Welcome to zvt

Contents

- Installation of zvt
  - Python version support
  - $ python -m pip install -U zvt
  - Get the Source Code
- Intro
  - Rethink market and programming
  - Core concepts building zvt
- Data
  - Data concepts
  - Record & query
  - Extend data
  - Adding new entity
  - Capital without country
- Factor
  - Factor concepts
  - Extending Factor
- Trader
  - Trader
- Machine learning
  - Tagger
  - MLMachine
- Drawer
  - Drawer concepts
- Contributor
- API
  - zvt
  - zvt.api
  - zvt.autocode
  - zvt.consts
  - zvt.contract

- zvt.domain
- zvt.factors
- zvt.fill_project
- zvt.informer
- zvt.main
- zvt.ml
- zvt.plugin
- zvt.recorders
- zvt.samples
- zvt.tag
- zvt.trader
- zvt.ui
- zvt.utils

Indices and tables

- Index
- Module Index
- Search Page

Installation of zvt

This part of the documentation covers the installation of zvt. The first step to using any software package is getting it properly installed.

Python version support

Officially Python 3.7, and 3.8.

$ python -m pip install -U zvt

To install zvt, simply run this simple command in your terminal of choice:

```
$ python -m pip install -U zvt
```

Get the Source Code

zvt is actively developed on GitHub, where the code is always available [https://github.com/zvtvz/zvt].

You can clone the public repository:

```
$ git clone git://github.com/zvtvz/zvt.git
```

Once you have a copy of the source, you can embed it in your own Python package, or install it into your site-packages easily:

```
$ cd zvt
$ python -m pip install .
```

Intro

This is a short introduction to zvt, you would learn the basic usage and glance a global view here.

Rethink market and programming

For practical trading, complex algorithm is fragile, complex algorithm building on complex facility is more fragile, complex algorithm building on complex facility by complex team is more and more fragile.

zvt want to provide a simple facility for building straightforward algorithm, it should be:

- **use the most basic programming concepts**
- **concise abstraction of the market**
- **correctness is obvious**

Core concepts building zvt

Technologies come and technologies go, but market insight is forever.
Your world is built by core concepts inside you, so it's you.
zvt world is built by core concepts inside market, so it's zvt.
We would show how core concepts building zvt in four aspects：

- Data
  - Data concepts
  - Record & query
  - Extend data
  - Adding new entity
  - Capital without country
- Factor
  - Factor concepts
  - Extending Factor

- Trader
  - Trader
- Machine learning
  - Tagger
  - MLMachine

Data

Without data, quant becomes a castle in the air.
So, what exactly is data?
zvt makes a concise and unified abstraction of data:

> In the world of zvt, there are two kinds of entities, one is tradable entity, the other is actor entity. Data is the events happened on them.

Note

Philosophically, entity is the existence described by itself, classification of existential concepts.

- Data concepts
  - TradableEntity
  - ActorEntity
  - Schema
  - IntervalLevel
  - Kdata(Quote, OHLC)
  - FinanceFactor
- Record & query
  - Dualism
  - Record data

    - Query data
- Extend data
    - Data structure
    - Data definition
    - Key steps to add data
    - Add new TradableEntity
- Adding new entity
    - Define entity Schema
    - Implement recorder for the entity
    - Define OHLC schema(kdata) for the entity
    - Implement recorder for OHLC schema(kdata)
    - Use them in zvt way
- Capital without country
    - Define entity Schema
    - Implement recorder for the entity
    - Define schema for the entity
    - Use them in zvt way

Data concepts

TradableEntity

`TradableEntity` is anything could be traded, it could be `Stock`, `Etf`, `Index`, future, cryptocurrency, or even a **sports match**.

Let's start with the real world tradable entities: China stock and USA stock ——— the world's most involved trading targets.

record and query stock:

```
>>> from zvt.domain import Stock
>>> Stock.record_data()
>>> df = Stock.query_data()
>>> print(df)
                   id        entity_id  timestamp entity_type
exchange    code   name  list_date end_date
0     stock_sh_600651  stock_sh_600651 1986-09-26       stock
sh  600651  *ST飞乐 1986-09-26     None
1     stock_sz_000004  stock_sz_000004 1990-12-01       stock
sz  000004   国华网安 1990-12-01     None
2     stock_sz_000005  stock_sz_000005 1990-12-10       stock
sz  000005   世纪星源 1990-12-10     None
3     stock_sh_600601  stock_sh_600601 1990-12-19       stock
sh  600601   方正科技 1990-12-19     None
4     stock_sh_600602  stock_sh_600602 1990-12-19       stock
sh  600602   云赛智联 1990-12-19     None
...               ...              ...        ...         ...
...     ...    ...        ...      ...
4615  stock_sh_603176  stock_sh_603176 2021-12-31       stock
sh  603176   汇通集团 2021-12-31     None
4616  stock_sh_600941  stock_sh_600941 2022-01-05       stock
sh  600941   中国移动 2022-01-05     None
4617  stock_sh_688262  stock_sh_688262 2022-01-06       stock
sh  688262   国芯科技 2022-01-06     None
4618  stock_sh_688176  stock_sh_688176 2022-01-07       stock
sh  688176   亚虹医药 2022-01-07     None
```

```
4619  stock_sz_301159  stock_sz_301159 2022-01-07       stock
sz  301159    N三维 2022-01-07     None

[4620 rows x 9 columns]
```

record and query stockus:

```
>>> from zvt.domain import Stockus
>>> Stockus.record_data()
>>> df = Stockus.query_data()
>>> print(df)
                      id            entity_id timestamp
entity_type exchange  code                            name
list_date end_date
0     stockus_nasdaq_CMII  stockus_nasdaq_CMII      None
stockus   nasdaq  CMII        CM Life Sciences II Inc-A
None     None
1     stockus_nasdaq_CBLI  stockus_nasdaq_CBLI      None
stockus   nasdaq  CBLI                     Cytocom Inc
None     None
2       stockus_nyse_XPOw    stockus_nyse_XPOw      None
stockus     nyse  XPOw             XPO Logistics Inc WI
None     None
3        stockus_nyse_WRI     stockus_nyse_WRI      None
stockus     nyse   WRI                        魏因加滕房地产投资
None     None
4       stockus_nyse_PROS    stockus_nyse_PROS      None
stockus     nyse  PROS              ProSight Global Inc
None     None
...                   ...                  ...       ...
...      ...   ...                              ...       ...
...
6345    stockus_nyse_FATH    stockus_nyse_FATH      None
stockus     nyse  FATH  Fathom Digital Manufacturing Co
None     None
6346    stockus_nyse_LOCL    stockus_nyse_LOCL      None
stockus     nyse  LOCL                Local Bounti Corp
None     None
6347     stockus_nyse_VLD     stockus_nyse_VLD      None
stockus     nyse   VLD                      Velo3D Inc
None     None
6348     stockus_nyse_AHI     stockus_nyse_AHI      None
stockus     nyse   AHI   Advanced Human Imaging Ltd ADR
```

```
None      None
6349     stockus_nyse_HLGN     stockus_nyse_HLGN       None
stockus      nyse   HLGN                        Heliogen Inc
None      None
```

what about other tradable entities?
Show current registered tradable entity type and its schema:

```
>>> from zvt.contract import zvt_context
>>> zvt_context.tradable_schema_map
{'stockus': zvt.domain.meta.stockus_meta.Stockus,
 'stockhk': zvt.domain.meta.stockhk_meta.Stockhk,
 'index': zvt.domain.meta.index_meta.Index,
 'etf': zvt.domain.meta.etf_meta.Etf,
 'stock': zvt.domain.meta.stock_meta.Stock,
 'block': zvt.domain.meta.block_meta.Block,
 'fund': zvt.domain.meta.fund_meta.Fund}
```

The key is **entity_type** and the value is its Schema.

From intuition, stockhk should be stock of hongkong:

```
>>> from zvt.domain import Stockhk
>>> Stockhk.record_data()
>>> df = Stockhk.query_data(index='code')
>>> print(df)

                     id         entity_id timestamp entity_type
exchange   code    name list_date end_date
code
00001  stockhk_hk_00001  stockhk_hk_00001       NaT     stockhk
hk  00001      长和      None     None
00002  stockhk_hk_00002  stockhk_hk_00002       NaT     stockhk
hk  00002    中电控股      None     None
00003  stockhk_hk_00003  stockhk_hk_00003       NaT     stockhk
hk  00003  香港中华煤气      None     None
00004  stockhk_hk_00004  stockhk_hk_00004       NaT     stockhk
hk  00004   九龙仓集团      None     None
00005  stockhk_hk_00005  stockhk_hk_00005       NaT     stockhk
hk  00005    汇丰控股      None     None
...                 ...                ...       ...         ...
...    ...     ...       ...      ...
```

```
09996  stockhk_hk_09996  stockhk_hk_09996       NaT     stockhk
hk  09996  沛嘉医疗-B      None     None
09997  stockhk_hk_09997  stockhk_hk_09997       NaT     stockhk
hk  09997    康基医疗       None     None
09998  stockhk_hk_09998  stockhk_hk_09998       NaT     stockhk
hk  09998    光荣控股       None     None
09999  stockhk_hk_09999  stockhk_hk_09999       NaT     stockhk
hk  09999    网易-S      None     None
80737  stockhk_hk_80737  stockhk_hk_80737       NaT     stockhk
hk  80737  湾区发展-R      None     None

[2597 rows x 9 columns]

>>> df[df.code=='00700']

                    id         entity_id timestamp entity_type
exchange   code  name list_date end_date
2112  stockhk_hk_00700  stockhk_hk_00700      None     stockhk
hk  00700  腾讯控股      None     None
```

From intuition, other tradable entities could be added to the system and used in the same way. Just follow [Add tradable entity]

ActorEntity

`ActorEntity` is the beings acting in the market, it could be government, company, fund or individual.

```
>>> from zvt.domain import StockInstitutionalInvestorHolder
>>> entity_ids = ["stock_sz_000338", "stock_sz_000001"]
>>>
StockInstitutionalInvestorHolder.record_data(entity_ids=entity_
ids)
>>> df =
StockInstitutionalInvestorHolder.query_data(entity_ids=entity_i
ds)
>>> print(df)
                                                      id
entity_id  timestamp    code  name                 actor_id
actor_type actor_code         actor_name report_period
report_date  holding_numbers  holding_ratio  holding_values
```

0 stock_sz_000001_1998-06-30_raised_fund_cn_184688
stock_sz_000001 1998-06-30 000001 平安银行
raised_fund_cn_184688 raised_fund 184688 基金
开元 half_year 1998-06-30 1.896697e+06 0.001771
3.269906e+07
1 stock_sz_000001_1998-09-30_raised_fund_cn_184688
stock_sz_000001 1998-09-30 000001 平安银行
raised_fund_cn_184688 raised_fund 184688 基金
开元 season3 1998-09-30 2.634093e+06 0.002460
4.151331e+07
2 stock_sz_000001_1998-12-31_raised_fund_cn_184688
stock_sz_000001 1998-12-31 000001 平安银行
raised_fund_cn_184688 raised_fund 184688 基金
开元 year 1998-12-31 2.673900e+06 0.002497
3.992133e+07
3 stock_sz_000001_1999-03-31_raised_fund_cn_184688
stock_sz_000001 1999-03-31 000001 平安银行
raised_fund_cn_184688 raised_fund 184688 基金
开元 season1 1999-03-31 2.378977e+06 0.002221
3.256820e+07
4 stock_sz_000001_1999-06-30_raised_fund_cn_500005
stock_sz_000001 1999-06-30 000001 平安银行
raised_fund_cn_500005 raised_fund 500005 基金
汉盛 half_year 1999-06-30 4.989611e+06 0.004659
1.386613e+08
... ...
... ...
... ...
... ...
22463 stock_sz_000338_2021-09-30_broker_cn_71067063
stock_sz_000338 2021-09-30 000338 潍柴动力
broker_cn_71067063 broker 71067063 东方红信和添
安4号 season3 2021-09-30 5.000000e+04 0.000012
8.580000e+05
22464 stock_sz_000338_2021-09-30_corporation_cn_1003...
stock_sz_000338 2021-09-30 000338 潍柴动力
corporation_cn_10030838 corporation 10030838 潍柴控股
集团有限公司 season3 2021-09-30 1.422551e+09
0.018071 2.441097e+10
22465 stock_sz_000338_2021-09-30_corporation_cn_1067...
stock_sz_000338 2021-09-30 000338 潍柴动力
corporation_cn_10671586 corporation 10671586 香港中央
结算有限公司 season3 2021-09-30 4.992710e+08

```
0.117713    8.567490e+09
22466  stock_sz_000338_2021-09-30_corporation_cn_1019...
stock_sz_000338 2021-09-30  000338  潍柴动力
corporation_cn_10196008  corporation   10196008       中国证券金
融股份有限公司       season3  2021-09-30     1.636089e+08
0.038574    2.807529e+09
22467  stock_sz_000338_2021-09-30_corporation_cn_1008...
stock_sz_000338 2021-09-30  000338  潍柴动力
corporation_cn_10086358  corporation   10086358  奥地利IVM技术咨
询维也纳有限公司       season3  2021-09-30     1.139387e+08
0.026863    1.955188e+09

[22468 rows x 14 columns]
```

Note

A good actor should know the good or bad actors in history, more importantly, the mind behind them.

Schema

Data structure describing `TradableEntity`, `ActorEntity` or events happen on them. Physically it's table with columns in sql database. One schema could have multiple storage with different providers.

From specific to general, all zvt schema usage is in the same way.

- from zvt.domain import {Schema}
- {Schema}.record_data
- {Schema}.query_data

Explore `domain` for pre defined schemas. And check record_data & query_data details

```
>>> from zvt.domain import *
>>> entity_ids = ["stock_sz_000338", "stock_sz_000001"]
>>> Stock1dHfqKdata.record_data(entity_ids=entity_ids,
provider="em")
>>> df = Stock1dHfqKdata.query_data(entity_ids=entity_ids,
```

```
provider="em")
>>> print(df)

                              id        entity_id  timestamp
provider    code  name level     open    close     high
low     volume      turnover  change_pct  turnover_rate
0      stock_sz_000001_1991-04-03  stock_sz_000001 1991-04-03
em  000001  平安银行    1d    49.00    49.00    49.00    49.00
1.0  5.000000e+03      0.2250         0.0000
1      stock_sz_000001_1991-04-04  stock_sz_000001 1991-04-04
em  000001  平安银行    1d    48.76    48.76    48.76    48.76
3.0  1.500000e+04     -0.0049         0.0000
2      stock_sz_000001_1991-04-05  stock_sz_000001 1991-04-05
em  000001  平安银行    1d    48.52    48.52    48.52    48.52
2.0  1.000000e+04     -0.0049         0.0000
3      stock_sz_000001_1991-04-06  stock_sz_000001 1991-04-06
em  000001  平安银行    1d    48.28    48.28    48.28    48.28
7.0  3.400000e+04     -0.0049         0.0000
4      stock_sz_000001_1991-04-08  stock_sz_000001 1991-04-08
em  000001  平安银行    1d    48.04    48.04    48.04    48.04
2.0  1.000000e+04     -0.0050         0.0000
...                           ...              ...        ...
...     ...   ...   ...      ...      ...      ...      ...
...           ...         ...            ...
10859  stock_sz_000338_2022-01-10  stock_sz_000338 2022-01-10
em  000338  潍柴动力    1d   314.38   314.38   320.37   312.69
956271.0  1.735153e+09      0.0149         0.0190
10860  stock_sz_000001_2022-01-11  stock_sz_000001 2022-01-11
em  000001  平安银行    1d  2974.07  2998.45  3019.58  2954.57
1581999.0  2.752485e+09      0.0121         0.0082
10861  stock_sz_000338_2022-01-11  stock_sz_000338 2022-01-11
em  000338  潍柴动力    1d   312.69   307.01   314.23   306.70
812187.0  1.444640e+09     -0.0234         0.0161
10862  stock_sz_000001_2022-01-12  stock_sz_000001 2022-01-12
em  000001  平安银行    1d  2998.45  2931.82  3004.95  2915.56
1502164.0  2.561266e+09     -0.0222         0.0077
10863  stock_sz_000338_2022-01-12  stock_sz_000338 2022-01-12
em  000338  潍柴动力    1d   307.01   305.78   309.62   302.86
882165.0  1.542044e+09     -0.0040         0.0175

[10864 rows x 15 columns]
```

The data of the schema is recorded in local database by default and could be updated incrementally.

Find them in this way:

```
{Schema}.get_storages()
```

e.g.

```
>>> Stock1dHfqKdata.get_storages()
[Engine(sqlite:////Users/foolcage/zvt-
home/data/joinquant_stock_1d_hfq_kdata.db?
check_same_thread=False),
 Engine(sqlite:////Users/foolcage/zvt-
home/data/em_stock_1d_hfq_kdata.db?check_same_thread=False)]
```

IntervalLevel

`IntervalLevel` is repeated fixed time interval, e.g, 5m, 1d. It's used in OHLC data for describing time window.

```
>>> from zvt.contract import *
>>> for level in IntervalLevel:
>>>     print(level.value)
tick
1m
5m
15m
30m
1h
4h
1d
1wk
1mon
```

Kdata(Quote, OHLC)

the candlestick data with OHLC.

the **TradableEntity** quote schema name follows below rules:

```
{entity_shema}{level}{adjust_type}Kdata
```

- entity_schema

TradableEntity class， e.g. Stock,Stockus.

- level

IntervalLevel value, e.g. 1d,1wk.

- adjust type

pre adjusted(qfq), post adjusted(hfq), or not adjusted(bfq).

```
>>> for adjust_type in AdjustType:
>>>     print(adjust_type.value)
```

Note

In order to be compatible with historical data, the qfq is an exception, {adjust_type} is left empty

The pre defined kdata schema could be found in **quotes**, it's separated by entity_schema, level, and adjust type.

e.g. Stock1dHfqKdata means China Stock daily hfq quotes.

```
>>> from zvt.domain import Stock1dHfqKdata
>>> Stock1dHfqKdata.record_data(code='000338', provider='em')
>>> df = Stock1dHfqKdata.query_data(code='000338',
provider='em')
>>> print(df)

                              id        entity_id  timestamp
provider    code  name level    open   close    high     low
volume      turnover  change_pct  turnover_rate
0     stock_sz_000338_2007-04-30  stock_sz_000338 2007-04-30
```

```
None  000338  潍柴动力    1d   70.00   64.93   71.00   62.88
207375.0  1.365189e+09      2.1720         0.1182
1     stock_sz_000338_2007-05-08  stock_sz_000338 2007-05-08
None  000338  潍柴动力    1d   66.60   64.00   68.00   62.88
86299.0  5.563198e+08     -0.0143         0.0492
2     stock_sz_000338_2007-05-09  stock_sz_000338 2007-05-09
None  000338  潍柴动力    1d   63.32   62.00   63.88   59.60
93823.0  5.782065e+08     -0.0313         0.0535
3     stock_sz_000338_2007-05-10  stock_sz_000338 2007-05-10
None  000338  潍柴动力    1d   61.50   62.49   64.48   61.01
47720.0  2.999226e+08      0.0079         0.0272
4     stock_sz_000338_2007-05-11  stock_sz_000338 2007-05-11
None  000338  潍柴动力    1d   61.90   60.65   61.90   59.70
39273.0  2.373126e+08     -0.0294         0.0224
...                          ...              ...        ...
...     ...   ...   ...     ...     ...     ...     ...
...           ...         ...            ...
3426  stock_sz_000338_2021-08-27  stock_sz_000338 2021-08-27
None  000338  潍柴动力    1d  331.97  345.95  345.95  329.82
1688497.0  3.370241e+09      0.0540         0.0398
3427  stock_sz_000338_2021-08-30  stock_sz_000338 2021-08-30
None  000338  潍柴动力    1d  345.95  342.72  346.10  337.96
1187601.0  2.377957e+09     -0.0093         0.0280
3428  stock_sz_000338_2021-08-31  stock_sz_000338 2021-08-31
None  000338  潍柴动力    1d  344.41  342.41  351.02  336.73
1143985.0  2.295195e+09     -0.0009         0.0270
3429  stock_sz_000338_2021-09-01  stock_sz_000338 2021-09-01
None  000338  潍柴动力    1d  341.03  336.42  341.03  328.28
1218697.0  2.383841e+09     -0.0175         0.0287
3430  stock_sz_000338_2021-09-02  stock_sz_000338 2021-09-02
None  000338  潍柴动力    1d  336.88  339.03  340.88  329.67
1023545.0  2.012006e+09      0.0078         0.0241

[3431 rows x 15 columns]
```

e.g. Stock30mHfqKdata means China Stock 30 minutes hfq quotes.

```
>>> from zvt.domain import Stock30mHfqKdata
>>> Stock30mHfqKdata.record_data(code='000338', provider='em')
>>> df = Stock30mHfqKdata.query_data(code='000338',
provider='em')
>>> print(df)
```

```
                                          id        entity_id
timestamp provider    code  name level    open   close    high
low    volume     turnover  change_pct  turnover_rate
0    stock_sz_000338_2022-01-07T10:00:00.000  stock_sz_000338
2022-01-07 10:00:00       em  000338  潍柴动力   30m  312.23
313.77  317.30  310.70  288036.0  521397671.0      0.0049
0.0057
1    stock_sz_000338_2022-01-07T10:30:00.000  stock_sz_000338
2022-01-07 10:30:00       em  000338  潍柴动力   30m  313.92
313.77  315.15  312.54  111887.0  201667653.0      0.0000
0.0022
2    stock_sz_000338_2022-01-07T11:00:00.000  stock_sz_000338
2022-01-07 11:00:00       em  000338  潍柴动力   30m  313.77
314.07  314.69  313.00   80072.0  144303962.0      0.0010
0.0016
3    stock_sz_000338_2022-01-07T11:30:00.000  stock_sz_000338
2022-01-07 11:30:00       em  000338  潍柴动力   30m  314.23
316.23  316.84  313.61  160797.0  291742498.0      0.0069
0.0032
4    stock_sz_000338_2022-01-07T13:30:00.000  stock_sz_000338
2022-01-07 13:30:00       em  000338  潍柴动力   30m  316.23
314.07  316.99  314.07  115775.0  210236422.0     -0.0068
0.0023
..                                          ...              ...
...      ...     ...   ...   ...     ...     ...     ...
...       ...          ...          ...            ...
251  stock_sz_000338_2022-02-28T11:30:00.000  stock_sz_000338
2022-02-28 11:30:00       em  000338  潍柴动力   30m  268.15
268.30  268.61  267.99   34581.0   52053276.0      0.0006
0.0007
252  stock_sz_000338_2022-02-28T13:30:00.000  stock_sz_000338
2022-02-28 13:30:00       em  000338  潍柴动力   30m  268.46
268.46  268.61  268.15   38019.0   57268380.0      0.0006
0.0008
253  stock_sz_000338_2022-02-28T14:00:00.000  stock_sz_000338
2022-02-28 14:00:00       em  000338  潍柴动力   30m  268.46
269.22  269.53  268.30   41713.0   62994140.0      0.0028
0.0008
254  stock_sz_000338_2022-02-28T14:30:00.000  stock_sz_000338
2022-02-28 14:30:00       em  000338  潍柴动力   30m  269.22
269.22  269.53  268.61   40815.0   61676966.0      0.0000
0.0008
255  stock_sz_000338_2022-02-28T15:00:00.000  stock_sz_000338
```

```
2022-02-28 15:00:00       em  000338  潍柴动力   30m  269.07
269.84  269.84  268.76   60190.0   91032952.0      0.0023
0.0012

[256 rows x 15 columns]
```

FinanceFactor

The usage is same as other entity events.

```
>>> from zvt.domain import FinanceFactor
>>> FinanceFactor.record_data(code='000338')
>>>
FinanceFactor.query_data(code='000338',columns=FinanceFactor.important_cols(),index='timestamp')

            basic_eps  total_op_income    net_profit
op_income_growth_yoy  net_profit_growth_yoy     roe    rota
gross_profit_margin  net_margin  timestamp
timestamp
2002-12-31        NaN     1.962000e+07  2.471000e+06
NaN                     NaN     NaN     NaN               0.2068
0.1259 2002-12-31
2003-12-31       1.27     3.574000e+09  2.739000e+08
181.2022                109.8778  0.7729  0.1783
0.2551      0.0766 2003-12-31
2004-12-31       1.75     6.188000e+09  5.369000e+08
0.7313                  0.9598  0.3245  0.1474
0.2489      0.0868 2004-12-31
2005-12-31       0.93     5.283000e+09  3.065000e+08
-0.1463                -0.4291  0.1327  0.0603
0.2252      0.0583 2005-12-31
2006-03-31       0.33     1.859000e+09  1.079000e+08
NaN                     NaN     NaN     NaN                  NaN
0.0598 2006-03-31
...               ...              ...           ...
...                     ...     ...     ...                  ...
...        ...
2020-08-28       0.59     9.449000e+10  4.680000e+09
0.0400                 -0.1148  0.0983  0.0229
0.1958      0.0603 2020-08-28
```

```
2020-10-31       0.90     1.474000e+11  7.106000e+09
0.1632                 0.0067  0.1502  0.0347
0.1949      0.0590 2020-10-31
2021-03-31       1.16     1.975000e+11  9.207000e+09
0.1327                 0.0112  0.1919  0.0444
0.1931      0.0571 2021-03-31
2021-04-30       0.42     6.547000e+10  3.344000e+09
0.6788                 0.6197  0.0622  0.0158
0.1916      0.0667 2021-04-30
2021-08-31       0.80     1.264000e+11  6.432000e+09
0.3375                 0.3742  0.1125  0.0287
0.1884      0.0653 2021-08-31

[66 rows x 10 columns]
```

Three financial tables

```
>>> BalanceSheet.record_data(code='000338')
>>> IncomeStatement.record_data(code='000338')
>>> CashFlowStatement.record_data(code='000338')
```

Note

Just remember, all schema usage is in the same way.

Record & query

Dualism

In and out, write and read, and so record and query.

Record data

Query data

Extend data

Data structure

zvt use sql database(default sqlite, and easy extend to others) to store data and provide uniform api to record and query data.(Thanks to the great Sqlalchemy [https://github.com/sqlalchemy/sqlalchemy])

As we know, TradableEntity and ActorEntity is everything about the market, data is the events happen on them. The Schema of the data would always in following structure:

It shows how we define the Entity in the market:

- entity_type

tradable entity type, e.g. stock, future, bond and so on.

- exchange

the exchange of entity

- code

the code of the entity

Naturally the id of the entity is defined:

```
{entity_type}_{exchange}_{code}
```

And the common fields:

- entity_id

entity id

- timestamp

happen time

- id

the id of the record, most of time, it's:

```
{entity_id}_{timestamp}
```

Data definition

zvt data is defined by three concepts:

- Database

The rule is one module define one db.

- Schema

The Schema defined in the module belongs to the Database defined inside.

- Provider

The Database could have different providers

Many pre supported data is in zvt.domain modules [https://github.com/zvtvz/zvt/blob/master/src/zvt/domain], e.g. Finance data [https://github.com/zvtvz/zvt/blob/master/src/zvt/domain/fundamental/finance.py], you could refer them when you want to extend data.

Key steps to add data

Let's show the key steps to add data.

1. Intent

I want to add some data about the news on Stock.

2. Find data source

The data source could be whatever you can see in the web ——— just write some web crawlers to get them, e.g, most of pre-implemented recorders [https://github.com/zvtvz/zvt/tree/master/src/zvt/recorders]. And you could use data from commercial data provider in the same way, e.g. joinquant recorders [https://github.com/zvtvz/zvt/tree/master/src/zvt/recorders/joinquant].

Let's show the crawler way here:

Open eastmoney wap [https://wap.eastmoney.com/quote/stock/0.002572.html] and Find news api [https://np-listapi.eastmoney.com/comm/wap/getListInfo?cb=callback&client=wap&type=1&mTypeAndCode=0.002572&pageSize=20&pageIndex=1&callback=jQuery18300174782479067403 52_1644568731256&_=1644568879493]

The news item result:


```
{
  "Art_ShowTime": "2022-02-09 14:12:30",
  "Art_Image": "",
  "Art_MediaName": "每日经济新闻",
  "Art_Code": "202202092271136373",
  "Art_Title": "首创证券维持索菲亚买入评级 公司简评报告：多重因素影响短期压制业绩 看好2022年利润修复",
  "Art_SortStart": "1644387150036373",
```

```
  "Art_VideoCount": 0,
  "Art_OriginUrl":
"http://finance.eastmoney.com/news/1354,202202092271136373.html
",
  "Art_Url":
"http://finance.eastmoney.com/a/202202092271136373.html"
}
```

3. Define Schema

No matter what the format of the data outside zvt is, we use zvt simple and uniform contract to define them.

```
>>> from sqlalchemy import Column, String
>>> from sqlalchemy.orm import declarative_base
>>> from zvt.contract import Mixin
>>> from zvt.contract.register import register_schema
>>> NewsBase = declarative_base()
>>> class StockNews(NewsBase, Mixin):
>>>     __tablename__ = "stock_news"
>>>     #: news title
>>>     news_title = Column(String)
>>>
>>> register_schema(providers=["em"], db_name="stock_news",
schema_base=NewsBase, entity_type="stock")
```

Check the defined db

```
>>> StockNews.get_storages()

[Engine(sqlite:////Users/foolcage/zvt-
home/data/em_stock_news.db?check_same_thread=False)]
```

As you see, the db file name format is:

```
{provider}_{db_name}
```

If you have another provider, e.g. sina, just register it too:

```
>>> register_schema(providers=["em", "sina"],
db_name="stock_news", schema_base=NewsBase,
```

```
entity_type="stock")
```

And you could find another db file:

```
>>> StockNews.get_storages()

[Engine(sqlite:////Users/foolcage/zvt-
home/data/sina_stock_news.db?check_same_thread=False)]
```

The advantage of this mechanism is:

- schema is the way you want
- provider could be switched seamlessly

4. Implement recorder

Let's implement a recorder for StockNews.

```
import pandas as pd
from zvt.contract.api import df_to_db
from zvt.contract.recorder import FixedCycleDataRecorder
from zvt.domain import Stock
from zvt.domain.misc.stock_news import StockNews
from zvt.recorders.em import em_api


class EMStockNewsRecorder(FixedCycleDataRecorder):
    original_page_url =
"https://wap.eastmoney.com/quote/stock/0.002572.html"
    url = "https://np-
listapi.eastmoney.com/comm/wap/getListInfo?
cb=callback&client=wap&type=1&mTypeAndCode=0.002572&pageSize=20
0&pageIndex=
{}&callback=jQuery18300174782479067403 52_1644568731256&_=164456
8879493"

    # the entity class you record for
    entity_schema = Stock
    # connect schema with recorder
    data_schema = StockNews
    # entity data provider
```

```
    entity_provider = "em"
    # data schema provider
    provider = "em"

    def record(self, entity, start, end, size, timestamps):
        news = em_api.get_news(entity_id=entity.id)
        df = pd.DataFrame.from_records(news)
        self.logger.info(df)
        df_to_db(df=df, data_schema=self.data_schema,
provider=self.provider, force_update=self.force_update)
```

After that, you could use the data in zvt way:

```
>>> StockNews.record_data(provider="em",
entity_id="stock_sz_002572")
>>> df = StockNews.query_data(entity_id="stock_sz_002572")
>>> print(df)

                                          id        entity_id
timestamp                                          news_title
0    stock_sz_002572_2019-05-20 19:50:42  stock_sz_002572 2019-
05-20 19:50:42                 索菲亚（002572.SZ）：股价回撤超65% 是
低吸机会吗？
1    stock_sz_002572_2019-05-24 22:39:47  stock_sz_002572 2019-
05-24 22:39:47                              索菲亚拟发行不超过5亿元
超短期融资券
2    stock_sz_002572_2019-05-24 22:48:54  stock_sz_002572 2019-
05-24 22:48:54               索菲亚(002572.SZ)拟终止发行不超10亿元的
可转换公司债券
3    stock_sz_002572_2019-06-04 14:06:42  stock_sz_002572 2019-
06-04 14:06:42               索菲亚(002572.SZ)截至5月底已累计回购
2.02亿元的股份
4    stock_sz_002572_2019-06-13 06:24:44  stock_sz_002572 2019-
06-13 06:24:44                                  索菲亚功臣王飚能否扶
起汉森中国
..                                     ...                ...
...                                                   ...
367  stock_sz_002572_2022-02-07 14:04:40  stock_sz_002572 2022-
02-07 14:04:40          开源证券维持索菲亚买入评级 近期获6份券商研报关注
目标均价涨幅59.82%
368  stock_sz_002572_2022-02-07 15:32:46  stock_sz_002572 2022-
02-07 15:32:46                              【调研快报】索菲亚接待机
构投资者调研
```

```
369  stock_sz_002572_2022-02-08 16:50:37  stock_sz_002572 2022-02-08 16:50:37                              索菲亚：公司承接了容东片区安置房项目
370  stock_sz_002572_2022-02-08 21:59:00  stock_sz_002572 2022-02-08 21:59:00       9亿坏账拖累净利下降九成 家具巨头索菲亚"甩包袱"起跑腰斩的股价能否趁势抬头？
371  stock_sz_002572_2022-02-09 14:12:30  stock_sz_002572 2022-02-09 14:12:30   首创证券维持索菲亚买入评级 公司简评报告：多重因素影响短期压制业绩 看好2022年利润修复

[372 rows x 4 columns]
```

Add new TradableEntity

It’s human nature to like the new and hate the old. Adding new TradableEntity is easy in zvt.

And from a higher perspective, trading is everywhere. you make trading everytime when you make the decision.

So you could treat Country as TradableEntity and make trading when making decision where to live or invest.

Let’s show the key steps to add new TradableEntity.

1. Define entity Schema

```
# -*- coding: utf-8 -*-

from sqlalchemy import Column, String, Float
from sqlalchemy.orm import declarative_base

from zvt.contract.schema import TradableEntity
from zvt.contract.register import register_schema,
register_entity

CountryMetaBase = declarative_base()


@register_entity(entity_type="country")
```

```
class Country(CountryMetaBase, TradableEntity):
    __tablename__ = "country"

    #: 区域
    #: region
    region = Column(String(length=128))
    #: 首都
    #: capital city
    capital_city = Column(String(length=128))
    #: 收入水平
    #: income level
    income_level = Column(String(length=64))
    #: 贷款类型
    #: lending type
    lending_type = Column(String(length=64))
    #: 经度
    #: longitude
    longitude = Column(Float)
    #: 纬度
    #: latitude
    latitude = Column(Float)


register_schema(providers=["wb"], db_name="country_meta",
schema_base=CountryMetaBase)
```

entity_type, exchange and code define the entity, for country, it's in following way:

```
entity_type = "country"
exchange = "galaxy"
code = "iso code"
```

e.g. country_galaxy_CN = China, country_galaxy_US = United States of America

2. Implement recorder for the entity

```
from zvt.contract.api import df_to_db
from zvt.contract.recorder import Recorder
from zvt.domain.meta.country_meta import Country
from zvt.recorders.wb import wb_api
```

```
class WBCountryRecorder(Recorder):
    provider = "wb"
    data_schema = Country

    def run(self):
        df = wb_api.get_countries()
        df_to_db(df=df, data_schema=self.data_schema,
provider=self.provider, force_update=self.force_update)
```

3. Define schema for the entity

e.g treasury yield of the country

```
# -*- coding: utf-8 -*-
from sqlalchemy import Column, String, Float
from sqlalchemy.orm import declarative_base

from zvt.contract import Mixin
from zvt.contract.register import register_schema

CurrencyBase = declarative_base()


class TreasuryYield(CurrencyBase, Mixin):
    __tablename__ = "treasury_yield"

    code = Column(String(length=32))

    # 2年期
    yield_2 = Column(Float)
    # 5年期
    yield_5 = Column(Float)
    # 10年期
    yield_10 = Column(Float)
    # 30年期
    yield_30 = Column(Float)


register_schema(providers=["em"], db_name="currency",
schema_base=CurrencyBase)
```

And the recorder [https://github.com/zvtvz/zvt/blob/master/src/zvt/recorders/em/macro/em_treasury_yield_recorder.py] for the schema

4. Use them in zvt way

Find the rich country:

```
>>> from zvt.domain import Country
>>> Country.record_data()
>>> df = Country.query_data()
>>> df[df['income_level']=='High income']

                   id          entity_id timestamp entity_type
exchange code                    name list_date end_date
region      capital_city income_level    lending_type
longitude   latitude
0    country_galaxy_AW  country_galaxy_AW      None     country
galaxy   AW                  Aruba      None     None  Latin
America & Caribbean         Oranjestad  High income  Not
classified  -70.016700  12.516700
7    country_galaxy_AD  country_galaxy_AD      None     country
galaxy   AD                 Andorra      None     None
Europe & Central Asia  Andorra la Vella  High income  Not
classified    1.521800  42.507500
9    country_galaxy_AE  country_galaxy_AE      None     country
galaxy   AE    United Arab Emirates      None     None  Middle
East & North Africa         Abu Dhabi  High income  Not
classified   54.370500  24.476400
13   country_galaxy_AG  country_galaxy_AG      None     country
galaxy   AG     Antigua and Barbuda      None     None  Latin
America & Caribbean       Saint John's  High income
IBRD  -61.845600  17.117500
14   country_galaxy_AU  country_galaxy_AU      None     country
galaxy   AU               Australia      None     None
East Asia & Pacific          Canberra  High income  Not
classified  149.129000 -35.282000
..                 ...                ...       ...         ...
...  ...                     ...       ...      ...
...               ...          ...             ...         ...
...
277  country_galaxy_TW  country_galaxy_TW      None     country
```

```
galaxy   TW           Taiwan, China      None     None
East Asia & Pacific                    High income  Not
classified         NaN        NaN
282  country_galaxy_UY  country_galaxy_UY      None     country
galaxy   UY                Uruguay      None     None  Latin
America & Caribbean         Montevideo  High income
IBRD  -56.067500 -34.894100
283  country_galaxy_US  country_galaxy_US      None     country
galaxy   US           United States      None     None
North America   Washington D.C.  High income  Not classified
-77.032000  38.889500
287  country_galaxy_VG  country_galaxy_VG      None     country
galaxy   VG  British Virgin Islands      None     None  Latin
America & Caribbean          Road Town  High income  Not
classified  -64.623056  18.431389
288  country_galaxy_VI  country_galaxy_VI      None     country
galaxy   VI   Virgin Islands (U.S.)      None     None  Latin
America & Caribbean   Charlotte Amalie  High income  Not
classified  -64.896300  18.335800

[80 rows x 15 columns]
```

Compare treasury yields of different maturities:

```
>>> from zvt.domain import TreasuryYield
>>> from zvt.api.intent import compare
>>> TreasuryYield.record_data()
>>> compare(codes=["US"], schema=TreasuryYield, columns=
["yield_2", "yield_10", "yield_30"])
```

Adding new entity

It's human nature to like the new and hate the old. Adding new TradableEntity is easy in zvt.

Adding new entity is nothing than a specific case of Adding data. Let's show the key steps below which add `Future`.

Define entity Schema

```
# -*- coding: utf-8 -*-
from sqlalchemy.ext.declarative import declarative_base

from zvt.contract.register import register_schema,
register_entity
from zvt.contract.schema import TradableEntity

FutureMetaBase = declarative_base()


@register_entity(entity_type="future")
class Future(FutureMetaBase, TradableEntity):
    __tablename__ = "future"


register_schema(providers=["em"], db_name="future_meta",
schema_base=FutureMetaBase)
```

Implement recorder for the entity

```
from zvt.contract.api import df_to_db
from zvt.contract.recorder import Recorder
from zvt.domain import Future
from zvt.recorders.em import em_api


class EMFutureRecorder(Recorder):
```

```
    provider = "em"
    data_schema = Future

    def run(self):
        df = em_api.get_tradable_list(entity_type="future")
        self.logger.info(df)
        df_to_db(df=df, data_schema=self.data_schema,
provider=self.provider, force_update=self.force_update)
```

Define OHLC schema(kdata) for the entity

zvt provide a standard way to generate OHLC schema for the tradable entity. All [OHLC schemas](https://github.com/zvtvz/zvt/blob/master/src/zvt/domain/quotes) is generated by [fill project script](https://github.com/zvtvz/zvt/blob/master/src/zvt/fill_project.py).

e.g generate Future OHLC schema.

```
gen_kdata_schema(
    pkg="zvt",
    providers=["em"],
    entity_type="future",
    levels=[IntervalLevel.LEVEL_1DAY],
    entity_in_submodule=True,
)
```

The OHLC schema definition principle is: **one level one file**

So we would define a common OHLC schema for each entity type in [quotes module](https://github.com/zvtvz/zvt/blob/master/src/zvt/domain/quotes/__init__.py).

e.g. Future common OHLC schema

```
class FutureKdataCommon(KdataCommon):
    #: 持仓量
    interest = Column(Float)
    #: 结算价
    settlement = Column(Float)
    #: 涨跌幅(按收盘价)
    # change_pct = Column(Float)
```

```
    #: 涨跌幅(按结算价)
    change_pct1 = Column(Float)
```

And we could relate the common kdata schema with the recorder and route level to specific schema automatically.

Implement recorder for OHLC schema(kdata)

Check [em quotes recorder](https://github.com/zvtvz/zvt/blob/master/src/zvt/recorders/em/quotes/em_kdata_recorder.py) [https://github.com/zvtvz/zvt/blob/master/src/zvt/recorders/em/quotes/em_kdata_recorder.py] for the details.

```
class EMFutureKdataRecorder(BaseEMStockKdataRecorder):
entity_provider = "em"
entity_schema = Future

data_schema = FutureKdataCommon
```

Use them in zvt way

Fetch the entity list:

```
>>> from zvt.domain import Future
>>> Future.record_data()
>>> df = Future.query_data()
>>> print(df)

                  id        entity_id  timestamp entity_type
exchange code    name  list_date end_date
0   future_cffex_IC  future_cffex_IC        NaT      future
cffex   IC  中证当月连续        NaT     None
1   future_cffex_IF  future_cffex_IF        NaT      future
cffex   IF  沪深当月连续        NaT     None
2   future_cffex_IH  future_cffex_IH        NaT      future
cffex   IH  上证当月连续        NaT     None
3    future_cffex_T   future_cffex_T        NaT      future
cffex    T  十债当季连续        NaT     None
4   future_cffex_TF  future_cffex_TF        NaT      future
cffex   TF  五债当季连续        NaT     None
..               ...              ...        ...         ...
```

```
...  ...     ...         ...      ...
65    future_ine_LU    future_ine_LU 2020-06-22      future
ine   LU  低硫燃油主力 2020-06-22     None
66   future_czce_PF   future_czce_PF 2020-10-12      future
czce   PF    短纤主力 2020-10-12     None
67    future_ine_BC    future_ine_BC 2020-11-19      future
ine   BC   国际铜主力 2020-11-19     None
68    future_dce_LH    future_dce_LH 2021-01-08      future
dce   LH    生猪主力 2021-01-08     None
69   future_czce_PK   future_czce_PK 2021-02-01      future
czce   PK    花生主力 2021-02-01     None

[70 rows x 9 columns]
```

Fetch the quotes:

```
>>> from zvt.domain import Future1dKdata
>>> Future1dKdata.record_data(code="CU")
>>> df = Future1dKdata.query_data(code="CU")
>>> print(df)

                             id       entity_id  timestamp
provider code  name level     open    close     high      low
volume      turnover  change_pct  turnover_rate interest
settlement change_pct1
0     future_shfe_CU_1996-04-03  future_shfe_CU 1996-04-03
em   CU  沪铜主力    1d  22930.0  22840.0  23000.0  22840.0
353.0  0.000000e+00      0.0000            0.0     None
None        None
1     future_shfe_CU_1996-04-04  future_shfe_CU 1996-04-04
em   CU  沪铜主力    1d  22700.0  22750.0  22820.0  22650.0
251.0  0.000000e+00     -0.0039            0.0     None
None        None
2     future_shfe_CU_1996-04-05  future_shfe_CU 1996-04-05
em   CU  沪铜主力    1d  22520.0  22780.0  22820.0  22500.0
298.0  0.000000e+00      0.0013            0.0     None
None        None
3     future_shfe_CU_1996-04-08  future_shfe_CU 1996-04-08
em   CU  沪铜主力    1d  22660.0  22650.0  22680.0  22600.0
98.0  0.000000e+00     -0.0057            0.0     None
None        None
4     future_shfe_CU_1996-04-09  future_shfe_CU 1996-04-09
em   CU  沪铜主力    1d  22830.0  22810.0  22860.0  22810.0
```

```
56.0  0.000000e+00      0.0071            0.0     None
None        None
...                          ...             ...        ...
...  ...   ...   ...      ...      ...      ...      ...
...           ...         ...            ...      ...
...         ...
6343  future_shfe_CU_2022-04-21  future_shfe_CU 2022-04-21
em   CU  沪铜主力    1d  74140.0  74480.0  74750.0  74140.0
48008.0  1.787678e+10     -0.0004            0.0     None
None        None
6344  future_shfe_CU_2022-04-22  future_shfe_CU 2022-04-22
em   CU  沪铜主力    1d  74800.0  75010.0  75200.0  74690.0
58874.0  2.205633e+10      0.0073            0.0     None
None        None
6345  future_shfe_CU_2022-04-25  future_shfe_CU 2022-04-25
em   CU  沪铜主力    1d  74900.0  73660.0  75190.0  73660.0
107455.0  3.989090e+10     -0.0168            0.0     None
None        None
6346  future_shfe_CU_2022-04-26  future_shfe_CU 2022-04-26
em   CU  沪铜主力    1d  73170.0  73260.0  73750.0  72500.0
113019.0  4.130931e+10     -0.0132            0.0     None
None        None
6347  future_shfe_CU_2022-04-27  future_shfe_CU 2022-04-27
em   CU  沪铜主力    1d  72990.0  73100.0  73560.0  72910.0
61563.0  2.254089e+10      0.0000            0.0     None
None        None

[6348 rows x 18 columns]
```

Capital without country

From a higher perspective, trading is everywhere. you make trading everytime when you make the decision.

So you could treat Country as TradableEntity and make trading when making decision where to live or invest.

It's nothing than a specific case of Adding new entity.

Let's show the key steps below.

Define entity Schema

```
# -*- coding: utf-8 -*-

from sqlalchemy import Column, String, Float
from sqlalchemy.orm import declarative_base

from zvt.contract.schema import TradableEntity
from zvt.contract.register import register_schema,
register_entity

CountryMetaBase = declarative_base()


@register_entity(entity_type="country")
class Country(CountryMetaBase, TradableEntity):
    __tablename__ = "country"

    #: 区域
    #: region
    region = Column(String(length=128))
    #: 首都
    #: capital city
    capital_city = Column(String(length=128))
    #: 收入水平
    #: income level
```

```
    income_level = Column(String(length=64))
    #: 贷款类型
    #: lending type
    lending_type = Column(String(length=64))
    #: 经度
    #: longitude
    longitude = Column(Float)
    #: 纬度
    #: latitude
    latitude = Column(Float)


register_schema(providers=["wb"], db_name="country_meta",
schema_base=CountryMetaBase)
```

entity_type, exchange and code define the entity, for country, it's in following way:

```
entity_type = "country"
exchange = "galaxy"
code = "iso code"
```

e.g. country_galaxy_CN = China, country_galaxy_US = United States of America

Implement recorder for the entity

```
from zvt.contract.api import df_to_db
from zvt.contract.recorder import Recorder
from zvt.domain.meta.country_meta import Country
from zvt.recorders.wb import wb_api


class WBCountryRecorder(Recorder):
    provider = "wb"
    data_schema = Country

    def run(self):
        df = wb_api.get_countries()
        df_to_db(df=df, data_schema=self.data_schema,
provider=self.provider, force_update=self.force_update)
```

Define schema for the entity

e.g treasury yield of the country

```
# -*- coding: utf-8 -*-
from sqlalchemy import Column, String, Float
from sqlalchemy.orm import declarative_base

from zvt.contract import Mixin
from zvt.contract.register import register_schema

MonetaryBase = declarative_base()


class TreasuryYield(MonetaryBase, Mixin):
    __tablename__ = "treasury_yield"

    code = Column(String(length=32))

    # 2年期
    yield_2 = Column(Float)
    # 5年期
    yield_5 = Column(Float)
    # 10年期
    yield_10 = Column(Float)
    # 30年期
    yield_30 = Column(Float)


register_schema(providers=["em"], db_name="monetary",
schema_base=MonetaryBase)
# the __all__ is generated
__all__ = ["TreasuryYield"]
```

And the recorder
[https://github.com/zvtvz/zvt/blob/master/src/zvt/recorders/em/macro/em_treasury_yield_recorder.py] for the schema

Use them in zvt way

Find the rich country:

```
>>> from zvt.domain import Country
>>> Country.record_data()
>>> df = Country.query_data()
>>> df[df['income_level']=='High income']

                   id          entity_id timestamp entity_type
exchange code                    name list_date end_date
region      capital_city income_level    lending_type
longitude   latitude
0    country_galaxy_AW  country_galaxy_AW      None     country
galaxy   AW                   Aruba      None     None  Latin
America & Caribbean         Oranjestad  High income  Not
classified  -70.016700  12.516700
7    country_galaxy_AD  country_galaxy_AD      None     country
galaxy   AD                 Andorra      None     None
Europe & Central Asia  Andorra la Vella  High income  Not
classified    1.521800  42.507500
9    country_galaxy_AE  country_galaxy_AE      None     country
galaxy   AE    United Arab Emirates      None     None  Middle
East & North Africa         Abu Dhabi  High income  Not
classified   54.370500  24.476400
13   country_galaxy_AG  country_galaxy_AG      None     country
galaxy   AG     Antigua and Barbuda      None     None  Latin
America & Caribbean       Saint John's  High income
IBRD  -61.845600  17.117500
14   country_galaxy_AU  country_galaxy_AU      None     country
galaxy   AU               Australia      None     None
East Asia & Pacific          Canberra  High income  Not
classified  149.129000 -35.282000
..                 ...                ...       ...         ...
...  ...                     ...       ...      ...
...               ...          ...             ...         ...
...
277  country_galaxy_TW  country_galaxy_TW      None     country
galaxy   TW           Taiwan, China      None     None
East Asia & Pacific                    High income  Not
classified         NaN        NaN
282  country_galaxy_UY  country_galaxy_UY      None     country
galaxy   UY                 Uruguay      None     None  Latin
America & Caribbean         Montevideo  High income
IBRD  -56.067500 -34.894100
```

```
283  country_galaxy_US  country_galaxy_US      None     country
galaxy   US           United States      None     None
North America   Washington D.C.  High income  Not classified
-77.032000  38.889500
287  country_galaxy_VG  country_galaxy_VG      None     country
galaxy   VG  British Virgin Islands      None     None  Latin
America & Caribbean          Road Town  High income  Not
classified  -64.623056  18.431389
288  country_galaxy_VI  country_galaxy_VI      None     country
galaxy   VI   Virgin Islands (U.S.)      None     None  Latin
America & Caribbean   Charlotte Amalie  High income  Not
classified  -64.896300  18.335800

[80 rows x 15 columns]
```

Compare treasury yields of different maturities:

```
>>> from zvt.domain import TreasuryYield
>>> from zvt.api.intent import compare
>>> TreasuryYield.record_data()
>>> compare(codes=["US"], schema=TreasuryYield, columns=
["yield_2", "yield_10", "yield_30"])
```

Factor

Without factor, the heart of quant is empty.
So, what exactly is factor?
zvt makes a concise and unified abstraction of factor:

> In the world of zvt, factor is data describing market. It's computed from from Schema, and save as new Schema if need.

- Factor concepts
  - Normal data
  - Factor
  - Transformer
  - Accumulator
  - TargetSelector
- Extending Factor
  - Rethink NormalData
  - Factor data structure
  - Write transformer
  - Factor with IntervalLevel
  - Stream computing
  - Factor persistence
  - Select the targets
  - Write accumulator

Factor concepts

Normal data

`NormalData` is the data containing pandas dataframe with multiple index which level 0 named entity_id and level 1 named timestamp:

entity_id	timestamp	col1	col2	col3	col4
stock_sz_000338	2020-05-05	1.2	0.5	0.3	a
…	2020-05-06	1.0	0.7	0.2	b
stock_sz_000778	2020-05-05	1.2	0.5	0.3	a
…	2020-05-06	1.0	0.7	0.2	b

This data structure is used heavily in zvt computing, you should be familiar with it. Pandas multiple index guide [https://pandas.pydata.org/pandas-docs/stable/user_guide/advanced.html#] is a good start.

Query data returning normal data in this way:

```
{Schema}.query_data(index=["entity_id, timestamp"])
```

e.g.

```
>>> from zvt.domain import *
>>> entity_ids = ["stock_sz_000338", "stock_sz_000001"]
>>> Stock1dHfqKdata.record_data(entity_ids=entity_ids,
```

```
provider="em")
>>> df = Stock1dHfqKdata.query_data(entity_ids=entity_ids,
provider="em", index=["entity_id", "timestamp"])
>>> print(df)

                                                      id
entity_id  timestamp provider    code  name level    open
close    high     low    volume      turnover  change_pct
turnover_rate
entity_id       timestamp
stock_sz_000001 1991-04-03  stock_sz_000001_1991-04-03
stock_sz_000001 1991-04-03       em  000001  平安银行    1d
49.00   49.00   49.00   49.00       1.0  5.000000e+03
0.2250         0.0000
                1991-04-04  stock_sz_000001_1991-04-04
stock_sz_000001 1991-04-04       em  000001  平安银行    1d
48.76   48.76   48.76   48.76       3.0  1.500000e+04
-0.0049         0.0000
                1991-04-05  stock_sz_000001_1991-04-05
stock_sz_000001 1991-04-05       em  000001  平安银行    1d
48.52   48.52   48.52   48.52       2.0  1.000000e+04
-0.0049         0.0000
                1991-04-06  stock_sz_000001_1991-04-06
stock_sz_000001 1991-04-06       em  000001  平安银行    1d
48.28   48.28   48.28   48.28       7.0  3.400000e+04
-0.0049         0.0000
                1991-04-08  stock_sz_000001_1991-04-08
stock_sz_000001 1991-04-08       em  000001  平安银行    1d
48.04   48.04   48.04   48.04       2.0  1.000000e+04
-0.0050         0.0000
...                                                  ...
...        ...      ...     ...   ...   ...     ...     ...
...     ...       ...           ...         ...            ...
stock_sz_000338 2022-01-17  stock_sz_000338_2022-01-17
stock_sz_000338 2022-01-17       em  000338  潍柴动力    1d
296.26  297.64  298.71  293.49  504866.0  8.546921e+08
0.0026         0.0100
                2022-01-18  stock_sz_000338_2022-01-18
stock_sz_000338 2022-01-18       em  000338  潍柴动力    1d
298.10  300.87  302.71  296.10  622455.0  1.064735e+09
0.0109         0.0124
                2022-01-19  stock_sz_000338_2022-01-19
stock_sz_000338 2022-01-19       em  000338  潍柴动力    1d
```

```
299.64  299.48  304.24  298.56  610096.0  1.049195e+09
-0.0046         0.0121
                2022-01-20  stock_sz_000338_2022-01-20
stock_sz_000338 2022-01-20       em  000338  潍柴动力    1d
298.10  294.87  299.18  290.11  812949.0  1.361764e+09
-0.0154         0.0161
                2022-01-21  stock_sz_000338_2022-01-21
stock_sz_000338 2022-01-21       em  000338  潍柴动力    1d
292.72  287.04  293.34  284.58  754156.0  1.234360e+09
-0.0266         0.0150

[10878 rows x 15 columns]
```

Factor

Factor is a computing facility to build *factor* according your mind ——— algorithm. It reads data from schema, use **Transformer**, **Accumulator** or your custom logic to compute, and save the result to new schema if need. It also provides a standard way to evaluate the targets which could be used by **TargetSelector** and **Trader** for backtesting or real trading.

Transformer

Computing factor which depends on input data only. Here is an example: **MaTransformer**

Accumulator

Computing factor which depends on input data and previous result of the factor. Here is an example:

Let's have a look by example:

```
>>> from zvt.factors import GoldCrossFactor
>>> from zvt.domain import Stock1dHfqKdata
>>> entity_ids = ["stock_sz_000338"]
>>> Stock1dHfqKdata.record_data(entity_ids=entity_ids,
provider="em")
```

```
>>> factor = GoldCrossFactor(entity_ids=entity_ids,
provider="em", start_timestamp="2018-01-01")
>>> print(factor.factor_df)
>>> print(factor.result_df)
>>> factor.draw(show=True)
                          level      turnover    high
id    open     low        entity_id  timestamp   close
turnover_rate     volume      diff       dea      macd  live
bull  live_count
entity_id       timestamp
stock_sz_000338 2018-01-02    1d  8.325588e+08  145.97
stock_sz_000338_2018-01-02  141.21  141.06  stock_sz_000338
2018-01-02  145.67         0.0225   972471.0       NaN
NaN       NaN    -1  False          -1
                2018-01-03    1d  7.530370e+08  147.66
stock_sz_000338_2018-01-03  146.13  144.29  stock_sz_000338
2018-01-03  144.44         0.0202   870225.0       NaN
NaN       NaN    -1  False          -2
                2018-01-04    1d  4.917067e+08  145.51
stock_sz_000338_2018-01-04  144.75  143.67  stock_sz_000338
2018-01-04  145.21         0.0133   574335.0       NaN
NaN       NaN    -1  False          -3
                2018-01-05    1d  5.282211e+08  146.59
stock_sz_000338_2018-01-05  146.44  143.21  stock_sz_000338
2018-01-05  143.21         0.0143   616244.0       NaN
NaN       NaN    -1  False          -4
                2018-01-08    1d  1.255871e+09  150.43
stock_sz_000338_2018-01-08  143.82  143.82  stock_sz_000338
2018-01-08  150.12         0.0331  1426567.0       NaN
NaN       NaN    -1  False          -5
...                          ...           ...     ...
...     ...     ...              ...        ...     ...
...        ...       ...       ...       ...   ...    ...
...
                2022-01-17    1d  8.546921e+08  298.71
stock_sz_000338_2022-01-17  296.26  293.49  stock_sz_000338
2022-01-17  297.64         0.0100   504866.0 -1.386687
1.781615 -6.336603    -1  False         -12
                2022-01-18    1d  1.064735e+09  302.71
stock_sz_000338_2022-01-18  298.10  296.10  stock_sz_000338
2022-01-18  300.87         0.0124   622455.0 -1.694421
1.086407 -5.561657    -1  False         -13
                2022-01-19    1d  1.049195e+09  304.24
```

```
stock_sz_000338_2022-01-19  299.64  298.56  stock_sz_000338
2022-01-19  299.48         0.0121   610096.0 -2.027097
0.463707 -4.981607    -1  False         -14
                2022-01-20    1d  1.361764e+09  299.18
stock_sz_000338_2022-01-20  298.10  290.11  stock_sz_000338
2022-01-20  294.87         0.0161   812949.0 -2.632389
-0.155513 -4.953753    -1  False         -15
                2022-01-21    1d  1.234360e+09  293.34
stock_sz_000338_2022-01-21  292.72  284.58  stock_sz_000338
2022-01-21  287.04         0.0150   754156.0 -3.701237
-0.864657 -5.673159    -1  False         -16

[987 rows x 17 columns]
                            filter_result
entity_id       timestamp
stock_sz_000338 2018-01-02          False
                2018-01-03          False
                2018-01-04          False
                2018-01-05          False
                2018-01-08          False
...                                   ...
                2022-01-17          False
                2022-01-18          False
                2022-01-19          False
                2022-01-20          False
                2022-01-21          False

[987 rows x 1 columns]
```

Follow [Extending factor]() to do the funny part.

TargetSelector

The class select targets according to Factors. You could calculate factors in the whole market and use selector to choose the targets.

```
from zvt.contract import IntervalLevel
from zvt.factors.target_selector import TargetSelector
from zvt.factors.ma.ma_factor import CrossMaFactor

entity_ids = ["stock_sz_000338"]
entity_type = "stock"
start_timestamp = "2018-01-01"
end_timestamp = "2019-06-30"
my_selector = TargetSelector(
    entity_ids=entity_ids, entity_schema=entity_type,
start_timestamp=start_timestamp, end_timestamp=end_timestamp
)
# add the factors
my_selector.add_factor(
    CrossMaFactor(
        entity_provider="em",
        provider="em",
        entity_ids=entity_ids,
        start_timestamp=start_timestamp,
        end_timestamp=end_timestamp,
        computing_window=10,
        windows=[5, 10],
        need_persist=False,
        level=IntervalLevel.LEVEL_1DAY,
        adjust_type="hfq",
    )
)
my_selector.run()
print(my_selector.open_long_df)
print(my_selector.open_short_df)
my_selector.get_open_long_targets("2019-06-27")
```

If not set entity_ids arguments, the selected targets would be in whole market. And it provides get_open_long_targets function to select targets on the timestamp. For multiple targets backtesting, this pre computed factor would be very fast.

Extending Factor

Rethink NormalData

Normal data format is as below:

entity_id	timestamp	y1	y2	y3	y4
stock_sz_000338	2008-01-07	86.64	87.08	85.60	85.74
	2008-01-08	86.13	94.24	86.13	92.53
	2008-01-09	94.11	97.10	93.13	95.63
stock_sh_601318	2008-01-07	103.92	103.92	100.46	102.81
	2008-01-08	103.22	107.76	103.22	106.51
	2008-01-09	105.92	107.52	105.22	106.63

calculate with others in specific time or intervals

calculate oneself in time intervals

Why use this format?
The reason is that human is comfortable for two-dimensional space and high-dimensional space could be reduced to it.

Obviously, It's complete and consistent. You could **calculate oneself in time intervals** or **calculate with others in specific time or intervals**. And it's easy to analyze the data with charts by Intent.

Factor data structure

The factor computing flow is as below:

- data_df

NormalData df read from the schema.

- factor_df

NormalData df computed by data_df using use `Transformer`, `Accumulator` or your custom logic.

- result_df

NormalData df containing columns **filter_result** and(or) **score_result** which calculated using factor_df or(and) data_df. Filter_result is True or False, score_result is from 0 to 1. You could use TargetSelector to select targets at specific time when filter_result is True and(or) score_result >=0.8 by default or do more precise control by setting other arguments.

Let's take BullFactor to illustrate the calculation process:

```
>>> from zvt.factors.macd import BullFactor
>>> from zvt.domain import Stock1dHfqKdata
>>> entity_ids = ["stock_sh_601318", "stock_sz_000338"]
>>> Stock1dHfqKdata.record_data(entity_ids=entity_ids,
provider="em")
>>> factor = BullFactor(entity_ids=entity_ids, provider="em",
start_timestamp='2019-01-01', end_timestamp='2019-06-10')
```

check the dfs:

```
>>> factor.data_df
>>> factor.factor_df
>>> factor.result_df
```

Write transformer

Transformer works as bellow:

entity_id	timestamp	open	high	low	close	ma5	ma10
stock_sz_000338	2008-01-07	86.64	87.08	85.60	85.74	85.74	85.74
	2008-01-08	86.13	94.24	86.13	92.53	92.53	92.53
	2008-01-09	94.11	97.10	93.13	95.63	95.63	95.63
stock_sh_601318	2008-01-07	103.92	103.92	100.46	102.81	102.81	102.81
	2008-01-08	103.22	107.76	103.22	106.51	106.51	106.51
	2008-01-09	105.92	107.52	105.22	106.63	106.63	106.63

What's in your mind is the NormalData format, and then practice the skills to manipulate it.

You could use other ta lib with it easily, e.g Bollinger Bands using [TA lib](https://github.com/bukosabino/ta#) [https://github.com/bukosabino/ta#]

install ta at first:

```
pip install --upgrade ta
```

write boll transformer and factor:

```
from typing import Optional, List

import pandas as pd
from ta.volatility import BollingerBands

from zvt.contract.factor import *
from zvt.factors import TechnicalFactor
```

```
class BollTransformer(Transformer):
    def transform_one(self, entity_id, df: pd.DataFrame) ->
pd.DataFrame:
        indicator_bb = BollingerBands(close=df["close"],
window=20, window_dev=2)

        # Add Bollinger Bands features
        df["bb_bbm"] = indicator_bb.bollinger_mavg()
        df["bb_bbh"] = indicator_bb.bollinger_hband()
        df["bb_bbl"] = indicator_bb.bollinger_lband()

        # Add Bollinger Band high indicator
        df["bb_bbhi"] =
indicator_bb.bollinger_hband_indicator()

        # Add Bollinger Band low indicator
        df["bb_bbli"] =
indicator_bb.bollinger_lband_indicator()

        # Add Width Size Bollinger Bands
        df["bb_bbw"] = indicator_bb.bollinger_wband()

        # Add Percentage Bollinger Bands
        df["bb_bbp"] = indicator_bb.bollinger_pband()
        return df


class BollFactor(TechnicalFactor):
    transformer = BollTransformer()

    def drawer_factor_df_list(self) ->
Optional[List[pd.DataFrame]]:
        # set the factor to show
        return [self.factor_df[["bb_bbm", "bb_bbh", "bb_bbl"]]]

    def compute_result(self):
        # set filter_result, which bb_bbli=1.0 buy and
bb_bbhi=1.0 sell
        super().compute_result()
        self.result_df = (self.factor_df["bb_bbli"] -
self.factor_df["bb_bbhi"]).to_frame(name="filter_result")
        self.result_df[self.result_df == 0] = None
```

```
        self.result_df[self.result_df == 1] = True
        self.result_df[self.result_df == -1] = False
```

Let’s show it:

```
>>> from zvt.domain import Stock1dHfqKdata

>>> provider = "em"
>>> entity_ids = ["stock_sz_000338", "stock_sh_601318"]
>>> Stock1dHfqKdata.record_data(entity_ids=entity_ids,
provider=provider,)
>>> factor = BollFactor(
>>> entity_ids=entity_ids, provider=provider,
entity_provider=provider, start_timestamp="2019-01-01"
>>> )
>>> factor.draw(show=True)
```

Most of ta lib support calculate single target by default, so we implement transform_one of the Transformer. If you want to calculate many targets at the same time you could implement transform directly and it would be faster.

And Transformer is stateless, so it’s easy to reuse in different factor if need.

Factor with IntervalLevel

After you write the transformer and construct the factor, it's easy to use in different levels.

Let's use BollFactor in IntervalLevel 30m:

```
>>> from zvt.domain import Stock30mHfqKdata

>>> provider = "em"
>>> entity_ids = ["stock_sz_000338", "stock_sh_601318"]

>>> Stock30mHfqKdata.record_data(entity_ids=entity_ids,
provider=provider)
>>> factor = BollFactor(
        entity_ids=entity_ids, provider=provider,
entity_provider=provider, start_timestamp="2021-01-01"
    )
>>> factor.draw(show=True)
```

Stream computing

The data is coming continuously and the factor using the data need computing continuously too.

It's simple and straightforward:

- {Schema}.record_data in one process
- {Factor}.move_on which call {Schema}.query_data in another process

We keep the simple enough philosophy: single process and thread. Enjoy programming and make everything clear.

Factor persistence

Getting data and computing factor continuously is cool. But…If It took a long time to calculate the factor and crashed.How would you feel?

Select the targets

You could select the targets according result_df of the factor by yourself. Or use TargetSelector do it for you.

entity_id	timestamp	filter_reult	score_result
stock_sz_000338	2008-01-06	True	0.5
	2008-01-07	True	0.4
	2008-01-08	True	0.1
stock_sh_601318	2008-01-06	True	0.9
	2008-01-07	True	0.6
	2008-01-08	True	0.3

entity_id	timestamp	filter_reult	score_result
stock_sz_000338	2008-01-06	True	0.5
	2008-01-07	True	0.4
	2008-01-08	True	0.1
stock_sh_601318	2008-01-06	True	0.9
	2008-01-07	True	0.6
	2008-01-08	True	0.3

Write accumulator

entity_id	timestamp	open	high	low	close	ma5	ma10	count
stock_sz_000338	2008-01-07	86.64	87.08	85.60	85.74	85.74	85.74	1
	2008-01-08	86.13	94.24	86.13	92.53	92.53	92.53	2
	2008-01-09	94.11	97.10	93.13	95.63	95.63	95.63	1
stock_sh_601318	2008-01-07	103.92	103.92	100.46	102.81	102.81	102.81	2
	2008-01-08	103.22	107.76	103.22	106.51	106.51	106.51	3
	2008-01-09	105.92	107.52	105.22	106.63	106.63	106.63	4

Trader

What we want in the end is to trade and make money.
So, what exactly is Trader?

> Trader is a simple facility helping you backtesting and generating trading signals using Factor, TargetSelector, MLMachine or you custom free style algorithm.

- Trader
  - Trader
  - TradingSignal

Trader

Trader

the backtest engine using TargetSelector, MLMachine or free style.

TradingSignal

the signal contains information about how to trade.

Machine learning

In today's world, you're not fashion without machine learning.
So, what could zvt do with ML?
Being a fashion-feeling person, zvt love ML in this way:

> zvt provide a simple facility to define X and y. you could adapt any sophisticated ML library to zvt.

Tagger

`Tagger` is a computing facility for classifying TradableEntity by different dimensions as tag which could be used as ml category feature.

People are divided into groups, things are grouped together.

The most important part for trading is which category is suitable to trade now. ML could help you find which category perform better in different cases ——— deterministic factor.

The tag could be **priori** or calculated from characteristics of the entity ——— some factor.

MLMachine

the ml engine.

Drawer

A picture is worth a thousand words.
So, what exactly is behind the picture?
it's drawer in zvt:

> Drawer is who can draw the picture with NormalData by Intent.

- Drawer concepts
  - Intent
  - Compare

Drawer concepts

Intent

There are many kinds of charts, but the intent is much less. zvt classify intent as compare, distribute and composite.

We could analyze the entity itself or look at it in the whole market by comparing with others. NormalData uses different structure to express different intents.

Compare

Based on zvt's world view and data model, you could compare anything you care about.

Think about that you want to compare S&P 500 index with Shanghai index. It's easy and comfortable with no pain.

```
>>> from zvt.api.intent import compare
>>> from zvt.domain import Indexus1dKdata, Index, Indexus,
Index1dKdata
>>> Index.record_data(provider="em")
>>> Indexus.record_data(provider="em")
>>> Index1dKdata.record_data(entity_id="index_sh_000001",
provider="em")
>>> Indexus1dKdata.record_data(entity_id="indexus_us_SPX",
provider="em")
>>> compare(entity_ids=["index_sh_000001", "indexus_us_SPX"],
start_timestamp="2000-01-01", scale_value=100)
```

Try some other style:

```
>>> compare(entity_ids=["index_sh_000001", "indexus_us_SPX"],
start_timestamp="2000-01-01",  schema_map_columns=
{Index1dKdata:["close"],Indexus1dKdata:["close"]})
```

Contributor

Thanks, my friends.

- 隋鹏飞
- Tiger [https://github.com/Tigerno1]
- 蓝小球
- 青玄
- *崇
- *阳
- 张凯
- Rubicon_陈小虾
- 看门大爷 [https://github.com/Mr-later]
- 芝华
- 雨多田光
- 叶金荣
- 冷冷
- Landy
- Sunny
- Ccc国毅
- Jim Tong
- tom
- 小七
- 小强
- Cascara Latte- 刘可烁

API

zvt

zvt.api

zvt.autocode

zvt.consts

zvt.contract

zvt.domain

zvt.factors

zvt.fill_project

zvt.informer

zvt.main

zvt.ml

zvt.plugin

zvt.recorders

zvt.samples

zvt.tag

zvt.trader

zvt.ui

zvt.utils

zvt

Functions

init_config([pkg_name, current_config])	init config
init_env(zvt_home, **kwargs)	init env
init_log([file_name, log_dir, simple_formatter])	
init_plugins()	
init_resources(resource_path)	
old_db_to_provider_dir(data_path)	

Modules

zvt.api

zvt.autocode

zvt.broker

zvt.common

zvt.consts

zvt.contract

zvt.domain

zvt.factors

zvt.fill_project

zvt.informer

zvt.main

zvt.ml

zvt.plugin

zvt.recorders

zvt.rest

zvt.samples

zvt.sched

zvt.tag

zvt.trader

zvt.trading

zvt.ui

zvt.utils

zvt.zhdate

zvt.init_config

zvt.init_config(*pkg_name: Optional[str] = None*, *current_config: Optional[dict] = None*, ***kwargs*) → dict

init config

zvt.init_env

zvt.init_env(*zvt_home: str*, ***kwargs*) → dict

init env

Parameters:: **zvt_home** – home path for zvt

zvt.init_log

zvt.init_log(*file_name='zvt.log'*, *log_dir=None*, *simple_formatter=True*)

zvt.init_plugins

zvt.init_plugins()

zvt.init_resources

zvt.init_resources(*resource_path*)

zvt.old_db_to_provider_dir

zvt.old_db_to_provider_dir(*data_path*)

zvt.api

Modules

zvt.api.intent

---

zvt.api.kdata

---

zvt.api.portfolio

---

zvt.api.selector

---

zvt.api.stats

---

zvt.api.utils

---

zvt.api.intent

Functions

compare([entity_ids, codes, schema, ...])	compare indicators(columns) of entities
compare_df(df[, chart_type])	compare indicators(columns) of entities in df
composite(entity_id, data_schema, columns[, ...])	composite indicators(columns) of entity
composite_all(data_schema, column, timestamp)	
composite_df(df)	composite indicators(columns) of entity in df
distribute(data_schema, columns[, ...])	distribute indicators(columns) of entities
distribute_df(df[, histnorm, nbinsx])	distribute indicators(columns) of entities in df

zvt.api.intent.compare

zvt.api.intent.compare(*entity_ids=None*, *codes=None*, *schema=None*, *columns=None*, *schema_map_columns: Optional[dict] = None*, *chart_type: ChartType = ChartType.line*, *start_timestamp=None*, *scale_value: Optional[int] = None*)

compare indicators(columns) of entities

Parameters::

- **entity_ids** –
- **codes** –
- **schema** –
- **columns** –
- **schema_map_columns** – key represents schema, value represents columns
- **chart_type** – “line”, “area”, “scatter”, default “line”
- **start_timestamp** – ”
- **scale_value** – compare with same value which scaled to scale_value

zvt.api.intent.compare_df

zvt.api.intent.compare_df(*df: DataFrame*, *chart_type: ChartType = ChartType.line*)

compare indicators(columns) of entities in df

Parameters::

- **df** – normal df
- **chart_type** –

zvt.api.intent.composite

zvt.api.intent.composite(*entity_id*, *data_schema*, *columns*, *filters=None*)

composite indicators(columns) of entity

Parameters::

- **entity_id** –
- **data_schema** –
- **columns** –
- **filters** –

zvt.api.intent.composite_all

zvt.api.intent.composite_all(*data_schema*, *column*, *timestamp*, *provider=None*, *entity_ids=None*, *filters=None*)

zvt.api.intent.composite_df

zvt.api.intent.composite_df(*df*)

composite indicators(columns) of entity in df

Parameters:: **df** –

zvt.api.intent.distribute

zvt.api.intent.distribute(*data_schema*, *columns*, *entity_ids=None*, *codes=None*, *histnorm='percent'*, *nbinsx=20*, *filters=None*)

distribute indicators(columns) of entities

Parameters::

- **data_schema** –
- **columns** –
- **entity_ids** –
- **codes** –
- **histnorm** – “percent”, “probability”, default “percent”
- **nbinsx** –
- **filters** –

zvt.api.intent.distribute_df

zvt.api.intent.distribute_df(*df*, *histnorm='percent'*, *nbinsx=20*)

distribute indicators(columns) of entities in df

Parameters::

- **df** – normal df
- **histnorm** – “percent”, “probability”, default “percent”
- **nbinsx** –

zvt.api.kdata

Functions

default_adjust_type(entity_type)	**type entity_type::** entity type, e.g stock, stockhk, stockus
generate_kdata_id(entity_id, timestamp, level)	
get_kdata([entity_id, entity_ids, level, ...])	
get_kdata_schema(entity_type[, level, ...])	
get_latest_kdata_date(entity_type[, ...])	
get_trade_dates(start, end)	
to_high_level_kdata(kdata_df, to_level)	

zvt.api.kdata.default_adjust_type

zvt.api.kdata.default_adjust_type(*entity_type: str*) → AdjustType

zvt.api.kdata.generate_kdata_id

zvt.api.kdata.generate_kdata_id(*entity_id*, *timestamp*, *level*)

zvt.api.kdata.get_kdata

zvt.api.kdata.get_kdata(*entity_id=None*, *entity_ids=None*, *level='1d'*, *provider=None*, *columns=None*, *return_type='df'*, *start_timestamp=None*, *end_timestamp=None*, *filters=None*, *session=None*, *order=None*, *limit=None*, *index='timestamp'*, *drop_index_col=False*, *adjust_type: Optional[AdjustType] = None*)

zvt.api.kdata.get_kdata_schema

zvt.api.kdata.get_kdata_schema(*entity_type: str*, *level: Union[IntervalLevel, str] = IntervalLevel.LEVEL_1DAY*, *adjust_type: Optional[Union[AdjustType, str]] = None*) → Mixin

zvt.api.kdata.get_latest_kdata_date

zvt.api.kdata.get_latest_kdata_date(*entity_type: str*, *provider: Optional[str] = None*, *level: Union[IntervalLevel, str] = IntervalLevel.LEVEL_1DAY*, *adjust_type: Optional[Union[AdjustType, str]] = None*) → Timestamp

zvt.api.kdata.get_trade_dates

zvt.api.kdata.get_trade_dates(*start*, *end*)

zvt.api.kdata.to_high_level_kdata

zvt.api.kdata.to_high_level_kdata(*kdata_df: DataFrame*, *to_level: IntervalLevel*)

zvt.api.portfolio

Functions

`get_etf_stocks`([code, codes, ids, ...])

`get_fund_stocks`([code, codes, ids, ...])

`get_portfolio_stocks`([portfolio_entity, ...])

`portfolio_relate_stock`(df, portfolio)

---

zvt.api.portfolio.get_etf_stocks

zvt.api.portfolio.get_etf_stocks(*code=None*, *codes=None*, *ids=None*, *timestamp=Timestamp('2024-08-28 22:08:12.449667')*, *provider=None*)

zvt.api.portfolio.get_fund_stocks

zvt.api.portfolio.get_fund_stocks(*code=None*, *codes=None*, *ids=None*, *timestamp=Timestamp('2024-08-28 22:08:12.449669')*, *provider=None*)

zvt.api.portfolio.get_portfolio_stocks

zvt.api.portfolio.get_portfolio_stocks(*portfolio_entity=<class 'zvt.domain.meta.fund_meta.Fund'>*, *code=None*, *codes=None*, *ids=None*, *timestamp=Timestamp('2024-08-28 22:08:12.449665')*, *provider=None*)

zvt.api.portfolio.portfolio_relate_stock

zvt.api.portfolio.portfolio_relate_stock(*df*, *portfolio*)

zvt.api.selector

Functions

get_big_cap_stock(timestamp[, provider])

---

get_big_players(start_timestamp[, ...])

---

get_dragon_and_tigger_player(start_timestamp)

---

get_entity_ids_by_filter([provider, ...])

---

get_entity_list_by_cap(timestamp, cap_start, ...)

---

get_good_players([timestamp, recent_days, ...])

---

get_limit_up_stocks(timestamp)

---

get_middle_and_big_stock(timestamp[, provider])

---

get_middle_cap_stock(timestamp[, provider])

---

get_mini_and_small_stock(timestamp[, provider])

---

get_mini_cap_stock(timestamp[, provider])

`get_player_performance`(start_timestamp[, ...])

`get_player_success_rate`(start_timestamp[, ...])

`get_players`(entity_id, start_timestamp, ...)

`get_small_cap_stock`(timestamp[, provider])

---

zvt.api.selector.get_big_cap_stock

zvt.api.selector.get_big_cap_stock(*timestamp*, *provider='em'*)

zvt.api.selector.get_big_players

zvt.api.selector.get_big_players(*start_timestamp*, *end_timestamp=None*, *count=40*)

zvt.api.selector.get_dragon_and_tigger_player

zvt.api.selector.get_dragon_and_tigger_player(*start_timestamp*, *end_timestamp=None*, *direction='in'*)

zvt.api.selector.get_entity_ids_by_filter

zvt.api.selector.get_entity_ids_by_filter(*provider='em'*, *ignore_delist=True*, *ignore_st=True*, *ignore_new_stock=False*, *target_date=None*, *entity_schema=<class 'zvt.domain.meta.stock_meta.Stock'>*, *entity_ids=None*, *ignore_bj=False*)

zvt.api.selector.get_entity_list_by_cap

zvt.api.selector.get_entity_list_by_cap(*timestamp*, *cap_start*, *cap_end*, *entity_type='stock'*, *provider=None*, *adjust_type=None*, *retry_times=20*)

zvt.api.selector.get_good_players

zvt.api.selector.get_good_players(*timestamp=Timestamp('2024-08-28 00:00:00')*, *recent_days=400*, *intervals=(3, 5, 10)*)

zvt.api.selector.get_limit_up_stocks

zvt.api.selector.get_limit_up_stocks(*timestamp*)

zvt.api.selector.get_middle_and_big_stock

zvt.api.selector.get_middle_and_big_stock(*timestamp*, *provider='em'*)

zvt.api.selector.get_middle_cap_stock

zvt.api.selector.get_middle_cap_stock(*timestamp*, *provider='em'*)

zvt.api.selector.get_mini_and_small_stock

zvt.api.selector.get_mini_and_small_stock(*timestamp*, *provider='em'*)

zvt.api.selector.get_mini_cap_stock

zvt.api.selector.get_mini_cap_stock(*timestamp*, *provider='em'*)

zvt.api.selector.get_player_performance

zvt.api.selector.get_player_performance(*start_timestamp*, *end_timestamp=None*, *days=5*, *players='机构专用'*, *provider='em'*, *buy_rate=5*)

zvt.api.selector.get_player_success_rate

zvt.api.selector.get_player_success_rate(*start_timestamp*, *end_timestamp=None*, *intervals=(3, 5, 10, 60)*, *players=('机构专用', '东方财富证券股份有限公司拉萨团结路第二证券营业部')*, *provider='em'*)

zvt.api.selector.get_players

zvt.api.selector.get_players(*entity_id*, *start_timestamp*, *end_timestamp*, *provider='em'*, *direction='in'*, *buy_rate=5*)

zvt.api.selector.get_small_cap_stock

zvt.api.selector.get_small_cap_stock(*timestamp*, *provider='em'*)

zvt.api.stats

Functions

get_change_ratio([entity_type, ...])	
get_performance(entity_ids[, ...])	
get_performance_stats([entity_type, ...])	
get_performance_stats_by_month([...])	
get_top_entities(data_schema, column[, ...])	get top entities in specific domain between time range
get_top_fund_holding_stocks([timestamp, pct, by])	
get_top_performance_by_month([entity_type, ...])	
get_top_performance_entities([entity_type, ...])	
get_top_performance_entities_by_periods(...)	
get_top_turnover_rate_entities([...])	

`get_top_volume_entities`([entity_type, ...])	
`show_industry_composition`(entity_ids, timestamp)	
`show_month_performance`()	

Classes

`TopType`(value)	An enumeration.
`WindowMethod`(value)	An enumeration.

zvt.api.stats.get_change_ratio

zvt.api.stats.get_change_ratio(*entity_type='stock'*, *start_timestamp=None*, *end_timestamp=None*, *adjust_type: Optional[Union[AdjustType, str]] = None*, *provider='em'*)

zvt.api.stats.get_performance

zvt.api.stats.get_performance(*entity_ids*, *start_timestamp=None*, *end_timestamp=None*, *adjust_type: Optional[Union[AdjustType, str]] = None*, *data_provider=None*)

zvt.api.stats.get_performance_stats

zvt.api.stats.get_performance_stats(*entity_type='stock', start_timestamp=None, end_timestamp=None, adjust_type: Optional[Union[AdjustType, str]] = None, data_provider=None, changes= ((- 1, - 0.5), (- 0.5, - 0.2), (- 0.2, 0), (0, 0.2), (0.2, 0.5), (0.5, 1), (1, 1000))*)

zvt.api.stats.get_performance_stats_by_month

zvt.api.stats.get_performance_stats_by_month(*entity_type='stock', start_timestamp='2015-01-01', end_timestamp=Timestamp('2024-08-28 22:08:13.917869'), adjust_type: Optional[Union[AdjustType, str]] = None, data_provider=None*)

zvt.api.stats.get_top_entities

zvt.api.stats.get_top_entities(*data_schema: Mixin*, *column: str*, *start_timestamp=None*, *end_timestamp=None*, *pct=0.1*, *method: WindowMethod = WindowMethod.change*, *return_type: Optional[TopType] = None*, *kdata_filters=None*, *show_name=False*, *data_provider=None*)

get top entities in specific domain between time range

Parameters::

- **data_schema** – schema in domain
- **column** – schema column
- **start_timestamp** –
- **end_timestamp** –
- **pct** – range (0,1]
- **method** –
- **return_type** –
- **entity_filters** –
- **kdata_filters** –
- **show_name** – show entity name

Returns::

zvt.api.stats.get_top_fund_holding_stocks

zvt.api.stats.get_top_fund_holding_stocks(*timestamp=None*, *pct=0.3*, *by=None*)

zvt.api.stats.get_top_performance_by_month

zvt.api.stats.get_top_performance_by_month(*entity_type='stock'*, *start_timestamp='2015-01-01'*, *end_timestamp=Timestamp('2024-08-28 22:08:13.917847')*, *list_days=None*, *data_provider=None*)

zvt.api.stats.get_top_performance_entities

zvt.api.stats.get_top_performance_entities(*entity_type='stock', start_timestamp=None, end_timestamp=None, pct=0.1, return_type=None, adjust_type: Optional[Union[AdjustType, str]] = None, entity_filters=None, kdata_filters=None, show_name=False, list_days=None, entity_provider=None, data_provider=None*)

zvt.api.stats.get_top_performance_entities_by_periods

zvt.api.stats.get_top_performance_entities_by_periods(*entity_provider*, *data_provider*, *target_date=None*, *periods=None*, *ignore_new_stock=True*, *ignore_st=True*, *entity_ids=None*, *entity_type='stock'*, *adjust_type=None*, *top_count=50*, *turnover_threshold=100000000*, *turnover_rate_threshold=0.02*, *return_type=TopType.positive*)

zvt.api.stats.get_top_turnover_rate_entities

zvt.api.stats.get_top_turnover_rate_entities(*entity_type='stock'*, *entity_ids=None*, *start_timestamp=None*, *end_timestamp=None*, *pct=0.1*, *return_type=TopType.positive*, *adjust_type: Optional[Union[AdjustType, str]] = None*, *method=WindowMethod.avg*, *data_provider=None*, *threshold=None*)

zvt.api.stats.get_top_volume_entities

zvt.api.stats.get_top_volume_entities(*entity_type='stock'*, *entity_ids=None*, *start_timestamp=None*, *end_timestamp=None*, *pct=0.1*, *return_type=TopType.positive*, *adjust_type: Optional[Union[AdjustType, str]] = None*, *method=WindowMethod.avg*, *data_provider=None*, *threshold=None*)

zvt.api.stats.show_industry_composition

zvt.api.stats.show_industry_composition(*entity_ids*, *timestamp*)

zvt.api.stats.show_month_performance

zvt.api.stats.show_month_performance()

TopType

class zvt.api.stats.TopType(*value*)

Bases: `Enum`

An enumeration.

__init__()

WindowMethod

class zvt.api.stats.WindowMethod(*value*)

Bases: `Enum`

An enumeration.

__init__()

zvt.api.utils

Functions

china_stock_code_to_id(code)

---

float_to_pct_str(value)

---

get_china_exchange(code)

---

get_recent_report(data_schema, timestamp[, ...])

---

get_recent_report_date([the_date, step])

---

get_recent_report_period([the_date, step])

---

to_report_period_type(report_date)

---

value_multiply(value, multiplier[, default])

---

value_to_pct(value[, default])

---

zvt.api.utils.china_stock_code_to_id

zvt.api.utils.china_stock_code_to_id(*code*)

zvt.api.utils.float_to_pct_str

zvt.api.utils.float_to_pct_str(*value*)

zvt.api.utils.get_china_exchange

zvt.api.utils.get_china_exchange(*code*)

zvt.api.utils.get_recent_report

zvt.api.utils.get_recent_report(*data_schema: Type[Mixin]*, *timestamp*, *entity_id=None*, *filters=None*, *max_step=2*)

zvt.api.utils.get_recent_report_date

zvt.api.utils.get_recent_report_date(*the_date=Timestamp('2024-08-28 22:08:12.449611')*, *step=0*)

zvt.api.utils.get_recent_report_period

zvt.api.utils.get_recent_report_period(*the_date=Timestamp('2024-08-28 22:08:12.449632')*, *step=0*)

zvt.api.utils.to_report_period_type

zvt.api.utils.to_report_period_type(*report_date*)

zvt.api.utils.value_multiply

zvt.api.utils.value_multiply(*value*, *multiplier*, *default=0*)

zvt.api.utils.value_to_pct

zvt.api.utils.value_to_pct(*value*, *default=0*)

zvt.autocode

Modules

zvt.autocode.generator

---

zvt.autocode.templates

---

zvt.autocode.generator

Functions

`all_sub_all`(sub_module)	
`all_sub_modules`(dir_path)	list all module name in specific directory
`fill_package`(dir_path)	
`fill_package_if_not_exist`(dir_path)	
`gen_exports`([dir_path, gen_flag, ...])	
`gen_kdata_schema`(pkg, providers, ...[, ...])	
`gen_plugin_project`(entity_type[, prefix, ...])	generate a standard plugin project

zvt.autocode.generator.all_sub_all

zvt.autocode.generator.all_sub_all(*sub_module*)

zvt.autocode.generator.all_sub_modules

zvt.autocode.generator.all_sub_modules(*dir_path: str*)

list all module name in specific directory

Parameters:: **dir_path** –

Returns::

zvt.autocode.generator.fill_package

zvt.autocode.generator.fill_package(*dir_path: str*)

zvt.autocode.generator.fill_package_if_not_exist

zvt.autocode.generator.fill_package_if_not_exist(*dir_path: str*)

zvt.autocode.generator.gen_exports

zvt.autocode.generator.gen_exports(*dir_path='./domain'*, *gen_flag='# the __all__ is generated'*, *export_from_package=False*, *export_modules=None*, *excludes=None*, *export_var=False*)

zvt.autocode.generator.gen_kdata_schema

zvt.autocode.generator.gen_kdata_schema(*pkg: str*, *providers: List[str]*, *entity_type: str*, *levels: List[IntervalLevel]*, *adjust_types=None*, *entity_in_submodule: bool = False*, *kdata_module='quotes'*)

zvt.autocode.generator.gen_plugin_project

zvt.autocode.generator.gen_plugin_project(*entity_type*, *prefix: str = 'zvt'*, *dir_path: str = '.'*, *providers=['joinquant']*)

generate a standard plugin project

Parameters::

- **entity_type** – the entity type of the plugin project
- **prefix** – project prefix
- **dir_path** – the root path for the project
- **providers** – the supported providers

zvt.autocode.templates

Functions

`all_tpls`(project, entity_type) return list of templates(location,Template)

---

zvt.autocode.templates.all_tpls

zvt.autocode.templates.all_tpls(*project: str*, *entity_type: str*)

return list of templates(location,Template)

Parameters:: **project** –

Returns::

zvt.broker

Modules

zvt.broker.qmt

---

zvt.broker.qmt

Modules

zvt.broker.qmt.errors

---

zvt.broker.qmt.errors

Exceptions

PositionOverflowError([message])	
QmtError([message])	
TraderError	Base class for exceptions in this module.

zvt.broker.qmt.errors.PositionOverflowError

exception zvt.broker.qmt.errors.PositionOverflowError(*message='超出仓位限制'*)

zvt.broker.qmt.errors.QmtError

exception zvt.broker.qmt.errors.QmtError(*message='qmt error'*)

zvt.broker.qmt.errors.TraderError

exception zvt.broker.qmt.errors.TraderError

Base class for exceptions in this module.

zvt.common

Modules

zvt.common.query_models

---

zvt.common.trading_models

---

zvt.common.query_models

Classes

`AbsoluteTimeRange`(*, start_timestamp, ...)	
`OrderByType`(value)	An enumeration.
`RelativeTimeRage`(*, interval, time_unit)	
`TimeRange`(*[, absolute_time_range, ...])	
`TimeUnit`(value)	An enumeration.

AbsoluteTimeRange

class zvt.common.query_models.AbsoluteTimeRange(*, *start_timestamp: datetime*, *end_timestamp: datetime*)

Bases: **`BaseModel`**

__init__(**data: Any*) → None

Create a new model by parsing and validating input data from keyword arguments.

Raises [*ValidationError*][pydantic_core.ValidationError] if the input data cannot be validated to form a valid model.

self is explicitly positional-only to allow *self* as a field name.

copy(*, *include: AbstractSetIntStr | MappingIntStrAny | None = None*, *exclude: AbstractSetIntStr | MappingIntStrAny | None = None*, *update: Dict[str, Any] | None = None*, *deep: bool = False*) → Model

Returns a copy of the model.

!!! warning "Deprecated"

This method is now deprecated; use *model_copy* instead.

If you need *include* or *exclude*, use:

```
`py data = self.model_dump(include=include, exclude=exclude, round_trip=True) data = {**data, **(update or {})} copied = self.model_validate(data) `
```

Args:

include: Optional set or mapping specifying which fields to include in the copied model. exclude: Optional set or mapping specifying which fields to exclude in the copied model. update: Optional dictionary of field-value pairs to override field values

in the copied model. deep: If True, the values of fields that are Pydantic models will be deep-copied.

Returns:

A copy of the model with included, excluded and updated fields as specified.

model_computed_fields*: ClassVar[dict[str, ComputedFieldInfo]] = {}*

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

model_config*: ClassVar[ConfigDict] = {}*

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

classmethod model_construct(*_fields_set: set[str] | None = None*, ***values: Any*) → Model

Creates a new instance of the *Model* class with validated data.

Creates a new model setting *__dict__* and *__pydantic_fields_set__* from trusted or pre-validated data. Default values are respected, but no other validation is performed. Behaves as if *Config.extra = 'allow'* was set since it adds all passed values

Args:

_fields_set: The set of field names accepted for the Model instance. values: Trusted or pre-validated data dictionary.

Returns:

A new instance of the *Model* class with validated data.

model_copy(*, *update: dict[str, Any] | None = None*, *deep: bool = False*) → Model

Usage docs: https://docs.pydantic.dev/2.6/concepts/serialization/#model_copy

Returns a copy of the model.

Args:

update: Values to change/add in the new model. Note: the data is not validated
before creating the new model. You should trust this data.

deep: Set to *True* to make a deep copy of the model.

Returns:

New model instance.

model_dump(*, *mode: Literal['json', 'python'] | str = 'python'*, *include: IncEx = None*, *exclude: IncEx = None*, *by_alias: bool = False*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*, *round_trip: bool = False*, *warnings: bool = True*) → dict[str, Any]

Usage docs: https://docs.pydantic.dev/2.6/concepts/serialization/#modelmodel_dump

Generate a dictionary representation of the model, optionally specifying which fields to include or exclude.

Args:

mode: The mode in which *to_python* should run.
If mode is 'json', the output will only contain JSON serializable types. If mode is 'python', the output may contain non-JSON-serializable Python objects.

include: A list of fields to include in the output. exclude: A list of fields to exclude from the output. by_alias: Whether to use the field's alias in the dictionary key if defined. exclude_unset: Whether to exclude fields that have not been explicitly set. exclude_defaults: Whether to exclude fields that are set to their default value. exclude_none: Whether to exclude fields that have a value of *None*. round_trip: If True, dumped values should be valid as input for non-idempotent types such as Json[T].

warnings: Whether to log warnings when invalid fields are encountered.

Returns:

A dictionary representation of the model.

model_dump_json(*, *indent: int | None = None*, *include: IncEx = None*, *exclude: IncEx = None*, *by_alias: bool = False*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*, *round_trip: bool = False*, *warnings: bool = True*) → str

Usage docs: https://docs.pydantic.dev/2.6/concepts/serialization/#modelmodel_dump_json

Generates a JSON representation of the model using Pydantic's *to_json* method.

Args:

indent: Indentation to use in the JSON output. If None is passed, the output will be compact. include: Field(s) to include in the JSON output. exclude: Field(s) to exclude from the JSON output. by_alias: Whether to serialize using field aliases. exclude_unset: Whether to exclude fields that have not been explicitly set. exclude_defaults: Whether to exclude fields that are set to their default value. exclude_none: Whether to exclude fields that have a value of *None*. round_trip: If True, dumped values should be valid as input for non-idempotent types such as Json[T]. warnings: Whether to log warnings when invalid fields are encountered.

Returns:

A JSON string representation of the model.

property model_extra*: dict[str, Any] | None*

Get extra fields set during validation.

Returns:

A dictionary of extra fields, or *None* if *config.extra* is not set to *"allow"*.

model_fields*: ClassVar[dict[str, FieldInfo]] = {'end_timestamp': FieldInfo(annotation=datetime, required=True), 'start_timestamp': FieldInfo(annotation=datetime, required=True)}*

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

property model_fields_set*: set[str]*

Returns the set of fields that have been explicitly set on this model instance.

Returns:

A set of strings representing the fields that have been set, i.e. that were not filled from defaults.

classmethod model_json_schema(*by_alias: bool = True*, *ref_template: str = '#/$defs/{model}'*, *schema_generator: type[pydantic.json_schema.GenerateJsonSchema] = <class 'pydantic.json_schema.GenerateJsonSchema'>*, *mode: ~typing.Literal['validation', 'serialization'] = 'validation'*) → dict[str, Any]

Generates a JSON schema for a model class.

Args:

by_alias: Whether to use attribute aliases or not. ref_template: The reference template. schema_generator: To override the logic used to generate the JSON schema, as a subclass of

GenerateJsonSchema with your desired modifications

mode: The mode in which to generate the schema.

Returns:

The JSON schema for the given model class.

classmethod model_parametrized_name(*params: tuple[type[Any], ...]*) → str

Compute the class name for parametrizations of generic classes.

This method can be overridden to achieve a custom naming scheme for generic BaseModels.

Args:

params: Tuple of types of the class. Given a generic class *Model* with 2 type variables and a concrete model *Model[str, int]*, the value *(str, int)* would be passed to *params*.

Returns:

String representing the new class where *params* are passed to *cls* as type variables.

Raises:

TypeError: Raised when trying to generate concrete names for non-generic models.

model_post_init(*_BaseModel__context: Any*) → None

Override this method to perform additional initialization after *__init__* and *model_construct*. This is useful if you want to do some validation that requires the entire model to be initialized.

classmethod model_rebuild(*, *force: bool = False*, *raise_errors: bool = True*, *_parent_namespace_depth: int = 2*, *_types_namespace: Optional[dict[str, Any]] = None*) → bool | None

Try to rebuild the pydantic-core schema for the model.

This may be necessary when one of the annotations is a ForwardRef which could not be resolved during the initial attempt to build the schema, and automatic rebuilding fails.

Args:

force: Whether to force the rebuilding of the model schema, defaults to *False*. raise_errors: Whether to raise errors, defaults to *True*. _parent_namespace_depth: The depth level of the parent namespace, defaults to 2. _types_namespace: The types namespace, defaults to *None*.

Returns:
Returns *None* if the schema is already "complete" and rebuilding was not required. If rebuilding _was_ required, returns *True* if rebuilding was successful, otherwise *False*.

classmethod model_validate(*obj: Any*, *, *strict: bool | None = None*, *from_attributes: bool | None = None*, *context: dict[str, Any] | None = None*) → Model

Validate a pydantic model instance.

Args:
obj: The object to validate. strict: Whether to enforce types strictly. from_attributes: Whether to extract data from object attributes. context: Additional context to pass to the validator.

Raises:
ValidationError: If the object could not be validated.

Returns:
The validated model instance.

classmethod model_validate_json(*json_data: str | bytes | bytearray*, *, *strict: bool | None = None*, *context: dict[str, Any] | None = None*) → Model

Usage docs: https://docs.pydantic.dev/2.6/concepts/json/#json-parsing

Validate the given JSON data against the Pydantic model.

Args:

json_data: The JSON data to validate. strict: Whether to enforce types strictly. context: Extra variables to pass to the validator.

Returns:
The validated Pydantic model.

Raises:
ValueError: If *json_data* is not a JSON string.

classmethod model_validate_strings(*obj: Any*, *, *strict: bool | None = None*, *context: dict[str, Any] | None = None*) → Model

Validate the given object contains string data against the Pydantic model.

Args:
obj: The object contains string data to validate. strict: Whether to enforce types strictly. context: Extra variables to pass to the validator.

Returns:
The validated Pydantic model.

OrderByType

class zvt.common.query_models.OrderByType(*value*)

Bases: **`Enum`**

An enumeration.

__init__()

RelativeTimeRage

class zvt.common.query_models.RelativeTimeRage(*, *interval: int*, *time_unit: TimeUnit*)

Bases: **BaseModel**

__init__(**data: Any*) → None

Create a new model by parsing and validating input data from keyword arguments.

Raises [*ValidationError*][pydantic_core.ValidationError] if the input data cannot be validated to form a valid model.

self is explicitly positional-only to allow *self* as a field name.

copy(*, *include: AbstractSetIntStr | MappingIntStrAny | None = None*, *exclude: AbstractSetIntStr | MappingIntStrAny | None = None*, *update: Dict[str, Any] | None = None*, *deep: bool = False*) → Model

Returns a copy of the model.

!!! warning "Deprecated"

This method is now deprecated; use *model_copy* instead.

If you need *include* or *exclude*, use:

```
`py data = self.model_dump(include=include, exclude=exclude, round_trip=True) data = {**data, **(update or {})} copied = self.model_validate(data) `
```

Args:

include: Optional set or mapping specifying which fields to include in the copied model. exclude: Optional set or mapping specifying which fields to exclude in the copied model. update: Optional dictionary of field-value pairs to override field values

in the copied model. deep: If True, the values of fields that are Pydantic models will be deep-copied.

Returns:

A copy of the model with included, excluded and updated fields as specified.

model_computed_fields*: ClassVar[dict[str, ComputedFieldInfo]] = {}*

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

model_config*: ClassVar[ConfigDict] = {}*

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

classmethod model_construct(*_fields_set: set[str] | None = None*, ***values: Any*) → Model

Creates a new instance of the *Model* class with validated data.

Creates a new model setting *__dict__* and *__pydantic_fields_set__* from trusted or pre-validated data. Default values are respected, but no other validation is performed. Behaves as if *Config.extra = 'allow'* was set since it adds all passed values

Args:

_fields_set: The set of field names accepted for the Model instance. values: Trusted or pre-validated data dictionary.

Returns:

A new instance of the *Model* class with validated data.

model_copy(*, *update: dict[str, Any] | None = None*, *deep: bool = False*) → Model

Usage docs: https://docs.pydantic.dev/2.6/concepts/serialization/#model_copy

Returns a copy of the model.

Args:

update: Values to change/add in the new model. Note: the data is not validated
before creating the new model. You should trust this data.

deep: Set to *True* to make a deep copy of the model.

Returns:

New model instance.

model_dump(*, *mode: Literal['json', 'python'] | str = 'python'*, *include: IncEx = None*, *exclude: IncEx = None*, *by_alias: bool = False*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*, *round_trip: bool = False*, *warnings: bool = True*) → dict[str, Any]

Usage docs: https://docs.pydantic.dev/2.6/concepts/serialization/#modelmodel_dump

Generate a dictionary representation of the model, optionally specifying which fields to include or exclude.

Args:

mode: The mode in which *to_python* should run.
If mode is 'json', the output will only contain JSON serializable types. If mode is 'python', the output may contain non-JSON-serializable Python objects.

include: A list of fields to include in the output. exclude: A list of fields to exclude from the output. by_alias: Whether to use the field's alias in the dictionary key if defined. exclude_unset: Whether to exclude fields that have not been explicitly set. exclude_defaults: Whether to exclude fields that are set to their default value. exclude_none: Whether to exclude fields that have a value of *None*. round_trip: If True, dumped values should be valid as input for non-idempotent types such as Json[T].

warnings: Whether to log warnings when invalid fields are encountered.

Returns:

A dictionary representation of the model.

model_dump_json(*, *indent: int | None = None*, *include: IncEx = None*, *exclude: IncEx = None*, *by_alias: bool = False*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*, *round_trip: bool = False*, *warnings: bool = True*) → str

Usage docs: https://docs.pydantic.dev/2.6/concepts/serialization/#modelmodel_dump_json

Generates a JSON representation of the model using Pydantic's *to_json* method.

Args:

indent: Indentation to use in the JSON output. If None is passed, the output will be compact. include: Field(s) to include in the JSON output. exclude: Field(s) to exclude from the JSON output. by_alias: Whether to serialize using field aliases. exclude_unset: Whether to exclude fields that have not been explicitly set. exclude_defaults: Whether to exclude fields that are set to their default value. exclude_none: Whether to exclude fields that have a value of *None*. round_trip: If True, dumped values should be valid as input for non-idempotent types such as Json[T]. warnings: Whether to log warnings when invalid fields are encountered.

Returns:

A JSON string representation of the model.

property model_extra*: dict[str, Any] | None*

Get extra fields set during validation.

Returns:

A dictionary of extra fields, or *None* if *config.extra* is not set to *"allow"*.

model_fields*: ClassVar[dict[str, FieldInfo]] = {'interval': FieldInfo(annotation=int, required=True), 'time_unit': FieldInfo(annotation=TimeUnit, required=True)}*

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

property model_fields_set*: set[str]*

Returns the set of fields that have been explicitly set on this model instance.

Returns:

A set of strings representing the fields that have been set, i.e. that were not filled from defaults.

classmethod model_json_schema(*by_alias: bool = True*, *ref_template: str = '#/$defs/{model}'*, *schema_generator: type[pydantic.json_schema.GenerateJsonSchema] = <class 'pydantic.json_schema.GenerateJsonSchema'>*, *mode: ~typing.Literal['validation', 'serialization'] = 'validation'*) → dict[str, Any]

Generates a JSON schema for a model class.

Args:

by_alias: Whether to use attribute aliases or not. ref_template: The reference template. schema_generator: To override the logic used to generate the JSON schema, as a subclass of

GenerateJsonSchema with your desired modifications

mode: The mode in which to generate the schema.

Returns:

The JSON schema for the given model class.

classmethod model_parametrized_name(*params: tuple[type[Any], ...]*) → str

Compute the class name for parametrizations of generic classes.

This method can be overridden to achieve a custom naming scheme for generic BaseModels.

Args:

params: Tuple of types of the class. Given a generic class *Model* with 2 type variables and a concrete model *Model[str, int]*, the value *(str, int)* would be passed to *params*.

Returns:

String representing the new class where *params* are passed to *cls* as type variables.

Raises:

TypeError: Raised when trying to generate concrete names for non-generic models.

model_post_init(*_BaseModel__context: Any*) → None

Override this method to perform additional initialization after *__init__* and *model_construct*. This is useful if you want to do some validation that requires the entire model to be initialized.

classmethod model_rebuild(*, *force: bool = False*, *raise_errors: bool = True*, *_parent_namespace_depth: int = 2*, *_types_namespace: Optional[dict[str, Any]] = None*) → bool | None

Try to rebuild the pydantic-core schema for the model.

This may be necessary when one of the annotations is a ForwardRef which could not be resolved during the initial attempt to build the schema, and automatic rebuilding fails.

Args:

force: Whether to force the rebuilding of the model schema, defaults to *False*. raise_errors: Whether to raise errors, defaults to *True*. _parent_namespace_depth: The depth level of the parent namespace, defaults to 2. _types_namespace: The types namespace, defaults to *None*.

Returns:
Returns *None* if the schema is already "complete" and rebuilding was not required. If rebuilding _was_ required, returns *True* if rebuilding was successful, otherwise *False*.

classmethod model_validate(*obj: Any*, *, *strict: bool | None = None*, *from_attributes: bool | None = None*, *context: dict[str, Any] | None = None*) → Model

Validate a pydantic model instance.

Args:
obj: The object to validate. strict: Whether to enforce types strictly. from_attributes: Whether to extract data from object attributes. context: Additional context to pass to the validator.

Raises:
ValidationError: If the object could not be validated.

Returns:
The validated model instance.

classmethod model_validate_json(*json_data: str | bytes | bytearray*, *, *strict: bool | None = None*, *context: dict[str, Any] | None = None*) → Model

Usage docs: https://docs.pydantic.dev/2.6/concepts/json/#json-parsing

Validate the given JSON data against the Pydantic model.

Args:

json_data: The JSON data to validate. strict: Whether to enforce types strictly. context: Extra variables to pass to the validator.

Returns:
The validated Pydantic model.

Raises:
ValueError: If *json_data* is not a JSON string.

classmethod model_validate_strings(*obj: Any*, *, *strict: bool | None = None*, *context: dict[str, Any] | None = None*) → Model

Validate the given object contains string data against the Pydantic model.

Args:
obj: The object contains string data to validate. strict: Whether to enforce types strictly. context: Extra variables to pass to the validator.

Returns:
The validated Pydantic model.

TimeRange

class zvt.common.query_models.TimeRange(*, *absolute_time_range: Optional[AbsoluteTimeRange] = None*, *relative_time_range: Optional[RelativeTimeRage] = None*)

Bases: **`BaseModel`**

__init__(**data: Any*) → None

Create a new model by parsing and validating input data from keyword arguments.

Raises [*ValidationError*][pydantic_core.ValidationError] if the input data cannot be validated to form a valid model.

self is explicitly positional-only to allow *self* as a field name.

copy(*, *include: AbstractSetIntStr | MappingIntStrAny | None = None*, *exclude: AbstractSetIntStr | MappingIntStrAny | None = None*, *update: Dict[str, Any] | None = None*, *deep: bool = False*) → Model

Returns a copy of the model.

!!! warning "Deprecated"

This method is now deprecated; use *model_copy* instead.

If you need *include* or *exclude*, use:

```
`py data = self.model_dump(include=include, exclude=exclude, round_trip=True) data = {**data, **(update or {})} copied = self.model_validate(data) `
```

Args:

include: Optional set or mapping specifying which fields to include in the copied model. exclude: Optional set or mapping specifying which fields to exclude in the copied model. update:

Optional dictionary of field-value pairs to override field values in the copied model. deep: If True, the values of fields that are Pydantic models will be deep-copied.

Returns:

A copy of the model with included, excluded and updated fields as specified.

model_computed_fields*: ClassVar[dict[str, ComputedFieldInfo]] = {}*

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

model_config*: ClassVar[ConfigDict] = {}*

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

classmethod model_construct(*_fields_set: set[str] | None = None*, ***values: Any*) → Model

Creates a new instance of the *Model* class with validated data.

Creates a new model setting *__dict__* and *__pydantic_fields_set__* from trusted or pre-validated data. Default values are respected, but no other validation is performed. Behaves as if *Config.extra = 'allow'* was set since it adds all passed values

Args:

_fields_set: The set of field names accepted for the Model instance. values: Trusted or pre-validated data dictionary.

Returns:

A new instance of the *Model* class with validated data.

model_copy(*, *update: dict[str, Any] | None = None*, *deep: bool = False*) → Model

Usage docs: https://docs.pydantic.dev/2.6/concepts/serialization/#model_copy

Returns a copy of the model.

Args:

update: Values to change/add in the new model. Note: the data is not validated
before creating the new model. You should trust this data.

deep: Set to *True* to make a deep copy of the model.

Returns:

New model instance.

model_dump(*, *mode: Literal['json', 'python'] | str = 'python'*, *include: IncEx = None*, *exclude: IncEx = None*, *by_alias: bool = False*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*, *round_trip: bool = False*, *warnings: bool = True*) → dict[str, Any]

Usage docs: https://docs.pydantic.dev/2.6/concepts/serialization/#modelmodel_dump

Generate a dictionary representation of the model, optionally specifying which fields to include or exclude.

Args:

mode: The mode in which *to_python* should run.
If mode is 'json', the output will only contain JSON serializable types. If mode is 'python', the output may contain non-JSON-serializable Python objects.

include: A list of fields to include in the output. exclude: A list of fields to exclude from the output. by_alias: Whether to use the field's alias in the dictionary key if defined. exclude_unset: Whether to exclude fields that have not been explicitly set. exclude_defaults: Whether to exclude fields that are set to their default value. exclude_none: Whether to exclude fields that have a value of *None*. round_trip: If True, dumped values should be valid as input for non-idempotent types such as Json[T].

warnings: Whether to log warnings when invalid fields are encountered.

Returns:

A dictionary representation of the model.

model_dump_json(*, *indent: int | None = None*, *include: IncEx = None*, *exclude: IncEx = None*, *by_alias: bool = False*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*, *round_trip: bool = False*, *warnings: bool = True*) → str

Usage docs: https://docs.pydantic.dev/2.6/concepts/serialization/#modelmodel_dump_json

Generates a JSON representation of the model using Pydantic's *to_json* method.

Args:

indent: Indentation to use in the JSON output. If None is passed, the output will be compact. include: Field(s) to include in the JSON output. exclude: Field(s) to exclude from the JSON output. by_alias: Whether to serialize using field aliases. exclude_unset: Whether to exclude fields that have not been explicitly set. exclude_defaults: Whether to exclude fields that are set to their default value. exclude_none: Whether to exclude fields that have a value of *None*. round_trip: If True, dumped values should be valid as input for non-idempotent types such as Json[T]. warnings: Whether to log warnings when invalid fields are encountered.

Returns:

A JSON string representation of the model.

property model_extra*: dict[str, Any] | None*

Get extra fields set during validation.

Returns:

A dictionary of extra fields, or *None* if *config.extra* is not set to *"allow"*.

model_fields*: ClassVar[dict[str, FieldInfo]] = {'absolute_time_range': FieldInfo(annotation=Union[AbsoluteTimeRange, NoneType], required=False), 'relative_time_range': FieldInfo(annotation=Union[RelativeTimeRage, NoneType], required=False)}*

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

property model_fields_set*: set[str]*

Returns the set of fields that have been explicitly set on this model instance.

Returns:

A set of strings representing the fields that have been set, i.e. that were not filled from defaults.

classmethod model_json_schema(*by_alias: bool = True*, *ref_template: str = '#/$defs/{model}'*, *schema_generator: type[pydantic.json_schema.GenerateJsonSchema] = <class 'pydantic.json_schema.GenerateJsonSchema'>*, *mode: ~typing.Literal['validation', 'serialization'] = 'validation'*) → dict[str, Any]

Generates a JSON schema for a model class.

Args:

by_alias: Whether to use attribute aliases or not. ref_template: The reference template. schema_generator: To override the logic used to generate the JSON schema, as a subclass of

GenerateJsonSchema with your desired modifications

mode: The mode in which to generate the schema.

Returns:

The JSON schema for the given model class.

classmethod model_parametrized_name(*params: tuple[type[Any], ...]*) → str

Compute the class name for parametrizations of generic classes.

This method can be overridden to achieve a custom naming scheme for generic BaseModels.

Args:

params: Tuple of types of the class. Given a generic class *Model* with 2 type variables and a concrete model *Model[str, int]*, the value *(str, int)* would be passed to *params*.

Returns:

String representing the new class where *params* are passed to *cls* as type variables.

Raises:

TypeError: Raised when trying to generate concrete names for non-generic models.

model_post_init(*_BaseModel__context: Any*) → None

Override this method to perform additional initialization after *__init__* and *model_construct*. This is useful if you want to do some validation that requires the entire model to be initialized.

classmethod model_rebuild(*, *force: bool = False*, *raise_errors: bool = True*, *_parent_namespace_depth: int = 2*, *_types_namespace: Optional[dict[str, Any]] = None*) → bool | None

Try to rebuild the pydantic-core schema for the model.

This may be necessary when one of the annotations is a ForwardRef which could not be resolved during the initial attempt to build the schema, and automatic rebuilding fails.

Args:

force: Whether to force the rebuilding of the model schema, defaults to *False*. raise_errors: Whether to raise errors, defaults to *True*. _parent_namespace_depth: The depth level of the parent namespace, defaults to 2. _types_namespace: The types namespace, defaults to *None*.

Returns:

Returns *None* if the schema is already "complete" and rebuilding was not required. If rebuilding _was_ required, returns *True* if rebuilding was successful, otherwise *False*.

classmethod model_validate(*obj: Any*, *, *strict: bool | None = None*, *from_attributes: bool | None = None*, *context: dict[str, Any] | None = None*) → Model

Validate a pydantic model instance.

Args:

obj: The object to validate. strict: Whether to enforce types strictly. from_attributes: Whether to extract data from object attributes. context: Additional context to pass to the validator.

Raises:

ValidationError: If the object could not be validated.

Returns:

The validated model instance.

classmethod model_validate_json(*json_data: str | bytes | bytearray*, *, *strict: bool | None = None*, *context: dict[str, Any] | None = None*) → Model

Usage docs: https://docs.pydantic.dev/2.6/concepts/json/#json-parsing

Validate the given JSON data against the Pydantic model.

Args:

json_data: The JSON data to validate. strict: Whether to enforce types strictly. context: Extra variables to pass to the validator.

Returns:
The validated Pydantic model.

Raises:
ValueError: If *json_data* is not a JSON string.

classmethod model_validate_strings(*obj: Any*, *, *strict: bool | None = None*, *context: dict[str, Any] | None = None*) → Model

Validate the given object contains string data against the Pydantic model.

Args:
obj: The object contains string data to validate. strict: Whether to enforce types strictly. context: Extra variables to pass to the validator.

Returns:
The validated Pydantic model.

TimeUnit

class zvt.common.query_models.TimeUnit(*value*)

Bases: **`Enum`**

An enumeration.

__init__()

zvt.common.trading_models

Classes

BuyParameter(*, entity_ids[, position_type, ...])	
PositionType(value)	An enumeration.
SellParameter(*, entity_ids[, sell_pcts])	
TradingResult(*[, success_entity_ids, ...])	

BuyParameter

class zvt.common.trading_models.BuyParameter(*, *entity_ids: List[str]*, *position_type: PositionType = PositionType.normal*, *position_pct: Optional[float] = None*, *weights: Optional[List[float]] = None*, *money_to_use: Optional[float] = None*)

Bases: **`BaseModel`**

__init__(**data: Any*) → None

Create a new model by parsing and validating input data from keyword arguments.

Raises [*ValidationError*][pydantic_core.ValidationError] if the input data cannot be validated to form a valid model.

self is explicitly positional-only to allow *self* as a field name.

copy(*, *include: AbstractSetIntStr | MappingIntStrAny | None = None*, *exclude: AbstractSetIntStr | MappingIntStrAny | None = None*, *update: Dict[str, Any] | None = None*, *deep: bool = False*) → Model

Returns a copy of the model.

!!! warning "Deprecated"

This method is now deprecated; use *model_copy* instead.

If you need *include* or *exclude*, use:

```
`py data = self.model_dump(include=include, exclude=exclude, round_trip=True) data = {**data, **(update or {})} copied = self.model_validate(data) `
```

Args:

include: Optional set or mapping specifying which fields to include in the copied model. exclude: Optional set or mapping

specifying which fields to exclude in the copied model. update: Optional dictionary of field-value pairs to override field values in the copied model. deep: If True, the values of fields that are Pydantic models will be deep-copied.

Returns:

A copy of the model with included, excluded and updated fields as specified.

model_computed_fields*: ClassVar[dict[str, ComputedFieldInfo]] = {}*

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

model_config*: ClassVar[ConfigDict] = {}*

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

classmethod model_construct(*_fields_set: set[str] | None = None*, ***values: Any*) → Model

Creates a new instance of the *Model* class with validated data.

Creates a new model setting *__dict__* and *__pydantic_fields_set__* from trusted or pre-validated data. Default values are respected, but no other validation is performed. Behaves as if *Config.extra = 'allow'* was set since it adds all passed values

Args:

_fields_set: The set of field names accepted for the Model instance. values: Trusted or pre-validated data dictionary.

Returns:

A new instance of the *Model* class with validated data.

model_copy(*, *update: dict[str, Any] | None = None*, *deep: bool = False*) → Model

Usage docs: https://docs.pydantic.dev/2.6/concepts/serialization/#model_copy

Returns a copy of the model.

Args:

update: Values to change/add in the new model. Note: the data is not validated
before creating the new model. You should trust this data.

deep: Set to *True* to make a deep copy of the model.

Returns:

New model instance.

model_dump(*, *mode: Literal['json', 'python'] | str = 'python'*, *include: IncEx = None*, *exclude: IncEx = None*, *by_alias: bool = False*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*, *round_trip: bool = False*, *warnings: bool = True*) → dict[str, Any]

Usage docs: https://docs.pydantic.dev/2.6/concepts/serialization/#modelmodel_dump

Generate a dictionary representation of the model, optionally specifying which fields to include or exclude.

Args:

mode: The mode in which *to_python* should run.
If mode is 'json', the output will only contain JSON serializable types. If mode is 'python', the output may contain non-JSON-serializable Python objects.

include: A list of fields to include in the output. exclude: A list of fields to exclude from the output. by_alias: Whether to use the field's alias in the dictionary key if defined. exclude_unset: Whether to exclude fields that have not been explicitly set. exclude_defaults: Whether to exclude fields that are set to their default value. exclude_none: Whether to exclude fields that have a value of *None*. round_trip: If True, dumped values should be

valid as input for non-idempotent types such as Json[T]. warnings: Whether to log warnings when invalid fields are encountered.

Returns:

A dictionary representation of the model.

model_dump_json(*, *indent: int | None = None*, *include: IncEx = None*, *exclude: IncEx = None*, *by_alias: bool = False*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*, *round_trip: bool = False*, *warnings: bool = True*) → str

Usage docs: https://docs.pydantic.dev/2.6/concepts/serialization/#modelmodel_dump_json

Generates a JSON representation of the model using Pydantic's *to_json* method.

Args:

indent: Indentation to use in the JSON output. If None is passed, the output will be compact. include: Field(s) to include in the JSON output. exclude: Field(s) to exclude from the JSON output. by_alias: Whether to serialize using field aliases. exclude_unset: Whether to exclude fields that have not been explicitly set. exclude_defaults: Whether to exclude fields that are set to their default value. exclude_none: Whether to exclude fields that have a value of *None*. round_trip: If True, dumped values should be valid as input for non-idempotent types such as Json[T]. warnings: Whether to log warnings when invalid fields are encountered.

Returns:

A JSON string representation of the model.

property model_extra*: dict[str, Any] | None*

Get extra fields set during validation.

Returns:

A dictionary of extra fields, or *None* if *config.extra* is not set to *"allow"*.

model_fields*: ClassVar[dict[str, FieldInfo]] = {'entity_ids': FieldInfo(annotation=List[str], required=True), 'money_to_use': FieldInfo(annotation=Union[float, NoneType], required=False), 'position_pct': FieldInfo(annotation=Union[float, NoneType], required=False), 'position_type': FieldInfo(annotation=PositionType, required=False, default=<PositionType.normal: 'normal'>), 'weights': FieldInfo(annotation=Union[List[float], NoneType], required=False)}*

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

property model_fields_set*: set[str]*

Returns the set of fields that have been explicitly set on this model instance.

Returns:

A set of strings representing the fields that have been set, i.e. that were not filled from defaults.

classmethod model_json_schema(*by_alias: bool = True*, *ref_template: str = '#/$defs/{model}'*, *schema_generator: type[pydantic.json_schema.GenerateJsonSchema] = <class 'pydantic.json_schema.GenerateJsonSchema'>*, *mode: ~typing.Literal['validation', 'serialization'] = 'validation'*) → dict[str, Any]

Generates a JSON schema for a model class.

Args:

by_alias: Whether to use attribute aliases or not. ref_template: The reference template. schema_generator: To override the logic used to generate the JSON schema, as a subclass of

GenerateJsonSchema with your desired modifications

mode: The mode in which to generate the schema.

Returns:

The JSON schema for the given model class.

classmethod model_parametrized_name(*params: tuple[type[Any], ...]*) → str

Compute the class name for parametrizations of generic classes.

This method can be overridden to achieve a custom naming scheme for generic BaseModels.

Args:

params: Tuple of types of the class. Given a generic class *Model* with 2 type variables and a concrete model *Model[str, int]*, the value *(str, int)* would be passed to *params*.

Returns:

String representing the new class where *params* are passed to *cls* as type variables.

Raises:

TypeError: Raised when trying to generate concrete names for non-generic models.

model_post_init(*_BaseModel__context: Any*) → None

Override this method to perform additional initialization after *__init__* and *model_construct*. This is useful if you want to do some validation that requires the entire model to be initialized.

classmethod model_rebuild(*, *force: bool = False*, *raise_errors: bool = True*, *_parent_namespace_depth: int = 2*, *_types_namespace: Optional[dict[str, Any]] = None*) → bool | None

Try to rebuild the pydantic-core schema for the model.

This may be necessary when one of the annotations is a ForwardRef which could not be resolved during the initial attempt to build the schema, and automatic rebuilding fails.

Args:

force: Whether to force the rebuilding of the model schema, defaults to *False*. raise_errors: Whether to raise errors, defaults to *True*. _parent_namespace_depth: The depth level of the parent namespace, defaults to 2. _types_namespace: The types namespace, defaults to *None*.

Returns:

Returns *None* if the schema is already "complete" and rebuilding was not required. If rebuilding _was_ required, returns *True* if rebuilding was successful, otherwise *False*.

classmethod model_validate(*obj: Any*, *, *strict: bool | None = None*, *from_attributes: bool | None = None*, *context: dict[str, Any] | None = None*) → Model

Validate a pydantic model instance.

Args:

obj: The object to validate. strict: Whether to enforce types strictly. from_attributes: Whether to extract data from object attributes. context: Additional context to pass to the validator.

Raises:

ValidationError: If the object could not be validated.

Returns:

The validated model instance.

classmethod model_validate_json(*json_data: str | bytes | bytearray*, *, *strict: bool | None = None*, *context: dict[str, Any] | None = None*) → Model

Usage docs: https://docs.pydantic.dev/2.6/concepts/json/#json-parsing

Validate the given JSON data against the Pydantic model.

Args:

json_data: The JSON data to validate. strict: Whether to enforce types strictly. context: Extra variables to pass to the validator.

Returns:
The validated Pydantic model.

Raises:
ValueError: If *json_data* is not a JSON string.

classmethod model_validate_strings(*obj: Any*, *, *strict: bool | None = None*, *context: dict[str, Any] | None = None*) → Model

Validate the given object contains string data against the Pydantic model.

Args:
obj: The object contains string data to validate. strict: Whether to enforce types strictly. context: Extra variables to pass to the validator.

Returns:
The validated Pydantic model.

PositionType

class zvt.common.trading_models.PositionType(*value*)

Bases: **`Enum`**

An enumeration.

__init__()

SellParameter

class zvt.common.trading_models.SellParameter(*, *entity_ids: List[str]*, *sell_pcts: Optional[List[float]] = None*)

Bases: **`BaseModel`**

__init__(**data: Any*) → None

Create a new model by parsing and validating input data from keyword arguments.

Raises [*ValidationError*][pydantic_core.ValidationError] if the input data cannot be validated to form a valid model.

self is explicitly positional-only to allow *self* as a field name.

copy(*, *include: AbstractSetIntStr | MappingIntStrAny | None = None*, *exclude: AbstractSetIntStr | MappingIntStrAny | None = None*, *update: Dict[str, Any] | None = None*, *deep: bool = False*) → Model

Returns a copy of the model.

!!! warning "Deprecated"

This method is now deprecated; use *model_copy* instead.

If you need *include* or *exclude*, use:

```
`py data = self.model_dump(include=include, exclude=exclude, round_trip=True) data = {**data, **(update or {})} copied = self.model_validate(data) `
```

Args:

include: Optional set or mapping specifying which fields to include in the copied model. exclude: Optional set or mapping specifying which fields to exclude in the copied model. update: Optional dictionary of field-value pairs to override field values

in the copied model. deep: If True, the values of fields that are Pydantic models will be deep-copied.

Returns:

A copy of the model with included, excluded and updated fields as specified.

model_computed_fields*: ClassVar[dict[str, ComputedFieldInfo]] = {}*

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

model_config*: ClassVar[ConfigDict] = {}*

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

classmethod model_construct(*_fields_set: set[str] | None = None*, ***values: Any*) → Model

Creates a new instance of the *Model* class with validated data.

Creates a new model setting *__dict__* and *__pydantic_fields_set__* from trusted or pre-validated data. Default values are respected, but no other validation is performed. Behaves as if *Config.extra = 'allow'* was set since it adds all passed values

Args:

_fields_set: The set of field names accepted for the Model instance. values: Trusted or pre-validated data dictionary.

Returns:

A new instance of the *Model* class with validated data.

model_copy(*, *update: dict[str, Any] | None = None*, *deep: bool = False*) → Model

Usage docs: https://docs.pydantic.dev/2.6/concepts/serialization/#model_copy

Returns a copy of the model.

Args:

update: Values to change/add in the new model. Note: the data is not validated
before creating the new model. You should trust this data.

deep: Set to *True* to make a deep copy of the model.

Returns:

New model instance.

model_dump(*, *mode: Literal['json', 'python'] | str = 'python'*, *include: IncEx = None*, *exclude: IncEx = None*, *by_alias: bool = False*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*, *round_trip: bool = False*, *warnings: bool = True*) → dict[str, Any]

Usage docs: https://docs.pydantic.dev/2.6/concepts/serialization/#modelmodel_dump

Generate a dictionary representation of the model, optionally specifying which fields to include or exclude.

Args:

mode: The mode in which *to_python* should run.
If mode is 'json', the output will only contain JSON serializable types. If mode is 'python', the output may contain non-JSON-serializable Python objects.

include: A list of fields to include in the output. exclude: A list of fields to exclude from the output. by_alias: Whether to use the field's alias in the dictionary key if defined. exclude_unset: Whether to exclude fields that have not been explicitly set. exclude_defaults: Whether to exclude fields that are set to their default value. exclude_none: Whether to exclude fields that have a value of *None*. round_trip: If True, dumped values should be valid as input for non-idempotent types such as Json[T].

warnings: Whether to log warnings when invalid fields are encountered.

Returns:

A dictionary representation of the model.

model_dump_json(*, *indent: int | None = None*, *include: IncEx = None*, *exclude: IncEx = None*, *by_alias: bool = False*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*, *round_trip: bool = False*, *warnings: bool = True*) → str

Usage docs: https://docs.pydantic.dev/2.6/concepts/serialization/#modelmodel_dump_json

Generates a JSON representation of the model using Pydantic's *to_json* method.

Args:

indent: Indentation to use in the JSON output. If None is passed, the output will be compact. include: Field(s) to include in the JSON output. exclude: Field(s) to exclude from the JSON output. by_alias: Whether to serialize using field aliases. exclude_unset: Whether to exclude fields that have not been explicitly set. exclude_defaults: Whether to exclude fields that are set to their default value. exclude_none: Whether to exclude fields that have a value of *None*. round_trip: If True, dumped values should be valid as input for non-idempotent types such as Json[T]. warnings: Whether to log warnings when invalid fields are encountered.

Returns:

A JSON string representation of the model.

property model_extra*: dict[str, Any] | None*

Get extra fields set during validation.

Returns:

A dictionary of extra fields, or *None* if *config.extra* is not set to *"allow"*.

model_fields*: ClassVar[dict[str, FieldInfo]] = {'entity_ids': FieldInfo(annotation=List[str], required=True), 'sell_pcts': FieldInfo(annotation=Union[List[float], NoneType], required=False)}*

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

property model_fields_set*: set[str]*

Returns the set of fields that have been explicitly set on this model instance.

Returns:

A set of strings representing the fields that have been set, i.e. that were not filled from defaults.

classmethod model_json_schema(*by_alias: bool = True*, *ref_template: str = '#/$defs/{model}'*, *schema_generator: type[pydantic.json_schema.GenerateJsonSchema] = <class 'pydantic.json_schema.GenerateJsonSchema'>*, *mode: ~typing.Literal['validation', 'serialization'] = 'validation'*) → dict[str, Any]

Generates a JSON schema for a model class.

Args:

by_alias: Whether to use attribute aliases or not. ref_template: The reference template. schema_generator: To override the logic used to generate the JSON schema, as a subclass of

GenerateJsonSchema with your desired modifications

mode: The mode in which to generate the schema.

Returns:

The JSON schema for the given model class.

classmethod model_parametrized_name(*params: tuple[type[Any], ...]*) → str

Compute the class name for parametrizations of generic classes.

This method can be overridden to achieve a custom naming scheme for generic BaseModels.

Args:

params: Tuple of types of the class. Given a generic class *Model* with 2 type variables and a concrete model *Model[str, int]*, the value *(str, int)* would be passed to *params*.

Returns:

String representing the new class where *params* are passed to *cls* as type variables.

Raises:

TypeError: Raised when trying to generate concrete names for non-generic models.

model_post_init(*_BaseModel__context: Any*) → None

Override this method to perform additional initialization after *__init__* and *model_construct*. This is useful if you want to do some validation that requires the entire model to be initialized.

classmethod model_rebuild(*, *force: bool = False*, *raise_errors: bool = True*, *_parent_namespace_depth: int = 2*, *_types_namespace: Optional[dict[str, Any]] = None*) → bool | None

Try to rebuild the pydantic-core schema for the model.

This may be necessary when one of the annotations is a ForwardRef which could not be resolved during the initial attempt to build the schema, and automatic rebuilding fails.

Args:

force: Whether to force the rebuilding of the model schema, defaults to *False*. raise_errors: Whether to raise errors, defaults to *True*. _parent_namespace_depth: The depth level of the parent namespace, defaults to 2. _types_namespace: The types namespace, defaults to *None*.

Returns:
Returns *None* if the schema is already "complete" and rebuilding was not required. If rebuilding _was_ required, returns *True* if rebuilding was successful, otherwise *False*.

classmethod model_validate(*obj: Any*, *, *strict: bool | None = None*, *from_attributes: bool | None = None*, *context: dict[str, Any] | None = None*) → Model

Validate a pydantic model instance.

Args:
obj: The object to validate. strict: Whether to enforce types strictly. from_attributes: Whether to extract data from object attributes. context: Additional context to pass to the validator.

Raises:
ValidationError: If the object could not be validated.

Returns:
The validated model instance.

classmethod model_validate_json(*json_data: str | bytes | bytearray*, *, *strict: bool | None = None*, *context: dict[str, Any] | None = None*) → Model

Usage docs: https://docs.pydantic.dev/2.6/concepts/json/#json-parsing

Validate the given JSON data against the Pydantic model.

Args:

json_data: The JSON data to validate. strict: Whether to enforce types strictly. context: Extra variables to pass to the validator.

Returns:
The validated Pydantic model.

Raises:
ValueError: If *json_data* is not a JSON string.

classmethod model_validate_strings(*obj: Any*, *, *strict: bool | None = None*, *context: dict[str, Any] | None = None*) → Model

Validate the given object contains string data against the Pydantic model.

Args:
obj: The object contains string data to validate. strict: Whether to enforce types strictly. context: Extra variables to pass to the validator.

Returns:
The validated Pydantic model.

TradingResult

class zvt.common.trading_models.TradingResult(*, *success_entity_ids: Optional[List[str]] = None*, *failed_entity_ids: Optional[List[str]] = None*)

Bases: **`BaseModel`**

__init__(**data: Any*) → None

Create a new model by parsing and validating input data from keyword arguments.

Raises [*ValidationError*][pydantic_core.ValidationError] if the input data cannot be validated to form a valid model.

self is explicitly positional-only to allow *self* as a field name.

copy(*, *include: AbstractSetIntStr | MappingIntStrAny | None = None*, *exclude: AbstractSetIntStr | MappingIntStrAny | None = None*, *update: Dict[str, Any] | None = None*, *deep: bool = False*) → Model

Returns a copy of the model.

!!! warning "Deprecated"

This method is now deprecated; use *model_copy* instead.

If you need *include* or *exclude*, use:

```
`py data = self.model_dump(include=include, exclude=exclude, round_trip=True) data = {**data, **(update or {})} copied = self.model_validate(data) `
```

Args:

include: Optional set or mapping specifying which fields to include in the copied model. exclude: Optional set or mapping specifying which fields to exclude in the copied model. update: Optional dictionary of field-value pairs to override field values

in the copied model. deep: If True, the values of fields that are Pydantic models will be deep-copied.

Returns:

A copy of the model with included, excluded and updated fields as specified.

model_computed_fields*: ClassVar[dict[str, ComputedFieldInfo]] = {}*

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

model_config*: ClassVar[ConfigDict] = {}*

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

classmethod model_construct(*_fields_set: set[str] | None = None*, ***values: Any*) → Model

Creates a new instance of the *Model* class with validated data.

Creates a new model setting *__dict__* and *__pydantic_fields_set__* from trusted or pre-validated data. Default values are respected, but no other validation is performed. Behaves as if *Config.extra = 'allow'* was set since it adds all passed values

Args:

_fields_set: The set of field names accepted for the Model instance. values: Trusted or pre-validated data dictionary.

Returns:

A new instance of the *Model* class with validated data.

model_copy(*, *update: dict[str, Any] | None = None*, *deep: bool = False*) → Model

Usage docs: https://docs.pydantic.dev/2.6/concepts/serialization/#model_copy

Returns a copy of the model.

Args:

update: Values to change/add in the new model. Note: the data is not validated
before creating the new model. You should trust this data.

deep: Set to *True* to make a deep copy of the model.

Returns:

New model instance.

model_dump(*, *mode: Literal['json', 'python'] | str = 'python'*, *include: IncEx = None*, *exclude: IncEx = None*, *by_alias: bool = False*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*, *round_trip: bool = False*, *warnings: bool = True*) → dict[str, Any]

Usage docs: https://docs.pydantic.dev/2.6/concepts/serialization/#modelmodel_dump

Generate a dictionary representation of the model, optionally specifying which fields to include or exclude.

Args:

mode: The mode in which *to_python* should run.
If mode is 'json', the output will only contain JSON serializable types. If mode is 'python', the output may contain non-JSON-serializable Python objects.

include: A list of fields to include in the output. exclude: A list of fields to exclude from the output. by_alias: Whether to use the field's alias in the dictionary key if defined. exclude_unset: Whether to exclude fields that have not been explicitly set. exclude_defaults: Whether to exclude fields that are set to their default value. exclude_none: Whether to exclude fields that have a value of *None*. round_trip: If True, dumped values should be valid as input for non-idempotent types such as Json[T].

warnings: Whether to log warnings when invalid fields are encountered.

Returns:

A dictionary representation of the model.

model_dump_json(*, *indent: int | None = None*, *include: IncEx = None*, *exclude: IncEx = None*, *by_alias: bool = False*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*, *round_trip: bool = False*, *warnings: bool = True*) → str

Usage docs: https://docs.pydantic.dev/2.6/concepts/serialization/#modelmodel_dump_json

Generates a JSON representation of the model using Pydantic's *to_json* method.

Args:

indent: Indentation to use in the JSON output. If None is passed, the output will be compact. include: Field(s) to include in the JSON output. exclude: Field(s) to exclude from the JSON output. by_alias: Whether to serialize using field aliases. exclude_unset: Whether to exclude fields that have not been explicitly set. exclude_defaults: Whether to exclude fields that are set to their default value. exclude_none: Whether to exclude fields that have a value of *None*. round_trip: If True, dumped values should be valid as input for non-idempotent types such as Json[T]. warnings: Whether to log warnings when invalid fields are encountered.

Returns:

A JSON string representation of the model.

property model_extra*: dict[str, Any] | None*

Get extra fields set during validation.

Returns:

A dictionary of extra fields, or *None* if *config.extra* is not set to *"allow"*.

model_fields*: ClassVar[dict[str, FieldInfo]] = {'failed_entity_ids': FieldInfo(annotation=Union[List[str], NoneType], required=False), 'success_entity_ids': FieldInfo(annotation=Union[List[str], NoneType], required=False)}*

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

property model_fields_set*: set[str]*

Returns the set of fields that have been explicitly set on this model instance.

Returns:

A set of strings representing the fields that have been set, i.e. that were not filled from defaults.

classmethod model_json_schema(*by_alias: bool = True*, *ref_template: str = '#/$defs/{model}'*, *schema_generator: type[pydantic.json_schema.GenerateJsonSchema] = <class 'pydantic.json_schema.GenerateJsonSchema'>*, *mode: ~typing.Literal['validation', 'serialization'] = 'validation'*) → dict[str, Any]

Generates a JSON schema for a model class.

Args:

by_alias: Whether to use attribute aliases or not. ref_template: The reference template. schema_generator: To override the logic used to generate the JSON schema, as a subclass of

GenerateJsonSchema with your desired modifications

mode: The mode in which to generate the schema.

Returns:

The JSON schema for the given model class.

classmethod model_parametrized_name(*params: tuple[type[Any], ...]*) → str

Compute the class name for parametrizations of generic classes.

This method can be overridden to achieve a custom naming scheme for generic BaseModels.

Args:

params: Tuple of types of the class. Given a generic class *Model* with 2 type variables and a concrete model *Model[str, int]*, the value *(str, int)* would be passed to *params*.

Returns:

String representing the new class where *params* are passed to *cls* as type variables.

Raises:

TypeError: Raised when trying to generate concrete names for non-generic models.

model_post_init(*_BaseModel__context: Any*) → None

Override this method to perform additional initialization after *__init__* and *model_construct*. This is useful if you want to do some validation that requires the entire model to be initialized.

classmethod model_rebuild(*, *force: bool = False*, *raise_errors: bool = True*, *_parent_namespace_depth: int = 2*, *_types_namespace: Optional[dict[str, Any]] = None*) → bool | None

Try to rebuild the pydantic-core schema for the model.

This may be necessary when one of the annotations is a ForwardRef which could not be resolved during the initial attempt to build the schema, and automatic rebuilding fails.

Args:

force: Whether to force the rebuilding of the model schema, defaults to *False*. raise_errors: Whether to raise errors, defaults to *True*. _parent_namespace_depth: The depth level of the parent namespace, defaults to 2. _types_namespace: The types namespace, defaults to *None*.

Returns:
Returns *None* if the schema is already "complete" and rebuilding was not required. If rebuilding _was_ required, returns *True* if rebuilding was successful, otherwise *False*.

classmethod model_validate(*obj: Any*, *, *strict: bool | None = None*, *from_attributes: bool | None = None*, *context: dict[str, Any] | None = None*) → Model

Validate a pydantic model instance.

Args:
obj: The object to validate. strict: Whether to enforce types strictly. from_attributes: Whether to extract data from object attributes. context: Additional context to pass to the validator.

Raises:
ValidationError: If the object could not be validated.

Returns:
The validated model instance.

classmethod model_validate_json(*json_data: str | bytes | bytearray*, *, *strict: bool | None = None*, *context: dict[str, Any] | None = None*) → Model

Usage docs: https://docs.pydantic.dev/2.6/concepts/json/#json-parsing

Validate the given JSON data against the Pydantic model.

Args:

json_data: The JSON data to validate. strict: Whether to enforce types strictly. context: Extra variables to pass to the validator.

Returns:
The validated Pydantic model.

Raises:
ValueError: If *json_data* is not a JSON string.

classmethod model_validate_strings(*obj: Any*, *, *strict: bool | None = None*, *context: dict[str, Any] | None = None*) → Model

Validate the given object contains string data against the Pydantic model.

Args:
obj: The object contains string data to validate. strict: Whether to enforce types strictly. context: Extra variables to pass to the validator.

Returns:
The validated Pydantic model.

zvt.consts

zvt.contract

Functions

`get_entity_exchanges`(entity_type)

Classes

`ActorType`(value)	An enumeration.
`AdjustType`(value)	split-adjusted type for `TradableEntity` quotes
`Exchange`(value)	An enumeration.
`IntervalLevel`(value)	Repeated fixed time interval, e.g, 5m, 1d.
`TradableType`(value)	An enumeration.

Modules

`zvt.contract.api`

`zvt.contract.base_service`

zvt.contract.context

zvt.contract.data_type

zvt.contract.drawer

zvt.contract.factor

zvt.contract.model

zvt.contract.normal_data

zvt.contract.reader

zvt.contract.recorder

zvt.contract.register

zvt.contract.schema

zvt.contract.zvt_info

zvt.contract.get_entity_exchanges

zvt.contract.get_entity_exchanges(*entity_type*)

ActorType

class zvt.contract.ActorType(*value*)

Bases: **`Enum`**

An enumeration.

__init__()

individual = *'individual'*

个人

raised_fund = *'raised_fund'*

公募基金

social_security = *'social_security'*

社保

insurance = *'insurance'*

保险

qfii = *'qfii'*

外资

trust = *'trust'*

信托

broker = *'qmt'*

券商

private_equity = *'private_equity'*

私募

corporation = *'corporation'*

公司(可能包括私募)

AdjustType

class zvt.contract.AdjustType(*value*)

Bases: `Enum`

split-adjusted type for `TradableEntity` quotes

__init__()

bfq = *'bfq'*

not adjusted 不复权

qfq = *'qfq'*

pre adjusted 前复权

hfq = *'hfq'*

post adjusted 后复权

Exchange

class zvt.contract.Exchange(*value*)

Bases: **`Enum`**

An enumeration.

__init__()

sh = *'sh'*

上证交易所

sz = *'sz'*

深证交易所

bj = *'bj'*

北交所

cn = *'cn'*

对于中国的非交易所的 标的

us = *'us'*

对于美国的非交易所的 标的

nasdaq = *'nasdaq'*

纳斯达克

nyse = *'nyse'*

纽交所

hk = *'hk'*

港交所

binance = *'binance'*

数字货币

shfe = *'shfe'*

上海期货交易所

dce = *'dce'*

大连商品交易所

czce = *'czce'*

郑州商品交易所

cffex = *'cffex'*

中国金融期货交易所

ine = *'ine'*

上海国际能源交易中心

forex = *'forex'*

外汇交易所(虚拟) currency exchange(virtual)

IntervalLevel

class zvt.contract.IntervalLevel(*value*)

Bases: **`Enum`**

Repeated fixed time interval, e.g, 5m, 1d.

__init__()

LEVEL_L2_QUOTE = *'l2quote'*

level l2 quote

LEVEL_TICK = *'tick'*

level tick

LEVEL_1MIN = *'1m'*

1 minute

LEVEL_5MIN = *'5m'*

5 minutes

LEVEL_15MIN = *'15m'*

15 minutes

LEVEL_30MIN = *'30m'*

30 minutes

LEVEL_1HOUR = *'1h'*

1 hour

LEVEL_4HOUR = *'4h'*

4 hours

LEVEL_1DAY = *'1d'*

1 day

LEVEL_1WEEK = *'1wk'*

1 week

LEVEL_1MON = *'1mon'*

1 month

to_ms()

To seconds count in the interval

Returns:: seconds count in the interval

TradableType

class zvt.contract.TradableType(*value*)

Bases: **`Enum`**

An enumeration.

__init__()

stock = *'stock'*

A股(中国) China stock

index = *'index'*

A股指数(中国) China index

block = *'block'*

A股板块(中国) China stock block

stockus = *'stockus'*

美股 USA stock

indexus = *'indexus'*

美股指数 USA index

stockhk = *'stockhk'*

港股 Hongkong Stock

future = *'future'*

期货(中国) China future

coin = *'coin'*

数字货币 Cryptocurrency

option = *'option'*

期权(中国) China option

fund = *'fund'*

基金(中国) China fund

currency = *'currency'*

货币汇率 currency exchange rate

zvt.contract.api

Functions

DBSession(provider[, db_name, data_schema])	get db session factory from (provider,db_name) or (provider,data_schema)
common_filter(query, data_schema[, ...])	build filter by the arguments
data_exist(session, schema, id)	whether exist data of the id
decode_entity_id(entity_id)	decode entity id to entity_type, exchange, code
del_data(data_schema[, filters, provider])	delete data by filters
df_to_db(df, data_schema, provider[, ...])	store the df to db
get_by_id(data_schema, id[, provider, session])	get one record by id from data schema
get_data(data_schema[, ids, entity_ids, ...])	query data by the arguments

get_data_count(data_schema[, filters, session])	get record count basing on the filters
get_db_engine(provider[, db_name, ...])	get db engine from (provider,db_name) or (provider,data_schema)
get_db_session(provider[, db_name, ...])	get db session from (provider,db_name) or (provider,data_schema)
get_db_session_factory(provider[, db_name, ...])	get db session factory from (provider,db_name) or (provider,data_schema)
get_entities([entity_schema, entity_type, ...])	get entities by the arguments
get_entity_code(entity_id)	get code by entity id
get_entity_exchange(entity_id)	get exchange by entity id
get_entity_ids([entity_type, entity_schema, ...])	get entity ids by the arguments
get_entity_schema(entity_type)	get entity schema from name

`get_entity_type`(entity_id)	get entity type by entity id
`get_group`(provider, data_schema, column[, ...])	
`get_providers`()	
`get_schema_by_name`(name)	get domain schema by the name
`get_schema_columns`(schema)	get all columns of the domain schema
`get_schemas`(provider)	get domain schemas supported by the provider

zvt.contract.api.DBSession

zvt.contract.api.DBSession(*provider: str*, *db_name: Optional[str] = None*, *data_schema: Optional[object] = None*)

get db session factory from (provider,db_name) or (provider,data_schema)

Parameters::
- **provider** – data provider
- **db_name** – db name
- **data_schema** – data schema

Returns:: db session factory

zvt.contract.api.common_filter

zvt.contract.api.common_filter(*query: Query*, *data_schema*, *start_timestamp=None*, *end_timestamp=None*, *filters=None*, *order=None*, *limit=None*, *distinct=None*, *time_field='timestamp'*)

build filter by the arguments

Parameters::

- **query** – sql query
- **data_schema** – data schema
- **start_timestamp** – start timestamp
- **end_timestamp** – end timestamp
- **filters** – sql filters
- **order** – sql order
- **limit** – sql limit size
- **time_field** – time field in columns

Returns:: result query

zvt.contract.api.data_exist

zvt.contract.api.data_exist(*session*, *schema*, *id*)

whether exist data of the id

Parameters::

- **session** –
- **schema** –
- **id** –

Returns::

zvt.contract.api.decode_entity_id

zvt.contract.api.decode_entity_id(*entity_id: str*)

decode entity id to entity_type, exchange, code

Parameters:: **entity_id** –

Returns:: tuple with format (entity_type, exchange, code)

zvt.contract.api.del_data

zvt.contract.api.del_data(*data_schema: Type[Mixin]*, *filters: Optional[List] = None*, *provider=None*)

delete data by filters

Parameters::

- **data_schema** – data schema
- **filters** – filters
- **provider** – data provider

zvt.contract.api.df_to_db

zvt.contract.api.df_to_db(*df: DataFrame*, *data_schema: DeclarativeMeta*, *provider: str*, *force_update: bool = False*, *sub_size: int = 5000*, *drop_duplicates: bool = True*, *dtype=None*, *session=None*) → object

store the df to db

Parameters::

- **df** – data with columns of the schema
- **data_schema** – data schema
- **provider** – data provider
- **force_update** – whether update the data with id existed
- **sub_size** – update batch size
- **drop_duplicates** – whether drop duplicates

Returns::

zvt.contract.api.get_by_id

zvt.contract.api.get_by_id(*data_schema*, *id: str*, *provider: Optional[str] = None*, *session: Optional[Session] = None*)

get one record by id from data schema

Parameters::
- **data_schema** – data schema
- **id** – the record id
- **provider** – data provider
- **session** – db session

Returns:: the record of the id

zvt.contract.api.get_data

zvt.contract.api.get_data(*data_schema: Type[Mixin]*, *ids: Optional[List[str]] = None*, *entity_ids: Optional[List[str]] = None*, *entity_id: Optional[str] = None*, *codes: Optional[List[str]] = None*, *code: Optional[str] = None*, *level: Optional[Union[IntervalLevel, str]] = None*, *provider: Optional[str] = None*, *columns: Optional[List] = None*, *col_label: Optional[dict] = None*, *return_type: str = 'df'*, *start_timestamp: Optional[Union[Timestamp, str]] = None*, *end_timestamp: Optional[Union[Timestamp, str]] = None*, *filters: Optional[List] = None*, *session: Optional[Session] = None*, *order=None*, *limit: Optional[int] = None*, *distinct=None*, *index: Optional[Union[str, list]] = None*, *drop_index_col=False*, *time_field: str = 'timestamp'*)

query data by the arguments

Parameters::

- **data_schema** –
- **ids** –
- **entity_ids** –
- **entity_id** –
- **codes** –
- **code** –
- **level** –
- **provider** –
- **columns** –
- **col_label** – dict with key(column), value(label)
- **return_type** – df, domain or dict. default is df
- **start_timestamp** –
- **end_timestamp** –
- **filters** –
- **session** –
- **order** –
- **limit** –
- **index** – index field name, str for single index, str list for multiple index
- **drop_index_col** – whether drop the col if it's in index, default False
- **time_field** –

Returns:: results basing on return_type.

zvt.contract.api.get_data_count

zvt.contract.api.get_data_count(*data_schema*, *filters=None*, *session=None*)

get record count basing on the filters

Parameters::

- **data_schema** –
- **filters** –
- **session** –

Returns::

zvt.contract.api.get_db_engine

zvt.contract.api.get_db_engine(*provider: str*, *db_name: Optional[str] = None*, *data_schema: Optional[object] = None*, *data_path: str = '/home/huzhe/zvt-home/data'*) → Engine

get db engine from (provider,db_name) or (provider,data_schema)

Parameters::
- **provider** – data provider
- **db_name** – db name
- **data_schema** – data schema
- **data_path** – data path

Returns:: db engine

zvt.contract.api.get_db_session

zvt.contract.api.get_db_session(*provider: str*, *db_name: Optional[str] = None*, *data_schema: Optional[object] = None*, *force_new: bool = False*) → Session

get db session from (provider,db_name) or (provider,data_schema)

Parameters::
- **provider** – data provider
- **db_name** – db name
- **data_schema** – data schema
- **force_new** – True for new session, otherwise use global session

Returns:: db session

zvt.contract.api.get_db_session_factory

zvt.contract.api.get_db_session_factory(*provider: str*, *db_name: Optional[str] = None*, *data_schema: Optional[object] = None*)

get db session factory from (provider,db_name) or (provider,data_schema)

Parameters::
- **provider** – data provider
- **db_name** – db name
- **data_schema** – data schema

Returns:: db session factory

zvt.contract.api.get_entities

zvt.contract.api.get_entities(*entity_schema: Optional[Type[TradableEntity]] = None*, *entity_type: Optional[str] = None*, *exchanges: Optional[List[str]] = None*, *ids: Optional[List[str]] = None*, *entity_ids: Optional[List[str]] = None*, *entity_id: Optional[str] = None*, *codes: Optional[List[str]] = None*, *code: Optional[str] = None*, *provider: Optional[str] = None*, *columns: Optional[List] = None*, *col_label: Optional[dict] = None*, *return_type: str = 'df'*, *start_timestamp: Optional[Union[Timestamp, str]] = None*, *end_timestamp: Optional[Union[Timestamp, str]] = None*, *filters: Optional[List] = None*, *session: Optional[Session] = None*, *order=None*, *limit: Optional[int] = None*, *index: Union[str, list] = 'code'*) → List

get entities by the arguments

Parameters::

- **entity_schema** –
- **entity_type** –
- **exchanges** –
- **ids** –
- **entity_ids** –
- **entity_id** –
- **codes** –
- **code** –
- **provider** –
- **columns** –
- **col_label** –
- **return_type** –
- **start_timestamp** –
- **end_timestamp** –
- **filters** –
- **session** –
- **order** –
- **limit** –
- **index** –

Returns::

zvt.contract.api.get_entity_code

zvt.contract.api.get_entity_code(*entity_id: str*)

get code by entity id

Parameters:: **entity_id** –

Returns:: code

zvt.contract.api.get_entity_exchange

zvt.contract.api.get_entity_exchange(*entity_id: str*)

get exchange by entity id

Parameters:: **entity_id** –

Returns:: exchange

zvt.contract.api.get_entity_ids

zvt.contract.api.get_entity_ids(*entity_type='stock'*, *entity_schema: Optional[TradableEntity] = None*, *exchanges=None*, *codes=None*, *provider=None*, *filters=None*, *entity_ids=None*)

get entity ids by the arguments

Parameters::

- **entity_type** –
- **entity_schema** –
- **exchanges** –
- **codes** –
- **provider** –
- **filters** –
- **entity_ids** –

Returns::

zvt.contract.api.get_entity_schema

zvt.contract.api.get_entity_schema(*entity_type: str*) → Type[TradableEntity]

get entity schema from name

Parameters::	**entity_type** – entity type, e.g. stock, stockus.
Returns::	the Schema of the entity

zvt.contract.api.get_entity_type

zvt.contract.api.get_entity_type(*entity_id: str*)

get entity type by entity id

Parameters:: **entity_id** –

Returns:: entity type

zvt.contract.api.get_group

zvt.contract.api.get_group(*provider*, *data_schema*, *column*, *group_func=<sqlalchemy.sql.functions._FunctionGenerator object>*, *session=None*)

zvt.contract.api.get_providers

zvt.contract.api.get_providers() → List[str]

zvt.contract.api.get_schema_by_name

zvt.contract.api.get_schema_by_name(*name: str*) → DeclarativeMeta

get domain schema by the name

Parameters:: **name** – schema name

Returns:: schema

zvt.contract.api.get_schema_columns

zvt.contract.api.get_schema_columns(*schema: DeclarativeMeta*) → List[str]

get all columns of the domain schema

Parameters:: **schema** – data schema

Returns:: columns of the schema

zvt.contract.api.get_schemas

zvt.contract.api.get_schemas(*provider: str*) → List[DeclarativeMeta]

get domain schemas supported by the provider

Parameters:: **provider** – data provider

Returns:: schemas provided by the provider

zvt.contract.base_service

Classes

`EntityStateService`(entity_ids)	StatefulService which saving one state one entity
`OneStateService`()	StatefulService which saving all states in one object
`StatefulService`()	Base service with state could be stored in state_schema

EntityStateService

class zvt.contract.base_service.EntityStateService(*entity_ids*)

Bases: `StatefulService`

StatefulService which saving one state one entity

__init__(*entity_ids*) → None

states*: dict*

entity_id:state

clear_state_data(*entity_id=None*)

clear state of the entity

Parameters:: **entity_id** – entity id

decode_state(*state: str*)

decode state

Parameters:: **state** –

Returns::

encode_state(*state: object*)

encode state

Parameters:: **state** –

Returns::

name = *None*

name of the service, default name of class if not set manually

state_schema*: Type[StateMixin] = None*

state schema

OneStateService

class zvt.contract.base_service.OneStateService

Bases: `StatefulService`

StatefulService which saving all states in one object

__init__() → None

clear_state_data(*entity_id=None*)

clear state of the entity

Parameters:: **entity_id** – entity id

decode_state(*state: str*)

decode state

Parameters:: **state** –

Returns::

encode_state(*state: object*)

encode state

Parameters:: **state** –

Returns::

name = *None*

name of the service, default name of class if not set manually

state_schema*: Type[StateMixin] = None*

state schema

StatefulService

class zvt.contract.base_service.StatefulService

Bases: `object`

Base service with state could be stored in state_schema

__init__() → None

state_schema*: Type[StateMixin] = None*

state schema

name = *None*

name of the service, default name of class if not set manually

clear_state_data(*entity_id=None*)

clear state of the entity

Parameters:: **entity_id** – entity id

decode_state(*state: str*)

decode state

Parameters:: **state** –

Returns::

encode_state(*state: object*)

encode state

Parameters:: **state** –

Returns::

zvt.contract.context

Module Attributes

`zvt_context` `Registry` instance

Classes

`Registry`() Class storing zvt registering meta

---

zvt.contract.context.zvt_context

zvt.contract.context.zvt_context = *<zvt.contract.context.Registry object>*

`Registry` instance

Registry

class zvt.contract.context.Registry

Bases: `object`

Class storing zvt registering meta

__init__() → None

providers

all registered providers

tradable_entity_types

all registered entity types(str)

tradable_entity_schemas

all entity schemas

schemas

all registered schemas

tradable_schema_map

tradable entity type -> schema

sessions

global sessions

db_engine_map

provider_dbname -> engine

db_session_map

provider_dbname -> session

provider_map_dbnames

provider -> [db_name1,db_name2…]

dbname_map_base

db_name -> [declarative_base1,declarative_base2…]

dbname_map_schemas

db_name -> [declarative_meta1,declarative_meta2…]

entity_map_schemas

entity_type -> related schemas

factor_cls_registry

factor class registry

zvt.contract.data_type

Classes

`Bean`()

Bean

class zvt.contract.data_type.Bean

Bases: **`object`**

__init__() → None

zvt.contract.drawer

Functions

annotations(annotation_df[, yref])	annotation_df format.

Classes

ChartType(value)	Chart type enum
Draw()	
Drawable()	
Drawer([main_df, factor_df_list, ...])	
Rect([x0, y0, x1, y1])	rect struct with left-bottom(x0, y0), right-top(x1, y1)
StackedDrawer(*drawers)	

zvt.contract.drawer.annotations

zvt.contract.drawer.annotations(*annotation_df: DataFrame*, *yref='y'*)

annotation_df format:

```
                                    value    flag    color
entity_id    timestamp
```

Parameters::

- **annotation_df** –
- **yref** – specific yaxis e.g, y,y2,y3

Returns::

ChartType

class zvt.contract.drawer.ChartType(*value*)

Bases: `Enum`

Chart type enum

__init__()

kline = *'kline'*

candlestick chart

line = *'line'*

line chart

area = *'area'*

area chart

scatter = *'scatter'*

scatter chart

histogram = *'histogram'*

histogram chart

pie = *'pie'*

pie chart

bar = *'bar'*

bar chart

Draw

class zvt.contract.drawer.Draw

Bases: `object`

__init__()

Drawable

class zvt.contract.drawer.Drawable

Bases: `object`

__init__()

Drawer

class zvt.contract.drawer.Drawer(*main_df: Optional[DataFrame] = None*, *factor_df_list: Optional[List[DataFrame]] = None*, *sub_df_list: Optional[DataFrame] = None*, *main_data: Optional[NormalData] = None*, *factor_data_list: Optional[List[NormalData]] = None*, *sub_data_list: Optional[NormalData] = None*, *sub_col_chart: Optional[dict] = None*, *rects: Optional[List[Rect]] = None*, *annotation_df: Optional[DataFrame] = None*, *scale_value: Optional[int] = None*)

Bases: `Draw`

__init__(*main_df: Optional[DataFrame] = None*, *factor_df_list: Optional[List[DataFrame]] = None*, *sub_df_list: Optional[DataFrame] = None*, *main_data: Optional[NormalData] = None*, *factor_data_list: Optional[List[NormalData]] = None*, *sub_data_list: Optional[NormalData] = None*, *sub_col_chart: Optional[dict] = None*, *rects: Optional[List[Rect]] = None*, *annotation_df: Optional[DataFrame] = None*, *scale_value: Optional[int] = None*) → None

Parameters::

- **main_df** – df for main chart
- **factor_df_list** – list of factor df on main chart
- **sub_df_list** – df for sub chart under main char
- **main_data** – NormalData wrap main_df,use either
- **factor_data_list** – list of NormalData wrap factor_df,use either
- **sub_data_list** – NormalData wrap sub_df,use either
- **annotation_df** –

factor_data_list*: List[NormalData]*

每一个df可能有多个column, 代表多个指标，对于连续型的，可以放在一个df里面 对于离散型的，比如一些特定模式的连线，放在多个df里面较好，因为index不同

sub_data_list: *List[NormalData]*

每一个df可能有多个column, 代表多个指标，对于连续型的，可以放在一个df里面 对于离散型的，比如一些特定模式的连线，放在多个df里面较好，因为index不同

sub_col_chart

幅图col对应的图形，line or bar

annotation_df

主图的标记数据

rects

list of rect

Rect

class zvt.contract.drawer.Rect(*x0=None*, *y0=None*, *x1=None*, *y1=None*)

Bases: `Bean`

rect struct with left-bottom(x0, y0), right-top(x1, y1)

__init__(*x0=None*, *y0=None*, *x1=None*, *y1=None*) → None

x0

left-bottom x0

y0

left-bottom y0

x1

right-top x1

y1

right-top y1

StackedDrawer

class zvt.contract.drawer.StackedDrawer(**drawers*)

Bases: `Draw`

__init__(**drawers*) → None

zvt.contract.factor

Classes

`Accumulator`([acc_window])	
`Factor`(data_schema[, entity_schema, ...])	
`FactorMeta`(name, bases, class_dict)	
`Indicator`()	
`ScoreFactor`(data_schema[, entity_schema, ...])	
`Scorer`()	
`TargetType`(value)	An enumeration.
`Transformer`()	

Accumulator

class zvt.contract.factor.Accumulator(*acc_window: int = 1*)

Bases: `Indicator`

__init__(*acc_window: int = 1*) → None

Parameters:: **acc_window** – the window size of acc for computing,default is 1

acc(*input_df: ~pandas.core.frame.DataFrame*, *acc_df: ~pandas.core.frame.DataFrame*, *states: dict) -> (<class 'pandas.core.frame.DataFrame'>*, *<class 'dict'>*)

Parameters::
- **input_df** – new input
- **acc_df** – previous result
- **states** – current states of the entity

Returns:: new result and states

acc_one(*entity_id*, *df: ~pandas.core.frame.DataFrame*, *acc_df: ~pandas.core.frame.DataFrame*, *state: dict) -> (<class 'pandas.core.frame.DataFrame'>*, *<class 'dict'>*)

df format:

```
             col1    col2    col3    ...
timestamp
             1.2     0.5     0.3     ...
             1.0     0.7     0.2     ...
```

the new result and state

Parameters::
- **df** – current input df
- **entity_id** – current computing entity_id
- **acc_df** – current result of the entity_id
- **state** – current state of the entity_id

Returns:: new result and state of the entity_id

Factor

class zvt.contract.factor.Factor(*data_schema: Type[Mixin], entity_schema: Optional[Type[TradableEntity]] = None, provider: Optional[str] = None, entity_provider: Optional[str] = None, entity_ids: Optional[List[str]] = None, exchanges: Optional[List[str]] = None, codes: Optional[List[str]] = None, start_timestamp: Optional[Union[Timestamp, str]] = None, end_timestamp: Optional[Union[Timestamp, str]] = None, columns: Optional[List] = None, filters: Optional[List] = None, order: Optional[object] = None, limit: Optional[int] = None, level: Union[str, IntervalLevel] = IntervalLevel.LEVEL_1DAY, category_field: str = 'entity_id', time_field: str = 'timestamp', keep_window: Optional[int] = None, keep_all_timestamp: bool = False, fill_method: str = 'ffill', effective_number: Optional[int] = None, transformer: Optional[Transformer] = None, accumulator: Optional[Accumulator] = None, need_persist: bool = False, only_compute_factor: bool = False, factor_name: Optional[str] = None, clear_state: bool = False, only_load_factor: bool = False*)

Bases: `DataReader`, `EntityStateService`, `DataListener`

__init__(*data_schema: Type[Mixin], entity_schema: Optional[Type[TradableEntity]] = None, provider: Optional[str] = None, entity_provider: Optional[str] = None, entity_ids: Optional[List[str]] = None, exchanges: Optional[List[str]] = None, codes: Optional[List[str]] = None, start_timestamp: Optional[Union[Timestamp, str]] = None, end_timestamp: Optional[Union[Timestamp, str]] = None, columns: Optional[List] = None, filters: Optional[List] = None, order: Optional[object] = None, limit: Optional[int] = None, level: Union[str, IntervalLevel] = IntervalLevel.LEVEL_1DAY, category_field: str = 'entity_id', time_field: str = 'timestamp', keep_window: Optional[int] = None, keep_all_timestamp: bool = False, fill_method: str = 'ffill', effective_number: Optional[int] = None, transformer: Optional[Transformer] = None, accumulator: Optional[Accumulator] = None, need_persist: bool = False, only_compute_factor: bool = False,*

factor_name: Optional[str] = None, *clear_state: bool = False*, *only_load_factor: bool = False*) → None

Parameters::

- **keep_all_timestamp** –
- **fill_method** –
- **effective_number** –
- **transformer** –
- **accumulator** –
- **need_persist** – whether persist factor
- **only_compute_factor** – only compute factor nor result
- **factor_name** –
- **clear_state** –
- **only_load_factor** – only load factor and compute result

state_schema

Schema for storing states

alias of `FactorState`

factor_schema*: Type[Mixin] = None*

define the schema for persist,its columns should be same as indicators in transformer or accumulator

name = *None*

name of the service, default name of class if not set manually

transformer*: Transformer = None*

transformer for this factor if not passed as __init__ argument

accumulator*: Accumulator = None*

accumulator for this factor if not passed as __init__ argument

pipe_df*: pd.DataFrame*

中间结果，不持久化 data_df->pipe_df

factor_df*: pd.DataFrame*

计算因子的结果，可持久化,通过对pipe_df的计算得到 pipe_df->factor_df

result_df*: pd.DataFrame*

result_df是用于选股的标准df,通过对factor_df的计算得到 factor_df->result_df

factor_col_map_object_hook() → dict

:return:{col:object_hook}

clear_state_data(*entity_id=None*)

clear state of the entity

Parameters:: **entity_id** – entity id

on_data_loaded(*data: DataFrame*)

Parameters:: **data** –

on_data_changed(*data: DataFrame*)

overwrite it for computing after data added

Parameters:: **data** –

on_entity_data_changed(*entity*, *added_data: DataFrame*)

overwrite it for computing after entity data added

Parameters::

- **entity** –
- **added_data** –

decode_state(*state: str*)

decode state

Parameters:: **state** –

Returns::

encode_state(*state: object*)

encode state

Parameters:: **state** –

Returns::

move_on(*to_timestamp: Optional[Union[Timestamp, str]] = None*, *timeout: int = 20*) → object

using continual fetching data in realtime 1)get the data happened before to_timestamp,if not set,get all the data which means to now 2)if computing_window set,the data_df would be cut for saving memory

Parameters::

- **to_timestamp** –
- **timeout** (*int*) –

Returns::

Return type::

states*: dict*

entity_id:state

FactorMeta

class zvt.contract.factor.FactorMeta(*name*, *bases*, *class_dict*)

Bases: **`type`**

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

mro()

Return a type's method resolution order.

Indicator

class zvt.contract.factor.Indicator

Bases: `object`

__init__() → None

ScoreFactor

class zvt.contract.factor.ScoreFactor(*data_schema: Type[Mixin], entity_schema: Optional[Type[TradableEntity]] = None, provider: Optional[str] = None, entity_provider: Optional[str] = None, entity_ids: Optional[List[str]] = None, exchanges: Optional[List[str]] = None, codes: Optional[List[str]] = None, start_timestamp: Optional[Union[Timestamp, str]] = None, end_timestamp: Optional[Union[Timestamp, str]] = None, columns: Optional[List] = None, filters: Optional[List] = None, order: Optional[object] = None, limit: Optional[int] = None, level: Union[str, IntervalLevel] = IntervalLevel.LEVEL_1DAY, category_field: str = 'entity_id', time_field: str = 'timestamp', keep_window: Optional[int] = None, keep_all_timestamp: bool = False, fill_method: str = 'ffill', effective_number: Optional[int] = None, transformer: Optional[Transformer] = None, accumulator: Optional[Accumulator] = None, need_persist: bool = False, only_compute_factor: bool = False, factor_name: Optional[str] = None, clear_state: bool = False, only_load_factor: bool = False*)

Bases: **Factor**

__init__(*data_schema: Type[Mixin], entity_schema: Optional[Type[TradableEntity]] = None, provider: Optional[str] = None, entity_provider: Optional[str] = None, entity_ids: Optional[List[str]] = None, exchanges: Optional[List[str]] = None, codes: Optional[List[str]] = None, start_timestamp: Optional[Union[Timestamp, str]] = None, end_timestamp: Optional[Union[Timestamp, str]] = None, columns: Optional[List] = None, filters: Optional[List] = None, order: Optional[object] = None, limit: Optional[int] = None, level: Union[str, IntervalLevel] = IntervalLevel.LEVEL_1DAY, category_field: str = 'entity_id', time_field: str = 'timestamp', keep_window: Optional[int] = None, keep_all_timestamp: bool = False, fill_method: str = 'ffill', effective_number: Optional[int] = None, transformer: Optional[Transformer] = None, accumulator: Optional[Accumulator] = None, need_persist: bool = False, only_compute_factor: bool = False,*

factor_name: Optional[str] = None, *clear_state: bool = False*, *only_load_factor: bool = False*) → None

Parameters::

- **keep_all_timestamp** –
- **fill_method** –
- **effective_number** –
- **transformer** –
- **accumulator** –
- **need_persist** – whether persist factor
- **only_compute_factor** – only compute factor nor result
- **factor_name** –
- **clear_state** –
- **only_load_factor** – only load factor and compute result

accumulator*: Accumulator = None*

accumulator for this factor if not passed as __init__ argument

clear_state_data(*entity_id=None*)

clear state of the entity

Parameters:: **entity_id** – entity id

decode_state(*state: str*)

decode state

Parameters:: **state** –

Returns::

encode_state(*state: object*)

encode state

Parameters:: **state** –

Returns::

factor_col_map_object_hook() → dict

:return:{col:object_hook}

factor_schema*: Type[Mixin] = None*

define the schema for persist,its columns should be same as indicators in transformer or accumulator

move_on(*to_timestamp: Optional[Union[Timestamp, str]] = None*, *timeout: int = 20*) → object

using continual fetching data in realtime 1)get the data happened before to_timestamp,if not set,get all the data which means to now 2)if computing_window set,the data_df would be cut for saving memory

Parameters::
- **to_timestamp** –
- **timeout** (*int*) –

Returns::

Return type::

name = *None*

name of the service, default name of class if not set manually

on_data_changed(*data: DataFrame*)

overwrite it for computing after data added

Parameters:: **data** –

on_data_loaded(*data: DataFrame*)

Parameters:: **data** –

on_entity_data_changed(*entity*, *added_data: DataFrame*)

overwrite it for computing after entity data added

Parameters::
- **entity** –
- **added_data** –

state_schema

alias of `FactorState`

transformer*: Transformer = None*

transformer for this factor if not passed as __init__ argument

pipe_df*: pd.DataFrame*

中间结果，不持久化 data_df->pipe_df

factor_df*: pd.DataFrame*

计算因子的结果，可持久化,通过对pipe_df的计算得到 pipe_df->factor_df

result_df*: pd.DataFrame*

result_df是用于选股的标准df,通过对factor_df的计算得到 factor_df->result_df

states*: dict*

entity_id:state

Scorer

class zvt.contract.factor.Scorer

Bases: `object`

__init__() → None

score(*input_df: DataFrame*) → DataFrame

Parameters:: **input_df** – current input df

Returns:: df with normal score

TargetType

class zvt.contract.factor.TargetType(*value*)

Bases: `Enum`

An enumeration.

__init__()

Transformer

class zvt.contract.factor.Transformer

Bases: `Indicator`

__init__() → None

transform(*input_df: DataFrame*) → DataFrame

input_df format:

```
                                col1    col2    col3    ...
entity_id    timestamp
                                1.2     0.5     0.3     ...
                                1.0     0.7     0.2     ...
```

the return result would change the columns and keep the format

Parameters:: **input_df** –

Returns::

transform_one(*entity_id: str*, *df: DataFrame*) → DataFrame

df format:

```
                 col1    col2    col3    ...
timestamp
                 1.2     0.5     0.3     ...
                 1.0     0.7     0.2     ...
```

the return result would change the columns and keep the format

Parameters::

- **entity_id** –
- **df** –

Returns::

zvt.contract.model

Classes

`CustomModel`()

`MixinModel`(*, id, entity_id, timestamp)

CustomModel

class zvt.contract.model.CustomModel

Bases: **`BaseModel`**

__init__(**data: Any*) → None

Create a new model by parsing and validating input data from keyword arguments.

Raises [*ValidationError*][pydantic_core.ValidationError] if the input data cannot be validated to form a valid model.

self is explicitly positional-only to allow *self* as a field name.

model_config*: ClassVar[ConfigDict] = {'allow_inf_nan': True, 'from_attributes': True}*

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

copy(*, *include: AbstractSetIntStr | MappingIntStrAny | None = None*, *exclude: AbstractSetIntStr | MappingIntStrAny | None = None*, *update: Dict[str, Any] | None = None*, *deep: bool = False*) → Model

Returns a copy of the model.

!!! warning "Deprecated"

This method is now deprecated; use *model_copy* instead.

If you need *include* or *exclude*, use:

```
`py data = self.model_dump(include=include, exclude=exclude, round_trip=True) data = {**data, **(update or {})} copied = self.model_validate(data) `
```

Args:

include: Optional set or mapping specifying which fields to include in the copied model. exclude: Optional set or mapping specifying which fields to exclude in the copied model. update: Optional dictionary of field-value pairs to override field values in the copied model. deep: If True, the values of fields that are Pydantic models will be deep-copied.

Returns:
A copy of the model with included, excluded and updated fields as specified.

model_computed_fields*: ClassVar[dict[str, ComputedFieldInfo]] = {}*
A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

classmethod model_construct(*_fields_set: set[str] | None = None*, ***values: Any*) → Model
Creates a new instance of the *Model* class with validated data.

Creates a new model setting *__dict__* and *__pydantic_fields_set__* from trusted or pre-validated data. Default values are respected, but no other validation is performed. Behaves as if *Config.extra = 'allow'* was set since it adds all passed values

Args:
_fields_set: The set of field names accepted for the Model instance. values: Trusted or pre-validated data dictionary.

Returns:
A new instance of the *Model* class with validated data.

model_copy(*, *update: dict[str, Any] | None = None*, *deep: bool = False*) → Model
Usage docs: https://docs.pydantic.dev/2.6/concepts/serialization/#model_copy

Returns a copy of the model.

Args:

update: Values to change/add in the new model. Note: the data is not validated
before creating the new model. You should trust this data.

deep: Set to *True* to make a deep copy of the model.

Returns:

New model instance.

model_dump(*, *mode: Literal['json', 'python'] | str = 'python'*, *include: IncEx = None*, *exclude: IncEx = None*, *by_alias: bool = False*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*, *round_trip: bool = False*, *warnings: bool = True*) → dict[str, Any]

Usage docs: https://docs.pydantic.dev/2.6/concepts/serialization/#modelmodel_dump

Generate a dictionary representation of the model, optionally specifying which fields to include or exclude.

Args:

mode: The mode in which *to_python* should run.
If mode is 'json', the output will only contain JSON serializable types. If mode is 'python', the output may contain non-JSON-serializable Python objects.

include: A list of fields to include in the output. exclude: A list of fields to exclude from the output. by_alias: Whether to use the field's alias in the dictionary key if defined. exclude_unset: Whether to exclude fields that have not been explicitly set. exclude_defaults: Whether to exclude fields that are set to their default value. exclude_none: Whether to exclude fields that have a value of *None*. round_trip: If True, dumped values should be valid as input for non-idempotent types such as Json[T].

warnings: Whether to log warnings when invalid fields are encountered.

Returns:

A dictionary representation of the model.

model_dump_json(*, *indent: int | None = None*, *include: IncEx = None*, *exclude: IncEx = None*, *by_alias: bool = False*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*, *round_trip: bool = False*, *warnings: bool = True*) → str

Usage docs: https://docs.pydantic.dev/2.6/concepts/serialization/#modelmodel_dump_json

Generates a JSON representation of the model using Pydantic's *to_json* method.

Args:

indent: Indentation to use in the JSON output. If None is passed, the output will be compact. include: Field(s) to include in the JSON output. exclude: Field(s) to exclude from the JSON output. by_alias: Whether to serialize using field aliases. exclude_unset: Whether to exclude fields that have not been explicitly set. exclude_defaults: Whether to exclude fields that are set to their default value. exclude_none: Whether to exclude fields that have a value of *None*. round_trip: If True, dumped values should be valid as input for non-idempotent types such as Json[T]. warnings: Whether to log warnings when invalid fields are encountered.

Returns:

A JSON string representation of the model.

property model_extra*: dict[str, Any] | None*

Get extra fields set during validation.

Returns:

A dictionary of extra fields, or *None* if *config.extra* is not set to *"allow"*.

model_fields*: ClassVar[dict[str, FieldInfo]] = {}*

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

property model_fields_set*: set[str]*

Returns the set of fields that have been explicitly set on this model instance.

Returns:

A set of strings representing the fields that have been set, i.e. that were not filled from defaults.

classmethod model_json_schema(*by_alias: bool = True*, *ref_template: str = '#/$defs/{model}'*, *schema_generator: type[pydantic.json_schema.GenerateJsonSchema] = <class 'pydantic.json_schema.GenerateJsonSchema'>*, *mode: ~typing.Literal['validation', 'serialization'] = 'validation'*) → dict[str, Any]

Generates a JSON schema for a model class.

Args:

by_alias: Whether to use attribute aliases or not. ref_template: The reference template. schema_generator: To override the logic used to generate the JSON schema, as a subclass of

GenerateJsonSchema with your desired modifications

mode: The mode in which to generate the schema.

Returns:

The JSON schema for the given model class.

classmethod model_parametrized_name(*params: tuple[type[Any], ...]*) → str

Compute the class name for parametrizations of generic classes.

This method can be overridden to achieve a custom naming scheme for generic BaseModels.

Args:

params: Tuple of types of the class. Given a generic class *Model* with 2 type variables and a concrete model *Model[str, int]*, the value *(str, int)* would be passed to *params*.

Returns:

String representing the new class where *params* are passed to *cls* as type variables.

Raises:

TypeError: Raised when trying to generate concrete names for non-generic models.

model_post_init(*_BaseModel__context: Any*) → None

Override this method to perform additional initialization after *__init__* and *model_construct*. This is useful if you want to do some validation that requires the entire model to be initialized.

classmethod model_rebuild(*, *force: bool = False*, *raise_errors: bool = True*, *_parent_namespace_depth: int = 2*, *_types_namespace: Optional[dict[str, Any]] = None*) → bool | None

Try to rebuild the pydantic-core schema for the model.

This may be necessary when one of the annotations is a ForwardRef which could not be resolved during the initial attempt to build the schema, and automatic rebuilding fails.

Args:

force: Whether to force the rebuilding of the model schema, defaults to *False*. raise_errors: Whether to raise errors, defaults

to *True*. _parent_namespace_depth: The depth level of the parent namespace, defaults to 2. _types_namespace: The types namespace, defaults to *None*.

Returns:

Returns *None* if the schema is already "complete" and rebuilding was not required. If rebuilding _was_ required, returns *True* if rebuilding was successful, otherwise *False*.

classmethod model_validate(*obj: Any*, *, *strict: bool | None = None*, *from_attributes: bool | None = None*, *context: dict[str, Any] | None = None*) → Model

Validate a pydantic model instance.

Args:

obj: The object to validate. strict: Whether to enforce types strictly. from_attributes: Whether to extract data from object attributes. context: Additional context to pass to the validator.

Raises:

ValidationError: If the object could not be validated.

Returns:

The validated model instance.

classmethod model_validate_json(*json_data: str | bytes | bytearray*, *, *strict: bool | None = None*, *context: dict[str, Any] | None = None*) → Model

Usage docs: https://docs.pydantic.dev/2.6/concepts/json/#json-parsing

Validate the given JSON data against the Pydantic model.

Args:

json_data: The JSON data to validate. strict: Whether to enforce types strictly. context: Extra variables to pass to the validator.

Returns:

The validated Pydantic model.

Raises:

ValueError: If *json_data* is not a JSON string.

classmethod model_validate_strings(*obj: Any*, *, *strict: bool | None = None*, *context: dict[str, Any] | None = None*) → Model

Validate the given object contains string data against the Pydantic model.

Args:

obj: The object contains string data to validate. strict: Whether to enforce types strictly. context: Extra variables to pass to the validator.

Returns:

The validated Pydantic model.

MixinModel

class zvt.contract.model.MixinModel(*, *id: str*, *entity_id: str*, *timestamp: datetime*)

Bases: `CustomModel`

__init__(**data: Any*) → None

Create a new model by parsing and validating input data from keyword arguments.

Raises [*ValidationError*][pydantic_core.ValidationError] if the input data cannot be validated to form a valid model.

self is explicitly positional-only to allow *self* as a field name.

copy(*, *include: AbstractSetIntStr | MappingIntStrAny | None = None*, *exclude: AbstractSetIntStr | MappingIntStrAny | None = None*, *update: Dict[str, Any] | None = None*, *deep: bool = False*) → Model

Returns a copy of the model.

!!! warning "Deprecated"

This method is now deprecated; use *model_copy* instead.

If you need *include* or *exclude*, use:

```
`py data = self.model_dump(include=include, exclude=exclude, round_trip=True) data = {**data, **(update or {})} copied = self.model_validate(data) `
```

Args:

include: Optional set or mapping specifying which fields to include in the copied model. exclude: Optional set or mapping specifying which fields to exclude in the copied model. update: Optional dictionary of field-value pairs to override field values

in the copied model. deep: If True, the values of fields that are Pydantic models will be deep-copied.

Returns:

A copy of the model with included, excluded and updated fields as specified.

model_computed_fields*: ClassVar[dict[str, ComputedFieldInfo]] = {}*

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

model_config*: ClassVar[ConfigDict] = {'allow_inf_nan': True, 'from_attributes': True}*

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

classmethod model_construct(*_fields_set: set[str] | None = None*, ***values: Any*) → Model

Creates a new instance of the *Model* class with validated data.

Creates a new model setting *__dict__* and *__pydantic_fields_set__* from trusted or pre-validated data. Default values are respected, but no other validation is performed. Behaves as if *Config.extra = 'allow'* was set since it adds all passed values

Args:

_fields_set: The set of field names accepted for the Model instance. values: Trusted or pre-validated data dictionary.

Returns:

A new instance of the *Model* class with validated data.

model_copy(*, *update: dict[str, Any] | None = None*, *deep: bool = False*) → Model

Usage docs: https://docs.pydantic.dev/2.6/concepts/serialization/#model_copy

Returns a copy of the model.

Args:

update: Values to change/add in the new model. Note: the data is not validated
before creating the new model. You should trust this data.

deep: Set to *True* to make a deep copy of the model.

Returns:

New model instance.

model_dump(*, *mode: Literal['json', 'python'] | str = 'python'*, *include: IncEx = None*, *exclude: IncEx = None*, *by_alias: bool = False*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*, *round_trip: bool = False*, *warnings: bool = True*) → dict[str, Any]

Usage docs: https://docs.pydantic.dev/2.6/concepts/serialization/#modelmodel_dump

Generate a dictionary representation of the model, optionally specifying which fields to include or exclude.

Args:

mode: The mode in which *to_python* should run.
If mode is 'json', the output will only contain JSON serializable types. If mode is 'python', the output may contain non-JSON-serializable Python objects.

include: A list of fields to include in the output. exclude: A list of fields to exclude from the output. by_alias: Whether to use the field's alias in the dictionary key if defined. exclude_unset: Whether to exclude fields that have not been explicitly set. exclude_defaults: Whether to exclude fields that are set to their default value. exclude_none: Whether to exclude fields that have a value of *None*. round_trip: If True, dumped values should be valid as input for non-idempotent types such as Json[T].

warnings: Whether to log warnings when invalid fields are encountered.

Returns:

A dictionary representation of the model.

model_dump_json(*, *indent: int | None = None*, *include: IncEx = None*, *exclude: IncEx = None*, *by_alias: bool = False*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*, *round_trip: bool = False*, *warnings: bool = True*) → str

Usage docs: https://docs.pydantic.dev/2.6/concepts/serialization/#modelmodel_dump_json

Generates a JSON representation of the model using Pydantic's *to_json* method.

Args:

indent: Indentation to use in the JSON output. If None is passed, the output will be compact. include: Field(s) to include in the JSON output. exclude: Field(s) to exclude from the JSON output. by_alias: Whether to serialize using field aliases. exclude_unset: Whether to exclude fields that have not been explicitly set. exclude_defaults: Whether to exclude fields that are set to their default value. exclude_none: Whether to exclude fields that have a value of *None*. round_trip: If True, dumped values should be valid as input for non-idempotent types such as Json[T]. warnings: Whether to log warnings when invalid fields are encountered.

Returns:

A JSON string representation of the model.

property model_extra*: dict[str, Any] | None*

Get extra fields set during validation.

Returns:

A dictionary of extra fields, or *None* if *config.extra* is not set to *"allow"*.

model_fields*: ClassVar[dict[str, FieldInfo]] = {'entity_id': FieldInfo(annotation=str, required=True), 'id': FieldInfo(annotation=str, required=True), 'timestamp': FieldInfo(annotation=datetime, required=True)}*

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

property model_fields_set*: set[str]*

Returns the set of fields that have been explicitly set on this model instance.

Returns:

A set of strings representing the fields that have been set, i.e. that were not filled from defaults.

classmethod model_json_schema(*by_alias: bool = True*, *ref_template: str = '#/$defs/{model}'*, *schema_generator: type[pydantic.json_schema.GenerateJsonSchema] = <class 'pydantic.json_schema.GenerateJsonSchema'>*, *mode: ~typing.Literal['validation', 'serialization'] = 'validation'*) → dict[str, Any]

Generates a JSON schema for a model class.

Args:

by_alias: Whether to use attribute aliases or not. ref_template: The reference template. schema_generator: To override the logic used to generate the JSON schema, as a subclass of

GenerateJsonSchema with your desired modifications

mode: The mode in which to generate the schema.

Returns:

The JSON schema for the given model class.

classmethod model_parametrized_name(*params: tuple[type[Any], ...]*) → str

Compute the class name for parametrizations of generic classes.

This method can be overridden to achieve a custom naming scheme for generic BaseModels.

Args:

params: Tuple of types of the class. Given a generic class *Model* with 2 type variables and a concrete model *Model[str, int]*, the value *(str, int)* would be passed to *params*.

Returns:

String representing the new class where *params* are passed to *cls* as type variables.

Raises:

TypeError: Raised when trying to generate concrete names for non-generic models.

model_post_init(*_BaseModel__context: Any*) → None

Override this method to perform additional initialization after *__init__* and *model_construct*. This is useful if you want to do some validation that requires the entire model to be initialized.

classmethod model_rebuild(*, *force: bool = False*, *raise_errors: bool = True*, *_parent_namespace_depth: int = 2*, *_types_namespace: Optional[dict[str, Any]] = None*) → bool | None

Try to rebuild the pydantic-core schema for the model.

This may be necessary when one of the annotations is a ForwardRef which could not be resolved during the initial attempt to build the schema, and automatic rebuilding fails.

Args:

force: Whether to force the rebuilding of the model schema, defaults to *False*. raise_errors: Whether to raise errors, defaults to *True*. _parent_namespace_depth: The depth level of the parent namespace, defaults to 2. _types_namespace: The types namespace, defaults to *None*.

Returns:
Returns *None* if the schema is already "complete" and rebuilding was not required. If rebuilding _was_ required, returns *True* if rebuilding was successful, otherwise *False*.

classmethod model_validate(*obj: Any*, *, *strict: bool | None = None*, *from_attributes: bool | None = None*, *context: dict[str, Any] | None = None*) → Model

Validate a pydantic model instance.

Args:
obj: The object to validate. strict: Whether to enforce types strictly. from_attributes: Whether to extract data from object attributes. context: Additional context to pass to the validator.

Raises:
ValidationError: If the object could not be validated.

Returns:
The validated model instance.

classmethod model_validate_json(*json_data: str | bytes | bytearray*, *, *strict: bool | None = None*, *context: dict[str, Any] | None = None*) → Model

Usage docs: https://docs.pydantic.dev/2.6/concepts/json/#json-parsing

Validate the given JSON data against the Pydantic model.

Args:

json_data: The JSON data to validate. strict: Whether to enforce types strictly. context: Extra variables to pass to the validator.

Returns:
The validated Pydantic model.

Raises:
ValueError: If *json_data* is not a JSON string.

classmethod model_validate_strings(*obj: Any*, *, *strict: bool | None = None*, *context: dict[str, Any] | None = None*) → Model

Validate the given object contains string data against the Pydantic model.

Args:
obj: The object contains string data to validate. strict: Whether to enforce types strictly. context: Extra variables to pass to the validator.

Returns:
The validated Pydantic model.

zvt.contract.normal_data

Classes

`NormalData`(df[, category_field, time_field, ...])

NormalData

class zvt.contract.normal_data.NormalData(*df*, *category_field='entity_id'*, *time_field='timestamp'*, *fill_index: bool = False*)

Bases: **`object`**

__init__(*df*, *category_field='entity_id'*, *time_field='timestamp'*, *fill_index: bool = False*) → None

normalize()

normalize data_df to:

```
                                  col1    col2    col3
entity_id    timestamp
```

zvt.contract.reader

Classes

`DataListener`()

`DataReader`(data_schema[, entity_schema, ...])

DataListener

class zvt.contract.reader.DataListener

Bases: `object`

__init__()

on_data_loaded(*data: DataFrame*) → object

Parameters:: **data** –

on_data_changed(*data: DataFrame*) → object

Parameters:: **data** –

on_entity_data_changed(*entity: str*, *added_data: DataFrame*) → object

Parameters::
- **entity** – the entity
- **added_data** – the data added for the entity

DataReader

class zvt.contract.reader.DataReader(*data_schema: Type[Mixin], entity_schema: Optional[Type[TradableEntity]] = None, provider: Optional[str] = None, entity_provider: Optional[str] = None, entity_ids: Optional[List[str]] = None, exchanges: Optional[List[str]] = None, codes: Optional[List[str]] = None, start_timestamp: Optional[Union[Timestamp, str]] = None, end_timestamp: Union[str, Timestamp] = Timestamp('2024-08-28 22:08:13.878693'), columns: Optional[List] = None, filters: Optional[List] = None, order: Optional[object] = None, limit: Optional[int] = None, level: Optional[IntervalLevel] = None, category_field: str = 'entity_id', time_field: str = 'timestamp', keep_window: Optional[int] = None*)

Bases: **Drawable**

__init__(*data_schema: Type[Mixin], entity_schema: Optional[Type[TradableEntity]] = None, provider: Optional[str] = None, entity_provider: Optional[str] = None, entity_ids: Optional[List[str]] = None, exchanges: Optional[List[str]] = None, codes: Optional[List[str]] = None, start_timestamp: Optional[Union[Timestamp, str]] = None, end_timestamp: Union[str, Timestamp] = Timestamp('2024-08-28 22:08:13.878693'), columns: Optional[List] = None, filters: Optional[List] = None, order: Optional[object] = None, limit: Optional[int] = None, level: Optional[IntervalLevel] = None, category_field: str = 'entity_id', time_field: str = 'timestamp', keep_window: Optional[int] = None*) → None

move_on(*to_timestamp: Optional[Union[Timestamp, str]] = None, timeout: int = 20*) → object

using continual fetching data in realtime 1)get the data happened before to_timestamp,if not set,get all the data which means to now 2)if computing_window set,the data_df would be cut for saving memory

Parameters::

- **to_timestamp** –
- **timeout** (*int*) –

Returns::

Return type::

zvt.contract.recorder

Classes

`EntityEventRecorder`([force_update, ...])

`FixedCycleDataRecorder`([force_update, ...])

`Meta`(name, bases, class_dict)

`Recorder`([force_update, sleeping_time])

`TimeSeriesDataRecorder`([force_update, ...])

`TimestampsDataRecorder`([force_update, ...])

EntityEventRecorder

class zvt.contract.recorder.EntityEventRecorder(*force_update=False*, *sleeping_time=10*, *exchanges=None*, *entity_id=None*, *entity_ids=None*, *code=None*, *codes=None*, *day_data=False*, *entity_filters=None*, *ignore_failed=True*, *return_unfinished=False*)

Bases: `Recorder`

__init__(*force_update=False*, *sleeping_time=10*, *exchanges=None*, *entity_id=None*, *entity_ids=None*, *code=None*, *codes=None*, *day_data=False*, *entity_filters=None*, *ignore_failed=True*, *return_unfinished=False*) → None

Parameters::

- **code** –
- **ignore_failed** –
- **entity_filters** –
- **exchanges** –
- **entity_id** – for record single entity
- **entity_ids** – set entity_ids or (entity_type,exchanges,codes)
- **codes** –
- **day_data** – one record per day,set to True if you want skip recording it when data of today exist
- **force_update** –
- **sleeping_time** –

entity_provider*: str = None*

overwrite them to fetch the entity list

exchanges

setup the entities you want to record

entity_ids

set entity_ids or (entity_type,exchanges,codes)

init_entities()

init the entities which we would record data for

clear_state_data(*entity_id=None*)

clear state of the entity

Parameters:: **entity_id** – entity id

decode_state(*state: str*)

decode state

Parameters:: **state** –

Returns::

encode_state(*state: object*)

encode state

Parameters:: **state** –

Returns::

name = *None*

name of the service, default name of class if not set manually

original_page_url = *None*

original page url

provider*: str* = *None*

overwrite them to set up the data you want to record

state_schema

alias of `RecorderState`

url = *None*

request url

session

using to do db operations

FixedCycleDataRecorder

class zvt.contract.recorder.FixedCycleDataRecorder(*force_update=True*, *sleeping_time=10*, *exchanges=None*, *entity_id=None*, *entity_ids=None*, *code=None*, *codes=None*, *day_data=False*, *entity_filters=None*, *ignore_failed=True*, *real_time=False*, *fix_duplicate_way='ignore'*, *start_timestamp=None*, *end_timestamp=None*, *level=IntervalLevel.LEVEL_1DAY*, *kdata_use_begin_time=False*, *one_day_trading_minutes=1440*, *return_unfinished=False*)

Bases: `TimeSeriesDataRecorder`

__init__(*force_update=True*, *sleeping_time=10*, *exchanges=None*, *entity_id=None*, *entity_ids=None*, *code=None*, *codes=None*, *day_data=False*, *entity_filters=None*, *ignore_failed=True*, *real_time=False*, *fix_duplicate_way='ignore'*, *start_timestamp=None*, *end_timestamp=None*, *level=IntervalLevel.LEVEL_1DAY*, *kdata_use_begin_time=False*, *one_day_trading_minutes=1440*, *return_unfinished=False*) → None

Parameters::

- **code** –
- **ignore_failed** –
- **entity_filters** –
- **exchanges** –
- **entity_id** – for record single entity
- **entity_ids** – set entity_ids or (entity_type,exchanges,codes)
- **codes** –
- **day_data** – one record per day,set to True if you want skip recording it when data of today exist
- **force_update** –
- **sleeping_time** –

clear_state_data(*entity_id=None*)

clear state of the entity

Parameters:: **entity_id** – entity id

decode_state(*state: str*)

decode state

Parameters:: **state** –
Returns::

encode_state(*state: object*)

encode state

Parameters:: **state** –
Returns::

entity_provider*: str = None*

overwrite them to fetch the entity list

generate_domain(*entity*, *original_data*)

generate the data_schema instance using entity and original_data,the original_data is from record result

Parameters::
- **entity** –
- **original_data** –

generate_domain_id(*entity*, *original_data*, *time_fmt='YYYY-MM-DD'*)

generate domain id from the entity and original data,the default id meaning:entity + event happen time

Parameters::
- **entity** –
- **original_data** –
- **time_fmt** –

Returns::
Return type::

get_data_map()

{‘original_field’:(‘domain_field’,transform_func)}

get_evaluated_time_field()

the timestamp field for evaluating time range of recorder,used in get_latest_saved_record

init_entities()

init the entities which we would record data for

name = *None*

name of the service, default name of class if not set manually

original_page_url = *None*

original page url

persist(*entity*, *domain_list*)

persist the domain list to db

Parameters::

- **entity** –
- **domain_list** –

provider*: str = None*

overwrite them to set up the data you want to record

record(*entity*, *start*, *end*, *size*, *timestamps*)

implement the recording logic in this method, should return json or domain list

Parameters::

- **entity** –
- **start** –
- **end** –
- **size** –
- **timestamps** –

state_schema

alias of `RecorderState`

url = *None*

request url

exchanges

setup the entities you want to record

entity_ids

set entity_ids or (entity_type,exchanges,codes)

session

using to do db operations

Meta

class zvt.contract.recorder.Meta(*name*, *bases*, *class_dict*)

Bases: **`type`**

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

mro()

Return a type's method resolution order.

Recorder

class zvt.contract.recorder.Recorder(*force_update: bool = False*, *sleeping_time: int = 10*)

Bases: `OneStateService`

__init__(*force_update: bool = False*, *sleeping_time: int = 10*) → None

provider*: str = None*

overwrite them to set up the data you want to record

original_page_url = *None*

original page url

url = *None*

request url

state_schema

alias of `RecorderState`

session

using to do db operations

clear_state_data(*entity_id=None*)

clear state of the entity

Parameters:: **entity_id** – entity id

decode_state(*state: str*)

decode state

Parameters:: **state** –

Returns::

encode_state(*state: object*)

encode state

Parameters:: **state** –

Returns::

name = *None*

name of the service, default name of class if not set manually

TimeSeriesDataRecorder

class zvt.contract.recorder.TimeSeriesDataRecorder(*force_update=False*, *sleeping_time=5*, *exchanges=None*, *entity_id=None*, *entity_ids=None*, *code=None*, *codes=None*, *day_data=False*, *entity_filters=None*, *ignore_failed=True*, *real_time=False*, *fix_duplicate_way='add'*, *start_timestamp=None*, *end_timestamp=None*, *return_unfinished=False*)

Bases: `EntityEventRecorder`

__init__(*force_update=False*, *sleeping_time=5*, *exchanges=None*, *entity_id=None*, *entity_ids=None*, *code=None*, *codes=None*, *day_data=False*, *entity_filters=None*, *ignore_failed=True*, *real_time=False*, *fix_duplicate_way='add'*, *start_timestamp=None*, *end_timestamp=None*, *return_unfinished=False*) → None

Parameters::

- **code** –
- **ignore_failed** –
- **entity_filters** –
- **exchanges** –
- **entity_id** – for record single entity
- **entity_ids** – set entity_ids or (entity_type,exchanges,codes)
- **codes** –
- **day_data** – one record per day,set to True if you want skip recording it when data of today exist
- **force_update** –
- **sleeping_time** –

get_data_map()

{'original_field':('domain_field',transform_func)}

record(*entity*, *start*, *end*, *size*, *timestamps*)

implement the recording logic in this method, should return json or domain list

Parameters::

- **entity** –
- **start** –
- **end** –
- **size** –
- **timestamps** –

get_evaluated_time_field()

the timestamp field for evaluating time range of recorder,used in get_latest_saved_record

generate_domain_id(*entity*, *original_data*, *time_fmt='YYYY-MM-DD'*)

generate domain id from the entity and original data,the default id meaning:entity + event happen time

Parameters::

- **entity** –
- **original_data** –
- **time_fmt** –

Returns::

Return type::

generate_domain(*entity*, *original_data*)

generate the data_schema instance using entity and original_data,the original_data is from record result

Parameters::

- **entity** –
- **original_data** –

persist(*entity*, *domain_list*)

persist the domain list to db

Parameters::

- **entity** –
- **domain_list** –

clear_state_data(*entity_id=None*)

clear state of the entity

Parameters:: **entity_id** – entity id

decode_state(*state: str*)

decode state

Parameters:: **state** –
Returns::

encode_state(*state: object*)

encode state

Parameters:: **state** –
Returns::

entity_provider*: str = None*

overwrite them to fetch the entity list

init_entities()

init the entities which we would record data for

name = *None*

name of the service, default name of class if not set manually

original_page_url = *None*

original page url

provider*: str = None*

overwrite them to set up the data you want to record

state_schema

alias of `RecorderState`

url = *None*

request url

exchanges

setup the entities you want to record

entity_ids

set entity_ids or (entity_type,exchanges,codes)

session

using to do db operations

TimestampsDataRecorder

class zvt.contract.recorder.TimestampsDataRecorder(*force_update=False*, *sleeping_time=5*, *exchanges=None*, *entity_id=None*, *entity_ids=None*, *code=None*, *codes=None*, *day_data=False*, *entity_filters=None*, *ignore_failed=True*, *real_time=False*, *fix_duplicate_way='add'*, *start_timestamp=None*, *end_timestamp=None*)

Bases: `TimeSeriesDataRecorder`

__init__(*force_update=False*, *sleeping_time=5*, *exchanges=None*, *entity_id=None*, *entity_ids=None*, *code=None*, *codes=None*, *day_data=False*, *entity_filters=None*, *ignore_failed=True*, *real_time=False*, *fix_duplicate_way='add'*, *start_timestamp=None*, *end_timestamp=None*) → None

Parameters::

- **code** –
- **ignore_failed** –
- **entity_filters** –
- **exchanges** –
- **entity_id** – for record single entity
- **entity_ids** – set entity_ids or (entity_type,exchanges,codes)
- **codes** –
- **day_data** – one record per day,set to True if you want skip recording it when data of today exist
- **force_update** –
- **sleeping_time** –

clear_state_data(*entity_id=None*)

clear state of the entity

Parameters:: **entity_id** – entity id

decode_state(*state: str*)

decode state

Parameters:: **state** –
Returns::

encode_state(*state: object*)

encode state

Parameters:: **state** –
Returns::

entity_provider*: str = None*

overwrite them to fetch the entity list

generate_domain(*entity*, *original_data*)

generate the data_schema instance using entity and original_data,the original_data is from record result

Parameters::
- **entity** –
- **original_data** –

generate_domain_id(*entity*, *original_data*, *time_fmt='YYYY-MM-DD'*)

generate domain id from the entity and original data,the default id meaning:entity + event happen time

Parameters::
- **entity** –
- **original_data** –
- **time_fmt** –

Returns::
Return type::

get_data_map()

{‘original_field’:(‘domain_field’,transform_func)}

get_evaluated_time_field()

the timestamp field for evaluating time range of recorder,used in get_latest_saved_record

init_entities()

init the entities which we would record data for

name = *None*

name of the service, default name of class if not set manually

original_page_url = *None*

original page url

persist(*entity*, *domain_list*)

persist the domain list to db

Parameters::

- **entity** –
- **domain_list** –

provider*: str = None*

overwrite them to set up the data you want to record

record(*entity*, *start*, *end*, *size*, *timestamps*)

implement the recording logic in this method, should return json or domain list

Parameters::

- **entity** –
- **start** –
- **end** –
- **size** –
- **timestamps** –

state_schema

alias of `RecorderState`

url = *None*

request url

exchanges

setup the entities you want to record

entity_ids

set entity_ids or (entity_type,exchanges,codes)

session

using to do db operations

zvt.contract.register

Functions

register_entity([entity_type])	function for register entity type
register_schema(providers, db_name, schema_base)	function for register schema,please declare them before register

zvt.contract.register.register_entity

zvt.contract.register.register_entity(*entity_type: Optional[str] = None*)

function for register entity type

Parameters:: **entity_type** –

Returns::

Return type::

zvt.contract.register.register_schema

zvt.contract.register.register_schema(*providers: List[str]*, *db_name: str*, *schema_base: DeclarativeMeta*, *entity_type: Optional[str] = None*)

function for register schema,please declare them before register

Parameters::

- **providers** – the supported providers for the schem
- **db_name** – database name for the schema
- **schema_base** –
- **entity_type** – the schema related entity_type

Returns::

Return type::

zvt.contract.schema

Classes

ActorEntity()	
ActorMeetTradable()	
Entity()	
Mixin()	Base class of schema.
NormalEntityMixin()	
NormalMixin()	
Portfolio()	composition of tradable entities
PortfolioStock()	
PortfolioStockHistory()	
TradableEntity()	tradable entity
TradableMeetActor()	

ActorEntity

class zvt.contract.schema.ActorEntity

Bases: `Entity`

__init__()

code = *Column(None, String(length=64), table=None)*

编码

end_date = *Column(None, DateTime(), table=None)*

退市日

entity_id = *Column(None, String(), table=None)*

entity id

entity_type = *Column(None, String(length=64), table=None)*

标的类型

exchange = *Column(None, String(length=32), table=None)*

所属交易所

classmethod get_providers() → List[str]

providers of the schema defined by cls

Returns:: providers

classmethod get_storages(*provider: Optional[str] = None*)

get the storages info

Parameters:: **provider** – provider

Returns:: storages

id = *Column(None, String(), table=None, primary_key=True, nullable=False)*

id

list_date = *Column(None, DateTime(), table=None)*

上市日

name = *Column(None, String(length=128), table=None)*

名字

classmethod query_data(*provider_index: int = 0*, *ids: Optional[List[str]] = None*, *entity_ids: Optional[List[str]] = None*, *entity_id: Optional[str] = None*, *codes: Optional[List[str]] = None*, *code: Optional[str] = None*, *level: Optional[Union[IntervalLevel, str]] = None*, *provider: Optional[str] = None*, *columns: Optional[List] = None*, *col_label: Optional[dict] = None*, *return_type: str = 'df'*, *start_timestamp: Optional[Union[Timestamp, str]] = None*, *end_timestamp: Optional[Union[Timestamp, str]] = None*, *filters: Optional[List] = None*, *session: Optional[Session] = None*, *order=None*, *limit: Optional[int] = None*, *distinct=None*, *index: Optional[Union[str, list]] = None*, *drop_index_col=False*, *time_field: str = 'timestamp'*)

query data by the arguments

Parameters::

- **provider_index** –
- **data_schema** –
- **ids** –
- **entity_ids** –
- **entity_id** –
- **codes** –
- **code** –
- **level** –
- **provider** –
- **columns** –
- **col_label** – dict with key(column), value(label
- **return_type** – df, domain or dict. default is df
- **start_timestamp** –
- **end_timestamp** –
- **filters** –
- **session** –
- **order** –
- **limit** –
- **index** – index field name, str for single index, str list for multiple index
- **drop_index_col** – whether drop the col if it's in index, default False
- **time_field** –

Returns:: results basing on return_type.

classmethod record_data(*provider_index: int = 0*, *provider: Optional[str] = None*, *force_update=None*, *sleeping_time=None*, *exchanges=None*, *entity_id=None*, *entity_ids=None*, *code=None*, *codes=None*, *real_time=None*, *fix_duplicate_way=None*, *start_timestamp=None*, *end_timestamp=None*, *one_day_trading_minutes=None*, ***kwargs*)

record data by the arguments

Parameters::

- **entity_id** –
- **provider_index** –
- **provider** –
- **force_update** –
- **sleeping_time** –
- **exchanges** –
- **entity_ids** –
- **code** –
- **codes** –
- **real_time** –
- **fix_duplicate_way** –
- **start_timestamp** –
- **end_timestamp** –
- **one_day_trading_minutes** –
- **kwargs** –

Returns::

classmethod register_provider(*provider*)

register the provider to the schema defined by cls

Parameters:: **provider** –

classmethod register_recorder_cls(*provider*, *recorder_cls*)

register the recorder for the schema

Parameters::

- **provider** –
- **recorder_cls** –

timestamp = *Column(None, DateTime(), table=None)*

the meaning could be different for different case,most time it means 'happen time'

ActorMeetTradable

class zvt.contract.schema.ActorMeetTradable

Bases: `Mixin`

__init__()

code = *Column(None, String(length=64), table=None)*

actor code

name = *Column(None, String(length=128), table=None)*

actor name

entity_id = *Column(None, String(), table=None)*

entity id

classmethod get_providers() → List[str]

providers of the schema defined by cls

Returns:: providers

classmethod get_storages(*provider: Optional[str] = None*)

get the storages info

Parameters:: **provider** – provider
Returns:: storages

id = *Column(None, String(), table=None, primary_key=True, nullable=False)*

id

classmethod query_data(*provider_index: int = 0*, *ids: Optional[List[str]] = None*, *entity_ids: Optional[List[str]] = None*, *entity_id: Optional[str] = None*, *codes: Optional[List[str]] = None*, *code: Optional[str] = None*, *level: Optional[Union[IntervalLevel, str]] = None*, *provider: Optional[str] = None*, *columns: Optional[List] = None*, *col_label:*

Optional[dict] = None, *return_type: str = 'df'*, *start_timestamp: Optional[Union[Timestamp, str]] = None*, *end_timestamp: Optional[Union[Timestamp, str]] = None*, *filters: Optional[List] = None*, *session: Optional[Session] = None*, *order=None*, *limit: Optional[int] = None*, *distinct=None*, *index: Optional[Union[str, list]] = None*, *drop_index_col=False*, *time_field: str = 'timestamp'*)

query data by the arguments

Parameters::

- **provider_index** –
- **data_schema** –
- **ids** –
- **entity_ids** –
- **entity_id** –
- **codes** –
- **code** –
- **level** –
- **provider** –
- **columns** –
- **col_label** – dict with key(column), value(label
- **return_type** – df, domain or dict. default is df
- **start_timestamp** –
- **end_timestamp** –
- **filters** –
- **session** –
- **order** –
- **limit** –
- **index** – index field name, str for single index, str list for multiple index
- **drop_index_col** – whether drop the col if it's in index, default False
- **time_field** –

Returns:: results basing on return_type.

classmethod record_data(*provider_index: int = 0*, *provider: Optional[str] = None*, *force_update=None*, *sleeping_time=None*, *exchanges=None*, *entity_id=None*, *entity_ids=None*, *code=None*, *codes=None*, *real_time=None*, *fix_duplicate_way=None*, *start_timestamp=None*, *end_timestamp=None*, *one_day_trading_minutes=None*, ***kwargs*)

record data by the arguments

Parameters::

- **entity_id** –
- **provider_index** –
- **provider** –
- **force_update** –
- **sleeping_time** –
- **exchanges** –
- **entity_ids** –
- **code** –
- **codes** –
- **real_time** –
- **fix_duplicate_way** –
- **start_timestamp** –
- **end_timestamp** –
- **one_day_trading_minutes** –
- **kwargs** –

Returns::

classmethod register_provider(*provider*)

register the provider to the schema defined by cls

Parameters:: **provider** –

classmethod register_recorder_cls(*provider*, *recorder_cls*)

register the recorder for the schema

Parameters::

- **provider** –
- **recorder_cls** –

timestamp = *Column(None, DateTime(), table=None)*

the meaning could be different for different case,most time it means 'happen time'

Entity

class zvt.contract.schema.Entity

Bases: `Mixin`

__init__()

entity_type = *Column(None, String(length=64), table=None)*

标的类型

exchange = *Column(None, String(length=32), table=None)*

所属交易所

code = *Column(None, String(length=64), table=None)*

编码

name = *Column(None, String(length=128), table=None)*

名字

list_date = *Column(None, DateTime(), table=None)*

上市日

end_date = *Column(None, DateTime(), table=None)*

退市日

entity_id = *Column(None, String(), table=None)*

entity id

classmethod get_providers() → List[str]

providers of the schema defined by cls

Returns:: providers

classmethod get_storages(*provider: Optional[str] = None*)

get the storages info

Parameters:: **provider** – provider
Returns:: storages

id = *Column(None, String(), table=None, primary_key=True, nullable=False)*

id

classmethod query_data(*provider_index: int = 0*, *ids: Optional[List[str]] = None*, *entity_ids: Optional[List[str]] = None*, *entity_id: Optional[str] = None*, *codes: Optional[List[str]] = None*, *code: Optional[str] = None*, *level: Optional[Union[IntervalLevel, str]] = None*, *provider: Optional[str] = None*, *columns: Optional[List] = None*, *col_label: Optional[dict] = None*, *return_type: str = 'df'*, *start_timestamp: Optional[Union[Timestamp, str]] = None*, *end_timestamp: Optional[Union[Timestamp, str]] = None*, *filters: Optional[List] = None*, *session: Optional[Session] = None*, *order=None*, *limit: Optional[int] = None*, *distinct=None*, *index: Optional[Union[str, list]] = None*, *drop_index_col=False*, *time_field: str = 'timestamp'*)

query data by the arguments

Parameters::

- **provider_index** –
- **data_schema** –
- **ids** –
- **entity_ids** –
- **entity_id** –
- **codes** –
- **code** –
- **level** –
- **provider** –
- **columns** –
- **col_label** – dict with key(column), value(label
- **return_type** – df, domain or dict. default is df
- **start_timestamp** –
- **end_timestamp** –
- **filters** –
- **session** –
- **order** –
- **limit** –
- **index** – index field name, str for single index, str list for multiple index
- **drop_index_col** – whether drop the col if it's in index, default False
- **time_field** –

Returns:: results basing on return_type.

classmethod record_data(*provider_index: int = 0*, *provider: Optional[str] = None*, *force_update=None*, *sleeping_time=None*, *exchanges=None*, *entity_id=None*, *entity_ids=None*, *code=None*, *codes=None*, *real_time=None*, *fix_duplicate_way=None*, *start_timestamp=None*, *end_timestamp=None*, *one_day_trading_minutes=None*, ***kwargs*)

record data by the arguments

Parameters::

- **entity_id** –
- **provider_index** –
- **provider** –
- **force_update** –
- **sleeping_time** –
- **exchanges** –
- **entity_ids** –
- **code** –
- **codes** –
- **real_time** –
- **fix_duplicate_way** –
- **start_timestamp** –
- **end_timestamp** –
- **one_day_trading_minutes** –
- **kwargs** –

Returns::

classmethod register_provider(*provider*)

register the provider to the schema defined by cls

Parameters:: **provider** –

classmethod register_recorder_cls(*provider*, *recorder_cls*)

register the recorder for the schema

Parameters::

- **provider** –
- **recorder_cls** –

timestamp = *Column(None, DateTime(), table=None)*

the meaning could be different for different case,most time it means 'happen time'

Mixin

class zvt.contract.schema.Mixin

Bases: `object`

Base class of schema.

__init__()

id = *Column(None, String(), table=None, primary_key=True, nullable=False)*

id

entity_id = *Column(None, String(), table=None)*

entity id

timestamp = *Column(None, DateTime(), table=None)*

the meaning could be different for different case,most time it means 'happen time'

classmethod register_recorder_cls(*provider*, *recorder_cls*)

register the recorder for the schema

Parameters::
- **provider** –
- **recorder_cls** –

classmethod register_provider(*provider*)

register the provider to the schema defined by cls

Parameters:: **provider** –

classmethod get_providers() → List[str]

providers of the schema defined by cls

Returns:: providers

classmethod query_data(*provider_index: int = 0*, *ids: Optional[List[str]] = None*, *entity_ids: Optional[List[str]] = None*, *entity_id: Optional[str] = None*, *codes: Optional[List[str]] = None*, *code: Optional[str] = None*, *level: Optional[Union[IntervalLevel, str]] = None*, *provider: Optional[str] = None*, *columns: Optional[List] = None*, *col_label: Optional[dict] = None*, *return_type: str = 'df'*, *start_timestamp: Optional[Union[Timestamp, str]] = None*, *end_timestamp: Optional[Union[Timestamp, str]] = None*, *filters: Optional[List] = None*, *session: Optional[Session] = None*, *order=None*, *limit: Optional[int] = None*, *distinct=None*, *index: Optional[Union[str, list]] = None*, *drop_index_col=False*, *time_field: str = 'timestamp'*)

query data by the arguments

Parameters::

- **provider_index** –
- **data_schema** –
- **ids** –
- **entity_ids** –
- **entity_id** –
- **codes** –
- **code** –
- **level** –
- **provider** –
- **columns** –
- **col_label** – dict with key(column), value(label
- **return_type** – df, domain or dict. default is df
- **start_timestamp** –
- **end_timestamp** –
- **filters** –
- **session** –
- **order** –
- **limit** –
- **index** – index field name, str for single index, str list for multiple index
- **drop_index_col** – whether drop the col if it's in index, default False
- **time_field** –

Returns:: results basing on return_type.

classmethod get_storages(*provider: Optional[str] = None*)

get the storages info

Parameters:: **provider** – provider

Returns:: storages

classmethod record_data(*provider_index: int = 0*, *provider: Optional[str] = None*, *force_update=None*, *sleeping_time=None*, *exchanges=None*, *entity_id=None*, *entity_ids=None*, *code=None*, *codes=None*, *real_time=None*, *fix_duplicate_way=None*, *start_timestamp=None*, *end_timestamp=None*, *one_day_trading_minutes=None*, ***kwargs*)

record data by the arguments

Parameters::

- **entity_id** –
- **provider_index** –
- **provider** –
- **force_update** –
- **sleeping_time** –
- **exchanges** –
- **entity_ids** –
- **code** –
- **codes** –
- **real_time** –
- **fix_duplicate_way** –
- **start_timestamp** –
- **end_timestamp** –
- **one_day_trading_minutes** –
- **kwargs** –

Returns::

NormalEntityMixin

class zvt.contract.schema.NormalEntityMixin

Bases: `TradableEntity`

__init__()

created_timestamp = *Column(None, DateTime(), table=None, default=ScalarElementColumnDefault(Timestamp('2024-08-28 22:08:09.719515')))*

the record created time in db

updated_timestamp = *Column(None, DateTime(), table=None)*

the record updated time in db, some recorder would check it for whether need to refresh

code = *Column(None, String(length=64), table=None)*

编码

classmethod could_short()

whether could be shorted

Returns::

end_date = *Column(None, DateTime(), table=None)*

退市日

entity_id = *Column(None, String(), table=None)*

entity id

entity_type = *Column(None, String(length=64), table=None)*

标的类型

exchange = *Column(None, String(length=32), table=None)*

所属交易所

classmethod get_interval_timestamps(*start_date*, *end_date*, *level: IntervalLevel*)

generate the timestamps for the level

Parameters::
- **start_date** –
- **end_date** –
- **level** –

classmethod get_providers() → List[str]

providers of the schema defined by cls

Returns:: providers

classmethod get_storages(*provider: Optional[str] = None*)

get the storages info

Parameters:: **provider** – provider

Returns:: storages

classmethod get_trading_dates(*start_date=None*, *end_date=None*)

overwrite it to get the trading dates of the entity

Parameters::
- **start_date** –
- **end_date** –

Returns:: list of dates

classmethod get_trading_intervals()

overwrite it to get the trading intervals of the entity

Returns:: list of time intervals, in format [(start,end)]

classmethod get_trading_t()

0 means t+0 1 means t+1

Returns::

id = *Column(None, String(), table=None, primary_key=True, nullable=False)*

id

classmethod is_finished_kdata_timestamp(*timestamp: Timestamp*, *level: IntervalLevel*)

Parameters::
- **timestamp** (*pd.Timestamp*) – the timestamp could be recorded in kdata of the level
- **level** (*zvt.domain.common.IntervalLevel*) –

Returns::

Return type:: bool

list_date = *Column(None, DateTime(), table=None)*

上市日

name = *Column(None, String(length=128), table=None)*

名字

classmethod query_data(*provider_index: int = 0*, *ids: Optional[List[str]] = None*, *entity_ids: Optional[List[str]] = None*, *entity_id: Optional[str] = None*, *codes: Optional[List[str]] = None*, *code: Optional[str] = None*, *level: Optional[Union[IntervalLevel, str]] = None*, *provider: Optional[str] = None*, *columns: Optional[List] = None*, *col_label: Optional[dict] = None*, *return_type: str = 'df'*, *start_timestamp: Optional[Union[Timestamp, str]] = None*, *end_timestamp: Optional[Union[Timestamp, str]] = None*, *filters: Optional[List] = None*, *session: Optional[Session] = None*, *order=None*, *limit: Optional[int] = None*, *distinct=None*, *index: Optional[Union[str, list]] = None*, *drop_index_col=False*, *time_field: str = 'timestamp'*)

query data by the arguments

Parameters::

- **provider_index** –
- **data_schema** –
- **ids** –
- **entity_ids** –
- **entity_id** –
- **codes** –
- **code** –
- **level** –
- **provider** –
- **columns** –
- **col_label** – dict with key(column), value(label
- **return_type** – df, domain or dict. default is df
- **start_timestamp** –
- **end_timestamp** –
- **filters** –
- **session** –
- **order** –
- **limit** –
- **index** – index field name, str for single index, str list for multiple index
- **drop_index_col** – whether drop the col if it's in index, default False
- **time_field** –

Returns:: results basing on return_type.

classmethod record_data(*provider_index: int = 0*, *provider: Optional[str] = None*, *force_update=None*, *sleeping_time=None*, *exchanges=None*, *entity_id=None*, *entity_ids=None*, *code=None*, *codes=None*, *real_time=None*, *fix_duplicate_way=None*, *start_timestamp=None*, *end_timestamp=None*, *one_day_trading_minutes=None*, ***kwargs*)

record data by the arguments

Parameters::

- **entity_id** –
- **provider_index** –
- **provider** –
- **force_update** –
- **sleeping_time** –
- **exchanges** –
- **entity_ids** –
- **code** –
- **codes** –
- **real_time** –
- **fix_duplicate_way** –
- **start_timestamp** –
- **end_timestamp** –
- **one_day_trading_minutes** –
- **kwargs** –

Returns::

classmethod register_provider(*provider*)

register the provider to the schema defined by cls

Parameters:: **provider** –

classmethod register_recorder_cls(*provider*, *recorder_cls*)

register the recorder for the schema

Parameters::

- **provider** –
- **recorder_cls** –

timestamp = *Column(None, DateTime(), table=None)*

the meaning could be different for different case,most time it means 'happen time'

NormalMixin

class zvt.contract.schema.NormalMixin

Bases: `Mixin`

__init__()

created_timestamp = *Column(None, DateTime(), table=None, default=ScalarElementColumnDefault(Timestamp('2024-08-28 22:08:09.719367')))*

the record created time in db

updated_timestamp = *Column(None, DateTime(), table=None)*

the record updated time in db, some recorder would check it for whether need to refresh

entity_id = *Column(None, String(), table=None)*

entity id

classmethod get_providers() → List[str]

providers of the schema defined by cls

Returns:: providers

classmethod get_storages(*provider: Optional[str] = None*)

get the storages info

Parameters:: **provider** – provider

Returns:: storages

id = *Column(None, String(), table=None, primary_key=True, nullable=False)*

id

classmethod query_data(*provider_index: int = 0*, *ids: Optional[List[str]] = None*, *entity_ids: Optional[List[str]] = None*, *entity_id: Optional[str]*

= None, *codes: Optional[List[str]] = None*, *code: Optional[str] = None*, *level: Optional[Union[IntervalLevel, str]] = None*, *provider: Optional[str] = None*, *columns: Optional[List] = None*, *col_label: Optional[dict] = None*, *return_type: str = 'df'*, *start_timestamp: Optional[Union[Timestamp, str]] = None*, *end_timestamp: Optional[Union[Timestamp, str]] = None*, *filters: Optional[List] = None*, *session: Optional[Session] = None*, *order=None*, *limit: Optional[int] = None*, *distinct=None*, *index: Optional[Union[str, list]] = None*, *drop_index_col=False*, *time_field: str = 'timestamp'*)

query data by the arguments

Parameters::

- **provider_index** –
- **data_schema** –
- **ids** –
- **entity_ids** –
- **entity_id** –
- **codes** –
- **code** –
- **level** –
- **provider** –
- **columns** –
- **col_label** – dict with key(column), value(label
- **return_type** – df, domain or dict. default is df
- **start_timestamp** –
- **end_timestamp** –
- **filters** –
- **session** –
- **order** –
- **limit** –
- **index** – index field name, str for single index, str list for multiple index
- **drop_index_col** – whether drop the col if it's in index, default False
- **time_field** –

Returns:: results basing on return_type.

classmethod record_data(*provider_index: int = 0*, *provider: Optional[str] = None*, *force_update=None*, *sleeping_time=None*, *exchanges=None*,

entity_id=None, *entity_ids=None*, *code=None*, *codes=None*, *real_time=None*, *fix_duplicate_way=None*, *start_timestamp=None*, *end_timestamp=None*, *one_day_trading_minutes=None*, ***kwargs*)

record data by the arguments

Parameters::

- **entity_id** –
- **provider_index** –
- **provider** –
- **force_update** –
- **sleeping_time** –
- **exchanges** –
- **entity_ids** –
- **code** –
- **codes** –
- **real_time** –
- **fix_duplicate_way** –
- **start_timestamp** –
- **end_timestamp** –
- **one_day_trading_minutes** –
- **kwargs** –

Returns::

classmethod register_provider(*provider*)

register the provider to the schema defined by cls

Parameters:: **provider** –

classmethod register_recorder_cls(*provider*, *recorder_cls*)

register the recorder for the schema

Parameters::

- **provider** –
- **recorder_cls** –

timestamp = *Column(None, DateTime(), table=None)*

the meaning could be different for different case,most time it means 'happen time'

Portfolio

class zvt.contract.schema.Portfolio

Bases: `TradableEntity`

composition of tradable entities

__init__()

classmethod get_stocks(*code=None*, *codes=None*, *ids=None*, *timestamp=Timestamp('2024-08-28 22:08:09.719543')*, *provider=None*)

the publishing policy of portfolio positions is different for different types, overwrite this function for get the holding stocks in specific date

Parameters::

- **code** – portfolio(etf/block/index…) code
- **codes** – portfolio(etf/block/index…) codes
- **ids** – portfolio(etf/block/index…) ids
- **timestamp** – the date of the holding stocks
- **provider** – the data provider

Returns::

code = *Column(None, String(length=64), table=None)*

编码

classmethod could_short()

whether could be shorted

Returns::

end_date = *Column(None, DateTime(), table=None)*

退市日

entity_id = *Column(None, String(), table=None)*

entity id

entity_type = *Column(None, String(length=64), table=None)*

标的类型

exchange = *Column(None, String(length=32), table=None)*

所属交易所

classmethod get_interval_timestamps(*start_date*, *end_date*, *level: IntervalLevel*)

generate the timestamps for the level

Parameters::
- **start_date** –
- **end_date** –
- **level** –

classmethod get_providers() → List[str]

providers of the schema defined by cls

Returns:: providers

classmethod get_storages(*provider: Optional[str] = None*)

get the storages info

Parameters:: **provider** – provider

Returns:: storages

classmethod get_trading_dates(*start_date=None*, *end_date=None*)

overwrite it to get the trading dates of the entity

Parameters::
- **start_date** –
- **end_date** –

Returns:: list of dates

classmethod get_trading_intervals()

overwrite it to get the trading intervals of the entity

Returns:: list of time intervals, in format [(start,end)]

classmethod get_trading_t()

0 means t+0 1 means t+1

Returns::

id = *Column(None, String(), table=None, primary_key=True, nullable=False)*

id

classmethod is_finished_kdata_timestamp(*timestamp: Timestamp*, *level: IntervalLevel*)

Parameters::
- **timestamp** (*pd.Timestamp*) – the timestamp could be recorded in kdata of the level
- **level** (*zvt.domain.common.IntervalLevel*) –

Returns::

Return type:: bool

list_date = *Column(None, DateTime(), table=None)*

上市日

name = *Column(None, String(length=128), table=None)*

名字

classmethod query_data(*provider_index: int = 0*, *ids: Optional[List[str]] = None*, *entity_ids: Optional[List[str]] = None*, *entity_id: Optional[str] = None*, *codes: Optional[List[str]] = None*, *code: Optional[str] = None*, *level: Optional[Union[IntervalLevel, str]] = None*, *provider: Optional[str] = None*, *columns: Optional[List] = None*, *col_label: Optional[dict] = None*, *return_type: str = 'df'*, *start_timestamp: Optional[Union[Timestamp, str]] = None*, *end_timestamp: Optional[Union[Timestamp, str]] = None*, *filters: Optional[List] = None*, *session: Optional[Session] = None*, *order=None*, *limit: Optional[int] = None*, *distinct=None*, *index: Optional[Union[str, list]] = None*, *drop_index_col=False*, *time_field: str = 'timestamp'*)

query data by the arguments

Parameters::

- **provider_index** –
- **data_schema** –
- **ids** –
- **entity_ids** –
- **entity_id** –
- **codes** –
- **code** –
- **level** –
- **provider** –
- **columns** –
- **col_label** – dict with key(column), value(label
- **return_type** – df, domain or dict. default is df
- **start_timestamp** –
- **end_timestamp** –
- **filters** –
- **session** –
- **order** –
- **limit** –
- **index** – index field name, str for single index, str list for multiple index
- **drop_index_col** – whether drop the col if it's in index, default False
- **time_field** –

Returns:: results basing on return_type.

classmethod record_data(*provider_index: int = 0*, *provider: Optional[str] = None*, *force_update=None*, *sleeping_time=None*, *exchanges=None*, *entity_id=None*, *entity_ids=None*, *code=None*, *codes=None*, *real_time=None*, *fix_duplicate_way=None*, *start_timestamp=None*, *end_timestamp=None*, *one_day_trading_minutes=None*, ***kwargs*)

record data by the arguments

Parameters::

- **entity_id** –
- **provider_index** –
- **provider** –
- **force_update** –
- **sleeping_time** –
- **exchanges** –
- **entity_ids** –
- **code** –
- **codes** –
- **real_time** –
- **fix_duplicate_way** –
- **start_timestamp** –
- **end_timestamp** –
- **one_day_trading_minutes** –
- **kwargs** –

Returns::

classmethod register_provider(*provider*)

register the provider to the schema defined by cls

Parameters:: **provider** –

classmethod register_recorder_cls(*provider*, *recorder_cls*)

register the recorder for the schema

Parameters::

- **provider** –
- **recorder_cls** –

timestamp = *Column(None, DateTime(), table=None)*

the meaning could be different for different case,most time it means 'happen time'

PortfolioStock

class zvt.contract.schema.PortfolioStock

Bases: `Mixin`

__init__()

entity_type = *Column(None, String(length=64), table=None)*

portfolio标的类型

exchange = *Column(None, String(length=32), table=None)*

portfolio所属交易所

code = *Column(None, String(length=64), table=None)*

portfolio编码

name = *Column(None, String(length=128), table=None)*

portfolio名字

entity_id = *Column(None, String(), table=None)*

entity id

classmethod get_providers() → List[str]

providers of the schema defined by cls

Returns:: providers

classmethod get_storages(*provider: Optional[str] = None*)

get the storages info

Parameters:: **provider** – provider

Returns:: storages

id = *Column(None, String(), table=None, primary_key=True, nullable=False)*

id

classmethod query_data(*provider_index: int = 0*, *ids: Optional[List[str]] = None*, *entity_ids: Optional[List[str]] = None*, *entity_id: Optional[str] = None*, *codes: Optional[List[str]] = None*, *code: Optional[str] = None*, *level: Optional[Union[IntervalLevel, str]] = None*, *provider: Optional[str] = None*, *columns: Optional[List] = None*, *col_label: Optional[dict] = None*, *return_type: str = 'df'*, *start_timestamp: Optional[Union[Timestamp, str]] = None*, *end_timestamp: Optional[Union[Timestamp, str]] = None*, *filters: Optional[List] = None*, *session: Optional[Session] = None*, *order=None*, *limit: Optional[int] = None*, *distinct=None*, *index: Optional[Union[str, list]] = None*, *drop_index_col=False*, *time_field: str = 'timestamp'*)

query data by the arguments

Parameters::

- **provider_index** –
- **data_schema** –
- **ids** –
- **entity_ids** –
- **entity_id** –
- **codes** –
- **code** –
- **level** –
- **provider** –
- **columns** –
- **col_label** – dict with key(column), value(label
- **return_type** – df, domain or dict. default is df
- **start_timestamp** –
- **end_timestamp** –
- **filters** –
- **session** –
- **order** –
- **limit** –
- **index** – index field name, str for single index, str list for multiple index
- **drop_index_col** – whether drop the col if it's in index, default False
- **time_field** –

Returns:: results basing on return_type.

classmethod record_data(*provider_index: int = 0*, *provider: Optional[str] = None*, *force_update=None*, *sleeping_time=None*, *exchanges=None*, *entity_id=None*, *entity_ids=None*, *code=None*, *codes=None*, *real_time=None*, *fix_duplicate_way=None*, *start_timestamp=None*, *end_timestamp=None*, *one_day_trading_minutes=None*, ***kwargs*)

record data by the arguments

Parameters::

- **entity_id** –
- **provider_index** –
- **provider** –
- **force_update** –
- **sleeping_time** –
- **exchanges** –
- **entity_ids** –
- **code** –
- **codes** –
- **real_time** –
- **fix_duplicate_way** –
- **start_timestamp** –
- **end_timestamp** –
- **one_day_trading_minutes** –
- **kwargs** –

Returns::

classmethod register_provider(*provider*)

register the provider to the schema defined by cls

Parameters:: **provider** –

classmethod register_recorder_cls(*provider*, *recorder_cls*)

register the recorder for the schema

Parameters::

- **provider** –
- **recorder_cls** –

timestamp = *Column(None, DateTime(), table=None)*

the meaning could be different for different case,most time it means ‘happen time’

PortfolioStockHistory

class zvt.contract.schema.PortfolioStockHistory

Bases: `PortfolioStock`

__init__()

report_period = *Column(None, String(length=32), table=None)*

报告期,season1,half_year,season3,year

report_date = *Column(None, DateTime(), table=None)*

3-31,6-30,9-30,12-31

proportion = *Column(None, Float(), table=None)*

占净值比例

shares = *Column(None, Float(), table=None)*

持有股票的数量

market_cap = *Column(None, Float(), table=None)*

持有股票的市值

code = *Column(None, String(length=64), table=None)*

portfolio编码

entity_id = *Column(None, String(), table=None)*

entity id

entity_type = *Column(None, String(length=64), table=None)*

portfolio标的类型

exchange = *Column(None, String(length=32), table=None)*

portfolio所属交易所

classmethod get_providers() → List[str]

providers of the schema defined by cls

Returns:: providers

classmethod get_storages(*provider: Optional[str] = None*)

get the storages info

Parameters:: **provider** – provider
Returns:: storages

id = *Column(None, String(), table=None, primary_key=True, nullable=False)*

id

name = *Column(None, String(length=128), table=None)*

portfolio名字

classmethod query_data(*provider_index: int = 0*, *ids: Optional[List[str]] = None*, *entity_ids: Optional[List[str]] = None*, *entity_id: Optional[str] = None*, *codes: Optional[List[str]] = None*, *code: Optional[str] = None*, *level: Optional[Union[IntervalLevel, str]] = None*, *provider: Optional[str] = None*, *columns: Optional[List] = None*, *col_label: Optional[dict] = None*, *return_type: str = 'df'*, *start_timestamp: Optional[Union[Timestamp, str]] = None*, *end_timestamp: Optional[Union[Timestamp, str]] = None*, *filters: Optional[List] = None*, *session: Optional[Session] = None*, *order=None*, *limit: Optional[int] = None*, *distinct=None*, *index: Optional[Union[str, list]] = None*, *drop_index_col=False*, *time_field: str = 'timestamp'*)

query data by the arguments

Parameters::

- **provider_index** –
- **data_schema** –
- **ids** –
- **entity_ids** –
- **entity_id** –
- **codes** –
- **code** –
- **level** –
- **provider** –
- **columns** –
- **col_label** – dict with key(column), value(label
- **return_type** – df, domain or dict. default is df
- **start_timestamp** –
- **end_timestamp** –
- **filters** –
- **session** –
- **order** –
- **limit** –
- **index** – index field name, str for single index, str list for multiple index
- **drop_index_col** – whether drop the col if it's in index, default False
- **time_field** –

Returns:: results basing on return_type.

classmethod record_data(*provider_index: int = 0*, *provider: Optional[str] = None*, *force_update=None*, *sleeping_time=None*, *exchanges=None*, *entity_id=None*, *entity_ids=None*, *code=None*, *codes=None*, *real_time=None*, *fix_duplicate_way=None*, *start_timestamp=None*, *end_timestamp=None*, *one_day_trading_minutes=None*, ***kwargs*)

record data by the arguments

Parameters::

- **entity_id** –
- **provider_index** –
- **provider** –
- **force_update** –
- **sleeping_time** –
- **exchanges** –
- **entity_ids** –
- **code** –
- **codes** –
- **real_time** –
- **fix_duplicate_way** –
- **start_timestamp** –
- **end_timestamp** –
- **one_day_trading_minutes** –
- **kwargs** –

Returns::

classmethod register_provider(*provider*)

register the provider to the schema defined by cls

Parameters:: **provider** –

classmethod register_recorder_cls(*provider*, *recorder_cls*)

register the recorder for the schema

Parameters::

- **provider** –
- **recorder_cls** –

timestamp = *Column(None, DateTime(), table=None)*

the meaning could be different for different case,most time it means 'happen time'

TradableEntity

class zvt.contract.schema.TradableEntity

Bases: `Entity`

tradable entity

__init__()

classmethod get_trading_dates(*start_date=None*, *end_date=None*)

overwrite it to get the trading dates of the entity

Parameters::
- **start_date** –
- **end_date** –

Returns:: list of dates

classmethod get_trading_intervals()

overwrite it to get the trading intervals of the entity

Returns:: list of time intervals, in format [(start,end)]

classmethod get_interval_timestamps(*start_date*, *end_date*, *level: IntervalLevel*)

generate the timestamps for the level

Parameters::
- **start_date** –
- **end_date** –
- **level** –

classmethod is_finished_kdata_timestamp(*timestamp: Timestamp*, *level: IntervalLevel*)

Parameters::
- **timestamp** (*pd.Timestamp*) – the timestamp could be recorded in kdata of the level
- **level** (*zvt.domain.common.IntervalLevel*) –

Returns::

Return type:: bool

classmethod could_short()

whether could be shorted

Returns::

classmethod get_trading_t()

0 means t+0 1 means t+1

Returns::

code = *Column(None, String(length=64), table=None)*

编码

end_date = *Column(None, DateTime(), table=None)*

退市日

entity_id = *Column(None, String(), table=None)*

entity id

entity_type = *Column(None, String(length=64), table=None)*

标的类型

exchange = *Column(None, String(length=32), table=None)*

所属交易所

classmethod get_providers() → List[str]

providers of the schema defined by cls

Returns:: providers

classmethod get_storages(*provider: Optional[str] = None*)

get the storages info

Parameters:: **provider** – provider

Returns:: storages

id = *Column(None, String(), table=None, primary_key=True, nullable=False)*

id

list_date = *Column(None, DateTime(), table=None)*

上市日

name = *Column(None, String(length=128), table=None)*

名字

classmethod query_data(*provider_index: int = 0*, *ids: Optional[List[str]] = None*, *entity_ids: Optional[List[str]] = None*, *entity_id: Optional[str] = None*, *codes: Optional[List[str]] = None*, *code: Optional[str] = None*, *level: Optional[Union[IntervalLevel, str]] = None*, *provider: Optional[str] = None*, *columns: Optional[List] = None*, *col_label: Optional[dict] = None*, *return_type: str = 'df'*, *start_timestamp: Optional[Union[Timestamp, str]] = None*, *end_timestamp: Optional[Union[Timestamp, str]] = None*, *filters: Optional[List] = None*, *session: Optional[Session] = None*, *order=None*, *limit: Optional[int] = None*, *distinct=None*, *index: Optional[Union[str, list]] = None*, *drop_index_col=False*, *time_field: str = 'timestamp'*)

query data by the arguments

Parameters::

- **provider_index** –
- **data_schema** –
- **ids** –
- **entity_ids** –
- **entity_id** –
- **codes** –
- **code** –
- **level** –
- **provider** –
- **columns** –
- **col_label** – dict with key(column), value(label
- **return_type** – df, domain or dict. default is df
- **start_timestamp** –
- **end_timestamp** –
- **filters** –
- **session** –
- **order** –
- **limit** –
- **index** – index field name, str for single index, str list for multiple index
- **drop_index_col** – whether drop the col if it's in index, default False
- **time_field** –

Returns:: results basing on return_type.

classmethod record_data(*provider_index: int = 0*, *provider: Optional[str] = None*, *force_update=None*, *sleeping_time=None*, *exchanges=None*, *entity_id=None*, *entity_ids=None*, *code=None*, *codes=None*, *real_time=None*, *fix_duplicate_way=None*, *start_timestamp=None*, *end_timestamp=None*, *one_day_trading_minutes=None*, ***kwargs*)

record data by the arguments

Parameters::

- **entity_id** –
- **provider_index** –
- **provider** –
- **force_update** –
- **sleeping_time** –
- **exchanges** –
- **entity_ids** –
- **code** –
- **codes** –
- **real_time** –
- **fix_duplicate_way** –
- **start_timestamp** –
- **end_timestamp** –
- **one_day_trading_minutes** –
- **kwargs** –

Returns::

classmethod register_provider(*provider*)

register the provider to the schema defined by cls

Parameters:: **provider** –

classmethod register_recorder_cls(*provider*, *recorder_cls*)

register the recorder for the schema

Parameters::

- **provider** –
- **recorder_cls** –

timestamp = *Column(None, DateTime(), table=None)*

the meaning could be different for different case,most time it means 'happen time'

TradableMeetActor

class zvt.contract.schema.TradableMeetActor

Bases: `Mixin`

__init__()

code = *Column(None, String(length=64), table=None)*

tradable code

name = *Column(None, String(length=128), table=None)*

tradable name

entity_id = *Column(None, String(), table=None)*

entity id

classmethod get_providers() → List[str]

providers of the schema defined by cls

Returns:: providers

classmethod get_storages(*provider: Optional[str] = None*)

get the storages info

Parameters:: **provider** – provider
Returns:: storages

id = *Column(None, String(), table=None, primary_key=True, nullable=False)*

id

classmethod query_data(*provider_index: int = 0*, *ids: Optional[List[str]] = None*, *entity_ids: Optional[List[str]] = None*, *entity_id: Optional[str] = None*, *codes: Optional[List[str]] = None*, *code: Optional[str] = None*, *level: Optional[Union[IntervalLevel, str]] = None*, *provider: Optional[str] = None*, *columns: Optional[List] = None*, *col_label:*

Optional[dict] = None, *return_type: str = 'df'*, *start_timestamp: Optional[Union[Timestamp, str]] = None*, *end_timestamp: Optional[Union[Timestamp, str]] = None*, *filters: Optional[List] = None*, *session: Optional[Session] = None*, *order=None*, *limit: Optional[int] = None*, *distinct=None*, *index: Optional[Union[str, list]] = None*, *drop_index_col=False*, *time_field: str = 'timestamp'*)

query data by the arguments

Parameters::

- **provider_index** –
- **data_schema** –
- **ids** –
- **entity_ids** –
- **entity_id** –
- **codes** –
- **code** –
- **level** –
- **provider** –
- **columns** –
- **col_label** – dict with key(column), value(label
- **return_type** – df, domain or dict. default is df
- **start_timestamp** –
- **end_timestamp** –
- **filters** –
- **session** –
- **order** –
- **limit** –
- **index** – index field name, str for single index, str list for multiple index
- **drop_index_col** – whether drop the col if it's in index, default False
- **time_field** –

Returns:: results basing on return_type.

classmethod record_data(*provider_index: int = 0*, *provider: Optional[str] = None*, *force_update=None*, *sleeping_time=None*, *exchanges=None*, *entity_id=None*, *entity_ids=None*, *code=None*, *codes=None*, *real_time=None*, *fix_duplicate_way=None*, *start_timestamp=None*, *end_timestamp=None*, *one_day_trading_minutes=None*, ***kwargs*)

record data by the arguments

Parameters::

- **entity_id** –
- **provider_index** –
- **provider** –
- **force_update** –
- **sleeping_time** –
- **exchanges** –
- **entity_ids** –
- **code** –
- **codes** –
- **real_time** –
- **fix_duplicate_way** –
- **start_timestamp** –
- **end_timestamp** –
- **one_day_trading_minutes** –
- **kwargs** –

Returns::

classmethod register_provider(*provider*)

register the provider to the schema defined by cls

Parameters:: **provider** –

classmethod register_recorder_cls(*provider*, *recorder_cls*)

register the recorder for the schema

Parameters::

- **provider** –
- **recorder_cls** –

timestamp = *Column(None, DateTime(), table=None)*

the meaning could be different for different case,most time it means 'happen time'

zvt.contract.zvt_info

Classes

`FactorState`(**kwargs)	Schema for storing factor state
`RecorderState`(**kwargs)	Schema for storing recorder state
`StateMixin`()	
`TaggerState`(**kwargs)	Schema for storing tagger state

FactorState

class zvt.contract.zvt_info.FactorState(***kwargs*)

Bases: **Base**, **StateMixin**

Schema for storing factor state

__init__(***kwargs*)

A simple constructor that allows initialization from kwargs.

Sets attributes on the constructed instance using the names and values in `kwargs`.

Only keys that are present as attributes of the instance's class are allowed. These could be, for example, any mapped columns or relationships.

entity_id

entity id

classmethod get_providers() → List[str]

providers of the schema defined by cls

Returns:: providers

classmethod get_storages(*provider: Optional[str] = None*)

get the storages info

Parameters:: **provider** – provider
Returns:: storages

id

id

classmethod query_data(*provider_index: int = 0*, *ids: Optional[List[str]] = None*, *entity_ids: Optional[List[str]] = None*, *entity_id: Optional[str]*

= None, *codes: Optional[List[str]] = None*, *code: Optional[str] = None*, *level: Optional[Union[IntervalLevel, str]] = None*, *provider: Optional[str] = None*, *columns: Optional[List] = None*, *col_label: Optional[dict] = None*, *return_type: str = 'df'*, *start_timestamp: Optional[Union[Timestamp, str]] = None*, *end_timestamp: Optional[Union[Timestamp, str]] = None*, *filters: Optional[List] = None*, *session: Optional[Session] = None*, *order=None*, *limit: Optional[int] = None*, *distinct=None*, *index: Optional[Union[str, list]] = None*, *drop_index_col=False*, *time_field: str = 'timestamp'*)

query data by the arguments

Parameters::

- **provider_index** –
- **data_schema** –
- **ids** –
- **entity_ids** –
- **entity_id** –
- **codes** –
- **code** –
- **level** –
- **provider** –
- **columns** –
- **col_label** – dict with key(column), value(label
- **return_type** – df, domain or dict. default is df
- **start_timestamp** –
- **end_timestamp** –
- **filters** –
- **session** –
- **order** –
- **limit** –
- **index** – index field name, str for single index, str list for multiple index
- **drop_index_col** – whether drop the col if it's in index, default False
- **time_field** –

Returns:: results basing on return_type.

classmethod record_data(*provider_index: int = 0*, *provider: Optional[str] = None*, *force_update=None*, *sleeping_time=None*, *exchanges=None*,

entity_id=None, *entity_ids=None*, *code=None*, *codes=None*, *real_time=None*, *fix_duplicate_way=None*, *start_timestamp=None*, *end_timestamp=None*, *one_day_trading_minutes=None*, ***kwargs*)

record data by the arguments

Parameters::

- **entity_id** –
- **provider_index** –
- **provider** –
- **force_update** –
- **sleeping_time** –
- **exchanges** –
- **entity_ids** –
- **code** –
- **codes** –
- **real_time** –
- **fix_duplicate_way** –
- **start_timestamp** –
- **end_timestamp** –
- **one_day_trading_minutes** –
- **kwargs** –

Returns::

classmethod register_provider(*provider*)

register the provider to the schema defined by cls

Parameters:: **provider** –

classmethod register_recorder_cls(*provider*, *recorder_cls*)

register the recorder for the schema

Parameters::

- **provider** –
- **recorder_cls** –

state

json string

state_name

the unique name of the service, e.g. recorder,factor,tag

timestamp

the meaning could be different for different case,most time it means ‘happen time’

RecorderState

class zvt.contract.zvt_info.RecorderState(***kwargs*)

Bases: **Base**, **StateMixin**

Schema for storing recorder state

__init__(***kwargs*)

A simple constructor that allows initialization from kwargs.

Sets attributes on the constructed instance using the names and values in `kwargs`.

Only keys that are present as attributes of the instance's class are allowed. These could be, for example, any mapped columns or relationships.

entity_id

entity id

classmethod get_providers() → List[str]

providers of the schema defined by cls

Returns:: providers

classmethod get_storages(*provider: Optional[str] = None*)

get the storages info

Parameters:: **provider** – provider
Returns:: storages

id

id

classmethod query_data(*provider_index: int = 0*, *ids: Optional[List[str]] = None*, *entity_ids: Optional[List[str]] = None*, *entity_id: Optional[str]*

= None, *codes: Optional[List[str]] = None*, *code: Optional[str] = None*, *level: Optional[Union[IntervalLevel, str]] = None*, *provider: Optional[str] = None*, *columns: Optional[List] = None*, *col_label: Optional[dict] = None*, *return_type: str = 'df'*, *start_timestamp: Optional[Union[Timestamp, str]] = None*, *end_timestamp: Optional[Union[Timestamp, str]] = None*, *filters: Optional[List] = None*, *session: Optional[Session] = None*, *order=None*, *limit: Optional[int] = None*, *distinct=None*, *index: Optional[Union[str, list]] = None*, *drop_index_col=False*, *time_field: str = 'timestamp'*)

query data by the arguments

Parameters::

- **provider_index** –
- **data_schema** –
- **ids** –
- **entity_ids** –
- **entity_id** –
- **codes** –
- **code** –
- **level** –
- **provider** –
- **columns** –
- **col_label** – dict with key(column), value(label
- **return_type** – df, domain or dict. default is df
- **start_timestamp** –
- **end_timestamp** –
- **filters** –
- **session** –
- **order** –
- **limit** –
- **index** – index field name, str for single index, str list for multiple index
- **drop_index_col** – whether drop the col if it's in index, default False
- **time_field** –

Returns:: results basing on return_type.

classmethod record_data(*provider_index: int = 0*, *provider: Optional[str] = None*, *force_update=None*, *sleeping_time=None*, *exchanges=None*,

entity_id=None, *entity_ids=None*, *code=None*, *codes=None*, *real_time=None*, *fix_duplicate_way=None*, *start_timestamp=None*, *end_timestamp=None*, *one_day_trading_minutes=None*, ***kwargs*)

record data by the arguments

Parameters::

- **entity_id** –
- **provider_index** –
- **provider** –
- **force_update** –
- **sleeping_time** –
- **exchanges** –
- **entity_ids** –
- **code** –
- **codes** –
- **real_time** –
- **fix_duplicate_way** –
- **start_timestamp** –
- **end_timestamp** –
- **one_day_trading_minutes** –
- **kwargs** –

Returns::

classmethod register_provider(*provider*)

register the provider to the schema defined by cls

Parameters:: **provider** –

classmethod register_recorder_cls(*provider*, *recorder_cls*)

register the recorder for the schema

Parameters::

- **provider** –
- **recorder_cls** –

state

json string

state_name

the unique name of the service, e.g. recorder,factor,tag

timestamp

the meaning could be different for different case,most time it means ‘happen time’

StateMixin

class zvt.contract.zvt_info.StateMixin

Bases: `Mixin`

__init__()

state_name = *Column(None, String(length=128), table=None)*

the unique name of the service, e.g. recorder,factor,tag

state = *Column(None, Text(), table=None)*

json string

entity_id = *Column(None, String(), table=None)*

entity id

classmethod get_providers() → List[str]

providers of the schema defined by cls

Returns:: providers

classmethod get_storages(*provider: Optional[str] = None*)

get the storages info

Parameters:: **provider** – provider

Returns:: storages

id = *Column(None, String(), table=None, primary_key=True, nullable=False)*

id

classmethod query_data(*provider_index: int = 0*, *ids: Optional[List[str]] = None*, *entity_ids: Optional[List[str]] = None*, *entity_id: Optional[str] = None*, *codes: Optional[List[str]] = None*, *code: Optional[str] = None*, *level: Optional[Union[IntervalLevel, str]] = None*, *provider: Optional[str] = None*, *columns: Optional[List] = None*, *col_label:*

Optional[dict] = None, *return_type: str = 'df'*, *start_timestamp: Optional[Union[Timestamp, str]] = None*, *end_timestamp: Optional[Union[Timestamp, str]] = None*, *filters: Optional[List] = None*, *session: Optional[Session] = None*, *order=None*, *limit: Optional[int] = None*, *distinct=None*, *index: Optional[Union[str, list]] = None*, *drop_index_col=False*, *time_field: str = 'timestamp'*)

query data by the arguments

Parameters::

- **provider_index** –
- **data_schema** –
- **ids** –
- **entity_ids** –
- **entity_id** –
- **codes** –
- **code** –
- **level** –
- **provider** –
- **columns** –
- **col_label** – dict with key(column), value(label
- **return_type** – df, domain or dict. default is df
- **start_timestamp** –
- **end_timestamp** –
- **filters** –
- **session** –
- **order** –
- **limit** –
- **index** – index field name, str for single index, str list for multiple index
- **drop_index_col** – whether drop the col if it's in index, default False
- **time_field** –

Returns:: results basing on return_type.

classmethod record_data(*provider_index: int = 0*, *provider: Optional[str] = None*, *force_update=None*, *sleeping_time=None*, *exchanges=None*, *entity_id=None*, *entity_ids=None*, *code=None*, *codes=None*, *real_time=None*, *fix_duplicate_way=None*, *start_timestamp=None*, *end_timestamp=None*, *one_day_trading_minutes=None*, ***kwargs*)

record data by the arguments

Parameters::

- **entity_id** –
- **provider_index** –
- **provider** –
- **force_update** –
- **sleeping_time** –
- **exchanges** –
- **entity_ids** –
- **code** –
- **codes** –
- **real_time** –
- **fix_duplicate_way** –
- **start_timestamp** –
- **end_timestamp** –
- **one_day_trading_minutes** –
- **kwargs** –

Returns::

classmethod register_provider(*provider*)

register the provider to the schema defined by cls

Parameters:: **provider** –

classmethod register_recorder_cls(*provider*, *recorder_cls*)

register the recorder for the schema

Parameters::

- **provider** –
- **recorder_cls** –

timestamp = *Column(None, DateTime(), table=None)*

the meaning could be different for different case,most time it means 'happen time'

TaggerState

class zvt.contract.zvt_info.TaggerState(***kwargs*)

Bases: **Base**, **StateMixin**

Schema for storing tagger state

__init__(***kwargs*)

A simple constructor that allows initialization from kwargs.

Sets attributes on the constructed instance using the names and values in `kwargs`.

Only keys that are present as attributes of the instance's class are allowed. These could be, for example, any mapped columns or relationships.

entity_id

entity id

classmethod get_providers() → List[str]

providers of the schema defined by cls

Returns:: providers

classmethod get_storages(*provider: Optional[str] = None*)

get the storages info

Parameters:: **provider** – provider
Returns:: storages

id

id

classmethod query_data(*provider_index: int = 0*, *ids: Optional[List[str]] = None*, *entity_ids: Optional[List[str]] = None*, *entity_id: Optional[str]*

= None, *codes: Optional[List[str]] = None*, *code: Optional[str] = None*, *level: Optional[Union[IntervalLevel, str]] = None*, *provider: Optional[str] = None*, *columns: Optional[List] = None*, *col_label: Optional[dict] = None*, *return_type: str = 'df'*, *start_timestamp: Optional[Union[Timestamp, str]] = None*, *end_timestamp: Optional[Union[Timestamp, str]] = None*, *filters: Optional[List] = None*, *session: Optional[Session] = None*, *order=None*, *limit: Optional[int] = None*, *distinct=None*, *index: Optional[Union[str, list]] = None*, *drop_index_col=False*, *time_field: str = 'timestamp'*)

query data by the arguments

Parameters::

- **provider_index** –
- **data_schema** –
- **ids** –
- **entity_ids** –
- **entity_id** –
- **codes** –
- **code** –
- **level** –
- **provider** –
- **columns** –
- **col_label** – dict with key(column), value(label
- **return_type** – df, domain or dict. default is df
- **start_timestamp** –
- **end_timestamp** –
- **filters** –
- **session** –
- **order** –
- **limit** –
- **index** – index field name, str for single index, str list for multiple index
- **drop_index_col** – whether drop the col if it's in index, default False
- **time_field** –

Returns:: results basing on return_type.

classmethod record_data(*provider_index: int = 0*, *provider: Optional[str] = None*, *force_update=None*, *sleeping_time=None*, *exchanges=None*,

entity_id=None, *entity_ids=None*, *code=None*, *codes=None*, *real_time=None*, *fix_duplicate_way=None*, *start_timestamp=None*, *end_timestamp=None*, *one_day_trading_minutes=None*, ***kwargs*)

record data by the arguments

Parameters::

- **entity_id** –
- **provider_index** –
- **provider** –
- **force_update** –
- **sleeping_time** –
- **exchanges** –
- **entity_ids** –
- **code** –
- **codes** –
- **real_time** –
- **fix_duplicate_way** –
- **start_timestamp** –
- **end_timestamp** –
- **one_day_trading_minutes** –
- **kwargs** –

Returns::

classmethod register_provider(*provider*)

register the provider to the schema defined by cls

Parameters:: **provider** –

classmethod register_recorder_cls(*provider*, *recorder_cls*)

register the recorder for the schema

Parameters::

- **provider** –
- **recorder_cls** –

state

json string

state_name

the unique name of the service, e.g. recorder,factor,tag

timestamp

the meaning could be different for different case,most time it means ‘happen time’

zvt.domain

Functions

get_future_name(code)	

Classes

BlockCategory(value)	An enumeration.
CompanyType(value)	An enumeration.
IndexCategory(value)	An enumeration.
ReportPeriod(value)	An enumeration.

Modules

zvt.domain.actor

zvt.domain.emotion

zvt.domain.fundamental

zvt.domain.macro

zvt.domain.meta

zvt.domain.misc

zvt.domain.quotes

zvt.domain.get_future_name

zvt.domain.get_future_name(*code*)

BlockCategory

class zvt.domain.BlockCategory(*value*)

Bases: **`Enum`**

An enumeration.

__init__()

industry = *'industry'*

行业版块

concept = *'concept'*

概念版块

area = *'area'*

区域版块

CompanyType

class zvt.domain.CompanyType(*value*)

Bases: `Enum`

An enumeration.

__init__()

IndexCategory

class zvt.domain.IndexCategory(*value*)

Bases: **`Enum`**

An enumeration.

__init__()

scope = *'scope'*

规模指数 常见的上证指数，深证指数等

industry = *'industry'*

行业指数

style = *'style'*

风格指数

fund = *'fund'*

基金指数

ReportPeriod

class zvt.domain.ReportPeriod(*value*)

Bases: **`Enum`**

An enumeration.

__init__()

zvt.domain.actor

Modules

zvt.domain.actor.actor_meta

---

zvt.domain.actor.stock_actor

---

zvt.domain.actor.actor_meta

Classes

`ActorMeta`(**kwargs)

ActorMeta

class zvt.domain.actor.actor_meta.ActorMeta(***kwargs*)

Bases: **Base**, **ActorEntity**

__init__(***kwargs*)

A simple constructor that allows initialization from kwargs.

Sets attributes on the constructed instance using the names and values in `kwargs`.

Only keys that are present as attributes of the instance's class are allowed. These could be, for example, any mapped columns or relationships.

code

编码

end_date

退市日

entity_id

entity id

entity_type

标的类型

exchange

所属交易所

classmethod get_providers() → List[str]

providers of the schema defined by cls

Returns:: providers

classmethod get_storages(*provider: Optional[str] = None*)

get the storages info

Parameters:: **provider** – provider

Returns:: storages

id

id

list_date

上市日

name

名字

classmethod query_data(*provider_index: int = 0*, *ids: Optional[List[str]] = None*, *entity_ids: Optional[List[str]] = None*, *entity_id: Optional[str] = None*, *codes: Optional[List[str]] = None*, *code: Optional[str] = None*, *level: Optional[Union[IntervalLevel, str]] = None*, *provider: Optional[str] = None*, *columns: Optional[List] = None*, *col_label: Optional[dict] = None*, *return_type: str = 'df'*, *start_timestamp: Optional[Union[Timestamp, str]] = None*, *end_timestamp: Optional[Union[Timestamp, str]] = None*, *filters: Optional[List] = None*, *session: Optional[Session] = None*, *order=None*, *limit: Optional[int] = None*, *distinct=None*, *index: Optional[Union[str, list]] = None*, *drop_index_col=False*, *time_field: str = 'timestamp'*)

query data by the arguments

Parameters::

- **provider_index** –
- **data_schema** –
- **ids** –
- **entity_ids** –
- **entity_id** –
- **codes** –
- **code** –
- **level** –
- **provider** –
- **columns** –
- **col_label** – dict with key(column), value(label
- **return_type** – df, domain or dict. default is df
- **start_timestamp** –
- **end_timestamp** –
- **filters** –
- **session** –
- **order** –
- **limit** –
- **index** – index field name, str for single index, str list for multiple index
- **drop_index_col** – whether drop the col if it's in index, default False
- **time_field** –

Returns:: results basing on return_type.

classmethod record_data(*provider_index: int = 0*, *provider: Optional[str] = None*, *force_update=None*, *sleeping_time=None*, *exchanges=None*, *entity_id=None*, *entity_ids=None*, *code=None*, *codes=None*, *real_time=None*, *fix_duplicate_way=None*, *start_timestamp=None*, *end_timestamp=None*, *one_day_trading_minutes=None*, ***kwargs*)

record data by the arguments

Parameters::

- **entity_id** –
- **provider_index** –
- **provider** –
- **force_update** –
- **sleeping_time** –
- **exchanges** –
- **entity_ids** –
- **code** –
- **codes** –
- **real_time** –
- **fix_duplicate_way** –
- **start_timestamp** –
- **end_timestamp** –
- **one_day_trading_minutes** –
- **kwargs** –

Returns::

classmethod register_provider(*provider*)

register the provider to the schema defined by cls

Parameters:: **provider** –

classmethod register_recorder_cls(*provider*, *recorder_cls*)

register the recorder for the schema

Parameters::

- **provider** –
- **recorder_cls** –

timestamp

the meaning could be different for different case,most time it means 'happen time'

zvt.domain.actor.stock_actor

Classes

`StockActorSummary`(**kwargs)

`StockInstitutionalInvestorHolder`(**kwargs)

`StockTopTenFreeHolder`(**kwargs)

`StockTopTenHolder`(**kwargs)

StockActorSummary

class zvt.domain.actor.stock_actor.StockActorSummary(**kwargs)

Bases: **Base**, **TradableMeetActor**

__init__(**kwargs)

A simple constructor that allows initialization from kwargs.

Sets attributes on the constructed instance using the names and values in `kwargs`.

Only keys that are present as attributes of the instance's class are allowed. These could be, for example, any mapped columns or relationships.

code

tradable code

name

tradable name

change_ratio

变动比例

is_complete

是否完成

actor_type

持股市值

holding_numbers

持股数

holding_ratio

持股比例

holding_values

持股市值

actor_code

actor_id

actor_name

entity_id

entity id

classmethod get_providers() → List[str]

providers of the schema defined by cls

Returns:: providers

classmethod get_storages(*provider: Optional[str] = None*)

get the storages info

Parameters:: **provider** – provider

Returns:: storages

id

id

classmethod query_data(*provider_index: int = 0*, *ids: Optional[List[str]] = None*, *entity_ids: Optional[List[str]] = None*, *entity_id: Optional[str] = None*, *codes: Optional[List[str]] = None*, *code: Optional[str] = None*, *level: Optional[Union[IntervalLevel, str]] = None*, *provider: Optional[str] = None*, *columns: Optional[List] = None*, *col_label: Optional[dict] = None*, *return_type: str = 'df'*, *start_timestamp: Optional[Union[Timestamp, str]] = None*, *end_timestamp: Optional[Union[Timestamp, str]] = None*, *filters: Optional[List] = None*, *session: Optional[Session] = None*, *order=None*, *limit: Optional[int] = None*, *distinct=None*, *index: Optional[Union[str, list]] = None*, *drop_index_col=False*, *time_field: str = 'timestamp'*)

query data by the arguments

Parameters::

- **provider_index** –
- **data_schema** –
- **ids** –
- **entity_ids** –
- **entity_id** –
- **codes** –
- **code** –
- **level** –
- **provider** –
- **columns** –
- **col_label** – dict with key(column), value(label
- **return_type** – df, domain or dict. default is df
- **start_timestamp** –
- **end_timestamp** –
- **filters** –
- **session** –
- **order** –
- **limit** –
- **index** – index field name, str for single index, str list for multiple index
- **drop_index_col** – whether drop the col if it's in index, default False
- **time_field** –

Returns:: results basing on return_type.

classmethod record_data(*provider_index: int = 0*, *provider: Optional[str] = None*, *force_update=None*, *sleeping_time=None*, *exchanges=None*, *entity_id=None*, *entity_ids=None*, *code=None*, *codes=None*, *real_time=None*, *fix_duplicate_way=None*, *start_timestamp=None*, *end_timestamp=None*, *one_day_trading_minutes=None*, ***kwargs*)

record data by the arguments

Parameters::

- **entity_id** –
- **provider_index** –
- **provider** –
- **force_update** –
- **sleeping_time** –
- **exchanges** –
- **entity_ids** –
- **code** –
- **codes** –
- **real_time** –
- **fix_duplicate_way** –
- **start_timestamp** –
- **end_timestamp** –
- **one_day_trading_minutes** –
- **kwargs** –

Returns::

classmethod register_provider(*provider*)

register the provider to the schema defined by cls

Parameters:: **provider** –

classmethod register_recorder_cls(*provider*, *recorder_cls*)

register the recorder for the schema

Parameters::

- **provider** –
- **recorder_cls** –

timestamp

the meaning could be different for different case,most time it means 'happen time'

StockInstitutionalInvestorHolder

class
zvt.domain.actor.stock_actor.StockInstitutionalInvestorHolder(***kwargs*)

Bases: **Base**, **TradableMeetActor**

__init__(***kwargs*)

A simple constructor that allows initialization from kwargs.

Sets attributes on the constructed instance using the names and values in `kwargs`.

Only keys that are present as attributes of the instance's class are allowed. These could be, for example, any mapped columns or relationships.

holding_numbers

持股数

holding_ratio

持股比例

holding_values

持股市值

actor_code

actor_id

actor_name

actor_type

code

tradable code

entity_id

entity id

classmethod get_providers() → List[str]

providers of the schema defined by cls

Returns:: providers

classmethod get_storages(*provider: Optional[str] = None*)

get the storages info

Parameters:: **provider** – provider

Returns:: storages

id

id

name

tradable name

classmethod query_data(*provider_index: int = 0*, *ids: Optional[List[str]] = None*, *entity_ids: Optional[List[str]] = None*, *entity_id: Optional[str] = None*, *codes: Optional[List[str]] = None*, *code: Optional[str] = None*, *level: Optional[Union[IntervalLevel, str]] = None*, *provider: Optional[str] = None*, *columns: Optional[List] = None*, *col_label: Optional[dict] = None*, *return_type: str = 'df'*, *start_timestamp: Optional[Union[Timestamp, str]] = None*, *end_timestamp: Optional[Union[Timestamp, str]] = None*, *filters: Optional[List] = None*, *session: Optional[Session] = None*, *order=None*, *limit: Optional[int] = None*, *distinct=None*, *index: Optional[Union[str, list]] = None*, *drop_index_col=False*, *time_field: str = 'timestamp'*)

query data by the arguments

Parameters::

- **provider_index** –
- **data_schema** –
- **ids** –
- **entity_ids** –
- **entity_id** –
- **codes** –
- **code** –
- **level** –
- **provider** –
- **columns** –
- **col_label** – dict with key(column), value(label
- **return_type** – df, domain or dict. default is df
- **start_timestamp** –
- **end_timestamp** –
- **filters** –
- **session** –
- **order** –
- **limit** –
- **index** – index field name, str for single index, str list for multiple index
- **drop_index_col** – whether drop the col if it's in index, default False
- **time_field** –

Returns:: results basing on return_type.

classmethod record_data(*provider_index: int = 0*, *provider: Optional[str] = None*, *force_update=None*, *sleeping_time=None*, *exchanges=None*, *entity_id=None*, *entity_ids=None*, *code=None*, *codes=None*, *real_time=None*, *fix_duplicate_way=None*, *start_timestamp=None*, *end_timestamp=None*, *one_day_trading_minutes=None*, ***kwargs*)

record data by the arguments

Parameters::

- **entity_id** –
- **provider_index** –
- **provider** –
- **force_update** –
- **sleeping_time** –
- **exchanges** –
- **entity_ids** –
- **code** –
- **codes** –
- **real_time** –
- **fix_duplicate_way** –
- **start_timestamp** –
- **end_timestamp** –
- **one_day_trading_minutes** –
- **kwargs** –

Returns::

classmethod register_provider(*provider*)

register the provider to the schema defined by cls

Parameters:: **provider** –

classmethod register_recorder_cls(*provider*, *recorder_cls*)

register the recorder for the schema

Parameters::

- **provider** –
- **recorder_cls** –

timestamp

the meaning could be different for different case,most time it means 'happen time'

StockTopTenFreeHolder

class zvt.domain.actor.stock_actor.StockTopTenFreeHolder(**kwargs)

Bases: **Base**, **TradableMeetActor**

__init__(**kwargs)

A simple constructor that allows initialization from kwargs.

Sets attributes on the constructed instance using the names and values in `kwargs`.

Only keys that are present as attributes of the instance's class are allowed. These could be, for example, any mapped columns or relationships.

holding_numbers

持股数

holding_ratio

持股比例

holding_values

持股市值

actor_code

actor_id

actor_name

actor_type

code

tradable code

entity_id

entity id

classmethod get_providers() → List[str]

providers of the schema defined by cls

Returns:: providers

classmethod get_storages(*provider: Optional[str] = None*)

get the storages info

Parameters:: **provider** – provider

Returns:: storages

id

id

name

tradable name

classmethod query_data(*provider_index: int = 0*, *ids: Optional[List[str]] = None*, *entity_ids: Optional[List[str]] = None*, *entity_id: Optional[str] = None*, *codes: Optional[List[str]] = None*, *code: Optional[str] = None*, *level: Optional[Union[IntervalLevel, str]] = None*, *provider: Optional[str] = None*, *columns: Optional[List] = None*, *col_label: Optional[dict] = None*, *return_type: str = 'df'*, *start_timestamp: Optional[Union[Timestamp, str]] = None*, *end_timestamp: Optional[Union[Timestamp, str]] = None*, *filters: Optional[List] = None*, *session: Optional[Session] = None*, *order=None*, *limit: Optional[int] = None*, *distinct=None*, *index: Optional[Union[str, list]] = None*, *drop_index_col=False*, *time_field: str = 'timestamp'*)

query data by the arguments

Parameters::

- **provider_index** –
- **data_schema** –
- **ids** –
- **entity_ids** –
- **entity_id** –
- **codes** –
- **code** –
- **level** –
- **provider** –
- **columns** –
- **col_label** – dict with key(column), value(label
- **return_type** – df, domain or dict. default is df
- **start_timestamp** –
- **end_timestamp** –
- **filters** –
- **session** –
- **order** –
- **limit** –
- **index** – index field name, str for single index, str list for multiple index
- **drop_index_col** – whether drop the col if it's in index, default False
- **time_field** –

Returns:: results basing on return_type.

classmethod record_data(*provider_index: int = 0*, *provider: Optional[str] = None*, *force_update=None*, *sleeping_time=None*, *exchanges=None*, *entity_id=None*, *entity_ids=None*, *code=None*, *codes=None*, *real_time=None*, *fix_duplicate_way=None*, *start_timestamp=None*, *end_timestamp=None*, *one_day_trading_minutes=None*, ***kwargs*)

record data by the arguments

Parameters::

- **entity_id** –
- **provider_index** –
- **provider** –
- **force_update** –
- **sleeping_time** –
- **exchanges** –
- **entity_ids** –
- **code** –
- **codes** –
- **real_time** –
- **fix_duplicate_way** –
- **start_timestamp** –
- **end_timestamp** –
- **one_day_trading_minutes** –
- **kwargs** –

Returns::

classmethod register_provider(*provider*)

register the provider to the schema defined by cls

Parameters:: **provider** –

classmethod register_recorder_cls(*provider*, *recorder_cls*)

register the recorder for the schema

Parameters::

- **provider** –
- **recorder_cls** –

timestamp

the meaning could be different for different case,most time it means 'happen time'

StockTopTenHolder

class zvt.domain.actor.stock_actor.StockTopTenHolder(**kwargs)

Bases: **Base**, **TradableMeetActor**

__init__(**kwargs)

A simple constructor that allows initialization from kwargs.

Sets attributes on the constructed instance using the names and values in `kwargs`.

Only keys that are present as attributes of the instance's class are allowed. These could be, for example, any mapped columns or relationships.

holding_numbers

持股数

holding_ratio

持股比例

holding_values

持股市值

actor_code

actor_id

actor_name

actor_type

code

tradable code

entity_id

entity id

classmethod get_providers() → List[str]

providers of the schema defined by cls

Returns:: providers

classmethod get_storages(*provider: Optional[str] = None*)

get the storages info

Parameters:: **provider** – provider

Returns:: storages

id

id

name

tradable name

classmethod query_data(*provider_index: int = 0*, *ids: Optional[List[str]] = None*, *entity_ids: Optional[List[str]] = None*, *entity_id: Optional[str] = None*, *codes: Optional[List[str]] = None*, *code: Optional[str] = None*, *level: Optional[Union[IntervalLevel, str]] = None*, *provider: Optional[str] = None*, *columns: Optional[List] = None*, *col_label: Optional[dict] = None*, *return_type: str = 'df'*, *start_timestamp: Optional[Union[Timestamp, str]] = None*, *end_timestamp: Optional[Union[Timestamp, str]] = None*, *filters: Optional[List] = None*, *session: Optional[Session] = None*, *order=None*, *limit: Optional[int] = None*, *distinct=None*, *index: Optional[Union[str, list]] = None*, *drop_index_col=False*, *time_field: str = 'timestamp'*)

query data by the arguments

Parameters::

- **provider_index** –
- **data_schema** –
- **ids** –
- **entity_ids** –
- **entity_id** –
- **codes** –
- **code** –
- **level** –
- **provider** –
- **columns** –
- **col_label** – dict with key(column), value(label
- **return_type** – df, domain or dict. default is df
- **start_timestamp** –
- **end_timestamp** –
- **filters** –
- **session** –
- **order** –
- **limit** –
- **index** – index field name, str for single index, str list for multiple index
- **drop_index_col** – whether drop the col if it's in index, default False
- **time_field** –

Returns:: results basing on return_type.

classmethod record_data(*provider_index: int = 0*, *provider: Optional[str] = None*, *force_update=None*, *sleeping_time=None*, *exchanges=None*, *entity_id=None*, *entity_ids=None*, *code=None*, *codes=None*, *real_time=None*, *fix_duplicate_way=None*, *start_timestamp=None*, *end_timestamp=None*, *one_day_trading_minutes=None*, ***kwargs*)

record data by the arguments

Parameters::

- **entity_id** –
- **provider_index** –
- **provider** –
- **force_update** –
- **sleeping_time** –
- **exchanges** –
- **entity_ids** –
- **code** –
- **codes** –
- **real_time** –
- **fix_duplicate_way** –
- **start_timestamp** –
- **end_timestamp** –
- **one_day_trading_minutes** –
- **kwargs** –

Returns::

classmethod register_provider(*provider*)

register the provider to the schema defined by cls

Parameters:: **provider** –

classmethod register_recorder_cls(*provider*, *recorder_cls*)

register the recorder for the schema

Parameters::

- **provider** –
- **recorder_cls** –

timestamp

the meaning could be different for different case,most time it means 'happen time'

zvt.domain.emotion

Modules

zvt.domain.emotion.emotion

---

zvt.domain.emotion.emotion

Classes

`Emotion`(**kwargs)

`LimitDownInfo`(**kwargs)

`LimitUpInfo`(**kwargs)

Emotion

class zvt.domain.emotion.emotion.Emotion(***kwargs*)

Bases: **`Base`**, **`Mixin`**

__init__(***kwargs*)

A simple constructor that allows initialization from kwargs.

Sets attributes on the constructed instance using the names and values in `kwargs`.

Only keys that are present as attributes of the instance's class are allowed. These could be, for example, any mapped columns or relationships.

limit_up_count

涨停数量

limit_up_open_count

炸板数

limit_up_success_rate

涨停封板成功率

max_height

连板高度

continuous_power

连板数x个数 相加

limit_down_count

跌停数量

limit_down_open_count

跌停打开

limit_down_success_rate

跌停封板成功率

entity_id

entity id

classmethod get_providers() → List[str]

providers of the schema defined by cls

Returns:: providers

classmethod get_storages(*provider: Optional[str] = None*)

get the storages info

Parameters:: **provider** – provider

Returns:: storages

id

id

classmethod query_data(*provider_index: int = 0*, *ids: Optional[List[str]] = None*, *entity_ids: Optional[List[str]] = None*, *entity_id: Optional[str] = None*, *codes: Optional[List[str]] = None*, *code: Optional[str] = None*, *level: Optional[Union[IntervalLevel, str]] = None*, *provider: Optional[str] = None*, *columns: Optional[List] = None*, *col_label: Optional[dict] = None*, *return_type: str = 'df'*, *start_timestamp: Optional[Union[Timestamp, str]] = None*, *end_timestamp: Optional[Union[Timestamp, str]] = None*, *filters: Optional[List] = None*, *session: Optional[Session] = None*, *order=None*, *limit: Optional[int] = None*, *distinct=None*, *index: Optional[Union[str, list]] = None*, *drop_index_col=False*, *time_field: str = 'timestamp'*)

query data by the arguments

Parameters::

- **provider_index** –
- **data_schema** –
- **ids** –
- **entity_ids** –
- **entity_id** –
- **codes** –
- **code** –
- **level** –
- **provider** –
- **columns** –
- **col_label** – dict with key(column), value(label
- **return_type** – df, domain or dict. default is df
- **start_timestamp** –
- **end_timestamp** –
- **filters** –
- **session** –
- **order** –
- **limit** –
- **index** – index field name, str for single index, str list for multiple index
- **drop_index_col** – whether drop the col if it's in index, default False
- **time_field** –

Returns:: results basing on return_type.

classmethod record_data(*provider_index: int = 0*, *provider: Optional[str] = None*, *force_update=None*, *sleeping_time=None*, *exchanges=None*, *entity_id=None*, *entity_ids=None*, *code=None*, *codes=None*, *real_time=None*, *fix_duplicate_way=None*, *start_timestamp=None*, *end_timestamp=None*, *one_day_trading_minutes=None*, ***kwargs*)

record data by the arguments

Parameters::

- **entity_id** –
- **provider_index** –
- **provider** –
- **force_update** –
- **sleeping_time** –
- **exchanges** –
- **entity_ids** –
- **code** –
- **codes** –
- **real_time** –
- **fix_duplicate_way** –
- **start_timestamp** –
- **end_timestamp** –
- **one_day_trading_minutes** –
- **kwargs** –

Returns::

classmethod register_provider(*provider*)

register the provider to the schema defined by cls

Parameters:: **provider** –

classmethod register_recorder_cls(*provider*, *recorder_cls*)

register the recorder for the schema

Parameters::

- **provider** –
- **recorder_cls** –

timestamp

the meaning could be different for different case,most time it means 'happen time'

LimitDownInfo

class zvt.domain.emotion.emotion.LimitDownInfo(**kwargs)

Bases: **Base**, **Mixin**

__init__(**kwargs)

A simple constructor that allows initialization from kwargs.

Sets attributes on the constructed instance using the names and values in `kwargs`.

Only keys that are present as attributes of the instance's class are allowed. These could be, for example, any mapped columns or relationships.

is_new

是否新股

is_again_limit

是否回封，是就是打开过，否相反

currency_value

流通市值

change_pct

涨幅

turnover_rate

换手率

entity_id

entity id

classmethod get_providers() → List[str]

providers of the schema defined by cls

Returns:: providers

classmethod get_storages(*provider: Optional[str] = None*)

get the storages info

Parameters:: **provider** – provider
Returns:: storages

id

id

classmethod query_data(*provider_index: int = 0*, *ids: Optional[List[str]] = None*, *entity_ids: Optional[List[str]] = None*, *entity_id: Optional[str] = None*, *codes: Optional[List[str]] = None*, *code: Optional[str] = None*, *level: Optional[Union[IntervalLevel, str]] = None*, *provider: Optional[str] = None*, *columns: Optional[List] = None*, *col_label: Optional[dict] = None*, *return_type: str = 'df'*, *start_timestamp: Optional[Union[Timestamp, str]] = None*, *end_timestamp: Optional[Union[Timestamp, str]] = None*, *filters: Optional[List] = None*, *session: Optional[Session] = None*, *order=None*, *limit: Optional[int] = None*, *distinct=None*, *index: Optional[Union[str, list]] = None*, *drop_index_col=False*, *time_field: str = 'timestamp'*)

query data by the arguments

Parameters::

- **provider_index** –
- **data_schema** –
- **ids** –
- **entity_ids** –
- **entity_id** –
- **codes** –
- **code** –
- **level** –
- **provider** –
- **columns** –
- **col_label** – dict with key(column), value(label
- **return_type** – df, domain or dict. default is df
- **start_timestamp** –
- **end_timestamp** –
- **filters** –
- **session** –
- **order** –
- **limit** –
- **index** – index field name, str for single index, str list for multiple index
- **drop_index_col** – whether drop the col if it's in index, default False
- **time_field** –

Returns:: results basing on return_type.

classmethod record_data(*provider_index: int = 0*, *provider: Optional[str] = None*, *force_update=None*, *sleeping_time=None*, *exchanges=None*, *entity_id=None*, *entity_ids=None*, *code=None*, *codes=None*, *real_time=None*, *fix_duplicate_way=None*, *start_timestamp=None*, *end_timestamp=None*, *one_day_trading_minutes=None*, ***kwargs*)

record data by the arguments

Parameters::

- **entity_id** –
- **provider_index** –
- **provider** –
- **force_update** –
- **sleeping_time** –
- **exchanges** –
- **entity_ids** –
- **code** –
- **codes** –
- **real_time** –
- **fix_duplicate_way** –
- **start_timestamp** –
- **end_timestamp** –
- **one_day_trading_minutes** –
- **kwargs** –

Returns::

classmethod register_provider(*provider*)

register the provider to the schema defined by cls

Parameters:: **provider** –

classmethod register_recorder_cls(*provider*, *recorder_cls*)

register the recorder for the schema

Parameters::

- **provider** –
- **recorder_cls** –

timestamp

the meaning could be different for different case,most time it means 'happen time'

LimitUpInfo

class zvt.domain.emotion.emotion.LimitUpInfo(**kwargs)

Bases: **Base**, **Mixin**

__init__(**kwargs)

A simple constructor that allows initialization from kwargs.

Sets attributes on the constructed instance using the names and values in `kwargs`.

Only keys that are present as attributes of the instance's class are allowed. These could be, for example, any mapped columns or relationships.

is_new

是否新股

is_again_limit

是否回封，是就是打开过，否相反

open_count

涨停打开次数,0代表封住就没开板

first_limit_up_time

首次封板时间

last_limit_up_time

最后封板时间

limit_up_type

涨停类型:换手板，一字板

order_amount

封单金额

success_rate
最近一年封板成功率

currency_value
流通市值

change_pct
涨幅

turnover_rate
换手率

reason
涨停原因

high_days
几天几板

high_days_count
最近几板，不一定是连板

entity_id
entity id

classmethod get_providers() → List[str]
providers of the schema defined by cls

Returns:: providers

classmethod get_storages(*provider: Optional[str] = None*)
get the storages info

Parameters:: **provider** – provider
Returns:: storages

id

id

classmethod query_data(*provider_index: int = 0*, *ids: Optional[List[str]] = None*, *entity_ids: Optional[List[str]] = None*, *entity_id: Optional[str] = None*, *codes: Optional[List[str]] = None*, *code: Optional[str] = None*, *level: Optional[Union[IntervalLevel, str]] = None*, *provider: Optional[str] = None*, *columns: Optional[List] = None*, *col_label: Optional[dict] = None*, *return_type: str = 'df'*, *start_timestamp: Optional[Union[Timestamp, str]] = None*, *end_timestamp: Optional[Union[Timestamp, str]] = None*, *filters: Optional[List] = None*, *session: Optional[Session] = None*, *order=None*, *limit: Optional[int] = None*, *distinct=None*, *index: Optional[Union[str, list]] = None*, *drop_index_col=False*, *time_field: str = 'timestamp'*)

query data by the arguments

Parameters::

- **provider_index** –
- **data_schema** –
- **ids** –
- **entity_ids** –
- **entity_id** –
- **codes** –
- **code** –
- **level** –
- **provider** –
- **columns** –
- **col_label** – dict with key(column), value(label
- **return_type** – df, domain or dict. default is df
- **start_timestamp** –
- **end_timestamp** –
- **filters** –
- **session** –
- **order** –
- **limit** –
- **index** – index field name, str for single index, str list for multiple index
- **drop_index_col** – whether drop the col if it's in index, default False
- **time_field** –

Returns:: results basing on return_type.

classmethod record_data(*provider_index: int = 0*, *provider: Optional[str] = None*, *force_update=None*, *sleeping_time=None*, *exchanges=None*, *entity_id=None*, *entity_ids=None*, *code=None*, *codes=None*, *real_time=None*, *fix_duplicate_way=None*, *start_timestamp=None*, *end_timestamp=None*, *one_day_trading_minutes=None*, ***kwargs*)

record data by the arguments

Parameters::

- **entity_id** –
- **provider_index** –
- **provider** –
- **force_update** –
- **sleeping_time** –
- **exchanges** –
- **entity_ids** –
- **code** –
- **codes** –
- **real_time** –
- **fix_duplicate_way** –
- **start_timestamp** –
- **end_timestamp** –
- **one_day_trading_minutes** –
- **kwargs** –

Returns::

classmethod register_provider(*provider*)

register the provider to the schema defined by cls

Parameters:: **provider** –

classmethod register_recorder_cls(*provider*, *recorder_cls*)

register the recorder for the schema

Parameters::

- **provider** –
- **recorder_cls** –

timestamp

the meaning could be different for different case,most time it means ‘happen time’

zvt.domain.fundamental

Modules

zvt.domain.fundamental.dividend_financing

---

zvt.domain.fundamental.finance

---

zvt.domain.fundamental.trading

---

zvt.domain.fundamental.valuation

---

zvt.domain.fundamental.dividend_financing

Classes

`DividendDetail`(**kwargs)

`DividendFinancing`(**kwargs)

`RightsIssueDetail`(**kwargs)

`SpoDetail`(**kwargs)

DividendDetail

class zvt.domain.fundamental.dividend_financing.DividendDetail(**kwargs)

Bases: **Base**, **Mixin**

__init__(**kwargs)

A simple constructor that allows initialization from kwargs.

Sets attributes on the constructed instance using the names and values in `kwargs`.

Only keys that are present as attributes of the instance's class are allowed. These could be, for example, any mapped columns or relationships.

announce_date

公告日

record_date

股权登记日

dividend_date

除权除息日

dividend

方案

entity_id

entity id

classmethod get_providers() → List[str]

providers of the schema defined by cls

Returns:: providers

classmethod get_storages(*provider: Optional[str] = None*)

get the storages info

Parameters:: **provider** – provider

Returns:: storages

id

id

classmethod query_data(*provider_index: int = 0*, *ids: Optional[List[str]] = None*, *entity_ids: Optional[List[str]] = None*, *entity_id: Optional[str] = None*, *codes: Optional[List[str]] = None*, *code: Optional[str] = None*, *level: Optional[Union[IntervalLevel, str]] = None*, *provider: Optional[str] = None*, *columns: Optional[List] = None*, *col_label: Optional[dict] = None*, *return_type: str = 'df'*, *start_timestamp: Optional[Union[Timestamp, str]] = None*, *end_timestamp: Optional[Union[Timestamp, str]] = None*, *filters: Optional[List] = None*, *session: Optional[Session] = None*, *order=None*, *limit: Optional[int] = None*, *distinct=None*, *index: Optional[Union[str, list]] = None*, *drop_index_col=False*, *time_field: str = 'timestamp'*)

query data by the arguments

Parameters::

- **provider_index** –
- **data_schema** –
- **ids** –
- **entity_ids** –
- **entity_id** –
- **codes** –
- **code** –
- **level** –
- **provider** –
- **columns** –
- **col_label** – dict with key(column), value(label
- **return_type** – df, domain or dict. default is df
- **start_timestamp** –
- **end_timestamp** –
- **filters** –
- **session** –
- **order** –
- **limit** –
- **index** – index field name, str for single index, str list for multiple index
- **drop_index_col** – whether drop the col if it's in index, default False
- **time_field** –

Returns:: results basing on return_type.

classmethod record_data(*provider_index: int = 0*, *provider: Optional[str] = None*, *force_update=None*, *sleeping_time=None*, *exchanges=None*, *entity_id=None*, *entity_ids=None*, *code=None*, *codes=None*, *real_time=None*, *fix_duplicate_way=None*, *start_timestamp=None*, *end_timestamp=None*, *one_day_trading_minutes=None*, ***kwargs*)

record data by the arguments

Parameters::

- **entity_id** –
- **provider_index** –
- **provider** –
- **force_update** –
- **sleeping_time** –
- **exchanges** –
- **entity_ids** –
- **code** –
- **codes** –
- **real_time** –
- **fix_duplicate_way** –
- **start_timestamp** –
- **end_timestamp** –
- **one_day_trading_minutes** –
- **kwargs** –

Returns::

classmethod register_provider(*provider*)

register the provider to the schema defined by cls

Parameters:: **provider** –

classmethod register_recorder_cls(*provider*, *recorder_cls*)

register the recorder for the schema

Parameters::

- **provider** –
- **recorder_cls** –

timestamp

the meaning could be different for different case,most time it means 'happen time'

DividendFinancing

class zvt.domain.fundamental.dividend_financing.DividendFinancing(**kwargs)

Bases: **Base**, **Mixin**

__init__(**kwargs)

A simple constructor that allows initialization from kwargs.

Sets attributes on the constructed instance using the names and values in `kwargs`.

Only keys that are present as attributes of the instance's class are allowed. These could be, for example, any mapped columns or relationships.

dividend_money

分红总额

ipo_issues

新股

spo_issues

增发

rights_issues

配股

entity_id

entity id

classmethod get_providers() → List[str]

providers of the schema defined by cls

Returns:: providers

classmethod get_storages(*provider: Optional[str] = None*)

get the storages info

Parameters:: **provider** – provider

Returns:: storages

id

id

classmethod query_data(*provider_index: int = 0*, *ids: Optional[List[str]] = None*, *entity_ids: Optional[List[str]] = None*, *entity_id: Optional[str] = None*, *codes: Optional[List[str]] = None*, *code: Optional[str] = None*, *level: Optional[Union[IntervalLevel, str]] = None*, *provider: Optional[str] = None*, *columns: Optional[List] = None*, *col_label: Optional[dict] = None*, *return_type: str = 'df'*, *start_timestamp: Optional[Union[Timestamp, str]] = None*, *end_timestamp: Optional[Union[Timestamp, str]] = None*, *filters: Optional[List] = None*, *session: Optional[Session] = None*, *order=None*, *limit: Optional[int] = None*, *distinct=None*, *index: Optional[Union[str, list]] = None*, *drop_index_col=False*, *time_field: str = 'timestamp'*)

query data by the arguments

Parameters::

- **provider_index** –
- **data_schema** –
- **ids** –
- **entity_ids** –
- **entity_id** –
- **codes** –
- **code** –
- **level** –
- **provider** –
- **columns** –
- **col_label** – dict with key(column), value(label
- **return_type** – df, domain or dict. default is df
- **start_timestamp** –
- **end_timestamp** –
- **filters** –
- **session** –
- **order** –
- **limit** –
- **index** – index field name, str for single index, str list for multiple index
- **drop_index_col** – whether drop the col if it's in index, default False
- **time_field** –

Returns:: results basing on return_type.

classmethod record_data(*provider_index: int = 0*, *provider: Optional[str] = None*, *force_update=None*, *sleeping_time=None*, *exchanges=None*, *entity_id=None*, *entity_ids=None*, *code=None*, *codes=None*, *real_time=None*, *fix_duplicate_way=None*, *start_timestamp=None*, *end_timestamp=None*, *one_day_trading_minutes=None*, ***kwargs*)

record data by the arguments

Parameters::

- **entity_id** –
- **provider_index** –
- **provider** –
- **force_update** –
- **sleeping_time** –
- **exchanges** –
- **entity_ids** –
- **code** –
- **codes** –
- **real_time** –
- **fix_duplicate_way** –
- **start_timestamp** –
- **end_timestamp** –
- **one_day_trading_minutes** –
- **kwargs** –

Returns::

classmethod register_provider(*provider*)

register the provider to the schema defined by cls

Parameters:: **provider** –

classmethod register_recorder_cls(*provider*, *recorder_cls*)

register the recorder for the schema

Parameters::

- **provider** –
- **recorder_cls** –

timestamp

the meaning could be different for different case,most time it means 'happen time'

RightsIssueDetail

class zvt.domain.fundamental.dividend_financing.RightsIssueDetail(***kwargs*)

Bases: **`Base`**, **`Mixin`**

__init__(***kwargs*)

A simple constructor that allows initialization from kwargs.

Sets attributes on the constructed instance using the names and values in `kwargs`.

Only keys that are present as attributes of the instance's class are allowed. These could be, for example, any mapped columns or relationships.

rights_issues

配股

entity_id

entity id

classmethod get_providers() → List[str]

providers of the schema defined by cls

Returns:: providers

classmethod get_storages(*provider: Optional[str] = None*)

get the storages info

Parameters:: **provider** – provider
Returns:: storages

id

id

classmethod query_data(*provider_index: int = 0*, *ids: Optional[List[str]] = None*, *entity_ids: Optional[List[str]] = None*, *entity_id: Optional[str] = None*, *codes: Optional[List[str]] = None*, *code: Optional[str] = None*, *level: Optional[Union[IntervalLevel, str]] = None*, *provider: Optional[str] = None*, *columns: Optional[List] = None*, *col_label: Optional[dict] = None*, *return_type: str = 'df'*, *start_timestamp: Optional[Union[Timestamp, str]] = None*, *end_timestamp: Optional[Union[Timestamp, str]] = None*, *filters: Optional[List] = None*, *session: Optional[Session] = None*, *order=None*, *limit: Optional[int] = None*, *distinct=None*, *index: Optional[Union[str, list]] = None*, *drop_index_col=False*, *time_field: str = 'timestamp'*)

query data by the arguments

Parameters::

- **provider_index** –
- **data_schema** –
- **ids** –
- **entity_ids** –
- **entity_id** –
- **codes** –
- **code** –
- **level** –
- **provider** –
- **columns** –
- **col_label** – dict with key(column), value(label
- **return_type** – df, domain or dict. default is df
- **start_timestamp** –
- **end_timestamp** –
- **filters** –
- **session** –
- **order** –
- **limit** –
- **index** – index field name, str for single index, str list for multiple index
- **drop_index_col** – whether drop the col if it's in index, default False
- **time_field** –

Returns:: results basing on return_type.

classmethod record_data(*provider_index: int = 0*, *provider: Optional[str] = None*, *force_update=None*, *sleeping_time=None*, *exchanges=None*, *entity_id=None*, *entity_ids=None*, *code=None*, *codes=None*, *real_time=None*, *fix_duplicate_way=None*, *start_timestamp=None*, *end_timestamp=None*, *one_day_trading_minutes=None*, ***kwargs*)

record data by the arguments

Parameters::

- **entity_id** –
- **provider_index** –
- **provider** –
- **force_update** –
- **sleeping_time** –
- **exchanges** –
- **entity_ids** –
- **code** –
- **codes** –
- **real_time** –
- **fix_duplicate_way** –
- **start_timestamp** –
- **end_timestamp** –
- **one_day_trading_minutes** –
- **kwargs** –

Returns::

classmethod register_provider(*provider*)

register the provider to the schema defined by cls

Parameters:: **provider** –

classmethod register_recorder_cls(*provider*, *recorder_cls*)

register the recorder for the schema

Parameters::

- **provider** –
- **recorder_cls** –

timestamp

the meaning could be different for different case,most time it means 'happen time'

SpoDetail

class zvt.domain.fundamental.dividend_financing.SpoDetail(***kwargs*)

Bases: **Base**, **Mixin**

__init__(***kwargs*)

A simple constructor that allows initialization from kwargs.

Sets attributes on the constructed instance using the names and values in `kwargs`.

Only keys that are present as attributes of the instance's class are allowed. These could be, for example, any mapped columns or relationships.

entity_id

entity id

classmethod get_providers() → List[str]

providers of the schema defined by cls

Returns:: providers

classmethod get_storages(*provider: Optional[str] = None*)

get the storages info

Parameters:: **provider** – provider
Returns:: storages

id

id

classmethod query_data(*provider_index: int = 0*, *ids: Optional[List[str]] = None*, *entity_ids: Optional[List[str]] = None*, *entity_id: Optional[str] = None*, *codes: Optional[List[str]] = None*, *code: Optional[str] = None*, *level: Optional[Union[IntervalLevel, str]] = None*, *provider:*

Optional[str] = None, *columns: Optional[List] = None*, *col_label: Optional[dict] = None*, *return_type: str = 'df'*, *start_timestamp: Optional[Union[Timestamp, str]] = None*, *end_timestamp: Optional[Union[Timestamp, str]] = None*, *filters: Optional[List] = None*, *session: Optional[Session] = None*, *order=None*, *limit: Optional[int] = None*, *distinct=None*, *index: Optional[Union[str, list]] = None*, *drop_index_col=False*, *time_field: str = 'timestamp'*)

query data by the arguments

Parameters::

- **provider_index** –
- **data_schema** –
- **ids** –
- **entity_ids** –
- **entity_id** –
- **codes** –
- **code** –
- **level** –
- **provider** –
- **columns** –
- **col_label** – dict with key(column), value(label
- **return_type** – df, domain or dict. default is df
- **start_timestamp** –
- **end_timestamp** –
- **filters** –
- **session** –
- **order** –
- **limit** –
- **index** – index field name, str for single index, str list for multiple index
- **drop_index_col** – whether drop the col if it's in index, default False
- **time_field** –

Returns:: results basing on return_type.

classmethod record_data(*provider_index: int = 0*, *provider: Optional[str] = None*, *force_update=None*, *sleeping_time=None*, *exchanges=None*, *entity_id=None*, *entity_ids=None*, *code=None*, *codes=None*,

real_time=None, *fix_duplicate_way=None*, *start_timestamp=None*, *end_timestamp=None*, *one_day_trading_minutes=None*, ***kwargs*)

record data by the arguments

Parameters::

- **entity_id** –
- **provider_index** –
- **provider** –
- **force_update** –
- **sleeping_time** –
- **exchanges** –
- **entity_ids** –
- **code** –
- **codes** –
- **real_time** –
- **fix_duplicate_way** –
- **start_timestamp** –
- **end_timestamp** –
- **one_day_trading_minutes** –
- **kwargs** –

Returns::

classmethod register_provider(*provider*)

register the provider to the schema defined by cls

Parameters:: **provider** –

classmethod register_recorder_cls(*provider*, *recorder_cls*)

register the recorder for the schema

Parameters::

- **provider** –
- **recorder_cls** –

timestamp

the meaning could be different for different case,most time it means 'happen time'

zvt.domain.fundamental.finance

Classes

`BalanceSheet`(**kwargs)

`CashFlowStatement`(**kwargs)

`FinanceFactor`(**kwargs)

`IncomeStatement`(**kwargs)

BalanceSheet

class zvt.domain.fundamental.finance.BalanceSheet(**kwargs)

Bases: **Base**, **Mixin**

__init__(**kwargs)

A simple constructor that allows initialization from kwargs.

Sets attributes on the constructed instance using the names and values in `kwargs`.

Only keys that are present as attributes of the instance's class are allowed. These could be, for example, any mapped columns or relationships.

cash_and_cash_equivalents

货币资金

note_receivable

应收票据

accounts_receivable

应收账款

advances_to_suppliers

预付款项

other_receivables

其他应收款

inventories

存货

current_portion_of_non_current_assets

一年内到期的非流动资产

other_current_assets
其他流动资产

total_current_assets
流动资产合计

fi_assets_saleable
可供出售金融资产

long_term_receivables
长期应收款

long_term_equity_investment
长期股权投资

real_estate_investment
投资性房地产

fixed_assets
固定资产

construction_in_process
在建工程

intangible_assets
无形资产

goodwill
商誉

long_term_prepaid_expenses
长期待摊费用

deferred_tax_assets

递延所得税资产

other_non_current_assets
其他非流动资产

total_non_current_assets
非流动资产合计

total_assets
资产总计

short_term_borrowing
短期借款

accept_money_deposits
吸收存款及同业存放

accounts_payable
应付账款

advances_from_customers
预收款项

employee_benefits_payable
应付职工薪酬

taxes_payable
应交税费

interest_payable
应付利息

other_payable
其他应付款

current_portion_of_non_current_liabilities

一年内到期的非流动负债

other_current_liabilities
其他流动负债

total_current_liabilities
流动负债合计

long_term_borrowing
长期借款

long_term_payable
长期应付款

deferred_revenue
递延收益

deferred_tax_liabilities
递延所得税负债

other_non_current_liabilities
其他非流动负债

total_non_current_liabilities
非流动负债合计

total_liabilities
负债合计

capital
实收资本（或股本）

capital_reserve
资本公积

special_reserve

专项储备

surplus_reserve
盈余公积

undistributed_profits
未分配利润

equity
归属于母公司股东权益合计

equity_as_minority_interest
少数股东权益

total_equity
股东权益合计

total_liabilities_and_equity
负债和股东权益合计

fi_cash_and_deposit_in_central_bank
银行相关 资产 现金及存放中央银行款项

fi_deposit_in_other_fi
存放同业款项

fi_expensive_metals
贵金属

fi_lending_to_other_fi
拆出资金

fi_financial_assets_effect_current_income
以公允价值计量且其变动计入当期损益的金融资产

fi_financial_derivative_asset

衍生金融资产

fi_buying_sell_back_fi__asset
买入返售金融资产

fi_interest_receivable
应收利息

fi_disbursing_loans_and_advances
发放贷款及垫款

fi_held_to_maturity_investment
持有至到期投资

fi_account_receivable_investment
应收款项类投资

fi_other_asset
其他资产

fi_borrowings_from_central_bank
向中央银行借款

fi_deposit_from_other_fi
同业和其他金融机构存放款项

fi_borrowings_from_fi
拆入资金

fi_financial_liability_effect_current_income
以公允价值计量且其变动计入当期损益的金融负债

fi_financial_derivative_liability
衍生金融负债

fi_sell_buy_back_fi_asset

卖出回购金融资产款

fi_savings_absorption
吸收存款

fi_notes_payable
存款证及应付票据

fi_estimated_liabilities
预计负债

fi_bond_payable
应付债券

fi_other_liability
其他负债

fi_capital
所有者权益(或股东权益) 股本

fi_other_equity_instruments
其他权益工具

fi_preferred_stock
其中:优先股

fi_generic_risk_reserve
一般风险准备

fi_client_fund
其中: 客户资金存款

fi_deposit_reservation_for_balance
结算备付金

fi_client_deposit_reservation_for_balance

其中: 客户备付金

fi_margin_out_fund
融出资金

fi_receivables
应收款项

fi_deposit_for_recognizance
存出保证金

fi_receiving_as_agent
代理买卖证券款

fi_short_financing_payable
应付短期融资款

fi_trade_risk_reserve
交易风险准备

fi_premiums_receivable
资产 应收保费

fi_reinsurance_premium_receivable
应收分保账款

fi_reinsurance_contract_reserve
应收分保合同准备金

fi_policy_pledge_loans
保户质押贷款

fi_time_deposit
发放贷款及垫款 定期存款

fi_deposit_for_capital_recognizance

存出资本保证金

fi_capital_in_independent_accounts
独立账户资产

fi_advance_from_customers
预收账款

fi_advance_premium
预收保费

fi_fees_and_commissions_payable
应付手续费及佣金

fi_dividend_payable_for_reinsurance
应付分保账款

fi_claims_payable
应付赔付款

fi_policy_holder_dividend_payable
应付保单红利

fi_policy_holder_deposits_and_investment_funds
保户储金及投资款

fi_contract_reserve
保险合同准备金

fi_independent_liability
独立账户负债

entity_id
entity id

classmethod get_providers() → List[str]

providers of the schema defined by cls

Returns:: providers

classmethod get_storages(*provider: Optional[str] = None*)

get the storages info

Parameters:: **provider** – provider
Returns:: storages

id

id

classmethod query_data(*provider_index: int = 0*, *ids: Optional[List[str]] = None*, *entity_ids: Optional[List[str]] = None*, *entity_id: Optional[str] = None*, *codes: Optional[List[str]] = None*, *code: Optional[str] = None*, *level: Optional[Union[IntervalLevel, str]] = None*, *provider: Optional[str] = None*, *columns: Optional[List] = None*, *col_label: Optional[dict] = None*, *return_type: str = 'df'*, *start_timestamp: Optional[Union[Timestamp, str]] = None*, *end_timestamp: Optional[Union[Timestamp, str]] = None*, *filters: Optional[List] = None*, *session: Optional[Session] = None*, *order=None*, *limit: Optional[int] = None*, *distinct=None*, *index: Optional[Union[str, list]] = None*, *drop_index_col=False*, *time_field: str = 'timestamp'*)

query data by the arguments

Parameters::

- **provider_index** –
- **data_schema** –
- **ids** –
- **entity_ids** –
- **entity_id** –
- **codes** –
- **code** –
- **level** –
- **provider** –
- **columns** –
- **col_label** – dict with key(column), value(label
- **return_type** – df, domain or dict. default is df
- **start_timestamp** –
- **end_timestamp** –
- **filters** –
- **session** –
- **order** –
- **limit** –
- **index** – index field name, str for single index, str list for multiple index
- **drop_index_col** – whether drop the col if it's in index, default False
- **time_field** –

Returns:: results basing on return_type.

classmethod record_data(*provider_index: int = 0*, *provider: Optional[str] = None*, *force_update=None*, *sleeping_time=None*, *exchanges=None*, *entity_id=None*, *entity_ids=None*, *code=None*, *codes=None*, *real_time=None*, *fix_duplicate_way=None*, *start_timestamp=None*, *end_timestamp=None*, *one_day_trading_minutes=None*, ***kwargs*)

record data by the arguments

Parameters::

- **entity_id** –
- **provider_index** –
- **provider** –
- **force_update** –
- **sleeping_time** –
- **exchanges** –
- **entity_ids** –
- **code** –
- **codes** –
- **real_time** –
- **fix_duplicate_way** –
- **start_timestamp** –
- **end_timestamp** –
- **one_day_trading_minutes** –
- **kwargs** –

Returns::

classmethod register_provider(*provider*)

register the provider to the schema defined by cls

Parameters:: **provider** –

classmethod register_recorder_cls(*provider*, *recorder_cls*)

register the recorder for the schema

Parameters::

- **provider** –
- **recorder_cls** –

timestamp

the meaning could be different for different case,most time it means 'happen time'

CashFlowStatement

class zvt.domain.fundamental.finance.CashFlowStatement(**kwargs)

Bases: **Base**, **Mixin**

__init__(**kwargs)

A simple constructor that allows initialization from kwargs.

Sets attributes on the constructed instance using the names and values in `kwargs`.

Only keys that are present as attributes of the instance's class are allowed. These could be, for example, any mapped columns or relationships.

cash_from_selling

销售商品、提供劳务收到的现金

tax_refund

收到的税费返还

cash_from_other_op

收到其他与经营活动有关的现金

total_op_cash_inflows

经营活动现金流入小计

cash_to_goods_services

购买商品、接受劳务支付的现金

cash_to_employees

支付给职工以及为职工支付的现金

taxes_and_surcharges

支付的各项税费

cash_to_other_related_op
支付其他与经营活动有关的现金

total_op_cash_outflows
经营活动现金流出小计

net_op_cash_flows
经营活动产生的现金流量净额

cash_from_disposal_of_investments
收回投资收到的现金

cash_from_returns_on_investments
取得投资收益收到的现金

cash_from_disposal_fixed_intangible_assets
处置固定资产、无形资产和其他长期资产收回的现金净额

cash_from_disposal_subsidiaries
处置子公司及其他营业单位收到的现金净额

cash_from_other_investing
收到其他与投资活动有关的现金

total_investing_cash_inflows
投资活动现金流入小计

cash_to_acquire_fixed_intangible_assets
购建固定资产、无形资产和其他长期资产支付的现金

cash_to_investments
投资支付的现金

cash_to_acquire_subsidiaries

取得子公司及其他营业单位支付的现金净额

cash_to_other_investing
支付其他与投资活动有关的现金

total_investing_cash_outflows
投资活动现金流出小计

net_investing_cash_flows
投资活动产生的现金流量净额

cash_from_accepting_investment
吸收投资收到的现金

cash_from_subsidiaries_accepting_minority_interest
子公司吸收少数股东投资收到的现金

cash_from_borrowings
取得借款收到的现金

cash_from_issuing_bonds
发行债券收到的现金

cash_from_other_financing
收到其他与筹资活动有关的现金

total_financing_cash_inflows
筹资活动现金流入小计

cash_to_repay_borrowings
偿还债务支付的现金

cash_to_pay_interest_dividend
分配股利、利润或偿付利息支付的现金

cash_to_pay_subsidiaries_minority_interest

子公司支付给少数股东的股利、利润

cash_to_other_financing
支付其他与筹资活动有关的现金

total_financing_cash_outflows
筹资活动现金流出小计

net_financing_cash_flows
筹资活动产生的现金流量净额

foreign_exchange_rate_effect
汇率变动对现金及现金等价物的影响

net_cash_increase
现金及现金等价物净增加额

cash_at_beginning
加: 期初现金及现金等价物余额

cash
期末现金及现金等价物余额

fi_deposit_increase
银行相关 客户存款和同业及其他金融机构存放款项净增加额

fi_borrow_from_central_bank_increase
向中央银行借款净增加额

fi_deposit_in_others_decrease
存放中央银行和同业款项及其他金融机构净减少额

fi_borrowing_and_sell_repurchase_increase
拆入资金及卖出回购金融资产款净增加额

fi_sell_repurchase_increase

其中:卖出回购金融资产款净增加额

fi_lending_and_buy_repurchase_decrease
拆出资金及买入返售金融资产净减少额

fi_buy_repurchase_decrease
买入返售金融资产净减少额

fi_cash_from_interest_commission
收取的利息、手续费及佣金的现金

fi_loan_advance_increase
客户贷款及垫款净增加额

fi_deposit_in_others_increase
存放中央银行和同业及其他金融机构款项净增加额

fi_lending_and_buy_repurchase_increase
拆出资金及买入返售金融资产净增加额

fi_lending_increase
其中:拆出资金净增加额

fi_borrowing_and_sell_repurchase_decrease
拆入资金及卖出回购金融资产款净减少额

fi_borrowing_decrease
其中:拆入资金净减少额

fi_sell_repurchase_decrease
卖出回购金融资产净减少额

fi_cash_to_interest_commission
支付利息、手续费及佣金的现金

fi_account_receivable_increase

应收账款净增加额

fi_cash_to_pay_interest
偿付债券利息支付的现金

fi_cash_from_premium_of_original
保险相关 收到原保险合同保费取得的现金

fi_insured_deposit_increase
保户储金及投资款净增加额

fi_bank_broker_sell_repurchase_increase
银行及证券业务卖出回购资金净增加额

fi_bank_broker_buy_repurchase_decrease
银行及证券业务买入返售资金净减少额

fi_cash_to_insurance_claim
支付原保险合同赔付等款项的现金

fi_cash_to_reinsurance
支付再保险业务现金净额

fi_lending_decrease
银行业务及证券业务拆借资金净减少额

fi_bank_broker_sell_repurchase_decrease
银行业务及证券业务卖出回购资金净减少额

fi_cash_to_dividends
支付保单红利的现金

fi_insured_pledge_loans_increase
保户质押贷款净增加额

fi_cash_to_acquire_subsidiaries

收购子公司及其他营业单位支付的现金净额

fi_cash_to_disposal_subsidiaries
处置子公司及其他营业单位流出的现金净额

fi_cash_to_sell_repurchase
支付卖出回购金融资产款现金净额

fi_borrowing_increase
券商相关 拆入资金净增加额

fi_cash_from_trading_agent
代理买卖证券收到的现金净额

fi_cash_from_repurchase_increase
回购业务资金净增加额

fi_disposal_trade_asset_decrease
处置交易性金融资产的净减少额

fi_repurchase_decrease
回购业务资金净减少额

fi_cash_to_agent_trade
代理买卖证券支付的现金净额（净减少额）

entity_id
entity id

classmethod get_providers() → List[str]
providers of the schema defined by cls

Returns:: providers

classmethod get_storages(*provider: Optional[str] = None*)
get the storages info

Parameters:: **provider** – provider
Returns:: storages

id

id

classmethod query_data(*provider_index: int = 0*, *ids: Optional[List[str]] = None*, *entity_ids: Optional[List[str]] = None*, *entity_id: Optional[str] = None*, *codes: Optional[List[str]] = None*, *code: Optional[str] = None*, *level: Optional[Union[IntervalLevel, str]] = None*, *provider: Optional[str] = None*, *columns: Optional[List] = None*, *col_label: Optional[dict] = None*, *return_type: str = 'df'*, *start_timestamp: Optional[Union[Timestamp, str]] = None*, *end_timestamp: Optional[Union[Timestamp, str]] = None*, *filters: Optional[List] = None*, *session: Optional[Session] = None*, *order=None*, *limit: Optional[int] = None*, *distinct=None*, *index: Optional[Union[str, list]] = None*, *drop_index_col=False*, *time_field: str = 'timestamp'*)

query data by the arguments

Parameters::

- **provider_index** –
- **data_schema** –
- **ids** –
- **entity_ids** –
- **entity_id** –
- **codes** –
- **code** –
- **level** –
- **provider** –
- **columns** –
- **col_label** – dict with key(column), value(label
- **return_type** – df, domain or dict. default is df
- **start_timestamp** –
- **end_timestamp** –
- **filters** –
- **session** –
- **order** –
- **limit** –
- **index** – index field name, str for single index, str list for multiple index
- **drop_index_col** – whether drop the col if it's in index, default False
- **time_field** –

Returns:: results basing on return_type.

classmethod record_data(*provider_index: int = 0*, *provider: Optional[str] = None*, *force_update=None*, *sleeping_time=None*, *exchanges=None*, *entity_id=None*, *entity_ids=None*, *code=None*, *codes=None*, *real_time=None*, *fix_duplicate_way=None*, *start_timestamp=None*, *end_timestamp=None*, *one_day_trading_minutes=None*, ***kwargs*)

record data by the arguments

Parameters::

- **entity_id** –
- **provider_index** –
- **provider** –
- **force_update** –
- **sleeping_time** –
- **exchanges** –
- **entity_ids** –
- **code** –
- **codes** –
- **real_time** –
- **fix_duplicate_way** –
- **start_timestamp** –
- **end_timestamp** –
- **one_day_trading_minutes** –
- **kwargs** –

Returns::

classmethod register_provider(*provider*)

register the provider to the schema defined by cls

Parameters:: **provider** –

classmethod register_recorder_cls(*provider*, *recorder_cls*)

register the recorder for the schema

Parameters::

- **provider** –
- **recorder_cls** –

timestamp

the meaning could be different for different case,most time it means 'happen time'

FinanceFactor

class zvt.domain.fundamental.finance.FinanceFactor(**kwargs)

Bases: **Base**, **Mixin**

__init__(**kwargs)

A simple constructor that allows initialization from kwargs.

Sets attributes on the constructed instance using the names and values in `kwargs`.

Only keys that are present as attributes of the instance's class are allowed. These could be, for example, any mapped columns or relationships.

basic_eps

基本每股收益(元)

deducted_eps

扣非每股收益(元)

diluted_eps

稀释每股收益(元)

bps

每股净资产(元)

capital_reserve_ps

每股资本公积(元)

undistributed_profit_ps

每股未分配利润(元)

op_cash_flow_ps

每股经营现金流(元)

total_op_income
营业总收入(元)

gross_profit
毛利润(元)

net_profit
归属净利润(元)

deducted_net_profit
扣非净利润(元)

op_income_growth_yoy
营业总收入同比增长

net_profit_growth_yoy
归属净利润同比增长

deducted_net_profit_growth_yoy
扣非净利润同比增长

op_income_growth_qoq
营业总收入滚动环比增长

net_profit_growth_qoq
归属净利润滚动环比增长

deducted_net_profit_growth_qoq
扣非净利润滚动环比增长

roe
净资产收益率(加权)

deducted_roe

净资产收益率(扣非/加权)

rota
总资产收益率(加权)

gross_profit_margin
毛利率

net_margin
净利率

advance_receipts_per_op_income
预收账款/营业收入

sales_net_cash_flow_per_op_income
销售净现金流/营业收入

op_net_cash_flow_per_op_income
经营净现金流/营业收入

actual_tax_rate
实际税率

current_ratio
流动比率

quick_ratio
速动比率

cash_flow_ratio
现金流量比率

debt_asset_ratio
资产负债率

em

权益乘数

equity_ratio
产权比率

total_assets_turnover_days
总资产周转天数(天)

inventory_turnover_days
存货周转天数(天)

receivables_turnover_days
应收账款周转天数(天)

total_assets_turnover
总资产周转率(次)

inventory_turnover
存货周转率(次)

receivables_turnover
应收账款周转率(次)

fi_total_deposit
存款总额

fi_total_loan
贷款总额

fi_loan_deposit_ratio
存贷款比例

fi_capital_adequacy_ratio
资本充足率

fi_core_capital_adequacy_ratio

核心资本充足率

fi_npl_ratio
不良贷款率

fi_npl_provision_coverage
不良贷款拨备覆盖率

fi_net_capital
资本净额

insurance_roi
总投资收益率

insurance_net_investment_yield
净投资收益率

insurance_earned_premium
已赚保费

insurance_payout
赔付支出

insurance_surrender_rate
退保率

insurance_solvency_adequacy_ratio
偿付能力充足率

broker_net_capital
净资本

broker_net_assets
净资产

broker_net_capital_assets_ratio

净资本/净资产

broker_self_operated_fixed_income_securities_net_capital_ratio
自营固定收益类证券规模/净资本

entity_id
entity id

classmethod get_providers() → List[str]
providers of the schema defined by cls

Returns:: providers

classmethod get_storages(*provider: Optional[str] = None*)
get the storages info

Parameters:: **provider** – provider
Returns:: storages

id
id

classmethod query_data(*provider_index: int = 0*, *ids: Optional[List[str]] = None*, *entity_ids: Optional[List[str]] = None*, *entity_id: Optional[str] = None*, *codes: Optional[List[str]] = None*, *code: Optional[str] = None*, *level: Optional[Union[IntervalLevel, str]] = None*, *provider: Optional[str] = None*, *columns: Optional[List] = None*, *col_label: Optional[dict] = None*, *return_type: str = 'df'*, *start_timestamp: Optional[Union[Timestamp, str]] = None*, *end_timestamp: Optional[Union[Timestamp, str]] = None*, *filters: Optional[List] = None*, *session: Optional[Session] = None*, *order=None*, *limit: Optional[int] = None*, *distinct=None*, *index: Optional[Union[str, list]] = None*, *drop_index_col=False*, *time_field: str = 'timestamp'*)
query data by the arguments

Parameters::

- **provider_index** –
- **data_schema** –
- **ids** –
- **entity_ids** –
- **entity_id** –
- **codes** –
- **code** –
- **level** –
- **provider** –
- **columns** –
- **col_label** – dict with key(column), value(label
- **return_type** – df, domain or dict. default is df
- **start_timestamp** –
- **end_timestamp** –
- **filters** –
- **session** –
- **order** –
- **limit** –
- **index** – index field name, str for single index, str list for multiple index
- **drop_index_col** – whether drop the col if it's in index, default False
- **time_field** –

Returns:: results basing on return_type.

classmethod record_data(*provider_index: int = 0*, *provider: Optional[str] = None*, *force_update=None*, *sleeping_time=None*, *exchanges=None*, *entity_id=None*, *entity_ids=None*, *code=None*, *codes=None*, *real_time=None*, *fix_duplicate_way=None*, *start_timestamp=None*, *end_timestamp=None*, *one_day_trading_minutes=None*, ***kwargs*)

record data by the arguments

Parameters::

- **entity_id** –
- **provider_index** –
- **provider** –
- **force_update** –
- **sleeping_time** –
- **exchanges** –
- **entity_ids** –
- **code** –
- **codes** –
- **real_time** –
- **fix_duplicate_way** –
- **start_timestamp** –
- **end_timestamp** –
- **one_day_trading_minutes** –
- **kwargs** –

Returns::

classmethod register_provider(*provider*)

register the provider to the schema defined by cls

Parameters:: **provider** –

classmethod register_recorder_cls(*provider*, *recorder_cls*)

register the recorder for the schema

Parameters::

- **provider** –
- **recorder_cls** –

timestamp

the meaning could be different for different case,most time it means 'happen time'

IncomeStatement

class zvt.domain.fundamental.finance.IncomeStatement(**kwargs)

Bases: **Base**, **Mixin**

__init__(**kwargs)

A simple constructor that allows initialization from kwargs.

Sets attributes on the constructed instance using the names and values in `kwargs`.

Only keys that are present as attributes of the instance's class are allowed. These could be, for example, any mapped columns or relationships.

operating_income

营业收入

total_operating_costs

营业总成本

operating_costs

营业成本

rd_costs

研发费用

net_change_in_insurance_contract_reserves

提取保险合同准备金净额

business_taxes_and_surcharges

营业税金及附加

sales_costs

销售费用

managing_costs
管理费用

financing_costs
财务费用

assets_devaluation
资产减值损失

investment_income
加: 投资收益

investment_income_from_related_enterprise
其中: 对联营企业和合营企业的投资收益

operating_profit
营业利润

non_operating_income
加: 营业外收入

non_operating_costs
减: 营业外支出

loss_on_disposal_non_current_asset
其中: 非流动资产处置净损失

total_profits
利润总额

tax_expense
减: 所得税费用

net_profit

净利润

net_profit_as_parent
其中: 归属于母公司股东的净利润

net_profit_as_minority_interest
少数股东损益

deducted_net_profit
扣除非经常性损益后的净利润

eps
每股收益 基本每股收益

diluted_eps
稀释每股收益

other_comprehensive_income
其他综合收益

other_comprehensive_income_as_parent
归属于母公司股东的其他综合收益

other_comprehensive_income_as_minority_interest
归属于少数股东的其他综合收益

total_comprehensive_income
综合收益总额

total_comprehensive_income_as_parent
归属于母公司所有者的综合收益总额

total_comprehensive_income_as_minority_interest
归属于少数股东的综合收益总额

fi_net_interest_income

银行相关 利息净收入

fi_interest_income
其中:利息收入

fi_interest_expenses
利息支出

fi_net_incomes_from_fees_and_commissions
手续费及佣金净收入

fi_incomes_from_fees_and_commissions
其中:手续费及佣金收入

fi_expenses_for_fees_and_commissions
手续费及佣金支出

fi_income_from_fair_value_change
公允价值变动收益

fi_income_from_exchange
汇兑收益

fi_other_income
其他业务收入

fi_operate_and_manage_expenses
业务及管理费

fi_net_income_from_premium
保险相关 已赚保费

fi_income_from_premium
其中:保险业务收入

fi_income_from_reinsurance_premium

分保费收入

fi_reinsurance_premium
减:分出保费

fi_undue_duty_reserve
提取未到期责任准备金

fi_net_income_from_bank_interest
银行业务利息净收入

fi_income_from_bank_interest
其中:银行业务利息收入

fi_expenses_for_bank_interest
银行业务利息支出

fi_net_incomes_from_fees_and_commissions_of_non_insurance
非保险业务手续费及佣金净收入

fi_incomes_from_fees_and_commissions_of_non_insurance
非保险业务手续费及佣金收入

fi_expenses_for_fees_and_commissions_of_non_insurance
非保险业务手续费及佣金支出

fi_insurance_surrender_costs
退保金

fi_insurance_claims_expenses
赔付支出

fi_amortized_insurance_claims_expenses
减:摊回赔付支出

fi_insurance_duty_reserve

提取保险责任准备金

fi_amortized_insurance_duty_reserve
减:摊回保险责任准备金

fi_dividend_expenses_to_insured
保单红利支出

fi_reinsurance_expenses
分保费用

fi_amortized_reinsurance_expenses
减:摊回分保费用

fi_other_op_expenses
其他业务成本

fi_net_incomes_from_trading_agent
其中:代理买卖证券业务净收入

fi_net_incomes_from_underwriting
证券承销业务净收入

fi_net_incomes_from_customer_asset_management
受托客户资产管理业务净收入

fi_fees_from_other
手续费及佣金净收入其他项目

fi_income_from_fair_value_change_of_fi_salable
其中:可供出售金融资产公允价值变动损益

entity_id
entity id

classmethod get_providers() → List[str]

providers of the schema defined by cls

Returns:: providers

classmethod get_storages(*provider: Optional[str] = None*)

get the storages info

Parameters:: **provider** – provider
Returns:: storages

id

id

classmethod query_data(*provider_index: int = 0*, *ids: Optional[List[str]] = None*, *entity_ids: Optional[List[str]] = None*, *entity_id: Optional[str] = None*, *codes: Optional[List[str]] = None*, *code: Optional[str] = None*, *level: Optional[Union[IntervalLevel, str]] = None*, *provider: Optional[str] = None*, *columns: Optional[List] = None*, *col_label: Optional[dict] = None*, *return_type: str = 'df'*, *start_timestamp: Optional[Union[Timestamp, str]] = None*, *end_timestamp: Optional[Union[Timestamp, str]] = None*, *filters: Optional[List] = None*, *session: Optional[Session] = None*, *order=None*, *limit: Optional[int] = None*, *distinct=None*, *index: Optional[Union[str, list]] = None*, *drop_index_col=False*, *time_field: str = 'timestamp'*)

query data by the arguments

Parameters::

- **provider_index** –
- **data_schema** –
- **ids** –
- **entity_ids** –
- **entity_id** –
- **codes** –
- **code** –
- **level** –
- **provider** –
- **columns** –
- **col_label** – dict with key(column), value(label
- **return_type** – df, domain or dict. default is df
- **start_timestamp** –
- **end_timestamp** –
- **filters** –
- **session** –
- **order** –
- **limit** –
- **index** – index field name, str for single index, str list for multiple index
- **drop_index_col** – whether drop the col if it's in index, default False
- **time_field** –

Returns:: results basing on return_type.

classmethod record_data(*provider_index: int = 0*, *provider: Optional[str] = None*, *force_update=None*, *sleeping_time=None*, *exchanges=None*, *entity_id=None*, *entity_ids=None*, *code=None*, *codes=None*, *real_time=None*, *fix_duplicate_way=None*, *start_timestamp=None*, *end_timestamp=None*, *one_day_trading_minutes=None*, ***kwargs*)

record data by the arguments

Parameters::

- **entity_id** –
- **provider_index** –
- **provider** –
- **force_update** –
- **sleeping_time** –
- **exchanges** –
- **entity_ids** –
- **code** –
- **codes** –
- **real_time** –
- **fix_duplicate_way** –
- **start_timestamp** –
- **end_timestamp** –
- **one_day_trading_minutes** –
- **kwargs** –

Returns::

classmethod register_provider(*provider*)

register the provider to the schema defined by cls

Parameters:: **provider** –

classmethod register_recorder_cls(*provider*, *recorder_cls*)

register the recorder for the schema

Parameters::

- **provider** –
- **recorder_cls** –

timestamp

the meaning could be different for different case,most time it means 'happen time'

zvt.domain.fundamental.trading

Classes

BigDealTrading(**kwargs)

DragonAndTiger(**kwargs)

HolderTrading(**kwargs)

ManagerTrading(**kwargs)

MarginTrading(**kwargs)

BigDealTrading

class zvt.domain.fundamental.trading.BigDealTrading(**kwargs)

Bases: **Base**, **Mixin**

__init__(**kwargs)

A simple constructor that allows initialization from kwargs.

Sets attributes on the constructed instance using the names and values in `kwargs`.

Only keys that are present as attributes of the instance's class are allowed. These could be, for example, any mapped columns or relationships.

turnover

成交额

price

成交价

sell_broker

卖出营业部

buy_broker

买入营业部

compare_rate

折/溢价率

entity_id

entity id

classmethod get_providers() → List[str]

providers of the schema defined by cls

Returns:: providers

classmethod get_storages(*provider: Optional[str] = None*)

get the storages info

Parameters:: **provider** – provider
Returns:: storages

id

id

classmethod query_data(*provider_index: int = 0*, *ids: Optional[List[str]] = None*, *entity_ids: Optional[List[str]] = None*, *entity_id: Optional[str] = None*, *codes: Optional[List[str]] = None*, *code: Optional[str] = None*, *level: Optional[Union[IntervalLevel, str]] = None*, *provider: Optional[str] = None*, *columns: Optional[List] = None*, *col_label: Optional[dict] = None*, *return_type: str = 'df'*, *start_timestamp: Optional[Union[Timestamp, str]] = None*, *end_timestamp: Optional[Union[Timestamp, str]] = None*, *filters: Optional[List] = None*, *session: Optional[Session] = None*, *order=None*, *limit: Optional[int] = None*, *distinct=None*, *index: Optional[Union[str, list]] = None*, *drop_index_col=False*, *time_field: str = 'timestamp'*)

query data by the arguments

Parameters::

- **provider_index** –
- **data_schema** –
- **ids** –
- **entity_ids** –
- **entity_id** –
- **codes** –
- **code** –
- **level** –
- **provider** –
- **columns** –
- **col_label** – dict with key(column), value(label
- **return_type** – df, domain or dict. default is df
- **start_timestamp** –
- **end_timestamp** –
- **filters** –
- **session** –
- **order** –
- **limit** –
- **index** – index field name, str for single index, str list for multiple index
- **drop_index_col** – whether drop the col if it's in index, default False
- **time_field** –

Returns:: results basing on return_type.

classmethod record_data(*provider_index: int = 0*, *provider: Optional[str] = None*, *force_update=None*, *sleeping_time=None*, *exchanges=None*, *entity_id=None*, *entity_ids=None*, *code=None*, *codes=None*, *real_time=None*, *fix_duplicate_way=None*, *start_timestamp=None*, *end_timestamp=None*, *one_day_trading_minutes=None*, ***kwargs*)

record data by the arguments

Parameters::

- **entity_id** –
- **provider_index** –
- **provider** –
- **force_update** –
- **sleeping_time** –
- **exchanges** –
- **entity_ids** –
- **code** –
- **codes** –
- **real_time** –
- **fix_duplicate_way** –
- **start_timestamp** –
- **end_timestamp** –
- **one_day_trading_minutes** –
- **kwargs** –

Returns::

classmethod register_provider(*provider*)

register the provider to the schema defined by cls

Parameters:: **provider** –

classmethod register_recorder_cls(*provider*, *recorder_cls*)

register the recorder for the schema

Parameters::

- **provider** –
- **recorder_cls** –

timestamp

the meaning could be different for different case,most time it means 'happen time'

DragonAndTiger

class zvt.domain.fundamental.trading.DragonAndTiger(**kwargs)

Bases: **Base**, **Mixin**

__init__(**kwargs)

A simple constructor that allows initialization from kwargs.

Sets attributes on the constructed instance using the names and values in `kwargs`.

Only keys that are present as attributes of the instance's class are allowed. These could be, for example, any mapped columns or relationships.

reason

异动原因

turnover

成交额

change_pct

涨幅

net_in

净买入

dep1

买入营业部

dep_1

卖出营业部

entity_id

entity id

classmethod get_providers() → List[str]

providers of the schema defined by cls

Returns:: providers

classmethod get_storages(*provider: Optional[str] = None*)

get the storages info

Parameters:: **provider** – provider

Returns:: storages

id

id

classmethod query_data(*provider_index: int = 0*, *ids: Optional[List[str]] = None*, *entity_ids: Optional[List[str]] = None*, *entity_id: Optional[str] = None*, *codes: Optional[List[str]] = None*, *code: Optional[str] = None*, *level: Optional[Union[IntervalLevel, str]] = None*, *provider: Optional[str] = None*, *columns: Optional[List] = None*, *col_label: Optional[dict] = None*, *return_type: str = 'df'*, *start_timestamp: Optional[Union[Timestamp, str]] = None*, *end_timestamp: Optional[Union[Timestamp, str]] = None*, *filters: Optional[List] = None*, *session: Optional[Session] = None*, *order=None*, *limit: Optional[int] = None*, *distinct=None*, *index: Optional[Union[str, list]] = None*, *drop_index_col=False*, *time_field: str = 'timestamp'*)

query data by the arguments

Parameters::

- **provider_index** –
- **data_schema** –
- **ids** –
- **entity_ids** –
- **entity_id** –
- **codes** –
- **code** –
- **level** –
- **provider** –
- **columns** –
- **col_label** – dict with key(column), value(label
- **return_type** – df, domain or dict. default is df
- **start_timestamp** –
- **end_timestamp** –
- **filters** –
- **session** –
- **order** –
- **limit** –
- **index** – index field name, str for single index, str list for multiple index
- **drop_index_col** – whether drop the col if it's in index, default False
- **time_field** –

Returns:: results basing on return_type.

classmethod record_data(*provider_index: int = 0*, *provider: Optional[str] = None*, *force_update=None*, *sleeping_time=None*, *exchanges=None*, *entity_id=None*, *entity_ids=None*, *code=None*, *codes=None*, *real_time=None*, *fix_duplicate_way=None*, *start_timestamp=None*, *end_timestamp=None*, *one_day_trading_minutes=None*, ***kwargs*)

record data by the arguments

Parameters::

- **entity_id** –
- **provider_index** –
- **provider** –
- **force_update** –
- **sleeping_time** –
- **exchanges** –
- **entity_ids** –
- **code** –
- **codes** –
- **real_time** –
- **fix_duplicate_way** –
- **start_timestamp** –
- **end_timestamp** –
- **one_day_trading_minutes** –
- **kwargs** –

Returns::

classmethod register_provider(*provider*)

register the provider to the schema defined by cls

Parameters:: **provider** –

classmethod register_recorder_cls(*provider*, *recorder_cls*)

register the recorder for the schema

Parameters::

- **provider** –
- **recorder_cls** –

timestamp

the meaning could be different for different case,most time it means 'happen time'

HolderTrading

class zvt.domain.fundamental.trading.HolderTrading(**kwargs*)

Bases: **Base**, **Mixin**

__init__(**kwargs*)

A simple constructor that allows initialization from kwargs.

Sets attributes on the constructed instance using the names and values in `kwargs`.

Only keys that are present as attributes of the instance's class are allowed. These could be, for example, any mapped columns or relationships.

holder_name

股东名称

volume

变动数量

change_pct

变动比例

holding_pct

变动后持股比例

entity_id

entity id

classmethod get_providers() → List[str]

providers of the schema defined by cls

Returns:: providers

classmethod get_storages(*provider: Optional[str] = None*)

get the storages info

Parameters:: **provider** – provider

Returns:: storages

id

id

classmethod query_data(*provider_index: int = 0*, *ids: Optional[List[str]] = None*, *entity_ids: Optional[List[str]] = None*, *entity_id: Optional[str] = None*, *codes: Optional[List[str]] = None*, *code: Optional[str] = None*, *level: Optional[Union[IntervalLevel, str]] = None*, *provider: Optional[str] = None*, *columns: Optional[List] = None*, *col_label: Optional[dict] = None*, *return_type: str = 'df'*, *start_timestamp: Optional[Union[Timestamp, str]] = None*, *end_timestamp: Optional[Union[Timestamp, str]] = None*, *filters: Optional[List] = None*, *session: Optional[Session] = None*, *order=None*, *limit: Optional[int] = None*, *distinct=None*, *index: Optional[Union[str, list]] = None*, *drop_index_col=False*, *time_field: str = 'timestamp'*)

query data by the arguments

Parameters::

- **provider_index** –
- **data_schema** –
- **ids** –
- **entity_ids** –
- **entity_id** –
- **codes** –
- **code** –
- **level** –
- **provider** –
- **columns** –
- **col_label** – dict with key(column), value(label
- **return_type** – df, domain or dict. default is df
- **start_timestamp** –
- **end_timestamp** –
- **filters** –
- **session** –
- **order** –
- **limit** –
- **index** – index field name, str for single index, str list for multiple index
- **drop_index_col** – whether drop the col if it's in index, default False
- **time_field** –

Returns:: results basing on return_type.

classmethod record_data(*provider_index: int = 0*, *provider: Optional[str] = None*, *force_update=None*, *sleeping_time=None*, *exchanges=None*, *entity_id=None*, *entity_ids=None*, *code=None*, *codes=None*, *real_time=None*, *fix_duplicate_way=None*, *start_timestamp=None*, *end_timestamp=None*, *one_day_trading_minutes=None*, ***kwargs*)

record data by the arguments

Parameters::

- **entity_id** –
- **provider_index** –
- **provider** –
- **force_update** –
- **sleeping_time** –
- **exchanges** –
- **entity_ids** –
- **code** –
- **codes** –
- **real_time** –
- **fix_duplicate_way** –
- **start_timestamp** –
- **end_timestamp** –
- **one_day_trading_minutes** –
- **kwargs** –

Returns::

classmethod register_provider(*provider*)

register the provider to the schema defined by cls

Parameters:: **provider** –

classmethod register_recorder_cls(*provider*, *recorder_cls*)

register the recorder for the schema

Parameters::

- **provider** –
- **recorder_cls** –

timestamp

the meaning could be different for different case,most time it means 'happen time'

ManagerTrading

class zvt.domain.fundamental.trading.ManagerTrading(**kwargs)

Bases: **Base**, **Mixin**

__init__(**kwargs)

A simple constructor that allows initialization from kwargs.

Sets attributes on the constructed instance using the names and values in `kwargs`.

Only keys that are present as attributes of the instance's class are allowed. These could be, for example, any mapped columns or relationships.

trading_person

变动人

volume

变动数量

price

交易均价

holding

结存股票

trading_way

交易方式

manager

董监高管

manager_position

高管职位

relationship_with_manager

与高管关系

entity_id

entity id

classmethod get_providers() → List[str]

providers of the schema defined by cls

Returns:: providers

classmethod get_storages(*provider: Optional[str] = None*)

get the storages info

Parameters:: **provider** – provider

Returns:: storages

id

id

classmethod query_data(*provider_index: int = 0*, *ids: Optional[List[str]] = None*, *entity_ids: Optional[List[str]] = None*, *entity_id: Optional[str] = None*, *codes: Optional[List[str]] = None*, *code: Optional[str] = None*, *level: Optional[Union[IntervalLevel, str]] = None*, *provider: Optional[str] = None*, *columns: Optional[List] = None*, *col_label: Optional[dict] = None*, *return_type: str = 'df'*, *start_timestamp: Optional[Union[Timestamp, str]] = None*, *end_timestamp: Optional[Union[Timestamp, str]] = None*, *filters: Optional[List] = None*, *session: Optional[Session] = None*, *order=None*, *limit: Optional[int] = None*, *distinct=None*, *index: Optional[Union[str, list]] = None*, *drop_index_col=False*, *time_field: str = 'timestamp'*)

query data by the arguments

Parameters::

- **provider_index** –
- **data_schema** –
- **ids** –
- **entity_ids** –
- **entity_id** –
- **codes** –
- **code** –
- **level** –
- **provider** –
- **columns** –
- **col_label** – dict with key(column), value(label
- **return_type** – df, domain or dict. default is df
- **start_timestamp** –
- **end_timestamp** –
- **filters** –
- **session** –
- **order** –
- **limit** –
- **index** – index field name, str for single index, str list for multiple index
- **drop_index_col** – whether drop the col if it's in index, default False
- **time_field** –

Returns:: results basing on return_type.

classmethod record_data(*provider_index: int = 0*, *provider: Optional[str] = None*, *force_update=None*, *sleeping_time=None*, *exchanges=None*, *entity_id=None*, *entity_ids=None*, *code=None*, *codes=None*, *real_time=None*, *fix_duplicate_way=None*, *start_timestamp=None*, *end_timestamp=None*, *one_day_trading_minutes=None*, ***kwargs*)

record data by the arguments

Parameters::

- **entity_id** –
- **provider_index** –
- **provider** –
- **force_update** –
- **sleeping_time** –
- **exchanges** –
- **entity_ids** –
- **code** –
- **codes** –
- **real_time** –
- **fix_duplicate_way** –
- **start_timestamp** –
- **end_timestamp** –
- **one_day_trading_minutes** –
- **kwargs** –

Returns::

classmethod register_provider(*provider*)

register the provider to the schema defined by cls

Parameters:: **provider** –

classmethod register_recorder_cls(*provider*, *recorder_cls*)

register the recorder for the schema

Parameters::

- **provider** –
- **recorder_cls** –

timestamp

the meaning could be different for different case,most time it means 'happen time'

MarginTrading

class zvt.domain.fundamental.trading.MarginTrading(**kwargs)

Bases: **Base**, **Mixin**

__init__(**kwargs)

A simple constructor that allows initialization from kwargs.

Sets attributes on the constructed instance using the names and values in `kwargs`.

Only keys that are present as attributes of the instance's class are allowed. These could be, for example, any mapped columns or relationships.

fin_value

融资余额(元)

fin_buy_value

融资买入额（元）

fin_refund_value

融资偿还额（元）

sec_value

融券余量（股）

sec_sell_value

融券卖出量（股）

sec_refund_value

融券偿还量（股）

fin_sec_value

融资融券余额（元）

entity_id

entity id

classmethod get_providers() → List[str]

providers of the schema defined by cls

Returns:: providers

classmethod get_storages(*provider: Optional[str] = None*)

get the storages info

Parameters:: **provider** – provider

Returns:: storages

id

id

classmethod query_data(*provider_index: int = 0*, *ids: Optional[List[str]] = None*, *entity_ids: Optional[List[str]] = None*, *entity_id: Optional[str] = None*, *codes: Optional[List[str]] = None*, *code: Optional[str] = None*, *level: Optional[Union[IntervalLevel, str]] = None*, *provider: Optional[str] = None*, *columns: Optional[List] = None*, *col_label: Optional[dict] = None*, *return_type: str = 'df'*, *start_timestamp: Optional[Union[Timestamp, str]] = None*, *end_timestamp: Optional[Union[Timestamp, str]] = None*, *filters: Optional[List] = None*, *session: Optional[Session] = None*, *order=None*, *limit: Optional[int] = None*, *distinct=None*, *index: Optional[Union[str, list]] = None*, *drop_index_col=False*, *time_field: str = 'timestamp'*)

query data by the arguments

Parameters::

- **provider_index** –
- **data_schema** –
- **ids** –
- **entity_ids** –
- **entity_id** –
- **codes** –
- **code** –
- **level** –
- **provider** –
- **columns** –
- **col_label** – dict with key(column), value(label
- **return_type** – df, domain or dict. default is df
- **start_timestamp** –
- **end_timestamp** –
- **filters** –
- **session** –
- **order** –
- **limit** –
- **index** – index field name, str for single index, str list for multiple index
- **drop_index_col** – whether drop the col if it's in index, default False
- **time_field** –

Returns:: results basing on return_type.

classmethod record_data(*provider_index: int = 0*, *provider: Optional[str] = None*, *force_update=None*, *sleeping_time=None*, *exchanges=None*, *entity_id=None*, *entity_ids=None*, *code=None*, *codes=None*, *real_time=None*, *fix_duplicate_way=None*, *start_timestamp=None*, *end_timestamp=None*, *one_day_trading_minutes=None*, ***kwargs*)

record data by the arguments

Parameters::

- **entity_id** –
- **provider_index** –
- **provider** –
- **force_update** –
- **sleeping_time** –
- **exchanges** –
- **entity_ids** –
- **code** –
- **codes** –
- **real_time** –
- **fix_duplicate_way** –
- **start_timestamp** –
- **end_timestamp** –
- **one_day_trading_minutes** –
- **kwargs** –

Returns::

classmethod register_provider(*provider*)

register the provider to the schema defined by cls

Parameters:: **provider** –

classmethod register_recorder_cls(*provider*, *recorder_cls*)

register the recorder for the schema

Parameters::

- **provider** –
- **recorder_cls** –

timestamp

the meaning could be different for different case,most time it means 'happen time'

zvt.domain.fundamental.valuation

Classes

`EtfValuation`(**kwargs)

`StockValuation`(**kwargs)

EtfValuation

class zvt.domain.fundamental.valuation.EtfValuation(**kwargs*)

Bases: **Base**, **Mixin**

__init__(**kwargs*)

A simple constructor that allows initialization from kwargs.

Sets attributes on the constructed instance using the names and values in `kwargs`.

Only keys that are present as attributes of the instance's class are allowed. These could be, for example, any mapped columns or relationships.

pe

静态pe

pe1

加权

pe_ttm

动态pe

pe_ttm1

加权

pb

市净率

pb1

加权

ps

市销率

ps1

加权

pcf

市现率

pcf1

加权

entity_id

entity id

classmethod get_providers() → List[str]

providers of the schema defined by cls

Returns:: providers

classmethod get_storages(*provider: Optional[str] = None*)

get the storages info

Parameters:: **provider** – provider

Returns:: storages

id

id

classmethod query_data(*provider_index: int = 0*, *ids: Optional[List[str]] = None*, *entity_ids: Optional[List[str]] = None*, *entity_id: Optional[str] = None*, *codes: Optional[List[str]] = None*, *code: Optional[str] = None*, *level: Optional[Union[IntervalLevel, str]] = None*, *provider: Optional[str] = None*, *columns: Optional[List] = None*, *col_label: Optional[dict] = None*, *return_type: str = 'df'*, *start_timestamp: Optional[Union[Timestamp, str]] = None*, *end_timestamp: Optional[Union[Timestamp, str]] = None*, *filters: Optional[List] = None*, *session: Optional[Session] = None*, *order=None*, *limit: Optional[int] =*

None, *distinct=None*, *index: Optional[Union[str, list]] = None*, *drop_index_col=False*, *time_field: str = 'timestamp'*)

query data by the arguments

Parameters::

- **provider_index** –
- **data_schema** –
- **ids** –
- **entity_ids** –
- **entity_id** –
- **codes** –
- **code** –
- **level** –
- **provider** –
- **columns** –
- **col_label** – dict with key(column), value(label
- **return_type** – df, domain or dict. default is df
- **start_timestamp** –
- **end_timestamp** –
- **filters** –
- **session** –
- **order** –
- **limit** –
- **index** – index field name, str for single index, str list for multiple index
- **drop_index_col** – whether drop the col if it's in index, default False
- **time_field** –

Returns:: results basing on return_type.

classmethod record_data(*provider_index: int = 0*, *provider: Optional[str] = None*, *force_update=None*, *sleeping_time=None*, *exchanges=None*, *entity_id=None*, *entity_ids=None*, *code=None*, *codes=None*, *real_time=None*, *fix_duplicate_way=None*, *start_timestamp=None*, *end_timestamp=None*, *one_day_trading_minutes=None*, ***kwargs*)

record data by the arguments

Parameters::

- **entity_id** –
- **provider_index** –
- **provider** –
- **force_update** –
- **sleeping_time** –
- **exchanges** –
- **entity_ids** –
- **code** –
- **codes** –
- **real_time** –
- **fix_duplicate_way** –
- **start_timestamp** –
- **end_timestamp** –
- **one_day_trading_minutes** –
- **kwargs** –

Returns::

classmethod register_provider(*provider*)

register the provider to the schema defined by cls

Parameters:: **provider** –

classmethod register_recorder_cls(*provider*, *recorder_cls*)

register the recorder for the schema

Parameters::

- **provider** –
- **recorder_cls** –

timestamp

the meaning could be different for different case,most time it means 'happen time'

StockValuation

class zvt.domain.fundamental.valuation.StockValuation(**kwargs)

Bases: **Base**, **Mixin**

__init__(**kwargs)

A simple constructor that allows initialization from kwargs.

Sets attributes on the constructed instance using the names and values in `kwargs`.

Only keys that are present as attributes of the instance's class are allowed. These could be, for example, any mapped columns or relationships.

capitalization

总股本(股)

circulating_cap

公司已发行的普通股股份总数(包含A股，B股和H股的总股本)

market_cap

市值

circulating_market_cap

流通市值

turnover_ratio

换手率

pe

静态pe

pe_ttm

动态pe

pb

市净率

ps

市销率

pcf

市现率

entity_id

entity id

classmethod get_providers() → List[str]

providers of the schema defined by cls

Returns:: providers

classmethod get_storages(*provider: Optional[str] = None*)

get the storages info

Parameters:: **provider** – provider

Returns:: storages

id

id

classmethod query_data(*provider_index: int = 0*, *ids: Optional[List[str]] = None*, *entity_ids: Optional[List[str]] = None*, *entity_id: Optional[str] = None*, *codes: Optional[List[str]] = None*, *code: Optional[str] = None*, *level: Optional[Union[IntervalLevel, str]] = None*, *provider: Optional[str] = None*, *columns: Optional[List] = None*, *col_label: Optional[dict] = None*, *return_type: str = 'df'*, *start_timestamp: Optional[Union[Timestamp, str]] = None*, *end_timestamp: Optional[Union[Timestamp, str]] = None*, *filters: Optional[List] = None*, *session: Optional[Session] = None*, *order=None*, *limit: Optional[int] =*

None, *distinct=None*, *index: Optional[Union[str, list]] = None*, *drop_index_col=False*, *time_field: str = 'timestamp'*)

query data by the arguments

Parameters::

- **provider_index** –
- **data_schema** –
- **ids** –
- **entity_ids** –
- **entity_id** –
- **codes** –
- **code** –
- **level** –
- **provider** –
- **columns** –
- **col_label** – dict with key(column), value(label
- **return_type** – df, domain or dict. default is df
- **start_timestamp** –
- **end_timestamp** –
- **filters** –
- **session** –
- **order** –
- **limit** –
- **index** – index field name, str for single index, str list for multiple index
- **drop_index_col** – whether drop the col if it's in index, default False
- **time_field** –

Returns:: results basing on return_type.

classmethod record_data(*provider_index: int = 0*, *provider: Optional[str] = None*, *force_update=None*, *sleeping_time=None*, *exchanges=None*, *entity_id=None*, *entity_ids=None*, *code=None*, *codes=None*, *real_time=None*, *fix_duplicate_way=None*, *start_timestamp=None*, *end_timestamp=None*, *one_day_trading_minutes=None*, ***kwargs*)

record data by the arguments

Parameters::

- **entity_id** –
- **provider_index** –
- **provider** –
- **force_update** –
- **sleeping_time** –
- **exchanges** –
- **entity_ids** –
- **code** –
- **codes** –
- **real_time** –
- **fix_duplicate_way** –
- **start_timestamp** –
- **end_timestamp** –
- **one_day_trading_minutes** –
- **kwargs** –

Returns::

classmethod register_provider(*provider*)

register the provider to the schema defined by cls

Parameters:: **provider** –

classmethod register_recorder_cls(*provider*, *recorder_cls*)

register the recorder for the schema

Parameters::

- **provider** –
- **recorder_cls** –

timestamp

the meaning could be different for different case,most time it means 'happen time'

zvt.domain.macro

Modules

zvt.domain.macro.macro

---

zvt.domain.macro.monetary

---

zvt.domain.macro.macro

Classes

Economy(**kwargs)

Economy

class zvt.domain.macro.macro.Economy(***kwargs*)

Bases: **`Base`**, **`Mixin`**

__init__(***kwargs*)

A simple constructor that allows initialization from kwargs.

Sets attributes on the constructed instance using the names and values in `kwargs`.

Only keys that are present as attributes of the instance's class are allowed. These could be, for example, any mapped columns or relationships.

entity_id

entity id

classmethod get_providers() → List[str]

providers of the schema defined by cls

Returns:: providers

classmethod get_storages(*provider: Optional[str] = None*)

get the storages info

Parameters:: **provider** – provider
Returns:: storages

id

id

classmethod query_data(*provider_index: int = 0*, *ids: Optional[List[str]] = None*, *entity_ids: Optional[List[str]] = None*, *entity_id: Optional[str] = None*, *codes: Optional[List[str]] = None*, *code: Optional[str] = None*, *level: Optional[Union[IntervalLevel, str]] = None*, *provider:*

Optional[str] = None, *columns: Optional[List] = None*, *col_label: Optional[dict] = None*, *return_type: str = 'df'*, *start_timestamp: Optional[Union[Timestamp, str]] = None*, *end_timestamp: Optional[Union[Timestamp, str]] = None*, *filters: Optional[List] = None*, *session: Optional[Session] = None*, *order=None*, *limit: Optional[int] = None*, *distinct=None*, *index: Optional[Union[str, list]] = None*, *drop_index_col=False*, *time_field: str = 'timestamp'*)

query data by the arguments

Parameters::

- **provider_index** –
- **data_schema** –
- **ids** –
- **entity_ids** –
- **entity_id** –
- **codes** –
- **code** –
- **level** –
- **provider** –
- **columns** –
- **col_label** – dict with key(column), value(label
- **return_type** – df, domain or dict. default is df
- **start_timestamp** –
- **end_timestamp** –
- **filters** –
- **session** –
- **order** –
- **limit** –
- **index** – index field name, str for single index, str list for multiple index
- **drop_index_col** – whether drop the col if it's in index, default False
- **time_field** –

Returns:: results basing on return_type.

classmethod record_data(*provider_index: int = 0*, *provider: Optional[str] = None*, *force_update=None*, *sleeping_time=None*, *exchanges=None*, *entity_id=None*, *entity_ids=None*, *code=None*, *codes=None*,

real_time=None, *fix_duplicate_way=None*, *start_timestamp=None*, *end_timestamp=None*, *one_day_trading_minutes=None*, ***kwargs*)

record data by the arguments

Parameters::

- **entity_id** –
- **provider_index** –
- **provider** –
- **force_update** –
- **sleeping_time** –
- **exchanges** –
- **entity_ids** –
- **code** –
- **codes** –
- **real_time** –
- **fix_duplicate_way** –
- **start_timestamp** –
- **end_timestamp** –
- **one_day_trading_minutes** –
- **kwargs** –

Returns::

classmethod register_provider(*provider*)

register the provider to the schema defined by cls

Parameters:: **provider** –

classmethod register_recorder_cls(*provider*, *recorder_cls*)

register the recorder for the schema

Parameters::

- **provider** –
- **recorder_cls** –

timestamp

the meaning could be different for different case,most time it means 'happen time'

zvt.domain.macro.monetary

Classes

`TreasuryYield`(**kwargs)

TreasuryYield

class zvt.domain.macro.monetary.TreasuryYield(***kwargs*)

Bases: **`Base`**, **`Mixin`**

__init__(***kwargs*)

A simple constructor that allows initialization from kwargs.

Sets attributes on the constructed instance using the names and values in `kwargs`.

Only keys that are present as attributes of the instance's class are allowed. These could be, for example, any mapped columns or relationships.

entity_id

entity id

classmethod get_providers() → List[str]

providers of the schema defined by cls

Returns:: providers

classmethod get_storages(*provider: Optional[str] = None*)

get the storages info

Parameters:: **provider** – provider
Returns:: storages

id

id

classmethod query_data(*provider_index: int = 0*, *ids: Optional[List[str]] = None*, *entity_ids: Optional[List[str]] = None*, *entity_id: Optional[str] = None*, *codes: Optional[List[str]] = None*, *code: Optional[str] = None*, *level: Optional[Union[IntervalLevel, str]] = None*, *provider:*

Optional[str] = None, *columns: Optional[List] = None*, *col_label: Optional[dict] = None*, *return_type: str = 'df'*, *start_timestamp: Optional[Union[Timestamp, str]] = None*, *end_timestamp: Optional[Union[Timestamp, str]] = None*, *filters: Optional[List] = None*, *session: Optional[Session] = None*, *order=None*, *limit: Optional[int] = None*, *distinct=None*, *index: Optional[Union[str, list]] = None*, *drop_index_col=False*, *time_field: str = 'timestamp'*)

query data by the arguments

Parameters::

- **provider_index** –
- **data_schema** –
- **ids** –
- **entity_ids** –
- **entity_id** –
- **codes** –
- **code** –
- **level** –
- **provider** –
- **columns** –
- **col_label** – dict with key(column), value(label
- **return_type** – df, domain or dict. default is df
- **start_timestamp** –
- **end_timestamp** –
- **filters** –
- **session** –
- **order** –
- **limit** –
- **index** – index field name, str for single index, str list for multiple index
- **drop_index_col** – whether drop the col if it's in index, default False
- **time_field** –

Returns:: results basing on return_type.

classmethod record_data(*provider_index: int = 0*, *provider: Optional[str] = None*, *force_update=None*, *sleeping_time=None*, *exchanges=None*, *entity_id=None*, *entity_ids=None*, *code=None*, *codes=None*,

real_time=None, *fix_duplicate_way=None*, *start_timestamp=None*, *end_timestamp=None*, *one_day_trading_minutes=None*, ***kwargs*)

record data by the arguments

Parameters::

- **entity_id** –
- **provider_index** –
- **provider** –
- **force_update** –
- **sleeping_time** –
- **exchanges** –
- **entity_ids** –
- **code** –
- **codes** –
- **real_time** –
- **fix_duplicate_way** –
- **start_timestamp** –
- **end_timestamp** –
- **one_day_trading_minutes** –
- **kwargs** –

Returns::

classmethod register_provider(*provider*)

register the provider to the schema defined by cls

Parameters:: **provider** –

classmethod register_recorder_cls(*provider*, *recorder_cls*)

register the recorder for the schema

Parameters::

- **provider** –
- **recorder_cls** –

timestamp

the meaning could be different for different case,most time it means 'happen time'

zvt.domain.meta

Modules

zvt.domain.meta.block_meta

---

zvt.domain.meta.country_meta

---

zvt.domain.meta.currency_meta

---

zvt.domain.meta.etf_meta

---

zvt.domain.meta.fund_meta

---

zvt.domain.meta.future_meta

---

zvt.domain.meta.index_meta

---

zvt.domain.meta.indexus_meta

---

zvt.domain.meta.stock_meta

---

zvt.domain.meta.stockhk_meta

---

zvt.domain.meta.stockus_meta

zvt.domain.meta.block_meta

Classes

`Block`(**kwargs)

`BlockStock`(**kwargs)

Block

class zvt.domain.meta.block_meta.Block(**kwargs)

Bases: **Base**, **Portfolio**

__init__(**kwargs)

A simple constructor that allows initialization from kwargs.

Sets attributes on the constructed instance using the names and values in `kwargs`.

Only keys that are present as attributes of the instance's class are allowed. These could be, for example, any mapped columns or relationships.

category

板块类型，行业(industry),概念(concept)

code

编码

classmethod could_short()

whether could be shorted

Returns::

end_date

退市日

entity_id

entity id

entity_type

标的类型

exchange

所属交易所

classmethod get_interval_timestamps(*start_date*, *end_date*, *level: IntervalLevel*)

generate the timestamps for the level

Parameters::
- **start_date** –
- **end_date** –
- **level** –

classmethod get_providers() → List[str]

providers of the schema defined by cls

Returns:: providers

classmethod get_stocks(*code=None*, *codes=None*, *ids=None*, *timestamp=Timestamp('2024-08-28 22:08:09.719543')*, *provider=None*)

the publishing policy of portfolio positions is different for different types, overwrite this function for get the holding stocks in specific date

Parameters::
- **code** – portfolio(etf/block/index…) code
- **codes** – portfolio(etf/block/index…) codes
- **ids** – portfolio(etf/block/index…) ids
- **timestamp** – the date of the holding stocks
- **provider** – the data provider

Returns::

classmethod get_storages(*provider: Optional[str] = None*)

get the storages info

Parameters:: **provider** – provider

Returns:: storages

classmethod get_trading_dates(*start_date=None*, *end_date=None*)

overwrite it to get the trading dates of the entity

Parameters::
- **start_date** –
- **end_date** –

Returns:: list of dates

classmethod get_trading_intervals()

overwrite it to get the trading intervals of the entity

Returns:: list of time intervals, in format [(start,end)]

classmethod get_trading_t()

0 means t+0 1 means t+1

Returns::

id

id

classmethod is_finished_kdata_timestamp(*timestamp: Timestamp*, *level: IntervalLevel*)

Parameters::
- **timestamp** (*pd.Timestamp*) – the timestamp could be recorded in kdata of the level
- **level** (*zvt.domain.common.IntervalLevel*) –

Returns::

Return type:: bool

list_date

上市日

name

名字

classmethod query_data(*provider_index: int = 0*, *ids: Optional[List[str]] = None*, *entity_ids: Optional[List[str]] = None*, *entity_id: Optional[str] = None*, *codes: Optional[List[str]] = None*, *code: Optional[str] = None*, *level: Optional[Union[IntervalLevel, str]] = None*, *provider: Optional[str] = None*, *columns: Optional[List] = None*, *col_label: Optional[dict] = None*, *return_type: str = 'df'*, *start_timestamp: Optional[Union[Timestamp, str]] = None*, *end_timestamp:*

Optional[Union[Timestamp, str]] = None, *filters: Optional[List] = None*, *session: Optional[Session] = None*, *order=None*, *limit: Optional[int] = None*, *distinct=None*, *index: Optional[Union[str, list]] = None*, *drop_index_col=False*, *time_field: str = 'timestamp'*)

query data by the arguments

Parameters::

- **provider_index** –
- **data_schema** –
- **ids** –
- **entity_ids** –
- **entity_id** –
- **codes** –
- **code** –
- **level** –
- **provider** –
- **columns** –
- **col_label** – dict with key(column), value(label
- **return_type** – df, domain or dict. default is df
- **start_timestamp** –
- **end_timestamp** –
- **filters** –
- **session** –
- **order** –
- **limit** –
- **index** – index field name, str for single index, str list for multiple index
- **drop_index_col** – whether drop the col if it's in index, default False
- **time_field** –

Returns:: results basing on return_type.

classmethod record_data(*provider_index: int = 0*, *provider: Optional[str] = None*, *force_update=None*, *sleeping_time=None*, *exchanges=None*, *entity_id=None*, *entity_ids=None*, *code=None*, *codes=None*, *real_time=None*, *fix_duplicate_way=None*, *start_timestamp=None*, *end_timestamp=None*, *one_day_trading_minutes=None*, ***kwargs*)

record data by the arguments

Parameters::

- **entity_id** –
- **provider_index** –
- **provider** –
- **force_update** –
- **sleeping_time** –
- **exchanges** –
- **entity_ids** –
- **code** –
- **codes** –
- **real_time** –
- **fix_duplicate_way** –
- **start_timestamp** –
- **end_timestamp** –
- **one_day_trading_minutes** –
- **kwargs** –

Returns::

classmethod register_provider(*provider*)

register the provider to the schema defined by cls

Parameters:: **provider** –

classmethod register_recorder_cls(*provider*, *recorder_cls*)

register the recorder for the schema

Parameters::

- **provider** –
- **recorder_cls** –

timestamp

the meaning could be different for different case,most time it means 'happen time'

BlockStock

class zvt.domain.meta.block_meta.BlockStock(***kwargs*)

Bases: **Base**, **PortfolioStock**

__init__(***kwargs*)

A simple constructor that allows initialization from kwargs.

Sets attributes on the constructed instance using the names and values in `kwargs`.

Only keys that are present as attributes of the instance's class are allowed. These could be, for example, any mapped columns or relationships.

code

portfolio编码

entity_id

entity id

entity_type

portfolio标的类型

exchange

portfolio所属交易所

classmethod get_providers() → List[str]

providers of the schema defined by cls

Returns:: providers

classmethod get_storages(*provider: Optional[str] = None*)

get the storages info

Parameters:: **provider** – provider
Returns:: storages

id

id

name

portfolio名字

classmethod query_data(*provider_index: int = 0*, *ids: Optional[List[str]] = None*, *entity_ids: Optional[List[str]] = None*, *entity_id: Optional[str] = None*, *codes: Optional[List[str]] = None*, *code: Optional[str] = None*, *level: Optional[Union[IntervalLevel, str]] = None*, *provider: Optional[str] = None*, *columns: Optional[List] = None*, *col_label: Optional[dict] = None*, *return_type: str = 'df'*, *start_timestamp: Optional[Union[Timestamp, str]] = None*, *end_timestamp: Optional[Union[Timestamp, str]] = None*, *filters: Optional[List] = None*, *session: Optional[Session] = None*, *order=None*, *limit: Optional[int] = None*, *distinct=None*, *index: Optional[Union[str, list]] = None*, *drop_index_col=False*, *time_field: str = 'timestamp'*)

query data by the arguments

Parameters::

- **provider_index** –
- **data_schema** –
- **ids** –
- **entity_ids** –
- **entity_id** –
- **codes** –
- **code** –
- **level** –
- **provider** –
- **columns** –
- **col_label** – dict with key(column), value(label
- **return_type** – df, domain or dict. default is df
- **start_timestamp** –
- **end_timestamp** –
- **filters** –
- **session** –
- **order** –
- **limit** –
- **index** – index field name, str for single index, str list for multiple index
- **drop_index_col** – whether drop the col if it's in index, default False
- **time_field** –

Returns:: results basing on return_type.

classmethod record_data(*provider_index: int = 0*, *provider: Optional[str] = None*, *force_update=None*, *sleeping_time=None*, *exchanges=None*, *entity_id=None*, *entity_ids=None*, *code=None*, *codes=None*, *real_time=None*, *fix_duplicate_way=None*, *start_timestamp=None*, *end_timestamp=None*, *one_day_trading_minutes=None*, ***kwargs*)

record data by the arguments

Parameters::

- **entity_id** –
- **provider_index** –
- **provider** –
- **force_update** –
- **sleeping_time** –
- **exchanges** –
- **entity_ids** –
- **code** –
- **codes** –
- **real_time** –
- **fix_duplicate_way** –
- **start_timestamp** –
- **end_timestamp** –
- **one_day_trading_minutes** –
- **kwargs** –

Returns::

classmethod register_provider(*provider*)

register the provider to the schema defined by cls

Parameters:: **provider** –

classmethod register_recorder_cls(*provider*, *recorder_cls*)

register the recorder for the schema

Parameters::

- **provider** –
- **recorder_cls** –

stock_code

stock_id

stock_name

timestamp

the meaning could be different for different case,most time it means 'happen time'

zvt.domain.meta.country_meta

Classes

`Country`(**kwargs)

Country

class zvt.domain.meta.country_meta.Country(**kwargs)

Bases: **Base**, **TradableEntity**

__init__(**kwargs)

A simple constructor that allows initialization from kwargs.

Sets attributes on the constructed instance using the names and values in `kwargs`.

Only keys that are present as attributes of the instance's class are allowed. These could be, for example, any mapped columns or relationships.

region

区域 region

capital_city

首都 capital city

income_level

收入水平 income level

lending_type

贷款类型 lending type

longitude

经度 longitude

latitude

纬度 latitude

code

编码

classmethod could_short()

whether could be shorted

Returns::

end_date

退市日

entity_id

entity id

entity_type

标的类型

exchange

所属交易所

classmethod get_interval_timestamps(*start_date*, *end_date*, *level: IntervalLevel*)

generate the timestamps for the level

Parameters::

- **start_date** –
- **end_date** –
- **level** –

classmethod get_providers() → List[str]

providers of the schema defined by cls

Returns:: providers

classmethod get_storages(*provider: Optional[str] = None*)

get the storages info

Parameters:: **provider** – provider

Returns:: storages

classmethod get_trading_dates(*start_date=None*, *end_date=None*)

overwrite it to get the trading dates of the entity

Parameters::
- **start_date** –
- **end_date** –

Returns:: list of dates

classmethod get_trading_intervals()

overwrite it to get the trading intervals of the entity

Returns:: list of time intervals, in format [(start,end)]

classmethod get_trading_t()

0 means t+0 1 means t+1

Returns::

id

id

classmethod is_finished_kdata_timestamp(*timestamp: Timestamp*, *level: IntervalLevel*)

Parameters::
- **timestamp** (*pd.Timestamp*) – the timestamp could be recorded in kdata of the level
- **level** (*zvt.domain.common.IntervalLevel*) –

Returns::

Return type:: bool

list_date

上市日

name

名字

classmethod query_data(*provider_index: int = 0*, *ids: Optional[List[str]] = None*, *entity_ids: Optional[List[str]] = None*, *entity_id: Optional[str] = None*, *codes: Optional[List[str]] = None*, *code: Optional[str] = None*, *level: Optional[Union[IntervalLevel, str]] = None*, *provider:*

Optional[str] = None, *columns: Optional[List] = None*, *col_label: Optional[dict] = None*, *return_type: str = 'df'*, *start_timestamp: Optional[Union[Timestamp, str]] = None*, *end_timestamp: Optional[Union[Timestamp, str]] = None*, *filters: Optional[List] = None*, *session: Optional[Session] = None*, *order=None*, *limit: Optional[int] = None*, *distinct=None*, *index: Optional[Union[str, list]] = None*, *drop_index_col=False*, *time_field: str = 'timestamp'*)

query data by the arguments

Parameters::

- **provider_index** –
- **data_schema** –
- **ids** –
- **entity_ids** –
- **entity_id** –
- **codes** –
- **code** –
- **level** –
- **provider** –
- **columns** –
- **col_label** – dict with key(column), value(label
- **return_type** – df, domain or dict. default is df
- **start_timestamp** –
- **end_timestamp** –
- **filters** –
- **session** –
- **order** –
- **limit** –
- **index** – index field name, str for single index, str list for multiple index
- **drop_index_col** – whether drop the col if it's in index, default False
- **time_field** –

Returns:: results basing on return_type.

classmethod record_data(*provider_index: int = 0*, *provider: Optional[str] = None*, *force_update=None*, *sleeping_time=None*, *exchanges=None*, *entity_id=None*, *entity_ids=None*, *code=None*, *codes=None*,

real_time=None, *fix_duplicate_way=None*, *start_timestamp=None*, *end_timestamp=None*, *one_day_trading_minutes=None*, ***kwargs*)

record data by the arguments

Parameters::

- **entity_id** –
- **provider_index** –
- **provider** –
- **force_update** –
- **sleeping_time** –
- **exchanges** –
- **entity_ids** –
- **code** –
- **codes** –
- **real_time** –
- **fix_duplicate_way** –
- **start_timestamp** –
- **end_timestamp** –
- **one_day_trading_minutes** –
- **kwargs** –

Returns::

classmethod register_provider(*provider*)

register the provider to the schema defined by cls

Parameters:: **provider** –

classmethod register_recorder_cls(*provider*, *recorder_cls*)

register the recorder for the schema

Parameters::

- **provider** –
- **recorder_cls** –

timestamp

the meaning could be different for different case,most time it means 'happen time'

zvt.domain.meta.currency_meta

Classes

`Currency`(**kwargs)

Currency

class zvt.domain.meta.currency_meta.Currency(**kwargs*)

Bases: **Base**, **TradableEntity**

__init__(**kwargs*)

A simple constructor that allows initialization from kwargs.

Sets attributes on the constructed instance using the names and values in `kwargs`.

Only keys that are present as attributes of the instance's class are allowed. These could be, for example, any mapped columns or relationships.

code

编码

classmethod could_short()

whether could be shorted

Returns::

end_date

退市日

entity_id

entity id

entity_type

标的类型

exchange

所属交易所

classmethod get_interval_timestamps(*start_date*, *end_date*, *level: IntervalLevel*)

generate the timestamps for the level

Parameters::
- **start_date** –
- **end_date** –
- **level** –

classmethod get_providers() → List[str]

providers of the schema defined by cls

Returns:: providers

classmethod get_storages(*provider: Optional[str] = None*)

get the storages info

Parameters:: **provider** – provider

Returns:: storages

classmethod get_trading_dates(*start_date=None*, *end_date=None*)

overwrite it to get the trading dates of the entity

Parameters::
- **start_date** –
- **end_date** –

Returns:: list of dates

classmethod get_trading_intervals()

overwrite it to get the trading intervals of the entity

Returns:: list of time intervals, in format [(start,end)]

classmethod get_trading_t()

0 means t+0 1 means t+1

Returns::

id

id

classmethod is_finished_kdata_timestamp(*timestamp: Timestamp*, *level: IntervalLevel*)

Parameters::
- **timestamp** (*pd.Timestamp*) – the timestamp could be recorded in kdata of the level
- **level** (*zvt.domain.common.IntervalLevel*) –

Returns::

Return type:: bool

list_date

上市日

name

名字

classmethod query_data(*provider_index: int = 0*, *ids: Optional[List[str]] = None*, *entity_ids: Optional[List[str]] = None*, *entity_id: Optional[str] = None*, *codes: Optional[List[str]] = None*, *code: Optional[str] = None*, *level: Optional[Union[IntervalLevel, str]] = None*, *provider: Optional[str] = None*, *columns: Optional[List] = None*, *col_label: Optional[dict] = None*, *return_type: str = 'df'*, *start_timestamp: Optional[Union[Timestamp, str]] = None*, *end_timestamp: Optional[Union[Timestamp, str]] = None*, *filters: Optional[List] = None*, *session: Optional[Session] = None*, *order=None*, *limit: Optional[int] = None*, *distinct=None*, *index: Optional[Union[str, list]] = None*, *drop_index_col=False*, *time_field: str = 'timestamp'*)

query data by the arguments

Parameters::

- **provider_index** –
- **data_schema** –
- **ids** –
- **entity_ids** –
- **entity_id** –
- **codes** –
- **code** –
- **level** –
- **provider** –
- **columns** –
- **col_label** – dict with key(column), value(label
- **return_type** – df, domain or dict. default is df
- **start_timestamp** –
- **end_timestamp** –
- **filters** –
- **session** –
- **order** –
- **limit** –
- **index** – index field name, str for single index, str list for multiple index
- **drop_index_col** – whether drop the col if it's in index, default False
- **time_field** –

Returns:: results basing on return_type.

classmethod record_data(*provider_index: int = 0*, *provider: Optional[str] = None*, *force_update=None*, *sleeping_time=None*, *exchanges=None*, *entity_id=None*, *entity_ids=None*, *code=None*, *codes=None*, *real_time=None*, *fix_duplicate_way=None*, *start_timestamp=None*, *end_timestamp=None*, *one_day_trading_minutes=None*, ***kwargs*)

record data by the arguments

Parameters::

- **entity_id** –
- **provider_index** –
- **provider** –
- **force_update** –
- **sleeping_time** –
- **exchanges** –
- **entity_ids** –
- **code** –
- **codes** –
- **real_time** –
- **fix_duplicate_way** –
- **start_timestamp** –
- **end_timestamp** –
- **one_day_trading_minutes** –
- **kwargs** –

Returns::

classmethod register_provider(*provider*)

register the provider to the schema defined by cls

Parameters:: **provider** –

classmethod register_recorder_cls(*provider*, *recorder_cls*)

register the recorder for the schema

Parameters::

- **provider** –
- **recorder_cls** –

timestamp

the meaning could be different for different case,most time it means 'happen time'

zvt.domain.meta.etf_meta

Classes

`Etf`(**kwargs)

`EtfStock`(**kwargs)

Etf

class zvt.domain.meta.etf_meta.Etf(***kwargs*)

Bases: **`Base`**, **`Portfolio`**

__init__(***kwargs*)

A simple constructor that allows initialization from kwargs.

Sets attributes on the constructed instance using the names and values in `kwargs`.

Only keys that are present as attributes of the instance's class are allowed. These could be, for example, any mapped columns or relationships.

classmethod get_stocks(*code=None*, *codes=None*, *ids=None*, *timestamp=Timestamp('2024-08-28 22:08:11.321110')*, *provider=None*)

the publishing policy of portfolio positions is different for different types, overwrite this function for get the holding stocks in specific date

Parameters::

- **code** – portfolio(etf/block/index…) code
- **codes** – portfolio(etf/block/index…) codes
- **ids** – portfolio(etf/block/index…) ids
- **timestamp** – the date of the holding stocks
- **provider** – the data provider

Returns::

code

编码

classmethod could_short()

whether could be shorted

Returns::

end_date

退市日

entity_id

entity id

entity_type

标的类型

exchange

所属交易所

classmethod get_interval_timestamps(*start_date*, *end_date*, *level: IntervalLevel*)

generate the timestamps for the level

Parameters::
- **start_date** –
- **end_date** –
- **level** –

classmethod get_providers() → List[str]

providers of the schema defined by cls

Returns:: providers

classmethod get_storages(*provider: Optional[str] = None*)

get the storages info

Parameters:: **provider** – provider

Returns:: storages

classmethod get_trading_dates(*start_date=None*, *end_date=None*)

overwrite it to get the trading dates of the entity

Parameters::
- **start_date** –
- **end_date** –

Returns:: list of dates

classmethod get_trading_intervals()

overwrite it to get the trading intervals of the entity

Returns:: list of time intervals, in format [(start,end)]

classmethod get_trading_t()

0 means t+0 1 means t+1

Returns::

id

id

classmethod is_finished_kdata_timestamp(*timestamp: Timestamp*, *level: IntervalLevel*)

Parameters::
- **timestamp** (*pd.Timestamp*) – the timestamp could be recorded in kdata of the level
- **level** (*zvt.domain.common.IntervalLevel*) –

Returns::

Return type:: bool

list_date

上市日

name

名字

classmethod query_data(*provider_index: int = 0*, *ids: Optional[List[str]] = None*, *entity_ids: Optional[List[str]] = None*, *entity_id: Optional[str] = None*, *codes: Optional[List[str]] = None*, *code: Optional[str] = None*, *level: Optional[Union[IntervalLevel, str]] = None*, *provider: Optional[str] = None*, *columns: Optional[List] = None*, *col_label: Optional[dict] = None*, *return_type: str = 'df'*, *start_timestamp: Optional[Union[Timestamp, str]] = None*, *end_timestamp: Optional[Union[Timestamp, str]] = None*, *filters: Optional[List] = None*, *session: Optional[Session] = None*, *order=None*, *limit: Optional[int] =*

None, *distinct=None*, *index: Optional[Union[str, list]] = None*, *drop_index_col=False*, *time_field: str = 'timestamp'*)

query data by the arguments

Parameters::

- **provider_index** –
- **data_schema** –
- **ids** –
- **entity_ids** –
- **entity_id** –
- **codes** –
- **code** –
- **level** –
- **provider** –
- **columns** –
- **col_label** – dict with key(column), value(label
- **return_type** – df, domain or dict. default is df
- **start_timestamp** –
- **end_timestamp** –
- **filters** –
- **session** –
- **order** –
- **limit** –
- **index** – index field name, str for single index, str list for multiple index
- **drop_index_col** – whether drop the col if it's in index, default False
- **time_field** –

Returns:: results basing on return_type.

classmethod record_data(*provider_index: int = 0*, *provider: Optional[str] = None*, *force_update=None*, *sleeping_time=None*, *exchanges=None*, *entity_id=None*, *entity_ids=None*, *code=None*, *codes=None*, *real_time=None*, *fix_duplicate_way=None*, *start_timestamp=None*, *end_timestamp=None*, *one_day_trading_minutes=None*, ***kwargs*)

record data by the arguments

Parameters::

- **entity_id** –
- **provider_index** –
- **provider** –
- **force_update** –
- **sleeping_time** –
- **exchanges** –
- **entity_ids** –
- **code** –
- **codes** –
- **real_time** –
- **fix_duplicate_way** –
- **start_timestamp** –
- **end_timestamp** –
- **one_day_trading_minutes** –
- **kwargs** –

Returns::

classmethod register_provider(*provider*)

register the provider to the schema defined by cls

Parameters:: **provider** –

classmethod register_recorder_cls(*provider*, *recorder_cls*)

register the recorder for the schema

Parameters::

- **provider** –
- **recorder_cls** –

timestamp

the meaning could be different for different case,most time it means 'happen time'

EtfStock

class zvt.domain.meta.etf_meta.EtfStock(***kwargs*)

Bases: **`Base`**, **`PortfolioStockHistory`**

__init__(***kwargs*)

A simple constructor that allows initialization from kwargs.

Sets attributes on the constructed instance using the names and values in `kwargs`.

Only keys that are present as attributes of the instance's class are allowed. These could be, for example, any mapped columns or relationships.

code

portfolio编码

entity_id

entity id

entity_type

portfolio标的类型

exchange

portfolio所属交易所

classmethod get_providers() → List[str]

providers of the schema defined by cls

Returns:: providers

classmethod get_storages(*provider: Optional[str] = None*)

get the storages info

Parameters:: **provider** – provider
Returns:: storages

id

id

market_cap

持有股票的市值

name

portfolio名字

proportion

占净值比例

classmethod query_data(*provider_index: int = 0*, *ids: Optional[List[str]] = None*, *entity_ids: Optional[List[str]] = None*, *entity_id: Optional[str] = None*, *codes: Optional[List[str]] = None*, *code: Optional[str] = None*, *level: Optional[Union[IntervalLevel, str]] = None*, *provider: Optional[str] = None*, *columns: Optional[List] = None*, *col_label: Optional[dict] = None*, *return_type: str = 'df'*, *start_timestamp: Optional[Union[Timestamp, str]] = None*, *end_timestamp: Optional[Union[Timestamp, str]] = None*, *filters: Optional[List] = None*, *session: Optional[Session] = None*, *order=None*, *limit: Optional[int] = None*, *distinct=None*, *index: Optional[Union[str, list]] = None*, *drop_index_col=False*, *time_field: str = 'timestamp'*)

query data by the arguments

Parameters::

- **provider_index** –
- **data_schema** –
- **ids** –
- **entity_ids** –
- **entity_id** –
- **codes** –
- **code** –
- **level** –
- **provider** –
- **columns** –
- **col_label** – dict with key(column), value(label
- **return_type** – df, domain or dict. default is df
- **start_timestamp** –
- **end_timestamp** –
- **filters** –
- **session** –
- **order** –
- **limit** –
- **index** – index field name, str for single index, str list for multiple index
- **drop_index_col** – whether drop the col if it's in index, default False
- **time_field** –

Returns:: results basing on return_type.

classmethod record_data(*provider_index: int = 0*, *provider: Optional[str] = None*, *force_update=None*, *sleeping_time=None*, *exchanges=None*, *entity_id=None*, *entity_ids=None*, *code=None*, *codes=None*, *real_time=None*, *fix_duplicate_way=None*, *start_timestamp=None*, *end_timestamp=None*, *one_day_trading_minutes=None*, ***kwargs*)

record data by the arguments

Parameters::

- **entity_id** –
- **provider_index** –
- **provider** –
- **force_update** –
- **sleeping_time** –
- **exchanges** –
- **entity_ids** –
- **code** –
- **codes** –
- **real_time** –
- **fix_duplicate_way** –
- **start_timestamp** –
- **end_timestamp** –
- **one_day_trading_minutes** –
- **kwargs** –

Returns::

classmethod register_provider(*provider*)

register the provider to the schema defined by cls

Parameters:: **provider** –

classmethod register_recorder_cls(*provider*, *recorder_cls*)

register the recorder for the schema

Parameters::

- **provider** –
- **recorder_cls** –

report_date

3-31,6-30,9-30,12-31

report_period

报告期,season1,half_year,season3,year

shares

持有股票的数量

stock_code

stock_id

stock_name

timestamp

the meaning could be different for different case,most time it means ‘happen time’

zvt.domain.meta.fund_meta

Classes

`Fund`(**kwargs)

`FundStock`(**kwargs)

Fund

class zvt.domain.meta.fund_meta.Fund(**kwargs)

Bases: **Base**, **Portfolio**

__init__(**kwargs)

A simple constructor that allows initialization from kwargs.

Sets attributes on the constructed instance using the names and values in `kwargs`.

Only keys that are present as attributes of the instance's class are allowed. These could be, for example, any mapped columns or relationships.

advisor

基金管理人

trustee

基金托管人

operate_mode_id

编码 基金运作方式 401001 开放式基金 401002 封闭式基金 401003 QDII 401004 FOF 401005 ETF 401006 LOF 基金运作方式编码

operate_mode

基金运作方式

underlying_asset_type_id

编码 基金类别 402001 股票型 402002 货币型 402003 债券型 402004 混合型 402005 基金型 402006 贵金属 402007 封闭式 投资标的类型编码

underlying_asset_type

投资标的类型

classmethod get_stocks(*code=None*, *codes=None*, *ids=None*, *timestamp=Timestamp('2024-08-28 22:08:11.554593')*, *provider=None*)

the publishing policy of portfolio positions is different for different types, overwrite this function for get the holding stocks in specific date

Parameters::

- **code** – portfolio(etf/block/index…) code
- **codes** – portfolio(etf/block/index…) codes
- **ids** – portfolio(etf/block/index…) ids
- **timestamp** – the date of the holding stocks
- **provider** – the data provider

Returns::

code

编码

classmethod could_short()

whether could be shorted

Returns::

end_date

退市日

entity_id

entity id

entity_type

标的类型

exchange

所属交易所

classmethod get_interval_timestamps(*start_date*, *end_date*, *level: IntervalLevel*)

generate the timestamps for the level

Parameters::
- **start_date** –
- **end_date** –
- **level** –

classmethod get_providers() → List[str]

providers of the schema defined by cls

Returns:: providers

classmethod get_storages(*provider: Optional[str] = None*)

get the storages info

Parameters:: **provider** – provider

Returns:: storages

classmethod get_trading_dates(*start_date=None*, *end_date=None*)

overwrite it to get the trading dates of the entity

Parameters::
- **start_date** –
- **end_date** –

Returns:: list of dates

classmethod get_trading_intervals()

overwrite it to get the trading intervals of the entity

Returns:: list of time intervals, in format [(start,end)]

classmethod get_trading_t()

0 means t+0 1 means t+1

Returns::

id

id

classmethod is_finished_kdata_timestamp(*timestamp: Timestamp*, *level: IntervalLevel*)

Parameters::
- **timestamp** (*pd.Timestamp*) – the timestamp could be recorded in kdata of the level
- **level** (*zvt.domain.common.IntervalLevel*) –

Returns::

Return type:: bool

list_date

上市日

name

名字

classmethod query_data(*provider_index: int = 0*, *ids: Optional[List[str]] = None*, *entity_ids: Optional[List[str]] = None*, *entity_id: Optional[str] = None*, *codes: Optional[List[str]] = None*, *code: Optional[str] = None*, *level: Optional[Union[IntervalLevel, str]] = None*, *provider: Optional[str] = None*, *columns: Optional[List] = None*, *col_label: Optional[dict] = None*, *return_type: str = 'df'*, *start_timestamp: Optional[Union[Timestamp, str]] = None*, *end_timestamp: Optional[Union[Timestamp, str]] = None*, *filters: Optional[List] = None*, *session: Optional[Session] = None*, *order=None*, *limit: Optional[int] = None*, *distinct=None*, *index: Optional[Union[str, list]] = None*, *drop_index_col=False*, *time_field: str = 'timestamp'*)

query data by the arguments

Parameters::

- **provider_index** –
- **data_schema** –
- **ids** –
- **entity_ids** –
- **entity_id** –
- **codes** –
- **code** –
- **level** –
- **provider** –
- **columns** –
- **col_label** – dict with key(column), value(label
- **return_type** – df, domain or dict. default is df
- **start_timestamp** –
- **end_timestamp** –
- **filters** –
- **session** –
- **order** –
- **limit** –
- **index** – index field name, str for single index, str list for multiple index
- **drop_index_col** – whether drop the col if it's in index, default False
- **time_field** –

Returns:: results basing on return_type.

classmethod record_data(*provider_index: int = 0*, *provider: Optional[str] = None*, *force_update=None*, *sleeping_time=None*, *exchanges=None*, *entity_id=None*, *entity_ids=None*, *code=None*, *codes=None*, *real_time=None*, *fix_duplicate_way=None*, *start_timestamp=None*, *end_timestamp=None*, *one_day_trading_minutes=None*, ***kwargs*)

record data by the arguments

Parameters::

- **entity_id** –
- **provider_index** –
- **provider** –
- **force_update** –
- **sleeping_time** –
- **exchanges** –
- **entity_ids** –
- **code** –
- **codes** –
- **real_time** –
- **fix_duplicate_way** –
- **start_timestamp** –
- **end_timestamp** –
- **one_day_trading_minutes** –
- **kwargs** –

Returns::

classmethod register_provider(*provider*)

register the provider to the schema defined by cls

Parameters:: **provider** –

classmethod register_recorder_cls(*provider*, *recorder_cls*)

register the recorder for the schema

Parameters::

- **provider** –
- **recorder_cls** –

timestamp

the meaning could be different for different case,most time it means 'happen time'

FundStock

class zvt.domain.meta.fund_meta.FundStock(***kwargs*)

Bases: **`Base`**, **`PortfolioStockHistory`**

__init__(***kwargs*)

A simple constructor that allows initialization from kwargs.

Sets attributes on the constructed instance using the names and values in `kwargs`.

Only keys that are present as attributes of the instance's class are allowed. These could be, for example, any mapped columns or relationships.

code

portfolio编码

entity_id

entity id

entity_type

portfolio标的类型

exchange

portfolio所属交易所

classmethod get_providers() → List[str]

providers of the schema defined by cls

Returns:: providers

classmethod get_storages(*provider: Optional[str] = None*)

get the storages info

Parameters:: **provider** – provider
Returns:: storages

id

id

market_cap

持有股票的市值

name

portfolio名字

proportion

占净值比例

classmethod query_data(*provider_index: int = 0*, *ids: Optional[List[str]] = None*, *entity_ids: Optional[List[str]] = None*, *entity_id: Optional[str] = None*, *codes: Optional[List[str]] = None*, *code: Optional[str] = None*, *level: Optional[Union[IntervalLevel, str]] = None*, *provider: Optional[str] = None*, *columns: Optional[List] = None*, *col_label: Optional[dict] = None*, *return_type: str = 'df'*, *start_timestamp: Optional[Union[Timestamp, str]] = None*, *end_timestamp: Optional[Union[Timestamp, str]] = None*, *filters: Optional[List] = None*, *session: Optional[Session] = None*, *order=None*, *limit: Optional[int] = None*, *distinct=None*, *index: Optional[Union[str, list]] = None*, *drop_index_col=False*, *time_field: str = 'timestamp'*)

query data by the arguments

Parameters::

- **provider_index** –
- **data_schema** –
- **ids** –
- **entity_ids** –
- **entity_id** –
- **codes** –
- **code** –
- **level** –
- **provider** –
- **columns** –
- **col_label** – dict with key(column), value(label
- **return_type** – df, domain or dict. default is df
- **start_timestamp** –
- **end_timestamp** –
- **filters** –
- **session** –
- **order** –
- **limit** –
- **index** – index field name, str for single index, str list for multiple index
- **drop_index_col** – whether drop the col if it's in index, default False
- **time_field** –

Returns:: results basing on return_type.

classmethod record_data(*provider_index: int = 0*, *provider: Optional[str] = None*, *force_update=None*, *sleeping_time=None*, *exchanges=None*, *entity_id=None*, *entity_ids=None*, *code=None*, *codes=None*, *real_time=None*, *fix_duplicate_way=None*, *start_timestamp=None*, *end_timestamp=None*, *one_day_trading_minutes=None*, ***kwargs*)

record data by the arguments

Parameters::

- **entity_id** –
- **provider_index** –
- **provider** –
- **force_update** –
- **sleeping_time** –
- **exchanges** –
- **entity_ids** –
- **code** –
- **codes** –
- **real_time** –
- **fix_duplicate_way** –
- **start_timestamp** –
- **end_timestamp** –
- **one_day_trading_minutes** –
- **kwargs** –

Returns::

classmethod register_provider(*provider*)

register the provider to the schema defined by cls

Parameters:: **provider** –

classmethod register_recorder_cls(*provider*, *recorder_cls*)

register the recorder for the schema

Parameters::

- **provider** –
- **recorder_cls** –

report_date

3-31,6-30,9-30,12-31

report_period

报告期,season1,half_year,season3,year

shares

持有股票的数量

stock_code

stock_id

stock_name

timestamp

the meaning could be different for different case,most time it means ‘happen time’

zvt.domain.meta.future_meta

Classes

Future(**kwargs)

Future

class zvt.domain.meta.future_meta.Future(**kwargs*)

Bases: **Base**, **TradableEntity**

__init__(**kwargs*)

A simple constructor that allows initialization from kwargs.

Sets attributes on the constructed instance using the names and values in `kwargs`.

Only keys that are present as attributes of the instance's class are allowed. These could be, for example, any mapped columns or relationships.

code

编码

classmethod could_short()

whether could be shorted

Returns::

end_date

退市日

entity_id

entity id

entity_type

标的类型

exchange

所属交易所

classmethod get_interval_timestamps(*start_date*, *end_date*, *level: IntervalLevel*)

generate the timestamps for the level

Parameters::
- **start_date** –
- **end_date** –
- **level** –

classmethod get_providers() → List[str]

providers of the schema defined by cls

Returns:: providers

classmethod get_storages(*provider: Optional[str] = None*)

get the storages info

Parameters:: **provider** – provider

Returns:: storages

classmethod get_trading_dates(*start_date=None*, *end_date=None*)

overwrite it to get the trading dates of the entity

Parameters::
- **start_date** –
- **end_date** –

Returns:: list of dates

classmethod get_trading_intervals()

overwrite it to get the trading intervals of the entity

Returns:: list of time intervals, in format [(start,end)]

classmethod get_trading_t()

0 means t+0 1 means t+1

Returns::

id

id

classmethod is_finished_kdata_timestamp(*timestamp: Timestamp*, *level: IntervalLevel*)

Parameters::
- **timestamp** (*pd.Timestamp*) – the timestamp could be recorded in kdata of the level
- **level** (*zvt.domain.common.IntervalLevel*) –

Returns::

Return type:: bool

list_date

上市日

name

名字

classmethod query_data(*provider_index: int = 0*, *ids: Optional[List[str]] = None*, *entity_ids: Optional[List[str]] = None*, *entity_id: Optional[str] = None*, *codes: Optional[List[str]] = None*, *code: Optional[str] = None*, *level: Optional[Union[IntervalLevel, str]] = None*, *provider: Optional[str] = None*, *columns: Optional[List] = None*, *col_label: Optional[dict] = None*, *return_type: str = 'df'*, *start_timestamp: Optional[Union[Timestamp, str]] = None*, *end_timestamp: Optional[Union[Timestamp, str]] = None*, *filters: Optional[List] = None*, *session: Optional[Session] = None*, *order=None*, *limit: Optional[int] = None*, *distinct=None*, *index: Optional[Union[str, list]] = None*, *drop_index_col=False*, *time_field: str = 'timestamp'*)

query data by the arguments

Parameters::

- **provider_index** –
- **data_schema** –
- **ids** –
- **entity_ids** –
- **entity_id** –
- **codes** –
- **code** –
- **level** –
- **provider** –
- **columns** –
- **col_label** – dict with key(column), value(label
- **return_type** – df, domain or dict. default is df
- **start_timestamp** –
- **end_timestamp** –
- **filters** –
- **session** –
- **order** –
- **limit** –
- **index** – index field name, str for single index, str list for multiple index
- **drop_index_col** – whether drop the col if it's in index, default False
- **time_field** –

Returns:: results basing on return_type.

classmethod record_data(*provider_index: int = 0*, *provider: Optional[str] = None*, *force_update=None*, *sleeping_time=None*, *exchanges=None*, *entity_id=None*, *entity_ids=None*, *code=None*, *codes=None*, *real_time=None*, *fix_duplicate_way=None*, *start_timestamp=None*, *end_timestamp=None*, *one_day_trading_minutes=None*, ***kwargs*)

record data by the arguments

Parameters::

- **entity_id** –
- **provider_index** –
- **provider** –
- **force_update** –
- **sleeping_time** –
- **exchanges** –
- **entity_ids** –
- **code** –
- **codes** –
- **real_time** –
- **fix_duplicate_way** –
- **start_timestamp** –
- **end_timestamp** –
- **one_day_trading_minutes** –
- **kwargs** –

Returns::

classmethod register_provider(*provider*)

register the provider to the schema defined by cls

Parameters:: **provider** –

classmethod register_recorder_cls(*provider*, *recorder_cls*)

register the recorder for the schema

Parameters::

- **provider** –
- **recorder_cls** –

timestamp

the meaning could be different for different case,most time it means 'happen time'

zvt.domain.meta.index_meta

Classes

`Index`(**kwargs)

`IndexStock`(**kwargs)

Index

class zvt.domain.meta.index_meta.Index(***kwargs*)

Bases: **Base**, **Portfolio**

__init__(***kwargs*)

A simple constructor that allows initialization from kwargs.

Sets attributes on the constructed instance using the names and values in `kwargs`.

Only keys that are present as attributes of the instance's class are allowed. These could be, for example, any mapped columns or relationships.

publisher

发布商

category

类别 see IndexCategory

base_point

基准点数

code

编码

classmethod could_short()

whether could be shorted

Returns::

end_date

退市日

entity_id

entity id

entity_type

标的类型

exchange

所属交易所

classmethod get_interval_timestamps(*start_date*, *end_date*, *level: IntervalLevel*)

generate the timestamps for the level

Parameters::

- **start_date** –
- **end_date** –
- **level** –

classmethod get_providers() → List[str]

providers of the schema defined by cls

Returns:: providers

classmethod get_stocks(*code=None*, *codes=None*, *ids=None*, *timestamp=Timestamp('2024-08-28 22:08:09.719543')*, *provider=None*)

the publishing policy of portfolio positions is different for different types, overwrite this function for get the holding stocks in specific date

Parameters::

- **code** – portfolio(etf/block/index…) code
- **codes** – portfolio(etf/block/index…) codes
- **ids** – portfolio(etf/block/index…) ids
- **timestamp** – the date of the holding stocks
- **provider** – the data provider

Returns::

classmethod get_storages(*provider: Optional[str] = None*)

get the storages info

Parameters:: **provider** – provider
Returns:: storages

classmethod get_trading_dates(*start_date=None*, *end_date=None*)

overwrite it to get the trading dates of the entity

Parameters::
- **start_date** –
- **end_date** –

Returns:: list of dates

classmethod get_trading_intervals()

overwrite it to get the trading intervals of the entity

Returns:: list of time intervals, in format [(start,end)]

classmethod get_trading_t()

0 means t+0 1 means t+1

Returns::

id

id

classmethod is_finished_kdata_timestamp(*timestamp: Timestamp*, *level: IntervalLevel*)

Parameters::
- **timestamp** (*pd.Timestamp*) – the timestamp could be recorded in kdata of the level
- **level** (*zvt.domain.common.IntervalLevel*) –

Returns::
Return type:: bool

list_date

上市日

name

名字

classmethod query_data(*provider_index: int = 0*, *ids: Optional[List[str]] = None*, *entity_ids: Optional[List[str]] = None*, *entity_id: Optional[str] = None*, *codes: Optional[List[str]] = None*, *code: Optional[str] = None*, *level: Optional[Union[IntervalLevel, str]] = None*, *provider: Optional[str] = None*, *columns: Optional[List] = None*, *col_label: Optional[dict] = None*, *return_type: str = 'df'*, *start_timestamp: Optional[Union[Timestamp, str]] = None*, *end_timestamp: Optional[Union[Timestamp, str]] = None*, *filters: Optional[List] = None*, *session: Optional[Session] = None*, *order=None*, *limit: Optional[int] = None*, *distinct=None*, *index: Optional[Union[str, list]] = None*, *drop_index_col=False*, *time_field: str = 'timestamp'*)

query data by the arguments

Parameters::

- **provider_index** –
- **data_schema** –
- **ids** –
- **entity_ids** –
- **entity_id** –
- **codes** –
- **code** –
- **level** –
- **provider** –
- **columns** –
- **col_label** – dict with key(column), value(label
- **return_type** – df, domain or dict. default is df
- **start_timestamp** –
- **end_timestamp** –
- **filters** –
- **session** –
- **order** –
- **limit** –
- **index** – index field name, str for single index, str list for multiple index
- **drop_index_col** – whether drop the col if it's in index, default False
- **time_field** –

Returns:: results basing on return_type.

classmethod record_data(*provider_index: int = 0*, *provider: Optional[str] = None*, *force_update=None*, *sleeping_time=None*, *exchanges=None*, *entity_id=None*, *entity_ids=None*, *code=None*, *codes=None*, *real_time=None*, *fix_duplicate_way=None*, *start_timestamp=None*, *end_timestamp=None*, *one_day_trading_minutes=None*, ***kwargs*)

record data by the arguments

Parameters::

- **entity_id** –
- **provider_index** –
- **provider** –
- **force_update** –
- **sleeping_time** –
- **exchanges** –
- **entity_ids** –
- **code** –
- **codes** –
- **real_time** –
- **fix_duplicate_way** –
- **start_timestamp** –
- **end_timestamp** –
- **one_day_trading_minutes** –
- **kwargs** –

Returns::

classmethod register_provider(*provider*)

register the provider to the schema defined by cls

Parameters:: **provider** –

classmethod register_recorder_cls(*provider*, *recorder_cls*)

register the recorder for the schema

Parameters::

- **provider** –
- **recorder_cls** –

timestamp

the meaning could be different for different case,most time it means 'happen time'

IndexStock

class zvt.domain.meta.index_meta.IndexStock(**kwargs)

Bases: **Base**, **PortfolioStockHistory**

__init__(**kwargs)

A simple constructor that allows initialization from kwargs.

Sets attributes on the constructed instance using the names and values in `kwargs`.

Only keys that are present as attributes of the instance's class are allowed. These could be, for example, any mapped columns or relationships.

code

portfolio编码

entity_id

entity id

entity_type

portfolio标的类型

exchange

portfolio所属交易所

classmethod get_providers() → List[str]

providers of the schema defined by cls

Returns:: providers

classmethod get_storages(*provider: Optional[str] = None*)

get the storages info

Parameters:: **provider** – provider
Returns:: storages

id

id

market_cap

持有股票的市值

name

portfolio名字

proportion

占净值比例

classmethod query_data(*provider_index: int = 0*, *ids: Optional[List[str]] = None*, *entity_ids: Optional[List[str]] = None*, *entity_id: Optional[str] = None*, *codes: Optional[List[str]] = None*, *code: Optional[str] = None*, *level: Optional[Union[IntervalLevel, str]] = None*, *provider: Optional[str] = None*, *columns: Optional[List] = None*, *col_label: Optional[dict] = None*, *return_type: str = 'df'*, *start_timestamp: Optional[Union[Timestamp, str]] = None*, *end_timestamp: Optional[Union[Timestamp, str]] = None*, *filters: Optional[List] = None*, *session: Optional[Session] = None*, *order=None*, *limit: Optional[int] = None*, *distinct=None*, *index: Optional[Union[str, list]] = None*, *drop_index_col=False*, *time_field: str = 'timestamp'*)

query data by the arguments

Parameters::

- **provider_index** –
- **data_schema** –
- **ids** –
- **entity_ids** –
- **entity_id** –
- **codes** –
- **code** –
- **level** –
- **provider** –
- **columns** –
- **col_label** – dict with key(column), value(label
- **return_type** – df, domain or dict. default is df
- **start_timestamp** –
- **end_timestamp** –
- **filters** –
- **session** –
- **order** –
- **limit** –
- **index** – index field name, str for single index, str list for multiple index
- **drop_index_col** – whether drop the col if it's in index, default False
- **time_field** –

Returns:: results basing on return_type.

classmethod record_data(*provider_index: int = 0*, *provider: Optional[str] = None*, *force_update=None*, *sleeping_time=None*, *exchanges=None*, *entity_id=None*, *entity_ids=None*, *code=None*, *codes=None*, *real_time=None*, *fix_duplicate_way=None*, *start_timestamp=None*, *end_timestamp=None*, *one_day_trading_minutes=None*, ***kwargs*)

record data by the arguments

Parameters::

- **entity_id** –
- **provider_index** –
- **provider** –
- **force_update** –
- **sleeping_time** –
- **exchanges** –
- **entity_ids** –
- **code** –
- **codes** –
- **real_time** –
- **fix_duplicate_way** –
- **start_timestamp** –
- **end_timestamp** –
- **one_day_trading_minutes** –
- **kwargs** –

Returns::

classmethod register_provider(*provider*)

register the provider to the schema defined by cls

Parameters:: **provider** –

classmethod register_recorder_cls(*provider*, *recorder_cls*)

register the recorder for the schema

Parameters::

- **provider** –
- **recorder_cls** –

report_date

3-31,6-30,9-30,12-31

report_period

报告期,season1,half_year,season3,year

shares

持有股票的数量

stock_code

stock_id

stock_name

timestamp

the meaning could be different for different case,most time it means ‘happen time’

zvt.domain.meta.indexus_meta

Classes

`Indexus`(**kwargs)

Indexus

class zvt.domain.meta.indexus_meta.Indexus(***kwargs*)

Bases: **`Base`**, **`Portfolio`**

__init__(***kwargs*)

A simple constructor that allows initialization from kwargs.

Sets attributes on the constructed instance using the names and values in `kwargs`.

Only keys that are present as attributes of the instance's class are allowed. These could be, for example, any mapped columns or relationships.

publisher

发布商

category

类别 see IndexCategory

base_point

基准点数

code

编码

classmethod could_short()

whether could be shorted

Returns::

end_date

退市日

entity_id

entity id

entity_type

标的类型

exchange

所属交易所

classmethod get_interval_timestamps(*start_date*, *end_date*, *level: IntervalLevel*)

generate the timestamps for the level

Parameters::

- **start_date** –
- **end_date** –
- **level** –

classmethod get_providers() → List[str]

providers of the schema defined by cls

Returns:: providers

classmethod get_stocks(*code=None*, *codes=None*, *ids=None*, *timestamp=Timestamp('2024-08-28 22:08:09.719543')*, *provider=None*)

the publishing policy of portfolio positions is different for different types, overwrite this function for get the holding stocks in specific date

Parameters::

- **code** – portfolio(etf/block/index…) code
- **codes** – portfolio(etf/block/index…) codes
- **ids** – portfolio(etf/block/index…) ids
- **timestamp** – the date of the holding stocks
- **provider** – the data provider

Returns::

classmethod get_storages(*provider: Optional[str] = None*)

get the storages info

Parameters:: **provider** – provider
Returns:: storages

classmethod get_trading_dates(*start_date=None*, *end_date=None*)

overwrite it to get the trading dates of the entity

Parameters::
- **start_date** –
- **end_date** –

Returns:: list of dates

classmethod get_trading_intervals()

overwrite it to get the trading intervals of the entity

Returns:: list of time intervals, in format [(start,end)]

classmethod get_trading_t()

0 means t+0 1 means t+1

Returns::

id

id

classmethod is_finished_kdata_timestamp(*timestamp: Timestamp*, *level: IntervalLevel*)

Parameters::
- **timestamp** (*pd.Timestamp*) – the timestamp could be recorded in kdata of the level
- **level** (*zvt.domain.common.IntervalLevel*) –

Returns::
Return type:: bool

list_date

上市日

name

名字

classmethod query_data(*provider_index: int = 0*, *ids: Optional[List[str]] = None*, *entity_ids: Optional[List[str]] = None*, *entity_id: Optional[str] = None*, *codes: Optional[List[str]] = None*, *code: Optional[str] = None*, *level: Optional[Union[IntervalLevel, str]] = None*, *provider: Optional[str] = None*, *columns: Optional[List] = None*, *col_label: Optional[dict] = None*, *return_type: str = 'df'*, *start_timestamp: Optional[Union[Timestamp, str]] = None*, *end_timestamp: Optional[Union[Timestamp, str]] = None*, *filters: Optional[List] = None*, *session: Optional[Session] = None*, *order=None*, *limit: Optional[int] = None*, *distinct=None*, *index: Optional[Union[str, list]] = None*, *drop_index_col=False*, *time_field: str = 'timestamp'*)

query data by the arguments

Parameters::

- **provider_index** –
- **data_schema** –
- **ids** –
- **entity_ids** –
- **entity_id** –
- **codes** –
- **code** –
- **level** –
- **provider** –
- **columns** –
- **col_label** – dict with key(column), value(label
- **return_type** – df, domain or dict. default is df
- **start_timestamp** –
- **end_timestamp** –
- **filters** –
- **session** –
- **order** –
- **limit** –
- **index** – index field name, str for single index, str list for multiple index
- **drop_index_col** – whether drop the col if it's in index, default False
- **time_field** –

Returns:: results basing on return_type.

classmethod record_data(*provider_index: int = 0*, *provider: Optional[str] = None*, *force_update=None*, *sleeping_time=None*, *exchanges=None*, *entity_id=None*, *entity_ids=None*, *code=None*, *codes=None*, *real_time=None*, *fix_duplicate_way=None*, *start_timestamp=None*, *end_timestamp=None*, *one_day_trading_minutes=None*, ***kwargs*)

record data by the arguments

Parameters::

- **entity_id** –
- **provider_index** –
- **provider** –
- **force_update** –
- **sleeping_time** –
- **exchanges** –
- **entity_ids** –
- **code** –
- **codes** –
- **real_time** –
- **fix_duplicate_way** –
- **start_timestamp** –
- **end_timestamp** –
- **one_day_trading_minutes** –
- **kwargs** –

Returns::

classmethod register_provider(*provider*)

register the provider to the schema defined by cls

Parameters:: **provider** –

classmethod register_recorder_cls(*provider*, *recorder_cls*)

register the recorder for the schema

Parameters::

- **provider** –
- **recorder_cls** –

timestamp

the meaning could be different for different case,most time it means 'happen time'

zvt.domain.meta.stock_meta

Classes

`Stock`(**kwargs)

`StockDetail`(**kwargs)

Stock

class zvt.domain.meta.stock_meta.Stock(**kwargs)

Bases: **Base**, **TradableEntity**

__init__(**kwargs)

A simple constructor that allows initialization from kwargs.

Sets attributes on the constructed instance using the names and values in `kwargs`.

Only keys that are present as attributes of the instance's class are allowed. These could be, for example, any mapped columns or relationships.

code

编码

classmethod could_short()

whether could be shorted

Returns::

end_date

退市日

entity_id

entity id

entity_type

标的类型

exchange

所属交易所

classmethod get_interval_timestamps(*start_date*, *end_date*, *level: IntervalLevel*)

generate the timestamps for the level

Parameters::
- **start_date** –
- **end_date** –
- **level** –

classmethod get_providers() → List[str]

providers of the schema defined by cls

Returns:: providers

classmethod get_storages(*provider: Optional[str] = None*)

get the storages info

Parameters:: **provider** – provider

Returns:: storages

classmethod get_trading_dates(*start_date=None*, *end_date=None*)

overwrite it to get the trading dates of the entity

Parameters::
- **start_date** –
- **end_date** –

Returns:: list of dates

classmethod get_trading_intervals()

overwrite it to get the trading intervals of the entity

Returns:: list of time intervals, in format [(start,end)]

classmethod get_trading_t()

0 means t+0 1 means t+1

Returns::

id

id

classmethod is_finished_kdata_timestamp(*timestamp: Timestamp*, *level: IntervalLevel*)

Parameters::
- **timestamp** (*pd.Timestamp*) – the timestamp could be recorded in kdata of the level
- **level** (*zvt.domain.common.IntervalLevel*) –

Returns::

Return type:: bool

list_date

上市日

name

名字

classmethod query_data(*provider_index: int = 0*, *ids: Optional[List[str]] = None*, *entity_ids: Optional[List[str]] = None*, *entity_id: Optional[str] = None*, *codes: Optional[List[str]] = None*, *code: Optional[str] = None*, *level: Optional[Union[IntervalLevel, str]] = None*, *provider: Optional[str] = None*, *columns: Optional[List] = None*, *col_label: Optional[dict] = None*, *return_type: str = 'df'*, *start_timestamp: Optional[Union[Timestamp, str]] = None*, *end_timestamp: Optional[Union[Timestamp, str]] = None*, *filters: Optional[List] = None*, *session: Optional[Session] = None*, *order=None*, *limit: Optional[int] = None*, *distinct=None*, *index: Optional[Union[str, list]] = None*, *drop_index_col=False*, *time_field: str = 'timestamp'*)

query data by the arguments

Parameters::

- **provider_index** –
- **data_schema** –
- **ids** –
- **entity_ids** –
- **entity_id** –
- **codes** –
- **code** –
- **level** –
- **provider** –
- **columns** –
- **col_label** – dict with key(column), value(label
- **return_type** – df, domain or dict. default is df
- **start_timestamp** –
- **end_timestamp** –
- **filters** –
- **session** –
- **order** –
- **limit** –
- **index** – index field name, str for single index, str list for multiple index
- **drop_index_col** – whether drop the col if it's in index, default False
- **time_field** –

Returns:: results basing on return_type.

classmethod record_data(*provider_index: int = 0*, *provider: Optional[str] = None*, *force_update=None*, *sleeping_time=None*, *exchanges=None*, *entity_id=None*, *entity_ids=None*, *code=None*, *codes=None*, *real_time=None*, *fix_duplicate_way=None*, *start_timestamp=None*, *end_timestamp=None*, *one_day_trading_minutes=None*, ***kwargs*)

record data by the arguments

Parameters::

- **entity_id** –
- **provider_index** –
- **provider** –
- **force_update** –
- **sleeping_time** –
- **exchanges** –
- **entity_ids** –
- **code** –
- **codes** –
- **real_time** –
- **fix_duplicate_way** –
- **start_timestamp** –
- **end_timestamp** –
- **one_day_trading_minutes** –
- **kwargs** –

Returns::

classmethod register_provider(*provider*)

register the provider to the schema defined by cls

Parameters:: **provider** –

classmethod register_recorder_cls(*provider*, *recorder_cls*)

register the recorder for the schema

Parameters::

- **provider** –
- **recorder_cls** –

timestamp

the meaning could be different for different case,most time it means 'happen time'

StockDetail

class zvt.domain.meta.stock_meta.StockDetail(**kwargs)

Bases: **Base**, **TradableEntity**

__init__(**kwargs)

A simple constructor that allows initialization from kwargs.

Sets attributes on the constructed instance using the names and values in `kwargs`.

Only keys that are present as attributes of the instance's class are allowed. These could be, for example, any mapped columns or relationships.

industries

所属行业

industry_indices

行业指数

concept_indices

所属板块

area_indices

所属区域

date_of_establishment

成立日期

profile

公司简介

main_business

主营业务

issues
发行量(股)

price
发行价格

raising_fund
募资净额(元)

issue_pe
发行市盈率

net_winning_rate
网上中签率

code
编码

classmethod could_short()
whether could be shorted

Returns::

end_date
退市日

entity_id
entity id

entity_type
标的类型

exchange
所属交易所

classmethod get_interval_timestamps(*start_date*, *end_date*, *level: IntervalLevel*)

generate the timestamps for the level

Parameters::
- **start_date** –
- **end_date** –
- **level** –

classmethod get_providers() → List[str]

providers of the schema defined by cls

Returns:: providers

classmethod get_storages(*provider: Optional[str] = None*)

get the storages info

Parameters:: **provider** – provider

Returns:: storages

classmethod get_trading_dates(*start_date=None*, *end_date=None*)

overwrite it to get the trading dates of the entity

Parameters::
- **start_date** –
- **end_date** –

Returns:: list of dates

classmethod get_trading_intervals()

overwrite it to get the trading intervals of the entity

Returns:: list of time intervals, in format [(start,end)]

classmethod get_trading_t()

0 means t+0 1 means t+1

Returns::

id

id

classmethod is_finished_kdata_timestamp(*timestamp: Timestamp*, *level: IntervalLevel*)

Parameters::
- **timestamp** (*pd.Timestamp*) – the timestamp could be recorded in kdata of the level
- **level** (*zvt.domain.common.IntervalLevel*) –

Returns::

Return type:: bool

list_date

上市日

name

名字

classmethod query_data(*provider_index: int = 0*, *ids: Optional[List[str]] = None*, *entity_ids: Optional[List[str]] = None*, *entity_id: Optional[str] = None*, *codes: Optional[List[str]] = None*, *code: Optional[str] = None*, *level: Optional[Union[IntervalLevel, str]] = None*, *provider: Optional[str] = None*, *columns: Optional[List] = None*, *col_label: Optional[dict] = None*, *return_type: str = 'df'*, *start_timestamp: Optional[Union[Timestamp, str]] = None*, *end_timestamp: Optional[Union[Timestamp, str]] = None*, *filters: Optional[List] = None*, *session: Optional[Session] = None*, *order=None*, *limit: Optional[int] = None*, *distinct=None*, *index: Optional[Union[str, list]] = None*, *drop_index_col=False*, *time_field: str = 'timestamp'*)

query data by the arguments

Parameters::

- **provider_index** –
- **data_schema** –
- **ids** –
- **entity_ids** –
- **entity_id** –
- **codes** –
- **code** –
- **level** –
- **provider** –
- **columns** –
- **col_label** – dict with key(column), value(label
- **return_type** – df, domain or dict. default is df
- **start_timestamp** –
- **end_timestamp** –
- **filters** –
- **session** –
- **order** –
- **limit** –
- **index** – index field name, str for single index, str list for multiple index
- **drop_index_col** – whether drop the col if it's in index, default False
- **time_field** –

Returns:: results basing on return_type.

classmethod record_data(*provider_index: int = 0*, *provider: Optional[str] = None*, *force_update=None*, *sleeping_time=None*, *exchanges=None*, *entity_id=None*, *entity_ids=None*, *code=None*, *codes=None*, *real_time=None*, *fix_duplicate_way=None*, *start_timestamp=None*, *end_timestamp=None*, *one_day_trading_minutes=None*, ***kwargs*)

record data by the arguments

Parameters::

- **entity_id** –
- **provider_index** –
- **provider** –
- **force_update** –
- **sleeping_time** –
- **exchanges** –
- **entity_ids** –
- **code** –
- **codes** –
- **real_time** –
- **fix_duplicate_way** –
- **start_timestamp** –
- **end_timestamp** –
- **one_day_trading_minutes** –
- **kwargs** –

Returns::

classmethod register_provider(*provider*)

register the provider to the schema defined by cls

Parameters:: **provider** –

classmethod register_recorder_cls(*provider*, *recorder_cls*)

register the recorder for the schema

Parameters::

- **provider** –
- **recorder_cls** –

timestamp

the meaning could be different for different case,most time it means 'happen time'

zvt.domain.meta.stockhk_meta

Classes

`Stockhk`(**kwargs)

Stockhk

class zvt.domain.meta.stockhk_meta.Stockhk(**kwargs)

Bases: **Base**, **TradableEntity**

__init__(**kwargs)

A simple constructor that allows initialization from kwargs.

Sets attributes on the constructed instance using the names and values in `kwargs`.

Only keys that are present as attributes of the instance's class are allowed. These could be, for example, any mapped columns or relationships.

south

是否属于港股通

code

编码

classmethod could_short()

whether could be shorted

Returns::

end_date

退市日

entity_id

entity id

entity_type

标的类型

exchange

所属交易所

classmethod get_interval_timestamps(*start_date*, *end_date*, *level: IntervalLevel*)

generate the timestamps for the level

Parameters::
- **start_date** –
- **end_date** –
- **level** –

classmethod get_providers() → List[str]

providers of the schema defined by cls

Returns:: providers

classmethod get_storages(*provider: Optional[str] = None*)

get the storages info

Parameters:: **provider** – provider

Returns:: storages

classmethod get_trading_dates(*start_date=None*, *end_date=None*)

overwrite it to get the trading dates of the entity

Parameters::
- **start_date** –
- **end_date** –

Returns:: list of dates

classmethod get_trading_intervals()

overwrite it to get the trading intervals of the entity

Returns:: list of time intervals, in format [(start,end)]

classmethod get_trading_t()

0 means t+0 1 means t+1

Returns::

id

id

classmethod is_finished_kdata_timestamp(*timestamp: Timestamp*, *level: IntervalLevel*)

Parameters::
- **timestamp** (*pd.Timestamp*) – the timestamp could be recorded in kdata of the level
- **level** (*zvt.domain.common.IntervalLevel*) –

Returns::

Return type:: bool

list_date

上市日

name

名字

classmethod query_data(*provider_index: int = 0*, *ids: Optional[List[str]] = None*, *entity_ids: Optional[List[str]] = None*, *entity_id: Optional[str] = None*, *codes: Optional[List[str]] = None*, *code: Optional[str] = None*, *level: Optional[Union[IntervalLevel, str]] = None*, *provider: Optional[str] = None*, *columns: Optional[List] = None*, *col_label: Optional[dict] = None*, *return_type: str = 'df'*, *start_timestamp: Optional[Union[Timestamp, str]] = None*, *end_timestamp: Optional[Union[Timestamp, str]] = None*, *filters: Optional[List] = None*, *session: Optional[Session] = None*, *order=None*, *limit: Optional[int] = None*, *distinct=None*, *index: Optional[Union[str, list]] = None*, *drop_index_col=False*, *time_field: str = 'timestamp'*)

query data by the arguments

Parameters::

- **provider_index** –
- **data_schema** –
- **ids** –
- **entity_ids** –
- **entity_id** –
- **codes** –
- **code** –
- **level** –
- **provider** –
- **columns** –
- **col_label** – dict with key(column), value(label
- **return_type** – df, domain or dict. default is df
- **start_timestamp** –
- **end_timestamp** –
- **filters** –
- **session** –
- **order** –
- **limit** –
- **index** – index field name, str for single index, str list for multiple index
- **drop_index_col** – whether drop the col if it's in index, default False
- **time_field** –

Returns:: results basing on return_type.

classmethod record_data(*provider_index: int = 0*, *provider: Optional[str] = None*, *force_update=None*, *sleeping_time=None*, *exchanges=None*, *entity_id=None*, *entity_ids=None*, *code=None*, *codes=None*, *real_time=None*, *fix_duplicate_way=None*, *start_timestamp=None*, *end_timestamp=None*, *one_day_trading_minutes=None*, ***kwargs*)

record data by the arguments

Parameters::

- **entity_id** –
- **provider_index** –
- **provider** –
- **force_update** –
- **sleeping_time** –
- **exchanges** –
- **entity_ids** –
- **code** –
- **codes** –
- **real_time** –
- **fix_duplicate_way** –
- **start_timestamp** –
- **end_timestamp** –
- **one_day_trading_minutes** –
- **kwargs** –

Returns::

classmethod register_provider(*provider*)

register the provider to the schema defined by cls

Parameters:: **provider** –

classmethod register_recorder_cls(*provider*, *recorder_cls*)

register the recorder for the schema

Parameters::

- **provider** –
- **recorder_cls** –

timestamp

the meaning could be different for different case,most time it means 'happen time'

zvt.domain.meta.stockus_meta

Classes

`Stockus`(**kwargs)

Stockus

class zvt.domain.meta.stockus_meta.Stockus(**kwargs)

Bases: **Base**, **TradableEntity**

__init__(**kwargs)

A simple constructor that allows initialization from kwargs.

Sets attributes on the constructed instance using the names and values in `kwargs`.

Only keys that are present as attributes of the instance's class are allowed. These could be, for example, any mapped columns or relationships.

code

编码

classmethod could_short()

whether could be shorted

Returns::

end_date

退市日

entity_id

entity id

entity_type

标的类型

exchange

所属交易所

classmethod get_interval_timestamps(*start_date*, *end_date*, *level: IntervalLevel*)

generate the timestamps for the level

Parameters::
- **start_date** –
- **end_date** –
- **level** –

classmethod get_providers() → List[str]

providers of the schema defined by cls

Returns:: providers

classmethod get_storages(*provider: Optional[str] = None*)

get the storages info

Parameters:: **provider** – provider

Returns:: storages

classmethod get_trading_dates(*start_date=None*, *end_date=None*)

overwrite it to get the trading dates of the entity

Parameters::
- **start_date** –
- **end_date** –

Returns:: list of dates

classmethod get_trading_intervals()

overwrite it to get the trading intervals of the entity

Returns:: list of time intervals, in format [(start,end)]

classmethod get_trading_t()

0 means t+0 1 means t+1

Returns::

id

id

classmethod is_finished_kdata_timestamp(*timestamp: Timestamp*, *level: IntervalLevel*)

Parameters::
- **timestamp** (*pd.Timestamp*) – the timestamp could be recorded in kdata of the level
- **level** (*zvt.domain.common.IntervalLevel*) –

Returns::

Return type:: bool

list_date

上市日

name

名字

classmethod query_data(*provider_index: int = 0*, *ids: Optional[List[str]] = None*, *entity_ids: Optional[List[str]] = None*, *entity_id: Optional[str] = None*, *codes: Optional[List[str]] = None*, *code: Optional[str] = None*, *level: Optional[Union[IntervalLevel, str]] = None*, *provider: Optional[str] = None*, *columns: Optional[List] = None*, *col_label: Optional[dict] = None*, *return_type: str = 'df'*, *start_timestamp: Optional[Union[Timestamp, str]] = None*, *end_timestamp: Optional[Union[Timestamp, str]] = None*, *filters: Optional[List] = None*, *session: Optional[Session] = None*, *order=None*, *limit: Optional[int] = None*, *distinct=None*, *index: Optional[Union[str, list]] = None*, *drop_index_col=False*, *time_field: str = 'timestamp'*)

query data by the arguments

Parameters::

- **provider_index** –
- **data_schema** –
- **ids** –
- **entity_ids** –
- **entity_id** –
- **codes** –
- **code** –
- **level** –
- **provider** –
- **columns** –
- **col_label** – dict with key(column), value(label
- **return_type** – df, domain or dict. default is df
- **start_timestamp** –
- **end_timestamp** –
- **filters** –
- **session** –
- **order** –
- **limit** –
- **index** – index field name, str for single index, str list for multiple index
- **drop_index_col** – whether drop the col if it's in index, default False
- **time_field** –

Returns:: results basing on return_type.

classmethod record_data(*provider_index: int = 0*, *provider: Optional[str] = None*, *force_update=None*, *sleeping_time=None*, *exchanges=None*, *entity_id=None*, *entity_ids=None*, *code=None*, *codes=None*, *real_time=None*, *fix_duplicate_way=None*, *start_timestamp=None*, *end_timestamp=None*, *one_day_trading_minutes=None*, ***kwargs*)

record data by the arguments

Parameters::

- **entity_id** –
- **provider_index** –
- **provider** –
- **force_update** –
- **sleeping_time** –
- **exchanges** –
- **entity_ids** –
- **code** –
- **codes** –
- **real_time** –
- **fix_duplicate_way** –
- **start_timestamp** –
- **end_timestamp** –
- **one_day_trading_minutes** –
- **kwargs** –

Returns::

classmethod register_provider(*provider*)

register the provider to the schema defined by cls

Parameters:: **provider** –

classmethod register_recorder_cls(*provider*, *recorder_cls*)

register the recorder for the schema

Parameters::

- **provider** –
- **recorder_cls** –

timestamp

the meaning could be different for different case,most time it means 'happen time'

zvt.domain.misc

Modules

zvt.domain.misc.holder

---

zvt.domain.misc.money_flow

---

zvt.domain.misc.overall

---

zvt.domain.misc.stock_news

---

zvt.domain.misc.holder

Classes

`HkHolder`(**kwargs)

`InstitutionalInvestorHolder`(**kwargs)

`TopTenHolder`(**kwargs)

`TopTenTradableHolder`(**kwargs)

HkHolder

class zvt.domain.misc.holder.HkHolder(**kwargs*)

Bases: **Base**, **Mixin**

__init__(**kwargs*)

A simple constructor that allows initialization from kwargs.

Sets attributes on the constructed instance using the names and values in `kwargs`.

Only keys that are present as attributes of the instance's class are allowed. These could be, for example, any mapped columns or relationships.

code

股票代码

name

股票名称

holder_code

市场通编码 三种类型：310001-沪股通，310002-深股通，310005-港股通

holder_name

市场通名称 三种类型：沪股通，深股通，港股通

share_number

持股数量

share_ratio

持股比例

entity_id

entity id

classmethod get_providers() → List[str]

providers of the schema defined by cls

Returns:: providers

classmethod get_storages(*provider: Optional[str] = None*)

get the storages info

Parameters:: **provider** – provider

Returns:: storages

id

id

classmethod query_data(*provider_index: int = 0*, *ids: Optional[List[str]] = None*, *entity_ids: Optional[List[str]] = None*, *entity_id: Optional[str] = None*, *codes: Optional[List[str]] = None*, *code: Optional[str] = None*, *level: Optional[Union[IntervalLevel, str]] = None*, *provider: Optional[str] = None*, *columns: Optional[List] = None*, *col_label: Optional[dict] = None*, *return_type: str = 'df'*, *start_timestamp: Optional[Union[Timestamp, str]] = None*, *end_timestamp: Optional[Union[Timestamp, str]] = None*, *filters: Optional[List] = None*, *session: Optional[Session] = None*, *order=None*, *limit: Optional[int] = None*, *distinct=None*, *index: Optional[Union[str, list]] = None*, *drop_index_col=False*, *time_field: str = 'timestamp'*)

query data by the arguments

Parameters::

- **provider_index** –
- **data_schema** –
- **ids** –
- **entity_ids** –
- **entity_id** –
- **codes** –
- **code** –
- **level** –
- **provider** –
- **columns** –
- **col_label** – dict with key(column), value(label
- **return_type** – df, domain or dict. default is df
- **start_timestamp** –
- **end_timestamp** –
- **filters** –
- **session** –
- **order** –
- **limit** –
- **index** – index field name, str for single index, str list for multiple index
- **drop_index_col** – whether drop the col if it's in index, default False
- **time_field** –

Returns:: results basing on return_type.

classmethod record_data(*provider_index: int = 0*, *provider: Optional[str] = None*, *force_update=None*, *sleeping_time=None*, *exchanges=None*, *entity_id=None*, *entity_ids=None*, *code=None*, *codes=None*, *real_time=None*, *fix_duplicate_way=None*, *start_timestamp=None*, *end_timestamp=None*, *one_day_trading_minutes=None*, ***kwargs*)

record data by the arguments

Parameters::

- **entity_id** –
- **provider_index** –
- **provider** –
- **force_update** –
- **sleeping_time** –
- **exchanges** –
- **entity_ids** –
- **code** –
- **codes** –
- **real_time** –
- **fix_duplicate_way** –
- **start_timestamp** –
- **end_timestamp** –
- **one_day_trading_minutes** –
- **kwargs** –

Returns::

classmethod register_provider(*provider*)

register the provider to the schema defined by cls

Parameters:: **provider** –

classmethod register_recorder_cls(*provider*, *recorder_cls*)

register the recorder for the schema

Parameters::

- **provider** –
- **recorder_cls** –

timestamp

the meaning could be different for different case,most time it means 'happen time'

InstitutionalInvestorHolder

class zvt.domain.misc.holder.InstitutionalInvestorHolder(**kwargs)

Bases: **Base**, **Mixin**

__init__(**kwargs)

A simple constructor that allows initialization from kwargs.

Sets attributes on the constructed instance using the names and values in `kwargs`.

Only keys that are present as attributes of the instance's class are allowed. These could be, for example, any mapped columns or relationships.

institutional_investor_type

机构类型

holder_code

股东代码

holder_name

股东名称

shareholding_numbers

持股数

shareholding_ratio

持股比例

entity_id

entity id

classmethod get_providers() → List[str]

providers of the schema defined by cls

Returns:: providers

classmethod get_storages(*provider: Optional[str] = None*)

get the storages info

Parameters:: **provider** – provider
Returns:: storages

id

id

classmethod query_data(*provider_index: int = 0*, *ids: Optional[List[str]] = None*, *entity_ids: Optional[List[str]] = None*, *entity_id: Optional[str] = None*, *codes: Optional[List[str]] = None*, *code: Optional[str] = None*, *level: Optional[Union[IntervalLevel, str]] = None*, *provider: Optional[str] = None*, *columns: Optional[List] = None*, *col_label: Optional[dict] = None*, *return_type: str = 'df'*, *start_timestamp: Optional[Union[Timestamp, str]] = None*, *end_timestamp: Optional[Union[Timestamp, str]] = None*, *filters: Optional[List] = None*, *session: Optional[Session] = None*, *order=None*, *limit: Optional[int] = None*, *distinct=None*, *index: Optional[Union[str, list]] = None*, *drop_index_col=False*, *time_field: str = 'timestamp'*)

query data by the arguments

Parameters::

- **provider_index** –
- **data_schema** –
- **ids** –
- **entity_ids** –
- **entity_id** –
- **codes** –
- **code** –
- **level** –
- **provider** –
- **columns** –
- **col_label** – dict with key(column), value(label
- **return_type** – df, domain or dict. default is df
- **start_timestamp** –
- **end_timestamp** –
- **filters** –
- **session** –
- **order** –
- **limit** –
- **index** – index field name, str for single index, str list for multiple index
- **drop_index_col** – whether drop the col if it's in index, default False
- **time_field** –

Returns:: results basing on return_type.

classmethod record_data(*provider_index: int = 0*, *provider: Optional[str] = None*, *force_update=None*, *sleeping_time=None*, *exchanges=None*, *entity_id=None*, *entity_ids=None*, *code=None*, *codes=None*, *real_time=None*, *fix_duplicate_way=None*, *start_timestamp=None*, *end_timestamp=None*, *one_day_trading_minutes=None*, ***kwargs*)

record data by the arguments

Parameters::

- **entity_id** –
- **provider_index** –
- **provider** –
- **force_update** –
- **sleeping_time** –
- **exchanges** –
- **entity_ids** –
- **code** –
- **codes** –
- **real_time** –
- **fix_duplicate_way** –
- **start_timestamp** –
- **end_timestamp** –
- **one_day_trading_minutes** –
- **kwargs** –

Returns::

classmethod register_provider(*provider*)

register the provider to the schema defined by cls

Parameters:: **provider** –

classmethod register_recorder_cls(*provider*, *recorder_cls*)

register the recorder for the schema

Parameters::

- **provider** –
- **recorder_cls** –

timestamp

the meaning could be different for different case,most time it means 'happen time'

TopTenHolder

class zvt.domain.misc.holder.TopTenHolder(**kwargs)

Bases: **Base**, **Mixin**

__init__(**kwargs)

A simple constructor that allows initialization from kwargs.

Sets attributes on the constructed instance using the names and values in `kwargs`.

Only keys that are present as attributes of the instance's class are allowed. These could be, for example, any mapped columns or relationships.

holder_code

股东代码

holder_name

股东名称

shareholding_numbers

持股数

shareholding_ratio

持股比例

change

变动

change_ratio

变动比例

entity_id

entity id

classmethod get_providers() → List[str]

providers of the schema defined by cls

Returns:: providers

classmethod get_storages(*provider: Optional[str] = None*)

get the storages info

Parameters:: **provider** – provider

Returns:: storages

id

id

classmethod query_data(*provider_index: int = 0*, *ids: Optional[List[str]] = None*, *entity_ids: Optional[List[str]] = None*, *entity_id: Optional[str] = None*, *codes: Optional[List[str]] = None*, *code: Optional[str] = None*, *level: Optional[Union[IntervalLevel, str]] = None*, *provider: Optional[str] = None*, *columns: Optional[List] = None*, *col_label: Optional[dict] = None*, *return_type: str = 'df'*, *start_timestamp: Optional[Union[Timestamp, str]] = None*, *end_timestamp: Optional[Union[Timestamp, str]] = None*, *filters: Optional[List] = None*, *session: Optional[Session] = None*, *order=None*, *limit: Optional[int] = None*, *distinct=None*, *index: Optional[Union[str, list]] = None*, *drop_index_col=False*, *time_field: str = 'timestamp'*)

query data by the arguments

Parameters::

- **provider_index** –
- **data_schema** –
- **ids** –
- **entity_ids** –
- **entity_id** –
- **codes** –
- **code** –
- **level** –
- **provider** –
- **columns** –
- **col_label** – dict with key(column), value(label
- **return_type** – df, domain or dict. default is df
- **start_timestamp** –
- **end_timestamp** –
- **filters** –
- **session** –
- **order** –
- **limit** –
- **index** – index field name, str for single index, str list for multiple index
- **drop_index_col** – whether drop the col if it's in index, default False
- **time_field** –

Returns:: results basing on return_type.

classmethod record_data(*provider_index: int = 0*, *provider: Optional[str] = None*, *force_update=None*, *sleeping_time=None*, *exchanges=None*, *entity_id=None*, *entity_ids=None*, *code=None*, *codes=None*, *real_time=None*, *fix_duplicate_way=None*, *start_timestamp=None*, *end_timestamp=None*, *one_day_trading_minutes=None*, ***kwargs*)

record data by the arguments

Parameters::

- **entity_id** –
- **provider_index** –
- **provider** –
- **force_update** –
- **sleeping_time** –
- **exchanges** –
- **entity_ids** –
- **code** –
- **codes** –
- **real_time** –
- **fix_duplicate_way** –
- **start_timestamp** –
- **end_timestamp** –
- **one_day_trading_minutes** –
- **kwargs** –

Returns::

classmethod register_provider(*provider*)

register the provider to the schema defined by cls

Parameters:: **provider** –

classmethod register_recorder_cls(*provider*, *recorder_cls*)

register the recorder for the schema

Parameters::

- **provider** –
- **recorder_cls** –

timestamp

the meaning could be different for different case,most time it means 'happen time'

TopTenTradableHolder

class zvt.domain.misc.holder.TopTenTradableHolder(***kwargs*)

Bases: **`Base`**, **`Mixin`**

__init__(***kwargs*)

A simple constructor that allows initialization from kwargs.

Sets attributes on the constructed instance using the names and values in `kwargs`.

Only keys that are present as attributes of the instance's class are allowed. These could be, for example, any mapped columns or relationships.

holder_code

股东代码

holder_name

股东名称

shareholding_numbers

持股数

shareholding_ratio

持股比例

change

变动

change_ratio

变动比例

entity_id

entity id

classmethod get_providers() → List[str]

providers of the schema defined by cls

Returns:: providers

classmethod get_storages(*provider: Optional[str] = None*)

get the storages info

Parameters:: **provider** – provider

Returns:: storages

id

id

classmethod query_data(*provider_index: int = 0*, *ids: Optional[List[str]] = None*, *entity_ids: Optional[List[str]] = None*, *entity_id: Optional[str] = None*, *codes: Optional[List[str]] = None*, *code: Optional[str] = None*, *level: Optional[Union[IntervalLevel, str]] = None*, *provider: Optional[str] = None*, *columns: Optional[List] = None*, *col_label: Optional[dict] = None*, *return_type: str = 'df'*, *start_timestamp: Optional[Union[Timestamp, str]] = None*, *end_timestamp: Optional[Union[Timestamp, str]] = None*, *filters: Optional[List] = None*, *session: Optional[Session] = None*, *order=None*, *limit: Optional[int] = None*, *distinct=None*, *index: Optional[Union[str, list]] = None*, *drop_index_col=False*, *time_field: str = 'timestamp'*)

query data by the arguments

Parameters::

- **provider_index** –
- **data_schema** –
- **ids** –
- **entity_ids** –
- **entity_id** –
- **codes** –
- **code** –
- **level** –
- **provider** –
- **columns** –
- **col_label** – dict with key(column), value(label
- **return_type** – df, domain or dict. default is df
- **start_timestamp** –
- **end_timestamp** –
- **filters** –
- **session** –
- **order** –
- **limit** –
- **index** – index field name, str for single index, str list for multiple index
- **drop_index_col** – whether drop the col if it's in index, default False
- **time_field** –

Returns:: results basing on return_type.

classmethod record_data(*provider_index: int = 0*, *provider: Optional[str] = None*, *force_update=None*, *sleeping_time=None*, *exchanges=None*, *entity_id=None*, *entity_ids=None*, *code=None*, *codes=None*, *real_time=None*, *fix_duplicate_way=None*, *start_timestamp=None*, *end_timestamp=None*, *one_day_trading_minutes=None*, ***kwargs*)

record data by the arguments

Parameters::

- **entity_id** –
- **provider_index** –
- **provider** –
- **force_update** –
- **sleeping_time** –
- **exchanges** –
- **entity_ids** –
- **code** –
- **codes** –
- **real_time** –
- **fix_duplicate_way** –
- **start_timestamp** –
- **end_timestamp** –
- **one_day_trading_minutes** –
- **kwargs** –

Returns::

classmethod register_provider(*provider*)

register the provider to the schema defined by cls

Parameters:: **provider** –

classmethod register_recorder_cls(*provider*, *recorder_cls*)

register the recorder for the schema

Parameters::

- **provider** –
- **recorder_cls** –

timestamp

the meaning could be different for different case,most time it means 'happen time'

zvt.domain.misc.money_flow

Classes

`BlockMoneyFlow`(**kwargs)

`IndexMoneyFlow`(**kwargs)

`StockMoneyFlow`(**kwargs)

BlockMoneyFlow

class zvt.domain.misc.money_flow.BlockMoneyFlow(**kwargs)

Bases: **Base**, **Mixin**

__init__(**kwargs)

A simple constructor that allows initialization from kwargs.

Sets attributes on the constructed instance using the names and values in `kwargs`.

Only keys that are present as attributes of the instance's class are allowed. These could be, for example, any mapped columns or relationships.

close

收盘价

net_inflows

净流入

net_inflow_rate

净流入率

net_main_inflows

主力=超大单+大单

net_huge_inflows

超大单

net_big_inflows

大单

net_medium_inflows

中单

net_small_inflows

小单

entity_id

entity id

classmethod get_providers() → List[str]

providers of the schema defined by cls

Returns:: providers

classmethod get_storages(*provider: Optional[str] = None*)

get the storages info

Parameters:: **provider** – provider

Returns:: storages

id

id

classmethod query_data(*provider_index: int = 0*, *ids: Optional[List[str]] = None*, *entity_ids: Optional[List[str]] = None*, *entity_id: Optional[str] = None*, *codes: Optional[List[str]] = None*, *code: Optional[str] = None*, *level: Optional[Union[IntervalLevel, str]] = None*, *provider: Optional[str] = None*, *columns: Optional[List] = None*, *col_label: Optional[dict] = None*, *return_type: str = 'df'*, *start_timestamp: Optional[Union[Timestamp, str]] = None*, *end_timestamp: Optional[Union[Timestamp, str]] = None*, *filters: Optional[List] = None*, *session: Optional[Session] = None*, *order=None*, *limit: Optional[int] = None*, *distinct=None*, *index: Optional[Union[str, list]] = None*, *drop_index_col=False*, *time_field: str = 'timestamp'*)

query data by the arguments

Parameters::

- **provider_index** –
- **data_schema** –
- **ids** –
- **entity_ids** –
- **entity_id** –
- **codes** –
- **code** –
- **level** –
- **provider** –
- **columns** –
- **col_label** – dict with key(column), value(label
- **return_type** – df, domain or dict. default is df
- **start_timestamp** –
- **end_timestamp** –
- **filters** –
- **session** –
- **order** –
- **limit** –
- **index** – index field name, str for single index, str list for multiple index
- **drop_index_col** – whether drop the col if it's in index, default False
- **time_field** –

Returns:: results basing on return_type.

classmethod record_data(*provider_index: int = 0*, *provider: Optional[str] = None*, *force_update=None*, *sleeping_time=None*, *exchanges=None*, *entity_id=None*, *entity_ids=None*, *code=None*, *codes=None*, *real_time=None*, *fix_duplicate_way=None*, *start_timestamp=None*, *end_timestamp=None*, *one_day_trading_minutes=None*, ***kwargs*)

record data by the arguments

Parameters::

- **entity_id** –
- **provider_index** –
- **provider** –
- **force_update** –
- **sleeping_time** –
- **exchanges** –
- **entity_ids** –
- **code** –
- **codes** –
- **real_time** –
- **fix_duplicate_way** –
- **start_timestamp** –
- **end_timestamp** –
- **one_day_trading_minutes** –
- **kwargs** –

Returns::

classmethod register_provider(*provider*)

register the provider to the schema defined by cls

Parameters:: **provider** –

classmethod register_recorder_cls(*provider*, *recorder_cls*)

register the recorder for the schema

Parameters::

- **provider** –
- **recorder_cls** –

timestamp

the meaning could be different for different case,most time it means 'happen time'

IndexMoneyFlow

class zvt.domain.misc.money_flow.IndexMoneyFlow(**kwargs)

Bases: **Base**, **Mixin**

__init__(**kwargs)

A simple constructor that allows initialization from kwargs.

Sets attributes on the constructed instance using the names and values in `kwargs`.

Only keys that are present as attributes of the instance's class are allowed. These could be, for example, any mapped columns or relationships.

net_inflows

净流入

net_inflow_rate

净流入率

net_main_inflows

主力=超大单+大单

net_huge_inflows

超大单

net_big_inflows

大单

net_medium_inflows

中单

net_small_inflows

小单

entity_id

entity id

classmethod get_providers() → List[str]

providers of the schema defined by cls

Returns:: providers

classmethod get_storages(*provider: Optional[str] = None*)

get the storages info

Parameters:: **provider** – provider

Returns:: storages

id

id

classmethod query_data(*provider_index: int = 0*, *ids: Optional[List[str]] = None*, *entity_ids: Optional[List[str]] = None*, *entity_id: Optional[str] = None*, *codes: Optional[List[str]] = None*, *code: Optional[str] = None*, *level: Optional[Union[IntervalLevel, str]] = None*, *provider: Optional[str] = None*, *columns: Optional[List] = None*, *col_label: Optional[dict] = None*, *return_type: str = 'df'*, *start_timestamp: Optional[Union[Timestamp, str]] = None*, *end_timestamp: Optional[Union[Timestamp, str]] = None*, *filters: Optional[List] = None*, *session: Optional[Session] = None*, *order=None*, *limit: Optional[int] = None*, *distinct=None*, *index: Optional[Union[str, list]] = None*, *drop_index_col=False*, *time_field: str = 'timestamp'*)

query data by the arguments

Parameters::

- **provider_index** –
- **data_schema** –
- **ids** –
- **entity_ids** –
- **entity_id** –
- **codes** –
- **code** –
- **level** –
- **provider** –
- **columns** –
- **col_label** – dict with key(column), value(label
- **return_type** – df, domain or dict. default is df
- **start_timestamp** –
- **end_timestamp** –
- **filters** –
- **session** –
- **order** –
- **limit** –
- **index** – index field name, str for single index, str list for multiple index
- **drop_index_col** – whether drop the col if it's in index, default False
- **time_field** –

Returns:: results basing on return_type.

classmethod record_data(*provider_index: int = 0*, *provider: Optional[str] = None*, *force_update=None*, *sleeping_time=None*, *exchanges=None*, *entity_id=None*, *entity_ids=None*, *code=None*, *codes=None*, *real_time=None*, *fix_duplicate_way=None*, *start_timestamp=None*, *end_timestamp=None*, *one_day_trading_minutes=None*, ***kwargs*)

record data by the arguments

Parameters::

- **entity_id** –
- **provider_index** –
- **provider** –
- **force_update** –
- **sleeping_time** –
- **exchanges** –
- **entity_ids** –
- **code** –
- **codes** –
- **real_time** –
- **fix_duplicate_way** –
- **start_timestamp** –
- **end_timestamp** –
- **one_day_trading_minutes** –
- **kwargs** –

Returns::

classmethod register_provider(*provider*)

register the provider to the schema defined by cls

Parameters:: **provider** –

classmethod register_recorder_cls(*provider*, *recorder_cls*)

register the recorder for the schema

Parameters::

- **provider** –
- **recorder_cls** –

timestamp

the meaning could be different for different case,most time it means 'happen time'

StockMoneyFlow

class zvt.domain.misc.money_flow.StockMoneyFlow(**kwargs)

Bases: **Base**, **Mixin**

__init__(**kwargs)

A simple constructor that allows initialization from kwargs.

Sets attributes on the constructed instance using the names and values in `kwargs`.

Only keys that are present as attributes of the instance's class are allowed. These could be, for example, any mapped columns or relationships.

close

收盘价

net_inflows

净流入

net_inflow_rate

净流入率

net_main_inflows

主力=超大单+大单

net_huge_inflows

超大单

net_big_inflows

大单

net_medium_inflows

中单

net_small_inflows

小单

entity_id

entity id

classmethod get_providers() → List[str]

providers of the schema defined by cls

Returns:: providers

classmethod get_storages(*provider: Optional[str] = None*)

get the storages info

Parameters:: **provider** – provider

Returns:: storages

id

id

classmethod query_data(*provider_index: int = 0*, *ids: Optional[List[str]] = None*, *entity_ids: Optional[List[str]] = None*, *entity_id: Optional[str] = None*, *codes: Optional[List[str]] = None*, *code: Optional[str] = None*, *level: Optional[Union[IntervalLevel, str]] = None*, *provider: Optional[str] = None*, *columns: Optional[List] = None*, *col_label: Optional[dict] = None*, *return_type: str = 'df'*, *start_timestamp: Optional[Union[Timestamp, str]] = None*, *end_timestamp: Optional[Union[Timestamp, str]] = None*, *filters: Optional[List] = None*, *session: Optional[Session] = None*, *order=None*, *limit: Optional[int] = None*, *distinct=None*, *index: Optional[Union[str, list]] = None*, *drop_index_col=False*, *time_field: str = 'timestamp'*)

query data by the arguments

Parameters::

- **provider_index** –
- **data_schema** –
- **ids** –
- **entity_ids** –
- **entity_id** –
- **codes** –
- **code** –
- **level** –
- **provider** –
- **columns** –
- **col_label** – dict with key(column), value(label
- **return_type** – df, domain or dict. default is df
- **start_timestamp** –
- **end_timestamp** –
- **filters** –
- **session** –
- **order** –
- **limit** –
- **index** – index field name, str for single index, str list for multiple index
- **drop_index_col** – whether drop the col if it's in index, default False
- **time_field** –

Returns:: results basing on return_type.

classmethod record_data(*provider_index: int = 0*, *provider: Optional[str] = None*, *force_update=None*, *sleeping_time=None*, *exchanges=None*, *entity_id=None*, *entity_ids=None*, *code=None*, *codes=None*, *real_time=None*, *fix_duplicate_way=None*, *start_timestamp=None*, *end_timestamp=None*, *one_day_trading_minutes=None*, ***kwargs*)

record data by the arguments

Parameters::

- **entity_id** –
- **provider_index** –
- **provider** –
- **force_update** –
- **sleeping_time** –
- **exchanges** –
- **entity_ids** –
- **code** –
- **codes** –
- **real_time** –
- **fix_duplicate_way** –
- **start_timestamp** –
- **end_timestamp** –
- **one_day_trading_minutes** –
- **kwargs** –

Returns::

classmethod register_provider(*provider*)

register the provider to the schema defined by cls

Parameters:: **provider** –

classmethod register_recorder_cls(*provider*, *recorder_cls*)

register the recorder for the schema

Parameters::

- **provider** –
- **recorder_cls** –

timestamp

the meaning could be different for different case,most time it means 'happen time'

zvt.domain.misc.overall

Classes

`CrossMarketSummary`(**kwargs)

`MarginTradingSummary`(**kwargs)

`StockSummary`(**kwargs)

CrossMarketSummary

class zvt.domain.misc.overall.CrossMarketSummary(***kwargs*)

Bases: **Base**, **Mixin**

__init__(***kwargs*)

A simple constructor that allows initialization from kwargs.

Sets attributes on the constructed instance using the names and values in `kwargs`.

Only keys that are present as attributes of the instance's class are allowed. These could be, for example, any mapped columns or relationships.

entity_id

entity id

classmethod get_providers() → List[str]

providers of the schema defined by cls

Returns:: providers

classmethod get_storages(*provider: Optional[str] = None*)

get the storages info

Parameters:: **provider** – provider
Returns:: storages

id

id

classmethod query_data(*provider_index: int = 0*, *ids: Optional[List[str]] = None*, *entity_ids: Optional[List[str]] = None*, *entity_id: Optional[str] = None*, *codes: Optional[List[str]] = None*, *code: Optional[str] = None*, *level: Optional[Union[IntervalLevel, str]] = None*, *provider:*

Optional[str] = None, *columns: Optional[List] = None*, *col_label: Optional[dict] = None*, *return_type: str = 'df'*, *start_timestamp: Optional[Union[Timestamp, str]] = None*, *end_timestamp: Optional[Union[Timestamp, str]] = None*, *filters: Optional[List] = None*, *session: Optional[Session] = None*, *order=None*, *limit: Optional[int] = None*, *distinct=None*, *index: Optional[Union[str, list]] = None*, *drop_index_col=False*, *time_field: str = 'timestamp'*)

query data by the arguments

Parameters::

- **provider_index** –
- **data_schema** –
- **ids** –
- **entity_ids** –
- **entity_id** –
- **codes** –
- **code** –
- **level** –
- **provider** –
- **columns** –
- **col_label** – dict with key(column), value(label
- **return_type** – df, domain or dict. default is df
- **start_timestamp** –
- **end_timestamp** –
- **filters** –
- **session** –
- **order** –
- **limit** –
- **index** – index field name, str for single index, str list for multiple index
- **drop_index_col** – whether drop the col if it's in index, default False
- **time_field** –

Returns:: results basing on return_type.

classmethod record_data(*provider_index: int = 0*, *provider: Optional[str] = None*, *force_update=None*, *sleeping_time=None*, *exchanges=None*, *entity_id=None*, *entity_ids=None*, *code=None*, *codes=None*,

real_time=None, *fix_duplicate_way=None*, *start_timestamp=None*, *end_timestamp=None*, *one_day_trading_minutes=None*, ***kwargs*)

record data by the arguments

Parameters::

- **entity_id** –
- **provider_index** –
- **provider** –
- **force_update** –
- **sleeping_time** –
- **exchanges** –
- **entity_ids** –
- **code** –
- **codes** –
- **real_time** –
- **fix_duplicate_way** –
- **start_timestamp** –
- **end_timestamp** –
- **one_day_trading_minutes** –
- **kwargs** –

Returns::

classmethod register_provider(*provider*)

register the provider to the schema defined by cls

Parameters:: **provider** –

classmethod register_recorder_cls(*provider*, *recorder_cls*)

register the recorder for the schema

Parameters::

- **provider** –
- **recorder_cls** –

timestamp

the meaning could be different for different case,most time it means 'happen time'

MarginTradingSummary

class zvt.domain.misc.overall.MarginTradingSummary(**kwargs)

Bases: **Base**, **Mixin**

__init__(**kwargs)

A simple constructor that allows initialization from kwargs.

Sets attributes on the constructed instance using the names and values in `kwargs`.

Only keys that are present as attributes of the instance's class are allowed. These could be, for example, any mapped columns or relationships.

margin_value

融资余额

margin_buy

买入额

short_value

融券余额

short_volume

卖出量

total_value

融资融券余额

entity_id

entity id

classmethod get_providers() → List[str]

providers of the schema defined by cls

Returns:: providers

classmethod get_storages(*provider: Optional[str] = None*)

get the storages info

Parameters:: **provider** – provider
Returns:: storages

id

id

classmethod query_data(*provider_index: int = 0*, *ids: Optional[List[str]] = None*, *entity_ids: Optional[List[str]] = None*, *entity_id: Optional[str] = None*, *codes: Optional[List[str]] = None*, *code: Optional[str] = None*, *level: Optional[Union[IntervalLevel, str]] = None*, *provider: Optional[str] = None*, *columns: Optional[List] = None*, *col_label: Optional[dict] = None*, *return_type: str = 'df'*, *start_timestamp: Optional[Union[Timestamp, str]] = None*, *end_timestamp: Optional[Union[Timestamp, str]] = None*, *filters: Optional[List] = None*, *session: Optional[Session] = None*, *order=None*, *limit: Optional[int] = None*, *distinct=None*, *index: Optional[Union[str, list]] = None*, *drop_index_col=False*, *time_field: str = 'timestamp'*)

query data by the arguments

Parameters::

- **provider_index** –
- **data_schema** –
- **ids** –
- **entity_ids** –
- **entity_id** –
- **codes** –
- **code** –
- **level** –
- **provider** –
- **columns** –
- **col_label** – dict with key(column), value(label
- **return_type** – df, domain or dict. default is df
- **start_timestamp** –
- **end_timestamp** –
- **filters** –
- **session** –
- **order** –
- **limit** –
- **index** – index field name, str for single index, str list for multiple index
- **drop_index_col** – whether drop the col if it's in index, default False
- **time_field** –

Returns:: results basing on return_type.

classmethod record_data(*provider_index: int = 0*, *provider: Optional[str] = None*, *force_update=None*, *sleeping_time=None*, *exchanges=None*, *entity_id=None*, *entity_ids=None*, *code=None*, *codes=None*, *real_time=None*, *fix_duplicate_way=None*, *start_timestamp=None*, *end_timestamp=None*, *one_day_trading_minutes=None*, ***kwargs*)

record data by the arguments

Parameters::

- **entity_id** –
- **provider_index** –
- **provider** –
- **force_update** –
- **sleeping_time** –
- **exchanges** –
- **entity_ids** –
- **code** –
- **codes** –
- **real_time** –
- **fix_duplicate_way** –
- **start_timestamp** –
- **end_timestamp** –
- **one_day_trading_minutes** –
- **kwargs** –

Returns::

classmethod register_provider(*provider*)

register the provider to the schema defined by cls

Parameters:: **provider** –

classmethod register_recorder_cls(*provider*, *recorder_cls*)

register the recorder for the schema

Parameters::

- **provider** –
- **recorder_cls** –

timestamp

the meaning could be different for different case,most time it means 'happen time'

StockSummary

class zvt.domain.misc.overall.StockSummary(***kwargs*)

Bases: **Base**, **Mixin**

__init__(***kwargs*)

A simple constructor that allows initialization from kwargs.

Sets attributes on the constructed instance using the names and values in `kwargs`.

Only keys that are present as attributes of the instance's class are allowed. These could be, for example, any mapped columns or relationships.

entity_id

entity id

classmethod get_providers() → List[str]

providers of the schema defined by cls

Returns:: providers

classmethod get_storages(*provider: Optional[str] = None*)

get the storages info

Parameters:: **provider** – provider
Returns:: storages

id

id

classmethod query_data(*provider_index: int = 0*, *ids: Optional[List[str]] = None*, *entity_ids: Optional[List[str]] = None*, *entity_id: Optional[str] = None*, *codes: Optional[List[str]] = None*, *code: Optional[str] = None*, *level: Optional[Union[IntervalLevel, str]] = None*, *provider:*

Optional[str] = None, *columns: Optional[List] = None*, *col_label: Optional[dict] = None*, *return_type: str = 'df'*, *start_timestamp: Optional[Union[Timestamp, str]] = None*, *end_timestamp: Optional[Union[Timestamp, str]] = None*, *filters: Optional[List] = None*, *session: Optional[Session] = None*, *order=None*, *limit: Optional[int] = None*, *distinct=None*, *index: Optional[Union[str, list]] = None*, *drop_index_col=False*, *time_field: str = 'timestamp'*)

query data by the arguments

Parameters::

- **provider_index** –
- **data_schema** –
- **ids** –
- **entity_ids** –
- **entity_id** –
- **codes** –
- **code** –
- **level** –
- **provider** –
- **columns** –
- **col_label** – dict with key(column), value(label
- **return_type** – df, domain or dict. default is df
- **start_timestamp** –
- **end_timestamp** –
- **filters** –
- **session** –
- **order** –
- **limit** –
- **index** – index field name, str for single index, str list for multiple index
- **drop_index_col** – whether drop the col if it's in index, default False
- **time_field** –

Returns:: results basing on return_type.

classmethod record_data(*provider_index: int = 0*, *provider: Optional[str] = None*, *force_update=None*, *sleeping_time=None*, *exchanges=None*, *entity_id=None*, *entity_ids=None*, *code=None*, *codes=None*,

real_time=None, *fix_duplicate_way=None*, *start_timestamp=None*, *end_timestamp=None*, *one_day_trading_minutes=None*, ***kwargs*)

record data by the arguments

Parameters::

- **entity_id** –
- **provider_index** –
- **provider** –
- **force_update** –
- **sleeping_time** –
- **exchanges** –
- **entity_ids** –
- **code** –
- **codes** –
- **real_time** –
- **fix_duplicate_way** –
- **start_timestamp** –
- **end_timestamp** –
- **one_day_trading_minutes** –
- **kwargs** –

Returns::

classmethod register_provider(*provider*)

register the provider to the schema defined by cls

Parameters:: **provider** –

classmethod register_recorder_cls(*provider*, *recorder_cls*)

register the recorder for the schema

Parameters::

- **provider** –
- **recorder_cls** –

timestamp

the meaning could be different for different case,most time it means 'happen time'

zvt.domain.misc.stock_news

Classes

StockNews(**kwargs)

StockNews

class zvt.domain.misc.stock_news.StockNews(**kwargs)

Bases: **Base**, **Mixin**

__init__(**kwargs)

A simple constructor that allows initialization from kwargs.

Sets attributes on the constructed instance using the names and values in `kwargs`.

Only keys that are present as attributes of the instance's class are allowed. These could be, for example, any mapped columns or relationships.

news_code

新闻编号

news_url

新闻地址

news_title

新闻标题

news_content

新闻内容

news_analysis

新闻解读

entity_id

entity id

classmethod get_providers() → List[str]

providers of the schema defined by cls

Returns:: providers

classmethod get_storages(*provider: Optional[str] = None*)

get the storages info

Parameters:: **provider** – provider
Returns:: storages

id

id

classmethod query_data(*provider_index: int = 0*, *ids: Optional[List[str]] = None*, *entity_ids: Optional[List[str]] = None*, *entity_id: Optional[str] = None*, *codes: Optional[List[str]] = None*, *code: Optional[str] = None*, *level: Optional[Union[IntervalLevel, str]] = None*, *provider: Optional[str] = None*, *columns: Optional[List] = None*, *col_label: Optional[dict] = None*, *return_type: str = 'df'*, *start_timestamp: Optional[Union[Timestamp, str]] = None*, *end_timestamp: Optional[Union[Timestamp, str]] = None*, *filters: Optional[List] = None*, *session: Optional[Session] = None*, *order=None*, *limit: Optional[int] = None*, *distinct=None*, *index: Optional[Union[str, list]] = None*, *drop_index_col=False*, *time_field: str = 'timestamp'*)

query data by the arguments

Parameters::

- **provider_index** –
- **data_schema** –
- **ids** –
- **entity_ids** –
- **entity_id** –
- **codes** –
- **code** –
- **level** –
- **provider** –
- **columns** –
- **col_label** – dict with key(column), value(label
- **return_type** – df, domain or dict. default is df
- **start_timestamp** –
- **end_timestamp** –
- **filters** –
- **session** –
- **order** –
- **limit** –
- **index** – index field name, str for single index, str list for multiple index
- **drop_index_col** – whether drop the col if it's in index, default False
- **time_field** –

Returns:: results basing on return_type.

classmethod record_data(*provider_index: int = 0*, *provider: Optional[str] = None*, *force_update=None*, *sleeping_time=None*, *exchanges=None*, *entity_id=None*, *entity_ids=None*, *code=None*, *codes=None*, *real_time=None*, *fix_duplicate_way=None*, *start_timestamp=None*, *end_timestamp=None*, *one_day_trading_minutes=None*, ***kwargs*)

record data by the arguments

Parameters::

- **entity_id** –
- **provider_index** –
- **provider** –
- **force_update** –
- **sleeping_time** –
- **exchanges** –
- **entity_ids** –
- **code** –
- **codes** –
- **real_time** –
- **fix_duplicate_way** –
- **start_timestamp** –
- **end_timestamp** –
- **one_day_trading_minutes** –
- **kwargs** –

Returns::

classmethod register_provider(*provider*)

register the provider to the schema defined by cls

Parameters:: **provider** –

classmethod register_recorder_cls(*provider*, *recorder_cls*)

register the recorder for the schema

Parameters::

- **provider** –
- **recorder_cls** –

timestamp

the meaning could be different for different case,most time it means 'happen time'

zvt.domain.quotes

Classes

BlockKdataCommon()

CurrencyKdataCommon()

EtfKdataCommon()

FutureKdataCommon()

IndexKdataCommon()

IndexusKdataCommon()

KdataCommon()

StockKdataCommon()

StockhkKdataCommon()

StockusKdataCommon()

TickCommon()

Modules

zvt.domain.quotes.block

zvt.domain.quotes.currency

zvt.domain.quotes.etf

zvt.domain.quotes.future

zvt.domain.quotes.index

zvt.domain.quotes.indexus

zvt.domain.quotes.stock

zvt.domain.quotes.stockhk

zvt.domain.quotes.stockus

zvt.domain.quotes.trade_day

BlockKdataCommon

class zvt.domain.quotes.BlockKdataCommon

Bases: `KdataCommon`

__init__()

entity_id = *Column(None, String(), table=None)*

entity id

classmethod get_providers() → List[str]

providers of the schema defined by cls

Returns:: providers

classmethod get_storages(*provider: Optional[str] = None*)

get the storages info

Parameters:: **provider** – provider

Returns:: storages

id = *Column(None, String(), table=None, primary_key=True, nullable=False)*

id

classmethod query_data(*provider_index: int = 0*, *ids: Optional[List[str]] = None*, *entity_ids: Optional[List[str]] = None*, *entity_id: Optional[str] = None*, *codes: Optional[List[str]] = None*, *code: Optional[str] = None*, *level: Optional[Union[IntervalLevel, str]] = None*, *provider: Optional[str] = None*, *columns: Optional[List] = None*, *col_label: Optional[dict] = None*, *return_type: str = 'df'*, *start_timestamp: Optional[Union[Timestamp, str]] = None*, *end_timestamp: Optional[Union[Timestamp, str]] = None*, *filters: Optional[List] = None*, *session: Optional[Session] = None*, *order=None*, *limit: Optional[int] = None*, *distinct=None*, *index: Optional[Union[str, list]] = None*, *drop_index_col=False*, *time_field: str = 'timestamp'*)

query data by the arguments

Parameters::

- **provider_index** –
- **data_schema** –
- **ids** –
- **entity_ids** –
- **entity_id** –
- **codes** –
- **code** –
- **level** –
- **provider** –
- **columns** –
- **col_label** – dict with key(column), value(label
- **return_type** – df, domain or dict. default is df
- **start_timestamp** –
- **end_timestamp** –
- **filters** –
- **session** –
- **order** –
- **limit** –
- **index** – index field name, str for single index, str list for multiple index
- **drop_index_col** – whether drop the col if it's in index, default False
- **time_field** –

Returns:: results basing on return_type.

classmethod record_data(*provider_index: int = 0*, *provider: Optional[str] = None*, *force_update=None*, *sleeping_time=None*, *exchanges=None*, *entity_id=None*, *entity_ids=None*, *code=None*, *codes=None*, *real_time=None*, *fix_duplicate_way=None*, *start_timestamp=None*, *end_timestamp=None*, *one_day_trading_minutes=None*, ***kwargs*)

record data by the arguments

Parameters::

- **entity_id** –
- **provider_index** –
- **provider** –
- **force_update** –
- **sleeping_time** –
- **exchanges** –
- **entity_ids** –
- **code** –
- **codes** –
- **real_time** –
- **fix_duplicate_way** –
- **start_timestamp** –
- **end_timestamp** –
- **one_day_trading_minutes** –
- **kwargs** –

Returns::

classmethod register_provider(*provider*)

register the provider to the schema defined by cls

Parameters:: **provider** –

classmethod register_recorder_cls(*provider*, *recorder_cls*)

register the recorder for the schema

Parameters::

- **provider** –
- **recorder_cls** –

timestamp = *Column(None, DateTime(), table=None)*

the meaning could be different for different case,most time it means 'happen time'

CurrencyKdataCommon

class zvt.domain.quotes.CurrencyKdataCommon

Bases: `KdataCommon`

__init__()

interest = *Column(None, Float(), table=None)*

持仓量

settlement = *Column(None, Float(), table=None)*

结算价

change_pct1 = *Column(None, Float(), table=None)*

涨跌幅(按结算价)

entity_id = *Column(None, String(), table=None)*

entity id

classmethod get_providers() → List[str]

providers of the schema defined by cls

Returns:: providers

classmethod get_storages(*provider: Optional[str] = None*)

get the storages info

Parameters:: **provider** – provider

Returns:: storages

id = *Column(None, String(), table=None, primary_key=True, nullable=False)*

id

classmethod query_data(*provider_index: int = 0*, *ids: Optional[List[str]] = None*, *entity_ids: Optional[List[str]] = None*, *entity_id: Optional[str] = None*, *codes: Optional[List[str]] = None*, *code: Optional[str] = None*, *level: Optional[Union[IntervalLevel, str]] = None*, *provider: Optional[str] = None*, *columns: Optional[List] = None*, *col_label: Optional[dict] = None*, *return_type: str = 'df'*, *start_timestamp: Optional[Union[Timestamp, str]] = None*, *end_timestamp: Optional[Union[Timestamp, str]] = None*, *filters: Optional[List] = None*, *session: Optional[Session] = None*, *order=None*, *limit: Optional[int] = None*, *distinct=None*, *index: Optional[Union[str, list]] = None*, *drop_index_col=False*, *time_field: str = 'timestamp'*)

query data by the arguments

Parameters::

- **provider_index** –
- **data_schema** –
- **ids** –
- **entity_ids** –
- **entity_id** –
- **codes** –
- **code** –
- **level** –
- **provider** –
- **columns** –
- **col_label** – dict with key(column), value(label
- **return_type** – df, domain or dict. default is df
- **start_timestamp** –
- **end_timestamp** –
- **filters** –
- **session** –
- **order** –
- **limit** –
- **index** – index field name, str for single index, str list for multiple index
- **drop_index_col** – whether drop the col if it's in index, default False
- **time_field** –

Returns:: results basing on return_type.

classmethod record_data(*provider_index: int = 0*, *provider: Optional[str] = None*, *force_update=None*, *sleeping_time=None*, *exchanges=None*, *entity_id=None*, *entity_ids=None*, *code=None*, *codes=None*, *real_time=None*, *fix_duplicate_way=None*, *start_timestamp=None*, *end_timestamp=None*, *one_day_trading_minutes=None*, ***kwargs*)

record data by the arguments

Parameters::

- **entity_id** –
- **provider_index** –
- **provider** –
- **force_update** –
- **sleeping_time** –
- **exchanges** –
- **entity_ids** –
- **code** –
- **codes** –
- **real_time** –
- **fix_duplicate_way** –
- **start_timestamp** –
- **end_timestamp** –
- **one_day_trading_minutes** –
- **kwargs** –

Returns::

classmethod register_provider(*provider*)

register the provider to the schema defined by cls

Parameters:: **provider** –

classmethod register_recorder_cls(*provider*, *recorder_cls*)

register the recorder for the schema

Parameters::

- **provider** –
- **recorder_cls** –

timestamp = *Column(None, DateTime(), table=None)*

the meaning could be different for different case,most time it means 'happen time'

EtfKdataCommon

class zvt.domain.quotes.EtfKdataCommon

Bases: `KdataCommon`

__init__()

entity_id = *Column(None, String(), table=None)*

entity id

classmethod get_providers() → List[str]

providers of the schema defined by cls

Returns:: providers

classmethod get_storages(*provider: Optional[str] = None*)

get the storages info

Parameters:: **provider** – provider

Returns:: storages

id = *Column(None, String(), table=None, primary_key=True, nullable=False)*

id

classmethod query_data(*provider_index: int = 0*, *ids: Optional[List[str]] = None*, *entity_ids: Optional[List[str]] = None*, *entity_id: Optional[str] = None*, *codes: Optional[List[str]] = None*, *code: Optional[str] = None*, *level: Optional[Union[IntervalLevel, str]] = None*, *provider: Optional[str] = None*, *columns: Optional[List] = None*, *col_label: Optional[dict] = None*, *return_type: str = 'df'*, *start_timestamp: Optional[Union[Timestamp, str]] = None*, *end_timestamp: Optional[Union[Timestamp, str]] = None*, *filters: Optional[List] = None*, *session: Optional[Session] = None*, *order=None*, *limit: Optional[int] = None*, *distinct=None*, *index: Optional[Union[str, list]] = None*, *drop_index_col=False*, *time_field: str = 'timestamp'*)

query data by the arguments

Parameters::

- **provider_index** –
- **data_schema** –
- **ids** –
- **entity_ids** –
- **entity_id** –
- **codes** –
- **code** –
- **level** –
- **provider** –
- **columns** –
- **col_label** – dict with key(column), value(label
- **return_type** – df, domain or dict. default is df
- **start_timestamp** –
- **end_timestamp** –
- **filters** –
- **session** –
- **order** –
- **limit** –
- **index** – index field name, str for single index, str list for multiple index
- **drop_index_col** – whether drop the col if it's in index, default False
- **time_field** –

Returns:: results basing on return_type.

classmethod record_data(*provider_index: int = 0*, *provider: Optional[str] = None*, *force_update=None*, *sleeping_time=None*, *exchanges=None*, *entity_id=None*, *entity_ids=None*, *code=None*, *codes=None*, *real_time=None*, *fix_duplicate_way=None*, *start_timestamp=None*, *end_timestamp=None*, *one_day_trading_minutes=None*, ***kwargs*)

record data by the arguments

Parameters::

- **entity_id** –
- **provider_index** –
- **provider** –
- **force_update** –
- **sleeping_time** –
- **exchanges** –
- **entity_ids** –
- **code** –
- **codes** –
- **real_time** –
- **fix_duplicate_way** –
- **start_timestamp** –
- **end_timestamp** –
- **one_day_trading_minutes** –
- **kwargs** –

Returns::

classmethod register_provider(*provider*)

register the provider to the schema defined by cls

Parameters:: **provider** –

classmethod register_recorder_cls(*provider*, *recorder_cls*)

register the recorder for the schema

Parameters::

- **provider** –
- **recorder_cls** –

timestamp = *Column(None, DateTime(), table=None)*

the meaning could be different for different case,most time it means 'happen time'

FutureKdataCommon

class zvt.domain.quotes.FutureKdataCommon

Bases: `KdataCommon`

__init__()

interest = *Column(None, Float(), table=None)*

持仓量

settlement = *Column(None, Float(), table=None)*

结算价

change_pct1 = *Column(None, Float(), table=None)*

涨跌幅(按结算价)

entity_id = *Column(None, String(), table=None)*

entity id

classmethod get_providers() → List[str]

providers of the schema defined by cls

Returns:: providers

classmethod get_storages(*provider: Optional[str] = None*)

get the storages info

Parameters:: **provider** – provider
Returns:: storages

id = *Column(None, String(), table=None, primary_key=True, nullable=False)*

id

classmethod query_data(*provider_index: int = 0*, *ids: Optional[List[str]] = None*, *entity_ids: Optional[List[str]] = None*, *entity_id: Optional[str] = None*, *codes: Optional[List[str]] = None*, *code: Optional[str] = None*, *level: Optional[Union[IntervalLevel, str]] = None*, *provider: Optional[str] = None*, *columns: Optional[List] = None*, *col_label: Optional[dict] = None*, *return_type: str = 'df'*, *start_timestamp: Optional[Union[Timestamp, str]] = None*, *end_timestamp: Optional[Union[Timestamp, str]] = None*, *filters: Optional[List] = None*, *session: Optional[Session] = None*, *order=None*, *limit: Optional[int] = None*, *distinct=None*, *index: Optional[Union[str, list]] = None*, *drop_index_col=False*, *time_field: str = 'timestamp'*)

query data by the arguments

Parameters::

- **provider_index** –
- **data_schema** –
- **ids** –
- **entity_ids** –
- **entity_id** –
- **codes** –
- **code** –
- **level** –
- **provider** –
- **columns** –
- **col_label** – dict with key(column), value(label
- **return_type** – df, domain or dict. default is df
- **start_timestamp** –
- **end_timestamp** –
- **filters** –
- **session** –
- **order** –
- **limit** –
- **index** – index field name, str for single index, str list for multiple index
- **drop_index_col** – whether drop the col if it's in index, default False
- **time_field** –

Returns:: results basing on return_type.

classmethod record_data(*provider_index: int = 0*, *provider: Optional[str] = None*, *force_update=None*, *sleeping_time=None*, *exchanges=None*, *entity_id=None*, *entity_ids=None*, *code=None*, *codes=None*, *real_time=None*, *fix_duplicate_way=None*, *start_timestamp=None*, *end_timestamp=None*, *one_day_trading_minutes=None*, ***kwargs*)

record data by the arguments

Parameters::

- **entity_id** –
- **provider_index** –
- **provider** –
- **force_update** –
- **sleeping_time** –
- **exchanges** –
- **entity_ids** –
- **code** –
- **codes** –
- **real_time** –
- **fix_duplicate_way** –
- **start_timestamp** –
- **end_timestamp** –
- **one_day_trading_minutes** –
- **kwargs** –

Returns::

classmethod register_provider(*provider*)

register the provider to the schema defined by cls

Parameters:: **provider** –

classmethod register_recorder_cls(*provider*, *recorder_cls*)

register the recorder for the schema

Parameters::

- **provider** –
- **recorder_cls** –

timestamp = *Column(None, DateTime(), table=None)*

the meaning could be different for different case,most time it means 'happen time'

IndexKdataCommon

class zvt.domain.quotes.IndexKdataCommon

Bases: `KdataCommon`

__init__()

entity_id = *Column(None, String(), table=None)*

entity id

classmethod get_providers() → List[str]

providers of the schema defined by cls

Returns:: providers

classmethod get_storages(*provider: Optional[str] = None*)

get the storages info

Parameters:: **provider** – provider

Returns:: storages

id = *Column(None, String(), table=None, primary_key=True, nullable=False)*

id

classmethod query_data(*provider_index: int = 0*, *ids: Optional[List[str]] = None*, *entity_ids: Optional[List[str]] = None*, *entity_id: Optional[str] = None*, *codes: Optional[List[str]] = None*, *code: Optional[str] = None*, *level: Optional[Union[IntervalLevel, str]] = None*, *provider: Optional[str] = None*, *columns: Optional[List] = None*, *col_label: Optional[dict] = None*, *return_type: str = 'df'*, *start_timestamp: Optional[Union[Timestamp, str]] = None*, *end_timestamp: Optional[Union[Timestamp, str]] = None*, *filters: Optional[List] = None*, *session: Optional[Session] = None*, *order=None*, *limit: Optional[int] = None*, *distinct=None*, *index: Optional[Union[str, list]] = None*, *drop_index_col=False*, *time_field: str = 'timestamp'*)

query data by the arguments

Parameters::

- **provider_index** –
- **data_schema** –
- **ids** –
- **entity_ids** –
- **entity_id** –
- **codes** –
- **code** –
- **level** –
- **provider** –
- **columns** –
- **col_label** – dict with key(column), value(label
- **return_type** – df, domain or dict. default is df
- **start_timestamp** –
- **end_timestamp** –
- **filters** –
- **session** –
- **order** –
- **limit** –
- **index** – index field name, str for single index, str list for multiple index
- **drop_index_col** – whether drop the col if it's in index, default False
- **time_field** –

Returns:: results basing on return_type.

classmethod record_data(*provider_index: int = 0*, *provider: Optional[str] = None*, *force_update=None*, *sleeping_time=None*, *exchanges=None*, *entity_id=None*, *entity_ids=None*, *code=None*, *codes=None*, *real_time=None*, *fix_duplicate_way=None*, *start_timestamp=None*, *end_timestamp=None*, *one_day_trading_minutes=None*, ***kwargs*)

record data by the arguments

Parameters::

- **entity_id** –
- **provider_index** –
- **provider** –
- **force_update** –
- **sleeping_time** –
- **exchanges** –
- **entity_ids** –
- **code** –
- **codes** –
- **real_time** –
- **fix_duplicate_way** –
- **start_timestamp** –
- **end_timestamp** –
- **one_day_trading_minutes** –
- **kwargs** –

Returns::

classmethod register_provider(*provider*)

register the provider to the schema defined by cls

Parameters:: **provider** –

classmethod register_recorder_cls(*provider*, *recorder_cls*)

register the recorder for the schema

Parameters::

- **provider** –
- **recorder_cls** –

timestamp = *Column(None, DateTime(), table=None)*

the meaning could be different for different case,most time it means 'happen time'

IndexusKdataCommon

class zvt.domain.quotes.IndexusKdataCommon

Bases: `KdataCommon`

__init__()

entity_id = *Column(None, String(), table=None)*

entity id

classmethod get_providers() → List[str]

providers of the schema defined by cls

Returns:: providers

classmethod get_storages(*provider: Optional[str] = None*)

get the storages info

Parameters:: **provider** – provider

Returns:: storages

id = *Column(None, String(), table=None, primary_key=True, nullable=False)*

id

classmethod query_data(*provider_index: int = 0*, *ids: Optional[List[str]] = None*, *entity_ids: Optional[List[str]] = None*, *entity_id: Optional[str] = None*, *codes: Optional[List[str]] = None*, *code: Optional[str] = None*, *level: Optional[Union[IntervalLevel, str]] = None*, *provider: Optional[str] = None*, *columns: Optional[List] = None*, *col_label: Optional[dict] = None*, *return_type: str = 'df'*, *start_timestamp: Optional[Union[Timestamp, str]] = None*, *end_timestamp: Optional[Union[Timestamp, str]] = None*, *filters: Optional[List] = None*, *session: Optional[Session] = None*, *order=None*, *limit: Optional[int] = None*, *distinct=None*, *index: Optional[Union[str, list]] = None*, *drop_index_col=False*, *time_field: str = 'timestamp'*)

query data by the arguments

Parameters::

- **provider_index** –
- **data_schema** –
- **ids** –
- **entity_ids** –
- **entity_id** –
- **codes** –
- **code** –
- **level** –
- **provider** –
- **columns** –
- **col_label** – dict with key(column), value(label
- **return_type** – df, domain or dict. default is df
- **start_timestamp** –
- **end_timestamp** –
- **filters** –
- **session** –
- **order** –
- **limit** –
- **index** – index field name, str for single index, str list for multiple index
- **drop_index_col** – whether drop the col if it's in index, default False
- **time_field** –

Returns:: results basing on return_type.

classmethod record_data(*provider_index: int = 0*, *provider: Optional[str] = None*, *force_update=None*, *sleeping_time=None*, *exchanges=None*, *entity_id=None*, *entity_ids=None*, *code=None*, *codes=None*, *real_time=None*, *fix_duplicate_way=None*, *start_timestamp=None*, *end_timestamp=None*, *one_day_trading_minutes=None*, ***kwargs*)

record data by the arguments

Parameters::

- **entity_id** –
- **provider_index** –
- **provider** –
- **force_update** –
- **sleeping_time** –
- **exchanges** –
- **entity_ids** –
- **code** –
- **codes** –
- **real_time** –
- **fix_duplicate_way** –
- **start_timestamp** –
- **end_timestamp** –
- **one_day_trading_minutes** –
- **kwargs** –

Returns::

classmethod register_provider(*provider*)

register the provider to the schema defined by cls

Parameters:: **provider** –

classmethod register_recorder_cls(*provider*, *recorder_cls*)

register the recorder for the schema

Parameters::

- **provider** –
- **recorder_cls** –

timestamp = *Column(None, DateTime(), table=None)*

the meaning could be different for different case,most time it means 'happen time'

KdataCommon

class zvt.domain.quotes.KdataCommon

Bases: `Mixin`

__init__()

entity_id = *Column(None, String(), table=None)*

entity id

classmethod get_providers() → List[str]

providers of the schema defined by cls

Returns:: providers

classmethod get_storages(*provider: Optional[str] = None*)

get the storages info

Parameters:: **provider** – provider

Returns:: storages

id = *Column(None, String(), table=None, primary_key=True, nullable=False)*

id

classmethod query_data(*provider_index: int = 0*, *ids: Optional[List[str]] = None*, *entity_ids: Optional[List[str]] = None*, *entity_id: Optional[str] = None*, *codes: Optional[List[str]] = None*, *code: Optional[str] = None*, *level: Optional[Union[IntervalLevel, str]] = None*, *provider: Optional[str] = None*, *columns: Optional[List] = None*, *col_label: Optional[dict] = None*, *return_type: str = 'df'*, *start_timestamp: Optional[Union[Timestamp, str]] = None*, *end_timestamp: Optional[Union[Timestamp, str]] = None*, *filters: Optional[List] = None*, *session: Optional[Session] = None*, *order=None*, *limit: Optional[int] = None*, *distinct=None*, *index: Optional[Union[str, list]] = None*, *drop_index_col=False*, *time_field: str = 'timestamp'*)

query data by the arguments

Parameters::

- **provider_index** –
- **data_schema** –
- **ids** –
- **entity_ids** –
- **entity_id** –
- **codes** –
- **code** –
- **level** –
- **provider** –
- **columns** –
- **col_label** – dict with key(column), value(label
- **return_type** – df, domain or dict. default is df
- **start_timestamp** –
- **end_timestamp** –
- **filters** –
- **session** –
- **order** –
- **limit** –
- **index** – index field name, str for single index, str list for multiple index
- **drop_index_col** – whether drop the col if it's in index, default False
- **time_field** –

Returns:: results basing on return_type.

classmethod record_data(*provider_index: int = 0*, *provider: Optional[str] = None*, *force_update=None*, *sleeping_time=None*, *exchanges=None*, *entity_id=None*, *entity_ids=None*, *code=None*, *codes=None*, *real_time=None*, *fix_duplicate_way=None*, *start_timestamp=None*, *end_timestamp=None*, *one_day_trading_minutes=None*, ***kwargs*)

record data by the arguments

Parameters::

- **entity_id** –
- **provider_index** –
- **provider** –
- **force_update** –
- **sleeping_time** –
- **exchanges** –
- **entity_ids** –
- **code** –
- **codes** –
- **real_time** –
- **fix_duplicate_way** –
- **start_timestamp** –
- **end_timestamp** –
- **one_day_trading_minutes** –
- **kwargs** –

Returns::

classmethod register_provider(*provider*)

register the provider to the schema defined by cls

Parameters:: **provider** –

classmethod register_recorder_cls(*provider*, *recorder_cls*)

register the recorder for the schema

Parameters::

- **provider** –
- **recorder_cls** –

timestamp = *Column(None, DateTime(), table=None)*

the meaning could be different for different case,most time it means 'happen time'

StockKdataCommon

class zvt.domain.quotes.StockKdataCommon

Bases: `KdataCommon`

__init__()

entity_id = *Column(None, String(), table=None)*

entity id

classmethod get_providers() → List[str]

providers of the schema defined by cls

Returns:: providers

classmethod get_storages(*provider: Optional[str] = None*)

get the storages info

Parameters:: **provider** – provider

Returns:: storages

id = *Column(None, String(), table=None, primary_key=True, nullable=False)*

id

classmethod query_data(*provider_index: int = 0*, *ids: Optional[List[str]] = None*, *entity_ids: Optional[List[str]] = None*, *entity_id: Optional[str] = None*, *codes: Optional[List[str]] = None*, *code: Optional[str] = None*, *level: Optional[Union[IntervalLevel, str]] = None*, *provider: Optional[str] = None*, *columns: Optional[List] = None*, *col_label: Optional[dict] = None*, *return_type: str = 'df'*, *start_timestamp: Optional[Union[Timestamp, str]] = None*, *end_timestamp: Optional[Union[Timestamp, str]] = None*, *filters: Optional[List] = None*, *session: Optional[Session] = None*, *order=None*, *limit: Optional[int] = None*, *distinct=None*, *index: Optional[Union[str, list]] = None*, *drop_index_col=False*, *time_field: str = 'timestamp'*)

query data by the arguments

Parameters::

- **provider_index** –
- **data_schema** –
- **ids** –
- **entity_ids** –
- **entity_id** –
- **codes** –
- **code** –
- **level** –
- **provider** –
- **columns** –
- **col_label** – dict with key(column), value(label
- **return_type** – df, domain or dict. default is df
- **start_timestamp** –
- **end_timestamp** –
- **filters** –
- **session** –
- **order** –
- **limit** –
- **index** – index field name, str for single index, str list for multiple index
- **drop_index_col** – whether drop the col if it's in index, default False
- **time_field** –

Returns:: results basing on return_type.

classmethod record_data(*provider_index: int = 0*, *provider: Optional[str] = None*, *force_update=None*, *sleeping_time=None*, *exchanges=None*, *entity_id=None*, *entity_ids=None*, *code=None*, *codes=None*, *real_time=None*, *fix_duplicate_way=None*, *start_timestamp=None*, *end_timestamp=None*, *one_day_trading_minutes=None*, ***kwargs*)

record data by the arguments

Parameters::

- **entity_id** –
- **provider_index** –
- **provider** –
- **force_update** –
- **sleeping_time** –
- **exchanges** –
- **entity_ids** –
- **code** –
- **codes** –
- **real_time** –
- **fix_duplicate_way** –
- **start_timestamp** –
- **end_timestamp** –
- **one_day_trading_minutes** –
- **kwargs** –

Returns::

classmethod register_provider(*provider*)

register the provider to the schema defined by cls

Parameters:: **provider** –

classmethod register_recorder_cls(*provider*, *recorder_cls*)

register the recorder for the schema

Parameters::

- **provider** –
- **recorder_cls** –

timestamp = *Column(None, DateTime(), table=None)*

the meaning could be different for different case,most time it means 'happen time'

StockhkKdataCommon

class zvt.domain.quotes.StockhkKdataCommon

Bases: `KdataCommon`

__init__()

entity_id = *Column(None, String(), table=None)*

entity id

classmethod get_providers() → List[str]

providers of the schema defined by cls

Returns:: providers

classmethod get_storages(*provider: Optional[str] = None*)

get the storages info

Parameters:: **provider** – provider

Returns:: storages

id = *Column(None, String(), table=None, primary_key=True, nullable=False)*

id

classmethod query_data(*provider_index: int = 0*, *ids: Optional[List[str]] = None*, *entity_ids: Optional[List[str]] = None*, *entity_id: Optional[str] = None*, *codes: Optional[List[str]] = None*, *code: Optional[str] = None*, *level: Optional[Union[IntervalLevel, str]] = None*, *provider: Optional[str] = None*, *columns: Optional[List] = None*, *col_label: Optional[dict] = None*, *return_type: str = 'df'*, *start_timestamp: Optional[Union[Timestamp, str]] = None*, *end_timestamp: Optional[Union[Timestamp, str]] = None*, *filters: Optional[List] = None*, *session: Optional[Session] = None*, *order=None*, *limit: Optional[int] = None*, *distinct=None*, *index: Optional[Union[str, list]] = None*, *drop_index_col=False*, *time_field: str = 'timestamp'*)

query data by the arguments

Parameters::

- **provider_index** –
- **data_schema** –
- **ids** –
- **entity_ids** –
- **entity_id** –
- **codes** –
- **code** –
- **level** –
- **provider** –
- **columns** –
- **col_label** – dict with key(column), value(label
- **return_type** – df, domain or dict. default is df
- **start_timestamp** –
- **end_timestamp** –
- **filters** –
- **session** –
- **order** –
- **limit** –
- **index** – index field name, str for single index, str list for multiple index
- **drop_index_col** – whether drop the col if it's in index, default False
- **time_field** –

Returns:: results basing on return_type.

classmethod record_data(*provider_index: int = 0*, *provider: Optional[str] = None*, *force_update=None*, *sleeping_time=None*, *exchanges=None*, *entity_id=None*, *entity_ids=None*, *code=None*, *codes=None*, *real_time=None*, *fix_duplicate_way=None*, *start_timestamp=None*, *end_timestamp=None*, *one_day_trading_minutes=None*, ***kwargs*)

record data by the arguments

Parameters::

- **entity_id** –
- **provider_index** –
- **provider** –
- **force_update** –
- **sleeping_time** –
- **exchanges** –
- **entity_ids** –
- **code** –
- **codes** –
- **real_time** –
- **fix_duplicate_way** –
- **start_timestamp** –
- **end_timestamp** –
- **one_day_trading_minutes** –
- **kwargs** –

Returns::

classmethod register_provider(*provider*)

register the provider to the schema defined by cls

Parameters:: **provider** –

classmethod register_recorder_cls(*provider*, *recorder_cls*)

register the recorder for the schema

Parameters::

- **provider** –
- **recorder_cls** –

timestamp = *Column(None, DateTime(), table=None)*

the meaning could be different for different case,most time it means 'happen time'

StockusKdataCommon

class zvt.domain.quotes.StockusKdataCommon

Bases: `KdataCommon`

__init__()

entity_id = *Column(None, String(), table=None)*

entity id

classmethod get_providers() → List[str]

providers of the schema defined by cls

Returns:: providers

classmethod get_storages(*provider: Optional[str] = None*)

get the storages info

Parameters:: **provider** – provider

Returns:: storages

id = *Column(None, String(), table=None, primary_key=True, nullable=False)*

id

classmethod query_data(*provider_index: int = 0*, *ids: Optional[List[str]] = None*, *entity_ids: Optional[List[str]] = None*, *entity_id: Optional[str] = None*, *codes: Optional[List[str]] = None*, *code: Optional[str] = None*, *level: Optional[Union[IntervalLevel, str]] = None*, *provider: Optional[str] = None*, *columns: Optional[List] = None*, *col_label: Optional[dict] = None*, *return_type: str = 'df'*, *start_timestamp: Optional[Union[Timestamp, str]] = None*, *end_timestamp: Optional[Union[Timestamp, str]] = None*, *filters: Optional[List] = None*, *session: Optional[Session] = None*, *order=None*, *limit: Optional[int] = None*, *distinct=None*, *index: Optional[Union[str, list]] = None*, *drop_index_col=False*, *time_field: str = 'timestamp'*)

query data by the arguments

Parameters::

- **provider_index** –
- **data_schema** –
- **ids** –
- **entity_ids** –
- **entity_id** –
- **codes** –
- **code** –
- **level** –
- **provider** –
- **columns** –
- **col_label** – dict with key(column), value(label
- **return_type** – df, domain or dict. default is df
- **start_timestamp** –
- **end_timestamp** –
- **filters** –
- **session** –
- **order** –
- **limit** –
- **index** – index field name, str for single index, str list for multiple index
- **drop_index_col** – whether drop the col if it's in index, default False
- **time_field** –

Returns:: results basing on return_type.

classmethod record_data(*provider_index: int = 0*, *provider: Optional[str] = None*, *force_update=None*, *sleeping_time=None*, *exchanges=None*, *entity_id=None*, *entity_ids=None*, *code=None*, *codes=None*, *real_time=None*, *fix_duplicate_way=None*, *start_timestamp=None*, *end_timestamp=None*, *one_day_trading_minutes=None*, ***kwargs*)

record data by the arguments

Parameters::

- **entity_id** –
- **provider_index** –
- **provider** –
- **force_update** –
- **sleeping_time** –
- **exchanges** –
- **entity_ids** –
- **code** –
- **codes** –
- **real_time** –
- **fix_duplicate_way** –
- **start_timestamp** –
- **end_timestamp** –
- **one_day_trading_minutes** –
- **kwargs** –

Returns::

classmethod register_provider(*provider*)

register the provider to the schema defined by cls

Parameters:: **provider** –

classmethod register_recorder_cls(*provider*, *recorder_cls*)

register the recorder for the schema

Parameters::

- **provider** –
- **recorder_cls** –

timestamp = *Column(None, DateTime(), table=None)*

the meaning could be different for different case,most time it means 'happen time'

TickCommon

class zvt.domain.quotes.TickCommon

Bases: `Mixin`

__init__()

time = *Column(None, Integer(), table=None)*

UNIX时间戳

open = *Column(None, Float(), table=None)*

开盘价

close = *Column(None, Float(), table=None)*

收盘价/当前价格

high = *Column(None, Float(), table=None)*

最高价

low = *Column(None, Float(), table=None)*

最低价

volume = *Column(None, Float(), table=None)*

成交量

turnover = *Column(None, Float(), table=None)*

成交金额

ask_price = *Column(None, Float(), table=None)*

委卖价

bid_price = *Column(None, Float(), table=None)*

委买价

ask_vol = *Column(None, JSON(), table=None)*

委卖量

bid_vol = *Column(None, JSON(), table=None)*

委买量

transaction_num = *Column(None, Integer(), table=None)*

成交笔数

entity_id = *Column(None, String(), table=None)*

entity id

classmethod get_providers() → List[str]

providers of the schema defined by cls

Returns:: providers

classmethod get_storages(*provider: Optional[str] = None*)

get the storages info

Parameters:: **provider** – provider

Returns:: storages

id = *Column(None, String(), table=None, primary_key=True, nullable=False)*

id

classmethod query_data(*provider_index: int = 0*, *ids: Optional[List[str]] = None*, *entity_ids: Optional[List[str]] = None*, *entity_id: Optional[str] = None*, *codes: Optional[List[str]] = None*, *code: Optional[str] = None*, *level: Optional[Union[IntervalLevel, str]] = None*, *provider: Optional[str] = None*, *columns: Optional[List] = None*, *col_label: Optional[dict] = None*, *return_type: str = 'df'*, *start_timestamp: Optional[Union[Timestamp, str]] = None*, *end_timestamp: Optional[Union[Timestamp, str]] = None*, *filters: Optional[List] = None*, *session: Optional[Session] = None*, *order=None*, *limit: Optional[int] =*

None, *distinct=None*, *index: Optional[Union[str, list]] = None*, *drop_index_col=False*, *time_field: str = 'timestamp'*)

query data by the arguments

Parameters::

- **provider_index** –
- **data_schema** –
- **ids** –
- **entity_ids** –
- **entity_id** –
- **codes** –
- **code** –
- **level** –
- **provider** –
- **columns** –
- **col_label** – dict with key(column), value(label
- **return_type** – df, domain or dict. default is df
- **start_timestamp** –
- **end_timestamp** –
- **filters** –
- **session** –
- **order** –
- **limit** –
- **index** – index field name, str for single index, str list for multiple index
- **drop_index_col** – whether drop the col if it's in index, default False
- **time_field** –

Returns:: results basing on return_type.

classmethod record_data(*provider_index: int = 0*, *provider: Optional[str] = None*, *force_update=None*, *sleeping_time=None*, *exchanges=None*, *entity_id=None*, *entity_ids=None*, *code=None*, *codes=None*, *real_time=None*, *fix_duplicate_way=None*, *start_timestamp=None*, *end_timestamp=None*, *one_day_trading_minutes=None*, ***kwargs*)

record data by the arguments

Parameters::

- **entity_id** –
- **provider_index** –
- **provider** –
- **force_update** –
- **sleeping_time** –
- **exchanges** –
- **entity_ids** –
- **code** –
- **codes** –
- **real_time** –
- **fix_duplicate_way** –
- **start_timestamp** –
- **end_timestamp** –
- **one_day_trading_minutes** –
- **kwargs** –

Returns::

classmethod register_provider(*provider*)

register the provider to the schema defined by cls

Parameters:: **provider** –

classmethod register_recorder_cls(*provider*, *recorder_cls*)

register the recorder for the schema

Parameters::

- **provider** –
- **recorder_cls** –

timestamp = *Column(None, DateTime(), table=None)*

the meaning could be different for different case,most time it means 'happen time'

zvt.domain.quotes.block

Modules

zvt.domain.quotes.block.block_1d_kdata

---

zvt.domain.quotes.block.block_1mon_kdata

---

zvt.domain.quotes.block.block_1wk_kdata

---

zvt.domain.quotes.block.block_1d_kdata

Classes

`Block1dKdata`(**kwargs)

Block1dKdata

class zvt.domain.quotes.block.block_1d_kdata.Block1dKdata(**kwargs*)

Bases: **Base**, **BlockKdataCommon**

__init__(**kwargs*)

A simple constructor that allows initialization from kwargs.

Sets attributes on the constructed instance using the names and values in `kwargs`.

Only keys that are present as attributes of the instance's class are allowed. These could be, for example, any mapped columns or relationships.

change_pct

close

code

entity_id

entity id

classmethod get_providers() → List[str]

providers of the schema defined by cls

Returns:: providers

classmethod get_storages(*provider: Optional[str] = None*)

get the storages info

Parameters:: **provider** – provider
Returns:: storages

high

id

id

level

low

name

open

provider

classmethod query_data(*provider_index: int = 0*, *ids: Optional[List[str]] = None*, *entity_ids: Optional[List[str]] = None*, *entity_id: Optional[str] = None*, *codes: Optional[List[str]] = None*, *code: Optional[str] = None*, *level: Optional[Union[IntervalLevel, str]] = None*, *provider: Optional[str] = None*, *columns: Optional[List] = None*, *col_label: Optional[dict] = None*, *return_type: str = 'df'*, *start_timestamp: Optional[Union[Timestamp, str]] = None*, *end_timestamp: Optional[Union[Timestamp, str]] = None*, *filters: Optional[List] = None*, *session: Optional[Session] = None*, *order=None*, *limit: Optional[int] = None*, *distinct=None*, *index: Optional[Union[str, list]] = None*, *drop_index_col=False*, *time_field: str = 'timestamp'*)

query data by the arguments

Parameters::

- **provider_index** –
- **data_schema** –
- **ids** –
- **entity_ids** –
- **entity_id** –
- **codes** –
- **code** –
- **level** –
- **provider** –
- **columns** –
- **col_label** – dict with key(column), value(label
- **return_type** – df, domain or dict. default is df
- **start_timestamp** –
- **end_timestamp** –
- **filters** –
- **session** –
- **order** –
- **limit** –
- **index** – index field name, str for single index, str list for multiple index
- **drop_index_col** – whether drop the col if it's in index, default False
- **time_field** –

Returns:: results basing on return_type.

classmethod record_data(*provider_index: int = 0*, *provider: Optional[str] = None*, *force_update=None*, *sleeping_time=None*, *exchanges=None*, *entity_id=None*, *entity_ids=None*, *code=None*, *codes=None*, *real_time=None*, *fix_duplicate_way=None*, *start_timestamp=None*, *end_timestamp=None*, *one_day_trading_minutes=None*, ***kwargs*)

record data by the arguments

Parameters::

- **entity_id** –
- **provider_index** –
- **provider** –
- **force_update** –
- **sleeping_time** –
- **exchanges** –
- **entity_ids** –
- **code** –
- **codes** –
- **real_time** –
- **fix_duplicate_way** –
- **start_timestamp** –
- **end_timestamp** –
- **one_day_trading_minutes** –
- **kwargs** –

Returns::

classmethod register_provider(*provider*)

register the provider to the schema defined by cls

Parameters:: **provider** –

classmethod register_recorder_cls(*provider*, *recorder_cls*)

register the recorder for the schema

Parameters::

- **provider** –
- **recorder_cls** –

timestamp

the meaning could be different for different case,most time it means 'happen time'

turnover

turnover_rate

volume

zvt.domain.quotes.block.block_1mon_kdata

Classes

`Block1monKdata`(**kwargs)

Block1monKdata

class zvt.domain.quotes.block.block_1mon_kdata.Block1monKdata(***kwargs*)

Bases: **Base**, **BlockKdataCommon**

__init__(***kwargs*)

A simple constructor that allows initialization from kwargs.

Sets attributes on the constructed instance using the names and values in `kwargs`.

Only keys that are present as attributes of the instance's class are allowed. These could be, for example, any mapped columns or relationships.

change_pct

close

code

entity_id

entity id

classmethod get_providers() → List[str]

providers of the schema defined by cls

Returns:: providers

classmethod get_storages(*provider: Optional[str] = None*)

get the storages info

Parameters:: **provider** – provider
Returns:: storages

high

id

id

level

low

name

open

provider

classmethod query_data(*provider_index: int = 0*, *ids: Optional[List[str]] = None*, *entity_ids: Optional[List[str]] = None*, *entity_id: Optional[str] = None*, *codes: Optional[List[str]] = None*, *code: Optional[str] = None*, *level: Optional[Union[IntervalLevel, str]] = None*, *provider: Optional[str] = None*, *columns: Optional[List] = None*, *col_label: Optional[dict] = None*, *return_type: str = 'df'*, *start_timestamp: Optional[Union[Timestamp, str]] = None*, *end_timestamp: Optional[Union[Timestamp, str]] = None*, *filters: Optional[List] = None*, *session: Optional[Session] = None*, *order=None*, *limit: Optional[int] = None*, *distinct=None*, *index: Optional[Union[str, list]] = None*, *drop_index_col=False*, *time_field: str = 'timestamp'*)

query data by the arguments

Parameters::

- **provider_index** –
- **data_schema** –
- **ids** –
- **entity_ids** –
- **entity_id** –
- **codes** –
- **code** –
- **level** –
- **provider** –
- **columns** –
- **col_label** – dict with key(column), value(label
- **return_type** – df, domain or dict. default is df
- **start_timestamp** –
- **end_timestamp** –
- **filters** –
- **session** –
- **order** –
- **limit** –
- **index** – index field name, str for single index, str list for multiple index
- **drop_index_col** – whether drop the col if it's in index, default False
- **time_field** –

Returns:: results basing on return_type.

classmethod record_data(*provider_index: int = 0*, *provider: Optional[str] = None*, *force_update=None*, *sleeping_time=None*, *exchanges=None*, *entity_id=None*, *entity_ids=None*, *code=None*, *codes=None*, *real_time=None*, *fix_duplicate_way=None*, *start_timestamp=None*, *end_timestamp=None*, *one_day_trading_minutes=None*, ***kwargs*)

record data by the arguments

Parameters::

- **entity_id** –
- **provider_index** –
- **provider** –
- **force_update** –
- **sleeping_time** –
- **exchanges** –
- **entity_ids** –
- **code** –
- **codes** –
- **real_time** –
- **fix_duplicate_way** –
- **start_timestamp** –
- **end_timestamp** –
- **one_day_trading_minutes** –
- **kwargs** –

Returns::

classmethod register_provider(*provider*)

register the provider to the schema defined by cls

Parameters:: **provider** –

classmethod register_recorder_cls(*provider*, *recorder_cls*)

register the recorder for the schema

Parameters::

- **provider** –
- **recorder_cls** –

timestamp

the meaning could be different for different case,most time it means 'happen time'

turnover

turnover_rate

volume

zvt.domain.quotes.block.block_1wk_kdata

Classes

`Block1wkKdata`(**kwargs)

Block1wkKdata

class zvt.domain.quotes.block.block_1wk_kdata.Block1wkKdata(**kwargs*)

Bases: **Base**, **BlockKdataCommon**

__init__(**kwargs*)

A simple constructor that allows initialization from kwargs.

Sets attributes on the constructed instance using the names and values in `kwargs`.

Only keys that are present as attributes of the instance's class are allowed. These could be, for example, any mapped columns or relationships.

change_pct

close

code

entity_id

entity id

classmethod get_providers() → List[str]

providers of the schema defined by cls

Returns:: providers

classmethod get_storages(*provider: Optional[str] = None*)

get the storages info

Parameters:: **provider** – provider
Returns:: storages

high

id

id

level

low

name

open

provider

classmethod query_data(*provider_index: int = 0*, *ids: Optional[List[str]] = None*, *entity_ids: Optional[List[str]] = None*, *entity_id: Optional[str] = None*, *codes: Optional[List[str]] = None*, *code: Optional[str] = None*, *level: Optional[Union[IntervalLevel, str]] = None*, *provider: Optional[str] = None*, *columns: Optional[List] = None*, *col_label: Optional[dict] = None*, *return_type: str = 'df'*, *start_timestamp: Optional[Union[Timestamp, str]] = None*, *end_timestamp: Optional[Union[Timestamp, str]] = None*, *filters: Optional[List] = None*, *session: Optional[Session] = None*, *order=None*, *limit: Optional[int] = None*, *distinct=None*, *index: Optional[Union[str, list]] = None*, *drop_index_col=False*, *time_field: str = 'timestamp'*)

query data by the arguments

Parameters::

- **provider_index** –
- **data_schema** –
- **ids** –
- **entity_ids** –
- **entity_id** –
- **codes** –
- **code** –
- **level** –
- **provider** –
- **columns** –
- **col_label** – dict with key(column), value(label
- **return_type** – df, domain or dict. default is df
- **start_timestamp** –
- **end_timestamp** –
- **filters** –
- **session** –
- **order** –
- **limit** –
- **index** – index field name, str for single index, str list for multiple index
- **drop_index_col** – whether drop the col if it's in index, default False
- **time_field** –

Returns:: results basing on return_type.

classmethod record_data(*provider_index: int = 0*, *provider: Optional[str] = None*, *force_update=None*, *sleeping_time=None*, *exchanges=None*, *entity_id=None*, *entity_ids=None*, *code=None*, *codes=None*, *real_time=None*, *fix_duplicate_way=None*, *start_timestamp=None*, *end_timestamp=None*, *one_day_trading_minutes=None*, ***kwargs*)

record data by the arguments

Parameters::

- **entity_id** –
- **provider_index** –
- **provider** –
- **force_update** –
- **sleeping_time** –
- **exchanges** –
- **entity_ids** –
- **code** –
- **codes** –
- **real_time** –
- **fix_duplicate_way** –
- **start_timestamp** –
- **end_timestamp** –
- **one_day_trading_minutes** –
- **kwargs** –

Returns::

classmethod register_provider(*provider*)

register the provider to the schema defined by cls

Parameters:: **provider** –

classmethod register_recorder_cls(*provider*, *recorder_cls*)

register the recorder for the schema

Parameters::

- **provider** –
- **recorder_cls** –

timestamp

the meaning could be different for different case,most time it means 'happen time'

turnover

turnover_rate

volume

zvt.domain.quotes.currency

Modules

zvt.domain.quotes.currency.currency_1d_kdata

---

zvt.domain.quotes.currency.currency_1d_kdata

Classes

`Currency1dKdata`(**kwargs)

Currency1dKdata

class zvt.domain.quotes.currency.currency_1d_kdata.Currency1dKdata(***kwargs*)

Bases: **Base**, **CurrencyKdataCommon**

__init__(***kwargs*)

A simple constructor that allows initialization from kwargs.

Sets attributes on the constructed instance using the names and values in `kwargs`.

Only keys that are present as attributes of the instance's class are allowed. These could be, for example, any mapped columns or relationships.

change_pct

change_pct1

涨跌幅(按结算价)

close

code

entity_id

entity id

classmethod get_providers() → List[str]

providers of the schema defined by cls

Returns:: providers

classmethod get_storages(*provider: Optional[str] = None*)

get the storages info

Parameters:: **provider** – provider
Returns:: storages

high

id

id

interest

持仓量

level

low

name

open

provider

classmethod query_data(*provider_index: int = 0*, *ids: Optional[List[str]] = None*, *entity_ids: Optional[List[str]] = None*, *entity_id: Optional[str] = None*, *codes: Optional[List[str]] = None*, *code: Optional[str] = None*, *level: Optional[Union[IntervalLevel, str]] = None*, *provider: Optional[str] = None*, *columns: Optional[List] = None*, *col_label: Optional[dict] = None*, *return_type: str = 'df'*, *start_timestamp: Optional[Union[Timestamp, str]] = None*, *end_timestamp: Optional[Union[Timestamp, str]] = None*, *filters: Optional[List] = None*, *session: Optional[Session] = None*, *order=None*, *limit: Optional[int] = None*, *distinct=None*, *index: Optional[Union[str, list]] = None*, *drop_index_col=False*, *time_field: str = 'timestamp'*)

query data by the arguments

Parameters::

- **provider_index** –
- **data_schema** –
- **ids** –
- **entity_ids** –
- **entity_id** –
- **codes** –
- **code** –
- **level** –
- **provider** –
- **columns** –
- **col_label** – dict with key(column), value(label
- **return_type** – df, domain or dict. default is df
- **start_timestamp** –
- **end_timestamp** –
- **filters** –
- **session** –
- **order** –
- **limit** –
- **index** – index field name, str for single index, str list for multiple index
- **drop_index_col** – whether drop the col if it's in index, default False
- **time_field** –

Returns:: results basing on return_type.

classmethod record_data(*provider_index: int = 0*, *provider: Optional[str] = None*, *force_update=None*, *sleeping_time=None*, *exchanges=None*, *entity_id=None*, *entity_ids=None*, *code=None*, *codes=None*, *real_time=None*, *fix_duplicate_way=None*, *start_timestamp=None*, *end_timestamp=None*, *one_day_trading_minutes=None*, ***kwargs*)

record data by the arguments

Parameters::

- **entity_id** –
- **provider_index** –
- **provider** –
- **force_update** –
- **sleeping_time** –
- **exchanges** –
- **entity_ids** –
- **code** –
- **codes** –
- **real_time** –
- **fix_duplicate_way** –
- **start_timestamp** –
- **end_timestamp** –
- **one_day_trading_minutes** –
- **kwargs** –

Returns::

classmethod register_provider(*provider*)

register the provider to the schema defined by cls

Parameters:: **provider** –

classmethod register_recorder_cls(*provider*, *recorder_cls*)

register the recorder for the schema

Parameters::

- **provider** –
- **recorder_cls** –

settlement

结算价

timestamp

the meaning could be different for different case,most time it means 'happen time'

turnover

turnover_rate

volume

zvt.domain.quotes.etf

Modules

zvt.domain.quotes.etf.etf_1d_kdata

---

zvt.domain.quotes.etf.etf_1d_kdata

Classes

`Etf1dKdata`(**kwargs)

Etf1dKdata

class zvt.domain.quotes.etf.etf_1d_kdata.Etf1dKdata(**kwargs)

Bases: **Base**, **EtfKdataCommon**

__init__(**kwargs)

A simple constructor that allows initialization from kwargs.

Sets attributes on the constructed instance using the names and values in `kwargs`.

Only keys that are present as attributes of the instance's class are allowed. These could be, for example, any mapped columns or relationships.

change_pct

close

code

cumulative_net_value

entity_id

entity id

classmethod get_providers() → List[str]

providers of the schema defined by cls

Returns:: providers

classmethod get_storages(*provider: Optional[str] = None*)

get the storages info

Parameters:: **provider** – provider

Returns:: storages

high

id

id

level

low

name

open

provider

classmethod query_data(*provider_index: int = 0*, *ids: Optional[List[str]] = None*, *entity_ids: Optional[List[str]] = None*, *entity_id: Optional[str] = None*, *codes: Optional[List[str]] = None*, *code: Optional[str] = None*, *level: Optional[Union[IntervalLevel, str]] = None*, *provider: Optional[str] = None*, *columns: Optional[List] = None*, *col_label: Optional[dict] = None*, *return_type: str = 'df'*, *start_timestamp: Optional[Union[Timestamp, str]] = None*, *end_timestamp: Optional[Union[Timestamp, str]] = None*, *filters: Optional[List] = None*, *session: Optional[Session] = None*, *order=None*, *limit: Optional[int] = None*, *distinct=None*, *index: Optional[Union[str, list]] = None*, *drop_index_col=False*, *time_field: str = 'timestamp'*)

query data by the arguments

Parameters::

- **provider_index** –
- **data_schema** –
- **ids** –
- **entity_ids** –
- **entity_id** –
- **codes** –
- **code** –
- **level** –
- **provider** –
- **columns** –
- **col_label** – dict with key(column), value(label
- **return_type** – df, domain or dict. default is df
- **start_timestamp** –
- **end_timestamp** –
- **filters** –
- **session** –
- **order** –
- **limit** –
- **index** – index field name, str for single index, str list for multiple index
- **drop_index_col** – whether drop the col if it's in index, default False
- **time_field** –

Returns:: results basing on return_type.

classmethod record_data(*provider_index: int = 0*, *provider: Optional[str] = None*, *force_update=None*, *sleeping_time=None*, *exchanges=None*, *entity_id=None*, *entity_ids=None*, *code=None*, *codes=None*, *real_time=None*, *fix_duplicate_way=None*, *start_timestamp=None*, *end_timestamp=None*, *one_day_trading_minutes=None*, ***kwargs*)

record data by the arguments

Parameters::

- **entity_id** –
- **provider_index** –
- **provider** –
- **force_update** –
- **sleeping_time** –
- **exchanges** –
- **entity_ids** –
- **code** –
- **codes** –
- **real_time** –
- **fix_duplicate_way** –
- **start_timestamp** –
- **end_timestamp** –
- **one_day_trading_minutes** –
- **kwargs** –

Returns::

classmethod register_provider(*provider*)

register the provider to the schema defined by cls

Parameters:: **provider** –

classmethod register_recorder_cls(*provider*, *recorder_cls*)

register the recorder for the schema

Parameters::

- **provider** –
- **recorder_cls** –

timestamp

the meaning could be different for different case,most time it means 'happen time'

turnover

turnover_rate

volume

zvt.domain.quotes.future

Modules

zvt.domain.quotes.future.future_1d_kdata

---

zvt.domain.quotes.future.future_1d_kdata

Classes

`Future1dKdata`(**kwargs)

Future1dKdata

class zvt.domain.quotes.future.future_1d_kdata.Future1dKdata(**kwargs)

Bases: **Base**, **FutureKdataCommon**

__init__(**kwargs)

A simple constructor that allows initialization from kwargs.

Sets attributes on the constructed instance using the names and values in `kwargs`.

Only keys that are present as attributes of the instance's class are allowed. These could be, for example, any mapped columns or relationships.

change_pct

change_pct1

涨跌幅(按结算价)

close

code

entity_id

entity id

classmethod get_providers() → List[str]

providers of the schema defined by cls

Returns:: providers

classmethod get_storages(*provider: Optional[str] = None*)

get the storages info

Parameters:: **provider** – provider

Returns:: storages

high

id

id

interest

持仓量

level

low

name

open

provider

classmethod query_data(*provider_index: int = 0*, *ids: Optional[List[str]] = None*, *entity_ids: Optional[List[str]] = None*, *entity_id: Optional[str] = None*, *codes: Optional[List[str]] = None*, *code: Optional[str] = None*, *level: Optional[Union[IntervalLevel, str]] = None*, *provider: Optional[str] = None*, *columns: Optional[List] = None*, *col_label: Optional[dict] = None*, *return_type: str = 'df'*, *start_timestamp: Optional[Union[Timestamp, str]] = None*, *end_timestamp: Optional[Union[Timestamp, str]] = None*, *filters: Optional[List] = None*, *session: Optional[Session] = None*, *order=None*, *limit: Optional[int] = None*, *distinct=None*, *index: Optional[Union[str, list]] = None*, *drop_index_col=False*, *time_field: str = 'timestamp'*)

query data by the arguments

Parameters::

- **provider_index** –
- **data_schema** –
- **ids** –
- **entity_ids** –
- **entity_id** –
- **codes** –
- **code** –
- **level** –
- **provider** –
- **columns** –
- **col_label** – dict with key(column), value(label
- **return_type** – df, domain or dict. default is df
- **start_timestamp** –
- **end_timestamp** –
- **filters** –
- **session** –
- **order** –
- **limit** –
- **index** – index field name, str for single index, str list for multiple index
- **drop_index_col** – whether drop the col if it's in index, default False
- **time_field** –

Returns:: results basing on return_type.

classmethod record_data(*provider_index: int = 0*, *provider: Optional[str] = None*, *force_update=None*, *sleeping_time=None*, *exchanges=None*, *entity_id=None*, *entity_ids=None*, *code=None*, *codes=None*, *real_time=None*, *fix_duplicate_way=None*, *start_timestamp=None*, *end_timestamp=None*, *one_day_trading_minutes=None*, ***kwargs*)

record data by the arguments

Parameters::

- **entity_id** –
- **provider_index** –
- **provider** –
- **force_update** –
- **sleeping_time** –
- **exchanges** –
- **entity_ids** –
- **code** –
- **codes** –
- **real_time** –
- **fix_duplicate_way** –
- **start_timestamp** –
- **end_timestamp** –
- **one_day_trading_minutes** –
- **kwargs** –

Returns::

classmethod register_provider(*provider*)

register the provider to the schema defined by cls

Parameters:: **provider** –

classmethod register_recorder_cls(*provider*, *recorder_cls*)

register the recorder for the schema

Parameters::

- **provider** –
- **recorder_cls** –

settlement

结算价

timestamp

the meaning could be different for different case,most time it means 'happen time'

turnover

turnover_rate

volume

zvt.domain.quotes.index

Modules

zvt.domain.quotes.index.index_1d_kdata

---

zvt.domain.quotes.index.index_1wk_kdata

---

zvt.domain.quotes.index.index_1d_kdata

Classes

`Index1dKdata`(**kwargs)

Index1dKdata

class zvt.domain.quotes.index.index_1d_kdata.Index1dKdata(***kwargs*)

Bases: **Base**, **IndexKdataCommon**

__init__(***kwargs*)

A simple constructor that allows initialization from kwargs.

Sets attributes on the constructed instance using the names and values in `kwargs`.

Only keys that are present as attributes of the instance's class are allowed. These could be, for example, any mapped columns or relationships.

change_pct

close

code

entity_id

entity id

classmethod get_providers() → List[str]

providers of the schema defined by cls

Returns:: providers

classmethod get_storages(*provider: Optional[str] = None*)

get the storages info

Parameters:: **provider** – provider
Returns:: storages

high

id

id

level

low

name

open

provider

classmethod query_data(*provider_index: int = 0*, *ids: Optional[List[str]] = None*, *entity_ids: Optional[List[str]] = None*, *entity_id: Optional[str] = None*, *codes: Optional[List[str]] = None*, *code: Optional[str] = None*, *level: Optional[Union[IntervalLevel, str]] = None*, *provider: Optional[str] = None*, *columns: Optional[List] = None*, *col_label: Optional[dict] = None*, *return_type: str = 'df'*, *start_timestamp: Optional[Union[Timestamp, str]] = None*, *end_timestamp: Optional[Union[Timestamp, str]] = None*, *filters: Optional[List] = None*, *session: Optional[Session] = None*, *order=None*, *limit: Optional[int] = None*, *distinct=None*, *index: Optional[Union[str, list]] = None*, *drop_index_col=False*, *time_field: str = 'timestamp'*)

query data by the arguments

Parameters::

- **provider_index** –
- **data_schema** –
- **ids** –
- **entity_ids** –
- **entity_id** –
- **codes** –
- **code** –
- **level** –
- **provider** –
- **columns** –
- **col_label** – dict with key(column), value(label
- **return_type** – df, domain or dict. default is df
- **start_timestamp** –
- **end_timestamp** –
- **filters** –
- **session** –
- **order** –
- **limit** –
- **index** – index field name, str for single index, str list for multiple index
- **drop_index_col** – whether drop the col if it's in index, default False
- **time_field** –

Returns:: results basing on return_type.

classmethod record_data(*provider_index: int = 0*, *provider: Optional[str] = None*, *force_update=None*, *sleeping_time=None*, *exchanges=None*, *entity_id=None*, *entity_ids=None*, *code=None*, *codes=None*, *real_time=None*, *fix_duplicate_way=None*, *start_timestamp=None*, *end_timestamp=None*, *one_day_trading_minutes=None*, ***kwargs*)

record data by the arguments

Parameters::

- **entity_id** –
- **provider_index** –
- **provider** –
- **force_update** –
- **sleeping_time** –
- **exchanges** –
- **entity_ids** –
- **code** –
- **codes** –
- **real_time** –
- **fix_duplicate_way** –
- **start_timestamp** –
- **end_timestamp** –
- **one_day_trading_minutes** –
- **kwargs** –

Returns::

classmethod register_provider(*provider*)

register the provider to the schema defined by cls

Parameters:: **provider** –

classmethod register_recorder_cls(*provider*, *recorder_cls*)

register the recorder for the schema

Parameters::

- **provider** –
- **recorder_cls** –

timestamp

the meaning could be different for different case,most time it means 'happen time'

turnover

turnover_rate

volume

zvt.domain.quotes.index.index_1wk_kdata

Classes

`Index1wkKdata`(**kwargs)

Index1wkKdata

class zvt.domain.quotes.index.index_1wk_kdata.Index1wkKdata(***kwargs*)

Bases: **Base**, **IndexKdataCommon**

__init__(***kwargs*)

A simple constructor that allows initialization from kwargs.

Sets attributes on the constructed instance using the names and values in `kwargs`.

Only keys that are present as attributes of the instance's class are allowed. These could be, for example, any mapped columns or relationships.

change_pct

close

code

entity_id

entity id

classmethod get_providers() → List[str]

providers of the schema defined by cls

Returns:: providers

classmethod get_storages(*provider: Optional[str] = None*)

get the storages info

Parameters:: **provider** – provider
Returns:: storages

high

id

id

level

low

name

open

provider

classmethod query_data(*provider_index: int = 0*, *ids: Optional[List[str]] = None*, *entity_ids: Optional[List[str]] = None*, *entity_id: Optional[str] = None*, *codes: Optional[List[str]] = None*, *code: Optional[str] = None*, *level: Optional[Union[IntervalLevel, str]] = None*, *provider: Optional[str] = None*, *columns: Optional[List] = None*, *col_label: Optional[dict] = None*, *return_type: str = 'df'*, *start_timestamp: Optional[Union[Timestamp, str]] = None*, *end_timestamp: Optional[Union[Timestamp, str]] = None*, *filters: Optional[List] = None*, *session: Optional[Session] = None*, *order=None*, *limit: Optional[int] = None*, *distinct=None*, *index: Optional[Union[str, list]] = None*, *drop_index_col=False*, *time_field: str = 'timestamp'*)

query data by the arguments

Parameters::

- **provider_index** –
- **data_schema** –
- **ids** –
- **entity_ids** –
- **entity_id** –
- **codes** –
- **code** –
- **level** –
- **provider** –
- **columns** –
- **col_label** – dict with key(column), value(label
- **return_type** – df, domain or dict. default is df
- **start_timestamp** –
- **end_timestamp** –
- **filters** –
- **session** –
- **order** –
- **limit** –
- **index** – index field name, str for single index, str list for multiple index
- **drop_index_col** – whether drop the col if it's in index, default False
- **time_field** –

Returns:: results basing on return_type.

classmethod record_data(*provider_index: int = 0*, *provider: Optional[str] = None*, *force_update=None*, *sleeping_time=None*, *exchanges=None*, *entity_id=None*, *entity_ids=None*, *code=None*, *codes=None*, *real_time=None*, *fix_duplicate_way=None*, *start_timestamp=None*, *end_timestamp=None*, *one_day_trading_minutes=None*, ***kwargs*)

record data by the arguments

Parameters::

- **entity_id** –
- **provider_index** –
- **provider** –
- **force_update** –
- **sleeping_time** –
- **exchanges** –
- **entity_ids** –
- **code** –
- **codes** –
- **real_time** –
- **fix_duplicate_way** –
- **start_timestamp** –
- **end_timestamp** –
- **one_day_trading_minutes** –
- **kwargs** –

Returns::

classmethod register_provider(*provider*)

register the provider to the schema defined by cls

Parameters:: **provider** –

classmethod register_recorder_cls(*provider*, *recorder_cls*)

register the recorder for the schema

Parameters::

- **provider** –
- **recorder_cls** –

timestamp

the meaning could be different for different case,most time it means 'happen time'

turnover

turnover_rate

volume

zvt.domain.quotes.indexus

Modules

zvt.domain.quotes.indexus.indexus_1d_kdata

---

zvt.domain.quotes.indexus.indexus_1d_kdata

Classes

`Indexus1dKdata`(**kwargs)

Indexus1dKdata

class zvt.domain.quotes.indexus.indexus_1d_kdata.Indexus1dKdata(***kwargs*)

Bases: **`Base`**, **`IndexusKdataCommon`**

__init__(***kwargs*)

A simple constructor that allows initialization from kwargs.

Sets attributes on the constructed instance using the names and values in `kwargs`.

Only keys that are present as attributes of the instance's class are allowed. These could be, for example, any mapped columns or relationships.

change_pct

close

code

entity_id

entity id

classmethod get_providers() → List[str]

providers of the schema defined by cls

Returns:: providers

classmethod get_storages(*provider: Optional[str] = None*)

get the storages info

Parameters:: **provider** – provider
Returns:: storages

high

id

id

level

low

name

open

provider

classmethod query_data(*provider_index: int = 0*, *ids: Optional[List[str]] = None*, *entity_ids: Optional[List[str]] = None*, *entity_id: Optional[str] = None*, *codes: Optional[List[str]] = None*, *code: Optional[str] = None*, *level: Optional[Union[IntervalLevel, str]] = None*, *provider: Optional[str] = None*, *columns: Optional[List] = None*, *col_label: Optional[dict] = None*, *return_type: str = 'df'*, *start_timestamp: Optional[Union[Timestamp, str]] = None*, *end_timestamp: Optional[Union[Timestamp, str]] = None*, *filters: Optional[List] = None*, *session: Optional[Session] = None*, *order=None*, *limit: Optional[int] = None*, *distinct=None*, *index: Optional[Union[str, list]] = None*, *drop_index_col=False*, *time_field: str = 'timestamp'*)

query data by the arguments

Parameters::

- **provider_index** –
- **data_schema** –
- **ids** –
- **entity_ids** –
- **entity_id** –
- **codes** –
- **code** –
- **level** –
- **provider** –
- **columns** –
- **col_label** – dict with key(column), value(label
- **return_type** – df, domain or dict. default is df
- **start_timestamp** –
- **end_timestamp** –
- **filters** –
- **session** –
- **order** –
- **limit** –
- **index** – index field name, str for single index, str list for multiple index
- **drop_index_col** – whether drop the col if it's in index, default False
- **time_field** –

Returns:: results basing on return_type.

classmethod record_data(*provider_index: int = 0*, *provider: Optional[str] = None*, *force_update=None*, *sleeping_time=None*, *exchanges=None*, *entity_id=None*, *entity_ids=None*, *code=None*, *codes=None*, *real_time=None*, *fix_duplicate_way=None*, *start_timestamp=None*, *end_timestamp=None*, *one_day_trading_minutes=None*, ***kwargs*)

record data by the arguments

Parameters::

- **entity_id** –
- **provider_index** –
- **provider** –
- **force_update** –
- **sleeping_time** –
- **exchanges** –
- **entity_ids** –
- **code** –
- **codes** –
- **real_time** –
- **fix_duplicate_way** –
- **start_timestamp** –
- **end_timestamp** –
- **one_day_trading_minutes** –
- **kwargs** –

Returns::

classmethod register_provider(*provider*)

register the provider to the schema defined by cls

Parameters:: **provider** –

classmethod register_recorder_cls(*provider*, *recorder_cls*)

register the recorder for the schema

Parameters::

- **provider** –
- **recorder_cls** –

timestamp

the meaning could be different for different case,most time it means 'happen time'

turnover

turnover_rate

volume

zvt.domain.quotes.stock

Modules

zvt.domain.quotes.stock.stock_15m_hfq_kdata

---

zvt.domain.quotes.stock.stock_15m_kdata

---

zvt.domain.quotes.stock.stock_1d_hfq_kdata

---

zvt.domain.quotes.stock.stock_1d_kdata

---

zvt.domain.quotes.stock.stock_1h_hfq_kdata

---

zvt.domain.quotes.stock.stock_1h_kdata

---

zvt.domain.quotes.stock.stock_1m_hfq_kdata

---

zvt.domain.quotes.stock.stock_1m_kdata

---

zvt.domain.quotes.stock.stock_1mon_hfq_kdata

---

zvt.domain.quotes.stock.stock_1mon_kdata

---

zvt.domain.quotes.stock.stock_1wk_hfq_kdata

zvt.domain.quotes.stock.stock_1wk_kdata

zvt.domain.quotes.stock.stock_30m_hfq_kdata

zvt.domain.quotes.stock.stock_30m_kdata

zvt.domain.quotes.stock.stock_4h_hfq_kdata

zvt.domain.quotes.stock.stock_4h_kdata

zvt.domain.quotes.stock.stock_5m_hfq_kdata

zvt.domain.quotes.stock.stock_5m_kdata

zvt.domain.quotes.stock.stock_quote

zvt.domain.quotes.stock.stock_15m_hfq_kdata

Classes

`Stock15mHfqKdata`(**kwargs)

Stock15mHfqKdata

class zvt.domain.quotes.stock.stock_15m_hfq_kdata.Stock15mHfqKdata(**kwargs)

Bases: **Base**, **StockKdataCommon**

__init__(**kwargs)

A simple constructor that allows initialization from kwargs.

Sets attributes on the constructed instance using the names and values in `kwargs`.

Only keys that are present as attributes of the instance's class are allowed. These could be, for example, any mapped columns or relationships.

change_pct

close

code

entity_id

entity id

classmethod get_providers() → List[str]

providers of the schema defined by cls

Returns:: providers

classmethod get_storages(*provider: Optional[str] = None*)

get the storages info

Parameters:: **provider** – provider

Returns:: storages

high

id

id

level

low

name

open

provider

classmethod query_data(*provider_index: int = 0*, *ids: Optional[List[str]] = None*, *entity_ids: Optional[List[str]] = None*, *entity_id: Optional[str] = None*, *codes: Optional[List[str]] = None*, *code: Optional[str] = None*, *level: Optional[Union[IntervalLevel, str]] = None*, *provider: Optional[str] = None*, *columns: Optional[List] = None*, *col_label: Optional[dict] = None*, *return_type: str = 'df'*, *start_timestamp: Optional[Union[Timestamp, str]] = None*, *end_timestamp: Optional[Union[Timestamp, str]] = None*, *filters: Optional[List] = None*, *session: Optional[Session] = None*, *order=None*, *limit: Optional[int] = None*, *distinct=None*, *index: Optional[Union[str, list]] = None*, *drop_index_col=False*, *time_field: str = 'timestamp'*)

query data by the arguments

Parameters::

- **provider_index** –
- **data_schema** –
- **ids** –
- **entity_ids** –
- **entity_id** –
- **codes** –
- **code** –
- **level** –
- **provider** –
- **columns** –
- **col_label** – dict with key(column), value(label
- **return_type** – df, domain or dict. default is df
- **start_timestamp** –
- **end_timestamp** –
- **filters** –
- **session** –
- **order** –
- **limit** –
- **index** – index field name, str for single index, str list for multiple index
- **drop_index_col** – whether drop the col if it's in index, default False
- **time_field** –

Returns:: results basing on return_type.

classmethod record_data(*provider_index: int = 0*, *provider: Optional[str] = None*, *force_update=None*, *sleeping_time=None*, *exchanges=None*, *entity_id=None*, *entity_ids=None*, *code=None*, *codes=None*, *real_time=None*, *fix_duplicate_way=None*, *start_timestamp=None*, *end_timestamp=None*, *one_day_trading_minutes=None*, ***kwargs*)

record data by the arguments

Parameters::

- **entity_id** –
- **provider_index** –
- **provider** –
- **force_update** –
- **sleeping_time** –
- **exchanges** –
- **entity_ids** –
- **code** –
- **codes** –
- **real_time** –
- **fix_duplicate_way** –
- **start_timestamp** –
- **end_timestamp** –
- **one_day_trading_minutes** –
- **kwargs** –

Returns::

classmethod register_provider(*provider*)

register the provider to the schema defined by cls

Parameters:: **provider** –

classmethod register_recorder_cls(*provider*, *recorder_cls*)

register the recorder for the schema

Parameters::

- **provider** –
- **recorder_cls** –

timestamp

the meaning could be different for different case,most time it means 'happen time'

turnover

turnover_rate

volume

zvt.domain.quotes.stock.stock_15m_kdata

Classes

`Stock15mKdata`(**kwargs)

Stock15mKdata

class zvt.domain.quotes.stock.stock_15m_kdata.Stock15mKdata(**kwargs*)

Bases: **Base**, **StockKdataCommon**

__init__(**kwargs*)

A simple constructor that allows initialization from kwargs.

Sets attributes on the constructed instance using the names and values in `kwargs`.

Only keys that are present as attributes of the instance's class are allowed. These could be, for example, any mapped columns or relationships.

change_pct

close

code

entity_id

entity id

classmethod get_providers() → List[str]

providers of the schema defined by cls

Returns:: providers

classmethod get_storages(*provider: Optional[str] = None*)

get the storages info

Parameters:: **provider** – provider
Returns:: storages

high

id

id

level

low

name

open

provider

classmethod query_data(*provider_index: int = 0*, *ids: Optional[List[str]] = None*, *entity_ids: Optional[List[str]] = None*, *entity_id: Optional[str] = None*, *codes: Optional[List[str]] = None*, *code: Optional[str] = None*, *level: Optional[Union[IntervalLevel, str]] = None*, *provider: Optional[str] = None*, *columns: Optional[List] = None*, *col_label: Optional[dict] = None*, *return_type: str = 'df'*, *start_timestamp: Optional[Union[Timestamp, str]] = None*, *end_timestamp: Optional[Union[Timestamp, str]] = None*, *filters: Optional[List] = None*, *session: Optional[Session] = None*, *order=None*, *limit: Optional[int] = None*, *distinct=None*, *index: Optional[Union[str, list]] = None*, *drop_index_col=False*, *time_field: str = 'timestamp'*)

query data by the arguments

Parameters::

- **provider_index** –
- **data_schema** –
- **ids** –
- **entity_ids** –
- **entity_id** –
- **codes** –
- **code** –
- **level** –
- **provider** –
- **columns** –
- **col_label** – dict with key(column), value(label
- **return_type** – df, domain or dict. default is df
- **start_timestamp** –
- **end_timestamp** –
- **filters** –
- **session** –
- **order** –
- **limit** –
- **index** – index field name, str for single index, str list for multiple index
- **drop_index_col** – whether drop the col if it's in index, default False
- **time_field** –

Returns:: results basing on return_type.

classmethod record_data(*provider_index: int = 0*, *provider: Optional[str] = None*, *force_update=None*, *sleeping_time=None*, *exchanges=None*, *entity_id=None*, *entity_ids=None*, *code=None*, *codes=None*, *real_time=None*, *fix_duplicate_way=None*, *start_timestamp=None*, *end_timestamp=None*, *one_day_trading_minutes=None*, ***kwargs*)

record data by the arguments

Parameters::

- **entity_id** –
- **provider_index** –
- **provider** –
- **force_update** –
- **sleeping_time** –
- **exchanges** –
- **entity_ids** –
- **code** –
- **codes** –
- **real_time** –
- **fix_duplicate_way** –
- **start_timestamp** –
- **end_timestamp** –
- **one_day_trading_minutes** –
- **kwargs** –

Returns::

classmethod register_provider(*provider*)

register the provider to the schema defined by cls

Parameters:: **provider** –

classmethod register_recorder_cls(*provider*, *recorder_cls*)

register the recorder for the schema

Parameters::

- **provider** –
- **recorder_cls** –

timestamp

the meaning could be different for different case,most time it means 'happen time'

turnover

turnover_rate

volume

zvt.domain.quotes.stock.stock_1d_hfq_kdata

Classes

`Stock1dHfqKdata`(**kwargs)

Stock1dHfqKdata

class zvt.domain.quotes.stock.stock_1d_hfq_kdata.Stock1dHfqKdata(***kwargs*)

Bases: **Base**, **StockKdataCommon**

__init__(***kwargs*)

A simple constructor that allows initialization from kwargs.

Sets attributes on the constructed instance using the names and values in `kwargs`.

Only keys that are present as attributes of the instance's class are allowed. These could be, for example, any mapped columns or relationships.

change_pct

close

code

entity_id

entity id

classmethod get_providers() → List[str]

providers of the schema defined by cls

Returns:: providers

classmethod get_storages(*provider: Optional[str] = None*)

get the storages info

Parameters:: **provider** – provider
Returns:: storages

high

id

id

level

low

name

open

provider

classmethod query_data(*provider_index: int = 0*, *ids: Optional[List[str]] = None*, *entity_ids: Optional[List[str]] = None*, *entity_id: Optional[str] = None*, *codes: Optional[List[str]] = None*, *code: Optional[str] = None*, *level: Optional[Union[IntervalLevel, str]] = None*, *provider: Optional[str] = None*, *columns: Optional[List] = None*, *col_label: Optional[dict] = None*, *return_type: str = 'df'*, *start_timestamp: Optional[Union[Timestamp, str]] = None*, *end_timestamp: Optional[Union[Timestamp, str]] = None*, *filters: Optional[List] = None*, *session: Optional[Session] = None*, *order=None*, *limit: Optional[int] = None*, *distinct=None*, *index: Optional[Union[str, list]] = None*, *drop_index_col=False*, *time_field: str = 'timestamp'*)

query data by the arguments

Parameters::

- **provider_index** –
- **data_schema** –
- **ids** –
- **entity_ids** –
- **entity_id** –
- **codes** –
- **code** –
- **level** –
- **provider** –
- **columns** –
- **col_label** – dict with key(column), value(label
- **return_type** – df, domain or dict. default is df
- **start_timestamp** –
- **end_timestamp** –
- **filters** –
- **session** –
- **order** –
- **limit** –
- **index** – index field name, str for single index, str list for multiple index
- **drop_index_col** – whether drop the col if it's in index, default False
- **time_field** –

Returns:: results basing on return_type.

classmethod record_data(*provider_index: int = 0*, *provider: Optional[str] = None*, *force_update=None*, *sleeping_time=None*, *exchanges=None*, *entity_id=None*, *entity_ids=None*, *code=None*, *codes=None*, *real_time=None*, *fix_duplicate_way=None*, *start_timestamp=None*, *end_timestamp=None*, *one_day_trading_minutes=None*, ***kwargs*)

record data by the arguments

Parameters::

- **entity_id** –
- **provider_index** –
- **provider** –
- **force_update** –
- **sleeping_time** –
- **exchanges** –
- **entity_ids** –
- **code** –
- **codes** –
- **real_time** –
- **fix_duplicate_way** –
- **start_timestamp** –
- **end_timestamp** –
- **one_day_trading_minutes** –
- **kwargs** –

Returns::

classmethod register_provider(*provider*)

register the provider to the schema defined by cls

Parameters:: **provider** –

classmethod register_recorder_cls(*provider*, *recorder_cls*)

register the recorder for the schema

Parameters::

- **provider** –
- **recorder_cls** –

timestamp

the meaning could be different for different case,most time it means 'happen time'

turnover

turnover_rate

volume

zvt.domain.quotes.stock.stock_1d_kdata

Classes

`Stock1dKdata`(**kwargs)

Stock1dKdata

class zvt.domain.quotes.stock.stock_1d_kdata.Stock1dKdata(**kwargs*)

Bases: **Base**, **StockKdataCommon**

__init__(**kwargs*)

A simple constructor that allows initialization from kwargs.

Sets attributes on the constructed instance using the names and values in `kwargs`.

Only keys that are present as attributes of the instance's class are allowed. These could be, for example, any mapped columns or relationships.

change_pct

close

code

entity_id

entity id

classmethod get_providers() → List[str]

providers of the schema defined by cls

Returns:: providers

classmethod get_storages(*provider: Optional[str] = None*)

get the storages info

Parameters:: **provider** – provider
Returns:: storages

high

id

id

level

low

name

open

provider

classmethod query_data(*provider_index: int = 0*, *ids: Optional[List[str]] = None*, *entity_ids: Optional[List[str]] = None*, *entity_id: Optional[str] = None*, *codes: Optional[List[str]] = None*, *code: Optional[str] = None*, *level: Optional[Union[IntervalLevel, str]] = None*, *provider: Optional[str] = None*, *columns: Optional[List] = None*, *col_label: Optional[dict] = None*, *return_type: str = 'df'*, *start_timestamp: Optional[Union[Timestamp, str]] = None*, *end_timestamp: Optional[Union[Timestamp, str]] = None*, *filters: Optional[List] = None*, *session: Optional[Session] = None*, *order=None*, *limit: Optional[int] = None*, *distinct=None*, *index: Optional[Union[str, list]] = None*, *drop_index_col=False*, *time_field: str = 'timestamp'*)

query data by the arguments

Parameters::

- **provider_index** –
- **data_schema** –
- **ids** –
- **entity_ids** –
- **entity_id** –
- **codes** –
- **code** –
- **level** –
- **provider** –
- **columns** –
- **col_label** – dict with key(column), value(label
- **return_type** – df, domain or dict. default is df
- **start_timestamp** –
- **end_timestamp** –
- **filters** –
- **session** –
- **order** –
- **limit** –
- **index** – index field name, str for single index, str list for multiple index
- **drop_index_col** – whether drop the col if it's in index, default False
- **time_field** –

Returns:: results basing on return_type.

classmethod record_data(*provider_index: int = 0*, *provider: Optional[str] = None*, *force_update=None*, *sleeping_time=None*, *exchanges=None*, *entity_id=None*, *entity_ids=None*, *code=None*, *codes=None*, *real_time=None*, *fix_duplicate_way=None*, *start_timestamp=None*, *end_timestamp=None*, *one_day_trading_minutes=None*, ***kwargs*)

record data by the arguments

Parameters::

- **entity_id** –
- **provider_index** –
- **provider** –
- **force_update** –
- **sleeping_time** –
- **exchanges** –
- **entity_ids** –
- **code** –
- **codes** –
- **real_time** –
- **fix_duplicate_way** –
- **start_timestamp** –
- **end_timestamp** –
- **one_day_trading_minutes** –
- **kwargs** –

Returns::

classmethod register_provider(*provider*)

register the provider to the schema defined by cls

Parameters:: **provider** –

classmethod register_recorder_cls(*provider*, *recorder_cls*)

register the recorder for the schema

Parameters::

- **provider** –
- **recorder_cls** –

timestamp

the meaning could be different for different case,most time it means 'happen time'

turnover

turnover_rate

volume

zvt.domain.quotes.stock.stock_1h_hfq_kdata

Classes

`Stock1hHfqKdata`(**kwargs)

Stock1hHfqKdata

class zvt.domain.quotes.stock.stock_1h_hfq_kdata.Stock1hHfqKdata(***kwargs*)

Bases: **Base**, **StockKdataCommon**

__init__(***kwargs*)

A simple constructor that allows initialization from kwargs.

Sets attributes on the constructed instance using the names and values in `kwargs`.

Only keys that are present as attributes of the instance's class are allowed. These could be, for example, any mapped columns or relationships.

change_pct

close

code

entity_id

entity id

classmethod get_providers() → List[str]

providers of the schema defined by cls

Returns:: providers

classmethod get_storages(*provider: Optional[str] = None*)

get the storages info

Parameters:: **provider** – provider
Returns:: storages

high

id

id

level

low

name

open

provider

classmethod query_data(*provider_index: int = 0*, *ids: Optional[List[str]] = None*, *entity_ids: Optional[List[str]] = None*, *entity_id: Optional[str] = None*, *codes: Optional[List[str]] = None*, *code: Optional[str] = None*, *level: Optional[Union[IntervalLevel, str]] = None*, *provider: Optional[str] = None*, *columns: Optional[List] = None*, *col_label: Optional[dict] = None*, *return_type: str = 'df'*, *start_timestamp: Optional[Union[Timestamp, str]] = None*, *end_timestamp: Optional[Union[Timestamp, str]] = None*, *filters: Optional[List] = None*, *session: Optional[Session] = None*, *order=None*, *limit: Optional[int] = None*, *distinct=None*, *index: Optional[Union[str, list]] = None*, *drop_index_col=False*, *time_field: str = 'timestamp'*)

query data by the arguments

Parameters::

- **provider_index** –
- **data_schema** –
- **ids** –
- **entity_ids** –
- **entity_id** –
- **codes** –
- **code** –
- **level** –
- **provider** –
- **columns** –
- **col_label** – dict with key(column), value(label
- **return_type** – df, domain or dict. default is df
- **start_timestamp** –
- **end_timestamp** –
- **filters** –
- **session** –
- **order** –
- **limit** –
- **index** – index field name, str for single index, str list for multiple index
- **drop_index_col** – whether drop the col if it's in index, default False
- **time_field** –

Returns:: results basing on return_type.

classmethod record_data(*provider_index: int = 0*, *provider: Optional[str] = None*, *force_update=None*, *sleeping_time=None*, *exchanges=None*, *entity_id=None*, *entity_ids=None*, *code=None*, *codes=None*, *real_time=None*, *fix_duplicate_way=None*, *start_timestamp=None*, *end_timestamp=None*, *one_day_trading_minutes=None*, ***kwargs*)

record data by the arguments

Parameters::

- **entity_id** –
- **provider_index** –
- **provider** –
- **force_update** –
- **sleeping_time** –
- **exchanges** –
- **entity_ids** –
- **code** –
- **codes** –
- **real_time** –
- **fix_duplicate_way** –
- **start_timestamp** –
- **end_timestamp** –
- **one_day_trading_minutes** –
- **kwargs** –

Returns::

classmethod register_provider(*provider*)

register the provider to the schema defined by cls

Parameters:: **provider** –

classmethod register_recorder_cls(*provider*, *recorder_cls*)

register the recorder for the schema

Parameters::

- **provider** –
- **recorder_cls** –

timestamp

the meaning could be different for different case,most time it means 'happen time'

turnover

turnover_rate

volume

zvt.domain.quotes.stock.stock_1h_kdata

Classes

`Stock1hKdata`(**kwargs)

Stock1hKdata

class zvt.domain.quotes.stock.stock_1h_kdata.Stock1hKdata(**kwargs*)

Bases: **Base**, **StockKdataCommon**

__init__(***kwargs*)

A simple constructor that allows initialization from kwargs.

Sets attributes on the constructed instance using the names and values in `kwargs`.

Only keys that are present as attributes of the instance's class are allowed. These could be, for example, any mapped columns or relationships.

change_pct

close

code

entity_id

entity id

classmethod get_providers() → List[str]

providers of the schema defined by cls

Returns:: providers

classmethod get_storages(*provider: Optional[str] = None*)

get the storages info

Parameters:: **provider** – provider
Returns:: storages

high

id

id

level

low

name

open

provider

classmethod query_data(*provider_index: int = 0*, *ids: Optional[List[str]] = None*, *entity_ids: Optional[List[str]] = None*, *entity_id: Optional[str] = None*, *codes: Optional[List[str]] = None*, *code: Optional[str] = None*, *level: Optional[Union[IntervalLevel, str]] = None*, *provider: Optional[str] = None*, *columns: Optional[List] = None*, *col_label: Optional[dict] = None*, *return_type: str = 'df'*, *start_timestamp: Optional[Union[Timestamp, str]] = None*, *end_timestamp: Optional[Union[Timestamp, str]] = None*, *filters: Optional[List] = None*, *session: Optional[Session] = None*, *order=None*, *limit: Optional[int] = None*, *distinct=None*, *index: Optional[Union[str, list]] = None*, *drop_index_col=False*, *time_field: str = 'timestamp'*)

query data by the arguments

Parameters::

- **provider_index** –
- **data_schema** –
- **ids** –
- **entity_ids** –
- **entity_id** –
- **codes** –
- **code** –
- **level** –
- **provider** –
- **columns** –
- **col_label** – dict with key(column), value(label
- **return_type** – df, domain or dict. default is df
- **start_timestamp** –
- **end_timestamp** –
- **filters** –
- **session** –
- **order** –
- **limit** –
- **index** – index field name, str for single index, str list for multiple index
- **drop_index_col** – whether drop the col if it's in index, default False
- **time_field** –

Returns:: results basing on return_type.

classmethod record_data(*provider_index: int = 0*, *provider: Optional[str] = None*, *force_update=None*, *sleeping_time=None*, *exchanges=None*, *entity_id=None*, *entity_ids=None*, *code=None*, *codes=None*, *real_time=None*, *fix_duplicate_way=None*, *start_timestamp=None*, *end_timestamp=None*, *one_day_trading_minutes=None*, ***kwargs*)

record data by the arguments

Parameters::

- **entity_id** –
- **provider_index** –
- **provider** –
- **force_update** –
- **sleeping_time** –
- **exchanges** –
- **entity_ids** –
- **code** –
- **codes** –
- **real_time** –
- **fix_duplicate_way** –
- **start_timestamp** –
- **end_timestamp** –
- **one_day_trading_minutes** –
- **kwargs** –

Returns::

classmethod register_provider(*provider*)

register the provider to the schema defined by cls

Parameters:: **provider** –

classmethod register_recorder_cls(*provider*, *recorder_cls*)

register the recorder for the schema

Parameters::

- **provider** –
- **recorder_cls** –

timestamp

the meaning could be different for different case,most time it means 'happen time'

turnover

turnover_rate

volume

zvt.domain.quotes.stock.stock_1m_hfq_kdata

Classes

`Stock1mHfqKdata`(**kwargs)

Stock1mHfqKdata

class zvt.domain.quotes.stock.stock_1m_hfq_kdata.Stock1mHfqKdata(**kwargs)

Bases: **Base**, **StockKdataCommon**

__init__(**kwargs)

A simple constructor that allows initialization from kwargs.

Sets attributes on the constructed instance using the names and values in `kwargs`.

Only keys that are present as attributes of the instance's class are allowed. These could be, for example, any mapped columns or relationships.

change_pct

close

code

entity_id

entity id

classmethod get_providers() → List[str]

providers of the schema defined by cls

Returns:: providers

classmethod get_storages(*provider: Optional[str] = None*)

get the storages info

Parameters:: **provider** – provider
Returns:: storages

high

id

id

level

low

name

open

provider

classmethod query_data(*provider_index: int = 0*, *ids: Optional[List[str]] = None*, *entity_ids: Optional[List[str]] = None*, *entity_id: Optional[str] = None*, *codes: Optional[List[str]] = None*, *code: Optional[str] = None*, *level: Optional[Union[IntervalLevel, str]] = None*, *provider: Optional[str] = None*, *columns: Optional[List] = None*, *col_label: Optional[dict] = None*, *return_type: str = 'df'*, *start_timestamp: Optional[Union[Timestamp, str]] = None*, *end_timestamp: Optional[Union[Timestamp, str]] = None*, *filters: Optional[List] = None*, *session: Optional[Session] = None*, *order=None*, *limit: Optional[int] = None*, *distinct=None*, *index: Optional[Union[str, list]] = None*, *drop_index_col=False*, *time_field: str = 'timestamp'*)

query data by the arguments

Parameters::

- **provider_index** –
- **data_schema** –
- **ids** –
- **entity_ids** –
- **entity_id** –
- **codes** –
- **code** –
- **level** –
- **provider** –
- **columns** –
- **col_label** – dict with key(column), value(label
- **return_type** – df, domain or dict. default is df
- **start_timestamp** –
- **end_timestamp** –
- **filters** –
- **session** –
- **order** –
- **limit** –
- **index** – index field name, str for single index, str list for multiple index
- **drop_index_col** – whether drop the col if it's in index, default False
- **time_field** –

Returns:: results basing on return_type.

classmethod record_data(*provider_index: int = 0*, *provider: Optional[str] = None*, *force_update=None*, *sleeping_time=None*, *exchanges=None*, *entity_id=None*, *entity_ids=None*, *code=None*, *codes=None*, *real_time=None*, *fix_duplicate_way=None*, *start_timestamp=None*, *end_timestamp=None*, *one_day_trading_minutes=None*, ***kwargs*)

record data by the arguments

Parameters::

- **entity_id** –
- **provider_index** –
- **provider** –
- **force_update** –
- **sleeping_time** –
- **exchanges** –
- **entity_ids** –
- **code** –
- **codes** –
- **real_time** –
- **fix_duplicate_way** –
- **start_timestamp** –
- **end_timestamp** –
- **one_day_trading_minutes** –
- **kwargs** –

Returns::

classmethod register_provider(*provider*)

register the provider to the schema defined by cls

Parameters:: **provider** –

classmethod register_recorder_cls(*provider*, *recorder_cls*)

register the recorder for the schema

Parameters::

- **provider** –
- **recorder_cls** –

timestamp

the meaning could be different for different case,most time it means 'happen time'

turnover

turnover_rate

volume

zvt.domain.quotes.stock.stock_1m_kdata

Classes

`Stock1mKdata`(**kwargs)

Stock1mKdata

class zvt.domain.quotes.stock.stock_1m_kdata.Stock1mKdata(***kwargs*)

Bases: **Base**, **StockKdataCommon**

__init__(***kwargs*)

A simple constructor that allows initialization from kwargs.

Sets attributes on the constructed instance using the names and values in `kwargs`.

Only keys that are present as attributes of the instance's class are allowed. These could be, for example, any mapped columns or relationships.

change_pct

close

code

entity_id

entity id

classmethod get_providers() → List[str]

providers of the schema defined by cls

Returns:: providers

classmethod get_storages(*provider: Optional[str] = None*)

get the storages info

Parameters:: **provider** – provider
Returns:: storages

high

id

id

level

low

name

open

provider

classmethod query_data(*provider_index: int = 0*, *ids: Optional[List[str]] = None*, *entity_ids: Optional[List[str]] = None*, *entity_id: Optional[str] = None*, *codes: Optional[List[str]] = None*, *code: Optional[str] = None*, *level: Optional[Union[IntervalLevel, str]] = None*, *provider: Optional[str] = None*, *columns: Optional[List] = None*, *col_label: Optional[dict] = None*, *return_type: str = 'df'*, *start_timestamp: Optional[Union[Timestamp, str]] = None*, *end_timestamp: Optional[Union[Timestamp, str]] = None*, *filters: Optional[List] = None*, *session: Optional[Session] = None*, *order=None*, *limit: Optional[int] = None*, *distinct=None*, *index: Optional[Union[str, list]] = None*, *drop_index_col=False*, *time_field: str = 'timestamp'*)

query data by the arguments

Parameters::

- **provider_index** –
- **data_schema** –
- **ids** –
- **entity_ids** –
- **entity_id** –
- **codes** –
- **code** –
- **level** –
- **provider** –
- **columns** –
- **col_label** – dict with key(column), value(label
- **return_type** – df, domain or dict. default is df
- **start_timestamp** –
- **end_timestamp** –
- **filters** –
- **session** –
- **order** –
- **limit** –
- **index** – index field name, str for single index, str list for multiple index
- **drop_index_col** – whether drop the col if it's in index, default False
- **time_field** –

Returns:: results basing on return_type.

classmethod record_data(*provider_index: int = 0*, *provider: Optional[str] = None*, *force_update=None*, *sleeping_time=None*, *exchanges=None*, *entity_id=None*, *entity_ids=None*, *code=None*, *codes=None*, *real_time=None*, *fix_duplicate_way=None*, *start_timestamp=None*, *end_timestamp=None*, *one_day_trading_minutes=None*, ***kwargs*)

record data by the arguments

Parameters::

- **entity_id** –
- **provider_index** –
- **provider** –
- **force_update** –
- **sleeping_time** –
- **exchanges** –
- **entity_ids** –
- **code** –
- **codes** –
- **real_time** –
- **fix_duplicate_way** –
- **start_timestamp** –
- **end_timestamp** –
- **one_day_trading_minutes** –
- **kwargs** –

Returns::

classmethod register_provider(*provider*)

register the provider to the schema defined by cls

Parameters:: **provider** –

classmethod register_recorder_cls(*provider*, *recorder_cls*)

register the recorder for the schema

Parameters::

- **provider** –
- **recorder_cls** –

timestamp

the meaning could be different for different case,most time it means 'happen time'

turnover

turnover_rate

volume

zvt.domain.quotes.stock.stock_1mon_hfq_kdata

Classes

`Stock1monHfqKdata`(**kwargs)

Stock1monHfqKdata

class zvt.domain.quotes.stock.stock_1mon_hfq_kdata.Stock1monHfqKdata(***kwargs*)

Bases: **Base**, **StockKdataCommon**

__init__(***kwargs*)

A simple constructor that allows initialization from kwargs.

Sets attributes on the constructed instance using the names and values in `kwargs`.

Only keys that are present as attributes of the instance's class are allowed. These could be, for example, any mapped columns or relationships.

change_pct

close

code

entity_id

entity id

classmethod get_providers() → List[str]

providers of the schema defined by cls

Returns:: providers

classmethod get_storages(*provider: Optional[str] = None*)

get the storages info

Parameters:: **provider** – provider
Returns:: storages

high

id

id

level

low

name

open

provider

classmethod query_data(*provider_index: int = 0*, *ids: Optional[List[str]] = None*, *entity_ids: Optional[List[str]] = None*, *entity_id: Optional[str] = None*, *codes: Optional[List[str]] = None*, *code: Optional[str] = None*, *level: Optional[Union[IntervalLevel, str]] = None*, *provider: Optional[str] = None*, *columns: Optional[List] = None*, *col_label: Optional[dict] = None*, *return_type: str = 'df'*, *start_timestamp: Optional[Union[Timestamp, str]] = None*, *end_timestamp: Optional[Union[Timestamp, str]] = None*, *filters: Optional[List] = None*, *session: Optional[Session] = None*, *order=None*, *limit: Optional[int] = None*, *distinct=None*, *index: Optional[Union[str, list]] = None*, *drop_index_col=False*, *time_field: str = 'timestamp'*)

query data by the arguments

Parameters::

- **provider_index** –
- **data_schema** –
- **ids** –
- **entity_ids** –
- **entity_id** –
- **codes** –
- **code** –
- **level** –
- **provider** –
- **columns** –
- **col_label** – dict with key(column), value(label
- **return_type** – df, domain or dict. default is df
- **start_timestamp** –
- **end_timestamp** –
- **filters** –
- **session** –
- **order** –
- **limit** –
- **index** – index field name, str for single index, str list for multiple index
- **drop_index_col** – whether drop the col if it's in index, default False
- **time_field** –

Returns:: results basing on return_type.

classmethod record_data(*provider_index: int = 0*, *provider: Optional[str] = None*, *force_update=None*, *sleeping_time=None*, *exchanges=None*, *entity_id=None*, *entity_ids=None*, *code=None*, *codes=None*, *real_time=None*, *fix_duplicate_way=None*, *start_timestamp=None*, *end_timestamp=None*, *one_day_trading_minutes=None*, ***kwargs*)

record data by the arguments

Parameters::

- **entity_id** –
- **provider_index** –
- **provider** –
- **force_update** –
- **sleeping_time** –
- **exchanges** –
- **entity_ids** –
- **code** –
- **codes** –
- **real_time** –
- **fix_duplicate_way** –
- **start_timestamp** –
- **end_timestamp** –
- **one_day_trading_minutes** –
- **kwargs** –

Returns::

classmethod register_provider(*provider*)

register the provider to the schema defined by cls

Parameters:: **provider** –

classmethod register_recorder_cls(*provider*, *recorder_cls*)

register the recorder for the schema

Parameters::

- **provider** –
- **recorder_cls** –

timestamp

the meaning could be different for different case,most time it means 'happen time'

turnover

turnover_rate

volume

zvt.domain.quotes.stock.stock_1mon_kdata

Classes

`Stock1monKdata`(**kwargs)

Stock1monKdata

class zvt.domain.quotes.stock.stock_1mon_kdata.Stock1monKdata(**kwargs)

Bases: **Base**, **StockKdataCommon**

__init__(**kwargs)

A simple constructor that allows initialization from kwargs.

Sets attributes on the constructed instance using the names and values in `kwargs`.

Only keys that are present as attributes of the instance's class are allowed. These could be, for example, any mapped columns or relationships.

change_pct

close

code

entity_id

entity id

classmethod get_providers() → List[str]

providers of the schema defined by cls

Returns:: providers

classmethod get_storages(*provider: Optional[str] = None*)

get the storages info

Parameters:: **provider** – provider
Returns:: storages

high

id

id

level

low

name

open

provider

classmethod query_data(*provider_index: int = 0*, *ids: Optional[List[str]] = None*, *entity_ids: Optional[List[str]] = None*, *entity_id: Optional[str] = None*, *codes: Optional[List[str]] = None*, *code: Optional[str] = None*, *level: Optional[Union[IntervalLevel, str]] = None*, *provider: Optional[str] = None*, *columns: Optional[List] = None*, *col_label: Optional[dict] = None*, *return_type: str = 'df'*, *start_timestamp: Optional[Union[Timestamp, str]] = None*, *end_timestamp: Optional[Union[Timestamp, str]] = None*, *filters: Optional[List] = None*, *session: Optional[Session] = None*, *order=None*, *limit: Optional[int] = None*, *distinct=None*, *index: Optional[Union[str, list]] = None*, *drop_index_col=False*, *time_field: str = 'timestamp'*)

query data by the arguments

Parameters::

- **provider_index** –
- **data_schema** –
- **ids** –
- **entity_ids** –
- **entity_id** –
- **codes** –
- **code** –
- **level** –
- **provider** –
- **columns** –
- **col_label** – dict with key(column), value(label
- **return_type** – df, domain or dict. default is df
- **start_timestamp** –
- **end_timestamp** –
- **filters** –
- **session** –
- **order** –
- **limit** –
- **index** – index field name, str for single index, str list for multiple index
- **drop_index_col** – whether drop the col if it's in index, default False
- **time_field** –

Returns:: results basing on return_type.

classmethod record_data(*provider_index: int = 0*, *provider: Optional[str] = None*, *force_update=None*, *sleeping_time=None*, *exchanges=None*, *entity_id=None*, *entity_ids=None*, *code=None*, *codes=None*, *real_time=None*, *fix_duplicate_way=None*, *start_timestamp=None*, *end_timestamp=None*, *one_day_trading_minutes=None*, ***kwargs*)

record data by the arguments

Parameters::

- **entity_id** –
- **provider_index** –
- **provider** –
- **force_update** –
- **sleeping_time** –
- **exchanges** –
- **entity_ids** –
- **code** –
- **codes** –
- **real_time** –
- **fix_duplicate_way** –
- **start_timestamp** –
- **end_timestamp** –
- **one_day_trading_minutes** –
- **kwargs** –

Returns::

classmethod register_provider(*provider*)

register the provider to the schema defined by cls

Parameters:: **provider** –

classmethod register_recorder_cls(*provider*, *recorder_cls*)

register the recorder for the schema

Parameters::

- **provider** –
- **recorder_cls** –

timestamp

the meaning could be different for different case,most time it means 'happen time'

turnover

turnover_rate

volume

zvt.domain.quotes.stock.stock_1wk_hfq_kdata

Classes

`Stock1wkHfqKdata`(**kwargs)

Stock1wkHfqKdata

class zvt.domain.quotes.stock.stock_1wk_hfq_kdata.Stock1wkHfqKdata(**kwargs)

Bases: **Base**, **StockKdataCommon**

__init__(**kwargs)

A simple constructor that allows initialization from kwargs.

Sets attributes on the constructed instance using the names and values in `kwargs`.

Only keys that are present as attributes of the instance's class are allowed. These could be, for example, any mapped columns or relationships.

change_pct

close

code

entity_id

entity id

classmethod get_providers() → List[str]

providers of the schema defined by cls

Returns:: providers

classmethod get_storages(*provider: Optional[str] = None*)

get the storages info

Parameters:: **provider** – provider
Returns:: storages

high

id

id

level

low

name

open

provider

classmethod query_data(*provider_index: int = 0*, *ids: Optional[List[str]] = None*, *entity_ids: Optional[List[str]] = None*, *entity_id: Optional[str] = None*, *codes: Optional[List[str]] = None*, *code: Optional[str] = None*, *level: Optional[Union[IntervalLevel, str]] = None*, *provider: Optional[str] = None*, *columns: Optional[List] = None*, *col_label: Optional[dict] = None*, *return_type: str = 'df'*, *start_timestamp: Optional[Union[Timestamp, str]] = None*, *end_timestamp: Optional[Union[Timestamp, str]] = None*, *filters: Optional[List] = None*, *session: Optional[Session] = None*, *order=None*, *limit: Optional[int] = None*, *distinct=None*, *index: Optional[Union[str, list]] = None*, *drop_index_col=False*, *time_field: str = 'timestamp'*)

query data by the arguments

Parameters::

- **provider_index** –
- **data_schema** –
- **ids** –
- **entity_ids** –
- **entity_id** –
- **codes** –
- **code** –
- **level** –
- **provider** –
- **columns** –
- **col_label** – dict with key(column), value(label
- **return_type** – df, domain or dict. default is df
- **start_timestamp** –
- **end_timestamp** –
- **filters** –
- **session** –
- **order** –
- **limit** –
- **index** – index field name, str for single index, str list for multiple index
- **drop_index_col** – whether drop the col if it's in index, default False
- **time_field** –

Returns:: results basing on return_type.

classmethod record_data(*provider_index: int = 0*, *provider: Optional[str] = None*, *force_update=None*, *sleeping_time=None*, *exchanges=None*, *entity_id=None*, *entity_ids=None*, *code=None*, *codes=None*, *real_time=None*, *fix_duplicate_way=None*, *start_timestamp=None*, *end_timestamp=None*, *one_day_trading_minutes=None*, ***kwargs*)

record data by the arguments

Parameters::

- **entity_id** –
- **provider_index** –
- **provider** –
- **force_update** –
- **sleeping_time** –
- **exchanges** –
- **entity_ids** –
- **code** –
- **codes** –
- **real_time** –
- **fix_duplicate_way** –
- **start_timestamp** –
- **end_timestamp** –
- **one_day_trading_minutes** –
- **kwargs** –

Returns::

classmethod register_provider(*provider*)

register the provider to the schema defined by cls

Parameters:: **provider** –

classmethod register_recorder_cls(*provider*, *recorder_cls*)

register the recorder for the schema

Parameters::

- **provider** –
- **recorder_cls** –

timestamp

the meaning could be different for different case,most time it means 'happen time'

turnover

turnover_rate

volume

zvt.domain.quotes.stock.stock_1wk_kdata

Classes

`Stock1wkKdata`(**kwargs)

Stock1wkKdata

class zvt.domain.quotes.stock.stock_1wk_kdata.Stock1wkKdata(**kwargs*)

Bases: **Base**, **StockKdataCommon**

__init__(**kwargs*)

A simple constructor that allows initialization from kwargs.

Sets attributes on the constructed instance using the names and values in `kwargs`.

Only keys that are present as attributes of the instance's class are allowed. These could be, for example, any mapped columns or relationships.

change_pct

close

code

entity_id

entity id

classmethod get_providers() → List[str]

providers of the schema defined by cls

Returns:: providers

classmethod get_storages(*provider: Optional[str] = None*)

get the storages info

Parameters:: **provider** – provider
Returns:: storages

high

id

id

level

low

name

open

provider

classmethod query_data(*provider_index: int = 0*, *ids: Optional[List[str]] = None*, *entity_ids: Optional[List[str]] = None*, *entity_id: Optional[str] = None*, *codes: Optional[List[str]] = None*, *code: Optional[str] = None*, *level: Optional[Union[IntervalLevel, str]] = None*, *provider: Optional[str] = None*, *columns: Optional[List] = None*, *col_label: Optional[dict] = None*, *return_type: str = 'df'*, *start_timestamp: Optional[Union[Timestamp, str]] = None*, *end_timestamp: Optional[Union[Timestamp, str]] = None*, *filters: Optional[List] = None*, *session: Optional[Session] = None*, *order=None*, *limit: Optional[int] = None*, *distinct=None*, *index: Optional[Union[str, list]] = None*, *drop_index_col=False*, *time_field: str = 'timestamp'*)

query data by the arguments

Parameters::

- **provider_index** –
- **data_schema** –
- **ids** –
- **entity_ids** –
- **entity_id** –
- **codes** –
- **code** –
- **level** –
- **provider** –
- **columns** –
- **col_label** – dict with key(column), value(label
- **return_type** – df, domain or dict. default is df
- **start_timestamp** –
- **end_timestamp** –
- **filters** –
- **session** –
- **order** –
- **limit** –
- **index** – index field name, str for single index, str list for multiple index
- **drop_index_col** – whether drop the col if it's in index, default False
- **time_field** –

Returns:: results basing on return_type.

classmethod record_data(*provider_index: int = 0*, *provider: Optional[str] = None*, *force_update=None*, *sleeping_time=None*, *exchanges=None*, *entity_id=None*, *entity_ids=None*, *code=None*, *codes=None*, *real_time=None*, *fix_duplicate_way=None*, *start_timestamp=None*, *end_timestamp=None*, *one_day_trading_minutes=None*, ***kwargs*)

record data by the arguments

Parameters::

- **entity_id** –
- **provider_index** –
- **provider** –
- **force_update** –
- **sleeping_time** –
- **exchanges** –
- **entity_ids** –
- **code** –
- **codes** –
- **real_time** –
- **fix_duplicate_way** –
- **start_timestamp** –
- **end_timestamp** –
- **one_day_trading_minutes** –
- **kwargs** –

Returns::

classmethod register_provider(*provider*)

register the provider to the schema defined by cls

Parameters:: **provider** –

classmethod register_recorder_cls(*provider*, *recorder_cls*)

register the recorder for the schema

Parameters::

- **provider** –
- **recorder_cls** –

timestamp

the meaning could be different for different case,most time it means 'happen time'

turnover

turnover_rate

volume

zvt.domain.quotes.stock.stock_30m_hfq_kdata

Classes

`Stock30mHfqKdata`(**kwargs)

Stock30mHfqKdata

class zvt.domain.quotes.stock.stock_30m_hfq_kdata.Stock30mHfqKdata(***kwargs*)

Bases: **Base**, **StockKdataCommon**

__init__(***kwargs*)

A simple constructor that allows initialization from kwargs.

Sets attributes on the constructed instance using the names and values in `kwargs`.

Only keys that are present as attributes of the instance's class are allowed. These could be, for example, any mapped columns or relationships.

change_pct

close

code

entity_id

entity id

classmethod get_providers() → List[str]

providers of the schema defined by cls

Returns:: providers

classmethod get_storages(*provider: Optional[str] = None*)

get the storages info

Parameters:: **provider** – provider

Returns:: storages

high

id

id

level

low

name

open

provider

classmethod query_data(*provider_index: int = 0*, *ids: Optional[List[str]] = None*, *entity_ids: Optional[List[str]] = None*, *entity_id: Optional[str] = None*, *codes: Optional[List[str]] = None*, *code: Optional[str] = None*, *level: Optional[Union[IntervalLevel, str]] = None*, *provider: Optional[str] = None*, *columns: Optional[List] = None*, *col_label: Optional[dict] = None*, *return_type: str = 'df'*, *start_timestamp: Optional[Union[Timestamp, str]] = None*, *end_timestamp: Optional[Union[Timestamp, str]] = None*, *filters: Optional[List] = None*, *session: Optional[Session] = None*, *order=None*, *limit: Optional[int] = None*, *distinct=None*, *index: Optional[Union[str, list]] = None*, *drop_index_col=False*, *time_field: str = 'timestamp'*)

query data by the arguments

Parameters::

- **provider_index** –
- **data_schema** –
- **ids** –
- **entity_ids** –
- **entity_id** –
- **codes** –
- **code** –
- **level** –
- **provider** –
- **columns** –
- **col_label** – dict with key(column), value(label
- **return_type** – df, domain or dict. default is df
- **start_timestamp** –
- **end_timestamp** –
- **filters** –
- **session** –
- **order** –
- **limit** –
- **index** – index field name, str for single index, str list for multiple index
- **drop_index_col** – whether drop the col if it's in index, default False
- **time_field** –

Returns:: results basing on return_type.

classmethod record_data(*provider_index: int = 0*, *provider: Optional[str] = None*, *force_update=None*, *sleeping_time=None*, *exchanges=None*, *entity_id=None*, *entity_ids=None*, *code=None*, *codes=None*, *real_time=None*, *fix_duplicate_way=None*, *start_timestamp=None*, *end_timestamp=None*, *one_day_trading_minutes=None*, ***kwargs*)

record data by the arguments

Parameters::

- **entity_id** –
- **provider_index** –
- **provider** –
- **force_update** –
- **sleeping_time** –
- **exchanges** –
- **entity_ids** –
- **code** –
- **codes** –
- **real_time** –
- **fix_duplicate_way** –
- **start_timestamp** –
- **end_timestamp** –
- **one_day_trading_minutes** –
- **kwargs** –

Returns::

classmethod register_provider(*provider*)

register the provider to the schema defined by cls

Parameters:: **provider** –

classmethod register_recorder_cls(*provider*, *recorder_cls*)

register the recorder for the schema

Parameters::

- **provider** –
- **recorder_cls** –

timestamp

the meaning could be different for different case,most time it means 'happen time'

turnover

turnover_rate

volume

zvt.domain.quotes.stock.stock_30m_kdata

Classes

`Stock30mKdata`(**kwargs)

Stock30mKdata

class zvt.domain.quotes.stock.stock_30m_kdata.Stock30mKdata(**kwargs*)

Bases: **Base**, **StockKdataCommon**

__init__(**kwargs*)

A simple constructor that allows initialization from kwargs.

Sets attributes on the constructed instance using the names and values in `kwargs`.

Only keys that are present as attributes of the instance's class are allowed. These could be, for example, any mapped columns or relationships.

change_pct

close

code

entity_id

entity id

classmethod get_providers() → List[str]

providers of the schema defined by cls

Returns:: providers

classmethod get_storages(*provider: Optional[str] = None*)

get the storages info

Parameters:: **provider** – provider
Returns:: storages

high

id

id

level

low

name

open

provider

classmethod query_data(*provider_index: int = 0*, *ids: Optional[List[str]] = None*, *entity_ids: Optional[List[str]] = None*, *entity_id: Optional[str] = None*, *codes: Optional[List[str]] = None*, *code: Optional[str] = None*, *level: Optional[Union[IntervalLevel, str]] = None*, *provider: Optional[str] = None*, *columns: Optional[List] = None*, *col_label: Optional[dict] = None*, *return_type: str = 'df'*, *start_timestamp: Optional[Union[Timestamp, str]] = None*, *end_timestamp: Optional[Union[Timestamp, str]] = None*, *filters: Optional[List] = None*, *session: Optional[Session] = None*, *order=None*, *limit: Optional[int] = None*, *distinct=None*, *index: Optional[Union[str, list]] = None*, *drop_index_col=False*, *time_field: str = 'timestamp'*)

query data by the arguments

Parameters::

- **provider_index** –
- **data_schema** –
- **ids** –
- **entity_ids** –
- **entity_id** –
- **codes** –
- **code** –
- **level** –
- **provider** –
- **columns** –
- **col_label** – dict with key(column), value(label
- **return_type** – df, domain or dict. default is df
- **start_timestamp** –
- **end_timestamp** –
- **filters** –
- **session** –
- **order** –
- **limit** –
- **index** – index field name, str for single index, str list for multiple index
- **drop_index_col** – whether drop the col if it's in index, default False
- **time_field** –

Returns:: results basing on return_type.

classmethod record_data(*provider_index: int = 0*, *provider: Optional[str] = None*, *force_update=None*, *sleeping_time=None*, *exchanges=None*, *entity_id=None*, *entity_ids=None*, *code=None*, *codes=None*, *real_time=None*, *fix_duplicate_way=None*, *start_timestamp=None*, *end_timestamp=None*, *one_day_trading_minutes=None*, ***kwargs*)

record data by the arguments

Parameters::

- **entity_id** –
- **provider_index** –
- **provider** –
- **force_update** –
- **sleeping_time** –
- **exchanges** –
- **entity_ids** –
- **code** –
- **codes** –
- **real_time** –
- **fix_duplicate_way** –
- **start_timestamp** –
- **end_timestamp** –
- **one_day_trading_minutes** –
- **kwargs** –

Returns::

classmethod register_provider(*provider*)

register the provider to the schema defined by cls

Parameters:: **provider** –

classmethod register_recorder_cls(*provider*, *recorder_cls*)

register the recorder for the schema

Parameters::

- **provider** –
- **recorder_cls** –

timestamp

the meaning could be different for different case,most time it means 'happen time'

turnover

turnover_rate

volume

zvt.domain.quotes.stock.stock_4h_hfq_kdata

Classes

`Stock4hHfqKdata`(**kwargs)

Stock4hHfqKdata

class zvt.domain.quotes.stock.stock_4h_hfq_kdata.Stock4hHfqKdata(**kwargs)

Bases: **Base**, **StockKdataCommon**

__init__(**kwargs)

A simple constructor that allows initialization from kwargs.

Sets attributes on the constructed instance using the names and values in `kwargs`.

Only keys that are present as attributes of the instance's class are allowed. These could be, for example, any mapped columns or relationships.

change_pct

close

code

entity_id

entity id

classmethod get_providers() → List[str]

providers of the schema defined by cls

Returns:: providers

classmethod get_storages(*provider: Optional[str] = None*)

get the storages info

Parameters:: **provider** – provider
Returns:: storages

high

id

id

level

low

name

open

provider

classmethod query_data(*provider_index: int = 0*, *ids: Optional[List[str]] = None*, *entity_ids: Optional[List[str]] = None*, *entity_id: Optional[str] = None*, *codes: Optional[List[str]] = None*, *code: Optional[str] = None*, *level: Optional[Union[IntervalLevel, str]] = None*, *provider: Optional[str] = None*, *columns: Optional[List] = None*, *col_label: Optional[dict] = None*, *return_type: str = 'df'*, *start_timestamp: Optional[Union[Timestamp, str]] = None*, *end_timestamp: Optional[Union[Timestamp, str]] = None*, *filters: Optional[List] = None*, *session: Optional[Session] = None*, *order=None*, *limit: Optional[int] = None*, *distinct=None*, *index: Optional[Union[str, list]] = None*, *drop_index_col=False*, *time_field: str = 'timestamp'*)

query data by the arguments

Parameters::

- **provider_index** –
- **data_schema** –
- **ids** –
- **entity_ids** –
- **entity_id** –
- **codes** –
- **code** –
- **level** –
- **provider** –
- **columns** –
- **col_label** – dict with key(column), value(label
- **return_type** – df, domain or dict. default is df
- **start_timestamp** –
- **end_timestamp** –
- **filters** –
- **session** –
- **order** –
- **limit** –
- **index** – index field name, str for single index, str list for multiple index
- **drop_index_col** – whether drop the col if it's in index, default False
- **time_field** –

Returns:: results basing on return_type.

classmethod record_data(*provider_index: int = 0*, *provider: Optional[str] = None*, *force_update=None*, *sleeping_time=None*, *exchanges=None*, *entity_id=None*, *entity_ids=None*, *code=None*, *codes=None*, *real_time=None*, *fix_duplicate_way=None*, *start_timestamp=None*, *end_timestamp=None*, *one_day_trading_minutes=None*, ***kwargs*)

record data by the arguments

Parameters::

- **entity_id** –
- **provider_index** –
- **provider** –
- **force_update** –
- **sleeping_time** –
- **exchanges** –
- **entity_ids** –
- **code** –
- **codes** –
- **real_time** –
- **fix_duplicate_way** –
- **start_timestamp** –
- **end_timestamp** –
- **one_day_trading_minutes** –
- **kwargs** –

Returns::

classmethod register_provider(*provider*)

register the provider to the schema defined by cls

Parameters:: **provider** –

classmethod register_recorder_cls(*provider*, *recorder_cls*)

register the recorder for the schema

Parameters::

- **provider** –
- **recorder_cls** –

timestamp

the meaning could be different for different case,most time it means 'happen time'

turnover

turnover_rate

volume

zvt.domain.quotes.stock.stock_4h_kdata

Classes

`Stock4hKdata`(**kwargs)

Stock4hKdata

class zvt.domain.quotes.stock.stock_4h_kdata.Stock4hKdata(**kwargs*)

Bases: **Base**, **StockKdataCommon**

__init__(**kwargs*)

A simple constructor that allows initialization from kwargs.

Sets attributes on the constructed instance using the names and values in `kwargs`.

Only keys that are present as attributes of the instance's class are allowed. These could be, for example, any mapped columns or relationships.

change_pct

close

code

entity_id

entity id

classmethod get_providers() → List[str]

providers of the schema defined by cls

Returns:: providers

classmethod get_storages(*provider: Optional[str] = None*)

get the storages info

Parameters:: **provider** – provider
Returns:: storages

high

id

id

level

low

name

open

provider

classmethod query_data(*provider_index: int = 0*, *ids: Optional[List[str]] = None*, *entity_ids: Optional[List[str]] = None*, *entity_id: Optional[str] = None*, *codes: Optional[List[str]] = None*, *code: Optional[str] = None*, *level: Optional[Union[IntervalLevel, str]] = None*, *provider: Optional[str] = None*, *columns: Optional[List] = None*, *col_label: Optional[dict] = None*, *return_type: str = 'df'*, *start_timestamp: Optional[Union[Timestamp, str]] = None*, *end_timestamp: Optional[Union[Timestamp, str]] = None*, *filters: Optional[List] = None*, *session: Optional[Session] = None*, *order=None*, *limit: Optional[int] = None*, *distinct=None*, *index: Optional[Union[str, list]] = None*, *drop_index_col=False*, *time_field: str = 'timestamp'*)

query data by the arguments

Parameters::

- **provider_index** –
- **data_schema** –
- **ids** –
- **entity_ids** –
- **entity_id** –
- **codes** –
- **code** –
- **level** –
- **provider** –
- **columns** –
- **col_label** – dict with key(column), value(label
- **return_type** – df, domain or dict. default is df
- **start_timestamp** –
- **end_timestamp** –
- **filters** –
- **session** –
- **order** –
- **limit** –
- **index** – index field name, str for single index, str list for multiple index
- **drop_index_col** – whether drop the col if it's in index, default False
- **time_field** –

Returns:: results basing on return_type.

classmethod record_data(*provider_index: int = 0*, *provider: Optional[str] = None*, *force_update=None*, *sleeping_time=None*, *exchanges=None*, *entity_id=None*, *entity_ids=None*, *code=None*, *codes=None*, *real_time=None*, *fix_duplicate_way=None*, *start_timestamp=None*, *end_timestamp=None*, *one_day_trading_minutes=None*, ***kwargs*)

record data by the arguments

Parameters::

- **entity_id** –
- **provider_index** –
- **provider** –
- **force_update** –
- **sleeping_time** –
- **exchanges** –
- **entity_ids** –
- **code** –
- **codes** –
- **real_time** –
- **fix_duplicate_way** –
- **start_timestamp** –
- **end_timestamp** –
- **one_day_trading_minutes** –
- **kwargs** –

Returns::

classmethod register_provider(*provider*)

register the provider to the schema defined by cls

Parameters:: **provider** –

classmethod register_recorder_cls(*provider*, *recorder_cls*)

register the recorder for the schema

Parameters::

- **provider** –
- **recorder_cls** –

timestamp

the meaning could be different for different case,most time it means 'happen time'

turnover

turnover_rate

volume

zvt.domain.quotes.stock.stock_5m_hfq_kdata

Classes

`Stock5mHfqKdata`(**kwargs)

Stock5mHfqKdata

class zvt.domain.quotes.stock.stock_5m_hfq_kdata.Stock5mHfqKdata(**kwargs)

Bases: **Base**, **StockKdataCommon**

__init__(**kwargs)

A simple constructor that allows initialization from kwargs.

Sets attributes on the constructed instance using the names and values in `kwargs`.

Only keys that are present as attributes of the instance's class are allowed. These could be, for example, any mapped columns or relationships.

change_pct

close

code

entity_id

entity id

classmethod get_providers() → List[str]

providers of the schema defined by cls

Returns:: providers

classmethod get_storages(*provider: Optional[str] = None*)

get the storages info

Parameters:: **provider** – provider
Returns:: storages

high

id

id

level

low

name

open

provider

classmethod query_data(*provider_index: int = 0*, *ids: Optional[List[str]] = None*, *entity_ids: Optional[List[str]] = None*, *entity_id: Optional[str] = None*, *codes: Optional[List[str]] = None*, *code: Optional[str] = None*, *level: Optional[Union[IntervalLevel, str]] = None*, *provider: Optional[str] = None*, *columns: Optional[List] = None*, *col_label: Optional[dict] = None*, *return_type: str = 'df'*, *start_timestamp: Optional[Union[Timestamp, str]] = None*, *end_timestamp: Optional[Union[Timestamp, str]] = None*, *filters: Optional[List] = None*, *session: Optional[Session] = None*, *order=None*, *limit: Optional[int] = None*, *distinct=None*, *index: Optional[Union[str, list]] = None*, *drop_index_col=False*, *time_field: str = 'timestamp'*)

query data by the arguments

Parameters::

- **provider_index** –
- **data_schema** –
- **ids** –
- **entity_ids** –
- **entity_id** –
- **codes** –
- **code** –
- **level** –
- **provider** –
- **columns** –
- **col_label** – dict with key(column), value(label
- **return_type** – df, domain or dict. default is df
- **start_timestamp** –
- **end_timestamp** –
- **filters** –
- **session** –
- **order** –
- **limit** –
- **index** – index field name, str for single index, str list for multiple index
- **drop_index_col** – whether drop the col if it's in index, default False
- **time_field** –

Returns:: results basing on return_type.

classmethod record_data(*provider_index: int = 0*, *provider: Optional[str] = None*, *force_update=None*, *sleeping_time=None*, *exchanges=None*, *entity_id=None*, *entity_ids=None*, *code=None*, *codes=None*, *real_time=None*, *fix_duplicate_way=None*, *start_timestamp=None*, *end_timestamp=None*, *one_day_trading_minutes=None*, ***kwargs*)

record data by the arguments

Parameters::

- **entity_id** –
- **provider_index** –
- **provider** –
- **force_update** –
- **sleeping_time** –
- **exchanges** –
- **entity_ids** –
- **code** –
- **codes** –
- **real_time** –
- **fix_duplicate_way** –
- **start_timestamp** –
- **end_timestamp** –
- **one_day_trading_minutes** –
- **kwargs** –

Returns::

classmethod register_provider(*provider*)

register the provider to the schema defined by cls

Parameters:: **provider** –

classmethod register_recorder_cls(*provider*, *recorder_cls*)

register the recorder for the schema

Parameters::

- **provider** –
- **recorder_cls** –

timestamp

the meaning could be different for different case,most time it means 'happen time'

turnover

turnover_rate

volume

zvt.domain.quotes.stock.stock_5m_kdata

Classes

`Stock5mKdata`(**kwargs)

Stock5mKdata

class zvt.domain.quotes.stock.stock_5m_kdata.Stock5mKdata(***kwargs*)

Bases: **Base**, **StockKdataCommon**

__init__(***kwargs*)

A simple constructor that allows initialization from kwargs.

Sets attributes on the constructed instance using the names and values in `kwargs`.

Only keys that are present as attributes of the instance's class are allowed. These could be, for example, any mapped columns or relationships.

change_pct

close

code

entity_id

entity id

classmethod get_providers() → List[str]

providers of the schema defined by cls

Returns:: providers

classmethod get_storages(*provider: Optional[str] = None*)

get the storages info

Parameters:: **provider** – provider
Returns:: storages

high

id

id

level

low

name

open

provider

classmethod query_data(*provider_index: int = 0*, *ids: Optional[List[str]] = None*, *entity_ids: Optional[List[str]] = None*, *entity_id: Optional[str] = None*, *codes: Optional[List[str]] = None*, *code: Optional[str] = None*, *level: Optional[Union[IntervalLevel, str]] = None*, *provider: Optional[str] = None*, *columns: Optional[List] = None*, *col_label: Optional[dict] = None*, *return_type: str = 'df'*, *start_timestamp: Optional[Union[Timestamp, str]] = None*, *end_timestamp: Optional[Union[Timestamp, str]] = None*, *filters: Optional[List] = None*, *session: Optional[Session] = None*, *order=None*, *limit: Optional[int] = None*, *distinct=None*, *index: Optional[Union[str, list]] = None*, *drop_index_col=False*, *time_field: str = 'timestamp'*)

query data by the arguments

Parameters::

- **provider_index** –
- **data_schema** –
- **ids** –
- **entity_ids** –
- **entity_id** –
- **codes** –
- **code** –
- **level** –
- **provider** –
- **columns** –
- **col_label** – dict with key(column), value(label
- **return_type** – df, domain or dict. default is df
- **start_timestamp** –
- **end_timestamp** –
- **filters** –
- **session** –
- **order** –
- **limit** –
- **index** – index field name, str for single index, str list for multiple index
- **drop_index_col** – whether drop the col if it's in index, default False
- **time_field** –

Returns:: results basing on return_type.

classmethod record_data(*provider_index: int = 0*, *provider: Optional[str] = None*, *force_update=None*, *sleeping_time=None*, *exchanges=None*, *entity_id=None*, *entity_ids=None*, *code=None*, *codes=None*, *real_time=None*, *fix_duplicate_way=None*, *start_timestamp=None*, *end_timestamp=None*, *one_day_trading_minutes=None*, ***kwargs*)

record data by the arguments

Parameters::

- **entity_id** –
- **provider_index** –
- **provider** –
- **force_update** –
- **sleeping_time** –
- **exchanges** –
- **entity_ids** –
- **code** –
- **codes** –
- **real_time** –
- **fix_duplicate_way** –
- **start_timestamp** –
- **end_timestamp** –
- **one_day_trading_minutes** –
- **kwargs** –

Returns::

classmethod register_provider(*provider*)

register the provider to the schema defined by cls

Parameters:: **provider** –

classmethod register_recorder_cls(*provider*, *recorder_cls*)

register the recorder for the schema

Parameters::

- **provider** –
- **recorder_cls** –

timestamp

the meaning could be different for different case,most time it means 'happen time'

turnover

turnover_rate

volume

zvt.domain.quotes.stock.stock_quote

Classes

StockQuote(**kwargs)

StockQuote

class zvt.domain.quotes.stock.stock_quote.StockQuote(**kwargs)

Bases: **Base**, **Mixin**

__init__(**kwargs)

A simple constructor that allows initialization from kwargs.

Sets attributes on the constructed instance using the names and values in `kwargs`.

Only keys that are present as attributes of the instance's class are allowed. These could be, for example, any mapped columns or relationships.

time

UNIX时间戳

price

最新价

is_limit_up

是否涨停

limit_up_amount

封涨停金额

is_limit_down

是否跌停

limit_down_amount

封跌停金额

ask_amount

5挡卖单金额

bid_amount
5挡买单金额

float_cap
流通市值

total_cap
总市值

entity_id
entity id

classmethod get_providers() → List[str]
providers of the schema defined by cls

Returns:: providers

classmethod get_storages(*provider: Optional[str] = None*)
get the storages info

Parameters:: **provider** – provider
Returns:: storages

id
id

classmethod query_data(*provider_index: int = 0*, *ids: Optional[List[str]] = None*, *entity_ids: Optional[List[str]] = None*, *entity_id: Optional[str] = None*, *codes: Optional[List[str]] = None*, *code: Optional[str] = None*, *level: Optional[Union[IntervalLevel, str]] = None*, *provider: Optional[str] = None*, *columns: Optional[List] = None*, *col_label: Optional[dict] = None*, *return_type: str = 'df'*, *start_timestamp: Optional[Union[Timestamp, str]] = None*, *end_timestamp: Optional[Union[Timestamp, str]] = None*, *filters: Optional[List] = None*, *session: Optional[Session] = None*, *order=None*, *limit: Optional[int] =*

None, *distinct=None*, *index: Optional[Union[str, list]] = None*, *drop_index_col=False*, *time_field: str = 'timestamp'*)

query data by the arguments

Parameters::

- **provider_index** –
- **data_schema** –
- **ids** –
- **entity_ids** –
- **entity_id** –
- **codes** –
- **code** –
- **level** –
- **provider** –
- **columns** –
- **col_label** – dict with key(column), value(label
- **return_type** – df, domain or dict. default is df
- **start_timestamp** –
- **end_timestamp** –
- **filters** –
- **session** –
- **order** –
- **limit** –
- **index** – index field name, str for single index, str list for multiple index
- **drop_index_col** – whether drop the col if it's in index, default False
- **time_field** –

Returns:: results basing on return_type.

classmethod record_data(*provider_index: int = 0*, *provider: Optional[str] = None*, *force_update=None*, *sleeping_time=None*, *exchanges=None*, *entity_id=None*, *entity_ids=None*, *code=None*, *codes=None*, *real_time=None*, *fix_duplicate_way=None*, *start_timestamp=None*, *end_timestamp=None*, *one_day_trading_minutes=None*, ***kwargs*)

record data by the arguments

Parameters::

- **entity_id** –
- **provider_index** –
- **provider** –
- **force_update** –
- **sleeping_time** –
- **exchanges** –
- **entity_ids** –
- **code** –
- **codes** –
- **real_time** –
- **fix_duplicate_way** –
- **start_timestamp** –
- **end_timestamp** –
- **one_day_trading_minutes** –
- **kwargs** –

Returns::

classmethod register_provider(*provider*)

register the provider to the schema defined by cls

Parameters:: **provider** –

classmethod register_recorder_cls(*provider*, *recorder_cls*)

register the recorder for the schema

Parameters::

- **provider** –
- **recorder_cls** –

timestamp

the meaning could be different for different case,most time it means 'happen time'

zvt.domain.quotes.stockhk

Modules

zvt.domain.quotes.stockhk.stockhk_1d_hfq_kdata

---

zvt.domain.quotes.stockhk.stockhk_1d_kdata

---

zvt.domain.quotes.stockhk.stockhk_1d_hfq_kdata

Classes

`Stockhk1dHfqKdata`(**kwargs)

Stockhk1dHfqKdata

class zvt.domain.quotes.stockhk.stockhk_1d_hfq_kdata.Stockhk1dHfqKdata(**kwargs)

Bases: **Base**, **StockhkKdataCommon**

__init__(**kwargs)

A simple constructor that allows initialization from kwargs.

Sets attributes on the constructed instance using the names and values in `kwargs`.

Only keys that are present as attributes of the instance's class are allowed. These could be, for example, any mapped columns or relationships.

change_pct

close

code

entity_id

entity id

classmethod get_providers() → List[str]

providers of the schema defined by cls

Returns:: providers

classmethod get_storages(*provider: Optional[str] = None*)

get the storages info

Parameters:: **provider** – provider
Returns:: storages

high

id

id

level

low

name

open

provider

classmethod query_data(*provider_index: int = 0*, *ids: Optional[List[str]] = None*, *entity_ids: Optional[List[str]] = None*, *entity_id: Optional[str] = None*, *codes: Optional[List[str]] = None*, *code: Optional[str] = None*, *level: Optional[Union[IntervalLevel, str]] = None*, *provider: Optional[str] = None*, *columns: Optional[List] = None*, *col_label: Optional[dict] = None*, *return_type: str = 'df'*, *start_timestamp: Optional[Union[Timestamp, str]] = None*, *end_timestamp: Optional[Union[Timestamp, str]] = None*, *filters: Optional[List] = None*, *session: Optional[Session] = None*, *order=None*, *limit: Optional[int] = None*, *distinct=None*, *index: Optional[Union[str, list]] = None*, *drop_index_col=False*, *time_field: str = 'timestamp'*)

query data by the arguments

Parameters::

- **provider_index** –
- **data_schema** –
- **ids** –
- **entity_ids** –
- **entity_id** –
- **codes** –
- **code** –
- **level** –
- **provider** –
- **columns** –
- **col_label** – dict with key(column), value(label
- **return_type** – df, domain or dict. default is df
- **start_timestamp** –
- **end_timestamp** –
- **filters** –
- **session** –
- **order** –
- **limit** –
- **index** – index field name, str for single index, str list for multiple index
- **drop_index_col** – whether drop the col if it's in index, default False
- **time_field** –

Returns:: results basing on return_type.

classmethod record_data(*provider_index: int = 0*, *provider: Optional[str] = None*, *force_update=None*, *sleeping_time=None*, *exchanges=None*, *entity_id=None*, *entity_ids=None*, *code=None*, *codes=None*, *real_time=None*, *fix_duplicate_way=None*, *start_timestamp=None*, *end_timestamp=None*, *one_day_trading_minutes=None*, ***kwargs*)

record data by the arguments

Parameters::

- **entity_id** –
- **provider_index** –
- **provider** –
- **force_update** –
- **sleeping_time** –
- **exchanges** –
- **entity_ids** –
- **code** –
- **codes** –
- **real_time** –
- **fix_duplicate_way** –
- **start_timestamp** –
- **end_timestamp** –
- **one_day_trading_minutes** –
- **kwargs** –

Returns::

classmethod register_provider(*provider*)

register the provider to the schema defined by cls

Parameters:: **provider** –

classmethod register_recorder_cls(*provider*, *recorder_cls*)

register the recorder for the schema

Parameters::

- **provider** –
- **recorder_cls** –

timestamp

the meaning could be different for different case,most time it means 'happen time'

turnover

turnover_rate

volume

zvt.domain.quotes.stockhk.stockhk_1d_kdata

Classes

`Stockhk1dKdata`(**kwargs)

Stockhk1dKdata

class zvt.domain.quotes.stockhk.stockhk_1d_kdata.Stockhk1dKdata(***kwargs*)

Bases: **Base**, **StockhkKdataCommon**

__init__(***kwargs*)

A simple constructor that allows initialization from kwargs.

Sets attributes on the constructed instance using the names and values in `kwargs`.

Only keys that are present as attributes of the instance's class are allowed. These could be, for example, any mapped columns or relationships.

change_pct

close

code

entity_id

entity id

classmethod get_providers() → List[str]

providers of the schema defined by cls

Returns:: providers

classmethod get_storages(*provider: Optional[str] = None*)

get the storages info

Parameters:: **provider** – provider
Returns:: storages

high

id

id

level

low

name

open

provider

classmethod query_data(*provider_index: int = 0*, *ids: Optional[List[str]] = None*, *entity_ids: Optional[List[str]] = None*, *entity_id: Optional[str] = None*, *codes: Optional[List[str]] = None*, *code: Optional[str] = None*, *level: Optional[Union[IntervalLevel, str]] = None*, *provider: Optional[str] = None*, *columns: Optional[List] = None*, *col_label: Optional[dict] = None*, *return_type: str = 'df'*, *start_timestamp: Optional[Union[Timestamp, str]] = None*, *end_timestamp: Optional[Union[Timestamp, str]] = None*, *filters: Optional[List] = None*, *session: Optional[Session] = None*, *order=None*, *limit: Optional[int] = None*, *distinct=None*, *index: Optional[Union[str, list]] = None*, *drop_index_col=False*, *time_field: str = 'timestamp'*)

query data by the arguments

Parameters::

- **provider_index** –
- **data_schema** –
- **ids** –
- **entity_ids** –
- **entity_id** –
- **codes** –
- **code** –
- **level** –
- **provider** –
- **columns** –
- **col_label** – dict with key(column), value(label
- **return_type** – df, domain or dict. default is df
- **start_timestamp** –
- **end_timestamp** –
- **filters** –
- **session** –
- **order** –
- **limit** –
- **index** – index field name, str for single index, str list for multiple index
- **drop_index_col** – whether drop the col if it's in index, default False
- **time_field** –

Returns:: results basing on return_type.

classmethod record_data(*provider_index: int = 0*, *provider: Optional[str] = None*, *force_update=None*, *sleeping_time=None*, *exchanges=None*, *entity_id=None*, *entity_ids=None*, *code=None*, *codes=None*, *real_time=None*, *fix_duplicate_way=None*, *start_timestamp=None*, *end_timestamp=None*, *one_day_trading_minutes=None*, ***kwargs*)

record data by the arguments

Parameters::

- **entity_id** –
- **provider_index** –
- **provider** –
- **force_update** –
- **sleeping_time** –
- **exchanges** –
- **entity_ids** –
- **code** –
- **codes** –
- **real_time** –
- **fix_duplicate_way** –
- **start_timestamp** –
- **end_timestamp** –
- **one_day_trading_minutes** –
- **kwargs** –

Returns::

classmethod register_provider(*provider*)

register the provider to the schema defined by cls

Parameters:: **provider** –

classmethod register_recorder_cls(*provider*, *recorder_cls*)

register the recorder for the schema

Parameters::

- **provider** –
- **recorder_cls** –

timestamp

the meaning could be different for different case,most time it means 'happen time'

turnover

turnover_rate

volume

zvt.domain.quotes.stockus

Modules

zvt.domain.quotes.stockus.stockus_1d_hfq_kdata

---

zvt.domain.quotes.stockus.stockus_1d_kdata

---

zvt.domain.quotes.stockus.stockus_1d_hfq_kdata

Classes

`Stockus1dHfqKdata`(**kwargs)

Stockus1dHfqKdata

class zvt.domain.quotes.stockus.stockus_1d_hfq_kdata.Stockus1dHfqKdata(***kwargs*)

Bases: **Base**, **StockusKdataCommon**

__init__(***kwargs*)

A simple constructor that allows initialization from kwargs.

Sets attributes on the constructed instance using the names and values in `kwargs`.

Only keys that are present as attributes of the instance's class are allowed. These could be, for example, any mapped columns or relationships.

change_pct

close

code

entity_id

entity id

classmethod get_providers() → List[str]

providers of the schema defined by cls

Returns:: providers

classmethod get_storages(*provider: Optional[str] = None*)

get the storages info

Parameters:: **provider** – provider
Returns:: storages

high

id

id

level

low

name

open

provider

classmethod query_data(*provider_index: int = 0*, *ids: Optional[List[str]] = None*, *entity_ids: Optional[List[str]] = None*, *entity_id: Optional[str] = None*, *codes: Optional[List[str]] = None*, *code: Optional[str] = None*, *level: Optional[Union[IntervalLevel, str]] = None*, *provider: Optional[str] = None*, *columns: Optional[List] = None*, *col_label: Optional[dict] = None*, *return_type: str = 'df'*, *start_timestamp: Optional[Union[Timestamp, str]] = None*, *end_timestamp: Optional[Union[Timestamp, str]] = None*, *filters: Optional[List] = None*, *session: Optional[Session] = None*, *order=None*, *limit: Optional[int] = None*, *distinct=None*, *index: Optional[Union[str, list]] = None*, *drop_index_col=False*, *time_field: str = 'timestamp'*)

query data by the arguments

Parameters::

- **provider_index** –
- **data_schema** –
- **ids** –
- **entity_ids** –
- **entity_id** –
- **codes** –
- **code** –
- **level** –
- **provider** –
- **columns** –
- **col_label** – dict with key(column), value(label
- **return_type** – df, domain or dict. default is df
- **start_timestamp** –
- **end_timestamp** –
- **filters** –
- **session** –
- **order** –
- **limit** –
- **index** – index field name, str for single index, str list for multiple index
- **drop_index_col** – whether drop the col if it's in index, default False
- **time_field** –

Returns:: results basing on return_type.

classmethod record_data(*provider_index: int = 0*, *provider: Optional[str] = None*, *force_update=None*, *sleeping_time=None*, *exchanges=None*, *entity_id=None*, *entity_ids=None*, *code=None*, *codes=None*, *real_time=None*, *fix_duplicate_way=None*, *start_timestamp=None*, *end_timestamp=None*, *one_day_trading_minutes=None*, ***kwargs*)

record data by the arguments

Parameters::

- **entity_id** –
- **provider_index** –
- **provider** –
- **force_update** –
- **sleeping_time** –
- **exchanges** –
- **entity_ids** –
- **code** –
- **codes** –
- **real_time** –
- **fix_duplicate_way** –
- **start_timestamp** –
- **end_timestamp** –
- **one_day_trading_minutes** –
- **kwargs** –

Returns::

classmethod register_provider(*provider*)

register the provider to the schema defined by cls

Parameters:: **provider** –

classmethod register_recorder_cls(*provider*, *recorder_cls*)

register the recorder for the schema

Parameters::

- **provider** –
- **recorder_cls** –

timestamp

the meaning could be different for different case,most time it means 'happen time'

turnover

turnover_rate

volume

zvt.domain.quotes.stockus.stockus_1d_kdata

Classes

`Stockus1dKdata`(**kwargs)

Stockus1dKdata

class zvt.domain.quotes.stockus.stockus_1d_kdata.Stockus1dKdata(***kwargs*)

Bases: **Base**, **StockusKdataCommon**

__init__(***kwargs*)

A simple constructor that allows initialization from kwargs.

Sets attributes on the constructed instance using the names and values in `kwargs`.

Only keys that are present as attributes of the instance's class are allowed. These could be, for example, any mapped columns or relationships.

change_pct

close

code

entity_id

entity id

classmethod get_providers() → List[str]

providers of the schema defined by cls

Returns:: providers

classmethod get_storages(*provider: Optional[str] = None*)

get the storages info

Parameters:: **provider** – provider
Returns:: storages

high

id

id

level

low

name

open

provider

classmethod query_data(*provider_index: int = 0*, *ids: Optional[List[str]] = None*, *entity_ids: Optional[List[str]] = None*, *entity_id: Optional[str] = None*, *codes: Optional[List[str]] = None*, *code: Optional[str] = None*, *level: Optional[Union[IntervalLevel, str]] = None*, *provider: Optional[str] = None*, *columns: Optional[List] = None*, *col_label: Optional[dict] = None*, *return_type: str = 'df'*, *start_timestamp: Optional[Union[Timestamp, str]] = None*, *end_timestamp: Optional[Union[Timestamp, str]] = None*, *filters: Optional[List] = None*, *session: Optional[Session] = None*, *order=None*, *limit: Optional[int] = None*, *distinct=None*, *index: Optional[Union[str, list]] = None*, *drop_index_col=False*, *time_field: str = 'timestamp'*)

query data by the arguments

Parameters::

- **provider_index** –
- **data_schema** –
- **ids** –
- **entity_ids** –
- **entity_id** –
- **codes** –
- **code** –
- **level** –
- **provider** –
- **columns** –
- **col_label** – dict with key(column), value(label
- **return_type** – df, domain or dict. default is df
- **start_timestamp** –
- **end_timestamp** –
- **filters** –
- **session** –
- **order** –
- **limit** –
- **index** – index field name, str for single index, str list for multiple index
- **drop_index_col** – whether drop the col if it's in index, default False
- **time_field** –

Returns:: results basing on return_type.

classmethod record_data(*provider_index: int = 0*, *provider: Optional[str] = None*, *force_update=None*, *sleeping_time=None*, *exchanges=None*, *entity_id=None*, *entity_ids=None*, *code=None*, *codes=None*, *real_time=None*, *fix_duplicate_way=None*, *start_timestamp=None*, *end_timestamp=None*, *one_day_trading_minutes=None*, ***kwargs*)

record data by the arguments

Parameters::

- **entity_id** –
- **provider_index** –
- **provider** –
- **force_update** –
- **sleeping_time** –
- **exchanges** –
- **entity_ids** –
- **code** –
- **codes** –
- **real_time** –
- **fix_duplicate_way** –
- **start_timestamp** –
- **end_timestamp** –
- **one_day_trading_minutes** –
- **kwargs** –

Returns::

classmethod register_provider(*provider*)

register the provider to the schema defined by cls

Parameters:: **provider** –

classmethod register_recorder_cls(*provider*, *recorder_cls*)

register the recorder for the schema

Parameters::

- **provider** –
- **recorder_cls** –

timestamp

the meaning could be different for different case,most time it means 'happen time'

turnover

turnover_rate

volume

zvt.domain.quotes.trade_day

Classes

`StockTradeDay`(**kwargs)

StockTradeDay

class zvt.domain.quotes.trade_day.StockTradeDay(***kwargs*)

Bases: **Base**, **Mixin**

__init__(***kwargs*)

A simple constructor that allows initialization from kwargs.

Sets attributes on the constructed instance using the names and values in `kwargs`.

Only keys that are present as attributes of the instance's class are allowed. These could be, for example, any mapped columns or relationships.

entity_id

entity id

classmethod get_providers() → List[str]

providers of the schema defined by cls

Returns:: providers

classmethod get_storages(*provider: Optional[str] = None*)

get the storages info

Parameters:: **provider** – provider
Returns:: storages

id

id

classmethod query_data(*provider_index: int = 0*, *ids: Optional[List[str]] = None*, *entity_ids: Optional[List[str]] = None*, *entity_id: Optional[str] = None*, *codes: Optional[List[str]] = None*, *code: Optional[str] = None*, *level: Optional[Union[IntervalLevel, str]] = None*, *provider:*

Optional[str] = None, *columns: Optional[List] = None*, *col_label: Optional[dict] = None*, *return_type: str = 'df'*, *start_timestamp: Optional[Union[Timestamp, str]] = None*, *end_timestamp: Optional[Union[Timestamp, str]] = None*, *filters: Optional[List] = None*, *session: Optional[Session] = None*, *order=None*, *limit: Optional[int] = None*, *distinct=None*, *index: Optional[Union[str, list]] = None*, *drop_index_col=False*, *time_field: str = 'timestamp'*)

query data by the arguments

Parameters::

- **provider_index** –
- **data_schema** –
- **ids** –
- **entity_ids** –
- **entity_id** –
- **codes** –
- **code** –
- **level** –
- **provider** –
- **columns** –
- **col_label** – dict with key(column), value(label
- **return_type** – df, domain or dict. default is df
- **start_timestamp** –
- **end_timestamp** –
- **filters** –
- **session** –
- **order** –
- **limit** –
- **index** – index field name, str for single index, str list for multiple index
- **drop_index_col** – whether drop the col if it's in index, default False
- **time_field** –

Returns:: results basing on return_type.

classmethod record_data(*provider_index: int = 0*, *provider: Optional[str] = None*, *force_update=None*, *sleeping_time=None*, *exchanges=None*, *entity_id=None*, *entity_ids=None*, *code=None*, *codes=None*,

real_time=None, *fix_duplicate_way=None*, *start_timestamp=None*, *end_timestamp=None*, *one_day_trading_minutes=None*, ***kwargs*)

record data by the arguments

Parameters::

- **entity_id** –
- **provider_index** –
- **provider** –
- **force_update** –
- **sleeping_time** –
- **exchanges** –
- **entity_ids** –
- **code** –
- **codes** –
- **real_time** –
- **fix_duplicate_way** –
- **start_timestamp** –
- **end_timestamp** –
- **one_day_trading_minutes** –
- **kwargs** –

Returns::

classmethod register_provider(*provider*)

register the provider to the schema defined by cls

Parameters:: **provider** –

classmethod register_recorder_cls(*provider*, *recorder_cls*)

register the recorder for the schema

Parameters::

- **provider** –
- **recorder_cls** –

timestamp

the meaning could be different for different case,most time it means 'happen time'

zvt.factors

Modules

zvt.factors.algorithm

---

zvt.factors.factor_models

---

zvt.factors.factor_service

---

zvt.factors.fundamental

---

zvt.factors.ma

---

zvt.factors.macd

---

zvt.factors.shape

---

zvt.factors.target_selector

---

zvt.factors.technical_factor

---

zvt.factors.top_stocks

---

zvt.factors.transformers

zvt.factors.z

zvt.factors.algorithm

Functions

`combine`(range_a, range_b)		
`distance`(range_a, range_b[, use_max])		
`ema`(s[, window])		
`intersect`(range_a, range_b)	range_a and range_b with format (start,end) in y axis	
`intersect_ranges`(range_list)		
`live_or_dead`(x)		
`ma`(s[, window])	**param s::**	
`macd`(s[, slow, fast, n, return_type, ...])		
`point_in_range`(point, range)	**param point::**	one point

Classes

`IntersectTransformer`([kdata_overlap])

`MaAndVolumeTransformer`([windows, ...])

`MaTransformer`([windows, cal_change_pct])

`MacdTransformer`([slow, fast, n, normal, ...])

`QuantileScorer`([score_levels])

`RankScorer`([ascending])

---

zvt.factors.algorithm.combine

zvt.factors.algorithm.combine(*range_a*, *range_b*)

zvt.factors.algorithm.distance

zvt.factors.algorithm.distance(*range_a*, *range_b*, *use_max=False*)

zvt.factors.algorithm.ema

zvt.factors.algorithm.ema(*s: Series*, *window: int = 12*) → Series

zvt.factors.algorithm.intersect

zvt.factors.algorithm.intersect(*range_a*, *range_b*)

range_a and range_b with format (start,end) in y axis

Parameters::

- **range_a** –
- **range_b** –

Returns::

zvt.factors.algorithm.intersect_ranges

zvt.factors.algorithm.intersect_ranges(*range_list*)

zvt.factors.algorithm.live_or_dead

zvt.factors.algorithm.live_or_dead(*x*)

zvt.factors.algorithm.ma

zvt.factors.algorithm.ma(*s: Series*, *window: int = 5*) → Series

Parameters::

- **s** –
- **window** –

Returns::

zvt.factors.algorithm.macd

zvt.factors.algorithm.macd(*s: Series*, *slow: int = 26*, *fast: int = 12*, *n: int = 9*, *return_type: str = 'df'*, *normal: bool = False*, *count_live_dead: bool = False*)

zvt.factors.algorithm.point_in_range

zvt.factors.algorithm.point_in_range(*point: float*, *range: tuple*)

Parameters::

- **point** – one point
- **range** – (start,end)

Returns::

IntersectTransformer

class zvt.factors.algorithm.IntersectTransformer(*kdata_overlap=0*)

Bases: **Transformer**

__init__(*kdata_overlap=0*) → None

transform(*input_df*) → DataFrame

Parameters:: **input_df** –

Returns::

transform_one(*entity_id: str*, *df: DataFrame*) → DataFrame

df format:

```
                col1    col2    col3    ...
timestamp
                1.2     0.5     0.3     ...
                1.0     0.7     0.2     ...
```

the return result would change the columns and keep the format

Parameters::

- **entity_id** –
- **df** –

Returns::

MaAndVolumeTransformer

class zvt.factors.algorithm.MaAndVolumeTransformer(*windows=None*, *vol_windows=None*, *kdata_overlap=0*)

Bases: **Transformer**

__init__(*windows=None*, *vol_windows=None*, *kdata_overlap=0*) → None

transform(*input_df*) → DataFrame

input_df format:

```
                          col1    col2    col3    ...
entity_id    timestamp
                          1.2     0.5     0.3     ...
                          1.0     0.7     0.2     ...
```

the return result would change the columns and keep the format

Parameters:: **input_df** –

Returns::

transform_one(*entity_id: str*, *df: DataFrame*) → DataFrame

df format:

```
             col1    col2    col3    ...
timestamp
             1.2     0.5     0.3     ...
             1.0     0.7     0.2     ...
```

the return result would change the columns and keep the format

Parameters::

- **entity_id** –
- **df** –

Returns::

MaTransformer

class zvt.factors.algorithm.MaTransformer(*windows=None*, *cal_change_pct=False*)

Bases: **Transformer**

__init__(*windows=None*, *cal_change_pct=False*) → None

transform(*input_df: DataFrame*) → DataFrame

input_df format:

```
                                col1    col2    col3    ...
entity_id    timestamp
                                1.2     0.5     0.3     ...
                                1.0     0.7     0.2     ...
```

the return result would change the columns and keep the format

Parameters:: **input_df** –

Returns::

transform_one(*entity_id*, *df: DataFrame*) → DataFrame

transform_one would not take effects if transform was implemented. Just show how to implement it here, most of time you should overwrite transform directly for performance.

Parameters::

- **entity_id** –
- **df** –

Returns::

MacdTransformer

class zvt.factors.algorithm.MacdTransformer(*slow=26*, *fast=12*, *n=9*, *normal=False*, *count_live_dead=False*)

Bases: **Transformer**

__init__(*slow=26*, *fast=12*, *n=9*, *normal=False*, *count_live_dead=False*) → None

transform(*input_df*) → DataFrame

input_df format:

```
                          col1    col2    col3    ...
entity_id    timestamp
                          1.2     0.5     0.3     ...
                          1.0     0.7     0.2     ...
```

the return result would change the columns and keep the format

Parameters:: **input_df** –

Returns::

transform_one(*entity_id*, *df: DataFrame*) → DataFrame

df format:

```
             col1    col2    col3    ...
timestamp
             1.2     0.5     0.3     ...
             1.0     0.7     0.2     ...
```

the return result would change the columns and keep the format

Parameters::

- **entity_id** –
- **df** –

Returns::

QuantileScorer

class zvt.factors.algorithm.QuantileScorer(*score_levels=[0, 0.1, 0.3, 0.5, 0.7, 0.9, 1.0]*)

Bases: `Scorer`

__init__(*score_levels=[0, 0.1, 0.3, 0.5, 0.7, 0.9, 1.0]*) → None

score(*input_df*)

Parameters:: **input_df** – current input df

Returns:: df with normal score

RankScorer

class zvt.factors.algorithm.RankScorer(*ascending=True*)

Bases: `Scorer`

__init__(*ascending=True*) → None

score(*input_df*) → DataFrame

Parameters:: **input_df** – current input df

Returns:: df with normal score

zvt.factors.factor_models

Classes

`FactorRequestModel`(*, factor_name, entity_ids)

`FactorResultModel`(*, entity_ids, tag_reason)

`KdataModel`(*, entity_id, code, name[, level])

`KdataRequestModel`(*, entity_ids[, ...])

`TradingSignalModel`(*, entity_id, ...[, ...])

FactorRequestModel

class zvt.factors.factor_models.FactorRequestModel(*, *factor_name: str*, *entity_ids: Optional[List[str]]*, *data_provider: str = 'em'*, *start_timestamp: datetime = Timestamp('2023-08-29 00:00:00')*, *level: IntervalLevel = IntervalLevel.LEVEL_1DAY*)

Bases: **`BaseModel`**

__init__(**data: Any*) → None

Create a new model by parsing and validating input data from keyword arguments.

Raises [*ValidationError*][pydantic_core.ValidationError] if the input data cannot be validated to form a valid model.

self is explicitly positional-only to allow *self* as a field name.

copy(*, *include: AbstractSetIntStr | MappingIntStrAny | None = None*, *exclude: AbstractSetIntStr | MappingIntStrAny | None = None*, *update: Dict[str, Any] | None = None*, *deep: bool = False*) → Model

Returns a copy of the model.

!!! warning "Deprecated"

This method is now deprecated; use *model_copy* instead.

If you need *include* or *exclude*, use:

```
`py data = self.model_dump(include=include, exclude=exclude, round_trip=True) data = {**data, **(update or {})} copied = self.model_validate(data) `
```

Args:

include: Optional set or mapping specifying which fields to include in the copied model. exclude: Optional set or mapping

specifying which fields to exclude in the copied model. update: Optional dictionary of field-value pairs to override field values in the copied model. deep: If True, the values of fields that are Pydantic models will be deep-copied.

Returns:

A copy of the model with included, excluded and updated fields as specified.

model_computed_fields*: ClassVar[dict[str, ComputedFieldInfo]] = {}*

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

model_config*: ClassVar[ConfigDict] = {}*

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

classmethod model_construct(*_fields_set: set[str] | None = None*, ***values: Any*) → Model

Creates a new instance of the *Model* class with validated data.

Creates a new model setting *__dict__* and *__pydantic_fields_set__* from trusted or pre-validated data. Default values are respected, but no other validation is performed. Behaves as if *Config.extra = 'allow'* was set since it adds all passed values

Args:

_fields_set: The set of field names accepted for the Model instance. values: Trusted or pre-validated data dictionary.

Returns:

A new instance of the *Model* class with validated data.

model_copy(*, *update: dict[str, Any] | None = None*, *deep: bool = False*) → Model

Usage docs: https://docs.pydantic.dev/2.6/concepts/serialization/#model_copy

Returns a copy of the model.

Args:

update: Values to change/add in the new model. Note: the data is not validated
before creating the new model. You should trust this data.

deep: Set to *True* to make a deep copy of the model.

Returns:

New model instance.

model_dump(*, *mode: Literal['json', 'python'] | str = 'python'*, *include: IncEx = None*, *exclude: IncEx = None*, *by_alias: bool = False*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*, *round_trip: bool = False*, *warnings: bool = True*) → dict[str, Any]

Usage docs: https://docs.pydantic.dev/2.6/concepts/serialization/#modelmodel_dump

Generate a dictionary representation of the model, optionally specifying which fields to include or exclude.

Args:

mode: The mode in which *to_python* should run.
If mode is 'json', the output will only contain JSON serializable types. If mode is 'python', the output may contain non-JSON-serializable Python objects.

include: A list of fields to include in the output. exclude: A list of fields to exclude from the output. by_alias: Whether to use the field's alias in the dictionary key if defined. exclude_unset: Whether to exclude fields that have not been explicitly set. exclude_defaults: Whether to exclude fields that are set to their default value. exclude_none: Whether to exclude fields that have a value of *None*. round_trip: If True, dumped values should be

valid as input for non-idempotent types such as Json[T]. warnings: Whether to log warnings when invalid fields are encountered.

Returns:

A dictionary representation of the model.

model_dump_json(*, *indent: int | None = None*, *include: IncEx = None*, *exclude: IncEx = None*, *by_alias: bool = False*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*, *round_trip: bool = False*, *warnings: bool = True*) → str

Usage docs: https://docs.pydantic.dev/2.6/concepts/serialization/#modelmodel_dump_json

Generates a JSON representation of the model using Pydantic's *to_json* method.

Args:

indent: Indentation to use in the JSON output. If None is passed, the output will be compact. include: Field(s) to include in the JSON output. exclude: Field(s) to exclude from the JSON output. by_alias: Whether to serialize using field aliases. exclude_unset: Whether to exclude fields that have not been explicitly set. exclude_defaults: Whether to exclude fields that are set to their default value. exclude_none: Whether to exclude fields that have a value of *None*. round_trip: If True, dumped values should be valid as input for non-idempotent types such as Json[T]. warnings: Whether to log warnings when invalid fields are encountered.

Returns:

A JSON string representation of the model.

property model_extra*: dict[str, Any] | None*

Get extra fields set during validation.

Returns:

A dictionary of extra fields, or *None* if *config.extra* is not set to *"allow"*.

model_fields*: ClassVar[dict[str, FieldInfo]] = {'data_provider': FieldInfo(annotation=str, required=False, default='em'), 'entity_ids': FieldInfo(annotation=Union[List[str], NoneType], required=True), 'factor_name': FieldInfo(annotation=str, required=True), 'level': FieldInfo(annotation=IntervalLevel, required=False, default= <IntervalLevel.LEVEL_1DAY: '1d'>), 'start_timestamp': FieldInfo(annotation=datetime, required=False, default=Timestamp('2023-08-29 00:00:00'))}*

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

property model_fields_set*: set[str]*

Returns the set of fields that have been explicitly set on this model instance.

Returns:

A set of strings representing the fields that have been set, i.e. that were not filled from defaults.

classmethod model_json_schema(*by_alias: bool = True*, *ref_template: str = '#/$defs/{model}'*, *schema_generator: type[pydantic.json_schema.GenerateJsonSchema] = <class 'pydantic.json_schema.GenerateJsonSchema'>*, *mode: ~typing.Literal['validation', 'serialization'] = 'validation'*) → dict[str, Any]

Generates a JSON schema for a model class.

Args:

by_alias: Whether to use attribute aliases or not. ref_template: The reference template. schema_generator: To override the logic used to generate the JSON schema, as a subclass of

GenerateJsonSchema with your desired modifications

mode: The mode in which to generate the schema.

Returns:

The JSON schema for the given model class.

classmethod model_parametrized_name(*params: tuple[type[Any], ...]*) → str

Compute the class name for parametrizations of generic classes.

This method can be overridden to achieve a custom naming scheme for generic BaseModels.

Args:

params: Tuple of types of the class. Given a generic class *Model* with 2 type variables and a concrete model *Model[str, int]*, the value *(str, int)* would be passed to *params*.

Returns:

String representing the new class where *params* are passed to *cls* as type variables.

Raises:

TypeError: Raised when trying to generate concrete names for non-generic models.

model_post_init(*_BaseModel__context: Any*) → None

Override this method to perform additional initialization after *__init__* and *model_construct*. This is useful if you want to do some validation that requires the entire model to be initialized.

classmethod model_rebuild(*, *force: bool = False*, *raise_errors: bool = True*, *_parent_namespace_depth: int = 2*, *_types_namespace: Optional[dict[str, Any]] = None*) → bool | None

Try to rebuild the pydantic-core schema for the model.

This may be necessary when one of the annotations is a ForwardRef which could not be resolved during the initial attempt to build the schema, and automatic rebuilding fails.

Args:

force: Whether to force the rebuilding of the model schema, defaults to *False*. raise_errors: Whether to raise errors, defaults to *True*. _parent_namespace_depth: The depth level of the parent namespace, defaults to 2. _types_namespace: The types namespace, defaults to *None*.

Returns:

Returns *None* if the schema is already "complete" and rebuilding was not required. If rebuilding _was_ required, returns *True* if rebuilding was successful, otherwise *False*.

classmethod model_validate(*obj: Any*, *, *strict: bool | None = None*, *from_attributes: bool | None = None*, *context: dict[str, Any] | None = None*) → Model

Validate a pydantic model instance.

Args:

obj: The object to validate. strict: Whether to enforce types strictly. from_attributes: Whether to extract data from object attributes. context: Additional context to pass to the validator.

Raises:

ValidationError: If the object could not be validated.

Returns:

The validated model instance.

classmethod model_validate_json(*json_data: str | bytes | bytearray*, *, *strict: bool | None = None*, *context: dict[str, Any] | None = None*) → Model

Usage docs: https://docs.pydantic.dev/2.6/concepts/json/#json-parsing

Validate the given JSON data against the Pydantic model.

Args:
json_data: The JSON data to validate. strict: Whether to enforce types strictly. context: Extra variables to pass to the validator.

Returns:
The validated Pydantic model.

Raises:
ValueError: If *json_data* is not a JSON string.

classmethod model_validate_strings(*obj: Any*, *, *strict: bool | None = None*, *context: dict[str, Any] | None = None*) → Model

Validate the given object contains string data against the Pydantic model.

Args:
obj: The object contains string data to validate. strict: Whether to enforce types strictly. context: Extra variables to pass to the validator.

Returns:
The validated Pydantic model.

FactorResultModel

class zvt.factors.factor_models.FactorResultModel(*, *entity_ids: Optional[List[str]]*, *tag_reason: str*)

Bases: **`BaseModel`**

__init__(**data: Any*) → None

Create a new model by parsing and validating input data from keyword arguments.

Raises [*ValidationError*][pydantic_core.ValidationError] if the input data cannot be validated to form a valid model.

self is explicitly positional-only to allow *self* as a field name.

copy(*, *include: AbstractSetIntStr | MappingIntStrAny | None = None*, *exclude: AbstractSetIntStr | MappingIntStrAny | None = None*, *update: Dict[str, Any] | None = None*, *deep: bool = False*) → Model

Returns a copy of the model.

!!! warning "Deprecated"

This method is now deprecated; use *model_copy* instead.

If you need *include* or *exclude*, use:

```
`py data = self.model_dump(include=include, exclude=exclude, round_trip=True) data = {**data, **(update or {})} copied = self.model_validate(data) `
```

Args:

include: Optional set or mapping specifying which fields to include in the copied model. exclude: Optional set or mapping specifying which fields to exclude in the copied model. update: Optional dictionary of field-value pairs to override field values

in the copied model. deep: If True, the values of fields that are Pydantic models will be deep-copied.

Returns:

A copy of the model with included, excluded and updated fields as specified.

model_computed_fields*: ClassVar[dict[str, ComputedFieldInfo]] = {}*

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

model_config*: ClassVar[ConfigDict] = {}*

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

classmethod model_construct(*_fields_set: set[str] | None = None*, ***values: Any*) → Model

Creates a new instance of the *Model* class with validated data.

Creates a new model setting *__dict__* and *__pydantic_fields_set__* from trusted or pre-validated data. Default values are respected, but no other validation is performed. Behaves as if *Config.extra = 'allow'* was set since it adds all passed values

Args:

_fields_set: The set of field names accepted for the Model instance. values: Trusted or pre-validated data dictionary.

Returns:

A new instance of the *Model* class with validated data.

model_copy(*, *update: dict[str, Any] | None = None*, *deep: bool = False*) → Model

Usage docs: https://docs.pydantic.dev/2.6/concepts/serialization/#model_copy

Returns a copy of the model.

Args:

update: Values to change/add in the new model. Note: the data is not validated
before creating the new model. You should trust this data.

deep: Set to *True* to make a deep copy of the model.

Returns:

New model instance.

model_dump(*, *mode: Literal['json', 'python'] | str = 'python'*, *include: IncEx = None*, *exclude: IncEx = None*, *by_alias: bool = False*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*, *round_trip: bool = False*, *warnings: bool = True*) → dict[str, Any]

Usage docs: https://docs.pydantic.dev/2.6/concepts/serialization/#modelmodel_dump

Generate a dictionary representation of the model, optionally specifying which fields to include or exclude.

Args:

mode: The mode in which *to_python* should run.
If mode is 'json', the output will only contain JSON serializable types. If mode is 'python', the output may contain non-JSON-serializable Python objects.

include: A list of fields to include in the output. exclude: A list of fields to exclude from the output. by_alias: Whether to use the field's alias in the dictionary key if defined. exclude_unset: Whether to exclude fields that have not been explicitly set. exclude_defaults: Whether to exclude fields that are set to their default value. exclude_none: Whether to exclude fields that have a value of *None*. round_trip: If True, dumped values should be valid as input for non-idempotent types such as Json[T].

warnings: Whether to log warnings when invalid fields are encountered.

Returns:

A dictionary representation of the model.

model_dump_json(*, *indent: int | None = None*, *include: IncEx = None*, *exclude: IncEx = None*, *by_alias: bool = False*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*, *round_trip: bool = False*, *warnings: bool = True*) → str

Usage docs: https://docs.pydantic.dev/2.6/concepts/serialization/#modelmodel_dump_json

Generates a JSON representation of the model using Pydantic's *to_json* method.

Args:

indent: Indentation to use in the JSON output. If None is passed, the output will be compact. include: Field(s) to include in the JSON output. exclude: Field(s) to exclude from the JSON output. by_alias: Whether to serialize using field aliases. exclude_unset: Whether to exclude fields that have not been explicitly set. exclude_defaults: Whether to exclude fields that are set to their default value. exclude_none: Whether to exclude fields that have a value of *None*. round_trip: If True, dumped values should be valid as input for non-idempotent types such as Json[T]. warnings: Whether to log warnings when invalid fields are encountered.

Returns:

A JSON string representation of the model.

property model_extra*: dict[str, Any] | None*

Get extra fields set during validation.

Returns:

A dictionary of extra fields, or *None* if *config.extra* is not set to *"allow"*.

model_fields*: ClassVar[dict[str, FieldInfo]] = {'entity_ids': FieldInfo(annotation=Union[List[str], NoneType], required=True), 'tag_reason': FieldInfo(annotation=str, required=True)}*

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

property model_fields_set*: set[str]*

Returns the set of fields that have been explicitly set on this model instance.

Returns:

A set of strings representing the fields that have been set, i.e. that were not filled from defaults.

classmethod model_json_schema(*by_alias: bool = True*, *ref_template: str = '#/$defs/{model}'*, *schema_generator: type[pydantic.json_schema.GenerateJsonSchema] = <class 'pydantic.json_schema.GenerateJsonSchema'>*, *mode: ~typing.Literal['validation', 'serialization'] = 'validation'*) → dict[str, Any]

Generates a JSON schema for a model class.

Args:

by_alias: Whether to use attribute aliases or not. ref_template: The reference template. schema_generator: To override the logic used to generate the JSON schema, as a subclass of

GenerateJsonSchema with your desired modifications

mode: The mode in which to generate the schema.

Returns:

The JSON schema for the given model class.

classmethod model_parametrized_name(*params: tuple[type[Any], ...]*) → str

Compute the class name for parametrizations of generic classes.

This method can be overridden to achieve a custom naming scheme for generic BaseModels.

Args:

params: Tuple of types of the class. Given a generic class *Model* with 2 type variables and a concrete model *Model[str, int]*, the value *(str, int)* would be passed to *params*.

Returns:

String representing the new class where *params* are passed to *cls* as type variables.

Raises:

TypeError: Raised when trying to generate concrete names for non-generic models.

model_post_init(*_BaseModel__context: Any*) → None

Override this method to perform additional initialization after *__init__* and *model_construct*. This is useful if you want to do some validation that requires the entire model to be initialized.

classmethod model_rebuild(*, *force: bool = False*, *raise_errors: bool = True*, *_parent_namespace_depth: int = 2*, *_types_namespace: Optional[dict[str, Any]] = None*) → bool | None

Try to rebuild the pydantic-core schema for the model.

This may be necessary when one of the annotations is a ForwardRef which could not be resolved during the initial attempt to build the schema, and automatic rebuilding fails.

Args:

force: Whether to force the rebuilding of the model schema, defaults to *False*. raise_errors: Whether to raise errors, defaults to *True*. _parent_namespace_depth: The depth level of the parent namespace, defaults to 2. _types_namespace: The types namespace, defaults to *None*.

Returns:
Returns *None* if the schema is already "complete" and rebuilding was not required. If rebuilding _was_ required, returns *True* if rebuilding was successful, otherwise *False*.

classmethod model_validate(*obj: Any*, *, *strict: bool | None = None*, *from_attributes: bool | None = None*, *context: dict[str, Any] | None = None*) → Model

Validate a pydantic model instance.

Args:
obj: The object to validate. strict: Whether to enforce types strictly. from_attributes: Whether to extract data from object attributes. context: Additional context to pass to the validator.

Raises:
ValidationError: If the object could not be validated.

Returns:
The validated model instance.

classmethod model_validate_json(*json_data: str | bytes | bytearray*, *, *strict: bool | None = None*, *context: dict[str, Any] | None = None*) → Model

Usage docs: https://docs.pydantic.dev/2.6/concepts/json/#json-parsing

Validate the given JSON data against the Pydantic model.

Args:

json_data: The JSON data to validate. strict: Whether to enforce types strictly. context: Extra variables to pass to the validator.

Returns:
The validated Pydantic model.

Raises:
ValueError: If *json_data* is not a JSON string.

classmethod model_validate_strings(*obj: Any*, *, *strict: bool | None = None*, *context: dict[str, Any] | None = None*) → Model

Validate the given object contains string data against the Pydantic model.

Args:
obj: The object contains string data to validate. strict: Whether to enforce types strictly. context: Extra variables to pass to the validator.

Returns:
The validated Pydantic model.

KdataModel

class zvt.factors.factor_models.KdataModel(*, *entity_id: str*, *code: str*, *name: str*, *level: IntervalLevel = IntervalLevel.LEVEL_1DAY*, *datas: List*)

Bases: **`BaseModel`**

__init__(**data: Any*) → None

Create a new model by parsing and validating input data from keyword arguments.

Raises [*ValidationError*][pydantic_core.ValidationError] if the input data cannot be validated to form a valid model.

self is explicitly positional-only to allow *self* as a field name.

copy(*, *include: AbstractSetIntStr | MappingIntStrAny | None = None*, *exclude: AbstractSetIntStr | MappingIntStrAny | None = None*, *update: Dict[str, Any] | None = None*, *deep: bool = False*) → Model

Returns a copy of the model.

!!! warning "Deprecated"

This method is now deprecated; use *model_copy* instead.

If you need *include* or *exclude*, use:

```
`py data = self.model_dump(include=include, exclude=exclude, round_trip=True) data = {**data, **(update or {})} copied = self.model_validate(data) `
```

Args:

include: Optional set or mapping specifying which fields to include in the copied model. exclude: Optional set or mapping specifying which fields to exclude in the copied model. update: Optional dictionary of field-value pairs to override field values

in the copied model. deep: If True, the values of fields that are Pydantic models will be deep-copied.

Returns:

A copy of the model with included, excluded and updated fields as specified.

model_computed_fields*: ClassVar[dict[str, ComputedFieldInfo]] = {}*

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

model_config*: ClassVar[ConfigDict] = {}*

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

classmethod model_construct(*_fields_set: set[str] | None = None*, ***values: Any*) → Model

Creates a new instance of the *Model* class with validated data.

Creates a new model setting *__dict__* and *__pydantic_fields_set__* from trusted or pre-validated data. Default values are respected, but no other validation is performed. Behaves as if *Config.extra = 'allow'* was set since it adds all passed values

Args:

_fields_set: The set of field names accepted for the Model instance. values: Trusted or pre-validated data dictionary.

Returns:

A new instance of the *Model* class with validated data.

model_copy(*, *update: dict[str, Any] | None = None*, *deep: bool = False*) → Model

Usage docs: https://docs.pydantic.dev/2.6/concepts/serialization/#model_copy

Returns a copy of the model.

Args:

update: Values to change/add in the new model. Note: the data is not validated
before creating the new model. You should trust this data.

deep: Set to *True* to make a deep copy of the model.

Returns:

New model instance.

model_dump(*, *mode: Literal['json', 'python'] | str = 'python'*, *include: IncEx = None*, *exclude: IncEx = None*, *by_alias: bool = False*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*, *round_trip: bool = False*, *warnings: bool = True*) → dict[str, Any]

Usage docs: https://docs.pydantic.dev/2.6/concepts/serialization/#modelmodel_dump

Generate a dictionary representation of the model, optionally specifying which fields to include or exclude.

Args:

mode: The mode in which *to_python* should run.
If mode is 'json', the output will only contain JSON serializable types. If mode is 'python', the output may contain non-JSON-serializable Python objects.

include: A list of fields to include in the output. exclude: A list of fields to exclude from the output. by_alias: Whether to use the field's alias in the dictionary key if defined. exclude_unset: Whether to exclude fields that have not been explicitly set. exclude_defaults: Whether to exclude fields that are set to their default value. exclude_none: Whether to exclude fields that have a value of *None*. round_trip: If True, dumped values should be valid as input for non-idempotent types such as Json[T].

warnings: Whether to log warnings when invalid fields are encountered.

Returns:

A dictionary representation of the model.

model_dump_json(*, *indent: int | None = None*, *include: IncEx = None*, *exclude: IncEx = None*, *by_alias: bool = False*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*, *round_trip: bool = False*, *warnings: bool = True*) → str

Usage docs: https://docs.pydantic.dev/2.6/concepts/serialization/#modelmodel_dump_json

Generates a JSON representation of the model using Pydantic's *to_json* method.

Args:

indent: Indentation to use in the JSON output. If None is passed, the output will be compact. include: Field(s) to include in the JSON output. exclude: Field(s) to exclude from the JSON output. by_alias: Whether to serialize using field aliases. exclude_unset: Whether to exclude fields that have not been explicitly set. exclude_defaults: Whether to exclude fields that are set to their default value. exclude_none: Whether to exclude fields that have a value of *None*. round_trip: If True, dumped values should be valid as input for non-idempotent types such as Json[T]. warnings: Whether to log warnings when invalid fields are encountered.

Returns:

A JSON string representation of the model.

property model_extra*: dict[str, Any] | None*

Get extra fields set during validation.

Returns:

A dictionary of extra fields, or *None* if *config.extra* is not set to *"allow"*.

model_fields*: ClassVar[dict[str, FieldInfo]] = {'code': FieldInfo(annotation=str, required=True), 'datas': FieldInfo(annotation=List, required=True), 'entity_id': FieldInfo(annotation=str, required=True), 'level': FieldInfo(annotation=IntervalLevel, required=False, default= <IntervalLevel.LEVEL_1DAY: '1d'>), 'name': FieldInfo(annotation=str, required=True)}*

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

property model_fields_set*: set[str]*

Returns the set of fields that have been explicitly set on this model instance.

Returns:

A set of strings representing the fields that have been set, i.e. that were not filled from defaults.

classmethod model_json_schema(*by_alias: bool = True*, *ref_template: str = '#/$defs/{model}'*, *schema_generator: type[pydantic.json_schema.GenerateJsonSchema] = <class 'pydantic.json_schema.GenerateJsonSchema'>*, *mode: ~typing.Literal['validation', 'serialization'] = 'validation'*) → dict[str, Any]

Generates a JSON schema for a model class.

Args:

by_alias: Whether to use attribute aliases or not. ref_template: The reference template. schema_generator: To override the logic used to generate the JSON schema, as a subclass of

GenerateJsonSchema with your desired modifications

mode: The mode in which to generate the schema.

Returns:

The JSON schema for the given model class.

classmethod model_parametrized_name(*params: tuple[type[Any], ...]*) → str

Compute the class name for parametrizations of generic classes.

This method can be overridden to achieve a custom naming scheme for generic BaseModels.

Args:

params: Tuple of types of the class. Given a generic class *Model* with 2 type variables and a concrete model *Model[str, int]*, the value *(str, int)* would be passed to *params*.

Returns:

String representing the new class where *params* are passed to *cls* as type variables.

Raises:

TypeError: Raised when trying to generate concrete names for non-generic models.

model_post_init(*_BaseModel__context: Any*) → None

Override this method to perform additional initialization after *__init__* and *model_construct*. This is useful if you want to do some validation that requires the entire model to be initialized.

classmethod model_rebuild(*, *force: bool = False*, *raise_errors: bool = True*, *_parent_namespace_depth: int = 2*, *_types_namespace: Optional[dict[str, Any]] = None*) → bool | None

Try to rebuild the pydantic-core schema for the model.

This may be necessary when one of the annotations is a ForwardRef which could not be resolved during the initial attempt to build the schema, and automatic rebuilding fails.

Args:

force: Whether to force the rebuilding of the model schema, defaults to *False*. raise_errors: Whether to raise errors, defaults to *True*. _parent_namespace_depth: The depth level of the parent namespace, defaults to 2. _types_namespace: The types namespace, defaults to *None*.

Returns:

Returns *None* if the schema is already "complete" and rebuilding was not required. If rebuilding _was_ required, returns *True* if rebuilding was successful, otherwise *False*.

classmethod model_validate(*obj: Any*, *, *strict: bool | None = None*, *from_attributes: bool | None = None*, *context: dict[str, Any] | None = None*) → Model

Validate a pydantic model instance.

Args:

obj: The object to validate. strict: Whether to enforce types strictly. from_attributes: Whether to extract data from object attributes. context: Additional context to pass to the validator.

Raises:

ValidationError: If the object could not be validated.

Returns:

The validated model instance.

classmethod model_validate_json(*json_data: str | bytes | bytearray*, *, *strict: bool | None = None*, *context: dict[str, Any] | None = None*) → Model

Usage docs: https://docs.pydantic.dev/2.6/concepts/json/#json-parsing

Validate the given JSON data against the Pydantic model.

Args:

json_data: The JSON data to validate. strict: Whether to enforce types strictly. context: Extra variables to pass to the validator.

Returns:
The validated Pydantic model.

Raises:
ValueError: If *json_data* is not a JSON string.

classmethod model_validate_strings(*obj: Any*, *, *strict: bool | None = None*, *context: dict[str, Any] | None = None*) → Model

Validate the given object contains string data against the Pydantic model.

Args:
obj: The object contains string data to validate. strict: Whether to enforce types strictly. context: Extra variables to pass to the validator.

Returns:
The validated Pydantic model.

KdataRequestModel

class zvt.factors.factor_models.KdataRequestModel(*, *entity_ids: List[str]*, *data_provider: str = 'em'*, *start_timestamp: datetime = Timestamp('2023-08-29 00:00:00')*, *end_timestamp: Optional[datetime] = None*, *level: IntervalLevel = IntervalLevel.LEVEL_1DAY*, *adjust_type: AdjustType = AdjustType.hfq*)

Bases: **`BaseModel`**

__init__(**data: Any*) → None

Create a new model by parsing and validating input data from keyword arguments.

Raises [*ValidationError*][pydantic_core.ValidationError] if the input data cannot be validated to form a valid model.

self is explicitly positional-only to allow *self* as a field name.

copy(*, *include: AbstractSetIntStr | MappingIntStrAny | None = None*, *exclude: AbstractSetIntStr | MappingIntStrAny | None = None*, *update: Dict[str, Any] | None = None*, *deep: bool = False*) → Model

Returns a copy of the model.

!!! warning "Deprecated"

This method is now deprecated; use *model_copy* instead.

If you need *include* or *exclude*, use:

```
`py data = self.model_dump(include=include, exclude=exclude, round_trip=True) data = {**data, **(update or {})} copied = self.model_validate(data) `
```

Args:

include: Optional set or mapping specifying which fields to include in the copied model. exclude: Optional set or mapping specifying which fields to exclude in the copied model. update: Optional dictionary of field-value pairs to override field values in the copied model. deep: If True, the values of fields that are Pydantic models will be deep-copied.

Returns:
A copy of the model with included, excluded and updated fields as specified.

model_computed_fields*: ClassVar[dict[str, ComputedFieldInfo]] = {}*
A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

model_config*: ClassVar[ConfigDict] = {}*
Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

classmethod model_construct(*_fields_set: set[str] | None = None*, ***values: Any*) → Model
Creates a new instance of the *Model* class with validated data.

Creates a new model setting *__dict__* and *__pydantic_fields_set__* from trusted or pre-validated data. Default values are respected, but no other validation is performed. Behaves as if *Config.extra = 'allow'* was set since it adds all passed values

Args:
_fields_set: The set of field names accepted for the Model instance. values: Trusted or pre-validated data dictionary.

Returns:
A new instance of the *Model* class with validated data.

model_copy(*, *update: dict[str, Any] | None = None*, *deep: bool = False*) → Model

Usage docs:
https://docs.pydantic.dev/2.6/concepts/serialization/#model_copy

Returns a copy of the model.

Args:

update: Values to change/add in the new model. Note: the data is not validated
before creating the new model. You should trust this data.

deep: Set to *True* to make a deep copy of the model.

Returns:

New model instance.

model_dump(*, *mode: Literal['json', 'python'] | str = 'python'*, *include: IncEx = None*, *exclude: IncEx = None*, *by_alias: bool = False*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*, *round_trip: bool = False*, *warnings: bool = True*) → dict[str, Any]

Usage docs:
https://docs.pydantic.dev/2.6/concepts/serialization/#modelmodel_dump

Generate a dictionary representation of the model, optionally specifying which fields to include or exclude.

Args:

mode: The mode in which *to_python* should run.
If mode is 'json', the output will only contain JSON serializable types. If mode is 'python', the output may contain non-JSON-serializable Python objects.

include: A list of fields to include in the output. exclude: A list of fields to exclude from the output. by_alias: Whether to use the field's alias in the dictionary key if defined. exclude_unset: Whether to exclude fields that have not been explicitly set.

exclude_defaults: Whether to exclude fields that are set to their default value. exclude_none: Whether to exclude fields that have a value of *None*. round_trip: If True, dumped values should be valid as input for non-idempotent types such as Json[T]. warnings: Whether to log warnings when invalid fields are encountered.

Returns:

A dictionary representation of the model.

model_dump_json(*, *indent: int | None = None*, *include: IncEx = None*, *exclude: IncEx = None*, *by_alias: bool = False*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*, *round_trip: bool = False*, *warnings: bool = True*) → str

Usage docs: https://docs.pydantic.dev/2.6/concepts/serialization/#modelmodel_dump_json

Generates a JSON representation of the model using Pydantic's *to_json* method.

Args:

indent: Indentation to use in the JSON output. If None is passed, the output will be compact. include: Field(s) to include in the JSON output. exclude: Field(s) to exclude from the JSON output. by_alias: Whether to serialize using field aliases. exclude_unset: Whether to exclude fields that have not been explicitly set. exclude_defaults: Whether to exclude fields that are set to their default value. exclude_none: Whether to exclude fields that have a value of *None*. round_trip: If True, dumped values should be valid as input for non-idempotent types such as Json[T]. warnings: Whether to log warnings when invalid fields are encountered.

Returns:

A JSON string representation of the model.

property model_extra*: dict[str, Any] | None*

Get extra fields set during validation.

Returns:
A dictionary of extra fields, or *None* if *config.extra* is not set to *"allow"*.

model_fields*: ClassVar[dict[str, FieldInfo]] = {'adjust_type': FieldInfo(annotation=AdjustType, required=False, default= <AdjustType.hfq: 'hfq'>), 'data_provider': FieldInfo(annotation=str, required=False, default='em'), 'end_timestamp': FieldInfo(annotation=Union[datetime, NoneType], required=False), 'entity_ids': FieldInfo(annotation=List[str], required=True), 'level': FieldInfo(annotation=IntervalLevel, required=False, default= <IntervalLevel.LEVEL_1DAY: '1d'>), 'start_timestamp': FieldInfo(annotation=datetime, required=False, default=Timestamp('2023-08-29 00:00:00'))}*

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

property model_fields_set*: set[str]*

Returns the set of fields that have been explicitly set on this model instance.

Returns:
A set of strings representing the fields that have been set,
i.e. that were not filled from defaults.

classmethod model_json_schema(*by_alias: bool = True*, *ref_template: str = '#/$defs/{model}'*, *schema_generator: type[pydantic.json_schema.GenerateJsonSchema] = <class 'pydantic.json_schema.GenerateJsonSchema'>*, *mode: ~typing.Literal['validation', 'serialization'] = 'validation'*) → dict[str, Any]

Generates a JSON schema for a model class.

Args:

by_alias: Whether to use attribute aliases or not. ref_template: The reference template. schema_generator: To override the logic used to generate the JSON schema, as a subclass of

GenerateJsonSchema with your desired modifications

mode: The mode in which to generate the schema.

Returns:

The JSON schema for the given model class.

classmethod model_parametrized_name(*params: tuple[type[Any], ...]*) → str

Compute the class name for parametrizations of generic classes.

This method can be overridden to achieve a custom naming scheme for generic BaseModels.

Args:

params: Tuple of types of the class. Given a generic class *Model* with 2 type variables and a concrete model *Model[str, int]*, the value *(str, int)* would be passed to *params*.

Returns:

String representing the new class where *params* are passed to *cls* as type variables.

Raises:

TypeError: Raised when trying to generate concrete names for non-generic models.

model_post_init(*_BaseModel__context: Any*) → None

Override this method to perform additional initialization after *__init__* and *model_construct*. This is useful if you want to do some validation that requires the entire model to be initialized.

classmethod model_rebuild(*, *force: bool = False*, *raise_errors: bool = True*, *_parent_namespace_depth: int = 2*, *_types_namespace: Optional[dict[str, Any]] = None*) → bool | None

Try to rebuild the pydantic-core schema for the model.

This may be necessary when one of the annotations is a ForwardRef which could not be resolved during the initial attempt to build the schema, and automatic rebuilding fails.

Args:

force: Whether to force the rebuilding of the model schema, defaults to *False*. raise_errors: Whether to raise errors, defaults to *True*. _parent_namespace_depth: The depth level of the parent namespace, defaults to 2. _types_namespace: The types namespace, defaults to *None*.

Returns:

Returns *None* if the schema is already "complete" and rebuilding was not required. If rebuilding _was_ required, returns *True* if rebuilding was successful, otherwise *False*.

classmethod model_validate(*obj: Any*, *, *strict: bool | None = None*, *from_attributes: bool | None = None*, *context: dict[str, Any] | None = None*) → Model

Validate a pydantic model instance.

Args:

obj: The object to validate. strict: Whether to enforce types strictly. from_attributes: Whether to extract data from object attributes. context: Additional context to pass to the validator.

Raises:

ValidationError: If the object could not be validated.

Returns:

The validated model instance.

classmethod model_validate_json(*json_data: str | bytes | bytearray*, *, *strict: bool | None = None*, *context: dict[str, Any] | None = None*) → Model

Usage docs: https://docs.pydantic.dev/2.6/concepts/json/#json-parsing

Validate the given JSON data against the Pydantic model.

Args:

json_data: The JSON data to validate. strict: Whether to enforce types strictly. context: Extra variables to pass to the validator.

Returns:

The validated Pydantic model.

Raises:

ValueError: If *json_data* is not a JSON string.

classmethod model_validate_strings(*obj: Any*, *, *strict: bool | None = None*, *context: dict[str, Any] | None = None*) → Model

Validate the given object contains string data against the Pydantic model.

Args:

obj: The object contains string data to validate. strict: Whether to enforce types strictly. context: Extra variables to pass to the validator.

Returns:

The validated Pydantic model.

TradingSignalModel

class zvt.factors.factor_models.TradingSignalModel(*, *entity_id: str*, *happen_timestamp: datetime*, *due_timestamp: datetime*, *trading_level: IntervalLevel = IntervalLevel.LEVEL_1DAY*, *trading_signal_type: TradingSignalType*, *position_pct: Optional[float] = 0.2*, *order_amount: Optional[float] = None*, *order_money: Optional[float] = None*)

Bases: **`BaseModel`**

__init__(***data: Any*) → None

Create a new model by parsing and validating input data from keyword arguments.

Raises [*ValidationError*][pydantic_core.ValidationError] if the input data cannot be validated to form a valid model.

self is explicitly positional-only to allow *self* as a field name.

copy(*, *include: AbstractSetIntStr | MappingIntStrAny | None = None*, *exclude: AbstractSetIntStr | MappingIntStrAny | None = None*, *update: Dict[str, Any] | None = None*, *deep: bool = False*) → Model

Returns a copy of the model.

!!! warning "Deprecated"

This method is now deprecated; use *model_copy* instead.

If you need *include* or *exclude*, use:

```
`py data = self.model_dump(include=include, exclude=exclude, round_trip=True) data = {**data, **(update or {})} copied = self.model_validate(data) `
```

Args:

include: Optional set or mapping specifying which fields to include in the copied model. exclude: Optional set or mapping specifying which fields to exclude in the copied model. update: Optional dictionary of field-value pairs to override field values in the copied model. deep: If True, the values of fields that are Pydantic models will be deep-copied.

Returns:
A copy of the model with included, excluded and updated fields as specified.

model_computed_fields*: ClassVar[dict[str, ComputedFieldInfo]] = {}*
A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

model_config*: ClassVar[ConfigDict] = {}*
Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

classmethod model_construct(*_fields_set: set[str] | None = None*, ***values: Any*) → Model
Creates a new instance of the *Model* class with validated data.

Creates a new model setting *__dict__* and *__pydantic_fields_set__* from trusted or pre-validated data. Default values are respected, but no other validation is performed. Behaves as if *Config.extra = 'allow'* was set since it adds all passed values

Args:
_fields_set: The set of field names accepted for the Model instance. values: Trusted or pre-validated data dictionary.

Returns:
A new instance of the *Model* class with validated data.

model_copy(*, *update: dict[str, Any] | None = None*, *deep: bool = False*) → Model

Usage docs:
https://docs.pydantic.dev/2.6/concepts/serialization/#model_copy

Returns a copy of the model.

Args:

update: Values to change/add in the new model. Note: the data is not validated
before creating the new model. You should trust this data.

deep: Set to *True* to make a deep copy of the model.

Returns:

New model instance.

model_dump(*, *mode: Literal['json', 'python'] | str = 'python'*, *include: IncEx = None*, *exclude: IncEx = None*, *by_alias: bool = False*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*, *round_trip: bool = False*, *warnings: bool = True*) → dict[str, Any]

Usage docs:
https://docs.pydantic.dev/2.6/concepts/serialization/#modelmodel_dump

Generate a dictionary representation of the model, optionally specifying which fields to include or exclude.

Args:

mode: The mode in which *to_python* should run.
If mode is 'json', the output will only contain JSON serializable types. If mode is 'python', the output may contain non-JSON-serializable Python objects.

include: A list of fields to include in the output. exclude: A list of fields to exclude from the output. by_alias: Whether to use the field's alias in the dictionary key if defined. exclude_unset: Whether to exclude fields that have not been explicitly set.

exclude_defaults: Whether to exclude fields that are set to their default value. exclude_none: Whether to exclude fields that have a value of *None*. round_trip: If True, dumped values should be valid as input for non-idempotent types such as Json[T]. warnings: Whether to log warnings when invalid fields are encountered.

Returns:

A dictionary representation of the model.

model_dump_json(*, *indent: int | None = None*, *include: IncEx = None*, *exclude: IncEx = None*, *by_alias: bool = False*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*, *round_trip: bool = False*, *warnings: bool = True*) → str

Usage docs: https://docs.pydantic.dev/2.6/concepts/serialization/#modelmodel_dump_json

Generates a JSON representation of the model using Pydantic's *to_json* method.

Args:

indent: Indentation to use in the JSON output. If None is passed, the output will be compact. include: Field(s) to include in the JSON output. exclude: Field(s) to exclude from the JSON output. by_alias: Whether to serialize using field aliases. exclude_unset: Whether to exclude fields that have not been explicitly set. exclude_defaults: Whether to exclude fields that are set to their default value. exclude_none: Whether to exclude fields that have a value of *None*. round_trip: If True, dumped values should be valid as input for non-idempotent types such as Json[T]. warnings: Whether to log warnings when invalid fields are encountered.

Returns:

A JSON string representation of the model.

property model_extra*: dict[str, Any] | None*

Get extra fields set during validation.

Returns:
A dictionary of extra fields, or *None* if *config.extra* is not set to *"allow"*.

model_fields*: ClassVar[dict[str, FieldInfo]] = {'due_timestamp': FieldInfo(annotation=datetime, required=True), 'entity_id': FieldInfo(annotation=str, required=True), 'happen_timestamp': FieldInfo(annotation=datetime, required=True), 'order_amount': FieldInfo(annotation=Union[float, NoneType], required=False), 'order_money': FieldInfo(annotation=Union[float, NoneType], required=False), 'position_pct': FieldInfo(annotation=Union[float, NoneType], required=False, default=0.2), 'trading_level': FieldInfo(annotation=IntervalLevel, required=False, default= <IntervalLevel.LEVEL_1DAY: '1d'>), 'trading_signal_type': FieldInfo(annotation=TradingSignalType, required=True)}*

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

property model_fields_set*: set[str]*

Returns the set of fields that have been explicitly set on this model instance.

Returns:
A set of strings representing the fields that have been set,
i.e. that were not filled from defaults.

classmethod model_json_schema(*by_alias: bool = True*, *ref_template: str = '#/$defs/{model}'*, *schema_generator: type[pydantic.json_schema.GenerateJsonSchema] = <class 'pydantic.json_schema.GenerateJsonSchema'>*, *mode: ~typing.Literal['validation', 'serialization'] = 'validation'*) → dict[str, Any]

Generates a JSON schema for a model class.

Args:

by_alias: Whether to use attribute aliases or not. ref_template: The reference template. schema_generator: To override the logic used to generate the JSON schema, as a subclass of

GenerateJsonSchema with your desired modifications

mode: The mode in which to generate the schema.

Returns:

The JSON schema for the given model class.

classmethod model_parametrized_name(*params: tuple[type[Any], ...]*) → str

Compute the class name for parametrizations of generic classes.

This method can be overridden to achieve a custom naming scheme for generic BaseModels.

Args:

params: Tuple of types of the class. Given a generic class *Model* with 2 type variables and a concrete model *Model[str, int]*, the value *(str, int)* would be passed to *params*.

Returns:

String representing the new class where *params* are passed to *cls* as type variables.

Raises:

TypeError: Raised when trying to generate concrete names for non-generic models.

model_post_init(*_BaseModel__context: Any*) → None

Override this method to perform additional initialization after *__init__* and *model_construct*. This is useful if you want to do some validation that requires the entire model to be initialized.

classmethod model_rebuild(*, *force: bool = False*, *raise_errors: bool = True*, *_parent_namespace_depth: int = 2*, *_types_namespace: Optional[dict[str, Any]] = None*) → bool | None

Try to rebuild the pydantic-core schema for the model.

This may be necessary when one of the annotations is a ForwardRef which could not be resolved during the initial attempt to build the schema, and automatic rebuilding fails.

Args:

force: Whether to force the rebuilding of the model schema, defaults to *False*. raise_errors: Whether to raise errors, defaults to *True*. _parent_namespace_depth: The depth level of the parent namespace, defaults to 2. _types_namespace: The types namespace, defaults to *None*.

Returns:

Returns *None* if the schema is already "complete" and rebuilding was not required. If rebuilding _was_ required, returns *True* if rebuilding was successful, otherwise *False*.

classmethod model_validate(*obj: Any*, *, *strict: bool | None = None*, *from_attributes: bool | None = None*, *context: dict[str, Any] | None = None*) → Model

Validate a pydantic model instance.

Args:

obj: The object to validate. strict: Whether to enforce types strictly. from_attributes: Whether to extract data from object attributes. context: Additional context to pass to the validator.

Raises:

ValidationError: If the object could not be validated.

Returns:

The validated model instance.

classmethod model_validate_json(*json_data: str | bytes | bytearray*, *, *strict: bool | None = None*, *context: dict[str, Any] | None = None*) → Model

Usage docs: https://docs.pydantic.dev/2.6/concepts/json/#json-parsing

Validate the given JSON data against the Pydantic model.

Args:

json_data: The JSON data to validate. strict: Whether to enforce types strictly. context: Extra variables to pass to the validator.

Returns:

The validated Pydantic model.

Raises:

ValueError: If *json_data* is not a JSON string.

classmethod model_validate_strings(*obj: Any*, *, *strict: bool | None = None*, *context: dict[str, Any] | None = None*) → Model

Validate the given object contains string data against the Pydantic model.

Args:

obj: The object contains string data to validate. strict: Whether to enforce types strictly. context: Extra variables to pass to the validator.

Returns:

The validated Pydantic model.

zvt.factors.factor_service

Functions

`query_factor_result`(factor_request_model)

`query_kdata`(kdata_request_model)

---

zvt.factors.factor_service.query_factor_result

zvt.factors.factor_service.query_factor_result(*factor_request_model: FactorRequestModel*)

zvt.factors.factor_service.query_kdata

zvt.factors.factor_service.query_kdata(*kdata_request_model: KdataRequestModel*)

zvt.factors.fundamental

Modules

[zvt.factors.fundamental.finance_factor](#)

---

zvt.factors.fundamental.finance_factor

Classes

`FinanceBaseFactor`(data_schema, ...)

`GoodCompanyFactor`(data_schema, ...[, ...])

FinanceBaseFactor

class zvt.factors.fundamental.finance_factor.FinanceBaseFactor(*data_schema: ~typing.Type[~zvt.contract.schema.Mixin] = <class 'zvt.domain.fundamental.finance.FinanceFactor'>, entity_schema: ~typing.Type[~zvt.contract.schema.TradableEntity] = <class 'zvt.domain.meta.stock_meta.Stock'>, provider: ~typing.Optional[str] = None, entity_provider: ~typing.Optional[str] = None, entity_ids: ~typing.Optional[~typing.List[str]] = None, exchanges: ~typing.Optional[~typing.List[str]] = None, codes: ~typing.Optional[~typing.List[str]] = None, start_timestamp: ~typing.Optional[~typing.Union[~pandas._libs.tslibs.timestamps.Timestamp, str]] = None, end_timestamp: ~typing.Optional[~typing.Union[~pandas._libs.tslibs.timestamps.Timestamp, str]] = None, columns: ~typing.Optional[~typing.List] = None, filters: ~typing.Optional[~typing.List] = None, order: ~typing.Optional[object] = None, limit: ~typing.Optional[int] = None, level: ~typing.Optional[~typing.Union[~zvt.contract.IntervalLevel, str]] = None, category_field: str = 'entity_id', time_field: str = 'timestamp', keep_window: ~typing.Optional[int] = None, keep_all_timestamp: bool = False, fill_method: str = 'ffill', effective_number: ~typing.Optional[int] = None, transformer: ~typing.Optional[~zvt.contract.factor.Transformer] = None, accumulator: ~typing.Optional[~zvt.contract.factor.Accumulator] = None, need_persist: bool = False, only_compute_factor: bool = False, factor_name: ~typing.Optional[str] = None, clear_state: bool = False, only_load_factor: bool = False*)

Bases: **Factor**

__init__(*data_schema: ~typing.Type[~zvt.contract.schema.Mixin] = <class 'zvt.domain.fundamental.finance.FinanceFactor'>, entity_schema: ~typing.Type[~zvt.contract.schema.TradableEntity] = <class 'zvt.domain.meta.stock_meta.Stock'>, provider: ~typing.Optional[str] = None, entity_provider: ~typing.Optional[str] = None, entity_ids: ~typing.Optional[~typing.List[str]] = None, exchanges:*

~typing.Optional[~typing.List[str]] = None, *codes: ~typing.Optional[~typing.List[str]] = None*, *start_timestamp: ~typing.Optional[~typing.Union[~pandas._libs.tslibs.timestamps.Timestamp*, *str]] = None*, *end_timestamp: ~typing.Optional[~typing.Union[~pandas._libs.tslibs.timestamps.Timestamp*, *str]] = None*, *columns: ~typing.Optional[~typing.List] = None*, *filters: ~typing.Optional[~typing.List] = None*, *order: ~typing.Optional[object] = None*, *limit: ~typing.Optional[int] = None*, *level: ~typing.Optional[~typing.Union[~zvt.contract.IntervalLevel*, *str]] = None*, *category_field: str = 'entity_id'*, *time_field: str = 'timestamp'*, *keep_window: ~typing.Optional[int] = None*, *keep_all_timestamp: bool = False*, *fill_method: str = 'ffill'*, *effective_number: ~typing.Optional[int] = None*, *transformer: ~typing.Optional[~zvt.contract.factor.Transformer] = None*, *accumulator: ~typing.Optional[~zvt.contract.factor.Accumulator] = None*, *need_persist: bool = False*, *only_compute_factor: bool = False*, *factor_name: ~typing.Optional[str] = None*, *clear_state: bool = False*, *only_load_factor: bool = False*) → None

Parameters::

- **keep_all_timestamp** –
- **fill_method** –
- **effective_number** –
- **transformer** –
- **accumulator** –
- **need_persist** – whether persist factor
- **only_compute_factor** – only compute factor nor result
- **factor_name** –
- **clear_state** –
- **only_load_factor** – only load factor and compute result

accumulator*: Accumulator = None*

accumulator for this factor if not passed as __init__ argument

clear_state_data(*entity_id=None*)

clear state of the entity

Parameters:: **entity_id** – entity id

decode_state(*state: str*)

decode state

Parameters:: **state** –

Returns::

encode_state(*state: object*)

encode state

Parameters:: **state** –

Returns::

factor_col_map_object_hook() → dict

:return:{col:object_hook}

factor_schema*: Type[Mixin] = None*

define the schema for persist,its columns should be same as indicators in transformer or accumulator

move_on(*to_timestamp: Optional[Union[Timestamp, str]] = None*, *timeout: int = 20*) → object

using continual fetching data in realtime 1)get the data happened before to_timestamp,if not set,get all the data which means to now 2)if computing_window set,the data_df would be cut for saving memory

Parameters::

- **to_timestamp** –
- **timeout** (*int*) –

Returns::

Return type::

name = *None*

name of the service, default name of class if not set manually

on_data_changed(*data: DataFrame*)

overwrite it for computing after data added

Parameters:: **data** –

on_data_loaded(*data: DataFrame*)

Parameters:: **data** –

on_entity_data_changed(*entity*, *added_data: DataFrame*)

overwrite it for computing after entity data added

Parameters::

- **entity** –
- **added_data** –

state_schema

alias of `FactorState`

transformer*: Transformer = None*

transformer for this factor if not passed as __init__ argument

pipe_df*: pd.DataFrame*

中间结果，不持久化 data_df->pipe_df

factor_df*: pd.DataFrame*

计算因子的结果，可持久化,通过对pipe_df的计算得到 pipe_df->factor_df

result_df*: pd.DataFrame*

result_df是用于选股的标准df,通过对factor_df的计算得到 factor_df->result_df

states*: dict*

entity_id:state

GoodCompanyFactor

class
zvt.factors.fundamental.finance_factor.GoodCompanyFactor(*data_schema: ~typing.Type[~zvt.contract.schema.Mixin] = <class 'zvt.domain.fundamental.finance.FinanceFactor'>*, *entity_schema: ~zvt.contract.schema.TradableEntity = <class 'zvt.domain.meta.stock_meta.Stock'>*, *provider: ~typing.Optional[str] = None*, *entity_provider: ~typing.Optional[str] = None*, *entity_ids: ~typing.Optional[~typing.List[str]] = None*, *exchanges: ~typing.Optional[~typing.List[str]] = None*, *codes: ~typing.Optional[~typing.List[str]] = None*, *start_timestamp: ~typing.Optional[~typing.Union[~pandas._libs.tslibs.timestamps.Timestamp*, *str]] = None*, *end_timestamp: ~typing.Optional[~typing.Union[~pandas._libs.tslibs.timestamps.Timestamp*, *str]] = None*, *columns: ~typing.List = (<sqlalchemy.orm.attributes.InstrumentedAttribute object>*, *<sqlalchemy.orm.attributes.InstrumentedAttribute object>*, *<sqlalchemy.orm.attributes.InstrumentedAttribute object>*, *<sqlalchemy.orm.attributes.InstrumentedAttribute object>*, *<sqlalchemy.orm.attributes.InstrumentedAttribute object>*, *<sqlalchemy.orm.attributes.InstrumentedAttribute object>*, *<sqlalchemy.orm.attributes.InstrumentedAttribute object>*, *<sqlalchemy.orm.attributes.InstrumentedAttribute object>)*, *filters: ~typing.List = (<sqlalchemy.sql.elements.BinaryExpression object>*, *<sqlalchemy.sql.elements.BinaryExpression object>*, *<sqlalchemy.sql.elements.BinaryExpression object>*, *<sqlalchemy.sql.elements.BinaryExpression object>*, *<sqlalchemy.sql.elements.BinaryExpression object>*, *<sqlalchemy.sql.elements.BinaryExpression object>*, *<sqlalchemy.sql.elements.BinaryExpression object>)*, *order: ~typing.Optional[object] = None*, *limit: ~typing.Optional[int] = None*, *level: ~typing.Union[str*, *~zvt.contract.IntervalLevel] = IntervalLevel.LEVEL_1DAY*, *category_field: str = 'entity_id'*, *time_field: str = 'timestamp'*, *keep_window: ~typing.Optional[int] = None*,

keep_all_timestamp: bool = True, fill_method: str = 'ffill', effective_number: ~typing.Optional[int] = None, transformer: ~typing.Optional[~zvt.contract.factor.Transformer] = None, accumulator: ~typing.Optional[~zvt.contract.factor.Accumulator] = None, need_persist: bool = False, only_compute_factor: bool = False, factor_name: ~typing.Optional[str] = None, clear_state: bool = False, only_load_factor: bool = False, window='1095d', count=8, col_period_threshold={'roe': 0.02})

Bases: **FinanceBaseFactor**

__init__(*data_schema: ~typing.Type[~zvt.contract.schema.Mixin] = <class 'zvt.domain.fundamental.finance.FinanceFactor'>, entity_schema: ~zvt.contract.schema.TradableEntity = <class 'zvt.domain.meta.stock_meta.Stock'>, provider: ~typing.Optional[str] = None, entity_provider: ~typing.Optional[str] = None, entity_ids: ~typing.Optional[~typing.List[str]] = None, exchanges: ~typing.Optional[~typing.List[str]] = None, codes: ~typing.Optional[~typing.List[str]] = None, start_timestamp: ~typing.Optional[~typing.Union[~pandas._libs.tslibs.timestamps.Timestamp, str]] = None, end_timestamp: ~typing.Optional[~typing.Union[~pandas._libs.tslibs.timestamps.Timestamp, str]] = None, columns: ~typing.List = (<sqlalchemy.orm.attributes.InstrumentedAttribute object>, <sqlalchemy.orm.attributes.InstrumentedAttribute object>, <sqlalchemy.orm.attributes.InstrumentedAttribute object>, <sqlalchemy.orm.attributes.InstrumentedAttribute object>, <sqlalchemy.orm.attributes.InstrumentedAttribute object>, <sqlalchemy.orm.attributes.InstrumentedAttribute object>, <sqlalchemy.orm.attributes.InstrumentedAttribute object>, <sqlalchemy.orm.attributes.InstrumentedAttribute object>), filters: ~typing.List = (<sqlalchemy.sql.elements.BinaryExpression object>, <sqlalchemy.sql.elements.BinaryExpression object>, <sqlalchemy.sql.elements.BinaryExpression object>, <sqlalchemy.sql.elements.BinaryExpression object>, <sqlalchemy.sql.elements.BinaryExpression object>, <sqlalchemy.sql.elements.BinaryExpression object>, <sqlalchemy.sql.elements.BinaryExpression object>), order: ~typing.Optional[object] = None, limit: ~typing.Optional[int] = None,*

level: ~typing.Union[str, ~zvt.contract.IntervalLevel] = IntervalLevel.LEVEL_1DAY, category_field: str = 'entity_id', time_field: str = 'timestamp', keep_window: ~typing.Optional[int] = None, keep_all_timestamp: bool = True, fill_method: str = 'ffill', effective_number: ~typing.Optional[int] = None, transformer: ~typing.Optional[~zvt.contract.factor.Transformer] = None, accumulator: ~typing.Optional[~zvt.contract.factor.Accumulator] = None, need_persist: bool = False, only_compute_factor: bool = False, factor_name: ~typing.Optional[str] = None, clear_state: bool = False, only_load_factor: bool = False, window='1095d', count=8, col_period_threshold={'roe': 0.02}) → None

Parameters::

- **keep_all_timestamp** –
- **fill_method** –
- **effective_number** –
- **transformer** –
- **accumulator** –
- **need_persist** – whether persist factor
- **only_compute_factor** – only compute factor nor result
- **factor_name** –
- **clear_state** –
- **only_load_factor** – only load factor and compute result

accumulator*: Accumulator = None*

accumulator for this factor if not passed as __init__ argument

clear_state_data(*entity_id=None*)

clear state of the entity

Parameters:: **entity_id** – entity id

decode_state(*state: str*)

decode state

Parameters:: **state** –

Returns::

encode_state(*state: object*)

encode state

Parameters:: **state** –

Returns::

factor_col_map_object_hook() → dict

:return:{col:object_hook}

factor_schema*: Type[Mixin] = None*

define the schema for persist,its columns should be same as indicators in transformer or accumulator

move_on(*to_timestamp: Optional[Union[Timestamp, str]] = None*, *timeout: int = 20*) → object

using continual fetching data in realtime 1)get the data happened before to_timestamp,if not set,get all the data which means to now 2)if computing_window set,the data_df would be cut for saving memory

Parameters::

- **to_timestamp** –
- **timeout** (*int*) –

Returns::

Return type::

name = *None*

name of the service, default name of class if not set manually

on_data_changed(*data: DataFrame*)

overwrite it for computing after data added

Parameters:: **data** –

on_data_loaded(*data: DataFrame*)

Parameters:: **data** –

on_entity_data_changed(*entity*, *added_data: DataFrame*)

overwrite it for computing after entity data added

Parameters::

- **entity** –
- **added_data** –

state_schema

alias of `FactorState`

transformer*: Transformer = None*

transformer for this factor if not passed as __init__ argument

pipe_df*: pd.DataFrame*

中间结果，不持久化 data_df->pipe_df

factor_df*: pd.DataFrame*

计算因子的结果，可持久化,通过对pipe_df的计算得到 pipe_df->factor_df

result_df*: pd.DataFrame*

result_df是用于选股的标准df,通过对factor_df的计算得到 factor_df->result_df

states*: dict*

entity_id:state

zvt.factors.ma

Modules

zvt.factors.ma.domain

---

zvt.factors.ma.ma_factor

---

zvt.factors.ma.ma_stats_factor

---

zvt.factors.ma.top_bottom_factor

---

zvt.factors.ma.domain

Modules

zvt.factors.ma.domain.common

---

zvt.factors.ma.domain.stock_1d_ma_factor

---

zvt.factors.ma.domain.stock_1d_ma_stats_factor

---

zvt.factors.ma.domain.common

Classes

`MaStatsFactorCommon`()

MaStatsFactorCommon

class zvt.factors.ma.domain.common.MaStatsFactorCommon

Bases: `Mixin`

__init__()

entity_id = *Column(None, String(), table=None)*

entity id

classmethod get_providers() → List[str]

providers of the schema defined by cls

Returns:: providers

classmethod get_storages(*provider: Optional[str] = None*)

get the storages info

Parameters:: **provider** – provider

Returns:: storages

id = *Column(None, String(), table=None, primary_key=True, nullable=False)*

id

classmethod query_data(*provider_index: int = 0*, *ids: Optional[List[str]] = None*, *entity_ids: Optional[List[str]] = None*, *entity_id: Optional[str] = None*, *codes: Optional[List[str]] = None*, *code: Optional[str] = None*, *level: Optional[Union[IntervalLevel, str]] = None*, *provider: Optional[str] = None*, *columns: Optional[List] = None*, *col_label: Optional[dict] = None*, *return_type: str = 'df'*, *start_timestamp: Optional[Union[Timestamp, str]] = None*, *end_timestamp: Optional[Union[Timestamp, str]] = None*, *filters: Optional[List] = None*, *session: Optional[Session] = None*, *order=None*, *limit: Optional[int] = None*, *distinct=None*, *index: Optional[Union[str, list]] = None*, *drop_index_col=False*, *time_field: str = 'timestamp'*)

query data by the arguments

Parameters::

- **provider_index** –
- **data_schema** –
- **ids** –
- **entity_ids** –
- **entity_id** –
- **codes** –
- **code** –
- **level** –
- **provider** –
- **columns** –
- **col_label** – dict with key(column), value(label
- **return_type** – df, domain or dict. default is df
- **start_timestamp** –
- **end_timestamp** –
- **filters** –
- **session** –
- **order** –
- **limit** –
- **index** – index field name, str for single index, str list for multiple index
- **drop_index_col** – whether drop the col if it's in index, default False
- **time_field** –

Returns:: results basing on return_type.

classmethod record_data(*provider_index: int = 0*, *provider: Optional[str] = None*, *force_update=None*, *sleeping_time=None*, *exchanges=None*, *entity_id=None*, *entity_ids=None*, *code=None*, *codes=None*, *real_time=None*, *fix_duplicate_way=None*, *start_timestamp=None*, *end_timestamp=None*, *one_day_trading_minutes=None*, ***kwargs*)

record data by the arguments

Parameters::

- **entity_id** –
- **provider_index** –
- **provider** –
- **force_update** –
- **sleeping_time** –
- **exchanges** –
- **entity_ids** –
- **code** –
- **codes** –
- **real_time** –
- **fix_duplicate_way** –
- **start_timestamp** –
- **end_timestamp** –
- **one_day_trading_minutes** –
- **kwargs** –

Returns::

classmethod register_provider(*provider*)

register the provider to the schema defined by cls

Parameters:: **provider** –

classmethod register_recorder_cls(*provider*, *recorder_cls*)

register the recorder for the schema

Parameters::

- **provider** –
- **recorder_cls** –

timestamp = *Column(None, DateTime(), table=None)*

the meaning could be different for different case,most time it means 'happen time'

zvt.factors.ma.domain.stock_1d_ma_factor

Classes

`Stock1dMaFactor`(**kwargs)

Stock1dMaFactor

class
zvt.factors.ma.domain.stock_1d_ma_factor.Stock1dMaFactor(**kwargs*)

Bases: **Base**, **Mixin**

__init__(**kwargs*)

A simple constructor that allows initialization from kwargs.

Sets attributes on the constructed instance using the names and values in `kwargs`.

Only keys that are present as attributes of the instance's class are allowed. These could be, for example, any mapped columns or relationships.

entity_id

entity id

classmethod get_providers() → List[str]

providers of the schema defined by cls

Returns:: providers

classmethod get_storages(*provider: Optional[str] = None*)

get the storages info

Parameters:: **provider** – provider
Returns:: storages

id

id

classmethod query_data(*provider_index: int = 0*, *ids: Optional[List[str]] = None*, *entity_ids: Optional[List[str]] = None*, *entity_id: Optional[str] = None*, *codes: Optional[List[str]] = None*, *code: Optional[str] = None*,

level: Optional[Union[IntervalLevel, str]] = None, *provider: Optional[str] = None*, *columns: Optional[List] = None*, *col_label: Optional[dict] = None*, *return_type: str = 'df'*, *start_timestamp: Optional[Union[Timestamp, str]] = None*, *end_timestamp: Optional[Union[Timestamp, str]] = None*, *filters: Optional[List] = None*, *session: Optional[Session] = None*, *order=None*, *limit: Optional[int] = None*, *distinct=None*, *index: Optional[Union[str, list]] = None*, *drop_index_col=False*, *time_field: str = 'timestamp'*)

query data by the arguments

Parameters::

- **provider_index** –
- **data_schema** –
- **ids** –
- **entity_ids** –
- **entity_id** –
- **codes** –
- **code** –
- **level** –
- **provider** –
- **columns** –
- **col_label** – dict with key(column), value(label
- **return_type** – df, domain or dict. default is df
- **start_timestamp** –
- **end_timestamp** –
- **filters** –
- **session** –
- **order** –
- **limit** –
- **index** – index field name, str for single index, str list for multiple index
- **drop_index_col** – whether drop the col if it's in index, default False
- **time_field** –

Returns:: results basing on return_type.

classmethod record_data(*provider_index: int = 0*, *provider: Optional[str] = None*, *force_update=None*, *sleeping_time=None*, *exchanges=None*, *entity_id=None*, *entity_ids=None*, *code=None*, *codes=None*,

real_time=None, *fix_duplicate_way=None*, *start_timestamp=None*, *end_timestamp=None*, *one_day_trading_minutes=None*, ***kwargs*)

record data by the arguments

Parameters::

- **entity_id** –
- **provider_index** –
- **provider** –
- **force_update** –
- **sleeping_time** –
- **exchanges** –
- **entity_ids** –
- **code** –
- **codes** –
- **real_time** –
- **fix_duplicate_way** –
- **start_timestamp** –
- **end_timestamp** –
- **one_day_trading_minutes** –
- **kwargs** –

Returns::

classmethod register_provider(*provider*)

register the provider to the schema defined by cls

Parameters:: **provider** –

classmethod register_recorder_cls(*provider*, *recorder_cls*)

register the recorder for the schema

Parameters::

- **provider** –
- **recorder_cls** –

timestamp

the meaning could be different for different case,most time it means 'happen time'

zvt.factors.ma.domain.stock_1d_ma_stats_factor

Classes

`Stock1dMaStatsFactor`(**kwargs)

Stock1dMaStatsFactor

class zvt.factors.ma.domain.stock_1d_ma_stats_factor.Stock1dMaStatsFactor(***kwargs*)

Bases: **Base**, **MaStatsFactorCommon**

__init__(***kwargs*)

A simple constructor that allows initialization from kwargs.

Sets attributes on the constructed instance using the names and values in `kwargs`.

Only keys that are present as attributes of the instance's class are allowed. These could be, for example, any mapped columns or relationships.

area

close

count

distance

entity_id

entity id

classmethod get_providers() → List[str]

providers of the schema defined by cls

Returns:: providers

classmethod get_storages(*provider: Optional[str] = None*)

get the storages info

Parameters:: **provider** – provider

Returns:: storages

high

id

id

live

low

ma10

ma120

ma144

ma250

ma34

ma5

ma55

ma89

open

classmethod query_data(*provider_index: int = 0*, *ids: Optional[List[str]] = None*, *entity_ids: Optional[List[str]] = None*, *entity_id: Optional[str] = None*, *codes: Optional[List[str]] = None*, *code: Optional[str] = None*, *level: Optional[Union[IntervalLevel, str]] = None*, *provider: Optional[str] = None*, *columns: Optional[List] = None*, *col_label: Optional[dict] = None*, *return_type: str = 'df'*, *start_timestamp: Optional[Union[Timestamp, str]] = None*, *end_timestamp: Optional[Union[Timestamp, str]] = None*, *filters: Optional[List] = None*, *session: Optional[Session] = None*, *order=None*, *limit: Optional[int] = None*, *distinct=None*, *index: Optional[Union[str, list]] = None*, *drop_index_col=False*, *time_field: str = 'timestamp'*)

query data by the arguments

Parameters::

- **provider_index** –
- **data_schema** –
- **ids** –
- **entity_ids** –
- **entity_id** –
- **codes** –
- **code** –
- **level** –
- **provider** –
- **columns** –
- **col_label** – dict with key(column), value(label
- **return_type** – df, domain or dict. default is df
- **start_timestamp** –
- **end_timestamp** –
- **filters** –
- **session** –
- **order** –
- **limit** –
- **index** – index field name, str for single index, str list for multiple index
- **drop_index_col** – whether drop the col if it's in index, default False
- **time_field** –

Returns:: results basing on return_type.

classmethod record_data(*provider_index: int = 0*, *provider: Optional[str] = None*, *force_update=None*, *sleeping_time=None*, *exchanges=None*, *entity_id=None*, *entity_ids=None*, *code=None*, *codes=None*, *real_time=None*, *fix_duplicate_way=None*, *start_timestamp=None*, *end_timestamp=None*, *one_day_trading_minutes=None*, ***kwargs*)

record data by the arguments

Parameters::

- **entity_id** –
- **provider_index** –
- **provider** –
- **force_update** –
- **sleeping_time** –
- **exchanges** –
- **entity_ids** –
- **code** –
- **codes** –
- **real_time** –
- **fix_duplicate_way** –
- **start_timestamp** –
- **end_timestamp** –
- **one_day_trading_minutes** –
- **kwargs** –

Returns::

classmethod register_provider(*provider*)

register the provider to the schema defined by cls

Parameters:: **provider** –

classmethod register_recorder_cls(*provider*, *recorder_cls*)

register the recorder for the schema

Parameters::

- **provider** –
- **recorder_cls** –

timestamp

the meaning could be different for different case,most time it means 'happen time'

turnover

vol_ma30

zvt.factors.ma.ma_factor

Functions

`get_ma_factor_schema`(entity_type[, level])

Classes

`CrossMaFactor`(entity_schema, provider, ...)

`CrossMaVolumeFactor`(entity_schema, provider, ...)

`MaFactor`(entity_schema, provider, ...[, windows])

`VolumeUpMaFactor`(entity_schema, provider, ...)

---

zvt.factors.ma.ma_factor.get_ma_factor_schema

zvt.factors.ma.ma_factor.get_ma_factor_schema(*entity_type: str*, *level: Union[IntervalLevel, str] = IntervalLevel.LEVEL_1DAY*)

CrossMaFactor

class zvt.factors.ma.ma_factor.CrossMaFactor(*entity_schema: ~typing.Type[~zvt.contract.schema.TradableEntity] = <class 'zvt.domain.meta.stock_meta.Stock'>, provider: ~typing.Optional[str] = None, entity_provider: ~typing.Optional[str] = None, entity_ids: ~typing.Optional[~typing.List[str]] = None, exchanges: ~typing.Optional[~typing.List[str]] = None, codes: ~typing.Optional[~typing.List[str]] = None, start_timestamp: ~typing.Optional[~typing.Union[~pandas._libs.tslibs.timestamps.Timestamp, str]] = None, end_timestamp: ~typing.Optional[~typing.Union[~pandas._libs.tslibs.timestamps.Timestamp, str]] = None, columns: ~typing.Optional[~typing.List] = None, filters: ~typing.Optional[~typing.List] = None, order: ~typing.Optional[object] = None, limit: ~typing.Optional[int] = None, level: ~typing.Union[str, ~zvt.contract.IntervalLevel] = IntervalLevel.LEVEL_1DAY, category_field: str = 'entity_id', time_field: str = 'timestamp', keep_window: ~typing.Optional[int] = None, keep_all_timestamp: bool = False, fill_method: str = 'ffill', effective_number: ~typing.Optional[int] = None, need_persist: bool = False, only_compute_factor: bool = False, factor_name: ~typing.Optional[str] = None, clear_state: bool = False, only_load_factor: bool = False, adjust_type: ~typing.Optional[~typing.Union[~zvt.contract.AdjustType, str]] = None, windows=None*)

Bases: **MaFactor**

__init__(*entity_schema: ~typing.Type[~zvt.contract.schema.TradableEntity] = <class 'zvt.domain.meta.stock_meta.Stock'>, provider: ~typing.Optional[str] = None, entity_provider: ~typing.Optional[str] = None, entity_ids: ~typing.Optional[~typing.List[str]] = None, exchanges: ~typing.Optional[~typing.List[str]] = None, codes: ~typing.Optional[~typing.List[str]] = None, start_timestamp: ~typing.Optional[~typing.Union[~pandas._libs.tslibs.timestamps.Timestamp, str]] = None, end_timestamp:*

~typing.Optional[~typing.Union[~pandas._libs.tslibs.timestamps.Timestamp, str]] = None, columns: ~typing.Optional[~typing.List] = None, filters: ~typing.Optional[~typing.List] = None, order: ~typing.Optional[object] = None, limit: ~typing.Optional[int] = None, level: ~typing.Union[str, ~zvt.contract.IntervalLevel] = IntervalLevel.LEVEL_1DAY, category_field: str = 'entity_id', time_field: str = 'timestamp', keep_window: ~typing.Optional[int] = None, keep_all_timestamp: bool = False, fill_method: str = 'ffill', effective_number: ~typing.Optional[int] = None, need_persist: bool = False, only_compute_factor: bool = False, factor_name: ~typing.Optional[str] = None, clear_state: bool = False, only_load_factor: bool = False, adjust_type: ~typing.Optional[~typing.Union[~zvt.contract.AdjustType, str]] = None, windows=None) → None

Parameters::

- **keep_all_timestamp** –
- **fill_method** –
- **effective_number** –
- **transformer** –
- **accumulator** –
- **need_persist** – whether persist factor
- **only_compute_factor** – only compute factor nor result
- **factor_name** –
- **clear_state** –
- **only_load_factor** – only load factor and compute result

accumulator*: Accumulator = None*

accumulator for this factor if not passed as __init__ argument

clear_state_data(*entity_id=None*)

clear state of the entity

Parameters:: **entity_id** – entity id

decode_state(*state: str*)

decode state

Parameters:: **state** –
Returns::

encode_state(*state: object*)

encode state

Parameters:: **state** –
Returns::

factor_col_map_object_hook() → dict

:return:{col:object_hook}

factor_schema*: Type[Mixin] = None*

define the schema for persist,its columns should be same as indicators in transformer or accumulator

move_on(*to_timestamp: Optional[Union[Timestamp, str]] = None*, *timeout: int = 20*) → object

using continual fetching data in realtime 1)get the data happened before to_timestamp,if not set,get all the data which means to now 2)if computing_window set,the data_df would be cut for saving memory

Parameters::
- **to_timestamp** –
- **timeout** (*int*) –

Returns::
Return type::

name = *None*

name of the service, default name of class if not set manually

on_data_changed(*data: DataFrame*)

overwrite it for computing after data added

Parameters:: **data** –

on_data_loaded(*data: DataFrame*)

Parameters:: **data** –

on_entity_data_changed(*entity*, *added_data: DataFrame*)

overwrite it for computing after entity data added

Parameters::

- **entity** –
- **added_data** –

state_schema

alias of `FactorState`

transformer*: Transformer = None*

transformer for this factor if not passed as __init__ argument

pipe_df*: pd.DataFrame*

中间结果，不持久化 data_df->pipe_df

factor_df*: pd.DataFrame*

计算因子的结果，可持久化,通过对pipe_df的计算得到 pipe_df->factor_df

result_df*: pd.DataFrame*

result_df是用于选股的标准df,通过对factor_df的计算得到 factor_df->result_df

states*: dict*

entity_id:state

CrossMaVolumeFactor

class zvt.factors.ma.ma_factor.CrossMaVolumeFactor(*entity_schema: ~typing.Type[~zvt.contract.schema.TradableEntity] = <class 'zvt.domain.meta.stock_meta.Stock'>, provider: ~typing.Optional[str] = None, entity_provider: ~typing.Optional[str] = None, entity_ids: ~typing.Optional[~typing.List[str]] = None, exchanges: ~typing.Optional[~typing.List[str]] = None, codes: ~typing.Optional[~typing.List[str]] = None, start_timestamp: ~typing.Optional[~typing.Union[~pandas._libs.tslibs.timestamps.Timestamp, str]] = None, end_timestamp: ~typing.Optional[~typing.Union[~pandas._libs.tslibs.timestamps.Timestamp, str]] = None, columns: ~typing.Optional[~typing.List] = None, filters: ~typing.Optional[~typing.List] = None, order: ~typing.Optional[object] = None, limit: ~typing.Optional[int] = None, level: ~typing.Union[str, ~zvt.contract.IntervalLevel] = IntervalLevel.LEVEL_1DAY, category_field: str = 'entity_id', time_field: str = 'timestamp', keep_window: ~typing.Optional[int] = None, keep_all_timestamp: bool = False, fill_method: str = 'ffill', effective_number: ~typing.Optional[int] = None, accumulator: ~typing.Optional[~zvt.contract.factor.Accumulator] = None, need_persist: bool = False, only_compute_factor: bool = False, factor_name: ~typing.Optional[str] = None, clear_state: bool = False, only_load_factor: bool = False, adjust_type: ~typing.Optional[~typing.Union[~zvt.contract.AdjustType, str]] = None, windows=[5, 10, 250], vol_windows=None, turnover_threshold=300000000, turnover_rate_threshold=0.02, up_intervals=40, over_mode='and'*)

Bases: `VolumeUpMaFactor`

__init__(*entity_schema: ~typing.Type[~zvt.contract.schema.TradableEntity] = <class 'zvt.domain.meta.stock_meta.Stock'>, provider: ~typing.Optional[str] = None, entity_provider: ~typing.Optional[str] = None, entity_ids: ~typing.Optional[~typing.List[str]] = None, exchanges: ~typing.Optional[~typing.List[str]] = None, codes: ~typing.Optional[~typing.List[str]] = None, start_timestamp:*

~typing.Optional[~typing.Union[~pandas._libs.tslibs.timestamps.Timestamp, str]] = None, end_timestamp: ~typing.Optional[~typing.Union[~pandas._libs.tslibs.timestamps.Timestamp, str]] = None, columns: ~typing.Optional[~typing.List] = None, filters: ~typing.Optional[~typing.List] = None, order: ~typing.Optional[object] = None, limit: ~typing.Optional[int] = None, level: ~typing.Union[str, ~zvt.contract.IntervalLevel] = IntervalLevel.LEVEL_1DAY, category_field: str = 'entity_id', time_field: str = 'timestamp', keep_window: ~typing.Optional[int] = None, keep_all_timestamp: bool = False, fill_method: str = 'ffill', effective_number: ~typing.Optional[int] = None, accumulator: ~typing.Optional[~zvt.contract.factor.Accumulator] = None, need_persist: bool = False, only_compute_factor: bool = False, factor_name: ~typing.Optional[str] = None, clear_state: bool = False, only_load_factor: bool = False, adjust_type: ~typing.Optional[~typing.Union[~zvt.contract.AdjustType, str]] = None, windows=[5, 10, 250], vol_windows=None, turnover_threshold=300000000, turnover_rate_threshold=0.02, up_intervals=40, over_mode='and') → None

Parameters::

- **keep_all_timestamp** –
- **fill_method** –
- **effective_number** –
- **transformer** –
- **accumulator** –
- **need_persist** – whether persist factor
- **only_compute_factor** – only compute factor nor result
- **factor_name** –
- **clear_state** –
- **only_load_factor** – only load factor and compute result

accumulator*: Accumulator = None*

accumulator for this factor if not passed as __init__ argument

clear_state_data(*entity_id=None*)

clear state of the entity

Parameters:: **entity_id** – entity id

decode_state(*state: str*)

decode state

Parameters:: **state** –

Returns::

encode_state(*state: object*)

encode state

Parameters:: **state** –

Returns::

factor_col_map_object_hook() → dict

:return:{col:object_hook}

factor_schema*: Type[Mixin] = None*

define the schema for persist,its columns should be same as indicators in transformer or accumulator

move_on(*to_timestamp: Optional[Union[Timestamp, str]] = None*, *timeout: int = 20*) → object

using continual fetching data in realtime 1)get the data happened before to_timestamp,if not set,get all the data which means to now 2)if computing_window set,the data_df would be cut for saving memory

Parameters::

- **to_timestamp** –
- **timeout** (*int*) –

Returns::

Return type::

name = *None*

name of the service, default name of class if not set manually

on_data_changed(*data: DataFrame*)

overwrite it for computing after data added

Parameters:: **data** –

on_data_loaded(*data: DataFrame*)

Parameters:: **data** –

on_entity_data_changed(*entity*, *added_data: DataFrame*)

overwrite it for computing after entity data added

Parameters::
- **entity** –
- **added_data** –

state_schema

alias of `FactorState`

transformer*: Transformer = None*

transformer for this factor if not passed as __init__ argument

pipe_df*: pd.DataFrame*

中间结果，不持久化 data_df->pipe_df

factor_df*: pd.DataFrame*

计算因子的结果，可持久化,通过对pipe_df的计算得到 pipe_df->factor_df

result_df*: pd.DataFrame*

result_df是用于选股的标准df,通过对factor_df的计算得到 factor_df->result_df

states*: dict*

entity_id:state

MaFactor

class zvt.factors.ma.ma_factor.MaFactor(*entity_schema: ~typing.Type[~zvt.contract.schema.TradableEntity] = <class 'zvt.domain.meta.stock_meta.Stock'>, provider: ~typing.Optional[str] = None, entity_provider: ~typing.Optional[str] = None, entity_ids: ~typing.Optional[~typing.List[str]] = None, exchanges: ~typing.Optional[~typing.List[str]] = None, codes: ~typing.Optional[~typing.List[str]] = None, start_timestamp: ~typing.Optional[~typing.Union[~pandas._libs.tslibs.timestamps.Timestamp, str]] = None, end_timestamp: ~typing.Optional[~typing.Union[~pandas._libs.tslibs.timestamps.Timestamp, str]] = None, columns: ~typing.Optional[~typing.List] = None, filters: ~typing.Optional[~typing.List] = None, order: ~typing.Optional[object] = None, limit: ~typing.Optional[int] = None, level: ~typing.Union[str, ~zvt.contract.IntervalLevel] = IntervalLevel.LEVEL_1DAY, category_field: str = 'entity_id', time_field: str = 'timestamp', keep_window: ~typing.Optional[int] = None, keep_all_timestamp: bool = False, fill_method: str = 'ffill', effective_number: ~typing.Optional[int] = None, need_persist: bool = False, only_compute_factor: bool = False, factor_name: ~typing.Optional[str] = None, clear_state: bool = False, only_load_factor: bool = False, adjust_type: ~typing.Optional[~typing.Union[~zvt.contract.AdjustType, str]] = None, windows=None*)

Bases: **TechnicalFactor**

__init__(*entity_schema: ~typing.Type[~zvt.contract.schema.TradableEntity] = <class 'zvt.domain.meta.stock_meta.Stock'>, provider: ~typing.Optional[str] = None, entity_provider: ~typing.Optional[str] = None, entity_ids: ~typing.Optional[~typing.List[str]] = None, exchanges: ~typing.Optional[~typing.List[str]] = None, codes: ~typing.Optional[~typing.List[str]] = None, start_timestamp: ~typing.Optional[~typing.Union[~pandas._libs.tslibs.timestamps.Timestamp, str]] = None, end_timestamp:*

~typing.Optional[~typing.Union[~pandas._libs.tslibs.timestamps.Timestamp, str]] = None, columns: ~typing.Optional[~typing.List] = None, filters: ~typing.Optional[~typing.List] = None, order: ~typing.Optional[object] = None, limit: ~typing.Optional[int] = None, level: ~typing.Union[str, ~zvt.contract.IntervalLevel] = IntervalLevel.LEVEL_1DAY, category_field: str = 'entity_id', time_field: str = 'timestamp', keep_window: ~typing.Optional[int] = None, keep_all_timestamp: bool = False, fill_method: str = 'ffill', effective_number: ~typing.Optional[int] = None, need_persist: bool = False, only_compute_factor: bool = False, factor_name: ~typing.Optional[str] = None, clear_state: bool = False, only_load_factor: bool = False, adjust_type: ~typing.Optional[~typing.Union[~zvt.contract.AdjustType, str]] = None, windows=None) → None

Parameters::

- **keep_all_timestamp** –
- **fill_method** –
- **effective_number** –
- **transformer** –
- **accumulator** –
- **need_persist** – whether persist factor
- **only_compute_factor** – only compute factor nor result
- **factor_name** –
- **clear_state** –
- **only_load_factor** – only load factor and compute result

factor_schema*: Type[Mixin] = None*

define the schema for persist,its columns should be same as indicators in transformer or accumulator

accumulator*: Accumulator = None*

accumulator for this factor if not passed as __init__ argument

clear_state_data(*entity_id=None*)

clear state of the entity

Parameters:: **entity_id** – entity id

decode_state(*state: str*)

decode state

Parameters:: **state** –

Returns::

encode_state(*state: object*)

encode state

Parameters:: **state** –

Returns::

factor_col_map_object_hook() → dict

:return:{col:object_hook}

move_on(*to_timestamp: Optional[Union[Timestamp, str]] = None*, *timeout: int = 20*) → object

using continual fetching data in realtime 1)get the data happened before to_timestamp,if not set,get all the data which means to now 2)if computing_window set,the data_df would be cut for saving memory

Parameters::

- **to_timestamp** –
- **timeout** (*int*) –

Returns::

Return type::

name = *None*

name of the service, default name of class if not set manually

on_data_changed(*data: DataFrame*)

overwrite it for computing after data added

Parameters:: **data** –

on_data_loaded(*data: DataFrame*)

Parameters:: **data** –

on_entity_data_changed(*entity*, *added_data: DataFrame*)

overwrite it for computing after entity data added

Parameters::

- **entity** –
- **added_data** –

state_schema

alias of `FactorState`

transformer*: Transformer = None*

transformer for this factor if not passed as __init__ argument

pipe_df*: pd.DataFrame*

中间结果，不持久化 data_df->pipe_df

factor_df*: pd.DataFrame*

计算因子的结果，可持久化,通过对pipe_df的计算得到 pipe_df->factor_df

result_df*: pd.DataFrame*

result_df是用于选股的标准df,通过对factor_df的计算得到 factor_df->result_df

states*: dict*

entity_id:state

VolumeUpMaFactor

class zvt.factors.ma.ma_factor.VolumeUpMaFactor(*entity_schema: ~typing.Type[~zvt.contract.schema.TradableEntity] = <class 'zvt.domain.meta.stock_meta.Stock'>*, *provider: ~typing.Optional[str] = None*, *entity_provider: ~typing.Optional[str] = None*, *entity_ids: ~typing.Optional[~typing.List[str]] = None*, *exchanges: ~typing.Optional[~typing.List[str]] = None*, *codes: ~typing.Optional[~typing.List[str]] = None*, *start_timestamp: ~typing.Optional[~typing.Union[~pandas._libs.tslibs.timestamps.Timestamp, str]] = None*, *end_timestamp: ~typing.Optional[~typing.Union[~pandas._libs.tslibs.timestamps.Timestamp, str]] = None*, *columns: ~typing.Optional[~typing.List] = None*, *filters: ~typing.Optional[~typing.List] = None*, *order: ~typing.Optional[object] = None*, *limit: ~typing.Optional[int] = None*, *level: ~typing.Union[str, ~zvt.contract.IntervalLevel] = IntervalLevel.LEVEL_1DAY*, *category_field: str = 'entity_id'*, *time_field: str = 'timestamp'*, *keep_window: ~typing.Optional[int] = None*, *keep_all_timestamp: bool = False*, *fill_method: str = 'ffill'*, *effective_number: ~typing.Optional[int] = None*, *accumulator: ~typing.Optional[~zvt.contract.factor.Accumulator] = None*, *need_persist: bool = False*, *only_compute_factor: bool = False*, *factor_name: ~typing.Optional[str] = None*, *clear_state: bool = False*, *only_load_factor: bool = False*, *adjust_type: ~typing.Optional[~typing.Union[~zvt.contract.AdjustType, str]] = None*, *windows=None*, *vol_windows=None*, *turnover_threshold=300000000*, *turnover_rate_threshold=0.02*, *up_intervals=40*, *over_mode='and'*)

Bases: `TechnicalFactor`

__init__(*entity_schema: ~typing.Type[~zvt.contract.schema.TradableEntity] = <class 'zvt.domain.meta.stock_meta.Stock'>*, *provider: ~typing.Optional[str] = None*, *entity_provider: ~typing.Optional[str] = None*, *entity_ids: ~typing.Optional[~typing.List[str]] = None*, *exchanges: ~typing.Optional[~typing.List[str]] = None*, *codes: ~typing.Optional[~typing.List[str]] = None*, *start_timestamp:*

~typing.Optional[~typing.Union[~pandas._libs.tslibs.timestamps.Timestamp, str]] = None, end_timestamp: ~typing.Optional[~typing.Union[~pandas._libs.tslibs.timestamps.Timestamp, str]] = None, columns: ~typing.Optional[~typing.List] = None, filters: ~typing.Optional[~typing.List] = None, order: ~typing.Optional[object] = None, limit: ~typing.Optional[int] = None, level: ~typing.Union[str, ~zvt.contract.IntervalLevel] = IntervalLevel.LEVEL_1DAY, category_field: str = 'entity_id', time_field: str = 'timestamp', keep_window: ~typing.Optional[int] = None, keep_all_timestamp: bool = False, fill_method: str = 'ffill', effective_number: ~typing.Optional[int] = None, accumulator: ~typing.Optional[~zvt.contract.factor.Accumulator] = None, need_persist: bool = False, only_compute_factor: bool = False, factor_name: ~typing.Optional[str] = None, clear_state: bool = False, only_load_factor: bool = False, adjust_type: ~typing.Optional[~typing.Union[~zvt.contract.AdjustType, str]] = None, windows=None, vol_windows=None, turnover_threshold=300000000, turnover_rate_threshold=0.02, up_intervals=40, over_mode='and') → None

Parameters::

- **keep_all_timestamp** –
- **fill_method** –
- **effective_number** –
- **transformer** –
- **accumulator** –
- **need_persist** – whether persist factor
- **only_compute_factor** – only compute factor nor result
- **factor_name** –
- **clear_state** –
- **only_load_factor** – only load factor and compute result

accumulator*: Accumulator = None*

accumulator for this factor if not passed as __init__ argument

clear_state_data(*entity_id=None*)

clear state of the entity

Parameters:: **entity_id** – entity id

decode_state(*state: str*)

decode state

Parameters:: **state** –

Returns::

encode_state(*state: object*)

encode state

Parameters:: **state** –

Returns::

factor_col_map_object_hook() → dict

:return:{col:object_hook}

factor_schema*: Type[Mixin] = None*

define the schema for persist,its columns should be same as indicators in transformer or accumulator

move_on(*to_timestamp: Optional[Union[Timestamp, str]] = None*, *timeout: int = 20*) → object

using continual fetching data in realtime 1)get the data happened before to_timestamp,if not set,get all the data which means to now 2)if computing_window set,the data_df would be cut for saving memory

Parameters::

- **to_timestamp** –
- **timeout** (*int*) –

Returns::

Return type::

name = *None*

name of the service, default name of class if not set manually

on_data_changed(*data: DataFrame*)

overwrite it for computing after data added

Parameters:: **data** –

on_data_loaded(*data: DataFrame*)

Parameters:: **data** –

on_entity_data_changed(*entity*, *added_data: DataFrame*)

overwrite it for computing after entity data added

Parameters::
- **entity** –
- **added_data** –

state_schema

alias of `FactorState`

transformer*: Transformer = None*

transformer for this factor if not passed as __init__ argument

pipe_df*: pd.DataFrame*

中间结果，不持久化 data_df->pipe_df

factor_df*: pd.DataFrame*

计算因子的结果，可持久化,通过对pipe_df的计算得到 pipe_df->factor_df

result_df*: pd.DataFrame*

result_df是用于选股的标准df,通过对factor_df的计算得到 factor_df->result_df

states*: dict*

entity_id:state

zvt.factors.ma.ma_stats_factor

Functions

`get_ma_stats_factor_schema`(entity_type[, level])

Classes

`MaStatsAccumulator`([acc_window, windows, ...])

`MaStatsFactor`(entity_schema, provider, ...)

`TFactor`(entity_schema, provider, ...[, ...])

---

zvt.factors.ma.ma_stats_factor.get_ma_stats_factor_schema

zvt.factors.ma.ma_stats_factor.get_ma_stats_factor_schema(*entity_type: str*, *level: Union[IntervalLevel, str] = IntervalLevel.LEVEL_1DAY*)

MaStatsAccumulator

class zvt.factors.ma.ma_stats_factor.MaStatsAccumulator(*acc_window: int = 250*, *windows=None*, *vol_windows=None*)

Bases: **Accumulator**

__init__(*acc_window: int = 250*, *windows=None*, *vol_windows=None*) → None

Parameters:: **acc_window** – the window size of acc for computing,default is 1

acc_one(*entity_id*, *df: ~pandas.core.frame.DataFrame*, *acc_df: ~pandas.core.frame.DataFrame*, *state: dict) -> (<class 'pandas.core.frame.DataFrame'>*, *<class 'dict'>*)

df format:

```
                col1    col2    col3    ...
timestamp
                1.2     0.5     0.3     ...
                1.0     0.7     0.2     ...
```

the new result and state

Parameters::
- **df** – current input df
- **entity_id** – current computing entity_id
- **acc_df** – current result of the entity_id
- **state** – current state of the entity_id

Returns:: new result and state of the entity_id

acc(*input_df: ~pandas.core.frame.DataFrame*, *acc_df: ~pandas.core.frame.DataFrame*, *states: dict) -> (<class 'pandas.core.frame.DataFrame'>*, *<class 'dict'>*)

Parameters::
- **input_df** – new input
- **acc_df** – previous result
- **states** – current states of the entity

Returns:: new result and states

MaStatsFactor

class zvt.factors.ma.ma_stats_factor.MaStatsFactor(*entity_schema: ~typing.Type[~zvt.contract.schema.TradableEntity] = <class 'zvt.domain.meta.stock_meta.Stock'>, provider: ~typing.Optional[str] = None, entity_provider: ~typing.Optional[str] = None, entity_ids: ~typing.Optional[~typing.List[str]] = None, exchanges: ~typing.Optional[~typing.List[str]] = None, codes: ~typing.Optional[~typing.List[str]] = None, start_timestamp: ~typing.Optional[~typing.Union[~pandas._libs.tslibs.timestamps.Timestamp, str]] = None, end_timestamp: ~typing.Optional[~typing.Union[~pandas._libs.tslibs.timestamps.Timestamp, str]] = None, filters: ~typing.Optional[~typing.List] = None, order: ~typing.Optional[object] = None, limit: ~typing.Optional[int] = None, level: ~typing.Union[str, ~zvt.contract.IntervalLevel] = IntervalLevel.LEVEL_1DAY, category_field: str = 'entity_id', time_field: str = 'timestamp', keep_window: ~typing.Optional[int] = None, keep_all_timestamp: bool = False, fill_method: str = 'ffill', effective_number: ~typing.Optional[int] = None, need_persist: bool = True, only_compute_factor: bool = False, factor_name: ~typing.Optional[str] = None, clear_state: bool = False, only_load_factor: bool = False, adjust_type: ~typing.Optional[~typing.Union[~zvt.contract.AdjustType, str]] = None, windows=None, vol_windows=None*)

Bases: **TechnicalFactor**

__init__(*entity_schema: ~typing.Type[~zvt.contract.schema.TradableEntity] = <class 'zvt.domain.meta.stock_meta.Stock'>, provider: ~typing.Optional[str] = None, entity_provider: ~typing.Optional[str] = None, entity_ids: ~typing.Optional[~typing.List[str]] = None, exchanges: ~typing.Optional[~typing.List[str]] = None, codes: ~typing.Optional[~typing.List[str]] = None, start_timestamp: ~typing.Optional[~typing.Union[~pandas._libs.tslibs.timestamps.Timestamp, str]] = None, end_timestamp: ~typing.Optional[~typing.Union[~pandas._libs.tslibs.timestamps.Timesta*

mp, str]] = None, filters: ~typing.Optional[~typing.List] = None, order: ~typing.Optional[object] = None, limit: ~typing.Optional[int] = None, level: ~typing.Union[str, ~zvt.contract.IntervalLevel] = IntervalLevel.LEVEL_1DAY, category_field: str = 'entity_id', time_field: str = 'timestamp', keep_window: ~typing.Optional[int] = None, keep_all_timestamp: bool = False, fill_method: str = 'ffill', effective_number: ~typing.Optional[int] = None, need_persist: bool = True, only_compute_factor: bool = False, factor_name: ~typing.Optional[str] = None, clear_state: bool = False, only_load_factor: bool = False, adjust_type: ~typing.Optional[~typing.Union[~zvt.contract.AdjustType, str]] = None, windows=None, vol_windows=None) → None

Parameters::

- **keep_all_timestamp** –
- **fill_method** –
- **effective_number** –
- **transformer** –
- **accumulator** –
- **need_persist** – whether persist factor
- **only_compute_factor** – only compute factor nor result
- **factor_name** –
- **clear_state** –
- **only_load_factor** – only load factor and compute result

factor_schema*: Type[Mixin] = None*

define the schema for persist,its columns should be same as indicators in transformer or accumulator

accumulator*: Accumulator = None*

accumulator for this factor if not passed as __init__ argument

clear_state_data(*entity_id=None*)

clear state of the entity

Parameters:: **entity_id** – entity id

decode_state(*state: str*)

decode state

Parameters:: **state** –
Returns::

encode_state(*state: object*)

encode state

Parameters:: **state** –
Returns::

factor_col_map_object_hook() → dict

:return:{col:object_hook}

move_on(*to_timestamp: Optional[Union[Timestamp, str]] = None*, *timeout: int = 20*) → object

using continual fetching data in realtime 1)get the data happened before to_timestamp,if not set,get all the data which means to now 2)if computing_window set,the data_df would be cut for saving memory

Parameters::
- **to_timestamp** –
- **timeout** (*int*) –

Returns::
Return type::

name = *None*

name of the service, default name of class if not set manually

on_data_changed(*data: DataFrame*)

overwrite it for computing after data added

Parameters:: **data** –

on_data_loaded(*data: DataFrame*)

Parameters:: **data** –

on_entity_data_changed(*entity*, *added_data: DataFrame*)

overwrite it for computing after entity data added

Parameters::

- **entity** –
- **added_data** –

state_schema

alias of `FactorState`

transformer*: Transformer = None*

transformer for this factor if not passed as __init__ argument

pipe_df*: pd.DataFrame*

中间结果，不持久化 data_df->pipe_df

factor_df*: pd.DataFrame*

计算因子的结果，可持久化,通过对pipe_df的计算得到 pipe_df->factor_df

result_df*: pd.DataFrame*

result_df是用于选股的标准df,通过对factor_df的计算得到 factor_df->result_df

states*: dict*

entity_id:state

TFactor

class zvt.factors.ma.ma_stats_factor.TFactor(*entity_schema: ~typing.Type[~zvt.contract.schema.TradableEntity] = <class 'zvt.domain.meta.stock_meta.Stock'>, provider: ~typing.Optional[str] = None, entity_provider: ~typing.Optional[str] = None, entity_ids: ~typing.Optional[~typing.List[str]] = None, exchanges: ~typing.Optional[~typing.List[str]] = None, codes: ~typing.Optional[~typing.List[str]] = None, start_timestamp: ~typing.Optional[~typing.Union[~pandas._libs.tslibs.timestamps.Timestamp, str]] = None, end_timestamp: ~typing.Optional[~typing.Union[~pandas._libs.tslibs.timestamps.Timestamp, str]] = None, filters: ~typing.Optional[~typing.List] = None, order: ~typing.Optional[object] = None, limit: ~typing.Optional[int] = None, level: ~typing.Union[str, ~zvt.contract.IntervalLevel] = IntervalLevel.LEVEL_1DAY, category_field: str = 'entity_id', time_field: str = 'timestamp', keep_window: ~typing.Optional[int] = None, keep_all_timestamp: bool = False, fill_method: str = 'ffill', effective_number: ~typing.Optional[int] = None, need_persist: bool = True, only_compute_factor: bool = False, factor_name: ~typing.Optional[str] = None, clear_state: bool = False, only_load_factor: bool = True, adjust_type: ~typing.Optional[~typing.Union[~zvt.contract.AdjustType, str]] = None, windows=None, vol_windows=None*)

Bases: **MaStatsFactor**

__init__(*entity_schema: ~typing.Type[~zvt.contract.schema.TradableEntity] = <class 'zvt.domain.meta.stock_meta.Stock'>, provider: ~typing.Optional[str] = None, entity_provider: ~typing.Optional[str] = None, entity_ids: ~typing.Optional[~typing.List[str]] = None, exchanges: ~typing.Optional[~typing.List[str]] = None, codes: ~typing.Optional[~typing.List[str]] = None, start_timestamp: ~typing.Optional[~typing.Union[~pandas._libs.tslibs.timestamps.Timestamp, str]] = None, end_timestamp: ~typing.Optional[~typing.Union[~pandas._libs.tslibs.timestamps.Timesta*

mp, *str]] = None*, *filters: ~typing.Optional[~typing.List] = None*, *order: ~typing.Optional[object] = None*, *limit: ~typing.Optional[int] = None*, *level: ~typing.Union[str*, *~zvt.contract.IntervalLevel] = IntervalLevel.LEVEL_1DAY*, *category_field: str = 'entity_id'*, *time_field: str = 'timestamp'*, *keep_window: ~typing.Optional[int] = None*, *keep_all_timestamp: bool = False*, *fill_method: str = 'ffill'*, *effective_number: ~typing.Optional[int] = None*, *need_persist: bool = True*, *only_compute_factor: bool = False*, *factor_name: ~typing.Optional[str] = None*, *clear_state: bool = False*, *only_load_factor: bool = True*, *adjust_type: ~typing.Optional[~typing.Union[~zvt.contract.AdjustType*, *str]] = None*, *windows=None*, *vol_windows=None*) → None

Parameters::

- **keep_all_timestamp** –
- **fill_method** –
- **effective_number** –
- **transformer** –
- **accumulator** –
- **need_persist** – whether persist factor
- **only_compute_factor** – only compute factor nor result
- **factor_name** –
- **clear_state** –
- **only_load_factor** – only load factor and compute result

accumulator*: Accumulator = None*

accumulator for this factor if not passed as __init__ argument

clear_state_data(*entity_id=None*)

clear state of the entity

Parameters:: **entity_id** – entity id

decode_state(*state: str*)

decode state

Parameters:: **state** –

Returns::

encode_state(*state: object*)

encode state

Parameters:: **state** –

Returns::

factor_col_map_object_hook() → dict

:return:{col:object_hook}

factor_schema*: Type[Mixin] = None*

define the schema for persist,its columns should be same as indicators in transformer or accumulator

move_on(*to_timestamp: Optional[Union[Timestamp, str]] = None*, *timeout: int = 20*) → object

using continual fetching data in realtime 1)get the data happened before to_timestamp,if not set,get all the data which means to now 2)if computing_window set,the data_df would be cut for saving memory

Parameters::

- **to_timestamp** –
- **timeout** (*int*) –

Returns::

Return type::

name = *None*

name of the service, default name of class if not set manually

on_data_changed(*data: DataFrame*)

overwrite it for computing after data added

Parameters:: **data** –

on_data_loaded(*data: DataFrame*)

Parameters:: **data** –

on_entity_data_changed(*entity*, *added_data: DataFrame*)

overwrite it for computing after entity data added

Parameters::

- **entity** –
- **added_data** –

state_schema

alias of `FactorState`

transformer*: Transformer = None*

transformer for this factor if not passed as __init__ argument

pipe_df*: pd.DataFrame*

中间结果，不持久化 data_df->pipe_df

factor_df*: pd.DataFrame*

计算因子的结果，可持久化,通过对pipe_df的计算得到 pipe_df->factor_df

result_df*: pd.DataFrame*

result_df是用于选股的标准df,通过对factor_df的计算得到 factor_df->result_df

states*: dict*

entity_id:state

zvt.factors.ma.top_bottom_factor

Classes

`TopBottomFactor`(entity_schema, provider, ...)

`TopBottomTransformer`([window])

TopBottomFactor

class zvt.factors.ma.top_bottom_factor.TopBottomFactor(*entity_schema: ~zvt.contract.schema.TradableEntity = <class 'zvt.domain.meta.stock_meta.Stock'>, provider: ~typing.Optional[str] = None, entity_provider: ~typing.Optional[str] = None, entity_ids: ~typing.Optional[~typing.List[str]] = None, exchanges: ~typing.Optional[~typing.List[str]] = None, codes: ~typing.Optional[~typing.List[str]] = None, start_timestamp: ~typing.Optional[~typing.Union[~pandas._libs.tslibs.timestamps.Timestamp, str]] = None, end_timestamp: ~typing.Optional[~typing.Union[~pandas._libs.tslibs.timestamps.Timestamp, str]] = None, columns: ~typing.List = ['id', 'entity_id', 'timestamp', 'level', 'open', 'close', 'high', 'low'], filters: ~typing.Optional[~typing.List] = None, order: ~typing.Optional[object] = None, limit: ~typing.Optional[int] = None, level: ~typing.Union[str, ~zvt.contract.IntervalLevel] = IntervalLevel.LEVEL_1DAY, category_field: str = 'entity_id', time_field: str = 'timestamp', keep_window: ~typing.Optional[int] = None, keep_all_timestamp: bool = False, fill_method: str = 'ffill', effective_number: ~typing.Optional[int] = None, accumulator: ~typing.Optional[~zvt.contract.factor.Accumulator] = None, need_persist: bool = False, only_compute_factor: bool = False, factor_name: ~typing.Optional[str] = None, clear_state: bool = False, only_load_factor: bool = False, adjust_type: ~typing.Optional[~typing.Union[~zvt.contract.AdjustType, str]] = None, window=30*)

Bases: `TechnicalFactor`

__init__(*entity_schema: ~zvt.contract.schema.TradableEntity = <class 'zvt.domain.meta.stock_meta.Stock'>, provider: ~typing.Optional[str] = None, entity_provider: ~typing.Optional[str] = None, entity_ids: ~typing.Optional[~typing.List[str]] = None, exchanges: ~typing.Optional[~typing.List[str]] = None, codes: ~typing.Optional[~typing.List[str]] = None, start_timestamp: ~typing.Optional[~typing.Union[~pandas._libs.tslibs.timestamps.Timesta*

mp, str]] = None, end_timestamp: ~typing.Optional[~typing.Union[~pandas._libs.tslibs.timestamps.Timestamp, str]] = None, columns: ~typing.List = ['id', 'entity_id', 'timestamp', 'level', 'open', 'close', 'high', 'low'], filters: ~typing.Optional[~typing.List] = None, order: ~typing.Optional[object] = None, limit: ~typing.Optional[int] = None, level: ~typing.Union[str, ~zvt.contract.IntervalLevel] = IntervalLevel.LEVEL_1DAY, category_field: str = 'entity_id', time_field: str = 'timestamp', keep_window: ~typing.Optional[int] = None, keep_all_timestamp: bool = False, fill_method: str = 'ffill', effective_number: ~typing.Optional[int] = None, accumulator: ~typing.Optional[~zvt.contract.factor.Accumulator] = None, need_persist: bool = False, only_compute_factor: bool = False, factor_name: ~typing.Optional[str] = None, clear_state: bool = False, only_load_factor: bool = False, adjust_type: ~typing.Optional[~typing.Union[~zvt.contract.AdjustType, str]] = None, window=30) → None

Parameters::

- **keep_all_timestamp** –
- **fill_method** –
- **effective_number** –
- **transformer** –
- **accumulator** –
- **need_persist** – whether persist factor
- **only_compute_factor** – only compute factor nor result
- **factor_name** –
- **clear_state** –
- **only_load_factor** – only load factor and compute result

accumulator*: Accumulator = None*

accumulator for this factor if not passed as __init__ argument

clear_state_data(*entity_id=None*)

clear state of the entity

Parameters:: **entity_id** – entity id

decode_state(*state: str*)

decode state

Parameters:: **state** –

Returns::

encode_state(*state: object*)

encode state

Parameters:: **state** –

Returns::

factor_col_map_object_hook() → dict

:return:{col:object_hook}

factor_schema*: Type[Mixin] = None*

define the schema for persist,its columns should be same as indicators in transformer or accumulator

move_on(*to_timestamp: Optional[Union[Timestamp, str]] = None*, *timeout: int = 20*) → object

using continual fetching data in realtime 1)get the data happened before to_timestamp,if not set,get all the data which means to now 2)if computing_window set,the data_df would be cut for saving memory

Parameters::

- **to_timestamp** –
- **timeout** (*int*) –

Returns::

Return type::

name = *None*

name of the service, default name of class if not set manually

on_data_changed(*data: DataFrame*)

overwrite it for computing after data added

Parameters:: **data** –

on_data_loaded(*data: DataFrame*)

Parameters:: **data** –

on_entity_data_changed(*entity*, *added_data: DataFrame*)

overwrite it for computing after entity data added

Parameters::

- **entity** –
- **added_data** –

state_schema

alias of `FactorState`

transformer*: Transformer = None*

transformer for this factor if not passed as __init__ argument

pipe_df*: pd.DataFrame*

中间结果，不持久化 data_df->pipe_df

factor_df*: pd.DataFrame*

计算因子的结果，可持久化,通过对pipe_df的计算得到 pipe_df->factor_df

result_df*: pd.DataFrame*

result_df是用于选股的标准df,通过对factor_df的计算得到 factor_df->result_df

states*: dict*

entity_id:state

TopBottomTransformer

class zvt.factors.ma.top_bottom_factor.TopBottomTransformer(*window=20*)

Bases: **Transformer**

__init__(*window=20*) → None

transform(*input_df*) → DataFrame

input_df format:

```
                                col1    col2    col3    ...
entity_id    timestamp
                                1.2     0.5     0.3     ...
                                1.0     0.7     0.2     ...
```

the return result would change the columns and keep the format

Parameters:: **input_df** –

Returns::

transform_one(*entity_id: str*, *df: DataFrame*) → DataFrame

df format:

```
             col1    col2    col3    ...
timestamp
             1.2     0.5     0.3     ...
             1.0     0.7     0.2     ...
```

the return result would change the columns and keep the format

Parameters::

- **entity_id** –
- **df** –

Returns::

zvt.factors.macd

Modules

zvt.factors.macd.macd_factor

---

zvt.factors.macd.macd_factor

Classes

`BullFactor`(entity_schema, provider, ...)

`GoldCrossFactor`(entity_schema, provider, ...)

`KeepBullFactor`(entity_schema, provider, ...)

`LiveOrDeadFactor`(entity_schema, provider, ...)

`MacdFactor`(entity_schema, provider, ...)

BullFactor

class zvt.factors.macd.macd_factor.BullFactor(*entity_schema: ~typing.Type[~zvt.contract.schema.TradableEntity] = <class 'zvt.domain.meta.stock_meta.Stock'>*, *provider: ~typing.Optional[str] = None*, *entity_provider: ~typing.Optional[str] = None*, *entity_ids: ~typing.Optional[~typing.List[str]] = None*, *exchanges: ~typing.Optional[~typing.List[str]] = None*, *codes: ~typing.Optional[~typing.List[str]] = None*, *start_timestamp: ~typing.Optional[~typing.Union[~pandas._libs.tslibs.timestamps.Timestamp, str]] = None*, *end_timestamp: ~typing.Optional[~typing.Union[~pandas._libs.tslibs.timestamps.Timestamp, str]] = None*, *columns: ~typing.Optional[~typing.List] = None*, *filters: ~typing.Optional[~typing.List] = None*, *order: ~typing.Optional[object] = None*, *limit: ~typing.Optional[int] = None*, *level: ~typing.Union[str, ~zvt.contract.IntervalLevel] = IntervalLevel.LEVEL_1DAY*, *category_field: str = 'entity_id'*, *time_field: str = 'timestamp'*, *keep_window: ~typing.Optional[int] = None*, *keep_all_timestamp: bool = False*, *fill_method: str = 'ffill'*, *effective_number: ~typing.Optional[int] = None*, *transformer: ~typing.Optional[~zvt.contract.factor.Transformer] = None*, *accumulator: ~typing.Optional[~zvt.contract.factor.Accumulator] = None*, *need_persist: bool = False*, *only_compute_factor: bool = False*, *factor_name: ~typing.Optional[str] = None*, *clear_state: bool = False*, *only_load_factor: bool = False*, *adjust_type: ~typing.Optional[~typing.Union[~zvt.contract.AdjustType, str]] = None*)

Bases: **MacdFactor**

__init__(*entity_schema: ~typing.Type[~zvt.contract.schema.TradableEntity] = <class 'zvt.domain.meta.stock_meta.Stock'>*, *provider: ~typing.Optional[str] = None*, *entity_provider: ~typing.Optional[str] = None*, *entity_ids: ~typing.Optional[~typing.List[str]] = None*, *exchanges: ~typing.Optional[~typing.List[str]] = None*, *codes: ~typing.Optional[~typing.List[str]] = None*, *start_timestamp: ~typing.Optional[~typing.Union[~pandas._libs.tslibs.timestamps.Timesta*

mp, str]] = None, end_timestamp: ~typing.Optional[~typing.Union[~pandas._libs.tslibs.timestamps.Timestamp, str]] = None, columns: ~typing.Optional[~typing.List] = None, filters: ~typing.Optional[~typing.List] = None, order: ~typing.Optional[object] = None, limit: ~typing.Optional[int] = None, level: ~typing.Union[str, ~zvt.contract.IntervalLevel] = IntervalLevel.LEVEL_1DAY, category_field: str = 'entity_id', time_field: str = 'timestamp', keep_window: ~typing.Optional[int] = None, keep_all_timestamp: bool = False, fill_method: str = 'ffill', effective_number: ~typing.Optional[int] = None, transformer: ~typing.Optional[~zvt.contract.factor.Transformer] = None, accumulator: ~typing.Optional[~zvt.contract.factor.Accumulator] = None, need_persist: bool = False, only_compute_factor: bool = False, factor_name: ~typing.Optional[str] = None, clear_state: bool = False, only_load_factor: bool = False, adjust_type: ~typing.Optional[~typing.Union[~zvt.contract.AdjustType, str]] = None) → None

Parameters::

- **keep_all_timestamp** –
- **fill_method** –
- **effective_number** –
- **transformer** –
- **accumulator** –
- **need_persist** – whether persist factor
- **only_compute_factor** – only compute factor nor result
- **factor_name** –
- **clear_state** –
- **only_load_factor** – only load factor and compute result

accumulator*: Accumulator = None*

accumulator for this factor if not passed as __init__ argument

clear_state_data(*entity_id=None*)

clear state of the entity

Parameters:: **entity_id** – entity id

decode_state(*state: str*)

decode state

Parameters:: **state** –

Returns::

encode_state(*state: object*)

encode state

Parameters:: **state** –

Returns::

factor_col_map_object_hook() → dict

:return:{col:object_hook}

factor_schema*: Type[Mixin] = None*

define the schema for persist,its columns should be same as indicators in transformer or accumulator

move_on(*to_timestamp: Optional[Union[Timestamp, str]] = None*, *timeout: int = 20*) → object

using continual fetching data in realtime 1)get the data happened before to_timestamp,if not set,get all the data which means to now 2)if computing_window set,the data_df would be cut for saving memory

Parameters::

- **to_timestamp** –
- **timeout** (*int*) –

Returns::

Return type::

name = *None*

name of the service, default name of class if not set manually

on_data_changed(*data: DataFrame*)

overwrite it for computing after data added

Parameters:: **data** –

on_data_loaded(*data: DataFrame*)

Parameters:: **data** –

on_entity_data_changed(*entity*, *added_data: DataFrame*)

overwrite it for computing after entity data added

Parameters::

- **entity** –
- **added_data** –

state_schema

alias of `FactorState`

transformer*: Transformer = <zvt.factors.algorithm.MacdTransformer object>*

transformer for this factor if not passed as __init__ argument

pipe_df*: pd.DataFrame*

中间结果，不持久化 data_df->pipe_df

factor_df*: pd.DataFrame*

计算因子的结果，可持久化,通过对pipe_df的计算得到 pipe_df->factor_df

result_df*: pd.DataFrame*

result_df是用于选股的标准df,通过对factor_df的计算得到 factor_df->result_df

states*: dict*

entity_id:state

GoldCrossFactor

class zvt.factors.macd.macd_factor.GoldCrossFactor(*entity_schema: ~typing.Type[~zvt.contract.schema.TradableEntity] = <class 'zvt.domain.meta.stock_meta.Stock'>*, *provider: ~typing.Optional[str] = None*, *entity_provider: ~typing.Optional[str] = None*, *entity_ids: ~typing.Optional[~typing.List[str]] = None*, *exchanges: ~typing.Optional[~typing.List[str]] = None*, *codes: ~typing.Optional[~typing.List[str]] = None*, *start_timestamp: ~typing.Optional[~typing.Union[~pandas._libs.tslibs.timestamps.Timestamp, str]] = None*, *end_timestamp: ~typing.Optional[~typing.Union[~pandas._libs.tslibs.timestamps.Timestamp, str]] = None*, *columns: ~typing.Optional[~typing.List] = None*, *filters: ~typing.Optional[~typing.List] = None*, *order: ~typing.Optional[object] = None*, *limit: ~typing.Optional[int] = None*, *level: ~typing.Union[str, ~zvt.contract.IntervalLevel] = IntervalLevel.LEVEL_1DAY*, *category_field: str = 'entity_id'*, *time_field: str = 'timestamp'*, *keep_window: ~typing.Optional[int] = None*, *keep_all_timestamp: bool = False*, *fill_method: str = 'ffill'*, *effective_number: ~typing.Optional[int] = None*, *transformer: ~typing.Optional[~zvt.contract.factor.Transformer] = None*, *accumulator: ~typing.Optional[~zvt.contract.factor.Accumulator] = None*, *need_persist: bool = False*, *only_compute_factor: bool = False*, *factor_name: ~typing.Optional[str] = None*, *clear_state: bool = False*, *only_load_factor: bool = False*, *adjust_type: ~typing.Optional[~typing.Union[~zvt.contract.AdjustType, str]] = None*)

Bases: **MacdFactor**

__init__(*entity_schema: ~typing.Type[~zvt.contract.schema.TradableEntity] = <class 'zvt.domain.meta.stock_meta.Stock'>*, *provider: ~typing.Optional[str] = None*, *entity_provider: ~typing.Optional[str] = None*, *entity_ids: ~typing.Optional[~typing.List[str]] = None*, *exchanges: ~typing.Optional[~typing.List[str]] = None*, *codes: ~typing.Optional[~typing.List[str]] = None*, *start_timestamp: ~typing.Optional[~typing.Union[~pandas._libs.tslibs.timestamps.Timesta*

mp, str]] = None, end_timestamp: ~typing.Optional[~typing.Union[~pandas._libs.tslibs.timestamps.Timestamp, str]] = None, columns: ~typing.Optional[~typing.List] = None, filters: ~typing.Optional[~typing.List] = None, order: ~typing.Optional[object] = None, limit: ~typing.Optional[int] = None, level: ~typing.Union[str, ~zvt.contract.IntervalLevel] = IntervalLevel.LEVEL_1DAY, category_field: str = 'entity_id', time_field: str = 'timestamp', keep_window: ~typing.Optional[int] = None, keep_all_timestamp: bool = False, fill_method: str = 'ffill', effective_number: ~typing.Optional[int] = None, transformer: ~typing.Optional[~zvt.contract.factor.Transformer] = None, accumulator: ~typing.Optional[~zvt.contract.factor.Accumulator] = None, need_persist: bool = False, only_compute_factor: bool = False, factor_name: ~typing.Optional[str] = None, clear_state: bool = False, only_load_factor: bool = False, adjust_type: ~typing.Optional[~typing.Union[~zvt.contract.AdjustType, str]] = None) → None

Parameters::

- **keep_all_timestamp** –
- **fill_method** –
- **effective_number** –
- **transformer** –
- **accumulator** –
- **need_persist** – whether persist factor
- **only_compute_factor** – only compute factor nor result
- **factor_name** –
- **clear_state** –
- **only_load_factor** – only load factor and compute result

accumulator*: Accumulator = None*

accumulator for this factor if not passed as __init__ argument

clear_state_data(*entity_id=None*)

clear state of the entity

Parameters:: **entity_id** – entity id

decode_state(*state: str*)

decode state

Parameters:: **state** –

Returns::

encode_state(*state: object*)

encode state

Parameters:: **state** –

Returns::

factor_col_map_object_hook() → dict

:return:{col:object_hook}

factor_schema*: Type[Mixin] = None*

define the schema for persist,its columns should be same as indicators in transformer or accumulator

move_on(*to_timestamp: Optional[Union[Timestamp, str]] = None*, *timeout: int = 20*) → object

using continual fetching data in realtime 1)get the data happened before to_timestamp,if not set,get all the data which means to now 2)if computing_window set,the data_df would be cut for saving memory

Parameters::

- **to_timestamp** –
- **timeout** (*int*) –

Returns::

Return type::

name = *None*

name of the service, default name of class if not set manually

on_data_changed(*data: DataFrame*)

overwrite it for computing after data added

Parameters:: **data** –

on_data_loaded(*data: DataFrame*)

Parameters:: **data** –

on_entity_data_changed(*entity*, *added_data: DataFrame*)

overwrite it for computing after entity data added

Parameters::

- **entity** –
- **added_data** –

state_schema

alias of `FactorState`

transformer*: Transformer = <zvt.factors.algorithm.MacdTransformer object>*

transformer for this factor if not passed as __init__ argument

pipe_df*: pd.DataFrame*

中间结果，不持久化 data_df->pipe_df

factor_df*: pd.DataFrame*

计算因子的结果，可持久化,通过对pipe_df的计算得到 pipe_df->factor_df

result_df*: pd.DataFrame*

result_df是用于选股的标准df,通过对factor_df的计算得到 factor_df->result_df

states*: dict*

entity_id:state

KeepBullFactor

class zvt.factors.macd.macd_factor.KeepBullFactor(*entity_schema: ~typing.Type[~zvt.contract.schema.TradableEntity] = <class 'zvt.domain.meta.stock_meta.Stock'>, provider: ~typing.Optional[str] = None, entity_provider: ~typing.Optional[str] = None, entity_ids: ~typing.Optional[~typing.List[str]] = None, exchanges: ~typing.Optional[~typing.List[str]] = None, codes: ~typing.Optional[~typing.List[str]] = None, start_timestamp: ~typing.Optional[~typing.Union[~pandas._libs.tslibs.timestamps.Timestamp, str]] = None, end_timestamp: ~typing.Optional[~typing.Union[~pandas._libs.tslibs.timestamps.Timestamp, str]] = None, columns: ~typing.Optional[~typing.List] = None, filters: ~typing.Optional[~typing.List] = None, order: ~typing.Optional[object] = None, limit: ~typing.Optional[int] = None, level: ~typing.Union[str, ~zvt.contract.IntervalLevel] = IntervalLevel.LEVEL_1DAY, category_field: str = 'entity_id', time_field: str = 'timestamp', keep_window: ~typing.Optional[int] = None, keep_all_timestamp: bool = False, fill_method: str = 'ffill', effective_number: ~typing.Optional[int] = None, transformer: ~typing.Optional[~zvt.contract.factor.Transformer] = None, accumulator: ~typing.Optional[~zvt.contract.factor.Accumulator] = None, need_persist: bool = False, only_compute_factor: bool = False, factor_name: ~typing.Optional[str] = None, clear_state: bool = False, only_load_factor: bool = False, adjust_type: ~typing.Optional[~typing.Union[~zvt.contract.AdjustType, str]] = None*)

Bases: **BullFactor**

__init__(*entity_schema: ~typing.Type[~zvt.contract.schema.TradableEntity] = <class 'zvt.domain.meta.stock_meta.Stock'>, provider: ~typing.Optional[str] = None, entity_provider: ~typing.Optional[str] = None, entity_ids: ~typing.Optional[~typing.List[str]] = None, exchanges: ~typing.Optional[~typing.List[str]] = None, codes: ~typing.Optional[~typing.List[str]] = None, start_timestamp: ~typing.Optional[~typing.Union[~pandas._libs.tslibs.timestamps.Timesta*

mp, str]] = None, end_timestamp: ~typing.Optional[~typing.Union[~pandas._libs.tslibs.timestamps.Timestamp, str]] = None, columns: ~typing.Optional[~typing.List] = None, filters: ~typing.Optional[~typing.List] = None, order: ~typing.Optional[object] = None, limit: ~typing.Optional[int] = None, level: ~typing.Union[str, ~zvt.contract.IntervalLevel] = IntervalLevel.LEVEL_1DAY, category_field: str = 'entity_id', time_field: str = 'timestamp', keep_window: ~typing.Optional[int] = None, keep_all_timestamp: bool = False, fill_method: str = 'ffill', effective_number: ~typing.Optional[int] = None, transformer: ~typing.Optional[~zvt.contract.factor.Transformer] = None, accumulator: ~typing.Optional[~zvt.contract.factor.Accumulator] = None, need_persist: bool = False, only_compute_factor: bool = False, factor_name: ~typing.Optional[str] = None, clear_state: bool = False, only_load_factor: bool = False, adjust_type: ~typing.Optional[~typing.Union[~zvt.contract.AdjustType, str]] = None) → None

Parameters::

- **keep_all_timestamp** –
- **fill_method** –
- **effective_number** –
- **transformer** –
- **accumulator** –
- **need_persist** – whether persist factor
- **only_compute_factor** – only compute factor nor result
- **factor_name** –
- **clear_state** –
- **only_load_factor** – only load factor and compute result

accumulator*: Accumulator = None*

accumulator for this factor if not passed as __init__ argument

clear_state_data(*entity_id=None*)

clear state of the entity

Parameters:: **entity_id** – entity id

decode_state(*state: str*)

decode state

Parameters:: **state** –

Returns::

encode_state(*state: object*)

encode state

Parameters:: **state** –

Returns::

factor_col_map_object_hook() → dict

:return:{col:object_hook}

factor_schema*: Type[Mixin] = None*

define the schema for persist,its columns should be same as indicators in transformer or accumulator

move_on(*to_timestamp: Optional[Union[Timestamp, str]] = None*, *timeout: int = 20*) → object

using continual fetching data in realtime 1)get the data happened before to_timestamp,if not set,get all the data which means to now 2)if computing_window set,the data_df would be cut for saving memory

Parameters::

- **to_timestamp** –
- **timeout** (*int*) –

Returns::

Return type::

name = *None*

name of the service, default name of class if not set manually

on_data_changed(*data: DataFrame*)

overwrite it for computing after data added

Parameters:: **data** –

on_data_loaded(*data: DataFrame*)

Parameters:: **data** –

on_entity_data_changed(*entity*, *added_data: DataFrame*)

overwrite it for computing after entity data added

Parameters::

- **entity** –
- **added_data** –

state_schema

alias of `FactorState`

transformer*: Transformer = <zvt.factors.algorithm.MacdTransformer object>*

transformer for this factor if not passed as __init__ argument

pipe_df*: pd.DataFrame*

中间结果，不持久化 data_df->pipe_df

factor_df*: pd.DataFrame*

计算因子的结果，可持久化,通过对pipe_df的计算得到 pipe_df->factor_df

result_df*: pd.DataFrame*

result_df是用于选股的标准df,通过对factor_df的计算得到 factor_df->result_df

states*: dict*

entity_id:state

LiveOrDeadFactor

class zvt.factors.macd.macd_factor.LiveOrDeadFactor(*entity_schema: ~typing.Type[~zvt.contract.schema.TradableEntity] = <class 'zvt.domain.meta.stock_meta.Stock'>*, *provider: ~typing.Optional[str] = None*, *entity_provider: ~typing.Optional[str] = None*, *entity_ids: ~typing.Optional[~typing.List[str]] = None*, *exchanges: ~typing.Optional[~typing.List[str]] = None*, *codes: ~typing.Optional[~typing.List[str]] = None*, *start_timestamp: ~typing.Optional[~typing.Union[~pandas._libs.tslibs.timestamps.Timestamp, str]] = None*, *end_timestamp: ~typing.Optional[~typing.Union[~pandas._libs.tslibs.timestamps.Timestamp, str]] = None*, *columns: ~typing.Optional[~typing.List] = None*, *filters: ~typing.Optional[~typing.List] = None*, *order: ~typing.Optional[object] = None*, *limit: ~typing.Optional[int] = None*, *level: ~typing.Union[str, ~zvt.contract.IntervalLevel] = IntervalLevel.LEVEL_1DAY*, *category_field: str = 'entity_id'*, *time_field: str = 'timestamp'*, *keep_window: ~typing.Optional[int] = None*, *keep_all_timestamp: bool = False*, *fill_method: str = 'ffill'*, *effective_number: ~typing.Optional[int] = None*, *transformer: ~typing.Optional[~zvt.contract.factor.Transformer] = None*, *accumulator: ~typing.Optional[~zvt.contract.factor.Accumulator] = None*, *need_persist: bool = False*, *only_compute_factor: bool = False*, *factor_name: ~typing.Optional[str] = None*, *clear_state: bool = False*, *only_load_factor: bool = False*, *adjust_type: ~typing.Optional[~typing.Union[~zvt.contract.AdjustType, str]] = None*)

Bases: **MacdFactor**

__init__(*entity_schema: ~typing.Type[~zvt.contract.schema.TradableEntity] = <class 'zvt.domain.meta.stock_meta.Stock'>*, *provider: ~typing.Optional[str] = None*, *entity_provider: ~typing.Optional[str] = None*, *entity_ids: ~typing.Optional[~typing.List[str]] = None*, *exchanges: ~typing.Optional[~typing.List[str]] = None*, *codes: ~typing.Optional[~typing.List[str]] = None*, *start_timestamp: ~typing.Optional[~typing.Union[~pandas._libs.tslibs.timestamps.Timesta*

mp, str]] = None, end_timestamp: ~typing.Optional[~typing.Union[~pandas._libs.tslibs.timestamps.Timestamp, str]] = None, columns: ~typing.Optional[~typing.List] = None, filters: ~typing.Optional[~typing.List] = None, order: ~typing.Optional[object] = None, limit: ~typing.Optional[int] = None, level: ~typing.Union[str, ~zvt.contract.IntervalLevel] = IntervalLevel.LEVEL_1DAY, category_field: str = 'entity_id', time_field: str = 'timestamp', keep_window: ~typing.Optional[int] = None, keep_all_timestamp: bool = False, fill_method: str = 'ffill', effective_number: ~typing.Optional[int] = None, transformer: ~typing.Optional[~zvt.contract.factor.Transformer] = None, accumulator: ~typing.Optional[~zvt.contract.factor.Accumulator] = None, need_persist: bool = False, only_compute_factor: bool = False, factor_name: ~typing.Optional[str] = None, clear_state: bool = False, only_load_factor: bool = False, adjust_type: ~typing.Optional[~typing.Union[~zvt.contract.AdjustType, str]] = None) → None

Parameters::

- **keep_all_timestamp** –
- **fill_method** –
- **effective_number** –
- **transformer** –
- **accumulator** –
- **need_persist** – whether persist factor
- **only_compute_factor** – only compute factor nor result
- **factor_name** –
- **clear_state** –
- **only_load_factor** – only load factor and compute result

accumulator*: Accumulator = None*

accumulator for this factor if not passed as __init__ argument

clear_state_data(*entity_id=None*)

clear state of the entity

Parameters:: **entity_id** – entity id

decode_state(*state: str*)

decode state

Parameters:: **state** –

Returns::

encode_state(*state: object*)

encode state

Parameters:: **state** –

Returns::

factor_col_map_object_hook() → dict

:return:{col:object_hook}

factor_schema*: Type[Mixin] = None*

define the schema for persist,its columns should be same as indicators in transformer or accumulator

move_on(*to_timestamp: Optional[Union[Timestamp, str]] = None*, *timeout: int = 20*) → object

using continual fetching data in realtime 1)get the data happened before to_timestamp,if not set,get all the data which means to now 2)if computing_window set,the data_df would be cut for saving memory

Parameters::

- **to_timestamp** –
- **timeout** (*int*) –

Returns::

Return type::

name = *None*

name of the service, default name of class if not set manually

on_data_changed(*data: DataFrame*)

overwrite it for computing after data added

Parameters:: **data** –

on_data_loaded(*data: DataFrame*)

Parameters:: **data** –

on_entity_data_changed(*entity*, *added_data: DataFrame*)

overwrite it for computing after entity data added

Parameters::

- **entity** –
- **added_data** –

state_schema

alias of `FactorState`

transformer*: Transformer = <zvt.factors.algorithm.MacdTransformer object>*

transformer for this factor if not passed as __init__ argument

pipe_df*: pd.DataFrame*

中间结果，不持久化 data_df->pipe_df

factor_df*: pd.DataFrame*

计算因子的结果，可持久化,通过对pipe_df的计算得到 pipe_df->factor_df

result_df*: pd.DataFrame*

result_df是用于选股的标准df,通过对factor_df的计算得到 factor_df->result_df

states*: dict*

entity_id:state

MacdFactor

class zvt.factors.macd.macd_factor.MacdFactor(*entity_schema: ~typing.Type[~zvt.contract.schema.TradableEntity] = <class 'zvt.domain.meta.stock_meta.Stock'>*, *provider: ~typing.Optional[str] = None*, *entity_provider: ~typing.Optional[str] = None*, *entity_ids: ~typing.Optional[~typing.List[str]] = None*, *exchanges: ~typing.Optional[~typing.List[str]] = None*, *codes: ~typing.Optional[~typing.List[str]] = None*, *start_timestamp: ~typing.Optional[~typing.Union[~pandas._libs.tslibs.timestamps.Timestamp, str]] = None*, *end_timestamp: ~typing.Optional[~typing.Union[~pandas._libs.tslibs.timestamps.Timestamp, str]] = None*, *columns: ~typing.Optional[~typing.List] = None*, *filters: ~typing.Optional[~typing.List] = None*, *order: ~typing.Optional[object] = None*, *limit: ~typing.Optional[int] = None*, *level: ~typing.Union[str, ~zvt.contract.IntervalLevel] = IntervalLevel.LEVEL_1DAY*, *category_field: str = 'entity_id'*, *time_field: str = 'timestamp'*, *keep_window: ~typing.Optional[int] = None*, *keep_all_timestamp: bool = False*, *fill_method: str = 'ffill'*, *effective_number: ~typing.Optional[int] = None*, *transformer: ~typing.Optional[~zvt.contract.factor.Transformer] = None*, *accumulator: ~typing.Optional[~zvt.contract.factor.Accumulator] = None*, *need_persist: bool = False*, *only_compute_factor: bool = False*, *factor_name: ~typing.Optional[str] = None*, *clear_state: bool = False*, *only_load_factor: bool = False*, *adjust_type: ~typing.Optional[~typing.Union[~zvt.contract.AdjustType, str]] = None*)

Bases: **TechnicalFactor**

__init__(*entity_schema: ~typing.Type[~zvt.contract.schema.TradableEntity] = <class 'zvt.domain.meta.stock_meta.Stock'>*, *provider: ~typing.Optional[str] = None*, *entity_provider: ~typing.Optional[str] = None*, *entity_ids: ~typing.Optional[~typing.List[str]] = None*, *exchanges: ~typing.Optional[~typing.List[str]] = None*, *codes: ~typing.Optional[~typing.List[str]] = None*, *start_timestamp: ~typing.Optional[~typing.Union[~pandas._libs.tslibs.timestamps.Timesta*

mp, str]] = None, end_timestamp: ~typing.Optional[~typing.Union[~pandas._libs.tslibs.timestamps.Timestamp, str]] = None, columns: ~typing.Optional[~typing.List] = None, filters: ~typing.Optional[~typing.List] = None, order: ~typing.Optional[object] = None, limit: ~typing.Optional[int] = None, level: ~typing.Union[str, ~zvt.contract.IntervalLevel] = IntervalLevel.LEVEL_1DAY, category_field: str = 'entity_id', time_field: str = 'timestamp', keep_window: ~typing.Optional[int] = None, keep_all_timestamp: bool = False, fill_method: str = 'ffill', effective_number: ~typing.Optional[int] = None, transformer: ~typing.Optional[~zvt.contract.factor.Transformer] = None, accumulator: ~typing.Optional[~zvt.contract.factor.Accumulator] = None, need_persist: bool = False, only_compute_factor: bool = False, factor_name: ~typing.Optional[str] = None, clear_state: bool = False, only_load_factor: bool = False, adjust_type: ~typing.Optional[~typing.Union[~zvt.contract.AdjustType, str]] = None) → None

Parameters::

- **keep_all_timestamp** –
- **fill_method** –
- **effective_number** –
- **transformer** –
- **accumulator** –
- **need_persist** – whether persist factor
- **only_compute_factor** – only compute factor nor result
- **factor_name** –
- **clear_state** –
- **only_load_factor** – only load factor and compute result

transformer*: Transformer = <zvt.factors.algorithm.MacdTransformer object>*

transformer for this factor if not passed as __init__ argument

accumulator*: Accumulator = None*

accumulator for this factor if not passed as __init__ argument

clear_state_data(*entity_id=None*)

clear state of the entity

Parameters:: **entity_id** – entity id

decode_state(*state: str*)

decode state

Parameters:: **state** –

Returns::

encode_state(*state: object*)

encode state

Parameters:: **state** –

Returns::

factor_col_map_object_hook() → dict

:return:{col:object_hook}

factor_schema*: Type[Mixin] = None*

define the schema for persist,its columns should be same as indicators in transformer or accumulator

move_on(*to_timestamp: Optional[Union[Timestamp, str]] = None*, *timeout: int = 20*) → object

using continual fetching data in realtime 1)get the data happened before to_timestamp,if not set,get all the data which means to now 2)if computing_window set,the data_df would be cut for saving memory

Parameters::

- **to_timestamp** –
- **timeout** (*int*) –

Returns::

Return type::

name = *None*

name of the service, default name of class if not set manually

on_data_changed(*data: DataFrame*)

overwrite it for computing after data added

Parameters:: **data** –

on_data_loaded(*data: DataFrame*)

Parameters:: **data** –

on_entity_data_changed(*entity*, *added_data: DataFrame*)

overwrite it for computing after entity data added

Parameters::

- **entity** –
- **added_data** –

state_schema

alias of `FactorState`

pipe_df*: pd.DataFrame*

中间结果，不持久化 data_df->pipe_df

factor_df*: pd.DataFrame*

计算因子的结果，可持久化,通过对pipe_df的计算得到 pipe_df->factor_df

result_df*: pd.DataFrame*

result_df是用于选股的标准df,通过对factor_df的计算得到 factor_df->result_df

states*: dict*

entity_id:state

zvt.factors.shape

Functions

a_include_b(a, b)	kdata a includes kdata b
decode_fenxing(dct)	
decode_rect(dct)	
fenxing_power(left, middle, right[, fenxing])	
get_direction(kdata, pre_kdata[, current])	
handle_duan(fenxing_list[, pre_duan_state])	
handle_first_fenxing(one_df[, step])	
handle_including(one_df, index, kdata, ...)	
handle_zhongshu(points, acc_df, end_index[, ...])	
is_down(kdata, pre_kdata)	

`is_up`(kdata, pre_kdata)

Classes

`Direction`(value)	An enumeration.
`FactorStateEncoder`(*[, skipkeys, ...])	
`Fenxing`(state, kdata, index)	

zvt.factors.shape.a_include_b

zvt.factors.shape.a_include_b(*a: Series*, *b: Series*) → bool

kdata a includes kdata b

Parameters::

- **a** –
- **b** –

Returns::

zvt.factors.shape.decode_fenxing

zvt.factors.shape.decode_fenxing(*dct*)

zvt.factors.shape.decode_rect

zvt.factors.shape.decode_rect(*dct*)

zvt.factors.shape.fenxing_power

zvt.factors.shape.fenxing_power(*left*, *middle*, *right*, *fenxing='tmp_ding'*)

zvt.factors.shape.get_direction

zvt.factors.shape.get_direction(*kdata*, *pre_kdata*, *current=Direction.up*) → Direction

zvt.factors.shape.handle_duan

zvt.factors.shape.handle_duan(*fenxing_list: List[Fenxing]*, *pre_duan_state='yi'*)

zvt.factors.shape.handle_first_fenxing

zvt.factors.shape.handle_first_fenxing(*one_df*, *step=11*)

zvt.factors.shape.handle_including

zvt.factors.shape.handle_including(*one_df*, *index*, *kdata*, *pre_index*, *pre_kdata*, *tmp_direction: Direction*)

zvt.factors.shape.handle_zhongshu

zvt.factors.shape.handle_zhongshu(*points: list*, *acc_df*, *end_index*, *zhongshu_col='zhongshu'*, *zhongshu_change_col='zhongshu_change'*)

zvt.factors.shape.is_down

zvt.factors.shape.is_down(*kdata*, *pre_kdata*)

zvt.factors.shape.is_up

zvt.factors.shape.is_up(*kdata*, *pre_kdata*)

Direction

class zvt.factors.shape.Direction(*value*)

Bases: `Enum`

An enumeration.

__init__()

FactorStateEncoder

class zvt.factors.shape.FactorStateEncoder(*, *skipkeys=False*, *ensure_ascii=True*, *check_circular=True*, *allow_nan=True*, *sort_keys=False*, *indent=None*, *separators=None*, *default=None*)

Bases: **`JSONEncoder`**

__init__(*, *skipkeys=False*, *ensure_ascii=True*, *check_circular=True*, *allow_nan=True*, *sort_keys=False*, *indent=None*, *separators=None*, *default=None*)

Constructor for JSONEncoder, with sensible defaults.

If skipkeys is false, then it is a TypeError to attempt encoding of keys that are not str, int, float or None. If skipkeys is True, such items are simply skipped.

If ensure_ascii is true, the output is guaranteed to be str objects with all incoming non-ASCII characters escaped. If ensure_ascii is false, the output can contain non-ASCII characters.

If check_circular is true, then lists, dicts, and custom encoded objects will be checked for circular references during encoding to prevent an infinite recursion (which would cause an RecursionError). Otherwise, no such check takes place.

If allow_nan is true, then NaN, Infinity, and -Infinity will be encoded as such. This behavior is not JSON specification compliant, but is consistent with most JavaScript based encoders and decoders. Otherwise, it will be a ValueError to encode such floats.

If sort_keys is true, then the output of dictionaries will be sorted by key; this is useful for regression tests to ensure that JSON serializations can be compared on a day-to-day basis.

If indent is a non-negative integer, then JSON array elements and object members will be pretty-printed with that indent level. An indent level of 0 will only insert newlines. None is the most compact representation.

If specified, separators should be an (item_separator, key_separator) tuple. The default is (', ', ': ') if *indent* is `None` and (',', ': ') otherwise. To get the most compact JSON representation, you should specify (',', ':') to eliminate whitespace.

If specified, default is a function that gets called for objects that can't otherwise be serialized. It should return a JSON encodable version of the object or raise a `TypeError`.

default(*object*)

Implement this method in a subclass such that it returns a serializable object for `o`, or calls the base implementation (to raise a `TypeError`).

For example, to support arbitrary iterators, you could implement default like this:

```
def default(self, o):
    try:
        iterable = iter(o)
    except TypeError:
        pass
    else:
        return list(iterable)
    # Let the base class default method raise the
TypeError
    return JSONEncoder.default(self, o)
```

encode(*o*)

Return a JSON string representation of a Python data structure.

```
>>> from json.encoder import JSONEncoder
>>> JSONEncoder().encode({"foo": ["bar", "baz"]})
'{"foo": ["bar", "baz"]}'
```

iterencode(*o*, *_one_shot=False*)

Encode the given object and yield each string representation as available.

For example:

```
for chunk in JSONEncoder().iterencode(bigobject):
    mysocket.write(chunk)
```

Fenxing

class zvt.factors.shape.Fenxing(*state*, *kdata*, *index*)

Bases: `Bean`

__init__(*state*, *kdata*, *index*) → None

zvt.factors.target_selector

Classes

`SelectMode`(value)	An enumeration.
`TargetSelector`([entity_ids, entity_schema, ...])	
`TradeType`(value)	An enumeration.

SelectMode

class zvt.factors.target_selector.SelectMode(*value*)

Bases: `Enum`

An enumeration.

__init__()

TargetSelector

class zvt.factors.target_selector.TargetSelector(*entity_ids=None*, *entity_schema=<class 'zvt.domain.meta.stock_meta.Stock'>*, *exchanges=None*, *codes=None*, *start_timestamp=None*, *end_timestamp=None*, *long_threshold=0.8*, *short_threshold=0.2*, *level=IntervalLevel.LEVEL_1DAY*, *provider=None*, *select_mode: ~zvt.factors.target_selector.SelectMode = SelectMode.condition_and*)

Bases: **`object`**

__init__(*entity_ids=None*, *entity_schema=<class 'zvt.domain.meta.stock_meta.Stock'>*, *exchanges=None*, *codes=None*, *start_timestamp=None*, *end_timestamp=None*, *long_threshold=0.8*, *short_threshold=0.2*, *level=IntervalLevel.LEVEL_1DAY*, *provider=None*, *select_mode: ~zvt.factors.target_selector.SelectMode = SelectMode.condition_and*) → None

TradeType

class zvt.factors.target_selector.TradeType(*value*)

Bases: **`Enum`**

An enumeration.

__init__()

zvt.factors.technical_factor

Classes

`TechnicalFactor`(entity_schema, provider, ...)

TechnicalFactor

class zvt.factors.technical_factor.TechnicalFactor(*entity_schema: ~typing.Type[~zvt.contract.schema.TradableEntity] = <class 'zvt.domain.meta.stock_meta.Stock'>*, *provider: ~typing.Optional[str] = None*, *entity_provider: ~typing.Optional[str] = None*, *entity_ids: ~typing.Optional[~typing.List[str]] = None*, *exchanges: ~typing.Optional[~typing.List[str]] = None*, *codes: ~typing.Optional[~typing.List[str]] = None*, *start_timestamp: ~typing.Optional[~typing.Union[~pandas._libs.tslibs.timestamps.Timestamp, str]] = None*, *end_timestamp: ~typing.Optional[~typing.Union[~pandas._libs.tslibs.timestamps.Timestamp, str]] = None*, *columns: ~typing.Optional[~typing.List] = None*, *filters: ~typing.Optional[~typing.List] = None*, *order: ~typing.Optional[object] = None*, *limit: ~typing.Optional[int] = None*, *level: ~typing.Union[str, ~zvt.contract.IntervalLevel] = IntervalLevel.LEVEL_1DAY*, *category_field: str = 'entity_id'*, *time_field: str = 'timestamp'*, *keep_window: ~typing.Optional[int] = None*, *keep_all_timestamp: bool = False*, *fill_method: str = 'ffill'*, *effective_number: ~typing.Optional[int] = None*, *transformer: ~typing.Optional[~zvt.contract.factor.Transformer] = None*, *accumulator: ~typing.Optional[~zvt.contract.factor.Accumulator] = None*, *need_persist: bool = False*, *only_compute_factor: bool = False*, *factor_name: ~typing.Optional[str] = None*, *clear_state: bool = False*, *only_load_factor: bool = False*, *adjust_type: ~typing.Optional[~typing.Union[~zvt.contract.AdjustType, str]] = None*)

Bases: **Factor**

__init__(*entity_schema: ~typing.Type[~zvt.contract.schema.TradableEntity] = <class 'zvt.domain.meta.stock_meta.Stock'>*, *provider: ~typing.Optional[str] = None*, *entity_provider: ~typing.Optional[str] = None*, *entity_ids: ~typing.Optional[~typing.List[str]] = None*, *exchanges: ~typing.Optional[~typing.List[str]] = None*, *codes: ~typing.Optional[~typing.List[str]] = None*, *start_timestamp: ~typing.Optional[~typing.Union[~pandas._libs.tslibs.timestamps.Timesta*

mp, str]] = None, end_timestamp: ~typing.Optional[~typing.Union[~pandas._libs.tslibs.timestamps.Timestamp, str]] = None, columns: ~typing.Optional[~typing.List] = None, filters: ~typing.Optional[~typing.List] = None, order: ~typing.Optional[object] = None, limit: ~typing.Optional[int] = None, level: ~typing.Union[str, ~zvt.contract.IntervalLevel] = IntervalLevel.LEVEL_1DAY, category_field: str = 'entity_id', time_field: str = 'timestamp', keep_window: ~typing.Optional[int] = None, keep_all_timestamp: bool = False, fill_method: str = 'ffill', effective_number: ~typing.Optional[int] = None, transformer: ~typing.Optional[~zvt.contract.factor.Transformer] = None, accumulator: ~typing.Optional[~zvt.contract.factor.Accumulator] = None, need_persist: bool = False, only_compute_factor: bool = False, factor_name: ~typing.Optional[str] = None, clear_state: bool = False, only_load_factor: bool = False, adjust_type: ~typing.Optional[~typing.Union[~zvt.contract.AdjustType, str]] = None) → None

Parameters::

- **keep_all_timestamp** –
- **fill_method** –
- **effective_number** –
- **transformer** –
- **accumulator** –
- **need_persist** – whether persist factor
- **only_compute_factor** – only compute factor nor result
- **factor_name** –
- **clear_state** –
- **only_load_factor** – only load factor and compute result

accumulator*: Accumulator = None*

accumulator for this factor if not passed as __init__ argument

clear_state_data(*entity_id=None*)

clear state of the entity

Parameters:: **entity_id** – entity id

decode_state(*state: str*)

decode state

Parameters:: **state** –

Returns::

encode_state(*state: object*)

encode state

Parameters:: **state** –

Returns::

factor_col_map_object_hook() → dict

:return:{col:object_hook}

factor_schema*: Type[Mixin] = None*

define the schema for persist,its columns should be same as indicators in transformer or accumulator

move_on(*to_timestamp: Optional[Union[Timestamp, str]] = None*, *timeout: int = 20*) → object

using continual fetching data in realtime 1)get the data happened before to_timestamp,if not set,get all the data which means to now 2)if computing_window set,the data_df would be cut for saving memory

Parameters::

- **to_timestamp** –
- **timeout** (*int*) –

Returns::

Return type::

name = *None*

name of the service, default name of class if not set manually

on_data_changed(*data: DataFrame*)

overwrite it for computing after data added

Parameters:: **data** –

on_data_loaded(*data: DataFrame*)

Parameters:: **data** –

on_entity_data_changed(*entity*, *added_data: DataFrame*)

overwrite it for computing after entity data added

Parameters::

- **entity** –
- **added_data** –

state_schema

alias of `FactorState`

transformer*: Transformer = None*

transformer for this factor if not passed as __init__ argument

pipe_df*: pd.DataFrame*

中间结果，不持久化 data_df->pipe_df

factor_df*: pd.DataFrame*

计算因子的结果，可持久化,通过对pipe_df的计算得到 pipe_df->factor_df

result_df*: pd.DataFrame*

result_df是用于选股的标准df,通过对factor_df的计算得到 factor_df->result_df

states*: dict*

entity_id:state

zvt.factors.top_stocks

Functions

`compute_top_stocks`([provider])

`get_top_stocks`(target_date[, return_type])

`get_vol_up_stocks`(target_date[, provider, ...])

`update_vol_up`()

`update_with_limit_up`()

Classes

`TopStocks`(**kwargs)

---

zvt.factors.top_stocks.compute_top_stocks

zvt.factors.top_stocks.compute_top_stocks(*provider='em'*)

zvt.factors.top_stocks.get_top_stocks

zvt.factors.top_stocks.get_top_stocks(*target_date*, *return_type='short'*)

zvt.factors.top_stocks.get_vol_up_stocks

zvt.factors.top_stocks.get_vol_up_stocks(*target_date*, *provider='em'*, *stock_type='small'*, *entity_ids=None*)

zvt.factors.top_stocks.update_vol_up

zvt.factors.top_stocks.update_vol_up()

zvt.factors.top_stocks.update_with_limit_up

zvt.factors.top_stocks.update_with_limit_up()

TopStocks

class zvt.factors.top_stocks.TopStocks(***kwargs*)

Bases: **`Base`**, **`Mixin`**

__init__(***kwargs*)

A simple constructor that allows initialization from kwargs.

Sets attributes on the constructed instance using the names and values in `kwargs`.

Only keys that are present as attributes of the instance's class are allowed. These could be, for example, any mapped columns or relationships.

entity_id

entity id

classmethod get_providers() → List[str]

providers of the schema defined by cls

Returns:: providers

classmethod get_storages(*provider: Optional[str] = None*)

get the storages info

Parameters:: **provider** – provider
Returns:: storages

id

id

classmethod query_data(*provider_index: int = 0*, *ids: Optional[List[str]] = None*, *entity_ids: Optional[List[str]] = None*, *entity_id: Optional[str] = None*, *codes: Optional[List[str]] = None*, *code: Optional[str] = None*, *level: Optional[Union[IntervalLevel, str]] = None*, *provider:*

Optional[str] = None, *columns: Optional[List] = None*, *col_label: Optional[dict] = None*, *return_type: str = 'df'*, *start_timestamp: Optional[Union[Timestamp, str]] = None*, *end_timestamp: Optional[Union[Timestamp, str]] = None*, *filters: Optional[List] = None*, *session: Optional[Session] = None*, *order=None*, *limit: Optional[int] = None*, *distinct=None*, *index: Optional[Union[str, list]] = None*, *drop_index_col=False*, *time_field: str = 'timestamp'*)

query data by the arguments

Parameters::

- **provider_index** –
- **data_schema** –
- **ids** –
- **entity_ids** –
- **entity_id** –
- **codes** –
- **code** –
- **level** –
- **provider** –
- **columns** –
- **col_label** – dict with key(column), value(label
- **return_type** – df, domain or dict. default is df
- **start_timestamp** –
- **end_timestamp** –
- **filters** –
- **session** –
- **order** –
- **limit** –
- **index** – index field name, str for single index, str list for multiple index
- **drop_index_col** – whether drop the col if it's in index, default False
- **time_field** –

Returns:: results basing on return_type.

classmethod record_data(*provider_index: int = 0*, *provider: Optional[str] = None*, *force_update=None*, *sleeping_time=None*, *exchanges=None*, *entity_id=None*, *entity_ids=None*, *code=None*, *codes=None*,

real_time=None, *fix_duplicate_way=None*, *start_timestamp=None*, *end_timestamp=None*, *one_day_trading_minutes=None*, ***kwargs*)

record data by the arguments

Parameters::

- **entity_id** –
- **provider_index** –
- **provider** –
- **force_update** –
- **sleeping_time** –
- **exchanges** –
- **entity_ids** –
- **code** –
- **codes** –
- **real_time** –
- **fix_duplicate_way** –
- **start_timestamp** –
- **end_timestamp** –
- **one_day_trading_minutes** –
- **kwargs** –

Returns::

classmethod register_provider(*provider*)

register the provider to the schema defined by cls

Parameters:: **provider** –

classmethod register_recorder_cls(*provider*, *recorder_cls*)

register the recorder for the schema

Parameters::

- **provider** –
- **recorder_cls** –

timestamp

the meaning could be different for different case,most time it means 'happen time'

zvt.factors.transformers

Classes

`CrossMaTransformer`([windows, cal_change_pct])

`FallBelowTransformer`([window, interval])

`SpecificTransformer`(buy_timestamp, ...)

CrossMaTransformer

class zvt.factors.transformers.CrossMaTransformer(*windows=None*, *cal_change_pct=False*)

Bases: **MaTransformer**

__init__(*windows=None*, *cal_change_pct=False*) → None

transform(*input_df: DataFrame*) → DataFrame

input_df format:

```
                          col1    col2    col3    ...
entity_id    timestamp
                          1.2     0.5     0.3     ...
                          1.0     0.7     0.2     ...
```

the return result would change the columns and keep the format

Parameters:: **input_df** –

Returns::

transform_one(*entity_id*, *df: DataFrame*) → DataFrame

transform_one would not take effects if transform was implemented. Just show how to implement it here, most of time you should overwrite transform directly for performance.

Parameters::

- **entity_id** –
- **df** –

Returns::

FallBelowTransformer

class zvt.factors.transformers.FallBelowTransformer(*window=10*, *interval=3*)

Bases: `Transformer`

__init__(*window=10*, *interval=3*) → None

transform(*input_df: DataFrame*) → DataFrame

input_df format:

```
                                col1    col2    col3    ...
entity_id    timestamp
                                1.2     0.5     0.3     ...
                                1.0     0.7     0.2     ...
```

the return result would change the columns and keep the format

Parameters:: **input_df** –

Returns::

transform_one(*entity_id: str*, *df: DataFrame*) → DataFrame

df format:

```
             col1    col2    col3    ...
timestamp
             1.2     0.5     0.3     ...
             1.0     0.7     0.2     ...
```

the return result would change the columns and keep the format

Parameters::

- **entity_id** –
- **df** –

Returns::

SpecificTransformer

class zvt.factors.transformers.SpecificTransformer(*buy_timestamp*, *sell_timestamp*)

Bases: `Transformer`

__init__(*buy_timestamp*, *sell_timestamp*) → None

transform(*input_df: DataFrame*) → DataFrame

input_df format:

```
                            col1    col2    col3    ...
entity_id    timestamp
                            1.2     0.5     0.3     ...
                            1.0     0.7     0.2     ...
```

the return result would change the columns and keep the format

Parameters:: **input_df** –

Returns::

transform_one(*entity_id: str*, *df: DataFrame*) → DataFrame

df format:

```
             col1    col2    col3    ...
timestamp
             1.2     0.5     0.3     ...
             1.0     0.7     0.2     ...
```

the return result would change the columns and keep the format

Parameters::

- **entity_id** –
- **df** –

Returns::

zvt.factors.z

Modules

zvt.factors.z.domain

---

zvt.factors.z.z_factor

---

zvt.factors.z.domain

Modules

zvt.factors.z.domain.common

---

zvt.factors.z.domain.stock_1d_z_factor

---

zvt.factors.z.domain.stock_1wk_z_factor

---

zvt.factors.z.domain.common

Classes

`ZFactorCommon`()

ZFactorCommon

class zvt.factors.z.domain.common.ZFactorCommon

Bases: `Mixin`

__init__()

entity_id = *Column(None, String(), table=None)*

entity id

classmethod get_providers() → List[str]

providers of the schema defined by cls

Returns:: providers

classmethod get_storages(*provider: Optional[str] = None*)

get the storages info

Parameters:: **provider** – provider

Returns:: storages

id = *Column(None, String(), table=None, primary_key=True, nullable=False)*

id

classmethod query_data(*provider_index: int = 0*, *ids: Optional[List[str]] = None*, *entity_ids: Optional[List[str]] = None*, *entity_id: Optional[str] = None*, *codes: Optional[List[str]] = None*, *code: Optional[str] = None*, *level: Optional[Union[IntervalLevel, str]] = None*, *provider: Optional[str] = None*, *columns: Optional[List] = None*, *col_label: Optional[dict] = None*, *return_type: str = 'df'*, *start_timestamp: Optional[Union[Timestamp, str]] = None*, *end_timestamp: Optional[Union[Timestamp, str]] = None*, *filters: Optional[List] = None*, *session: Optional[Session] = None*, *order=None*, *limit: Optional[int] = None*, *distinct=None*, *index: Optional[Union[str, list]] = None*, *drop_index_col=False*, *time_field: str = 'timestamp'*)

query data by the arguments

Parameters::

- **provider_index** –
- **data_schema** –
- **ids** –
- **entity_ids** –
- **entity_id** –
- **codes** –
- **code** –
- **level** –
- **provider** –
- **columns** –
- **col_label** – dict with key(column), value(label
- **return_type** – df, domain or dict. default is df
- **start_timestamp** –
- **end_timestamp** –
- **filters** –
- **session** –
- **order** –
- **limit** –
- **index** – index field name, str for single index, str list for multiple index
- **drop_index_col** – whether drop the col if it's in index, default False
- **time_field** –

Returns:: results basing on return_type.

classmethod record_data(*provider_index: int = 0*, *provider: Optional[str] = None*, *force_update=None*, *sleeping_time=None*, *exchanges=None*, *entity_id=None*, *entity_ids=None*, *code=None*, *codes=None*, *real_time=None*, *fix_duplicate_way=None*, *start_timestamp=None*, *end_timestamp=None*, *one_day_trading_minutes=None*, ***kwargs*)

record data by the arguments

Parameters::

- **entity_id** –
- **provider_index** –
- **provider** –
- **force_update** –
- **sleeping_time** –
- **exchanges** –
- **entity_ids** –
- **code** –
- **codes** –
- **real_time** –
- **fix_duplicate_way** –
- **start_timestamp** –
- **end_timestamp** –
- **one_day_trading_minutes** –
- **kwargs** –

Returns::

classmethod register_provider(*provider*)

register the provider to the schema defined by cls

Parameters:: **provider** –

classmethod register_recorder_cls(*provider*, *recorder_cls*)

register the recorder for the schema

Parameters::

- **provider** –
- **recorder_cls** –

timestamp = *Column(None, DateTime(), table=None)*

the meaning could be different for different case,most time it means 'happen time'

zvt.factors.z.domain.stock_1d_z_factor

Classes

`Stock1dZFactor`(**kwargs)

Stock1dZFactor

class zvt.factors.z.domain.stock_1d_z_factor.Stock1dZFactor(**kwargs)

Bases: **Base**, **ZFactorCommon**

__init__(**kwargs)

A simple constructor that allows initialization from kwargs.

Sets attributes on the constructed instance using the names and values in `kwargs`.

Only keys that are present as attributes of the instance's class are allowed. These could be, for example, any mapped columns or relationships.

bi_change

bi_di

bi_ding

bi_interval

bi_slope

bi_value

bi_zhongshu

bi_zhongshu_change

close

current_change

current_direction

current_interval

current_slope

current_zhongshu_change

current_zhongshu_y0

current_zhongshu_y1

duan_change

duan_di

duan_ding

duan_interval

duan_slope

duan_state

duan_value

entity_id

entity id

fenxing_power

classmethod get_providers() → List[str]

providers of the schema defined by cls

Returns:: providers

classmethod get_storages(*provider: Optional[str] = None*)

get the storages info

Parameters:: **provider** – provider

Returns:: storages

high

id

id

level

low

open

opposite_change

opposite_interval

opposite_slope

classmethod query_data(*provider_index: int = 0*, *ids: Optional[List[str]] = None*, *entity_ids: Optional[List[str]] = None*, *entity_id: Optional[str] = None*, *codes: Optional[List[str]] = None*, *code: Optional[str] = None*, *level: Optional[Union[IntervalLevel, str]] = None*, *provider: Optional[str] = None*, *columns: Optional[List] = None*, *col_label: Optional[dict] = None*, *return_type: str = 'df'*, *start_timestamp: Optional[Union[Timestamp, str]] = None*, *end_timestamp: Optional[Union[Timestamp, str]] = None*, *filters: Optional[List] = None*, *session: Optional[Session] = None*, *order=None*, *limit: Optional[int] = None*, *distinct=None*, *index: Optional[Union[str, list]] = None*, *drop_index_col=False*, *time_field: str = 'timestamp'*)

query data by the arguments

Parameters::

- **provider_index** –
- **data_schema** –
- **ids** –
- **entity_ids** –
- **entity_id** –
- **codes** –
- **code** –
- **level** –
- **provider** –
- **columns** –
- **col_label** – dict with key(column), value(label
- **return_type** – df, domain or dict. default is df
- **start_timestamp** –
- **end_timestamp** –
- **filters** –
- **session** –
- **order** –
- **limit** –
- **index** – index field name, str for single index, str list for multiple index
- **drop_index_col** – whether drop the col if it's in index, default False
- **time_field** –

Returns:: results basing on return_type.

classmethod record_data(*provider_index: int = 0*, *provider: Optional[str] = None*, *force_update=None*, *sleeping_time=None*, *exchanges=None*, *entity_id=None*, *entity_ids=None*, *code=None*, *codes=None*, *real_time=None*, *fix_duplicate_way=None*, *start_timestamp=None*, *end_timestamp=None*, *one_day_trading_minutes=None*, ***kwargs*)

record data by the arguments

Parameters::

- **entity_id** –
- **provider_index** –
- **provider** –
- **force_update** –
- **sleeping_time** –
- **exchanges** –
- **entity_ids** –
- **code** –
- **codes** –
- **real_time** –
- **fix_duplicate_way** –
- **start_timestamp** –
- **end_timestamp** –
- **one_day_trading_minutes** –
- **kwargs** –

Returns::

classmethod register_provider(*provider*)

register the provider to the schema defined by cls

Parameters:: **provider** –

classmethod register_recorder_cls(*provider*, *recorder_cls*)

register the recorder for the schema

Parameters::

- **provider** –
- **recorder_cls** –

timestamp

the meaning could be different for different case,most time it means 'happen time'

tmp_di

tmp_ding

tmp_direction

turnover

volume

zhongshu

zhongshu_change

zvt.factors.z.domain.stock_1wk_z_factor

Classes

`Stock1wkZFactor`(**kwargs)

Stock1wkZFactor

class zvt.factors.z.domain.stock_1wk_z_factor.Stock1wkZFactor(**kwargs)

Bases: **Base**, **ZFactorCommon**

__init__(**kwargs)

A simple constructor that allows initialization from kwargs.

Sets attributes on the constructed instance using the names and values in `kwargs`.

Only keys that are present as attributes of the instance's class are allowed. These could be, for example, any mapped columns or relationships.

bi_change

bi_di

bi_ding

bi_interval

bi_slope

bi_value

bi_zhongshu

bi_zhongshu_change

close

current_change

current_direction

current_interval

current_slope

current_zhongshu_change

current_zhongshu_y0

current_zhongshu_y1

duan_change

duan_di

duan_ding

duan_interval

duan_slope

duan_state

duan_value

entity_id

entity id

fenxing_power

classmethod get_providers() → List[str]

providers of the schema defined by cls

Returns:: providers

classmethod get_storages(*provider: Optional[str] = None*)

get the storages info

Parameters:: **provider** – provider

Returns:: storages

high

id

id

level

low

open

opposite_change

opposite_interval

opposite_slope

classmethod query_data(*provider_index: int = 0*, *ids: Optional[List[str]] = None*, *entity_ids: Optional[List[str]] = None*, *entity_id: Optional[str] = None*, *codes: Optional[List[str]] = None*, *code: Optional[str] = None*, *level: Optional[Union[IntervalLevel, str]] = None*, *provider: Optional[str] = None*, *columns: Optional[List] = None*, *col_label: Optional[dict] = None*, *return_type: str = 'df'*, *start_timestamp: Optional[Union[Timestamp, str]] = None*, *end_timestamp: Optional[Union[Timestamp, str]] = None*, *filters: Optional[List] = None*, *session: Optional[Session] = None*, *order=None*, *limit: Optional[int] = None*, *distinct=None*, *index: Optional[Union[str, list]] = None*, *drop_index_col=False*, *time_field: str = 'timestamp'*)

query data by the arguments

Parameters::

- **provider_index** –
- **data_schema** –
- **ids** –
- **entity_ids** –
- **entity_id** –
- **codes** –
- **code** –
- **level** –
- **provider** –
- **columns** –
- **col_label** – dict with key(column), value(label
- **return_type** – df, domain or dict. default is df
- **start_timestamp** –
- **end_timestamp** –
- **filters** –
- **session** –
- **order** –
- **limit** –
- **index** – index field name, str for single index, str list for multiple index
- **drop_index_col** – whether drop the col if it's in index, default False
- **time_field** –

Returns:: results basing on return_type.

classmethod record_data(*provider_index: int = 0*, *provider: Optional[str] = None*, *force_update=None*, *sleeping_time=None*, *exchanges=None*, *entity_id=None*, *entity_ids=None*, *code=None*, *codes=None*, *real_time=None*, *fix_duplicate_way=None*, *start_timestamp=None*, *end_timestamp=None*, *one_day_trading_minutes=None*, ***kwargs*)

record data by the arguments

Parameters::

- **entity_id** –
- **provider_index** –
- **provider** –
- **force_update** –
- **sleeping_time** –
- **exchanges** –
- **entity_ids** –
- **code** –
- **codes** –
- **real_time** –
- **fix_duplicate_way** –
- **start_timestamp** –
- **end_timestamp** –
- **one_day_trading_minutes** –
- **kwargs** –

Returns::

classmethod register_provider(*provider*)

register the provider to the schema defined by cls

Parameters:: **provider** –

classmethod register_recorder_cls(*provider*, *recorder_cls*)

register the recorder for the schema

Parameters::

- **provider** –
- **recorder_cls** –

timestamp

the meaning could be different for different case,most time it means 'happen time'

tmp_di

tmp_ding

tmp_direction

turnover

volume

zhongshu

zhongshu_change

zvt.factors.z.z_factor

Functions

`get_z_factor_schema`(entity_type[, level])

Classes

`ZAccumulator`([acc_window])

`ZFactor`(entity_schema, provider, ...)

`ZState`([state])

---

zvt.factors.z.z_factor.get_z_factor_schema

zvt.factors.z.z_factor.get_z_factor_schema(*entity_type: str*, *level: Union[IntervalLevel, str] = IntervalLevel.LEVEL_1DAY*)

ZAccumulator

class zvt.factors.z.z_factor.ZAccumulator(*acc_window: int = 1*)

Bases: **Accumulator**

__init__(*acc_window: int = 1*) → None

算法和概念 <实体> 某种状态的k线 [实体] 连续实体排列

两k线的关系有三种: 上涨，下跌，包含 上涨: k线高点比之前高，低点比之前高 下跌: k线低点比之前低，高点比之前低 包含: k线高点比之前高，低点比之前低;反方向，即被包含 处理包含关系，长的k线缩短，上涨时，低点取max(low1,low2)；下跌时，高点取min(high1,high2)

第一个顶(底)分型: 出现连续4根下跌(上涨)k线 之后开始寻找 候选底(顶)分型，寻找的过程中有以下状态

<临时顶>: 中间k线比两边的高点高,是一条特定的k线 <临时底>: 中间k线比两边的高点高,是一条特定的k线

<候选顶分型>: 连续的<临时顶>取最大 <候选底分型>: 连续的<临时底>取最小 任何时刻只能有一个候选，其之前是一个确定的分型

<上升k线>: <下降k线>: <连接k线>: 分型之间的k线都可以认为是连接k线，以上为演化过程的中间态 distance(<候选顶分型>, <连接k线>)>=4 则 <候选顶分型> 变成顶分型 distance(<候选底分型>, <连接k线>)>=4 则 <候选底分型> 变成底分型

<顶分型><连接k线><候选底分型> <底分型><连接k线><候选顶分型>

acc_one(*entity_id*, *df: ~pandas.core.frame.DataFrame*, *acc_df: ~pandas.core.frame.DataFrame*, *state: dict) -> (<class 'pandas.core.frame.DataFrame'>*, *<class 'dict'>*)

df format:

```
                col1    col2    col3    ...
timestamp
                1.2     0.5     0.3     ...
                1.0     0.7     0.2     ...
```

the new result and state

Parameters::
- **df** – current input df
- **entity_id** – current computing entity_id
- **acc_df** – current result of the entity_id
- **state** – current state of the entity_id

Returns:: new result and state of the entity_id

acc(*input_df: ~pandas.core.frame.DataFrame*, *acc_df: ~pandas.core.frame.DataFrame*, *states: dict) -> (<class 'pandas.core.frame.DataFrame'>*, *<class 'dict'>*)

Parameters::
- **input_df** – new input
- **acc_df** – previous result
- **states** – current states of the entity

Returns:: new result and states

ZFactor

class zvt.factors.z.z_factor.ZFactor(*entity_schema: ~typing.Type[~zvt.contract.schema.TradableEntity] = <class 'zvt.domain.meta.stock_meta.Stock'>*, *provider: ~typing.Optional[str] = None*, *entity_provider: ~typing.Optional[str] = None*, *entity_ids: ~typing.Optional[~typing.List[str]] = None*, *exchanges: ~typing.Optional[~typing.List[str]] = None*, *codes: ~typing.Optional[~typing.List[str]] = None*, *start_timestamp: ~typing.Optional[~typing.Union[~pandas._libs.tslibs.timestamps.Timestamp, str]] = None*, *end_timestamp: ~typing.Optional[~typing.Union[~pandas._libs.tslibs.timestamps.Timestamp, str]] = None*, *columns: ~typing.Optional[~typing.List] = None*, *filters: ~typing.Optional[~typing.List] = None*, *order: ~typing.Optional[object] = None*, *limit: ~typing.Optional[int] = None*, *level: ~typing.Union[str, ~zvt.contract.IntervalLevel] = IntervalLevel.LEVEL_1DAY*, *category_field: str = 'entity_id'*, *time_field: str = 'timestamp'*, *keep_window: ~typing.Optional[int] = None*, *keep_all_timestamp: bool = False*, *fill_method: str = 'ffill'*, *effective_number: ~typing.Optional[int] = None*, *transformer: ~typing.Optional[~zvt.contract.factor.Transformer] = None*, *accumulator: ~typing.Optional[~zvt.contract.factor.Accumulator] = None*, *need_persist: bool = False*, *only_compute_factor: bool = False*, *factor_name: ~typing.Optional[str] = None*, *clear_state: bool = False*, *only_load_factor: bool = False*, *adjust_type: ~typing.Optional[~typing.Union[~zvt.contract.AdjustType, str]] = None*)

Bases: **TechnicalFactor**

__init__(*entity_schema: ~typing.Type[~zvt.contract.schema.TradableEntity] = <class 'zvt.domain.meta.stock_meta.Stock'>*, *provider: ~typing.Optional[str] = None*, *entity_provider: ~typing.Optional[str] = None*, *entity_ids: ~typing.Optional[~typing.List[str]] = None*, *exchanges: ~typing.Optional[~typing.List[str]] = None*, *codes: ~typing.Optional[~typing.List[str]] = None*, *start_timestamp: ~typing.Optional[~typing.Union[~pandas._libs.tslibs.timestamps.Timesta*

mp, str]] = None, end_timestamp: ~typing.Optional[~typing.Union[~pandas._libs.tslibs.timestamps.Timestamp, str]] = None, columns: ~typing.Optional[~typing.List] = None, filters: ~typing.Optional[~typing.List] = None, order: ~typing.Optional[object] = None, limit: ~typing.Optional[int] = None, level: ~typing.Union[str, ~zvt.contract.IntervalLevel] = IntervalLevel.LEVEL_1DAY, category_field: str = 'entity_id', time_field: str = 'timestamp', keep_window: ~typing.Optional[int] = None, keep_all_timestamp: bool = False, fill_method: str = 'ffill', effective_number: ~typing.Optional[int] = None, transformer: ~typing.Optional[~zvt.contract.factor.Transformer] = None, accumulator: ~typing.Optional[~zvt.contract.factor.Accumulator] = None, need_persist: bool = False, only_compute_factor: bool = False, factor_name: ~typing.Optional[str] = None, clear_state: bool = False, only_load_factor: bool = False, adjust_type: ~typing.Optional[~typing.Union[~zvt.contract.AdjustType, str]] = None) → None

Parameters::

- **keep_all_timestamp** –
- **fill_method** –
- **effective_number** –
- **transformer** –
- **accumulator** –
- **need_persist** – whether persist factor
- **only_compute_factor** – only compute factor nor result
- **factor_name** –
- **clear_state** –
- **only_load_factor** – only load factor and compute result

accumulator*: Accumulator = <zvt.factors.z.z_factor.ZAccumulator object>*

accumulator for this factor if not passed as __init__ argument

factor_schema*: Type[Mixin] = None*

define the schema for persist,its columns should be same as indicators in transformer or accumulator

factor_col_map_object_hook() → dict

:return:{col:object_hook}

clear_state_data(*entity_id=None*)

clear state of the entity

Parameters:: **entity_id** – entity id

decode_state(*state: str*)

decode state

Parameters:: **state** –

Returns::

encode_state(*state: object*)

encode state

Parameters:: **state** –

Returns::

move_on(*to_timestamp: Optional[Union[Timestamp, str]] = None*, *timeout: int = 20*) → object

using continual fetching data in realtime 1)get the data happened before to_timestamp,if not set,get all the data which means to now 2)if computing_window set,the data_df would be cut for saving memory

Parameters::

- **to_timestamp** –
- **timeout** (*int*) –

Returns::

Return type::

name = *None*

name of the service, default name of class if not set manually

on_data_changed(*data: DataFrame*)

overwrite it for computing after data added

Parameters:: **data** –

on_data_loaded(*data: DataFrame*)

Parameters:: **data** –

on_entity_data_changed(*entity*, *added_data: DataFrame*)

overwrite it for computing after entity data added

Parameters::
- **entity** –
- **added_data** –

state_schema

alias of `FactorState`

transformer*: Transformer = None*

transformer for this factor if not passed as __init__ argument

pipe_df*: pd.DataFrame*

中间结果，不持久化 data_df->pipe_df

factor_df*: pd.DataFrame*

计算因子的结果，可持久化,通过对pipe_df的计算得到 pipe_df->factor_df

result_df*: pd.DataFrame*

result_df是用于选股的标准df,通过对factor_df的计算得到 factor_df->result_df

states*: dict*

entity_id:state

ZState

class zvt.factors.z.z_factor.ZState(*state: Optional[dict] = None*)

Bases: `Bean`

__init__(*state: Optional[dict] = None*) → None

zvt.fill_project

Functions

`gen_kdata_schemas()` generate kdata(OHLC) schemas for tradable entity

---

zvt.fill_project.gen_kdata_schemas

zvt.fill_project.gen_kdata_schemas()

generate kdata(OHLC) schemas for tradable entity

zvt.informer

Modules

zvt.informer.inform_utils

---

zvt.informer.informer

---

zvt.informer.inform_utils

Functions

`add_to_eastmoney`(codes, group[, ...])

`clean_groups`(keep)

`inform_email`(entity_ids, entity_type, ...)

---

zvt.informer.inform_utils.add_to_eastmoney

zvt.informer.inform_utils.add_to_eastmoney(*codes*, *group*, *entity_type='stock'*, *over_write=True*)

zvt.informer.inform_utils.clean_groups

zvt.informer.inform_utils.clean_groups(*keep*)

zvt.informer.inform_utils.inform_email

zvt.informer.inform_utils.inform_email(*entity_ids*, *entity_type*, *target_date*, *title*, *provider*)

zvt.informer.informer

Classes

`EmailInformer`([ssl])

`Informer`()

`WechatInformer`()

EmailInformer

class zvt.informer.informer.EmailInformer(*ssl=True*)

Bases: `Informer`

__init__(*ssl=True*) → None

Informer

class zvt.informer.informer.Informer

Bases: `object`

__init__()

WechatInformer

class zvt.informer.informer.WechatInformer

Bases: `Informer`

__init__() → None

zvt.main

Functions

`main`()

`serve_layout`()

`tab_content`(active_tab)

---

zvt.main.main

zvt.main.main()

zvt.main.serve_layout

zvt.main.serve_layout()

zvt.main.tab_content

zvt.main.tab_content(*active_tab*)

zvt.ml

Modules

zvt.ml.lables

---

zvt.ml.ml

---

zvt.ml.lables

Classes

`BehaviorCategory`(value)	An enumeration.
`RelativePerformance`(value)	An enumeration.

BehaviorCategory

class zvt.ml.lables.BehaviorCategory(*value*)

Bases: `Enum`

An enumeration.

__init__()

RelativePerformance

class zvt.ml.lables.RelativePerformance(*value*)

Bases: **`Enum`**

An enumeration.

__init__()

zvt.ml.ml

Functions

`cal_behavior_cls`(s, predict_range)

`cal_change`(s, predict_range)

`cal_predict`(s, predict_range)

`cal_relative_performance`(s)

Classes

`MLMachine`([entity_ids, start_timestamp, ...])

`MaStockMLMachine`([entity_ids, ...])

`StockMLMachine`([entity_ids, ...])

---

zvt.ml.ml.cal_behavior_cls

zvt.ml.ml.cal_behavior_cls(*s: Series*, *predict_range*)

zvt.ml.ml.cal_change

zvt.ml.ml.cal_change(*s: Series*, *predict_range*)

zvt.ml.ml.cal_predict

zvt.ml.ml.cal_predict(*s: Series*, *predict_range*)

zvt.ml.ml.cal_relative_performance

zvt.ml.ml.cal_relative_performance(*s: Series*)

MLMachine

class zvt.ml.ml.MLMachine(*entity_ids: Optional[List[str]] = None*, *start_timestamp: Union[str, Timestamp] = '2015-01-01'*, *end_timestamp: Union[str, Timestamp] = '2021-12-01'*, *predict_start_timestamp: Union[str, Timestamp] = '2021-06-01'*, *predict_steps: int = 20*, *level: Union[IntervalLevel, str] = IntervalLevel.LEVEL_1DAY*, *adjust_type: Optional[Union[AdjustType, str]] = None*, *data_provider: Optional[str] = None*, *label_method: str = 'raw'*)

Bases: `object`

__init__(*entity_ids: Optional[List[str]] = None*, *start_timestamp: Union[str, Timestamp] = '2015-01-01'*, *end_timestamp: Union[str, Timestamp] = '2021-12-01'*, *predict_start_timestamp: Union[str, Timestamp] = '2021-06-01'*, *predict_steps: int = 20*, *level: Union[IntervalLevel, str] = IntervalLevel.LEVEL_1DAY*, *adjust_type: Optional[Union[AdjustType, str]] = None*, *data_provider: Optional[str] = None*, *label_method: str = 'raw'*) → None

Parameters::

- **entity_ids** –
- **start_timestamp** –
- **end_timestamp** –
- **predict_start_timestamp** –
- **predict_steps** –
- **level** –
- **adjust_type** –
- **data_provider** –
- **label_method** – raw, change, or behavior_cls

build_feature(*entity_ids: List[str]*, *start_timestamp: Timestamp*, *end_timestamp: Timestamp*) → DataFrame

result df format

col1 col2 col3 …

entity_id timestamp

1.2 0.5 0.3 … 1.0 0.7 0.2 …

Parameters::

- **entity_ids** – entity id list
- **start_timestamp** –
- **end_timestamp** –

Return type:: pd.DataFrame

MaStockMLMachine

class zvt.ml.ml.MaStockMLMachine(*entity_ids: Optional[List[str]] = None*, *start_timestamp: Union[str, Timestamp] = '2015-01-01'*, *end_timestamp: Union[str, Timestamp] = '2021-12-01'*, *predict_start_timestamp: Union[str, Timestamp] = '2021-06-01'*, *predict_steps: int = 20*, *level: Union[IntervalLevel, str] = IntervalLevel.LEVEL_1DAY*, *adjust_type: Optional[Union[AdjustType, str]] = None*, *data_provider: Optional[str] = None*, *label_method: str = 'raw'*)

Bases: `StockMLMachine`

__init__(*entity_ids: Optional[List[str]] = None*, *start_timestamp: Union[str, Timestamp] = '2015-01-01'*, *end_timestamp: Union[str, Timestamp] = '2021-12-01'*, *predict_start_timestamp: Union[str, Timestamp] = '2021-06-01'*, *predict_steps: int = 20*, *level: Union[IntervalLevel, str] = IntervalLevel.LEVEL_1DAY*, *adjust_type: Optional[Union[AdjustType, str]] = None*, *data_provider: Optional[str] = None*, *label_method: str = 'raw'*) → None

Parameters::

- **entity_ids** –
- **start_timestamp** –
- **end_timestamp** –
- **predict_start_timestamp** –
- **predict_steps** –
- **level** –
- **adjust_type** –
- **data_provider** –
- **label_method** – raw, change, or behavior_cls

build_feature(*entity_ids: List[str]*, *start_timestamp: Timestamp*, *end_timestamp: Timestamp*) → DataFrame

Parameters::

- **entity_ids** –
- **start_timestamp** –
- **end_timestamp** –

Returns::

entity_schema

alias of `Stock`

StockMLMachine

class zvt.ml.ml.StockMLMachine(*entity_ids: Optional[List[str]] = None*, *start_timestamp: Union[str, Timestamp] = '2015-01-01'*, *end_timestamp: Union[str, Timestamp] = '2021-12-01'*, *predict_start_timestamp: Union[str, Timestamp] = '2021-06-01'*, *predict_steps: int = 20*, *level: Union[IntervalLevel, str] = IntervalLevel.LEVEL_1DAY*, *adjust_type: Optional[Union[AdjustType, str]] = None*, *data_provider: Optional[str] = None*, *label_method: str = 'raw'*)

Bases: `MLMachine`

__init__(*entity_ids: Optional[List[str]] = None*, *start_timestamp: Union[str, Timestamp] = '2015-01-01'*, *end_timestamp: Union[str, Timestamp] = '2021-12-01'*, *predict_start_timestamp: Union[str, Timestamp] = '2021-06-01'*, *predict_steps: int = 20*, *level: Union[IntervalLevel, str] = IntervalLevel.LEVEL_1DAY*, *adjust_type: Optional[Union[AdjustType, str]] = None*, *data_provider: Optional[str] = None*, *label_method: str = 'raw'*) → None

Parameters::

- **entity_ids** –
- **start_timestamp** –
- **end_timestamp** –
- **predict_start_timestamp** –
- **predict_steps** –
- **level** –
- **adjust_type** –
- **data_provider** –
- **label_method** – raw, change, or behavior_cls

entity_schema

alias of `Stock`

build_feature(*entity_ids: List[str]*, *start_timestamp: Timestamp*, *end_timestamp: Timestamp*) → DataFrame

result df format

col1 col2 col3 …

entity_id timestamp
1.2 0.5 0.3 … 1.0 0.7 0.2 …

Parameters::
- **entity_ids** – entity id list
- **start_timestamp** –
- **end_timestamp** –

Return type:: pd.DataFrame

zvt.plugin

Functions

export()

---

main()

---

zvt.plugin.export

zvt.plugin.export()

zvt.plugin.main

zvt.plugin.main()

zvt.recorders

Functions

init_main_index([provider])

Modules

zvt.recorders.consts

---

zvt.recorders.eastmoney

---

zvt.recorders.em

---

zvt.recorders.exchange

---

zvt.recorders.joinquant

---

zvt.recorders.jqka

---

zvt.recorders.qmt

---

zvt.recorders.sina

---

zvt.recorders.wb

zvt.recorders.init_main_index

zvt.recorders.init_main_index(*provider='exchange'*)

zvt.recorders.consts

zvt.recorders.eastmoney

Modules

zvt.recorders.eastmoney.common

---

zvt.recorders.eastmoney.dividend_financing

---

zvt.recorders.eastmoney.finance

---

zvt.recorders.eastmoney.holder

---

zvt.recorders.eastmoney.meta

---

zvt.recorders.eastmoney.trading

---

zvt.recorders.eastmoney.common

Functions

call_eastmoney_api([url, method, param, ...])

---

company_type_flag(security_item)

---

get_company_type(stock_domain)

---

get_fc(security_item)

---

get_from_path_fields(the_json, path_fields)

Classes

ApiWrapper()

BaseEastmoneyRecorder()

EastmoneyApiWrapper()

EastmoneyMoreDataRecorder([force_update, ...])

EastmoneyPageabeDataRecorder([force_update, ...])

EastmoneyTimestampsDataRecorder([...])

zvt.recorders.eastmoney.common.call_eastmoney_api

zvt.recorders.eastmoney.common.call_eastmoney_api(*url=None*, *method='post'*, *param=None*, *path_fields=None*)

zvt.recorders.eastmoney.common.company_type_flag

zvt.recorders.eastmoney.common.company_type_flag(*security_item*)

zvt.recorders.eastmoney.common.get_company_type

zvt.recorders.eastmoney.common.get_company_type(*stock_domain: StockDetail*)

zvt.recorders.eastmoney.common.get_fc

zvt.recorders.eastmoney.common.get_fc(*security_item*)

zvt.recorders.eastmoney.common.get_from_path_fields

zvt.recorders.eastmoney.common.get_from_path_fields(*the_json*, *path_fields*)

ApiWrapper

class zvt.recorders.eastmoney.common.ApiWrapper

Bases: `object`

__init__()

BaseEastmoneyRecorder

class zvt.recorders.eastmoney.common.BaseEastmoneyRecorder

Bases: `object`

__init__()

EastmoneyApiWrapper

class zvt.recorders.eastmoney.common.EastmoneyApiWrapper

Bases: `ApiWrapper`

__init__()

EastmoneyMoreDataRecorder

class zvt.recorders.eastmoney.common.EastmoneyMoreDataRecorder(*force_update=False*, *sleeping_time=5*, *exchanges=None*, *entity_id=None*, *entity_ids=None*, *code=None*, *codes=None*, *day_data=False*, *entity_filters=None*, *ignore_failed=True*, *real_time=False*, *fix_duplicate_way='add'*, *start_timestamp=None*, *end_timestamp=None*, *return_unfinished=False*)

Bases: `BaseEastmoneyRecorder`, `TimeSeriesDataRecorder`

__init__(*force_update=False*, *sleeping_time=5*, *exchanges=None*, *entity_id=None*, *entity_ids=None*, *code=None*, *codes=None*, *day_data=False*, *entity_filters=None*, *ignore_failed=True*, *real_time=False*, *fix_duplicate_way='add'*, *start_timestamp=None*, *end_timestamp=None*, *return_unfinished=False*) → None

entity_provider*: str = 'eastmoney'*

overwrite them to fetch the entity list

entity_schema

alias of `StockDetail`

provider*: str = 'eastmoney'*

overwrite them to set up the data you want to record

clear_state_data(*entity_id=None*)

clear state of the entity

Parameters:: **entity_id** – entity id

decode_state(*state: str*)

decode state

Parameters:: **state** –

Returns::

encode_state(*state: object*)

encode state

Parameters:: **state** –

Returns::

generate_domain(*entity*, *original_data*)

generate the data_schema instance using entity and original_data,the original_data is from record result

Parameters::
- **entity** –
- **original_data** –

generate_domain_id(*entity*, *original_data*, *time_fmt='YYYY-MM-DD'*)

generate domain id from the entity and original data,the default id meaning:entity + event happen time

Parameters::
- **entity** –
- **original_data** –
- **time_fmt** –

Returns::

Return type::

get_data_map()

{‘original_field’:(‘domain_field’,transform_func)}

get_evaluated_time_field()

the timestamp field for evaluating time range of recorder,used in get_latest_saved_record

init_entities()

init the entities which we would record data for

name = *None*

name of the service, default name of class if not set manually

original_page_url = *None*

original page url

persist(*entity*, *domain_list*)

persist the domain list to db

Parameters::

- **entity** –
- **domain_list** –

record(*entity_item*, *start*, *end*, *size*, *timestamps*)

implement the recording logic in this method, should return json or domain list

Parameters::

- **entity** –
- **start** –
- **end** –
- **size** –
- **timestamps** –

state_schema

alias of `RecorderState`

url = *None*

request url

exchanges

setup the entities you want to record

entity_ids

set entity_ids or (entity_type,exchanges,codes)

session

using to do db operations

EastmoneyPageabeDataRecorder

class zvt.recorders.eastmoney.common.EastmoneyPageabeDataRecorder(*force_update=False*, *sleeping_time=5*, *exchanges=None*, *entity_id=None*, *entity_ids=None*, *code=None*, *codes=None*, *day_data=False*, *entity_filters=None*, *ignore_failed=True*, *real_time=False*, *fix_duplicate_way='add'*, *start_timestamp=None*, *end_timestamp=None*, *return_unfinished=False*)

Bases: `BaseEastmoneyRecorder`, `TimeSeriesDataRecorder`

__init__(*force_update=False*, *sleeping_time=5*, *exchanges=None*, *entity_id=None*, *entity_ids=None*, *code=None*, *codes=None*, *day_data=False*, *entity_filters=None*, *ignore_failed=True*, *real_time=False*, *fix_duplicate_way='add'*, *start_timestamp=None*, *end_timestamp=None*, *return_unfinished=False*) → None

entity_provider*: str = 'eastmoney'*

overwrite them to fetch the entity list

entity_schema

alias of `StockDetail`

provider*: str = 'eastmoney'*

overwrite them to set up the data you want to record

clear_state_data(*entity_id=None*)

clear state of the entity

Parameters:: **entity_id** – entity id

decode_state(*state: str*)

decode state

Parameters:: **state** –

Returns::

encode_state(*state: object*)

encode state

Parameters:: **state** –

Returns::

generate_domain(*entity*, *original_data*)

generate the data_schema instance using entity and original_data,the original_data is from record result

Parameters::
- **entity** –
- **original_data** –

generate_domain_id(*entity*, *original_data*, *time_fmt='YYYY-MM-DD'*)

generate domain id from the entity and original data,the default id meaning:entity + event happen time

Parameters::
- **entity** –
- **original_data** –
- **time_fmt** –

Returns::

Return type::

get_data_map()

{‘original_field’:(‘domain_field’,transform_func)}

get_evaluated_time_field()

the timestamp field for evaluating time range of recorder,used in get_latest_saved_record

init_entities()

init the entities which we would record data for

name = *None*

name of the service, default name of class if not set manually

original_page_url = *None*

original page url

persist(*entity*, *domain_list*)

persist the domain list to db

Parameters::

- **entity** –
- **domain_list** –

record(*entity_item*, *start*, *end*, *size*, *timestamps*)

implement the recording logic in this method, should return json or domain list

Parameters::

- **entity** –
- **start** –
- **end** –
- **size** –
- **timestamps** –

state_schema

alias of `RecorderState`

url = *None*

request url

exchanges

setup the entities you want to record

entity_ids

set entity_ids or (entity_type,exchanges,codes)

session

using to do db operations

EastmoneyTimestampsDataRecorder

class zvt.recorders.eastmoney.common.EastmoneyTimestampsDataRecorder(*force_update=False*, *sleeping_time=5*, *exchanges=None*, *entity_id=None*, *entity_ids=None*, *code=None*, *codes=None*, *day_data=False*, *entity_filters=None*, *ignore_failed=True*, *real_time=False*, *fix_duplicate_way='add'*, *start_timestamp=None*, *end_timestamp=None*)

Bases: `BaseEastmoneyRecorder`, `TimestampsDataRecorder`

__init__(*force_update=False*, *sleeping_time=5*, *exchanges=None*, *entity_id=None*, *entity_ids=None*, *code=None*, *codes=None*, *day_data=False*, *entity_filters=None*, *ignore_failed=True*, *real_time=False*, *fix_duplicate_way='add'*, *start_timestamp=None*, *end_timestamp=None*) → None

entity_provider*: str = 'eastmoney'*

overwrite them to fetch the entity list

entity_schema

alias of `StockDetail`

provider*: str = 'eastmoney'*

overwrite them to set up the data you want to record

clear_state_data(*entity_id=None*)

clear state of the entity

Parameters:: **entity_id** – entity id

decode_state(*state: str*)

decode state

Parameters:: **state** –
Returns::

encode_state(*state: object*)

encode state

Parameters:: **state** –
Returns::

generate_domain(*entity*, *original_data*)

generate the data_schema instance using entity and original_data,the original_data is from record result

Parameters::
- **entity** –
- **original_data** –

generate_domain_id(*entity*, *original_data*, *time_fmt='YYYY-MM-DD'*)

generate domain id from the entity and original data,the default id meaning:entity + event happen time

Parameters::
- **entity** –
- **original_data** –
- **time_fmt** –

Returns::
Return type::

get_data_map()

{‘original_field’:(‘domain_field’,transform_func)}

get_evaluated_time_field()

the timestamp field for evaluating time range of recorder,used in get_latest_saved_record

init_entities()

init the entities which we would record data for

name = *None*

name of the service, default name of class if not set manually

original_page_url = *None*

original page url

persist(*entity*, *domain_list*)

persist the domain list to db

Parameters::

- **entity** –
- **domain_list** –

record(*entity_item*, *start*, *end*, *size*, *timestamps*)

implement the recording logic in this method, should return json or domain list

Parameters::

- **entity** –
- **start** –
- **end** –
- **size** –
- **timestamps** –

state_schema

alias of `RecorderState`

url = *None*

request url

exchanges

setup the entities you want to record

entity_ids

set entity_ids or (entity_type,exchanges,codes)

session

using to do db operations

zvt.recorders.eastmoney.dividend_financing

Modules

zvt.recorders.eastmoney.dividend_financing.eastmoney_dividend_detail_recorder

---

zvt.recorders.eastmoney.dividend_financing.eastmoney_dividend_financing_recorder

---

zvt.recorders.eastmoney.dividend_financing.eastmoney_rights_issue_detail_recorder

---

zvt.recorders.eastmoney.dividend_financing.eastmoney_spo_detail_recorder

---

zvt.recorders.eastmoney.dividend_financing.eastmoney_dividend_detail_recorder

Classes

`DividendDetailRecorder`([force_update, ...])

DividendDetailRecorder

class zvt.recorders.eastmoney.dividend_financing.eastmoney_dividend_detail_recorder.DividendDetailRecorder(*force_update=False*, *sleeping_time=5*, *exchanges=None*, *entity_id=None*, *entity_ids=None*, *code=None*, *codes=None*, *day_data=False*, *entity_filters=None*, *ignore_failed=True*, *real_time=False*, *fix_duplicate_way='add'*, *start_timestamp=None*, *end_timestamp=None*, *return_unfinished=False*)

Bases: `EastmoneyPageabeDataRecorder`

__init__(*force_update=False*, *sleeping_time=5*, *exchanges=None*, *entity_id=None*, *entity_ids=None*, *code=None*, *codes=None*, *day_data=False*, *entity_filters=None*, *ignore_failed=True*, *real_time=False*, *fix_duplicate_way='add'*, *start_timestamp=None*, *end_timestamp=None*, *return_unfinished=False*) → None

data_schema

alias of `DividendDetail`

url = *'https://emh5.eastmoney.com/api/FenHongRongZi/GetFenHongSongZhuanList'*

request url

get_data_map()

{‘original_field’:(‘domain_field’,transform_func)}

clear_state_data(*entity_id=None*)

clear state of the entity

Parameters:: **entity_id** – entity id

decode_state(*state: str*)

decode state

Parameters:: **state** –
Returns::

encode_state(*state: object*)

encode state

Parameters:: **state** –
Returns::

entity_provider*: str = 'eastmoney'*

overwrite them to fetch the entity list

entity_schema

alias of `StockDetail`

generate_domain(*entity*, *original_data*)

generate the data_schema instance using entity and original_data,the original_data is from record result

Parameters::
- **entity** –
- **original_data** –

generate_domain_id(*entity*, *original_data*, *time_fmt='YYYY-MM-DD'*)

generate domain id from the entity and original data,the default id meaning:entity + event happen time

Parameters::
- **entity** –
- **original_data** –
- **time_fmt** –

Returns::
Return type::

get_evaluated_time_field()

the timestamp field for evaluating time range of recorder,used in get_latest_saved_record

init_entities()

init the entities which we would record data for

name = *None*

name of the service, default name of class if not set manually

original_page_url = *None*

original page url

persist(*entity*, *domain_list*)

persist the domain list to db

Parameters::

- **entity** –
- **domain_list** –

provider*: str = 'eastmoney'*

overwrite them to set up the data you want to record

record(*entity_item*, *start*, *end*, *size*, *timestamps*)

implement the recording logic in this method, should return json or domain list

Parameters::

- **entity** –
- **start** –
- **end** –
- **size** –
- **timestamps** –

state_schema

alias of `RecorderState`

exchanges

setup the entities you want to record

entity_ids

set entity_ids or (entity_type,exchanges,codes)

session

using to do db operations

zvt.recorders.eastmoney.dividend_financing.eastmoney_dividend_financing_recorder

Classes

`DividendFinancingRecorder`([force_update, ...])

DividendFinancingRecorder

class zvt.recorders.eastmoney.dividend_financing.eastmoney_dividend_financing_recorder.DividendFinancingRecorder(*force_update=False*, *sleeping_time=5*, *exchanges=None*, *entity_id=None*, *entity_ids=None*, *code=None*, *codes=None*, *day_data=False*, *entity_filters=None*, *ignore_failed=True*, *real_time=False*, *fix_duplicate_way='add'*, *start_timestamp=None*, *end_timestamp=None*, *return_unfinished=False*)

Bases: `EastmoneyPageabeDataRecorder`

__init__(*force_update=False*, *sleeping_time=5*, *exchanges=None*, *entity_id=None*, *entity_ids=None*, *code=None*, *codes=None*, *day_data=False*, *entity_filters=None*, *ignore_failed=True*, *real_time=False*, *fix_duplicate_way='add'*, *start_timestamp=None*, *end_timestamp=None*, *return_unfinished=False*) → None

data_schema

alias of `DividendFinancing`

url = *'https://emh5.eastmoney.com/api/FenHongRongZi/GetLiNianFenHongRongZiList'*

request url

get_data_map()

{‘original_field’:(‘domain_field’,transform_func)}

clear_state_data(*entity_id=None*)

clear state of the entity

Parameters:: **entity_id** – entity id

decode_state(*state: str*)

decode state

Parameters:: **state** –
Returns::

encode_state(*state: object*)

encode state

Parameters:: **state** –
Returns::

entity_provider*: str = 'eastmoney'*

overwrite them to fetch the entity list

entity_schema

alias of `StockDetail`

generate_domain(*entity*, *original_data*)

generate the data_schema instance using entity and original_data,the original_data is from record result

Parameters::
- **entity** –
- **original_data** –

generate_domain_id(*entity*, *original_data*, *time_fmt='YYYY-MM-DD'*)

generate domain id from the entity and original data,the default id meaning:entity + event happen time

Parameters::
- **entity** –
- **original_data** –
- **time_fmt** –

Returns::
Return type::

get_evaluated_time_field()

the timestamp field for evaluating time range of recorder,used in get_latest_saved_record

init_entities()

init the entities which we would record data for

name = *None*

name of the service, default name of class if not set manually

original_page_url = *None*

original page url

persist(*entity*, *domain_list*)

persist the domain list to db

Parameters::

- **entity** –
- **domain_list** –

provider*: str = 'eastmoney'*

overwrite them to set up the data you want to record

record(*entity_item*, *start*, *end*, *size*, *timestamps*)

implement the recording logic in this method, should return json or domain list

Parameters::

- **entity** –
- **start** –
- **end** –
- **size** –
- **timestamps** –

state_schema

alias of `RecorderState`

exchanges

setup the entities you want to record

entity_ids

set entity_ids or (entity_type,exchanges,codes)

session

using to do db operations

zvt.recorders.eastmoney.dividend_financing.eastmoney_rights_issue_detail_recorder

Classes

`RightsIssueDetailRecorder`([force_update, ...])

RightsIssueDetailRecorder

class zvt.recorders.eastmoney.dividend_financing.eastmoney_rights_issue_detail_recorder.RightsIssueDetailRecorder(*force_update=False*, *sleeping_time=5*, *exchanges=None*, *entity_id=None*, *entity_ids=None*, *code=None*, *codes=None*, *day_data=False*, *entity_filters=None*, *ignore_failed=True*, *real_time=False*, *fix_duplicate_way='add'*, *start_timestamp=None*, *end_timestamp=None*, *return_unfinished=False*)

Bases: `EastmoneyPageabeDataRecorder`

__init__(*force_update=False*, *sleeping_time=5*, *exchanges=None*, *entity_id=None*, *entity_ids=None*, *code=None*, *codes=None*, *day_data=False*, *entity_filters=None*, *ignore_failed=True*, *real_time=False*, *fix_duplicate_way='add'*, *start_timestamp=None*, *end_timestamp=None*, *return_unfinished=False*) → None

data_schema

alias of `RightsIssueDetail`

url = *'https://emh5.eastmoney.com/api/FenHongRongZi/GetPeiGuMingXiList'*

request url

get_data_map()

{'original_field':('domain_field',transform_func)}

clear_state_data(*entity_id=None*)

clear state of the entity

Parameters:: **entity_id** – entity id

decode_state(*state: str*)

decode state

Parameters:: **state** –
Returns::

encode_state(*state: object*)

encode state

Parameters:: **state** –
Returns::

entity_provider*: str = 'eastmoney'*

overwrite them to fetch the entity list

entity_schema

alias of `StockDetail`

generate_domain(*entity*, *original_data*)

generate the data_schema instance using entity and original_data,the original_data is from record result

Parameters::
- **entity** –
- **original_data** –

generate_domain_id(*entity*, *original_data*, *time_fmt='YYYY-MM-DD'*)

generate domain id from the entity and original data,the default id meaning:entity + event happen time

Parameters::
- **entity** –
- **original_data** –
- **time_fmt** –

Returns::
Return type::

get_evaluated_time_field()

the timestamp field for evaluating time range of recorder,used in get_latest_saved_record

init_entities()

init the entities which we would record data for

name = *None*

name of the service, default name of class if not set manually

original_page_url = *None*

original page url

persist(*entity*, *domain_list*)

persist the domain list to db

Parameters::

- **entity** –
- **domain_list** –

provider*: str = 'eastmoney'*

overwrite them to set up the data you want to record

record(*entity_item*, *start*, *end*, *size*, *timestamps*)

implement the recording logic in this method, should return json or domain list

Parameters::

- **entity** –
- **start** –
- **end** –
- **size** –
- **timestamps** –

state_schema

alias of `RecorderState`

exchanges

setup the entities you want to record

entity_ids

set entity_ids or (entity_type,exchanges,codes)

session

using to do db operations

zvt.recorders.eastmoney.dividend_financing.eastmoney_spo_detail_recorder

Classes

`SPODetailRecorder`([force_update, ...])

SPODetailRecorder

class zvt.recorders.eastmoney.dividend_financing.eastmoney_spo_detail_recorder.SPODetailRecorder(*force_update=False*, *sleeping_time=5*, *exchanges=None*, *entity_id=None*, *entity_ids=None*, *code=None*, *codes=None*, *day_data=False*, *entity_filters=None*, *ignore_failed=True*, *real_time=False*, *fix_duplicate_way='add'*, *start_timestamp=None*, *end_timestamp=None*, *return_unfinished=False*)

Bases: `EastmoneyPageabeDataRecorder`

__init__(*force_update=False*, *sleeping_time=5*, *exchanges=None*, *entity_id=None*, *entity_ids=None*, *code=None*, *codes=None*, *day_data=False*, *entity_filters=None*, *ignore_failed=True*, *real_time=False*, *fix_duplicate_way='add'*, *start_timestamp=None*, *end_timestamp=None*, *return_unfinished=False*) → None

data_schema

alias of `SpoDetail`

url = *'https://emh5.eastmoney.com/api/FenHongRongZi/GetZengFaMingXiList'*

request url

get_data_map()

{'original_field':('domain_field',transform_func)}

clear_state_data(*entity_id=None*)

clear state of the entity

Parameters:: **entity_id** – entity id

decode_state(*state: str*)

decode state

Parameters:: **state** –
Returns::

encode_state(*state: object*)

encode state

Parameters:: **state** –
Returns::

entity_provider*: str = 'eastmoney'*

overwrite them to fetch the entity list

entity_schema

alias of `StockDetail`

generate_domain(*entity*, *original_data*)

generate the data_schema instance using entity and original_data,the original_data is from record result

Parameters::
- **entity** –
- **original_data** –

generate_domain_id(*entity*, *original_data*, *time_fmt='YYYY-MM-DD'*)

generate domain id from the entity and original data,the default id meaning:entity + event happen time

Parameters::
- **entity** –
- **original_data** –
- **time_fmt** –

Returns::
Return type::

get_evaluated_time_field()

the timestamp field for evaluating time range of recorder,used in get_latest_saved_record

init_entities()

init the entities which we would record data for

name = *None*

name of the service, default name of class if not set manually

original_page_url = *None*

original page url

persist(*entity*, *domain_list*)

persist the domain list to db

Parameters::

- **entity** –
- **domain_list** –

provider*: str = 'eastmoney'*

overwrite them to set up the data you want to record

record(*entity_item*, *start*, *end*, *size*, *timestamps*)

implement the recording logic in this method, should return json or domain list

Parameters::

- **entity** –
- **start** –
- **end** –
- **size** –
- **timestamps** –

state_schema

alias of `RecorderState`

exchanges

setup the entities you want to record

entity_ids

set entity_ids or (entity_type,exchanges,codes)

session

using to do db operations

zvt.recorders.eastmoney.finance

Modules

zvt.recorders.eastmoney.finance.base_china_stock_finance_recorder

---

zvt.recorders.eastmoney.finance.eastmoney_balance_sheet_recorder

---

zvt.recorders.eastmoney.finance.eastmoney_cash_flow_recorder

---

zvt.recorders.eastmoney.finance.eastmoney_finance_factor_recorder

---

zvt.recorders.eastmoney.finance.eastmoney_income_statement_recorder

---

zvt.recorders.eastmoney.finance.base_china_stock_finance_recorder

Functions

`to_jq_report_period`(timestamp)

Classes

`BaseChinaStockFinanceRecorder`([exchanges, ...])

---

zvt.recorders.eastmoney.finance.base_china_stock_finance_recorder.to_jq_report_period

zvt.recorders.eastmoney.finance.base_china_stock_finance_recorder.to_jq_report_period(*timestamp*)

BaseChinaStockFinanceRecorder

class zvt.recorders.eastmoney.finance.base_china_stock_finance_recorder.BaseChinaStockFinanceRecorder(*exchanges=None*, *entity_id=None*, *entity_ids=None*, *code=None*, *codes=None*, *day_data=False*, *force_update=False*, *sleeping_time=5*, *real_time=False*, *fix_duplicate_way='add'*, *start_timestamp=None*, *end_timestamp=None*)

Bases: `EastmoneyTimestampsDataRecorder`

__init__(*exchanges=None*, *entity_id=None*, *entity_ids=None*, *code=None*, *codes=None*, *day_data=False*, *force_update=False*, *sleeping_time=5*, *real_time=False*, *fix_duplicate_way='add'*, *start_timestamp=None*, *end_timestamp=None*) → None

record(*entity*, *start*, *end*, *size*, *timestamps*)

implement the recording logic in this method, should return json or domain list

Parameters::

- **entity** –
- **start** –
- **end** –
- **size** –
- **timestamps** –

clear_state_data(*entity_id=None*)

clear state of the entity

Parameters:: **entity_id** – entity id

decode_state(*state: str*)

decode state

Parameters:: **state** –

Returns::

encode_state(*state: object*)

encode state

Parameters:: **state** –

Returns::

entity_provider*: str = 'eastmoney'*

overwrite them to fetch the entity list

entity_schema

alias of `StockDetail`

generate_domain(*entity*, *original_data*)

generate the data_schema instance using entity and original_data,the original_data is from record result

Parameters::

- **entity** –
- **original_data** –

generate_domain_id(*entity*, *original_data*, *time_fmt='YYYY-MM-DD'*)

generate domain id from the entity and original data,the default id meaning:entity + event happen time

Parameters::

- **entity** –
- **original_data** –
- **time_fmt** –

Returns::

Return type::

get_data_map()

{‘original_field’:(‘domain_field’,transform_func)}

get_evaluated_time_field()

the timestamp field for evaluating time range of recorder,used in get_latest_saved_record

init_entities()

init the entities which we would record data for

name = *None*

name of the service, default name of class if not set manually

original_page_url = *None*

original page url

persist(*entity*, *domain_list*)

persist the domain list to db

Parameters::

- **entity** –
- **domain_list** –

provider*: str = 'eastmoney'*

overwrite them to set up the data you want to record

state_schema

alias of `RecorderState`

url = *None*

request url

exchanges

setup the entities you want to record

entity_ids

set entity_ids or (entity_type,exchanges,codes)

session

using to do db operations

zvt.recorders.eastmoney.finance.eastmoney_balance_sheet_recorder

Classes

`ChinaStockBalanceSheetRecorder`([exchanges, ...])

ChinaStockBalanceSheetRecorder

class zvt.recorders.eastmoney.finance.eastmoney_balance_sheet_recorder.ChinaStockBalanceSheetRecorder(*exchanges=None*, *entity_id=None*, *entity_ids=None*, *code=None*, *codes=None*, *day_data=False*, *force_update=False*, *sleeping_time=5*, *real_time=False*, *fix_duplicate_way='add'*, *start_timestamp=None*, *end_timestamp=None*)

Bases: `BaseChinaStockFinanceRecorder`

__init__(*exchanges=None*, *entity_id=None*, *entity_ids=None*, *code=None*, *codes=None*, *day_data=False*, *force_update=False*, *sleeping_time=5*, *real_time=False*, *fix_duplicate_way='add'*, *start_timestamp=None*, *end_timestamp=None*) → None

data_schema

alias of `BalanceSheet`

url = *'https://emh5.eastmoney.com/api/CaiWuFenXi/GetZiChanFuZhaiBiaoList'*

request url

get_data_map()

{'original_field':('domain_field',transform_func)}

clear_state_data(*entity_id=None*)

clear state of the entity

Parameters:: **entity_id** – entity id

decode_state(*state: str*)

decode state

Parameters:: **state** –

Returns::

encode_state(*state: object*)

encode state

Parameters:: **state** –

Returns::

entity_provider*: str = 'eastmoney'*

overwrite them to fetch the entity list

entity_schema

alias of `StockDetail`

generate_domain(*entity*, *original_data*)

generate the data_schema instance using entity and original_data,the original_data is from record result

Parameters::

- **entity** –
- **original_data** –

generate_domain_id(*entity*, *original_data*, *time_fmt='YYYY-MM-DD'*)

generate domain id from the entity and original data,the default id meaning:entity + event happen time

Parameters::

- **entity** –
- **original_data** –
- **time_fmt** –

Returns::

Return type::

get_evaluated_time_field()

the timestamp field for evaluating time range of recorder,used in get_latest_saved_record

init_entities()

init the entities which we would record data for

name = *None*

name of the service, default name of class if not set manually

original_page_url = *None*

original page url

persist(*entity*, *domain_list*)

persist the domain list to db

Parameters::

- **entity** –
- **domain_list** –

provider*: str = 'eastmoney'*

overwrite them to set up the data you want to record

record(*entity*, *start*, *end*, *size*, *timestamps*)

implement the recording logic in this method, should return json or domain list

Parameters::

- **entity** –
- **start** –
- **end** –
- **size** –
- **timestamps** –

state_schema

alias of `RecorderState`

exchanges

setup the entities you want to record

entity_ids

set entity_ids or (entity_type,exchanges,codes)

session

using to do db operations

zvt.recorders.eastmoney.finance.eastmoney_cash_flow_recorder

Classes

`ChinaStockCashFlowRecorder`([exchanges, ...])

ChinaStockCashFlowRecorder

class zvt.recorders.eastmoney.finance.eastmoney_cash_flow_recorder.ChinaStockCashFlowRecorder(*exchanges=None*, *entity_id=None*, *entity_ids=None*, *code=None*, *codes=None*, *day_data=False*, *force_update=False*, *sleeping_time=5*, *real_time=False*, *fix_duplicate_way='add'*, *start_timestamp=None*, *end_timestamp=None*)

Bases: `BaseChinaStockFinanceRecorder`

__init__(*exchanges=None*, *entity_id=None*, *entity_ids=None*, *code=None*, *codes=None*, *day_data=False*, *force_update=False*, *sleeping_time=5*, *real_time=False*, *fix_duplicate_way='add'*, *start_timestamp=None*, *end_timestamp=None*) → None

data_schema

alias of `CashFlowStatement`

url = *'https://emh5.eastmoney.com/api/CaiWuFenXi/GetXianJinLiuLiangBiaoList'*

request url

get_data_map()

{'original_field':('domain_field',transform_func)}

clear_state_data(*entity_id=None*)

clear state of the entity

Parameters:: **entity_id** – entity id

decode_state(*state: str*)

decode state

Parameters:: **state** –

Returns::

encode_state(*state: object*)

encode state

Parameters:: **state** –

Returns::

entity_provider*: str = 'eastmoney'*

overwrite them to fetch the entity list

entity_schema

alias of `StockDetail`

generate_domain(*entity*, *original_data*)

generate the data_schema instance using entity and original_data,the original_data is from record result

Parameters::

- **entity** –
- **original_data** –

generate_domain_id(*entity*, *original_data*, *time_fmt='YYYY-MM-DD'*)

generate domain id from the entity and original data,the default id meaning:entity + event happen time

Parameters::

- **entity** –
- **original_data** –
- **time_fmt** –

Returns::

Return type::

get_evaluated_time_field()

the timestamp field for evaluating time range of recorder,used in get_latest_saved_record

init_entities()

init the entities which we would record data for

name = *None*

name of the service, default name of class if not set manually

original_page_url = *None*

original page url

persist(*entity*, *domain_list*)

persist the domain list to db

Parameters::

- **entity** –
- **domain_list** –

provider*: str = 'eastmoney'*

overwrite them to set up the data you want to record

record(*entity*, *start*, *end*, *size*, *timestamps*)

implement the recording logic in this method, should return json or domain list

Parameters::

- **entity** –
- **start** –
- **end** –
- **size** –
- **timestamps** –

state_schema

alias of `RecorderState`

exchanges

setup the entities you want to record

entity_ids

set entity_ids or (entity_type,exchanges,codes)

session

using to do db operations

zvt.recorders.eastmoney.finance.eastmoney_finance_factor_recorder

Classes

`ChinaStockFinanceFactorRecorder`([exchanges, ...])

ChinaStockFinanceFactorRecorder

class zvt.recorders.eastmoney.finance.eastmoney_finance_factor_recorder.ChinaStockFinanceFactorRecorder(*exchanges=None*, *entity_id=None*, *entity_ids=None*, *code=None*, *codes=None*, *day_data=False*, *force_update=False*, *sleeping_time=5*, *real_time=False*, *fix_duplicate_way='add'*, *start_timestamp=None*, *end_timestamp=None*)

Bases: `BaseChinaStockFinanceRecorder`

__init__(*exchanges=None*, *entity_id=None*, *entity_ids=None*, *code=None*, *codes=None*, *day_data=False*, *force_update=False*, *sleeping_time=5*, *real_time=False*, *fix_duplicate_way='add'*, *start_timestamp=None*, *end_timestamp=None*) → None

url = *'https://emh5.eastmoney.com/api/CaiWuFenXi/GetZhuYaoZhiBiaoList'*

request url

data_schema

alias of `FinanceFactor`

get_data_map()

{'original_field':('domain_field',transform_func)}

clear_state_data(*entity_id=None*)

clear state of the entity

Parameters:: **entity_id** – entity id

decode_state(*state: str*)

decode state

Parameters:: **state** –

Returns::

encode_state(*state: object*)

encode state

Parameters:: **state** –

Returns::

entity_provider*: str = 'eastmoney'*

overwrite them to fetch the entity list

entity_schema

alias of `StockDetail`

generate_domain(*entity*, *original_data*)

generate the data_schema instance using entity and original_data,the original_data is from record result

Parameters::

- **entity** –
- **original_data** –

generate_domain_id(*entity*, *original_data*, *time_fmt='YYYY-MM-DD'*)

generate domain id from the entity and original data,the default id meaning:entity + event happen time

Parameters::

- **entity** –
- **original_data** –
- **time_fmt** –

Returns::

Return type::

get_evaluated_time_field()

the timestamp field for evaluating time range of recorder,used in get_latest_saved_record

init_entities()

init the entities which we would record data for

name = *None*

name of the service, default name of class if not set manually

original_page_url = *None*

original page url

persist(*entity*, *domain_list*)

persist the domain list to db

Parameters::

- **entity** –
- **domain_list** –

provider*: str = 'eastmoney'*

overwrite them to set up the data you want to record

record(*entity*, *start*, *end*, *size*, *timestamps*)

implement the recording logic in this method, should return json or domain list

Parameters::

- **entity** –
- **start** –
- **end** –
- **size** –
- **timestamps** –

state_schema

alias of `RecorderState`

exchanges

setup the entities you want to record

entity_ids

set entity_ids or (entity_type,exchanges,codes)

session

using to do db operations

zvt.recorders.eastmoney.finance.eastmoney_income_statement_recorder

Classes

`ChinaStockIncomeStatementRecorder`([...])

ChinaStockIncomeStatementRecorder

class zvt.recorders.eastmoney.finance.eastmoney_income_statement_recorder.ChinaStockIncomeStatementRecorder(*exchanges=None*, *entity_id=None*, *entity_ids=None*, *code=None*, *codes=None*, *day_data=False*, *force_update=False*, *sleeping_time=5*, *real_time=False*, *fix_duplicate_way='add'*, *start_timestamp=None*, *end_timestamp=None*)

Bases: `BaseChinaStockFinanceRecorder`

__init__(*exchanges=None*, *entity_id=None*, *entity_ids=None*, *code=None*, *codes=None*, *day_data=False*, *force_update=False*, *sleeping_time=5*, *real_time=False*, *fix_duplicate_way='add'*, *start_timestamp=None*, *end_timestamp=None*) → None

data_schema

alias of `IncomeStatement`

url = *'https://emh5.eastmoney.com/api/CaiWuFenXi/GetLiRunBiaoList'*

request url

get_data_map()

{'original_field':('domain_field',transform_func)}

clear_state_data(*entity_id=None*)

clear state of the entity

Parameters:: **entity_id** – entity id

decode_state(*state: str*)

decode state

Parameters:: **state** –

Returns::

encode_state(*state: object*)

encode state

Parameters:: **state** –

Returns::

entity_provider*: str = 'eastmoney'*

overwrite them to fetch the entity list

entity_schema

alias of `StockDetail`

generate_domain(*entity*, *original_data*)

generate the data_schema instance using entity and original_data,the original_data is from record result

Parameters::

- **entity** –
- **original_data** –

generate_domain_id(*entity*, *original_data*, *time_fmt='YYYY-MM-DD'*)

generate domain id from the entity and original data,the default id meaning:entity + event happen time

Parameters::

- **entity** –
- **original_data** –
- **time_fmt** –

Returns::

Return type::

get_evaluated_time_field()

the timestamp field for evaluating time range of recorder,used in get_latest_saved_record

init_entities()

init the entities which we would record data for

name = *None*

name of the service, default name of class if not set manually

original_page_url = *None*

original page url

persist(*entity*, *domain_list*)

persist the domain list to db

Parameters::

- **entity** –
- **domain_list** –

provider*: str = 'eastmoney'*

overwrite them to set up the data you want to record

record(*entity*, *start*, *end*, *size*, *timestamps*)

implement the recording logic in this method, should return json or domain list

Parameters::

- **entity** –
- **start** –
- **end** –
- **size** –
- **timestamps** –

state_schema

alias of `RecorderState`

exchanges

setup the entities you want to record

entity_ids

set entity_ids or (entity_type,exchanges,codes)

session

using to do db operations

zvt.recorders.eastmoney.holder

Modules

zvt.recorders.eastmoney.holder.eastmoney_stock_actor_recorder

---

zvt.recorders.eastmoney.holder.eastmoney_top_ten_holder_recorder

---

zvt.recorders.eastmoney.holder.eastmoney_top_ten_tradable_holder_recorder

---

zvt.recorders.eastmoney.holder.eastmoney_stock_actor_recorder

Classes

`EastmoneyActorRecorder`([force_update, ...])

EastmoneyActorRecorder

class zvt.recorders.eastmoney.holder.eastmoney_stock_actor_recorder.EastmoneyActorRecorder(*force_update: bool = False*, *sleeping_time: int = 10*)

Bases: `Recorder`

__init__(*force_update: bool = False*, *sleeping_time: int = 10*) → None

name = *'eastmoney_actor_recorder'*

name of the service, default name of class if not set manually

provider*: str = 'eastmoney'*

overwrite them to set up the data you want to record

data_schema

alias of `ActorMeta`

url = *"https://datacenter.eastmoney.com/securities/api/data/v1/get?reportName=RPT_FREEHOLDERS_BASIC_INFO&columns=HOLDER_NAME,END_DATE,HOLDER_NEW,HOLDER_NUM,HOLDER_CODE&quoteColumns=&filter=(END_DATE='{}')&pageNumber={}&pageSize={}&sortTypes=-1,-1&sortColumns=HOLDER_NUM,HOLDER_NAME&source=SECURITIES&client=SW"*

request url

clear_state_data(*entity_id=None*)

clear state of the entity

Parameters:: **entity_id** – entity id

decode_state(*state: str*)

decode state

Parameters:: **state** –
Returns::

encode_state(*state: object*)

encode state

Parameters:: **state** –
Returns::

original_page_url = *None*

original page url

state_schema

alias of `RecorderState`

session

using to do db operations

zvt.recorders.eastmoney.holder.eastmoney_top_ten_holder_recorder

Classes

`TopTenHolderRecorder`([force_update, ...])

TopTenHolderRecorder

class zvt.recorders.eastmoney.holder.eastmoney_top_ten_holder_recorder.TopTenHolderRecorder(*force_update=False*, *sleeping_time=5*, *exchanges=None*, *entity_id=None*, *entity_ids=None*, *code=None*, *codes=None*, *day_data=False*, *entity_filters=None*, *ignore_failed=True*, *real_time=False*, *fix_duplicate_way='add'*, *start_timestamp=None*, *end_timestamp=None*)

Bases: `EastmoneyTimestampsDataRecorder`

__init__(*force_update=False*, *sleeping_time=5*, *exchanges=None*, *entity_id=None*, *entity_ids=None*, *code=None*, *codes=None*, *day_data=False*, *entity_filters=None*, *ignore_failed=True*, *real_time=False*, *fix_duplicate_way='add'*, *start_timestamp=None*, *end_timestamp=None*) → None

provider*: str = 'eastmoney'*

overwrite them to set up the data you want to record

data_schema

alias of `TopTenHolder`

url = *'https://emh5.eastmoney.com/api/GuBenGuDong/GetShiDaGuDong'*

request url

get_data_map()

{‘original_field’:(‘domain_field’,transform_func)}

generate_domain_id(*entity*, *original_data*)

generate domain id from the entity and original data,the default id meaning:entity + event happen time

Parameters::
- **entity** –
- **original_data** –
- **time_fmt** –

Returns::
Return type::

clear_state_data(*entity_id=None*)

clear state of the entity

Parameters:: **entity_id** – entity id

decode_state(*state: str*)

decode state

Parameters:: **state** –
Returns::

encode_state(*state: object*)

encode state

Parameters:: **state** –
Returns::

entity_provider*: str = 'eastmoney'*

overwrite them to fetch the entity list

entity_schema

alias of `StockDetail`

generate_domain(*entity*, *original_data*)

generate the data_schema instance using entity and original_data,the original_data is from record result

Parameters::
- **entity** –
- **original_data** –

get_evaluated_time_field()

the timestamp field for evaluating time range of recorder,used in get_latest_saved_record

init_entities()

init the entities which we would record data for

name = *None*

name of the service, default name of class if not set manually

original_page_url = *None*

original page url

persist(*entity*, *domain_list*)

persist the domain list to db

Parameters::

- **entity** –
- **domain_list** –

record(*entity_item*, *start*, *end*, *size*, *timestamps*)

implement the recording logic in this method, should return json or domain list

Parameters::

- **entity** –
- **start** –
- **end** –
- **size** –
- **timestamps** –

state_schema

alias of `RecorderState`

exchanges

setup the entities you want to record

entity_ids

set entity_ids or (entity_type,exchanges,codes)

session

using to do db operations

zvt.recorders.eastmoney.holder.eastmoney_top_ten_tradable_holder_recorder

Classes

`TopTenTradableHolderRecorder`([force_update, ...])

TopTenTradableHolderRecorder

class zvt.recorders.eastmoney.holder.eastmoney_top_ten_tradable_holder_recorder.TopTenTradableHolderRecorder(*force_update=False*, *sleeping_time=5*, *exchanges=None*, *entity_id=None*, *entity_ids=None*, *code=None*, *codes=None*, *day_data=False*, *entity_filters=None*, *ignore_failed=True*, *real_time=False*, *fix_duplicate_way='add'*, *start_timestamp=None*, *end_timestamp=None*)

Bases: `TopTenHolderRecorder`

__init__(*force_update=False*, *sleeping_time=5*, *exchanges=None*, *entity_id=None*, *entity_ids=None*, *code=None*, *codes=None*, *day_data=False*, *entity_filters=None*, *ignore_failed=True*, *real_time=False*, *fix_duplicate_way='add'*, *start_timestamp=None*, *end_timestamp=None*) → None

provider*: str = 'eastmoney'*

overwrite them to set up the data you want to record

data_schema

alias of `TopTenTradableHolder`

url = *'https://emh5.eastmoney.com/api/GuBenGuDong/GetShiDaLiuTongGuDong'*

request url

clear_state_data(*entity_id=None*)

clear state of the entity

Parameters:: **entity_id** – entity id

decode_state(*state: str*)

decode state

Parameters:: **state** –
Returns::

encode_state(*state: object*)

encode state

Parameters:: **state** –
Returns::

entity_provider*: str = 'eastmoney'*

overwrite them to fetch the entity list

entity_schema

alias of `StockDetail`

generate_domain(*entity*, *original_data*)

generate the data_schema instance using entity and original_data,the original_data is from record result

Parameters::
- **entity** –
- **original_data** –

generate_domain_id(*entity*, *original_data*)

generate domain id from the entity and original data,the default id meaning:entity + event happen time

Parameters::
- **entity** –
- **original_data** –
- **time_fmt** –

Returns::
Return type::

get_data_map()

{‘original_field’:(‘domain_field’,transform_func)}

get_evaluated_time_field()

the timestamp field for evaluating time range of recorder,used in get_latest_saved_record

init_entities()

init the entities which we would record data for

name = *None*

name of the service, default name of class if not set manually

original_page_url = *None*

original page url

persist(*entity*, *domain_list*)

persist the domain list to db

Parameters::

- **entity** –
- **domain_list** –

record(*entity_item*, *start*, *end*, *size*, *timestamps*)

implement the recording logic in this method, should return json or domain list

Parameters::

- **entity** –
- **start** –
- **end** –
- **size** –
- **timestamps** –

state_schema

alias of `RecorderState`

exchanges

setup the entities you want to record

entity_ids

set entity_ids or (entity_type,exchanges,codes)

session

using to do db operations

zvt.recorders.eastmoney.meta

Modules

zvt.recorders.eastmoney.meta.eastmoney_block_meta_recorder

---

zvt.recorders.eastmoney.meta.eastmoney_stock_meta_recorder

---

zvt.recorders.eastmoney.meta.eastmoney_block_meta_recorder

Classes

`EastmoneyBlockRecorder`([force_update, ...])

`EastmoneyBlockStockRecorder`([force_update, ...])

EastmoneyBlockRecorder

class zvt.recorders.eastmoney.meta.eastmoney_block_meta_recorder.EastmoneyBlockRecorder(*force_update: bool = False*, *sleeping_time: int = 10*)

Bases: `Recorder`

__init__(*force_update: bool = False*, *sleeping_time: int = 10*) → None

provider*: str = 'eastmoney'*

overwrite them to set up the data you want to record

data_schema

alias of `Block`

clear_state_data(*entity_id=None*)

clear state of the entity

Parameters:: **entity_id** – entity id

decode_state(*state: str*)

decode state

Parameters:: **state** –

Returns::

encode_state(*state: object*)

encode state

Parameters:: **state** –

Returns::

name = *None*

name of the service, default name of class if not set manually

original_page_url = *None*

original page url

state_schema

alias of `RecorderState`

url = *None*

request url

session

using to do db operations

EastmoneyBlockStockRecorder

class zvt.recorders.eastmoney.meta.eastmoney_block_meta_recorder.EastmoneyBlockStockRecorder(*force_update=False*, *sleeping_time=5*, *exchanges=None*, *entity_id=None*, *entity_ids=None*, *code=None*, *codes=None*, *day_data=False*, *entity_filters=None*, *ignore_failed=True*, *real_time=False*, *fix_duplicate_way='add'*, *start_timestamp=None*, *end_timestamp=None*, *return_unfinished=False*)

Bases: `TimeSeriesDataRecorder`

__init__(*force_update=False*, *sleeping_time=5*, *exchanges=None*, *entity_id=None*, *entity_ids=None*, *code=None*, *codes=None*, *day_data=False*, *entity_filters=None*, *ignore_failed=True*, *real_time=False*, *fix_duplicate_way='add'*, *start_timestamp=None*, *end_timestamp=None*, *return_unfinished=False*) → None

Parameters::

- **code** –
- **ignore_failed** –
- **entity_filters** –
- **exchanges** –
- **entity_id** – for record single entity
- **entity_ids** – set entity_ids or (entity_type,exchanges,codes)
- **codes** –
- **day_data** – one record per day,set to True if you want skip recording it when data of today exist
- **force_update** –
- **sleeping_time** –

entity_provider*: str = 'eastmoney'*

overwrite them to fetch the entity list

entity_schema

alias of `Block`

provider*: str = 'eastmoney'*

overwrite them to set up the data you want to record

data_schema

alias of `BlockStock`

record(*entity*, *start*, *end*, *size*, *timestamps*)

implement the recording logic in this method, should return json or domain list

Parameters::

- **entity** –
- **start** –
- **end** –
- **size** –
- **timestamps** –

clear_state_data(*entity_id=None*)

clear state of the entity

Parameters:: **entity_id** – entity id

decode_state(*state: str*)

decode state

Parameters:: **state** –

Returns::

encode_state(*state: object*)

encode state

Parameters:: **state** –

Returns::

generate_domain(*entity*, *original_data*)

generate the data_schema instance using entity and original_data,the original_data is from record result

Parameters::

- **entity** –
- **original_data** –

generate_domain_id(*entity*, *original_data*, *time_fmt='YYYY-MM-DD'*)

generate domain id from the entity and original data,the default id meaning:entity + event happen time

Parameters::

- **entity** –
- **original_data** –
- **time_fmt** –

Returns::

Return type::

get_data_map()

{'original_field':('domain_field',transform_func)}

get_evaluated_time_field()

the timestamp field for evaluating time range of recorder,used in get_latest_saved_record

init_entities()

init the entities which we would record data for

name = *None*

name of the service, default name of class if not set manually

original_page_url = *None*

original page url

persist(*entity*, *domain_list*)

persist the domain list to db

Parameters::

- **entity** –
- **domain_list** –

state_schema

alias of `RecorderState`

url = *None*

request url

exchanges

setup the entities you want to record

entity_ids

set entity_ids or (entity_type,exchanges,codes)

session

using to do db operations

zvt.recorders.eastmoney.meta.eastmoney_stock_meta_recorder

Classes

`EastmoneyStockDetailRecorder`([force_update, ...])

`EastmoneyStockRecorder`([force_update, ...])

EastmoneyStockDetailRecorder

class zvt.recorders.eastmoney.meta.eastmoney_stock_meta_recorder.EastmoneyStockDetailRecorder(*force_update=False*, *sleeping_time=5*, *code=None*, *codes=None*)

Bases: `Recorder`

__init__(*force_update=False*, *sleeping_time=5*, *code=None*, *codes=None*) → None

provider*: str = 'eastmoney'*

overwrite them to set up the data you want to record

data_schema

alias of `StockDetail`

clear_state_data(*entity_id=None*)

clear state of the entity

Parameters:: **entity_id** – entity id

decode_state(*state: str*)

decode state

Parameters:: **state** –
Returns::

encode_state(*state: object*)

encode state

Parameters:: **state** –
Returns::

name = *None*

name of the service, default name of class if not set manually

original_page_url = *None*

original page url

state_schema

alias of `RecorderState`

url = *None*

request url

session

using to do db operations

EastmoneyStockRecorder

class zvt.recorders.eastmoney.meta.eastmoney_stock_meta_recorder.EastmoneyStockRecorder(*force_update: bool = False*, *sleeping_time: int = 10*)

Bases: `ExchangeStockMetaRecorder`

__init__(*force_update: bool = False*, *sleeping_time: int = 10*) → None

data_schema

alias of `Stock`

provider*: str = 'eastmoney'*

overwrite them to set up the data you want to record

clear_state_data(*entity_id=None*)

clear state of the entity

Parameters:: **entity_id** – entity id

decode_state(*state: str*)

decode state

Parameters:: **state** –
Returns::

encode_state(*state: object*)

encode state

Parameters:: **state** –
Returns::

name = *None*

name of the service, default name of class if not set manually

original_page_url = *'http://www.sse.com.cn/assortment/stock/list/share/'*

original page url

state_schema

alias of `RecorderState`

url = *None*

request url

session

using to do db operations

zvt.recorders.eastmoney.trading

Modules

zvt.recorders.eastmoney.trading.eastmoney_holder_trading_recorder

---

zvt.recorders.eastmoney.trading.eastmoney_manager_trading_recorder

---

zvt.recorders.eastmoney.trading.eastmoney_holder_trading_recorder

Classes

`HolderTradingRecorder`([force_update, ...])

HolderTradingRecorder

class zvt.recorders.eastmoney.trading.eastmoney_holder_trading_recorder.HolderTradingRecorder(*force_update=False*, *sleeping_time=5*, *exchanges=None*, *entity_id=None*, *entity_ids=None*, *code=None*, *codes=None*, *day_data=False*, *entity_filters=None*, *ignore_failed=True*, *real_time=False*, *fix_duplicate_way='add'*, *start_timestamp=None*, *end_timestamp=None*, *return_unfinished=False*)

Bases: `EastmoneyMoreDataRecorder`

__init__(*force_update=False*, *sleeping_time=5*, *exchanges=None*, *entity_id=None*, *entity_ids=None*, *code=None*, *codes=None*, *day_data=False*, *entity_filters=None*, *ignore_failed=True*, *real_time=False*, *fix_duplicate_way='add'*, *start_timestamp=None*, *end_timestamp=None*, *return_unfinished=False*) → None

data_schema

alias of `HolderTrading`

url = *'https://emh5.eastmoney.com/api/JiaoYiShuJu/GetGuDongZengJian'*

request url

get_data_map()

{'original_field':('domain_field',transform_func)}

generate_domain_id(*entity*, *original_data*)

generate domain id from the entity and original data,the default id meaning:entity + event happen time

Parameters::

- **entity** –
- **original_data** –
- **time_fmt** –

Returns::

Return type::

clear_state_data(*entity_id=None*)

clear state of the entity

Parameters:: **entity_id** – entity id

decode_state(*state: str*)

decode state

Parameters:: **state** –

Returns::

encode_state(*state: object*)

encode state

Parameters:: **state** –

Returns::

entity_provider*: str = 'eastmoney'*

overwrite them to fetch the entity list

entity_schema

alias of `StockDetail`

generate_domain(*entity*, *original_data*)

generate the data_schema instance using entity and original_data,the original_data is from record result

Parameters::

- **entity** –
- **original_data** –

get_evaluated_time_field()

the timestamp field for evaluating time range of recorder,used in get_latest_saved_record

init_entities()

init the entities which we would record data for

name = *None*

name of the service, default name of class if not set manually

original_page_url = *None*

original page url

persist(*entity*, *domain_list*)

persist the domain list to db

Parameters::

- **entity** –
- **domain_list** –

provider*: str = 'eastmoney'*

overwrite them to set up the data you want to record

record(*entity_item*, *start*, *end*, *size*, *timestamps*)

implement the recording logic in this method, should return json or domain list

Parameters::

- **entity** –
- **start** –
- **end** –
- **size** –
- **timestamps** –

state_schema

alias of `RecorderState`

exchanges

setup the entities you want to record

entity_ids

set entity_ids or (entity_type,exchanges,codes)

session

using to do db operations

zvt.recorders.eastmoney.trading.eastmoney_manager_trading_recorder

Classes

`ManagerTradingRecorder`([force_update, ...])

ManagerTradingRecorder

class zvt.recorders.eastmoney.trading.eastmoney_manager_trading_recorder.ManagerTradingRecorder(*force_update=False*, *sleeping_time=5*, *exchanges=None*, *entity_id=None*, *entity_ids=None*, *code=None*, *codes=None*, *day_data=False*, *entity_filters=None*, *ignore_failed=True*, *real_time=False*, *fix_duplicate_way='add'*, *start_timestamp=None*, *end_timestamp=None*, *return_unfinished=False*)

Bases: `EastmoneyMoreDataRecorder`

__init__(*force_update=False*, *sleeping_time=5*, *exchanges=None*, *entity_id=None*, *entity_ids=None*, *code=None*, *codes=None*, *day_data=False*, *entity_filters=None*, *ignore_failed=True*, *real_time=False*, *fix_duplicate_way='add'*, *start_timestamp=None*, *end_timestamp=None*, *return_unfinished=False*) → None

data_schema

alias of `ManagerTrading`

url = *'https://emh5.eastmoney.com/api/JiaoYiShuJu/GetGaoGuanZengJian'*

request url

get_data_map()

{'original_field':('domain_field',transform_func)}

generate_domain_id(*entity*, *original_data*)

generate domain id from the entity and original data,the default id meaning:entity + event happen time

Parameters::

- **entity** –
- **original_data** –
- **time_fmt** –

Returns::

Return type::

clear_state_data(*entity_id=None*)

clear state of the entity

Parameters:: **entity_id** – entity id

decode_state(*state: str*)

decode state

Parameters:: **state** –

Returns::

encode_state(*state: object*)

encode state

Parameters:: **state** –

Returns::

entity_provider*: str = 'eastmoney'*

overwrite them to fetch the entity list

entity_schema

alias of `StockDetail`

generate_domain(*entity*, *original_data*)

generate the data_schema instance using entity and original_data,the original_data is from record result

Parameters::

- **entity** –
- **original_data** –

get_evaluated_time_field()

the timestamp field for evaluating time range of recorder,used in get_latest_saved_record

init_entities()

init the entities which we would record data for

name = *None*

name of the service, default name of class if not set manually

original_page_url = *None*

original page url

persist(*entity*, *domain_list*)

persist the domain list to db

Parameters::

- **entity** –
- **domain_list** –

provider*: str = 'eastmoney'*

overwrite them to set up the data you want to record

record(*entity_item*, *start*, *end*, *size*, *timestamps*)

implement the recording logic in this method, should return json or domain list

Parameters::

- **entity** –
- **start** –
- **end** –
- **size** –
- **timestamps** –

state_schema

alias of `RecorderState`

exchanges

setup the entities you want to record

entity_ids

set entity_ids or (entity_type,exchanges,codes)

session

using to do db operations

zvt.recorders.em

Modules

zvt.recorders.em.actor

---

zvt.recorders.em.em_api

---

zvt.recorders.em.macro

---

zvt.recorders.em.meta

---

zvt.recorders.em.misc

---

zvt.recorders.em.quotes

---

zvt.recorders.em.trading

---

zvt.recorders.em.actor

Modules

zvt.recorders.em.actor.em_stock_actor_summary_recorder

---

zvt.recorders.em.actor.em_stock_ii_recorder

---

zvt.recorders.em.actor.em_stock_top_ten_free_recorder

---

zvt.recorders.em.actor.em_stock_top_ten_recorder

---

zvt.recorders.em.actor.em_stock_actor_summary_recorder

Classes

`EMStockActorSummaryRecorder`([force_update, ...])

EMStockActorSummaryRecorder

class zvt.recorders.em.actor.em_stock_actor_summary_recorder.EMStockActorSummaryRecorder(*force_update=False*, *sleeping_time=5*, *exchanges=None*, *entity_id=None*, *entity_ids=None*, *code=None*, *codes=None*, *day_data=False*, *entity_filters=None*, *ignore_failed=True*, *real_time=False*, *fix_duplicate_way='add'*, *start_timestamp=None*, *end_timestamp=None*)

Bases: `TimestampsDataRecorder`

__init__(*force_update=False*, *sleeping_time=5*, *exchanges=None*, *entity_id=None*, *entity_ids=None*, *code=None*, *codes=None*, *day_data=False*, *entity_filters=None*, *ignore_failed=True*, *real_time=False*, *fix_duplicate_way='add'*, *start_timestamp=None*, *end_timestamp=None*) → None

Parameters::

- **code** –
- **ignore_failed** –
- **entity_filters** –
- **exchanges** –
- **entity_id** – for record single entity
- **entity_ids** – set entity_ids or (entity_type,exchanges,codes)
- **codes** –
- **day_data** – one record per day,set to True if you want skip recording it when data of today exist
- **force_update** –
- **sleeping_time** –

entity_provider*: str = 'em'*

overwrite them to fetch the entity list

entity_schema

alias of `Stock`

provider*: str = 'em'*

overwrite them to set up the data you want to record

data_schema

alias of `StockActorSummary`

record(*entity*, *start*, *end*, *size*, *timestamps*)

implement the recording logic in this method, should return json or domain list

Parameters::

- **entity** –
- **start** –
- **end** –
- **size** –
- **timestamps** –

clear_state_data(*entity_id=None*)

clear state of the entity

Parameters:: **entity_id** – entity id

decode_state(*state: str*)

decode state

Parameters:: **state** –

Returns::

encode_state(*state: object*)

encode state

Parameters:: **state** –

Returns::

generate_domain(*entity*, *original_data*)

generate the data_schema instance using entity and original_data,the original_data is from record result

Parameters::

- **entity** –
- **original_data** –

generate_domain_id(*entity*, *original_data*, *time_fmt='YYYY-MM-DD'*)

generate domain id from the entity and original data,the default id meaning:entity + event happen time

Parameters::

- **entity** –
- **original_data** –
- **time_fmt** –

Returns::

Return type::

get_data_map()

{'original_field':('domain_field',transform_func)}

get_evaluated_time_field()

the timestamp field for evaluating time range of recorder,used in get_latest_saved_record

init_entities()

init the entities which we would record data for

name = *None*

name of the service, default name of class if not set manually

original_page_url = *None*

original page url

persist(*entity*, *domain_list*)

persist the domain list to db

Parameters::

- **entity** –
- **domain_list** –

state_schema

alias of `RecorderState`

url = *None*

request url

exchanges

setup the entities you want to record

entity_ids

set entity_ids or (entity_type,exchanges,codes)

session

using to do db operations

zvt.recorders.em.actor.em_stock_ii_recorder

Classes

`EMStockIIRecorder`([force_update, ...])

EMStockIIRecorder

class zvt.recorders.em.actor.em_stock_ii_recorder.EMStockIIRecorder(*force_update=False*, *sleeping_time=5*, *exchanges=None*, *entity_id=None*, *entity_ids=None*, *code=None*, *codes=None*, *day_data=False*, *entity_filters=None*, *ignore_failed=True*, *real_time=False*, *fix_duplicate_way='add'*, *start_timestamp=None*, *end_timestamp=None*)

Bases: `TimestampsDataRecorder`

__init__(*force_update=False*, *sleeping_time=5*, *exchanges=None*, *entity_id=None*, *entity_ids=None*, *code=None*, *codes=None*, *day_data=False*, *entity_filters=None*, *ignore_failed=True*, *real_time=False*, *fix_duplicate_way='add'*, *start_timestamp=None*, *end_timestamp=None*) → None

Parameters::

- **code** –
- **ignore_failed** –
- **entity_filters** –
- **exchanges** –
- **entity_id** – for record single entity
- **entity_ids** – set entity_ids or (entity_type,exchanges,codes)
- **codes** –
- **day_data** – one record per day,set to True if you want skip recording it when data of today exist
- **force_update** –
- **sleeping_time** –

entity_provider*: str = 'em'*

overwrite them to fetch the entity list

entity_schema

alias of `Stock`

provider*: str = 'em'*

overwrite them to set up the data you want to record

data_schema

alias of `StockInstitutionalInvestorHolder`

record(*entity*, *start*, *end*, *size*, *timestamps*)

implement the recording logic in this method, should return json or domain list

Parameters::

- **entity** –
- **start** –
- **end** –
- **size** –
- **timestamps** –

clear_state_data(*entity_id=None*)

clear state of the entity

Parameters:: **entity_id** – entity id

decode_state(*state: str*)

decode state

Parameters:: **state** –

Returns::

encode_state(*state: object*)

encode state

Parameters:: **state** –

Returns::

generate_domain(*entity*, *original_data*)

generate the data_schema instance using entity and original_data,the original_data is from record result

Parameters::

- **entity** –
- **original_data** –

generate_domain_id(*entity*, *original_data*, *time_fmt='YYYY-MM-DD'*)

generate domain id from the entity and original data,the default id meaning:entity + event happen time

Parameters::

- **entity** –
- **original_data** –
- **time_fmt** –

Returns::

Return type::

get_data_map()

{'original_field':('domain_field',transform_func)}

get_evaluated_time_field()

the timestamp field for evaluating time range of recorder,used in get_latest_saved_record

init_entities()

init the entities which we would record data for

name = *None*

name of the service, default name of class if not set manually

original_page_url = *None*

original page url

persist(*entity*, *domain_list*)

persist the domain list to db

Parameters::

- **entity** –
- **domain_list** –

state_schema

alias of `RecorderState`

url = *None*

request url

exchanges

setup the entities you want to record

entity_ids

set entity_ids or (entity_type,exchanges,codes)

session

using to do db operations

zvt.recorders.em.actor.em_stock_top_ten_free_recorder

Classes

`EMStockTopTenFreeRecorder`([force_update, ...])

EMStockTopTenFreeRecorder

class zvt.recorders.em.actor.em_stock_top_ten_free_recorder.EMStockTopTenFreeRecorder(*force_update=False*, *sleeping_time=5*, *exchanges=None*, *entity_id=None*, *entity_ids=None*, *code=None*, *codes=None*, *day_data=False*, *entity_filters=None*, *ignore_failed=True*, *real_time=False*, *fix_duplicate_way='add'*, *start_timestamp=None*, *end_timestamp=None*)

Bases: `TimestampsDataRecorder`

__init__(*force_update=False*, *sleeping_time=5*, *exchanges=None*, *entity_id=None*, *entity_ids=None*, *code=None*, *codes=None*, *day_data=False*, *entity_filters=None*, *ignore_failed=True*, *real_time=False*, *fix_duplicate_way='add'*, *start_timestamp=None*, *end_timestamp=None*) → None

Parameters::

- **code** –
- **ignore_failed** –
- **entity_filters** –
- **exchanges** –
- **entity_id** – for record single entity
- **entity_ids** – set entity_ids or (entity_type,exchanges,codes)
- **codes** –
- **day_data** – one record per day,set to True if you want skip recording it when data of today exist
- **force_update** –
- **sleeping_time** –

entity_provider*: str = 'em'*

overwrite them to fetch the entity list

entity_schema

alias of `Stock`

provider*: str = 'em'*

overwrite them to set up the data you want to record

data_schema

alias of `StockTopTenFreeHolder`

record(*entity*, *start*, *end*, *size*, *timestamps*)

implement the recording logic in this method, should return json or domain list

Parameters::

- **entity** –
- **start** –
- **end** –
- **size** –
- **timestamps** –

clear_state_data(*entity_id=None*)

clear state of the entity

Parameters:: **entity_id** – entity id

decode_state(*state: str*)

decode state

Parameters:: **state** –

Returns::

encode_state(*state: object*)

encode state

Parameters:: **state** –

Returns::

generate_domain(*entity*, *original_data*)

generate the data_schema instance using entity and original_data,the original_data is from record result

Parameters::

- **entity** –
- **original_data** –

generate_domain_id(*entity*, *original_data*, *time_fmt='YYYY-MM-DD'*)

generate domain id from the entity and original data,the default id meaning:entity + event happen time

Parameters::

- **entity** –
- **original_data** –
- **time_fmt** –

Returns::

Return type::

get_data_map()

{'original_field':('domain_field',transform_func)}

get_evaluated_time_field()

the timestamp field for evaluating time range of recorder,used in get_latest_saved_record

init_entities()

init the entities which we would record data for

name = *None*

name of the service, default name of class if not set manually

original_page_url = *None*

original page url

persist(*entity*, *domain_list*)

persist the domain list to db

Parameters::

- **entity** –
- **domain_list** –

state_schema

alias of `RecorderState`

url = *None*

request url

exchanges

setup the entities you want to record

entity_ids

set entity_ids or (entity_type,exchanges,codes)

session

using to do db operations

zvt.recorders.em.actor.em_stock_top_ten_recorder

Classes

`EMStockTopTenRecorder`([force_update, ...])

EMStockTopTenRecorder

class zvt.recorders.em.actor.em_stock_top_ten_recorder.EMStockTopTenRecorder(*force_update=False*, *sleeping_time=5*, *exchanges=None*, *entity_id=None*, *entity_ids=None*, *code=None*, *codes=None*, *day_data=False*, *entity_filters=None*, *ignore_failed=True*, *real_time=False*, *fix_duplicate_way='add'*, *start_timestamp=None*, *end_timestamp=None*)

Bases: `TimestampsDataRecorder`

__init__(*force_update=False*, *sleeping_time=5*, *exchanges=None*, *entity_id=None*, *entity_ids=None*, *code=None*, *codes=None*, *day_data=False*, *entity_filters=None*, *ignore_failed=True*, *real_time=False*, *fix_duplicate_way='add'*, *start_timestamp=None*, *end_timestamp=None*) → None

Parameters::

- **code** –
- **ignore_failed** –
- **entity_filters** –
- **exchanges** –
- **entity_id** – for record single entity
- **entity_ids** – set entity_ids or (entity_type,exchanges,codes)
- **codes** –
- **day_data** – one record per day,set to True if you want skip recording it when data of today exist
- **force_update** –
- **sleeping_time** –

entity_provider*: str = 'em'*

overwrite them to fetch the entity list

entity_schema

alias of `Stock`

provider*: str = 'em'*

overwrite them to set up the data you want to record

data_schema

alias of `StockTopTenHolder`

record(*entity*, *start*, *end*, *size*, *timestamps*)

implement the recording logic in this method, should return json or domain list

Parameters::

- **entity** –
- **start** –
- **end** –
- **size** –
- **timestamps** –

clear_state_data(*entity_id=None*)

clear state of the entity

Parameters:: **entity_id** – entity id

decode_state(*state: str*)

decode state

Parameters:: **state** –

Returns::

encode_state(*state: object*)

encode state

Parameters:: **state** –

Returns::

generate_domain(*entity*, *original_data*)

generate the data_schema instance using entity and original_data,the original_data is from record result

Parameters::

- **entity** –
- **original_data** –

generate_domain_id(*entity*, *original_data*, *time_fmt='YYYY-MM-DD'*)

generate domain id from the entity and original data,the default id meaning:entity + event happen time

Parameters::

- **entity** –
- **original_data** –
- **time_fmt** –

Returns::

Return type::

get_data_map()

{'original_field':('domain_field',transform_func)}

get_evaluated_time_field()

the timestamp field for evaluating time range of recorder,used in get_latest_saved_record

init_entities()

init the entities which we would record data for

name = *None*

name of the service, default name of class if not set manually

original_page_url = *None*

original page url

persist(*entity*, *domain_list*)

persist the domain list to db

Parameters::

- **entity** –
- **domain_list** –

state_schema

alias of `RecorderState`

url = *None*

request url

exchanges

setup the entities you want to record

entity_ids

set entity_ids or (entity_type,exchanges,codes)

session

using to do db operations

zvt.recorders.em.em_api

Functions

actor_type_to_org_type(actor_type)

---

generate_filters([code, trade_date, ...])

---

get_basic_info(entity_id)

---

get_block_stocks(block_id[, name])

---

get_controlling_shareholder(code)

---

get_dragon_and_tiger(code[, start_date])

---

get_dragon_and_tiger_list(start_date[, end_date])

---

get_em_data(request_type, fields[, session, ...])

---

get_exchange(code)

---

get_free_holder_report_dates(code)

---

get_free_holders(code, end_date)

get_future_list()

get_holder_report_dates(code)

get_holders(code, end_date)

get_ii_holder(code, report_date, org_type)

get_ii_holder_report_dates(code)

get_ii_summary(code, report_date, org_type)

get_kdata(entity_id[, session, level, ...])

get_news(entity_id[, ps, index, ...])

get_quotes()

get_tradable_list([entity_type, exchange, ...])

get_treasury_yield([pn, ps, fetch_all])

get_url(type, sty[, source, filters, ...])

to_em_entity_flag(exchange)

`to_em_fc`(entity_id)

`to_em_fq_flag`(adjust_type)

`to_em_level_flag`(level)

`to_em_sec_id`(entity_id)

`to_zvt_code`(code)

---

zvt.recorders.em.em_api.actor_type_to_org_type

zvt.recorders.em.em_api.actor_type_to_org_type(*actor_type: ActorType*)

zvt.recorders.em.em_api.generate_filters

zvt.recorders.em.em_api.generate_filters(*code=None*, *trade_date=None*, *report_date=None*, *end_date=None*, *org_type=None*, *field_op: Optional[dict] = None*)

zvt.recorders.em.em_api.get_basic_info

zvt.recorders.em.em_api.get_basic_info(*entity_id*)

zvt.recorders.em.em_api.get_block_stocks

zvt.recorders.em.em_api.get_block_stocks(*block_id*, *name=''*)

zvt.recorders.em.em_api.get_controlling_shareholder

zvt.recorders.em.em_api.get_controlling_shareholder(*code*)

zvt.recorders.em.em_api.get_dragon_and_tiger

zvt.recorders.em.em_api.get_dragon_and_tiger(*code*, *start_date=None*)

zvt.recorders.em.em_api.get_dragon_and_tiger_list

zvt.recorders.em.em_api.get_dragon_and_tiger_list(*start_date*, *end_date=None*)

zvt.recorders.em.em_api.get_em_data

zvt.recorders.em.em_api.get_em_data(*request_type*, *fields*, *session=None*, *source='SECURITIES'*, *filters=None*, *sort_by=''*, *sort='asc'*, *pn=1*, *ps=2000*, *fetch_all=True*, *fetch_count=1*, *params=None*)

zvt.recorders.em.em_api.get_exchange

zvt.recorders.em.em_api.get_exchange(*code*)

zvt.recorders.em.em_api.get_free_holder_report_dates

zvt.recorders.em.em_api.get_free_holder_report_dates(*code*)

zvt.recorders.em.em_api.get_free_holders

zvt.recorders.em.em_api.get_free_holders(*code*, *end_date*)

zvt.recorders.em.em_api.get_future_list

zvt.recorders.em.em_api.get_future_list()

zvt.recorders.em.em_api.get_holder_report_dates

zvt.recorders.em.em_api.get_holder_report_dates(*code*)

zvt.recorders.em.em_api.get_holders

zvt.recorders.em.em_api.get_holders(*code*, *end_date*)

zvt.recorders.em.em_api.get_ii_holder

zvt.recorders.em.em_api.get_ii_holder(*code*, *report_date*, *org_type*)

zvt.recorders.em.em_api.get_ii_holder_report_dates

zvt.recorders.em.em_api.get_ii_holder_report_dates(*code*)

zvt.recorders.em.em_api.get_ii_summary

zvt.recorders.em.em_api.get_ii_summary(*code*, *report_date*, *org_type*)

zvt.recorders.em.em_api.get_kdata

zvt.recorders.em.em_api.get_kdata(*entity_id*, *session=None*, *level=IntervalLevel.LEVEL_1DAY*, *adjust_type=AdjustType.qfq*, *limit=10000*)

zvt.recorders.em.em_api.get_news

zvt.recorders.em.em_api.get_news(*entity_id*, *ps=200*, *index=1*, *start_timestamp=None*, *session=None*, *latest_code=None*)

zvt.recorders.em.em_api.get_quotes

zvt.recorders.em.em_api.get_quotes()

zvt.recorders.em.em_api.get_tradable_list

zvt.recorders.em.em_api.get_tradable_list(*entity_type: Union[TradableType, str] = 'stock', exchange: Optional[Union[Exchange, str]] = None, limit: int = 10000, hk_south=False, block_category=BlockCategory.concept*)

zvt.recorders.em.em_api.get_treasury_yield

zvt.recorders.em.em_api.get_treasury_yield(*pn=1*, *ps=2000*, *fetch_all=True*)

zvt.recorders.em.em_api.get_url

zvt.recorders.em.em_api.get_url(*type*, *sty*, *source='SECURITIES'*, *filters=None*, *order_by=''*, *order='asc'*, *pn=1*, *ps=2000*, *params=None*)

zvt.recorders.em.em_api.to_em_entity_flag

zvt.recorders.em.em_api.to_em_entity_flag(*exchange: Union[Exchange, str]*)

zvt.recorders.em.em_api.to_em_fc

zvt.recorders.em.em_api.to_em_fc(*entity_id*)

zvt.recorders.em.em_api.to_em_fq_flag

zvt.recorders.em.em_api.to_em_fq_flag(*adjust_type: AdjustType*)

zvt.recorders.em.em_api.to_em_level_flag

zvt.recorders.em.em_api.to_em_level_flag(*level: IntervalLevel*)

zvt.recorders.em.em_api.to_em_sec_id

zvt.recorders.em.em_api.to_em_sec_id(*entity_id*)

zvt.recorders.em.em_api.to_zvt_code

zvt.recorders.em.em_api.to_zvt_code(*code*)

zvt.recorders.em.macro

Modules

zvt.recorders.em.macro.em_treasury_yield_recorder

---

zvt.recorders.em.macro.em_treasury_yield_recorder

Classes

`EMTreasuryYieldRecorder`([force_update, ...])

EMTreasuryYieldRecorder

class zvt.recorders.em.macro.em_treasury_yield_recorder.EMTreasuryYieldRecorder(*force_update=True*, *sleeping_time=10*, *entity_filters=None*, *ignore_failed=True*, *real_time=False*, *fix_duplicate_way='ignore'*, *start_timestamp=None*, *end_timestamp=None*, *level=IntervalLevel.LEVEL_1DAY*, *kdata_use_begin_time=False*, *one_day_trading_minutes=1440*, *return_unfinished=False*)

Bases: `FixedCycleDataRecorder`

__init__(*force_update=True*, *sleeping_time=10*, *entity_filters=None*, *ignore_failed=True*, *real_time=False*, *fix_duplicate_way='ignore'*, *start_timestamp=None*, *end_timestamp=None*, *level=IntervalLevel.LEVEL_1DAY*, *kdata_use_begin_time=False*, *one_day_trading_minutes=1440*, *return_unfinished=False*) → None

Parameters::

- **code** –
- **ignore_failed** –
- **entity_filters** –
- **exchanges** –
- **entity_id** – for record single entity
- **entity_ids** – set entity_ids or (entity_type,exchanges,codes)
- **codes** –
- **day_data** – one record per day,set to True if you want skip recording it when data of today exist
- **force_update** –
- **sleeping_time** –

entity_schema

alias of `Country`

data_schema

alias of `TreasuryYield`

entity_provider*: str = 'wb'*

overwrite them to fetch the entity list

provider*: str = 'em'*

overwrite them to set up the data you want to record

record(*entity*, *start*, *end*, *size*, *timestamps*)

implement the recording logic in this method, should return json or domain list

Parameters::

- **entity** –
- **start** –
- **end** –
- **size** –
- **timestamps** –

clear_state_data(*entity_id=None*)

clear state of the entity

Parameters:: **entity_id** – entity id

decode_state(*state: str*)

decode state

Parameters:: **state** –

Returns::

encode_state(*state: object*)

encode state

Parameters:: **state** –

Returns::

generate_domain(*entity*, *original_data*)

generate the data_schema instance using entity and original_data,the original_data is from record result

Parameters::

- **entity** –
- **original_data** –

generate_domain_id(*entity*, *original_data*, *time_fmt='YYYY-MM-DD'*)

generate domain id from the entity and original data,the default id meaning:entity + event happen time

Parameters::

- **entity** –
- **original_data** –
- **time_fmt** –

Returns::

Return type::

get_data_map()

{'original_field':('domain_field',transform_func)}

get_evaluated_time_field()

the timestamp field for evaluating time range of recorder,used in get_latest_saved_record

init_entities()

init the entities which we would record data for

name = *None*

name of the service, default name of class if not set manually

original_page_url = *None*

original page url

persist(*entity*, *domain_list*)

persist the domain list to db

Parameters::

- **entity** –
- **domain_list** –

state_schema

alias of `RecorderState`

url = *None*

request url

exchanges

setup the entities you want to record

entity_ids

set entity_ids or (entity_type,exchanges,codes)

session

using to do db operations

zvt.recorders.em.meta

Modules

zvt.recorders.em.meta.em_block_meta_recorder

---

zvt.recorders.em.meta.em_currency_meta_recorder

---

zvt.recorders.em.meta.em_future_meta_recorder

---

zvt.recorders.em.meta.em_index_meta_recorder

---

zvt.recorders.em.meta.em_indexus_meta_recorder

---

zvt.recorders.em.meta.em_stock_meta_recorder

---

zvt.recorders.em.meta.em_stockhk_meta_recorder

---

zvt.recorders.em.meta.em_stockus_meta_recorder

---

zvt.recorders.em.meta.em_block_meta_recorder

Classes

`EMBlockRecorder`([force_update, sleeping_time])

`EMBlockStockRecorder`([force_update, ...])

EMBlockRecorder

class zvt.recorders.em.meta.em_block_meta_recorder.EMBlockRecorder(*force_update: bool = False*, *sleeping_time: int = 10*)

Bases: `Recorder`

__init__(*force_update: bool = False*, *sleeping_time: int = 10*) → None

provider*: str = 'em'*

overwrite them to set up the data you want to record

data_schema

alias of `Block`

clear_state_data(*entity_id=None*)

clear state of the entity

Parameters:: **entity_id** – entity id

decode_state(*state: str*)

decode state

Parameters:: **state** –

Returns::

encode_state(*state: object*)

encode state

Parameters:: **state** –

Returns::

name = *None*

name of the service, default name of class if not set manually

original_page_url = *None*

original page url

state_schema

alias of `RecorderState`

url = *None*

request url

session

using to do db operations

EMBlockStockRecorder

class zvt.recorders.em.meta.em_block_meta_recorder.EMBlockStockRecorder(*force_update=False*, *sleeping_time=5*, *exchanges=None*, *entity_id=None*, *entity_ids=None*, *code=None*, *codes=None*, *day_data=False*, *entity_filters=None*, *ignore_failed=True*, *real_time=False*, *fix_duplicate_way='add'*, *start_timestamp=None*, *end_timestamp=None*, *return_unfinished=False*)

Bases: `TimeSeriesDataRecorder`

__init__(*force_update=False*, *sleeping_time=5*, *exchanges=None*, *entity_id=None*, *entity_ids=None*, *code=None*, *codes=None*, *day_data=False*, *entity_filters=None*, *ignore_failed=True*, *real_time=False*, *fix_duplicate_way='add'*, *start_timestamp=None*, *end_timestamp=None*, *return_unfinished=False*) → None

Parameters::

- **code** –
- **ignore_failed** –
- **entity_filters** –
- **exchanges** –
- **entity_id** – for record single entity
- **entity_ids** – set entity_ids or (entity_type,exchanges,codes)
- **codes** –
- **day_data** – one record per day,set to True if you want skip recording it when data of today exist
- **force_update** –
- **sleeping_time** –

entity_provider*: str = 'em'*

overwrite them to fetch the entity list

entity_schema

alias of `Block`

provider*: str = 'em'*

overwrite them to set up the data you want to record

data_schema

alias of `BlockStock`

record(*entity*, *start*, *end*, *size*, *timestamps*)

implement the recording logic in this method, should return json or domain list

Parameters::

- **entity** –
- **start** –
- **end** –
- **size** –
- **timestamps** –

clear_state_data(*entity_id=None*)

clear state of the entity

Parameters:: **entity_id** – entity id

decode_state(*state: str*)

decode state

Parameters:: **state** –

Returns::

encode_state(*state: object*)

encode state

Parameters:: **state** –

Returns::

generate_domain(*entity*, *original_data*)

generate the data_schema instance using entity and original_data,the original_data is from record result

Parameters::

- **entity** –
- **original_data** –

generate_domain_id(*entity*, *original_data*, *time_fmt='YYYY-MM-DD'*)

generate domain id from the entity and original data,the default id meaning:entity + event happen time

Parameters::

- **entity** –
- **original_data** –
- **time_fmt** –

Returns::

Return type::

get_data_map()

{'original_field':('domain_field',transform_func)}

get_evaluated_time_field()

the timestamp field for evaluating time range of recorder,used in get_latest_saved_record

init_entities()

init the entities which we would record data for

name = *None*

name of the service, default name of class if not set manually

original_page_url = *None*

original page url

persist(*entity*, *domain_list*)

persist the domain list to db

Parameters::

- **entity** –
- **domain_list** –

state_schema

alias of `RecorderState`

url = *None*

request url

exchanges

setup the entities you want to record

entity_ids

set entity_ids or (entity_type,exchanges,codes)

session

using to do db operations

zvt.recorders.em.meta.em_currency_meta_recorder

Classes

`EMCurrencyRecorder`([force_update, sleeping_time])

EMCurrencyRecorder

class zvt.recorders.em.meta.em_currency_meta_recorder.EMCurrencyRecorder(*force_update: bool = False*, *sleeping_time: int = 10*)

Bases: `Recorder`

__init__(*force_update: bool = False*, *sleeping_time: int = 10*) → None

provider*: str = 'em'*

overwrite them to set up the data you want to record

data_schema

alias of `Currency`

clear_state_data(*entity_id=None*)

clear state of the entity

Parameters:: **entity_id** – entity id

decode_state(*state: str*)

decode state

Parameters:: **state** –
Returns::

encode_state(*state: object*)

encode state

Parameters:: **state** –
Returns::

name = *None*

name of the service, default name of class if not set manually

original_page_url = *None*

original page url

state_schema

alias of `RecorderState`

url = *None*

request url

session

using to do db operations

zvt.recorders.em.meta.em_future_meta_recorder

Classes

`EMFutureRecorder`([force_update, sleeping_time])

EMFutureRecorder

class zvt.recorders.em.meta.em_future_meta_recorder.EMFutureRecorder(*force_update: bool = False*, *sleeping_time: int = 10*)

Bases: `Recorder`

__init__(*force_update: bool = False*, *sleeping_time: int = 10*) → None

provider*: str = 'em'*

overwrite them to set up the data you want to record

data_schema

alias of `Future`

clear_state_data(*entity_id=None*)

clear state of the entity

Parameters:: **entity_id** – entity id

decode_state(*state: str*)

decode state

Parameters:: **state** –
Returns::

encode_state(*state: object*)

encode state

Parameters:: **state** –
Returns::

name = *None*

name of the service, default name of class if not set manually

original_page_url = *None*

original page url

state_schema

alias of `RecorderState`

url = *None*

request url

session

using to do db operations

zvt.recorders.em.meta.em_index_meta_recorder

Classes

`EMIndexRecorder`([force_update, sleeping_time])

EMIndexRecorder

class zvt.recorders.em.meta.em_index_meta_recorder.EMIndexRecorder(*force_update: bool = False*, *sleeping_time: int = 10*)

Bases: `Recorder`

__init__(*force_update: bool = False*, *sleeping_time: int = 10*) → None

provider*: str = 'em'*

overwrite them to set up the data you want to record

data_schema

alias of `Index`

clear_state_data(*entity_id=None*)

clear state of the entity

Parameters:: **entity_id** – entity id

decode_state(*state: str*)

decode state

Parameters:: **state** –
Returns::

encode_state(*state: object*)

encode state

Parameters:: **state** –
Returns::

name = *None*

name of the service, default name of class if not set manually

original_page_url = *None*

original page url

state_schema

alias of `RecorderState`

url = *None*

request url

session

using to do db operations

zvt.recorders.em.meta.em_indexus_meta_recorder

Classes

`EMIndexusRecorder`([force_update, sleeping_time])

EMIndexusRecorder

class zvt.recorders.em.meta.em_indexus_meta_recorder.EMIndexusRecorder(*force_update: bool = False*, *sleeping_time: int = 10*)

Bases: `Recorder`

__init__(*force_update: bool = False*, *sleeping_time: int = 10*) → None

provider*: str = 'em'*

overwrite them to set up the data you want to record

data_schema

alias of `Indexus`

clear_state_data(*entity_id=None*)

clear state of the entity

Parameters:: **entity_id** – entity id

decode_state(*state: str*)

decode state

Parameters:: **state** –
Returns::

encode_state(*state: object*)

encode state

Parameters:: **state** –
Returns::

name = *None*

name of the service, default name of class if not set manually

original_page_url = *None*

original page url

state_schema

alias of `RecorderState`

url = *None*

request url

session

using to do db operations

zvt.recorders.em.meta.em_stock_meta_recorder

Classes

`EMStockRecorder`([force_update, sleeping_time])

EMStockRecorder

class zvt.recorders.em.meta.em_stock_meta_recorder.EMStockRecorder(*force_update: bool = False*, *sleeping_time: int = 10*)

Bases: `Recorder`

__init__(*force_update: bool = False*, *sleeping_time: int = 10*) → None

provider*: str = 'em'*

overwrite them to set up the data you want to record

data_schema

alias of `Stock`

clear_state_data(*entity_id=None*)

clear state of the entity

Parameters:: **entity_id** – entity id

decode_state(*state: str*)

decode state

Parameters:: **state** –

Returns::

encode_state(*state: object*)

encode state

Parameters:: **state** –

Returns::

name = *None*

name of the service, default name of class if not set manually

original_page_url = *None*

original page url

state_schema

alias of `RecorderState`

url = *None*

request url

session

using to do db operations

zvt.recorders.em.meta.em_stockhk_meta_recorder

Classes

`EMStockhkRecorder`([force_update, sleeping_time])

EMStockhkRecorder

class zvt.recorders.em.meta.em_stockhk_meta_recorder.EMStockhkRecorder(*force_update: bool = False*, *sleeping_time: int = 10*)

Bases: `Recorder`

__init__(*force_update: bool = False*, *sleeping_time: int = 10*) → None

provider*: str = 'em'*

overwrite them to set up the data you want to record

data_schema

alias of `Stockhk`

clear_state_data(*entity_id=None*)

clear state of the entity

Parameters:: **entity_id** – entity id

decode_state(*state: str*)

decode state

Parameters:: **state** –
Returns::

encode_state(*state: object*)

encode state

Parameters:: **state** –
Returns::

name = *None*

name of the service, default name of class if not set manually

original_page_url = *None*

original page url

state_schema

alias of `RecorderState`

url = *None*

request url

session

using to do db operations

zvt.recorders.em.meta.em_stockus_meta_recorder

Classes

`EMStockusRecorder`([force_update, sleeping_time])

EMStockusRecorder

class zvt.recorders.em.meta.em_stockus_meta_recorder.EMStockusRecorder(*force_update: bool = False*, *sleeping_time: int = 10*)

Bases: `Recorder`

__init__(*force_update: bool = False*, *sleeping_time: int = 10*) → None

provider*: str = 'em'*

overwrite them to set up the data you want to record

data_schema

alias of `Stockus`

clear_state_data(*entity_id=None*)

clear state of the entity

Parameters:: **entity_id** – entity id

decode_state(*state: str*)

decode state

Parameters:: **state** –
Returns::

encode_state(*state: object*)

encode state

Parameters:: **state** –
Returns::

name = *None*

name of the service, default name of class if not set manually

original_page_url = *None*

original page url

state_schema

alias of `RecorderState`

url = *None*

request url

session

using to do db operations

zvt.recorders.em.misc

Modules

zvt.recorders.em.misc.em_stock_news_recorder

---

zvt.recorders.em.misc.em_stock_news_recorder

Classes

`EMStockNewsRecorder`([force_update, ...])

EMStockNewsRecorder

class zvt.recorders.em.misc.em_stock_news_recorder.EMStockNewsRecorder(*force_update=True*, *sleeping_time=10*, *exchanges=None*, *entity_id=None*, *entity_ids=None*, *code=None*, *codes=None*, *day_data=False*, *entity_filters=None*, *ignore_failed=True*, *real_time=False*, *fix_duplicate_way='ignore'*, *start_timestamp=None*, *end_timestamp=None*, *level=IntervalLevel.LEVEL_1DAY*, *kdata_use_begin_time=False*, *one_day_trading_minutes=1440*, *return_unfinished=False*)

Bases: `FixedCycleDataRecorder`

__init__(*force_update=True*, *sleeping_time=10*, *exchanges=None*, *entity_id=None*, *entity_ids=None*, *code=None*, *codes=None*, *day_data=False*, *entity_filters=None*, *ignore_failed=True*, *real_time=False*, *fix_duplicate_way='ignore'*, *start_timestamp=None*, *end_timestamp=None*, *level=IntervalLevel.LEVEL_1DAY*, *kdata_use_begin_time=False*, *one_day_trading_minutes=1440*, *return_unfinished=False*) → None

Parameters::

- **code** –
- **ignore_failed** –
- **entity_filters** –
- **exchanges** –
- **entity_id** – for record single entity
- **entity_ids** – set entity_ids or (entity_type,exchanges,codes)
- **codes** –
- **day_data** – one record per day,set to True if you want skip recording it when data of today exist
- **force_update** –
- **sleeping_time** –

original_page_url = *'https://wap.eastmoney.com/quote/stock/0.002572.html'*

original page url

url = *'https://np-listapi.eastmoney.com/comm/wap/getListInfo?cb=callback&client=wap&type=1&mTypeAndCode=0.002572&pageSize=200&pageIndex={}&callback=jQuery18300174782479067403 52_1644568731256&_=1644568879493'*

request url

entity_schema

alias of `Stock`

data_schema

alias of `StockNews`

entity_provider*: str = 'em'*

overwrite them to fetch the entity list

provider*: str = 'em'*

overwrite them to set up the data you want to record

record(*entity*, *start*, *end*, *size*, *timestamps*)

implement the recording logic in this method, should return json or domain list

Parameters::

- **entity** –
- **start** –
- **end** –
- **size** –
- **timestamps** –

clear_state_data(*entity_id=None*)

clear state of the entity

Parameters:: **entity_id** – entity id

decode_state(*state: str*)

decode state

Parameters:: **state** –
Returns::

encode_state(*state: object*)

encode state

Parameters:: **state** –
Returns::

generate_domain(*entity*, *original_data*)

generate the data_schema instance using entity and original_data,the original_data is from record result

Parameters::
- **entity** –
- **original_data** –

generate_domain_id(*entity*, *original_data*, *time_fmt='YYYY-MM-DD'*)

generate domain id from the entity and original data,the default id meaning:entity + event happen time

Parameters::
- **entity** –
- **original_data** –
- **time_fmt** –

Returns::
Return type::

get_data_map()

{'original_field':('domain_field',transform_func)}

get_evaluated_time_field()

the timestamp field for evaluating time range of recorder,used in get_latest_saved_record

init_entities()

init the entities which we would record data for

name = *None*

name of the service, default name of class if not set manually

persist(*entity*, *domain_list*)

persist the domain list to db

Parameters::

- **entity** –
- **domain_list** –

state_schema

alias of `RecorderState`

exchanges

setup the entities you want to record

entity_ids

set entity_ids or (entity_type,exchanges,codes)

session

using to do db operations

zvt.recorders.em.quotes

Modules

zvt.recorders.em.quotes.em_kdata_recorder

---

zvt.recorders.em.quotes.em_kdata_recorder

Classes

BaseEMStockKdataRecorder([force_update, ...])

EMBlockKdataRecorder([force_update, ...])

EMCurrencyKdataRecorder([force_update, ...])

EMFutureKdataRecorder([force_update, ...])

EMIndexKdataRecorder([force_update, ...])

EMIndexusKdataRecorder([force_update, ...])

EMStockKdataRecorder([force_update, ...])

EMStockhkKdataRecorder([force_update, ...])

EMStockusKdataRecorder([force_update, ...])

BaseEMStockKdataRecorder

class zvt.recorders.em.quotes.em_kdata_recorder.BaseEMStockKdataRecorder(*force_update=True*, *sleeping_time=10*, *exchanges=None*, *entity_id=None*, *entity_ids=None*, *code=None*, *codes=None*, *day_data=False*, *entity_filters=None*, *ignore_failed=True*, *real_time=False*, *fix_duplicate_way='ignore'*, *start_timestamp=None*, *end_timestamp=None*, *level=IntervalLevel.LEVEL_1DAY*, *kdata_use_begin_time=False*, *one_day_trading_minutes=1440*, *adjust_type=AdjustType.qfq*, *return_unfinished=False*)

Bases: `FixedCycleDataRecorder`

__init__(*force_update=True*, *sleeping_time=10*, *exchanges=None*, *entity_id=None*, *entity_ids=None*, *code=None*, *codes=None*, *day_data=False*, *entity_filters=None*, *ignore_failed=True*, *real_time=False*, *fix_duplicate_way='ignore'*, *start_timestamp=None*, *end_timestamp=None*, *level=IntervalLevel.LEVEL_1DAY*, *kdata_use_begin_time=False*, *one_day_trading_minutes=1440*, *adjust_type=AdjustType.qfq*, *return_unfinished=False*) → None

Parameters::

- **code** –
- **ignore_failed** –
- **entity_filters** –
- **exchanges** –
- **entity_id** – for record single entity
- **entity_ids** – set entity_ids or (entity_type,exchanges,codes)
- **codes** –
- **day_data** – one record per day,set to True if you want skip recording it when data of today exist
- **force_update** –
- **sleeping_time** –

entity_provider*: str = 'em'*

overwrite them to fetch the entity list

provider*: str = 'em'*

overwrite them to set up the data you want to record

record(*entity*, *start*, *end*, *size*, *timestamps*)

implement the recording logic in this method, should return json or domain list

Parameters::

- **entity** –
- **start** –
- **end** –
- **size** –
- **timestamps** –

clear_state_data(*entity_id=None*)

clear state of the entity

Parameters:: **entity_id** – entity id

decode_state(*state: str*)

decode state

Parameters:: **state** –

Returns::

encode_state(*state: object*)

encode state

Parameters:: **state** –

Returns::

generate_domain(*entity*, *original_data*)

generate the data_schema instance using entity and original_data,the original_data is from record result

Parameters::

- **entity** –
- **original_data** –

generate_domain_id(*entity*, *original_data*, *time_fmt='YYYY-MM-DD'*)

generate domain id from the entity and original data,the default id meaning:entity + event happen time

Parameters::
- **entity** –
- **original_data** –
- **time_fmt** –

Returns::

Return type::

get_data_map()

{'original_field':('domain_field',transform_func)}

get_evaluated_time_field()

the timestamp field for evaluating time range of recorder,used in get_latest_saved_record

init_entities()

init the entities which we would record data for

name = *None*

name of the service, default name of class if not set manually

original_page_url = *None*

original page url

persist(*entity*, *domain_list*)

persist the domain list to db

Parameters::
- **entity** –
- **domain_list** –

state_schema

alias of `RecorderState`

url = *None*

request url

exchanges

setup the entities you want to record

entity_ids

set entity_ids or (entity_type,exchanges,codes)

session

using to do db operations

EMBlockKdataRecorder

class zvt.recorders.em.quotes.em_kdata_recorder.EMBlockKdataRecorder(*force_update=True*, *sleeping_time=10*, *exchanges=None*, *entity_id=None*, *entity_ids=None*, *code=None*, *codes=None*, *day_data=False*, *entity_filters=None*, *ignore_failed=True*, *real_time=False*, *fix_duplicate_way='ignore'*, *start_timestamp=None*, *end_timestamp=None*, *level=IntervalLevel.LEVEL_1DAY*, *kdata_use_begin_time=False*, *one_day_trading_minutes=1440*, *adjust_type=AdjustType.qfq*, *return_unfinished=False*)

Bases: `BaseEMStockKdataRecorder`

__init__(*force_update=True*, *sleeping_time=10*, *exchanges=None*, *entity_id=None*, *entity_ids=None*, *code=None*, *codes=None*, *day_data=False*, *entity_filters=None*, *ignore_failed=True*, *real_time=False*, *fix_duplicate_way='ignore'*, *start_timestamp=None*, *end_timestamp=None*, *level=IntervalLevel.LEVEL_1DAY*, *kdata_use_begin_time=False*, *one_day_trading_minutes=1440*, *adjust_type=AdjustType.qfq*, *return_unfinished=False*) → None

Parameters::

- **code** –
- **ignore_failed** –
- **entity_filters** –
- **exchanges** –
- **entity_id** – for record single entity
- **entity_ids** – set entity_ids or (entity_type,exchanges,codes)
- **codes** –
- **day_data** – one record per day,set to True if you want skip recording it when data of today exist
- **force_update** –
- **sleeping_time** –

entity_provider*: str = 'em'*

overwrite them to fetch the entity list

entity_schema

alias of `Block`

data_schema

alias of `BlockKdataCommon`

clear_state_data(*entity_id=None*)

clear state of the entity

Parameters:: **entity_id** – entity id

decode_state(*state: str*)

decode state

Parameters:: **state** –

Returns::

encode_state(*state: object*)

encode state

Parameters:: **state** –

Returns::

generate_domain(*entity*, *original_data*)

generate the data_schema instance using entity and original_data,the original_data is from record result

Parameters::

- **entity** –
- **original_data** –

generate_domain_id(*entity*, *original_data*, *time_fmt='YYYY-MM-DD'*)

generate domain id from the entity and original data,the default id meaning:entity + event happen time

Parameters::

- **entity** –
- **original_data** –
- **time_fmt** –

Returns::
Return type::

get_data_map()
{'original_field':('domain_field',transform_func)}

get_evaluated_time_field()
the timestamp field for evaluating time range of recorder,used in get_latest_saved_record

init_entities()
init the entities which we would record data for

name = *None*
name of the service, default name of class if not set manually

original_page_url = *None*
original page url

persist(*entity*, *domain_list*)
persist the domain list to db

Parameters::
- **entity** –
- **domain_list** –

provider*: str = 'em'*
overwrite them to set up the data you want to record

record(*entity*, *start*, *end*, *size*, *timestamps*)
implement the recording logic in this method, should return json or domain list

Parameters::
- **entity** –
- **start** –
- **end** –
- **size** –
- **timestamps** –

state_schema

alias of `RecorderState`

url = *None*

request url

exchanges

setup the entities you want to record

entity_ids

set entity_ids or (entity_type,exchanges,codes)

session

using to do db operations

EMCurrencyKdataRecorder

class zvt.recorders.em.quotes.em_kdata_recorder.EMCurrencyKdataRecorder(*force_update=True*, *sleeping_time=10*, *exchanges=None*, *entity_id=None*, *entity_ids=None*, *code=None*, *codes=None*, *day_data=False*, *entity_filters=None*, *ignore_failed=True*, *real_time=False*, *fix_duplicate_way='ignore'*, *start_timestamp=None*, *end_timestamp=None*, *level=IntervalLevel.LEVEL_1DAY*, *kdata_use_begin_time=False*, *one_day_trading_minutes=1440*, *adjust_type=AdjustType.qfq*, *return_unfinished=False*)

Bases: `BaseEMStockKdataRecorder`

__init__(*force_update=True*, *sleeping_time=10*, *exchanges=None*, *entity_id=None*, *entity_ids=None*, *code=None*, *codes=None*, *day_data=False*, *entity_filters=None*, *ignore_failed=True*, *real_time=False*, *fix_duplicate_way='ignore'*, *start_timestamp=None*, *end_timestamp=None*, *level=IntervalLevel.LEVEL_1DAY*, *kdata_use_begin_time=False*, *one_day_trading_minutes=1440*, *adjust_type=AdjustType.qfq*, *return_unfinished=False*) → None

Parameters::

- **code** –
- **ignore_failed** –
- **entity_filters** –
- **exchanges** –
- **entity_id** – for record single entity
- **entity_ids** – set entity_ids or (entity_type,exchanges,codes)
- **codes** –
- **day_data** – one record per day,set to True if you want skip recording it when data of today exist
- **force_update** –
- **sleeping_time** –

entity_provider*: str = 'em'*

overwrite them to fetch the entity list

entity_schema

alias of `Currency`

data_schema

alias of `CurrencyKdataCommon`

clear_state_data(*entity_id=None*)

clear state of the entity

Parameters:: **entity_id** – entity id

decode_state(*state: str*)

decode state

Parameters:: **state** –

Returns::

encode_state(*state: object*)

encode state

Parameters:: **state** –

Returns::

generate_domain(*entity*, *original_data*)

generate the data_schema instance using entity and original_data,the original_data is from record result

Parameters::

- **entity** –
- **original_data** –

generate_domain_id(*entity*, *original_data*, *time_fmt='YYYY-MM-DD'*)

generate domain id from the entity and original data,the default id meaning:entity + event happen time

Parameters::

- **entity** –
- **original_data** –
- **time_fmt** –

Returns::
Return type::

get_data_map()
{'original_field':('domain_field',transform_func)}

get_evaluated_time_field()
the timestamp field for evaluating time range of recorder,used in get_latest_saved_record

init_entities()
init the entities which we would record data for

name = *None*
name of the service, default name of class if not set manually

original_page_url = *None*
original page url

persist(*entity*, *domain_list*)
persist the domain list to db

Parameters::
- **entity** –
- **domain_list** –

provider*: str = 'em'*
overwrite them to set up the data you want to record

record(*entity*, *start*, *end*, *size*, *timestamps*)
implement the recording logic in this method, should return json or domain list

Parameters::
- **entity** –
- **start** –
- **end** –
- **size** –
- **timestamps** –

state_schema

alias of `RecorderState`

url = *None*

request url

exchanges

setup the entities you want to record

entity_ids

set entity_ids or (entity_type,exchanges,codes)

session

using to do db operations

EMFutureKdataRecorder

class zvt.recorders.em.quotes.em_kdata_recorder.EMFutureKdataRecorder(*force_update=True*, *sleeping_time=10*, *exchanges=None*, *entity_id=None*, *entity_ids=None*, *code=None*, *codes=None*, *day_data=False*, *entity_filters=None*, *ignore_failed=True*, *real_time=False*, *fix_duplicate_way='ignore'*, *start_timestamp=None*, *end_timestamp=None*, *level=IntervalLevel.LEVEL_1DAY*, *kdata_use_begin_time=False*, *one_day_trading_minutes=1440*, *adjust_type=AdjustType.qfq*, *return_unfinished=False*)

Bases: `BaseEMStockKdataRecorder`

__init__(*force_update=True*, *sleeping_time=10*, *exchanges=None*, *entity_id=None*, *entity_ids=None*, *code=None*, *codes=None*, *day_data=False*, *entity_filters=None*, *ignore_failed=True*, *real_time=False*, *fix_duplicate_way='ignore'*, *start_timestamp=None*, *end_timestamp=None*, *level=IntervalLevel.LEVEL_1DAY*, *kdata_use_begin_time=False*, *one_day_trading_minutes=1440*, *adjust_type=AdjustType.qfq*, *return_unfinished=False*) → None

Parameters::

- **code** –
- **ignore_failed** –
- **entity_filters** –
- **exchanges** –
- **entity_id** – for record single entity
- **entity_ids** – set entity_ids or (entity_type,exchanges,codes)
- **codes** –
- **day_data** – one record per day,set to True if you want skip recording it when data of today exist
- **force_update** –
- **sleeping_time** –

entity_provider*: str = 'em'*

overwrite them to fetch the entity list

entity_schema

alias of `Future`

data_schema

alias of `FutureKdataCommon`

clear_state_data(*entity_id=None*)

clear state of the entity

Parameters:: **entity_id** – entity id

decode_state(*state: str*)

decode state

Parameters:: **state** –

Returns::

encode_state(*state: object*)

encode state

Parameters:: **state** –

Returns::

generate_domain(*entity*, *original_data*)

generate the data_schema instance using entity and original_data,the original_data is from record result

Parameters::

- **entity** –
- **original_data** –

generate_domain_id(*entity*, *original_data*, *time_fmt='YYYY-MM-DD'*)

generate domain id from the entity and original data,the default id meaning:entity + event happen time

Parameters::

- **entity** –
- **original_data** –
- **time_fmt** –

Returns::
Return type::

get_data_map()
{'original_field':('domain_field',transform_func)}

get_evaluated_time_field()
the timestamp field for evaluating time range of recorder,used in get_latest_saved_record

init_entities()
init the entities which we would record data for

name = *None*
name of the service, default name of class if not set manually

original_page_url = *None*
original page url

persist(*entity*, *domain_list*)
persist the domain list to db

Parameters::
- **entity** –
- **domain_list** –

provider*: str = 'em'*
overwrite them to set up the data you want to record

record(*entity*, *start*, *end*, *size*, *timestamps*)
implement the recording logic in this method, should return json or domain list

Parameters::
- **entity** –
- **start** –
- **end** –
- **size** –
- **timestamps** –

state_schema

alias of `RecorderState`

url = *None*

request url

exchanges

setup the entities you want to record

entity_ids

set entity_ids or (entity_type,exchanges,codes)

session

using to do db operations

EMIndexKdataRecorder

class zvt.recorders.em.quotes.em_kdata_recorder.EMIndexKdataRecorder(*force_update=True*, *sleeping_time=10*, *exchanges=None*, *entity_id=None*, *entity_ids=None*, *code=None*, *codes=None*, *day_data=False*, *entity_filters=None*, *ignore_failed=True*, *real_time=False*, *fix_duplicate_way='ignore'*, *start_timestamp=None*, *end_timestamp=None*, *level=IntervalLevel.LEVEL_1DAY*, *kdata_use_begin_time=False*, *one_day_trading_minutes=1440*, *adjust_type=AdjustType.qfq*, *return_unfinished=False*)

Bases: `BaseEMStockKdataRecorder`

__init__(*force_update=True*, *sleeping_time=10*, *exchanges=None*, *entity_id=None*, *entity_ids=None*, *code=None*, *codes=None*, *day_data=False*, *entity_filters=None*, *ignore_failed=True*, *real_time=False*, *fix_duplicate_way='ignore'*, *start_timestamp=None*, *end_timestamp=None*, *level=IntervalLevel.LEVEL_1DAY*, *kdata_use_begin_time=False*, *one_day_trading_minutes=1440*, *adjust_type=AdjustType.qfq*, *return_unfinished=False*) → None

Parameters::

- **code** –
- **ignore_failed** –
- **entity_filters** –
- **exchanges** –
- **entity_id** – for record single entity
- **entity_ids** – set entity_ids or (entity_type,exchanges,codes)
- **codes** –
- **day_data** – one record per day,set to True if you want skip recording it when data of today exist
- **force_update** –
- **sleeping_time** –

entity_provider*: str = 'em'*

overwrite them to fetch the entity list

entity_schema

alias of `Index`

data_schema

alias of `IndexKdataCommon`

clear_state_data(*entity_id=None*)

clear state of the entity

Parameters:: **entity_id** – entity id

decode_state(*state: str*)

decode state

Parameters:: **state** –

Returns::

encode_state(*state: object*)

encode state

Parameters:: **state** –

Returns::

generate_domain(*entity*, *original_data*)

generate the data_schema instance using entity and original_data,the original_data is from record result

Parameters::

- **entity** –
- **original_data** –

generate_domain_id(*entity*, *original_data*, *time_fmt='YYYY-MM-DD'*)

generate domain id from the entity and original data,the default id meaning:entity + event happen time

Parameters::

- **entity** –
- **original_data** –
- **time_fmt** –

Returns::
Return type::

get_data_map()
{'original_field':('domain_field',transform_func)}

get_evaluated_time_field()
the timestamp field for evaluating time range of recorder,used in get_latest_saved_record

init_entities()
init the entities which we would record data for

name = *None*
name of the service, default name of class if not set manually

original_page_url = *None*
original page url

persist(*entity*, *domain_list*)
persist the domain list to db

Parameters::
- **entity** –
- **domain_list** –

provider*: str = 'em'*
overwrite them to set up the data you want to record

record(*entity*, *start*, *end*, *size*, *timestamps*)
implement the recording logic in this method, should return json or domain list

Parameters::
- **entity** –
- **start** –
- **end** –
- **size** –
- **timestamps** –

state_schema

alias of `RecorderState`

url = *None*

request url

exchanges

setup the entities you want to record

entity_ids

set entity_ids or (entity_type,exchanges,codes)

session

using to do db operations

EMIndexusKdataRecorder

class zvt.recorders.em.quotes.em_kdata_recorder.EMIndexusKdataRecorder(*force_update=True*, *sleeping_time=10*, *exchanges=None*, *entity_id=None*, *entity_ids=None*, *code=None*, *codes=None*, *day_data=False*, *entity_filters=None*, *ignore_failed=True*, *real_time=False*, *fix_duplicate_way='ignore'*, *start_timestamp=None*, *end_timestamp=None*, *level=IntervalLevel.LEVEL_1DAY*, *kdata_use_begin_time=False*, *one_day_trading_minutes=1440*, *adjust_type=AdjustType.qfq*, *return_unfinished=False*)

Bases: `BaseEMStockKdataRecorder`

__init__(*force_update=True*, *sleeping_time=10*, *exchanges=None*, *entity_id=None*, *entity_ids=None*, *code=None*, *codes=None*, *day_data=False*, *entity_filters=None*, *ignore_failed=True*, *real_time=False*, *fix_duplicate_way='ignore'*, *start_timestamp=None*, *end_timestamp=None*, *level=IntervalLevel.LEVEL_1DAY*, *kdata_use_begin_time=False*, *one_day_trading_minutes=1440*, *adjust_type=AdjustType.qfq*, *return_unfinished=False*) → None

Parameters::

- **code** –
- **ignore_failed** –
- **entity_filters** –
- **exchanges** –
- **entity_id** – for record single entity
- **entity_ids** – set entity_ids or (entity_type,exchanges,codes)
- **codes** –
- **day_data** – one record per day,set to True if you want skip recording it when data of today exist
- **force_update** –
- **sleeping_time** –

entity_provider*: str = 'em'*

overwrite them to fetch the entity list

entity_schema

alias of `Indexus`

data_schema

alias of `IndexusKdataCommon`

clear_state_data(*entity_id=None*)

clear state of the entity

Parameters:: **entity_id** – entity id

decode_state(*state: str*)

decode state

Parameters:: **state** –

Returns::

encode_state(*state: object*)

encode state

Parameters:: **state** –

Returns::

generate_domain(*entity*, *original_data*)

generate the data_schema instance using entity and original_data,the original_data is from record result

Parameters::

- **entity** –
- **original_data** –

generate_domain_id(*entity*, *original_data*, *time_fmt='YYYY-MM-DD'*)

generate domain id from the entity and original data,the default id meaning:entity + event happen time

Parameters::

- **entity** –
- **original_data** –
- **time_fmt** –

Returns::
Return type::

get_data_map()
{'original_field':('domain_field',transform_func)}

get_evaluated_time_field()
the timestamp field for evaluating time range of recorder,used in get_latest_saved_record

init_entities()
init the entities which we would record data for

name = *None*
name of the service, default name of class if not set manually

original_page_url = *None*
original page url

persist(*entity*, *domain_list*)
persist the domain list to db

Parameters::
- **entity** –
- **domain_list** –

provider*: str = 'em'*
overwrite them to set up the data you want to record

record(*entity*, *start*, *end*, *size*, *timestamps*)
implement the recording logic in this method, should return json or domain list

Parameters::
- **entity** –
- **start** –
- **end** –
- **size** –
- **timestamps** –

state_schema

alias of `RecorderState`

url = *None*

request url

exchanges

setup the entities you want to record

entity_ids

set entity_ids or (entity_type,exchanges,codes)

session

using to do db operations

EMStockKdataRecorder

class zvt.recorders.em.quotes.em_kdata_recorder.EMStockKdataRecorder(*force_update=True*, *sleeping_time=10*, *exchanges=None*, *entity_id=None*, *entity_ids=None*, *code=None*, *codes=None*, *day_data=False*, *entity_filters=None*, *ignore_failed=True*, *real_time=False*, *fix_duplicate_way='ignore'*, *start_timestamp=None*, *end_timestamp=None*, *level=IntervalLevel.LEVEL_1DAY*, *kdata_use_begin_time=False*, *one_day_trading_minutes=1440*, *adjust_type=AdjustType.qfq*, *return_unfinished=False*)

Bases: `BaseEMStockKdataRecorder`

__init__(*force_update=True*, *sleeping_time=10*, *exchanges=None*, *entity_id=None*, *entity_ids=None*, *code=None*, *codes=None*, *day_data=False*, *entity_filters=None*, *ignore_failed=True*, *real_time=False*, *fix_duplicate_way='ignore'*, *start_timestamp=None*, *end_timestamp=None*, *level=IntervalLevel.LEVEL_1DAY*, *kdata_use_begin_time=False*, *one_day_trading_minutes=1440*, *adjust_type=AdjustType.qfq*, *return_unfinished=False*) → None

Parameters::

- **code** –
- **ignore_failed** –
- **entity_filters** –
- **exchanges** –
- **entity_id** – for record single entity
- **entity_ids** – set entity_ids or (entity_type,exchanges,codes)
- **codes** –
- **day_data** – one record per day,set to True if you want skip recording it when data of today exist
- **force_update** –
- **sleeping_time** –

entity_schema

alias of `Stock`

data_schema

alias of `StockKdataCommon`

clear_state_data(*entity_id=None*)

clear state of the entity

Parameters:: **entity_id** – entity id

decode_state(*state: str*)

decode state

Parameters:: **state** –

Returns::

encode_state(*state: object*)

encode state

Parameters:: **state** –

Returns::

entity_provider*: str = 'em'*

overwrite them to fetch the entity list

generate_domain(*entity*, *original_data*)

generate the data_schema instance using entity and original_data,the original_data is from record result

Parameters::

- **entity** –
- **original_data** –

generate_domain_id(*entity*, *original_data*, *time_fmt='YYYY-MM-DD'*)

generate domain id from the entity and original data,the default id meaning:entity + event happen time

Parameters::

- **entity** –
- **original_data** –
- **time_fmt** –

Returns::
Return type::

get_data_map()
{'original_field':('domain_field',transform_func)}

get_evaluated_time_field()
the timestamp field for evaluating time range of recorder,used in get_latest_saved_record

init_entities()
init the entities which we would record data for

name = *None*
name of the service, default name of class if not set manually

original_page_url = *None*
original page url

persist(*entity*, *domain_list*)
persist the domain list to db

Parameters::
- **entity** –
- **domain_list** –

provider*: str = 'em'*
overwrite them to set up the data you want to record

record(*entity*, *start*, *end*, *size*, *timestamps*)
implement the recording logic in this method, should return json or domain list

Parameters::
- **entity** –
- **start** –
- **end** –
- **size** –
- **timestamps** –

state_schema

alias of `RecorderState`

url = *None*

request url

exchanges

setup the entities you want to record

entity_ids

set entity_ids or (entity_type,exchanges,codes)

session

using to do db operations

EMStockhkKdataRecorder

class zvt.recorders.em.quotes.em_kdata_recorder.EMStockhkKdataRecorder(*force_update=True*, *sleeping_time=10*, *exchanges=None*, *entity_id=None*, *entity_ids=None*, *code=None*, *codes=None*, *day_data=False*, *entity_filters=None*, *ignore_failed=True*, *real_time=False*, *fix_duplicate_way='ignore'*, *start_timestamp=None*, *end_timestamp=None*, *level=IntervalLevel.LEVEL_1DAY*, *kdata_use_begin_time=False*, *one_day_trading_minutes=1440*, *adjust_type=AdjustType.qfq*, *return_unfinished=False*)

Bases: `BaseEMStockKdataRecorder`

__init__(*force_update=True*, *sleeping_time=10*, *exchanges=None*, *entity_id=None*, *entity_ids=None*, *code=None*, *codes=None*, *day_data=False*, *entity_filters=None*, *ignore_failed=True*, *real_time=False*, *fix_duplicate_way='ignore'*, *start_timestamp=None*, *end_timestamp=None*, *level=IntervalLevel.LEVEL_1DAY*, *kdata_use_begin_time=False*, *one_day_trading_minutes=1440*, *adjust_type=AdjustType.qfq*, *return_unfinished=False*) → None

Parameters::

- **code** –
- **ignore_failed** –
- **entity_filters** –
- **exchanges** –
- **entity_id** – for record single entity
- **entity_ids** – set entity_ids or (entity_type,exchanges,codes)
- **codes** –
- **day_data** – one record per day,set to True if you want skip recording it when data of today exist
- **force_update** –
- **sleeping_time** –

entity_provider*: str = 'em'*

overwrite them to fetch the entity list

entity_schema

alias of `Stockhk`

data_schema

alias of `StockhkKdataCommon`

clear_state_data(*entity_id=None*)

clear state of the entity

Parameters:: **entity_id** – entity id

decode_state(*state: str*)

decode state

Parameters:: **state** –

Returns::

encode_state(*state: object*)

encode state

Parameters:: **state** –

Returns::

generate_domain(*entity*, *original_data*)

generate the data_schema instance using entity and original_data,the original_data is from record result

Parameters::

- **entity** –
- **original_data** –

generate_domain_id(*entity*, *original_data*, *time_fmt='YYYY-MM-DD'*)

generate domain id from the entity and original data,the default id meaning:entity + event happen time

Parameters::

- **entity** –
- **original_data** –
- **time_fmt** –

Returns::
Return type::

get_data_map()
{'original_field':('domain_field',transform_func)}

get_evaluated_time_field()
the timestamp field for evaluating time range of recorder,used in get_latest_saved_record

init_entities()
init the entities which we would record data for

name = *None*
name of the service, default name of class if not set manually

original_page_url = *None*
original page url

persist(*entity*, *domain_list*)
persist the domain list to db

Parameters::
- **entity** –
- **domain_list** –

provider*: str = 'em'*
overwrite them to set up the data you want to record

record(*entity*, *start*, *end*, *size*, *timestamps*)
implement the recording logic in this method, should return json or domain list

Parameters::
- **entity** –
- **start** –
- **end** –
- **size** –
- **timestamps** –

state_schema

alias of `RecorderState`

url = *None*

request url

exchanges

setup the entities you want to record

entity_ids

set entity_ids or (entity_type,exchanges,codes)

session

using to do db operations

EMStockusKdataRecorder

class zvt.recorders.em.quotes.em_kdata_recorder.EMStockusKdataRecorder(*force_update=True*, *sleeping_time=10*, *exchanges=None*, *entity_id=None*, *entity_ids=None*, *code=None*, *codes=None*, *day_data=False*, *entity_filters=None*, *ignore_failed=True*, *real_time=False*, *fix_duplicate_way='ignore'*, *start_timestamp=None*, *end_timestamp=None*, *level=IntervalLevel.LEVEL_1DAY*, *kdata_use_begin_time=False*, *one_day_trading_minutes=1440*, *adjust_type=AdjustType.qfq*, *return_unfinished=False*)

Bases: `BaseEMStockKdataRecorder`

__init__(*force_update=True*, *sleeping_time=10*, *exchanges=None*, *entity_id=None*, *entity_ids=None*, *code=None*, *codes=None*, *day_data=False*, *entity_filters=None*, *ignore_failed=True*, *real_time=False*, *fix_duplicate_way='ignore'*, *start_timestamp=None*, *end_timestamp=None*, *level=IntervalLevel.LEVEL_1DAY*, *kdata_use_begin_time=False*, *one_day_trading_minutes=1440*, *adjust_type=AdjustType.qfq*, *return_unfinished=False*) → None

Parameters::

- **code** –
- **ignore_failed** –
- **entity_filters** –
- **exchanges** –
- **entity_id** – for record single entity
- **entity_ids** – set entity_ids or (entity_type,exchanges,codes)
- **codes** –
- **day_data** – one record per day,set to True if you want skip recording it when data of today exist
- **force_update** –
- **sleeping_time** –

entity_provider*: str = 'em'*

overwrite them to fetch the entity list

entity_schema

alias of `Stockus`

data_schema

alias of `StockusKdataCommon`

clear_state_data(*entity_id=None*)

clear state of the entity

Parameters:: **entity_id** – entity id

decode_state(*state: str*)

decode state

Parameters:: **state** –

Returns::

encode_state(*state: object*)

encode state

Parameters:: **state** –

Returns::

generate_domain(*entity*, *original_data*)

generate the data_schema instance using entity and original_data,the original_data is from record result

Parameters::

- **entity** –
- **original_data** –

generate_domain_id(*entity*, *original_data*, *time_fmt='YYYY-MM-DD'*)

generate domain id from the entity and original data,the default id meaning:entity + event happen time

Parameters::

- **entity** –
- **original_data** –
- **time_fmt** –

Returns::
Return type::

get_data_map()
{'original_field':('domain_field',transform_func)}

get_evaluated_time_field()
the timestamp field for evaluating time range of recorder,used in get_latest_saved_record

init_entities()
init the entities which we would record data for

name = *None*
name of the service, default name of class if not set manually

original_page_url = *None*
original page url

persist(*entity*, *domain_list*)
persist the domain list to db

Parameters::
- **entity** –
- **domain_list** –

provider*: str = 'em'*
overwrite them to set up the data you want to record

record(*entity*, *start*, *end*, *size*, *timestamps*)
implement the recording logic in this method, should return json or domain list

Parameters::
- **entity** –
- **start** –
- **end** –
- **size** –
- **timestamps** –

state_schema

alias of `RecorderState`

url = *None*

request url

exchanges

setup the entities you want to record

entity_ids

set entity_ids or (entity_type,exchanges,codes)

session

using to do db operations

zvt.recorders.em.trading

Modules

zvt.recorders.em.trading.em_dragon_and_tiger_recorder

---

zvt.recorders.em.trading.em_dragon_and_tiger_recorder

Classes

`EMDragonAndTigerRecorder`([force_update, ...])

EMDragonAndTigerRecorder

class zvt.recorders.em.trading.em_dragon_and_tiger_recorder.EMDragonAndTigerRecorder(*force_update=True*, *sleeping_time=10*, *exchanges=None*, *entity_id=None*, *entity_ids=None*, *code=None*, *codes=None*, *day_data=False*, *entity_filters=None*, *ignore_failed=True*, *real_time=False*, *fix_duplicate_way='ignore'*, *start_timestamp=None*, *end_timestamp=None*, *level=IntervalLevel.LEVEL_1DAY*, *kdata_use_begin_time=False*, *one_day_trading_minutes=1440*, *return_unfinished=False*)

Bases: `FixedCycleDataRecorder`

__init__(*force_update=True*, *sleeping_time=10*, *exchanges=None*, *entity_id=None*, *entity_ids=None*, *code=None*, *codes=None*, *day_data=False*, *entity_filters=None*, *ignore_failed=True*, *real_time=False*, *fix_duplicate_way='ignore'*, *start_timestamp=None*, *end_timestamp=None*, *level=IntervalLevel.LEVEL_1DAY*, *kdata_use_begin_time=False*, *one_day_trading_minutes=1440*, *return_unfinished=False*) → None

Parameters::

- **code** –
- **ignore_failed** –
- **entity_filters** –
- **exchanges** –
- **entity_id** – for record single entity
- **entity_ids** – set entity_ids or (entity_type,exchanges,codes)
- **codes** –
- **day_data** – one record per day,set to True if you want skip recording it when data of today exist
- **force_update** –
- **sleeping_time** –

entity_provider*: str = 'em'*

overwrite them to fetch the entity list

entity_schema

alias of `Stock`

provider*: str = 'em'*

overwrite them to set up the data you want to record

data_schema

alias of `DragonAndTiger`

record(*entity*, *start*, *end*, *size*, *timestamps*)

implement the recording logic in this method, should return json or domain list

Parameters::

- **entity** –
- **start** –
- **end** –
- **size** –
- **timestamps** –

clear_state_data(*entity_id=None*)

clear state of the entity

Parameters:: **entity_id** – entity id

decode_state(*state: str*)

decode state

Parameters:: **state** –

Returns::

encode_state(*state: object*)

encode state

Parameters:: **state** –

Returns::

generate_domain(*entity*, *original_data*)

generate the data_schema instance using entity and original_data,the original_data is from record result

Parameters::
- **entity** –
- **original_data** –

generate_domain_id(*entity*, *original_data*, *time_fmt='YYYY-MM-DD'*)

generate domain id from the entity and original data,the default id meaning:entity + event happen time

Parameters::
- **entity** –
- **original_data** –
- **time_fmt** –

Returns::

Return type::

get_data_map()

{'original_field':('domain_field',transform_func)}

get_evaluated_time_field()

the timestamp field for evaluating time range of recorder,used in get_latest_saved_record

init_entities()

init the entities which we would record data for

name = *None*

name of the service, default name of class if not set manually

original_page_url = *None*

original page url

persist(*entity*, *domain_list*)

persist the domain list to db

Parameters::
- **entity** –
- **domain_list** –

state_schema

alias of `RecorderState`

url = *None*

request url

exchanges

setup the entities you want to record

entity_ids

set entity_ids or (entity_type,exchanges,codes)

session

using to do db operations

zvt.recorders.exchange

Modules

zvt.recorders.exchange.api

---

zvt.recorders.exchange.exchange_etf_meta_recorder

---

zvt.recorders.exchange.exchange_index_recorder

---

zvt.recorders.exchange.exchange_index_stock_recorder

---

zvt.recorders.exchange.exchange_stock_meta_recorder

---

zvt.recorders.exchange.exchange_stock_summary_recorder

---

zvt.recorders.exchange.api

Modules

zvt.recorders.exchange.api.cn_index_api

---

zvt.recorders.exchange.api.cn_index_stock_api

---

zvt.recorders.exchange.api.cs_index_api

---

zvt.recorders.exchange.api.cs_index_stock_api

---

zvt.recorders.exchange.api.cn_index_api

Functions

`get_cn_index`([index_type, category])

---

zvt.recorders.exchange.api.cn_index_api.get_cn_index

zvt.recorders.exchange.api.cn_index_api.get_cn_index(*index_type='cni'*, *category=IndexCategory.style*)

zvt.recorders.exchange.api.cn_index_stock_api

Functions

`get_cn_index_stock`(code, timestamp[, name])

---

zvt.recorders.exchange.api.cn_index_stock_api.get_cn_index_stock

zvt.recorders.exchange.api.cn_index_stock_api.get_cn_index_stock(*code*, *timestamp*, *name=None*)

zvt.recorders.exchange.api.cs_index_api

Functions

`get_cs_index`([index_type])

---

zvt.recorders.exchange.api.cs_index_api.get_cs_index

zvt.recorders.exchange.api.cs_index_api.get_cs_index(*index_type='sh'*)

zvt.recorders.exchange.api.cs_index_stock_api

Functions

`get_cs_index_stock`(code, timestamp[, name])

---

zvt.recorders.exchange.api.cs_index_stock_api.get_cs_index_stock

zvt.recorders.exchange.api.cs_index_stock_api.get_cs_index_stock(*code*, *timestamp*, *name=None*)

zvt.recorders.exchange.exchange_etf_meta_recorder

Classes

`ChinaETFListSpider`([force_update, ...])

ChinaETFListSpider

class zvt.recorders.exchange.exchange_etf_meta_recorder.ChinaETFListSpider(*force_update=False*, *sleeping_time=10.0*, *provider='exchange'*)

Bases: `Recorder`

__init__(*force_update=False*, *sleeping_time=10.0*, *provider='exchange'*) → None

data_schema

alias of `EtfStock`

provider*: str = None*

overwrite them to set up the data you want to record

static populate_sh_etf_type(*df: DataFrame*)

填充沪市 ETF 代码对应的 TYPE 到列表数据中 :param df: ETF 列表数据 :return: 包含 ETF 对应 TYPE 的列表数据

static parse_sz_etf_underlying_index(*df: DataFrame*)

解析深市 ETF 对应跟踪的指数代码 :param df: ETF 列表数据 :return: 解析完成 ETF 对应指数代码的列表数据

clear_state_data(*entity_id=None*)

clear state of the entity

Parameters:: **entity_id** – entity id

decode_state(*state: str*)

decode state

Parameters:: **state** –

Returns::

encode_state(*state: object*)

encode state

Parameters:: **state** –

Returns::

name = *None*

name of the service, default name of class if not set manually

original_page_url = *None*

original page url

state_schema

alias of `RecorderState`

url = *None*

request url

session

using to do db operations

zvt.recorders.exchange.exchange_index_recorder

Classes

`ExchangeIndexRecorder`([force_update, ...])

ExchangeIndexRecorder

class zvt.recorders.exchange.exchange_index_recorder.ExchangeIndexRecorder(*force_update: bool = False*, *sleeping_time: int = 10*)

Bases: `Recorder`

__init__(*force_update: bool = False*, *sleeping_time: int = 10*) → None

provider*: str = 'exchange'*

overwrite them to set up the data you want to record

data_schema

alias of `Index`

clear_state_data(*entity_id=None*)

clear state of the entity

Parameters:: **entity_id** – entity id

decode_state(*state: str*)

decode state

Parameters:: **state** –
Returns::

encode_state(*state: object*)

encode state

Parameters:: **state** –
Returns::

name = *None*

name of the service, default name of class if not set manually

original_page_url = *None*

original page url

state_schema

alias of `RecorderState`

url = *None*

request url

session

using to do db operations

zvt.recorders.exchange.exchange_index_stock_recorder

Classes

`ExchangeIndexStockRecorder`([force_update, ...])

ExchangeIndexStockRecorder

class zvt.recorders.exchange.exchange_index_stock_recorder.ExchangeIndexStockRecorder(*force_update=False*, *sleeping_time=5*, *exchanges=None*, *entity_id=None*, *entity_ids=None*, *code=None*, *codes=None*, *day_data=False*, *entity_filters=None*, *ignore_failed=True*, *real_time=False*, *fix_duplicate_way='add'*, *start_timestamp=None*, *end_timestamp=None*, *record_history=False*)

Bases: `TimestampsDataRecorder`

__init__(*force_update=False*, *sleeping_time=5*, *exchanges=None*, *entity_id=None*, *entity_ids=None*, *code=None*, *codes=None*, *day_data=False*, *entity_filters=None*, *ignore_failed=True*, *real_time=False*, *fix_duplicate_way='add'*, *start_timestamp=None*, *end_timestamp=None*, *record_history=False*) → None

Parameters::

- **code** –
- **ignore_failed** –
- **entity_filters** –
- **exchanges** –
- **entity_id** – for record single entity
- **entity_ids** – set entity_ids or (entity_type,exchanges,codes)
- **codes** –
- **day_data** – one record per day,set to True if you want skip recording it when data of today exist
- **force_update** –
- **sleeping_time** –

entity_provider*: str = 'exchange'*

overwrite them to fetch the entity list

entity_schema

alias of `Index`

provider*: str = 'exchange'*

overwrite them to set up the data you want to record

data_schema

alias of `IndexStock`

record(*entity*, *start*, *end*, *size*, *timestamps*)

implement the recording logic in this method, should return json or domain list

Parameters::

- **entity** –
- **start** –
- **end** –
- **size** –
- **timestamps** –

clear_state_data(*entity_id=None*)

clear state of the entity

Parameters:: **entity_id** – entity id

decode_state(*state: str*)

decode state

Parameters:: **state** –

Returns::

encode_state(*state: object*)

encode state

Parameters:: **state** –

Returns::

generate_domain(*entity*, *original_data*)

generate the data_schema instance using entity and original_data,the original_data is from record result

Parameters::

- **entity** –
- **original_data** –

generate_domain_id(*entity*, *original_data*, *time_fmt='YYYY-MM-DD'*)

generate domain id from the entity and original data,the default id meaning:entity + event happen time

Parameters::

- **entity** –
- **original_data** –
- **time_fmt** –

Returns::

Return type::

get_data_map()

{'original_field':('domain_field',transform_func)}

get_evaluated_time_field()

the timestamp field for evaluating time range of recorder,used in get_latest_saved_record

init_entities()

init the entities which we would record data for

name = *None*

name of the service, default name of class if not set manually

original_page_url = *None*

original page url

persist(*entity*, *domain_list*)

persist the domain list to db

Parameters::

- **entity** –
- **domain_list** –

state_schema

alias of `RecorderState`

url = *None*

request url

exchanges

setup the entities you want to record

entity_ids

set entity_ids or (entity_type,exchanges,codes)

session

using to do db operations

zvt.recorders.exchange.exchange_stock_meta_recorder

Classes

`ExchangeStockMetaRecorder`([force_update, ...])

ExchangeStockMetaRecorder

class zvt.recorders.exchange.exchange_stock_meta_recorder.ExchangeStockMetaRecorder(*force_update: bool = False*, *sleeping_time: int = 10*)

Bases: `Recorder`

__init__(*force_update: bool = False*, *sleeping_time: int = 10*) → None

data_schema

alias of `Stock`

provider*: str = 'exchange'*

overwrite them to set up the data you want to record

original_page_url = *'http://www.sse.com.cn/assortment/stock/list/share/'*

original page url

clear_state_data(*entity_id=None*)

clear state of the entity

Parameters:: **entity_id** – entity id

decode_state(*state: str*)

decode state

Parameters:: **state** –

Returns::

encode_state(*state: object*)

encode state

Parameters:: **state** –

Returns::

name = *None*

name of the service, default name of class if not set manually

state_schema

alias of `RecorderState`

url = *None*

request url

session

using to do db operations

zvt.recorders.exchange.exchange_stock_summary_recorder

Classes

`ExchangeStockSummaryRecorder`([force_update, ...])

ExchangeStockSummaryRecorder

class zvt.recorders.exchange.exchange_stock_summary_recorder.ExchangeStockSummaryRecorder(*force_update=False*, *sleeping_time=5*, *exchanges=None*, *entity_id=None*, *entity_ids=None*, *code=None*, *day_data=False*, *entity_filters=None*, *ignore_failed=True*, *real_time=False*, *fix_duplicate_way='add'*, *start_timestamp=None*, *end_timestamp=None*)

Bases: `TimestampsDataRecorder`

__init__(*force_update=False*, *sleeping_time=5*, *exchanges=None*, *entity_id=None*, *entity_ids=None*, *code=None*, *day_data=False*, *entity_filters=None*, *ignore_failed=True*, *real_time=False*, *fix_duplicate_way='add'*, *start_timestamp=None*, *end_timestamp=None*) → None

Parameters::

- **code** –
- **ignore_failed** –
- **entity_filters** –
- **exchanges** –
- **entity_id** – for record single entity
- **entity_ids** – set entity_ids or (entity_type,exchanges,codes)
- **codes** –
- **day_data** – one record per day,set to True if you want skip recording it when data of today exist
- **force_update** –
- **sleeping_time** –

entity_provider*: str = 'exchange'*

overwrite them to fetch the entity list

entity_schema

alias of `Index`

provider*: str = 'exchange'*

overwrite them to set up the data you want to record

data_schema

alias of `StockSummary`

original_page_url = *'http://www.sse.com.cn/market/stockdata/overview/day/'*

original page url

url = *'http://query.sse.com.cn/marketdata/tradedata/queryTradingByProdTypeData.do?jsonCallBack=jsonpCallback30731&searchDate={}&prodType=gp&_=1515717065511'*

request url

record(*entity*, *start*, *end*, *size*, *timestamps*)

implement the recording logic in this method, should return json or domain list

Parameters::

- **entity** –
- **start** –
- **end** –
- **size** –
- **timestamps** –

get_data_map()

{'original_field':('domain_field',transform_func)}

clear_state_data(*entity_id=None*)

clear state of the entity

Parameters:: **entity_id** – entity id

decode_state(*state: str*)

decode state

Parameters:: **state** –

Returns::

encode_state(*state: object*)

encode state

Parameters:: **state** –

Returns::

generate_domain(*entity*, *original_data*)

generate the data_schema instance using entity and original_data,the original_data is from record result

Parameters::
- **entity** –
- **original_data** –

generate_domain_id(*entity*, *original_data*, *time_fmt='YYYY-MM-DD'*)

generate domain id from the entity and original data,the default id meaning:entity + event happen time

Parameters::
- **entity** –
- **original_data** –
- **time_fmt** –

Returns::

Return type::

get_evaluated_time_field()

the timestamp field for evaluating time range of recorder,used in get_latest_saved_record

init_entities()

init the entities which we would record data for

name = *None*

name of the service, default name of class if not set manually

persist(*entity*, *domain_list*)

persist the domain list to db

Parameters::

- **entity** –
- **domain_list** –

state_schema

alias of `RecorderState`

exchanges

setup the entities you want to record

entity_ids

set entity_ids or (entity_type,exchanges,codes)

session

using to do db operations

zvt.recorders.joinquant

Modules

zvt.recorders.joinquant.common

---

zvt.recorders.joinquant.fundamental

---

zvt.recorders.joinquant.meta

---

zvt.recorders.joinquant.misc

---

zvt.recorders.joinquant.overall

---

zvt.recorders.joinquant.quotes

---

zvt.recorders.joinquant.common

Functions

`jq_to_report_period`(jq_report_type)

`to_entity_id`(jq_code, entity_type)

`to_jq_entity_id`(security_item)

`to_jq_trading_level`(trading_level)

---

zvt.recorders.joinquant.common.jq_to_report_period

zvt.recorders.joinquant.common.jq_to_report_period(*jq_report_type*)

zvt.recorders.joinquant.common.to_entity_id

zvt.recorders.joinquant.common.to_entity_id(*jq_code: str*, *entity_type*)

zvt.recorders.joinquant.common.to_jq_entity_id

zvt.recorders.joinquant.common.to_jq_entity_id(*security_item*)

zvt.recorders.joinquant.common.to_jq_trading_level

zvt.recorders.joinquant.common.to_jq_trading_level(*trading_level: IntervalLevel*)

zvt.recorders.joinquant.fundamental

Modules

zvt.recorders.joinquant.fundamental.jq_etf_valuation_recorder

---

zvt.recorders.joinquant.fundamental.jq_margin_trading_recorder

---

zvt.recorders.joinquant.fundamental.jq_stock_valuation_recorder

---

zvt.recorders.joinquant.fundamental.jq_etf_valuation_recorder

Classes

`JqChinaEtfValuationRecorder`([force_update, ...])

JqChinaEtfValuationRecorder

class zvt.recorders.joinquant.fundamental.jq_etf_valuation_recorder.JqChinaEtfValuationRecorder(*force_update=False*, *sleeping_time=5*, *exchanges=None*, *entity_id=None*, *entity_ids=None*, *code=None*, *codes=None*, *day_data=False*, *entity_filters=None*, *ignore_failed=True*, *real_time=False*, *fix_duplicate_way='add'*, *start_timestamp=None*, *end_timestamp=None*, *return_unfinished=False*)

Bases: `TimeSeriesDataRecorder`

__init__(*force_update=False*, *sleeping_time=5*, *exchanges=None*, *entity_id=None*, *entity_ids=None*, *code=None*, *codes=None*, *day_data=False*, *entity_filters=None*, *ignore_failed=True*, *real_time=False*, *fix_duplicate_way='add'*, *start_timestamp=None*, *end_timestamp=None*, *return_unfinished=False*) → None

Parameters::

- **code** –
- **ignore_failed** –
- **entity_filters** –
- **exchanges** –
- **entity_id** – for record single entity
- **entity_ids** – set entity_ids or (entity_type,exchanges,codes)
- **codes** –
- **day_data** – one record per day,set to True if you want skip recording it when data of today exist
- **force_update** –
- **sleeping_time** –

entity_provider*: str = 'joinquant'*

overwrite them to fetch the entity list

entity_schema

alias of `Etf`

provider*: str = 'joinquant'*

overwrite them to set up the data you want to record

data_schema

alias of `EtfValuation`

record(*entity*, *start*, *end*, *size*, *timestamps*)

implement the recording logic in this method, should return json or domain list

Parameters::

- **entity** –
- **start** –
- **end** –
- **size** –
- **timestamps** –

clear_state_data(*entity_id=None*)

clear state of the entity

Parameters:: **entity_id** – entity id

decode_state(*state: str*)

decode state

Parameters:: **state** –

Returns::

encode_state(*state: object*)

encode state

Parameters:: **state** –

Returns::

generate_domain(*entity*, *original_data*)

generate the data_schema instance using entity and original_data,the original_data is from record result

Parameters::

- **entity** –
- **original_data** –

generate_domain_id(*entity*, *original_data*, *time_fmt='YYYY-MM-DD'*)

generate domain id from the entity and original data,the default id meaning:entity + event happen time

Parameters::

- **entity** –
- **original_data** –
- **time_fmt** –

Returns::

Return type::

get_data_map()

{'original_field':('domain_field',transform_func)}

get_evaluated_time_field()

the timestamp field for evaluating time range of recorder,used in get_latest_saved_record

init_entities()

init the entities which we would record data for

name = *None*

name of the service, default name of class if not set manually

original_page_url = *None*

original page url

persist(*entity*, *domain_list*)

persist the domain list to db

Parameters::

- **entity** –
- **domain_list** –

state_schema

alias of `RecorderState`

url = *None*

request url

exchanges

setup the entities you want to record

entity_ids

set entity_ids or (entity_type,exchanges,codes)

session

using to do db operations

zvt.recorders.joinquant.fundamental.jq_margin_trading_recorder

Classes

`MarginTradingRecorder`([force_update, ...])

MarginTradingRecorder

class zvt.recorders.joinquant.fundamental.jq_margin_trading_recorder.MarginTradingRecorder(*force_update=False*, *sleeping_time=5*, *exchanges=None*, *entity_id=None*, *entity_ids=None*, *code=None*, *codes=None*, *day_data=False*, *entity_filters=None*, *ignore_failed=True*, *real_time=False*, *fix_duplicate_way='add'*, *start_timestamp=None*, *end_timestamp=None*, *return_unfinished=False*)

Bases: `TimeSeriesDataRecorder`

__init__(*force_update=False*, *sleeping_time=5*, *exchanges=None*, *entity_id=None*, *entity_ids=None*, *code=None*, *codes=None*, *day_data=False*, *entity_filters=None*, *ignore_failed=True*, *real_time=False*, *fix_duplicate_way='add'*, *start_timestamp=None*, *end_timestamp=None*, *return_unfinished=False*) → None

Parameters::

- **code** –
- **ignore_failed** –
- **entity_filters** –
- **exchanges** –
- **entity_id** – for record single entity
- **entity_ids** – set entity_ids or (entity_type,exchanges,codes)
- **codes** –
- **day_data** – one record per day,set to True if you want skip recording it when data of today exist
- **force_update** –
- **sleeping_time** –

entity_provider*: str = 'joinquant'*

overwrite them to fetch the entity list

entity_schema

alias of `Stock`

provider*: str = 'joinquant'*

overwrite them to set up the data you want to record

data_schema

alias of `MarginTrading`

record(*entity*, *start*, *end*, *size*, *timestamps*)

implement the recording logic in this method, should return json or domain list

Parameters::

- **entity** –
- **start** –
- **end** –
- **size** –
- **timestamps** –

clear_state_data(*entity_id=None*)

clear state of the entity

Parameters:: **entity_id** – entity id

decode_state(*state: str*)

decode state

Parameters:: **state** –

Returns::

encode_state(*state: object*)

encode state

Parameters:: **state** –

Returns::

generate_domain(*entity*, *original_data*)

generate the data_schema instance using entity and original_data,the original_data is from record result

Parameters::

- **entity** –
- **original_data** –

generate_domain_id(*entity*, *original_data*, *time_fmt='YYYY-MM-DD'*)

generate domain id from the entity and original data,the default id meaning:entity + event happen time

Parameters::

- **entity** –
- **original_data** –
- **time_fmt** –

Returns::

Return type::

get_data_map()

{'original_field':('domain_field',transform_func)}

get_evaluated_time_field()

the timestamp field for evaluating time range of recorder,used in get_latest_saved_record

init_entities()

init the entities which we would record data for

name = *None*

name of the service, default name of class if not set manually

original_page_url = *None*

original page url

persist(*entity*, *domain_list*)

persist the domain list to db

Parameters::

- **entity** –
- **domain_list** –

state_schema

alias of `RecorderState`

url = *None*

request url

exchanges

setup the entities you want to record

entity_ids

set entity_ids or (entity_type,exchanges,codes)

session

using to do db operations

zvt.recorders.joinquant.fundamental.jq_stock_valuation_recorder

Classes

`JqChinaStockValuationRecorder`([...])

JqChinaStockValuationRecorder

class zvt.recorders.joinquant.fundamental.jq_stock_valuation_recorder.JqChinaStockValuationRecorder(*force_update=False*, *sleeping_time=5*, *exchanges=None*, *entity_id=None*, *entity_ids=None*, *code=None*, *codes=None*, *day_data=False*, *entity_filters=None*, *ignore_failed=True*, *real_time=False*, *fix_duplicate_way='add'*, *start_timestamp=None*, *end_timestamp=None*, *return_unfinished=False*)

Bases: `TimeSeriesDataRecorder`

__init__(*force_update=False*, *sleeping_time=5*, *exchanges=None*, *entity_id=None*, *entity_ids=None*, *code=None*, *codes=None*, *day_data=False*, *entity_filters=None*, *ignore_failed=True*, *real_time=False*, *fix_duplicate_way='add'*, *start_timestamp=None*, *end_timestamp=None*, *return_unfinished=False*) → None

Parameters::

- **code** –
- **ignore_failed** –
- **entity_filters** –
- **exchanges** –
- **entity_id** – for record single entity
- **entity_ids** – set entity_ids or (entity_type,exchanges,codes)
- **codes** –
- **day_data** – one record per day,set to True if you want skip recording it when data of today exist
- **force_update** –
- **sleeping_time** –

entity_provider*: str = 'joinquant'*

overwrite them to fetch the entity list

entity_schema

alias of `Stock`

provider*: str = 'joinquant'*

overwrite them to set up the data you want to record

data_schema

alias of `StockValuation`

record(*entity*, *start*, *end*, *size*, *timestamps*)

implement the recording logic in this method, should return json or domain list

Parameters::

- **entity** –
- **start** –
- **end** –
- **size** –
- **timestamps** –

clear_state_data(*entity_id=None*)

clear state of the entity

Parameters:: **entity_id** – entity id

decode_state(*state: str*)

decode state

Parameters:: **state** –

Returns::

encode_state(*state: object*)

encode state

Parameters:: **state** –

Returns::

generate_domain(*entity*, *original_data*)

generate the data_schema instance using entity and original_data,the original_data is from record result

Parameters::

- **entity** –
- **original_data** –

generate_domain_id(*entity*, *original_data*, *time_fmt='YYYY-MM-DD'*)

generate domain id from the entity and original data,the default id meaning:entity + event happen time

Parameters::

- **entity** –
- **original_data** –
- **time_fmt** –

Returns::

Return type::

get_data_map()

{'original_field':('domain_field',transform_func)}

get_evaluated_time_field()

the timestamp field for evaluating time range of recorder,used in get_latest_saved_record

init_entities()

init the entities which we would record data for

name = *None*

name of the service, default name of class if not set manually

original_page_url = *None*

original page url

persist(*entity*, *domain_list*)

persist the domain list to db

Parameters::

- **entity** –
- **domain_list** –

state_schema

alias of `RecorderState`

url = *None*

request url

exchanges

setup the entities you want to record

entity_ids

set entity_ids or (entity_type,exchanges,codes)

session

using to do db operations

zvt.recorders.joinquant.meta

Modules

zvt.recorders.joinquant.meta.jq_fund_meta_recorder

---

zvt.recorders.joinquant.meta.jq_stock_meta_recorder

---

zvt.recorders.joinquant.meta.jq_trade_day_recorder

---

zvt.recorders.joinquant.meta.jq_fund_meta_recorder

Classes

`JqChinaFundRecorder`([force_update, ...])

`JqChinaFundStockRecorder`([force_update, ...])

JqChinaFundRecorder

class zvt.recorders.joinquant.meta.jq_fund_meta_recorder.JqChinaFundRecorder(*force_update: bool = False*, *sleeping_time: int = 10*)

Bases: `Recorder`

__init__(*force_update: bool = False*, *sleeping_time: int = 10*) → None

provider*: str = 'joinquant'*

overwrite them to set up the data you want to record

data_schema

alias of `Fund`

clear_state_data(*entity_id=None*)

clear state of the entity

Parameters:: **entity_id** – entity id

decode_state(*state: str*)

decode state

Parameters:: **state** –
Returns::

encode_state(*state: object*)

encode state

Parameters:: **state** –
Returns::

name = *None*

name of the service, default name of class if not set manually

original_page_url = *None*

original page url

state_schema

alias of `RecorderState`

url = *None*

request url

session

using to do db operations

JqChinaFundStockRecorder

class zvt.recorders.joinquant.meta.jq_fund_meta_recorder.JqChinaFundStockRecorder(*force_update=False*, *sleeping_time=5*, *exchanges=None*, *entity_id=None*, *entity_ids=None*, *code=None*, *codes=None*, *day_data=False*, *entity_filters=None*, *ignore_failed=True*, *real_time=False*, *fix_duplicate_way='add'*, *start_timestamp=None*, *end_timestamp=None*, *return_unfinished=False*)

Bases: `TimeSeriesDataRecorder`

__init__(*force_update=False*, *sleeping_time=5*, *exchanges=None*, *entity_id=None*, *entity_ids=None*, *code=None*, *codes=None*, *day_data=False*, *entity_filters=None*, *ignore_failed=True*, *real_time=False*, *fix_duplicate_way='add'*, *start_timestamp=None*, *end_timestamp=None*, *return_unfinished=False*) → None

Parameters::

- **code** –
- **ignore_failed** –
- **entity_filters** –
- **exchanges** –
- **entity_id** – for record single entity
- **entity_ids** – set entity_ids or (entity_type,exchanges,codes)
- **codes** –
- **day_data** – one record per day,set to True if you want skip recording it when data of today exist
- **force_update** –
- **sleeping_time** –

entity_provider*: str = 'joinquant'*

overwrite them to fetch the entity list

entity_schema

alias of `Fund`

provider*: str = 'joinquant'*

overwrite them to set up the data you want to record

data_schema

alias of `FundStock`

init_entities()

init the entities which we would record data for

record(*entity*, *start*, *end*, *size*, *timestamps*)

implement the recording logic in this method, should return json or domain list

Parameters::

- **entity** –
- **start** –
- **end** –
- **size** –
- **timestamps** –

clear_state_data(*entity_id=None*)

clear state of the entity

Parameters:: **entity_id** – entity id

decode_state(*state: str*)

decode state

Parameters:: **state** –

Returns::

encode_state(*state: object*)

encode state

Parameters:: **state** –

Returns::

generate_domain(*entity*, *original_data*)

generate the data_schema instance using entity and original_data,the original_data is from record result

Parameters::
- **entity** –
- **original_data** –

generate_domain_id(*entity*, *original_data*, *time_fmt='YYYY-MM-DD'*)

generate domain id from the entity and original data,the default id meaning:entity + event happen time

Parameters::
- **entity** –
- **original_data** –
- **time_fmt** –

Returns::

Return type::

get_data_map()

{'original_field':('domain_field',transform_func)}

get_evaluated_time_field()

the timestamp field for evaluating time range of recorder,used in get_latest_saved_record

name = *None*

name of the service, default name of class if not set manually

original_page_url = *None*

original page url

persist(*entity*, *domain_list*)

persist the domain list to db

Parameters::
- **entity** –
- **domain_list** –

state_schema

alias of `RecorderState`

url = *None*

request url

exchanges

setup the entities you want to record

entity_ids

set entity_ids or (entity_type,exchanges,codes)

session

using to do db operations

zvt.recorders.joinquant.meta.jq_stock_meta_recorder

Classes

`BaseJqChinaMetaRecorder`([force_update, ...])

`JqChinaEtfRecorder`([force_update, sleeping_time])

`JqChinaStockEtfPortfolioRecorder`([...])

`JqChinaStockRecorder`([force_update, ...])

BaseJqChinaMetaRecorder

class zvt.recorders.joinquant.meta.jq_stock_meta_recorder.BaseJqChinaMetaRecorder(*force_update=True*, *sleeping_time=10*)

Bases: `Recorder`

__init__(*force_update=True*, *sleeping_time=10*) → None

provider*: str = 'joinquant'*

overwrite them to set up the data you want to record

clear_state_data(*entity_id=None*)

clear state of the entity

Parameters:: **entity_id** – entity id

decode_state(*state: str*)

decode state

Parameters:: **state** –

Returns::

encode_state(*state: object*)

encode state

Parameters:: **state** –

Returns::

name = *None*

name of the service, default name of class if not set manually

original_page_url = *None*

original page url

state_schema

alias of `RecorderState`

url = *None*

request url

session

using to do db operations

JqChinaEtfRecorder

class zvt.recorders.joinquant.meta.jq_stock_meta_recorder.JqChinaEtfRecorder(*force_update=True*, *sleeping_time=10*)

Bases: `BaseJqChinaMetaRecorder`

__init__(*force_update=True*, *sleeping_time=10*) → None

data_schema

alias of `Etf`

clear_state_data(*entity_id=None*)

clear state of the entity

Parameters:: **entity_id** – entity id

decode_state(*state: str*)

decode state

Parameters:: **state** –

Returns::

encode_state(*state: object*)

encode state

Parameters:: **state** –

Returns::

name = *None*

name of the service, default name of class if not set manually

original_page_url = *None*

original page url

provider*: str = 'joinquant'*

overwrite them to set up the data you want to record

state_schema

alias of `RecorderState`

url = *None*

request url

session

using to do db operations

JqChinaStockEtfPortfolioRecorder

class zvt.recorders.joinquant.meta.jq_stock_meta_recorder.JqChinaStockEtfPortfolioRecorder(*force_update=False*, *sleeping_time=5*, *exchanges=None*, *entity_id=None*, *entity_ids=None*, *code=None*, *codes=None*, *day_data=False*, *entity_filters=None*, *ignore_failed=True*, *real_time=False*, *fix_duplicate_way='add'*, *start_timestamp=None*, *end_timestamp=None*, *return_unfinished=False*)

Bases: `TimeSeriesDataRecorder`

__init__(*force_update=False*, *sleeping_time=5*, *exchanges=None*, *entity_id=None*, *entity_ids=None*, *code=None*, *codes=None*, *day_data=False*, *entity_filters=None*, *ignore_failed=True*, *real_time=False*, *fix_duplicate_way='add'*, *start_timestamp=None*, *end_timestamp=None*, *return_unfinished=False*) → None

Parameters::

- **code** –
- **ignore_failed** –
- **entity_filters** –
- **exchanges** –
- **entity_id** – for record single entity
- **entity_ids** – set entity_ids or (entity_type,exchanges,codes)
- **codes** –
- **day_data** – one record per day,set to True if you want skip recording it when data of today exist
- **force_update** –
- **sleeping_time** –

entity_provider*: str = 'joinquant'*

overwrite them to fetch the entity list

entity_schema

alias of `Etf`

provider*: str = 'joinquant'*

overwrite them to set up the data you want to record

data_schema

alias of `EtfStock`

record(*entity*, *start*, *end*, *size*, *timestamps*)

implement the recording logic in this method, should return json or domain list

Parameters::

- **entity** –
- **start** –
- **end** –
- **size** –
- **timestamps** –

clear_state_data(*entity_id=None*)

clear state of the entity

Parameters:: **entity_id** – entity id

decode_state(*state: str*)

decode state

Parameters:: **state** –

Returns::

encode_state(*state: object*)

encode state

Parameters:: **state** –

Returns::

generate_domain(*entity*, *original_data*)

generate the data_schema instance using entity and original_data,the original_data is from record result

Parameters::

- **entity** –
- **original_data** –

generate_domain_id(*entity*, *original_data*, *time_fmt='YYYY-MM-DD'*)

generate domain id from the entity and original data,the default id meaning:entity + event happen time

Parameters::

- **entity** –
- **original_data** –
- **time_fmt** –

Returns::

Return type::

get_data_map()

{'original_field':('domain_field',transform_func)}

get_evaluated_time_field()

the timestamp field for evaluating time range of recorder,used in get_latest_saved_record

init_entities()

init the entities which we would record data for

name = *None*

name of the service, default name of class if not set manually

original_page_url = *None*

original page url

persist(*entity*, *domain_list*)

persist the domain list to db

Parameters::

- **entity** –
- **domain_list** –

state_schema

alias of `RecorderState`

url = *None*

request url

exchanges

setup the entities you want to record

entity_ids

set entity_ids or (entity_type,exchanges,codes)

session

using to do db operations

JqChinaStockRecorder

class zvt.recorders.joinquant.meta.jq_stock_meta_recorder.JqChinaStockRecorder(*force_update=True*, *sleeping_time=10*)

Bases: `BaseJqChinaMetaRecorder`

__init__(*force_update=True*, *sleeping_time=10*) → None

data_schema

alias of `Stock`

clear_state_data(*entity_id=None*)

clear state of the entity

Parameters:: **entity_id** – entity id

decode_state(*state: str*)

decode state

Parameters:: **state** –

Returns::

encode_state(*state: object*)

encode state

Parameters:: **state** –

Returns::

name = *None*

name of the service, default name of class if not set manually

original_page_url = *None*

original page url

provider*: str = 'joinquant'*

overwrite them to set up the data you want to record

state_schema

alias of `RecorderState`

url = *None*

request url

session

using to do db operations

zvt.recorders.joinquant.meta.jq_trade_day_recorder

Classes

`StockTradeDayRecorder`([exchanges, ...])

StockTradeDayRecorder

class zvt.recorders.joinquant.meta.jq_trade_day_recorder.StockTradeDayRecorder(*exchanges=None*, *entity_id=None*, *entity_ids=None*, *day_data=False*, *force_update=False*, *sleeping_time=5*, *real_time=False*, *fix_duplicate_way='add'*, *start_timestamp=None*, *end_timestamp=None*, *entity_filters=None*)

Bases: `TimeSeriesDataRecorder`

__init__(*exchanges=None*, *entity_id=None*, *entity_ids=None*, *day_data=False*, *force_update=False*, *sleeping_time=5*, *real_time=False*, *fix_duplicate_way='add'*, *start_timestamp=None*, *end_timestamp=None*, *entity_filters=None*) → None

Parameters::

- **code** –
- **ignore_failed** –
- **entity_filters** –
- **exchanges** –
- **entity_id** – for record single entity
- **entity_ids** – set entity_ids or (entity_type,exchanges,codes)
- **codes** –
- **day_data** – one record per day,set to True if you want skip recording it when data of today exist
- **force_update** –
- **sleeping_time** –

entity_provider*: str = 'joinquant'*

overwrite them to fetch the entity list

entity_schema

alias of `Stock`

provider*: str = 'joinquant'*

overwrite them to set up the data you want to record

data_schema

alias of `StockTradeDay`

record(*entity*, *start*, *end*, *size*, *timestamps*)

implement the recording logic in this method, should return json or domain list

Parameters::

- **entity** –
- **start** –
- **end** –
- **size** –
- **timestamps** –

clear_state_data(*entity_id=None*)

clear state of the entity

Parameters:: **entity_id** – entity id

decode_state(*state: str*)

decode state

Parameters:: **state** –

Returns::

encode_state(*state: object*)

encode state

Parameters:: **state** –

Returns::

generate_domain(*entity*, *original_data*)

generate the data_schema instance using entity and original_data,the original_data is from record result

Parameters::

- **entity** –
- **original_data** –

generate_domain_id(*entity*, *original_data*, *time_fmt='YYYY-MM-DD'*)

generate domain id from the entity and original data,the default id meaning:entity + event happen time

Parameters::
- **entity** –
- **original_data** –
- **time_fmt** –

Returns::

Return type::

get_data_map()

{'original_field':('domain_field',transform_func)}

get_evaluated_time_field()

the timestamp field for evaluating time range of recorder,used in get_latest_saved_record

init_entities()

init the entities which we would record data for

name = *None*

name of the service, default name of class if not set manually

original_page_url = *None*

original page url

persist(*entity*, *domain_list*)

persist the domain list to db

Parameters::
- **entity** –
- **domain_list** –

state_schema

alias of `RecorderState`

url = *None*

request url

exchanges

setup the entities you want to record

entity_ids

set entity_ids or (entity_type,exchanges,codes)

session

using to do db operations

zvt.recorders.joinquant.misc

Modules

zvt.recorders.joinquant.misc.jq_hk_holder_recorder

---

zvt.recorders.joinquant.misc.jq_index_money_flow_recorder

---

zvt.recorders.joinquant.misc.jq_stock_money_flow_recorder

---

zvt.recorders.joinquant.misc.jq_hk_holder_recorder

Classes

`JoinquantHkHolderRecorder`([day_data, ...])

JoinquantHkHolderRecorder

class zvt.recorders.joinquant.misc.jq_hk_holder_recorder.JoinquantHkHolderRecorder(*day_data=False*, *force_update=False*, *sleeping_time=5*, *real_time=False*, *start_timestamp=None*, *end_timestamp=None*)

Bases: `TimestampsDataRecorder`

__init__(*day_data=False*, *force_update=False*, *sleeping_time=5*, *real_time=False*, *start_timestamp=None*, *end_timestamp=None*) → None

Parameters::

- **code** –
- **ignore_failed** –
- **entity_filters** –
- **exchanges** –
- **entity_id** – for record single entity
- **entity_ids** – set entity_ids or (entity_type,exchanges,codes)
- **codes** –
- **day_data** – one record per day,set to True if you want skip recording it when data of today exist
- **force_update** –
- **sleeping_time** –

entity_provider*: str = 'exchange'*

overwrite them to fetch the entity list

entity_schema

alias of `Index`

provider*: str = 'joinquant'*

overwrite them to set up the data you want to record

data_schema

alias of `HkHolder`

record(*entity*, *start*, *end*, *size*, *timestamps*)

implement the recording logic in this method, should return json or domain list

Parameters::
- **entity** –
- **start** –
- **end** –
- **size** –
- **timestamps** –

clear_state_data(*entity_id=None*)

clear state of the entity

Parameters:: **entity_id** – entity id

decode_state(*state: str*)

decode state

Parameters:: **state** –
Returns::

encode_state(*state: object*)

encode state

Parameters:: **state** –
Returns::

generate_domain(*entity*, *original_data*)

generate the data_schema instance using entity and original_data,the original_data is from record result

Parameters::
- **entity** –
- **original_data** –

generate_domain_id(*entity*, *original_data*, *time_fmt='YYYY-MM-DD'*)

generate domain id from the entity and original data,the default id meaning:entity + event happen time

Parameters::

- **entity** –
- **original_data** –
- **time_fmt** –

Returns::

Return type::

get_data_map()

{'original_field':('domain_field',transform_func)}

get_evaluated_time_field()

the timestamp field for evaluating time range of recorder,used in get_latest_saved_record

init_entities()

init the entities which we would record data for

name = *None*

name of the service, default name of class if not set manually

original_page_url = *None*

original page url

persist(*entity*, *domain_list*)

persist the domain list to db

Parameters::

- **entity** –
- **domain_list** –

state_schema

alias of `RecorderState`

url = *None*

request url

exchanges

setup the entities you want to record

entity_ids

set entity_ids or (entity_type,exchanges,codes)

session

using to do db operations

zvt.recorders.joinquant.misc.jq_index_money_flow_recorder

Classes

`JoinquantIndexMoneyFlowRecorder`([...])

JoinquantIndexMoneyFlowRecorder

class zvt.recorders.joinquant.misc.jq_index_money_flow_recorder.JoinquantIndexMoneyFlowRecorder(*force_update=True*, *sleeping_time=10*, *exchanges=None*, *entity_id=None*, *entity_ids=None*, *code=None*, *codes=None*, *day_data=False*, *entity_filters=None*, *ignore_failed=True*, *real_time=False*, *fix_duplicate_way='ignore'*, *start_timestamp=None*, *end_timestamp=None*, *level=IntervalLevel.LEVEL_1DAY*, *kdata_use_begin_time=False*, *one_day_trading_minutes=1440*, *return_unfinished=False*)

Bases: `FixedCycleDataRecorder`

__init__(*force_update=True*, *sleeping_time=10*, *exchanges=None*, *entity_id=None*, *entity_ids=None*, *code=None*, *codes=None*, *day_data=False*, *entity_filters=None*, *ignore_failed=True*, *real_time=False*, *fix_duplicate_way='ignore'*, *start_timestamp=None*, *end_timestamp=None*, *level=IntervalLevel.LEVEL_1DAY*, *kdata_use_begin_time=False*, *one_day_trading_minutes=1440*, *return_unfinished=False*) → None

Parameters::

- **code** –
- **ignore_failed** –
- **entity_filters** –
- **exchanges** –
- **entity_id** – for record single entity
- **entity_ids** – set entity_ids or (entity_type,exchanges,codes)
- **codes** –
- **day_data** – one record per day,set to True if you want skip recording it when data of today exist
- **force_update** –
- **sleeping_time** –

entity_provider*: str = 'exchange'*

overwrite them to fetch the entity list

entity_schema

alias of `Index`

provider*: str = 'joinquant'*

overwrite them to set up the data you want to record

data_schema

alias of `IndexMoneyFlow`

record(*entity*, *start*, *end*, *size*, *timestamps*)

implement the recording logic in this method, should return json or domain list

Parameters::

- **entity** –
- **start** –
- **end** –
- **size** –
- **timestamps** –

clear_state_data(*entity_id=None*)

clear state of the entity

Parameters:: **entity_id** – entity id

decode_state(*state: str*)

decode state

Parameters:: **state** –

Returns::

encode_state(*state: object*)

encode state

Parameters:: **state** –

Returns::

generate_domain(*entity*, *original_data*)

generate the data_schema instance using entity and original_data,the original_data is from record result

Parameters::
- **entity** –
- **original_data** –

generate_domain_id(*entity*, *original_data*, *time_fmt='YYYY-MM-DD'*)

generate domain id from the entity and original data,the default id meaning:entity + event happen time

Parameters::
- **entity** –
- **original_data** –
- **time_fmt** –

Returns::

Return type::

get_data_map()

{'original_field':('domain_field',transform_func)}

get_evaluated_time_field()

the timestamp field for evaluating time range of recorder,used in get_latest_saved_record

init_entities()

init the entities which we would record data for

name = *None*

name of the service, default name of class if not set manually

original_page_url = *None*

original page url

persist(*entity*, *domain_list*)

persist the domain list to db

Parameters::
- **entity** –
- **domain_list** –

state_schema

alias of `RecorderState`

url = *None*

request url

exchanges

setup the entities you want to record

entity_ids

set entity_ids or (entity_type,exchanges,codes)

session

using to do db operations

zvt.recorders.joinquant.misc.jq_stock_money_flow_recorder

Classes

`JoinquantStockMoneyFlowRecorder`([...])

JoinquantStockMoneyFlowRecorder

class zvt.recorders.joinquant.misc.jq_stock_money_flow_recorder.JoinquantStockMoneyFlowRecorder(*force_update=True*, *sleeping_time=10*, *exchanges=None*, *entity_id=None*, *entity_ids=None*, *code=None*, *codes=None*, *day_data=False*, *entity_filters=None*, *ignore_failed=True*, *real_time=False*, *fix_duplicate_way='ignore'*, *start_timestamp=None*, *end_timestamp=None*, *level=IntervalLevel.LEVEL_1DAY*, *kdata_use_begin_time=False*, *one_day_trading_minutes=1440*, *compute_index_money_flow=False*, *return_unfinished=False*)

Bases: `FixedCycleDataRecorder`

__init__(*force_update=True*, *sleeping_time=10*, *exchanges=None*, *entity_id=None*, *entity_ids=None*, *code=None*, *codes=None*, *day_data=False*, *entity_filters=None*, *ignore_failed=True*, *real_time=False*, *fix_duplicate_way='ignore'*, *start_timestamp=None*, *end_timestamp=None*, *level=IntervalLevel.LEVEL_1DAY*, *kdata_use_begin_time=False*, *one_day_trading_minutes=1440*, *compute_index_money_flow=False*, *return_unfinished=False*) → None

Parameters::

- **code** –
- **ignore_failed** –
- **entity_filters** –
- **exchanges** –
- **entity_id** – for record single entity
- **entity_ids** – set entity_ids or (entity_type,exchanges,codes)
- **codes** –
- **day_data** – one record per day,set to True if you want skip recording it when data of today exist
- **force_update** –
- **sleeping_time** –

entity_provider*: str = 'joinquant'*

overwrite them to fetch the entity list

entity_schema

alias of `Stock`

provider*: str = 'joinquant'*

overwrite them to set up the data you want to record

data_schema

alias of `StockMoneyFlow`

generate_domain_id(*entity*, *original_data*)

generate domain id from the entity and original data,the default id meaning:entity + event happen time

Parameters::

- **entity** –
- **original_data** –
- **time_fmt** –

Returns::

Return type::

record(*entity*, *start*, *end*, *size*, *timestamps*)

implement the recording logic in this method, should return json or domain list

Parameters::

- **entity** –
- **start** –
- **end** –
- **size** –
- **timestamps** –

clear_state_data(*entity_id=None*)

clear state of the entity

Parameters:: **entity_id** – entity id

decode_state(*state: str*)

decode state

Parameters:: **state** –
Returns::

encode_state(*state: object*)

encode state

Parameters:: **state** –
Returns::

generate_domain(*entity*, *original_data*)

generate the data_schema instance using entity and original_data,the original_data is from record result

Parameters::
- **entity** –
- **original_data** –

get_data_map()

{'original_field':('domain_field',transform_func)}

get_evaluated_time_field()

the timestamp field for evaluating time range of recorder,used in get_latest_saved_record

init_entities()

init the entities which we would record data for

name = *None*

name of the service, default name of class if not set manually

original_page_url = *None*

original page url

persist(*entity*, *domain_list*)

persist the domain list to db

Parameters::

- **entity** –
- **domain_list** –

state_schema

alias of `RecorderState`

url = *None*

request url

exchanges

setup the entities you want to record

entity_ids

set entity_ids or (entity_type,exchanges,codes)

session

using to do db operations

zvt.recorders.joinquant.overall

Modules

zvt.recorders.joinquant.overall.jq_cross_market_recorder

---

zvt.recorders.joinquant.overall.jq_margin_trading_recorder

---

zvt.recorders.joinquant.overall.jq_stock_summary_recorder

---

zvt.recorders.joinquant.overall.jq_cross_market_recorder

Classes

`CrossMarketSummaryRecorder`([force_update, ...])

CrossMarketSummaryRecorder

class zvt.recorders.joinquant.overall.jq_cross_market_recorder.CrossMarketSummaryRecorder(*force_update=False*, *sleeping_time=5*, *real_time=False*, *fix_duplicate_way='add'*)

Bases: `TimeSeriesDataRecorder`

__init__(*force_update=False*, *sleeping_time=5*, *real_time=False*, *fix_duplicate_way='add'*) → None

Parameters::

- **code** –
- **ignore_failed** –
- **entity_filters** –
- **exchanges** –
- **entity_id** – for record single entity
- **entity_ids** – set entity_ids or (entity_type,exchanges,codes)
- **codes** –
- **day_data** – one record per day,set to True if you want skip recording it when data of today exist
- **force_update** –
- **sleeping_time** –

entity_provider*: str = 'joinquant'*

overwrite them to fetch the entity list

entity_schema

alias of `Index`

provider*: str = 'joinquant'*

overwrite them to set up the data you want to record

data_schema

alias of `CrossMarketSummary`

init_entities()

init the entities which we would record data for

record(*entity*, *start*, *end*, *size*, *timestamps*)

implement the recording logic in this method, should return json or domain list

Parameters::

- **entity** –
- **start** –
- **end** –
- **size** –
- **timestamps** –

get_data_map()

{'original_field':('domain_field',transform_func)}

clear_state_data(*entity_id=None*)

clear state of the entity

Parameters:: **entity_id** – entity id

decode_state(*state: str*)

decode state

Parameters:: **state** –

Returns::

encode_state(*state: object*)

encode state

Parameters:: **state** –

Returns::

generate_domain(*entity*, *original_data*)

generate the data_schema instance using entity and original_data,the original_data is from record result

Parameters::

- **entity** –
- **original_data** –

generate_domain_id(*entity*, *original_data*, *time_fmt='YYYY-MM-DD'*)

generate domain id from the entity and original data,the default id meaning:entity + event happen time

Parameters::

- **entity** –
- **original_data** –
- **time_fmt** –

Returns::

Return type::

get_evaluated_time_field()

the timestamp field for evaluating time range of recorder,used in get_latest_saved_record

name = *None*

name of the service, default name of class if not set manually

original_page_url = *None*

original page url

persist(*entity*, *domain_list*)

persist the domain list to db

Parameters::

- **entity** –
- **domain_list** –

state_schema

alias of `RecorderState`

url = *None*

request url

exchanges

setup the entities you want to record

entity_ids

set entity_ids or (entity_type,exchanges,codes)

session

using to do db operations

zvt.recorders.joinquant.overall.jq_margin_trading_recorder

Classes

`MarginTradingSummaryRecorder`([force_update, ...])

MarginTradingSummaryRecorder

class zvt.recorders.joinquant.overall.jq_margin_trading_recorder.MarginTradingSummaryRecorder(*force_update=False*, *sleeping_time=5*, *exchanges=None*, *entity_id=None*, *entity_ids=None*, *day_data=False*, *entity_filters=None*, *ignore_failed=True*, *real_time=False*, *fix_duplicate_way='add'*, *start_timestamp=None*, *end_timestamp=None*)

Bases: `TimeSeriesDataRecorder`

__init__(*force_update=False*, *sleeping_time=5*, *exchanges=None*, *entity_id=None*, *entity_ids=None*, *day_data=False*, *entity_filters=None*, *ignore_failed=True*, *real_time=False*, *fix_duplicate_way='add'*, *start_timestamp=None*, *end_timestamp=None*) → None

Parameters::

- **code** –
- **ignore_failed** –
- **entity_filters** –
- **exchanges** –
- **entity_id** – for record single entity
- **entity_ids** – set entity_ids or (entity_type,exchanges,codes)
- **codes** –
- **day_data** – one record per day,set to True if you want skip recording it when data of today exist
- **force_update** –
- **sleeping_time** –

entity_provider*: str = 'exchange'*

overwrite them to fetch the entity list

entity_schema

alias of `Index`

provider*: str = 'joinquant'*

overwrite them to set up the data you want to record

data_schema

alias of `MarginTradingSummary`

record(*entity*, *start*, *end*, *size*, *timestamps*)

implement the recording logic in this method, should return json or domain list

Parameters::

- **entity** –
- **start** –
- **end** –
- **size** –
- **timestamps** –

get_data_map()

{'original_field':('domain_field',transform_func)}

clear_state_data(*entity_id=None*)

clear state of the entity

Parameters:: **entity_id** – entity id

decode_state(*state: str*)

decode state

Parameters:: **state** –

Returns::

encode_state(*state: object*)

encode state

Parameters:: **state** –

Returns::

generate_domain(*entity*, *original_data*)

generate the data_schema instance using entity and original_data,the original_data is from record result

Parameters::

- **entity** –
- **original_data** –

generate_domain_id(*entity*, *original_data*, *time_fmt='YYYY-MM-DD'*)

generate domain id from the entity and original data,the default id meaning:entity + event happen time

Parameters::

- **entity** –
- **original_data** –
- **time_fmt** –

Returns::

Return type::

get_evaluated_time_field()

the timestamp field for evaluating time range of recorder,used in get_latest_saved_record

init_entities()

init the entities which we would record data for

name = *None*

name of the service, default name of class if not set manually

original_page_url = *None*

original page url

persist(*entity*, *domain_list*)

persist the domain list to db

Parameters::

- **entity** –
- **domain_list** –

state_schema

alias of `RecorderState`

url = *None*

request url

exchanges

setup the entities you want to record

entity_ids

set entity_ids or (entity_type,exchanges,codes)

session

using to do db operations

zvt.recorders.joinquant.overall.jq_stock_summary_recorder

Classes

`StockSummaryRecorder`([force_update, ...])

StockSummaryRecorder

class zvt.recorders.joinquant.overall.jq_stock_summary_recorder.StockSummaryRecorder(*force_update=False*, *sleeping_time=5*, *exchanges=None*, *entity_id=None*, *entity_ids=None*, *day_data=False*, *entity_filters=None*, *ignore_failed=True*, *real_time=False*, *fix_duplicate_way='add'*, *start_timestamp=None*, *end_timestamp=None*)

Bases: `TimeSeriesDataRecorder`

__init__(*force_update=False*, *sleeping_time=5*, *exchanges=None*, *entity_id=None*, *entity_ids=None*, *day_data=False*, *entity_filters=None*, *ignore_failed=True*, *real_time=False*, *fix_duplicate_way='add'*, *start_timestamp=None*, *end_timestamp=None*) → None

Parameters::

- **code** –
- **ignore_failed** –
- **entity_filters** –
- **exchanges** –
- **entity_id** – for record single entity
- **entity_ids** – set entity_ids or (entity_type,exchanges,codes)
- **codes** –
- **day_data** – one record per day,set to True if you want skip recording it when data of today exist
- **force_update** –
- **sleeping_time** –

entity_provider*: str = 'exchange'*

overwrite them to fetch the entity list

entity_schema

alias of `Index`

provider*: str = 'joinquant'*

overwrite them to set up the data you want to record

data_schema

alias of `StockSummary`

record(*entity*, *start*, *end*, *size*, *timestamps*)

implement the recording logic in this method, should return json or domain list

Parameters::

- **entity** –
- **start** –
- **end** –
- **size** –
- **timestamps** –

get_data_map()

{'original_field':('domain_field',transform_func)}

clear_state_data(*entity_id=None*)

clear state of the entity

Parameters:: **entity_id** – entity id

decode_state(*state: str*)

decode state

Parameters:: **state** –

Returns::

encode_state(*state: object*)

encode state

Parameters:: **state** –

Returns::

generate_domain(*entity*, *original_data*)

generate the data_schema instance using entity and original_data,the original_data is from record result

Parameters::

- **entity** –
- **original_data** –

generate_domain_id(*entity*, *original_data*, *time_fmt='YYYY-MM-DD'*)

generate domain id from the entity and original data,the default id meaning:entity + event happen time

Parameters::

- **entity** –
- **original_data** –
- **time_fmt** –

Returns::

Return type::

get_evaluated_time_field()

the timestamp field for evaluating time range of recorder,used in get_latest_saved_record

init_entities()

init the entities which we would record data for

name = *None*

name of the service, default name of class if not set manually

original_page_url = *None*

original page url

persist(*entity*, *domain_list*)

persist the domain list to db

Parameters::

- **entity** –
- **domain_list** –

state_schema

alias of `RecorderState`

url = *None*

request url

exchanges

setup the entities you want to record

entity_ids

set entity_ids or (entity_type,exchanges,codes)

session

using to do db operations

zvt.recorders.joinquant.quotes

Modules

zvt.recorders.joinquant.quotes.jq_index_kdata_recorder

---

zvt.recorders.joinquant.quotes.jq_stock_kdata_recorder

---

zvt.recorders.joinquant.quotes.jq_index_kdata_recorder

Classes

`JqChinaIndexKdataRecorder`([force_update, ...])

JqChinaIndexKdataRecorder

class zvt.recorders.joinquant.quotes.jq_index_kdata_recorder.JqChinaIndexKdataRecorder(*force_update=True*, *sleeping_time=10*, *exchanges=None*, *entity_id=None*, *entity_ids=None*, *code=None*, *codes=None*, *day_data=False*, *entity_filters=None*, *ignore_failed=True*, *real_time=False*, *fix_duplicate_way='ignore'*, *start_timestamp=None*, *end_timestamp=None*, *level=IntervalLevel.LEVEL_1DAY*, *kdata_use_begin_time=False*, *one_day_trading_minutes=1440*, *return_unfinished=False*)

Bases: `FixedCycleDataRecorder`

__init__(*force_update=True*, *sleeping_time=10*, *exchanges=None*, *entity_id=None*, *entity_ids=None*, *code=None*, *codes=None*, *day_data=False*, *entity_filters=None*, *ignore_failed=True*, *real_time=False*, *fix_duplicate_way='ignore'*, *start_timestamp=None*, *end_timestamp=None*, *level=IntervalLevel.LEVEL_1DAY*, *kdata_use_begin_time=False*, *one_day_trading_minutes=1440*, *return_unfinished=False*) → None

Parameters::

- **code** –
- **ignore_failed** –
- **entity_filters** –
- **exchanges** –
- **entity_id** – for record single entity
- **entity_ids** – set entity_ids or (entity_type,exchanges,codes)
- **codes** –
- **day_data** – one record per day,set to True if you want skip recording it when data of today exist
- **force_update** –
- **sleeping_time** –

entity_provider*: str = 'joinquant'*

overwrite them to fetch the entity list

entity_schema

alias of `Index`

provider*: str = 'joinquant'*

overwrite them to set up the data you want to record

data_schema

alias of `IndexKdataCommon`

init_entities()

init the entities which we would record data for

generate_domain_id(*entity*, *original_data*)

generate domain id from the entity and original data,the default id meaning:entity + event happen time

Parameters::

- **entity** –
- **original_data** –
- **time_fmt** –

Returns::

Return type::

record(*entity*, *start*, *end*, *size*, *timestamps*)

implement the recording logic in this method, should return json or domain list

Parameters::

- **entity** –
- **start** –
- **end** –
- **size** –
- **timestamps** –

clear_state_data(*entity_id=None*)

clear state of the entity

Parameters:: **entity_id** – entity id

decode_state(*state: str*)

decode state

Parameters:: **state** –
Returns::

encode_state(*state: object*)

encode state

Parameters:: **state** –
Returns::

generate_domain(*entity*, *original_data*)

generate the data_schema instance using entity and original_data,the original_data is from record result

Parameters::
- **entity** –
- **original_data** –

get_data_map()

{'original_field':('domain_field',transform_func)}

get_evaluated_time_field()

the timestamp field for evaluating time range of recorder,used in get_latest_saved_record

name = *None*

name of the service, default name of class if not set manually

original_page_url = *None*

original page url

persist(*entity*, *domain_list*)

persist the domain list to db

Parameters::
- **entity** –
- **domain_list** –

state_schema

alias of `RecorderState`

url = *None*

request url

exchanges

setup the entities you want to record

entity_ids

set entity_ids or (entity_type,exchanges,codes)

session

using to do db operations

zvt.recorders.joinquant.quotes.jq_stock_kdata_recorder

Classes

`JqChinaStockKdataRecorder`([force_update, ...])

JqChinaStockKdataRecorder

class zvt.recorders.joinquant.quotes.jq_stock_kdata_recorder.JqChinaStockKdataRecorder(*force_update=True*, *sleeping_time=10*, *exchanges=None*, *entity_id=None*, *entity_ids=None*, *code=None*, *codes=None*, *day_data=False*, *entity_filters=None*, *ignore_failed=True*, *real_time=False*, *fix_duplicate_way='ignore'*, *start_timestamp=None*, *end_timestamp=None*, *level=IntervalLevel.LEVEL_1DAY*, *kdata_use_begin_time=False*, *one_day_trading_minutes=1440*, *adjust_type=AdjustType.qfq*, *return_unfinished=False*)

Bases: `FixedCycleDataRecorder`

__init__(*force_update=True*, *sleeping_time=10*, *exchanges=None*, *entity_id=None*, *entity_ids=None*, *code=None*, *codes=None*, *day_data=False*, *entity_filters=None*, *ignore_failed=True*, *real_time=False*, *fix_duplicate_way='ignore'*, *start_timestamp=None*, *end_timestamp=None*, *level=IntervalLevel.LEVEL_1DAY*, *kdata_use_begin_time=False*, *one_day_trading_minutes=1440*, *adjust_type=AdjustType.qfq*, *return_unfinished=False*) → None

Parameters::

- **code** –
- **ignore_failed** –
- **entity_filters** –
- **exchanges** –
- **entity_id** – for record single entity
- **entity_ids** – set entity_ids or (entity_type,exchanges,codes)
- **codes** –
- **day_data** – one record per day,set to True if you want skip recording it when data of today exist
- **force_update** –
- **sleeping_time** –

entity_provider*: str = 'joinquant'*

overwrite them to fetch the entity list

entity_schema

alias of `Stock`

provider*: str = 'joinquant'*

overwrite them to set up the data you want to record

data_schema

alias of `StockKdataCommon`

init_entities()

init the entities which we would record data for

generate_domain_id(*entity*, *original_data*)

generate domain id from the entity and original data,the default id meaning:entity + event happen time

Parameters::

- **entity** –
- **original_data** –
- **time_fmt** –

Returns::

Return type::

record(*entity*, *start*, *end*, *size*, *timestamps*)

implement the recording logic in this method, should return json or domain list

Parameters::

- **entity** –
- **start** –
- **end** –
- **size** –
- **timestamps** –

clear_state_data(*entity_id=None*)

clear state of the entity

Parameters:: **entity_id** – entity id

decode_state(*state: str*)

decode state

Parameters:: **state** –

Returns::

encode_state(*state: object*)

encode state

Parameters:: **state** –

Returns::

generate_domain(*entity*, *original_data*)

generate the data_schema instance using entity and original_data,the original_data is from record result

Parameters::

- **entity** –
- **original_data** –

get_data_map()

{‘original_field’:(‘domain_field’,transform_func)}

get_evaluated_time_field()

the timestamp field for evaluating time range of recorder,used in get_latest_saved_record

name = *None*

name of the service, default name of class if not set manually

original_page_url = *None*

original page url

persist(*entity*, *domain_list*)

persist the domain list to db

Parameters::

- **entity** –
- **domain_list** –

state_schema

alias of `RecorderState`

url = *None*

request url

exchanges

setup the entities you want to record

entity_ids

set entity_ids or (entity_type,exchanges,codes)

session

using to do db operations

zvt.recorders.jqka

Modules

zvt.recorders.jqka.emotion

---

zvt.recorders.jqka.jqka_api

---

zvt.recorders.jqka.emotion

Modules

zvt.recorders.jqka.emotion.JqkaEmotionRecorder([...])

---

zvt.recorders.jqka.emotion.JqkaEmotionRecorder

Module Attributes

zvt.recorders.jqka.jqka_api

Functions

`get_continuous_limit_up`(date)

`get_jkqa_data`(url[, pn, ps, fetch_all, headers])

`get_limit_down`(date)

`get_limit_stats`(date)

`get_limit_up`(date)

---

zvt.recorders.jqka.jqka_api.get_continuous_limit_up

zvt.recorders.jqka.jqka_api.get_continuous_limit_up(*date: str*)

zvt.recorders.jqka.jqka_api.get_jkqa_data

zvt.recorders.jqka.jqka_api.get_jkqa_data(*url*, *pn=1*, *ps=200*, *fetch_all=True*, *headers={'Accept': 'application/json, text/plain, */*', 'Accept-Encoding': 'gzip, deflate, br', 'Accept-Language': 'zh-CN,zh;q=0.9,en;q=0.8', 'Connection': 'keep-alive', 'Host': 'data.10jqka.com.cn', 'Referer': 'https://data.10jqka.com.cn/datacenterph/limitup/limtupInfo.html?fontzoom=no&client_userid=cA2fp&share_hxapp=gsc&share_action=webpage_share.1&back_source=wxhy', 'Sec-Fetch-Dest': 'empty', 'Sec-Fetch-Mode': 'cors', 'Sec-Fetch-Site': 'same-origin', 'User-Agent': 'Mozilla/5.0 (Linux; Android 6.0; Nexus 5 Build/MRA58N) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/109.0.0.0 Mobile Safari/537.36', 'sec-ch-ua': '"Not_A Brand";v="99", "Google Chrome";v="109", "Chromium";v="109"', 'sec-ch-ua-mobile': '?1', 'sec-ch-ua-platform': '"Android"'}*)

zvt.recorders.jqka.jqka_api.get_limit_down

zvt.recorders.jqka.jqka_api.get_limit_down(*date: str*)

zvt.recorders.jqka.jqka_api.get_limit_stats

zvt.recorders.jqka.jqka_api.get_limit_stats(*date: str*)

zvt.recorders.jqka.jqka_api.get_limit_up

zvt.recorders.jqka.jqka_api.get_limit_up(*date: str*)

zvt.recorders.qmt

Modules

zvt.recorders.qmt.meta

---

zvt.recorders.qmt.meta

Modules

zvt.recorders.sina

Modules

zvt.recorders.sina.meta

---

zvt.recorders.sina.money_flow

---

zvt.recorders.sina.quotes

---

zvt.recorders.sina.meta

Modules

zvt.recorders.sina.meta.sina_block_recorder

---

zvt.recorders.sina.meta.sina_block_recorder

Classes

`SinaBlockRecorder`([force_update, sleeping_time])

`SinaChinaBlockStockRecorder`([force_update, ...])

SinaBlockRecorder

class zvt.recorders.sina.meta.sina_block_recorder.SinaBlockRecorder(*force_update: bool = False*, *sleeping_time: int = 10*)

Bases: `Recorder`

__init__(*force_update: bool = False*, *sleeping_time: int = 10*) → None

provider*: str = 'sina'*

overwrite them to set up the data you want to record

data_schema

alias of `Block`

clear_state_data(*entity_id=None*)

clear state of the entity

Parameters:: **entity_id** – entity id

decode_state(*state: str*)

decode state

Parameters:: **state** –
Returns::

encode_state(*state: object*)

encode state

Parameters:: **state** –
Returns::

name = *None*

name of the service, default name of class if not set manually

original_page_url = *None*

original page url

state_schema

alias of `RecorderState`

url = *None*

request url

session

using to do db operations

SinaChinaBlockStockRecorder

class zvt.recorders.sina.meta.sina_block_recorder.SinaChinaBlockStockRecorder(*force_update=False*, *sleeping_time=5*, *exchanges=None*, *entity_id=None*, *entity_ids=None*, *code=None*, *codes=None*, *day_data=False*, *entity_filters=None*, *ignore_failed=True*, *real_time=False*, *fix_duplicate_way='add'*, *start_timestamp=None*, *end_timestamp=None*, *return_unfinished=False*)

Bases: `TimeSeriesDataRecorder`

__init__(*force_update=False*, *sleeping_time=5*, *exchanges=None*, *entity_id=None*, *entity_ids=None*, *code=None*, *codes=None*, *day_data=False*, *entity_filters=None*, *ignore_failed=True*, *real_time=False*, *fix_duplicate_way='add'*, *start_timestamp=None*, *end_timestamp=None*, *return_unfinished=False*) → None

Parameters::

- **code** –
- **ignore_failed** –
- **entity_filters** –
- **exchanges** –
- **entity_id** – for record single entity
- **entity_ids** – set entity_ids or (entity_type,exchanges,codes)
- **codes** –
- **day_data** – one record per day,set to True if you want skip recording it when data of today exist
- **force_update** –
- **sleeping_time** –

entity_provider*: str = 'sina'*

overwrite them to fetch the entity list

entity_schema

alias of `Block`

provider*: str = 'sina'*

overwrite them to set up the data you want to record

data_schema

alias of `BlockStock`

record(*entity*, *start*, *end*, *size*, *timestamps*)

implement the recording logic in this method, should return json or domain list

Parameters::

- **entity** –
- **start** –
- **end** –
- **size** –
- **timestamps** –

clear_state_data(*entity_id=None*)

clear state of the entity

Parameters:: **entity_id** – entity id

decode_state(*state: str*)

decode state

Parameters:: **state** –

Returns::

encode_state(*state: object*)

encode state

Parameters:: **state** –

Returns::

generate_domain(*entity*, *original_data*)

generate the data_schema instance using entity and original_data,the original_data is from record result

Parameters::

- **entity** –
- **original_data** –

generate_domain_id(*entity*, *original_data*, *time_fmt='YYYY-MM-DD'*)

generate domain id from the entity and original data,the default id meaning:entity + event happen time

Parameters::

- **entity** –
- **original_data** –
- **time_fmt** –

Returns::

Return type::

get_data_map()

{'original_field':('domain_field',transform_func)}

get_evaluated_time_field()

the timestamp field for evaluating time range of recorder,used in get_latest_saved_record

init_entities()

init the entities which we would record data for

name = *None*

name of the service, default name of class if not set manually

original_page_url = *None*

original page url

persist(*entity*, *domain_list*)

persist the domain list to db

Parameters::

- **entity** –
- **domain_list** –

state_schema

alias of `RecorderState`

url = *None*

request url

exchanges

setup the entities you want to record

entity_ids

set entity_ids or (entity_type,exchanges,codes)

session

using to do db operations

zvt.recorders.sina.money_flow

Modules

zvt.recorders.sina.money_flow.sina_block_money_flow_recorder

---

zvt.recorders.sina.money_flow.sina_stock_money_flow_recorder

---

zvt.recorders.sina.money_flow.sina_block_money_flow_recorder

Classes

`SinaBlockMoneyFlowRecorder`([force_update, ...])

SinaBlockMoneyFlowRecorder

class zvt.recorders.sina.money_flow.sina_block_money_flow_recorder.SinaBlockMoneyFlowRecorder(*force_update=True*, *sleeping_time=10*, *exchanges=None*, *entity_id=None*, *entity_ids=None*, *code=None*, *codes=None*, *day_data=False*, *entity_filters=None*, *ignore_failed=True*, *real_time=False*, *fix_duplicate_way='ignore'*, *start_timestamp=None*, *end_timestamp=None*, *level=IntervalLevel.LEVEL_1DAY*, *kdata_use_begin_time=False*, *one_day_trading_minutes=1440*, *return_unfinished=False*)

Bases: `FixedCycleDataRecorder`

__init__(*force_update=True*, *sleeping_time=10*, *exchanges=None*, *entity_id=None*, *entity_ids=None*, *code=None*, *codes=None*, *day_data=False*, *entity_filters=None*, *ignore_failed=True*, *real_time=False*, *fix_duplicate_way='ignore'*, *start_timestamp=None*, *end_timestamp=None*, *level=IntervalLevel.LEVEL_1DAY*, *kdata_use_begin_time=False*, *one_day_trading_minutes=1440*, *return_unfinished=False*) → None

Parameters::

- **code** –
- **ignore_failed** –
- **entity_filters** –
- **exchanges** –
- **entity_id** – for record single entity
- **entity_ids** – set entity_ids or (entity_type,exchanges,codes)
- **codes** –
- **day_data** – one record per day,set to True if you want skip recording it when data of today exist
- **force_update** –
- **sleeping_time** –

entity_provider*: str = 'sina'*

overwrite them to fetch the entity list

entity_schema

alias of `Block`

provider*: str = 'sina'*

overwrite them to set up the data you want to record

data_schema

alias of `BlockMoneyFlow`

url = *'http://vip.stock.finance.sina.com.cn/quotes_service/api/json_v2.php/MoneyFlow.ssl_bkzj_zjlrqs?page=1&num={}&sort=opendate&asc=0&bankuai={}%2F{}'*

request url

get_data_map()

{'original_field':('domain_field',transform_func)}

record(*entity*, *start*, *end*, *size*, *timestamps*)

implement the recording logic in this method, should return json or domain list

Parameters::

- **entity** –
- **start** –
- **end** –
- **size** –
- **timestamps** –

clear_state_data(*entity_id=None*)

clear state of the entity

Parameters:: **entity_id** – entity id

decode_state(*state: str*)

decode state

Parameters:: **state** –
Returns::

encode_state(*state: object*)

encode state

Parameters:: **state** –
Returns::

generate_domain(*entity*, *original_data*)

generate the data_schema instance using entity and original_data,the original_data is from record result

Parameters::
- **entity** –
- **original_data** –

generate_domain_id(*entity*, *original_data*, *time_fmt='YYYY-MM-DD'*)

generate domain id from the entity and original data,the default id meaning:entity + event happen time

Parameters::
- **entity** –
- **original_data** –
- **time_fmt** –

Returns::
Return type::

get_evaluated_time_field()

the timestamp field for evaluating time range of recorder,used in get_latest_saved_record

init_entities()

init the entities which we would record data for

name = *None*

name of the service, default name of class if not set manually

original_page_url = *None*

original page url

persist(*entity*, *domain_list*)

persist the domain list to db

Parameters::

- **entity** –
- **domain_list** –

state_schema

alias of `RecorderState`

exchanges

setup the entities you want to record

entity_ids

set entity_ids or (entity_type,exchanges,codes)

session

using to do db operations

zvt.recorders.sina.money_flow.sina_stock_money_flow_recorder

Classes

`SinaStockMoneyFlowRecorder`([force_update, ...])

SinaStockMoneyFlowRecorder

class zvt.recorders.sina.money_flow.sina_stock_money_flow_recorder.SinaStockMoneyFlowRecorder(*force_update=True*, *sleeping_time=10*, *exchanges=None*, *entity_id=None*, *entity_ids=None*, *code=None*, *codes=None*, *day_data=False*, *entity_filters=None*, *ignore_failed=True*, *real_time=False*, *fix_duplicate_way='ignore'*, *start_timestamp=None*, *end_timestamp=None*, *level=IntervalLevel.LEVEL_1DAY*, *kdata_use_begin_time=False*, *one_day_trading_minutes=1440*, *return_unfinished=False*)

Bases: `FixedCycleDataRecorder`

__init__(*force_update=True*, *sleeping_time=10*, *exchanges=None*, *entity_id=None*, *entity_ids=None*, *code=None*, *codes=None*, *day_data=False*, *entity_filters=None*, *ignore_failed=True*, *real_time=False*, *fix_duplicate_way='ignore'*, *start_timestamp=None*, *end_timestamp=None*, *level=IntervalLevel.LEVEL_1DAY*, *kdata_use_begin_time=False*, *one_day_trading_minutes=1440*, *return_unfinished=False*) → None

Parameters::

- **code** –
- **ignore_failed** –
- **entity_filters** –
- **exchanges** –
- **entity_id** – for record single entity
- **entity_ids** – set entity_ids or (entity_type,exchanges,codes)
- **codes** –
- **day_data** – one record per day,set to True if you want skip recording it when data of today exist
- **force_update** –
- **sleeping_time** –

entity_provider*: str = 'joinquant'*

overwrite them to fetch the entity list

entity_schema

alias of `Stock`

provider*: str = 'sina'*

overwrite them to set up the data you want to record

data_schema

alias of `StockMoneyFlow`

url = *'http://vip.stock.finance.sina.com.cn/quotes_service/api/json_v2.php/MoneyFlow.ssl_qsfx_lscjfb?page=1&num={}&sort=opendate&asc=0&daima={}'*

request url

init_entities()

init the entities which we would record data for

get_data_map()

{'original_field':('domain_field',transform_func)}

record(*entity*, *start*, *end*, *size*, *timestamps*)

implement the recording logic in this method, should return json or domain list

Parameters::

- **entity** –
- **start** –
- **end** –
- **size** –
- **timestamps** –

clear_state_data(*entity_id=None*)

clear state of the entity

Parameters:: **entity_id** – entity id

decode_state(*state: str*)

decode state

Parameters:: **state** –

Returns::

encode_state(*state: object*)

encode state

Parameters:: **state** –

Returns::

generate_domain(*entity*, *original_data*)

generate the data_schema instance using entity and original_data,the original_data is from record result

Parameters::

- **entity** –
- **original_data** –

generate_domain_id(*entity*, *original_data*, *time_fmt='YYYY-MM-DD'*)

generate domain id from the entity and original data,the default id meaning:entity + event happen time

Parameters::

- **entity** –
- **original_data** –
- **time_fmt** –

Returns::

Return type::

get_evaluated_time_field()

the timestamp field for evaluating time range of recorder,used in get_latest_saved_record

name = *None*

name of the service, default name of class if not set manually

original_page_url = *None*

original page url

persist(*entity*, *domain_list*)

persist the domain list to db

Parameters::

- **entity** –
- **domain_list** –

state_schema

alias of `RecorderState`

exchanges

setup the entities you want to record

entity_ids

set entity_ids or (entity_type,exchanges,codes)

session

using to do db operations

zvt.recorders.sina.quotes

Modules

zvt.recorders.sina.quotes.sina_etf_kdata_recorder

---

zvt.recorders.sina.quotes.sina_index_kdata_recorder

---

zvt.recorders.sina.quotes.sina_etf_kdata_recorder

Classes

`ChinaETFDayKdataRecorder`([force_update, ...])

ChinaETFDayKdataRecorder

class zvt.recorders.sina.quotes.sina_etf_kdata_recorder.ChinaETFDayKdataRecorder(*force_update=True*, *sleeping_time=10*, *exchanges=None*, *entity_id=None*, *entity_ids=None*, *code=None*, *codes=None*, *day_data=False*, *entity_filters=None*, *ignore_failed=True*, *real_time=False*, *fix_duplicate_way='ignore'*, *start_timestamp=None*, *end_timestamp=None*, *level=IntervalLevel.LEVEL_1DAY*, *kdata_use_begin_time=False*, *one_day_trading_minutes=1440*, *return_unfinished=False*)

Bases: `FixedCycleDataRecorder`

__init__(*force_update=True*, *sleeping_time=10*, *exchanges=None*, *entity_id=None*, *entity_ids=None*, *code=None*, *codes=None*, *day_data=False*, *entity_filters=None*, *ignore_failed=True*, *real_time=False*, *fix_duplicate_way='ignore'*, *start_timestamp=None*, *end_timestamp=None*, *level=IntervalLevel.LEVEL_1DAY*, *kdata_use_begin_time=False*, *one_day_trading_minutes=1440*, *return_unfinished=False*) → None

Parameters::

- **code** –
- **ignore_failed** –
- **entity_filters** –
- **exchanges** –
- **entity_id** – for record single entity
- **entity_ids** – set entity_ids or (entity_type,exchanges,codes)
- **codes** –
- **day_data** – one record per day,set to True if you want skip recording it when data of today exist
- **force_update** –
- **sleeping_time** –

entity_provider*: str = 'exchange'*

overwrite them to fetch the entity list

entity_schema

alias of `Etf`

provider*: str = 'sina'*

overwrite them to set up the data you want to record

data_schema

alias of `Etf1dKdata`

url = *'http://money.finance.sina.com.cn/quotes_service/api/json_v2.php/CN_MarketData.getKLineData?symbol={}{}&scale=240&&datalen={}&ma=no'*

request url

get_data_map()

{'original_field':('domain_field',transform_func)}

generate_domain_id(*entity*, *original_data*)

generate domain id from the entity and original data,the default id meaning:entity + event happen time

Parameters::

- **entity** –
- **original_data** –
- **time_fmt** –

Returns::

Return type::

record(*entity*, *start*, *end*, *size*, *timestamps*)

implement the recording logic in this method, should return json or domain list

Parameters::

- **entity** –
- **start** –
- **end** –
- **size** –
- **timestamps** –

clear_state_data(*entity_id=None*)

clear state of the entity

Parameters:: **entity_id** – entity id

decode_state(*state: str*)

decode state

Parameters:: **state** –

Returns::

encode_state(*state: object*)

encode state

Parameters:: **state** –

Returns::

generate_domain(*entity*, *original_data*)

generate the data_schema instance using entity and original_data,the original_data is from record result

Parameters::

- **entity** –
- **original_data** –

get_evaluated_time_field()

the timestamp field for evaluating time range of recorder,used in get_latest_saved_record

init_entities()

init the entities which we would record data for

name = *None*

name of the service, default name of class if not set manually

original_page_url = *None*

original page url

persist(*entity*, *domain_list*)

persist the domain list to db

Parameters::

- **entity** –
- **domain_list** –

state_schema

alias of `RecorderState`

exchanges

setup the entities you want to record

entity_ids

set entity_ids or (entity_type,exchanges,codes)

session

using to do db operations

zvt.recorders.sina.quotes.sina_index_kdata_recorder

Classes

`ChinaIndexDayKdataRecorder`([force_update, ...])

ChinaIndexDayKdataRecorder

class zvt.recorders.sina.quotes.sina_index_kdata_recorder.ChinaIndexDayKdataRecorder(*force_update=True*, *sleeping_time=10*, *exchanges=None*, *entity_id=None*, *entity_ids=None*, *code=None*, *codes=None*, *day_data=False*, *entity_filters=None*, *ignore_failed=True*, *real_time=False*, *fix_duplicate_way='ignore'*, *start_timestamp=None*, *end_timestamp=None*, *level=IntervalLevel.LEVEL_1DAY*, *kdata_use_begin_time=False*, *one_day_trading_minutes=1440*, *return_unfinished=False*)

Bases: `FixedCycleDataRecorder`

__init__(*force_update=True*, *sleeping_time=10*, *exchanges=None*, *entity_id=None*, *entity_ids=None*, *code=None*, *codes=None*, *day_data=False*, *entity_filters=None*, *ignore_failed=True*, *real_time=False*, *fix_duplicate_way='ignore'*, *start_timestamp=None*, *end_timestamp=None*, *level=IntervalLevel.LEVEL_1DAY*, *kdata_use_begin_time=False*, *one_day_trading_minutes=1440*, *return_unfinished=False*) → None

Parameters::

- **code** –
- **ignore_failed** –
- **entity_filters** –
- **exchanges** –
- **entity_id** – for record single entity
- **entity_ids** – set entity_ids or (entity_type,exchanges,codes)
- **codes** –
- **day_data** – one record per day,set to True if you want skip recording it when data of today exist
- **force_update** –
- **sleeping_time** –

entity_provider*: str = 'exchange'*

overwrite them to fetch the entity list

entity_schema

alias of `Index`

provider*: str = 'sina'*

overwrite them to set up the data you want to record

data_schema

alias of `Index1dKdata`

url = *'http://vip.stock.finance.sina.com.cn/corp/go.php/vMS_MarketHistory/stockid/{}/type/S.phtml?year={}&jidu={}'*

request url

get_data_map()

{'original_field':('domain_field',transform_func)}

generate_domain_id(*entity*, *original_data*)

generate domain id from the entity and original data,the default id meaning:entity + event happen time

Parameters::

- **entity** –
- **original_data** –
- **time_fmt** –

Returns::

Return type::

record(*entity*, *start*, *end*, *size*, *timestamps*)

implement the recording logic in this method, should return json or domain list

Parameters::

- **entity** –
- **start** –
- **end** –
- **size** –
- **timestamps** –

clear_state_data(*entity_id=None*)

clear state of the entity

Parameters:: **entity_id** – entity id

decode_state(*state: str*)

decode state

Parameters:: **state** –
Returns::

encode_state(*state: object*)

encode state

Parameters:: **state** –
Returns::

generate_domain(*entity*, *original_data*)

generate the data_schema instance using entity and original_data,the original_data is from record result

Parameters::
- **entity** –
- **original_data** –

get_evaluated_time_field()

the timestamp field for evaluating time range of recorder,used in get_latest_saved_record

init_entities()

init the entities which we would record data for

name = *None*

name of the service, default name of class if not set manually

original_page_url = *None*

original page url

persist(*entity*, *domain_list*)

persist the domain list to db

Parameters::

- **entity** –
- **domain_list** –

state_schema

alias of `RecorderState`

exchanges

setup the entities you want to record

entity_ids

set entity_ids or (entity_type,exchanges,codes)

session

using to do db operations

zvt.recorders.wb

Modules

zvt.recorders.wb.wb_api

---

zvt.recorders.wb.wb_country_recorder

---

zvt.recorders.wb.wb_economy_recorder

---

zvt.recorders.wb.wb_api

Functions

get_countries()	
get_economy_data(entity_id[, indicators, date])	
get_incomelevels([incomelevel, language])	Return a DataFrame that describes one, multiple or all income levels, indexed by the IL id.
get_indicator_data(indicator[, ...])	
get_indicators([indicator, language, ...])	Return a DataFrame that describes one, multiple or all indicators, indexed by the indicator id.
get_lendingtypes([lendingtype, language])	Return a DataFrame that describes one, multiple or all lending types, indexed by the LT id.
get_regions([region, language])	Return a DataFrame that describes one, multiple or all regions, indexed by the region id.

`get_sources`([source, language])	Return a DataFrame that describes one, multiple or all sources, indexed by the source id.
`get_topics`([topic, language])	Return a DataFrame that describes one, multiple or all sources, indexed by the source id.

zvt.recorders.wb.wb_api.get_countries

zvt.recorders.wb.wb_api.get_countries()

zvt.recorders.wb.wb_api.get_economy_data

zvt.recorders.wb.wb_api.get_economy_data(*entity_id*, *indicators=None*, *date=None*)

zvt.recorders.wb.wb_api.get_incomelevels

zvt.recorders.wb.wb_api.get_incomelevels(*incomelevel=None*, *language=None*, ***params*)

Return a DataFrame that describes one, multiple or all income levels, indexed by the IL id. :param incomelevel: None (all income levels), the id of an income level, or a list of multiple ids :param language: Desired language

zvt.recorders.wb.wb_api.get_indicator_data

zvt.recorders.wb.wb_api.get_indicator_data(*indicator*, *indicator_name=None*, *country=None*, *date=None*)

zvt.recorders.wb.wb_api.get_indicators

zvt.recorders.wb.wb_api.get_indicators(*indicator=None*, *language=None*, *id_or_value=None*, ***params*)

Return a DataFrame that describes one, multiple or all indicators, indexed by the indicator id. :param indicator: None (all indicators), the id of an indicator, or a list of multiple ids :param language: Desired language :param id_or_value: Choose either ‘id’ or ‘value’ for columns ‘source’ and ‘topics’

zvt.recorders.wb.wb_api.get_lendingtypes

zvt.recorders.wb.wb_api.get_lendingtypes(*lendingtype=None*, *language=None*, ***params*)

Return a DataFrame that describes one, multiple or all lending types, indexed by the LT id. :param lendingtype: None (all lending types), the id of a lending type, or a list of multiple ids :param language: Desired language

zvt.recorders.wb.wb_api.get_regions

zvt.recorders.wb.wb_api.get_regions(*region=None*, *language=None*, ***params*)

Return a DataFrame that describes one, multiple or all regions, indexed by the region id. :param region: None (all regions), the id of a region, or a list of multiple ids :param language: Desired language

zvt.recorders.wb.wb_api.get_sources

zvt.recorders.wb.wb_api.get_sources(*source=None*, *language=None*, ***params*)

Return a DataFrame that describes one, multiple or all sources, indexed by the source id. :param source: None (all sources), the id of a source, or a list of multiple ids :param language: Desired language

zvt.recorders.wb.wb_api.get_topics

zvt.recorders.wb.wb_api.get_topics(*topic=None*, *language=None*, ***params*)

Return a DataFrame that describes one, multiple or all sources, indexed by the source id. :param topic: None (all topics), the id of a topic, or a list of multiple ids :param language: Desired language

zvt.recorders.wb.wb_country_recorder

Classes

`WBCountryRecorder`([force_update, sleeping_time])

WBCountryRecorder

class zvt.recorders.wb.wb_country_recorder.WBCountryRecorder(*force_update: bool = False*, *sleeping_time: int = 10*)

Bases: `Recorder`

__init__(*force_update: bool = False*, *sleeping_time: int = 10*) → None

provider*: str = 'wb'*

overwrite them to set up the data you want to record

data_schema

alias of `Country`

clear_state_data(*entity_id=None*)

clear state of the entity

Parameters:: **entity_id** – entity id

decode_state(*state: str*)

decode state

Parameters:: **state** –
Returns::

encode_state(*state: object*)

encode state

Parameters:: **state** –
Returns::

name = *None*

name of the service, default name of class if not set manually

original_page_url = *None*

original page url

state_schema

alias of `RecorderState`

url = *None*

request url

session

using to do db operations

zvt.recorders.wb.wb_economy_recorder

Classes

`WBEconomyRecorder`([force_update, ...])

WBEconomyRecorder

class zvt.recorders.wb.wb_economy_recorder.WBEconomyRecorder(*force_update=True*, *sleeping_time=10*, *exchanges=None*, *entity_id=None*, *entity_ids=None*, *code=None*, *codes=None*, *day_data=False*, *entity_filters=None*, *ignore_failed=True*, *real_time=False*, *fix_duplicate_way='ignore'*, *start_timestamp=None*, *end_timestamp=None*, *level=IntervalLevel.LEVEL_1DAY*, *kdata_use_begin_time=False*, *one_day_trading_minutes=1440*, *return_unfinished=False*)

Bases: `FixedCycleDataRecorder`

__init__(*force_update=True*, *sleeping_time=10*, *exchanges=None*, *entity_id=None*, *entity_ids=None*, *code=None*, *codes=None*, *day_data=False*, *entity_filters=None*, *ignore_failed=True*, *real_time=False*, *fix_duplicate_way='ignore'*, *start_timestamp=None*, *end_timestamp=None*, *level=IntervalLevel.LEVEL_1DAY*, *kdata_use_begin_time=False*, *one_day_trading_minutes=1440*, *return_unfinished=False*) → None

Parameters::

- **code** –
- **ignore_failed** –
- **entity_filters** –
- **exchanges** –
- **entity_id** – for record single entity
- **entity_ids** – set entity_ids or (entity_type,exchanges,codes)
- **codes** –
- **day_data** – one record per day,set to True if you want skip recording it when data of today exist
- **force_update** –
- **sleeping_time** –

entity_schema

alias of `Country`

data_schema

alias of `Economy`

entity_provider*: str = 'wb'*

overwrite them to fetch the entity list

provider*: str = 'wb'*

overwrite them to set up the data you want to record

record(*entity*, *start*, *end*, *size*, *timestamps*)

implement the recording logic in this method, should return json or domain list

Parameters::

- **entity** –
- **start** –
- **end** –
- **size** –
- **timestamps** –

clear_state_data(*entity_id=None*)

clear state of the entity

Parameters:: **entity_id** – entity id

decode_state(*state: str*)

decode state

Parameters:: **state** –

Returns::

encode_state(*state: object*)

encode state

Parameters:: **state** –

Returns::

generate_domain(*entity*, *original_data*)

generate the data_schema instance using entity and original_data,the original_data is from record result

Parameters::
- **entity** –
- **original_data** –

generate_domain_id(*entity*, *original_data*, *time_fmt='YYYY-MM-DD'*)

generate domain id from the entity and original data,the default id meaning:entity + event happen time

Parameters::
- **entity** –
- **original_data** –
- **time_fmt** –

Returns::

Return type::

get_data_map()

{'original_field':('domain_field',transform_func)}

get_evaluated_time_field()

the timestamp field for evaluating time range of recorder,used in get_latest_saved_record

init_entities()

init the entities which we would record data for

name = *None*

name of the service, default name of class if not set manually

original_page_url = *None*

original page url

persist(*entity*, *domain_list*)

persist the domain list to db

Parameters::
- **entity** –
- **domain_list** –

state_schema

alias of `RecorderState`

url = *None*

request url

exchanges

setup the entities you want to record

entity_ids

set entity_ids or (entity_type,exchanges,codes)

session

using to do db operations

zvt.rest

Modules

zvt.rest.data

---

zvt.rest.factor

---

zvt.rest.trading

---

zvt.rest.work

---

zvt.rest.data

Functions

`get_data_providers`()	Get data providers
`get_data_schemas`(provider)	Get schemas by provider
`query_data`(provider, schema)	Get schemas by provider

zvt.rest.data.get_data_providers

zvt.rest.data.get_data_providers()

Get data providers

zvt.rest.data.get_data_schemas

zvt.rest.data.get_data_schemas(*provider*)

Get schemas by provider

zvt.rest.data.query_data

zvt.rest.data.query_data(*provider: str*, *schema: str*)

Get schemas by provider

zvt.rest.factor

Functions

`get_factors`()

`query_factor_result`(factor_request_model)

`query_kdata`(kdata_request_model)

---

zvt.rest.factor.get_factors

zvt.rest.factor.get_factors()

zvt.rest.factor.query_factor_result

zvt.rest.factor.query_factor_result(*factor_request_model: FactorRequestModel*)

zvt.rest.factor.query_kdata

zvt.rest.factor.query_kdata(*kdata_request_model: KdataRequestModel*)

zvt.rest.trading

Functions

build_query_stock_quote_setting(...)

---

build_trading_plan(build_trading_plan_model)

---

buy(buy_position_strategy)

---

get_current_trading_plan()

---

get_future_trading_plan()

---

get_query_stock_quote_setting()

---

query_trading_plan(query_trading_plan_model)

---

sell(sell_position_strategy)

---

zvt.rest.trading.build_query_stock_quote_setting

zvt.rest.trading.build_query_stock_quote_setting(*build_query_stock_quote_setting_model: BuildQueryStockQuoteSettingModel*)

zvt.rest.trading.build_trading_plan

zvt.rest.trading.build_trading_plan(*build_trading_plan_model: BuildTradingPlanModel*)

zvt.rest.trading.buy

zvt.rest.trading.buy(*buy_position_strategy: BuyParameter*)

zvt.rest.trading.get_current_trading_plan

zvt.rest.trading.get_current_trading_plan()

zvt.rest.trading.get_future_trading_plan

zvt.rest.trading.get_future_trading_plan()

zvt.rest.trading.get_query_stock_quote_setting

zvt.rest.trading.get_query_stock_quote_setting()

zvt.rest.trading.query_trading_plan

zvt.rest.trading.query_trading_plan(*query_trading_plan_model: QueryTradingPlanModel*)

zvt.rest.trading.sell

zvt.rest.trading.sell(*sell_position_strategy: SellParameter*)

zvt.rest.work

Functions

activate_sub_tags(activate_sub_tags_model)	Activate sub tags
batch_set_stock_tags(batch_set_stock_tags_model)	
build_stock_tags(set_stock_tags_model_list)	Set stock tags in batch
create_hidden_tag_info(tag_info)	
create_main_tag_info(tag_info)	
create_stock_pool_info(...)	
create_stock_pools(create_stock_pools_model)	
create_sub_tag_info(tag_info)	
get_hidden_tag_info()	Get hidden_tag info
get_main_tag_info()	Get main_tag info

get_stock_pool_info()	
get_stock_pools(stock_pool_name)	
get_sub_tag_info()	Get sub_tag info
query_simple_stock_tags(...)	Get simple entity tags
query_stock_tag_stats(...)	Get stock tag stats
query_stock_tags(query_stock_tags_model)	Get entity tags
set_stock_tags(set_stock_tags_model)	Set stock tags

zvt.rest.work.activate_sub_tags

zvt.rest.work.activate_sub_tags(*activate_sub_tags_model: ActivateSubTagsModel*)

Activate sub tags

zvt.rest.work.batch_set_stock_tags

zvt.rest.work.batch_set_stock_tags(*batch_set_stock_tags_model: BatchSetStockTagsModel*)

zvt.rest.work.build_stock_tags

zvt.rest.work.build_stock_tags(*set_stock_tags_model_list: List[SetStockTagsModel]*)

Set stock tags in batch

zvt.rest.work.create_hidden_tag_info

zvt.rest.work.create_hidden_tag_info(*tag_info: CreateTagInfoModel*)

zvt.rest.work.create_main_tag_info

zvt.rest.work.create_main_tag_info(*tag_info: CreateTagInfoModel*)

zvt.rest.work.create_stock_pool_info

zvt.rest.work.create_stock_pool_info(*create_stock_pool_info_model: CreateStockPoolInfoModel*)

zvt.rest.work.create_stock_pools

zvt.rest.work.create_stock_pools(*create_stock_pools_model: CreateStockPoolsModel*)

zvt.rest.work.create_sub_tag_info

zvt.rest.work.create_sub_tag_info(*tag_info: CreateTagInfoModel*)

zvt.rest.work.get_hidden_tag_info

zvt.rest.work.get_hidden_tag_info()

Get hidden_tag info

zvt.rest.work.get_main_tag_info

zvt.rest.work.get_main_tag_info()

Get main_tag info

zvt.rest.work.get_stock_pool_info

zvt.rest.work.get_stock_pool_info()

zvt.rest.work.get_stock_pools

zvt.rest.work.get_stock_pools(*stock_pool_name: str*)

zvt.rest.work.get_sub_tag_info

zvt.rest.work.get_sub_tag_info()

Get sub_tag info

zvt.rest.work.query_simple_stock_tags

zvt.rest.work.query_simple_stock_tags(*query_simple_stock_tags_model: QuerySimpleStockTagsModel*)

Get simple entity tags

zvt.rest.work.query_stock_tag_stats

zvt.rest.work.query_stock_tag_stats(*query_stock_tag_stats_model: QueryStockTagStatsModel*)

Get stock tag stats

zvt.rest.work.query_stock_tags

zvt.rest.work.query_stock_tags(*query_stock_tags_model: QueryStockTagsModel*)

Get entity tags

zvt.rest.work.set_stock_tags

zvt.rest.work.set_stock_tags(*set_stock_tags_model: SetStockTagsModel*)

Set stock tags

zvt.samples

Modules

zvt.samples.stock_traders

---

zvt.samples.stock_traders

Classes

`MyBullTrader`([entity_ids, exchanges, codes, ...])

`MyMaTrader`([entity_ids, exchanges, codes, ...])

MyBullTrader

class zvt.samples.stock_traders.MyBullTrader(*entity_ids: Optional[List[str]] = None*, *exchanges: Optional[List[str]] = None*, *codes: Optional[List[str]] = None*, *start_timestamp: Optional[Union[Timestamp, str]] = None*, *end_timestamp: Optional[Union[Timestamp, str]] = None*, *provider: Optional[str] = None*, *level: Union[str, IntervalLevel] = IntervalLevel.LEVEL_1DAY*, *trader_name: Optional[str] = None*, *real_time: bool = False*, *kdata_use_begin_time: bool = False*, *draw_result: bool = True*, *rich_mode: bool = False*, *adjust_type: AdjustType = AdjustType.hfq*, *profit_threshold=(3, - 0.3)*, *keep_history=False*)

Bases: `StockTrader`

__init__(*entity_ids: Optional[List[str]] = None*, *exchanges: Optional[List[str]] = None*, *codes: Optional[List[str]] = None*, *start_timestamp: Optional[Union[Timestamp, str]] = None*, *end_timestamp: Optional[Union[Timestamp, str]] = None*, *provider: Optional[str] = None*, *level: Union[str, IntervalLevel] = IntervalLevel.LEVEL_1DAY*, *trader_name: Optional[str] = None*, *real_time: bool = False*, *kdata_use_begin_time: bool = False*, *draw_result: bool = True*, *rich_mode: bool = False*, *adjust_type: AdjustType = AdjustType.hfq*, *profit_threshold=(3, - 0.3)*, *keep_history=False*) → None

init_factors(*entity_ids*, *entity_schema*, *exchanges*, *codes*, *start_timestamp*, *end_timestamp*, *adjust_type=None*)

overwrite it to init factors if you want to use factor computing model
:param adjust_type:

entity_schema

alias of `Stock`

handle_factor_targets(*timestamp: Timestamp*)

select targets from factors :param timestamp: the timestamp for next kdata coming

init_entities(*timestamp*)

init the entities for timestamp

Parameters:: **timestamp** –

Returns::

on_factor_targets_filtered(*timestamp*, *level*, *factor: Factor*, *long_targets: List[str]*, *short_targets: List[str]*) → Tuple[List[str], List[str]]

overwrite it to filter the targets from factor

Parameters::

- **timestamp** – the event time
- **level** – the level
- **factor** – the factor
- **long_targets** – the long targets from the factor
- **short_targets** – the short targets from the factor

Returns:: filtered long targets, filtered short targets

on_targets_selected_from_levels(*timestamp*) → Tuple[List[str], List[str]]

this method's called in every min level cycle to select targets in all levels generated by the previous cycle the default implementation is selecting the targets in all levels overwrite it for your custom logic

Parameters:: **timestamp** – current event time

Returns:: long targets, short targets

on_time(*timestamp: Timestamp*)

called in every min level cycle

Parameters:: **timestamp** – event time

update_targets_by_level(*level: IntervalLevel*, *long_targets: List[str]*, *short_targets: List[str]*) → None

the trading signals is generated in min level,before that,we should cache targets of all levels

Parameters::

- **level** –
- **long_targets** –
- **short_targets** –

MyMaTrader

class zvt.samples.stock_traders.MyMaTrader(*entity_ids: Optional[List[str]] = None*, *exchanges: Optional[List[str]] = None*, *codes: Optional[List[str]] = None*, *start_timestamp: Optional[Union[Timestamp, str]] = None*, *end_timestamp: Optional[Union[Timestamp, str]] = None*, *provider: Optional[str] = None*, *level: Union[str, IntervalLevel] = IntervalLevel.LEVEL_1DAY*, *trader_name: Optional[str] = None*, *real_time: bool = False*, *kdata_use_begin_time: bool = False*, *draw_result: bool = True*, *rich_mode: bool = False*, *adjust_type: AdjustType = AdjustType.hfq*, *profit_threshold=(3, - 0.3)*, *keep_history=False*)

Bases: `StockTrader`

__init__(*entity_ids: Optional[List[str]] = None*, *exchanges: Optional[List[str]] = None*, *codes: Optional[List[str]] = None*, *start_timestamp: Optional[Union[Timestamp, str]] = None*, *end_timestamp: Optional[Union[Timestamp, str]] = None*, *provider: Optional[str] = None*, *level: Union[str, IntervalLevel] = IntervalLevel.LEVEL_1DAY*, *trader_name: Optional[str] = None*, *real_time: bool = False*, *kdata_use_begin_time: bool = False*, *draw_result: bool = True*, *rich_mode: bool = False*, *adjust_type: AdjustType = AdjustType.hfq*, *profit_threshold=(3, - 0.3)*, *keep_history=False*) → None

init_factors(*entity_ids*, *entity_schema*, *exchanges*, *codes*, *start_timestamp*, *end_timestamp*, *adjust_type=None*)

overwrite it to init factors if you want to use factor computing model
:param adjust_type:

entity_schema

alias of `Stock`

handle_factor_targets(*timestamp: Timestamp*)

select targets from factors :param timestamp: the timestamp for next kdata coming

init_entities(*timestamp*)

init the entities for timestamp

Parameters:: **timestamp** –

Returns::

on_factor_targets_filtered(*timestamp*, *level*, *factor: Factor*, *long_targets: List[str]*, *short_targets: List[str]*) → Tuple[List[str], List[str]]

overwrite it to filter the targets from factor

Parameters::

- **timestamp** – the event time
- **level** – the level
- **factor** – the factor
- **long_targets** – the long targets from the factor
- **short_targets** – the short targets from the factor

Returns:: filtered long targets, filtered short targets

on_targets_selected_from_levels(*timestamp*) → Tuple[List[str], List[str]]

this method's called in every min level cycle to select targets in all levels generated by the previous cycle the default implementation is selecting the targets in all levels overwrite it for your custom logic

Parameters:: **timestamp** – current event time

Returns:: long targets, short targets

on_time(*timestamp: Timestamp*)

called in every min level cycle

Parameters:: **timestamp** – event time

update_targets_by_level(*level: IntervalLevel*, *long_targets: List[str]*, *short_targets: List[str]*) → None

the trading signals is generated in min level,before that,we should cache targets of all levels

Parameters::

- **level** –
- **long_targets** –
- **short_targets** –

zvt.sched

Modules

zvt.sched.sched

---

zvt.sched.sched

Functions

`sched_tasks`()

---

zvt.sched.sched.sched_tasks

zvt.sched.sched.sched_tasks()

zvt.tag

Modules

zvt.tag.common

---

zvt.tag.tag_models

---

zvt.tag.tag_schemas

---

zvt.tag.tag_service

---

zvt.tag.tag_stats

---

zvt.tag.tag_utils

---

zvt.tag.tagger

---

zvt.tag.common

Classes

StockPoolType(value)	An enumeration.
TagType(value)	An enumeration.

StockPoolType

class zvt.tag.common.StockPoolType(*value*)

Bases: `Enum`

An enumeration.

__init__()

TagType

class zvt.tag.common.TagType(*value*)

Bases: `Enum`

An enumeration.

__init__()

zvt.tag.tag_models

Classes

`ActivateSubTagsModel`(*, sub_tags)

`ActivateSubTagsResultModel`(*, tag_entity_ids)

`BatchSetStockTagsModel`(*, entity_ids, tag[, ...])

`CreateStockPoolInfoModel`(*, stock_pool_type, ...)

`CreateStockPoolsModel`(*, stock_pool_name, ...)

`CreateTagInfoModel`(*, tag[, tag_reason])

`QuerySimpleStockTagsModel`(*, entity_ids)

`QueryStockTagStatsModel`(*[, ...])

`QueryStockTagsModel`(*, entity_ids)

`SetStockTagsModel`(*, entity_id, main_tag[, ...])

`SimpleStockTagsModel`(*, entity_id, name, ...)

StockPoolInfoModel(*, id, entity_id, ...)

StockPoolModel(*, id, entity_id, timestamp, ...)

StockPoolsModel(*, id, entity_id, timestamp, ...)

StockTagDetailsModel(*, entity_id, main_tag, ...)

StockTagStatsModel(*, id, entity_id, ...)

StockTagsModel(*, id, entity_id, timestamp, ...)

TagInfoModel(*, id, entity_id, timestamp, tag)

TagParameter(*, main_tag[, main_tag_reason, ...])

ActivateSubTagsModel

class zvt.tag.tag_models.ActivateSubTagsModel(*, *sub_tags: List[str]*)

Bases: `CustomModel`

__init__(**data: Any*) → None

Create a new model by parsing and validating input data from keyword arguments.

Raises [*ValidationError*][pydantic_core.ValidationError] if the input data cannot be validated to form a valid model.

self is explicitly positional-only to allow *self* as a field name.

copy(*, *include: AbstractSetIntStr | MappingIntStrAny | None = None*, *exclude: AbstractSetIntStr | MappingIntStrAny | None = None*, *update: Dict[str, Any] | None = None*, *deep: bool = False*) → Model

Returns a copy of the model.

!!! warning "Deprecated"

This method is now deprecated; use *model_copy* instead.

If you need *include* or *exclude*, use:

```
`py data = self.model_dump(include=include, exclude=exclude, round_trip=True) data = {**data, **(update or {})} copied = self.model_validate(data) `
```

Args:

include: Optional set or mapping specifying which fields to include in the copied model. exclude: Optional set or mapping specifying which fields to exclude in the copied model. update: Optional dictionary of field-value pairs to override field values in the copied model. deep: If True, the values of fields that are Pydantic models will be deep-copied.

Returns:

A copy of the model with included, excluded and updated fields as specified.

model_computed_fields*: ClassVar[dict[str, ComputedFieldInfo]] = {}*

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

model_config*: ClassVar[ConfigDict] = {'allow_inf_nan': True, 'from_attributes': True}*

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

classmethod model_construct(*_fields_set: set[str] | None = None*, ***values: Any*) → Model

Creates a new instance of the *Model* class with validated data.

Creates a new model setting *__dict__* and *__pydantic_fields_set__* from trusted or pre-validated data. Default values are respected, but no other validation is performed. Behaves as if *Config.extra = 'allow'* was set since it adds all passed values

Args:

_fields_set: The set of field names accepted for the Model instance. values: Trusted or pre-validated data dictionary.

Returns:

A new instance of the *Model* class with validated data.

model_copy(*, *update: dict[str, Any] | None = None*, *deep: bool = False*) → Model

Usage docs: https://docs.pydantic.dev/2.6/concepts/serialization/#model_copy

Returns a copy of the model.

Args:

update: Values to change/add in the new model. Note: the data is not validated
before creating the new model. You should trust this data.

deep: Set to *True* to make a deep copy of the model.

Returns:
New model instance.

model_dump(*, *mode: Literal['json', 'python'] | str = 'python'*, *include: IncEx = None*, *exclude: IncEx = None*, *by_alias: bool = False*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*, *round_trip: bool = False*, *warnings: bool = True*) → dict[str, Any]

Usage docs: https://docs.pydantic.dev/2.6/concepts/serialization/#modelmodel_dump

Generate a dictionary representation of the model, optionally specifying which fields to include or exclude.

Args:
mode: The mode in which *to_python* should run.
If mode is 'json', the output will only contain JSON serializable types. If mode is 'python', the output may contain non-JSON-serializable Python objects.

include: A list of fields to include in the output. exclude: A list of fields to exclude from the output. by_alias: Whether to use the field's alias in the dictionary key if defined. exclude_unset: Whether to exclude fields that have not been explicitly set. exclude_defaults: Whether to exclude fields that are set to their default value. exclude_none: Whether to exclude fields that have a value of *None*. round_trip: If True, dumped values should be valid as input for non-idempotent types such as Json[T]. warnings: Whether to log warnings when invalid fields are encountered.

Returns:

A dictionary representation of the model.

model_dump_json(*, *indent: int | None = None*, *include: IncEx = None*, *exclude: IncEx = None*, *by_alias: bool = False*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*, *round_trip: bool = False*, *warnings: bool = True*) → str

Usage docs: https://docs.pydantic.dev/2.6/concepts/serialization/#modelmodel_dump_json

Generates a JSON representation of the model using Pydantic's *to_json* method.

Args:

indent: Indentation to use in the JSON output. If None is passed, the output will be compact. include: Field(s) to include in the JSON output. exclude: Field(s) to exclude from the JSON output. by_alias: Whether to serialize using field aliases. exclude_unset: Whether to exclude fields that have not been explicitly set. exclude_defaults: Whether to exclude fields that are set to their default value. exclude_none: Whether to exclude fields that have a value of *None*. round_trip: If True, dumped values should be valid as input for non-idempotent types such as Json[T]. warnings: Whether to log warnings when invalid fields are encountered.

Returns:

A JSON string representation of the model.

property model_extra*: dict[str, Any] | None*

Get extra fields set during validation.

Returns:

A dictionary of extra fields, or *None* if *config.extra* is not set to *"allow"*.

model_fields*: ClassVar[dict[str, FieldInfo]] = {'sub_tags': FieldInfo(annotation=List[str], required=True)}*

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

property model_fields_set*: set[str]*

Returns the set of fields that have been explicitly set on this model instance.

Returns:

A set of strings representing the fields that have been set, i.e. that were not filled from defaults.

classmethod model_json_schema(*by_alias: bool = True*, *ref_template: str = '#/$defs/{model}'*, *schema_generator: type[pydantic.json_schema.GenerateJsonSchema] = <class 'pydantic.json_schema.GenerateJsonSchema'>*, *mode: ~typing.Literal['validation', 'serialization'] = 'validation'*) → dict[str, Any]

Generates a JSON schema for a model class.

Args:

by_alias: Whether to use attribute aliases or not. ref_template: The reference template. schema_generator: To override the logic used to generate the JSON schema, as a subclass of

GenerateJsonSchema with your desired modifications

mode: The mode in which to generate the schema.

Returns:

The JSON schema for the given model class.

classmethod model_parametrized_name(*params: tuple[type[Any], ...]*) → str

Compute the class name for parametrizations of generic classes.

This method can be overridden to achieve a custom naming scheme for generic BaseModels.

Args:

params: Tuple of types of the class. Given a generic class *Model* with 2 type variables and a concrete model *Model[str, int]*, the value *(str, int)* would be passed to *params*.

Returns:

String representing the new class where *params* are passed to *cls* as type variables.

Raises:

TypeError: Raised when trying to generate concrete names for non-generic models.

model_post_init(*_BaseModel__context: Any*) → None

Override this method to perform additional initialization after *__init__* and *model_construct*. This is useful if you want to do some validation that requires the entire model to be initialized.

classmethod model_rebuild(*, *force: bool = False*, *raise_errors: bool = True*, *_parent_namespace_depth: int = 2*, *_types_namespace: Optional[dict[str, Any]] = None*) → bool | None

Try to rebuild the pydantic-core schema for the model.

This may be necessary when one of the annotations is a ForwardRef which could not be resolved during the initial attempt to build the schema, and automatic rebuilding fails.

Args:

force: Whether to force the rebuilding of the model schema, defaults to *False*. raise_errors: Whether to raise errors, defaults to *True*. _parent_namespace_depth: The depth level of the parent namespace, defaults to 2. _types_namespace: The types namespace, defaults to *None*.

Returns:

Returns *None* if the schema is already "complete" and rebuilding was not required. If rebuilding _was_ required, returns *True* if rebuilding was successful, otherwise *False*.

classmethod model_validate(*obj: Any*, *, *strict: bool | None = None*, *from_attributes: bool | None = None*, *context: dict[str, Any] | None = None*) → Model

Validate a pydantic model instance.

Args:

obj: The object to validate. strict: Whether to enforce types strictly. from_attributes: Whether to extract data from object attributes. context: Additional context to pass to the validator.

Raises:

ValidationError: If the object could not be validated.

Returns:

The validated model instance.

classmethod model_validate_json(*json_data: str | bytes | bytearray*, *, *strict: bool | None = None*, *context: dict[str, Any] | None = None*) → Model

Usage docs: https://docs.pydantic.dev/2.6/concepts/json/#json-parsing

Validate the given JSON data against the Pydantic model.

Args:

json_data: The JSON data to validate. strict: Whether to enforce types strictly. context: Extra variables to pass to the validator.

Returns:

The validated Pydantic model.

Raises:

ValueError: If *json_data* is not a JSON string.

classmethod model_validate_strings(*obj: Any*, *, *strict: bool | None = None*, *context: dict[str, Any] | None = None*) → Model

Validate the given object contains string data against the Pydantic model.

Args:

obj: The object contains string data to validate. strict: Whether to enforce types strictly. context: Extra variables to pass to the validator.

Returns:

The validated Pydantic model.

ActivateSubTagsResultModel

class zvt.tag.tag_models.ActivateSubTagsResultModel(*, *tag_entity_ids: Dict[str, Optional[List[str]]]*)

Bases: `CustomModel`

__init__(**data: Any*) → None

Create a new model by parsing and validating input data from keyword arguments.

Raises [*ValidationError*][pydantic_core.ValidationError] if the input data cannot be validated to form a valid model.

self is explicitly positional-only to allow *self* as a field name.

copy(*, *include: AbstractSetIntStr | MappingIntStrAny | None = None*, *exclude: AbstractSetIntStr | MappingIntStrAny | None = None*, *update: Dict[str, Any] | None = None*, *deep: bool = False*) → Model

Returns a copy of the model.

!!! warning "Deprecated"

This method is now deprecated; use *model_copy* instead.

If you need *include* or *exclude*, use:

```
`py data = self.model_dump(include=include, exclude=exclude, round_trip=True) data = {**data, **(update or {})} copied = self.model_validate(data) `
```

Args:

include: Optional set or mapping specifying which fields to include in the copied model. exclude: Optional set or mapping specifying which fields to exclude in the copied model. update: Optional dictionary of field-value pairs to override field values

in the copied model. deep: If True, the values of fields that are Pydantic models will be deep-copied.

Returns:

A copy of the model with included, excluded and updated fields as specified.

model_computed_fields*: ClassVar[dict[str, ComputedFieldInfo]] = {}*

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

model_config*: ClassVar[ConfigDict] = {'allow_inf_nan': True, 'from_attributes': True}*

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

classmethod model_construct(*_fields_set: set[str] | None = None*, ***values: Any*) → Model

Creates a new instance of the *Model* class with validated data.

Creates a new model setting *__dict__* and *__pydantic_fields_set__* from trusted or pre-validated data. Default values are respected, but no other validation is performed. Behaves as if *Config.extra = 'allow'* was set since it adds all passed values

Args:

_fields_set: The set of field names accepted for the Model instance. values: Trusted or pre-validated data dictionary.

Returns:

A new instance of the *Model* class with validated data.

model_copy(*, *update: dict[str, Any] | None = None*, *deep: bool = False*) → Model

Usage docs: https://docs.pydantic.dev/2.6/concepts/serialization/#model_copy

Returns a copy of the model.

Args:

update: Values to change/add in the new model. Note: the data is not validated
before creating the new model. You should trust this data.

deep: Set to *True* to make a deep copy of the model.

Returns:

New model instance.

model_dump(*, *mode: Literal['json', 'python'] | str = 'python'*, *include: IncEx = None*, *exclude: IncEx = None*, *by_alias: bool = False*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*, *round_trip: bool = False*, *warnings: bool = True*) → dict[str, Any]

Usage docs: https://docs.pydantic.dev/2.6/concepts/serialization/#modelmodel_dump

Generate a dictionary representation of the model, optionally specifying which fields to include or exclude.

Args:

mode: The mode in which *to_python* should run.
If mode is 'json', the output will only contain JSON serializable types. If mode is 'python', the output may contain non-JSON-serializable Python objects.

include: A list of fields to include in the output. exclude: A list of fields to exclude from the output. by_alias: Whether to use the field's alias in the dictionary key if defined. exclude_unset: Whether to exclude fields that have not been explicitly set. exclude_defaults: Whether to exclude fields that are set to their default value. exclude_none: Whether to exclude fields that have a value of *None*. round_trip: If True, dumped values should be valid as input for non-idempotent types such as Json[T].

warnings: Whether to log warnings when invalid fields are encountered.

Returns:

A dictionary representation of the model.

model_dump_json(*, *indent: int | None = None*, *include: IncEx = None*, *exclude: IncEx = None*, *by_alias: bool = False*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*, *round_trip: bool = False*, *warnings: bool = True*) → str

Usage docs: https://docs.pydantic.dev/2.6/concepts/serialization/#modelmodel_dump_json

Generates a JSON representation of the model using Pydantic's *to_json* method.

Args:

indent: Indentation to use in the JSON output. If None is passed, the output will be compact. include: Field(s) to include in the JSON output. exclude: Field(s) to exclude from the JSON output. by_alias: Whether to serialize using field aliases. exclude_unset: Whether to exclude fields that have not been explicitly set. exclude_defaults: Whether to exclude fields that are set to their default value. exclude_none: Whether to exclude fields that have a value of *None*. round_trip: If True, dumped values should be valid as input for non-idempotent types such as Json[T]. warnings: Whether to log warnings when invalid fields are encountered.

Returns:

A JSON string representation of the model.

property model_extra*: dict[str, Any] | None*

Get extra fields set during validation.

Returns:

A dictionary of extra fields, or *None* if *config.extra* is not set to *"allow"*.

model_fields*: ClassVar[dict[str, FieldInfo]] = {'tag_entity_ids': FieldInfo(annotation=Dict[str, Union[List[str], NoneType]], required=True)}*

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

property model_fields_set*: set[str]*

Returns the set of fields that have been explicitly set on this model instance.

Returns:

A set of strings representing the fields that have been set, i.e. that were not filled from defaults.

classmethod model_json_schema(*by_alias: bool = True*, *ref_template: str = '#/$defs/{model}'*, *schema_generator: type[pydantic.json_schema.GenerateJsonSchema] = <class 'pydantic.json_schema.GenerateJsonSchema'>*, *mode: ~typing.Literal['validation', 'serialization'] = 'validation'*) → dict[str, Any]

Generates a JSON schema for a model class.

Args:

by_alias: Whether to use attribute aliases or not. ref_template: The reference template. schema_generator: To override the logic used to generate the JSON schema, as a subclass of

GenerateJsonSchema with your desired modifications

mode: The mode in which to generate the schema.

Returns:

The JSON schema for the given model class.

classmethod model_parametrized_name(*params: tuple[type[Any], ...]*) → str

Compute the class name for parametrizations of generic classes.

This method can be overridden to achieve a custom naming scheme for generic BaseModels.

Args:

params: Tuple of types of the class. Given a generic class *Model* with 2 type variables and a concrete model *Model[str, int]*, the value *(str, int)* would be passed to *params*.

Returns:

String representing the new class where *params* are passed to *cls* as type variables.

Raises:

TypeError: Raised when trying to generate concrete names for non-generic models.

model_post_init(*_BaseModel__context: Any*) → None

Override this method to perform additional initialization after *__init__* and *model_construct*. This is useful if you want to do some validation that requires the entire model to be initialized.

classmethod model_rebuild(*, *force: bool = False*, *raise_errors: bool = True*, *_parent_namespace_depth: int = 2*, *_types_namespace: Optional[dict[str, Any]] = None*) → bool | None

Try to rebuild the pydantic-core schema for the model.

This may be necessary when one of the annotations is a ForwardRef which could not be resolved during the initial attempt to build the schema, and automatic rebuilding fails.

Args:

force: Whether to force the rebuilding of the model schema, defaults to *False*. raise_errors: Whether to raise errors, defaults to *True*. _parent_namespace_depth: The depth level of the parent namespace, defaults to 2. _types_namespace: The types namespace, defaults to *None*.

Returns:
Returns *None* if the schema is already "complete" and rebuilding was not required. If rebuilding _was_ required, returns *True* if rebuilding was successful, otherwise *False*.

classmethod model_validate(*obj: Any*, *, *strict: bool | None = None*, *from_attributes: bool | None = None*, *context: dict[str, Any] | None = None*) → Model

Validate a pydantic model instance.

Args:
obj: The object to validate. strict: Whether to enforce types strictly. from_attributes: Whether to extract data from object attributes. context: Additional context to pass to the validator.

Raises:
ValidationError: If the object could not be validated.

Returns:
The validated model instance.

classmethod model_validate_json(*json_data: str | bytes | bytearray*, *, *strict: bool | None = None*, *context: dict[str, Any] | None = None*) → Model

Usage docs: https://docs.pydantic.dev/2.6/concepts/json/#json-parsing

Validate the given JSON data against the Pydantic model.

Args:

json_data: The JSON data to validate. strict: Whether to enforce types strictly. context: Extra variables to pass to the validator.

Returns:
The validated Pydantic model.

Raises:
ValueError: If *json_data* is not a JSON string.

classmethod model_validate_strings(*obj: Any*, *, *strict: bool | None = None*, *context: dict[str, Any] | None = None*) → Model

Validate the given object contains string data against the Pydantic model.

Args:
obj: The object contains string data to validate. strict: Whether to enforce types strictly. context: Extra variables to pass to the validator.

Returns:
The validated Pydantic model.

BatchSetStockTagsModel

class zvt.tag.tag_models.BatchSetStockTagsModel(*, *entity_ids: List[str]*, *tag: str*, *tag_reason: Optional[str] = None*, *tag_type: TagType*)

Bases: `CustomModel`

__init__(**data: Any*) → None

Create a new model by parsing and validating input data from keyword arguments.

Raises [*ValidationError*][pydantic_core.ValidationError] if the input data cannot be validated to form a valid model.

self is explicitly positional-only to allow *self* as a field name.

copy(*, *include: AbstractSetIntStr | MappingIntStrAny | None = None*, *exclude: AbstractSetIntStr | MappingIntStrAny | None = None*, *update: Dict[str, Any] | None = None*, *deep: bool = False*) → Model

Returns a copy of the model.

!!! warning "Deprecated"

This method is now deprecated; use *model_copy* instead.

If you need *include* or *exclude*, use:

```
`py data = self.model_dump(include=include, exclude=exclude, round_trip=True) data = {**data, **(update or {})} copied = self.model_validate(data) `
```

Args:

include: Optional set or mapping specifying which fields to include in the copied model. exclude: Optional set or mapping specifying which fields to exclude in the copied model. update: Optional dictionary of field-value pairs to override field values

in the copied model. deep: If True, the values of fields that are Pydantic models will be deep-copied.

Returns:

A copy of the model with included, excluded and updated fields as specified.

model_computed_fields*: ClassVar[dict[str, ComputedFieldInfo]] = {}*

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

model_config*: ClassVar[ConfigDict] = {'allow_inf_nan': True, 'from_attributes': True}*

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

classmethod model_construct(*_fields_set: set[str] | None = None*, ***values: Any*) → Model

Creates a new instance of the *Model* class with validated data.

Creates a new model setting *__dict__* and *__pydantic_fields_set__* from trusted or pre-validated data. Default values are respected, but no other validation is performed. Behaves as if *Config.extra = 'allow'* was set since it adds all passed values

Args:

_fields_set: The set of field names accepted for the Model instance. values: Trusted or pre-validated data dictionary.

Returns:

A new instance of the *Model* class with validated data.

model_copy(*, *update: dict[str, Any] | None = None*, *deep: bool = False*) → Model

Usage docs: https://docs.pydantic.dev/2.6/concepts/serialization/#model_copy

Returns a copy of the model.

Args:

update: Values to change/add in the new model. Note: the data is not validated
before creating the new model. You should trust this data.

deep: Set to *True* to make a deep copy of the model.

Returns:

New model instance.

model_dump(*, *mode: Literal['json', 'python'] | str = 'python'*, *include: IncEx = None*, *exclude: IncEx = None*, *by_alias: bool = False*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*, *round_trip: bool = False*, *warnings: bool = True*) → dict[str, Any]

Usage docs: https://docs.pydantic.dev/2.6/concepts/serialization/#modelmodel_dump

Generate a dictionary representation of the model, optionally specifying which fields to include or exclude.

Args:

mode: The mode in which *to_python* should run.
If mode is 'json', the output will only contain JSON serializable types. If mode is 'python', the output may contain non-JSON-serializable Python objects.

include: A list of fields to include in the output. exclude: A list of fields to exclude from the output. by_alias: Whether to use the field's alias in the dictionary key if defined. exclude_unset: Whether to exclude fields that have not been explicitly set. exclude_defaults: Whether to exclude fields that are set to their default value. exclude_none: Whether to exclude fields that have a value of *None*. round_trip: If True, dumped values should be valid as input for non-idempotent types such as Json[T].

warnings: Whether to log warnings when invalid fields are encountered.

Returns:

A dictionary representation of the model.

model_dump_json(*, *indent: int | None = None*, *include: IncEx = None*, *exclude: IncEx = None*, *by_alias: bool = False*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*, *round_trip: bool = False*, *warnings: bool = True*) → str

Usage docs: https://docs.pydantic.dev/2.6/concepts/serialization/#modelmodel_dump_json

Generates a JSON representation of the model using Pydantic's *to_json* method.

Args:

indent: Indentation to use in the JSON output. If None is passed, the output will be compact. include: Field(s) to include in the JSON output. exclude: Field(s) to exclude from the JSON output. by_alias: Whether to serialize using field aliases. exclude_unset: Whether to exclude fields that have not been explicitly set. exclude_defaults: Whether to exclude fields that are set to their default value. exclude_none: Whether to exclude fields that have a value of *None*. round_trip: If True, dumped values should be valid as input for non-idempotent types such as Json[T]. warnings: Whether to log warnings when invalid fields are encountered.

Returns:

A JSON string representation of the model.

property model_extra*: dict[str, Any] | None*

Get extra fields set during validation.

Returns:

A dictionary of extra fields, or *None* if *config.extra* is not set to *"allow"*.

model_fields*: ClassVar[dict[str, FieldInfo]] = {'entity_ids': FieldInfo(annotation=List[str], required=True), 'tag': FieldInfo(annotation=str, required=True), 'tag_reason': FieldInfo(annotation=Union[str, NoneType], required=False), 'tag_type': FieldInfo(annotation=TagType, required=True)}*

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

property model_fields_set*: set[str]*

Returns the set of fields that have been explicitly set on this model instance.

Returns:

A set of strings representing the fields that have been set, i.e. that were not filled from defaults.

classmethod model_json_schema(*by_alias: bool = True*, *ref_template: str = '#/$defs/{model}'*, *schema_generator: type[pydantic.json_schema.GenerateJsonSchema] = <class 'pydantic.json_schema.GenerateJsonSchema'>*, *mode: ~typing.Literal['validation', 'serialization'] = 'validation'*) → dict[str, Any]

Generates a JSON schema for a model class.

Args:

by_alias: Whether to use attribute aliases or not. ref_template: The reference template. schema_generator: To override the logic used to generate the JSON schema, as a subclass of

GenerateJsonSchema with your desired modifications

mode: The mode in which to generate the schema.

Returns:

The JSON schema for the given model class.

classmethod model_parametrized_name(*params: tuple[type[Any], ...]*) → str

Compute the class name for parametrizations of generic classes.

This method can be overridden to achieve a custom naming scheme for generic BaseModels.

Args:

params: Tuple of types of the class. Given a generic class *Model* with 2 type variables and a concrete model *Model[str, int]*, the value *(str, int)* would be passed to *params*.

Returns:

String representing the new class where *params* are passed to *cls* as type variables.

Raises:

TypeError: Raised when trying to generate concrete names for non-generic models.

model_post_init(*_BaseModel__context: Any*) → None

Override this method to perform additional initialization after *__init__* and *model_construct*. This is useful if you want to do some validation that requires the entire model to be initialized.

classmethod model_rebuild(*, *force: bool = False*, *raise_errors: bool = True*, *_parent_namespace_depth: int = 2*, *_types_namespace: Optional[dict[str, Any]] = None*) → bool | None

Try to rebuild the pydantic-core schema for the model.

This may be necessary when one of the annotations is a ForwardRef which could not be resolved during the initial attempt to build the schema, and automatic rebuilding fails.

Args:

force: Whether to force the rebuilding of the model schema, defaults to *False*. raise_errors: Whether to raise errors, defaults to *True*. _parent_namespace_depth: The depth level of the parent namespace, defaults to 2. _types_namespace: The types namespace, defaults to *None*.

Returns:

Returns *None* if the schema is already "complete" and rebuilding was not required. If rebuilding _was_ required, returns *True* if rebuilding was successful, otherwise *False*.

classmethod model_validate(*obj: Any*, *, *strict: bool | None = None*, *from_attributes: bool | None = None*, *context: dict[str, Any] | None = None*) → Model

Validate a pydantic model instance.

Args:

obj: The object to validate. strict: Whether to enforce types strictly. from_attributes: Whether to extract data from object attributes. context: Additional context to pass to the validator.

Raises:

ValidationError: If the object could not be validated.

Returns:

The validated model instance.

classmethod model_validate_json(*json_data: str | bytes | bytearray*, *, *strict: bool | None = None*, *context: dict[str, Any] | None = None*) → Model

Usage docs: https://docs.pydantic.dev/2.6/concepts/json/#json-parsing

Validate the given JSON data against the Pydantic model.

Args:

json_data: The JSON data to validate. strict: Whether to enforce types strictly. context: Extra variables to pass to the validator.

Returns:
The validated Pydantic model.

Raises:
ValueError: If *json_data* is not a JSON string.

classmethod model_validate_strings(*obj: Any*, *, *strict: bool | None = None*, *context: dict[str, Any] | None = None*) → Model

Validate the given object contains string data against the Pydantic model.

Args:
obj: The object contains string data to validate. strict: Whether to enforce types strictly. context: Extra variables to pass to the validator.

Returns:
The validated Pydantic model.

CreateStockPoolInfoModel

class zvt.tag.tag_models.CreateStockPoolInfoModel(*, *stock_pool_type: StockPoolType*, *stock_pool_name: str*)

Bases: `CustomModel`

__init__(**data: Any*) → None

Create a new model by parsing and validating input data from keyword arguments.

Raises [*ValidationError*][pydantic_core.ValidationError] if the input data cannot be validated to form a valid model.

self is explicitly positional-only to allow *self* as a field name.

copy(*, *include: AbstractSetIntStr | MappingIntStrAny | None = None*, *exclude: AbstractSetIntStr | MappingIntStrAny | None = None*, *update: Dict[str, Any] | None = None*, *deep: bool = False*) → Model

Returns a copy of the model.

!!! warning "Deprecated"

This method is now deprecated; use *model_copy* instead.

If you need *include* or *exclude*, use:

```
`py data = self.model_dump(include=include, exclude=exclude, round_trip=True) data = {**data, **(update or {})} copied = self.model_validate(data) `
```

Args:

include: Optional set or mapping specifying which fields to include in the copied model. exclude: Optional set or mapping specifying which fields to exclude in the copied model. update: Optional dictionary of field-value pairs to override field values

in the copied model. deep: If True, the values of fields that are Pydantic models will be deep-copied.

Returns:

A copy of the model with included, excluded and updated fields as specified.

model_computed_fields*: ClassVar[dict[str, ComputedFieldInfo]] = {}*

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

model_config*: ClassVar[ConfigDict] = {'allow_inf_nan': True, 'from_attributes': True}*

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

classmethod model_construct(*_fields_set: set[str] | None = None*, ***values: Any*) → Model

Creates a new instance of the *Model* class with validated data.

Creates a new model setting *__dict__* and *__pydantic_fields_set__* from trusted or pre-validated data. Default values are respected, but no other validation is performed. Behaves as if *Config.extra = 'allow'* was set since it adds all passed values

Args:

_fields_set: The set of field names accepted for the Model instance. values: Trusted or pre-validated data dictionary.

Returns:

A new instance of the *Model* class with validated data.

model_copy(*, *update: dict[str, Any] | None = None*, *deep: bool = False*) → Model

Usage docs: https://docs.pydantic.dev/2.6/concepts/serialization/#model_copy

Returns a copy of the model.

Args:

update: Values to change/add in the new model. Note: the data is not validated
before creating the new model. You should trust this data.

deep: Set to *True* to make a deep copy of the model.

Returns:

New model instance.

model_dump(*, *mode: Literal['json', 'python'] | str = 'python'*, *include: IncEx = None*, *exclude: IncEx = None*, *by_alias: bool = False*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*, *round_trip: bool = False*, *warnings: bool = True*) → dict[str, Any]

Usage docs: https://docs.pydantic.dev/2.6/concepts/serialization/#modelmodel_dump

Generate a dictionary representation of the model, optionally specifying which fields to include or exclude.

Args:

mode: The mode in which *to_python* should run.
If mode is 'json', the output will only contain JSON serializable types. If mode is 'python', the output may contain non-JSON-serializable Python objects.

include: A list of fields to include in the output. exclude: A list of fields to exclude from the output. by_alias: Whether to use the field's alias in the dictionary key if defined. exclude_unset: Whether to exclude fields that have not been explicitly set. exclude_defaults: Whether to exclude fields that are set to their default value. exclude_none: Whether to exclude fields that have a value of *None*. round_trip: If True, dumped values should be valid as input for non-idempotent types such as Json[T].

warnings: Whether to log warnings when invalid fields are encountered.

Returns:

A dictionary representation of the model.

model_dump_json(*, *indent: int | None = None*, *include: IncEx = None*, *exclude: IncEx = None*, *by_alias: bool = False*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*, *round_trip: bool = False*, *warnings: bool = True*) → str

Usage docs: https://docs.pydantic.dev/2.6/concepts/serialization/#modelmodel_dump_json

Generates a JSON representation of the model using Pydantic's *to_json* method.

Args:

indent: Indentation to use in the JSON output. If None is passed, the output will be compact. include: Field(s) to include in the JSON output. exclude: Field(s) to exclude from the JSON output. by_alias: Whether to serialize using field aliases. exclude_unset: Whether to exclude fields that have not been explicitly set. exclude_defaults: Whether to exclude fields that are set to their default value. exclude_none: Whether to exclude fields that have a value of *None*. round_trip: If True, dumped values should be valid as input for non-idempotent types such as Json[T]. warnings: Whether to log warnings when invalid fields are encountered.

Returns:

A JSON string representation of the model.

property model_extra*: dict[str, Any] | None*

Get extra fields set during validation.

Returns:

A dictionary of extra fields, or *None* if *config.extra* is not set to *"allow"*.

model_fields*: ClassVar[dict[str, FieldInfo]] = {'stock_pool_name': FieldInfo(annotation=str, required=True), 'stock_pool_type': FieldInfo(annotation=StockPoolType, required=True)}*

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

property model_fields_set*: set[str]*

Returns the set of fields that have been explicitly set on this model instance.

Returns:

A set of strings representing the fields that have been set, i.e. that were not filled from defaults.

classmethod model_json_schema(*by_alias: bool = True*, *ref_template: str = '#/$defs/{model}'*, *schema_generator: type[pydantic.json_schema.GenerateJsonSchema] = <class 'pydantic.json_schema.GenerateJsonSchema'>*, *mode: ~typing.Literal['validation', 'serialization'] = 'validation'*) → dict[str, Any]

Generates a JSON schema for a model class.

Args:

by_alias: Whether to use attribute aliases or not. ref_template: The reference template. schema_generator: To override the logic used to generate the JSON schema, as a subclass of

GenerateJsonSchema with your desired modifications

mode: The mode in which to generate the schema.

Returns:

The JSON schema for the given model class.

classmethod model_parametrized_name(*params: tuple[type[Any], ...]*) → str

Compute the class name for parametrizations of generic classes.

This method can be overridden to achieve a custom naming scheme for generic BaseModels.

Args:

params: Tuple of types of the class. Given a generic class *Model* with 2 type variables and a concrete model *Model[str, int]*, the value *(str, int)* would be passed to *params*.

Returns:

String representing the new class where *params* are passed to *cls* as type variables.

Raises:

TypeError: Raised when trying to generate concrete names for non-generic models.

model_post_init(*_BaseModel__context: Any*) → None

Override this method to perform additional initialization after *__init__* and *model_construct*. This is useful if you want to do some validation that requires the entire model to be initialized.

classmethod model_rebuild(*, *force: bool = False*, *raise_errors: bool = True*, *_parent_namespace_depth: int = 2*, *_types_namespace: Optional[dict[str, Any]] = None*) → bool | None

Try to rebuild the pydantic-core schema for the model.

This may be necessary when one of the annotations is a ForwardRef which could not be resolved during the initial attempt to build the schema, and automatic rebuilding fails.

Args:

force: Whether to force the rebuilding of the model schema, defaults to *False*. raise_errors: Whether to raise errors, defaults to *True*. _parent_namespace_depth: The depth level of the parent namespace, defaults to 2. _types_namespace: The types namespace, defaults to *None*.

Returns:
Returns *None* if the schema is already "complete" and rebuilding was not required. If rebuilding _was_ required, returns *True* if rebuilding was successful, otherwise *False*.

classmethod model_validate(*obj: Any*, *, *strict: bool | None = None*, *from_attributes: bool | None = None*, *context: dict[str, Any] | None = None*) → Model

Validate a pydantic model instance.

Args:
obj: The object to validate. strict: Whether to enforce types strictly. from_attributes: Whether to extract data from object attributes. context: Additional context to pass to the validator.

Raises:
ValidationError: If the object could not be validated.

Returns:
The validated model instance.

classmethod model_validate_json(*json_data: str | bytes | bytearray*, *, *strict: bool | None = None*, *context: dict[str, Any] | None = None*) → Model

Usage docs: https://docs.pydantic.dev/2.6/concepts/json/#json-parsing

Validate the given JSON data against the Pydantic model.

Args:

json_data: The JSON data to validate. strict: Whether to enforce types strictly. context: Extra variables to pass to the validator.

Returns:
The validated Pydantic model.

Raises:
ValueError: If *json_data* is not a JSON string.

classmethod model_validate_strings(*obj: Any*, *, *strict: bool | None = None*, *context: dict[str, Any] | None = None*) → Model

Validate the given object contains string data against the Pydantic model.

Args:
obj: The object contains string data to validate. strict: Whether to enforce types strictly. context: Extra variables to pass to the validator.

Returns:
The validated Pydantic model.

CreateStockPoolsModel

class zvt.tag.tag_models.CreateStockPoolsModel(*, *stock_pool_name: str*, *entity_ids: List[str]*)

Bases: `CustomModel`

__init__(**data: Any*) → None

Create a new model by parsing and validating input data from keyword arguments.

Raises [*ValidationError*][pydantic_core.ValidationError] if the input data cannot be validated to form a valid model.

self is explicitly positional-only to allow *self* as a field name.

copy(*, *include: AbstractSetIntStr | MappingIntStrAny | None = None*, *exclude: AbstractSetIntStr | MappingIntStrAny | None = None*, *update: Dict[str, Any] | None = None*, *deep: bool = False*) → Model

Returns a copy of the model.

!!! warning "Deprecated"

This method is now deprecated; use *model_copy* instead.

If you need *include* or *exclude*, use:

```
`py data = self.model_dump(include=include, exclude=exclude, round_trip=True) data = {**data, **(update or {})} copied = self.model_validate(data) `
```

Args:

include: Optional set or mapping specifying which fields to include in the copied model. exclude: Optional set or mapping specifying which fields to exclude in the copied model. update: Optional dictionary of field-value pairs to override field values

in the copied model. deep: If True, the values of fields that are Pydantic models will be deep-copied.

Returns:

A copy of the model with included, excluded and updated fields as specified.

model_computed_fields*: ClassVar[dict[str, ComputedFieldInfo]] = {}*

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

model_config*: ClassVar[ConfigDict] = {'allow_inf_nan': True, 'from_attributes': True}*

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

classmethod model_construct(*_fields_set: set[str] | None = None*, ***values: Any*) → Model

Creates a new instance of the *Model* class with validated data.

Creates a new model setting *__dict__* and *__pydantic_fields_set__* from trusted or pre-validated data. Default values are respected, but no other validation is performed. Behaves as if *Config.extra = 'allow'* was set since it adds all passed values

Args:

_fields_set: The set of field names accepted for the Model instance. values: Trusted or pre-validated data dictionary.

Returns:

A new instance of the *Model* class with validated data.

model_copy(*, *update: dict[str, Any] | None = None*, *deep: bool = False*) → Model

Usage docs: https://docs.pydantic.dev/2.6/concepts/serialization/#model_copy

Returns a copy of the model.

Args:

update: Values to change/add in the new model. Note: the data is not validated
before creating the new model. You should trust this data.

deep: Set to *True* to make a deep copy of the model.

Returns:

New model instance.

model_dump(*, *mode: Literal['json', 'python'] | str = 'python'*, *include: IncEx = None*, *exclude: IncEx = None*, *by_alias: bool = False*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*, *round_trip: bool = False*, *warnings: bool = True*) → dict[str, Any]

Usage docs: https://docs.pydantic.dev/2.6/concepts/serialization/#modelmodel_dump

Generate a dictionary representation of the model, optionally specifying which fields to include or exclude.

Args:

mode: The mode in which *to_python* should run.
If mode is 'json', the output will only contain JSON serializable types. If mode is 'python', the output may contain non-JSON-serializable Python objects.

include: A list of fields to include in the output. exclude: A list of fields to exclude from the output. by_alias: Whether to use the field's alias in the dictionary key if defined. exclude_unset: Whether to exclude fields that have not been explicitly set. exclude_defaults: Whether to exclude fields that are set to their default value. exclude_none: Whether to exclude fields that have a value of *None*. round_trip: If True, dumped values should be valid as input for non-idempotent types such as Json[T].

warnings: Whether to log warnings when invalid fields are encountered.

Returns:

A dictionary representation of the model.

model_dump_json(*, *indent: int | None = None*, *include: IncEx = None*, *exclude: IncEx = None*, *by_alias: bool = False*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*, *round_trip: bool = False*, *warnings: bool = True*) → str

Usage docs: https://docs.pydantic.dev/2.6/concepts/serialization/#modelmodel_dump_json

Generates a JSON representation of the model using Pydantic's *to_json* method.

Args:

indent: Indentation to use in the JSON output. If None is passed, the output will be compact. include: Field(s) to include in the JSON output. exclude: Field(s) to exclude from the JSON output. by_alias: Whether to serialize using field aliases. exclude_unset: Whether to exclude fields that have not been explicitly set. exclude_defaults: Whether to exclude fields that are set to their default value. exclude_none: Whether to exclude fields that have a value of *None*. round_trip: If True, dumped values should be valid as input for non-idempotent types such as Json[T]. warnings: Whether to log warnings when invalid fields are encountered.

Returns:

A JSON string representation of the model.

property model_extra*: dict[str, Any] | None*

Get extra fields set during validation.

Returns:

A dictionary of extra fields, or *None* if *config.extra* is not set to *"allow"*.

model_fields*: ClassVar[dict[str, FieldInfo]] = {'entity_ids': FieldInfo(annotation=List[str], required=True), 'stock_pool_name': FieldInfo(annotation=str, required=True)}*

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

property model_fields_set*: set[str]*

Returns the set of fields that have been explicitly set on this model instance.

Returns:

A set of strings representing the fields that have been set, i.e. that were not filled from defaults.

classmethod model_json_schema(*by_alias: bool = True*, *ref_template: str = '#/$defs/{model}'*, *schema_generator: type[pydantic.json_schema.GenerateJsonSchema] = <class 'pydantic.json_schema.GenerateJsonSchema'>*, *mode: ~typing.Literal['validation', 'serialization'] = 'validation'*) → dict[str, Any]

Generates a JSON schema for a model class.

Args:

by_alias: Whether to use attribute aliases or not. ref_template: The reference template. schema_generator: To override the logic used to generate the JSON schema, as a subclass of

GenerateJsonSchema with your desired modifications

mode: The mode in which to generate the schema.

Returns:

The JSON schema for the given model class.

classmethod model_parametrized_name(*params: tuple[type[Any], ...]*) → str

Compute the class name for parametrizations of generic classes.

This method can be overridden to achieve a custom naming scheme for generic BaseModels.

Args:

params: Tuple of types of the class. Given a generic class *Model* with 2 type variables and a concrete model *Model[str, int]*, the value *(str, int)* would be passed to *params*.

Returns:

String representing the new class where *params* are passed to *cls* as type variables.

Raises:

TypeError: Raised when trying to generate concrete names for non-generic models.

model_post_init(*_BaseModel__context: Any*) → None

Override this method to perform additional initialization after *__init__* and *model_construct*. This is useful if you want to do some validation that requires the entire model to be initialized.

classmethod model_rebuild(*, *force: bool = False*, *raise_errors: bool = True*, *_parent_namespace_depth: int = 2*, *_types_namespace: Optional[dict[str, Any]] = None*) → bool | None

Try to rebuild the pydantic-core schema for the model.

This may be necessary when one of the annotations is a ForwardRef which could not be resolved during the initial attempt to build the schema, and automatic rebuilding fails.

Args:

force: Whether to force the rebuilding of the model schema, defaults to *False*. raise_errors: Whether to raise errors, defaults to *True*. _parent_namespace_depth: The depth level of the parent namespace, defaults to 2. _types_namespace: The types namespace, defaults to *None*.

Returns:
Returns *None* if the schema is already "complete" and rebuilding was not required. If rebuilding _was_ required, returns *True* if rebuilding was successful, otherwise *False*.

classmethod model_validate(*obj: Any*, *, *strict: bool | None = None*, *from_attributes: bool | None = None*, *context: dict[str, Any] | None = None*) → Model

Validate a pydantic model instance.

Args:
obj: The object to validate. strict: Whether to enforce types strictly. from_attributes: Whether to extract data from object attributes. context: Additional context to pass to the validator.

Raises:
ValidationError: If the object could not be validated.

Returns:
The validated model instance.

classmethod model_validate_json(*json_data: str | bytes | bytearray*, *, *strict: bool | None = None*, *context: dict[str, Any] | None = None*) → Model

Usage docs: https://docs.pydantic.dev/2.6/concepts/json/#json-parsing

Validate the given JSON data against the Pydantic model.

Args:

json_data: The JSON data to validate. strict: Whether to enforce types strictly. context: Extra variables to pass to the validator.

Returns:
The validated Pydantic model.

Raises:
ValueError: If *json_data* is not a JSON string.

classmethod model_validate_strings(*obj: Any*, *, *strict: bool | None = None*, *context: dict[str, Any] | None = None*) → Model

Validate the given object contains string data against the Pydantic model.

Args:
obj: The object contains string data to validate. strict: Whether to enforce types strictly. context: Extra variables to pass to the validator.

Returns:
The validated Pydantic model.

CreateTagInfoModel

class zvt.tag.tag_models.CreateTagInfoModel(*, *tag: str*, *tag_reason: Optional[str] = None*)

Bases: `CustomModel`

__init__(**data: Any*) → None

Create a new model by parsing and validating input data from keyword arguments.

Raises [*ValidationError*][pydantic_core.ValidationError] if the input data cannot be validated to form a valid model.

self is explicitly positional-only to allow *self* as a field name.

copy(*, *include: AbstractSetIntStr | MappingIntStrAny | None = None*, *exclude: AbstractSetIntStr | MappingIntStrAny | None = None*, *update: Dict[str, Any] | None = None*, *deep: bool = False*) → Model

Returns a copy of the model.

!!! warning "Deprecated"

This method is now deprecated; use *model_copy* instead.

If you need *include* or *exclude*, use:

```
`py data = self.model_dump(include=include, exclude=exclude, round_trip=True) data = {**data, **(update or {})} copied = self.model_validate(data) `
```

Args:

include: Optional set or mapping specifying which fields to include in the copied model. exclude: Optional set or mapping specifying which fields to exclude in the copied model. update: Optional dictionary of field-value pairs to override field values

in the copied model. deep: If True, the values of fields that are Pydantic models will be deep-copied.

Returns:

A copy of the model with included, excluded and updated fields as specified.

model_computed_fields*: ClassVar[dict[str, ComputedFieldInfo]] = {}*

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

model_config*: ClassVar[ConfigDict] = {'allow_inf_nan': True, 'from_attributes': True}*

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

classmethod model_construct(*_fields_set: set[str] | None = None*, ***values: Any*) → Model

Creates a new instance of the *Model* class with validated data.

Creates a new model setting *__dict__* and *__pydantic_fields_set__* from trusted or pre-validated data. Default values are respected, but no other validation is performed. Behaves as if *Config.extra = 'allow'* was set since it adds all passed values

Args:

_fields_set: The set of field names accepted for the Model instance. values: Trusted or pre-validated data dictionary.

Returns:

A new instance of the *Model* class with validated data.

model_copy(*, *update: dict[str, Any] | None = None*, *deep: bool = False*) → Model

Usage docs: https://docs.pydantic.dev/2.6/concepts/serialization/#model_copy

Returns a copy of the model.

Args:

update: Values to change/add in the new model. Note: the data is not validated
before creating the new model. You should trust this data.

deep: Set to *True* to make a deep copy of the model.

Returns:

New model instance.

model_dump(*, *mode: Literal['json', 'python'] | str = 'python'*, *include: IncEx = None*, *exclude: IncEx = None*, *by_alias: bool = False*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*, *round_trip: bool = False*, *warnings: bool = True*) → dict[str, Any]

Usage docs: https://docs.pydantic.dev/2.6/concepts/serialization/#modelmodel_dump

Generate a dictionary representation of the model, optionally specifying which fields to include or exclude.

Args:

mode: The mode in which *to_python* should run.
If mode is 'json', the output will only contain JSON serializable types. If mode is 'python', the output may contain non-JSON-serializable Python objects.

include: A list of fields to include in the output. exclude: A list of fields to exclude from the output. by_alias: Whether to use the field's alias in the dictionary key if defined. exclude_unset: Whether to exclude fields that have not been explicitly set. exclude_defaults: Whether to exclude fields that are set to their default value. exclude_none: Whether to exclude fields that have a value of *None*. round_trip: If True, dumped values should be valid as input for non-idempotent types such as Json[T].

warnings: Whether to log warnings when invalid fields are encountered.

Returns:

A dictionary representation of the model.

model_dump_json(*, *indent: int | None = None*, *include: IncEx = None*, *exclude: IncEx = None*, *by_alias: bool = False*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*, *round_trip: bool = False*, *warnings: bool = True*) → str

Usage docs: https://docs.pydantic.dev/2.6/concepts/serialization/#modelmodel_dump_json

Generates a JSON representation of the model using Pydantic's *to_json* method.

Args:

indent: Indentation to use in the JSON output. If None is passed, the output will be compact. include: Field(s) to include in the JSON output. exclude: Field(s) to exclude from the JSON output. by_alias: Whether to serialize using field aliases. exclude_unset: Whether to exclude fields that have not been explicitly set. exclude_defaults: Whether to exclude fields that are set to their default value. exclude_none: Whether to exclude fields that have a value of *None*. round_trip: If True, dumped values should be valid as input for non-idempotent types such as Json[T]. warnings: Whether to log warnings when invalid fields are encountered.

Returns:

A JSON string representation of the model.

property model_extra*: dict[str, Any] | None*

Get extra fields set during validation.

Returns:

A dictionary of extra fields, or *None* if *config.extra* is not set to *"allow"*.

model_fields*: ClassVar[dict[str, FieldInfo]] = {'tag': FieldInfo(annotation=str, required=True), 'tag_reason': FieldInfo(annotation=Union[str, NoneType], required=False)}*

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

property model_fields_set*: set[str]*

Returns the set of fields that have been explicitly set on this model instance.

Returns:

A set of strings representing the fields that have been set, i.e. that were not filled from defaults.

classmethod model_json_schema(*by_alias: bool = True*, *ref_template: str = '#/$defs/{model}'*, *schema_generator: type[pydantic.json_schema.GenerateJsonSchema] = <class 'pydantic.json_schema.GenerateJsonSchema'>*, *mode: ~typing.Literal['validation', 'serialization'] = 'validation'*) → dict[str, Any]

Generates a JSON schema for a model class.

Args:

by_alias: Whether to use attribute aliases or not. ref_template: The reference template. schema_generator: To override the logic used to generate the JSON schema, as a subclass of

GenerateJsonSchema with your desired modifications

mode: The mode in which to generate the schema.

Returns:

The JSON schema for the given model class.

classmethod model_parametrized_name(*params: tuple[type[Any], ...]*) → str

Compute the class name for parametrizations of generic classes.

This method can be overridden to achieve a custom naming scheme for generic BaseModels.

Args:

params: Tuple of types of the class. Given a generic class *Model* with 2 type variables and a concrete model *Model[str, int]*, the value *(str, int)* would be passed to *params*.

Returns:

String representing the new class where *params* are passed to *cls* as type variables.

Raises:

TypeError: Raised when trying to generate concrete names for non-generic models.

model_post_init(*_BaseModel__context: Any*) → None

Override this method to perform additional initialization after *__init__* and *model_construct*. This is useful if you want to do some validation that requires the entire model to be initialized.

classmethod model_rebuild(*, *force: bool = False*, *raise_errors: bool = True*, *_parent_namespace_depth: int = 2*, *_types_namespace: Optional[dict[str, Any]] = None*) → bool | None

Try to rebuild the pydantic-core schema for the model.

This may be necessary when one of the annotations is a ForwardRef which could not be resolved during the initial attempt to build the schema, and automatic rebuilding fails.

Args:

force: Whether to force the rebuilding of the model schema, defaults to *False*. raise_errors: Whether to raise errors, defaults to *True*. _parent_namespace_depth: The depth level of the parent namespace, defaults to 2. _types_namespace: The types namespace, defaults to *None*.

Returns:
Returns *None* if the schema is already "complete" and rebuilding was not required. If rebuilding _was_ required, returns *True* if rebuilding was successful, otherwise *False*.

classmethod model_validate(*obj: Any*, *, *strict: bool | None = None*, *from_attributes: bool | None = None*, *context: dict[str, Any] | None = None*) → Model

Validate a pydantic model instance.

Args:
obj: The object to validate. strict: Whether to enforce types strictly. from_attributes: Whether to extract data from object attributes. context: Additional context to pass to the validator.

Raises:
ValidationError: If the object could not be validated.

Returns:
The validated model instance.

classmethod model_validate_json(*json_data: str | bytes | bytearray*, *, *strict: bool | None = None*, *context: dict[str, Any] | None = None*) → Model

Usage docs: https://docs.pydantic.dev/2.6/concepts/json/#json-parsing

Validate the given JSON data against the Pydantic model.

Args:

json_data: The JSON data to validate. strict: Whether to enforce types strictly. context: Extra variables to pass to the validator.

Returns:
The validated Pydantic model.

Raises:
ValueError: If *json_data* is not a JSON string.

classmethod model_validate_strings(*obj: Any*, *, *strict: bool | None = None*, *context: dict[str, Any] | None = None*) → Model

Validate the given object contains string data against the Pydantic model.

Args:
obj: The object contains string data to validate. strict: Whether to enforce types strictly. context: Extra variables to pass to the validator.

Returns:
The validated Pydantic model.

QuerySimpleStockTagsModel

class zvt.tag.tag_models.QuerySimpleStockTagsModel(*, *entity_ids: List[str]*)

Bases: `CustomModel`

__init__(**data: Any*) → None

Create a new model by parsing and validating input data from keyword arguments.

Raises [*ValidationError*][pydantic_core.ValidationError] if the input data cannot be validated to form a valid model.

self is explicitly positional-only to allow *self* as a field name.

copy(*, *include: AbstractSetIntStr | MappingIntStrAny | None = None*, *exclude: AbstractSetIntStr | MappingIntStrAny | None = None*, *update: Dict[str, Any] | None = None*, *deep: bool = False*) → Model

Returns a copy of the model.

!!! warning "Deprecated"

This method is now deprecated; use *model_copy* instead.

If you need *include* or *exclude*, use:

```
`py data = self.model_dump(include=include, exclude=exclude, round_trip=True) data = {**data, **(update or {})} copied = self.model_validate(data) `
```

Args:

include: Optional set or mapping specifying which fields to include in the copied model. exclude: Optional set or mapping specifying which fields to exclude in the copied model. update: Optional dictionary of field-value pairs to override field values

in the copied model. deep: If True, the values of fields that are Pydantic models will be deep-copied.

Returns:

A copy of the model with included, excluded and updated fields as specified.

model_computed_fields*: ClassVar[dict[str, ComputedFieldInfo]] = {}*

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

model_config*: ClassVar[ConfigDict] = {'allow_inf_nan': True, 'from_attributes': True}*

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

classmethod model_construct(*_fields_set: set[str] | None = None*, ***values: Any*) → Model

Creates a new instance of the *Model* class with validated data.

Creates a new model setting *__dict__* and *__pydantic_fields_set__* from trusted or pre-validated data. Default values are respected, but no other validation is performed. Behaves as if *Config.extra = 'allow'* was set since it adds all passed values

Args:

_fields_set: The set of field names accepted for the Model instance. values: Trusted or pre-validated data dictionary.

Returns:

A new instance of the *Model* class with validated data.

model_copy(*, *update: dict[str, Any] | None = None*, *deep: bool = False*) → Model

Usage docs: https://docs.pydantic.dev/2.6/concepts/serialization/#model_copy

Returns a copy of the model.

Args:

update: Values to change/add in the new model. Note: the data is not validated
before creating the new model. You should trust this data.

deep: Set to *True* to make a deep copy of the model.

Returns:

New model instance.

model_dump(*, *mode: Literal['json', 'python'] | str = 'python'*, *include: IncEx = None*, *exclude: IncEx = None*, *by_alias: bool = False*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*, *round_trip: bool = False*, *warnings: bool = True*) → dict[str, Any]

Usage docs: https://docs.pydantic.dev/2.6/concepts/serialization/#modelmodel_dump

Generate a dictionary representation of the model, optionally specifying which fields to include or exclude.

Args:

mode: The mode in which *to_python* should run.
If mode is 'json', the output will only contain JSON serializable types. If mode is 'python', the output may contain non-JSON-serializable Python objects.

include: A list of fields to include in the output. exclude: A list of fields to exclude from the output. by_alias: Whether to use the field's alias in the dictionary key if defined. exclude_unset: Whether to exclude fields that have not been explicitly set. exclude_defaults: Whether to exclude fields that are set to their default value. exclude_none: Whether to exclude fields that have a value of *None*. round_trip: If True, dumped values should be valid as input for non-idempotent types such as Json[T].

warnings: Whether to log warnings when invalid fields are encountered.

Returns:

A dictionary representation of the model.

model_dump_json(*, *indent: int | None = None*, *include: IncEx = None*, *exclude: IncEx = None*, *by_alias: bool = False*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*, *round_trip: bool = False*, *warnings: bool = True*) → str

Usage docs: https://docs.pydantic.dev/2.6/concepts/serialization/#modelmodel_dump_json

Generates a JSON representation of the model using Pydantic's *to_json* method.

Args:

indent: Indentation to use in the JSON output. If None is passed, the output will be compact. include: Field(s) to include in the JSON output. exclude: Field(s) to exclude from the JSON output. by_alias: Whether to serialize using field aliases. exclude_unset: Whether to exclude fields that have not been explicitly set. exclude_defaults: Whether to exclude fields that are set to their default value. exclude_none: Whether to exclude fields that have a value of *None*. round_trip: If True, dumped values should be valid as input for non-idempotent types such as Json[T]. warnings: Whether to log warnings when invalid fields are encountered.

Returns:

A JSON string representation of the model.

property model_extra*: dict[str, Any] | None*

Get extra fields set during validation.

Returns:

A dictionary of extra fields, or *None* if *config.extra* is not set to *"allow"*.

model_fields*: ClassVar[dict[str, FieldInfo]] = {'entity_ids': FieldInfo(annotation=List[str], required=True)}*

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

property model_fields_set*: set[str]*

Returns the set of fields that have been explicitly set on this model instance.

Returns:

A set of strings representing the fields that have been set, i.e. that were not filled from defaults.

classmethod model_json_schema(*by_alias: bool = True*, *ref_template: str = '#/$defs/{model}'*, *schema_generator: type[pydantic.json_schema.GenerateJsonSchema] = <class 'pydantic.json_schema.GenerateJsonSchema'>*, *mode: ~typing.Literal['validation', 'serialization'] = 'validation'*) → dict[str, Any]

Generates a JSON schema for a model class.

Args:

by_alias: Whether to use attribute aliases or not. ref_template: The reference template. schema_generator: To override the logic used to generate the JSON schema, as a subclass of

GenerateJsonSchema with your desired modifications

mode: The mode in which to generate the schema.

Returns:

The JSON schema for the given model class.

classmethod model_parametrized_name(*params: tuple[type[Any], ...]*) → str

Compute the class name for parametrizations of generic classes.

This method can be overridden to achieve a custom naming scheme for generic BaseModels.

Args:

params: Tuple of types of the class. Given a generic class *Model* with 2 type variables and a concrete model *Model[str, int]*, the value *(str, int)* would be passed to *params*.

Returns:

String representing the new class where *params* are passed to *cls* as type variables.

Raises:

TypeError: Raised when trying to generate concrete names for non-generic models.

model_post_init(*_BaseModel__context: Any*) → None

Override this method to perform additional initialization after *__init__* and *model_construct*. This is useful if you want to do some validation that requires the entire model to be initialized.

classmethod model_rebuild(*, *force: bool = False*, *raise_errors: bool = True*, *_parent_namespace_depth: int = 2*, *_types_namespace: Optional[dict[str, Any]] = None*) → bool | None

Try to rebuild the pydantic-core schema for the model.

This may be necessary when one of the annotations is a ForwardRef which could not be resolved during the initial attempt to build the schema, and automatic rebuilding fails.

Args:

force: Whether to force the rebuilding of the model schema, defaults to *False*. raise_errors: Whether to raise errors, defaults

to *True*. _parent_namespace_depth: The depth level of the parent namespace, defaults to 2. _types_namespace: The types namespace, defaults to *None*.

Returns:

Returns *None* if the schema is already "complete" and rebuilding was not required. If rebuilding _was_ required, returns *True* if rebuilding was successful, otherwise *False*.

classmethod model_validate(*obj: Any*, *, *strict: bool | None = None*, *from_attributes: bool | None = None*, *context: dict[str, Any] | None = None*) → Model

Validate a pydantic model instance.

Args:

obj: The object to validate. strict: Whether to enforce types strictly. from_attributes: Whether to extract data from object attributes. context: Additional context to pass to the validator.

Raises:

ValidationError: If the object could not be validated.

Returns:

The validated model instance.

classmethod model_validate_json(*json_data: str | bytes | bytearray*, *, *strict: bool | None = None*, *context: dict[str, Any] | None = None*) → Model

Usage docs: https://docs.pydantic.dev/2.6/concepts/json/#json-parsing

Validate the given JSON data against the Pydantic model.

Args:

json_data: The JSON data to validate. strict: Whether to enforce types strictly. context: Extra variables to pass to the validator.

Returns:

The validated Pydantic model.

Raises:

ValueError: If *json_data* is not a JSON string.

classmethod model_validate_strings(*obj: Any*, *, *strict: bool | None = None*, *context: dict[str, Any] | None = None*) → Model

Validate the given object contains string data against the Pydantic model.

Args:

obj: The object contains string data to validate. strict: Whether to enforce types strictly. context: Extra variables to pass to the validator.

Returns:

The validated Pydantic model.

QueryStockTagStatsModel

class zvt.tag.tag_models.QueryStockTagStatsModel(*, *stock_pool_name: Optional[str] = None*, *entity_ids: Optional[List[str]] = None*)

Bases: `CustomModel`

__init__(**data: Any*) → None

Create a new model by parsing and validating input data from keyword arguments.

Raises [*ValidationError*][pydantic_core.ValidationError] if the input data cannot be validated to form a valid model.

self is explicitly positional-only to allow *self* as a field name.

copy(*, *include: AbstractSetIntStr | MappingIntStrAny | None = None*, *exclude: AbstractSetIntStr | MappingIntStrAny | None = None*, *update: Dict[str, Any] | None = None*, *deep: bool = False*) → Model

Returns a copy of the model.

!!! warning "Deprecated"

This method is now deprecated; use *model_copy* instead.

If you need *include* or *exclude*, use:

```
`py data = self.model_dump(include=include, exclude=exclude, round_trip=True) data = {**data, **(update or {})} copied = self.model_validate(data) `
```

Args:

include: Optional set or mapping specifying which fields to include in the copied model. exclude: Optional set or mapping specifying which fields to exclude in the copied model. update: Optional dictionary of field-value pairs to override field values

in the copied model. deep: If True, the values of fields that are Pydantic models will be deep-copied.

Returns:

A copy of the model with included, excluded and updated fields as specified.

model_computed_fields*: ClassVar[dict[str, ComputedFieldInfo]] = {}*

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

model_config*: ClassVar[ConfigDict] = {'allow_inf_nan': True, 'from_attributes': True}*

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

classmethod model_construct(*_fields_set: set[str] | None = None*, ***values: Any*) → Model

Creates a new instance of the *Model* class with validated data.

Creates a new model setting *__dict__* and *__pydantic_fields_set__* from trusted or pre-validated data. Default values are respected, but no other validation is performed. Behaves as if *Config.extra = 'allow'* was set since it adds all passed values

Args:

_fields_set: The set of field names accepted for the Model instance. values: Trusted or pre-validated data dictionary.

Returns:

A new instance of the *Model* class with validated data.

model_copy(*, *update: dict[str, Any] | None = None*, *deep: bool = False*) → Model

Usage docs: https://docs.pydantic.dev/2.6/concepts/serialization/#model_copy

Returns a copy of the model.

Args:

update: Values to change/add in the new model. Note: the data is not validated
before creating the new model. You should trust this data.

deep: Set to *True* to make a deep copy of the model.

Returns:

New model instance.

model_dump(*, *mode: Literal['json', 'python'] | str = 'python'*, *include: IncEx = None*, *exclude: IncEx = None*, *by_alias: bool = False*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*, *round_trip: bool = False*, *warnings: bool = True*) → dict[str, Any]

Usage docs: https://docs.pydantic.dev/2.6/concepts/serialization/#modelmodel_dump

Generate a dictionary representation of the model, optionally specifying which fields to include or exclude.

Args:

mode: The mode in which *to_python* should run.
If mode is 'json', the output will only contain JSON serializable types. If mode is 'python', the output may contain non-JSON-serializable Python objects.

include: A list of fields to include in the output. exclude: A list of fields to exclude from the output. by_alias: Whether to use the field's alias in the dictionary key if defined. exclude_unset: Whether to exclude fields that have not been explicitly set. exclude_defaults: Whether to exclude fields that are set to their default value. exclude_none: Whether to exclude fields that have a value of *None*. round_trip: If True, dumped values should be valid as input for non-idempotent types such as Json[T].

warnings: Whether to log warnings when invalid fields are encountered.

Returns:

A dictionary representation of the model.

model_dump_json(*, *indent: int | None = None*, *include: IncEx = None*, *exclude: IncEx = None*, *by_alias: bool = False*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*, *round_trip: bool = False*, *warnings: bool = True*) → str

Usage docs: https://docs.pydantic.dev/2.6/concepts/serialization/#modelmodel_dump_json

Generates a JSON representation of the model using Pydantic's *to_json* method.

Args:

indent: Indentation to use in the JSON output. If None is passed, the output will be compact. include: Field(s) to include in the JSON output. exclude: Field(s) to exclude from the JSON output. by_alias: Whether to serialize using field aliases. exclude_unset: Whether to exclude fields that have not been explicitly set. exclude_defaults: Whether to exclude fields that are set to their default value. exclude_none: Whether to exclude fields that have a value of *None*. round_trip: If True, dumped values should be valid as input for non-idempotent types such as Json[T]. warnings: Whether to log warnings when invalid fields are encountered.

Returns:

A JSON string representation of the model.

property model_extra*: dict[str, Any] | None*

Get extra fields set during validation.

Returns:

A dictionary of extra fields, or *None* if *config.extra* is not set to *"allow"*.

model_fields*: ClassVar[dict[str, FieldInfo]] = {'entity_ids': FieldInfo(annotation=Union[List[str], NoneType], required=False), 'stock_pool_name': FieldInfo(annotation=Union[str, NoneType], required=False)}*

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

property model_fields_set*: set[str]*

Returns the set of fields that have been explicitly set on this model instance.

Returns:

A set of strings representing the fields that have been set, i.e. that were not filled from defaults.

classmethod model_json_schema(*by_alias: bool = True*, *ref_template: str = '#/$defs/{model}'*, *schema_generator: type[pydantic.json_schema.GenerateJsonSchema] = <class 'pydantic.json_schema.GenerateJsonSchema'>*, *mode: ~typing.Literal['validation', 'serialization'] = 'validation'*) → dict[str, Any]

Generates a JSON schema for a model class.

Args:

by_alias: Whether to use attribute aliases or not. ref_template: The reference template. schema_generator: To override the logic used to generate the JSON schema, as a subclass of

GenerateJsonSchema with your desired modifications

mode: The mode in which to generate the schema.

Returns:

The JSON schema for the given model class.

classmethod model_parametrized_name(*params: tuple[type[Any], ...]*) → str

Compute the class name for parametrizations of generic classes.

This method can be overridden to achieve a custom naming scheme for generic BaseModels.

Args:

params: Tuple of types of the class. Given a generic class *Model* with 2 type variables and a concrete model *Model[str, int]*, the value *(str, int)* would be passed to *params*.

Returns:

String representing the new class where *params* are passed to *cls* as type variables.

Raises:

TypeError: Raised when trying to generate concrete names for non-generic models.

model_post_init(*_BaseModel__context: Any*) → None

Override this method to perform additional initialization after *__init__* and *model_construct*. This is useful if you want to do some validation that requires the entire model to be initialized.

classmethod model_rebuild(*, *force: bool = False*, *raise_errors: bool = True*, *_parent_namespace_depth: int = 2*, *_types_namespace: Optional[dict[str, Any]] = None*) → bool | None

Try to rebuild the pydantic-core schema for the model.

This may be necessary when one of the annotations is a ForwardRef which could not be resolved during the initial attempt to build the schema, and automatic rebuilding fails.

Args:

force: Whether to force the rebuilding of the model schema, defaults to *False*. raise_errors: Whether to raise errors, defaults to *True*. _parent_namespace_depth: The depth level of the parent namespace, defaults to 2. _types_namespace: The types namespace, defaults to *None*.

Returns:
Returns *None* if the schema is already "complete" and rebuilding was not required. If rebuilding _was_ required, returns *True* if rebuilding was successful, otherwise *False*.

classmethod model_validate(*obj: Any*, *, *strict: bool | None = None*, *from_attributes: bool | None = None*, *context: dict[str, Any] | None = None*) → Model

Validate a pydantic model instance.

Args:
obj: The object to validate. strict: Whether to enforce types strictly. from_attributes: Whether to extract data from object attributes. context: Additional context to pass to the validator.

Raises:
ValidationError: If the object could not be validated.

Returns:
The validated model instance.

classmethod model_validate_json(*json_data: str | bytes | bytearray*, *, *strict: bool | None = None*, *context: dict[str, Any] | None = None*) → Model

Usage docs: https://docs.pydantic.dev/2.6/concepts/json/#json-parsing

Validate the given JSON data against the Pydantic model.

Args:

json_data: The JSON data to validate. strict: Whether to enforce types strictly. context: Extra variables to pass to the validator.

Returns:
The validated Pydantic model.

Raises:
ValueError: If *json_data* is not a JSON string.

classmethod model_validate_strings(*obj: Any*, *, *strict: bool | None = None*, *context: dict[str, Any] | None = None*) → Model

Validate the given object contains string data against the Pydantic model.

Args:
obj: The object contains string data to validate. strict: Whether to enforce types strictly. context: Extra variables to pass to the validator.

Returns:
The validated Pydantic model.

QueryStockTagsModel

class zvt.tag.tag_models.QueryStockTagsModel(*, *entity_ids: List[str]*)

Bases: `CustomModel`

__init__(**data: Any*) → None

Create a new model by parsing and validating input data from keyword arguments.

Raises [*ValidationError*][pydantic_core.ValidationError] if the input data cannot be validated to form a valid model.

self is explicitly positional-only to allow *self* as a field name.

copy(*, *include: AbstractSetIntStr | MappingIntStrAny | None = None*, *exclude: AbstractSetIntStr | MappingIntStrAny | None = None*, *update: Dict[str, Any] | None = None*, *deep: bool = False*) → Model

Returns a copy of the model.

!!! warning "Deprecated"

This method is now deprecated; use *model_copy* instead.

If you need *include* or *exclude*, use:

```
`py data = self.model_dump(include=include, exclude=exclude, round_trip=True) data = {**data, **(update or {})} copied = self.model_validate(data) `
```

Args:

include: Optional set or mapping specifying which fields to include in the copied model. exclude: Optional set or mapping specifying which fields to exclude in the copied model. update: Optional dictionary of field-value pairs to override field values in the copied model. deep: If True, the values of fields that are Pydantic models will be deep-copied.

Returns:

A copy of the model with included, excluded and updated fields as specified.

model_computed_fields*: ClassVar[dict[str, ComputedFieldInfo]] = {}*

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

model_config*: ClassVar[ConfigDict] = {'allow_inf_nan': True, 'from_attributes': True}*

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

classmethod model_construct(*_fields_set: set[str] | None = None*, ***values: Any*) → Model

Creates a new instance of the *Model* class with validated data.

Creates a new model setting *__dict__* and *__pydantic_fields_set__* from trusted or pre-validated data. Default values are respected, but no other validation is performed. Behaves as if *Config.extra = 'allow'* was set since it adds all passed values

Args:

_fields_set: The set of field names accepted for the Model instance. values: Trusted or pre-validated data dictionary.

Returns:

A new instance of the *Model* class with validated data.

model_copy(*, *update: dict[str, Any] | None = None*, *deep: bool = False*) → Model

Usage docs: https://docs.pydantic.dev/2.6/concepts/serialization/#model_copy

Returns a copy of the model.

Args:

update: Values to change/add in the new model. Note: the data is not validated
before creating the new model. You should trust this data.

deep: Set to *True* to make a deep copy of the model.

Returns:
New model instance.

model_dump(*, *mode: Literal['json', 'python'] | str = 'python'*, *include: IncEx = None*, *exclude: IncEx = None*, *by_alias: bool = False*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*, *round_trip: bool = False*, *warnings: bool = True*) → dict[str, Any]

Usage docs: https://docs.pydantic.dev/2.6/concepts/serialization/#modelmodel_dump

Generate a dictionary representation of the model, optionally specifying which fields to include or exclude.

Args:
mode: The mode in which *to_python* should run.
If mode is 'json', the output will only contain JSON serializable types. If mode is 'python', the output may contain non-JSON-serializable Python objects.

include: A list of fields to include in the output. exclude: A list of fields to exclude from the output. by_alias: Whether to use the field's alias in the dictionary key if defined. exclude_unset: Whether to exclude fields that have not been explicitly set. exclude_defaults: Whether to exclude fields that are set to their default value. exclude_none: Whether to exclude fields that have a value of *None*. round_trip: If True, dumped values should be valid as input for non-idempotent types such as Json[T]. warnings: Whether to log warnings when invalid fields are encountered.

Returns:

A dictionary representation of the model.

model_dump_json(*, *indent: int | None = None*, *include: IncEx = None*, *exclude: IncEx = None*, *by_alias: bool = False*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*, *round_trip: bool = False*, *warnings: bool = True*) → str

Usage docs: https://docs.pydantic.dev/2.6/concepts/serialization/#modelmodel_dump_json

Generates a JSON representation of the model using Pydantic's *to_json* method.

Args:

indent: Indentation to use in the JSON output. If None is passed, the output will be compact. include: Field(s) to include in the JSON output. exclude: Field(s) to exclude from the JSON output. by_alias: Whether to serialize using field aliases. exclude_unset: Whether to exclude fields that have not been explicitly set. exclude_defaults: Whether to exclude fields that are set to their default value. exclude_none: Whether to exclude fields that have a value of *None*. round_trip: If True, dumped values should be valid as input for non-idempotent types such as Json[T]. warnings: Whether to log warnings when invalid fields are encountered.

Returns:

A JSON string representation of the model.

property model_extra*: dict[str, Any] | None*

Get extra fields set during validation.

Returns:

A dictionary of extra fields, or *None* if *config.extra* is not set to *"allow"*.

model_fields*: ClassVar[dict[str, FieldInfo]] = {'entity_ids': FieldInfo(annotation=List[str], required=True)}*

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

property model_fields_set*: set[str]*

Returns the set of fields that have been explicitly set on this model instance.

Returns:

A set of strings representing the fields that have been set, i.e. that were not filled from defaults.

classmethod model_json_schema(*by_alias: bool = True*, *ref_template: str = '#/$defs/{model}'*, *schema_generator: type[pydantic.json_schema.GenerateJsonSchema] = <class 'pydantic.json_schema.GenerateJsonSchema'>*, *mode: ~typing.Literal['validation', 'serialization'] = 'validation'*) → dict[str, Any]

Generates a JSON schema for a model class.

Args:

by_alias: Whether to use attribute aliases or not. ref_template: The reference template. schema_generator: To override the logic used to generate the JSON schema, as a subclass of

GenerateJsonSchema with your desired modifications

mode: The mode in which to generate the schema.

Returns:

The JSON schema for the given model class.

classmethod model_parametrized_name(*params: tuple[type[Any], ...]*) → str

Compute the class name for parametrizations of generic classes.

This method can be overridden to achieve a custom naming scheme for generic BaseModels.

Args:

params: Tuple of types of the class. Given a generic class *Model* with 2 type variables and a concrete model *Model[str, int]*, the value *(str, int)* would be passed to *params*.

Returns:

String representing the new class where *params* are passed to *cls* as type variables.

Raises:

TypeError: Raised when trying to generate concrete names for non-generic models.

model_post_init(*_BaseModel__context: Any*) → None

Override this method to perform additional initialization after *__init__* and *model_construct*. This is useful if you want to do some validation that requires the entire model to be initialized.

classmethod model_rebuild(*, *force: bool = False*, *raise_errors: bool = True*, *_parent_namespace_depth: int = 2*, *_types_namespace: Optional[dict[str, Any]] = None*) → bool | None

Try to rebuild the pydantic-core schema for the model.

This may be necessary when one of the annotations is a ForwardRef which could not be resolved during the initial attempt to build the schema, and automatic rebuilding fails.

Args:

force: Whether to force the rebuilding of the model schema, defaults to *False*. raise_errors: Whether to raise errors, defaults to *True*. _parent_namespace_depth: The depth level of the parent namespace, defaults to 2. _types_namespace: The types namespace, defaults to *None*.

Returns:

Returns *None* if the schema is already "complete" and rebuilding was not required. If rebuilding _was_ required, returns *True* if rebuilding was successful, otherwise *False*.

classmethod model_validate(*obj: Any*, *, *strict: bool | None = None*, *from_attributes: bool | None = None*, *context: dict[str, Any] | None = None*) → Model

Validate a pydantic model instance.

Args:

obj: The object to validate. strict: Whether to enforce types strictly. from_attributes: Whether to extract data from object attributes. context: Additional context to pass to the validator.

Raises:

ValidationError: If the object could not be validated.

Returns:

The validated model instance.

classmethod model_validate_json(*json_data: str | bytes | bytearray*, *, *strict: bool | None = None*, *context: dict[str, Any] | None = None*) → Model

Usage docs: https://docs.pydantic.dev/2.6/concepts/json/#json-parsing

Validate the given JSON data against the Pydantic model.

Args:

json_data: The JSON data to validate. strict: Whether to enforce types strictly. context: Extra variables to pass to the validator.

Returns:

The validated Pydantic model.

Raises:

ValueError: If *json_data* is not a JSON string.

classmethod model_validate_strings(*obj: Any*, *, *strict: bool | None = None*, *context: dict[str, Any] | None = None*) → Model

Validate the given object contains string data against the Pydantic model.

Args:

obj: The object contains string data to validate. strict: Whether to enforce types strictly. context: Extra variables to pass to the validator.

Returns:

The validated Pydantic model.

SetStockTagsModel

class zvt.tag.tag_models.SetStockTagsModel(*, *entity_id: str*, *main_tag: str*, *main_tag_reason: Optional[str] = None*, *sub_tag: Optional[str]*, *sub_tag_reason: Optional[str] = None*, *active_hidden_tags: Optional[Dict[str, str]]*)

Bases: `CustomModel`

__init__(**data: Any*) → None

Create a new model by parsing and validating input data from keyword arguments.

Raises [*ValidationError*][pydantic_core.ValidationError] if the input data cannot be validated to form a valid model.

self is explicitly positional-only to allow *self* as a field name.

copy(*, *include: AbstractSetIntStr | MappingIntStrAny | None = None*, *exclude: AbstractSetIntStr | MappingIntStrAny | None = None*, *update: Dict[str, Any] | None = None*, *deep: bool = False*) → Model

Returns a copy of the model.

!!! warning "Deprecated"

This method is now deprecated; use *model_copy* instead.

If you need *include* or *exclude*, use:

```
`py data = self.model_dump(include=include, exclude=exclude, round_trip=True) data = {**data, **(update or {})} copied = self.model_validate(data) `
```

Args:

include: Optional set or mapping specifying which fields to include in the copied model. exclude: Optional set or mapping

specifying which fields to exclude in the copied model. update: Optional dictionary of field-value pairs to override field values in the copied model. deep: If True, the values of fields that are Pydantic models will be deep-copied.

Returns:

A copy of the model with included, excluded and updated fields as specified.

model_computed_fields*: ClassVar[dict[str, ComputedFieldInfo]] = {}*

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

model_config*: ClassVar[ConfigDict] = {'allow_inf_nan': True, 'from_attributes': True}*

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

classmethod model_construct(*_fields_set: set[str] | None = None*, ***values: Any*) → Model

Creates a new instance of the *Model* class with validated data.

Creates a new model setting *__dict__* and *__pydantic_fields_set__* from trusted or pre-validated data. Default values are respected, but no other validation is performed. Behaves as if *Config.extra = 'allow'* was set since it adds all passed values

Args:

_fields_set: The set of field names accepted for the Model instance. values: Trusted or pre-validated data dictionary.

Returns:

A new instance of the *Model* class with validated data.

model_copy(*, *update: dict[str, Any] | None = None*, *deep: bool = False*) → Model

Usage docs: https://docs.pydantic.dev/2.6/concepts/serialization/#model_copy

Returns a copy of the model.

Args:

update: Values to change/add in the new model. Note: the data is not validated
before creating the new model. You should trust this data.

deep: Set to *True* to make a deep copy of the model.

Returns:

New model instance.

model_dump(*, *mode: Literal['json', 'python'] | str = 'python'*, *include: IncEx = None*, *exclude: IncEx = None*, *by_alias: bool = False*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*, *round_trip: bool = False*, *warnings: bool = True*) → dict[str, Any]

Usage docs: https://docs.pydantic.dev/2.6/concepts/serialization/#modelmodel_dump

Generate a dictionary representation of the model, optionally specifying which fields to include or exclude.

Args:

mode: The mode in which *to_python* should run.
If mode is 'json', the output will only contain JSON serializable types. If mode is 'python', the output may contain non-JSON-serializable Python objects.

include: A list of fields to include in the output. exclude: A list of fields to exclude from the output. by_alias: Whether to use the field's alias in the dictionary key if defined. exclude_unset: Whether to exclude fields that have not been explicitly set. exclude_defaults: Whether to exclude fields that are set to their default value. exclude_none: Whether to exclude fields that have a value of *None*. round_trip: If True, dumped values should be

valid as input for non-idempotent types such as Json[T]. warnings: Whether to log warnings when invalid fields are encountered.

Returns:

A dictionary representation of the model.

model_dump_json(*, *indent: int | None = None*, *include: IncEx = None*, *exclude: IncEx = None*, *by_alias: bool = False*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*, *round_trip: bool = False*, *warnings: bool = True*) → str

Usage docs: https://docs.pydantic.dev/2.6/concepts/serialization/#modelmodel_dump_json

Generates a JSON representation of the model using Pydantic's *to_json* method.

Args:

indent: Indentation to use in the JSON output. If None is passed, the output will be compact. include: Field(s) to include in the JSON output. exclude: Field(s) to exclude from the JSON output. by_alias: Whether to serialize using field aliases. exclude_unset: Whether to exclude fields that have not been explicitly set. exclude_defaults: Whether to exclude fields that are set to their default value. exclude_none: Whether to exclude fields that have a value of *None*. round_trip: If True, dumped values should be valid as input for non-idempotent types such as Json[T]. warnings: Whether to log warnings when invalid fields are encountered.

Returns:

A JSON string representation of the model.

property model_extra*: dict[str, Any] | None*

Get extra fields set during validation.

Returns:

A dictionary of extra fields, or *None* if *config.extra* is not set to *"allow"*.

model_fields*: ClassVar[dict[str, FieldInfo]] = {'active_hidden_tags': FieldInfo(annotation=Union[Dict[str, str], NoneType], required=True), 'entity_id': FieldInfo(annotation=str, required=True), 'main_tag': FieldInfo(annotation=str, required=True), 'main_tag_reason': FieldInfo(annotation=Union[str, NoneType], required=False), 'sub_tag': FieldInfo(annotation=Union[str, NoneType], required=True), 'sub_tag_reason': FieldInfo(annotation=Union[str, NoneType], required=False)}*

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

property model_fields_set*: set[str]*

Returns the set of fields that have been explicitly set on this model instance.

Returns:

A set of strings representing the fields that have been set, i.e. that were not filled from defaults.

classmethod model_json_schema(*by_alias: bool = True*, *ref_template: str = '#/$defs/{model}'*, *schema_generator: type[pydantic.json_schema.GenerateJsonSchema] = <class 'pydantic.json_schema.GenerateJsonSchema'>*, *mode: ~typing.Literal['validation', 'serialization'] = 'validation'*) → dict[str, Any]

Generates a JSON schema for a model class.

Args:

by_alias: Whether to use attribute aliases or not. ref_template: The reference template. schema_generator: To override the logic used to generate the JSON schema, as a subclass of

GenerateJsonSchema with your desired modifications

mode: The mode in which to generate the schema.

Returns:

The JSON schema for the given model class.

classmethod model_parametrized_name(*params: tuple[type[Any], ...]*) → str

Compute the class name for parametrizations of generic classes.

This method can be overridden to achieve a custom naming scheme for generic BaseModels.

Args:

params: Tuple of types of the class. Given a generic class *Model* with 2 type variables and a concrete model *Model[str, int]*, the value *(str, int)* would be passed to *params*.

Returns:

String representing the new class where *params* are passed to *cls* as type variables.

Raises:

TypeError: Raised when trying to generate concrete names for non-generic models.

model_post_init(*_BaseModel__context: Any*) → None

Override this method to perform additional initialization after *__init__* and *model_construct*. This is useful if you want to do some validation that requires the entire model to be initialized.

classmethod model_rebuild(*, *force: bool = False*, *raise_errors: bool = True*, *_parent_namespace_depth: int = 2*, *_types_namespace: Optional[dict[str, Any]] = None*) → bool | None

Try to rebuild the pydantic-core schema for the model.

This may be necessary when one of the annotations is a ForwardRef which could not be resolved during the initial attempt to build the schema, and automatic rebuilding fails.

Args:

force: Whether to force the rebuilding of the model schema, defaults to *False*. raise_errors: Whether to raise errors, defaults to *True*. _parent_namespace_depth: The depth level of the parent namespace, defaults to 2. _types_namespace: The types namespace, defaults to *None*.

Returns:

Returns *None* if the schema is already "complete" and rebuilding was not required. If rebuilding _was_ required, returns *True* if rebuilding was successful, otherwise *False*.

classmethod model_validate(*obj: Any*, *, *strict: bool | None = None*, *from_attributes: bool | None = None*, *context: dict[str, Any] | None = None*) → Model

Validate a pydantic model instance.

Args:

obj: The object to validate. strict: Whether to enforce types strictly. from_attributes: Whether to extract data from object attributes. context: Additional context to pass to the validator.

Raises:

ValidationError: If the object could not be validated.

Returns:

The validated model instance.

classmethod model_validate_json(*json_data: str | bytes | bytearray*, *, *strict: bool | None = None*, *context: dict[str, Any] | None = None*) → Model

Usage docs: https://docs.pydantic.dev/2.6/concepts/json/#json-parsing

Validate the given JSON data against the Pydantic model.

Args:
json_data: The JSON data to validate. strict: Whether to enforce types strictly. context: Extra variables to pass to the validator.

Returns:
The validated Pydantic model.

Raises:
ValueError: If *json_data* is not a JSON string.

classmethod model_validate_strings(*obj: Any*, *, *strict: bool | None = None*, *context: dict[str, Any] | None = None*) → Model

Validate the given object contains string data against the Pydantic model.

Args:
obj: The object contains string data to validate. strict: Whether to enforce types strictly. context: Extra variables to pass to the validator.

Returns:
The validated Pydantic model.

SimpleStockTagsModel

class zvt.tag.tag_models.SimpleStockTagsModel(*, *entity_id: str*, *name: str*, *main_tag: str*, *main_tag_reason: Optional[str] = None*, *main_tags: Dict[str, str]*, *sub_tag: Optional[str]*, *sub_tag_reason: Optional[str] = None*, *sub_tags: Optional[Dict[str, str]]*, *active_hidden_tags: Optional[Dict[str, str]]*)

Bases: `CustomModel`

__init__(**data: Any*) → None

Create a new model by parsing and validating input data from keyword arguments.

Raises [*ValidationError*][pydantic_core.ValidationError] if the input data cannot be validated to form a valid model.

self is explicitly positional-only to allow *self* as a field name.

copy(*, *include: AbstractSetIntStr | MappingIntStrAny | None = None*, *exclude: AbstractSetIntStr | MappingIntStrAny | None = None*, *update: Dict[str, Any] | None = None*, *deep: bool = False*) → Model

Returns a copy of the model.

!!! warning "Deprecated"

This method is now deprecated; use *model_copy* instead.

If you need *include* or *exclude*, use:

```
`py data = self.model_dump(include=include, exclude=exclude, round_trip=True) data = {**data, **(update or {})} copied = self.model_validate(data) `
```

Args:

include: Optional set or mapping specifying which fields to include in the copied model. exclude: Optional set or mapping specifying which fields to exclude in the copied model. update: Optional dictionary of field-value pairs to override field values in the copied model. deep: If True, the values of fields that are Pydantic models will be deep-copied.

Returns:

A copy of the model with included, excluded and updated fields as specified.

model_computed_fields*: ClassVar[dict[str, ComputedFieldInfo]] = {}*

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

model_config*: ClassVar[ConfigDict] = {'allow_inf_nan': True, 'from_attributes': True}*

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

classmethod model_construct(*_fields_set: set[str] | None = None*, ***values: Any*) → Model

Creates a new instance of the *Model* class with validated data.

Creates a new model setting *__dict__* and *__pydantic_fields_set__* from trusted or pre-validated data. Default values are respected, but no other validation is performed. Behaves as if *Config.extra = 'allow'* was set since it adds all passed values

Args:

_fields_set: The set of field names accepted for the Model instance. values: Trusted or pre-validated data dictionary.

Returns:

A new instance of the *Model* class with validated data.

model_copy(*, *update: dict[str, Any] | None = None*, *deep: bool = False*) → Model

Usage docs: https://docs.pydantic.dev/2.6/concepts/serialization/#model_copy

Returns a copy of the model.

Args:

update: Values to change/add in the new model. Note: the data is not validated
before creating the new model. You should trust this data.

deep: Set to *True* to make a deep copy of the model.

Returns:

New model instance.

model_dump(*, *mode: Literal['json', 'python'] | str = 'python'*, *include: IncEx = None*, *exclude: IncEx = None*, *by_alias: bool = False*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*, *round_trip: bool = False*, *warnings: bool = True*) → dict[str, Any]

Usage docs: https://docs.pydantic.dev/2.6/concepts/serialization/#modelmodel_dump

Generate a dictionary representation of the model, optionally specifying which fields to include or exclude.

Args:

mode: The mode in which *to_python* should run.
If mode is 'json', the output will only contain JSON serializable types. If mode is 'python', the output may contain non-JSON-serializable Python objects.

include: A list of fields to include in the output. exclude: A list of fields to exclude from the output. by_alias: Whether to use the

field's alias in the dictionary key if defined. exclude_unset: Whether to exclude fields that have not been explicitly set. exclude_defaults: Whether to exclude fields that are set to their default value. exclude_none: Whether to exclude fields that have a value of *None*. round_trip: If True, dumped values should be valid as input for non-idempotent types such as Json[T]. warnings: Whether to log warnings when invalid fields are encountered.

Returns:

A dictionary representation of the model.

model_dump_json(*, *indent: int | None = None*, *include: IncEx = None*, *exclude: IncEx = None*, *by_alias: bool = False*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*, *round_trip: bool = False*, *warnings: bool = True*) → str

Usage docs: https://docs.pydantic.dev/2.6/concepts/serialization/#modelmodel_dump_json

Generates a JSON representation of the model using Pydantic's *to_json* method.

Args:

indent: Indentation to use in the JSON output. If None is passed, the output will be compact. include: Field(s) to include in the JSON output. exclude: Field(s) to exclude from the JSON output. by_alias: Whether to serialize using field aliases. exclude_unset: Whether to exclude fields that have not been explicitly set. exclude_defaults: Whether to exclude fields that are set to their default value. exclude_none: Whether to exclude fields that have a value of *None*. round_trip: If True, dumped values should be valid as input for non-idempotent types such as Json[T]. warnings: Whether to log warnings when invalid fields are encountered.

Returns:

A JSON string representation of the model.

property model_extra*: dict[str, Any] | None*

Get extra fields set during validation.

Returns:

A dictionary of extra fields, or *None* if *config.extra* is not set to *"allow"*.

model_fields*: ClassVar[dict[str, FieldInfo]] = {'active_hidden_tags': FieldInfo(annotation=Union[Dict[str, str], NoneType], required=True), 'entity_id': FieldInfo(annotation=str, required=True), 'main_tag': FieldInfo(annotation=str, required=True), 'main_tag_reason': FieldInfo(annotation=Union[str, NoneType], required=False), 'main_tags': FieldInfo(annotation=Dict[str, str], required=True), 'name': FieldInfo(annotation=str, required=True), 'sub_tag': FieldInfo(annotation=Union[str, NoneType], required=True), 'sub_tag_reason': FieldInfo(annotation=Union[str, NoneType], required=False), 'sub_tags': FieldInfo(annotation=Union[Dict[str, str], NoneType], required=True)}*

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

property model_fields_set*: set[str]*

Returns the set of fields that have been explicitly set on this model instance.

Returns:

A set of strings representing the fields that have been set,
i.e. that were not filled from defaults.

classmethod model_json_schema(*by_alias: bool = True*, *ref_template: str = '#/$defs/{model}'*, *schema_generator: type[pydantic.json_schema.GenerateJsonSchema] = <class 'pydantic.json_schema.GenerateJsonSchema'>*, *mode: ~typing.Literal['validation', 'serialization'] = 'validation'*) → dict[str, Any]

Generates a JSON schema for a model class.

Args:

by_alias: Whether to use attribute aliases or not. ref_template: The reference template. schema_generator: To override the logic used to generate the JSON schema, as a subclass of

GenerateJsonSchema with your desired modifications

mode: The mode in which to generate the schema.

Returns:

The JSON schema for the given model class.

classmethod model_parametrized_name(*params: tuple[type[Any], ...]*) → str

Compute the class name for parametrizations of generic classes.

This method can be overridden to achieve a custom naming scheme for generic BaseModels.

Args:

params: Tuple of types of the class. Given a generic class *Model* with 2 type variables and a concrete model *Model[str, int]*, the value *(str, int)* would be passed to *params*.

Returns:

String representing the new class where *params* are passed to *cls* as type variables.

Raises:

TypeError: Raised when trying to generate concrete names for non-generic models.

model_post_init(*_BaseModel__context: Any*) → None

Override this method to perform additional initialization after *__init__* and *model_construct*. This is useful if you want to do some validation that requires the entire model to be initialized.

classmethod model_rebuild(*, *force: bool = False*, *raise_errors: bool = True*, *_parent_namespace_depth: int = 2*, *_types_namespace: Optional[dict[str, Any]] = None*) → bool | None

Try to rebuild the pydantic-core schema for the model.

This may be necessary when one of the annotations is a ForwardRef which could not be resolved during the initial attempt to build the schema, and automatic rebuilding fails.

Args:

force: Whether to force the rebuilding of the model schema, defaults to *False*. raise_errors: Whether to raise errors, defaults to *True*. _parent_namespace_depth: The depth level of the parent namespace, defaults to 2. _types_namespace: The types namespace, defaults to *None*.

Returns:

Returns *None* if the schema is already "complete" and rebuilding was not required. If rebuilding _was_ required, returns *True* if rebuilding was successful, otherwise *False*.

classmethod model_validate(*obj: Any*, *, *strict: bool | None = None*, *from_attributes: bool | None = None*, *context: dict[str, Any] | None = None*) → Model

Validate a pydantic model instance.

Args:

obj: The object to validate. strict: Whether to enforce types strictly. from_attributes: Whether to extract data from object attributes. context: Additional context to pass to the validator.

Raises:

ValidationError: If the object could not be validated.

Returns:

The validated model instance.

classmethod model_validate_json(*json_data: str | bytes | bytearray*, *, *strict: bool | None = None*, *context: dict[str, Any] | None = None*) → Model

Usage docs: https://docs.pydantic.dev/2.6/concepts/json/#json-parsing

Validate the given JSON data against the Pydantic model.

Args:

json_data: The JSON data to validate. strict: Whether to enforce types strictly. context: Extra variables to pass to the validator.

Returns:

The validated Pydantic model.

Raises:

ValueError: If *json_data* is not a JSON string.

classmethod model_validate_strings(*obj: Any*, *, *strict: bool | None = None*, *context: dict[str, Any] | None = None*) → Model

Validate the given object contains string data against the Pydantic model.

Args:

obj: The object contains string data to validate. strict: Whether to enforce types strictly. context: Extra variables to pass to the validator.

Returns:

The validated Pydantic model.

StockPoolInfoModel

class zvt.tag.tag_models.StockPoolInfoModel(*, *id: str*, *entity_id: str*, *timestamp: datetime*, *stock_pool_type: StockPoolType*, *stock_pool_name: str*)

Bases: `MixinModel`

__init__(**data: Any*) → None

Create a new model by parsing and validating input data from keyword arguments.

Raises [*ValidationError*][pydantic_core.ValidationError] if the input data cannot be validated to form a valid model.

self is explicitly positional-only to allow *self* as a field name.

copy(*, *include: AbstractSetIntStr | MappingIntStrAny | None = None*, *exclude: AbstractSetIntStr | MappingIntStrAny | None = None*, *update: Dict[str, Any] | None = None*, *deep: bool = False*) → Model

Returns a copy of the model.

!!! warning "Deprecated"

This method is now deprecated; use *model_copy* instead.

If you need *include* or *exclude*, use:

```
`py data = self.model_dump(include=include, exclude=exclude, round_trip=True) data = {**data, **(update or {})} copied = self.model_validate(data) `
```

Args:

include: Optional set or mapping specifying which fields to include in the copied model. exclude: Optional set or mapping specifying which fields to exclude in the copied model. update:

Optional dictionary of field-value pairs to override field values in the copied model. deep: If True, the values of fields that are Pydantic models will be deep-copied.

Returns:

A copy of the model with included, excluded and updated fields as specified.

model_computed_fields*: ClassVar[dict[str, ComputedFieldInfo]] = {}*

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

model_config*: ClassVar[ConfigDict] = {'allow_inf_nan': True, 'from_attributes': True}*

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

classmethod model_construct(*_fields_set: set[str] | None = None*, ***values: Any*) → Model

Creates a new instance of the *Model* class with validated data.

Creates a new model setting *__dict__* and *__pydantic_fields_set__* from trusted or pre-validated data. Default values are respected, but no other validation is performed. Behaves as if *Config.extra = 'allow'* was set since it adds all passed values

Args:

_fields_set: The set of field names accepted for the Model instance. values: Trusted or pre-validated data dictionary.

Returns:

A new instance of the *Model* class with validated data.

model_copy(*, *update: dict[str, Any] | None = None*, *deep: bool = False*) → Model

Usage docs: https://docs.pydantic.dev/2.6/concepts/serialization/#model_copy

Returns a copy of the model.

Args:

update: Values to change/add in the new model. Note: the data is not validated
before creating the new model. You should trust this data.

deep: Set to *True* to make a deep copy of the model.

Returns:

New model instance.

model_dump(*, *mode: Literal['json', 'python'] | str = 'python'*, *include: IncEx = None*, *exclude: IncEx = None*, *by_alias: bool = False*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*, *round_trip: bool = False*, *warnings: bool = True*) → dict[str, Any]

Usage docs: https://docs.pydantic.dev/2.6/concepts/serialization/#modelmodel_dump

Generate a dictionary representation of the model, optionally specifying which fields to include or exclude.

Args:

mode: The mode in which *to_python* should run.
If mode is 'json', the output will only contain JSON serializable types. If mode is 'python', the output may contain non-JSON-serializable Python objects.

include: A list of fields to include in the output. exclude: A list of fields to exclude from the output. by_alias: Whether to use the field's alias in the dictionary key if defined. exclude_unset: Whether to exclude fields that have not been explicitly set. exclude_defaults: Whether to exclude fields that are set to their default value. exclude_none: Whether to exclude fields that have a value of *None*. round_trip: If True, dumped values should be

valid as input for non-idempotent types such as Json[T]. warnings: Whether to log warnings when invalid fields are encountered.

Returns:

A dictionary representation of the model.

model_dump_json(*, *indent: int | None = None*, *include: IncEx = None*, *exclude: IncEx = None*, *by_alias: bool = False*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*, *round_trip: bool = False*, *warnings: bool = True*) → str

Usage docs: https://docs.pydantic.dev/2.6/concepts/serialization/#modelmodel_dump_json

Generates a JSON representation of the model using Pydantic's *to_json* method.

Args:

indent: Indentation to use in the JSON output. If None is passed, the output will be compact. include: Field(s) to include in the JSON output. exclude: Field(s) to exclude from the JSON output. by_alias: Whether to serialize using field aliases. exclude_unset: Whether to exclude fields that have not been explicitly set. exclude_defaults: Whether to exclude fields that are set to their default value. exclude_none: Whether to exclude fields that have a value of *None*. round_trip: If True, dumped values should be valid as input for non-idempotent types such as Json[T]. warnings: Whether to log warnings when invalid fields are encountered.

Returns:

A JSON string representation of the model.

property model_extra*: dict[str, Any] | None*

Get extra fields set during validation.

Returns:

A dictionary of extra fields, or *None* if *config.extra* is not set to *"allow"*.

model_fields*: ClassVar[dict[str, FieldInfo]] = {'entity_id': FieldInfo(annotation=str, required=True), 'id': FieldInfo(annotation=str, required=True), 'stock_pool_name': FieldInfo(annotation=str, required=True), 'stock_pool_type': FieldInfo(annotation=StockPoolType, required=True), 'timestamp': FieldInfo(annotation=datetime, required=True)}*

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

property model_fields_set*: set[str]*

Returns the set of fields that have been explicitly set on this model instance.

Returns:

A set of strings representing the fields that have been set, i.e. that were not filled from defaults.

classmethod model_json_schema(*by_alias: bool = True*, *ref_template: str = '#/$defs/{model}'*, *schema_generator: type[pydantic.json_schema.GenerateJsonSchema] = <class 'pydantic.json_schema.GenerateJsonSchema'>*, *mode: ~typing.Literal['validation', 'serialization'] = 'validation'*) → dict[str, Any]

Generates a JSON schema for a model class.

Args:

by_alias: Whether to use attribute aliases or not. ref_template: The reference template. schema_generator: To override the logic used to generate the JSON schema, as a subclass of

GenerateJsonSchema with your desired modifications

mode: The mode in which to generate the schema.

Returns:

The JSON schema for the given model class.

classmethod model_parametrized_name(*params: tuple[type[Any], ...]*) → str

Compute the class name for parametrizations of generic classes.

This method can be overridden to achieve a custom naming scheme for generic BaseModels.

Args:

params: Tuple of types of the class. Given a generic class *Model* with 2 type variables and a concrete model *Model[str, int]*, the value *(str, int)* would be passed to *params*.

Returns:

String representing the new class where *params* are passed to *cls* as type variables.

Raises:

TypeError: Raised when trying to generate concrete names for non-generic models.

model_post_init(*_BaseModel__context: Any*) → None

Override this method to perform additional initialization after *__init__* and *model_construct*. This is useful if you want to do some validation that requires the entire model to be initialized.

classmethod model_rebuild(*, *force: bool = False*, *raise_errors: bool = True*, *_parent_namespace_depth: int = 2*, *_types_namespace: Optional[dict[str, Any]] = None*) → bool | None

Try to rebuild the pydantic-core schema for the model.

This may be necessary when one of the annotations is a ForwardRef which could not be resolved during the initial attempt to build the schema, and automatic rebuilding fails.

Args:

force: Whether to force the rebuilding of the model schema, defaults to *False*. raise_errors: Whether to raise errors, defaults to *True*. _parent_namespace_depth: The depth level of the parent namespace, defaults to 2. _types_namespace: The types namespace, defaults to *None*.

Returns:

Returns *None* if the schema is already "complete" and rebuilding was not required. If rebuilding _was_ required, returns *True* if rebuilding was successful, otherwise *False*.

classmethod model_validate(*obj: Any*, *, *strict: bool | None = None*, *from_attributes: bool | None = None*, *context: dict[str, Any] | None = None*) → Model

Validate a pydantic model instance.

Args:

obj: The object to validate. strict: Whether to enforce types strictly. from_attributes: Whether to extract data from object attributes. context: Additional context to pass to the validator.

Raises:

ValidationError: If the object could not be validated.

Returns:

The validated model instance.

classmethod model_validate_json(*json_data: str | bytes | bytearray*, *, *strict: bool | None = None*, *context: dict[str, Any] | None = None*) → Model

Usage docs: https://docs.pydantic.dev/2.6/concepts/json/#json-parsing

Validate the given JSON data against the Pydantic model.

Args:

json_data: The JSON data to validate. strict: Whether to enforce types strictly. context: Extra variables to pass to the validator.

Returns:
The validated Pydantic model.

Raises:
ValueError: If *json_data* is not a JSON string.

classmethod model_validate_strings(*obj: Any*, *, *strict: bool | None = None*, *context: dict[str, Any] | None = None*) → Model

Validate the given object contains string data against the Pydantic model.

Args:
obj: The object contains string data to validate. strict: Whether to enforce types strictly. context: Extra variables to pass to the validator.

Returns:
The validated Pydantic model.

StockPoolModel

class zvt.tag.tag_models.StockPoolModel(*, *id: str*, *entity_id: str*, *timestamp: datetime*, *stock_pool_name: str*, *entity_ids: List[str]*)

Bases: **MixinModel**

__init__(**data: Any*) → None

Create a new model by parsing and validating input data from keyword arguments.

Raises [*ValidationError*][pydantic_core.ValidationError] if the input data cannot be validated to form a valid model.

self is explicitly positional-only to allow *self* as a field name.

copy(*, *include: AbstractSetIntStr | MappingIntStrAny | None = None*, *exclude: AbstractSetIntStr | MappingIntStrAny | None = None*, *update: Dict[str, Any] | None = None*, *deep: bool = False*) → Model

Returns a copy of the model.

!!! warning "Deprecated"

This method is now deprecated; use *model_copy* instead.

If you need *include* or *exclude*, use:

```
`py data = self.model_dump(include=include, exclude=exclude, round_trip=True) data = {**data, **(update or {})} copied = self.model_validate(data) `
```

Args:

include: Optional set or mapping specifying which fields to include in the copied model. exclude: Optional set or mapping specifying which fields to exclude in the copied model. update: Optional dictionary of field-value pairs to override field values

in the copied model. deep: If True, the values of fields that are Pydantic models will be deep-copied.

Returns:

A copy of the model with included, excluded and updated fields as specified.

model_computed_fields*: ClassVar[dict[str, ComputedFieldInfo]] = {}*

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

model_config*: ClassVar[ConfigDict] = {'allow_inf_nan': True, 'from_attributes': True}*

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

classmethod model_construct(*_fields_set: set[str] | None = None*, ***values: Any*) → Model

Creates a new instance of the *Model* class with validated data.

Creates a new model setting *__dict__* and *__pydantic_fields_set__* from trusted or pre-validated data. Default values are respected, but no other validation is performed. Behaves as if *Config.extra = 'allow'* was set since it adds all passed values

Args:

_fields_set: The set of field names accepted for the Model instance. values: Trusted or pre-validated data dictionary.

Returns:

A new instance of the *Model* class with validated data.

model_copy(*, *update: dict[str, Any] | None = None*, *deep: bool = False*) → Model

Usage docs: https://docs.pydantic.dev/2.6/concepts/serialization/#model_copy

Returns a copy of the model.

Args:

update: Values to change/add in the new model. Note: the data is not validated
before creating the new model. You should trust this data.

deep: Set to *True* to make a deep copy of the model.

Returns:

New model instance.

model_dump(*, *mode: Literal['json', 'python'] | str = 'python'*, *include: IncEx = None*, *exclude: IncEx = None*, *by_alias: bool = False*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*, *round_trip: bool = False*, *warnings: bool = True*) → dict[str, Any]

Usage docs: https://docs.pydantic.dev/2.6/concepts/serialization/#modelmodel_dump

Generate a dictionary representation of the model, optionally specifying which fields to include or exclude.

Args:

mode: The mode in which *to_python* should run.
If mode is 'json', the output will only contain JSON serializable types. If mode is 'python', the output may contain non-JSON-serializable Python objects.

include: A list of fields to include in the output. exclude: A list of fields to exclude from the output. by_alias: Whether to use the field's alias in the dictionary key if defined. exclude_unset: Whether to exclude fields that have not been explicitly set. exclude_defaults: Whether to exclude fields that are set to their default value. exclude_none: Whether to exclude fields that have a value of *None*. round_trip: If True, dumped values should be valid as input for non-idempotent types such as Json[T].

warnings: Whether to log warnings when invalid fields are encountered.

Returns:

A dictionary representation of the model.

model_dump_json(*, *indent: int | None = None*, *include: IncEx = None*, *exclude: IncEx = None*, *by_alias: bool = False*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*, *round_trip: bool = False*, *warnings: bool = True*) → str

Usage docs: https://docs.pydantic.dev/2.6/concepts/serialization/#modelmodel_dump_json

Generates a JSON representation of the model using Pydantic's *to_json* method.

Args:

indent: Indentation to use in the JSON output. If None is passed, the output will be compact. include: Field(s) to include in the JSON output. exclude: Field(s) to exclude from the JSON output. by_alias: Whether to serialize using field aliases. exclude_unset: Whether to exclude fields that have not been explicitly set. exclude_defaults: Whether to exclude fields that are set to their default value. exclude_none: Whether to exclude fields that have a value of *None*. round_trip: If True, dumped values should be valid as input for non-idempotent types such as Json[T]. warnings: Whether to log warnings when invalid fields are encountered.

Returns:

A JSON string representation of the model.

property model_extra*: dict[str, Any] | None*

Get extra fields set during validation.

Returns:

A dictionary of extra fields, or *None* if *config.extra* is not set to *"allow"*.

model_fields*: ClassVar[dict[str, FieldInfo]] = {'entity_id': FieldInfo(annotation=str, required=True), 'entity_ids': FieldInfo(annotation=List[str], required=True), 'id': FieldInfo(annotation=str, required=True), 'stock_pool_name': FieldInfo(annotation=str, required=True), 'timestamp': FieldInfo(annotation=datetime, required=True)}*

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

property model_fields_set*: set[str]*

Returns the set of fields that have been explicitly set on this model instance.

Returns:

A set of strings representing the fields that have been set, i.e. that were not filled from defaults.

classmethod model_json_schema(*by_alias: bool = True*, *ref_template: str = '#/$defs/{model}'*, *schema_generator: type[pydantic.json_schema.GenerateJsonSchema] = <class 'pydantic.json_schema.GenerateJsonSchema'>*, *mode: ~typing.Literal['validation', 'serialization'] = 'validation'*) → dict[str, Any]

Generates a JSON schema for a model class.

Args:

by_alias: Whether to use attribute aliases or not. ref_template: The reference template. schema_generator: To override the logic used to generate the JSON schema, as a subclass of

GenerateJsonSchema with your desired modifications

mode: The mode in which to generate the schema.

Returns:

The JSON schema for the given model class.

classmethod model_parametrized_name(*params: tuple[type[Any], ...]*) → str

Compute the class name for parametrizations of generic classes.

This method can be overridden to achieve a custom naming scheme for generic BaseModels.

Args:

params: Tuple of types of the class. Given a generic class *Model* with 2 type variables and a concrete model *Model[str, int]*, the value *(str, int)* would be passed to *params*.

Returns:

String representing the new class where *params* are passed to *cls* as type variables.

Raises:

TypeError: Raised when trying to generate concrete names for non-generic models.

model_post_init(*_BaseModel__context: Any*) → None

Override this method to perform additional initialization after *__init__* and *model_construct*. This is useful if you want to do some validation that requires the entire model to be initialized.

classmethod model_rebuild(*, *force: bool = False*, *raise_errors: bool = True*, *_parent_namespace_depth: int = 2*, *_types_namespace: Optional[dict[str, Any]] = None*) → bool | None

Try to rebuild the pydantic-core schema for the model.

This may be necessary when one of the annotations is a ForwardRef which could not be resolved during the initial attempt to build the schema, and automatic rebuilding fails.

Args:

force: Whether to force the rebuilding of the model schema, defaults to *False*. raise_errors: Whether to raise errors, defaults to *True*. _parent_namespace_depth: The depth level of the parent namespace, defaults to 2. _types_namespace: The types namespace, defaults to *None*.

Returns:

Returns *None* if the schema is already "complete" and rebuilding was not required. If rebuilding _was_ required, returns *True* if rebuilding was successful, otherwise *False*.

classmethod model_validate(*obj: Any*, *, *strict: bool | None = None*, *from_attributes: bool | None = None*, *context: dict[str, Any] | None = None*) → Model

Validate a pydantic model instance.

Args:

obj: The object to validate. strict: Whether to enforce types strictly. from_attributes: Whether to extract data from object attributes. context: Additional context to pass to the validator.

Raises:

ValidationError: If the object could not be validated.

Returns:

The validated model instance.

classmethod model_validate_json(*json_data: str | bytes | bytearray*, *, *strict: bool | None = None*, *context: dict[str, Any] | None = None*) → Model

Usage docs: https://docs.pydantic.dev/2.6/concepts/json/#json-parsing

Validate the given JSON data against the Pydantic model.

Args:

json_data: The JSON data to validate. strict: Whether to enforce types strictly. context: Extra variables to pass to the validator.

Returns:
The validated Pydantic model.

Raises:
ValueError: If *json_data* is not a JSON string.

classmethod model_validate_strings(*obj: Any*, *, *strict: bool | None = None*, *context: dict[str, Any] | None = None*) → Model

Validate the given object contains string data against the Pydantic model.

Args:
obj: The object contains string data to validate. strict: Whether to enforce types strictly. context: Extra variables to pass to the validator.

Returns:
The validated Pydantic model.

StockPoolsModel

class zvt.tag.tag_models.StockPoolsModel(*, *id: str*, *entity_id: str*, *timestamp: datetime*, *stock_pool_name: str*, *entity_ids: List[str]*)

Bases: **MixinModel**

__init__(**data: Any*) → None

Create a new model by parsing and validating input data from keyword arguments.

Raises [*ValidationError*][pydantic_core.ValidationError] if the input data cannot be validated to form a valid model.

self is explicitly positional-only to allow *self* as a field name.

copy(*, *include: AbstractSetIntStr | MappingIntStrAny | None = None*, *exclude: AbstractSetIntStr | MappingIntStrAny | None = None*, *update: Dict[str, Any] | None = None*, *deep: bool = False*) → Model

Returns a copy of the model.

!!! warning "Deprecated"

This method is now deprecated; use *model_copy* instead.

If you need *include* or *exclude*, use:

```
`py data = self.model_dump(include=include, exclude=exclude, round_trip=True) data = {**data, **(update or {})} copied = self.model_validate(data) `
```

Args:

include: Optional set or mapping specifying which fields to include in the copied model. exclude: Optional set or mapping specifying which fields to exclude in the copied model. update: Optional dictionary of field-value pairs to override field values

in the copied model. deep: If True, the values of fields that are Pydantic models will be deep-copied.

Returns:

A copy of the model with included, excluded and updated fields as specified.

model_computed_fields*: ClassVar[dict[str, ComputedFieldInfo]] = {}*

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

model_config*: ClassVar[ConfigDict] = {'allow_inf_nan': True, 'from_attributes': True}*

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

classmethod model_construct(*_fields_set: set[str] | None = None*, ***values: Any*) → Model

Creates a new instance of the *Model* class with validated data.

Creates a new model setting *__dict__* and *__pydantic_fields_set__* from trusted or pre-validated data. Default values are respected, but no other validation is performed. Behaves as if *Config.extra = 'allow'* was set since it adds all passed values

Args:

_fields_set: The set of field names accepted for the Model instance. values: Trusted or pre-validated data dictionary.

Returns:

A new instance of the *Model* class with validated data.

model_copy(*, *update: dict[str, Any] | None = None*, *deep: bool = False*) → Model

Usage docs: https://docs.pydantic.dev/2.6/concepts/serialization/#model_copy

Returns a copy of the model.

Args:

update: Values to change/add in the new model. Note: the data is not validated
before creating the new model. You should trust this data.

deep: Set to *True* to make a deep copy of the model.

Returns:

New model instance.

model_dump(*, *mode: Literal['json', 'python'] | str = 'python'*, *include: IncEx = None*, *exclude: IncEx = None*, *by_alias: bool = False*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*, *round_trip: bool = False*, *warnings: bool = True*) → dict[str, Any]

Usage docs: https://docs.pydantic.dev/2.6/concepts/serialization/#modelmodel_dump

Generate a dictionary representation of the model, optionally specifying which fields to include or exclude.

Args:

mode: The mode in which *to_python* should run.
If mode is 'json', the output will only contain JSON serializable types. If mode is 'python', the output may contain non-JSON-serializable Python objects.

include: A list of fields to include in the output. exclude: A list of fields to exclude from the output. by_alias: Whether to use the field's alias in the dictionary key if defined. exclude_unset: Whether to exclude fields that have not been explicitly set. exclude_defaults: Whether to exclude fields that are set to their default value. exclude_none: Whether to exclude fields that have a value of *None*. round_trip: If True, dumped values should be valid as input for non-idempotent types such as Json[T].

warnings: Whether to log warnings when invalid fields are encountered.

Returns:

A dictionary representation of the model.

model_dump_json(*, *indent: int | None = None*, *include: IncEx = None*, *exclude: IncEx = None*, *by_alias: bool = False*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*, *round_trip: bool = False*, *warnings: bool = True*) → str

Usage docs: https://docs.pydantic.dev/2.6/concepts/serialization/#modelmodel_dump_json

Generates a JSON representation of the model using Pydantic's *to_json* method.

Args:

indent: Indentation to use in the JSON output. If None is passed, the output will be compact. include: Field(s) to include in the JSON output. exclude: Field(s) to exclude from the JSON output. by_alias: Whether to serialize using field aliases. exclude_unset: Whether to exclude fields that have not been explicitly set. exclude_defaults: Whether to exclude fields that are set to their default value. exclude_none: Whether to exclude fields that have a value of *None*. round_trip: If True, dumped values should be valid as input for non-idempotent types such as Json[T]. warnings: Whether to log warnings when invalid fields are encountered.

Returns:

A JSON string representation of the model.

property model_extra*: dict[str, Any] | None*

Get extra fields set during validation.

Returns:

A dictionary of extra fields, or *None* if *config.extra* is not set to *"allow"*.

model_fields*: ClassVar[dict[str, FieldInfo]] = {'entity_id': FieldInfo(annotation=str, required=True), 'entity_ids': FieldInfo(annotation=List[str], required=True), 'id': FieldInfo(annotation=str, required=True), 'stock_pool_name': FieldInfo(annotation=str, required=True), 'timestamp': FieldInfo(annotation=datetime, required=True)}*

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

property model_fields_set*: set[str]*

Returns the set of fields that have been explicitly set on this model instance.

Returns:

A set of strings representing the fields that have been set, i.e. that were not filled from defaults.

classmethod model_json_schema(*by_alias: bool = True*, *ref_template: str = '#/$defs/{model}'*, *schema_generator: type[pydantic.json_schema.GenerateJsonSchema] = <class 'pydantic.json_schema.GenerateJsonSchema'>*, *mode: ~typing.Literal['validation', 'serialization'] = 'validation'*) → dict[str, Any]

Generates a JSON schema for a model class.

Args:

by_alias: Whether to use attribute aliases or not. ref_template: The reference template. schema_generator: To override the logic used to generate the JSON schema, as a subclass of

GenerateJsonSchema with your desired modifications

mode: The mode in which to generate the schema.

Returns:

The JSON schema for the given model class.

classmethod model_parametrized_name(*params: tuple[type[Any], ...]*) → str

Compute the class name for parametrizations of generic classes.

This method can be overridden to achieve a custom naming scheme for generic BaseModels.

Args:

params: Tuple of types of the class. Given a generic class *Model* with 2 type variables and a concrete model *Model[str, int]*, the value *(str, int)* would be passed to *params*.

Returns:

String representing the new class where *params* are passed to *cls* as type variables.

Raises:

TypeError: Raised when trying to generate concrete names for non-generic models.

model_post_init(*_BaseModel__context: Any*) → None

Override this method to perform additional initialization after *__init__* and *model_construct*. This is useful if you want to do some validation that requires the entire model to be initialized.

classmethod model_rebuild(*, *force: bool = False*, *raise_errors: bool = True*, *_parent_namespace_depth: int = 2*, *_types_namespace: Optional[dict[str, Any]] = None*) → bool | None

Try to rebuild the pydantic-core schema for the model.

This may be necessary when one of the annotations is a ForwardRef which could not be resolved during the initial attempt to build the schema, and automatic rebuilding fails.

Args:

force: Whether to force the rebuilding of the model schema, defaults to *False*. raise_errors: Whether to raise errors, defaults to *True*. _parent_namespace_depth: The depth level of the parent namespace, defaults to 2. _types_namespace: The types namespace, defaults to *None*.

Returns:

Returns *None* if the schema is already "complete" and rebuilding was not required. If rebuilding _was_ required, returns *True* if rebuilding was successful, otherwise *False*.

classmethod model_validate(*obj: Any*, *, *strict: bool | None = None*, *from_attributes: bool | None = None*, *context: dict[str, Any] | None = None*) → Model

Validate a pydantic model instance.

Args:

obj: The object to validate. strict: Whether to enforce types strictly. from_attributes: Whether to extract data from object attributes. context: Additional context to pass to the validator.

Raises:

ValidationError: If the object could not be validated.

Returns:

The validated model instance.

classmethod model_validate_json(*json_data: str | bytes | bytearray*, *, *strict: bool | None = None*, *context: dict[str, Any] | None = None*) → Model

Usage docs: https://docs.pydantic.dev/2.6/concepts/json/#json-parsing

Validate the given JSON data against the Pydantic model.

Args:

json_data: The JSON data to validate. strict: Whether to enforce types strictly. context: Extra variables to pass to the validator.

Returns:
The validated Pydantic model.

Raises:
ValueError: If *json_data* is not a JSON string.

classmethod model_validate_strings(*obj: Any*, *, *strict: bool | None = None*, *context: dict[str, Any] | None = None*) → Model

Validate the given object contains string data against the Pydantic model.

Args:
obj: The object contains string data to validate. strict: Whether to enforce types strictly. context: Extra variables to pass to the validator.

Returns:
The validated Pydantic model.

StockTagDetailsModel

class zvt.tag.tag_models.StockTagDetailsModel(*, *entity_id: str*, *main_tag: str*, *sub_tag: Optional[str]*, *hidden_tags: Optional[List[str]]*, *code: str*, *name: str*, *recent_reduction: Optional[bool] = None*, *recent_acquisition: Optional[bool] = None*, *recent_unlock: Optional[bool] = None*, *recent_additional_or_rights_issue: Optional[bool] = None*, *recent_positive_earnings_news: Optional[bool] = None*, *recent_negative_earnings_news: Optional[bool] = None*, *recent_dragon_and_tiger_count: Optional[int] = None*, *recent_violation_alert: Optional[bool] = None*, *recent_positive_news: Optional[bool] = None*, *recent_negative_news: Optional[bool] = None*, *recent_news_summary: Optional[Dict[str, str]] = None*)

Bases: `CustomModel`

__init__(**data: Any*) → None

Create a new model by parsing and validating input data from keyword arguments.

Raises [*ValidationError*][pydantic_core.ValidationError] if the input data cannot be validated to form a valid model.

self is explicitly positional-only to allow *self* as a field name.

code*: str*

代码

name*: str*

名字

recent_reduction*: Optional[bool]*

减持

recent_acquisition*: Optional[bool]*

增持

recent_unlock*: Optional[bool]*
解禁

recent_additional_or_rights_issue*: Optional[bool]*
增发配股

recent_positive_earnings_news*: Optional[bool]*
业绩利好

recent_negative_earnings_news*: Optional[bool]*
业绩利空

recent_dragon_and_tiger_count*: Optional[int]*
上榜次数

recent_violation_alert*: Optional[bool]*
违规行为

recent_positive_news*: Optional[bool]*
利好

recent_negative_news*: Optional[bool]*
利空

recent_news_summary*: Optional[Dict[str, str]]*
新闻总结

copy(*, *include: AbstractSetIntStr | MappingIntStrAny | None = None*, *exclude: AbstractSetIntStr | MappingIntStrAny | None = None*, *update: Dict[str, Any] | None = None*, *deep: bool = False*) → Model
Returns a copy of the model.

!!! warning "Deprecated"
This method is now deprecated; use *model_copy* instead.

If you need *include* or *exclude*, use:

`py data = self.model_dump(include=include, exclude=exclude, round_trip=True) data = {**data, **(update or {})} copied = self.model_validate(data) `

Args:

include: Optional set or mapping specifying which fields to include in the copied model. exclude: Optional set or mapping specifying which fields to exclude in the copied model. update: Optional dictionary of field-value pairs to override field values in the copied model. deep: If True, the values of fields that are Pydantic models will be deep-copied.

Returns:

A copy of the model with included, excluded and updated fields as specified.

model_computed_fields*: ClassVar[dict[str, ComputedFieldInfo]] = {}*

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

model_config*: ClassVar[ConfigDict] = {'allow_inf_nan': True, 'from_attributes': True}*

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

classmethod model_construct(*_fields_set: set[str] | None = None*, ***values: Any*) → Model

Creates a new instance of the *Model* class with validated data.

Creates a new model setting __*dict*__ and __*pydantic_fields_set*__ from trusted or pre-validated data. Default values are respected, but no other validation is performed. Behaves as if *Config.extra = 'allow'* was set since it adds all passed values

Args:

_fields_set: The set of field names accepted for the Model instance. values: Trusted or pre-validated data dictionary.

Returns:
A new instance of the *Model* class with validated data.

model_copy(*, *update: dict[str, Any] | None = None*, *deep: bool = False*) → Model

Usage docs:
https://docs.pydantic.dev/2.6/concepts/serialization/#model_copy

Returns a copy of the model.

Args:
update: Values to change/add in the new model. Note: the data is not validated
before creating the new model. You should trust this data.

deep: Set to *True* to make a deep copy of the model.

Returns:
New model instance.

model_dump(*, *mode: Literal['json', 'python'] | str = 'python'*, *include: IncEx = None*, *exclude: IncEx = None*, *by_alias: bool = False*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*, *round_trip: bool = False*, *warnings: bool = True*) → dict[str, Any]

Usage docs:
https://docs.pydantic.dev/2.6/concepts/serialization/#modelmodel_dump

Generate a dictionary representation of the model, optionally specifying which fields to include or exclude.

Args:
mode: The mode in which *to_python* should run.

If mode is ‘json’, the output will only contain JSON serializable types. If mode is ‘python’, the output may contain non-JSON-serializable Python objects.

include: A list of fields to include in the output. exclude: A list of fields to exclude from the output. by_alias: Whether to use the field’s alias in the dictionary key if defined. exclude_unset: Whether to exclude fields that have not been explicitly set. exclude_defaults: Whether to exclude fields that are set to their default value. exclude_none: Whether to exclude fields that have a value of *None*. round_trip: If True, dumped values should be valid as input for non-idempotent types such as Json[T]. warnings: Whether to log warnings when invalid fields are encountered.

Returns:

A dictionary representation of the model.

model_dump_json(*, *indent: int | None = None*, *include: IncEx = None*, *exclude: IncEx = None*, *by_alias: bool = False*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*, *round_trip: bool = False*, *warnings: bool = True*) → str

Usage docs: https://docs.pydantic.dev/2.6/concepts/serialization/#modelmodel_dump_json

Generates a JSON representation of the model using Pydantic’s *to_json* method.

Args:

indent: Indentation to use in the JSON output. If None is passed, the output will be compact. include: Field(s) to include in the JSON output. exclude: Field(s) to exclude from the JSON output. by_alias: Whether to serialize using field aliases. exclude_unset: Whether to exclude fields that have not been explicitly set. exclude_defaults: Whether to exclude fields that are set to their default value. exclude_none: Whether to exclude fields that have a value of *None*. round_trip: If True, dumped

values should be valid as input for non-idempotent types such as Json[T]. warnings: Whether to log warnings when invalid fields are encountered.

Returns:

A JSON string representation of the model.

property model_extra*: dict[str, Any] | None*

Get extra fields set during validation.

Returns:

A dictionary of extra fields, or *None* if *config.extra* is not set to *"allow"*.

model_fields*: ClassVar[dict[str, FieldInfo]] = {'code': FieldInfo(annotation=str, required=True), 'entity_id': FieldInfo(annotation=str, required=True), 'hidden_tags': FieldInfo(annotation=Union[List[str], NoneType], required=True), 'main_tag': FieldInfo(annotation=str, required=True), 'name': FieldInfo(annotation=str, required=True), 'recent_acquisition': FieldInfo(annotation=Union[bool, NoneType], required=False), 'recent_additional_or_rights_issue': FieldInfo(annotation=Union[bool, NoneType], required=False), 'recent_dragon_and_tiger_count': FieldInfo(annotation=Union[int, NoneType], required=False), 'recent_negative_earnings_news': FieldInfo(annotation=Union[bool, NoneType], required=False), 'recent_negative_news': FieldInfo(annotation=Union[bool, NoneType], required=False), 'recent_news_summary': FieldInfo(annotation=Union[Dict[str, str], NoneType], required=False), 'recent_positive_earnings_news': FieldInfo(annotation=Union[bool, NoneType], required=False), 'recent_positive_news': FieldInfo(annotation=Union[bool, NoneType], required=False), 'recent_reduction': FieldInfo(annotation=Union[bool, NoneType], required=False), 'recent_unlock': FieldInfo(annotation=Union[bool, NoneType], required=False), 'recent_violation_alert': FieldInfo(annotation=Union[bool, NoneType], required=False), 'sub_tag': FieldInfo(annotation=Union[str, NoneType], required=True)}*

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

property model_fields_set*: set[str]*

Returns the set of fields that have been explicitly set on this model instance.

Returns:

A set of strings representing the fields that have been set, i.e. that were not filled from defaults.

classmethod model_json_schema(*by_alias: bool = True*, *ref_template: str = '#/$defs/{model}'*, *schema_generator: type[pydantic.json_schema.GenerateJsonSchema] = <class 'pydantic.json_schema.GenerateJsonSchema'>*, *mode: ~typing.Literal['validation', 'serialization'] = 'validation'*) → dict[str, Any]

Generates a JSON schema for a model class.

Args:

by_alias: Whether to use attribute aliases or not. ref_template: The reference template. schema_generator: To override the logic used to generate the JSON schema, as a subclass of

GenerateJsonSchema with your desired modifications

mode: The mode in which to generate the schema.

Returns:

The JSON schema for the given model class.

classmethod model_parametrized_name(*params: tuple[type[Any], ...]*) → str

Compute the class name for parametrizations of generic classes.

This method can be overridden to achieve a custom naming scheme for generic BaseModels.

Args:

params: Tuple of types of the class. Given a generic class *Model* with 2 type variables and a concrete model *Model[str, int]*, the value *(str, int)* would be passed to *params*.

Returns:

String representing the new class where *params* are passed to *cls* as type variables.

Raises:

TypeError: Raised when trying to generate concrete names for non-generic models.

model_post_init(*_BaseModel__context: Any*) → None

Override this method to perform additional initialization after *__init__* and *model_construct*. This is useful if you want to do some validation that requires the entire model to be initialized.

classmethod model_rebuild(*, *force: bool = False*, *raise_errors: bool = True*, *_parent_namespace_depth: int = 2*, *_types_namespace: Optional[dict[str, Any]] = None*) → bool | None

Try to rebuild the pydantic-core schema for the model.

This may be necessary when one of the annotations is a ForwardRef which could not be resolved during the initial attempt to build the schema, and automatic rebuilding fails.

Args:

force: Whether to force the rebuilding of the model schema, defaults to *False*. raise_errors: Whether to raise errors, defaults to *True*. _parent_namespace_depth: The depth level of the parent namespace, defaults to 2. _types_namespace: The types namespace, defaults to *None*.

Returns:

Returns *None* if the schema is already "complete" and rebuilding was not required. If rebuilding _was_ required, returns *True* if rebuilding was successful, otherwise *False*.

classmethod model_validate(*obj: Any*, *, *strict: bool | None = None*, *from_attributes: bool | None = None*, *context: dict[str, Any] | None = None*) → Model

Validate a pydantic model instance.

Args:

obj: The object to validate. strict: Whether to enforce types strictly. from_attributes: Whether to extract data from object attributes. context: Additional context to pass to the validator.

Raises:

ValidationError: If the object could not be validated.

Returns:

The validated model instance.

classmethod model_validate_json(*json_data: str | bytes | bytearray*, *, *strict: bool | None = None*, *context: dict[str, Any] | None = None*) → Model

Usage docs: https://docs.pydantic.dev/2.6/concepts/json/#json-parsing

Validate the given JSON data against the Pydantic model.

Args:

json_data: The JSON data to validate. strict: Whether to enforce types strictly. context: Extra variables to pass to the validator.

Returns:

The validated Pydantic model.

Raises:

ValueError: If *json_data* is not a JSON string.

classmethod model_validate_strings(*obj: Any*, *, *strict: bool | None = None*, *context: dict[str, Any] | None = None*) → Model

Validate the given object contains string data against the Pydantic model.

Args:

obj: The object contains string data to validate. strict: Whether to enforce types strictly. context: Extra variables to pass to the validator.

Returns:

The validated Pydantic model.

StockTagStatsModel

class zvt.tag.tag_models.StockTagStatsModel(*, *id: str*, *entity_id: str*, *timestamp: datetime*, *main_tag: str*, *turnover: float*, *entity_count: int*, *position: int*, *is_main_line: bool*, *main_line_continuous_days: int*, *entity_ids: List[str]*, *stock_details: List[StockTagDetailsModel]*)

Bases: `MixinModel`

__init__(**data: Any*) → None

Create a new model by parsing and validating input data from keyword arguments.

Raises [*ValidationError*][pydantic_core.ValidationError] if the input data cannot be validated to form a valid model.

self is explicitly positional-only to allow *self* as a field name.

copy(*, *include: AbstractSetIntStr | MappingIntStrAny | None = None*, *exclude: AbstractSetIntStr | MappingIntStrAny | None = None*, *update: Dict[str, Any] | None = None*, *deep: bool = False*) → Model

Returns a copy of the model.

!!! warning "Deprecated"

This method is now deprecated; use *model_copy* instead.

If you need *include* or *exclude*, use:

```
`py data = self.model_dump(include=include, exclude=exclude, round_trip=True) data = {**data, **(update or {})} copied = self.model_validate(data) `
```

Args:

include: Optional set or mapping specifying which fields to include in the copied model. exclude: Optional set or mapping

specifying which fields to exclude in the copied model. update: Optional dictionary of field-value pairs to override field values in the copied model. deep: If True, the values of fields that are Pydantic models will be deep-copied.

Returns:

A copy of the model with included, excluded and updated fields as specified.

model_computed_fields*: ClassVar[dict[str, ComputedFieldInfo]] = {}*

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

model_config*: ClassVar[ConfigDict] = {'allow_inf_nan': True, 'from_attributes': True}*

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

classmethod model_construct(*_fields_set: set[str] | None = None*, ***values: Any*) → Model

Creates a new instance of the *Model* class with validated data.

Creates a new model setting *__dict__* and *__pydantic_fields_set__* from trusted or pre-validated data. Default values are respected, but no other validation is performed. Behaves as if *Config.extra = 'allow'* was set since it adds all passed values

Args:

_fields_set: The set of field names accepted for the Model instance. values: Trusted or pre-validated data dictionary.

Returns:

A new instance of the *Model* class with validated data.

model_copy(*, *update: dict[str, Any] | None = None*, *deep: bool = False*) → Model

Usage docs: https://docs.pydantic.dev/2.6/concepts/serialization/#model_copy

Returns a copy of the model.

Args:

update: Values to change/add in the new model. Note: the data is not validated
before creating the new model. You should trust this data.

deep: Set to *True* to make a deep copy of the model.

Returns:

New model instance.

model_dump(*, *mode: Literal['json', 'python'] | str = 'python'*, *include: IncEx = None*, *exclude: IncEx = None*, *by_alias: bool = False*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*, *round_trip: bool = False*, *warnings: bool = True*) → dict[str, Any]

Usage docs: https://docs.pydantic.dev/2.6/concepts/serialization/#modelmodel_dump

Generate a dictionary representation of the model, optionally specifying which fields to include or exclude.

Args:

mode: The mode in which *to_python* should run.
If mode is 'json', the output will only contain JSON serializable types. If mode is 'python', the output may contain non-JSON-serializable Python objects.

include: A list of fields to include in the output. exclude: A list of fields to exclude from the output. by_alias: Whether to use the field's alias in the dictionary key if defined. exclude_unset: Whether to exclude fields that have not been explicitly set. exclude_defaults: Whether to exclude fields that are set to their default value. exclude_none: Whether to exclude fields that have a value of *None*. round_trip: If True, dumped values should be

valid as input for non-idempotent types such as Json[T]. warnings: Whether to log warnings when invalid fields are encountered.

Returns:

A dictionary representation of the model.

model_dump_json(*, *indent: int | None = None*, *include: IncEx = None*, *exclude: IncEx = None*, *by_alias: bool = False*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*, *round_trip: bool = False*, *warnings: bool = True*) → str

Usage docs: https://docs.pydantic.dev/2.6/concepts/serialization/#modelmodel_dump_json

Generates a JSON representation of the model using Pydantic's *to_json* method.

Args:

indent: Indentation to use in the JSON output. If None is passed, the output will be compact. include: Field(s) to include in the JSON output. exclude: Field(s) to exclude from the JSON output. by_alias: Whether to serialize using field aliases. exclude_unset: Whether to exclude fields that have not been explicitly set. exclude_defaults: Whether to exclude fields that are set to their default value. exclude_none: Whether to exclude fields that have a value of *None*. round_trip: If True, dumped values should be valid as input for non-idempotent types such as Json[T]. warnings: Whether to log warnings when invalid fields are encountered.

Returns:

A JSON string representation of the model.

property model_extra*: dict[str, Any] | None*

Get extra fields set during validation.

Returns:

A dictionary of extra fields, or *None* if *config.extra* is not set to *"allow"*.

model_fields*: ClassVar[dict[str, FieldInfo]] = {'entity_count': FieldInfo(annotation=int, required=True), 'entity_id': FieldInfo(annotation=str, required=True), 'entity_ids': FieldInfo(annotation=List[str], required=True), 'id': FieldInfo(annotation=str, required=True), 'is_main_line': FieldInfo(annotation=bool, required=True), 'main_line_continuous_days': FieldInfo(annotation=int, required=True), 'main_tag': FieldInfo(annotation=str, required=True), 'position': FieldInfo(annotation=int, required=True), 'stock_details': FieldInfo(annotation=List[StockTagDetailsModel], required=True), 'timestamp': FieldInfo(annotation=datetime, required=True), 'turnover': FieldInfo(annotation=float, required=True)}*

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

property model_fields_set*: set[str]*

Returns the set of fields that have been explicitly set on this model instance.

Returns:

A set of strings representing the fields that have been set, i.e. that were not filled from defaults.

classmethod model_json_schema(*by_alias: bool = True*, *ref_template: str = '#/$defs/{model}'*, *schema_generator: type[pydantic.json_schema.GenerateJsonSchema] = <class 'pydantic.json_schema.GenerateJsonSchema'>*, *mode: ~typing.Literal['validation', 'serialization'] = 'validation'*) → dict[str, Any]

Generates a JSON schema for a model class.

Args:

by_alias: Whether to use attribute aliases or not. ref_template: The reference template. schema_generator: To override the logic used to generate the JSON schema, as a subclass of

GenerateJsonSchema with your desired modifications

mode: The mode in which to generate the schema.

Returns:

The JSON schema for the given model class.

classmethod model_parametrized_name(*params: tuple[type[Any], ...]*) → str

Compute the class name for parametrizations of generic classes.

This method can be overridden to achieve a custom naming scheme for generic BaseModels.

Args:

params: Tuple of types of the class. Given a generic class *Model* with 2 type variables and a concrete model *Model[str, int]*, the value *(str, int)* would be passed to *params*.

Returns:

String representing the new class where *params* are passed to *cls* as type variables.

Raises:

TypeError: Raised when trying to generate concrete names for non-generic models.

model_post_init(*_BaseModel__context: Any*) → None

Override this method to perform additional initialization after *__init__* and *model_construct*. This is useful if you want to do some validation that requires the entire model to be initialized.

classmethod model_rebuild(*, *force: bool = False*, *raise_errors: bool = True*, *_parent_namespace_depth: int = 2*, *_types_namespace:*

Optional[dict[str, Any]] = None) → bool | None

Try to rebuild the pydantic-core schema for the model.

This may be necessary when one of the annotations is a ForwardRef which could not be resolved during the initial attempt to build the schema, and automatic rebuilding fails.

Args:

force: Whether to force the rebuilding of the model schema, defaults to *False*. raise_errors: Whether to raise errors, defaults to *True*. _parent_namespace_depth: The depth level of the parent namespace, defaults to 2. _types_namespace: The types namespace, defaults to *None*.

Returns:

Returns *None* if the schema is already "complete" and rebuilding was not required. If rebuilding _was_ required, returns *True* if rebuilding was successful, otherwise *False*.

classmethod model_validate(*obj: Any*, *, *strict: bool | None = None*, *from_attributes: bool | None = None*, *context: dict[str, Any] | None = None*) → Model

Validate a pydantic model instance.

Args:

obj: The object to validate. strict: Whether to enforce types strictly. from_attributes: Whether to extract data from object attributes. context: Additional context to pass to the validator.

Raises:

ValidationError: If the object could not be validated.

Returns:

The validated model instance.

classmethod model_validate_json(*json_data: str | bytes | bytearray*, *, *strict: bool | None = None*, *context: dict[str, Any] | None = None*) →

Model

Usage docs: https://docs.pydantic.dev/2.6/concepts/json/#json-parsing

Validate the given JSON data against the Pydantic model.

Args:

json_data: The JSON data to validate. strict: Whether to enforce types strictly. context: Extra variables to pass to the validator.

Returns:

The validated Pydantic model.

Raises:

ValueError: If *json_data* is not a JSON string.

classmethod model_validate_strings(*obj: Any*, *, *strict: bool | None = None*, *context: dict[str, Any] | None = None*) → Model

Validate the given object contains string data against the Pydantic model.

Args:

obj: The object contains string data to validate. strict: Whether to enforce types strictly. context: Extra variables to pass to the validator.

Returns:

The validated Pydantic model.

StockTagsModel

class zvt.tag.tag_models.StockTagsModel(*, *id: str*, *entity_id: str*, *timestamp: datetime*, *main_tag: str*, *main_tag_reason: Optional[str] = None*, *main_tags: Dict[str, str]*, *sub_tag: Optional[str]*, *sub_tag_reason: Optional[str] = None*, *sub_tags: Optional[Dict[str, str]]*, *active_hidden_tags: Optional[Dict[str, str]]*, *hidden_tags: Optional[Dict[str, str]]*, *set_by_user: bool = False*)

Bases: **MixinModel**

__init__(**data: Any*) → None

Create a new model by parsing and validating input data from keyword arguments.

Raises [*ValidationError*][pydantic_core.ValidationError] if the input data cannot be validated to form a valid model.

self is explicitly positional-only to allow *self* as a field name.

copy(*, *include: AbstractSetIntStr | MappingIntStrAny | None = None*, *exclude: AbstractSetIntStr | MappingIntStrAny | None = None*, *update: Dict[str, Any] | None = None*, *deep: bool = False*) → Model

Returns a copy of the model.

!!! warning "Deprecated"

This method is now deprecated; use *model_copy* instead.

If you need *include* or *exclude*, use:

```
`py data = self.model_dump(include=include, exclude=exclude, round_trip=True) data = {**data, **(update or {})} copied = self.model_validate(data) `
```

Args:

include: Optional set or mapping specifying which fields to include in the copied model. exclude: Optional set or mapping specifying which fields to exclude in the copied model. update: Optional dictionary of field-value pairs to override field values in the copied model. deep: If True, the values of fields that are Pydantic models will be deep-copied.

Returns:

A copy of the model with included, excluded and updated fields as specified.

model_computed_fields*: ClassVar[dict[str, ComputedFieldInfo]] = {}*

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

model_config*: ClassVar[ConfigDict] = {'allow_inf_nan': True, 'from_attributes': True}*

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

classmethod model_construct(*_fields_set: set[str] | None = None*, ***values: Any*) → Model

Creates a new instance of the *Model* class with validated data.

Creates a new model setting *__dict__* and *__pydantic_fields_set__* from trusted or pre-validated data. Default values are respected, but no other validation is performed. Behaves as if *Config.extra = 'allow'* was set since it adds all passed values

Args:

_fields_set: The set of field names accepted for the Model instance. values: Trusted or pre-validated data dictionary.

Returns:

A new instance of the *Model* class with validated data.

model_copy(*, *update: dict[str, Any] | None = None*, *deep: bool = False*) → Model

Usage docs: https://docs.pydantic.dev/2.6/concepts/serialization/#model_copy

Returns a copy of the model.

Args:

update: Values to change/add in the new model. Note: the data is not validated
before creating the new model. You should trust this data.

deep: Set to *True* to make a deep copy of the model.

Returns:

New model instance.

model_dump(*, *mode: Literal['json', 'python'] | str = 'python'*, *include: IncEx = None*, *exclude: IncEx = None*, *by_alias: bool = False*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*, *round_trip: bool = False*, *warnings: bool = True*) → dict[str, Any]

Usage docs: https://docs.pydantic.dev/2.6/concepts/serialization/#modelmodel_dump

Generate a dictionary representation of the model, optionally specifying which fields to include or exclude.

Args:

mode: The mode in which *to_python* should run.
If mode is 'json', the output will only contain JSON serializable types. If mode is 'python', the output may contain non-JSON-serializable Python objects.

include: A list of fields to include in the output. exclude: A list of fields to exclude from the output. by_alias: Whether to use the

field's alias in the dictionary key if defined. exclude_unset: Whether to exclude fields that have not been explicitly set. exclude_defaults: Whether to exclude fields that are set to their default value. exclude_none: Whether to exclude fields that have a value of *None*. round_trip: If True, dumped values should be valid as input for non-idempotent types such as Json[T]. warnings: Whether to log warnings when invalid fields are encountered.

Returns:

A dictionary representation of the model.

model_dump_json(*, *indent: int | None = None*, *include: IncEx = None*, *exclude: IncEx = None*, *by_alias: bool = False*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*, *round_trip: bool = False*, *warnings: bool = True*) → str

Usage docs: https://docs.pydantic.dev/2.6/concepts/serialization/#modelmodel_dump_json

Generates a JSON representation of the model using Pydantic's *to_json* method.

Args:

indent: Indentation to use in the JSON output. If None is passed, the output will be compact. include: Field(s) to include in the JSON output. exclude: Field(s) to exclude from the JSON output. by_alias: Whether to serialize using field aliases. exclude_unset: Whether to exclude fields that have not been explicitly set. exclude_defaults: Whether to exclude fields that are set to their default value. exclude_none: Whether to exclude fields that have a value of *None*. round_trip: If True, dumped values should be valid as input for non-idempotent types such as Json[T]. warnings: Whether to log warnings when invalid fields are encountered.

Returns:

A JSON string representation of the model.

property model_extra*: dict[str, Any] | None*

Get extra fields set during validation.

Returns:

A dictionary of extra fields, or *None* if *config.extra* is not set to *"allow"*.

model_fields*: ClassVar[dict[str, FieldInfo]] = {'active_hidden_tags': FieldInfo(annotation=Union[Dict[str, str], NoneType], required=True), 'entity_id': FieldInfo(annotation=str, required=True), 'hidden_tags': FieldInfo(annotation=Union[Dict[str, str], NoneType], required=True), 'id': FieldInfo(annotation=str, required=True), 'main_tag': FieldInfo(annotation=str, required=True), 'main_tag_reason': FieldInfo(annotation=Union[str, NoneType], required=False), 'main_tags': FieldInfo(annotation=Dict[str, str], required=True), 'set_by_user': FieldInfo(annotation=bool, required=False, default=False), 'sub_tag': FieldInfo(annotation=Union[str, NoneType], required=True), 'sub_tag_reason': FieldInfo(annotation=Union[str, NoneType], required=False), 'sub_tags': FieldInfo(annotation=Union[Dict[str, str], NoneType], required=True), 'timestamp': FieldInfo(annotation=datetime, required=True)}*

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

property model_fields_set*: set[str]*

Returns the set of fields that have been explicitly set on this model instance.

Returns:

A set of strings representing the fields that have been set, i.e. that were not filled from defaults.

classmethod model_json_schema(*by_alias: bool = True*, *ref_template: str = '#/$defs/{model}'*, *schema_generator: type[pydantic.json_schema.GenerateJsonSchema] = <class*

'pydantic.json_schema.GenerateJsonSchema'>, *mode: ~typing.Literal['validation', 'serialization'] = 'validation'*) → dict[str, Any]

Generates a JSON schema for a model class.

Args:

by_alias: Whether to use attribute aliases or not. ref_template: The reference template. schema_generator: To override the logic used to generate the JSON schema, as a subclass of

GenerateJsonSchema with your desired modifications

mode: The mode in which to generate the schema.

Returns:

The JSON schema for the given model class.

classmethod model_parametrized_name(*params: tuple[type[Any], ...]*) → str

Compute the class name for parametrizations of generic classes.

This method can be overridden to achieve a custom naming scheme for generic BaseModels.

Args:

params: Tuple of types of the class. Given a generic class *Model* with 2 type variables and a concrete model *Model[str, int]*, the value *(str, int)* would be passed to *params*.

Returns:

String representing the new class where *params* are passed to *cls* as type variables.

Raises:

TypeError: Raised when trying to generate concrete names for non-generic models.

model_post_init(*_BaseModel__context: Any*) → None

Override this method to perform additional initialization after *__init__* and *model_construct*. This is useful if you want to do some validation that requires the entire model to be initialized.

classmethod model_rebuild(*, *force: bool = False*, *raise_errors: bool = True*, *_parent_namespace_depth: int = 2*, *_types_namespace: Optional[dict[str, Any]] = None*) → bool | None

Try to rebuild the pydantic-core schema for the model.

This may be necessary when one of the annotations is a ForwardRef which could not be resolved during the initial attempt to build the schema, and automatic rebuilding fails.

Args:

force: Whether to force the rebuilding of the model schema, defaults to *False*. raise_errors: Whether to raise errors, defaults to *True*. _parent_namespace_depth: The depth level of the parent namespace, defaults to 2. _types_namespace: The types namespace, defaults to *None*.

Returns:

Returns *None* if the schema is already "complete" and rebuilding was not required. If rebuilding _was_ required, returns *True* if rebuilding was successful, otherwise *False*.

classmethod model_validate(*obj: Any*, *, *strict: bool | None = None*, *from_attributes: bool | None = None*, *context: dict[str, Any] | None = None*) → Model

Validate a pydantic model instance.

Args:

obj: The object to validate. strict: Whether to enforce types strictly. from_attributes: Whether to extract data from object attributes. context: Additional context to pass to the validator.

Raises:

ValidationError: If the object could not be validated.

Returns:
The validated model instance.

classmethod model_validate_json(*json_data: str | bytes | bytearray*, *, *strict: bool | None = None*, *context: dict[str, Any] | None = None*) → Model

Usage docs: https://docs.pydantic.dev/2.6/concepts/json/#json-parsing

Validate the given JSON data against the Pydantic model.

Args:
json_data: The JSON data to validate. strict: Whether to enforce types strictly. context: Extra variables to pass to the validator.

Returns:
The validated Pydantic model.

Raises:
ValueError: If *json_data* is not a JSON string.

classmethod model_validate_strings(*obj: Any*, *, *strict: bool | None = None*, *context: dict[str, Any] | None = None*) → Model

Validate the given object contains string data against the Pydantic model.

Args:
obj: The object contains string data to validate. strict: Whether to enforce types strictly. context: Extra variables to pass to the validator.

Returns:
The validated Pydantic model.

TagInfoModel

class zvt.tag.tag_models.TagInfoModel(*, *id: str*, *entity_id: str*, *timestamp: datetime*, *tag: str*, *tag_reason: Optional[str] = None*)

Bases: **MixinModel**

__init__(**data: Any*) → None

Create a new model by parsing and validating input data from keyword arguments.

Raises [*ValidationError*][pydantic_core.ValidationError] if the input data cannot be validated to form a valid model.

self is explicitly positional-only to allow *self* as a field name.

copy(*, *include: AbstractSetIntStr | MappingIntStrAny | None = None*, *exclude: AbstractSetIntStr | MappingIntStrAny | None = None*, *update: Dict[str, Any] | None = None*, *deep: bool = False*) → Model

Returns a copy of the model.

!!! warning "Deprecated"

This method is now deprecated; use *model_copy* instead.

If you need *include* or *exclude*, use:

```
`py data = self.model_dump(include=include, exclude=exclude, round_trip=True) data = {**data, **(update or {})} copied = self.model_validate(data) `
```

Args:

include: Optional set or mapping specifying which fields to include in the copied model. exclude: Optional set or mapping specifying which fields to exclude in the copied model. update: Optional dictionary of field-value pairs to override field values

in the copied model. deep: If True, the values of fields that are Pydantic models will be deep-copied.

Returns:

A copy of the model with included, excluded and updated fields as specified.

model_computed_fields*: ClassVar[dict[str, ComputedFieldInfo]] = {}*

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

model_config*: ClassVar[ConfigDict] = {'allow_inf_nan': True, 'from_attributes': True}*

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

classmethod model_construct(*_fields_set: set[str] | None = None*, ***values: Any*) → Model

Creates a new instance of the *Model* class with validated data.

Creates a new model setting *__dict__* and *__pydantic_fields_set__* from trusted or pre-validated data. Default values are respected, but no other validation is performed. Behaves as if *Config.extra = 'allow'* was set since it adds all passed values

Args:

_fields_set: The set of field names accepted for the Model instance. values: Trusted or pre-validated data dictionary.

Returns:

A new instance of the *Model* class with validated data.

model_copy(*, *update: dict[str, Any] | None = None*, *deep: bool = False*) → Model

Usage docs: https://docs.pydantic.dev/2.6/concepts/serialization/#model_copy

Returns a copy of the model.

Args:

update: Values to change/add in the new model. Note: the data is not validated
before creating the new model. You should trust this data.

deep: Set to *True* to make a deep copy of the model.

Returns:

New model instance.

model_dump(*, *mode: Literal['json', 'python'] | str = 'python'*, *include: IncEx = None*, *exclude: IncEx = None*, *by_alias: bool = False*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*, *round_trip: bool = False*, *warnings: bool = True*) → dict[str, Any]

Usage docs: https://docs.pydantic.dev/2.6/concepts/serialization/#modelmodel_dump

Generate a dictionary representation of the model, optionally specifying which fields to include or exclude.

Args:

mode: The mode in which *to_python* should run.
If mode is 'json', the output will only contain JSON serializable types. If mode is 'python', the output may contain non-JSON-serializable Python objects.

include: A list of fields to include in the output. exclude: A list of fields to exclude from the output. by_alias: Whether to use the field's alias in the dictionary key if defined. exclude_unset: Whether to exclude fields that have not been explicitly set. exclude_defaults: Whether to exclude fields that are set to their default value. exclude_none: Whether to exclude fields that have a value of *None*. round_trip: If True, dumped values should be valid as input for non-idempotent types such as Json[T].

warnings: Whether to log warnings when invalid fields are encountered.

Returns:

A dictionary representation of the model.

model_dump_json(*, *indent: int | None = None*, *include: IncEx = None*, *exclude: IncEx = None*, *by_alias: bool = False*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*, *round_trip: bool = False*, *warnings: bool = True*) → str

Usage docs: https://docs.pydantic.dev/2.6/concepts/serialization/#modelmodel_dump_json

Generates a JSON representation of the model using Pydantic's *to_json* method.

Args:

indent: Indentation to use in the JSON output. If None is passed, the output will be compact. include: Field(s) to include in the JSON output. exclude: Field(s) to exclude from the JSON output. by_alias: Whether to serialize using field aliases. exclude_unset: Whether to exclude fields that have not been explicitly set. exclude_defaults: Whether to exclude fields that are set to their default value. exclude_none: Whether to exclude fields that have a value of *None*. round_trip: If True, dumped values should be valid as input for non-idempotent types such as Json[T]. warnings: Whether to log warnings when invalid fields are encountered.

Returns:

A JSON string representation of the model.

property model_extra*: dict[str, Any] | None*

Get extra fields set during validation.

Returns:

A dictionary of extra fields, or *None* if *config.extra* is not set to *"allow"*.

model_fields*: ClassVar[dict[str, FieldInfo]] = {'entity_id': FieldInfo(annotation=str, required=True), 'id': FieldInfo(annotation=str, required=True), 'tag': FieldInfo(annotation=str, required=True), 'tag_reason': FieldInfo(annotation=Union[str, NoneType], required=False), 'timestamp': FieldInfo(annotation=datetime, required=True)}*

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

property model_fields_set*: set[str]*

Returns the set of fields that have been explicitly set on this model instance.

Returns:

A set of strings representing the fields that have been set, i.e. that were not filled from defaults.

classmethod model_json_schema(*by_alias: bool = True*, *ref_template: str = '#/$defs/{model}'*, *schema_generator: type[pydantic.json_schema.GenerateJsonSchema] = <class 'pydantic.json_schema.GenerateJsonSchema'>*, *mode: ~typing.Literal['validation', 'serialization'] = 'validation'*) → dict[str, Any]

Generates a JSON schema for a model class.

Args:

by_alias: Whether to use attribute aliases or not. ref_template: The reference template. schema_generator: To override the logic used to generate the JSON schema, as a subclass of

GenerateJsonSchema with your desired modifications

mode: The mode in which to generate the schema.

Returns:

The JSON schema for the given model class.

classmethod model_parametrized_name(*params: tuple[type[Any], ...]*) → str

Compute the class name for parametrizations of generic classes.

This method can be overridden to achieve a custom naming scheme for generic BaseModels.

Args:

params: Tuple of types of the class. Given a generic class *Model* with 2 type variables and a concrete model *Model[str, int]*, the value *(str, int)* would be passed to *params*.

Returns:

String representing the new class where *params* are passed to *cls* as type variables.

Raises:

TypeError: Raised when trying to generate concrete names for non-generic models.

model_post_init(*_BaseModel__context: Any*) → None

Override this method to perform additional initialization after *__init__* and *model_construct*. This is useful if you want to do some validation that requires the entire model to be initialized.

classmethod model_rebuild(*, *force: bool = False*, *raise_errors: bool = True*, *_parent_namespace_depth: int = 2*, *_types_namespace: Optional[dict[str, Any]] = None*) → bool | None

Try to rebuild the pydantic-core schema for the model.

This may be necessary when one of the annotations is a ForwardRef which could not be resolved during the initial attempt to build the schema, and automatic rebuilding fails.

Args:

force: Whether to force the rebuilding of the model schema, defaults to *False*. raise_errors: Whether to raise errors, defaults to *True*. _parent_namespace_depth: The depth level of the parent namespace, defaults to 2. _types_namespace: The types namespace, defaults to *None*.

Returns:

Returns *None* if the schema is already "complete" and rebuilding was not required. If rebuilding _was_ required, returns *True* if rebuilding was successful, otherwise *False*.

classmethod model_validate(*obj: Any*, *, *strict: bool | None = None*, *from_attributes: bool | None = None*, *context: dict[str, Any] | None = None*) → Model

Validate a pydantic model instance.

Args:

obj: The object to validate. strict: Whether to enforce types strictly. from_attributes: Whether to extract data from object attributes. context: Additional context to pass to the validator.

Raises:

ValidationError: If the object could not be validated.

Returns:

The validated model instance.

classmethod model_validate_json(*json_data: str | bytes | bytearray*, *, *strict: bool | None = None*, *context: dict[str, Any] | None = None*) → Model

Usage docs: https://docs.pydantic.dev/2.6/concepts/json/#json-parsing

Validate the given JSON data against the Pydantic model.

Args:

json_data: The JSON data to validate. strict: Whether to enforce types strictly. context: Extra variables to pass to the validator.

Returns:
The validated Pydantic model.

Raises:
ValueError: If *json_data* is not a JSON string.

classmethod model_validate_strings(*obj: Any*, *, *strict: bool | None = None*, *context: dict[str, Any] | None = None*) → Model

Validate the given object contains string data against the Pydantic model.

Args:
obj: The object contains string data to validate. strict: Whether to enforce types strictly. context: Extra variables to pass to the validator.

Returns:
The validated Pydantic model.

TagParameter

class zvt.tag.tag_models.TagParameter(*, *main_tag: str*, *main_tag_reason: Optional[str] = None*, *sub_tag: Optional[str] = None*, *sub_tag_reason: Optional[str] = None*)

Bases: `CustomModel`

__init__(**data: Any*) → None

Create a new model by parsing and validating input data from keyword arguments.

Raises [*ValidationError*][pydantic_core.ValidationError] if the input data cannot be validated to form a valid model.

self is explicitly positional-only to allow *self* as a field name.

copy(*, *include: AbstractSetIntStr | MappingIntStrAny | None = None*, *exclude: AbstractSetIntStr | MappingIntStrAny | None = None*, *update: Dict[str, Any] | None = None*, *deep: bool = False*) → Model

Returns a copy of the model.

!!! warning "Deprecated"

This method is now deprecated; use *model_copy* instead.

If you need *include* or *exclude*, use:

```
`py data = self.model_dump(include=include, exclude=exclude, round_trip=True) data = {**data, **(update or {})} copied = self.model_validate(data) `
```

Args:

include: Optional set or mapping specifying which fields to include in the copied model. exclude: Optional set or mapping specifying which fields to exclude in the copied model. update:

Optional dictionary of field-value pairs to override field values in the copied model. deep: If True, the values of fields that are Pydantic models will be deep-copied.

Returns:

A copy of the model with included, excluded and updated fields as specified.

model_computed_fields*: ClassVar[dict[str, ComputedFieldInfo]] = {}*

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

model_config*: ClassVar[ConfigDict] = {'allow_inf_nan': True, 'from_attributes': True}*

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

classmethod model_construct(*_fields_set: set[str] | None = None*, ***values: Any*) → Model

Creates a new instance of the *Model* class with validated data.

Creates a new model setting *__dict__* and *__pydantic_fields_set__* from trusted or pre-validated data. Default values are respected, but no other validation is performed. Behaves as if *Config.extra = 'allow'* was set since it adds all passed values

Args:

_fields_set: The set of field names accepted for the Model instance. values: Trusted or pre-validated data dictionary.

Returns:

A new instance of the *Model* class with validated data.

model_copy(*, *update: dict[str, Any] | None = None*, *deep: bool = False*) → Model

Usage docs: https://docs.pydantic.dev/2.6/concepts/serialization/#model_copy

Returns a copy of the model.

Args:

update: Values to change/add in the new model. Note: the data is not validated
before creating the new model. You should trust this data.

deep: Set to *True* to make a deep copy of the model.

Returns:

New model instance.

model_dump(*, *mode: Literal['json', 'python'] | str = 'python'*, *include: IncEx = None*, *exclude: IncEx = None*, *by_alias: bool = False*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*, *round_trip: bool = False*, *warnings: bool = True*) → dict[str, Any]

Usage docs: https://docs.pydantic.dev/2.6/concepts/serialization/#modelmodel_dump

Generate a dictionary representation of the model, optionally specifying which fields to include or exclude.

Args:

mode: The mode in which *to_python* should run.
If mode is 'json', the output will only contain JSON serializable types. If mode is 'python', the output may contain non-JSON-serializable Python objects.

include: A list of fields to include in the output. exclude: A list of fields to exclude from the output. by_alias: Whether to use the field's alias in the dictionary key if defined. exclude_unset: Whether to exclude fields that have not been explicitly set. exclude_defaults: Whether to exclude fields that are set to their default value. exclude_none: Whether to exclude fields that have a value of *None*. round_trip: If True, dumped values should be

valid as input for non-idempotent types such as Json[T]. warnings: Whether to log warnings when invalid fields are encountered.

Returns:

A dictionary representation of the model.

model_dump_json(*, *indent: int | None = None*, *include: IncEx = None*, *exclude: IncEx = None*, *by_alias: bool = False*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*, *round_trip: bool = False*, *warnings: bool = True*) → str

Usage docs: https://docs.pydantic.dev/2.6/concepts/serialization/#modelmodel_dump_json

Generates a JSON representation of the model using Pydantic's *to_json* method.

Args:

indent: Indentation to use in the JSON output. If None is passed, the output will be compact. include: Field(s) to include in the JSON output. exclude: Field(s) to exclude from the JSON output. by_alias: Whether to serialize using field aliases. exclude_unset: Whether to exclude fields that have not been explicitly set. exclude_defaults: Whether to exclude fields that are set to their default value. exclude_none: Whether to exclude fields that have a value of *None*. round_trip: If True, dumped values should be valid as input for non-idempotent types such as Json[T]. warnings: Whether to log warnings when invalid fields are encountered.

Returns:

A JSON string representation of the model.

property model_extra*: dict[str, Any] | None*

Get extra fields set during validation.

Returns:

A dictionary of extra fields, or *None* if *config.extra* is not set to *"allow"*.

model_fields*: ClassVar[dict[str, FieldInfo]] = {'main_tag': FieldInfo(annotation=str, required=True), 'main_tag_reason': FieldInfo(annotation=Union[str, NoneType], required=False), 'sub_tag': FieldInfo(annotation=Union[str, NoneType], required=False), 'sub_tag_reason': FieldInfo(annotation=Union[str, NoneType], required=False)}*

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

property model_fields_set*: set[str]*

Returns the set of fields that have been explicitly set on this model instance.

Returns:

A set of strings representing the fields that have been set, i.e. that were not filled from defaults.

classmethod model_json_schema(*by_alias: bool = True*, *ref_template: str = '#/$defs/{model}'*, *schema_generator: type[pydantic.json_schema.GenerateJsonSchema] = <class 'pydantic.json_schema.GenerateJsonSchema'>*, *mode: ~typing.Literal['validation', 'serialization'] = 'validation'*) → dict[str, Any]

Generates a JSON schema for a model class.

Args:

by_alias: Whether to use attribute aliases or not. ref_template: The reference template. schema_generator: To override the logic used to generate the JSON schema, as a subclass of

GenerateJsonSchema with your desired modifications

mode: The mode in which to generate the schema.

Returns:

The JSON schema for the given model class.

classmethod model_parametrized_name(*params: tuple[type[Any], ...]*) → str

Compute the class name for parametrizations of generic classes.

This method can be overridden to achieve a custom naming scheme for generic BaseModels.

Args:

params: Tuple of types of the class. Given a generic class *Model* with 2 type variables and a concrete model *Model[str, int]*, the value *(str, int)* would be passed to *params*.

Returns:

String representing the new class where *params* are passed to *cls* as type variables.

Raises:

TypeError: Raised when trying to generate concrete names for non-generic models.

model_post_init(*_BaseModel__context: Any*) → None

Override this method to perform additional initialization after *__init__* and *model_construct*. This is useful if you want to do some validation that requires the entire model to be initialized.

classmethod model_rebuild(*, *force: bool = False*, *raise_errors: bool = True*, *_parent_namespace_depth: int = 2*, *_types_namespace: Optional[dict[str, Any]] = None*) → bool | None

Try to rebuild the pydantic-core schema for the model.

This may be necessary when one of the annotations is a ForwardRef which could not be resolved during the initial attempt to build the schema, and automatic rebuilding fails.

Args:

force: Whether to force the rebuilding of the model schema, defaults to *False*. raise_errors: Whether to raise errors, defaults to *True*. _parent_namespace_depth: The depth level of the parent namespace, defaults to 2. _types_namespace: The types namespace, defaults to *None*.

Returns:

Returns *None* if the schema is already "complete" and rebuilding was not required. If rebuilding _was_ required, returns *True* if rebuilding was successful, otherwise *False*.

classmethod model_validate(*obj: Any*, *, *strict: bool | None = None*, *from_attributes: bool | None = None*, *context: dict[str, Any] | None = None*) → Model

Validate a pydantic model instance.

Args:

obj: The object to validate. strict: Whether to enforce types strictly. from_attributes: Whether to extract data from object attributes. context: Additional context to pass to the validator.

Raises:

ValidationError: If the object could not be validated.

Returns:

The validated model instance.

classmethod model_validate_json(*json_data: str | bytes | bytearray*, *, *strict: bool | None = None*, *context: dict[str, Any] | None = None*) → Model

Usage docs: https://docs.pydantic.dev/2.6/concepts/json/#json-parsing

Validate the given JSON data against the Pydantic model.

Args:

json_data: The JSON data to validate. strict: Whether to enforce types strictly. context: Extra variables to pass to the validator.

Returns:
The validated Pydantic model.

Raises:
ValueError: If *json_data* is not a JSON string.

classmethod model_validate_strings(*obj: Any*, *, *strict: bool | None = None*, *context: dict[str, Any] | None = None*) → Model

Validate the given object contains string data against the Pydantic model.

Args:
obj: The object contains string data to validate. strict: Whether to enforce types strictly. context: Extra variables to pass to the validator.

Returns:
The validated Pydantic model.

zvt.tag.tag_schemas

Classes

HiddenTagInfo(**kwargs)	
MainTagInfo(**kwargs)	
StockPoolInfo(**kwargs)	
StockPools(**kwargs)	
StockSystemTags(**kwargs)	
StockTags(**kwargs)	Schema for storing stock tags
SubTagInfo(**kwargs)	
TagStats(**kwargs)	

HiddenTagInfo

class zvt.tag.tag_schemas.HiddenTagInfo(***kwargs*)

Bases: **Base**, **Mixin**

__init__(***kwargs*)

A simple constructor that allows initialization from kwargs.

Sets attributes on the constructed instance using the names and values in `kwargs`.

Only keys that are present as attributes of the instance's class are allowed. These could be, for example, any mapped columns or relationships.

entity_id

entity id

classmethod get_providers() → List[str]

providers of the schema defined by cls

Returns:: providers

classmethod get_storages(*provider: Optional[str] = None*)

get the storages info

Parameters:: **provider** – provider
Returns:: storages

id

id

classmethod query_data(*provider_index: int = 0*, *ids: Optional[List[str]] = None*, *entity_ids: Optional[List[str]] = None*, *entity_id: Optional[str] = None*, *codes: Optional[List[str]] = None*, *code: Optional[str] = None*, *level: Optional[Union[IntervalLevel, str]] = None*, *provider:*

Optional[str] = None, *columns: Optional[List] = None*, *col_label: Optional[dict] = None*, *return_type: str = 'df'*, *start_timestamp: Optional[Union[Timestamp, str]] = None*, *end_timestamp: Optional[Union[Timestamp, str]] = None*, *filters: Optional[List] = None*, *session: Optional[Session] = None*, *order=None*, *limit: Optional[int] = None*, *distinct=None*, *index: Optional[Union[str, list]] = None*, *drop_index_col=False*, *time_field: str = 'timestamp'*)

query data by the arguments

Parameters::

- **provider_index** –
- **data_schema** –
- **ids** –
- **entity_ids** –
- **entity_id** –
- **codes** –
- **code** –
- **level** –
- **provider** –
- **columns** –
- **col_label** – dict with key(column), value(label
- **return_type** – df, domain or dict. default is df
- **start_timestamp** –
- **end_timestamp** –
- **filters** –
- **session** –
- **order** –
- **limit** –
- **index** – index field name, str for single index, str list for multiple index
- **drop_index_col** – whether drop the col if it's in index, default False
- **time_field** –

Returns:: results basing on return_type.

classmethod record_data(*provider_index: int = 0*, *provider: Optional[str] = None*, *force_update=None*, *sleeping_time=None*, *exchanges=None*, *entity_id=None*, *entity_ids=None*, *code=None*, *codes=None*,

real_time=None, *fix_duplicate_way=None*, *start_timestamp=None*, *end_timestamp=None*, *one_day_trading_minutes=None*, ***kwargs*)

record data by the arguments

Parameters::

- **entity_id** –
- **provider_index** –
- **provider** –
- **force_update** –
- **sleeping_time** –
- **exchanges** –
- **entity_ids** –
- **code** –
- **codes** –
- **real_time** –
- **fix_duplicate_way** –
- **start_timestamp** –
- **end_timestamp** –
- **one_day_trading_minutes** –
- **kwargs** –

Returns::

classmethod register_provider(*provider*)

register the provider to the schema defined by cls

Parameters:: **provider** –

classmethod register_recorder_cls(*provider*, *recorder_cls*)

register the recorder for the schema

Parameters::

- **provider** –
- **recorder_cls** –

timestamp

the meaning could be different for different case,most time it means 'happen time'

MainTagInfo

class zvt.tag.tag_schemas.MainTagInfo(***kwargs*)

Bases: **`Base`**, **`Mixin`**

__init__(***kwargs*)

A simple constructor that allows initialization from kwargs.

Sets attributes on the constructed instance using the names and values in `kwargs`.

Only keys that are present as attributes of the instance's class are allowed. These could be, for example, any mapped columns or relationships.

entity_id

entity id

classmethod get_providers() → List[str]

providers of the schema defined by cls

Returns:: providers

classmethod get_storages(*provider: Optional[str] = None*)

get the storages info

Parameters:: **provider** – provider
Returns:: storages

id

id

classmethod query_data(*provider_index: int = 0*, *ids: Optional[List[str]] = None*, *entity_ids: Optional[List[str]] = None*, *entity_id: Optional[str] = None*, *codes: Optional[List[str]] = None*, *code: Optional[str] = None*, *level: Optional[Union[IntervalLevel, str]] = None*, *provider:*

Optional[str] = None, *columns: Optional[List] = None*, *col_label: Optional[dict] = None*, *return_type: str = 'df'*, *start_timestamp: Optional[Union[Timestamp, str]] = None*, *end_timestamp: Optional[Union[Timestamp, str]] = None*, *filters: Optional[List] = None*, *session: Optional[Session] = None*, *order=None*, *limit: Optional[int] = None*, *distinct=None*, *index: Optional[Union[str, list]] = None*, *drop_index_col=False*, *time_field: str = 'timestamp'*)

query data by the arguments

Parameters::

- **provider_index** –
- **data_schema** –
- **ids** –
- **entity_ids** –
- **entity_id** –
- **codes** –
- **code** –
- **level** –
- **provider** –
- **columns** –
- **col_label** – dict with key(column), value(label
- **return_type** – df, domain or dict. default is df
- **start_timestamp** –
- **end_timestamp** –
- **filters** –
- **session** –
- **order** –
- **limit** –
- **index** – index field name, str for single index, str list for multiple index
- **drop_index_col** – whether drop the col if it's in index, default False
- **time_field** –

Returns:: results basing on return_type.

classmethod record_data(*provider_index: int = 0*, *provider: Optional[str] = None*, *force_update=None*, *sleeping_time=None*, *exchanges=None*, *entity_id=None*, *entity_ids=None*, *code=None*, *codes=None*,

real_time=None, *fix_duplicate_way=None*, *start_timestamp=None*, *end_timestamp=None*, *one_day_trading_minutes=None*, ***kwargs*)

record data by the arguments

Parameters::

- **entity_id** –
- **provider_index** –
- **provider** –
- **force_update** –
- **sleeping_time** –
- **exchanges** –
- **entity_ids** –
- **code** –
- **codes** –
- **real_time** –
- **fix_duplicate_way** –
- **start_timestamp** –
- **end_timestamp** –
- **one_day_trading_minutes** –
- **kwargs** –

Returns::

classmethod register_provider(*provider*)

register the provider to the schema defined by cls

Parameters:: **provider** –

classmethod register_recorder_cls(*provider*, *recorder_cls*)

register the recorder for the schema

Parameters::

- **provider** –
- **recorder_cls** –

timestamp

the meaning could be different for different case,most time it means 'happen time'

StockPoolInfo

class zvt.tag.tag_schemas.StockPoolInfo(***kwargs*)

Bases: **Base**, **Mixin**

__init__(***kwargs*)

A simple constructor that allows initialization from kwargs.

Sets attributes on the constructed instance using the names and values in `kwargs`.

Only keys that are present as attributes of the instance's class are allowed. These could be, for example, any mapped columns or relationships.

entity_id

entity id

classmethod get_providers() → List[str]

providers of the schema defined by cls

Returns:: providers

classmethod get_storages(*provider: Optional[str] = None*)

get the storages info

Parameters:: **provider** – provider
Returns:: storages

id

id

classmethod query_data(*provider_index: int = 0*, *ids: Optional[List[str]] = None*, *entity_ids: Optional[List[str]] = None*, *entity_id: Optional[str] = None*, *codes: Optional[List[str]] = None*, *code: Optional[str] = None*, *level: Optional[Union[IntervalLevel, str]] = None*, *provider:*

Optional[str] = None, *columns: Optional[List] = None*, *col_label: Optional[dict] = None*, *return_type: str = 'df'*, *start_timestamp: Optional[Union[Timestamp, str]] = None*, *end_timestamp: Optional[Union[Timestamp, str]] = None*, *filters: Optional[List] = None*, *session: Optional[Session] = None*, *order=None*, *limit: Optional[int] = None*, *distinct=None*, *index: Optional[Union[str, list]] = None*, *drop_index_col=False*, *time_field: str = 'timestamp'*)

query data by the arguments

Parameters::

- **provider_index** –
- **data_schema** –
- **ids** –
- **entity_ids** –
- **entity_id** –
- **codes** –
- **code** –
- **level** –
- **provider** –
- **columns** –
- **col_label** – dict with key(column), value(label
- **return_type** – df, domain or dict. default is df
- **start_timestamp** –
- **end_timestamp** –
- **filters** –
- **session** –
- **order** –
- **limit** –
- **index** – index field name, str for single index, str list for multiple index
- **drop_index_col** – whether drop the col if it's in index, default False
- **time_field** –

Returns:: results basing on return_type.

classmethod record_data(*provider_index: int = 0*, *provider: Optional[str] = None*, *force_update=None*, *sleeping_time=None*, *exchanges=None*, *entity_id=None*, *entity_ids=None*, *code=None*, *codes=None*,

real_time=None, *fix_duplicate_way=None*, *start_timestamp=None*, *end_timestamp=None*, *one_day_trading_minutes=None*, ***kwargs*)

record data by the arguments

Parameters::

- **entity_id** –
- **provider_index** –
- **provider** –
- **force_update** –
- **sleeping_time** –
- **exchanges** –
- **entity_ids** –
- **code** –
- **codes** –
- **real_time** –
- **fix_duplicate_way** –
- **start_timestamp** –
- **end_timestamp** –
- **one_day_trading_minutes** –
- **kwargs** –

Returns::

classmethod register_provider(*provider*)

register the provider to the schema defined by cls

Parameters:: **provider** –

classmethod register_recorder_cls(*provider*, *recorder_cls*)

register the recorder for the schema

Parameters::

- **provider** –
- **recorder_cls** –

timestamp

the meaning could be different for different case,most time it means 'happen time'

StockPools

class zvt.tag.tag_schemas.StockPools(**kwargs*)

Bases: **Base**, **Mixin**

__init__(**kwargs*)

A simple constructor that allows initialization from kwargs.

Sets attributes on the constructed instance using the names and values in `kwargs`.

Only keys that are present as attributes of the instance's class are allowed. These could be, for example, any mapped columns or relationships.

entity_id

entity id

classmethod get_providers() → List[str]

providers of the schema defined by cls

Returns:: providers

classmethod get_storages(*provider: Optional[str] = None*)

get the storages info

Parameters:: **provider** – provider
Returns:: storages

id

id

classmethod query_data(*provider_index: int = 0*, *ids: Optional[List[str]] = None*, *entity_ids: Optional[List[str]] = None*, *entity_id: Optional[str] = None*, *codes: Optional[List[str]] = None*, *code: Optional[str] = None*, *level: Optional[Union[IntervalLevel, str]] = None*, *provider:*

Optional[str] = None, *columns: Optional[List] = None*, *col_label: Optional[dict] = None*, *return_type: str = 'df'*, *start_timestamp: Optional[Union[Timestamp, str]] = None*, *end_timestamp: Optional[Union[Timestamp, str]] = None*, *filters: Optional[List] = None*, *session: Optional[Session] = None*, *order=None*, *limit: Optional[int] = None*, *distinct=None*, *index: Optional[Union[str, list]] = None*, *drop_index_col=False*, *time_field: str = 'timestamp'*)

query data by the arguments

Parameters::

- **provider_index** –
- **data_schema** –
- **ids** –
- **entity_ids** –
- **entity_id** –
- **codes** –
- **code** –
- **level** –
- **provider** –
- **columns** –
- **col_label** – dict with key(column), value(label
- **return_type** – df, domain or dict. default is df
- **start_timestamp** –
- **end_timestamp** –
- **filters** –
- **session** –
- **order** –
- **limit** –
- **index** – index field name, str for single index, str list for multiple index
- **drop_index_col** – whether drop the col if it's in index, default False
- **time_field** –

Returns:: results basing on return_type.

classmethod record_data(*provider_index: int = 0*, *provider: Optional[str] = None*, *force_update=None*, *sleeping_time=None*, *exchanges=None*, *entity_id=None*, *entity_ids=None*, *code=None*, *codes=None*,

real_time=None, *fix_duplicate_way=None*, *start_timestamp=None*, *end_timestamp=None*, *one_day_trading_minutes=None*, ***kwargs*)

record data by the arguments

Parameters::

- **entity_id** –
- **provider_index** –
- **provider** –
- **force_update** –
- **sleeping_time** –
- **exchanges** –
- **entity_ids** –
- **code** –
- **codes** –
- **real_time** –
- **fix_duplicate_way** –
- **start_timestamp** –
- **end_timestamp** –
- **one_day_trading_minutes** –
- **kwargs** –

Returns::

classmethod register_provider(*provider*)

register the provider to the schema defined by cls

Parameters:: **provider** –

classmethod register_recorder_cls(*provider*, *recorder_cls*)

register the recorder for the schema

Parameters::

- **provider** –
- **recorder_cls** –

timestamp

the meaning could be different for different case,most time it means 'happen time'

StockSystemTags

class zvt.tag.tag_schemas.StockSystemTags(**kwargs)

Bases: **Base**, **Mixin**

__init__(**kwargs)

A simple constructor that allows initialization from kwargs.

Sets attributes on the constructed instance using the names and values in `kwargs`.

Only keys that are present as attributes of the instance's class are allowed. These could be, for example, any mapped columns or relationships.

code

编码

name

名字

recent_reduction

减持

recent_acquisition

增持

recent_unlock

解禁

recent_additional_or_rights_issue

增发配股

recent_positive_earnings_news

业绩利好

recent_negative_earnings_news
业绩利空

recent_dragon_and_tiger_count
上榜次数

recent_violation_alert
违规行为

recent_positive_news
利好

recent_negative_news
利空

recent_news_summary
新闻总结

entity_id
entity id

classmethod get_providers() → List[str]
providers of the schema defined by cls

Returns:: providers

classmethod get_storages(*provider: Optional[str] = None*)
get the storages info

Parameters:: **provider** – provider
Returns:: storages

id
id

classmethod query_data(*provider_index: int = 0*, *ids: Optional[List[str]] = None*, *entity_ids: Optional[List[str]] = None*, *entity_id: Optional[str] = None*, *codes: Optional[List[str]] = None*, *code: Optional[str] = None*, *level: Optional[Union[IntervalLevel, str]] = None*, *provider: Optional[str] = None*, *columns: Optional[List] = None*, *col_label: Optional[dict] = None*, *return_type: str = 'df'*, *start_timestamp: Optional[Union[Timestamp, str]] = None*, *end_timestamp: Optional[Union[Timestamp, str]] = None*, *filters: Optional[List] = None*, *session: Optional[Session] = None*, *order=None*, *limit: Optional[int] = None*, *distinct=None*, *index: Optional[Union[str, list]] = None*, *drop_index_col=False*, *time_field: str = 'timestamp'*)

query data by the arguments

Parameters::

- **provider_index** –
- **data_schema** –
- **ids** –
- **entity_ids** –
- **entity_id** –
- **codes** –
- **code** –
- **level** –
- **provider** –
- **columns** –
- **col_label** – dict with key(column), value(label
- **return_type** – df, domain or dict. default is df
- **start_timestamp** –
- **end_timestamp** –
- **filters** –
- **session** –
- **order** –
- **limit** –
- **index** – index field name, str for single index, str list for multiple index
- **drop_index_col** – whether drop the col if it's in index, default False
- **time_field** –

Returns:: results basing on return_type.

classmethod record_data(*provider_index: int = 0*, *provider: Optional[str] = None*, *force_update=None*, *sleeping_time=None*, *exchanges=None*, *entity_id=None*, *entity_ids=None*, *code=None*, *codes=None*, *real_time=None*, *fix_duplicate_way=None*, *start_timestamp=None*, *end_timestamp=None*, *one_day_trading_minutes=None*, ***kwargs*)

record data by the arguments

Parameters::

- **entity_id** –
- **provider_index** –
- **provider** –
- **force_update** –
- **sleeping_time** –
- **exchanges** –
- **entity_ids** –
- **code** –
- **codes** –
- **real_time** –
- **fix_duplicate_way** –
- **start_timestamp** –
- **end_timestamp** –
- **one_day_trading_minutes** –
- **kwargs** –

Returns::

classmethod register_provider(*provider*)

register the provider to the schema defined by cls

Parameters:: **provider** –

classmethod register_recorder_cls(*provider*, *recorder_cls*)

register the recorder for the schema

Parameters::

- **provider** –
- **recorder_cls** –

timestamp

the meaning could be different for different case,most time it means 'happen time'

StockTags

class zvt.tag.tag_schemas.StockTags(**kwargs*)

Bases: **Base**, **Mixin**

Schema for storing stock tags

__init__(**kwargs*)

A simple constructor that allows initialization from kwargs.

Sets attributes on the constructed instance using the names and values in `kwargs`.

Only keys that are present as attributes of the instance's class are allowed. These could be, for example, any mapped columns or relationships.

entity_id

entity id

classmethod get_providers() → List[str]

providers of the schema defined by cls

Returns:: providers

classmethod get_storages(*provider: Optional[str] = None*)

get the storages info

Parameters:: **provider** – provider
Returns:: storages

id

id

classmethod query_data(*provider_index: int = 0*, *ids: Optional[List[str]] = None*, *entity_ids: Optional[List[str]] = None*, *entity_id: Optional[str]*

= None, *codes: Optional[List[str]] = None*, *code: Optional[str] = None*, *level: Optional[Union[IntervalLevel, str]] = None*, *provider: Optional[str] = None*, *columns: Optional[List] = None*, *col_label: Optional[dict] = None*, *return_type: str = 'df'*, *start_timestamp: Optional[Union[Timestamp, str]] = None*, *end_timestamp: Optional[Union[Timestamp, str]] = None*, *filters: Optional[List] = None*, *session: Optional[Session] = None*, *order=None*, *limit: Optional[int] = None*, *distinct=None*, *index: Optional[Union[str, list]] = None*, *drop_index_col=False*, *time_field: str = 'timestamp'*)

query data by the arguments

Parameters::

- **provider_index** –
- **data_schema** –
- **ids** –
- **entity_ids** –
- **entity_id** –
- **codes** –
- **code** –
- **level** –
- **provider** –
- **columns** –
- **col_label** – dict with key(column), value(label
- **return_type** – df, domain or dict. default is df
- **start_timestamp** –
- **end_timestamp** –
- **filters** –
- **session** –
- **order** –
- **limit** –
- **index** – index field name, str for single index, str list for multiple index
- **drop_index_col** – whether drop the col if it's in index, default False
- **time_field** –

Returns:: results basing on return_type.

classmethod record_data(*provider_index: int = 0*, *provider: Optional[str] = None*, *force_update=None*, *sleeping_time=None*, *exchanges=None*,

entity_id=None, *entity_ids=None*, *code=None*, *codes=None*, *real_time=None*, *fix_duplicate_way=None*, *start_timestamp=None*, *end_timestamp=None*, *one_day_trading_minutes=None*, ***kwargs*)

record data by the arguments

Parameters::

- **entity_id** –
- **provider_index** –
- **provider** –
- **force_update** –
- **sleeping_time** –
- **exchanges** –
- **entity_ids** –
- **code** –
- **codes** –
- **real_time** –
- **fix_duplicate_way** –
- **start_timestamp** –
- **end_timestamp** –
- **one_day_trading_minutes** –
- **kwargs** –

Returns::

classmethod register_provider(*provider*)

register the provider to the schema defined by cls

Parameters:: **provider** –

classmethod register_recorder_cls(*provider*, *recorder_cls*)

register the recorder for the schema

Parameters::

- **provider** –
- **recorder_cls** –

timestamp

the meaning could be different for different case,most time it means 'happen time'

SubTagInfo

class zvt.tag.tag_schemas.SubTagInfo(***kwargs*)

Bases: **`Base`**, **`Mixin`**

__init__(***kwargs*)

A simple constructor that allows initialization from kwargs.

Sets attributes on the constructed instance using the names and values in `kwargs`.

Only keys that are present as attributes of the instance's class are allowed. These could be, for example, any mapped columns or relationships.

entity_id

entity id

classmethod get_providers() → List[str]

providers of the schema defined by cls

Returns:: providers

classmethod get_storages(*provider: Optional[str] = None*)

get the storages info

Parameters:: **provider** – provider
Returns:: storages

id

id

classmethod query_data(*provider_index: int = 0*, *ids: Optional[List[str]] = None*, *entity_ids: Optional[List[str]] = None*, *entity_id: Optional[str] = None*, *codes: Optional[List[str]] = None*, *code: Optional[str] = None*, *level: Optional[Union[IntervalLevel, str]] = None*, *provider:*

Optional[str] = None, *columns: Optional[List] = None*, *col_label: Optional[dict] = None*, *return_type: str = 'df'*, *start_timestamp: Optional[Union[Timestamp, str]] = None*, *end_timestamp: Optional[Union[Timestamp, str]] = None*, *filters: Optional[List] = None*, *session: Optional[Session] = None*, *order=None*, *limit: Optional[int] = None*, *distinct=None*, *index: Optional[Union[str, list]] = None*, *drop_index_col=False*, *time_field: str = 'timestamp'*)

query data by the arguments

Parameters::

- **provider_index** –
- **data_schema** –
- **ids** –
- **entity_ids** –
- **entity_id** –
- **codes** –
- **code** –
- **level** –
- **provider** –
- **columns** –
- **col_label** – dict with key(column), value(label
- **return_type** – df, domain or dict. default is df
- **start_timestamp** –
- **end_timestamp** –
- **filters** –
- **session** –
- **order** –
- **limit** –
- **index** – index field name, str for single index, str list for multiple index
- **drop_index_col** – whether drop the col if it's in index, default False
- **time_field** –

Returns:: results basing on return_type.

classmethod record_data(*provider_index: int = 0*, *provider: Optional[str] = None*, *force_update=None*, *sleeping_time=None*, *exchanges=None*, *entity_id=None*, *entity_ids=None*, *code=None*, *codes=None*,

real_time=None, *fix_duplicate_way=None*, *start_timestamp=None*, *end_timestamp=None*, *one_day_trading_minutes=None*, ***kwargs*)

record data by the arguments

Parameters::

- **entity_id** –
- **provider_index** –
- **provider** –
- **force_update** –
- **sleeping_time** –
- **exchanges** –
- **entity_ids** –
- **code** –
- **codes** –
- **real_time** –
- **fix_duplicate_way** –
- **start_timestamp** –
- **end_timestamp** –
- **one_day_trading_minutes** –
- **kwargs** –

Returns::

classmethod register_provider(*provider*)

register the provider to the schema defined by cls

Parameters:: **provider** –

classmethod register_recorder_cls(*provider*, *recorder_cls*)

register the recorder for the schema

Parameters::

- **provider** –
- **recorder_cls** –

timestamp

the meaning could be different for different case,most time it means 'happen time'

TagStats

class zvt.tag.tag_schemas.TagStats(***kwargs*)

Bases: **`Base`**, **`Mixin`**

__init__(***kwargs*)

A simple constructor that allows initialization from kwargs.

Sets attributes on the constructed instance using the names and values in `kwargs`.

Only keys that are present as attributes of the instance's class are allowed. These could be, for example, any mapped columns or relationships.

entity_id

entity id

classmethod get_providers() → List[str]

providers of the schema defined by cls

Returns:: providers

classmethod get_storages(*provider: Optional[str] = None*)

get the storages info

Parameters:: **provider** – provider
Returns:: storages

id

id

classmethod query_data(*provider_index: int = 0*, *ids: Optional[List[str]] = None*, *entity_ids: Optional[List[str]] = None*, *entity_id: Optional[str] = None*, *codes: Optional[List[str]] = None*, *code: Optional[str] = None*, *level: Optional[Union[IntervalLevel, str]] = None*, *provider:*

Optional[str] = None, *columns: Optional[List] = None*, *col_label: Optional[dict] = None*, *return_type: str = 'df'*, *start_timestamp: Optional[Union[Timestamp, str]] = None*, *end_timestamp: Optional[Union[Timestamp, str]] = None*, *filters: Optional[List] = None*, *session: Optional[Session] = None*, *order=None*, *limit: Optional[int] = None*, *distinct=None*, *index: Optional[Union[str, list]] = None*, *drop_index_col=False*, *time_field: str = 'timestamp'*)

query data by the arguments

Parameters::

- **provider_index** –
- **data_schema** –
- **ids** –
- **entity_ids** –
- **entity_id** –
- **codes** –
- **code** –
- **level** –
- **provider** –
- **columns** –
- **col_label** – dict with key(column), value(label
- **return_type** – df, domain or dict. default is df
- **start_timestamp** –
- **end_timestamp** –
- **filters** –
- **session** –
- **order** –
- **limit** –
- **index** – index field name, str for single index, str list for multiple index
- **drop_index_col** – whether drop the col if it's in index, default False
- **time_field** –

Returns:: results basing on return_type.

classmethod record_data(*provider_index: int = 0*, *provider: Optional[str] = None*, *force_update=None*, *sleeping_time=None*, *exchanges=None*, *entity_id=None*, *entity_ids=None*, *code=None*, *codes=None*,

real_time=None, *fix_duplicate_way=None*, *start_timestamp=None*, *end_timestamp=None*, *one_day_trading_minutes=None*, ***kwargs*)

record data by the arguments

Parameters::

- **entity_id** –
- **provider_index** –
- **provider** –
- **force_update** –
- **sleeping_time** –
- **exchanges** –
- **entity_ids** –
- **code** –
- **codes** –
- **real_time** –
- **fix_duplicate_way** –
- **start_timestamp** –
- **end_timestamp** –
- **one_day_trading_minutes** –
- **kwargs** –

Returns::

classmethod register_provider(*provider*)

register the provider to the schema defined by cls

Parameters:: **provider** –

classmethod register_recorder_cls(*provider*, *recorder_cls*)

register the recorder for the schema

Parameters::

- **provider** –
- **recorder_cls** –

timestamp

the meaning could be different for different case,most time it means 'happen time'

zvt.tag.tag_service

Functions

activate_main_tag_by_industry(industry)

---

activate_main_tag_by_sub_tags(...)

---

batch_set_stock_tags(batch_set_stock_tags_model)

---

build_stock_pool(create_stock_pools_model, ...)

---

build_stock_pool_info(...)

---

build_stock_tags(set_stock_tags_model, ...)

---

build_tag_parameter(tag_type, tag, ...)

---

query_stock_tag_stats(...)

---

refresh_all_main_tag_by_sub_tag()

---

refresh_main_tag_by_sub_tag(stock_tag)

---

stock_tags_need_update(stock_tags, ...)

Classes

StockAutoTagger([force])

zvt.tag.tag_service.activate_main_tag_by_industry

zvt.tag.tag_service.activate_main_tag_by_industry(*industry: str*)

zvt.tag.tag_service.activate_main_tag_by_sub_tags

zvt.tag.tag_service.activate_main_tag_by_sub_tags(*activate_sub_tags_model: ActivateSubTagsModel*)

zvt.tag.tag_service.batch_set_stock_tags

zvt.tag.tag_service.batch_set_stock_tags(*batch_set_stock_tags_model: BatchSetStockTagsModel*)

zvt.tag.tag_service.build_stock_pool

zvt.tag.tag_service.build_stock_pool(*create_stock_pools_model: CreateStockPoolsModel*, *target_date*)

zvt.tag.tag_service.build_stock_pool_info

zvt.tag.tag_service.build_stock_pool_info(*create_stock_pool_info_model: CreateStockPoolInfoModel*, *timestamp*)

zvt.tag.tag_service.build_stock_tags

zvt.tag.tag_service.build_stock_tags(*set_stock_tags_model: SetStockTagsModel*, *timestamp: Timestamp*, *set_by_user: bool*, *keep_current=False*)

zvt.tag.tag_service.build_tag_parameter

zvt.tag.tag_service.build_tag_parameter(*tag_type: TagType*, *tag*, *tag_reason*, *stock_tag: StockTags*)

zvt.tag.tag_service.query_stock_tag_stats

zvt.tag.tag_service.query_stock_tag_stats(*query_stock_tag_stats_model: QueryStockTagStatsModel*)

zvt.tag.tag_service.refresh_all_main_tag_by_sub_tag

zvt.tag.tag_service.refresh_all_main_tag_by_sub_tag()

zvt.tag.tag_service.refresh_main_tag_by_sub_tag

zvt.tag.tag_service.refresh_main_tag_by_sub_tag(*stock_tag: StockTags*) → StockTags

zvt.tag.tag_service.stock_tags_need_update

zvt.tag.tag_service.stock_tags_need_update(*stock_tags: StockTags*, *set_stock_tags_model: SetStockTagsModel*)

StockAutoTagger

class zvt.tag.tag_service.StockAutoTagger(*force=False*)

Bases: `StockTagger`

__init__(*force=False*) → None

clear_state_data(*entity_id=None*)

clear state of the entity

Parameters:: **entity_id** – entity id

data_schema

alias of `StockTags`

decode_state(*state: str*)

decode state

Parameters:: **state** –
Returns::

encode_state(*state: object*)

encode state

Parameters:: **state** –
Returns::

entity_schema

alias of `Stock`

name = *None*

name of the service, default name of class if not set manually

state_schema

alias of `TaggerState`

zvt.tag.tag_stats

Functions

`build_stock_pool_tag_stats`(stock_pool_name)

`build_system_stock_pools`()

---

zvt.tag.tag_stats.build_stock_pool_tag_stats

zvt.tag.tag_stats.build_stock_pool_tag_stats(*stock_pool_name*, *force_rebuild_latest=False*)

zvt.tag.tag_stats.build_system_stock_pools

zvt.tag.tag_stats.build_system_stock_pools()

zvt.tag.tag_utils

Functions

build_initial_hidden_tag_info()

---

build_initial_main_tag_info()

---

build_initial_stock_pool_info()

---

build_initial_sub_tag_info()

---

check_missed_concept()

---

check_missed_industry()

---

get_concept_list()

---

get_concept_main_tag_mapping()

---

get_hidden_tags()

---

get_industry_list()

---

get_industry_main_tag_mapping()

get_initial_hidden_tag_info()

get_initial_main_tag_info()

get_initial_stock_pool_info()

get_initial_sub_tag_info()

get_initial_sub_tags()

get_main_tag_concept_mapping()

get_main_tag_industry_mapping()

get_main_tags()

get_stock_pool_names()

get_sub_tags()

industry_to_main_tag(industry)

match_tag(alias)

match_tag_by_type(alias[, tag_type])

zvt.tag.tag_utils.build_initial_hidden_tag_info

zvt.tag.tag_utils.build_initial_hidden_tag_info()

zvt.tag.tag_utils.build_initial_main_tag_info

zvt.tag.tag_utils.build_initial_main_tag_info()

zvt.tag.tag_utils.build_initial_stock_pool_info

zvt.tag.tag_utils.build_initial_stock_pool_info()

zvt.tag.tag_utils.build_initial_sub_tag_info

zvt.tag.tag_utils.build_initial_sub_tag_info()

zvt.tag.tag_utils.check_missed_concept

zvt.tag.tag_utils.check_missed_concept()

zvt.tag.tag_utils.check_missed_industry

zvt.tag.tag_utils.check_missed_industry()

zvt.tag.tag_utils.get_concept_list

zvt.tag.tag_utils.get_concept_list()

zvt.tag.tag_utils.get_concept_main_tag_mapping

zvt.tag.tag_utils.get_concept_main_tag_mapping() → Dict[str, str]

zvt.tag.tag_utils.get_hidden_tags

zvt.tag.tag_utils.get_hidden_tags()

zvt.tag.tag_utils.get_industry_list

zvt.tag.tag_utils.get_industry_list()

zvt.tag.tag_utils.get_industry_main_tag_mapping

zvt.tag.tag_utils.get_industry_main_tag_mapping() → Dict[str, str]

zvt.tag.tag_utils.get_initial_hidden_tag_info

zvt.tag.tag_utils.get_initial_hidden_tag_info()

zvt.tag.tag_utils.get_initial_main_tag_info

zvt.tag.tag_utils.get_initial_main_tag_info()

zvt.tag.tag_utils.get_initial_stock_pool_info

zvt.tag.tag_utils.get_initial_stock_pool_info()

zvt.tag.tag_utils.get_initial_sub_tag_info

zvt.tag.tag_utils.get_initial_sub_tag_info()

zvt.tag.tag_utils.get_initial_sub_tags

zvt.tag.tag_utils.get_initial_sub_tags() → List[str]

zvt.tag.tag_utils.get_main_tag_concept_mapping

zvt.tag.tag_utils.get_main_tag_concept_mapping() → Dict[str, List[str]]

zvt.tag.tag_utils.get_main_tag_industry_mapping

zvt.tag.tag_utils.get_main_tag_industry_mapping() → Dict[str, List[str]]

zvt.tag.tag_utils.get_main_tags

zvt.tag.tag_utils.get_main_tags()

zvt.tag.tag_utils.get_stock_pool_names

zvt.tag.tag_utils.get_stock_pool_names()

zvt.tag.tag_utils.get_sub_tags

zvt.tag.tag_utils.get_sub_tags()

zvt.tag.tag_utils.industry_to_main_tag

zvt.tag.tag_utils.industry_to_main_tag(*industry*)

zvt.tag.tag_utils.match_tag

zvt.tag.tag_utils.match_tag(*alias*)

zvt.tag.tag_utils.match_tag_by_type

zvt.tag.tag_utils.match_tag_by_type(*alias*, *tag_type='main_tag'*)

zvt.tag.tagger

Classes

StockTagger([force])

Tagger([force])

StockTagger

class zvt.tag.tagger.StockTagger(*force=False*)

Bases: `Tagger`

__init__(*force=False*) → None

data_schema

alias of `StockTags`

entity_schema

alias of `Stock`

clear_state_data(*entity_id=None*)

clear state of the entity

Parameters:: **entity_id** – entity id

decode_state(*state: str*)

decode state

Parameters:: **state** –

Returns::

encode_state(*state: object*)

encode state

Parameters:: **state** –

Returns::

name = *None*

name of the service, default name of class if not set manually

state_schema

alias of `TaggerState`

Tagger

class zvt.tag.tagger.Tagger(*force=False*)

Bases: `OneStateService`

__init__(*force=False*) → None

state_schema

alias of `TaggerState`

clear_state_data(*entity_id=None*)

clear state of the entity

Parameters:: **entity_id** – entity id

decode_state(*state: str*)

decode state

Parameters:: **state** –

Returns::

encode_state(*state: object*)

encode state

Parameters:: **state** –

Returns::

name = *None*

name of the service, default name of class if not set manually

zvt.trader

Functions

trading_signal_type_to_order_type(...)

Classes

AccountService()

OrderType(value) An enumeration.

TradingListener()

TradingSignal(entity_id, due_timestamp, ...)

TradingSignalType(value) An enumeration.

Modules

zvt.trader.errors

zvt.trader.sim_account

zvt.trader.trader

zvt.trader.trader_info_api

zvt.trader.trader_schemas

zvt.trader.trading_signal_type_to_order_type

zvt.trader.trading_signal_type_to_order_type(*trading_signal_type*)

AccountService

class zvt.trader.AccountService

Bases: `TradingListener`

__init__()

get_current_position(*entity_id*, *create_if_not_exist=False*)

overwrite it to provide your real position

Parameters:: **entity_id** –

OrderType

class zvt.trader.OrderType(*value*)

Bases: **`Enum`**

An enumeration.

__init__()

TradingListener

class zvt.trader.TradingListener

Bases: `object`

__init__()

TradingSignal

class zvt.trader.TradingSignal(*entity_id: str*, *due_timestamp: Union[str, Timestamp]*, *happen_timestamp: Union[str, Timestamp]*, *trading_level: IntervalLevel*, *trading_signal_type: TradingSignalType*, *position_pct: Optional[float] = None*, *order_money: Optional[float] = None*, *order_amount: Optional[int] = None*)

Bases: `object`

__init__(*entity_id: str*, *due_timestamp: Union[str, Timestamp]*, *happen_timestamp: Union[str, Timestamp]*, *trading_level: IntervalLevel*, *trading_signal_type: TradingSignalType*, *position_pct: Optional[float] = None*, *order_money: Optional[float] = None*, *order_amount: Optional[int] = None*)

Parameters::

- **entity_id** – the entity id
- **due_timestamp** – the signal due time
- **happen_timestamp** – the time when generating the signal
- **trading_level** – the level
- **trading_signal_type** –
- **position_pct** – percentage of account to order
- **order_money** – money to order
- **order_amount** – amount to order

TradingSignalType

class zvt.trader.TradingSignalType(*value*)

Bases: **`Enum`**

An enumeration.

__init__()

zvt.trader.errors

Exceptions

InvalidOrderError([message])	
InvalidOrderParamError([message])	
NotEnoughMoneyError([message])	
NotEnoughPositionError([message])	
TraderError	Base class for exceptions in this module.
WrongKdataError([message])	

zvt.trader.errors.InvalidOrderError

exception zvt.trader.errors.InvalidOrderError(*message='invalid order'*)

zvt.trader.errors.InvalidOrderParamError

exception zvt.trader.errors.InvalidOrderParamError(*message='invalid order param'*)

zvt.trader.errors.NotEnoughMoneyError

exception zvt.trader.errors.NotEnoughMoneyError(*message='not enough money'*)

zvt.trader.errors.NotEnoughPositionError

exception zvt.trader.errors.NotEnoughPositionError(*message='not enough position'*)

zvt.trader.errors.TraderError

exception zvt.trader.errors.TraderError

Base class for exceptions in this module.

zvt.trader.errors.WrongKdataError

exception zvt.trader.errors.WrongKdataError(*message='wrong kdata'*)

zvt.trader.sim_account

Classes

`AccountDayStatsSchema`(*args, **kwargs)

`PositionSchema`(*args, **kwargs)

`SimAccountService`(entity_schema, ...[, ...])

AccountDayStatsSchema

class zvt.trader.sim_account.AccountDayStatsSchema(**args*, ***kwargs*)

Bases: **`SQLAlchemyAutoSchema`**

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

class Meta

Bases: **`object`**

model

alias of **`AccountStats`**

OPTIONS_CLASS

alias of **`SQLAlchemyAutoSchemaOpts`**

dump(*obj: Any*, *, *many: Optional[bool] = None*)

Serialize an object to native Python data types according to this Schema's fields.

Parameters::
- **obj** – The object to serialize.
- **many** – Whether to serialize *obj* as a collectio
  If *None*, the value for *self.many* is used.

Returns:: Serialized data

New in version 1.0.0.

Changed in version 3.0.0b7: This method returns the serialized data rather than a `(data, errors)` duple. A **`ValidationError`** is raised if `obj` is invalid.

Changed in version 3.0.0rc9: Validation no longer occurs upon serialization.

dumps(*obj: Any*, **args*, *many: Optional[bool] = None*, ***kwargs*)

Same as **`dump()`**, except return a JSON-encoded string.

Parameters::
- **obj** – The object to serialize.
- **many** – Whether to serialize *obj* as a collectio
  If *None*, the value for *self.many* is used.

Returns:: A `json` string

New in version 1.0.0.

Changed in version 3.0.0b7: This method returns the serialized data rather than a `(data, errors)` duple. A **`ValidationError`** is raised if `obj` is invalid.

error_messages*: Dict[str, str] = {}*

Overrides for default schema-level error messages

classmethod from_dict(*fields: dict[str, marshmallow.fields.Field | type]*, *, *name: str = 'GeneratedSchema'*) → type

Generate a *Schema* class given a dictionary of fields.

```
from marshmallow import Schema, fields

PersonSchema = Schema.from_dict({"name": fields.Str()})
print(PersonSchema().load({"name": "David"}))  # =>
{'name': 'David'}
```

Generated schemas are not added to the class registry and therefore cannot be referred to by name in *Nested* fields.

Parameters::
- **fields** (*dict*) – Dictionary mapping field names to field instances.
- **name** (*str*) – Optional name for the class, which will appear in the `repr` for the class.

New in version 3.0.0.

get_attribute(*obj: Any*, *attr: str*, *default: Any*)

Defines how to pull values from an object to serialize.

New in version 2.0.0.

Changed in version 3.0.0a1: Changed position of `obj` and `attr`.

get_instance(*data*)

Retrieve an existing record by primary key(s). If the schema instance is transient, return None.

Parameters:: **data** – Serialized data to inform lookup.

handle_error(*error: ValidationError*, *data: Any*, *, *many: bool*, ***kwargs*)

Custom error handler function for the schema.

Parameters::

- **error** – The *ValidationError* raised during (de)serialization.
- **data** – The original input data.
- **many** – Value of `many` on dump or load.
- **partial** – Value of `partial` on load.

New in version 2.0.0.

Changed in version 3.0.0rc9: Receives *many* and *partial* (on deserialization) as keyword arguments.

load(*data*, *, *session=None*, *instance=None*, *transient=False*, ***kwargs*)

Deserialize data to internal representation.

Parameters::

- **session** – Optional SQLAlchemy session.
- **instance** – Optional existing instance to modify.
- **transient** – Optional switch to allow transient instantiation.

loads(*json_data: str*, *, *many: Optional[bool] = None*, *partial: Optional[Union[bool, Sequence[str], AbstractSet[str]]] = None*, *unknown: Optional[str] = None*, ***kwargs*)

Same as `load()`, except it takes a JSON string as input.

Parameters::

- **json_data** – A JSON string of the data to deserialize.
- **many** – Whether to deserialize *obj* as a collection. If *None*, the value for *self.many* is used.
- **partial** – Whether to ignore missing fields and not require any fields declared. Propagates down to `Nested` fields as well. If its value is an iterable, only missing fields listed in that iterable will be ignored. Use dot delimiters to specify nested fields.
- **unknown** – Whether to exclude, include, or raise an error for unknown fields in the data. Use *EXCLUDE*, *INCLUDE* or *RAISE*. If *None* the value for *self.unknown* is used.

Returns:: Deserialized data

New in version 1.0.0.

Changed in version 3.0.0b7: This method returns the deserialized data rather than a `(data, errors)` duple. A **`ValidationError`** is raised if invalid data are passed.

make_instance(*data*, ***kwargs*)

Deserialize data to an instance of the model if self.load_instance is True.

Update an existing row if specified in *self.instance* or loaded by primary key(s) in the data; else create a new row.

Parameters:: **data** – Data to deserialize.

on_bind_field(*field_name: str*, *field_obj: Field*) → None

Hook to modify a field when it is bound to the *Schema*.

No-op by default.

set_class

alias of **`OrderedSet`**

validate(*data*, *, *session=None*, ***kwargs*)

Validate *data* against the schema, returning a dictionary of validation errors.

Parameters::

- **data** – The data to validate.
- **many** – Whether to validate *data* as a collection. If *None*, the value for *self.many* is used.
- **partial** – Whether to ignore missing fields and not require any fields declared. Propagates down to `Nested` fields as well. If its value is an iterable, only missing fields listed in that iterable will be ignored. Use dot delimiters to specify nested fields.

Returns:: A dictionary of validation errors.

New in version 1.1.0.

fields*: Dict[str, ma_fields.Field]*

Dictionary mapping field_names -> **`Field`** objects

PositionSchema

class zvt.trader.sim_account.PositionSchema(**args*, ***kwargs*)

Bases: **`SQLAlchemyAutoSchema`**

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

class Meta

Bases: **`object`**

model

alias of **`Position`**

OPTIONS_CLASS

alias of **`SQLAlchemyAutoSchemaOpts`**

dump(*obj: Any*, *, *many: Optional[bool] = None*)

Serialize an object to native Python data types according to this Schema's fields.

Parameters::
- **obj** – The object to serialize.
- **many** – Whether to serialize *obj* as a collectio
  If *None*, the value for *self.many* is used.

Returns:: Serialized data

New in version 1.0.0.

Changed in version 3.0.0b7: This method returns the serialized data rather than a `(data, errors)` duple. A **`ValidationError`** is raised if `obj` is invalid.

Changed in version 3.0.0rc9: Validation no longer occurs upon serialization.

dumps(*obj: Any*, **args*, *many: Optional[bool] = None*, ***kwargs*)

Same as **`dump()`**, except return a JSON-encoded string.

Parameters::
- **obj** – The object to serialize.
- **many** – Whether to serialize *obj* as a collectio
  If *None*, the value for *self.many* is used.

Returns:: A `json` string

New in version 1.0.0.

Changed in version 3.0.0b7: This method returns the serialized data rather than a `(data, errors)` duple. A **`ValidationError`** is raised if `obj` is invalid.

error_messages*: Dict[str, str] = {}*

Overrides for default schema-level error messages

classmethod from_dict(*fields: dict[str, marshmallow.fields.Field | type]*, *, *name: str = 'GeneratedSchema'*) → type

Generate a *Schema* class given a dictionary of fields.

```
from marshmallow import Schema, fields

PersonSchema = Schema.from_dict({"name": fields.Str()})
print(PersonSchema().load({"name": "David"}))  # =>
{'name': 'David'}
```

Generated schemas are not added to the class registry and therefore cannot be referred to by name in *Nested* fields.

Parameters::
- **fields** (*dict*) – Dictionary mapping field names to field instances.
- **name** (*str*) – Optional name for the class, which will appear in the `repr` for the class.

New in version 3.0.0.

get_attribute(*obj: Any*, *attr: str*, *default: Any*)

Defines how to pull values from an object to serialize.

New in version 2.0.0.

Changed in version 3.0.0a1: Changed position of `obj` and `attr`.

get_instance(*data*)

Retrieve an existing record by primary key(s). If the schema instance is transient, return None.

Parameters:: **data** – Serialized data to inform lookup.

handle_error(*error: ValidationError*, *data: Any*, *, *many: bool*, ***kwargs*)

Custom error handler function for the schema.

Parameters::

- **error** – The *ValidationError* raised during (de)serialization.
- **data** – The original input data.
- **many** – Value of `many` on dump or load.
- **partial** – Value of `partial` on load.

New in version 2.0.0.

Changed in version 3.0.0rc9: Receives *many* and *partial* (on deserialization) as keyword arguments.

load(*data*, *, *session=None*, *instance=None*, *transient=False*, ***kwargs*)

Deserialize data to internal representation.

Parameters::

- **session** – Optional SQLAlchemy session.
- **instance** – Optional existing instance to modify.
- **transient** – Optional switch to allow transient instantiation.

loads(*json_data: str*, *, *many: Optional[bool] = None*, *partial: Optional[Union[bool, Sequence[str], AbstractSet[str]]] = None*, *unknown: Optional[str] = None*, ***kwargs*)

Same as `load()`, except it takes a JSON string as input.

Parameters::

- **json_data** – A JSON string of the data to deserialize.
- **many** – Whether to deserialize *obj* as a collection. If *None*, the value for *self.many* is used.
- **partial** – Whether to ignore missing fields and not require any fields declared. Propagates down to `Nested` fields as well. If its value is an iterable, only missing fields listed in that iterable will be ignored. Use dot delimiters to specify nested fields.
- **unknown** – Whether to exclude, include, or raise an error for unknown fields in the data. Use *EXCLUDE*, *INCLUDE* or *RAISE*. If *None* the value for *self.unknown* is used.

Returns:: Deserialized data

New in version 1.0.0.

Changed in version 3.0.0b7: This method returns the deserialized data rather than a `(data, errors)` duple. A **`ValidationError`** is raised if invalid data are passed.

make_instance(*data*, ***kwargs*)

Deserialize data to an instance of the model if self.load_instance is True.

Update an existing row if specified in *self.instance* or loaded by primary key(s) in the data; else create a new row.

Parameters:: **data** – Data to deserialize.

on_bind_field(*field_name: str*, *field_obj: Field*) → None

Hook to modify a field when it is bound to the *Schema*.

No-op by default.

set_class

alias of **`OrderedSet`**

validate(*data*, *, *session=None*, ***kwargs*)

Validate *data* against the schema, returning a dictionary of validation errors.

Parameters::

- **data** – The data to validate.
- **many** – Whether to validate *data* as a collection. If *None*, the value for *self.many* is used.
- **partial** – Whether to ignore missing fields and not require any fields declared. Propagates down to `Nested` fields as well. If its value is an iterable, only missing fields listed in that iterable will be ignored. Use dot delimiters to specify nested fields.

Returns:: A dictionary of validation errors.

New in version 1.1.0.

fields*: Dict[str, ma_fields.Field]*

Dictionary mapping field_names -> **`Field`** objects

SimAccountService

class zvt.trader.sim_account.SimAccountService(*entity_schema: TradableEntity*, *trader_name*, *timestamp*, *provider=None*, *level=IntervalLevel.LEVEL_1DAY*, *base_capital=1000000*, *buy_cost=0.001*, *sell_cost=0.001*, *slippage=0.001*, *rich_mode=True*, *adjust_type: Optional[AdjustType] = None*, *keep_history=False*, *real_time=False*, *kdata_use_begin_time=False*)

Bases: `AccountService`

__init__(*entity_schema: TradableEntity*, *trader_name*, *timestamp*, *provider=None*, *level=IntervalLevel.LEVEL_1DAY*, *base_capital=1000000*, *buy_cost=0.001*, *sell_cost=0.001*, *slippage=0.001*, *rich_mode=True*, *adjust_type: Optional[AdjustType] = None*, *keep_history=False*, *real_time=False*, *kdata_use_begin_time=False*)

get_current_position(*entity_id*, *create_if_not_exist=False*) → Optional[Position]

get position for entity_id

Parameters::

- **entity_id** – the entity id
- **create_if_not_exist** – create an empty positio if not exist in current account

Returns::

update_position(*current_position*, *order_amount*, *current_price*, *order_type*, *timestamp*)

Parameters::

- **timestamp** –
- **current_position** (*Position*) –
- **order_amount** –
- **current_price** –
- **order_type** –

zvt.trader.trader

Classes

`StockTrader`([entity_ids, exchanges, codes, ...])

`Trader`([entity_ids, exchanges, codes, ...])

StockTrader

class zvt.trader.trader.StockTrader(*entity_ids: Optional[List[str]] = None*, *exchanges: Optional[List[str]] = None*, *codes: Optional[List[str]] = None*, *start_timestamp: Optional[Union[Timestamp, str]] = None*, *end_timestamp: Optional[Union[Timestamp, str]] = None*, *provider: Optional[str] = None*, *level: Union[str, IntervalLevel] = IntervalLevel.LEVEL_1DAY*, *trader_name: Optional[str] = None*, *real_time: bool = False*, *kdata_use_begin_time: bool = False*, *draw_result: bool = True*, *rich_mode: bool = False*, *adjust_type: AdjustType = AdjustType.hfq*, *profit_threshold=(3, - 0.3)*, *keep_history=False*)

Bases: `Trader`

__init__(*entity_ids: Optional[List[str]] = None*, *exchanges: Optional[List[str]] = None*, *codes: Optional[List[str]] = None*, *start_timestamp: Optional[Union[Timestamp, str]] = None*, *end_timestamp: Optional[Union[Timestamp, str]] = None*, *provider: Optional[str] = None*, *level: Union[str, IntervalLevel] = IntervalLevel.LEVEL_1DAY*, *trader_name: Optional[str] = None*, *real_time: bool = False*, *kdata_use_begin_time: bool = False*, *draw_result: bool = True*, *rich_mode: bool = False*, *adjust_type: AdjustType = AdjustType.hfq*, *profit_threshold=(3, - 0.3)*, *keep_history=False*) → None

entity_schema

alias of `Stock`

handle_factor_targets(*timestamp: Timestamp*)

select targets from factors :param timestamp: the timestamp for next kdata coming

init_entities(*timestamp*)

init the entities for timestamp

Parameters:: **timestamp** –

Returns::

init_factors(*entity_ids*, *entity_schema*, *exchanges*, *codes*, *start_timestamp*, *end_timestamp*, *adjust_type=None*)

overwrite it to init factors if you want to use factor computing model
:param adjust_type:

on_factor_targets_filtered(*timestamp*, *level*, *factor: Factor*, *long_targets: List[str]*, *short_targets: List[str]*) → Tuple[List[str], List[str]]

overwrite it to filter the targets from factor

Parameters::
- **timestamp** – the event time
- **level** – the level
- **factor** – the factor
- **long_targets** – the long targets from the factor
- **short_targets** – the short targets from the factor

Returns:: filtered long targets, filtered short targets

on_targets_selected_from_levels(*timestamp*) → Tuple[List[str], List[str]]

this method's called in every min level cycle to select targets in all levels generated by the previous cycle the default implementation is selecting the targets in all levels overwrite it for your custom logic

Parameters:: **timestamp** – current event time
Returns:: long targets, short targets

on_time(*timestamp: Timestamp*)

called in every min level cycle

Parameters:: **timestamp** – event time

update_targets_by_level(*level: IntervalLevel*, *long_targets: List[str]*, *short_targets: List[str]*) → None

the trading signals is generated in min level,before that,we should cache targets of all levels

Parameters::

- **level** –
- **long_targets** –
- **short_targets** –

Trader

class zvt.trader.trader.Trader(*entity_ids: Optional[List[str]] = None*, *exchanges: Optional[List[str]] = None*, *codes: Optional[List[str]] = None*, *start_timestamp: Optional[Union[Timestamp, str]] = None*, *end_timestamp: Optional[Union[Timestamp, str]] = None*, *provider: Optional[str] = None*, *level: Union[str, IntervalLevel] = IntervalLevel.LEVEL_1DAY*, *trader_name: Optional[str] = None*, *real_time: bool = False*, *kdata_use_begin_time: bool = False*, *draw_result: bool = True*, *rich_mode: bool = False*, *adjust_type: Optional[AdjustType] = None*, *profit_threshold=(3, - 0.3)*, *keep_history=False*, *pre_load_days=365*)

Bases: `object`

__init__(*entity_ids: Optional[List[str]] = None*, *exchanges: Optional[List[str]] = None*, *codes: Optional[List[str]] = None*, *start_timestamp: Optional[Union[Timestamp, str]] = None*, *end_timestamp: Optional[Union[Timestamp, str]] = None*, *provider: Optional[str] = None*, *level: Union[str, IntervalLevel] = IntervalLevel.LEVEL_1DAY*, *trader_name: Optional[str] = None*, *real_time: bool = False*, *kdata_use_begin_time: bool = False*, *draw_result: bool = True*, *rich_mode: bool = False*, *adjust_type: Optional[AdjustType] = None*, *profit_threshold=(3, - 0.3)*, *keep_history=False*, *pre_load_days=365*) → None

init_entities(*timestamp*)

init the entities for timestamp

Parameters:: **timestamp** –

Returns::

init_factors(*entity_ids*, *entity_schema*, *exchanges*, *codes*, *start_timestamp*, *end_timestamp*, *adjust_type=None*)

overwrite it to init factors if you want to use factor computing model

:param adjust_type:

update_targets_by_level(*level: IntervalLevel*, *long_targets: List[str]*, *short_targets: List[str]*) → None

the trading signals is generated in min level,before that,we should cache targets of all levels

Parameters::
- **level** –
- **long_targets** –
- **short_targets** –

on_targets_selected_from_levels(*timestamp*) → Tuple[List[str], List[str]]

this method's called in every min level cycle to select targets in all levels generated by the previous cycle the default implementation is selecting the targets in all levels overwrite it for your custom logic

Parameters:: **timestamp** – current event time

Returns:: long targets, short targets

on_factor_targets_filtered(*timestamp*, *level*, *factor: Factor*, *long_targets: List[str]*, *short_targets: List[str]*) → Tuple[List[str], List[str]]

overwrite it to filter the targets from factor

Parameters::
- **timestamp** – the event time
- **level** – the level
- **factor** – the factor
- **long_targets** – the long targets from the factor
- **short_targets** – the short targets from the factor

Returns:: filtered long targets, filtered short targets

on_time(*timestamp: Timestamp*)

called in every min level cycle

Parameters:: **timestamp** – event time

handle_factor_targets(*timestamp: Timestamp*)

select targets from factors :param timestamp: the timestamp for next kdata coming

zvt.trader.trader_info_api

Functions

`clear_trader`(trader_name[, session])

`get_order_securities`(trader_name)

`get_trader_info`([trader_name, return_type, ...])

Classes

`AccountStatsReader`([start_timestamp, ...])

`OrderReader`([start_timestamp, ...])

---

zvt.trader.trader_info_api.clear_trader

zvt.trader.trader_info_api.clear_trader(*trader_name*, *session=None*)

zvt.trader.trader_info_api.get_order_securities

zvt.trader.trader_info_api.get_order_securities(*trader_name*)

zvt.trader.trader_info_api.get_trader_info

zvt.trader.trader_info_api.get_trader_info(*trader_name=None*, *return_type='df'*, *start_timestamp=None*, *end_timestamp=None*, *filters=None*, *session=None*, *order=None*, *limit=None*) → List[TraderInfo]

AccountStatsReader

class zvt.trader.trader_info_api.AccountStatsReader(*start_timestamp: Optional[Union[Timestamp, str]] = None*, *end_timestamp: Optional[Union[Timestamp, str]] = None*, *columns: Optional[List] = None*, *filters: Optional[List] = None*, *order: Optional[object] = None*, *level: IntervalLevel = IntervalLevel.LEVEL_1DAY*, *trader_names: Optional[List[str]] = None*)

Bases: `DataReader`

__init__(*start_timestamp: Optional[Union[Timestamp, str]] = None*, *end_timestamp: Optional[Union[Timestamp, str]] = None*, *columns: Optional[List] = None*, *filters: Optional[List] = None*, *order: Optional[object] = None*, *level: IntervalLevel = IntervalLevel.LEVEL_1DAY*, *trader_names: Optional[List[str]] = None*) → None

move_on(*to_timestamp: Optional[Union[Timestamp, str]] = None*, *timeout: int = 20*) → object

using continual fetching data in realtime 1)get the data happened before to_timestamp,if not set,get all the data which means to now 2)if computing_window set,the data_df would be cut for saving memory

Parameters::
- **to_timestamp** –
- **timeout** (*int*) –

Returns::

Return type::

OrderReader

class zvt.trader.trader_info_api.OrderReader(*start_timestamp: Optional[Union[Timestamp, str]] = None*, *end_timestamp: Optional[Union[Timestamp, str]] = None*, *columns: Optional[List] = None*, *filters: Optional[List] = None*, *order: Optional[object] = None*, *level: Optional[IntervalLevel] = None*, *trader_names: Optional[List[str]] = None*)

Bases: `DataReader`

__init__(*start_timestamp: Optional[Union[Timestamp, str]] = None*, *end_timestamp: Optional[Union[Timestamp, str]] = None*, *columns: Optional[List] = None*, *filters: Optional[List] = None*, *order: Optional[object] = None*, *level: Optional[IntervalLevel] = None*, *trader_names: Optional[List[str]] = None*) → None

move_on(*to_timestamp: Optional[Union[Timestamp, str]] = None*, *timeout: int = 20*) → object

using continual fetching data in realtime 1)get the data happened before to_timestamp,if not set,get all the data which means to now 2)if computing_window set,the data_df would be cut for saving memory

Parameters::

- **to_timestamp** –
- **timeout** (*int*) –

Returns::

Return type::

zvt.trader.trader_schemas

Classes

`AccountStats`(**kwargs)	account stats of every day
`Order`(**kwargs)	
`Position`(**kwargs)	
`TraderInfo`(**kwargs)	trader info

AccountStats

class zvt.trader.trader_schemas.AccountStats(**kwargs)

Bases: **Base**, **Mixin**

account stats of every day

__init__(**kwargs)

A simple constructor that allows initialization from kwargs.

Sets attributes on the constructed instance using the names and values in `kwargs`.

Only keys that are present as attributes of the instance's class are allowed. These could be, for example, any mapped columns or relationships.

trader_name

机器人名字

cash

可用现金

positions

具体仓位

value

市值

all_value

市值+cash

profit

盈亏

profit_rate

盈亏比例

closing

收盘计算

entity_id

entity id

classmethod get_providers() → List[str]

providers of the schema defined by cls

Returns:: providers

classmethod get_storages(*provider: Optional[str] = None*)

get the storages info

Parameters:: **provider** – provider

Returns:: storages

id

id

classmethod query_data(*provider_index: int = 0*, *ids: Optional[List[str]] = None*, *entity_ids: Optional[List[str]] = None*, *entity_id: Optional[str] = None*, *codes: Optional[List[str]] = None*, *code: Optional[str] = None*, *level: Optional[Union[IntervalLevel, str]] = None*, *provider: Optional[str] = None*, *columns: Optional[List] = None*, *col_label: Optional[dict] = None*, *return_type: str = 'df'*, *start_timestamp: Optional[Union[Timestamp, str]] = None*, *end_timestamp: Optional[Union[Timestamp, str]] = None*, *filters: Optional[List] = None*, *session: Optional[Session] = None*, *order=None*, *limit: Optional[int] = None*, *distinct=None*, *index: Optional[Union[str, list]] = None*, *drop_index_col=False*, *time_field: str = 'timestamp'*)

query data by the arguments

Parameters::

- **provider_index** –
- **data_schema** –
- **ids** –
- **entity_ids** –
- **entity_id** –
- **codes** –
- **code** –
- **level** –
- **provider** –
- **columns** –
- **col_label** – dict with key(column), value(label
- **return_type** – df, domain or dict. default is df
- **start_timestamp** –
- **end_timestamp** –
- **filters** –
- **session** –
- **order** –
- **limit** –
- **index** – index field name, str for single index, str list for multiple index
- **drop_index_col** – whether drop the col if it's in index, default False
- **time_field** –

Returns:: results basing on return_type.

classmethod record_data(*provider_index: int = 0*, *provider: Optional[str] = None*, *force_update=None*, *sleeping_time=None*, *exchanges=None*, *entity_id=None*, *entity_ids=None*, *code=None*, *codes=None*, *real_time=None*, *fix_duplicate_way=None*, *start_timestamp=None*, *end_timestamp=None*, *one_day_trading_minutes=None*, ***kwargs*)

record data by the arguments

Parameters::

- **entity_id** –
- **provider_index** –
- **provider** –
- **force_update** –
- **sleeping_time** –
- **exchanges** –
- **entity_ids** –
- **code** –
- **codes** –
- **real_time** –
- **fix_duplicate_way** –
- **start_timestamp** –
- **end_timestamp** –
- **one_day_trading_minutes** –
- **kwargs** –

Returns::

classmethod register_provider(*provider*)

register the provider to the schema defined by cls

Parameters:: **provider** –

classmethod register_recorder_cls(*provider*, *recorder_cls*)

register the recorder for the schema

Parameters::

- **provider** –
- **recorder_cls** –

timestamp

the meaning could be different for different case,most time it means 'happen time'

Order

class zvt.trader.trader_schemas.Order(**kwargs)

Bases: **Base**, **Mixin**

__init__(**kwargs)

A simple constructor that allows initialization from kwargs.

Sets attributes on the constructed instance using the names and values in `kwargs`.

Only keys that are present as attributes of the instance's class are allowed. These could be, for example, any mapped columns or relationships.

trader_name

机器人名字

order_price

订单价格

order_amount

订单数量

order_type

订单类型

status

订单状态

level

产生订单的selector/factor level

entity_id

entity id

classmethod get_providers() → List[str]

providers of the schema defined by cls

Returns:: providers

classmethod get_storages(*provider: Optional[str] = None*)

get the storages info

Parameters:: **provider** – provider

Returns:: storages

id

id

classmethod query_data(*provider_index: int = 0*, *ids: Optional[List[str]] = None*, *entity_ids: Optional[List[str]] = None*, *entity_id: Optional[str] = None*, *codes: Optional[List[str]] = None*, *code: Optional[str] = None*, *level: Optional[Union[IntervalLevel, str]] = None*, *provider: Optional[str] = None*, *columns: Optional[List] = None*, *col_label: Optional[dict] = None*, *return_type: str = 'df'*, *start_timestamp: Optional[Union[Timestamp, str]] = None*, *end_timestamp: Optional[Union[Timestamp, str]] = None*, *filters: Optional[List] = None*, *session: Optional[Session] = None*, *order=None*, *limit: Optional[int] = None*, *distinct=None*, *index: Optional[Union[str, list]] = None*, *drop_index_col=False*, *time_field: str = 'timestamp'*)

query data by the arguments

Parameters::

- **provider_index** –
- **data_schema** –
- **ids** –
- **entity_ids** –
- **entity_id** –
- **codes** –
- **code** –
- **level** –
- **provider** –
- **columns** –
- **col_label** – dict with key(column), value(label
- **return_type** – df, domain or dict. default is df
- **start_timestamp** –
- **end_timestamp** –
- **filters** –
- **session** –
- **order** –
- **limit** –
- **index** – index field name, str for single index, str list for multiple index
- **drop_index_col** – whether drop the col if it's in index, default False
- **time_field** –

Returns:: results basing on return_type.

classmethod record_data(*provider_index: int = 0*, *provider: Optional[str] = None*, *force_update=None*, *sleeping_time=None*, *exchanges=None*, *entity_id=None*, *entity_ids=None*, *code=None*, *codes=None*, *real_time=None*, *fix_duplicate_way=None*, *start_timestamp=None*, *end_timestamp=None*, *one_day_trading_minutes=None*, ***kwargs*)

record data by the arguments

Parameters::

- **entity_id** –
- **provider_index** –
- **provider** –
- **force_update** –
- **sleeping_time** –
- **exchanges** –
- **entity_ids** –
- **code** –
- **codes** –
- **real_time** –
- **fix_duplicate_way** –
- **start_timestamp** –
- **end_timestamp** –
- **one_day_trading_minutes** –
- **kwargs** –

Returns::

classmethod register_provider(*provider*)

register the provider to the schema defined by cls

Parameters:: **provider** –

classmethod register_recorder_cls(*provider*, *recorder_cls*)

register the recorder for the schema

Parameters::

- **provider** –
- **recorder_cls** –

timestamp

the meaning could be different for different case,most time it means 'happen time'

Position

class zvt.trader.trader_schemas.Position(**kwargs)

Bases: **Base**, **Mixin**

__init__(**kwargs)

A simple constructor that allows initialization from kwargs.

Sets attributes on the constructed instance using the names and values in `kwargs`.

Only keys that are present as attributes of the instance's class are allowed. These could be, for example, any mapped columns or relationships.

trader_name

机器人名字

account_stats_id

账户id

long_amount

做多数量

available_long

可平多数量

average_long_price

平均做多价格

short_amount

做空数量

available_short

可平空数量

average_short_price
平均做空价格

profit
盈亏

profit_rate
盈亏比例

value
市值 或者 占用的保证金(方便起见，总是100%)

trading_t
交易类型(0代表T+0,1代表T+1)

entity_id
entity id

classmethod get_providers() → List[str]
providers of the schema defined by cls

Returns:: providers

classmethod get_storages(*provider: Optional[str] = None*)
get the storages info

Parameters:: **provider** – provider
Returns:: storages

id
id

classmethod query_data(*provider_index: int = 0*, *ids: Optional[List[str]] = None*, *entity_ids: Optional[List[str]] = None*, *entity_id: Optional[str] = None*, *codes: Optional[List[str]] = None*, *code: Optional[str] = None*, *level: Optional[Union[IntervalLevel, str]] = None*, *provider:*

Optional[str] = None, *columns: Optional[List] = None*, *col_label: Optional[dict] = None*, *return_type: str = 'df'*, *start_timestamp: Optional[Union[Timestamp, str]] = None*, *end_timestamp: Optional[Union[Timestamp, str]] = None*, *filters: Optional[List] = None*, *session: Optional[Session] = None*, *order=None*, *limit: Optional[int] = None*, *distinct=None*, *index: Optional[Union[str, list]] = None*, *drop_index_col=False*, *time_field: str = 'timestamp'*)

query data by the arguments

Parameters::

- **provider_index** –
- **data_schema** –
- **ids** –
- **entity_ids** –
- **entity_id** –
- **codes** –
- **code** –
- **level** –
- **provider** –
- **columns** –
- **col_label** – dict with key(column), value(label
- **return_type** – df, domain or dict. default is df
- **start_timestamp** –
- **end_timestamp** –
- **filters** –
- **session** –
- **order** –
- **limit** –
- **index** – index field name, str for single index, str list for multiple index
- **drop_index_col** – whether drop the col if it's in index, default False
- **time_field** –

Returns:: results basing on return_type.

classmethod record_data(*provider_index: int = 0*, *provider: Optional[str] = None*, *force_update=None*, *sleeping_time=None*, *exchanges=None*, *entity_id=None*, *entity_ids=None*, *code=None*, *codes=None*,

real_time=None, *fix_duplicate_way=None*, *start_timestamp=None*, *end_timestamp=None*, *one_day_trading_minutes=None*, ***kwargs*)

record data by the arguments

Parameters::

- **entity_id** –
- **provider_index** –
- **provider** –
- **force_update** –
- **sleeping_time** –
- **exchanges** –
- **entity_ids** –
- **code** –
- **codes** –
- **real_time** –
- **fix_duplicate_way** –
- **start_timestamp** –
- **end_timestamp** –
- **one_day_trading_minutes** –
- **kwargs** –

Returns::

classmethod register_provider(*provider*)

register the provider to the schema defined by cls

Parameters:: **provider** –

classmethod register_recorder_cls(*provider*, *recorder_cls*)

register the recorder for the schema

Parameters::

- **provider** –
- **recorder_cls** –

timestamp

the meaning could be different for different case,most time it means 'happen time'

TraderInfo

class zvt.trader.trader_schemas.TraderInfo(***kwargs*)

Bases: **`Base`**, **`Mixin`**

trader info

__init__(***kwargs*)

A simple constructor that allows initialization from kwargs.

Sets attributes on the constructed instance using the names and values in `kwargs`.

Only keys that are present as attributes of the instance's class are allowed. These could be, for example, any mapped columns or relationships.

trader_name

机器人名字

entity_id

entity id

classmethod get_providers() → List[str]

providers of the schema defined by cls

Returns:: providers

classmethod get_storages(*provider: Optional[str] = None*)

get the storages info

Parameters:: **provider** – provider
Returns:: storages

id

id

classmethod query_data(*provider_index: int = 0*, *ids: Optional[List[str]] = None*, *entity_ids: Optional[List[str]] = None*, *entity_id: Optional[str] = None*, *codes: Optional[List[str]] = None*, *code: Optional[str] = None*, *level: Optional[Union[IntervalLevel, str]] = None*, *provider: Optional[str] = None*, *columns: Optional[List] = None*, *col_label: Optional[dict] = None*, *return_type: str = 'df'*, *start_timestamp: Optional[Union[Timestamp, str]] = None*, *end_timestamp: Optional[Union[Timestamp, str]] = None*, *filters: Optional[List] = None*, *session: Optional[Session] = None*, *order=None*, *limit: Optional[int] = None*, *distinct=None*, *index: Optional[Union[str, list]] = None*, *drop_index_col=False*, *time_field: str = 'timestamp'*)

query data by the arguments

Parameters::

- **provider_index** –
- **data_schema** –
- **ids** –
- **entity_ids** –
- **entity_id** –
- **codes** –
- **code** –
- **level** –
- **provider** –
- **columns** –
- **col_label** – dict with key(column), value(label
- **return_type** – df, domain or dict. default is df
- **start_timestamp** –
- **end_timestamp** –
- **filters** –
- **session** –
- **order** –
- **limit** –
- **index** – index field name, str for single index, str list for multiple index
- **drop_index_col** – whether drop the col if it's in index, default False
- **time_field** –

Returns:: results basing on return_type.

classmethod record_data(*provider_index: int = 0*, *provider: Optional[str] = None*, *force_update=None*, *sleeping_time=None*, *exchanges=None*, *entity_id=None*, *entity_ids=None*, *code=None*, *codes=None*, *real_time=None*, *fix_duplicate_way=None*, *start_timestamp=None*, *end_timestamp=None*, *one_day_trading_minutes=None*, ***kwargs*)

record data by the arguments

Parameters::

- **entity_id** –
- **provider_index** –
- **provider** –
- **force_update** –
- **sleeping_time** –
- **exchanges** –
- **entity_ids** –
- **code** –
- **codes** –
- **real_time** –
- **fix_duplicate_way** –
- **start_timestamp** –
- **end_timestamp** –
- **one_day_trading_minutes** –
- **kwargs** –

Returns::

classmethod register_provider(*provider*)

register the provider to the schema defined by cls

Parameters:: **provider** –

classmethod register_recorder_cls(*provider*, *recorder_cls*)

register the recorder for the schema

Parameters::

- **provider** –
- **recorder_cls** –

timestamp

the meaning could be different for different case,most time it means 'happen time'

zvt.trading

Modules

zvt.trading.common

---

zvt.trading.trading_models

---

zvt.trading.trading_schemas

---

zvt.trading.trading_service

---

zvt.trading.common

Classes

`ExecutionStatus`(value) An enumeration.

ExecutionStatus

class zvt.trading.common.ExecutionStatus(*value*)

Bases: `Enum`

An enumeration.

__init__()

zvt.trading.trading_models

Classes

BuildQueryStockQuoteSettingModel(*, ...[, ...])

BuildTradingPlanModel(*, stock_id, ...)

QueryStockQuoteModel(*[, main_tag, ...])

QueryStockQuoteSettingModel(*[, ...])

QueryTagQuoteModel(*, stock_pool_name, main_tags)

QueryTradingPlanModel(*, time_range)

StockQuoteModel(*, id, entity_id, timestamp, ...)

StockQuoteStatsModel(*, limit_up_count, ...)

TagQuoteStatsModel(*, main_tag, ...)

TradingPlanModel(*, id, entity_id, ...[, status])

BuildQueryStockQuoteSettingModel

class zvt.trading.trading_models.BuildQueryStockQuoteSettingModel(*, *stock_pool_name: str*, *main_tags: Optional[List[str]] = None*)

Bases: **`BaseModel`**

__init__(**data: Any*) → None

Create a new model by parsing and validating input data from keyword arguments.

Raises [*ValidationError*][pydantic_core.ValidationError] if the input data cannot be validated to form a valid model.

self is explicitly positional-only to allow *self* as a field name.

copy(*, *include: AbstractSetIntStr | MappingIntStrAny | None = None*, *exclude: AbstractSetIntStr | MappingIntStrAny | None = None*, *update: Dict[str, Any] | None = None*, *deep: bool = False*) → Model

Returns a copy of the model.

!!! warning "Deprecated"

This method is now deprecated; use *model_copy* instead.

If you need *include* or *exclude*, use:

```
`py data = self.model_dump(include=include, exclude=exclude, round_trip=True) data = {**data, **(update or {})} copied = self.model_validate(data) `
```

Args:

include: Optional set or mapping specifying which fields to include in the copied model. exclude: Optional set or mapping

specifying which fields to exclude in the copied model. update: Optional dictionary of field-value pairs to override field values in the copied model. deep: If True, the values of fields that are Pydantic models will be deep-copied.

Returns:

A copy of the model with included, excluded and updated fields as specified.

model_computed_fields*: ClassVar[dict[str, ComputedFieldInfo]] = {}*

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

model_config*: ClassVar[ConfigDict] = {}*

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

classmethod model_construct(*_fields_set: set[str] | None = None*, ***values: Any*) → Model

Creates a new instance of the *Model* class with validated data.

Creates a new model setting *__dict__* and *__pydantic_fields_set__* from trusted or pre-validated data. Default values are respected, but no other validation is performed. Behaves as if *Config.extra = 'allow'* was set since it adds all passed values

Args:

_fields_set: The set of field names accepted for the Model instance. values: Trusted or pre-validated data dictionary.

Returns:

A new instance of the *Model* class with validated data.

model_copy(*, *update: dict[str, Any] | None = None*, *deep: bool = False*) → Model

Usage docs: https://docs.pydantic.dev/2.6/concepts/serialization/#model_copy

Returns a copy of the model.

Args:

update: Values to change/add in the new model. Note: the data is not validated
before creating the new model. You should trust this data.

deep: Set to *True* to make a deep copy of the model.

Returns:

New model instance.

model_dump(*, *mode: Literal['json', 'python'] | str = 'python'*, *include: IncEx = None*, *exclude: IncEx = None*, *by_alias: bool = False*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*, *round_trip: bool = False*, *warnings: bool = True*) → dict[str, Any]

Usage docs: https://docs.pydantic.dev/2.6/concepts/serialization/#modelmodel_dump

Generate a dictionary representation of the model, optionally specifying which fields to include or exclude.

Args:

mode: The mode in which *to_python* should run.
If mode is 'json', the output will only contain JSON serializable types. If mode is 'python', the output may contain non-JSON-serializable Python objects.

include: A list of fields to include in the output. exclude: A list of fields to exclude from the output. by_alias: Whether to use the field's alias in the dictionary key if defined. exclude_unset: Whether to exclude fields that have not been explicitly set. exclude_defaults: Whether to exclude fields that are set to their default value. exclude_none: Whether to exclude fields that have a value of *None*. round_trip: If True, dumped values should be

valid as input for non-idempotent types such as Json[T]. warnings: Whether to log warnings when invalid fields are encountered.

Returns:

A dictionary representation of the model.

model_dump_json(*, *indent: int | None = None*, *include: IncEx = None*, *exclude: IncEx = None*, *by_alias: bool = False*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*, *round_trip: bool = False*, *warnings: bool = True*) → str

Usage docs: https://docs.pydantic.dev/2.6/concepts/serialization/#modelmodel_dump_json

Generates a JSON representation of the model using Pydantic's *to_json* method.

Args:

indent: Indentation to use in the JSON output. If None is passed, the output will be compact. include: Field(s) to include in the JSON output. exclude: Field(s) to exclude from the JSON output. by_alias: Whether to serialize using field aliases. exclude_unset: Whether to exclude fields that have not been explicitly set. exclude_defaults: Whether to exclude fields that are set to their default value. exclude_none: Whether to exclude fields that have a value of *None*. round_trip: If True, dumped values should be valid as input for non-idempotent types such as Json[T]. warnings: Whether to log warnings when invalid fields are encountered.

Returns:

A JSON string representation of the model.

property model_extra*: dict[str, Any] | None*

Get extra fields set during validation.

Returns:

A dictionary of extra fields, or *None* if *config.extra* is not set to *"allow"*.

model_fields*: ClassVar[dict[str, FieldInfo]] = {'main_tags': FieldInfo(annotation=Union[List[str], NoneType], required=False), 'stock_pool_name': FieldInfo(annotation=str, required=True)}*

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

property model_fields_set*: set[str]*

Returns the set of fields that have been explicitly set on this model instance.

Returns:

A set of strings representing the fields that have been set, i.e. that were not filled from defaults.

classmethod model_json_schema(*by_alias: bool = True*, *ref_template: str = '#/$defs/{model}'*, *schema_generator: type[pydantic.json_schema.GenerateJsonSchema] = <class 'pydantic.json_schema.GenerateJsonSchema'>*, *mode: ~typing.Literal['validation', 'serialization'] = 'validation'*) → dict[str, Any]

Generates a JSON schema for a model class.

Args:

by_alias: Whether to use attribute aliases or not. ref_template: The reference template. schema_generator: To override the logic used to generate the JSON schema, as a subclass of

GenerateJsonSchema with your desired modifications

mode: The mode in which to generate the schema.

Returns:

The JSON schema for the given model class.

classmethod model_parametrized_name(*params: tuple[type[Any], ...]*) → str

Compute the class name for parametrizations of generic classes.

This method can be overridden to achieve a custom naming scheme for generic BaseModels.

Args:

params: Tuple of types of the class. Given a generic class *Model* with 2 type variables and a concrete model *Model[str, int]*, the value *(str, int)* would be passed to *params*.

Returns:

String representing the new class where *params* are passed to *cls* as type variables.

Raises:

TypeError: Raised when trying to generate concrete names for non-generic models.

model_post_init(*_BaseModel__context: Any*) → None

Override this method to perform additional initialization after *__init__* and *model_construct*. This is useful if you want to do some validation that requires the entire model to be initialized.

classmethod model_rebuild(*, *force: bool = False*, *raise_errors: bool = True*, *_parent_namespace_depth: int = 2*, *_types_namespace: Optional[dict[str, Any]] = None*) → bool | None

Try to rebuild the pydantic-core schema for the model.

This may be necessary when one of the annotations is a ForwardRef which could not be resolved during the initial attempt to build the schema, and automatic rebuilding fails.

Args:

force: Whether to force the rebuilding of the model schema, defaults to *False*. raise_errors: Whether to raise errors, defaults to *True*. _parent_namespace_depth: The depth level of the parent namespace, defaults to 2. _types_namespace: The types namespace, defaults to *None*.

Returns:
Returns *None* if the schema is already "complete" and rebuilding was not required. If rebuilding _was_ required, returns *True* if rebuilding was successful, otherwise *False*.

classmethod model_validate(*obj: Any*, *, *strict: bool | None = None*, *from_attributes: bool | None = None*, *context: dict[str, Any] | None = None*) → Model

Validate a pydantic model instance.

Args:
obj: The object to validate. strict: Whether to enforce types strictly. from_attributes: Whether to extract data from object attributes. context: Additional context to pass to the validator.

Raises:
ValidationError: If the object could not be validated.

Returns:
The validated model instance.

classmethod model_validate_json(*json_data: str | bytes | bytearray*, *, *strict: bool | None = None*, *context: dict[str, Any] | None = None*) → Model

Usage docs: https://docs.pydantic.dev/2.6/concepts/json/#json-parsing

Validate the given JSON data against the Pydantic model.

Args:

json_data: The JSON data to validate. strict: Whether to enforce types strictly. context: Extra variables to pass to the validator.

Returns:
The validated Pydantic model.

Raises:
ValueError: If *json_data* is not a JSON string.

classmethod model_validate_strings(*obj: Any*, *, *strict: bool | None = None*, *context: dict[str, Any] | None = None*) → Model

Validate the given object contains string data against the Pydantic model.

Args:
obj: The object contains string data to validate. strict: Whether to enforce types strictly. context: Extra variables to pass to the validator.

Returns:
The validated Pydantic model.

BuildTradingPlanModel

class zvt.trading.trading_models.BuildTradingPlanModel(*, *stock_id: str*, *trading_date: datetime*, *expected_open_pct: float*, *buy_price: Optional[float]*, *sell_price: Optional[float]*, *trading_reason: str*, *trading_signal_type: TradingSignalType*)

Bases: **`BaseModel`**

__init__(**data: Any*) → None

Create a new model by parsing and validating input data from keyword arguments.

Raises [*ValidationError*][pydantic_core.ValidationError] if the input data cannot be validated to form a valid model.

self is explicitly positional-only to allow *self* as a field name.

copy(*, *include: AbstractSetIntStr | MappingIntStrAny | None = None*, *exclude: AbstractSetIntStr | MappingIntStrAny | None = None*, *update: Dict[str, Any] | None = None*, *deep: bool = False*) → Model

Returns a copy of the model.

!!! warning "Deprecated"

This method is now deprecated; use *model_copy* instead.

If you need *include* or *exclude*, use:

```
`py data = self.model_dump(include=include, exclude=exclude, round_trip=True) data = {**data, **(update or {})} copied = self.model_validate(data) `
```

Args:

include: Optional set or mapping specifying which fields to include in the copied model. exclude: Optional set or mapping

specifying which fields to exclude in the copied model. update: Optional dictionary of field-value pairs to override field values in the copied model. deep: If True, the values of fields that are Pydantic models will be deep-copied.

Returns:

A copy of the model with included, excluded and updated fields as specified.

model_computed_fields*: ClassVar[dict[str, ComputedFieldInfo]] = {}*

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

model_config*: ClassVar[ConfigDict] = {}*

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

classmethod model_construct(*_fields_set: set[str] | None = None*, ***values: Any*) → Model

Creates a new instance of the *Model* class with validated data.

Creates a new model setting *__dict__* and *__pydantic_fields_set__* from trusted or pre-validated data. Default values are respected, but no other validation is performed. Behaves as if *Config.extra = 'allow'* was set since it adds all passed values

Args:

_fields_set: The set of field names accepted for the Model instance. values: Trusted or pre-validated data dictionary.

Returns:

A new instance of the *Model* class with validated data.

model_copy(*, *update: dict[str, Any] | None = None*, *deep: bool = False*) → Model

Usage docs: https://docs.pydantic.dev/2.6/concepts/serialization/#model_copy

Returns a copy of the model.

Args:

update: Values to change/add in the new model. Note: the data is not validated
before creating the new model. You should trust this data.

deep: Set to *True* to make a deep copy of the model.

Returns:

New model instance.

model_dump(*, *mode: Literal['json', 'python'] | str = 'python'*, *include: IncEx = None*, *exclude: IncEx = None*, *by_alias: bool = False*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*, *round_trip: bool = False*, *warnings: bool = True*) → dict[str, Any]

Usage docs: https://docs.pydantic.dev/2.6/concepts/serialization/#modelmodel_dump

Generate a dictionary representation of the model, optionally specifying which fields to include or exclude.

Args:

mode: The mode in which *to_python* should run.
If mode is 'json', the output will only contain JSON serializable types. If mode is 'python', the output may contain non-JSON-serializable Python objects.

include: A list of fields to include in the output. exclude: A list of fields to exclude from the output. by_alias: Whether to use the field's alias in the dictionary key if defined. exclude_unset: Whether to exclude fields that have not been explicitly set. exclude_defaults: Whether to exclude fields that are set to their default value. exclude_none: Whether to exclude fields that have a value of *None*. round_trip: If True, dumped values should be

valid as input for non-idempotent types such as Json[T]. warnings: Whether to log warnings when invalid fields are encountered.

Returns:

A dictionary representation of the model.

model_dump_json(*, *indent: int | None = None*, *include: IncEx = None*, *exclude: IncEx = None*, *by_alias: bool = False*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*, *round_trip: bool = False*, *warnings: bool = True*) → str

Usage docs: https://docs.pydantic.dev/2.6/concepts/serialization/#modelmodel_dump_json

Generates a JSON representation of the model using Pydantic's *to_json* method.

Args:

indent: Indentation to use in the JSON output. If None is passed, the output will be compact. include: Field(s) to include in the JSON output. exclude: Field(s) to exclude from the JSON output. by_alias: Whether to serialize using field aliases. exclude_unset: Whether to exclude fields that have not been explicitly set. exclude_defaults: Whether to exclude fields that are set to their default value. exclude_none: Whether to exclude fields that have a value of *None*. round_trip: If True, dumped values should be valid as input for non-idempotent types such as Json[T]. warnings: Whether to log warnings when invalid fields are encountered.

Returns:

A JSON string representation of the model.

property model_extra*: dict[str, Any] | None*

Get extra fields set during validation.

Returns:

A dictionary of extra fields, or *None* if *config.extra* is not set to *"allow"*.

model_fields*: ClassVar[dict[str, FieldInfo]] = {'buy_price': FieldInfo(annotation=Union[float, NoneType], required=True), 'expected_open_pct': FieldInfo(annotation=float, required=True), 'sell_price': FieldInfo(annotation=Union[float, NoneType], required=True), 'stock_id': FieldInfo(annotation=str, required=True), 'trading_date': FieldInfo(annotation=datetime, required=True), 'trading_reason': FieldInfo(annotation=str, required=True), 'trading_signal_type': FieldInfo(annotation=TradingSignalType, required=True)}*

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

property model_fields_set*: set[str]*

Returns the set of fields that have been explicitly set on this model instance.

Returns:

A set of strings representing the fields that have been set, i.e. that were not filled from defaults.

classmethod model_json_schema(*by_alias: bool = True*, *ref_template: str = '#/$defs/{model}'*, *schema_generator: type[pydantic.json_schema.GenerateJsonSchema] = <class 'pydantic.json_schema.GenerateJsonSchema'>*, *mode: ~typing.Literal['validation', 'serialization'] = 'validation'*) → dict[str, Any]

Generates a JSON schema for a model class.

Args:

by_alias: Whether to use attribute aliases or not. ref_template: The reference template. schema_generator: To override the logic used to generate the JSON schema, as a subclass of

GenerateJsonSchema with your desired modifications

mode: The mode in which to generate the schema.

Returns:

The JSON schema for the given model class.

classmethod model_parametrized_name(*params: tuple[type[Any], ...]*) → str

Compute the class name for parametrizations of generic classes.

This method can be overridden to achieve a custom naming scheme for generic BaseModels.

Args:

params: Tuple of types of the class. Given a generic class *Model* with 2 type variables and a concrete model *Model[str, int]*, the value *(str, int)* would be passed to *params*.

Returns:

String representing the new class where *params* are passed to *cls* as type variables.

Raises:

TypeError: Raised when trying to generate concrete names for non-generic models.

model_post_init(*_BaseModel__context: Any*) → None

Override this method to perform additional initialization after *__init__* and *model_construct*. This is useful if you want to do some validation that requires the entire model to be initialized.

classmethod model_rebuild(*, *force: bool = False*, *raise_errors: bool = True*, *_parent_namespace_depth: int = 2*, *_types_namespace: Optional[dict[str, Any]] = None*) → bool | None

Try to rebuild the pydantic-core schema for the model.

This may be necessary when one of the annotations is a ForwardRef which could not be resolved during the initial attempt to build the schema, and automatic rebuilding fails.

Args:

force: Whether to force the rebuilding of the model schema, defaults to *False*. raise_errors: Whether to raise errors, defaults to *True*. _parent_namespace_depth: The depth level of the parent namespace, defaults to 2. _types_namespace: The types namespace, defaults to *None*.

Returns:

Returns *None* if the schema is already "complete" and rebuilding was not required. If rebuilding _was_ required, returns *True* if rebuilding was successful, otherwise *False*.

classmethod model_validate(*obj: Any*, *, *strict: bool | None = None*, *from_attributes: bool | None = None*, *context: dict[str, Any] | None = None*) → Model

Validate a pydantic model instance.

Args:

obj: The object to validate. strict: Whether to enforce types strictly. from_attributes: Whether to extract data from object attributes. context: Additional context to pass to the validator.

Raises:

ValidationError: If the object could not be validated.

Returns:

The validated model instance.

classmethod model_validate_json(*json_data: str | bytes | bytearray*, *, *strict: bool | None = None*, *context: dict[str, Any] | None = None*) → Model

Usage docs: https://docs.pydantic.dev/2.6/concepts/json/#json-parsing

Validate the given JSON data against the Pydantic model.

Args:
json_data: The JSON data to validate. strict: Whether to enforce types strictly. context: Extra variables to pass to the validator.

Returns:
The validated Pydantic model.

Raises:
ValueError: If *json_data* is not a JSON string.

classmethod model_validate_strings(*obj: Any*, *, *strict: bool | None = None*, *context: dict[str, Any] | None = None*) → Model

Validate the given object contains string data against the Pydantic model.

Args:
obj: The object contains string data to validate. strict: Whether to enforce types strictly. context: Extra variables to pass to the validator.

Returns:
The validated Pydantic model.

QueryStockQuoteModel

class zvt.trading.trading_models.QueryStockQuoteModel(*, *main_tag: Optional[str] = None*, *entity_ids: Optional[List[str]] = None*, *stock_pool_name: Optional[str] = None*, *order_by_type: Optional[OrderByType] = OrderByType.desc*, *order_by_field: Optional[str] = 'change_pct'*)

Bases: `CustomModel`

__init__(**data: Any*) → None

Create a new model by parsing and validating input data from keyword arguments.

Raises [*ValidationError*][pydantic_core.ValidationError] if the input data cannot be validated to form a valid model.

self is explicitly positional-only to allow *self* as a field name.

copy(*, *include: AbstractSetIntStr | MappingIntStrAny | None = None*, *exclude: AbstractSetIntStr | MappingIntStrAny | None = None*, *update: Dict[str, Any] | None = None*, *deep: bool = False*) → Model

Returns a copy of the model.

!!! warning "Deprecated"

This method is now deprecated; use *model_copy* instead.

If you need *include* or *exclude*, use:

```
`py data = self.model_dump(include=include, exclude=exclude, round_trip=True) data = {**data, **(update or {})} copied = self.model_validate(data) `
```

Args:

include: Optional set or mapping specifying which fields to include in the copied model. exclude: Optional set or mapping specifying which fields to exclude in the copied model. update: Optional dictionary of field-value pairs to override field values in the copied model. deep: If True, the values of fields that are Pydantic models will be deep-copied.

Returns:

A copy of the model with included, excluded and updated fields as specified.

model_computed_fields*: ClassVar[dict[str, ComputedFieldInfo]] = {}*

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

model_config*: ClassVar[ConfigDict] = {'allow_inf_nan': True, 'from_attributes': True}*

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

classmethod model_construct(*_fields_set: set[str] | None = None*, ***values: Any*) → Model

Creates a new instance of the *Model* class with validated data.

Creates a new model setting *__dict__* and *__pydantic_fields_set__* from trusted or pre-validated data. Default values are respected, but no other validation is performed. Behaves as if *Config.extra = 'allow'* was set since it adds all passed values

Args:

_fields_set: The set of field names accepted for the Model instance. values: Trusted or pre-validated data dictionary.

Returns:

A new instance of the *Model* class with validated data.

model_copy(*, *update: dict[str, Any] | None = None*, *deep: bool = False*) → Model

Usage docs: https://docs.pydantic.dev/2.6/concepts/serialization/#model_copy

Returns a copy of the model.

Args:

update: Values to change/add in the new model. Note: the data is not validated
before creating the new model. You should trust this data.

deep: Set to *True* to make a deep copy of the model.

Returns:

New model instance.

model_dump(*, *mode: Literal['json', 'python'] | str = 'python'*, *include: IncEx = None*, *exclude: IncEx = None*, *by_alias: bool = False*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*, *round_trip: bool = False*, *warnings: bool = True*) → dict[str, Any]

Usage docs: https://docs.pydantic.dev/2.6/concepts/serialization/#modelmodel_dump

Generate a dictionary representation of the model, optionally specifying which fields to include or exclude.

Args:

mode: The mode in which *to_python* should run.
If mode is 'json', the output will only contain JSON serializable types. If mode is 'python', the output may contain non-JSON-serializable Python objects.

include: A list of fields to include in the output. exclude: A list of fields to exclude from the output. by_alias: Whether to use the

field's alias in the dictionary key if defined. exclude_unset: Whether to exclude fields that have not been explicitly set. exclude_defaults: Whether to exclude fields that are set to their default value. exclude_none: Whether to exclude fields that have a value of *None*. round_trip: If True, dumped values should be valid as input for non-idempotent types such as Json[T]. warnings: Whether to log warnings when invalid fields are encountered.

Returns:

A dictionary representation of the model.

model_dump_json(*, *indent: int | None = None*, *include: IncEx = None*, *exclude: IncEx = None*, *by_alias: bool = False*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*, *round_trip: bool = False*, *warnings: bool = True*) → str

Usage docs: https://docs.pydantic.dev/2.6/concepts/serialization/#modelmodel_dump_json

Generates a JSON representation of the model using Pydantic's *to_json* method.

Args:

indent: Indentation to use in the JSON output. If None is passed, the output will be compact. include: Field(s) to include in the JSON output. exclude: Field(s) to exclude from the JSON output. by_alias: Whether to serialize using field aliases. exclude_unset: Whether to exclude fields that have not been explicitly set. exclude_defaults: Whether to exclude fields that are set to their default value. exclude_none: Whether to exclude fields that have a value of *None*. round_trip: If True, dumped values should be valid as input for non-idempotent types such as Json[T]. warnings: Whether to log warnings when invalid fields are encountered.

Returns:

A JSON string representation of the model.

property model_extra*: dict[str, Any] | None*

Get extra fields set during validation.

Returns:

A dictionary of extra fields, or *None* if *config.extra* is not set to *"allow"*.

model_fields*: ClassVar[dict[str, FieldInfo]] = {'entity_ids': FieldInfo(annotation=Union[List[str], NoneType], required=False), 'main_tag': FieldInfo(annotation=Union[str, NoneType], required=False), 'order_by_field': FieldInfo(annotation=Union[str, NoneType], required=False, default='change_pct'), 'order_by_type': FieldInfo(annotation=Union[OrderByType, NoneType], required=False, default=<OrderByType.desc: 'desc'>), 'stock_pool_name': FieldInfo(annotation=Union[str, NoneType], required=False)}*

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

property model_fields_set*: set[str]*

Returns the set of fields that have been explicitly set on this model instance.

Returns:

A set of strings representing the fields that have been set, i.e. that were not filled from defaults.

classmethod model_json_schema(*by_alias: bool = True*, *ref_template: str = '#/$defs/{model}'*, *schema_generator: type[pydantic.json_schema.GenerateJsonSchema] = <class 'pydantic.json_schema.GenerateJsonSchema'>*, *mode: ~typing.Literal['validation', 'serialization'] = 'validation'*) → dict[str, Any]

Generates a JSON schema for a model class.

Args:

by_alias: Whether to use attribute aliases or not. ref_template: The reference template. schema_generator: To override the logic used to generate the JSON schema, as a subclass of

GenerateJsonSchema with your desired modifications

mode: The mode in which to generate the schema.

Returns:

The JSON schema for the given model class.

classmethod model_parametrized_name(*params: tuple[type[Any], ...]*) → str

Compute the class name for parametrizations of generic classes.

This method can be overridden to achieve a custom naming scheme for generic BaseModels.

Args:

params: Tuple of types of the class. Given a generic class *Model* with 2 type variables and a concrete model *Model[str, int]*, the value *(str, int)* would be passed to *params*.

Returns:

String representing the new class where *params* are passed to *cls* as type variables.

Raises:

TypeError: Raised when trying to generate concrete names for non-generic models.

model_post_init(*_BaseModel__context: Any*) → None

Override this method to perform additional initialization after *__init__* and *model_construct*. This is useful if you want to do some validation that requires the entire model to be initialized.

classmethod model_rebuild(*, *force: bool = False*, *raise_errors: bool = True*, *_parent_namespace_depth: int = 2*, *_types_namespace: Optional[dict[str, Any]] = None*) → bool | None

Try to rebuild the pydantic-core schema for the model.

This may be necessary when one of the annotations is a ForwardRef which could not be resolved during the initial attempt to build the schema, and automatic rebuilding fails.

Args:

force: Whether to force the rebuilding of the model schema, defaults to *False*. raise_errors: Whether to raise errors, defaults to *True*. _parent_namespace_depth: The depth level of the parent namespace, defaults to 2. _types_namespace: The types namespace, defaults to *None*.

Returns:

Returns *None* if the schema is already "complete" and rebuilding was not required. If rebuilding _was_ required, returns *True* if rebuilding was successful, otherwise *False*.

classmethod model_validate(*obj: Any*, *, *strict: bool | None = None*, *from_attributes: bool | None = None*, *context: dict[str, Any] | None = None*) → Model

Validate a pydantic model instance.

Args:

obj: The object to validate. strict: Whether to enforce types strictly. from_attributes: Whether to extract data from object attributes. context: Additional context to pass to the validator.

Raises:

ValidationError: If the object could not be validated.

Returns:

The validated model instance.

classmethod model_validate_json(*json_data: str | bytes | bytearray*, *, *strict: bool | None = None*, *context: dict[str, Any] | None = None*) → Model

Usage docs: https://docs.pydantic.dev/2.6/concepts/json/#json-parsing

Validate the given JSON data against the Pydantic model.

Args:

json_data: The JSON data to validate. strict: Whether to enforce types strictly. context: Extra variables to pass to the validator.

Returns:

The validated Pydantic model.

Raises:

ValueError: If *json_data* is not a JSON string.

classmethod model_validate_strings(*obj: Any*, *, *strict: bool | None = None*, *context: dict[str, Any] | None = None*) → Model

Validate the given object contains string data against the Pydantic model.

Args:

obj: The object contains string data to validate. strict: Whether to enforce types strictly. context: Extra variables to pass to the validator.

Returns:

The validated Pydantic model.

QueryStockQuoteSettingModel

class zvt.trading.trading_models.QueryStockQuoteSettingModel(*, *stock_pool_name: Optional[str] = None*, *main_tags: Optional[List[str]] = None*)

Bases: `CustomModel`

__init__(**data: Any*) → None

Create a new model by parsing and validating input data from keyword arguments.

Raises [*ValidationError*][pydantic_core.ValidationError] if the input data cannot be validated to form a valid model.

self is explicitly positional-only to allow *self* as a field name.

copy(*, *include: AbstractSetIntStr | MappingIntStrAny | None = None*, *exclude: AbstractSetIntStr | MappingIntStrAny | None = None*, *update: Dict[str, Any] | None = None*, *deep: bool = False*) → Model

Returns a copy of the model.

!!! warning "Deprecated"

This method is now deprecated; use *model_copy* instead.

If you need *include* or *exclude*, use:

```
`py data = self.model_dump(include=include, exclude=exclude, round_trip=True) data = {**data, **(update or {})} copied = self.model_validate(data) `
```

Args:

include: Optional set or mapping specifying which fields to include in the copied model. exclude: Optional set or mapping specifying which fields to exclude in the copied model. update:

Optional dictionary of field-value pairs to override field values in the copied model. deep: If True, the values of fields that are Pydantic models will be deep-copied.

Returns:

A copy of the model with included, excluded and updated fields as specified.

model_computed_fields*: ClassVar[dict[str, ComputedFieldInfo]] = {}*

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

model_config*: ClassVar[ConfigDict] = {'allow_inf_nan': True, 'from_attributes': True}*

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

classmethod model_construct(*_fields_set: set[str] | None = None*, ***values: Any*) → Model

Creates a new instance of the *Model* class with validated data.

Creates a new model setting *__dict__* and *__pydantic_fields_set__* from trusted or pre-validated data. Default values are respected, but no other validation is performed. Behaves as if *Config.extra = 'allow'* was set since it adds all passed values

Args:

_fields_set: The set of field names accepted for the Model instance. values: Trusted or pre-validated data dictionary.

Returns:

A new instance of the *Model* class with validated data.

model_copy(*, *update: dict[str, Any] | None = None*, *deep: bool = False*) → Model

Usage docs: https://docs.pydantic.dev/2.6/concepts/serialization/#model_copy

Returns a copy of the model.

Args:

update: Values to change/add in the new model. Note: the data is not validated
before creating the new model. You should trust this data.

deep: Set to *True* to make a deep copy of the model.

Returns:

New model instance.

model_dump(*, *mode: Literal['json', 'python'] | str = 'python'*, *include: IncEx = None*, *exclude: IncEx = None*, *by_alias: bool = False*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*, *round_trip: bool = False*, *warnings: bool = True*) → dict[str, Any]

Usage docs: https://docs.pydantic.dev/2.6/concepts/serialization/#modelmodel_dump

Generate a dictionary representation of the model, optionally specifying which fields to include or exclude.

Args:

mode: The mode in which *to_python* should run.
If mode is 'json', the output will only contain JSON serializable types. If mode is 'python', the output may contain non-JSON-serializable Python objects.

include: A list of fields to include in the output. exclude: A list of fields to exclude from the output. by_alias: Whether to use the field's alias in the dictionary key if defined. exclude_unset: Whether to exclude fields that have not been explicitly set. exclude_defaults: Whether to exclude fields that are set to their default value. exclude_none: Whether to exclude fields that have a value of *None*. round_trip: If True, dumped values should be

valid as input for non-idempotent types such as Json[T]. warnings: Whether to log warnings when invalid fields are encountered.

Returns:

A dictionary representation of the model.

model_dump_json(*, *indent: int | None = None*, *include: IncEx = None*, *exclude: IncEx = None*, *by_alias: bool = False*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*, *round_trip: bool = False*, *warnings: bool = True*) → str

Usage docs: https://docs.pydantic.dev/2.6/concepts/serialization/#modelmodel_dump_json

Generates a JSON representation of the model using Pydantic's *to_json* method.

Args:

indent: Indentation to use in the JSON output. If None is passed, the output will be compact. include: Field(s) to include in the JSON output. exclude: Field(s) to exclude from the JSON output. by_alias: Whether to serialize using field aliases. exclude_unset: Whether to exclude fields that have not been explicitly set. exclude_defaults: Whether to exclude fields that are set to their default value. exclude_none: Whether to exclude fields that have a value of *None*. round_trip: If True, dumped values should be valid as input for non-idempotent types such as Json[T]. warnings: Whether to log warnings when invalid fields are encountered.

Returns:

A JSON string representation of the model.

property model_extra*: dict[str, Any] | None*

Get extra fields set during validation.

Returns:

A dictionary of extra fields, or *None* if *config.extra* is not set to *"allow"*.

model_fields*: ClassVar[dict[str, FieldInfo]] = {'main_tags': FieldInfo(annotation=Union[List[str], NoneType], required=False), 'stock_pool_name': FieldInfo(annotation=Union[str, NoneType], required=False)}*

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

property model_fields_set*: set[str]*

Returns the set of fields that have been explicitly set on this model instance.

Returns:

A set of strings representing the fields that have been set, i.e. that were not filled from defaults.

classmethod model_json_schema(*by_alias: bool = True*, *ref_template: str = '#/$defs/{model}'*, *schema_generator: type[pydantic.json_schema.GenerateJsonSchema] = <class 'pydantic.json_schema.GenerateJsonSchema'>*, *mode: ~typing.Literal['validation', 'serialization'] = 'validation'*) → dict[str, Any]

Generates a JSON schema for a model class.

Args:

by_alias: Whether to use attribute aliases or not. ref_template: The reference template. schema_generator: To override the logic used to generate the JSON schema, as a subclass of

GenerateJsonSchema with your desired modifications

mode: The mode in which to generate the schema.

Returns:

The JSON schema for the given model class.

classmethod model_parametrized_name(*params: tuple[type[Any], ...]*) → str

Compute the class name for parametrizations of generic classes.

This method can be overridden to achieve a custom naming scheme for generic BaseModels.

Args:

params: Tuple of types of the class. Given a generic class *Model* with 2 type variables and a concrete model *Model[str, int]*, the value *(str, int)* would be passed to *params*.

Returns:

String representing the new class where *params* are passed to *cls* as type variables.

Raises:

TypeError: Raised when trying to generate concrete names for non-generic models.

model_post_init(*_BaseModel__context: Any*) → None

Override this method to perform additional initialization after *__init__* and *model_construct*. This is useful if you want to do some validation that requires the entire model to be initialized.

classmethod model_rebuild(*, *force: bool = False*, *raise_errors: bool = True*, *_parent_namespace_depth: int = 2*, *_types_namespace: Optional[dict[str, Any]] = None*) → bool | None

Try to rebuild the pydantic-core schema for the model.

This may be necessary when one of the annotations is a ForwardRef which could not be resolved during the initial attempt to build the schema, and automatic rebuilding fails.

Args:

force: Whether to force the rebuilding of the model schema, defaults to *False*. raise_errors: Whether to raise errors, defaults to *True*. _parent_namespace_depth: The depth level of the parent namespace, defaults to 2. _types_namespace: The types namespace, defaults to *None*.

Returns:
Returns *None* if the schema is already "complete" and rebuilding was not required. If rebuilding _was_ required, returns *True* if rebuilding was successful, otherwise *False*.

classmethod model_validate(*obj: Any*, *, *strict: bool | None = None*, *from_attributes: bool | None = None*, *context: dict[str, Any] | None = None*) → Model

Validate a pydantic model instance.

Args:
obj: The object to validate. strict: Whether to enforce types strictly. from_attributes: Whether to extract data from object attributes. context: Additional context to pass to the validator.

Raises:
ValidationError: If the object could not be validated.

Returns:
The validated model instance.

classmethod model_validate_json(*json_data: str | bytes | bytearray*, *, *strict: bool | None = None*, *context: dict[str, Any] | None = None*) → Model

Usage docs: https://docs.pydantic.dev/2.6/concepts/json/#json-parsing

Validate the given JSON data against the Pydantic model.

Args:

json_data: The JSON data to validate. strict: Whether to enforce types strictly. context: Extra variables to pass to the validator.

Returns:
The validated Pydantic model.

Raises:
ValueError: If *json_data* is not a JSON string.

classmethod model_validate_strings(*obj: Any*, *, *strict: bool | None = None*, *context: dict[str, Any] | None = None*) → Model

Validate the given object contains string data against the Pydantic model.

Args:
obj: The object contains string data to validate. strict: Whether to enforce types strictly. context: Extra variables to pass to the validator.

Returns:
The validated Pydantic model.

QueryTagQuoteModel

class zvt.trading.trading_models.QueryTagQuoteModel(*, *stock_pool_name: str*, *main_tags: List[str]*)

Bases: `CustomModel`

__init__(**data: Any*) → None

Create a new model by parsing and validating input data from keyword arguments.

Raises [*ValidationError*][pydantic_core.ValidationError] if the input data cannot be validated to form a valid model.

self is explicitly positional-only to allow *self* as a field name.

copy(*, *include: AbstractSetIntStr | MappingIntStrAny | None = None*, *exclude: AbstractSetIntStr | MappingIntStrAny | None = None*, *update: Dict[str, Any] | None = None*, *deep: bool = False*) → Model

Returns a copy of the model.

!!! warning "Deprecated"

This method is now deprecated; use *model_copy* instead.

If you need *include* or *exclude*, use:

```
`py data = self.model_dump(include=include, exclude=exclude, round_trip=True) data = {**data, **(update or {})} copied = self.model_validate(data) `
```

Args:

include: Optional set or mapping specifying which fields to include in the copied model. exclude: Optional set or mapping specifying which fields to exclude in the copied model. update: Optional dictionary of field-value pairs to override field values

in the copied model. deep: If True, the values of fields that are Pydantic models will be deep-copied.

Returns:

A copy of the model with included, excluded and updated fields as specified.

model_computed_fields*: ClassVar[dict[str, ComputedFieldInfo]] = {}*

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

model_config*: ClassVar[ConfigDict] = {'allow_inf_nan': True, 'from_attributes': True}*

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

classmethod model_construct(*_fields_set: set[str] | None = None*, ***values: Any*) → Model

Creates a new instance of the *Model* class with validated data.

Creates a new model setting *__dict__* and *__pydantic_fields_set__* from trusted or pre-validated data. Default values are respected, but no other validation is performed. Behaves as if *Config.extra = 'allow'* was set since it adds all passed values

Args:

_fields_set: The set of field names accepted for the Model instance. values: Trusted or pre-validated data dictionary.

Returns:

A new instance of the *Model* class with validated data.

model_copy(*, *update: dict[str, Any] | None = None*, *deep: bool = False*) → Model

Usage docs: https://docs.pydantic.dev/2.6/concepts/serialization/#model_copy

Returns a copy of the model.

Args:

update: Values to change/add in the new model. Note: the data is not validated
before creating the new model. You should trust this data.

deep: Set to *True* to make a deep copy of the model.

Returns:

New model instance.

model_dump(*, *mode: Literal['json', 'python'] | str = 'python'*, *include: IncEx = None*, *exclude: IncEx = None*, *by_alias: bool = False*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*, *round_trip: bool = False*, *warnings: bool = True*) → dict[str, Any]

Usage docs: https://docs.pydantic.dev/2.6/concepts/serialization/#modelmodel_dump

Generate a dictionary representation of the model, optionally specifying which fields to include or exclude.

Args:

mode: The mode in which *to_python* should run.
If mode is 'json', the output will only contain JSON serializable types. If mode is 'python', the output may contain non-JSON-serializable Python objects.

include: A list of fields to include in the output. exclude: A list of fields to exclude from the output. by_alias: Whether to use the field's alias in the dictionary key if defined. exclude_unset: Whether to exclude fields that have not been explicitly set. exclude_defaults: Whether to exclude fields that are set to their default value. exclude_none: Whether to exclude fields that have a value of *None*. round_trip: If True, dumped values should be valid as input for non-idempotent types such as Json[T].

warnings: Whether to log warnings when invalid fields are encountered.

Returns:

A dictionary representation of the model.

model_dump_json(*, *indent: int | None = None*, *include: IncEx = None*, *exclude: IncEx = None*, *by_alias: bool = False*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*, *round_trip: bool = False*, *warnings: bool = True*) → str

Usage docs: https://docs.pydantic.dev/2.6/concepts/serialization/#modelmodel_dump_json

Generates a JSON representation of the model using Pydantic's *to_json* method.

Args:

indent: Indentation to use in the JSON output. If None is passed, the output will be compact. include: Field(s) to include in the JSON output. exclude: Field(s) to exclude from the JSON output. by_alias: Whether to serialize using field aliases. exclude_unset: Whether to exclude fields that have not been explicitly set. exclude_defaults: Whether to exclude fields that are set to their default value. exclude_none: Whether to exclude fields that have a value of *None*. round_trip: If True, dumped values should be valid as input for non-idempotent types such as Json[T]. warnings: Whether to log warnings when invalid fields are encountered.

Returns:

A JSON string representation of the model.

property model_extra*: dict[str, Any] | None*

Get extra fields set during validation.

Returns:

A dictionary of extra fields, or *None* if *config.extra* is not set to *"allow"*.

model_fields*: ClassVar[dict[str, FieldInfo]] = {'main_tags': FieldInfo(annotation=List[str], required=True), 'stock_pool_name': FieldInfo(annotation=str, required=True)}*

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

property model_fields_set*: set[str]*

Returns the set of fields that have been explicitly set on this model instance.

Returns:

A set of strings representing the fields that have been set, i.e. that were not filled from defaults.

classmethod model_json_schema(*by_alias: bool = True*, *ref_template: str = '#/$defs/{model}'*, *schema_generator: type[pydantic.json_schema.GenerateJsonSchema] = <class 'pydantic.json_schema.GenerateJsonSchema'>*, *mode: ~typing.Literal['validation', 'serialization'] = 'validation'*) → dict[str, Any]

Generates a JSON schema for a model class.

Args:

by_alias: Whether to use attribute aliases or not. ref_template: The reference template. schema_generator: To override the logic used to generate the JSON schema, as a subclass of

GenerateJsonSchema with your desired modifications

mode: The mode in which to generate the schema.

Returns:

The JSON schema for the given model class.

classmethod model_parametrized_name(*params: tuple[type[Any], ...]*) → str

Compute the class name for parametrizations of generic classes.

This method can be overridden to achieve a custom naming scheme for generic BaseModels.

Args:

params: Tuple of types of the class. Given a generic class *Model* with 2 type variables and a concrete model *Model[str, int]*, the value *(str, int)* would be passed to *params*.

Returns:

String representing the new class where *params* are passed to *cls* as type variables.

Raises:

TypeError: Raised when trying to generate concrete names for non-generic models.

model_post_init(*_BaseModel__context: Any*) → None

Override this method to perform additional initialization after *__init__* and *model_construct*. This is useful if you want to do some validation that requires the entire model to be initialized.

classmethod model_rebuild(*, *force: bool = False*, *raise_errors: bool = True*, *_parent_namespace_depth: int = 2*, *_types_namespace: Optional[dict[str, Any]] = None*) → bool | None

Try to rebuild the pydantic-core schema for the model.

This may be necessary when one of the annotations is a ForwardRef which could not be resolved during the initial attempt to build the schema, and automatic rebuilding fails.

Args:

force: Whether to force the rebuilding of the model schema, defaults to *False*. raise_errors: Whether to raise errors, defaults to *True*. _parent_namespace_depth: The depth level of the parent namespace, defaults to 2. _types_namespace: The types namespace, defaults to *None*.

Returns:
Returns *None* if the schema is already "complete" and rebuilding was not required. If rebuilding _was_ required, returns *True* if rebuilding was successful, otherwise *False*.

classmethod model_validate(*obj: Any*, *, *strict: bool | None = None*, *from_attributes: bool | None = None*, *context: dict[str, Any] | None = None*) → Model

Validate a pydantic model instance.

Args:
obj: The object to validate. strict: Whether to enforce types strictly. from_attributes: Whether to extract data from object attributes. context: Additional context to pass to the validator.

Raises:
ValidationError: If the object could not be validated.

Returns:
The validated model instance.

classmethod model_validate_json(*json_data: str | bytes | bytearray*, *, *strict: bool | None = None*, *context: dict[str, Any] | None = None*) → Model

Usage docs: https://docs.pydantic.dev/2.6/concepts/json/#json-parsing

Validate the given JSON data against the Pydantic model.

Args:

json_data: The JSON data to validate. strict: Whether to enforce types strictly. context: Extra variables to pass to the validator.

Returns:
The validated Pydantic model.

Raises:
ValueError: If *json_data* is not a JSON string.

classmethod model_validate_strings(*obj: Any*, *, *strict: bool | None = None*, *context: dict[str, Any] | None = None*) → Model

Validate the given object contains string data against the Pydantic model.

Args:
obj: The object contains string data to validate. strict: Whether to enforce types strictly. context: Extra variables to pass to the validator.

Returns:
The validated Pydantic model.

QueryTradingPlanModel

class zvt.trading.trading_models.QueryTradingPlanModel(*, *time_range: TimeRange*)

Bases: **`BaseModel`**

__init__(***data: Any*) → None

Create a new model by parsing and validating input data from keyword arguments.

Raises [*ValidationError*][pydantic_core.ValidationError] if the input data cannot be validated to form a valid model.

self is explicitly positional-only to allow *self* as a field name.

copy(*, *include: AbstractSetIntStr | MappingIntStrAny | None = None*, *exclude: AbstractSetIntStr | MappingIntStrAny | None = None*, *update: Dict[str, Any] | None = None*, *deep: bool = False*) → Model

Returns a copy of the model.

!!! warning "Deprecated"

This method is now deprecated; use *model_copy* instead.

If you need *include* or *exclude*, use:

```
`py data = self.model_dump(include=include, exclude=exclude, round_trip=True) data = {**data, **(update or {})} copied = self.model_validate(data) `
```

Args:

include: Optional set or mapping specifying which fields to include in the copied model. exclude: Optional set or mapping specifying which fields to exclude in the copied model. update: Optional dictionary of field-value pairs to override field values

in the copied model. deep: If True, the values of fields that are Pydantic models will be deep-copied.

Returns:

A copy of the model with included, excluded and updated fields as specified.

model_computed_fields*: ClassVar[dict[str, ComputedFieldInfo]] = {}*

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

model_config*: ClassVar[ConfigDict] = {}*

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

classmethod model_construct(*_fields_set: set[str] | None = None*, ***values: Any*) → Model

Creates a new instance of the *Model* class with validated data.

Creates a new model setting *__dict__* and *__pydantic_fields_set__* from trusted or pre-validated data. Default values are respected, but no other validation is performed. Behaves as if *Config.extra = 'allow'* was set since it adds all passed values

Args:

_fields_set: The set of field names accepted for the Model instance. values: Trusted or pre-validated data dictionary.

Returns:

A new instance of the *Model* class with validated data.

model_copy(*, *update: dict[str, Any] | None = None*, *deep: bool = False*) → Model

Usage docs: https://docs.pydantic.dev/2.6/concepts/serialization/#model_copy

Returns a copy of the model.

Args:

update: Values to change/add in the new model. Note: the data is not validated
before creating the new model. You should trust this data.

deep: Set to *True* to make a deep copy of the model.

Returns:

New model instance.

model_dump(*, *mode: Literal['json', 'python'] | str = 'python'*, *include: IncEx = None*, *exclude: IncEx = None*, *by_alias: bool = False*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*, *round_trip: bool = False*, *warnings: bool = True*) → dict[str, Any]

Usage docs: https://docs.pydantic.dev/2.6/concepts/serialization/#modelmodel_dump

Generate a dictionary representation of the model, optionally specifying which fields to include or exclude.

Args:

mode: The mode in which *to_python* should run.
If mode is 'json', the output will only contain JSON serializable types. If mode is 'python', the output may contain non-JSON-serializable Python objects.

include: A list of fields to include in the output. exclude: A list of fields to exclude from the output. by_alias: Whether to use the field's alias in the dictionary key if defined. exclude_unset: Whether to exclude fields that have not been explicitly set. exclude_defaults: Whether to exclude fields that are set to their default value. exclude_none: Whether to exclude fields that have a value of *None*. round_trip: If True, dumped values should be valid as input for non-idempotent types such as Json[T].

warnings: Whether to log warnings when invalid fields are encountered.

Returns:

A dictionary representation of the model.

model_dump_json(*, *indent: int | None = None*, *include: IncEx = None*, *exclude: IncEx = None*, *by_alias: bool = False*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*, *round_trip: bool = False*, *warnings: bool = True*) → str

Usage docs: https://docs.pydantic.dev/2.6/concepts/serialization/#modelmodel_dump_json

Generates a JSON representation of the model using Pydantic's *to_json* method.

Args:

indent: Indentation to use in the JSON output. If None is passed, the output will be compact. include: Field(s) to include in the JSON output. exclude: Field(s) to exclude from the JSON output. by_alias: Whether to serialize using field aliases. exclude_unset: Whether to exclude fields that have not been explicitly set. exclude_defaults: Whether to exclude fields that are set to their default value. exclude_none: Whether to exclude fields that have a value of *None*. round_trip: If True, dumped values should be valid as input for non-idempotent types such as Json[T]. warnings: Whether to log warnings when invalid fields are encountered.

Returns:

A JSON string representation of the model.

property model_extra*: dict[str, Any] | None*

Get extra fields set during validation.

Returns:

A dictionary of extra fields, or *None* if *config.extra* is not set to *"allow"*.

model_fields*: ClassVar[dict[str, FieldInfo]] = {'time_range': FieldInfo(annotation=TimeRange, required=True)}*

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

property model_fields_set*: set[str]*

Returns the set of fields that have been explicitly set on this model instance.

Returns:

A set of strings representing the fields that have been set, i.e. that were not filled from defaults.

classmethod model_json_schema(*by_alias: bool = True*, *ref_template: str = '#/$defs/{model}'*, *schema_generator: type[pydantic.json_schema.GenerateJsonSchema] = <class 'pydantic.json_schema.GenerateJsonSchema'>*, *mode: ~typing.Literal['validation', 'serialization'] = 'validation'*) → dict[str, Any]

Generates a JSON schema for a model class.

Args:

by_alias: Whether to use attribute aliases or not. ref_template: The reference template. schema_generator: To override the logic used to generate the JSON schema, as a subclass of

GenerateJsonSchema with your desired modifications

mode: The mode in which to generate the schema.

Returns:

The JSON schema for the given model class.

classmethod model_parametrized_name(*params: tuple[type[Any], ...]*) → str

Compute the class name for parametrizations of generic classes.

This method can be overridden to achieve a custom naming scheme for generic BaseModels.

Args:

params: Tuple of types of the class. Given a generic class *Model* with 2 type variables and a concrete model *Model[str, int]*, the value *(str, int)* would be passed to *params*.

Returns:

String representing the new class where *params* are passed to *cls* as type variables.

Raises:

TypeError: Raised when trying to generate concrete names for non-generic models.

model_post_init(*_BaseModel__context: Any*) → None

Override this method to perform additional initialization after *__init__* and *model_construct*. This is useful if you want to do some validation that requires the entire model to be initialized.

classmethod model_rebuild(*, *force: bool = False*, *raise_errors: bool = True*, *_parent_namespace_depth: int = 2*, *_types_namespace: Optional[dict[str, Any]] = None*) → bool | None

Try to rebuild the pydantic-core schema for the model.

This may be necessary when one of the annotations is a ForwardRef which could not be resolved during the initial attempt to build the schema, and automatic rebuilding fails.

Args:

force: Whether to force the rebuilding of the model schema, defaults to *False*. raise_errors: Whether to raise errors, defaults

to *True*. _parent_namespace_depth: The depth level of the parent namespace, defaults to 2. _types_namespace: The types namespace, defaults to *None*.

Returns:

Returns *None* if the schema is already "complete" and rebuilding was not required. If rebuilding _was_ required, returns *True* if rebuilding was successful, otherwise *False*.

classmethod model_validate(*obj: Any*, *, *strict: bool | None = None*, *from_attributes: bool | None = None*, *context: dict[str, Any] | None = None*) → Model

Validate a pydantic model instance.

Args:

obj: The object to validate. strict: Whether to enforce types strictly. from_attributes: Whether to extract data from object attributes. context: Additional context to pass to the validator.

Raises:

ValidationError: If the object could not be validated.

Returns:

The validated model instance.

classmethod model_validate_json(*json_data: str | bytes | bytearray*, *, *strict: bool | None = None*, *context: dict[str, Any] | None = None*) → Model

Usage docs: https://docs.pydantic.dev/2.6/concepts/json/#json-parsing

Validate the given JSON data against the Pydantic model.

Args:

json_data: The JSON data to validate. strict: Whether to enforce types strictly. context: Extra variables to pass to the validator.

Returns:

The validated Pydantic model.

Raises:

ValueError: If *json_data* is not a JSON string.

classmethod model_validate_strings(*obj: Any*, *, *strict: bool | None = None*, *context: dict[str, Any] | None = None*) → Model

Validate the given object contains string data against the Pydantic model.

Args:

obj: The object contains string data to validate. strict: Whether to enforce types strictly. context: Extra variables to pass to the validator.

Returns:

The validated Pydantic model.

StockQuoteModel

class zvt.trading.trading_models.StockQuoteModel(*, *id: str*, *entity_id: str*, *timestamp: datetime*, *code: str*, *name: str*, *time: int*, *price: float*, *change_pct: float*, *turnover: float*, *turnover_rate: float*, *is_limit_up: bool*, *limit_up_amount: Optional[float] = None*, *is_limit_down: bool*, *limit_down_amount: Optional[float] = None*, *ask_amount: float*, *bid_amount: float*, *float_cap: float*, *total_cap: float*, *main_tag: str = None*, *sub_tag: Optional[str] = None*, *hidden_tags: Optional[List[str]] = None*)

Bases: **MixinModel**

__init__(**data: Any*) → None

Create a new model by parsing and validating input data from keyword arguments.

Raises [*ValidationError*][pydantic_core.ValidationError] if the input data cannot be validated to form a valid model.

self is explicitly positional-only to allow *self* as a field name.

code*: str*

代码

name*: str*

名字

time*: int*

UNIX时间戳

price*: float*

最新价

is_limit_up*: bool*

是否涨停

limit_up_amount*: Optional[float]*

封涨停金额

is_limit_down*: bool*

是否跌停

limit_down_amount*: Optional[float]*

封跌停金额

ask_amount*: float*

5挡卖单金额

bid_amount*: float*

5挡买单金额

float_cap*: float*

流通市值

total_cap*: float*

总市值

copy(*, *include: AbstractSetIntStr | MappingIntStrAny | None = None*, *exclude: AbstractSetIntStr | MappingIntStrAny | None = None*, *update: Dict[str, Any] | None = None*, *deep: bool = False*) → Model

Returns a copy of the model.

!!! warning "Deprecated"

This method is now deprecated; use *model_copy* instead.

If you need *include* or *exclude*, use:

```
`py data = self.model_dump(include=include,
exclude=exclude, round_trip=True) data = {**data, **
(update or {})} copied = self.model_validate(data) `
```

Args:

include: Optional set or mapping specifying which fields to include in the copied model. exclude: Optional set or mapping specifying which fields to exclude in the copied model. update: Optional dictionary of field-value pairs to override field values in the copied model. deep: If True, the values of fields that are Pydantic models will be deep-copied.

Returns:

A copy of the model with included, excluded and updated fields as specified.

model_computed_fields*: ClassVar[dict[str, ComputedFieldInfo]] = {}*

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

model_config*: ClassVar[ConfigDict] = {'allow_inf_nan': True, 'from_attributes': True}*

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

classmethod model_construct(*_fields_set: set[str] | None = None*, ***values: Any*) → Model

Creates a new instance of the *Model* class with validated data.

Creates a new model setting *__dict__* and *__pydantic_fields_set__* from trusted or pre-validated data. Default values are respected, but no other validation is performed. Behaves as if *Config.extra = 'allow'* was set since it adds all passed values

Args:

_fields_set: The set of field names accepted for the Model instance. values: Trusted or pre-validated data dictionary.

Returns:

A new instance of the *Model* class with validated data.

model_copy(*, *update: dict[str, Any] | None = None*, *deep: bool = False*) → Model

Usage docs: https://docs.pydantic.dev/2.6/concepts/serialization/#model_copy

Returns a copy of the model.

Args:

update: Values to change/add in the new model. Note: the data is not validated
before creating the new model. You should trust this data.

deep: Set to *True* to make a deep copy of the model.

Returns:

New model instance.

model_dump(*, *mode: Literal['json', 'python'] | str = 'python'*, *include: IncEx = None*, *exclude: IncEx = None*, *by_alias: bool = False*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*, *round_trip: bool = False*, *warnings: bool = True*) → dict[str, Any]

Usage docs: https://docs.pydantic.dev/2.6/concepts/serialization/#modelmodel_dump

Generate a dictionary representation of the model, optionally specifying which fields to include or exclude.

Args:

mode: The mode in which *to_python* should run.
If mode is 'json', the output will only contain JSON serializable types. If mode is 'python', the output may contain non-JSON-serializable Python objects.

include: A list of fields to include in the output. exclude: A list of fields to exclude from the output. by_alias: Whether to use the

field's alias in the dictionary key if defined. exclude_unset: Whether to exclude fields that have not been explicitly set. exclude_defaults: Whether to exclude fields that are set to their default value. exclude_none: Whether to exclude fields that have a value of *None*. round_trip: If True, dumped values should be valid as input for non-idempotent types such as Json[T]. warnings: Whether to log warnings when invalid fields are encountered.

Returns:

A dictionary representation of the model.

model_dump_json(*, *indent: int | None = None*, *include: IncEx = None*, *exclude: IncEx = None*, *by_alias: bool = False*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*, *round_trip: bool = False*, *warnings: bool = True*) → str

Usage docs: https://docs.pydantic.dev/2.6/concepts/serialization/#modelmodel_dump_json

Generates a JSON representation of the model using Pydantic's *to_json* method.

Args:

indent: Indentation to use in the JSON output. If None is passed, the output will be compact. include: Field(s) to include in the JSON output. exclude: Field(s) to exclude from the JSON output. by_alias: Whether to serialize using field aliases. exclude_unset: Whether to exclude fields that have not been explicitly set. exclude_defaults: Whether to exclude fields that are set to their default value. exclude_none: Whether to exclude fields that have a value of *None*. round_trip: If True, dumped values should be valid as input for non-idempotent types such as Json[T]. warnings: Whether to log warnings when invalid fields are encountered.

Returns:

A JSON string representation of the model.

property model_extra*: dict[str, Any] | None*

Get extra fields set during validation.

Returns:

A dictionary of extra fields, or *None* if *config.extra* is not set to *"allow"*.

model_fields*: ClassVar[dict[str, FieldInfo]] = {'ask_amount': FieldInfo(annotation=float, required=True), 'bid_amount': FieldInfo(annotation=float, required=True), 'change_pct': FieldInfo(annotation=float, required=True), 'code': FieldInfo(annotation=str, required=True), 'entity_id': FieldInfo(annotation=str, required=True), 'float_cap': FieldInfo(annotation=float, required=True), 'hidden_tags': FieldInfo(annotation=Union[List[str], NoneType], required=False), 'id': FieldInfo(annotation=str, required=True), 'is_limit_down': FieldInfo(annotation=bool, required=True), 'is_limit_up': FieldInfo(annotation=bool, required=True), 'limit_down_amount': FieldInfo(annotation=Union[float, NoneType], required=False), 'limit_up_amount': FieldInfo(annotation=Union[float, NoneType], required=False), 'main_tag': FieldInfo(annotation=str, required=False), 'name': FieldInfo(annotation=str, required=True), 'price': FieldInfo(annotation=float, required=True), 'sub_tag': FieldInfo(annotation=Union[str, NoneType], required=False), 'time': FieldInfo(annotation=int, required=True), 'timestamp': FieldInfo(annotation=datetime, required=True), 'total_cap': FieldInfo(annotation=float, required=True), 'turnover': FieldInfo(annotation=float, required=True), 'turnover_rate': FieldInfo(annotation=float, required=True)}*

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

property model_fields_set*: set[str]*

Returns the set of fields that have been explicitly set on this model instance.

Returns:

A set of strings representing the fields that have been set, i.e. that were not filled from defaults.

classmethod model_json_schema(*by_alias: bool = True*, *ref_template: str = '#/$defs/{model}'*, *schema_generator: type[pydantic.json_schema.GenerateJsonSchema] = <class 'pydantic.json_schema.GenerateJsonSchema'>*, *mode: ~typing.Literal['validation', 'serialization'] = 'validation'*) → dict[str, Any]

Generates a JSON schema for a model class.

Args:

by_alias: Whether to use attribute aliases or not. ref_template: The reference template. schema_generator: To override the logic used to generate the JSON schema, as a subclass of

GenerateJsonSchema with your desired modifications

mode: The mode in which to generate the schema.

Returns:

The JSON schema for the given model class.

classmethod model_parametrized_name(*params: tuple[type[Any], ...]*) → str

Compute the class name for parametrizations of generic classes.

This method can be overridden to achieve a custom naming scheme for generic BaseModels.

Args:

params: Tuple of types of the class. Given a generic class *Model* with 2 type variables and a concrete model *Model[str, int]*, the value *(str, int)* would be passed to *params*.

Returns:

String representing the new class where *params* are passed to *cls* as type variables.

Raises:

TypeError: Raised when trying to generate concrete names for non-generic models.

model_post_init(*_BaseModel__context: Any*) → None

Override this method to perform additional initialization after *__init__* and *model_construct*. This is useful if you want to do some validation that requires the entire model to be initialized.

classmethod model_rebuild(*, *force: bool = False*, *raise_errors: bool = True*, *_parent_namespace_depth: int = 2*, *_types_namespace: Optional[dict[str, Any]] = None*) → bool | None

Try to rebuild the pydantic-core schema for the model.

This may be necessary when one of the annotations is a ForwardRef which could not be resolved during the initial attempt to build the schema, and automatic rebuilding fails.

Args:

force: Whether to force the rebuilding of the model schema, defaults to *False*. raise_errors: Whether to raise errors, defaults to *True*. _parent_namespace_depth: The depth level of the parent namespace, defaults to 2. _types_namespace: The types namespace, defaults to *None*.

Returns:

Returns *None* if the schema is already "complete" and rebuilding was not required. If rebuilding _was_ required, returns *True* if rebuilding was successful, otherwise *False*.

classmethod model_validate(*obj: Any*, *, *strict: bool | None = None*, *from_attributes: bool | None = None*, *context: dict[str, Any] | None = None*) → Model

Validate a pydantic model instance.

Args:
obj: The object to validate. strict: Whether to enforce types strictly. from_attributes: Whether to extract data from object attributes. context: Additional context to pass to the validator.

Raises:
ValidationError: If the object could not be validated.

Returns:
The validated model instance.

classmethod model_validate_json(*json_data: str | bytes | bytearray*, *, *strict: bool | None = None*, *context: dict[str, Any] | None = None*) → Model

Usage docs: https://docs.pydantic.dev/2.6/concepts/json/#json-parsing

Validate the given JSON data against the Pydantic model.

Args:
json_data: The JSON data to validate. strict: Whether to enforce types strictly. context: Extra variables to pass to the validator.

Returns:
The validated Pydantic model.

Raises:
ValueError: If *json_data* is not a JSON string.

classmethod model_validate_strings(*obj: Any*, *, *strict: bool | None = None*, *context: dict[str, Any] | None = None*) → Model

Validate the given object contains string data against the Pydantic model.

Args:
obj: The object contains string data to validate. strict: Whether to enforce types strictly. context: Extra variables to pass to the validator.

Returns:

The validated Pydantic model.

StockQuoteStatsModel

class zvt.trading.trading_models.StockQuoteStatsModel(*, *limit_up_count: int*, *limit_down_count: int*, *up_count: int*, *down_count: int*, *change_pct: float*, *turnover: float*, *quotes: List[StockQuoteModel]*)

Bases: `CustomModel`

__init__(**data: Any*) → None

Create a new model by parsing and validating input data from keyword arguments.

Raises [*ValidationError*][pydantic_core.ValidationError] if the input data cannot be validated to form a valid model.

self is explicitly positional-only to allow *self* as a field name.

limit_up_count*: int*

涨停数

limit_down_count*: int*

跌停数

up_count*: int*

上涨数

down_count*: int*

下跌数

change_pct*: float*

涨幅

turnover*: float*

成交额

copy(*, *include: AbstractSetIntStr | MappingIntStrAny | None = None*, *exclude: AbstractSetIntStr | MappingIntStrAny | None = None*, *update: Dict[str, Any] | None = None*, *deep: bool = False*) → Model

Returns a copy of the model.

!!! warning "Deprecated"

This method is now deprecated; use *model_copy* instead.

If you need *include* or *exclude*, use:

```
`py data = self.model_dump(include=include, exclude=exclude, round_trip=True) data = {**data, **(update or {})} copied = self.model_validate(data) `
```

Args:

include: Optional set or mapping specifying which fields to include in the copied model. exclude: Optional set or mapping specifying which fields to exclude in the copied model. update: Optional dictionary of field-value pairs to override field values in the copied model. deep: If True, the values of fields that are Pydantic models will be deep-copied.

Returns:

A copy of the model with included, excluded and updated fields as specified.

model_computed_fields*: ClassVar[dict[str, ComputedFieldInfo]] = {}*

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

model_config*: ClassVar[ConfigDict] = {'allow_inf_nan': True, 'from_attributes': True}*

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

classmethod model_construct(*_fields_set: set[str] | None = None*, ***values: Any*) → Model

Creates a new instance of the *Model* class with validated data.

Creates a new model setting *__dict__* and *__pydantic_fields_set__* from trusted or pre-validated data. Default values are respected, but no other validation is performed. Behaves as if *Config.extra = 'allow'* was set since it adds all passed values

Args:
_fields_set: The set of field names accepted for the Model instance. values: Trusted or pre-validated data dictionary.

Returns:
A new instance of the *Model* class with validated data.

model_copy(*, *update: dict[str, Any] | None = None*, *deep: bool = False*) → Model

Usage docs: https://docs.pydantic.dev/2.6/concepts/serialization/#model_copy

Returns a copy of the model.

Args:
update: Values to change/add in the new model. Note: the data is not validated
before creating the new model. You should trust this data.

deep: Set to *True* to make a deep copy of the model.

Returns:
New model instance.

model_dump(*, *mode: Literal['json', 'python'] | str = 'python'*, *include: IncEx = None*, *exclude: IncEx = None*, *by_alias: bool = False*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*, *round_trip: bool = False*, *warnings: bool = True*) → dict[str, Any]

Usage docs:
https://docs.pydantic.dev/2.6/concepts/serialization/#modelmodel_dump

Generate a dictionary representation of the model, optionally specifying which fields to include or exclude.

Args:

mode: The mode in which *to_python* should run.
If mode is 'json', the output will only contain JSON serializable types. If mode is 'python', the output may contain non-JSON-serializable Python objects.

include: A list of fields to include in the output. exclude: A list of fields to exclude from the output. by_alias: Whether to use the field's alias in the dictionary key if defined. exclude_unset: Whether to exclude fields that have not been explicitly set. exclude_defaults: Whether to exclude fields that are set to their default value. exclude_none: Whether to exclude fields that have a value of *None*. round_trip: If True, dumped values should be valid as input for non-idempotent types such as Json[T]. warnings: Whether to log warnings when invalid fields are encountered.

Returns:

A dictionary representation of the model.

model_dump_json(*, *indent: int | None = None*, *include: IncEx = None*, *exclude: IncEx = None*, *by_alias: bool = False*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*, *round_trip: bool = False*, *warnings: bool = True*) → str

Usage docs:
https://docs.pydantic.dev/2.6/concepts/serialization/#modelmodel_dump_json

Generates a JSON representation of the model using Pydantic's *to_json* method.

Args:

indent: Indentation to use in the JSON output. If None is passed, the output will be compact. include: Field(s) to include in the JSON output. exclude: Field(s) to exclude from the JSON output. by_alias: Whether to serialize using field aliases. exclude_unset: Whether to exclude fields that have not been explicitly set. exclude_defaults: Whether to exclude fields that are set to their default value. exclude_none: Whether to exclude fields that have a value of *None*. round_trip: If True, dumped values should be valid as input for non-idempotent types such as Json[T]. warnings: Whether to log warnings when invalid fields are encountered.

Returns:

A JSON string representation of the model.

property model_extra*: dict[str, Any] | None*

Get extra fields set during validation.

Returns:

A dictionary of extra fields, or *None* if *config.extra* is not set to *"allow"*.

model_fields*: ClassVar[dict[str, FieldInfo]] = {'change_pct': FieldInfo(annotation=float, required=True), 'down_count': FieldInfo(annotation=int, required=True), 'limit_down_count': FieldInfo(annotation=int, required=True), 'limit_up_count': FieldInfo(annotation=int, required=True), 'quotes': FieldInfo(annotation=List[StockQuoteModel], required=True), 'turnover': FieldInfo(annotation=float, required=True), 'up_count': FieldInfo(annotation=int, required=True)}*

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

property model_fields_set*: set[str]*

Returns the set of fields that have been explicitly set on this model instance.

Returns:
A set of strings representing the fields that have been set,
i.e. that were not filled from defaults.

classmethod model_json_schema(*by_alias: bool = True*, *ref_template: str = '#/$defs/{model}'*, *schema_generator: type[pydantic.json_schema.GenerateJsonSchema] = <class 'pydantic.json_schema.GenerateJsonSchema'>*, *mode: ~typing.Literal['validation', 'serialization'] = 'validation'*) → dict[str, Any]

Generates a JSON schema for a model class.

Args:
by_alias: Whether to use attribute aliases or not. ref_template: The reference template. schema_generator: To override the logic used to generate the JSON schema, as a subclass of

GenerateJsonSchema with your desired modifications

mode: The mode in which to generate the schema.

Returns:
The JSON schema for the given model class.

classmethod model_parametrized_name(*params: tuple[type[Any], ...]*) → str

Compute the class name for parametrizations of generic classes.

This method can be overridden to achieve a custom naming scheme for generic BaseModels.

Args:
params: Tuple of types of the class. Given a generic class

Model with 2 type variables and a concrete model *Model[str, int]*, the value *(str, int)* would be passed to *params*.

Returns:

String representing the new class where *params* are passed to *cls* as type variables.

Raises:

TypeError: Raised when trying to generate concrete names for non-generic models.

model_post_init(*_BaseModel__context: Any*) → None

Override this method to perform additional initialization after *__init__* and *model_construct*. This is useful if you want to do some validation that requires the entire model to be initialized.

classmethod model_rebuild(*, *force: bool = False*, *raise_errors: bool = True*, *_parent_namespace_depth: int = 2*, *_types_namespace: Optional[dict[str, Any]] = None*) → bool | None

Try to rebuild the pydantic-core schema for the model.

This may be necessary when one of the annotations is a ForwardRef which could not be resolved during the initial attempt to build the schema, and automatic rebuilding fails.

Args:

force: Whether to force the rebuilding of the model schema, defaults to *False*. raise_errors: Whether to raise errors, defaults to *True*. _parent_namespace_depth: The depth level of the parent namespace, defaults to 2. _types_namespace: The types namespace, defaults to *None*.

Returns:

Returns *None* if the schema is already “complete” and rebuilding was not required. If rebuilding _was_ required, returns *True* if rebuilding was successful, otherwise *False*.

classmethod model_validate(*obj: Any*, *, *strict: bool | None = None*, *from_attributes: bool | None = None*, *context: dict[str, Any] | None = None*) → Model

Validate a pydantic model instance.

Args:

obj: The object to validate. strict: Whether to enforce types strictly. from_attributes: Whether to extract data from object attributes. context: Additional context to pass to the validator.

Raises:

ValidationError: If the object could not be validated.

Returns:

The validated model instance.

classmethod model_validate_json(*json_data: str | bytes | bytearray*, *, *strict: bool | None = None*, *context: dict[str, Any] | None = None*) → Model

Usage docs: https://docs.pydantic.dev/2.6/concepts/json/#json-parsing

Validate the given JSON data against the Pydantic model.

Args:

json_data: The JSON data to validate. strict: Whether to enforce types strictly. context: Extra variables to pass to the validator.

Returns:

The validated Pydantic model.

Raises:

ValueError: If *json_data* is not a JSON string.

classmethod model_validate_strings(*obj: Any*, *, *strict: bool | None = None*, *context: dict[str, Any] | None = None*) → Model

Validate the given object contains string data against the Pydantic model.

Args:
    obj: The object contains string data to validate. strict: Whether to enforce types strictly. context: Extra variables to pass to the validator.

Returns:
    The validated Pydantic model.

TagQuoteStatsModel

class zvt.trading.trading_models.TagQuoteStatsModel(*, *main_tag: str*, *limit_up_count: int*, *limit_down_count: int*, *up_count: int*, *down_count: int*, *change_pct: float*, *turnover: float*)

Bases: `CustomModel`

__init__(**data: Any*) → None

Create a new model by parsing and validating input data from keyword arguments.

Raises [*ValidationError*][pydantic_core.ValidationError] if the input data cannot be validated to form a valid model.

self is explicitly positional-only to allow *self* as a field name.

limit_up_count*: int*

涨停数

limit_down_count*: int*

跌停数

up_count*: int*

上涨数

down_count*: int*

下跌数

change_pct*: float*

涨幅

turnover*: float*

成交额

copy(*, *include: AbstractSetIntStr | MappingIntStrAny | None = None*, *exclude: AbstractSetIntStr | MappingIntStrAny | None = None*, *update: Dict[str, Any] | None = None*, *deep: bool = False*) → Model

Returns a copy of the model.

!!! warning "Deprecated"

This method is now deprecated; use *model_copy* instead.

If you need *include* or *exclude*, use:

```
`py data = self.model_dump(include=include, exclude=exclude, round_trip=True) data = {**data, **(update or {})} copied = self.model_validate(data) `
```

Args:

include: Optional set or mapping specifying which fields to include in the copied model. exclude: Optional set or mapping specifying which fields to exclude in the copied model. update: Optional dictionary of field-value pairs to override field values in the copied model. deep: If True, the values of fields that are Pydantic models will be deep-copied.

Returns:

A copy of the model with included, excluded and updated fields as specified.

model_computed_fields*: ClassVar[dict[str, ComputedFieldInfo]] = {}*

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

model_config*: ClassVar[ConfigDict] = {'allow_inf_nan': True, 'from_attributes': True}*

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

classmethod model_construct(*_fields_set: set[str] | None = None*, ***values: Any*) → Model

Creates a new instance of the *Model* class with validated data.

Creates a new model setting *__dict__* and *__pydantic_fields_set__* from trusted or pre-validated data. Default values are respected, but no other validation is performed. Behaves as if *Config.extra = 'allow'* was set since it adds all passed values

Args:
_fields_set: The set of field names accepted for the Model instance. values: Trusted or pre-validated data dictionary.

Returns:
A new instance of the *Model* class with validated data.

model_copy(*, *update: dict[str, Any] | None = None*, *deep: bool = False*) → Model

Usage docs: https://docs.pydantic.dev/2.6/concepts/serialization/#model_copy

Returns a copy of the model.

Args:
update: Values to change/add in the new model. Note: the data is not validated
before creating the new model. You should trust this data.

deep: Set to *True* to make a deep copy of the model.

Returns:
New model instance.

model_dump(*, *mode: Literal['json', 'python'] | str = 'python'*, *include: IncEx = None*, *exclude: IncEx = None*, *by_alias: bool = False*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*, *round_trip: bool = False*, *warnings: bool = True*) → dict[str, Any]

Usage docs:
https://docs.pydantic.dev/2.6/concepts/serialization/#modelmodel_dump

Generate a dictionary representation of the model, optionally specifying which fields to include or exclude.

Args:

mode: The mode in which *to_python* should run.
If mode is 'json', the output will only contain JSON serializable types. If mode is 'python', the output may contain non-JSON-serializable Python objects.

include: A list of fields to include in the output. exclude: A list of fields to exclude from the output. by_alias: Whether to use the field's alias in the dictionary key if defined. exclude_unset: Whether to exclude fields that have not been explicitly set. exclude_defaults: Whether to exclude fields that are set to their default value. exclude_none: Whether to exclude fields that have a value of *None*. round_trip: If True, dumped values should be valid as input for non-idempotent types such as Json[T]. warnings: Whether to log warnings when invalid fields are encountered.

Returns:

A dictionary representation of the model.

model_dump_json(*, *indent: int | None = None*, *include: IncEx = None*, *exclude: IncEx = None*, *by_alias: bool = False*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*, *round_trip: bool = False*, *warnings: bool = True*) → str

Usage docs:
https://docs.pydantic.dev/2.6/concepts/serialization/#modelmodel_dump_json

Generates a JSON representation of the model using Pydantic's *to_json* method.

Args:

indent: Indentation to use in the JSON output. If None is passed, the output will be compact. include: Field(s) to include in the JSON output. exclude: Field(s) to exclude from the JSON output. by_alias: Whether to serialize using field aliases. exclude_unset: Whether to exclude fields that have not been explicitly set. exclude_defaults: Whether to exclude fields that are set to their default value. exclude_none: Whether to exclude fields that have a value of *None*. round_trip: If True, dumped values should be valid as input for non-idempotent types such as Json[T]. warnings: Whether to log warnings when invalid fields are encountered.

Returns:

A JSON string representation of the model.

property model_extra*: dict[str, Any] | None*

Get extra fields set during validation.

Returns:

A dictionary of extra fields, or *None* if *config.extra* is not set to *"allow"*.

model_fields*: ClassVar[dict[str, FieldInfo]] = {'change_pct': FieldInfo(annotation=float, required=True), 'down_count': FieldInfo(annotation=int, required=True), 'limit_down_count': FieldInfo(annotation=int, required=True), 'limit_up_count': FieldInfo(annotation=int, required=True), 'main_tag': FieldInfo(annotation=str, required=True), 'turnover': FieldInfo(annotation=float, required=True), 'up_count': FieldInfo(annotation=int, required=True)}*

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

property model_fields_set*: set[str]*

Returns the set of fields that have been explicitly set on this model instance.

Returns:
A set of strings representing the fields that have been set,
i.e. that were not filled from defaults.

classmethod model_json_schema(*by_alias: bool = True*, *ref_template: str = '#/$defs/{model}'*, *schema_generator: type[pydantic.json_schema.GenerateJsonSchema] = <class 'pydantic.json_schema.GenerateJsonSchema'>*, *mode: ~typing.Literal['validation', 'serialization'] = 'validation'*) → dict[str, Any]

Generates a JSON schema for a model class.

Args:
by_alias: Whether to use attribute aliases or not. ref_template: The reference template. schema_generator: To override the logic used to generate the JSON schema, as a subclass of

GenerateJsonSchema with your desired modifications

mode: The mode in which to generate the schema.

Returns:
The JSON schema for the given model class.

classmethod model_parametrized_name(*params: tuple[type[Any], ...]*) → str

Compute the class name for parametrizations of generic classes.

This method can be overridden to achieve a custom naming scheme for generic BaseModels.

Args:
params: Tuple of types of the class. Given a generic class

Model with 2 type variables and a concrete model *Model[str, int]*, the value *(str, int)* would be passed to *params*.

Returns:

String representing the new class where *params* are passed to *cls* as type variables.

Raises:

TypeError: Raised when trying to generate concrete names for non-generic models.

model_post_init(*_BaseModel__context: Any*) → None

Override this method to perform additional initialization after *__init__* and *model_construct*. This is useful if you want to do some validation that requires the entire model to be initialized.

classmethod model_rebuild(*, *force: bool = False*, *raise_errors: bool = True*, *_parent_namespace_depth: int = 2*, *_types_namespace: Optional[dict[str, Any]] = None*) → bool | None

Try to rebuild the pydantic-core schema for the model.

This may be necessary when one of the annotations is a ForwardRef which could not be resolved during the initial attempt to build the schema, and automatic rebuilding fails.

Args:

force: Whether to force the rebuilding of the model schema, defaults to *False*. raise_errors: Whether to raise errors, defaults to *True*. _parent_namespace_depth: The depth level of the parent namespace, defaults to 2. _types_namespace: The types namespace, defaults to *None*.

Returns:

Returns *None* if the schema is already “complete” and rebuilding was not required. If rebuilding _was_ required, returns *True* if rebuilding was successful, otherwise *False*.

classmethod model_validate(*obj: Any*, *, *strict: bool | None = None*, *from_attributes: bool | None = None*, *context: dict[str, Any] | None = None*) → Model

Validate a pydantic model instance.

Args:

obj: The object to validate. strict: Whether to enforce types strictly. from_attributes: Whether to extract data from object attributes. context: Additional context to pass to the validator.

Raises:

ValidationError: If the object could not be validated.

Returns:

The validated model instance.

classmethod model_validate_json(*json_data: str | bytes | bytearray*, *, *strict: bool | None = None*, *context: dict[str, Any] | None = None*) → Model

Usage docs: https://docs.pydantic.dev/2.6/concepts/json/#json-parsing

Validate the given JSON data against the Pydantic model.

Args:

json_data: The JSON data to validate. strict: Whether to enforce types strictly. context: Extra variables to pass to the validator.

Returns:

The validated Pydantic model.

Raises:

ValueError: If *json_data* is not a JSON string.

classmethod model_validate_strings(*obj: Any*, *, *strict: bool | None = None*, *context: dict[str, Any] | None = None*) → Model

Validate the given object contains string data against the Pydantic model.

Args:
    obj: The object contains string data to validate. strict: Whether to enforce types strictly. context: Extra variables to pass to the validator.

Returns:
    The validated Pydantic model.

TradingPlanModel

class zvt.trading.trading_models.TradingPlanModel(*, *id: str*, *entity_id: str*, *timestamp: datetime*, *stock_id: str*, *stock_code: str*, *stock_name: str*, *trading_date: datetime*, *expected_open_pct: float*, *buy_price: Optional[float]*, *sell_price: Optional[float]*, *trading_reason: str*, *trading_signal_type: TradingSignalType*, *status: ExecutionStatus* = *ExecutionStatus.init*, *review: Optional[str]*)

Bases: `MixinModel`

__init__(**data: Any*) → None

Create a new model by parsing and validating input data from keyword arguments.

Raises [*ValidationError*][pydantic_core.ValidationError] if the input data cannot be validated to form a valid model.

self is explicitly positional-only to allow *self* as a field name.

copy(*, *include: AbstractSetIntStr | MappingIntStrAny | None = None*, *exclude: AbstractSetIntStr | MappingIntStrAny | None = None*, *update: Dict[str, Any] | None = None*, *deep: bool = False*) → Model

Returns a copy of the model.

!!! warning "Deprecated"

This method is now deprecated; use *model_copy* instead.

If you need *include* or *exclude*, use:

```
`py data = self.model_dump(include=include, exclude=exclude, round_trip=True) data = {**data, **(update or {})} copied = self.model_validate(data) `
```

Args:

include: Optional set or mapping specifying which fields to include in the copied model. exclude: Optional set or mapping specifying which fields to exclude in the copied model. update: Optional dictionary of field-value pairs to override field values in the copied model. deep: If True, the values of fields that are Pydantic models will be deep-copied.

Returns:

A copy of the model with included, excluded and updated fields as specified.

model_computed_fields*: ClassVar[dict[str, ComputedFieldInfo]] = {}*

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

model_config*: ClassVar[ConfigDict] = {'allow_inf_nan': True, 'from_attributes': True}*

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

classmethod model_construct(*_fields_set: set[str] | None = None*, ***values: Any*) → Model

Creates a new instance of the *Model* class with validated data.

Creates a new model setting *__dict__* and *__pydantic_fields_set__* from trusted or pre-validated data. Default values are respected, but no other validation is performed. Behaves as if *Config.extra = 'allow'* was set since it adds all passed values

Args:

_fields_set: The set of field names accepted for the Model instance. values: Trusted or pre-validated data dictionary.

Returns:

A new instance of the *Model* class with validated data.

model_copy(*, *update: dict[str, Any] | None = None*, *deep: bool = False*) → Model

Usage docs: https://docs.pydantic.dev/2.6/concepts/serialization/#model_copy

Returns a copy of the model.

Args:

update: Values to change/add in the new model. Note: the data is not validated
before creating the new model. You should trust this data.

deep: Set to *True* to make a deep copy of the model.

Returns:

New model instance.

model_dump(*, *mode: Literal['json', 'python'] | str = 'python'*, *include: IncEx = None*, *exclude: IncEx = None*, *by_alias: bool = False*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*, *round_trip: bool = False*, *warnings: bool = True*) → dict[str, Any]

Usage docs: https://docs.pydantic.dev/2.6/concepts/serialization/#modelmodel_dump

Generate a dictionary representation of the model, optionally specifying which fields to include or exclude.

Args:

mode: The mode in which *to_python* should run.
If mode is 'json', the output will only contain JSON serializable types. If mode is 'python', the output may contain non-JSON-serializable Python objects.

include: A list of fields to include in the output. exclude: A list of fields to exclude from the output. by_alias: Whether to use the

field's alias in the dictionary key if defined. exclude_unset: Whether to exclude fields that have not been explicitly set. exclude_defaults: Whether to exclude fields that are set to their default value. exclude_none: Whether to exclude fields that have a value of *None*. round_trip: If True, dumped values should be valid as input for non-idempotent types such as Json[T]. warnings: Whether to log warnings when invalid fields are encountered.

Returns:

A dictionary representation of the model.

model_dump_json(*, *indent: int | None = None*, *include: IncEx = None*, *exclude: IncEx = None*, *by_alias: bool = False*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*, *round_trip: bool = False*, *warnings: bool = True*) → str

Usage docs: https://docs.pydantic.dev/2.6/concepts/serialization/#modelmodel_dump_json

Generates a JSON representation of the model using Pydantic's *to_json* method.

Args:

indent: Indentation to use in the JSON output. If None is passed, the output will be compact. include: Field(s) to include in the JSON output. exclude: Field(s) to exclude from the JSON output. by_alias: Whether to serialize using field aliases. exclude_unset: Whether to exclude fields that have not been explicitly set. exclude_defaults: Whether to exclude fields that are set to their default value. exclude_none: Whether to exclude fields that have a value of *None*. round_trip: If True, dumped values should be valid as input for non-idempotent types such as Json[T]. warnings: Whether to log warnings when invalid fields are encountered.

Returns:

A JSON string representation of the model.

property model_extra*: dict[str, Any] | None*

Get extra fields set during validation.

Returns:

A dictionary of extra fields, or *None* if *config.extra* is not set to *"allow"*.

model_fields*: ClassVar[dict[str, FieldInfo]] = {'buy_price': FieldInfo(annotation=Union[float, NoneType], required=True), 'entity_id': FieldInfo(annotation=str, required=True), 'expected_open_pct': FieldInfo(annotation=float, required=True), 'id': FieldInfo(annotation=str, required=True), 'review': FieldInfo(annotation=Union[str, NoneType], required=True), 'sell_price': FieldInfo(annotation=Union[float, NoneType], required=True), 'status': FieldInfo(annotation=ExecutionStatus, required=False, default=<ExecutionStatus.init: 'init'>), 'stock_code': FieldInfo(annotation=str, required=True), 'stock_id': FieldInfo(annotation=str, required=True), 'stock_name': FieldInfo(annotation=str, required=True), 'timestamp': FieldInfo(annotation=datetime, required=True), 'trading_date': FieldInfo(annotation=datetime, required=True), 'trading_reason': FieldInfo(annotation=str, required=True), 'trading_signal_type': FieldInfo(annotation=TradingSignalType, required=True)}*

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

property model_fields_set*: set[str]*

Returns the set of fields that have been explicitly set on this model instance.

Returns:

A set of strings representing the fields that have been set, i.e. that were not filled from defaults.

classmethod model_json_schema(*by_alias: bool = True*, *ref_template: str = '#/$defs/{model}'*, *schema_generator: type[pydantic.json_schema.GenerateJsonSchema] = <class 'pydantic.json_schema.GenerateJsonSchema'>*, *mode: ~typing.Literal['validation', 'serialization'] = 'validation'*) → dict[str, Any]

Generates a JSON schema for a model class.

Args:

by_alias: Whether to use attribute aliases or not. ref_template: The reference template. schema_generator: To override the logic used to generate the JSON schema, as a subclass of

GenerateJsonSchema with your desired modifications

mode: The mode in which to generate the schema.

Returns:

The JSON schema for the given model class.

classmethod model_parametrized_name(*params: tuple[type[Any], ...]*) → str

Compute the class name for parametrizations of generic classes.

This method can be overridden to achieve a custom naming scheme for generic BaseModels.

Args:

params: Tuple of types of the class. Given a generic class *Model* with 2 type variables and a concrete model *Model[str, int]*, the value *(str, int)* would be passed to *params*.

Returns:

String representing the new class where *params* are passed to *cls* as type variables.

Raises:

TypeError: Raised when trying to generate concrete names for non-generic models.

model_post_init(*_BaseModel__context: Any*) → None

Override this method to perform additional initialization after *__init__* and *model_construct*. This is useful if you want to do some validation that requires the entire model to be initialized.

classmethod model_rebuild(*, *force: bool = False*, *raise_errors: bool = True*, *_parent_namespace_depth: int = 2*, *_types_namespace: Optional[dict[str, Any]] = None*) → bool | None

Try to rebuild the pydantic-core schema for the model.

This may be necessary when one of the annotations is a ForwardRef which could not be resolved during the initial attempt to build the schema, and automatic rebuilding fails.

Args:

force: Whether to force the rebuilding of the model schema, defaults to *False*. raise_errors: Whether to raise errors, defaults to *True*. _parent_namespace_depth: The depth level of the parent namespace, defaults to 2. _types_namespace: The types namespace, defaults to *None*.

Returns:

Returns *None* if the schema is already "complete" and rebuilding was not required. If rebuilding _was_ required, returns *True* if rebuilding was successful, otherwise *False*.

classmethod model_validate(*obj: Any*, *, *strict: bool | None = None*, *from_attributes: bool | None = None*, *context: dict[str, Any] | None = None*) → Model

Validate a pydantic model instance.

Args:

obj: The object to validate. strict: Whether to enforce types strictly. from_attributes: Whether to extract data from object

attributes. context: Additional context to pass to the validator.

Raises:

ValidationError: If the object could not be validated.

Returns:

The validated model instance.

classmethod model_validate_json(*json_data: str | bytes | bytearray*, *, *strict: bool | None = None*, *context: dict[str, Any] | None = None*) → Model

Usage docs: https://docs.pydantic.dev/2.6/concepts/json/#json-parsing

Validate the given JSON data against the Pydantic model.

Args:

json_data: The JSON data to validate. strict: Whether to enforce types strictly. context: Extra variables to pass to the validator.

Returns:

The validated Pydantic model.

Raises:

ValueError: If *json_data* is not a JSON string.

classmethod model_validate_strings(*obj: Any*, *, *strict: bool | None = None*, *context: dict[str, Any] | None = None*) → Model

Validate the given object contains string data against the Pydantic model.

Args:

obj: The object contains string data to validate. strict: Whether to enforce types strictly. context: Extra variables to pass to the validator.

Returns:

The validated Pydantic model.

zvt.trading.trading_schemas

Classes

`QueryStockQuoteSetting`(**kwargs)

`TradingPlan`(**kwargs)

QueryStockQuoteSetting

class zvt.trading.trading_schemas.QueryStockQuoteSetting(***kwargs*)

Bases: **Base**, **Mixin**

__init__(***kwargs*)

A simple constructor that allows initialization from kwargs.

Sets attributes on the constructed instance using the names and values in `kwargs`.

Only keys that are present as attributes of the instance's class are allowed. These could be, for example, any mapped columns or relationships.

entity_id

entity id

classmethod get_providers() → List[str]

providers of the schema defined by cls

Returns:: providers

classmethod get_storages(*provider: Optional[str] = None*)

get the storages info

Parameters:: **provider** – provider
Returns:: storages

id

id

classmethod query_data(*provider_index: int = 0*, *ids: Optional[List[str]] = None*, *entity_ids: Optional[List[str]] = None*, *entity_id: Optional[str] = None*, *codes: Optional[List[str]] = None*, *code: Optional[str] = None*, *level: Optional[Union[IntervalLevel, str]] = None*, *provider:*

Optional[str] = None, *columns: Optional[List] = None*, *col_label: Optional[dict] = None*, *return_type: str = 'df'*, *start_timestamp: Optional[Union[Timestamp, str]] = None*, *end_timestamp: Optional[Union[Timestamp, str]] = None*, *filters: Optional[List] = None*, *session: Optional[Session] = None*, *order=None*, *limit: Optional[int] = None*, *distinct=None*, *index: Optional[Union[str, list]] = None*, *drop_index_col=False*, *time_field: str = 'timestamp'*)

query data by the arguments

Parameters::

- **provider_index** –
- **data_schema** –
- **ids** –
- **entity_ids** –
- **entity_id** –
- **codes** –
- **code** –
- **level** –
- **provider** –
- **columns** –
- **col_label** – dict with key(column), value(label
- **return_type** – df, domain or dict. default is df
- **start_timestamp** –
- **end_timestamp** –
- **filters** –
- **session** –
- **order** –
- **limit** –
- **index** – index field name, str for single index, str list for multiple index
- **drop_index_col** – whether drop the col if it's in index, default False
- **time_field** –

Returns:: results basing on return_type.

classmethod record_data(*provider_index: int = 0*, *provider: Optional[str] = None*, *force_update=None*, *sleeping_time=None*, *exchanges=None*, *entity_id=None*, *entity_ids=None*, *code=None*, *codes=None*,

real_time=None, *fix_duplicate_way=None*, *start_timestamp=None*, *end_timestamp=None*, *one_day_trading_minutes=None*, ***kwargs*)

record data by the arguments

Parameters::

- **entity_id** –
- **provider_index** –
- **provider** –
- **force_update** –
- **sleeping_time** –
- **exchanges** –
- **entity_ids** –
- **code** –
- **codes** –
- **real_time** –
- **fix_duplicate_way** –
- **start_timestamp** –
- **end_timestamp** –
- **one_day_trading_minutes** –
- **kwargs** –

Returns::

classmethod register_provider(*provider*)

register the provider to the schema defined by cls

Parameters:: **provider** –

classmethod register_recorder_cls(*provider*, *recorder_cls*)

register the recorder for the schema

Parameters::

- **provider** –
- **recorder_cls** –

timestamp

the meaning could be different for different case,most time it means 'happen time'

TradingPlan

class zvt.trading.trading_schemas.TradingPlan(***kwargs*)

Bases: **Base**, **Mixin**

__init__(***kwargs*)

A simple constructor that allows initialization from kwargs.

Sets attributes on the constructed instance using the names and values in `kwargs`.

Only keys that are present as attributes of the instance's class are allowed. These could be, for example, any mapped columns or relationships.

entity_id

entity id

classmethod get_providers() → List[str]

providers of the schema defined by cls

Returns:: providers

classmethod get_storages(*provider: Optional[str] = None*)

get the storages info

Parameters:: **provider** – provider
Returns:: storages

id

id

classmethod query_data(*provider_index: int = 0*, *ids: Optional[List[str]] = None*, *entity_ids: Optional[List[str]] = None*, *entity_id: Optional[str] = None*, *codes: Optional[List[str]] = None*, *code: Optional[str] = None*, *level: Optional[Union[IntervalLevel, str]] = None*, *provider:*

Optional[str] = None, *columns: Optional[List] = None*, *col_label: Optional[dict] = None*, *return_type: str = 'df'*, *start_timestamp: Optional[Union[Timestamp, str]] = None*, *end_timestamp: Optional[Union[Timestamp, str]] = None*, *filters: Optional[List] = None*, *session: Optional[Session] = None*, *order=None*, *limit: Optional[int] = None*, *distinct=None*, *index: Optional[Union[str, list]] = None*, *drop_index_col=False*, *time_field: str = 'timestamp'*)

query data by the arguments

Parameters::

- **provider_index** –
- **data_schema** –
- **ids** –
- **entity_ids** –
- **entity_id** –
- **codes** –
- **code** –
- **level** –
- **provider** –
- **columns** –
- **col_label** – dict with key(column), value(label
- **return_type** – df, domain or dict. default is df
- **start_timestamp** –
- **end_timestamp** –
- **filters** –
- **session** –
- **order** –
- **limit** –
- **index** – index field name, str for single index, str list for multiple index
- **drop_index_col** – whether drop the col if it's in index, default False
- **time_field** –

Returns:: results basing on return_type.

classmethod record_data(*provider_index: int = 0*, *provider: Optional[str] = None*, *force_update=None*, *sleeping_time=None*, *exchanges=None*, *entity_id=None*, *entity_ids=None*, *code=None*, *codes=None*,

real_time=None, *fix_duplicate_way=None*, *start_timestamp=None*, *end_timestamp=None*, *one_day_trading_minutes=None*, ***kwargs*)

record data by the arguments

Parameters::

- **entity_id** –
- **provider_index** –
- **provider** –
- **force_update** –
- **sleeping_time** –
- **exchanges** –
- **entity_ids** –
- **code** –
- **codes** –
- **real_time** –
- **fix_duplicate_way** –
- **start_timestamp** –
- **end_timestamp** –
- **one_day_trading_minutes** –
- **kwargs** –

Returns::

classmethod register_provider(*provider*)

register the provider to the schema defined by cls

Parameters:: **provider** –

classmethod register_recorder_cls(*provider*, *recorder_cls*)

register the recorder for the schema

Parameters::

- **provider** –
- **recorder_cls** –

timestamp

the meaning could be different for different case,most time it means 'happen time'

zvt.trading.trading_service

Functions

build_query_stock_quote_setting(...)

---

build_trading_plan(build_trading_plan_model)

---

buy_stocks()

---

check_trading_plan()

---

get_current_trading_plan()

---

get_future_trading_plan()

---

query_stock_quotes(query_stock_quote_model)

---

query_tag_quotes(query_tag_quote_model)

---

query_trading_plan(query_trading_plan_model)

---

sell_stocks()

---

zvt.trading.trading_service.build_query_stock_quote_setting

zvt.trading.trading_service.build_query_stock_quote_setting(*build_query_stock_quote_setting_model: BuildQueryStockQuoteSettingModel*, *timestamp*)

zvt.trading.trading_service.build_trading_plan

zvt.trading.trading_service.build_trading_plan(*build_trading_plan_model: BuildTradingPlanModel*)

zvt.trading.trading_service.buy_stocks

zvt.trading.trading_service.buy_stocks()

zvt.trading.trading_service.check_trading_plan

zvt.trading.trading_service.check_trading_plan()

zvt.trading.trading_service.get_current_trading_plan

zvt.trading.trading_service.get_current_trading_plan()

zvt.trading.trading_service.get_future_trading_plan

zvt.trading.trading_service.get_future_trading_plan()

zvt.trading.trading_service.query_stock_quotes

zvt.trading.trading_service.query_stock_quotes(*query_stock_quote_model: QueryStockQuoteModel*)

zvt.trading.trading_service.query_tag_quotes

zvt.trading.trading_service.query_tag_quotes(*query_tag_quote_model: QueryTagQuoteModel*)

zvt.trading.trading_service.query_trading_plan

zvt.trading.trading_service.query_trading_plan(*query_trading_plan_model: QueryTradingPlanModel*)

zvt.trading.trading_service.sell_stocks

zvt.trading.trading_service.sell_stocks()

zvt.ui

Modules

zvt.ui.assets

---

zvt.ui.components

---

zvt.ui.assets

zvt.ui.components

Modules

zvt.ui.components.dcc_components

---

zvt.ui.components.dcc_components

Functions

`get_account_stats_figure`(account_stats_reader)

`get_trading_signals_figure`(order_reader, ...)

`order_type_color`(order_type)

`order_type_flag`(order_type)

---

zvt.ui.components.dcc_components.get_account_stats_figure

zvt.ui.components.dcc_components.get_account_stats_figure(*account_stats_reader: AccountStatsReader*)

zvt.ui.components.dcc_components.get_trading_signals_figure

zvt.ui.components.dcc_components.get_trading_signals_figure(*order_reader: OrderReader*, *entity_id: str*, *start_timestamp=None*, *end_timestamp=None*, *adjust_type=None*)

zvt.ui.components.dcc_components.order_type_color

zvt.ui.components.dcc_components.order_type_color(*order_type*)

zvt.ui.components.dcc_components.order_type_flag

zvt.ui.components.dcc_components.order_type_flag(*order_type*)

zvt.utils

Modules

zvt.utils.decorator

---

zvt.utils.file_utils

---

zvt.utils.git_utils

---

zvt.utils.model_utils

---

zvt.utils.pd_utils

---

zvt.utils.recorder_utils

---

zvt.utils.str_utils

---

zvt.utils.time_utils

---

zvt.utils.utils

---

zvt.utils.zip_utils

---

zvt.utils.decorator

Functions

`to_string`(cls)

---

zvt.utils.decorator.to_string

zvt.utils.decorator.to_string(*cls*)

zvt.utils.file_utils

Functions

`list_all_files`([dir_path, ext, excludes, ...])	list all files with extension in specific directory recursively

---

zvt.utils.file_utils.list_all_files

zvt.utils.file_utils.list_all_files(*dir_path: str = ./domain'*, *ext: Optional[str] = '.py'*, *excludes=None*, *includes=None*, *return_base_name=False*) → List[str]

list all files with extension in specific directory recursively

Parameters::

- **includes** – including files, None means all
- **dir_path** – the directory path
- **ext** – file extension
- **excludes** – excluding files
- **return_base_name** – return file name if True otherwise abs path

Returns::

zvt.utils.git_utils

Functions

`get_git_user_email`()

`get_git_user_name`()

---

zvt.utils.git_utils.get_git_user_email

zvt.utils.git_utils.get_git_user_email()

zvt.utils.git_utils.get_git_user_name

zvt.utils.git_utils.get_git_user_name()

zvt.utils.model_utils

Functions

`update_model`(db_model, schema)

---

zvt.utils.model_utils.update_model

zvt.utils.model_utils.update_model(*db_model*, *schema*)

zvt.utils.pd_utils

Functions

df_subset(df[, columns])

---

drop_continue_duplicate(s[, col])

---

fill_with_same_index(df_list)

---

group_by_entity_id(input_df)

---

index_df(df[, index, inplace, drop, time_field])

---

is_filter_result_df(df)

---

is_normal_df(df[, category_field, time_filed])

---

is_score_result_df(df)

---

merge_filter_result(input_df, filter_result)

---

normal_index_df(df[, category_field, ...])

---

normalize_group_compute_result(group_result)

`pd_is_not_null`(df)

---

zvt.utils.pd_utils.df_subset

zvt.utils.pd_utils.df_subset(*df*, *columns=None*)

zvt.utils.pd_utils.drop_continue_duplicate

zvt.utils.pd_utils.drop_continue_duplicate(*s: Union[Series, DataFrame]*, *col=None*)

zvt.utils.pd_utils.fill_with_same_index

zvt.utils.pd_utils.fill_with_same_index(*df_list: List[DataFrame]*)

zvt.utils.pd_utils.group_by_entity_id

zvt.utils.pd_utils.group_by_entity_id(*input_df: DataFrame*)

zvt.utils.pd_utils.index_df

zvt.utils.pd_utils.index_df(*df*, *index='timestamp'*, *inplace=True*, *drop=False*, *time_field='timestamp'*)

zvt.utils.pd_utils.is_filter_result_df

zvt.utils.pd_utils.is_filter_result_df(*df: DataFrame*)

zvt.utils.pd_utils.is_normal_df

zvt.utils.pd_utils.is_normal_df(*df*, *category_field='entity_id'*, *time_filed='timestamp'*)

zvt.utils.pd_utils.is_score_result_df

zvt.utils.pd_utils.is_score_result_df(*df: DataFrame*)

zvt.utils.pd_utils.merge_filter_result

zvt.utils.pd_utils.merge_filter_result(*input_df: DataFrame*, *filter_result: Series*)

zvt.utils.pd_utils.normal_index_df

zvt.utils.pd_utils.normal_index_df(*df*, *category_field='entity_id'*, *time_filed='timestamp'*, *drop=True*, *default_entity='entity'*)

zvt.utils.pd_utils.normalize_group_compute_result

zvt.utils.pd_utils.normalize_group_compute_result(*group_result*)

zvt.utils.pd_utils.pd_is_not_null

zvt.utils.pd_utils.pd_is_not_null(*df: Union[DataFrame, Series]*)

zvt.utils.recorder_utils

Functions

`run_data_recorder`(domain[, entity_provider, ...])

---

zvt.utils.recorder_utils.run_data_recorder

zvt.utils.recorder_utils.run_data_recorder(*domain: Type[Mixin]*, *entity_provider=None*, *data_provider=None*, *entity_ids=None*, *retry_times=10*, *sleeping_time=10*, *return_unfinished=False*, ***recorder_kv*)

zvt.utils.str_utils

Functions

`to_camel_str`(input)

`to_snake_str`(input)

---

zvt.utils.str_utils.to_camel_str

zvt.utils.str_utils.to_camel_str(*input: str*) → str

zvt.utils.str_utils.to_snake_str

zvt.utils.str_utils.to_snake_str(*input: str*) → str

zvt.utils.time_utils

Functions

count_interval(start_date, end_date)	
current_date()	
date_and_time(the_date, the_time)	
date_time_by_interval(the_time[, interval, unit])	
day_offset_today([offset])	
evaluate_size_from_timestamp(...[, ...])	given from timestamp,level,one_day_trading_minutes,this func evaluate size of kdata to current.
get_local_timezone()	
get_year_quarter(time)	
get_year_quarters(start[, end])	
is_finished_kdata_timestamp(timestamp, level)	
is_in_same_interval(t1, t2, level)	
is_same_date(one, two)	
is_same_time(one, two)	
month_end_date(the_date)	

month_start_date(the_date)

month_start_end_ranges(start_date, end_date)

next_timestamp_on_level(current_timestamp, level)

now_pd_timestamp()

now_time_str([fmt])

now_timestamp()

pre_month([t])

pre_month_end_date([t])

pre_month_start_date([t])

recent_year_date()

split_time_interval(start, end[, method, ...])

to_pd_timestamp(the_time)

to_time_str(the_time[, fmt])

to_timestamp(the_time)

today()

tomorrow_date()

zvt.utils.time_utils.count_interval

zvt.utils.time_utils.count_interval(*start_date*, *end_date*)

zvt.utils.time_utils.current_date

zvt.utils.time_utils.current_date() → Timestamp

zvt.utils.time_utils.date_and_time

zvt.utils.time_utils.date_and_time(*the_date*, *the_time*)

zvt.utils.time_utils.date_time_by_interval

zvt.utils.time_utils.date_time_by_interval(*the_time*, *interval=1*, *unit: TimeUnit = TimeUnit.day*)

zvt.utils.time_utils.day_offset_today

zvt.utils.time_utils.day_offset_today(*offset=0*)

zvt.utils.time_utils.evaluate_size_from_timestamp

zvt.utils.time_utils.evaluate_size_from_timestamp(*start_timestamp*, *level: IntervalLevel*, *one_day_trading_minutes*, *end_timestamp: Optional[Timestamp] = None*)

given from timestamp,level,one_day_trading_minutes,this func evaluate size of kdata to current. it maybe a little bigger than the real size for fetching all the kdata.

Parameters::

- **start_timestamp** (*pd.Timestamp*) –
- **level** (*IntervalLevel*) –
- **one_day_trading_minutes** (*int*) –

zvt.utils.time_utils.get_local_timezone

zvt.utils.time_utils.get_local_timezone()

zvt.utils.time_utils.get_year_quarter

zvt.utils.time_utils.get_year_quarter(*time*)

zvt.utils.time_utils.get_year_quarters

zvt.utils.time_utils.get_year_quarters(*start*, *end=Timestamp('2024-08-28 22:08:09.719204')*)

zvt.utils.time_utils.is_finished_kdata_timestamp

zvt.utils.time_utils.is_finished_kdata_timestamp(*timestamp*, *level: IntervalLevel*)

zvt.utils.time_utils.is_in_same_interval

zvt.utils.time_utils.is_in_same_interval(*t1: Timestamp*, *t2: Timestamp*, *level: IntervalLevel*)

zvt.utils.time_utils.is_same_date

zvt.utils.time_utils.is_same_date(*one*, *two*)

zvt.utils.time_utils.is_same_time

zvt.utils.time_utils.is_same_time(*one*, *two*)

zvt.utils.time_utils.month_end_date

zvt.utils.time_utils.month_end_date(*the_date*)

zvt.utils.time_utils.month_start_date

zvt.utils.time_utils.month_start_date(*the_date*)

zvt.utils.time_utils.month_start_end_ranges

zvt.utils.time_utils.month_start_end_ranges(*start_date*, *end_date*)

zvt.utils.time_utils.next_timestamp_on_level

zvt.utils.time_utils.next_timestamp_on_level(*current_timestamp: Timestamp*, *level: IntervalLevel*) → Timestamp

zvt.utils.time_utils.now_pd_timestamp

zvt.utils.time_utils.now_pd_timestamp() → Timestamp

zvt.utils.time_utils.now_time_str

zvt.utils.time_utils.now_time_str(*fmt='YYYY-MM-DD'*)

zvt.utils.time_utils.now_timestamp

zvt.utils.time_utils.now_timestamp()

zvt.utils.time_utils.pre_month

zvt.utils.time_utils.pre_month(*t=Timestamp('2024-08-28 22:08:09.719129')*)

zvt.utils.time_utils.pre_month_end_date

zvt.utils.time_utils.pre_month_end_date(*t=Timestamp('2024-08-28 00:00:00')*)

zvt.utils.time_utils.pre_month_start_date

zvt.utils.time_utils.pre_month_start_date(*t=Timestamp('2024-08-28 00:00:00')*)

zvt.utils.time_utils.recent_year_date

zvt.utils.time_utils.recent_year_date()

zvt.utils.time_utils.split_time_interval

zvt.utils.time_utils.split_time_interval(*start*, *end*, *method=None*, *interval=30*, *freq='D'*)

zvt.utils.time_utils.to_pd_timestamp

zvt.utils.time_utils.to_pd_timestamp(*the_time*) → Timestamp

zvt.utils.time_utils.to_time_str

zvt.utils.time_utils.to_time_str(*the_time*, *fmt='YYYY-MM-DD'*)

zvt.utils.time_utils.to_timestamp

zvt.utils.time_utils.to_timestamp(*the_time*)

zvt.utils.time_utils.today

zvt.utils.time_utils.today() → Timestamp

zvt.utils.time_utils.tomorrow_date

zvt.utils.time_utils.tomorrow_date()

zvt.utils.utils

Functions

add_func_to_value(the_map, the_func)	
add_to_map_list(the_map, key, value)	
chrome_copy_header_to_dict(src)	
compare_dicts(dict1, dict2)	
fill_dict(src, dst)	Fills items from the source dictionary (src) into the destination dictionary (dst) if the keys are not already present in dst.
fill_domain_from_dict(the_domain, the_dict)	use field map and related func to fill properties from the dict to the domain
first_item_to_float(the_list)	
flatten_list(input_list)	

iterate_with_step(data[, sub_size])

---

json_callback_param(the_str)

---

marshal_object_for_ui(object)

---

multiple_number(number, factor)

---

parse_url_params(url)

---

pct_to_float(the_str[, default])

---

read_csv(f, encoding[, sep, na_values])

---

second_item_to_float(the_list)

---

set_one_and_only_one(**kwargs)

---

to_float(the_str[, default])

---

to_positive_number(number)

---

to_str(str_or_list)

---

url_unquote(url)

---

zvt.utils.utils.add_func_to_value

zvt.utils.utils.add_func_to_value(*the_map*, *the_func*)

zvt.utils.utils.add_to_map_list

zvt.utils.utils.add_to_map_list(*the_map*, *key*, *value*)

zvt.utils.utils.chrome_copy_header_to_dict

zvt.utils.utils.chrome_copy_header_to_dict(*src*)

zvt.utils.utils.compare_dicts

zvt.utils.utils.compare_dicts(*dict1*, *dict2*)

zvt.utils.utils.fill_dict

zvt.utils.utils.fill_dict(*src*, *dst*)

Fills items from the source dictionary (src) into the destination dictionary (dst) if the keys are not already present in dst.

Args:

src (dict): The source dictionary to copy items from. dst (dict): The destination dictionary to copy items into.

Returns:

dict: The updated destination dictionary with new items from the source dictionary.

zvt.utils.utils.fill_domain_from_dict

zvt.utils.utils.fill_domain_from_dict(*the_domain*, *the_dict: dict*, *the_map: ~typing.Optional[dict] = None*, *default_func=<function <lambda>>*)

use field map and related func to fill properties from the dict to the domain

Parameters::

- **the_domain** (*DeclarativeMeta*) –
- **the_dict** (*dict*) –
- **the_map** (*dict*) –
- **default_func** (*function*) –

zvt.utils.utils.first_item_to_float

zvt.utils.utils.first_item_to_float(*the_list*)

zvt.utils.utils.flatten_list

zvt.utils.utils.flatten_list(*input_list*)

zvt.utils.utils.iterate_with_step

zvt.utils.utils.iterate_with_step(*data*, *sub_size=100*)

zvt.utils.utils.json_callback_param

zvt.utils.utils.json_callback_param(*the_str*)

zvt.utils.utils.marshal_object_for_ui

zvt.utils.utils.marshal_object_for_ui(*object*)

zvt.utils.utils.multiple_number

zvt.utils.utils.multiple_number(*number*, *factor*)

zvt.utils.utils.parse_url_params

zvt.utils.utils.parse_url_params(*url*)

zvt.utils.utils.pct_to_float

zvt.utils.utils.pct_to_float(*the_str*, *default=None*)

zvt.utils.utils.read_csv

zvt.utils.utils.read_csv(*f*, *encoding*, *sep=None*, *na_values=None*)

zvt.utils.utils.second_item_to_float

zvt.utils.utils.second_item_to_float(*the_list*)

zvt.utils.utils.set_one_and_only_one

zvt.utils.utils.set_one_and_only_one(**_kwargs_)

zvt.utils.utils.to_float

zvt.utils.utils.to_float(*the_str*, *default=None*)

zvt.utils.utils.to_positive_number

zvt.utils.utils.to_positive_number(*number*)

zvt.utils.utils.to_str

zvt.utils.utils.to_str(*str_or_list*)

zvt.utils.utils.url_unquote

zvt.utils.utils.url_unquote(*url*)

zvt.utils.zip_utils

Functions

`unzip`(zip_file, dst_dir)

`zip_dir`(src_dir[, dst_dir, zip_file_name])

---

zvt.utils.zip_utils.unzip

zvt.utils.zip_utils.unzip(*zip_file*, *dst_dir*)

zvt.utils.zip_utils.zip_dir

zvt.utils.zip_utils.zip_dir(*src_dir*, *dst_dir=None*, *zip_file_name=None*)

zvt.zhdate

Modules

`zvt.zhdate.constants`	
`zvt.zhdate.zhdate`	-- *coding: utf-8* -- thanks to https://github.com/CutePandaSh/zhdate
`zvt.zhdate.ztime`	

zvt.zhdate.constants

Module Attributes

CHINESEYEARCODE	从1900年到2100年的农历月份数据代码 20位二进制代码表示一个年份的数据。
CHINESENEWYEAR	从1900年，至2100年每年的农历春节的公历日期

zvt.zhdate.constants.CHINESEYEARCODE

zvt.zhdate.constants.CHINESEYEARCODE = *[19416, 19168, 42352, 21717, 53856, 55632, 91476, 22176, 39632, 21970, 19168, 42422, 42192, 53840, 119381, 46400, 54944, 44450, 38320, 84343, 18800, 42160, 46261, 27216, 27968, 109396, 11104, 38256, 21234, 18800, 25958, 54432, 59984, 92821, 23248, 11104, 100067, 37600, 116951, 51536, 54432, 120998, 46416, 22176, 107956, 9680, 37584, 53938, 43344, 46423, 27808, 46416, 86869, 19872, 42416, 83315, 21168, 43432, 59728, 27296, 44710, 43856, 19296, 43748, 42352, 21088, 62051, 55632, 23383, 22176, 38608, 19925, 19152, 42192, 54484, 53840, 54616, 46400, 46752, 103846, 38320, 18864, 43380, 42160, 45690, 27216, 27968, 44870, 43872, 38256, 19189, 18800, 25776, 29859, 59984, 27480, 23232, 43872, 38613, 37600, 51552, 55636, 54432, 55888, 30034, 22176, 43959, 9680, 37584, 51893, 43344, 46240, 47780, 44368, 21977, 19360, 42416, 86390, 21168, 43312, 31060, 27296, 44368, 23378, 19296, 42726, 42208, 53856, 60005, 54576, 23200, 30371, 38608, 19195, 19152, 42192, 118966, 53840, 54560, 56645, 46496, 22224, 21938, 18864, 42359, 42160, 43600, 111189, 27936, 44448, 84835, 37744, 18936, 18800, 25776, 92326, 59984, 27296, 108228, 43744, 37600, 53987, 51552, 54615, 54432, 55888, 23893, 22176, 42704, 21972, 21200, 43448, 43344, 46240, 46758, 44368, 21920, 43940, 42416, 21168, 45683, 26928, 29495, 27296, 44368, 84821, 19296, 42352, 21732, 53600, 59752, 54560, 55968, 92838, 22224, 19168, 43476, 41680, 53584, 62034, 54560]*

从1900年到2100年的农历月份数据代码 20位二进制代码表示一个年份的数据。

前四位0:表示闰月为29天，1:表示闰月为30天 中间12位：从左起表示1-12月每月的大小，1为30天，0为29天 最后四位：表示闰月的月份，0表示当年无闰月

前四位和最后四位应该结合使用，如果最后四位为0，则不考虑前四位 例： 1901年代码为 19168，转成二进制为

0b100101011100000, 最后四位为0，当年无闰月，月份数据为 010010101110 分别代表12月的大小情况 1903年代码为 21717，转成二进制为 0b101010011010101，最后四位为5，当年为闰五月，首四位为0，闰月为29天，月份数据为 010101001101 分别代表12月的大小情况

zvt.zhdate.constants.CHINESENEWYEAR

zvt.zhdate.constants.CHINESENEWYEAR = *['19000131', '19010219', '19020208', '19030129', '19040216', '19050204', '19060125', '19070213', '19080202', '19090122', '19100210', '19110130', '19120218', '19130206', '19140126', '19150214', '19160203', '19170123', '19180211', '19190201', '19200220', '19210208', '19220128', '19230216', '19240205', '19250124', '19260213', '19270202', '19280123', '19290210', '19300130', '19310217', '19320206', '19330126', '19340214', '19350204', '19360124', '19370211', '19380131', '19390219', '19400208', '19410127', '19420215', '19430205', '19440125', '19450213', '19460202', '19470122', '19480210', '19490129', '19500217', '19510206', '19520127', '19530214', '19540203', '19550124', '19560212', '19570131', '19580218', '19590208', '19600128', '19610215', '19620205', '19630125', '19640213', '19650202', '19660121', '19670209', '19680130', '19690217', '19700206', '19710127', '19720215', '19730203', '19740123', '19750211', '19760131', '19770218', '19780207', '19790128', '19800216', '19810205', '19820125', '19830213', '19840202', '19850220', '19860209', '19870129', '19880217', '19890206', '19900127', '19910215', '19920204', '19930123', '19940210', '19950131', '19960219', '19970207', '19980128', '19990216', '20000205', '20010124', '20020212', '20030201', '20040122', '20050209', '20060129', '20070218', '20080207', '20090126', '20100214', '20110203', '20120123', '20130210', '20140131', '20150219', '20160208', '20170128', '20180216', '20190205', '20200125', '20210212', '20220201', '20230122', '20240210', '20250129', '20260217', '20270206', '20280126', '20290213', '20300203', '20310123', '20320211', '20330131', '20340219', '20350208', '20360128', '20370215', '20380204', '20390124', '20400212', '20410201', '20420122', '20430210', '20440130', '20450217', '20460206', '20470126', '20480214', '20490202', '20500123', '20510211', '20520201', '20530219', '20540208', '20550128', '20560215', '20570204', '20580124', '20590212', '20600202', '20610121', '20620209', '20630129', '20640217', '20650205', '20660126', '20670214', '20680203', '20690123', '20700211', '20710131', '20720219', '20730207', '20740127', '20750215', '20760205', '20770124', '20780212', '20790202', '20800122', '20810209',*

'20820129', '20830217', '20840206', '20850126', '20860214', '20870203', '20880124', '20890210', '20900130', '20910218', '20920207', '20930127', '20940215', '20950205', '20960125', '20970212', '20980201', '20990121', '21000209']

从1900年，至2100年每年的农历春节的公历日期

zvt.zhdate.zhdate

-- *coding: utf-8* -- thanks to https://github.com/CutePandaSh/zhdate

Classes

`ZhDate`(lunar_year, lunar_month, lunar_day[, ...])

ZhDate

class zvt.zhdate.zhdate.ZhDate(*lunar_year*, *lunar_month*, *lunar_day*, *leap_month=False*)

Bases: `object`

__init__(*lunar_year*, *lunar_month*, *lunar_day*, *leap_month=False*)

初始化函数

Arguments:

lunar_year {int} – 农历年 lunar_month {int} – 农历月份
lunar_day {int} – 农历日

Keyword Arguments:

leap_month {bool} – 是否是在农历闰月中 (default: {False})

to_datetime()

农历日期转换称公历日期

Returns:

datetime – 当前农历对应的公历日期

static from_datetime(*dt*)

静态方法，从公历日期生成农历日期

Arguments:

dt {datetime} – 公历的日期

Returns:

ZhDate – 生成的农历日期对象

static validate(*year*, *month*, *day*, *leap*)

农历日期校验

Arguments:

year {int} – 农历年份 month {int} – 农历月份 day {int} – 农历日期 leap {bool} – 农历是否为闰月日期

Returns:

bool – 校验是否通过

static decode(*year_code*)

解析年度农历代码函数

Arguments:

year_code {int} – 从年度代码数组中获取的代码整数

Returns:

list[int,] – 当前年度代码解析以后形成的每月天数数组，已将闰月嵌入对应位置，即有闰月的年份返回的列表长度为13，否则为12

static month_days(*year*)

根据年份返回当前农历月份天数list

Arguments:

year {int} – 1900到2100的之间的整数

Returns:

[int] – 农历年份所对应的农历月份天数列表

zvt.zhdate.ztime

Functions

`holiday_distance`([timestamp, days])

---

zvt.zhdate.ztime.holiday_distance

zvt.zhdate.ztime.holiday_distance(*timestamp=None*, *days=15*)

Python Module Index

Z

z

zvt

zvt.api

zvt.api.intent

zvt.api.kdata

zvt.api.portfolio

zvt.api.selector

zvt.api.stats

zvt.api.utils

zvt.autocode

zvt.autocode.generator

zvt.autocode.templates

zvt.broker

zvt.broker.qmt

zvt.broker.qmt.errors

zvt.common

zvt.common.query_models

zvt.common.trading_models

zvt.consts

zvt.contract

zvt.contract.api

zvt.contract.base_service

zvt.contract.context

zvt.contract.data_type

zvt.contract.drawer

zvt.contract.factor

zvt.contract.model

zvt.contract.normal_data

zvt.contract.reader

zvt.contract.recorder

zvt.contract.register

zvt.contract.schema

zvt.contract.zvt_info

zvt.domain

zvt.domain.actor

zvt.domain.actor.actor_meta

zvt.domain.actor.stock_actor

zvt.domain.emotion

zvt.domain.emotion.emotion

zvt.domain.fundamental

zvt.domain.fundamental.dividend_financing

zvt.domain.fundamental.finance

zvt.domain.fundamental.trading

zvt.domain.fundamental.valuation

zvt.domain.macro

zvt.domain.macro.macro

zvt.domain.macro.monetary

zvt.domain.meta

zvt.domain.meta.block_meta

zvt.domain.meta.country_meta

zvt.domain.meta.currency_meta

zvt.domain.meta.etf_meta

zvt.domain.meta.fund_meta

zvt.domain.meta.future_meta

zvt.domain.meta.index_meta

zvt.domain.meta.indexus_meta

zvt.domain.meta.stock_meta

zvt.domain.meta.stockhk_meta

zvt.domain.meta.stockus_meta

zvt.domain.misc

zvt.domain.misc.holder

zvt.domain.misc.money_flow

zvt.domain.misc.overall

zvt.domain.misc.stock_news

zvt.domain.quotes

zvt.domain.quotes.block

zvt.domain.quotes.block.block_1d_kdata

zvt.domain.quotes.block.block_1mon_kdata

zvt.domain.quotes.block.block_1wk_kdata

zvt.domain.quotes.currency

zvt.domain.quotes.currency.currency_1d_kdata

zvt.domain.quotes.etf

zvt.domain.quotes.etf.etf_1d_kdata

zvt.domain.quotes.future

zvt.domain.quotes.future.future_1d_kdata

zvt.domain.quotes.index

zvt.domain.quotes.index.index_1d_kdata

zvt.domain.quotes.index.index_1wk_kdata

zvt.domain.quotes.indexus

zvt.domain.quotes.indexus.indexus_1d_kdata

zvt.domain.quotes.stock

zvt.domain.quotes.stock.stock_15m_hfq_kdata

zvt.domain.quotes.stock.stock_15m_kdata

zvt.domain.quotes.stock.stock_1d_hfq_kdata

zvt.domain.quotes.stock.stock_1d_kdata

zvt.domain.quotes.stock.stock_1h_hfq_kdata

zvt.domain.quotes.stock.stock_1h_kdata

zvt.domain.quotes.stock.stock_1m_hfq_kdata

zvt.domain.quotes.stock.stock_1m_kdata

zvt.domain.quotes.stock.stock_1mon_hfq_kdata

zvt.domain.quotes.stock.stock_1mon_kdata

zvt.domain.quotes.stock.stock_1wk_hfq_kdata

zvt.domain.quotes.stock.stock_1wk_kdata

zvt.domain.quotes.stock.stock_30m_hfq_kdata

zvt.domain.quotes.stock.stock_30m_kdata

zvt.domain.quotes.stock.stock_4h_hfq_kdata

zvt.domain.quotes.stock.stock_4h_kdata

zvt.domain.quotes.stock.stock_5m_hfq_kdata

zvt.domain.quotes.stock.stock_5m_kdata

zvt.domain.quotes.stock.stock_quote

zvt.domain.quotes.stockhk

zvt.domain.quotes.stockhk.stockhk_1d_hfq_kdata

zvt.domain.quotes.stockhk.stockhk_1d_kdata

zvt.domain.quotes.stockus

zvt.domain.quotes.stockus.stockus_1d_hfq_kdata

zvt.domain.quotes.stockus.stockus_1d_kdata

zvt.domain.quotes.trade_day

zvt.factors

zvt.factors.algorithm

zvt.factors.factor_models

zvt.factors.factor_service

zvt.factors.fundamental

zvt.factors.fundamental.finance_factor

zvt.factors.ma

zvt.factors.ma.domain

zvt.factors.ma.domain.common

zvt.factors.ma.domain.stock_1d_ma_factor

zvt.factors.ma.domain.stock_1d_ma_stats_factor

zvt.factors.ma.ma_factor

zvt.factors.ma.ma_stats_factor

zvt.factors.ma.top_bottom_factor

zvt.factors.macd

zvt.factors.macd.macd_factor

zvt.factors.shape

zvt.factors.target_selector

zvt.factors.technical_factor

zvt.factors.top_stocks

zvt.factors.transformers

zvt.factors.z

zvt.factors.z.domain

zvt.factors.z.domain.common

zvt.factors.z.domain.stock_1d_z_factor

zvt.factors.z.domain.stock_1wk_z_factor

zvt.factors.z.z_factor

zvt.fill_project

zvt.informer

zvt.informer.inform_utils

zvt.informer.informer

zvt.main

zvt.ml

zvt.ml.lables

zvt.ml.ml

zvt.plugin

zvt.recorders

zvt.recorders.consts

zvt.recorders.eastmoney

zvt.recorders.eastmoney.common

zvt.recorders.eastmoney.dividend_financing

zvt.recorders.eastmoney.dividend_financing.eastmoney_dividend_detail_recorder

zvt.recorders.eastmoney.dividend_financing.eastmoney_dividend_financing_recorder

zvt.recorders.eastmoney.dividend_financing.eastmoney_rights_issue_detail_recorder

zvt.recorders.eastmoney.dividend_financing.eastmoney_spo_detail_recorder

zvt.recorders.eastmoney.finance

zvt.recorders.eastmoney.finance.base_china_stock_finance_recorder

zvt.recorders.eastmoney.finance.eastmoney_balance_sheet_recorder

zvt.recorders.eastmoney.finance.eastmoney_cash_flow_recorder

zvt.recorders.eastmoney.finance.eastmoney_finance_factor_recorder

zvt.recorders.eastmoney.finance.eastmoney_income_statement_recorder

zvt.recorders.eastmoney.holder

zvt.recorders.eastmoney.holder.eastmoney_stock_actor_recorder

zvt.recorders.eastmoney.holder.eastmoney_top_ten_holder_recorder

zvt.recorders.eastmoney.holder.eastmoney_top_ten_tradable_holder_recorder

zvt.recorders.eastmoney.meta

zvt.recorders.eastmoney.meta.eastmoney_block_meta_recorder

zvt.recorders.eastmoney.meta.eastmoney_stock_meta_recorder

zvt.recorders.eastmoney.trading

zvt.recorders.eastmoney.trading.eastmoney_holder_trading_recorder

zvt.recorders.eastmoney.trading.eastmoney_manager_trading_recorder

zvt.recorders.em

zvt.recorders.em.actor

zvt.recorders.em.actor.em_stock_actor_summary_recorder

zvt.recorders.em.actor.em_stock_ii_recorder

zvt.recorders.em.actor.em_stock_top_ten_free_recorder

zvt.recorders.em.actor.em_stock_top_ten_recorder

zvt.recorders.em.em_api

zvt.recorders.em.macro

zvt.recorders.em.macro.em_treasury_yield_recorder

zvt.recorders.em.meta

zvt.recorders.em.meta.em_block_meta_recorder

zvt.recorders.em.meta.em_currency_meta_recorder

zvt.recorders.em.meta.em_future_meta_recorder

zvt.recorders.em.meta.em_index_meta_recorder

zvt.recorders.em.meta.em_indexus_meta_recorder

zvt.recorders.em.meta.em_stock_meta_recorder

zvt.recorders.em.meta.em_stockhk_meta_recorder

zvt.recorders.em.meta.em_stockus_meta_recorder

zvt.recorders.em.misc

zvt.recorders.em.misc.em_stock_news_recorder

zvt.recorders.em.quotes

zvt.recorders.em.quotes.em_kdata_recorder

zvt.recorders.em.trading

zvt.recorders.em.trading.em_dragon_and_tiger_recorder

zvt.recorders.exchange

zvt.recorders.exchange.api

zvt.recorders.exchange.api.cn_index_api

zvt.recorders.exchange.api.cn_index_stock_api

zvt.recorders.exchange.api.cs_index_api

zvt.recorders.exchange.api.cs_index_stock_api

zvt.recorders.exchange.exchange_etf_meta_recorder

zvt.recorders.exchange.exchange_index_recorder

zvt.recorders.exchange.exchange_index_stock_recorder

zvt.recorders.exchange.exchange_stock_meta_recorder

zvt.recorders.exchange.exchange_stock_summary_recorder

zvt.recorders.joinquant

zvt.recorders.joinquant.common

zvt.recorders.joinquant.fundamental

zvt.recorders.joinquant.fundamental.jq_etf_valuation_recorder

zvt.recorders.joinquant.fundamental.jq_margin_trading_recorder

zvt.recorders.joinquant.fundamental.jq_stock_valuation_recorder

zvt.recorders.joinquant.meta

zvt.recorders.joinquant.meta.jq_fund_meta_recorder

zvt.recorders.joinquant.meta.jq_stock_meta_recorder

zvt.recorders.joinquant.meta.jq_trade_day_recorder

zvt.recorders.joinquant.misc

zvt.recorders.joinquant.misc.jq_hk_holder_recorder

zvt.recorders.joinquant.misc.jq_index_money_flow_recorder

zvt.recorders.joinquant.misc.jq_stock_money_flow_recorder

zvt.recorders.joinquant.overall

zvt.recorders.joinquant.overall.jq_cross_market_recorder

zvt.recorders.joinquant.overall.jq_margin_trading_recorder

zvt.recorders.joinquant.overall.jq_stock_summary_recorder

zvt.recorders.joinquant.quotes

zvt.recorders.joinquant.quotes.jq_index_kdata_recorder

zvt.recorders.joinquant.quotes.jq_stock_kdata_recorder

zvt.recorders.jqka

zvt.recorders.jqka.emotion

zvt.recorders.jqka.emotion.JqkaEmotionRecorder

zvt.recorders.jqka.jqka_api

zvt.recorders.qmt

zvt.recorders.qmt.meta

zvt.recorders.sina

zvt.recorders.sina.meta

zvt.recorders.sina.meta.sina_block_recorder

zvt.recorders.sina.money_flow

zvt.recorders.sina.money_flow.sina_block_money_flow_recorder

zvt.recorders.sina.money_flow.sina_stock_money_flow_recorder

zvt.recorders.sina.quotes

zvt.recorders.sina.quotes.sina_etf_kdata_recorder

zvt.recorders.sina.quotes.sina_index_kdata_recorder

zvt.recorders.wb

zvt.recorders.wb.wb_api

zvt.recorders.wb.wb_country_recorder

zvt.recorders.wb.wb_economy_recorder

zvt.rest

zvt.rest.data

zvt.rest.factor

zvt.rest.trading

zvt.rest.work

zvt.samples

zvt.samples.stock_traders

zvt.sched

zvt.sched.sched

zvt.tag

zvt.tag.common

zvt.tag.tag_models

zvt.tag.tag_schemas

zvt.tag.tag_service

zvt.tag.tag_stats

zvt.tag.tag_utils

zvt.tag.tagger

zvt.trader

zvt.trader.errors

zvt.trader.sim_account

zvt.trader.trader

zvt.trader.trader_info_api

zvt.trader.trader_schemas

zvt.trading

zvt.trading.common

zvt.trading.trading_models

zvt.trading.trading_schemas

zvt.trading.trading_service

zvt.ui

zvt.ui.assets

zvt.ui.components

zvt.ui.components.dcc_components

zvt.utils

zvt.utils.decorator

zvt.utils.file_utils

zvt.utils.git_utils

zvt.utils.model_utils

zvt.utils.pd_utils

zvt.utils.recorder_utils

zvt.utils.str_utils

zvt.utils.time_utils

zvt.utils.utils

zvt.utils.zip_utils

zvt.zhdate

zvt.zhdate.constants

zvt.zhdate.zhdate

zvt.zhdate.ztime

Index

_ | A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | W | X | Y | Z

_

__init__() (zvt.api.stats.TopType method)
(zvt.api.stats.WindowMethod method)
(zvt.common.query_models.AbsoluteTimeRange method)
(zvt.common.query_models.OrderByType method)
(zvt.common.query_models.RelativeTimeRage method)
(zvt.common.query_models.TimeRange method)
(zvt.common.query_models.TimeUnit method)
(zvt.common.trading_models.BuyParameter method)
(zvt.common.trading_models.PositionType method)
(zvt.common.trading_models.SellParameter method)
(zvt.common.trading_models.TradingResult method)
(zvt.contract.ActorType method)
(zvt.contract.AdjustType method)
(zvt.contract.base_service.EntityStateService method)
(zvt.contract.base_service.OneStateService method)
(zvt.contract.base_service.StatefulService method)
(zvt.contract.context.Registry method)
(zvt.contract.data_type.Bean method)
(zvt.contract.drawer.ChartType method)
(zvt.contract.drawer.Draw method)
(zvt.contract.drawer.Drawable method)
(zvt.contract.drawer.Drawer method)
(zvt.contract.drawer.Rect method)
(zvt.contract.drawer.StackedDrawer method)
(zvt.contract.Exchange method)
(zvt.contract.factor.Accumulator method)
(zvt.contract.factor.Factor method)
(zvt.contract.factor.FactorMeta method)
(zvt.contract.factor.Indicator method)
(zvt.contract.factor.ScoreFactor method)
(zvt.contract.factor.Scorer method)
(zvt.contract.factor.TargetType method)
(zvt.contract.factor.Transformer method)
(zvt.contract.IntervalLevel method)
(zvt.contract.model.CustomModel method)
(zvt.contract.model.MixinModel method)
(zvt.contract.normal_data.NormalData method)
(zvt.contract.reader.DataListener method)
(zvt.contract.reader.DataReader method)
(zvt.contract.recorder.EntityEventRecorder method)
(zvt.contract.recorder.FixedCycleDataRecorder method)
(zvt.contract.recorder.Meta method)
(zvt.contract.recorder.Recorder method)
(zvt.contract.recorder.TimeSeriesDataRecorder method)
(zvt.contract.recorder.TimestampsDataRecorder method)
(zvt.contract.schema.ActorEntity method)
(zvt.contract.schema.ActorMeetTradable method)

(zvt.contract.schema.Entity method)
(zvt.contract.schema.Mixin method)
(zvt.contract.schema.NormalEntityMixin method)
(zvt.contract.schema.NormalMixin method)
(zvt.contract.schema.Portfolio method)
(zvt.contract.schema.PortfolioStock method)
(zvt.contract.schema.PortfolioStockHistory method)
(zvt.contract.schema.TradableEntity method)
(zvt.contract.schema.TradableMeetActor method)
(zvt.contract.TradableType method)
(zvt.contract.zvt_info.FactorState method)
(zvt.contract.zvt_info.RecorderState method)
(zvt.contract.zvt_info.StateMixin method)
(zvt.contract.zvt_info.TaggerState method)
(zvt.domain.actor.actor_meta.ActorMeta method)
(zvt.domain.actor.stock_actor.StockActorSummary method)
(zvt.domain.actor.stock_actor.StockInstitutionalInvestorHolder method)
(zvt.domain.actor.stock_actor.StockTopTenFreeHolder method)
(zvt.domain.actor.stock_actor.StockTopTenHolder method)
(zvt.domain.BlockCategory method)
(zvt.domain.CompanyType method)
(zvt.domain.emotion.emotion.Emotion method)
(zvt.domain.emotion.emotion.LimitDownInfo method)
(zvt.domain.emotion.emotion.LimitUpInfo method)
(zvt.domain.fundamental.dividend_financing.DividendDetail method)
(zvt.domain.fundamental.dividend_financing.DividendFinancing method)
(zvt.domain.fundamental.dividend_financing.RightsIssueDetail method)
(zvt.domain.fundamental.dividend_financing.SpoDetail method)
(zvt.domain.fundamental.finance.BalanceSheet method)
(zvt.domain.fundamental.finance.CashFlowStatement method)
(zvt.domain.fundamental.finance.FinanceFactor method)
(zvt.domain.fundamental.finance.IncomeStatement method)
(zvt.domain.fundamental.trading.BigDealTrading method)
(zvt.domain.fundamental.trading.DragonAndTiger method)
(zvt.domain.fundamental.trading.HolderTrading method)
(zvt.domain.fundamental.trading.ManagerTrading method)
(zvt.domain.fundamental.trading.MarginTrading method)
(zvt.domain.fundamental.valuation.EtfValuation method)
(zvt.domain.fundamental.valuation.StockValuation method)
(zvt.domain.IndexCategory method)
(zvt.domain.macro.macro.Economy method)
(zvt.domain.macro.monetary.TreasuryYield method)
(zvt.domain.meta.block_meta.Block method)
(zvt.domain.meta.block_meta.BlockStock method)
(zvt.domain.meta.country_meta.Country method)
(zvt.domain.meta.currency_meta.Currency method)
(zvt.domain.meta.etf_meta.Etf method)
(zvt.domain.meta.etf_meta.EtfStock method)
(zvt.domain.meta.fund_meta.Fund method)
(zvt.domain.meta.fund_meta.FundStock method)
(zvt.domain.meta.future_meta.Future method)
(zvt.domain.meta.index_meta.Index method)
(zvt.domain.meta.index_meta.IndexStock method)
(zvt.domain.meta.indexus_meta.Indexus method)
(zvt.domain.meta.stock_meta.Stock method)
(zvt.domain.meta.stock_meta.StockDetail method)

(zvt.domain.meta.stockhk_meta.Stockhk method)
(zvt.domain.meta.stockus_meta.Stockus method)
(zvt.domain.misc.holder.HkHolder method)
(zvt.domain.misc.holder.InstitutionalInvestorHolder method)
(zvt.domain.misc.holder.TopTenHolder method)
(zvt.domain.misc.holder.TopTenTradableHolder method)
(zvt.domain.misc.money_flow.BlockMoneyFlow method)
(zvt.domain.misc.money_flow.IndexMoneyFlow method)
(zvt.domain.misc.money_flow.StockMoneyFlow method)
(zvt.domain.misc.overall.CrossMarketSummary method)
(zvt.domain.misc.overall.MarginTradingSummary method)
(zvt.domain.misc.overall.StockSummary method)
(zvt.domain.misc.stock_news.StockNews method)
(zvt.domain.quotes.block.block_1d_kdata.Block1dKdata method)
(zvt.domain.quotes.block.block_1mon_kdata.Block1monKdata method)
(zvt.domain.quotes.block.block_1wk_kdata.Block1wkKdata method)
(zvt.domain.quotes.BlockKdataCommon method)
(zvt.domain.quotes.currency.currency_1d_kdata.Currency1dKdata method)
(zvt.domain.quotes.CurrencyKdataCommon method)
(zvt.domain.quotes.etf.etf_1d_kdata.Etf1dKdata method)
(zvt.domain.quotes.EtfKdataCommon method)
(zvt.domain.quotes.future.future_1d_kdata.Future1dKdata method)
(zvt.domain.quotes.FutureKdataCommon method)
(zvt.domain.quotes.index.index_1d_kdata.Index1dKdata method)
(zvt.domain.quotes.index.index_1wk_kdata.Index1wkKdata method)
(zvt.domain.quotes.IndexKdataCommon method)
(zvt.domain.quotes.indexus.indexus_1d_kdata.Indexus1dKdata method)
(zvt.domain.quotes.IndexusKdataCommon method)
(zvt.domain.quotes.KdataCommon method)
(zvt.domain.quotes.stock.stock_15m_hfq_kdata.Stock15mHfqKdata method)
(zvt.domain.quotes.stock.stock_15m_kdata.Stock15mKdata method)
(zvt.domain.quotes.stock.stock_1d_hfq_kdata.Stock1dHfqKdata method)
(zvt.domain.quotes.stock.stock_1d_kdata.Stock1dKdata method)
(zvt.domain.quotes.stock.stock_1h_hfq_kdata.Stock1hHfqKdata method)
(zvt.domain.quotes.stock.stock_1h_kdata.Stock1hKdata method)
(zvt.domain.quotes.stock.stock_1m_hfq_kdata.Stock1mHfqKdata method)
(zvt.domain.quotes.stock.stock_1m_kdata.Stock1mKdata method)
(zvt.domain.quotes.stock.stock_1mon_hfq_kdata.Stock1monHfqKdata method)
(zvt.domain.quotes.stock.stock_1mon_kdata.Stock1monKdata method)
(zvt.domain.quotes.stock.stock_1wk_hfq_kdata.Stock1wkHfqKdata method)
(zvt.domain.quotes.stock.stock_1wk_kdata.Stock1wkKdata method)
(zvt.domain.quotes.stock.stock_30m_hfq_kdata.Stock30mHfqKdata method)
(zvt.domain.quotes.stock.stock_30m_kdata.Stock30mKdata method)
(zvt.domain.quotes.stock.stock_4h_hfq_kdata.Stock4hHfqKdata method)
(zvt.domain.quotes.stock.stock_4h_kdata.Stock4hKdata method)
(zvt.domain.quotes.stock.stock_5m_hfq_kdata.Stock5mHfqKdata method)
(zvt.domain.quotes.stock.stock_5m_kdata.Stock5mKdata method)
(zvt.domain.quotes.stock.stock_quote.StockQuote method)
(zvt.domain.quotes.stockhk.stockhk_1d_hfq_kdata.Stockhk1dHfqKdata method)
(zvt.domain.quotes.stockhk.stockhk_1d_kdata.Stockhk1dKdata method)
(zvt.domain.quotes.StockhkKdataCommon method)
(zvt.domain.quotes.StockKdataCommon method)
(zvt.domain.quotes.stockus.stockus_1d_hfq_kdata.Stockus1dHfqKdata method)
(zvt.domain.quotes.stockus.stockus_1d_kdata.Stockus1dKdata method)
(zvt.domain.quotes.StockusKdataCommon method)
(zvt.domain.quotes.TickCommon method)

(zvt.domain.quotes.trade_day.StockTradeDay method)
(zvt.domain.ReportPeriod method)
(zvt.factors.algorithm.IntersectTransformer method)
(zvt.factors.algorithm.MaAndVolumeTransformer method)
(zvt.factors.algorithm.MacdTransformer method)
(zvt.factors.algorithm.MaTransformer method)
(zvt.factors.algorithm.QuantileScorer method)
(zvt.factors.algorithm.RankScorer method)
(zvt.factors.factor_models.FactorRequestModel method)
(zvt.factors.factor_models.FactorResultModel method)
(zvt.factors.factor_models.KdataModel method)
(zvt.factors.factor_models.KdataRequestModel method)
(zvt.factors.factor_models.TradingSignalModel method)
(zvt.factors.fundamental.finance_factor.FinanceBaseFactor method)
(zvt.factors.fundamental.finance_factor.GoodCompanyFactor method)
(zvt.factors.ma.domain.common.MaStatsFactorCommon method)
(zvt.factors.ma.domain.stock_1d_ma_factor.Stock1dMaFactor method)
(zvt.factors.ma.domain.stock_1d_ma_stats_factor.Stock1dMaStatsFactor method)
(zvt.factors.ma.ma_factor.CrossMaFactor method)
(zvt.factors.ma.ma_factor.CrossMaVolumeFactor method)
(zvt.factors.ma.ma_factor.MaFactor method)
(zvt.factors.ma.ma_factor.VolumeUpMaFactor method)
(zvt.factors.ma.ma_stats_factor.MaStatsAccumulator method)
(zvt.factors.ma.ma_stats_factor.MaStatsFactor method)
(zvt.factors.ma.ma_stats_factor.TFactor method)
(zvt.factors.ma.top_bottom_factor.TopBottomFactor method)
(zvt.factors.ma.top_bottom_factor.TopBottomTransformer method)
(zvt.factors.macd.macd_factor.BullFactor method)
(zvt.factors.macd.macd_factor.GoldCrossFactor method)
(zvt.factors.macd.macd_factor.KeepBullFactor method)
(zvt.factors.macd.macd_factor.LiveOrDeadFactor method)
(zvt.factors.macd.macd_factor.MacdFactor method)
(zvt.factors.shape.Direction method)
(zvt.factors.shape.FactorStateEncoder method)
(zvt.factors.shape.Fenxing method)
(zvt.factors.target_selector.SelectMode method)
(zvt.factors.target_selector.TargetSelector method)
(zvt.factors.target_selector.TradeType method)
(zvt.factors.technical_factor.TechnicalFactor method)
(zvt.factors.top_stocks.TopStocks method)
(zvt.factors.transformers.CrossMaTransformer method)
(zvt.factors.transformers.FallBelowTransformer method)
(zvt.factors.transformers.SpecificTransformer method)
(zvt.factors.z.domain.common.ZFactorCommon method)
(zvt.factors.z.domain.stock_1d_z_factor.Stock1dZFactor method)
(zvt.factors.z.domain.stock_1wk_z_factor.Stock1wkZFactor method)
(zvt.factors.z.z_factor.ZAccumulator method)
(zvt.factors.z.z_factor.ZFactor method)
(zvt.factors.z.z_factor.ZState method)
(zvt.informer.informer.EmailInformer method)
(zvt.informer.informer.Informer method)
(zvt.informer.informer.WechatInformer method)
(zvt.ml.lables.BehaviorCategory method)
(zvt.ml.lables.RelativePerformance method)
(zvt.ml.ml.MaStockMLMachine method)
(zvt.ml.ml.MLMachine method)

(zvt.ml.ml.StockMLMachine method)
(zvt.recorders.eastmoney.common.ApiWrapper method)
(zvt.recorders.eastmoney.common.BaseEastmoneyRecorder method)
(zvt.recorders.eastmoney.common.EastmoneyApiWrapper method)
(zvt.recorders.eastmoney.common.EastmoneyMoreDataRecorder method)
(zvt.recorders.eastmoney.common.EastmoneyPageabeDataRecorder method)
(zvt.recorders.eastmoney.common.EastmoneyTimestampsDataRecorder method)
(zvt.recorders.eastmoney.dividend_financing.eastmoney_dividend_detail_recorder.DividendDetailRecord
method)
(zvt.recorders.eastmoney.dividend_financing.eastmoney_dividend_financing_recorder.DividendFinancing
method)
(zvt.recorders.eastmoney.dividend_financing.eastmoney_rights_issue_detail_recorder.RightsIssueDetailR
method)
(zvt.recorders.eastmoney.dividend_financing.eastmoney_spo_detail_recorder.SPODetailRecorder method
(zvt.recorders.eastmoney.finance.base_china_stock_finance_recorder.BaseChinaStockFinanceRecorder m
(zvt.recorders.eastmoney.finance.eastmoney_balance_sheet_recorder.ChinaStockBalanceSheetRecorder n
(zvt.recorders.eastmoney.finance.eastmoney_cash_flow_recorder.ChinaStockCashFlowRecorder method)
(zvt.recorders.eastmoney.finance.eastmoney_finance_factor_recorder.ChinaStockFinanceFactorRecorder
(zvt.recorders.eastmoney.finance.eastmoney_income_statement_recorder.ChinaStockIncomeStatementRe
method)
(zvt.recorders.eastmoney.holder.eastmoney_stock_actor_recorder.EastmoneyActorRecorder method)
(zvt.recorders.eastmoney.holder.eastmoney_top_ten_holder_recorder.TopTenHolderRecorder method)
(zvt.recorders.eastmoney.holder.eastmoney_top_ten_tradable_holder_recorder.TopTenTradableHolderRec
method)
(zvt.recorders.eastmoney.meta.eastmoney_block_meta_recorder.EastmoneyBlockRecorder method)
(zvt.recorders.eastmoney.meta.eastmoney_block_meta_recorder.EastmoneyBlockStockRecorder method)
(zvt.recorders.eastmoney.meta.eastmoney_stock_meta_recorder.EastmoneyStockDetailRecorder method)
(zvt.recorders.eastmoney.meta.eastmoney_stock_meta_recorder.EastmoneyStockRecorder method)
(zvt.recorders.eastmoney.trading.eastmoney_holder_trading_recorder.HolderTradingRecorder method)
(zvt.recorders.eastmoney.trading.eastmoney_manager_trading_recorder.ManagerTradingRecorder method
(zvt.recorders.em.actor.em_stock_actor_summary_recorder.EMStockActorSummaryRecorder method)
(zvt.recorders.em.actor.em_stock_ii_recorder.EMStockIIRecorder method)
(zvt.recorders.em.actor.em_stock_top_ten_free_recorder.EMStockTopTenFreeRecorder method)
(zvt.recorders.em.actor.em_stock_top_ten_recorder.EMStockTopTenRecorder method)
(zvt.recorders.em.macro.em_treasury_yield_recorder.EMTreasuryYieldRecorder method)
(zvt.recorders.em.meta.em_block_meta_recorder.EMBlockRecorder method)
(zvt.recorders.em.meta.em_block_meta_recorder.EMBlockStockRecorder method)
(zvt.recorders.em.meta.em_currency_meta_recorder.EMCurrencyRecorder method)
(zvt.recorders.em.meta.em_future_meta_recorder.EMFutureRecorder method)
(zvt.recorders.em.meta.em_index_meta_recorder.EMIndexRecorder method)
(zvt.recorders.em.meta.em_indexus_meta_recorder.EMIndexusRecorder method)
(zvt.recorders.em.meta.em_stock_meta_recorder.EMStockRecorder method)
(zvt.recorders.em.meta.em_stockhk_meta_recorder.EMStockhkRecorder method)
(zvt.recorders.em.meta.em_stockus_meta_recorder.EMStockusRecorder method)
(zvt.recorders.em.misc.em_stock_news_recorder.EMStockNewsRecorder method)
(zvt.recorders.em.quotes.em_kdata_recorder.BaseEMStockKdataRecorder method)
(zvt.recorders.em.quotes.em_kdata_recorder.EMBlockKdataRecorder method)
(zvt.recorders.em.quotes.em_kdata_recorder.EMCurrencyKdataRecorder method)
(zvt.recorders.em.quotes.em_kdata_recorder.EMFutureKdataRecorder method)
(zvt.recorders.em.quotes.em_kdata_recorder.EMIndexKdataRecorder method)
(zvt.recorders.em.quotes.em_kdata_recorder.EMIndexusKdataRecorder method)
(zvt.recorders.em.quotes.em_kdata_recorder.EMStockhkKdataRecorder method)
(zvt.recorders.em.quotes.em_kdata_recorder.EMStockKdataRecorder method)
(zvt.recorders.em.quotes.em_kdata_recorder.EMStockusKdataRecorder method)
(zvt.recorders.em.trading.em_dragon_and_tiger_recorder.EMDragonAndTigerRecorder method)
(zvt.recorders.exchange.exchange_etf_meta_recorder.ChinaETFListSpider method)

(zvt.recorders.exchange.exchange_index_recorder.ExchangeIndexRecorder method)
(zvt.recorders.exchange.exchange_index_stock_recorder.ExchangeIndexStockRecorder method)
(zvt.recorders.exchange.exchange_stock_meta_recorder.ExchangeStockMetaRecorder method)
(zvt.recorders.exchange.exchange_stock_summary_recorder.ExchangeStockSummaryRecorder method)
(zvt.recorders.joinquant.fundamental.jq_etf_valuation_recorder.JqChinaEtfValuationRecorder method)
(zvt.recorders.joinquant.fundamental.jq_margin_trading_recorder.MarginTradingRecorder method)
(zvt.recorders.joinquant.fundamental.jq_stock_valuation_recorder.JqChinaStockValuationRecorder metho
(zvt.recorders.joinquant.meta.jq_fund_meta_recorder.JqChinaFundRecorder method)
(zvt.recorders.joinquant.meta.jq_fund_meta_recorder.JqChinaFundStockRecorder method)
(zvt.recorders.joinquant.meta.jq_stock_meta_recorder.BaseJqChinaMetaRecorder method)
(zvt.recorders.joinquant.meta.jq_stock_meta_recorder.JqChinaEtfRecorder method)
(zvt.recorders.joinquant.meta.jq_stock_meta_recorder.JqChinaStockEtfPortfolioRecorder method)
(zvt.recorders.joinquant.meta.jq_stock_meta_recorder.JqChinaStockRecorder method)
(zvt.recorders.joinquant.meta.jq_trade_day_recorder.StockTradeDayRecorder method)
(zvt.recorders.joinquant.misc.jq_hk_holder_recorder.JoinquantHkHolderRecorder method)
(zvt.recorders.joinquant.misc.jq_index_money_flow_recorder.JoinquantIndexMoneyFlowRecorder metho
(zvt.recorders.joinquant.misc.jq_stock_money_flow_recorder.JoinquantStockMoneyFlowRecorder metho
(zvt.recorders.joinquant.overall.jq_cross_market_recorder.CrossMarketSummaryRecorder method)
(zvt.recorders.joinquant.overall.jq_margin_trading_recorder.MarginTradingSummaryRecorder method)
(zvt.recorders.joinquant.overall.jq_stock_summary_recorder.StockSummaryRecorder method)
(zvt.recorders.joinquant.quotes.jq_index_kdata_recorder.JqChinaIndexKdataRecorder method)
(zvt.recorders.joinquant.quotes.jq_stock_kdata_recorder.JqChinaStockKdataRecorder method)
(zvt.recorders.sina.meta.sina_block_recorder.SinaBlockRecorder method)
(zvt.recorders.sina.meta.sina_block_recorder.SinaChinaBlockStockRecorder method)
(zvt.recorders.sina.money_flow.sina_block_money_flow_recorder.SinaBlockMoneyFlowRecorder metho
(zvt.recorders.sina.money_flow.sina_stock_money_flow_recorder.SinaStockMoneyFlowRecorder metho
(zvt.recorders.sina.quotes.sina_etf_kdata_recorder.ChinaETFDayKdataRecorder method)
(zvt.recorders.sina.quotes.sina_index_kdata_recorder.ChinaIndexDayKdataRecorder method)
(zvt.recorders.wb.wb_country_recorder.WBCountryRecorder method)
(zvt.recorders.wb.wb_economy_recorder.WBEconomyRecorder method)
(zvt.samples.stock_traders.MyBullTrader method)
(zvt.samples.stock_traders.MyMaTrader method)
(zvt.tag.common.StockPoolType method)
(zvt.tag.common.TagType method)
(zvt.tag.tag_models.ActivateSubTagsModel method)
(zvt.tag.tag_models.ActivateSubTagsResultModel method)
(zvt.tag.tag_models.BatchSetStockTagsModel method)
(zvt.tag.tag_models.CreateStockPoolInfoModel method)
(zvt.tag.tag_models.CreateStockPoolsModel method)
(zvt.tag.tag_models.CreateTagInfoModel method)
(zvt.tag.tag_models.QuerySimpleStockTagsModel method)
(zvt.tag.tag_models.QueryStockTagsModel method)
(zvt.tag.tag_models.QueryStockTagStatsModel method)
(zvt.tag.tag_models.SetStockTagsModel method)
(zvt.tag.tag_models.SimpleStockTagsModel method)
(zvt.tag.tag_models.StockPoolInfoModel method)
(zvt.tag.tag_models.StockPoolModel method)
(zvt.tag.tag_models.StockPoolsModel method)
(zvt.tag.tag_models.StockTagDetailsModel method)
(zvt.tag.tag_models.StockTagsModel method)
(zvt.tag.tag_models.StockTagStatsModel method)
(zvt.tag.tag_models.TagInfoModel method)
(zvt.tag.tag_models.TagParameter method)
(zvt.tag.tag_schemas.HiddenTagInfo method)
(zvt.tag.tag_schemas.MainTagInfo method)
(zvt.tag.tag_schemas.StockPoolInfo method)

(zvt.tag.tag_schemas.StockPools method)
(zvt.tag.tag_schemas.StockSystemTags method)
(zvt.tag.tag_schemas.StockTags method)
(zvt.tag.tag_schemas.SubTagInfo method)
(zvt.tag.tag_schemas.TagStats method)
(zvt.tag.tag_service.StockAutoTagger method)
(zvt.tag.tagger.StockTagger method)
(zvt.tag.tagger.Tagger method)
(zvt.trader.AccountService method)
(zvt.trader.OrderType method)
(zvt.trader.sim_account.AccountDayStatsSchema method)
(zvt.trader.sim_account.PositionSchema method)
(zvt.trader.sim_account.SimAccountService method)
(zvt.trader.trader.StockTrader method)
(zvt.trader.trader.Trader method)
(zvt.trader.trader_info_api.AccountStatsReader method)
(zvt.trader.trader_info_api.OrderReader method)
(zvt.trader.trader_schemas.AccountStats method)
(zvt.trader.trader_schemas.Order method)
(zvt.trader.trader_schemas.Position method)
(zvt.trader.trader_schemas.TraderInfo method)
(zvt.trader.TradingListener method)
(zvt.trader.TradingSignal method)
(zvt.trader.TradingSignalType method)
(zvt.trading.common.ExecutionStatus method)
(zvt.trading.trading_models.BuildQueryStockQuoteSettingModel method)
(zvt.trading.trading_models.BuildTradingPlanModel method)
(zvt.trading.trading_models.QueryStockQuoteModel method)
(zvt.trading.trading_models.QueryStockQuoteSettingModel method)
(zvt.trading.trading_models.QueryTagQuoteModel method)
(zvt.trading.trading_models.QueryTradingPlanModel method)
(zvt.trading.trading_models.StockQuoteModel method)
(zvt.trading.trading_models.StockQuoteStatsModel method)
(zvt.trading.trading_models.TagQuoteStatsModel method)
(zvt.trading.trading_models.TradingPlanModel method)
(zvt.trading.trading_schemas.QueryStockQuoteSetting method)
(zvt.trading.trading_schemas.TradingPlan method)
(zvt.zhdate.zhdate.ZhDate method)

A

a_include_b() (in module zvt.factors.shape)
AbsoluteTimeRange (class in zvt.common.query_models)
acc() (zvt.contract.factor.Accumulator method)
(zvt.factors.ma.ma_stats_factor.MaStatsAccumulator method)
(zvt.factors.z.z_factor.ZAccumulator method)
acc_one() (zvt.contract.factor.Accumulator method)
(zvt.factors.ma.ma_stats_factor.MaStatsAccumulator method)
(zvt.factors.z.z_factor.ZAccumulator method)
accept_money_deposits
(zvt.domain.fundamental.finance.BalanceSheet attribute)
account_stats_id (zvt.trader.trader_schemas.Position attribute)
AccountDayStatsSchema (class in zvt.trader.sim_account)
AccountDayStatsSchema.Meta (class in zvt.trader.sim_account)
actor_name (zvt.domain.actor.stock_a
(zvt.domain.actor.stock_actor.St
(zvt.domain.actor.stock_actor.St
(zvt.domain.actor.stock_actor.St
actor_type (zvt.domain.actor.stock_ac
(zvt.domain.actor.stock_actor.St
(zvt.domain.actor.stock_actor.St
(zvt.domain.actor.stock_actor.St
actor_type_to_org_type() (in module
ActorEntity (class in zvt.contract.sch
ActorMeetTradable (class in zvt.cont
ActorMeta (class in zvt.domain.actor
ActorType (class in zvt.contract)
actual_tax_rate (zvt.domain.fundame

accounts_payable (zvt.domain.fundamental.finance.BalanceSheet attribute)
accounts_receivable (zvt.domain.fundamental.finance.BalanceSheet attribute)
AccountService (class in zvt.trader)
AccountStats (class in zvt.trader.trader_schemas)
AccountStatsReader (class in zvt.trader.trader_info_api)
Accumulator (class in zvt.contract.factor)
accumulator (zvt.contract.factor.Factor attribute)
(zvt.contract.factor.ScoreFactor attribute)
(zvt.factors.fundamental.finance_factor.FinanceBaseFactor attribute)
(zvt.factors.fundamental.finance_factor.GoodCompanyFactor attribute)
(zvt.factors.ma.ma_factor.CrossMaFactor attribute)
(zvt.factors.ma.ma_factor.CrossMaVolumeFactor attribute)
(zvt.factors.ma.ma_factor.MaFactor attribute)
(zvt.factors.ma.ma_factor.VolumeUpMaFactor attribute)
(zvt.factors.ma.ma_stats_factor.MaStatsFactor attribute)
(zvt.factors.ma.ma_stats_factor.TFactor attribute)
(zvt.factors.ma.top_bottom_factor.TopBottomFactor attribute)
(zvt.factors.macd.macd_factor.BullFactor attribute)
(zvt.factors.macd.macd_factor.GoldCrossFactor attribute)
(zvt.factors.macd.macd_factor.KeepBullFactor attribute)
(zvt.factors.macd.macd_factor.LiveOrDeadFactor attribute)
(zvt.factors.macd.macd_factor.MacdFactor attribute)
(zvt.factors.technical_factor.TechnicalFactor attribute)
(zvt.factors.z.z_factor.ZFactor attribute)
activate_main_tag_by_industry() (in module zvt.tag.tag_service)
activate_main_tag_by_sub_tags() (in module zvt.tag.tag_service)
activate_sub_tags() (in module zvt.rest.work)
ActivateSubTagsModel (class in zvt.tag.tag_models)
ActivateSubTagsResultModel (class in zvt.tag.tag_models)
actor_code (zvt.domain.actor.stock_actor.StockActorSummary attribute)
(zvt.domain.actor.stock_actor.StockInstitutionalInvestorHolder attribute)
(zvt.domain.actor.stock_actor.StockTopTenFreeHolder attribute)
(zvt.domain.actor.stock_actor.StockTopTenHolder attribute)
actor_id (zvt.domain.actor.stock_actor.StockActorSummary attribute)
(zvt.domain.actor.stock_actor.StockInstitutionalInvestorHolder attribute)
(zvt.domain.actor.stock_actor.StockTopTenFreeHolder attribute)
(zvt.domain.actor.stock_actor.StockTopTenHolder attribute)
add_func_to_value() (in module zvt.u
add_to_eastmoney() (in module zvt.i
add_to_map_list() (in module zvt.util
AdjustType (class in zvt.contract)
advance_receipts_per_op_income (zvt.domain.fundamental.finance.Fin
advances_from_customers (zvt.doma attribute)
advances_to_suppliers (zvt.domain.fu attribute)
advisor (zvt.domain.meta.fund_meta.
all_sub_all() (in module zvt.autocode
all_sub_modules() (in module zvt.aut
all_tpls() (in module zvt.autocode.ten
all_value (zvt.trader.trader_schemas.
annotation_df (zvt.contract.drawer.D
annotations() (in module zvt.contract
announce_date (zvt.domain.fundame attribute)
ApiWrapper (class in zvt.recorders.ea
area (zvt.contract.drawer.ChartType a
(zvt.domain.BlockCategory attri
(zvt.factors.ma.domain.stock_1c attribute)
area_indices (zvt.domain.meta.stock_
ask_amount (zvt.domain.quotes.stock
(zvt.trading.trading_models.Stoc
ask_price (zvt.domain.quotes.TickCo
ask_vol (zvt.domain.quotes.TickCom
assets_devaluation (zvt.domain.funda attribute)
available_long (zvt.trader.trader_sche
available_short (zvt.trader.trader_sch
average_long_price (zvt.trader.trader
average_short_price (zvt.trader.trade

B

BalanceSheet (class in zvt.domain.fundamental.finance)
bar (zvt.contract.drawer.ChartType attribute)
base_point (zvt.domain.meta.index_meta.Index attribute)
(zvt.domain.meta.indexus_meta.Indexus attribute)
block (zvt.contract.TradableType att
Block1dKdata (class in zvt.domain.quotes.block.block_1d_
Block1monKdata (class in zvt.domain.quotes.block.block_1mo

BaseChinaStockFinanceRecorder (class in zvt.recorders.eastmoney.finance.base_china_stock_finance_recorder)
BaseEastmoneyRecorder (class in zvt.recorders.eastmoney.common)
BaseEMStockKdataRecorder (class in zvt.recorders.em.quotes.em_kdata_recorder)
BaseJqChinaMetaRecorder (class in zvt.recorders.joinquant.meta.jq_stock_meta_recorder)
basic_eps (zvt.domain.fundamental.finance.FinanceFactor attribute)
batch_set_stock_tags() (in module zvt.rest.work)
(in module zvt.tag.tag_service)
BatchSetStockTagsModel (class in zvt.tag.tag_models)
Bean (class in zvt.contract.data_type)
BehaviorCategory (class in zvt.ml.lables)
bfq (zvt.contract.AdjustType attribute)
bi_change (zvt.factors.z.domain.stock_1d_z_factor.Stock1dZFactor attribute)
(zvt.factors.z.domain.stock_1wk_z_factor.Stock1wkZFactor attribute)
bi_di (zvt.factors.z.domain.stock_1d_z_factor.Stock1dZFactor attribute)
(zvt.factors.z.domain.stock_1wk_z_factor.Stock1wkZFactor attribute)
bi_ding (zvt.factors.z.domain.stock_1d_z_factor.Stock1dZFactor attribute)
(zvt.factors.z.domain.stock_1wk_z_factor.Stock1wkZFactor attribute)
bi_interval (zvt.factors.z.domain.stock_1d_z_factor.Stock1dZFactor attribute)
(zvt.factors.z.domain.stock_1wk_z_factor.Stock1wkZFactor attribute)
bi_slope (zvt.factors.z.domain.stock_1d_z_factor.Stock1dZFactor attribute)
(zvt.factors.z.domain.stock_1wk_z_factor.Stock1wkZFactor attribute)
bi_value (zvt.factors.z.domain.stock_1d_z_factor.Stock1dZFactor attribute)
(zvt.factors.z.domain.stock_1wk_z_factor.Stock1wkZFactor attribute)
bi_zhongshu (zvt.factors.z.domain.stock_1d_z_factor.Stock1dZFactor attribute)
(zvt.factors.z.domain.stock_1wk_z_factor.Stock1wkZFactor attribute)
bi_zhongshu_change (zvt.factors.z.domain.stock_1d_z_factor.Stock1dZFactor attribute)
(zvt.factors.z.domain.stock_1wk_z_factor.Stock1wkZFactor attribute)
bid_amount (zvt.domain.quotes.stock.stock_quote.StockQuote attribute)
(zvt.trading.trading_models.StockQuoteModel attribute)
bid_price (zvt.domain.quotes.TickCommon attribute)
bid_vol (zvt.domain.quotes.TickCommon attribute)
BigDealTrading (class in zvt.domain.fundamental.trading)
binance (zvt.contract.Exchange attribute)
bj (zvt.contract.Exchange attribute)
Block (class in zvt.domain.meta.block_meta)
Block1wkKdata (class in zvt.domain.quotes.block.block_1wk
BlockCategory (class in zvt.domain
BlockKdataCommon (class in zvt.d
BlockMoneyFlow (class in zvt.dom
BlockStock (class in zvt.domain.me
bps (zvt.domain.fundamental.financ
broker (zvt.contract.ActorType attri
broker_net_assets (zvt.domain.fundamental.finance.Fi
broker_net_capital (zvt.domain.fundamental.finance.Fi
broker_net_capital_assets_ratio (zvt.domain.fundamental.finance.Fi
broker_self_operated_fixed_income (zvt.domain.fundamental.finance.Fi
build_feature() (zvt.ml.ml.MaStock
(zvt.ml.ml.MLMachine metho
(zvt.ml.ml.StockMLMachine n
build_initial_hidden_tag_info() (in
build_initial_main_tag_info() (in m
build_initial_stock_pool_info() (in
build_initial_sub_tag_info() (in mo
build_query_stock_quote_setting() (
(in module zvt.trading.trading_
build_stock_pool() (in module zvt.t
build_stock_pool_info() (in module
build_stock_pool_tag_stats() (in mo
build_stock_tags() (in module zvt.re
(in module zvt.tag.tag_service)
build_system_stock_pools() (in mo
build_tag_parameter() (in module z
build_trading_plan() (in module zvt
(in module zvt.trading.trading_
BuildQueryStockQuoteSettingMode
zvt.trading.trading_models)
BuildTradingPlanModel (class in zv
BullFactor (class in zvt.factors.mac
business_taxes_and_surcharges (zvt.domain.fundamental.finance.In
buy() (in module zvt.rest.trading)
buy_broker (zvt.domain.fundamenta
attribute)
buy_stocks() (in module zvt.trading
BuyParameter (class in zvt.common

C

cal_behavior_cls() (in module zvt.ml.ml)
cal_change() (in module zvt.ml.ml)
cal_predict() (in module zvt.ml.ml)
cal_relative_performance() (in module zvt.ml.ml)
call_eastmoney_api() (in module zvt.recorders.eastmoney.common)
capital (zvt.domain.fundamental.finance.BalanceSheet attribute)
capital_city (zvt.domain.meta.country_meta.Country attribute)
capital_reserve (zvt.domain.fundamental.finance.BalanceSheet attribute)
capital_reserve_ps (zvt.domain.fundamental.finance.FinanceFactor attribute)
capitalization (zvt.domain.fundamental.valuation.StockValuation attribute)
cash (zvt.domain.fundamental.finance.CashFlowStatement attribute)
(zvt.trader.trader_schemas.AccountStats attribute)
cash_and_cash_equivalents (zvt.domain.fundamental.finance.BalanceSheet attribute)
cash_at_beginning (zvt.domain.fundamental.finance.CashFlowStatement attribute)
cash_flow_ratio (zvt.domain.fundamental.finance.FinanceFactor attribute)
cash_from_accepting_investment (zvt.domain.fundamental.finance.CashFlowStatement attribute)
cash_from_borrowings (zvt.domain.fundamental.finance.CashFlowStatement attribute)
cash_from_disposal_fixed_intangible_assets (zvt.domain.fundamental.finance.CashFlowStatement attribute)
cash_from_disposal_of_investments (zvt.domain.fundamental.finance.CashFlowStatement attribute)
cash_from_disposal_subsidiaries (zvt.domain.fundamental.finance.CashFlowStatement attribute)
cash_from_issuing_bonds (zvt.domain.fundamental.finance.CashFlowStatement attribute)
cash_from_other_financing (zvt.domain.fundamental.finance.CashFlowStatement attribute)
cash_from_other_investing (zvt.domain.fundamental.finance.CashFlowStatement attribute)
cash_from_other_op (zvt.domain.fundamental.finance.CashFlowStatement attribute)
cash_from_returns_on_investments (zvt.domain.fundamental.finance.CashFlowStatement attribute)
cash_from_selling (zvt.domain.fundamental.finance.CashFlowStatement attribute)
cash_from_subsidiaries_accepting_minority_interest (zvt.domain.fundamental.finance.CashFlowStatement att
cash_to_acquire_fixed_intangible_assets (zvt.domain.fundamental.finance.CashFlowStatement attribute)
cash_to_acquire_subsidiaries (zvt.domain.fundamental.finance.CashFlowStatement attribute)
cash_to_employees (zvt.domain.fundamental.finance.CashFlowStatement attribute)
cash_to_goods_services (zvt.domain.fundamental.finance.CashFlowStatement attribute)
cash_to_investments (zvt.domain.fundamental.finance.CashFlowStatement attribute)
cash_to_other_financing (zvt.domain.fundamental.finance.CashFlowStatement attribute)
cash_to_other_investing (zvt.domain.fundamental.finance.CashFlowStatement attribute)
cash_to_other_related_op (zvt.domain.fundamental.finance.CashFlowStatement attribute)
cash_to_pay_interest_dividend (zvt.domain.fundamental.finance.CashFlowStatement attribute)
cash_to_pay_subsidiaries_minority_interest (zvt.domain.fundamental.finance.CashFlowStatement attribute)
cash_to_repay_borrowings (zvt.domain.fundamental.finance.CashFlowStatement attribute)
CashFlowStatement (class in zvt.domain.fundamental.finance)
category (zvt.domain.meta.block_meta.Block attribute)
(zvt.domain.meta.index_meta.Index attribute)
(zvt.domain.meta.indexus_meta.Indexus attribute)
cffex (zvt.contract.Exchange attribute)
change (zvt.domain.misc.holder.TopTenHolder attribute)
(zvt.domain.misc.holder.TopTenTradableHolder attribute)
change_pct (zvt.domain.emotion.emotion.LimitDownInfo attribute)
(zvt.domain.emotion.emotion.LimitUpInfo attribute)
(zvt.domain.fundamental.trading.DragonAndTiger attribute)
(zvt.domain.fundamental.trading.HolderTrading attribute)
(zvt.domain.quotes.block.block_1d_kdata.Block1dKdata attribute)
(zvt.domain.quotes.block.block_1mon_kdata.Block1monKdata attribute)
(zvt.domain.quotes.block.block_1wk_kdata.Block1wkKdata attribute)
(zvt.domain.quotes.currency.currency_1d_kdata.Currency1dKdata attribute)

(zvt.domain.quotes.etf.etf_1d_kdata.Etf1dKdata attribute)
(zvt.domain.quotes.future.future_1d_kdata.Future1dKdata attribute)
(zvt.domain.quotes.index.index_1d_kdata.Index1dKdata attribute)
(zvt.domain.quotes.index.index_1wk_kdata.Index1wkKdata attribute)
(zvt.domain.quotes.indexus.indexus_1d_kdata.Indexus1dKdata attribute)
(zvt.domain.quotes.stock.stock_15m_hfq_kdata.Stock15mHfqKdata attribute)
(zvt.domain.quotes.stock.stock_15m_kdata.Stock15mKdata attribute)
(zvt.domain.quotes.stock.stock_1d_hfq_kdata.Stock1dHfqKdata attribute)
(zvt.domain.quotes.stock.stock_1d_kdata.Stock1dKdata attribute)
(zvt.domain.quotes.stock.stock_1h_hfq_kdata.Stock1hHfqKdata attribute)
(zvt.domain.quotes.stock.stock_1h_kdata.Stock1hKdata attribute)
(zvt.domain.quotes.stock.stock_1m_hfq_kdata.Stock1mHfqKdata attribute)
(zvt.domain.quotes.stock.stock_1m_kdata.Stock1mKdata attribute)
(zvt.domain.quotes.stock.stock_1mon_hfq_kdata.Stock1monHfqKdata attribute)
(zvt.domain.quotes.stock.stock_1mon_kdata.Stock1monKdata attribute)
(zvt.domain.quotes.stock.stock_1wk_hfq_kdata.Stock1wkHfqKdata attribute)
(zvt.domain.quotes.stock.stock_1wk_kdata.Stock1wkKdata attribute)
(zvt.domain.quotes.stock.stock_30m_hfq_kdata.Stock30mHfqKdata attribute)
(zvt.domain.quotes.stock.stock_30m_kdata.Stock30mKdata attribute)
(zvt.domain.quotes.stock.stock_4h_hfq_kdata.Stock4hHfqKdata attribute)
(zvt.domain.quotes.stock.stock_4h_kdata.Stock4hKdata attribute)
(zvt.domain.quotes.stock.stock_5m_hfq_kdata.Stock5mHfqKdata attribute)
(zvt.domain.quotes.stock.stock_5m_kdata.Stock5mKdata attribute)
(zvt.domain.quotes.stockhk.stockhk_1d_hfq_kdata.Stockhk1dHfqKdata attribute)
(zvt.domain.quotes.stockhk.stockhk_1d_kdata.Stockhk1dKdata attribute)
(zvt.domain.quotes.stockus.stockus_1d_hfq_kdata.Stockus1dHfqKdata attribute)
(zvt.domain.quotes.stockus.stockus_1d_kdata.Stockus1dKdata attribute)
(zvt.trading.trading_models.StockQuoteStatsModel attribute)
(zvt.trading.trading_models.TagQuoteStatsModel attribute)
change_pct1 (zvt.domain.quotes.currency.currency_1d_kdata.Currency1dKdata attribute)
(zvt.domain.quotes.CurrencyKdataCommon attribute)
(zvt.domain.quotes.future.future_1d_kdata.Future1dKdata attribute)
(zvt.domain.quotes.FutureKdataCommon attribute)
change_ratio (zvt.domain.actor.stock_actor.StockActorSummary attribute)
(zvt.domain.misc.holder.TopTenHolder attribute)
(zvt.domain.misc.holder.TopTenTradableHolder attribute)
ChartType (class in zvt.contract.drawer)
check_missed_concept() (in module zvt.tag.tag_utils)
check_missed_industry() (in module zvt.tag.tag_utils)
check_trading_plan() (in module zvt.trading.trading_service)
china_stock_code_to_id() (in module zvt.api.utils)
ChinaETFDayKdataRecorder (class in zvt.recorders.sina.quotes.sina_etf_kdata_recorder)
ChinaETFListSpider (class in zvt.recorders.exchange.exchange_etf_meta_recorder)
ChinaIndexDayKdataRecorder (class in zvt.recorders.sina.quotes.sina_index_kdata_recorder)
ChinaStockBalanceSheetRecorder (class in zvt.recorders.eastmoney.finance.eastmoney_balance_sheet_recorde
ChinaStockCashFlowRecorder (class in zvt.recorders.eastmoney.finance.eastmoney_cash_flow_recorder)
ChinaStockFinanceFactorRecorder (class in zvt.recorders.eastmoney.finance.eastmoney_finance_factor_recorc
ChinaStockIncomeStatementRecorder (class in
zvt.recorders.eastmoney.finance.eastmoney_income_statement_recorder)
CHINESENEWYEAR (in module zvt.zhdate.constants)
CHINESEYEARCODE (in module zvt.zhdate.constants)
chrome_copy_header_to_dict() (in module zvt.utils.utils)
circulating_cap (zvt.domain.fundamental.valuation.StockValuation attribute)
circulating_market_cap (zvt.domain.fundamental.valuation.StockValuation attribute)
clean_groups() (in module zvt.informer.inform_utils)
clear_state_data() (zvt.contract.base_service.EntityStateService method)

(zvt.contract.base_service.OneStateService method)
(zvt.contract.base_service.StatefulService method)
(zvt.contract.factor.Factor method)
(zvt.contract.factor.ScoreFactor method)
(zvt.contract.recorder.EntityEventRecorder method)
(zvt.contract.recorder.FixedCycleDataRecorder method)
(zvt.contract.recorder.Recorder method)
(zvt.contract.recorder.TimeSeriesDataRecorder method)
(zvt.contract.recorder.TimestampsDataRecorder method)
(zvt.factors.fundamental.finance_factor.FinanceBaseFactor method)
(zvt.factors.fundamental.finance_factor.GoodCompanyFactor method)
(zvt.factors.ma.ma_factor.CrossMaFactor method)
(zvt.factors.ma.ma_factor.CrossMaVolumeFactor method)
(zvt.factors.ma.ma_factor.MaFactor method)
(zvt.factors.ma.ma_factor.VolumeUpMaFactor method)
(zvt.factors.ma.ma_stats_factor.MaStatsFactor method)
(zvt.factors.ma.ma_stats_factor.TFactor method)
(zvt.factors.ma.top_bottom_factor.TopBottomFactor method)
(zvt.factors.macd.macd_factor.BullFactor method)
(zvt.factors.macd.macd_factor.GoldCrossFactor method)
(zvt.factors.macd.macd_factor.KeepBullFactor method)
(zvt.factors.macd.macd_factor.LiveOrDeadFactor method)
(zvt.factors.macd.macd_factor.MacdFactor method)
(zvt.factors.technical_factor.TechnicalFactor method)
(zvt.factors.z.z_factor.ZFactor method)
(zvt.recorders.eastmoney.common.EastmoneyMoreDataRecorder method)
(zvt.recorders.eastmoney.common.EastmoneyPageabeDataRecorder method)
(zvt.recorders.eastmoney.common.EastmoneyTimestampsDataRecorder method)
(zvt.recorders.eastmoney.dividend_financing.eastmoney_dividend_detail_recorder.DividendDetailRecord
method)
(zvt.recorders.eastmoney.dividend_financing.eastmoney_dividend_financing_recorder.DividendFinancing
method)
(zvt.recorders.eastmoney.dividend_financing.eastmoney_rights_issue_detail_recorder.RightsIssueDetailR
method)
(zvt.recorders.eastmoney.dividend_financing.eastmoney_spo_detail_recorder.SPODetailRecorder method
(zvt.recorders.eastmoney.finance.base_china_stock_finance_recorder.BaseChinaStockFinanceRecorder m
(zvt.recorders.eastmoney.finance.eastmoney_balance_sheet_recorder.ChinaStockBalanceSheetRecorder n
(zvt.recorders.eastmoney.finance.eastmoney_cash_flow_recorder.ChinaStockCashFlowRecorder method)
(zvt.recorders.eastmoney.finance.eastmoney_finance_factor_recorder.ChinaStockFinanceFactorRecorder_
(zvt.recorders.eastmoney.finance.eastmoney_income_statement_recorder.ChinaStockIncomeStatementRe
method)
(zvt.recorders.eastmoney.holder.eastmoney_stock_actor_recorder.EastmoneyActorRecorder method)
(zvt.recorders.eastmoney.holder.eastmoney_top_ten_holder_recorder.TopTenHolderRecorder method)
(zvt.recorders.eastmoney.holder.eastmoney_top_ten_tradable_holder_recorder.TopTenTradableHolderRec
method)
(zvt.recorders.eastmoney.meta.eastmoney_block_meta_recorder.EastmoneyBlockRecorder method)
(zvt.recorders.eastmoney.meta.eastmoney_block_meta_recorder.EastmoneyBlockStockRecorder method)
(zvt.recorders.eastmoney.meta.eastmoney_stock_meta_recorder.EastmoneyStockDetailRecorder method)
(zvt.recorders.eastmoney.meta.eastmoney_stock_meta_recorder.EastmoneyStockRecorder method)
(zvt.recorders.eastmoney.trading.eastmoney_holder_trading_recorder.HolderTradingRecorder method)
(zvt.recorders.eastmoney.trading.eastmoney_manager_trading_recorder.ManagerTradingRecorder method
(zvt.recorders.em.actor.em_stock_actor_summary_recorder.EMStockActorSummaryRecorder method)
(zvt.recorders.em.actor.em_stock_ii_recorder.EMStockIIRecorder method)
(zvt.recorders.em.actor.em_stock_top_ten_free_recorder.EMStockTopTenFreeRecorder method)
(zvt.recorders.em.actor.em_stock_top_ten_recorder.EMStockTopTenRecorder method)
(zvt.recorders.em.macro.em_treasury_yield_recorder.EMTreasuryYieldRecorder method)

(zvt.recorders.em.meta.em_block_meta_recorder.EMBlockRecorder method)
(zvt.recorders.em.meta.em_block_meta_recorder.EMBlockStockRecorder method)
(zvt.recorders.em.meta.em_currency_meta_recorder.EMCurrencyRecorder method)
(zvt.recorders.em.meta.em_future_meta_recorder.EMFutureRecorder method)
(zvt.recorders.em.meta.em_index_meta_recorder.EMIndexRecorder method)
(zvt.recorders.em.meta.em_indexus_meta_recorder.EMIndexusRecorder method)
(zvt.recorders.em.meta.em_stock_meta_recorder.EMStockRecorder method)
(zvt.recorders.em.meta.em_stockhk_meta_recorder.EMStockhkRecorder method)
(zvt.recorders.em.meta.em_stockus_meta_recorder.EMStockusRecorder method)
(zvt.recorders.em.misc.em_stock_news_recorder.EMStockNewsRecorder method)
(zvt.recorders.em.quotes.em_kdata_recorder.BaseEMStockKdataRecorder method)
(zvt.recorders.em.quotes.em_kdata_recorder.EMBlockKdataRecorder method)
(zvt.recorders.em.quotes.em_kdata_recorder.EMCurrencyKdataRecorder method)
(zvt.recorders.em.quotes.em_kdata_recorder.EMFutureKdataRecorder method)
(zvt.recorders.em.quotes.em_kdata_recorder.EMIndexKdataRecorder method)
(zvt.recorders.em.quotes.em_kdata_recorder.EMIndexusKdataRecorder method)
(zvt.recorders.em.quotes.em_kdata_recorder.EMStockhkKdataRecorder method)
(zvt.recorders.em.quotes.em_kdata_recorder.EMStockKdataRecorder method)
(zvt.recorders.em.quotes.em_kdata_recorder.EMStockusKdataRecorder method)
(zvt.recorders.em.trading.em_dragon_and_tiger_recorder.EMDragonAndTigerRecorder method)
(zvt.recorders.exchange.exchange_etf_meta_recorder.ChinaETFListSpider method)
(zvt.recorders.exchange.exchange_index_recorder.ExchangeIndexRecorder method)
(zvt.recorders.exchange.exchange_index_stock_recorder.ExchangeIndexStockRecorder method)
(zvt.recorders.exchange.exchange_stock_meta_recorder.ExchangeStockMetaRecorder method)
(zvt.recorders.exchange.exchange_stock_summary_recorder.ExchangeStockSummaryRecorder method)
(zvt.recorders.joinquant.fundamental.jq_etf_valuation_recorder.JqChinaEtfValuationRecorder method)
(zvt.recorders.joinquant.fundamental.jq_margin_trading_recorder.MarginTradingRecorder method)
(zvt.recorders.joinquant.fundamental.jq_stock_valuation_recorder.JqChinaStockValuationRecorder metho
(zvt.recorders.joinquant.meta.jq_fund_meta_recorder.JqChinaFundRecorder method)
(zvt.recorders.joinquant.meta.jq_fund_meta_recorder.JqChinaFundStockRecorder method)
(zvt.recorders.joinquant.meta.jq_stock_meta_recorder.BaseJqChinaMetaRecorder method)
(zvt.recorders.joinquant.meta.jq_stock_meta_recorder.JqChinaEtfRecorder method)
(zvt.recorders.joinquant.meta.jq_stock_meta_recorder.JqChinaStockEtfPortfolioRecorder method)
(zvt.recorders.joinquant.meta.jq_stock_meta_recorder.JqChinaStockRecorder method)
(zvt.recorders.joinquant.meta.jq_trade_day_recorder.StockTradeDayRecorder method)
(zvt.recorders.joinquant.misc.jq_hk_holder_recorder.JoinquantHkHolderRecorder method)
(zvt.recorders.joinquant.misc.jq_index_money_flow_recorder.JoinquantIndexMoneyFlowRecorder metho
(zvt.recorders.joinquant.misc.jq_stock_money_flow_recorder.JoinquantStockMoneyFlowRecorder metho
(zvt.recorders.joinquant.overall.jq_cross_market_recorder.CrossMarketSummaryRecorder method)
(zvt.recorders.joinquant.overall.jq_margin_trading_recorder.MarginTradingSummaryRecorder method)
(zvt.recorders.joinquant.overall.jq_stock_summary_recorder.StockSummaryRecorder method)
(zvt.recorders.joinquant.quotes.jq_index_kdata_recorder.JqChinaIndexKdataRecorder method)
(zvt.recorders.joinquant.quotes.jq_stock_kdata_recorder.JqChinaStockKdataRecorder method)
(zvt.recorders.sina.meta.sina_block_recorder.SinaBlockRecorder method)
(zvt.recorders.sina.meta.sina_block_recorder.SinaChinaBlockStockRecorder method)
(zvt.recorders.sina.money_flow.sina_block_money_flow_recorder.SinaBlockMoneyFlowRecorder metho
(zvt.recorders.sina.money_flow.sina_stock_money_flow_recorder.SinaStockMoneyFlowRecorder metho
(zvt.recorders.sina.quotes.sina_etf_kdata_recorder.ChinaETFDayKdataRecorder method)
(zvt.recorders.sina.quotes.sina_index_kdata_recorder.ChinaIndexDayKdataRecorder method)
(zvt.recorders.wb.wb_country_recorder.WBCountryRecorder method)
(zvt.recorders.wb.wb_economy_recorder.WBEconomyRecorder method)
(zvt.tag.tag_service.StockAutoTagger method)
(zvt.tag.tagger.StockTagger method)
(zvt.tag.tagger.Tagger method)
clear_trader() (in module zvt.trader.trader_info_api)
close (zvt.domain.misc.money_flow.BlockMoneyFlow attribute)

(zvt.domain.misc.money_flow.StockMoneyFlow attribute)
(zvt.domain.quotes.block.block_1d_kdata.Block1dKdata attribute)
(zvt.domain.quotes.block.block_1mon_kdata.Block1monKdata attribute)
(zvt.domain.quotes.block.block_1wk_kdata.Block1wkKdata attribute)
(zvt.domain.quotes.currency.currency_1d_kdata.Currency1dKdata attribute)
(zvt.domain.quotes.etf.etf_1d_kdata.Etf1dKdata attribute)
(zvt.domain.quotes.future.future_1d_kdata.Future1dKdata attribute)
(zvt.domain.quotes.index.index_1d_kdata.Index1dKdata attribute)
(zvt.domain.quotes.index.index_1wk_kdata.Index1wkKdata attribute)
(zvt.domain.quotes.indexus.indexus_1d_kdata.Indexus1dKdata attribute)
(zvt.domain.quotes.stock.stock_15m_hfq_kdata.Stock15mHfqKdata attribute)
(zvt.domain.quotes.stock.stock_15m_kdata.Stock15mKdata attribute)
(zvt.domain.quotes.stock.stock_1d_hfq_kdata.Stock1dHfqKdata attribute)
(zvt.domain.quotes.stock.stock_1d_kdata.Stock1dKdata attribute)
(zvt.domain.quotes.stock.stock_1h_hfq_kdata.Stock1hHfqKdata attribute)
(zvt.domain.quotes.stock.stock_1h_kdata.Stock1hKdata attribute)
(zvt.domain.quotes.stock.stock_1m_hfq_kdata.Stock1mHfqKdata attribute)
(zvt.domain.quotes.stock.stock_1m_kdata.Stock1mKdata attribute)
(zvt.domain.quotes.stock.stock_1mon_hfq_kdata.Stock1monHfqKdata attribute)
(zvt.domain.quotes.stock.stock_1mon_kdata.Stock1monKdata attribute)
(zvt.domain.quotes.stock.stock_1wk_hfq_kdata.Stock1wkHfqKdata attribute)
(zvt.domain.quotes.stock.stock_1wk_kdata.Stock1wkKdata attribute)
(zvt.domain.quotes.stock.stock_30m_hfq_kdata.Stock30mHfqKdata attribute)
(zvt.domain.quotes.stock.stock_30m_kdata.Stock30mKdata attribute)
(zvt.domain.quotes.stock.stock_4h_hfq_kdata.Stock4hHfqKdata attribute)
(zvt.domain.quotes.stock.stock_4h_kdata.Stock4hKdata attribute)
(zvt.domain.quotes.stock.stock_5m_hfq_kdata.Stock5mHfqKdata attribute)
(zvt.domain.quotes.stock.stock_5m_kdata.Stock5mKdata attribute)
(zvt.domain.quotes.stockhk.stockhk_1d_hfq_kdata.Stockhk1dHfqKdata attribute)
(zvt.domain.quotes.stockhk.stockhk_1d_kdata.Stockhk1dKdata attribute)
(zvt.domain.quotes.stockus.stockus_1d_hfq_kdata.Stockus1dHfqKdata attribute)
(zvt.domain.quotes.stockus.stockus_1d_kdata.Stockus1dKdata attribute)
(zvt.domain.quotes.TickCommon attribute)
(zvt.factors.ma.domain.stock_1d_ma_stats_factor.Stock1dMaStatsFactor attribute)
(zvt.factors.z.domain.stock_1d_z_factor.Stock1dZFactor attribute)
(zvt.factors.z.domain.stock_1wk_z_factor.Stock1wkZFactor attribute)

D

data_exist() (in module zvt.contract.api)
data_schema
(zvt.recorders.eastmoney.dividend_financing.eastmoney_dividend_detail_recorder.DividendDetailRecorder att
(zvt.recorders.eastmoney.dividend_financing.eastmoney_dividend_financing_recorder.DividendFinancing
attribute)
(zvt.recorders.eastmoney.dividend_financing.eastmoney_rights_issue_detail_recorder.RightsIssueDetailR
attribute)
(zvt.recorders.eastmoney.dividend_financing.eastmoney_spo_detail_recorder.SPODetailRecorder attribut
(zvt.recorders.eastmoney.finance.eastmoney_balance_sheet_recorder.ChinaStockBalanceSheetRecorder a
(zvt.recorders.eastmoney.finance.eastmoney_cash_flow_recorder.ChinaStockCashFlowRecorder attribute
(zvt.recorders.eastmoney.finance.eastmoney_finance_factor_recorder.ChinaStockFinanceFactorRecorder_
(zvt.recorders.eastmoney.finance.eastmoney_income_statement_recorder.ChinaStockIncomeStatementRe
attribute)
(zvt.recorders.eastmoney.holder.eastmoney_stock_actor_recorder.EastmoneyActorRecorder attribute)
(zvt.recorders.eastmoney.holder.eastmoney_top_ten_holder_recorder.TopTenHolderRecorder attribute)

(zvt.recorders.eastmoney.holder.eastmoney_top_ten_tradable_holder_recorder.TopTenTradableHolderRec
attribute)
(zvt.recorders.eastmoney.meta.eastmoney_block_meta_recorder.EastmoneyBlockRecorder attribute)
(zvt.recorders.eastmoney.meta.eastmoney_block_meta_recorder.EastmoneyBlockStockRecorder attribute
(zvt.recorders.eastmoney.meta.eastmoney_stock_meta_recorder.EastmoneyStockDetailRecorder attribute
(zvt.recorders.eastmoney.meta.eastmoney_stock_meta_recorder.EastmoneyStockRecorder attribute)
(zvt.recorders.eastmoney.trading.eastmoney_holder_trading_recorder.HolderTradingRecorder attribute)
(zvt.recorders.eastmoney.trading.eastmoney_manager_trading_recorder.ManagerTradingRecorder attribut
(zvt.recorders.em.actor.em_stock_actor_summary_recorder.EMStockActorSummaryRecorder attribute)
(zvt.recorders.em.actor.em_stock_ii_recorder.EMStockIIRecorder attribute)
(zvt.recorders.em.actor.em_stock_top_ten_free_recorder.EMStockTopTenFreeRecorder attribute)
(zvt.recorders.em.actor.em_stock_top_ten_recorder.EMStockTopTenRecorder attribute)
(zvt.recorders.em.macro.em_treasury_yield_recorder.EMTreasuryYieldRecorder attribute)
(zvt.recorders.em.meta.em_block_meta_recorder.EMBlockRecorder attribute)
(zvt.recorders.em.meta.em_block_meta_recorder.EMBlockStockRecorder attribute)
(zvt.recorders.em.meta.em_currency_meta_recorder.EMCurrencyRecorder attribute)
(zvt.recorders.em.meta.em_future_meta_recorder.EMFutureRecorder attribute)
(zvt.recorders.em.meta.em_index_meta_recorder.EMIndexRecorder attribute)
(zvt.recorders.em.meta.em_indexus_meta_recorder.EMIndexusRecorder attribute)
(zvt.recorders.em.meta.em_stock_meta_recorder.EMStockRecorder attribute)
(zvt.recorders.em.meta.em_stockhk_meta_recorder.EMStockhkRecorder attribute)
(zvt.recorders.em.meta.em_stockus_meta_recorder.EMStockusRecorder attribute)
(zvt.recorders.em.misc.em_stock_news_recorder.EMStockNewsRecorder attribute)
(zvt.recorders.em.quotes.em_kdata_recorder.EMBlockKdataRecorder attribute)
(zvt.recorders.em.quotes.em_kdata_recorder.EMCurrencyKdataRecorder attribute)
(zvt.recorders.em.quotes.em_kdata_recorder.EMFutureKdataRecorder attribute)
(zvt.recorders.em.quotes.em_kdata_recorder.EMIndexKdataRecorder attribute)
(zvt.recorders.em.quotes.em_kdata_recorder.EMIndexusKdataRecorder attribute)
(zvt.recorders.em.quotes.em_kdata_recorder.EMStockhkKdataRecorder attribute)
(zvt.recorders.em.quotes.em_kdata_recorder.EMStockKdataRecorder attribute)
(zvt.recorders.em.quotes.em_kdata_recorder.EMStockusKdataRecorder attribute)
(zvt.recorders.em.trading.em_dragon_and_tiger_recorder.EMDragonAndTigerRecorder attribute)
(zvt.recorders.exchange.exchange_etf_meta_recorder.ChinaETFListSpider attribute)
(zvt.recorders.exchange.exchange_index_recorder.ExchangeIndexRecorder attribute)
(zvt.recorders.exchange.exchange_index_stock_recorder.ExchangeIndexStockRecorder attribute)
(zvt.recorders.exchange.exchange_stock_meta_recorder.ExchangeStockMetaRecorder attribute)
(zvt.recorders.exchange.exchange_stock_summary_recorder.ExchangeStockSummaryRecorder attribute)
(zvt.recorders.joinquant.fundamental.jq_etf_valuation_recorder.JqChinaEtfValuationRecorder attribute)
(zvt.recorders.joinquant.fundamental.jq_margin_trading_recorder.MarginTradingRecorder attribute)
(zvt.recorders.joinquant.fundamental.jq_stock_valuation_recorder.JqChinaStockValuationRecorder attribu
(zvt.recorders.joinquant.meta.jq_fund_meta_recorder.JqChinaFundRecorder attribute)
(zvt.recorders.joinquant.meta.jq_fund_meta_recorder.JqChinaFundStockRecorder attribute)
(zvt.recorders.joinquant.meta.jq_stock_meta_recorder.JqChinaEtfRecorder attribute)
(zvt.recorders.joinquant.meta.jq_stock_meta_recorder.JqChinaStockEtfPortfolioRecorder attribute)
(zvt.recorders.joinquant.meta.jq_stock_meta_recorder.JqChinaStockRecorder attribute)
(zvt.recorders.joinquant.meta.jq_trade_day_recorder.StockTradeDayRecorder attribute)
(zvt.recorders.joinquant.misc.jq_hk_holder_recorder.JoinquantHkHolderRecorder attribute)
(zvt.recorders.joinquant.misc.jq_index_money_flow_recorder.JoinquantIndexMoneyFlowRecorder attribu
(zvt.recorders.joinquant.misc.jq_stock_money_flow_recorder.JoinquantStockMoneyFlowRecorder attribu
(zvt.recorders.joinquant.overall.jq_cross_market_recorder.CrossMarketSummaryRecorder attribute)
(zvt.recorders.joinquant.overall.jq_margin_trading_recorder.MarginTradingSummaryRecorder attribute)
(zvt.recorders.joinquant.overall.jq_stock_summary_recorder.StockSummaryRecorder attribute)
(zvt.recorders.joinquant.quotes.jq_index_kdata_recorder.JqChinaIndexKdataRecorder attribute)
(zvt.recorders.joinquant.quotes.jq_stock_kdata_recorder.JqChinaStockKdataRecorder attribute)
(zvt.recorders.sina.meta.sina_block_recorder.SinaBlockRecorder attribute)
(zvt.recorders.sina.meta.sina_block_recorder.SinaChinaBlockStockRecorder attribute)

(zvt.recorders.sina.money_flow.sina_block_money_flow_recorder.SinaBlockMoneyFlowRecorder attribu
(zvt.recorders.sina.money_flow.sina_stock_money_flow_recorder.SinaStockMoneyFlowRecorder attribut
(zvt.recorders.sina.quotes.sina_etf_kdata_recorder.ChinaETFDayKdataRecorder attribute)
(zvt.recorders.sina.quotes.sina_index_kdata_recorder.ChinaIndexDayKdataRecorder attribute)
(zvt.recorders.wb.wb_country_recorder.WBCountryRecorder attribute)
(zvt.recorders.wb.wb_economy_recorder.WBEconomyRecorder attribute)
(zvt.tag.tag_service.StockAutoTagger attribute)
(zvt.tag.tagger.StockTagger attribute)
DataListener (class in zvt.contract.reader)
DataReader (class in zvt.contract.reader)
date_and_time() (in module zvt.utils.time_utils)
date_of_establishment (zvt.domain.meta.stock_meta.StockDetail attribute)
date_time_by_interval() (in module zvt.utils.time_utils)
day_offset_today() (in module zvt.utils.time_utils)
db_engine_map (zvt.contract.context.Registry attribute)
db_session_map (zvt.contract.context.Registry attribute)
dbname_map_base (zvt.contract.context.Registry attribute)
dbname_map_schemas (zvt.contract.context.Registry attribute)
DBSession() (in module zvt.contract.api)
dce (zvt.contract.Exchange attribute)
debt_asset_ratio (zvt.domain.fundamental.finance.FinanceFactor attribute)
decode() (zvt.zhdate.zhdate.ZhDate static method)
decode_entity_id() (in module zvt.contract.api)
decode_fenxing() (in module zvt.factors.shape)
decode_rect() (in module zvt.factors.shape)
decode_state() (zvt.contract.base_service.EntityStateService method)
(zvt.contract.base_service.OneStateService method)
(zvt.contract.base_service.StatefulService method)
(zvt.contract.factor.Factor method)
(zvt.contract.factor.ScoreFactor method)
(zvt.contract.recorder.EntityEventRecorder method)
(zvt.contract.recorder.FixedCycleDataRecorder method)
(zvt.contract.recorder.Recorder method)
(zvt.contract.recorder.TimeSeriesDataRecorder method)
(zvt.contract.recorder.TimestampsDataRecorder method)
(zvt.factors.fundamental.finance_factor.FinanceBaseFactor method)
(zvt.factors.fundamental.finance_factor.GoodCompanyFactor method)
(zvt.factors.ma.ma_factor.CrossMaFactor method)
(zvt.factors.ma.ma_factor.CrossMaVolumeFactor method)
(zvt.factors.ma.ma_factor.MaFactor method)
(zvt.factors.ma.ma_factor.VolumeUpMaFactor method)
(zvt.factors.ma.ma_stats_factor.MaStatsFactor method)
(zvt.factors.ma.ma_stats_factor.TFactor method)
(zvt.factors.ma.top_bottom_factor.TopBottomFactor method)
(zvt.factors.macd.macd_factor.BullFactor method)
(zvt.factors.macd.macd_factor.GoldCrossFactor method)
(zvt.factors.macd.macd_factor.KeepBullFactor method)
(zvt.factors.macd.macd_factor.LiveOrDeadFactor method)
(zvt.factors.macd.macd_factor.MacdFactor method)
(zvt.factors.technical_factor.TechnicalFactor method)
(zvt.factors.z.z_factor.ZFactor method)
(zvt.recorders.eastmoney.common.EastmoneyMoreDataRecorder method)
(zvt.recorders.eastmoney.common.EastmoneyPageabeDataRecorder method)
(zvt.recorders.eastmoney.common.EastmoneyTimestampsDataRecorder method)
(zvt.recorders.eastmoney.dividend_financing.eastmoney_dividend_detail_recorder.DividendDetailRecord
method)

(zvt.recorders.eastmoney.dividend_financing.eastmoney_dividend_financing_recorder.DividendFinancing
method)
(zvt.recorders.eastmoney.dividend_financing.eastmoney_rights_issue_detail_recorder.RightsIssueDetailR
method)
(zvt.recorders.eastmoney.dividend_financing.eastmoney_spo_detail_recorder.SPODetailRecorder method
(zvt.recorders.eastmoney.finance.base_china_stock_finance_recorder.BaseChinaStockFinanceRecorder m
(zvt.recorders.eastmoney.finance.eastmoney_balance_sheet_recorder.ChinaStockBalanceSheetRecorder n
(zvt.recorders.eastmoney.finance.eastmoney_cash_flow_recorder.ChinaStockCashFlowRecorder method)
(zvt.recorders.eastmoney.finance.eastmoney_finance_factor_recorder.ChinaStockFinanceFactorRecorder
(zvt.recorders.eastmoney.finance.eastmoney_income_statement_recorder.ChinaStockIncomeStatementRe
method)
(zvt.recorders.eastmoney.holder.eastmoney_stock_actor_recorder.EastmoneyActorRecorder method)
(zvt.recorders.eastmoney.holder.eastmoney_top_ten_holder_recorder.TopTenHolderRecorder method)
(zvt.recorders.eastmoney.holder.eastmoney_top_ten_tradable_holder_recorder.TopTenTradableHolderRec
method)
(zvt.recorders.eastmoney.meta.eastmoney_block_meta_recorder.EastmoneyBlockRecorder method)
(zvt.recorders.eastmoney.meta.eastmoney_block_meta_recorder.EastmoneyBlockStockRecorder method)
(zvt.recorders.eastmoney.meta.eastmoney_stock_meta_recorder.EastmoneyStockDetailRecorder method)
(zvt.recorders.eastmoney.meta.eastmoney_stock_meta_recorder.EastmoneyStockRecorder method)
(zvt.recorders.eastmoney.trading.eastmoney_holder_trading_recorder.HolderTradingRecorder method)
(zvt.recorders.eastmoney.trading.eastmoney_manager_trading_recorder.ManagerTradingRecorder method
(zvt.recorders.em.actor.em_stock_actor_summary_recorder.EMStockActorSummaryRecorder method)
(zvt.recorders.em.actor.em_stock_ii_recorder.EMStockIIRecorder method)
(zvt.recorders.em.actor.em_stock_top_ten_free_recorder.EMStockTopTenFreeRecorder method)
(zvt.recorders.em.actor.em_stock_top_ten_recorder.EMStockTopTenRecorder method)
(zvt.recorders.em.macro.em_treasury_yield_recorder.EMTreasuryYieldRecorder method)
(zvt.recorders.em.meta.em_block_meta_recorder.EMBlockRecorder method)
(zvt.recorders.em.meta.em_block_meta_recorder.EMBlockStockRecorder method)
(zvt.recorders.em.meta.em_currency_meta_recorder.EMCurrencyRecorder method)
(zvt.recorders.em.meta.em_future_meta_recorder.EMFutureRecorder method)
(zvt.recorders.em.meta.em_index_meta_recorder.EMIndexRecorder method)
(zvt.recorders.em.meta.em_indexus_meta_recorder.EMIndexusRecorder method)
(zvt.recorders.em.meta.em_stock_meta_recorder.EMStockRecorder method)
(zvt.recorders.em.meta.em_stockhk_meta_recorder.EMStockhkRecorder method)
(zvt.recorders.em.meta.em_stockus_meta_recorder.EMStockusRecorder method)
(zvt.recorders.em.misc.em_stock_news_recorder.EMStockNewsRecorder method)
(zvt.recorders.em.quotes.em_kdata_recorder.BaseEMStockKdataRecorder method)
(zvt.recorders.em.quotes.em_kdata_recorder.EMBlockKdataRecorder method)
(zvt.recorders.em.quotes.em_kdata_recorder.EMCurrencyKdataRecorder method)
(zvt.recorders.em.quotes.em_kdata_recorder.EMFutureKdataRecorder method)
(zvt.recorders.em.quotes.em_kdata_recorder.EMIndexKdataRecorder method)
(zvt.recorders.em.quotes.em_kdata_recorder.EMIndexusKdataRecorder method)
(zvt.recorders.em.quotes.em_kdata_recorder.EMStockhkKdataRecorder method)
(zvt.recorders.em.quotes.em_kdata_recorder.EMStockKdataRecorder method)
(zvt.recorders.em.quotes.em_kdata_recorder.EMStockusKdataRecorder method)
(zvt.recorders.em.trading.em_dragon_and_tiger_recorder.EMDragonAndTigerRecorder method)
(zvt.recorders.exchange.exchange_etf_meta_recorder.ChinaETFListSpider method)
(zvt.recorders.exchange.exchange_index_recorder.ExchangeIndexRecorder method)
(zvt.recorders.exchange.exchange_index_stock_recorder.ExchangeIndexStockRecorder method)
(zvt.recorders.exchange.exchange_stock_meta_recorder.ExchangeStockMetaRecorder method)
(zvt.recorders.exchange.exchange_stock_summary_recorder.ExchangeStockSummaryRecorder method)
(zvt.recorders.joinquant.fundamental.jq_etf_valuation_recorder.JqChinaEtfValuationRecorder method)
(zvt.recorders.joinquant.fundamental.jq_margin_trading_recorder.MarginTradingRecorder method)
(zvt.recorders.joinquant.fundamental.jq_stock_valuation_recorder.JqChinaStockValuationRecorder metho
(zvt.recorders.joinquant.meta.jq_fund_meta_recorder.JqChinaFundRecorder method)
(zvt.recorders.joinquant.meta.jq_fund_meta_recorder.JqChinaFundStockRecorder method)

(zvt.recorders.joinquant.meta.jq_stock_meta_recorder.BaseJqChinaMetaRecorder method)
(zvt.recorders.joinquant.meta.jq_stock_meta_recorder.JqChinaEtfRecorder method)
(zvt.recorders.joinquant.meta.jq_stock_meta_recorder.JqChinaStockEtfPortfolioRecorder method)
(zvt.recorders.joinquant.meta.jq_stock_meta_recorder.JqChinaStockRecorder method)
(zvt.recorders.joinquant.meta.jq_trade_day_recorder.StockTradeDayRecorder method)
(zvt.recorders.joinquant.misc.jq_hk_holder_recorder.JoinquantHkHolderRecorder method)
(zvt.recorders.joinquant.misc.jq_index_money_flow_recorder.JoinquantIndexMoneyFlowRecorder metho
(zvt.recorders.joinquant.misc.jq_stock_money_flow_recorder.JoinquantStockMoneyFlowRecorder metho
(zvt.recorders.joinquant.overall.jq_cross_market_recorder.CrossMarketSummaryRecorder method)
(zvt.recorders.joinquant.overall.jq_margin_trading_recorder.MarginTradingSummaryRecorder method)
(zvt.recorders.joinquant.overall.jq_stock_summary_recorder.StockSummaryRecorder method)
(zvt.recorders.joinquant.quotes.jq_index_kdata_recorder.JqChinaIndexKdataRecorder method)
(zvt.recorders.joinquant.quotes.jq_stock_kdata_recorder.JqChinaStockKdataRecorder method)
(zvt.recorders.sina.meta.sina_block_recorder.SinaBlockRecorder method)
(zvt.recorders.sina.meta.sina_block_recorder.SinaChinaBlockStockRecorder method)
(zvt.recorders.sina.money_flow.sina_block_money_flow_recorder.SinaBlockMoneyFlowRecorder metho
(zvt.recorders.sina.money_flow.sina_stock_money_flow_recorder.SinaStockMoneyFlowRecorder method
(zvt.recorders.sina.quotes.sina_etf_kdata_recorder.ChinaETFDayKdataRecorder method)
(zvt.recorders.sina.quotes.sina_index_kdata_recorder.ChinaIndexDayKdataRecorder method)
(zvt.recorders.wb.wb_country_recorder.WBCountryRecorder method)
(zvt.recorders.wb.wb_economy_recorder.WBEconomyRecorder method)
(zvt.tag.tag_service.StockAutoTagger method)
(zvt.tag.tagger.StockTagger method)
(zvt.tag.tagger.Tagger method)

E

EastmoneyActorRecorder (class in zvt.recorders.eastmoney.holder.eastmoney_stock_actor_recorder)
EastmoneyApiWrapper (class in zvt.recorders.eastmoney.common)
EastmoneyBlockRecorder (class in zvt.recorders.eastmoney.meta.eastmoney_block_meta_recorder)
EastmoneyBlockStockRecorder (class in zvt.recorders.eastmoney.meta.eastmoney_block_meta_recorder)
EastmoneyMoreDataRecorder (class in zvt.recorders.eastmoney.common)
EastmoneyPageabeDataRecorder (class in zvt.recorders.eastmoney.common)
EastmoneyStockDetailRecorder (class in zvt.recorders.eastmoney.meta.eastmoney_stock_meta_recorder)
EastmoneyStockRecorder (class in zvt.recorders.eastmoney.meta.eastmoney_stock_meta_recorder)
EastmoneyTimestampsDataRecorder (class in zvt.recorders.eastmoney.common)
Economy (class in zvt.domain.macro.macro)
em (zvt.domain.fundamental.finance.FinanceFactor attribute)
ema() (in module zvt.factors.algorithm)
EmailInformer (class in zvt.informer.informer)
EMBlockKdataRecorder (class in zvt.recorders.em.quotes.em_kdata_recorder)
EMBlockRecorder (class in zvt.recorders.em.meta.em_block_meta_recorder)
EMBlockStockRecorder (class in zvt.recorders.em.meta.em_block_meta_recorder)
EMCurrencyKdataRecorder (class in zvt.recorders.em.quotes.em_kdata_recorder)
EMCurrencyRecorder (class in zvt.recorders.em.meta.em_currency_meta_recorder)
EMDragonAndTigerRecorder (class in zvt.recorders.em.trading.em_dragon_and_tiger_recorder)
EMFutureKdataRecorder (class in zvt.recorders.em.quotes.em_kdata_recorder)
EMFutureRecorder (class in zvt.recorders.em.meta.em_future_meta_recorder)
EMIndexKdataRecorder (class in zvt.recorders.em.quotes.em_kdata_recorder)
EMIndexRecorder (class in zvt.recorders.em.meta.em_index_meta_recorder)
EMIndexusKdataRecorder (class in zvt.recorders.em.quotes.em_kdata_recorder)
EMIndexusRecorder (class in zvt.recorders.em.meta.em_indexus_meta_recorder)
Emotion (class in zvt.domain.emotion.emotion)
employee_benefits_payable (zvt.domain.fundamental.finance.BalanceSheet attribute)
EMStockActorSummaryRecorder (class in zvt.recorders.em.actor.em_stock_actor_summary_recorder)

EMStockhkKdataRecorder (class in zvt.recorders.em.quotes.em_kdata_recorder)
EMStockhkRecorder (class in zvt.recorders.em.meta.em_stockhk_meta_recorder)
EMStockIIRecorder (class in zvt.recorders.em.actor.em_stock_ii_recorder)
EMStockKdataRecorder (class in zvt.recorders.em.quotes.em_kdata_recorder)
EMStockNewsRecorder (class in zvt.recorders.em.misc.em_stock_news_recorder)
EMStockRecorder (class in zvt.recorders.em.meta.em_stock_meta_recorder)
EMStockTopTenFreeRecorder (class in zvt.recorders.em.actor.em_stock_top_ten_free_recorder)
EMStockTopTenRecorder (class in zvt.recorders.em.actor.em_stock_top_ten_recorder)
EMStockusKdataRecorder (class in zvt.recorders.em.quotes.em_kdata_recorder)
EMStockusRecorder (class in zvt.recorders.em.meta.em_stockus_meta_recorder)
EMTreasuryYieldRecorder (class in zvt.recorders.em.macro.em_treasury_yield_recorder)
encode() (zvt.factors.shape.FactorStateEncoder method)
encode_state() (zvt.contract.base_service.EntityStateService method)
(zvt.contract.base_service.OneStateService method)
(zvt.contract.base_service.StatefulService method)
(zvt.contract.factor.Factor method)
(zvt.contract.factor.ScoreFactor method)
(zvt.contract.recorder.EntityEventRecorder method)
(zvt.contract.recorder.FixedCycleDataRecorder method)
(zvt.contract.recorder.Recorder method)
(zvt.contract.recorder.TimeSeriesDataRecorder method)
(zvt.contract.recorder.TimestampsDataRecorder method)
(zvt.factors.fundamental.finance_factor.FinanceBaseFactor method)
(zvt.factors.fundamental.finance_factor.GoodCompanyFactor method)
(zvt.factors.ma.ma_factor.CrossMaFactor method)
(zvt.factors.ma.ma_factor.CrossMaVolumeFactor method)
(zvt.factors.ma.ma_factor.MaFactor method)
(zvt.factors.ma.ma_factor.VolumeUpMaFactor method)
(zvt.factors.ma.ma_stats_factor.MaStatsFactor method)
(zvt.factors.ma.ma_stats_factor.TFactor method)
(zvt.factors.ma.top_bottom_factor.TopBottomFactor method)
(zvt.factors.macd.macd_factor.BullFactor method)
(zvt.factors.macd.macd_factor.GoldCrossFactor method)
(zvt.factors.macd.macd_factor.KeepBullFactor method)
(zvt.factors.macd.macd_factor.LiveOrDeadFactor method)
(zvt.factors.macd.macd_factor.MacdFactor method)
(zvt.factors.technical_factor.TechnicalFactor method)
(zvt.factors.z.z_factor.ZFactor method)
(zvt.recorders.eastmoney.common.EastmoneyMoreDataRecorder method)
(zvt.recorders.eastmoney.common.EastmoneyPageabeDataRecorder method)
(zvt.recorders.eastmoney.common.EastmoneyTimestampsDataRecorder method)
(zvt.recorders.eastmoney.dividend_financing.eastmoney_dividend_detail_recorder.DividendDetailRecord
method)
(zvt.recorders.eastmoney.dividend_financing.eastmoney_dividend_financing_recorder.DividendFinancing
method)
(zvt.recorders.eastmoney.dividend_financing.eastmoney_rights_issue_detail_recorder.RightsIssueDetailR
method)
(zvt.recorders.eastmoney.dividend_financing.eastmoney_spo_detail_recorder.SPODetailRecorder method
(zvt.recorders.eastmoney.finance.base_china_stock_finance_recorder.BaseChinaStockFinanceRecorder m
(zvt.recorders.eastmoney.finance.eastmoney_balance_sheet_recorder.ChinaStockBalanceSheetRecorder n
(zvt.recorders.eastmoney.finance.eastmoney_cash_flow_recorder.ChinaStockCashFlowRecorder method)
(zvt.recorders.eastmoney.finance.eastmoney_finance_factor_recorder.ChinaStockFinanceFactorRecorder
(zvt.recorders.eastmoney.finance.eastmoney_income_statement_recorder.ChinaStockIncomeStatementRe
method)
(zvt.recorders.eastmoney.holder.eastmoney_stock_actor_recorder.EastmoneyActorRecorder method)
(zvt.recorders.eastmoney.holder.eastmoney_top_ten_holder_recorder.TopTenHolderRecorder method)

(zvt.recorders.eastmoney.holder.eastmoney_top_ten_tradable_holder_recorder.TopTenTradableHolderRec
method)
(zvt.recorders.eastmoney.meta.eastmoney_block_meta_recorder.EastmoneyBlockRecorder method)
(zvt.recorders.eastmoney.meta.eastmoney_block_meta_recorder.EastmoneyBlockStockRecorder method)
(zvt.recorders.eastmoney.meta.eastmoney_stock_meta_recorder.EastmoneyStockDetailRecorder method)
(zvt.recorders.eastmoney.meta.eastmoney_stock_meta_recorder.EastmoneyStockRecorder method)
(zvt.recorders.eastmoney.trading.eastmoney_holder_trading_recorder.HolderTradingRecorder method)
(zvt.recorders.eastmoney.trading.eastmoney_manager_trading_recorder.ManagerTradingRecorder method
(zvt.recorders.em.actor.em_stock_actor_summary_recorder.EMStockActorSummaryRecorder method)
(zvt.recorders.em.actor.em_stock_ii_recorder.EMStockIIRecorder method)
(zvt.recorders.em.actor.em_stock_top_ten_free_recorder.EMStockTopTenFreeRecorder method)
(zvt.recorders.em.actor.em_stock_top_ten_recorder.EMStockTopTenRecorder method)
(zvt.recorders.em.macro.em_treasury_yield_recorder.EMTreasuryYieldRecorder method)
(zvt.recorders.em.meta.em_block_meta_recorder.EMBlockRecorder method)
(zvt.recorders.em.meta.em_block_meta_recorder.EMBlockStockRecorder method)
(zvt.recorders.em.meta.em_currency_meta_recorder.EMCurrencyRecorder method)
(zvt.recorders.em.meta.em_future_meta_recorder.EMFutureRecorder method)
(zvt.recorders.em.meta.em_index_meta_recorder.EMIndexRecorder method)
(zvt.recorders.em.meta.em_indexus_meta_recorder.EMIndexusRecorder method)
(zvt.recorders.em.meta.em_stock_meta_recorder.EMStockRecorder method)
(zvt.recorders.em.meta.em_stockhk_meta_recorder.EMStockhkRecorder method)
(zvt.recorders.em.meta.em_stockus_meta_recorder.EMStockusRecorder method)
(zvt.recorders.em.misc.em_stock_news_recorder.EMStockNewsRecorder method)
(zvt.recorders.em.quotes.em_kdata_recorder.BaseEMStockKdataRecorder method)
(zvt.recorders.em.quotes.em_kdata_recorder.EMBlockKdataRecorder method)
(zvt.recorders.em.quotes.em_kdata_recorder.EMCurrencyKdataRecorder method)
(zvt.recorders.em.quotes.em_kdata_recorder.EMFutureKdataRecorder method)
(zvt.recorders.em.quotes.em_kdata_recorder.EMIndexKdataRecorder method)
(zvt.recorders.em.quotes.em_kdata_recorder.EMIndexusKdataRecorder method)
(zvt.recorders.em.quotes.em_kdata_recorder.EMStockhkKdataRecorder method)
(zvt.recorders.em.quotes.em_kdata_recorder.EMStockKdataRecorder method)
(zvt.recorders.em.quotes.em_kdata_recorder.EMStockusKdataRecorder method)
(zvt.recorders.em.trading.em_dragon_and_tiger_recorder.EMDragonAndTigerRecorder method)
(zvt.recorders.exchange.exchange_etf_meta_recorder.ChinaETFListSpider method)
(zvt.recorders.exchange.exchange_index_recorder.ExchangeIndexRecorder method)
(zvt.recorders.exchange.exchange_index_stock_recorder.ExchangeIndexStockRecorder method)
(zvt.recorders.exchange.exchange_stock_meta_recorder.ExchangeStockMetaRecorder method)
(zvt.recorders.exchange.exchange_stock_summary_recorder.ExchangeStockSummaryRecorder method)
(zvt.recorders.joinquant.fundamental.jq_etf_valuation_recorder.JqChinaEtfValuationRecorder method)
(zvt.recorders.joinquant.fundamental.jq_margin_trading_recorder.MarginTradingRecorder method)
(zvt.recorders.joinquant.fundamental.jq_stock_valuation_recorder.JqChinaStockValuationRecorder metho
(zvt.recorders.joinquant.meta.jq_fund_meta_recorder.JqChinaFundRecorder method)
(zvt.recorders.joinquant.meta.jq_fund_meta_recorder.JqChinaFundStockRecorder method)
(zvt.recorders.joinquant.meta.jq_stock_meta_recorder.BaseJqChinaMetaRecorder method)
(zvt.recorders.joinquant.meta.jq_stock_meta_recorder.JqChinaEtfRecorder method)
(zvt.recorders.joinquant.meta.jq_stock_meta_recorder.JqChinaStockEtfPortfolioRecorder method)
(zvt.recorders.joinquant.meta.jq_stock_meta_recorder.JqChinaStockRecorder method)
(zvt.recorders.joinquant.meta.jq_trade_day_recorder.StockTradeDayRecorder method)
(zvt.recorders.joinquant.misc.jq_hk_holder_recorder.JoinquantHkHolderRecorder method)
(zvt.recorders.joinquant.misc.jq_index_money_flow_recorder.JoinquantIndexMoneyFlowRecorder metho
(zvt.recorders.joinquant.misc.jq_stock_money_flow_recorder.JoinquantStockMoneyFlowRecorder metho
(zvt.recorders.joinquant.overall.jq_cross_market_recorder.CrossMarketSummaryRecorder method)
(zvt.recorders.joinquant.overall.jq_margin_trading_recorder.MarginTradingSummaryRecorder method)
(zvt.recorders.joinquant.overall.jq_stock_summary_recorder.StockSummaryRecorder method)
(zvt.recorders.joinquant.quotes.jq_index_kdata_recorder.JqChinaIndexKdataRecorder method)
(zvt.recorders.joinquant.quotes.jq_stock_kdata_recorder.JqChinaStockKdataRecorder method)

(zvt.recorders.sina.meta.sina_block_recorder.SinaBlockRecorder method)
(zvt.recorders.sina.meta.sina_block_recorder.SinaChinaBlockStockRecorder method)
(zvt.recorders.sina.money_flow.sina_block_money_flow_recorder.SinaBlockMoneyFlowRecorder metho
(zvt.recorders.sina.money_flow.sina_stock_money_flow_recorder.SinaStockMoneyFlowRecorder metho
(zvt.recorders.sina.quotes.sina_etf_kdata_recorder.ChinaETFDayKdataRecorder method)
(zvt.recorders.sina.quotes.sina_index_kdata_recorder.ChinaIndexDayKdataRecorder method)
(zvt.recorders.wb.wb_country_recorder.WBCountryRecorder method)
(zvt.recorders.wb.wb_economy_recorder.WBEconomyRecorder method)
(zvt.tag.tag_service.StockAutoTagger method)
(zvt.tag.tagger.StockTagger method)
(zvt.tag.tagger.Tagger method)
end_date (zvt.contract.schema.ActorEntity attribute)
(zvt.contract.schema.Entity attribute)
(zvt.contract.schema.NormalEntityMixin attribute)
(zvt.contract.schema.Portfolio attribute)
(zvt.contract.schema.TradableEntity attribute)
(zvt.domain.actor.actor_meta.ActorMeta attribute)
(zvt.domain.meta.block_meta.Block attribute)
(zvt.domain.meta.country_meta.Country attribute)
(zvt.domain.meta.currency_meta.Currency attribute)
(zvt.domain.meta.etf_meta.Etf attribute)
(zvt.domain.meta.fund_meta.Fund attribute)
(zvt.domain.meta.future_meta.Future attribute)
(zvt.domain.meta.index_meta.Index attribute)
(zvt.domain.meta.indexus_meta.Indexus attribute)
(zvt.domain.meta.stock_meta.Stock attribute)
(zvt.domain.meta.stock_meta.StockDetail attribute)
(zvt.domain.meta.stockhk_meta.Stockhk attribute)
(zvt.domain.meta.stockus_meta.Stockus attribute)
Entity (class in zvt.contract.schema)
entity_id (zvt.contract.schema.ActorEntity attribute)
(zvt.contract.schema.ActorMeetTradable attribute)
(zvt.contract.schema.Entity attribute)
(zvt.contract.schema.Mixin attribute)
(zvt.contract.schema.NormalEntityMixin attribute)
(zvt.contract.schema.NormalMixin attribute)
(zvt.contract.schema.Portfolio attribute)
(zvt.contract.schema.PortfolioStock attribute)
(zvt.contract.schema.PortfolioStockHistory attribute)
(zvt.contract.schema.TradableEntity attribute)
(zvt.contract.schema.TradableMeetActor attribute)
(zvt.contract.zvt_info.FactorState attribute)
(zvt.contract.zvt_info.RecorderState attribute)
(zvt.contract.zvt_info.StateMixin attribute)
(zvt.contract.zvt_info.TaggerState attribute)
(zvt.domain.actor.actor_meta.ActorMeta attribute)
(zvt.domain.actor.stock_actor.StockActorSummary attribute)
(zvt.domain.actor.stock_actor.StockInstitutionalInvestorHolder attribute)
(zvt.domain.actor.stock_actor.StockTopTenFreeHolder attribute)
(zvt.domain.actor.stock_actor.StockTopTenHolder attribute)
(zvt.domain.emotion.emotion.Emotion attribute)
(zvt.domain.emotion.emotion.LimitDownInfo attribute)
(zvt.domain.emotion.emotion.LimitUpInfo attribute)
(zvt.domain.fundamental.dividend_financing.DividendDetail attribute)
(zvt.domain.fundamental.dividend_financing.DividendFinancing attribute)
(zvt.domain.fundamental.dividend_financing.RightsIssueDetail attribute)

(zvt.domain.fundamental.dividend_financing.SpoDetail attribute)
(zvt.domain.fundamental.finance.BalanceSheet attribute)
(zvt.domain.fundamental.finance.CashFlowStatement attribute)
(zvt.domain.fundamental.finance.FinanceFactor attribute)
(zvt.domain.fundamental.finance.IncomeStatement attribute)
(zvt.domain.fundamental.trading.BigDealTrading attribute)
(zvt.domain.fundamental.trading.DragonAndTiger attribute)
(zvt.domain.fundamental.trading.HolderTrading attribute)
(zvt.domain.fundamental.trading.ManagerTrading attribute)
(zvt.domain.fundamental.trading.MarginTrading attribute)
(zvt.domain.fundamental.valuation.EtfValuation attribute)
(zvt.domain.fundamental.valuation.StockValuation attribute)
(zvt.domain.macro.macro.Economy attribute)
(zvt.domain.macro.monetary.TreasuryYield attribute)
(zvt.domain.meta.block_meta.Block attribute)
(zvt.domain.meta.block_meta.BlockStock attribute)
(zvt.domain.meta.country_meta.Country attribute)
(zvt.domain.meta.currency_meta.Currency attribute)
(zvt.domain.meta.etf_meta.Etf attribute)
(zvt.domain.meta.etf_meta.EtfStock attribute)
(zvt.domain.meta.fund_meta.Fund attribute)
(zvt.domain.meta.fund_meta.FundStock attribute)
(zvt.domain.meta.future_meta.Future attribute)
(zvt.domain.meta.index_meta.Index attribute)
(zvt.domain.meta.index_meta.IndexStock attribute)
(zvt.domain.meta.indexus_meta.Indexus attribute)
(zvt.domain.meta.stock_meta.Stock attribute)
(zvt.domain.meta.stock_meta.StockDetail attribute)
(zvt.domain.meta.stockhk_meta.Stockhk attribute)
(zvt.domain.meta.stockus_meta.Stockus attribute)
(zvt.domain.misc.holder.HkHolder attribute)
(zvt.domain.misc.holder.InstitutionalInvestorHolder attribute)
(zvt.domain.misc.holder.TopTenHolder attribute)
(zvt.domain.misc.holder.TopTenTradableHolder attribute)
(zvt.domain.misc.money_flow.BlockMoneyFlow attribute)
(zvt.domain.misc.money_flow.IndexMoneyFlow attribute)
(zvt.domain.misc.money_flow.StockMoneyFlow attribute)
(zvt.domain.misc.overall.CrossMarketSummary attribute)
(zvt.domain.misc.overall.MarginTradingSummary attribute)
(zvt.domain.misc.overall.StockSummary attribute)
(zvt.domain.misc.stock_news.StockNews attribute)
(zvt.domain.quotes.block.block_1d_kdata.Block1dKdata attribute)
(zvt.domain.quotes.block.block_1mon_kdata.Block1monKdata attribute)
(zvt.domain.quotes.block.block_1wk_kdata.Block1wkKdata attribute)
(zvt.domain.quotes.BlockKdataCommon attribute)
(zvt.domain.quotes.currency.currency_1d_kdata.Currency1dKdata attribute)
(zvt.domain.quotes.CurrencyKdataCommon attribute)
(zvt.domain.quotes.etf.etf_1d_kdata.Etf1dKdata attribute)
(zvt.domain.quotes.EtfKdataCommon attribute)
(zvt.domain.quotes.future.future_1d_kdata.Future1dKdata attribute)
(zvt.domain.quotes.FutureKdataCommon attribute)
(zvt.domain.quotes.index.index_1d_kdata.Index1dKdata attribute)
(zvt.domain.quotes.index.index_1wk_kdata.Index1wkKdata attribute)
(zvt.domain.quotes.IndexKdataCommon attribute)
(zvt.domain.quotes.indexus.indexus_1d_kdata.Indexus1dKdata attribute)
(zvt.domain.quotes.IndexusKdataCommon attribute)

(zvt.domain.quotes.KdataCommon attribute)
(zvt.domain.quotes.stock.stock_15m_hfq_kdata.Stock15mHfqKdata attribute)
(zvt.domain.quotes.stock.stock_15m_kdata.Stock15mKdata attribute)
(zvt.domain.quotes.stock.stock_1d_hfq_kdata.Stock1dHfqKdata attribute)
(zvt.domain.quotes.stock.stock_1d_kdata.Stock1dKdata attribute)
(zvt.domain.quotes.stock.stock_1h_hfq_kdata.Stock1hHfqKdata attribute)
(zvt.domain.quotes.stock.stock_1h_kdata.Stock1hKdata attribute)
(zvt.domain.quotes.stock.stock_1m_hfq_kdata.Stock1mHfqKdata attribute)
(zvt.domain.quotes.stock.stock_1m_kdata.Stock1mKdata attribute)
(zvt.domain.quotes.stock.stock_1mon_hfq_kdata.Stock1monHfqKdata attribute)
(zvt.domain.quotes.stock.stock_1mon_kdata.Stock1monKdata attribute)
(zvt.domain.quotes.stock.stock_1wk_hfq_kdata.Stock1wkHfqKdata attribute)
(zvt.domain.quotes.stock.stock_1wk_kdata.Stock1wkKdata attribute)
(zvt.domain.quotes.stock.stock_30m_hfq_kdata.Stock30mHfqKdata attribute)
(zvt.domain.quotes.stock.stock_30m_kdata.Stock30mKdata attribute)
(zvt.domain.quotes.stock.stock_4h_hfq_kdata.Stock4hHfqKdata attribute)
(zvt.domain.quotes.stock.stock_4h_kdata.Stock4hKdata attribute)
(zvt.domain.quotes.stock.stock_5m_hfq_kdata.Stock5mHfqKdata attribute)
(zvt.domain.quotes.stock.stock_5m_kdata.Stock5mKdata attribute)
(zvt.domain.quotes.stock.stock_quote.StockQuote attribute)
(zvt.domain.quotes.stockhk.stockhk_1d_hfq_kdata.Stockhk1dHfqKdata attribute)
(zvt.domain.quotes.stockhk.stockhk_1d_kdata.Stockhk1dKdata attribute)
(zvt.domain.quotes.StockhkKdataCommon attribute)
(zvt.domain.quotes.StockKdataCommon attribute)
(zvt.domain.quotes.stockus.stockus_1d_hfq_kdata.Stockus1dHfqKdata attribute)
(zvt.domain.quotes.stockus.stockus_1d_kdata.Stockus1dKdata attribute)
(zvt.domain.quotes.StockusKdataCommon attribute)
(zvt.domain.quotes.TickCommon attribute)
(zvt.domain.quotes.trade_day.StockTradeDay attribute)
(zvt.factors.ma.domain.common.MaStatsFactorCommon attribute)
(zvt.factors.ma.domain.stock_1d_ma_factor.Stock1dMaFactor attribute)
(zvt.factors.ma.domain.stock_1d_ma_stats_factor.Stock1dMaStatsFactor attribute)
(zvt.factors.top_stocks.TopStocks attribute)
(zvt.factors.z.domain.common.ZFactorCommon attribute)
(zvt.factors.z.domain.stock_1d_z_factor.Stock1dZFactor attribute)
(zvt.factors.z.domain.stock_1wk_z_factor.Stock1wkZFactor attribute)
(zvt.tag.tag_schemas.HiddenTagInfo attribute)
(zvt.tag.tag_schemas.MainTagInfo attribute)
(zvt.tag.tag_schemas.StockPoolInfo attribute)
(zvt.tag.tag_schemas.StockPools attribute)
(zvt.tag.tag_schemas.StockSystemTags attribute)
(zvt.tag.tag_schemas.StockTags attribute)
(zvt.tag.tag_schemas.SubTagInfo attribute)
(zvt.tag.tag_schemas.TagStats attribute)
(zvt.trader.trader_schemas.AccountStats attribute)
(zvt.trader.trader_schemas.Order attribute)
(zvt.trader.trader_schemas.Position attribute)
(zvt.trader.trader_schemas.TraderInfo attribute)
(zvt.trading.trading_schemas.QueryStockQuoteSetting attribute)
(zvt.trading.trading_schemas.TradingPlan attribute)
entity_ids (zvt.contract.recorder.EntityEventRecorder attribute)
(zvt.contract.recorder.FixedCycleDataRecorder attribute)
(zvt.contract.recorder.TimeSeriesDataRecorder attribute)
(zvt.contract.recorder.TimestampsDataRecorder attribute)
(zvt.recorders.eastmoney.common.EastmoneyMoreDataRecorder attribute)
(zvt.recorders.eastmoney.common.EastmoneyPageabeDataRecorder attribute)

(zvt.recorders.eastmoney.common.EastmoneyTimestampsDataRecorder attribute)
(zvt.recorders.eastmoney.dividend_financing.eastmoney_dividend_detail_recorder.DividendDetailRecord
attribute)
(zvt.recorders.eastmoney.dividend_financing.eastmoney_dividend_financing_recorder.DividendFinancing
attribute)
(zvt.recorders.eastmoney.dividend_financing.eastmoney_rights_issue_detail_recorder.RightsIssueDetailR
attribute)
(zvt.recorders.eastmoney.dividend_financing.eastmoney_spo_detail_recorder.SPODetailRecorder attribut
(zvt.recorders.eastmoney.finance.base_china_stock_finance_recorder.BaseChinaStockFinanceRecorder at
(zvt.recorders.eastmoney.finance.eastmoney_balance_sheet_recorder.ChinaStockBalanceSheetRecorder a
(zvt.recorders.eastmoney.finance.eastmoney_cash_flow_recorder.ChinaStockCashFlowRecorder attribute
(zvt.recorders.eastmoney.finance.eastmoney_finance_factor_recorder.ChinaStockFinanceFactorRecorder_
(zvt.recorders.eastmoney.finance.eastmoney_income_statement_recorder.ChinaStockIncomeStatementRe
attribute)
(zvt.recorders.eastmoney.holder.eastmoney_top_ten_holder_recorder.TopTenHolderRecorder attribute)
(zvt.recorders.eastmoney.holder.eastmoney_top_ten_tradable_holder_recorder.TopTenTradableHolderRec
attribute)
(zvt.recorders.eastmoney.meta.eastmoney_block_meta_recorder.EastmoneyBlockStockRecorder attribute
(zvt.recorders.eastmoney.trading.eastmoney_holder_trading_recorder.HolderTradingRecorder attribute)
(zvt.recorders.eastmoney.trading.eastmoney_manager_trading_recorder.ManagerTradingRecorder attribut
(zvt.recorders.em.actor.em_stock_actor_summary_recorder.EMStockActorSummaryRecorder attribute)
(zvt.recorders.em.actor.em_stock_ii_recorder.EMStockIIRecorder attribute)
(zvt.recorders.em.actor.em_stock_top_ten_free_recorder.EMStockTopTenFreeRecorder attribute)
(zvt.recorders.em.actor.em_stock_top_ten_recorder.EMStockTopTenRecorder attribute)
(zvt.recorders.em.macro.em_treasury_yield_recorder.EMTreasuryYieldRecorder attribute)
(zvt.recorders.em.meta.em_block_meta_recorder.EMBlockStockRecorder attribute)
(zvt.recorders.em.misc.em_stock_news_recorder.EMStockNewsRecorder attribute)
(zvt.recorders.em.quotes.em_kdata_recorder.BaseEMStockKdataRecorder attribute)
(zvt.recorders.em.quotes.em_kdata_recorder.EMBlockKdataRecorder attribute)
(zvt.recorders.em.quotes.em_kdata_recorder.EMCurrencyKdataRecorder attribute)
(zvt.recorders.em.quotes.em_kdata_recorder.EMFutureKdataRecorder attribute)
(zvt.recorders.em.quotes.em_kdata_recorder.EMIndexKdataRecorder attribute)
(zvt.recorders.em.quotes.em_kdata_recorder.EMIndexusKdataRecorder attribute)
(zvt.recorders.em.quotes.em_kdata_recorder.EMStockhkKdataRecorder attribute)
(zvt.recorders.em.quotes.em_kdata_recorder.EMStockKdataRecorder attribute)
(zvt.recorders.em.quotes.em_kdata_recorder.EMStockusKdataRecorder attribute)
(zvt.recorders.em.trading.em_dragon_and_tiger_recorder.EMDragonAndTigerRecorder attribute)
(zvt.recorders.exchange.exchange_index_stock_recorder.ExchangeIndexStockRecorder attribute)
(zvt.recorders.exchange.exchange_stock_summary_recorder.ExchangeStockSummaryRecorder attribute)
(zvt.recorders.joinquant.fundamental.jq_etf_valuation_recorder.JqChinaEtfValuationRecorder attribute)
(zvt.recorders.joinquant.fundamental.jq_margin_trading_recorder.MarginTradingRecorder attribute)
(zvt.recorders.joinquant.fundamental.jq_stock_valuation_recorder.JqChinaStockValuationRecorder attribu
(zvt.recorders.joinquant.meta.jq_fund_meta_recorder.JqChinaFundStockRecorder attribute)
(zvt.recorders.joinquant.meta.jq_stock_meta_recorder.JqChinaStockEtfPortfolioRecorder attribute)
(zvt.recorders.joinquant.meta.jq_trade_day_recorder.StockTradeDayRecorder attribute)
(zvt.recorders.joinquant.misc.jq_hk_holder_recorder.JoinquantHkHolderRecorder attribute)
(zvt.recorders.joinquant.misc.jq_index_money_flow_recorder.JoinquantIndexMoneyFlowRecorder attribu
(zvt.recorders.joinquant.misc.jq_stock_money_flow_recorder.JoinquantStockMoneyFlowRecorder attribu
(zvt.recorders.joinquant.overall.jq_cross_market_recorder.CrossMarketSummaryRecorder attribute)
(zvt.recorders.joinquant.overall.jq_margin_trading_recorder.MarginTradingSummaryRecorder attribute)
(zvt.recorders.joinquant.overall.jq_stock_summary_recorder.StockSummaryRecorder attribute)
(zvt.recorders.joinquant.quotes.jq_index_kdata_recorder.JqChinaIndexKdataRecorder attribute)
(zvt.recorders.joinquant.quotes.jq_stock_kdata_recorder.JqChinaStockKdataRecorder attribute)
(zvt.recorders.sina.meta.sina_block_recorder.SinaChinaBlockStockRecorder attribute)
(zvt.recorders.sina.money_flow.sina_block_money_flow_recorder.SinaBlockMoneyFlowRecorder attribu
(zvt.recorders.sina.money_flow.sina_stock_money_flow_recorder.SinaStockMoneyFlowRecorder attribut

(zvt.recorders.sina.quotes.sina_etf_kdata_recorder.ChinaETFDayKdataRecorder attribute)
(zvt.recorders.sina.quotes.sina_index_kdata_recorder.ChinaIndexDayKdataRecorder attribute)
(zvt.recorders.wb.wb_economy_recorder.WBEconomyRecorder attribute)

F

Factor (class in zvt.contract.factor)
factor_cls_registry (zvt.contract.context.Registry attribute)
factor_col_map_object_hook() (zvt.contract.factor.Factor method)
(zvt.contract.factor.ScoreFactor method)
(zvt.factors.fundamental.finance_factor.FinanceBaseFactor method)
(zvt.factors.fundamental.finance_factor.GoodCompanyFactor method)
(zvt.factors.ma.ma_factor.CrossMaFactor method)
(zvt.factors.ma.ma_factor.CrossMaVolumeFactor method)
(zvt.factors.ma.ma_factor.MaFactor method)
(zvt.factors.ma.ma_factor.VolumeUpMaFactor method)
(zvt.factors.ma.ma_stats_factor.MaStatsFactor method)
(zvt.factors.ma.ma_stats_factor.TFactor method)
(zvt.factors.ma.top_bottom_factor.TopBottomFactor method)
(zvt.factors.macd.macd_factor.BullFactor method)
(zvt.factors.macd.macd_factor.GoldCrossFactor method)
(zvt.factors.macd.macd_factor.KeepBullFactor method)
(zvt.factors.macd.macd_factor.LiveOrDeadFactor method)
(zvt.factors.macd.macd_factor.MacdFactor method)
(zvt.factors.technical_factor.TechnicalFactor method)
(zvt.factors.z.z_factor.ZFactor method)
factor_data_list (zvt.contract.drawer.Drawer attribute)
factor_df (zvt.contract.factor.Factor attribute)
(zvt.contract.factor.ScoreFactor attribute)
(zvt.factors.fundamental.finance_factor.FinanceBaseFactor attribute)
(zvt.factors.fundamental.finance_factor.GoodCompanyFactor attribute)
(zvt.factors.ma.ma_factor.CrossMaFactor attribute)
(zvt.factors.ma.ma_factor.CrossMaVolumeFactor attribute)
(zvt.factors.ma.ma_factor.MaFactor attribute)
(zvt.factors.ma.ma_factor.VolumeUpMaFactor attribute)
(zvt.factors.ma.ma_stats_factor.MaStatsFactor attribute)
(zvt.factors.ma.ma_stats_factor.TFactor attribute)
(zvt.factors.ma.top_bottom_factor.TopBottomFactor attribute)
(zvt.factors.macd.macd_factor.BullFactor attribute)
(zvt.factors.macd.macd_factor.GoldCrossFactor attribute)
(zvt.factors.macd.macd_factor.KeepBullFactor attribute)
(zvt.factors.macd.macd_factor.LiveOrDeadFactor attribute)
(zvt.factors.macd.macd_factor.MacdFactor attribute)
(zvt.factors.technical_factor.TechnicalFactor attribute)
(zvt.factors.z.z_factor.ZFactor attribute)
factor_schema (zvt.contract.factor.Factor attribute)
(zvt.contract.factor.ScoreFactor attribute)
(zvt.factors.fundamental.finance_factor.FinanceBaseFactor attribute)
fi_deposit_reservation_for_balance
(zvt.domain.fundamental.finance.Bala
fi_disbursing_loans_and_advances
(zvt.domain.fundamental.finance.Bala
fi_disposal_trade_asset_decrease
(zvt.domain.fundamental.finance.Cash
fi_dividend_expenses_to_insured
(zvt.domain.fundamental.finance.Inco
fi_dividend_payable_for_reinsurance
(zvt.domain.fundamental.finance.Bala
fi_estimated_liabilities
(zvt.domain.fundamental.finance.Bala
fi_expenses_for_bank_interest
(zvt.domain.fundamental.finance.Inco
fi_expenses_for_fees_and_commissio
(zvt.domain.fundamental.finance.Inco
fi_expenses_for_fees_and_commissio
(zvt.domain.fundamental.finance.Inco
fi_expensive_metals
(zvt.domain.fundamental.finance.Bala
fi_fees_and_commissions_payable
(zvt.domain.fundamental.finance.Bala
fi_fees_from_other
(zvt.domain.fundamental.finance.Inco
fi_financial_assets_effect_current_inc
(zvt.domain.fundamental.finance.Bala
fi_financial_derivative_asset
(zvt.domain.fundamental.finance.Bala
fi_financial_derivative_liability
(zvt.domain.fundamental.finance.Bala
fi_financial_liability_effect_current_i
(zvt.domain.fundamental.finance.Bala
fi_generic_risk_reserve
(zvt.domain.fundamental.finance.Bala
fi_held_to_maturity_investment
(zvt.domain.fundamental.finance.Bala
fi_income_from_bank_interest
(zvt.domain.fundamental.finance.Inco
fi_income_from_exchange
(zvt.domain.fundamental.finance.Inco
fi_income_from_fair_value_change
(zvt.domain.fundamental.finance.Inco
fi_income_from_fair_value_change_o
(zvt.domain.fundamental.finance.Inco
fi_income_from_premium
(zvt.domain.fundamental.finance.Inco
fi_income_from_reinsurance_premiu
(zvt.domain.fundamental.finance.Inco

(zvt.factors.fundamental.finance_factor.GoodCompanyFactor attribute)
(zvt.factors.ma.ma_factor.CrossMaFactor attribute)
(zvt.factors.ma.ma_factor.CrossMaVolumeFactor attribute)
(zvt.factors.ma.ma_factor.MaFactor attribute)
(zvt.factors.ma.ma_factor.VolumeUpMaFactor attribute)
(zvt.factors.ma.ma_stats_factor.MaStatsFactor attribute)
(zvt.factors.ma.ma_stats_factor.TFactor attribute)
(zvt.factors.ma.top_bottom_factor.TopBottomFactor attribute)
(zvt.factors.macd.macd_factor.BullFactor attribute)
(zvt.factors.macd.macd_factor.GoldCrossFactor attribute)
(zvt.factors.macd.macd_factor.KeepBullFactor attribute)
(zvt.factors.macd.macd_factor.LiveOrDeadFactor attribute)
(zvt.factors.macd.macd_factor.MacdFactor attribute)
(zvt.factors.technical_factor.TechnicalFactor attribute)
(zvt.factors.z.z_factor.ZFactor attribute)
FactorMeta (class in zvt.contract.factor)
FactorRequestModel (class in zvt.factors.factor_models)
FactorResultModel (class in zvt.factors.factor_models)
FactorState (class in zvt.contract.zvt_info)
FactorStateEncoder (class in zvt.factors.shape)
FallBelowTransformer (class in zvt.factors.transformers)
Fenxing (class in zvt.factors.shape)
fenxing_power (zvt.factors.z.domain.stock_1d_z_factor.Stock1dZFactor attribute)
(zvt.factors.z.domain.stock_1wk_z_factor.Stock1wkZFactor attribute)
fenxing_power() (in module zvt.factors.shape)
fi_account_receivable_increase (zvt.domain.fundamental.finance.CashFlowStatement attribute)
fi_account_receivable_investment (zvt.domain.fundamental.finance.BalanceSheet attribute)
fi_advance_from_customers (zvt.domain.fundamental.finance.BalanceSheet attribute)
fi_advance_premium (zvt.domain.fundamental.finance.BalanceSheet attribute)
fi_amortized_insurance_claims_expenses (zvt.domain.fundamental.finance.IncomeStatement attribute)
fi_amortized_insurance_duty_reserve (zvt.domain.fundamental.finance.IncomeStatement attribute)
fi_amortized_reinsurance_expenses (zvt.domain.fundamental.finance.IncomeStatement attribute)
fi_assets_saleable (zvt.domain.fundamental.finance.BalanceSheet attribute)
fi_bank_broker_buy_repurchase_decrease (zvt.domain.fundamental.finance.CashFlowStatement attribute)
fi_bank_broker_sell_repurchase_decrease (zvt.domain.fundamental.finance.CashFlowStatement attribute)
fi_bank_broker_sell_repurchase_increase (zvt.domain.fundamental.finance.CashFlowStatement attribute)
fi_bond_payable (zvt.domain.fundamental.finance.BalanceSheet attribute)
fi_borrow_from_central_bank_increase (zvt.domain.fundamental.finance.CashFlowStatement attribute)
fi_incomes_from_fees_and_commissi
(zvt.domain.fundamental.finance.Inco
fi_incomes_from_fees_and_commissi
(zvt.domain.fundamental.finance.Inco
fi_independent_liability
(zvt.domain.fundamental.finance.Bala
fi_insurance_claims_expenses
(zvt.domain.fundamental.finance.Inco
fi_insurance_duty_reserve
(zvt.domain.fundamental.finance.Inco
fi_insurance_surrender_costs
(zvt.domain.fundamental.finance.Inco
fi_insured_deposit_increase
(zvt.domain.fundamental.finance.Cash
fi_insured_pledge_loans_increase
(zvt.domain.fundamental.finance.Cash
fi_interest_expenses
(zvt.domain.fundamental.finance.Inco
fi_interest_income
(zvt.domain.fundamental.finance.Inco
fi_interest_receivable
(zvt.domain.fundamental.finance.Bala
fi_lending_and_buy_repurchase_decre
(zvt.domain.fundamental.finance.Cash
fi_lending_and_buy_repurchase_incre
(zvt.domain.fundamental.finance.Cash
fi_lending_decrease
(zvt.domain.fundamental.finance.Cash
fi_lending_increase
(zvt.domain.fundamental.finance.Cash
fi_lending_to_other_fi
(zvt.domain.fundamental.finance.Bala
fi_loan_advance_increase
(zvt.domain.fundamental.finance.Cash
fi_loan_deposit_ratio
(zvt.domain.fundamental.finance.Fina
fi_margin_out_fund
(zvt.domain.fundamental.finance.Bala
fi_net_capital (zvt.domain.fundamenta
attribute)
fi_net_income_from_bank_interest
(zvt.domain.fundamental.finance.Inco
fi_net_income_from_premium
(zvt.domain.fundamental.finance.Inco
fi_net_incomes_from_customer_asset_
(zvt.domain.fundamental.finance.Inco
fi_net_incomes_from_fees_and_comm
(zvt.domain.fundamental.finance.Inco
fi_net_incomes_from_fees_and_comm
(zvt.domain.fundamental.finance.Inco
fi_net_incomes_from_trading_agent
(zvt.domain.fundamental.finance.Inco
fi_net_incomes_from_underwriting
(zvt.domain.fundamental.finance.Inco
fi_net_interest_income
(zvt.domain.fundamental.finance.Inco

fi_borrowing_and_sell_repurchase_decrease (zvt.domain.fundamental.finance.CashFlowStatement attribute)
fi_borrowing_and_sell_repurchase_increase (zvt.domain.fundamental.finance.CashFlowStatement attribute)
fi_borrowing_decrease (zvt.domain.fundamental.finance.CashFlowStatement attribute)
fi_borrowing_increase (zvt.domain.fundamental.finance.CashFlowStatement attribute)
fi_borrowings_from_central_bank (zvt.domain.fundamental.finance.BalanceSheet attribute)
fi_borrowings_from_fi (zvt.domain.fundamental.finance.BalanceSheet attribute)
fi_buy_repurchase_decrease (zvt.domain.fundamental.finance.CashFlowStatement attribute)
fi_buying_sell_back_fi__asset (zvt.domain.fundamental.finance.BalanceSheet attribute)
fi_capital (zvt.domain.fundamental.finance.BalanceSheet attribute)
fi_capital_adequacy_ratio (zvt.domain.fundamental.finance.FinanceFactor attribute)
fi_capital_in_independent_accounts (zvt.domain.fundamental.finance.BalanceSheet attribute)
fi_cash_and_deposit_in_central_bank (zvt.domain.fundamental.finance.BalanceSheet attribute)
fi_cash_from_interest_commission (zvt.domain.fundamental.finance.CashFlowStatement attribute)
fi_cash_from_premium_of_original (zvt.domain.fundamental.finance.CashFlowStatement attribute)
fi_cash_from_repurchase_increase (zvt.domain.fundamental.finance.CashFlowStatement attribute)
fi_cash_from_trading_agent (zvt.domain.fundamental.finance.CashFlowStatement attribute)
fi_cash_to_acquire_subsidiaries (zvt.domain.fundamental.finance.CashFlowStatement attribute)
fi_cash_to_agent_trade (zvt.domain.fundamental.finance.CashFlowStatement attribute)
fi_cash_to_disposal_subsidiaries (zvt.domain.fundamental.finance.CashFlowStatement attribute)
fi_cash_to_dividends (zvt.domain.fundamental.finance.CashFlowStatement attribute)
fi_cash_to_insurance_claim (zvt.domain.fundamental.finance.CashFlowStatement attribute)
fi_cash_to_interest_commission (zvt.domain.fundamental.finance.CashFlowStatement attribute)
fi_cash_to_pay_interest (zvt.domain.fundamental.finance.CashFlowStatement attribute)
fi_cash_to_reinsurance (zvt.domain.fundamental.finance.CashFlowStatement attribute)
fi_cash_to_sell_repurchase (zvt.domain.fundamental.finance.CashFlowStatement attribute)
fi_claims_payable (zvt.domain.fundamental.finance.BalanceSheet attribute)
fi_client_deposit_reservation_for_balance (zvt.domain.fundamental.finance.BalanceSheet attribute)
fi_client_fund (zvt.domain.fundamental.finance.BalanceSheet attribute)
fi_notes_payable (zvt.domain.fundame attribute)
fi_npl_provision_coverage (zvt.domain.fundamental.finance.Fina
fi_npl_ratio (zvt.domain.fundamental.f attribute)
fi_operate_and_manage_expenses (zvt.domain.fundamental.finance.Inco
fi_other_asset (zvt.domain.fundamenta attribute)
fi_other_equity_instruments (zvt.domain.fundamental.finance.Bala
fi_other_income (zvt.domain.fundamental.finance.Inco
fi_other_liability (zvt.domain.fundame attribute)
fi_other_op_expenses (zvt.domain.fundamental.finance.Inco
fi_policy_holder_deposits_and_invest (zvt.domain.fundamental.finance.Bala
fi_policy_holder_dividend_payable (zvt.domain.fundamental.finance.Bala
fi_policy_pledge_loans (zvt.domain.fundamental.finance.Bala
fi_preferred_stock (zvt.domain.fundamental.finance.Bala
fi_premiums_receivable (zvt.domain.fundamental.finance.Bala
fi_receivables (zvt.domain.fundamenta attribute)
fi_receiving_as_agent (zvt.domain.fundamental.finance.Bala
fi_reinsurance_contract_reserve (zvt.domain.fundamental.finance.Bala
fi_reinsurance_expenses (zvt.domain.fundamental.finance.Inco
fi_reinsurance_premium (zvt.domain.fundamental.finance.Inco
fi_reinsurance_premium_receivable (zvt.domain.fundamental.finance.Bala
fi_repurchase_decrease (zvt.domain.fundamental.finance.Cash
fi_savings_absorption (zvt.domain.fundamental.finance.Bala
fi_sell_buy_back_fi_asset (zvt.domain.fundamental.finance.Bala
fi_sell_repurchase_decrease (zvt.domain.fundamental.finance.Cash
fi_sell_repurchase_increase (zvt.domain.fundamental.finance.Cash
fi_short_financing_payable (zvt.domain.fundamental.finance.Bala
fi_time_deposit (zvt.domain.fundamen attribute)
fi_total_deposit (zvt.domain.fundamen attribute)

fi_contract_reserve
(zvt.domain.fundamental.finance.BalanceSheet attribute)
fi_core_capital_adequacy_ratio
(zvt.domain.fundamental.finance.FinanceFactor attribute)
fi_deposit_for_capital_recognizance
(zvt.domain.fundamental.finance.BalanceSheet attribute)
fi_deposit_for_recognizance
(zvt.domain.fundamental.finance.BalanceSheet attribute)
fi_deposit_from_other_fi
(zvt.domain.fundamental.finance.BalanceSheet attribute)
fi_deposit_in_other_fi
(zvt.domain.fundamental.finance.BalanceSheet attribute)
fi_deposit_in_others_decrease
(zvt.domain.fundamental.finance.CashFlowStatement attribute)
fi_deposit_in_others_increase
(zvt.domain.fundamental.finance.CashFlowStatement attribute)
fi_deposit_increase
(zvt.domain.fundamental.finance.CashFlowStatement attribute)
fi_total_loan (zvt.domain.fundamental
attribute)
fi_trade_risk_reserve
(zvt.domain.fundamental.finance.Bala
fi_undue_duty_reserve
(zvt.domain.fundamental.finance.Incor
fields (zvt.trader.sim_account.Account
attribute)
(zvt.trader.sim_account.PositionS
fill_dict() (in module zvt.utils.utils)
fill_domain_from_dict() (in module zv
fill_package() (in module zvt.autocode
fill_package_if_not_exist() (in module
fill_with_same_index() (in module zvt
fin_buy_value (zvt.domain.fundament
attribute)
fin_refund_value
(zvt.domain.fundamental.trading.Marg
fin_sec_value (zvt.domain.fundamenta
attribute)
fin_value (zvt.domain.fundamental.tra
attribute)
FinanceBaseFactor (class in
zvt.factors.fundamental.finance_factor
FinanceFactor (class in zvt.domain.fun
financing_costs
(zvt.domain.fundamental.finance.Incor
first_item_to_float() (in module zvt.uti
first_limit_up_time (zvt.domain.emoti
attribute)
fixed_assets (zvt.domain.fundamental.
attribute)
FixedCycleDataRecorder (class in zvt.
flatten_list() (in module zvt.utils.utils)
float_cap (zvt.domain.quotes.stock.sto
attribute)
(zvt.trading.trading_models.Stock
float_to_pct_str() (in module zvt.api.ut
foreign_exchange_rate_effect
(zvt.domain.fundamental.finance.Cash
forex (zvt.contract.Exchange attribute)
from_datetime() (zvt.zhdate.zhdate.Zh
from_dict() (zvt.trader.sim_account.Ac
class method)
(zvt.trader.sim_account.PositionS
Fund (class in zvt.domain.meta.fund_m
fund (zvt.contract.TradableType attribu
(zvt.domain.IndexCategory attribu
FundStock (class in zvt.domain.meta.f
Future (class in zvt.domain.meta.future
future (zvt.contract.TradableType attrib
Future1dKdata (class in
zvt.domain.quotes.future.future_1d_kd
FutureKdataCommon (class in zvt.don

G

gen_exports() (in module zvt.autocode.generator)
gen_kdata_schema() (in module zvt.autocode.generator)
gen_kdata_schemas() (in module zvt.fill_project)
gen_plugin_project() (in module zvt.autocode.generator)
generate_domain() (zvt.contract.recorder.FixedCycleDataRecorder method)
(zvt.contract.recorder.TimeSeriesDataRecorder method)
(zvt.contract.recorder.TimestampsDataRecorder method)
(zvt.recorders.eastmoney.common.EastmoneyMoreDataRecorder method)
(zvt.recorders.eastmoney.common.EastmoneyPageabeDataRecorder method)
(zvt.recorders.eastmoney.common.EastmoneyTimestampsDataRecorder method)
(zvt.recorders.eastmoney.dividend_financing.eastmoney_dividend_detail_recorder.DividendDetailRecord method)
(zvt.recorders.eastmoney.dividend_financing.eastmoney_dividend_financing_recorder.DividendFinancing method)
(zvt.recorders.eastmoney.dividend_financing.eastmoney_rights_issue_detail_recorder.RightsIssueDetailR method)
(zvt.recorders.eastmoney.dividend_financing.eastmoney_spo_detail_recorder.SPODetailRecorder method
(zvt.recorders.eastmoney.finance.base_china_stock_finance_recorder.BaseChinaStockFinanceRecorder m
(zvt.recorders.eastmoney.finance.eastmoney_balance_sheet_recorder.ChinaStockBalanceSheetRecorder n
(zvt.recorders.eastmoney.finance.eastmoney_cash_flow_recorder.ChinaStockCashFlowRecorder method)
(zvt.recorders.eastmoney.finance.eastmoney_finance_factor_recorder.ChinaStockFinanceFactorRecorder
(zvt.recorders.eastmoney.finance.eastmoney_income_statement_recorder.ChinaStockIncomeStatementRe method)
(zvt.recorders.eastmoney.holder.eastmoney_top_ten_holder_recorder.TopTenHolderRecorder method)
(zvt.recorders.eastmoney.holder.eastmoney_top_ten_tradable_holder_recorder.TopTenTradableHolderRec method)
(zvt.recorders.eastmoney.meta.eastmoney_block_meta_recorder.EastmoneyBlockStockRecorder method)
(zvt.recorders.eastmoney.trading.eastmoney_holder_trading_recorder.HolderTradingRecorder method)
(zvt.recorders.eastmoney.trading.eastmoney_manager_trading_recorder.ManagerTradingRecorder method
(zvt.recorders.em.actor.em_stock_actor_summary_recorder.EMStockActorSummaryRecorder method)
(zvt.recorders.em.actor.em_stock_ii_recorder.EMStockIIRecorder method)
(zvt.recorders.em.actor.em_stock_top_ten_free_recorder.EMStockTopTenFreeRecorder method)
(zvt.recorders.em.actor.em_stock_top_ten_recorder.EMStockTopTenRecorder method)
(zvt.recorders.em.macro.em_treasury_yield_recorder.EMTreasuryYieldRecorder method)
(zvt.recorders.em.meta.em_block_meta_recorder.EMBlockStockRecorder method)
(zvt.recorders.em.misc.em_stock_news_recorder.EMStockNewsRecorder method)
(zvt.recorders.em.quotes.em_kdata_recorder.BaseEMStockKdataRecorder method)
(zvt.recorders.em.quotes.em_kdata_recorder.EMBlockKdataRecorder method)
(zvt.recorders.em.quotes.em_kdata_recorder.EMCurrencyKdataRecorder method)
(zvt.recorders.em.quotes.em_kdata_recorder.EMFutureKdataRecorder method)
(zvt.recorders.em.quotes.em_kdata_recorder.EMIndexKdataRecorder method)
(zvt.recorders.em.quotes.em_kdata_recorder.EMIndexusKdataRecorder method)
(zvt.recorders.em.quotes.em_kdata_recorder.EMStockhkKdataRecorder method)
(zvt.recorders.em.quotes.em_kdata_recorder.EMStockKdataRecorder method)
(zvt.recorders.em.quotes.em_kdata_recorder.EMStockusKdataRecorder method)
(zvt.recorders.em.trading.em_dragon_and_tiger_recorder.EMDragonAndTigerRecorder method)
(zvt.recorders.exchange.exchange_index_stock_recorder.ExchangeIndexStockRecorder method)
(zvt.recorders.exchange.exchange_stock_summary_recorder.ExchangeStockSummaryRecorder method)
(zvt.recorders.joinquant.fundamental.jq_etf_valuation_recorder.JqChinaEtfValuationRecorder method)
(zvt.recorders.joinquant.fundamental.jq_margin_trading_recorder.MarginTradingRecorder method)
(zvt.recorders.joinquant.fundamental.jq_stock_valuation_recorder.JqChinaStockValuationRecorder metho
(zvt.recorders.joinquant.meta.jq_fund_meta_recorder.JqChinaFundStockRecorder method)
(zvt.recorders.joinquant.meta.jq_stock_meta_recorder.JqChinaStockEtfPortfolioRecorder method)

(zvt.recorders.joinquant.meta.jq_trade_day_recorder.StockTradeDayRecorder method)
(zvt.recorders.joinquant.misc.jq_hk_holder_recorder.JoinquantHkHolderRecorder method)
(zvt.recorders.joinquant.misc.jq_index_money_flow_recorder.JoinquantIndexMoneyFlowRecorder metho
(zvt.recorders.joinquant.misc.jq_stock_money_flow_recorder.JoinquantStockMoneyFlowRecorder metho
(zvt.recorders.joinquant.overall.jq_cross_market_recorder.CrossMarketSummaryRecorder method)
(zvt.recorders.joinquant.overall.jq_margin_trading_recorder.MarginTradingSummaryRecorder method)
(zvt.recorders.joinquant.overall.jq_stock_summary_recorder.StockSummaryRecorder method)
(zvt.recorders.joinquant.quotes.jq_index_kdata_recorder.JqChinaIndexKdataRecorder method)
(zvt.recorders.joinquant.quotes.jq_stock_kdata_recorder.JqChinaStockKdataRecorder method)
(zvt.recorders.sina.meta.sina_block_recorder.SinaChinaBlockStockRecorder method)
(zvt.recorders.sina.money_flow.sina_block_money_flow_recorder.SinaBlockMoneyFlowRecorder metho
(zvt.recorders.sina.money_flow.sina_stock_money_flow_recorder.SinaStockMoneyFlowRecorder method
(zvt.recorders.sina.quotes.sina_etf_kdata_recorder.ChinaETFDayKdataRecorder method)
(zvt.recorders.sina.quotes.sina_index_kdata_recorder.ChinaIndexDayKdataRecorder method)
(zvt.recorders.wb.wb_economy_recorder.WBEconomyRecorder method)
generate_domain_id() (zvt.contract.recorder.FixedCycleDataRecorder method)
(zvt.contract.recorder.TimeSeriesDataRecorder method)
(zvt.contract.recorder.TimestampsDataRecorder method)
(zvt.recorders.eastmoney.common.EastmoneyMoreDataRecorder method)
(zvt.recorders.eastmoney.common.EastmoneyPageabeDataRecorder method)
(zvt.recorders.eastmoney.common.EastmoneyTimestampsDataRecorder method)
(zvt.recorders.eastmoney.dividend_financing.eastmoney_dividend_detail_recorder.DividendDetailRecord
method)
(zvt.recorders.eastmoney.dividend_financing.eastmoney_dividend_financing_recorder.DividendFinancing
method)
(zvt.recorders.eastmoney.dividend_financing.eastmoney_rights_issue_detail_recorder.RightsIssueDetailR
method)
(zvt.recorders.eastmoney.dividend_financing.eastmoney_spo_detail_recorder.SPODetailRecorder method
(zvt.recorders.eastmoney.finance.base_china_stock_finance_recorder.BaseChinaStockFinanceRecorder m
(zvt.recorders.eastmoney.finance.eastmoney_balance_sheet_recorder.ChinaStockBalanceSheetRecorder n
(zvt.recorders.eastmoney.finance.eastmoney_cash_flow_recorder.ChinaStockCashFlowRecorder method)
(zvt.recorders.eastmoney.finance.eastmoney_finance_factor_recorder.ChinaStockFinanceFactorRecorder
(zvt.recorders.eastmoney.finance.eastmoney_income_statement_recorder.ChinaStockIncomeStatementRe
method)
(zvt.recorders.eastmoney.holder.eastmoney_top_ten_holder_recorder.TopTenHolderRecorder method)
(zvt.recorders.eastmoney.holder.eastmoney_top_ten_tradable_holder_recorder.TopTenTradableHolderRec
method)
(zvt.recorders.eastmoney.meta.eastmoney_block_meta_recorder.EastmoneyBlockStockRecorder method)
(zvt.recorders.eastmoney.trading.eastmoney_holder_trading_recorder.HolderTradingRecorder method)
(zvt.recorders.eastmoney.trading.eastmoney_manager_trading_recorder.ManagerTradingRecorder method
(zvt.recorders.em.actor.em_stock_actor_summary_recorder.EMStockActorSummaryRecorder method)
(zvt.recorders.em.actor.em_stock_ii_recorder.EMStockIIRecorder method)
(zvt.recorders.em.actor.em_stock_top_ten_free_recorder.EMStockTopTenFreeRecorder method)
(zvt.recorders.em.actor.em_stock_top_ten_recorder.EMStockTopTenRecorder method)
(zvt.recorders.em.macro.em_treasury_yield_recorder.EMTreasuryYieldRecorder method)
(zvt.recorders.em.meta.em_block_meta_recorder.EMBlockStockRecorder method)
(zvt.recorders.em.misc.em_stock_news_recorder.EMStockNewsRecorder method)
(zvt.recorders.em.quotes.em_kdata_recorder.BaseEMStockKdataRecorder method)
(zvt.recorders.em.quotes.em_kdata_recorder.EMBlockKdataRecorder method)
(zvt.recorders.em.quotes.em_kdata_recorder.EMCurrencyKdataRecorder method)
(zvt.recorders.em.quotes.em_kdata_recorder.EMFutureKdataRecorder method)
(zvt.recorders.em.quotes.em_kdata_recorder.EMIndexKdataRecorder method)
(zvt.recorders.em.quotes.em_kdata_recorder.EMIndexusKdataRecorder method)
(zvt.recorders.em.quotes.em_kdata_recorder.EMStockhkKdataRecorder method)
(zvt.recorders.em.quotes.em_kdata_recorder.EMStockKdataRecorder method)
(zvt.recorders.em.quotes.em_kdata_recorder.EMStockusKdataRecorder method)

(zvt.recorders.em.trading.em_dragon_and_tiger_recorder.EMDragonAndTigerRecorder method)
(zvt.recorders.exchange.exchange_index_stock_recorder.ExchangeIndexStockRecorder method)
(zvt.recorders.exchange.exchange_stock_summary_recorder.ExchangeStockSummaryRecorder method)
(zvt.recorders.joinquant.fundamental.jq_etf_valuation_recorder.JqChinaEtfValuationRecorder method)
(zvt.recorders.joinquant.fundamental.jq_margin_trading_recorder.MarginTradingRecorder method)
(zvt.recorders.joinquant.fundamental.jq_stock_valuation_recorder.JqChinaStockValuationRecorder metho
(zvt.recorders.joinquant.meta.jq_fund_meta_recorder.JqChinaFundStockRecorder method)
(zvt.recorders.joinquant.meta.jq_stock_meta_recorder.JqChinaStockEtfPortfolioRecorder method)
(zvt.recorders.joinquant.meta.jq_trade_day_recorder.StockTradeDayRecorder method)
(zvt.recorders.joinquant.misc.jq_hk_holder_recorder.JoinquantHkHolderRecorder method)
(zvt.recorders.joinquant.misc.jq_index_money_flow_recorder.JoinquantIndexMoneyFlowRecorder metho
(zvt.recorders.joinquant.misc.jq_stock_money_flow_recorder.JoinquantStockMoneyFlowRecorder metho
(zvt.recorders.joinquant.overall.jq_cross_market_recorder.CrossMarketSummaryRecorder method)
(zvt.recorders.joinquant.overall.jq_margin_trading_recorder.MarginTradingSummaryRecorder method)
(zvt.recorders.joinquant.overall.jq_stock_summary_recorder.StockSummaryRecorder method)
(zvt.recorders.joinquant.quotes.jq_index_kdata_recorder.JqChinaIndexKdataRecorder method)
(zvt.recorders.joinquant.quotes.jq_stock_kdata_recorder.JqChinaStockKdataRecorder method)
(zvt.recorders.sina.meta.sina_block_recorder.SinaChinaBlockStockRecorder method)
(zvt.recorders.sina.money_flow.sina_block_money_flow_recorder.SinaBlockMoneyFlowRecorder metho
(zvt.recorders.sina.money_flow.sina_stock_money_flow_recorder.SinaStockMoneyFlowRecorder metho
(zvt.recorders.sina.quotes.sina_etf_kdata_recorder.ChinaETFDayKdataRecorder method)
(zvt.recorders.sina.quotes.sina_index_kdata_recorder.ChinaIndexDayKdataRecorder method)
(zvt.recorders.wb.wb_economy_recorder.WBEconomyRecorder method)
generate_filters() (in module zvt.recorders.em.em_api)
generate_kdata_id() (in module zvt.api.kdata)
get_account_stats_figure() (in module zvt.ui.components.dcc_components)
get_attribute() (zvt.trader.sim_account.AccountDayStatsSchema method)
(zvt.trader.sim_account.PositionSchema method)
get_basic_info() (in module zvt.recorders.em.em_api)
get_big_cap_stock() (in module zvt.api.selector)
get_big_players() (in module zvt.api.selector)
get_block_stocks() (in module zvt.recorders.em.em_api)
get_by_id() (in module zvt.contract.api)
get_change_ratio() (in module zvt.api.stats)
get_china_exchange() (in module zvt.api.utils)
get_cn_index() (in module zvt.recorders.exchange.api.cn_index_api)
get_cn_index_stock() (in module zvt.recorders.exchange.api.cn_index_stock_api)
get_company_type() (in module zvt.recorders.eastmoney.common)
get_concept_list() (in module zvt.tag.tag_utils)
get_concept_main_tag_mapping() (in module zvt.tag.tag_utils)
get_continuous_limit_up() (in module zvt.recorders.jqka.jqka_api)
get_controlling_shareholder() (in module zvt.recorders.em.em_api)
get_countries() (in module zvt.recorders.wb.wb_api)
get_cs_index() (in module zvt.recorders.exchange.api.cs_index_api)
get_cs_index_stock() (in module zvt.recorders.exchange.api.cs_index_stock_api)
get_current_position() (zvt.trader.AccountService method)
(zvt.trader.sim_account.SimAccountService method)
get_current_trading_plan() (in module zvt.rest.trading)
(in module zvt.trading.trading_service)
get_data() (in module zvt.contract.api)
get_data_count() (in module zvt.contract.api)
get_data_map() (zvt.contract.recorder.FixedCycleDataRecorder method)
(zvt.contract.recorder.TimeSeriesDataRecorder method)
(zvt.contract.recorder.TimestampsDataRecorder method)
(zvt.recorders.eastmoney.common.EastmoneyMoreDataRecorder method)
(zvt.recorders.eastmoney.common.EastmoneyPageabeDataRecorder method)

(zvt.recorders.eastmoney.common.EastmoneyTimestampsDataRecorder method)
(zvt.recorders.eastmoney.dividend_financing.eastmoney_dividend_detail_recorder.DividendDetailRecord
method)
(zvt.recorders.eastmoney.dividend_financing.eastmoney_dividend_financing_recorder.DividendFinancing
method)
(zvt.recorders.eastmoney.dividend_financing.eastmoney_rights_issue_detail_recorder.RightsIssueDetailR
method)
(zvt.recorders.eastmoney.dividend_financing.eastmoney_spo_detail_recorder.SPODetailRecorder method
(zvt.recorders.eastmoney.finance.base_china_stock_finance_recorder.BaseChinaStockFinanceRecorder m
(zvt.recorders.eastmoney.finance.eastmoney_balance_sheet_recorder.ChinaStockBalanceSheetRecorder m
(zvt.recorders.eastmoney.finance.eastmoney_cash_flow_recorder.ChinaStockCashFlowRecorder method)
(zvt.recorders.eastmoney.finance.eastmoney_finance_factor_recorder.ChinaStockFinanceFactorRecorder
(zvt.recorders.eastmoney.finance.eastmoney_income_statement_recorder.ChinaStockIncomeStatementRe
method)
(zvt.recorders.eastmoney.holder.eastmoney_top_ten_holder_recorder.TopTenHolderRecorder method)
(zvt.recorders.eastmoney.holder.eastmoney_top_ten_tradable_holder_recorder.TopTenTradableHolderRec
method)
(zvt.recorders.eastmoney.meta.eastmoney_block_meta_recorder.EastmoneyBlockStockRecorder method)
(zvt.recorders.eastmoney.trading.eastmoney_holder_trading_recorder.HolderTradingRecorder method)
(zvt.recorders.eastmoney.trading.eastmoney_manager_trading_recorder.ManagerTradingRecorder method
(zvt.recorders.em.actor.em_stock_actor_summary_recorder.EMStockActorSummaryRecorder method)
(zvt.recorders.em.actor.em_stock_ii_recorder.EMStockIIRecorder method)
(zvt.recorders.em.actor.em_stock_top_ten_free_recorder.EMStockTopTenFreeRecorder method)
(zvt.recorders.em.actor.em_stock_top_ten_recorder.EMStockTopTenRecorder method)
(zvt.recorders.em.macro.em_treasury_yield_recorder.EMTreasuryYieldRecorder method)
(zvt.recorders.em.meta.em_block_meta_recorder.EMBlockStockRecorder method)
(zvt.recorders.em.misc.em_stock_news_recorder.EMStockNewsRecorder method)
(zvt.recorders.em.quotes.em_kdata_recorder.BaseEMStockKdataRecorder method)
(zvt.recorders.em.quotes.em_kdata_recorder.EMBlockKdataRecorder method)
(zvt.recorders.em.quotes.em_kdata_recorder.EMCurrencyKdataRecorder method)
(zvt.recorders.em.quotes.em_kdata_recorder.EMFutureKdataRecorder method)
(zvt.recorders.em.quotes.em_kdata_recorder.EMIndexKdataRecorder method)
(zvt.recorders.em.quotes.em_kdata_recorder.EMIndexusKdataRecorder method)
(zvt.recorders.em.quotes.em_kdata_recorder.EMStockhkKdataRecorder method)
(zvt.recorders.em.quotes.em_kdata_recorder.EMStockKdataRecorder method)
(zvt.recorders.em.quotes.em_kdata_recorder.EMStockusKdataRecorder method)
(zvt.recorders.em.trading.em_dragon_and_tiger_recorder.EMDragonAndTigerRecorder method)
(zvt.recorders.exchange.exchange_index_stock_recorder.ExchangeIndexStockRecorder method)
(zvt.recorders.exchange.exchange_stock_summary_recorder.ExchangeStockSummaryRecorder method)
(zvt.recorders.joinquant.fundamental.jq_etf_valuation_recorder.JqChinaEtfValuationRecorder method)
(zvt.recorders.joinquant.fundamental.jq_margin_trading_recorder.MarginTradingRecorder method)
(zvt.recorders.joinquant.fundamental.jq_stock_valuation_recorder.JqChinaStockValuationRecorder metho
(zvt.recorders.joinquant.meta.jq_fund_meta_recorder.JqChinaFundStockRecorder method)
(zvt.recorders.joinquant.meta.jq_stock_meta_recorder.JqChinaStockEtfPortfolioRecorder method)
(zvt.recorders.joinquant.meta.jq_trade_day_recorder.StockTradeDayRecorder method)
(zvt.recorders.joinquant.misc.jq_hk_holder_recorder.JoinquantHkHolderRecorder method)
(zvt.recorders.joinquant.misc.jq_index_money_flow_recorder.JoinquantIndexMoneyFlowRecorder metho
(zvt.recorders.joinquant.misc.jq_stock_money_flow_recorder.JoinquantStockMoneyFlowRecorder metho
(zvt.recorders.joinquant.overall.jq_cross_market_recorder.CrossMarketSummaryRecorder method)
(zvt.recorders.joinquant.overall.jq_margin_trading_recorder.MarginTradingSummaryRecorder method)
(zvt.recorders.joinquant.overall.jq_stock_summary_recorder.StockSummaryRecorder method)
(zvt.recorders.joinquant.quotes.jq_index_kdata_recorder.JqChinaIndexKdataRecorder method)
(zvt.recorders.joinquant.quotes.jq_stock_kdata_recorder.JqChinaStockKdataRecorder method)
(zvt.recorders.sina.meta.sina_block_recorder.SinaChinaBlockStockRecorder method)
(zvt.recorders.sina.money_flow.sina_block_money_flow_recorder.SinaBlockMoneyFlowRecorder metho
(zvt.recorders.sina.money_flow.sina_stock_money_flow_recorder.SinaStockMoneyFlowRecorder metho

(zvt.recorders.sina.quotes.sina_etf_kdata_recorder.ChinaETFDayKdataRecorder method)
(zvt.recorders.sina.quotes.sina_index_kdata_recorder.ChinaIndexDayKdataRecorder method)
(zvt.recorders.wb.wb_economy_recorder.WBEconomyRecorder method)
get_data_providers() (in module zvt.rest.data)
get_data_schemas() (in module zvt.rest.data)
get_db_engine() (in module zvt.contract.api)
get_db_session() (in module zvt.contract.api)
get_db_session_factory() (in module zvt.contract.api)
get_direction() (in module zvt.factors.shape)
get_dragon_and_tiger() (in module zvt.recorders.em.em_api)
get_dragon_and_tiger_list() (in module zvt.recorders.em.em_api)
get_dragon_and_tigger_player() (in module zvt.api.selector)
get_economy_data() (in module zvt.recorders.wb.wb_api)
get_em_data() (in module zvt.recorders.em.em_api)
get_entities() (in module zvt.contract.api)
get_entity_code() (in module zvt.contract.api)
get_entity_exchange() (in module zvt.contract.api)
get_entity_exchanges() (in module zvt.contract)
get_entity_ids() (in module zvt.contract.api)
get_entity_ids_by_filter() (in module zvt.api.selector)
get_entity_list_by_cap() (in module zvt.api.selector)
get_entity_schema() (in module zvt.contract.api)
get_entity_type() (in module zvt.contract.api)
get_etf_stocks() (in module zvt.api.portfolio)
get_evaluated_time_field() (zvt.contract.recorder.FixedCycleDataRecorder method)
(zvt.contract.recorder.TimeSeriesDataRecorder method)
(zvt.contract.recorder.TimestampsDataRecorder method)
(zvt.recorders.eastmoney.common.EastmoneyMoreDataRecorder method)
(zvt.recorders.eastmoney.common.EastmoneyPageabeDataRecorder method)
(zvt.recorders.eastmoney.common.EastmoneyTimestampsDataRecorder method)
(zvt.recorders.eastmoney.dividend_financing.eastmoney_dividend_detail_recorder.DividendDetailRecord
method)
(zvt.recorders.eastmoney.dividend_financing.eastmoney_dividend_financing_recorder.DividendFinancing
method)
(zvt.recorders.eastmoney.dividend_financing.eastmoney_rights_issue_detail_recorder.RightsIssueDetailR
method)
(zvt.recorders.eastmoney.dividend_financing.eastmoney_spo_detail_recorder.SPODetailRecorder method
(zvt.recorders.eastmoney.finance.base_china_stock_finance_recorder.BaseChinaStockFinanceRecorder m
(zvt.recorders.eastmoney.finance.eastmoney_balance_sheet_recorder.ChinaStockBalanceSheetRecorder n
(zvt.recorders.eastmoney.finance.eastmoney_cash_flow_recorder.ChinaStockCashFlowRecorder method)
(zvt.recorders.eastmoney.finance.eastmoney_finance_factor_recorder.ChinaStockFinanceFactorRecorder
(zvt.recorders.eastmoney.finance.eastmoney_income_statement_recorder.ChinaStockIncomeStatementRe
method)
(zvt.recorders.eastmoney.holder.eastmoney_top_ten_holder_recorder.TopTenHolderRecorder method)
(zvt.recorders.eastmoney.holder.eastmoney_top_ten_tradable_holder_recorder.TopTenTradableHolderRec
method)
(zvt.recorders.eastmoney.meta.eastmoney_block_meta_recorder.EastmoneyBlockStockRecorder method)
(zvt.recorders.eastmoney.trading.eastmoney_holder_trading_recorder.HolderTradingRecorder method)
(zvt.recorders.eastmoney.trading.eastmoney_manager_trading_recorder.ManagerTradingRecorder method
(zvt.recorders.em.actor.em_stock_actor_summary_recorder.EMStockActorSummaryRecorder method)
(zvt.recorders.em.actor.em_stock_ii_recorder.EMStockIIRecorder method)
(zvt.recorders.em.actor.em_stock_top_ten_free_recorder.EMStockTopTenFreeRecorder method)
(zvt.recorders.em.actor.em_stock_top_ten_recorder.EMStockTopTenRecorder method)
(zvt.recorders.em.macro.em_treasury_yield_recorder.EMTreasuryYieldRecorder method)
(zvt.recorders.em.meta.em_block_meta_recorder.EMBlockStockRecorder method)
(zvt.recorders.em.misc.em_stock_news_recorder.EMStockNewsRecorder method)

(zvt.recorders.em.quotes.em_kdata_recorder.BaseEMStockKdataRecorder method)
(zvt.recorders.em.quotes.em_kdata_recorder.EMBlockKdataRecorder method)
(zvt.recorders.em.quotes.em_kdata_recorder.EMCurrencyKdataRecorder method)
(zvt.recorders.em.quotes.em_kdata_recorder.EMFutureKdataRecorder method)
(zvt.recorders.em.quotes.em_kdata_recorder.EMIndexKdataRecorder method)
(zvt.recorders.em.quotes.em_kdata_recorder.EMIndexusKdataRecorder method)
(zvt.recorders.em.quotes.em_kdata_recorder.EMStockhkKdataRecorder method)
(zvt.recorders.em.quotes.em_kdata_recorder.EMStockKdataRecorder method)
(zvt.recorders.em.quotes.em_kdata_recorder.EMStockusKdataRecorder method)
(zvt.recorders.em.trading.em_dragon_and_tiger_recorder.EMDragonAndTigerRecorder method)
(zvt.recorders.exchange.exchange_index_stock_recorder.ExchangeIndexStockRecorder method)
(zvt.recorders.exchange.exchange_stock_summary_recorder.ExchangeStockSummaryRecorder method)
(zvt.recorders.joinquant.fundamental.jq_etf_valuation_recorder.JqChinaEtfValuationRecorder method)
(zvt.recorders.joinquant.fundamental.jq_margin_trading_recorder.MarginTradingRecorder method)
(zvt.recorders.joinquant.fundamental.jq_stock_valuation_recorder.JqChinaStockValuationRecorder metho
(zvt.recorders.joinquant.meta.jq_fund_meta_recorder.JqChinaFundStockRecorder method)
(zvt.recorders.joinquant.meta.jq_stock_meta_recorder.JqChinaStockEtfPortfolioRecorder method)
(zvt.recorders.joinquant.meta.jq_trade_day_recorder.StockTradeDayRecorder method)
(zvt.recorders.joinquant.misc.jq_hk_holder_recorder.JoinquantHkHolderRecorder method)
(zvt.recorders.joinquant.misc.jq_index_money_flow_recorder.JoinquantIndexMoneyFlowRecorder metho
(zvt.recorders.joinquant.misc.jq_stock_money_flow_recorder.JoinquantStockMoneyFlowRecorder metho
(zvt.recorders.joinquant.overall.jq_cross_market_recorder.CrossMarketSummaryRecorder method)
(zvt.recorders.joinquant.overall.jq_margin_trading_recorder.MarginTradingSummaryRecorder method)
(zvt.recorders.joinquant.overall.jq_stock_summary_recorder.StockSummaryRecorder method)
(zvt.recorders.joinquant.quotes.jq_index_kdata_recorder.JqChinaIndexKdataRecorder method)
(zvt.recorders.joinquant.quotes.jq_stock_kdata_recorder.JqChinaStockKdataRecorder method)
(zvt.recorders.sina.meta.sina_block_recorder.SinaChinaBlockStockRecorder method)
(zvt.recorders.sina.money_flow.sina_block_money_flow_recorder.SinaBlockMoneyFlowRecorder metho
(zvt.recorders.sina.money_flow.sina_stock_money_flow_recorder.SinaStockMoneyFlowRecorder metho
(zvt.recorders.sina.quotes.sina_etf_kdata_recorder.ChinaETFDayKdataRecorder method)
(zvt.recorders.sina.quotes.sina_index_kdata_recorder.ChinaIndexDayKdataRecorder method)
(zvt.recorders.wb.wb_economy_recorder.WBEconomyRecorder method)
get_exchange() (in module zvt.recorders.em.em_api)
get_factors() (in module zvt.rest.factor)
get_fc() (in module zvt.recorders.eastmoney.common)
get_free_holder_report_dates() (in module zvt.recorders.em.em_api)
get_free_holders() (in module zvt.recorders.em.em_api)
get_from_path_fields() (in module zvt.recorders.eastmoney.common)
get_fund_stocks() (in module zvt.api.portfolio)
get_future_list() (in module zvt.recorders.em.em_api)
get_future_name() (in module zvt.domain)
get_future_trading_plan() (in module zvt.rest.trading)
(in module zvt.trading.trading_service)
get_git_user_email() (in module zvt.utils.git_utils)
get_git_user_name() (in module zvt.utils.git_utils)
get_good_players() (in module zvt.api.selector)
get_group() (in module zvt.contract.api)
get_hidden_tag_info() (in module zvt.rest.work)
get_hidden_tags() (in module zvt.tag.tag_utils)
get_holder_report_dates() (in module zvt.recorders.em.em_api)
get_holders() (in module zvt.recorders.em.em_api)
get_ii_holder() (in module zvt.recorders.em.em_api)
get_ii_holder_report_dates() (in module zvt.recorders.em.em_api)
get_ii_summary() (in module zvt.recorders.em.em_api)
get_incomelevels() (in module zvt.recorders.wb.wb_api)
get_indicator_data() (in module zvt.recorders.wb.wb_api)

get_indicators() (in module zvt.recorders.wb.wb_api)
get_industry_list() (in module zvt.tag.tag_utils)
get_industry_main_tag_mapping() (in module zvt.tag.tag_utils)
get_initial_hidden_tag_info() (in module zvt.tag.tag_utils)
get_initial_main_tag_info() (in module zvt.tag.tag_utils)
get_initial_stock_pool_info() (in module zvt.tag.tag_utils)
get_initial_sub_tag_info() (in module zvt.tag.tag_utils)
get_initial_sub_tags() (in module zvt.tag.tag_utils)
get_instance() (zvt.trader.sim_account.AccountDayStatsSchema method)
(zvt.trader.sim_account.PositionSchema method)
get_interval_timestamps() (zvt.contract.schema.NormalEntityMixin class method)
(zvt.contract.schema.Portfolio class method)
(zvt.contract.schema.TradableEntity class method)
(zvt.domain.meta.block_meta.Block class method)
(zvt.domain.meta.country_meta.Country class method)
(zvt.domain.meta.currency_meta.Currency class method)
(zvt.domain.meta.etf_meta.Etf class method)
(zvt.domain.meta.fund_meta.Fund class method)
(zvt.domain.meta.future_meta.Future class method)
(zvt.domain.meta.index_meta.Index class method)
(zvt.domain.meta.indexus_meta.Indexus class method)
(zvt.domain.meta.stock_meta.Stock class method)
(zvt.domain.meta.stock_meta.StockDetail class method)
(zvt.domain.meta.stockhk_meta.Stockhk class method)
(zvt.domain.meta.stockus_meta.Stockus class method)
get_jkqa_data() (in module zvt.recorders.jqka.jqka_api)
get_kdata() (in module zvt.api.kdata)
(in module zvt.recorders.em.em_api)
get_kdata_schema() (in module zvt.api.kdata)
get_latest_kdata_date() (in module zvt.api.kdata)
get_lendingtypes() (in module zvt.recorders.wb.wb_api)
get_limit_down() (in module zvt.recorders.jqka.jqka_api)
get_limit_stats() (in module zvt.recorders.jqka.jqka_api)
get_limit_up() (in module zvt.recorders.jqka.jqka_api)
get_limit_up_stocks() (in module zvt.api.selector)
get_local_timezone() (in module zvt.utils.time_utils)
get_ma_factor_schema() (in module zvt.factors.ma.ma_factor)
get_ma_stats_factor_schema() (in module zvt.factors.ma.ma_stats_factor)
get_main_tag_concept_mapping() (in module zvt.tag.tag_utils)
get_main_tag_industry_mapping() (in module zvt.tag.tag_utils)
get_main_tag_info() (in module zvt.rest.work)
get_main_tags() (in module zvt.tag.tag_utils)
get_middle_and_big_stock() (in module zvt.api.selector)
get_middle_cap_stock() (in module zvt.api.selector)
get_mini_and_small_stock() (in module zvt.api.selector)
get_mini_cap_stock() (in module zvt.api.selector)
get_news() (in module zvt.recorders.em.em_api)
get_order_securities() (in module zvt.trader.trader_info_api)
get_performance() (in module zvt.api.stats)

H

handle_duan() (in module zvt.factors.shape)
handle_error() (zvt.trader.sim_account.AccountDayStatsSchema method)
(zvt.trader.sim_account.PositionSchema method)
handle_factor_targets() (zvt.samples.stock_traders.MyBullTrader method)
(zvt.samples.stock_traders.MyMaTrader method)
(zvt.trader.trader.StockTrader method)
(zvt.trader.trader.Trader method)
handle_first_fenxing() (in module zvt.factors.shape)
handle_including() (in module zvt.factors.shape)
handle_zhongshu() (in module zvt.factors.shape)
hfq (zvt.contract.AdjustType attribute)
HiddenTagInfo (class in zvt.tag.tag_schemas)
high (zvt.domain.quotes.block.block_1d_kdata.Block1dKdata attribute)
(zvt.domain.quotes.block.block_1mon_kdata.Block1monKdata attribute)
(zvt.domain.quotes.block.block_1wk_kdata.Block1wkKdata attribute)
high_days (zvt.domain.emo
high_days_count (zvt.doma
attribute)
histogram (zvt.contract.dra
hk (zvt.contract.Exchange a
HkHolder (class in zvt.dom
holder_code (zvt.domain.m
(zvt.domain.misc.hold
(zvt.domain.misc.hold
(zvt.domain.misc.hold
holder_name (zvt.domain.f
attribute)
(zvt.domain.misc.hold
(zvt.domain.misc.hold
(zvt.domain.misc.hold
(zvt.domain.misc.hold
HolderTrading (class in zvt

(zvt.domain.quotes.currency.currency_1d_kdata.Currency1dKdata attribute)
(zvt.domain.quotes.etf.etf_1d_kdata.Etf1dKdata attribute)
(zvt.domain.quotes.future.future_1d_kdata.Future1dKdata attribute)
(zvt.domain.quotes.index.index_1d_kdata.Index1dKdata attribute)
(zvt.domain.quotes.index.index_1wk_kdata.Index1wkKdata attribute)
(zvt.domain.quotes.indexus.indexus_1d_kdata.Indexus1dKdata attribute)
(zvt.domain.quotes.stock.stock_15m_hfq_kdata.Stock15mHfqKdata attribute)
(zvt.domain.quotes.stock.stock_15m_kdata.Stock15mKdata attribute)
(zvt.domain.quotes.stock.stock_1d_hfq_kdata.Stock1dHfqKdata attribute)
(zvt.domain.quotes.stock.stock_1d_kdata.Stock1dKdata attribute)
(zvt.domain.quotes.stock.stock_1h_hfq_kdata.Stock1hHfqKdata attribute)
(zvt.domain.quotes.stock.stock_1h_kdata.Stock1hKdata attribute)
(zvt.domain.quotes.stock.stock_1m_hfq_kdata.Stock1mHfqKdata attribute)
(zvt.domain.quotes.stock.stock_1m_kdata.Stock1mKdata attribute)
(zvt.domain.quotes.stock.stock_1mon_hfq_kdata.Stock1monHfqKdata attribute)
(zvt.domain.quotes.stock.stock_1mon_kdata.Stock1monKdata attribute)
(zvt.domain.quotes.stock.stock_1wk_hfq_kdata.Stock1wkHfqKdata attribute)
(zvt.domain.quotes.stock.stock_1wk_kdata.Stock1wkKdata attribute)
(zvt.domain.quotes.stock.stock_30m_hfq_kdata.Stock30mHfqKdata attribute)
(zvt.domain.quotes.stock.stock_30m_kdata.Stock30mKdata attribute)
(zvt.domain.quotes.stock.stock_4h_hfq_kdata.Stock4hHfqKdata attribute)
(zvt.domain.quotes.stock.stock_4h_kdata.Stock4hKdata attribute)
(zvt.domain.quotes.stock.stock_5m_hfq_kdata.Stock5mHfqKdata attribute)
(zvt.domain.quotes.stock.stock_5m_kdata.Stock5mKdata attribute)
(zvt.domain.quotes.stockhk.stockhk_1d_hfq_kdata.Stockhk1dHfqKdata attribute)
(zvt.domain.quotes.stockhk.stockhk_1d_kdata.Stockhk1dKdata attribute)
(zvt.domain.quotes.stockus.stockus_1d_hfq_kdata.Stockus1dHfqKdata attribute)
(zvt.domain.quotes.stockus.stockus_1d_kdata.Stockus1dKdata attribute)
(zvt.domain.quotes.TickCommon attribute)
(zvt.factors.ma.domain.stock_1d_ma_stats_factor.Stock1dMaStatsFactor attribute)
(zvt.factors.z.domain.stock_1d_z_factor.Stock1dZFactor attribute)
(zvt.factors.z.domain.stock_1wk_z_factor.Stock1wkZFactor attribute)

I

id (zvt.contract.schema.ActorEntity attribute)
(zvt.contract.schema.ActorMeetTradable attribute)
(zvt.contract.schema.Entity attribute)
(zvt.contract.schema.Mixin attribute)
(zvt.contract.schema.NormalEntityMixin attribute)
HolderTradingRecorder (cl
zvt.recorders.eastmoney.tra
holding (zvt.domain.fundar
holding_numbers (zvt.dom
attribute)
(zvt.domain.actor.stoc
attribute)
(zvt.domain.actor.stoc
attribute)
(zvt.domain.actor.stoc
holding_pct (zvt.domain.fu
attribute)
holding_ratio (zvt.domain.a
attribute)
(zvt.domain.actor.stoc
attribute)
(zvt.domain.actor.stoc
attribute)
(zvt.domain.actor.stoc
holding_values (zvt.domai
attribute)
(zvt.domain.actor.stoc
attribute)
(zvt.domain.actor.stoc
attribute)
(zvt.domain.actor.stoc
holiday_distance() (in mod
Indicator (class in zvt.contr
individual (zvt.contract.Act
industries (zvt.domain.meta
industry (zvt.domain.Block
(zvt.domain.IndexCat

(zvt.contract.schema.NormalMixin attribute)
(zvt.contract.schema.Portfolio attribute)
(zvt.contract.schema.PortfolioStock attribute)
(zvt.contract.schema.PortfolioStockHistory attribute)
(zvt.contract.schema.TradableEntity attribute)
(zvt.contract.schema.TradableMeetActor attribute)
(zvt.contract.zvt_info.FactorState attribute)
(zvt.contract.zvt_info.RecorderState attribute)
(zvt.contract.zvt_info.StateMixin attribute)
(zvt.contract.zvt_info.TaggerState attribute)
(zvt.domain.actor.actor_meta.ActorMeta attribute)
(zvt.domain.actor.stock_actor.StockActorSummary attribute)
(zvt.domain.actor.stock_actor.StockInstitutionalInvestorHolder attribute)
(zvt.domain.actor.stock_actor.StockTopTenFreeHolder attribute)
(zvt.domain.actor.stock_actor.StockTopTenHolder attribute)
(zvt.domain.emotion.emotion.Emotion attribute)
(zvt.domain.emotion.emotion.LimitDownInfo attribute)
(zvt.domain.emotion.emotion.LimitUpInfo attribute)
(zvt.domain.fundamental.dividend_financing.DividendDetail attribute)
(zvt.domain.fundamental.dividend_financing.DividendFinancing attribute)
(zvt.domain.fundamental.dividend_financing.RightsIssueDetail attribute)
(zvt.domain.fundamental.dividend_financing.SpoDetail attribute)
(zvt.domain.fundamental.finance.BalanceSheet attribute)
(zvt.domain.fundamental.finance.CashFlowStatement attribute)
(zvt.domain.fundamental.finance.FinanceFactor attribute)
(zvt.domain.fundamental.finance.IncomeStatement attribute)
(zvt.domain.fundamental.trading.BigDealTrading attribute)
(zvt.domain.fundamental.trading.DragonAndTiger attribute)
(zvt.domain.fundamental.trading.HolderTrading attribute)
(zvt.domain.fundamental.trading.ManagerTrading attribute)
(zvt.domain.fundamental.trading.MarginTrading attribute)
(zvt.domain.fundamental.valuation.EtfValuation attribute)
(zvt.domain.fundamental.valuation.StockValuation attribute)
(zvt.domain.macro.macro.Economy attribute)
(zvt.domain.macro.monetary.TreasuryYield attribute)
(zvt.domain.meta.block_meta.Block attribute)
(zvt.domain.meta.block_meta.BlockStock attribute)
(zvt.domain.meta.country_meta.Country attribute)
(zvt.domain.meta.currency_meta.Currency attribute)
(zvt.domain.meta.etf_meta.Etf attribute)
(zvt.domain.meta.etf_meta.EtfStock attribute)
(zvt.domain.meta.fund_meta.Fund attribute)
(zvt.domain.meta.fund_meta.FundStock attribute)
(zvt.domain.meta.future_meta.Future attribute)
(zvt.domain.meta.index_meta.Index attribute)
(zvt.domain.meta.index_meta.IndexStock attribute)
(zvt.domain.meta.indexus_meta.Indexus attribute)
(zvt.domain.meta.stock_meta.Stock attribute)
(zvt.domain.meta.stock_meta.StockDetail attribute)
(zvt.domain.meta.stockhk_meta.Stockhk attribute)
(zvt.domain.meta.stockus_meta.Stockus attribute)
(zvt.domain.misc.holder.HkHolder attribute)
(zvt.domain.misc.holder.InstitutionalInvestorHolder attribute)
(zvt.domain.misc.holder.TopTenHolder attribute)
industry_indices (zvt.doma
industry_to_main_tag() (in
ine (zvt.contract.Exchange
inform_email() (in module
Informer (class in zvt.infor
init_config() (in module zv
init_entities() (zvt.contract.
(zvt.contract.recorder.
(zvt.contract.recorder.
(zvt.contract.recorder.
(zvt.recorders.eastmor
(zvt.recorders.eastmor
(zvt.recorders.eastmor
(zvt.recorders.eastmor
method)
(zvt.recorders.eastmor
method)
(zvt.recorders.eastmor
method)
(zvt.recorders.eastmor
(zvt.recorders.eastmor
(zvt.recorders.eastmor
(zvt.recorders.eastmor
(zvt.recorders.eastmor
(zvt.recorders.eastmor
method)
(zvt.recorders.eastmor
(zvt.recorders.eastmor
method)
(zvt.recorders.eastmor
(zvt.recorders.eastmor
(zvt.recorders.eastmor
(zvt.recorders.em.acto
(zvt.recorders.em.acto
(zvt.recorders.em.acto
(zvt.recorders.em.acto
(zvt.recorders.em.mac
(zvt.recorders.em.met
(zvt.recorders.em.misc
(zvt.recorders.em.quot
(zvt.recorders.em.quot
(zvt.recorders.em.quot
(zvt.recorders.em.quot
(zvt.recorders.em.quot
(zvt.recorders.em.quot
(zvt.recorders.em.quot
(zvt.recorders.em.quot
(zvt.recorders.em.quot
(zvt.recorders.em.trad
(zvt.recorders.exchang
(zvt.recorders.exchang
(zvt.recorders.joinqua
(zvt.recorders.joinqua
(zvt.recorders.joinqua
(zvt.recorders.joinqua
(zvt.recorders.joinqua

(zvt.domain.misc.holder.TopTenTradableHolder attribute)
(zvt.domain.misc.money_flow.BlockMoneyFlow attribute)
(zvt.domain.misc.money_flow.IndexMoneyFlow attribute)
(zvt.domain.misc.money_flow.StockMoneyFlow attribute)
(zvt.domain.misc.overall.CrossMarketSummary attribute)
(zvt.domain.misc.overall.MarginTradingSummary attribute)
(zvt.domain.misc.overall.StockSummary attribute)
(zvt.domain.misc.stock_news.StockNews attribute)
(zvt.domain.quotes.block.block_1d_kdata.Block1dKdata attribute)
(zvt.domain.quotes.block.block_1mon_kdata.Block1monKdata attribute)
(zvt.domain.quotes.block.block_1wk_kdata.Block1wkKdata attribute)
(zvt.domain.quotes.BlockKdataCommon attribute)
(zvt.domain.quotes.currency.currency_1d_kdata.Currency1dKdata attribute)
(zvt.domain.quotes.CurrencyKdataCommon attribute)
(zvt.domain.quotes.etf.etf_1d_kdata.Etf1dKdata attribute)
(zvt.domain.quotes.EtfKdataCommon attribute)
(zvt.domain.quotes.future.future_1d_kdata.Future1dKdata attribute)
(zvt.domain.quotes.FutureKdataCommon attribute)
(zvt.domain.quotes.index.index_1d_kdata.Index1dKdata attribute)
(zvt.domain.quotes.index.index_1wk_kdata.Index1wkKdata attribute)
(zvt.domain.quotes.IndexKdataCommon attribute)
(zvt.domain.quotes.indexus.indexus_1d_kdata.Indexus1dKdata attribute)
(zvt.domain.quotes.IndexusKdataCommon attribute)
(zvt.domain.quotes.KdataCommon attribute)
(zvt.domain.quotes.stock.stock_15m_hfq_kdata.Stock15mHfqKdata attribute)
(zvt.domain.quotes.stock.stock_15m_kdata.Stock15mKdata attribute)
(zvt.domain.quotes.stock.stock_1d_hfq_kdata.Stock1dHfqKdata attribute)
(zvt.domain.quotes.stock.stock_1d_kdata.Stock1dKdata attribute)
(zvt.domain.quotes.stock.stock_1h_hfq_kdata.Stock1hHfqKdata attribute)
(zvt.domain.quotes.stock.stock_1h_kdata.Stock1hKdata attribute)
(zvt.domain.quotes.stock.stock_1m_hfq_kdata.Stock1mHfqKdata attribute)
(zvt.domain.quotes.stock.stock_1m_kdata.Stock1mKdata attribute)
(zvt.domain.quotes.stock.stock_1mon_hfq_kdata.Stock1monHfqKdata attribute)
(zvt.domain.quotes.stock.stock_1mon_kdata.Stock1monKdata attribute)
(zvt.domain.quotes.stock.stock_1wk_hfq_kdata.Stock1wkHfqKdata attribute)
(zvt.domain.quotes.stock.stock_1wk_kdata.Stock1wkKdata attribute)
(zvt.domain.quotes.stock.stock_30m_hfq_kdata.Stock30mHfqKdata attribute)
(zvt.domain.quotes.stock.stock_30m_kdata.Stock30mKdata attribute)
(zvt.domain.quotes.stock.stock_4h_hfq_kdata.Stock4hHfqKdata attribute)
(zvt.domain.quotes.stock.stock_4h_kdata.Stock4hKdata attribute)
(zvt.domain.quotes.stock.stock_5m_hfq_kdata.Stock5mHfqKdata attribute)
(zvt.domain.quotes.stock.stock_5m_kdata.Stock5mKdata attribute)
(zvt.domain.quotes.stock.stock_quote.StockQuote attribute)
(zvt.recorders.joinqua
(zvt.recorders.joinqua
(zvt.recorders.joinqua
(zvt.recorders.joinqua
(zvt.recorders.joinqua
(zvt.recorders.joinqua
(zvt.recorders.joinqua
(zvt.recorders.joinqua
(zvt.recorders.joinqua
(zvt.recorders.sina.me
(zvt.recorders.sina.mo
(zvt.recorders.sina.mo
(zvt.recorders.sina.qu
(zvt.recorders.sina.qu
(zvt.recorders.wb.wb_
(zvt.samples.stock_tra
(zvt.samples.stock_tra
(zvt.trader.trader.Stoc
(zvt.trader.trader.Trad
init_env() (in module zvt)
init_factors() (zvt.samples.
(zvt.samples.stock_tra
(zvt.trader.trader.Stoc
(zvt.trader.trader.Trad
init_log() (in module zvt)
init_main_index() (in mod
init_plugins() (in module z
init_resources() (in module
institutional_investor_type
InstitutionalInvestorHolder
insurance (zvt.contract.Act
insurance_earned_premiu
insurance_net_investment
insurance_payout (zvt.dom
insurance_roi (zvt.domain.
insurance_solvency_adequ
insurance_surrender_rate (
intangible_assets (zvt.dom
interest (zvt.domain.quotes
(zvt.domain.quotes.C
(zvt.domain.quotes.fu
(zvt.domain.quotes.F
interest_payable (zvt.dom
intersect() (in module zvt.f
intersect_ranges() (in mod
IntersectTransformer (class
IntervalLevel (class in zvt.
InvalidOrderError
InvalidOrderParamError
inventories (zvt.domain.fu
inventory_turnover (zvt.do
inventory_turnover_days (
investment_income (zvt.do
investment_income_from
ipo_issues (zvt.domain.fun
is_again_limit (zvt.domain

(zvt.domain.quotes.stockhk.stockhk_1d_hfq_kdata.Stockhk1dHfqKdata attribute)
(zvt.domain.quotes.stockhk.stockhk_1d_kdata.Stockhk1dKdata attribute)
(zvt.domain.quotes.StockhkKdataCommon attribute)
(zvt.domain.quotes.StockKdataCommon attribute)
(zvt.domain.quotes.stockus.stockus_1d_hfq_kdata.Stockus1dHfqKdata attribute)
(zvt.domain.quotes.stockus.stockus_1d_kdata.Stockus1dKdata attribute)
(zvt.domain.quotes.StockusKdataCommon attribute)
(zvt.domain.quotes.TickCommon attribute)
(zvt.domain.quotes.trade_day.StockTradeDay attribute)
(zvt.factors.ma.domain.common.MaStatsFactorCommon attribute)
(zvt.factors.ma.domain.stock_1d_ma_factor.Stock1dMaFactor attribute)
(zvt.factors.ma.domain.stock_1d_ma_stats_factor.Stock1dMaStatsFactor attribute)
(zvt.factors.top_stocks.TopStocks attribute)
(zvt.factors.z.domain.common.ZFactorCommon attribute)
(zvt.factors.z.domain.stock_1d_z_factor.Stock1dZFactor attribute)
(zvt.factors.z.domain.stock_1wk_z_factor.Stock1wkZFactor attribute)
(zvt.tag.tag_schemas.HiddenTagInfo attribute)
(zvt.tag.tag_schemas.MainTagInfo attribute)
(zvt.tag.tag_schemas.StockPoolInfo attribute)
(zvt.tag.tag_schemas.StockPools attribute)
(zvt.tag.tag_schemas.StockSystemTags attribute)
(zvt.tag.tag_schemas.StockTags attribute)
(zvt.tag.tag_schemas.SubTagInfo attribute)
(zvt.tag.tag_schemas.TagStats attribute)
(zvt.trader.trader_schemas.AccountStats attribute)
(zvt.trader.trader_schemas.Order attribute)
(zvt.trader.trader_schemas.Position attribute)
(zvt.trader.trader_schemas.TraderInfo attribute)
(zvt.trading.trading_schemas.QueryStockQuoteSetting attribute)
(zvt.trading.trading_schemas.TradingPlan attribute)
income_level (zvt.domain.meta.country_meta.Country attribute)
IncomeStatement (class in zvt.domain.fundamental.finance)
Index (class in zvt.domain.meta.index_meta)
index (zvt.contract.TradableType attribute)
Index1dKdata (class in zvt.domain.quotes.index.index_1d_kdata)
Index1wkKdata (class in zvt.domain.quotes.index.index_1wk_kdata)
index_df() (in module zvt.utils.pd_utils)
IndexCategory (class in zvt.domain)
IndexKdataCommon (class in zvt.domain.quotes)
IndexMoneyFlow (class in zvt.domain.misc.money_flow)
IndexStock (class in zvt.domain.meta.index_meta)
Indexus (class in zvt.domain.meta.indexus_meta)
indexus (zvt.contract.TradableType attribute)
Indexus1dKdata (class in zvt.domain.quotes.indexus.indexus_1d_kdata)
IndexusKdataCommon (class in zvt.domain.quotes)

(zvt.domain.emotion.e
is_complete (zvt.domain.ac
is_down() (in module zvt.fa
is_filter_result_df() (in mod
is_finished_kdata_timestan
(zvt.contract.schema.N
(zvt.contract.schema.F
(zvt.contract.schema.T
(zvt.domain.meta.bloc
(zvt.domain.meta.cour
(zvt.domain.meta.curr
(zvt.domain.meta.etf_
(zvt.domain.meta.func
(zvt.domain.meta.futu
(zvt.domain.meta.inde
(zvt.domain.meta.inde
(zvt.domain.meta.stoc
(zvt.domain.meta.stoc
(zvt.domain.meta.stoc
(zvt.domain.meta.stoc
is_in_same_interval() (in m
is_limit_down (zvt.domain
(zvt.trading.trading_m
is_limit_up (zvt.domain.qu
(zvt.trading.trading_m
is_new (zvt.domain.emotio
(zvt.domain.emotion.e
is_normal_df() (in module
is_same_date() (in module
is_same_time() (in module
is_score_result_df() (in mo
is_up() (in module zvt.facto
issue_pe (zvt.domain.meta.
issues (zvt.domain.meta.stc
iterate_with_step() (in mod
iterencode() (zvt.factors.sha

J

JoinquantHkHolderRecorder (class in zvt.recorders.joinquant.misc.jq_hk_holder_recorder)

JqChinaFundStockRecorder (class in zvt.recorders.joinquant.meta.jq_fund_met

JoinquantIndexMoneyFlowRecorder (class in zvt.recorders.joinquant.misc.jq_index_money_flow_recorder)
JoinquantStockMoneyFlowRecorder (class in zvt.recorders.joinquant.misc.jq_stock_money_flow_recorder)
jq_to_report_period() (in module zvt.recorders.joinquant.common)
JqChinaEtfRecorder (class in zvt.recorders.joinquant.meta.jq_stock_meta_recorder)
JqChinaEtfValuationRecorder (class in zvt.recorders.joinquant.fundamental.jq_etf_valuation_recorder)
JqChinaFundRecorder (class in zvt.recorders.joinquant.meta.jq_fund_meta_recorder)
JqChinaIndexKdataRecorder (class in zvt.recorders.joinquant.quotes.jq_index_k
JqChinaStockEtfPortfolioRecorder (class
zvt.recorders.joinquant.meta.jq_stock_me
JqChinaStockKdataRecorder (class in zvt.recorders.joinquant.quotes.jq_stock_k
JqChinaStockRecorder (class in zvt.recorders.joinquant.meta.jq_stock_me
JqChinaStockValuationRecorder (class in zvt.recorders.joinquant.fundamental.jq_st
json_callback_param() (in module zvt.util

K

KdataCommon (class in zvt.domain.quotes)
KdataModel (class in zvt.factors.factor_models)
KdataRequestModel (class in zvt.factors.factor_models)
KeepBullFactor (class in zvt.factors.macd.macd_factor)
kline (zvt.contract.drawer.ChartType attribute)

L

last_limit_up_time (zvt.domain.emotion.emotion.LimitUpInfo attribute)
latitude (zvt.domain.meta.country_meta.Country attribute)
lending_type (zvt.domain.meta.country_meta.Country attribute)
level (zvt.domain.quotes.block.block_1d_kdata.Block1dKdata attribute)
(zvt.domain.quotes.block.block_1mon_kdata.Block1monKdata attribute)
(zvt.domain.quotes.block.block_1wk_kdata.Block1wkKdata attribute)
(zvt.domain.quotes.currency.currency_1d_kdata.Currency1dKdata attribute)
(zvt.domain.quotes.etf.etf_1d_kdata.Etf1dKdata attribute)
(zvt.domain.quotes.future.future_1d_kdata.Future1dKdata attribute)
(zvt.domain.quotes.index.index_1d_kdata.Index1dKdata attribute)
(zvt.domain.quotes.index.index_1wk_kdata.Index1wkKdata attribute)
(zvt.domain.quotes.indexus.indexus_1d_kdata.Indexus1dKdata attribute)
(zvt.domain.quotes.stock.stock_15m_hfq_kdata.Stock15mHfqKdata attribute)
(zvt.domain.quotes.stock.stock_15m_kdata.Stock15mKdata attribute)
(zvt.domain.quotes.stock.stock_1d_hfq_kdata.Stock1dHfqKdata attribute)
(zvt.domain.quotes.stock.stock_1d_kdata.Stock1dKdata attribute)
(zvt.domain.quotes.stock.stock_1h_hfq_kdata.Stock1hHfqKdata attribute)
(zvt.domain.quotes.stock.stock_1h_kdata.Stock1hKdata attribute)
(zvt.domain.quotes.stock.stock_1m_hfq_kdata.Stock1mHfqKdata attribute)
(zvt.domain.quotes.stock.stock_1m_kdata.Stock1mKdata attribute)
(zvt.domain.quotes.stock.stock_1mon_hfq_kdata.Stock1monHfqKdata attribute)
(zvt.domain.quotes.stock.stock_1mon_kdata.Stock1monKdata attribute)
list_date (zvt.contract.schem
(zvt.contract.schema.E
(zvt.contract.schema.N
(zvt.contract.schema.P
(zvt.contract.schema.T
(zvt.domain.actor.actor
(zvt.domain.meta.block
(zvt.domain.meta.coun
(zvt.domain.meta.curre
(zvt.domain.meta.etf_n
(zvt.domain.meta.fund_
(zvt.domain.meta.futur
(zvt.domain.meta.index
(zvt.domain.meta.index
(zvt.domain.meta.stock
(zvt.domain.meta.stock
(zvt.domain.meta.stock
(zvt.domain.meta.stock
live (zvt.factors.ma.domain.
attribute)
live_or_dead() (in module z
LiveOrDeadFactor (class in
load() (zvt.trader.sim_accou
(zvt.trader.sim_account
loads() (zvt.trader.sim_acco
(zvt.trader.sim_account
long_amount (zvt.trader.trad
long_term_borrowing (zvt.d
attribute)
long_term_equity_investme
attribute)

(zvt.domain.quotes.stock.stock_1wk_hfq_kdata.Stock1wkHfqKdata attribute)
(zvt.domain.quotes.stock.stock_1wk_kdata.Stock1wkKdata attribute)
(zvt.domain.quotes.stock.stock_30m_hfq_kdata.Stock30mHfqKdata attribute)
(zvt.domain.quotes.stock.stock_30m_kdata.Stock30mKdata attribute)
(zvt.domain.quotes.stock.stock_4h_hfq_kdata.Stock4hHfqKdata attribute)
(zvt.domain.quotes.stock.stock_4h_kdata.Stock4hKdata attribute)
(zvt.domain.quotes.stock.stock_5m_hfq_kdata.Stock5mHfqKdata attribute)
(zvt.domain.quotes.stock.stock_5m_kdata.Stock5mKdata attribute)
(zvt.domain.quotes.stockhk.stockhk_1d_hfq_kdata.Stockhk1dHfqKdata attribute)
(zvt.domain.quotes.stockhk.stockhk_1d_kdata.Stockhk1dKdata attribute)
(zvt.domain.quotes.stockus.stockus_1d_hfq_kdata.Stockus1dHfqKdata attribute)
(zvt.domain.quotes.stockus.stockus_1d_kdata.Stockus1dKdata attribute)
(zvt.factors.z.domain.stock_1d_z_factor.Stock1dZFactor attribute)
(zvt.factors.z.domain.stock_1wk_z_factor.Stock1wkZFactor attribute)
(zvt.trader.trader_schemas.Order attribute)
LEVEL_15MIN (zvt.contract.IntervalLevel attribute)
LEVEL_1DAY (zvt.contract.IntervalLevel attribute)
LEVEL_1HOUR (zvt.contract.IntervalLevel attribute)
LEVEL_1MIN (zvt.contract.IntervalLevel attribute)
LEVEL_1MON (zvt.contract.IntervalLevel attribute)
LEVEL_1WEEK (zvt.contract.IntervalLevel attribute)
LEVEL_30MIN (zvt.contract.IntervalLevel attribute)
LEVEL_4HOUR (zvt.contract.IntervalLevel attribute)
LEVEL_5MIN (zvt.contract.IntervalLevel attribute)
LEVEL_L2_QUOTE (zvt.contract.IntervalLevel attribute)
LEVEL_TICK (zvt.contract.IntervalLevel attribute)
limit_down_amount (zvt.domain.quotes.stock.stock_quote.StockQuote attribute)
(zvt.trading.trading_models.StockQuoteModel attribute)
limit_down_count (zvt.domain.emotion.emotion.Emotion attribute)
(zvt.trading.trading_models.StockQuoteStatsModel attribute)
(zvt.trading.trading_models.TagQuoteStatsModel attribute)
limit_down_open_count (zvt.domain.emotion.emotion.Emotion attribute)
limit_down_success_rate (zvt.domain.emotion.emotion.Emotion attribute)
limit_up_amount (zvt.domain.quotes.stock.stock_quote.StockQuote attribute)
(zvt.trading.trading_models.StockQuoteModel attribute)
limit_up_count (zvt.domain.emotion.emotion.Emotion attribute)
(zvt.trading.trading_models.StockQuoteStatsModel attribute)
(zvt.trading.trading_models.TagQuoteStatsModel attribute)
limit_up_open_count (zvt.domain.emotion.emotion.Emotion attribute)
limit_up_success_rate (zvt.domain.emotion.emotion.Emotion attribute)
limit_up_type (zvt.domain.emotion.emotion.LimitUpInfo attribute)
LimitDownInfo (class in zvt.domain.emotion.emotion)
LimitUpInfo (class in zvt.domain.emotion.emotion)
line (zvt.contract.drawer.ChartType attribute)
list_all_files() (in module zvt.utils.file_utils)
long_term_payable (zvt.don
long_term_prepaid_expense
attribute)
long_term_receivables (zvt.
attribute)
longitude (zvt.domain.meta.
loss_on_disposal_non_curre
(zvt.domain.fundamental.fin
low (zvt.domain.quotes.bloc
(zvt.domain.quotes.blo
attribute)
(zvt.domain.quotes.blo
(zvt.domain.quotes.cur
attribute)
(zvt.domain.quotes.etf.
(zvt.domain.quotes.futu
(zvt.domain.quotes.ind
(zvt.domain.quotes.ind
(zvt.domain.quotes.ind
attribute)
(zvt.domain.quotes.sto
attribute)
(zvt.domain.quotes.sto
(zvt.domain.quotes.sto
attribute)
(zvt.domain.quotes.sto
(zvt.domain.quotes.sto
attribute)
(zvt.domain.quotes.sto
(zvt.domain.quotes.sto
attribute)
(zvt.domain.quotes.sto
(zvt.domain.quotes.sto
attribute)
(zvt.domain.quotes.sto
(zvt.domain.quotes.sto
attribute)
(zvt.domain.quotes.sto
(zvt.domain.quotes.sto
attribute)
(zvt.domain.quotes.sto
(zvt.domain.quotes.sto
attribute)
(zvt.domain.quotes.sto
(zvt.domain.quotes.sto
attribute)
(zvt.domain.quotes.sto
(zvt.domain.quotes.sto
attribute)
(zvt.domain.quotes.sto
(zvt.domain.quotes.sto
attribute)
(zvt.domain.quotes.sto
(zvt.domain.quotes.sto
attribute)
(zvt.domain.quotes.sto
(zvt.domain.quotes.sto
attribute)
(zvt.domain.quotes.sto
(zvt.domain.quotes.Tic

(zvt.factors.ma.domain
attribute)
(zvt.factors.z.domain.st
(zvt.factors.z.domain.st

M

ma() (in module zvt.factors.algorithm)
ma10
(zvt.factors.ma.domain.stock_1d_ma_stats_factor.Stock1dMaStatsFactor
attribute)
ma120
(zvt.factors.ma.domain.stock_1d_ma_stats_factor.Stock1dMaStatsFactor
attribute)
ma144
(zvt.factors.ma.domain.stock_1d_ma_stats_factor.Stock1dMaStatsFactor
attribute)
ma250
(zvt.factors.ma.domain.stock_1d_ma_stats_factor.Stock1dMaStatsFactor
attribute)
ma34
(zvt.factors.ma.domain.stock_1d_ma_stats_factor.Stock1dMaStatsFactor
attribute)
ma5
(zvt.factors.ma.domain.stock_1d_ma_stats_factor.Stock1dMaStatsFactor
attribute)
ma55
(zvt.factors.ma.domain.stock_1d_ma_stats_factor.Stock1dMaStatsFactor
attribute)
ma89
(zvt.factors.ma.domain.stock_1d_ma_stats_factor.Stock1dMaStatsFactor
attribute)
MaAndVolumeTransformer (class in zvt.factors.algorithm)
macd() (in module zvt.factors.algorithm)
MacdFactor (class in zvt.factors.macd.macd_factor)
MacdTransformer (class in zvt.factors.algorithm)
MaFactor (class in zvt.factors.ma.ma_factor)
main() (in module zvt.main)
(in module zvt.plugin)
main_business (zvt.domain.meta.stock_meta.StockDetail attribute)
MainTagInfo (class in zvt.tag.tag_schemas)
make_instance() (zvt.trader.sim_account.AccountDayStatsSchema
method)
(zvt.trader.sim_account.PositionSchema method)
manager (zvt.domain.fundamental.trading.ManagerTrading attribute)
manager_position (zvt.domain.fundamental.trading.ManagerTrading
attribute)
ManagerTrading (class in zvt.domain.fundamental.trading)
ManagerTradingRecorder (class in
zvt.recorders.eastmoney.trading.eastmoney_manager_trading_recorder)
managing_costs (zvt.domain.fundamental.finance.IncomeStatement
attribute)
margin_buy (zvt.domain.misc.overall.MarginTradingSummary attribute)
margin_value (zvt.domain.misc.overall.MarginTradingSummary
attribute)
model_rebuild() (zvt.common.q
(zvt.common.query_model
(zvt.common.query_model
(zvt.common.trading_mode
(zvt.common.trading_mode
(zvt.common.trading_mode
(zvt.contract.model.Custom
(zvt.contract.model.MixinM
(zvt.factors.factor_models.
(zvt.factors.factor_models.
(zvt.factors.factor_models.
(zvt.factors.factor_models.
(zvt.factors.factor_models.
(zvt.tag.tag_models.Activa
(zvt.tag.tag_models.Activa
(zvt.tag.tag_models.BatchS
(zvt.tag.tag_models.Create
(zvt.tag.tag_models.Create
(zvt.tag.tag_models.Create
(zvt.tag.tag_models.Query
(zvt.tag.tag_models.Query
(zvt.tag.tag_models.Query
(zvt.tag.tag_models.SetSto
(zvt.tag.tag_models.Simple
(zvt.tag.tag_models.StockP
(zvt.tag.tag_models.StockP
(zvt.tag.tag_models.StockP
(zvt.tag.tag_models.StockT
(zvt.tag.tag_models.StockT
(zvt.tag.tag_models.StockT
(zvt.tag.tag_models.TagInf
(zvt.tag.tag_models.TagPar
(zvt.trading.trading_models
(zvt.trading.trading_models
(zvt.trading.trading_models
(zvt.trading.trading_models
(zvt.trading.trading_models
(zvt.trading.trading_models
(zvt.trading.trading_models
(zvt.trading.trading_models
(zvt.trading.trading_models
(zvt.trading.trading_models
model_validate() (zvt.common.c
(zvt.common.query_model
(zvt.common.query_model
(zvt.common.trading_mode
(zvt.common.trading_mode
(zvt.common.trading_mode

MarginTrading (class in zvt.domain.fundamental.trading)
MarginTradingRecorder (class in zvt.recorders.joinquant.fundamental.jq_margin_trading_recorder)
MarginTradingSummary (class in zvt.domain.misc.overall)
MarginTradingSummaryRecorder (class in zvt.recorders.joinquant.overall.jq_margin_trading_recorder)
market_cap (zvt.contract.schema.PortfolioStockHistory attribute)
(zvt.domain.fundamental.valuation.StockValuation attribute)
(zvt.domain.meta.etf_meta.EtfStock attribute)
(zvt.domain.meta.fund_meta.FundStock attribute)
(zvt.domain.meta.index_meta.IndexStock attribute)
marshal_object_for_ui() (in module zvt.utils.utils)
MaStatsAccumulator (class in zvt.factors.ma.ma_stats_factor)
MaStatsFactor (class in zvt.factors.ma.ma_stats_factor)
MaStatsFactorCommon (class in zvt.factors.ma.domain.common)
MaStockMLMachine (class in zvt.ml.ml)
match_tag() (in module zvt.tag.tag_utils)
match_tag_by_type() (in module zvt.tag.tag_utils)
MaTransformer (class in zvt.factors.algorithm)
max_height (zvt.domain.emotion.emotion.Emotion attribute)
merge_filter_result() (in module zvt.utils.pd_utils)
Meta (class in zvt.contract.recorder)
Mixin (class in zvt.contract.schema)
MixinModel (class in zvt.contract.model)
MLMachine (class in zvt.ml.ml)
model (zvt.trader.sim_account.AccountDayStatsSchema.Meta attribute)
(zvt.trader.sim_account.PositionSchema.Meta attribute)
model_computed_fields
(zvt.common.query_models.AbsoluteTimeRange attribute)
(zvt.common.query_models.RelativeTimeRage attribute)
(zvt.common.query_models.TimeRange attribute)
(zvt.common.trading_models.BuyParameter attribute)
(zvt.common.trading_models.SellParameter attribute)
(zvt.common.trading_models.TradingResult attribute)
(zvt.contract.model.CustomModel attribute)
(zvt.contract.model.MixinModel attribute)
(zvt.factors.factor_models.FactorRequestModel attribute)
(zvt.factors.factor_models.FactorResultModel attribute)
(zvt.factors.factor_models.KdataModel attribute)
(zvt.factors.factor_models.KdataRequestModel attribute)
(zvt.factors.factor_models.TradingSignalModel attribute)
(zvt.tag.tag_models.ActivateSubTagsModel attribute)
(zvt.tag.tag_models.ActivateSubTagsResultModel attribute)
(zvt.tag.tag_models.BatchSetStockTagsModel attribute)
(zvt.tag.tag_models.CreateStockPoolInfoModel attribute)
(zvt.tag.tag_models.CreateStockPoolsModel attribute)
(zvt.tag.tag_models.CreateTagInfoModel attribute)
(zvt.tag.tag_models.QuerySimpleStockTagsModel attribute)
(zvt.tag.tag_models.QueryStockTagsModel attribute)
(zvt.tag.tag_models.QueryStockTagStatsModel attribute)
(zvt.tag.tag_models.SetStockTagsModel attribute)
(zvt.tag.tag_models.SimpleStockTagsModel attribute)
(zvt.tag.tag_models.StockPoolInfoModel attribute)
(zvt.tag.tag_models.StockPoolModel attribute)
(zvt.tag.tag_models.StockPoolsModel attribute)
(zvt.tag.tag_models.StockTagDetailsModel attribute)
(zvt.contract.model.Custom
(zvt.contract.model.MixinM
(zvt.factors.factor_models.
(zvt.factors.factor_models.
(zvt.factors.factor_models.
(zvt.factors.factor_models.
(zvt.factors.factor_models.
(zvt.tag.tag_models.Activa
(zvt.tag.tag_models.Activa
(zvt.tag.tag_models.BatchS
(zvt.tag.tag_models.Create
(zvt.tag.tag_models.Create
(zvt.tag.tag_models.Create
(zvt.tag.tag_models.Query
(zvt.tag.tag_models.Query
(zvt.tag.tag_models.Query
(zvt.tag.tag_models.SetSto
(zvt.tag.tag_models.Simple
(zvt.tag.tag_models.StockP
(zvt.tag.tag_models.StockP
(zvt.tag.tag_models.StockP
(zvt.tag.tag_models.StockT
(zvt.tag.tag_models.StockT
(zvt.tag.tag_models.StockT
(zvt.tag.tag_models.TagInf
(zvt.tag.tag_models.TagPar
(zvt.trading.trading_model
(zvt.trading.trading_model
(zvt.trading.trading_model
(zvt.trading.trading_model
(zvt.trading.trading_model
(zvt.trading.trading_model
(zvt.trading.trading_model
(zvt.trading.trading_model
(zvt.trading.trading_model
model_validate_json() (zvt.com
(zvt.common.query_model
(zvt.common.query_model
(zvt.common.trading_mod
(zvt.common.trading_mod
(zvt.common.trading_mod
(zvt.contract.model.Custom
(zvt.contract.model.MixinM
(zvt.factors.factor_models.
(zvt.factors.factor_models.
(zvt.factors.factor_models.
(zvt.factors.factor_models.
(zvt.factors.factor_models.
(zvt.tag.tag_models.Activa
(zvt.tag.tag_models.Activa
(zvt.tag.tag_models.BatchS
(zvt.tag.tag_models.Create
(zvt.tag.tag_models.Create
(zvt.tag.tag_models.Create
(zvt.tag.tag_models.Query

(zvt.tag.tag_models.StockTagsModel attribute)
(zvt.tag.tag_models.StockTagStatsModel attribute)
(zvt.tag.tag_models.TagInfoModel attribute)
(zvt.tag.tag_models.TagParameter attribute)
(zvt.trading.trading_models.BuildQueryStockQuoteSettingModel attribute)
(zvt.trading.trading_models.BuildTradingPlanModel attribute)
(zvt.trading.trading_models.QueryStockQuoteModel attribute)
(zvt.trading.trading_models.QueryStockQuoteSettingModel attribute)
(zvt.trading.trading_models.QueryTagQuoteModel attribute)
(zvt.trading.trading_models.QueryTradingPlanModel attribute)
(zvt.trading.trading_models.StockQuoteModel attribute)
(zvt.trading.trading_models.StockQuoteStatsModel attribute)
(zvt.trading.trading_models.TagQuoteStatsModel attribute)
(zvt.trading.trading_models.TradingPlanModel attribute)

model_config (zvt.common.query_models.AbsoluteTimeRange attribute)
(zvt.common.query_models.RelativeTimeRage attribute)
(zvt.common.query_models.TimeRange attribute)
(zvt.common.trading_models.BuyParameter attribute)
(zvt.common.trading_models.SellParameter attribute)
(zvt.common.trading_models.TradingResult attribute)
(zvt.contract.model.CustomModel attribute)
(zvt.contract.model.MixinModel attribute)
(zvt.factors.factor_models.FactorRequestModel attribute)
(zvt.factors.factor_models.FactorResultModel attribute)
(zvt.factors.factor_models.KdataModel attribute)
(zvt.factors.factor_models.KdataRequestModel attribute)
(zvt.factors.factor_models.TradingSignalModel attribute)
(zvt.tag.tag_models.ActivateSubTagsModel attribute)
(zvt.tag.tag_models.ActivateSubTagsResultModel attribute)
(zvt.tag.tag_models.BatchSetStockTagsModel attribute)
(zvt.tag.tag_models.CreateStockPoolInfoModel attribute)
(zvt.tag.tag_models.CreateStockPoolsModel attribute)
(zvt.tag.tag_models.CreateTagInfoModel attribute)
(zvt.tag.tag_models.QuerySimpleStockTagsModel attribute)
(zvt.tag.tag_models.QueryStockTagsModel attribute)
(zvt.tag.tag_models.QueryStockTagStatsModel attribute)
(zvt.tag.tag_models.SetStockTagsModel attribute)
(zvt.tag.tag_models.SimpleStockTagsModel attribute)
(zvt.tag.tag_models.StockPoolInfoModel attribute)
(zvt.tag.tag_models.StockPoolModel attribute)
(zvt.tag.tag_models.StockPoolsModel attribute)
(zvt.tag.tag_models.StockTagDetailsModel attribute)
(zvt.tag.tag_models.StockTagsModel attribute)
(zvt.tag.tag_models.StockTagStatsModel attribute)
(zvt.tag.tag_models.TagInfoModel attribute)
(zvt.tag.tag_models.TagParameter attribute)
(zvt.trading.trading_models.BuildQueryStockQuoteSettingModel attribute)
(zvt.trading.trading_models.BuildTradingPlanModel attribute)
(zvt.trading.trading_models.QueryStockQuoteModel attribute)
(zvt.trading.trading_models.QueryStockQuoteSettingModel attribute)
(zvt.trading.trading_models.QueryTagQuoteModel attribute)

(zvt.tag.tag_models.Query
(zvt.tag.tag_models.Query
(zvt.tag.tag_models.SetSto
(zvt.tag.tag_models.Simple
(zvt.tag.tag_models.StockP
(zvt.tag.tag_models.StockP
(zvt.tag.tag_models.StockP
(zvt.tag.tag_models.StockT
(zvt.tag.tag_models.StockT
(zvt.tag.tag_models.StockT
(zvt.tag.tag_models.TagInf
(zvt.tag.tag_models.TagPar
(zvt.trading.trading_models
(zvt.trading.trading_models
(zvt.trading.trading_models
(zvt.trading.trading_models
(zvt.trading.trading_models
(zvt.trading.trading_models
(zvt.trading.trading_models
(zvt.trading.trading_models
(zvt.trading.trading_models
(zvt.trading.trading_models
(zvt.trading.trading_models

model_validate_strings() (zvt.co
(zvt.common.query_model
(zvt.common.query_model
(zvt.common.trading_mode
(zvt.common.trading_mode
(zvt.common.trading_mode
(zvt.contract.model.Custom
(zvt.contract.model.MixinM
(zvt.factors.factor_models.
(zvt.factors.factor_models.
(zvt.factors.factor_models.
(zvt.factors.factor_models.
(zvt.factors.factor_models.
(zvt.tag.tag_models.Activa
(zvt.tag.tag_models.Activa
(zvt.tag.tag_models.BatchS
(zvt.tag.tag_models.Create
(zvt.tag.tag_models.Create
(zvt.tag.tag_models.Create
(zvt.tag.tag_models.Query
(zvt.tag.tag_models.Query
(zvt.tag.tag_models.Query
(zvt.tag.tag_models.SetSto
(zvt.tag.tag_models.Simple
(zvt.tag.tag_models.StockP
(zvt.tag.tag_models.StockP
(zvt.tag.tag_models.StockP
(zvt.tag.tag_models.StockT
(zvt.tag.tag_models.StockT
(zvt.tag.tag_models.StockT
(zvt.tag.tag_models.TagInf
(zvt.tag.tag_models.TagPar
(zvt.trading.trading_models
(zvt.trading.trading_models

(zvt.trading.trading_models.QueryTradingPlanModel attribute)
(zvt.trading.trading_models.StockQuoteModel attribute)
(zvt.trading.trading_models.StockQuoteStatsModel attribute)
(zvt.trading.trading_models.TagQuoteStatsModel attribute)
(zvt.trading.trading_models.TradingPlanModel attribute)
model_construct() (zvt.common.query_models.AbsoluteTimeRange class method)
(zvt.common.query_models.RelativeTimeRage class method)
(zvt.common.query_models.TimeRange class method)
(zvt.common.trading_models.BuyParameter class method)
(zvt.common.trading_models.SellParameter class method)
(zvt.common.trading_models.TradingResult class method)
(zvt.contract.model.CustomModel class method)
(zvt.contract.model.MixinModel class method)
(zvt.factors.factor_models.FactorRequestModel class method)
(zvt.factors.factor_models.FactorResultModel class method)
(zvt.factors.factor_models.KdataModel class method)
(zvt.factors.factor_models.KdataRequestModel class method)
(zvt.factors.factor_models.TradingSignalModel class method)
(zvt.tag.tag_models.ActivateSubTagsModel class method)
(zvt.tag.tag_models.ActivateSubTagsResultModel class method)
(zvt.tag.tag_models.BatchSetStockTagsModel class method)
(zvt.tag.tag_models.CreateStockPoolInfoModel class method)
(zvt.tag.tag_models.CreateStockPoolsModel class method)
(zvt.tag.tag_models.CreateTagInfoModel class method)
(zvt.tag.tag_models.QuerySimpleStockTagsModel class method)
(zvt.tag.tag_models.QueryStockTagsModel class method)
(zvt.tag.tag_models.QueryStockTagStatsModel class method)
(zvt.tag.tag_models.SetStockTagsModel class method)
(zvt.tag.tag_models.SimpleStockTagsModel class method)
(zvt.tag.tag_models.StockPoolInfoModel class method)
(zvt.tag.tag_models.StockPoolModel class method)
(zvt.tag.tag_models.StockPoolsModel class method)
(zvt.tag.tag_models.StockTagDetailsModel class method)
(zvt.tag.tag_models.StockTagsModel class method)
(zvt.tag.tag_models.StockTagStatsModel class method)
(zvt.tag.tag_models.TagInfoModel class method)
(zvt.tag.tag_models.TagParameter class method)
(zvt.trading.trading_models.BuildQueryStockQuoteSettingModel class method)
(zvt.trading.trading_models.BuildTradingPlanModel class method)
(zvt.trading.trading_models.QueryStockQuoteModel class method)
(zvt.trading.trading_models.QueryStockQuoteSettingModel class method)
(zvt.trading.trading_models.QueryTagQuoteModel class method)
(zvt.trading.trading_models.QueryTradingPlanModel class method)
(zvt.trading.trading_models.StockQuoteModel class method)
(zvt.trading.trading_models.StockQuoteStatsModel class method)
(zvt.trading.trading_models.TagQuoteStatsModel class method)
(zvt.trading.trading_models.TradingPlanModel class method)
model_copy() (zvt.common.query_models.AbsoluteTimeRange method)
(zvt.common.query_models.RelativeTimeRage method)
(zvt.common.query_models.TimeRange method)
(zvt.common.trading_models.BuyParameter method)
(zvt.common.trading_models.SellParameter method)
(zvt.common.trading_models.TradingResult method)
(zvt.trading.trading_models
(zvt.trading.trading_models
(zvt.trading.trading_models
(zvt.trading.trading_models
(zvt.trading.trading_models
(zvt.trading.trading_models
(zvt.trading.trading_models
(zvt.trading.trading_models
module
zvt
zvt.api
zvt.api.intent
zvt.api.kdata
zvt.api.portfolio
zvt.api.selector
zvt.api.stats
zvt.api.utils
zvt.autocode
zvt.autocode.generator
zvt.autocode.templates
zvt.broker
zvt.broker.qmt
zvt.broker.qmt.errors
zvt.common
zvt.common.query_models
zvt.common.trading_mode
zvt.consts
zvt.contract
zvt.contract.api
zvt.contract.base_service
zvt.contract.context
zvt.contract.data_type
zvt.contract.drawer
zvt.contract.factor
zvt.contract.model
zvt.contract.normal_data
zvt.contract.reader
zvt.contract.recorder
zvt.contract.register
zvt.contract.schema
zvt.contract.zvt_info
zvt.domain
zvt.domain.actor
zvt.domain.actor.actor_met
zvt.domain.actor.stock_act
zvt.domain.emotion
zvt.domain.emotion.emotic
zvt.domain.fundamental
zvt.domain.fundamental.di
zvt.domain.fundamental.fir
zvt.domain.fundamental.tra
zvt.domain.fundamental.va
zvt.domain.macro
zvt.domain.macro.macro
zvt.domain.macro.monetar
zvt.domain.meta

(zvt.contract.model.CustomModel method)
(zvt.contract.model.MixinModel method)
(zvt.factors.factor_models.FactorRequestModel method)
(zvt.factors.factor_models.FactorResultModel method)
(zvt.factors.factor_models.KdataModel method)
(zvt.factors.factor_models.KdataRequestModel method)
(zvt.factors.factor_models.TradingSignalModel method)
(zvt.tag.tag_models.ActivateSubTagsModel method)
(zvt.tag.tag_models.ActivateSubTagsResultModel method)
(zvt.tag.tag_models.BatchSetStockTagsModel method)
(zvt.tag.tag_models.CreateStockPoolInfoModel method)
(zvt.tag.tag_models.CreateStockPoolsModel method)
(zvt.tag.tag_models.CreateTagInfoModel method)
(zvt.tag.tag_models.QuerySimpleStockTagsModel method)
(zvt.tag.tag_models.QueryStockTagsModel method)
(zvt.tag.tag_models.QueryStockTagStatsModel method)
(zvt.tag.tag_models.SetStockTagsModel method)
(zvt.tag.tag_models.SimpleStockTagsModel method)
(zvt.tag.tag_models.StockPoolInfoModel method)
(zvt.tag.tag_models.StockPoolModel method)
(zvt.tag.tag_models.StockPoolsModel method)
(zvt.tag.tag_models.StockTagDetailsModel method)
(zvt.tag.tag_models.StockTagsModel method)
(zvt.tag.tag_models.StockTagStatsModel method)
(zvt.tag.tag_models.TagInfoModel method)
(zvt.tag.tag_models.TagParameter method)
(zvt.trading.trading_models.BuildQueryStockQuoteSettingModel method)
(zvt.trading.trading_models.BuildTradingPlanModel method)
(zvt.trading.trading_models.QueryStockQuoteModel method)
(zvt.trading.trading_models.QueryStockQuoteSettingModel method)
(zvt.trading.trading_models.QueryTagQuoteModel method)
(zvt.trading.trading_models.QueryTradingPlanModel method)
(zvt.trading.trading_models.StockQuoteModel method)
(zvt.trading.trading_models.StockQuoteStatsModel method)
(zvt.trading.trading_models.TagQuoteStatsModel method)
(zvt.trading.trading_models.TradingPlanModel method)
model_dump() (zvt.common.query_models.AbsoluteTimeRange method)
(zvt.common.query_models.RelativeTimeRage method)
(zvt.common.query_models.TimeRange method)
(zvt.common.trading_models.BuyParameter method)
(zvt.common.trading_models.SellParameter method)
(zvt.common.trading_models.TradingResult method)
(zvt.contract.model.CustomModel method)
(zvt.contract.model.MixinModel method)
(zvt.factors.factor_models.FactorRequestModel method)
(zvt.factors.factor_models.FactorResultModel method)
(zvt.factors.factor_models.KdataModel method)
(zvt.factors.factor_models.KdataRequestModel method)
(zvt.factors.factor_models.TradingSignalModel method)
(zvt.tag.tag_models.ActivateSubTagsModel method)
(zvt.tag.tag_models.ActivateSubTagsResultModel method)
(zvt.tag.tag_models.BatchSetStockTagsModel method)
(zvt.tag.tag_models.CreateStockPoolInfoModel method)

zvt.domain.meta.block_me
zvt.domain.meta.country_n
zvt.domain.meta.currency_
zvt.domain.meta.etf_meta
zvt.domain.meta.fund_meta
zvt.domain.meta.future_me
zvt.domain.meta.index_me
zvt.domain.meta.indexus_n
zvt.domain.meta.stock_met
zvt.domain.meta.stockhk_n
zvt.domain.meta.stockus_n
zvt.domain.misc
zvt.domain.misc.holder
zvt.domain.misc.money_flo
zvt.domain.misc.overall
zvt.domain.misc.stock_new
zvt.domain.quotes
zvt.domain.quotes.block
zvt.domain.quotes.block.bl
zvt.domain.quotes.block.bl
zvt.domain.quotes.block.bl
zvt.domain.quotes.currency
zvt.domain.quotes.currency
zvt.domain.quotes.etf
zvt.domain.quotes.etf.etf_1
zvt.domain.quotes.future
zvt.domain.quotes.future.fu
zvt.domain.quotes.index
zvt.domain.quotes.index.in
zvt.domain.quotes.index.in
zvt.domain.quotes.indexus
zvt.domain.quotes.indexus.
zvt.domain.quotes.stock
zvt.domain.quotes.stock.sto
zvt.domain.quotes.stock.sto
zvt.domain.quotes.stock.sto
zvt.domain.quotes.stock.sto
zvt.domain.quotes.stock.sto
zvt.domain.quotes.stock.sto
zvt.domain.quotes.stock.sto
zvt.domain.quotes.stock.sto
zvt.domain.quotes.stock.sto
zvt.domain.quotes.stock.sto
zvt.domain.quotes.stock.sto
zvt.domain.quotes.stock.sto
zvt.domain.quotes.stock.sto
zvt.domain.quotes.stock.sto
zvt.domain.quotes.stock.sto
zvt.domain.quotes.stock.sto
zvt.domain.quotes.stock.sto
zvt.domain.quotes.stock.sto
zvt.domain.quotes.stock.sto
zvt.domain.quotes.stock.sto
zvt.domain.quotes.stock.sto
zvt.domain.quotes.stock.sto
zvt.domain.quotes.stock.sto
zvt.domain.quotes.stockhk
zvt.domain.quotes.stockhk.
zvt.domain.quotes.stockhk.
zvt.domain.quotes.stockus

(zvt.tag.tag_models.CreateStockPoolsModel method)
(zvt.tag.tag_models.CreateTagInfoModel method)
(zvt.tag.tag_models.QuerySimpleStockTagsModel method)
(zvt.tag.tag_models.QueryStockTagsModel method)
(zvt.tag.tag_models.QueryStockTagStatsModel method)
(zvt.tag.tag_models.SetStockTagsModel method)
(zvt.tag.tag_models.SimpleStockTagsModel method)
(zvt.tag.tag_models.StockPoolInfoModel method)
(zvt.tag.tag_models.StockPoolModel method)
(zvt.tag.tag_models.StockPoolsModel method)
(zvt.tag.tag_models.StockTagDetailsModel method)
(zvt.tag.tag_models.StockTagsModel method)
(zvt.tag.tag_models.StockTagStatsModel method)
(zvt.tag.tag_models.TagInfoModel method)
(zvt.tag.tag_models.TagParameter method)
(zvt.trading.trading_models.BuildQueryStockQuoteSettingModel method)
(zvt.trading.trading_models.BuildTradingPlanModel method)
(zvt.trading.trading_models.QueryStockQuoteModel method)
(zvt.trading.trading_models.QueryStockQuoteSettingModel method)
(zvt.trading.trading_models.QueryTagQuoteModel method)
(zvt.trading.trading_models.QueryTradingPlanModel method)
(zvt.trading.trading_models.StockQuoteModel method)
(zvt.trading.trading_models.StockQuoteStatsModel method)
(zvt.trading.trading_models.TagQuoteStatsModel method)
(zvt.trading.trading_models.TradingPlanModel method)
model_dump_json() (zvt.common.query_models.AbsoluteTimeRange method)
(zvt.common.query_models.RelativeTimeRage method)
(zvt.common.query_models.TimeRange method)
(zvt.common.trading_models.BuyParameter method)
(zvt.common.trading_models.SellParameter method)
(zvt.common.trading_models.TradingResult method)
(zvt.contract.model.CustomModel method)
(zvt.contract.model.MixinModel method)
(zvt.factors.factor_models.FactorRequestModel method)
(zvt.factors.factor_models.FactorResultModel method)
(zvt.factors.factor_models.KdataModel method)
(zvt.factors.factor_models.KdataRequestModel method)
(zvt.factors.factor_models.TradingSignalModel method)
(zvt.tag.tag_models.ActivateSubTagsModel method)
(zvt.tag.tag_models.ActivateSubTagsResultModel method)
(zvt.tag.tag_models.BatchSetStockTagsModel method)
(zvt.tag.tag_models.CreateStockPoolInfoModel method)
(zvt.tag.tag_models.CreateStockPoolsModel method)
(zvt.tag.tag_models.CreateTagInfoModel method)
(zvt.tag.tag_models.QuerySimpleStockTagsModel method)
(zvt.tag.tag_models.QueryStockTagsModel method)
(zvt.tag.tag_models.QueryStockTagStatsModel method)
(zvt.tag.tag_models.SetStockTagsModel method)
(zvt.tag.tag_models.SimpleStockTagsModel method)
(zvt.tag.tag_models.StockPoolInfoModel method)
(zvt.tag.tag_models.StockPoolModel method)
(zvt.tag.tag_models.StockPoolsModel method)
(zvt.tag.tag_models.StockTagDetailsModel method)

zvt.domain.quotes.stockus.
zvt.domain.quotes.stockus.
zvt.domain.quotes.trade_da
zvt.factors
zvt.factors.algorithm
zvt.factors.factor_models
zvt.factors.factor_service
zvt.factors.fundamental
zvt.factors.fundamental.fin
zvt.factors.ma
zvt.factors.ma.domain
zvt.factors.ma.domain.com
zvt.factors.ma.domain.stoc
zvt.factors.ma.domain.stoc
zvt.factors.ma.ma_factor
zvt.factors.ma.ma_stats_fa
zvt.factors.ma.top_bottom_
zvt.factors.macd
zvt.factors.macd.macd_fact
zvt.factors.shape
zvt.factors.target_selector
zvt.factors.technical_factor
zvt.factors.top_stocks
zvt.factors.transformers
zvt.factors.z
zvt.factors.z.domain
zvt.factors.z.domain.comm
zvt.factors.z.domain.stock_
zvt.factors.z.domain.stock_
zvt.factors.z.z_factor
zvt.fill_project
zvt.informer
zvt.informer.inform_utils
zvt.informer.informer
zvt.main
zvt.ml
zvt.ml.lables
zvt.ml.ml
zvt.plugin
zvt.recorders
zvt.recorders.consts
zvt.recorders.eastmoney
zvt.recorders.eastmoney.co
zvt.recorders.eastmoney.div
zvt.recorders.eastmoney.div
zvt.recorders.eastmoney.div
zvt.recorders.eastmoney.div
zvt.recorders.eastmoney.div
zvt.recorders.eastmoney.fin
zvt.recorders.eastmoney.fin
zvt.recorders.eastmoney.fin
zvt.recorders.eastmoney.fin
zvt.recorders.eastmoney.fin
zvt.recorders.eastmoney.fin
zvt.recorders.eastmoney.ho
zvt.recorders.eastmoney.ho

(zvt.tag.tag_models.StockTagsModel method)
(zvt.tag.tag_models.StockTagStatsModel method)
(zvt.tag.tag_models.TagInfoModel method)
(zvt.tag.tag_models.TagParameter method)
(zvt.trading.trading_models.BuildQueryStockQuoteSettingModel method)
(zvt.trading.trading_models.BuildTradingPlanModel method)
(zvt.trading.trading_models.QueryStockQuoteModel method)
(zvt.trading.trading_models.QueryStockQuoteSettingModel method)
(zvt.trading.trading_models.QueryTagQuoteModel method)
(zvt.trading.trading_models.QueryTradingPlanModel method)
(zvt.trading.trading_models.StockQuoteModel method)
(zvt.trading.trading_models.StockQuoteStatsModel method)
(zvt.trading.trading_models.TagQuoteStatsModel method)
(zvt.trading.trading_models.TradingPlanModel method)
model_extra (zvt.common.query_models.AbsoluteTimeRange property)
(zvt.common.query_models.RelativeTimeRage property)
(zvt.common.query_models.TimeRange property)
(zvt.common.trading_models.BuyParameter property)
(zvt.common.trading_models.SellParameter property)
(zvt.common.trading_models.TradingResult property)
(zvt.contract.model.CustomModel property)
(zvt.contract.model.MixinModel property)
(zvt.factors.factor_models.FactorRequestModel property)
(zvt.factors.factor_models.FactorResultModel property)
(zvt.factors.factor_models.KdataModel property)
(zvt.factors.factor_models.KdataRequestModel property)
(zvt.factors.factor_models.TradingSignalModel property)
(zvt.tag.tag_models.ActivateSubTagsModel property)
(zvt.tag.tag_models.ActivateSubTagsResultModel property)
(zvt.tag.tag_models.BatchSetStockTagsModel property)
(zvt.tag.tag_models.CreateStockPoolInfoModel property)
(zvt.tag.tag_models.CreateStockPoolsModel property)
(zvt.tag.tag_models.CreateTagInfoModel property)
(zvt.tag.tag_models.QuerySimpleStockTagsModel property)
(zvt.tag.tag_models.QueryStockTagsModel property)
(zvt.tag.tag_models.QueryStockTagStatsModel property)
(zvt.tag.tag_models.SetStockTagsModel property)
(zvt.tag.tag_models.SimpleStockTagsModel property)
(zvt.tag.tag_models.StockPoolInfoModel property)
(zvt.tag.tag_models.StockPoolModel property)
(zvt.tag.tag_models.StockPoolsModel property)
(zvt.tag.tag_models.StockTagDetailsModel property)
(zvt.tag.tag_models.StockTagsModel property)
(zvt.tag.tag_models.StockTagStatsModel property)
(zvt.tag.tag_models.TagInfoModel property)
(zvt.tag.tag_models.TagParameter property)
(zvt.trading.trading_models.BuildQueryStockQuoteSettingModel property)
(zvt.trading.trading_models.BuildTradingPlanModel property)
(zvt.trading.trading_models.QueryStockQuoteModel property)
(zvt.trading.trading_models.QueryStockQuoteSettingModel property)
(zvt.trading.trading_models.QueryTagQuoteModel property)
(zvt.trading.trading_models.QueryTradingPlanModel property)
zvt.recorders.eastmoney.ho
zvt.recorders.eastmoney.ho
zvt.recorders.eastmoney.me
zvt.recorders.eastmoney.me
zvt.recorders.eastmoney.me
zvt.recorders.eastmoney.tra
zvt.recorders.eastmoney.tra
zvt.recorders.eastmoney.tra
zvt.recorders.em
zvt.recorders.em.actor
zvt.recorders.em.actor.em_
zvt.recorders.em.actor.em_
zvt.recorders.em.actor.em_
zvt.recorders.em.actor.em_
zvt.recorders.em.em_api
zvt.recorders.em.macro
zvt.recorders.em.macro.em
zvt.recorders.em.meta
zvt.recorders.em.meta.em_
zvt.recorders.em.meta.em_
zvt.recorders.em.meta.em_
zvt.recorders.em.meta.em_
zvt.recorders.em.meta.em_
zvt.recorders.em.meta.em_
zvt.recorders.em.meta.em_
zvt.recorders.em.meta.em_
zvt.recorders.em.misc
zvt.recorders.em.misc.em_
zvt.recorders.em.quotes
zvt.recorders.em.quotes.em
zvt.recorders.em.trading
zvt.recorders.em.trading.en
zvt.recorders.exchange
zvt.recorders.exchange.api
zvt.recorders.exchange.api.
zvt.recorders.exchange.api.
zvt.recorders.exchange.api.
zvt.recorders.exchange.api.
zvt.recorders.exchange.exc
zvt.recorders.exchange.exc
zvt.recorders.exchange.exc
zvt.recorders.exchange.exc
zvt.recorders.exchange.exc
zvt.recorders.joinquant
zvt.recorders.joinquant.con
zvt.recorders.joinquant.fun
zvt.recorders.joinquant.fun
zvt.recorders.joinquant.fun
zvt.recorders.joinquant.fun
zvt.recorders.joinquant.met
zvt.recorders.joinquant.met
zvt.recorders.joinquant.met
zvt.recorders.joinquant.met
zvt.recorders.joinquant.mis
zvt.recorders.joinquant.mis
zvt.recorders.joinquant.mis

(zvt.trading.trading_models.StockQuoteModel property)
(zvt.trading.trading_models.StockQuoteStatsModel property)
(zvt.trading.trading_models.TagQuoteStatsModel property)
(zvt.trading.trading_models.TradingPlanModel property)
model_fields (zvt.common.query_models.AbsoluteTimeRange attribute)
(zvt.common.query_models.RelativeTimeRage attribute)
(zvt.common.query_models.TimeRange attribute)
(zvt.common.trading_models.BuyParameter attribute)
(zvt.common.trading_models.SellParameter attribute)
(zvt.common.trading_models.TradingResult attribute)
(zvt.contract.model.CustomModel attribute)
(zvt.contract.model.MixinModel attribute)
(zvt.factors.factor_models.FactorRequestModel attribute)
(zvt.factors.factor_models.FactorResultModel attribute)
(zvt.factors.factor_models.KdataModel attribute)
(zvt.factors.factor_models.KdataRequestModel attribute)
(zvt.factors.factor_models.TradingSignalModel attribute)
(zvt.tag.tag_models.ActivateSubTagsModel attribute)
(zvt.tag.tag_models.ActivateSubTagsResultModel attribute)
(zvt.tag.tag_models.BatchSetStockTagsModel attribute)
(zvt.tag.tag_models.CreateStockPoolInfoModel attribute)
(zvt.tag.tag_models.CreateStockPoolsModel attribute)
(zvt.tag.tag_models.CreateTagInfoModel attribute)
(zvt.tag.tag_models.QuerySimpleStockTagsModel attribute)
(zvt.tag.tag_models.QueryStockTagsModel attribute)
(zvt.tag.tag_models.QueryStockTagStatsModel attribute)
(zvt.tag.tag_models.SetStockTagsModel attribute)
(zvt.tag.tag_models.SimpleStockTagsModel attribute)
(zvt.tag.tag_models.StockPoolInfoModel attribute)
(zvt.tag.tag_models.StockPoolModel attribute)
(zvt.tag.tag_models.StockPoolsModel attribute)
(zvt.tag.tag_models.StockTagDetailsModel attribute)
(zvt.tag.tag_models.StockTagsModel attribute)
(zvt.tag.tag_models.StockTagStatsModel attribute)
(zvt.tag.tag_models.TagInfoModel attribute)
(zvt.tag.tag_models.TagParameter attribute)
(zvt.trading.trading_models.BuildQueryStockQuoteSettingModel attribute)
(zvt.trading.trading_models.BuildTradingPlanModel attribute)
(zvt.trading.trading_models.QueryStockQuoteModel attribute)
(zvt.trading.trading_models.QueryStockQuoteSettingModel attribute)
(zvt.trading.trading_models.QueryTagQuoteModel attribute)
(zvt.trading.trading_models.QueryTradingPlanModel attribute)
(zvt.trading.trading_models.StockQuoteModel attribute)
(zvt.trading.trading_models.StockQuoteStatsModel attribute)
(zvt.trading.trading_models.TagQuoteStatsModel attribute)
(zvt.trading.trading_models.TradingPlanModel attribute)
model_fields_set (zvt.common.query_models.AbsoluteTimeRange property)
(zvt.common.query_models.RelativeTimeRage property)
(zvt.common.query_models.TimeRange property)
(zvt.common.trading_models.BuyParameter property)
(zvt.common.trading_models.SellParameter property)
(zvt.common.trading_models.TradingResult property)
(zvt.contract.model.CustomModel property)
zvt.recorders.joinquant.mis
zvt.recorders.joinquant.ove
zvt.recorders.joinquant.ove
zvt.recorders.joinquant.ove
zvt.recorders.joinquant.ove
zvt.recorders.joinquant.quo
zvt.recorders.joinquant.quo
zvt.recorders.joinquant.quo
zvt.recorders.jqka
zvt.recorders.jqka.emotion
zvt.recorders.jqka.emotion.
zvt.recorders.jqka.jqka_api
zvt.recorders.qmt
zvt.recorders.qmt.meta
zvt.recorders.sina
zvt.recorders.sina.meta
zvt.recorders.sina.meta.sina
zvt.recorders.sina.money_f
zvt.recorders.sina.money_f
zvt.recorders.sina.money_f
zvt.recorders.sina.quotes
zvt.recorders.sina.quotes.si
zvt.recorders.sina.quotes.si
zvt.recorders.wb
zvt.recorders.wb.wb_api
zvt.recorders.wb.wb_count
zvt.recorders.wb.wb_econc
zvt.rest
zvt.rest.data
zvt.rest.factor
zvt.rest.trading
zvt.rest.work
zvt.samples
zvt.samples.stock_traders
zvt.sched
zvt.sched.sched
zvt.tag
zvt.tag.common
zvt.tag.tag_models
zvt.tag.tag_schemas
zvt.tag.tag_service
zvt.tag.tag_stats
zvt.tag.tag_utils
zvt.tag.tagger
zvt.trader
zvt.trader.errors
zvt.trader.sim_account
zvt.trader.trader
zvt.trader.trader_info_api
zvt.trader.trader_schemas
zvt.trading
zvt.trading.common
zvt.trading.trading_models
zvt.trading.trading_schema
zvt.trading.trading_service
zvt.ui

(zvt.contract.model.MixinModel property)
(zvt.factors.factor_models.FactorRequestModel property)
(zvt.factors.factor_models.FactorResultModel property)
(zvt.factors.factor_models.KdataModel property)
(zvt.factors.factor_models.KdataRequestModel property)
(zvt.factors.factor_models.TradingSignalModel property)
(zvt.tag.tag_models.ActivateSubTagsModel property)
(zvt.tag.tag_models.ActivateSubTagsResultModel property)
(zvt.tag.tag_models.BatchSetStockTagsModel property)
(zvt.tag.tag_models.CreateStockPoolInfoModel property)
(zvt.tag.tag_models.CreateStockPoolsModel property)
(zvt.tag.tag_models.CreateTagInfoModel property)
(zvt.tag.tag_models.QuerySimpleStockTagsModel property)
(zvt.tag.tag_models.QueryStockTagsModel property)
(zvt.tag.tag_models.QueryStockTagStatsModel property)
(zvt.tag.tag_models.SetStockTagsModel property)
(zvt.tag.tag_models.SimpleStockTagsModel property)
(zvt.tag.tag_models.StockPoolInfoModel property)
(zvt.tag.tag_models.StockPoolModel property)
(zvt.tag.tag_models.StockPoolsModel property)
(zvt.tag.tag_models.StockTagDetailsModel property)
(zvt.tag.tag_models.StockTagsModel property)
(zvt.tag.tag_models.StockTagStatsModel property)
(zvt.tag.tag_models.TagInfoModel property)
(zvt.tag.tag_models.TagParameter property)
(zvt.trading.trading_models.BuildQueryStockQuoteSettingModel property)
(zvt.trading.trading_models.BuildTradingPlanModel property)
(zvt.trading.trading_models.QueryStockQuoteModel property)
(zvt.trading.trading_models.QueryStockQuoteSettingModel property)
(zvt.trading.trading_models.QueryTagQuoteModel property)
(zvt.trading.trading_models.QueryTradingPlanModel property)
(zvt.trading.trading_models.StockQuoteModel property)
(zvt.trading.trading_models.StockQuoteStatsModel property)
(zvt.trading.trading_models.TagQuoteStatsModel property)
(zvt.trading.trading_models.TradingPlanModel property)
model_json_schema() (zvt.common.query_models.AbsoluteTimeRange class method)
(zvt.common.query_models.RelativeTimeRage class method)
(zvt.common.query_models.TimeRange class method)
(zvt.common.trading_models.BuyParameter class method)
(zvt.common.trading_models.SellParameter class method)
(zvt.common.trading_models.TradingResult class method)
(zvt.contract.model.CustomModel class method)
(zvt.contract.model.MixinModel class method)
(zvt.factors.factor_models.FactorRequestModel class method)
(zvt.factors.factor_models.FactorResultModel class method)
(zvt.factors.factor_models.KdataModel class method)
(zvt.factors.factor_models.KdataRequestModel class method)
(zvt.factors.factor_models.TradingSignalModel class method)
(zvt.tag.tag_models.ActivateSubTagsModel class method)
(zvt.tag.tag_models.ActivateSubTagsResultModel class method)
(zvt.tag.tag_models.BatchSetStockTagsModel class method)
(zvt.tag.tag_models.CreateStockPoolInfoModel class method)
(zvt.tag.tag_models.CreateStockPoolsModel class method)
zvt.ui.assets
zvt.ui.components
zvt.ui.components.dcc_con
zvt.utils
zvt.utils.decorator
zvt.utils.file_utils
zvt.utils.git_utils
zvt.utils.model_utils
zvt.utils.pd_utils
zvt.utils.recorder_utils
zvt.utils.str_utils
zvt.utils.time_utils
zvt.utils.utils
zvt.utils.zip_utils
zvt.zhdate
zvt.zhdate.constants
zvt.zhdate.zhdate
zvt.zhdate.ztime
month_days() (zvt.zhdate.zhdate
month_end_date() (in module zv
month_start_date() (in module z
month_start_end_ranges() (in m
move_on() (zvt.contract.factor.F
(zvt.contract.factor.ScoreFa
(zvt.contract.reader.DataRe
(zvt.factors.fundamental.fi
(zvt.factors.fundamental.fi
(zvt.factors.ma.ma_factor.
(zvt.factors.ma.ma_factor.
(zvt.factors.ma.ma_factor.
(zvt.factors.ma.ma_factor.
(zvt.factors.ma.ma_stats_fa
(zvt.factors.ma.ma_stats_fa
(zvt.factors.ma.top_bottom
(zvt.factors.macd.macd_fac
(zvt.factors.macd.macd_fac
(zvt.factors.macd.macd_fac
(zvt.factors.macd.macd_fac
(zvt.factors.macd.macd_fac
(zvt.factors.technical_facto
(zvt.factors.z.z_factor.ZFac
(zvt.trader.trader_info_api.
(zvt.trader.trader_info_api.
mro() (zvt.contract.factor.Factor
(zvt.contract.recorder.Meta
multiple_number() (in module z
MyBullTrader (class in zvt.samp
MyMaTrader (class in zvt.sampl

(zvt.tag.tag_models.CreateTagInfoModel class method)
(zvt.tag.tag_models.QuerySimpleStockTagsModel class method)
(zvt.tag.tag_models.QueryStockTagsModel class method)
(zvt.tag.tag_models.QueryStockTagStatsModel class method)
(zvt.tag.tag_models.SetStockTagsModel class method)
(zvt.tag.tag_models.SimpleStockTagsModel class method)
(zvt.tag.tag_models.StockPoolInfoModel class method)
(zvt.tag.tag_models.StockPoolModel class method)
(zvt.tag.tag_models.StockPoolsModel class method)
(zvt.tag.tag_models.StockTagDetailsModel class method)
(zvt.tag.tag_models.StockTagsModel class method)
(zvt.tag.tag_models.StockTagStatsModel class method)
(zvt.tag.tag_models.TagInfoModel class method)
(zvt.tag.tag_models.TagParameter class method)
(zvt.trading.trading_models.BuildQueryStockQuoteSettingModel class method)
(zvt.trading.trading_models.BuildTradingPlanModel class method)
(zvt.trading.trading_models.QueryStockQuoteModel class method)
(zvt.trading.trading_models.QueryStockQuoteSettingModel class method)
(zvt.trading.trading_models.QueryTagQuoteModel class method)
(zvt.trading.trading_models.QueryTradingPlanModel class method)
(zvt.trading.trading_models.StockQuoteModel class method)
(zvt.trading.trading_models.StockQuoteStatsModel class method)
(zvt.trading.trading_models.TagQuoteStatsModel class method)
(zvt.trading.trading_models.TradingPlanModel class method)
model_parametrized_name()
(zvt.common.query_models.AbsoluteTimeRange class method)
(zvt.common.query_models.RelativeTimeRage class method)
(zvt.common.query_models.TimeRange class method)
(zvt.common.trading_models.BuyParameter class method)
(zvt.common.trading_models.SellParameter class method)
(zvt.common.trading_models.TradingResult class method)
(zvt.contract.model.CustomModel class method)
(zvt.contract.model.MixinModel class method)
(zvt.factors.factor_models.FactorRequestModel class method)
(zvt.factors.factor_models.FactorResultModel class method)
(zvt.factors.factor_models.KdataModel class method)
(zvt.factors.factor_models.KdataRequestModel class method)
(zvt.factors.factor_models.TradingSignalModel class method)
(zvt.tag.tag_models.ActivateSubTagsModel class method)
(zvt.tag.tag_models.ActivateSubTagsResultModel class method)
(zvt.tag.tag_models.BatchSetStockTagsModel class method)
(zvt.tag.tag_models.CreateStockPoolInfoModel class method)
(zvt.tag.tag_models.CreateStockPoolsModel class method)
(zvt.tag.tag_models.CreateTagInfoModel class method)
(zvt.tag.tag_models.QuerySimpleStockTagsModel class method)
(zvt.tag.tag_models.QueryStockTagsModel class method)
(zvt.tag.tag_models.QueryStockTagStatsModel class method)
(zvt.tag.tag_models.SetStockTagsModel class method)
(zvt.tag.tag_models.SimpleStockTagsModel class method)
(zvt.tag.tag_models.StockPoolInfoModel class method)
(zvt.tag.tag_models.StockPoolModel class method)
(zvt.tag.tag_models.StockPoolsModel class method)
(zvt.tag.tag_models.StockTagDetailsModel class method)
(zvt.tag.tag_models.StockTagsModel class method)

(zvt.tag.tag_models.StockTagStatsModel class method)
(zvt.tag.tag_models.TagInfoModel class method)
(zvt.tag.tag_models.TagParameter class method)
(zvt.trading.trading_models.BuildQueryStockQuoteSettingModel class method)
(zvt.trading.trading_models.BuildTradingPlanModel class method)
(zvt.trading.trading_models.QueryStockQuoteModel class method)
(zvt.trading.trading_models.QueryStockQuoteSettingModel class method)
(zvt.trading.trading_models.QueryTagQuoteModel class method)
(zvt.trading.trading_models.QueryTradingPlanModel class method)
(zvt.trading.trading_models.StockQuoteModel class method)
(zvt.trading.trading_models.StockQuoteStatsModel class method)
(zvt.trading.trading_models.TagQuoteStatsModel class method)
(zvt.trading.trading_models.TradingPlanModel class method)
model_post_init() (zvt.common.query_models.AbsoluteTimeRange method)
(zvt.common.query_models.RelativeTimeRage method)
(zvt.common.query_models.TimeRange method)
(zvt.common.trading_models.BuyParameter method)
(zvt.common.trading_models.SellParameter method)
(zvt.common.trading_models.TradingResult method)
(zvt.contract.model.CustomModel method)
(zvt.contract.model.MixinModel method)
(zvt.factors.factor_models.FactorRequestModel method)
(zvt.factors.factor_models.FactorResultModel method)
(zvt.factors.factor_models.KdataModel method)
(zvt.factors.factor_models.KdataRequestModel method)
(zvt.factors.factor_models.TradingSignalModel method)
(zvt.tag.tag_models.ActivateSubTagsModel method)
(zvt.tag.tag_models.ActivateSubTagsResultModel method)
(zvt.tag.tag_models.BatchSetStockTagsModel method)
(zvt.tag.tag_models.CreateStockPoolInfoModel method)
(zvt.tag.tag_models.CreateStockPoolsModel method)
(zvt.tag.tag_models.CreateTagInfoModel method)
(zvt.tag.tag_models.QuerySimpleStockTagsModel method)
(zvt.tag.tag_models.QueryStockTagsModel method)
(zvt.tag.tag_models.QueryStockTagStatsModel method)
(zvt.tag.tag_models.SetStockTagsModel method)
(zvt.tag.tag_models.SimpleStockTagsModel method)
(zvt.tag.tag_models.StockPoolInfoModel method)
(zvt.tag.tag_models.StockPoolModel method)
(zvt.tag.tag_models.StockPoolsModel method)
(zvt.tag.tag_models.StockTagDetailsModel method)
(zvt.tag.tag_models.StockTagsModel method)
(zvt.tag.tag_models.StockTagStatsModel method)
(zvt.tag.tag_models.TagInfoModel method)
(zvt.tag.tag_models.TagParameter method)
(zvt.trading.trading_models.BuildQueryStockQuoteSettingModel method)
(zvt.trading.trading_models.BuildTradingPlanModel method)
(zvt.trading.trading_models.QueryStockQuoteModel method)
(zvt.trading.trading_models.QueryStockQuoteSettingModel method)
(zvt.trading.trading_models.QueryTagQuoteModel method)
(zvt.trading.trading_models.QueryTradingPlanModel method)

(zvt.trading.trading_models.StockQuoteModel method)
(zvt.trading.trading_models.StockQuoteStatsModel method)
(zvt.trading.trading_models.TagQuoteStatsModel method)
(zvt.trading.trading_models.TradingPlanModel method)

N

name (zvt.contract.base_service.EntityStateService attribute)
(zvt.contract.base_service.OneStateService attribute)
(zvt.contract.base_service.StatefulService attribute)
(zvt.contract.factor.Factor attribute)
(zvt.contract.factor.ScoreFactor attribute)
(zvt.contract.recorder.EntityEventRecorder attribute)
(zvt.contract.recorder.FixedCycleDataRecorder attribute)
(zvt.contract.recorder.Recorder attribute)
(zvt.contract.recorder.TimeSeriesDataRecorder attribute)
(zvt.contract.recorder.TimestampsDataRecorder attribute)
(zvt.contract.schema.ActorEntity attribute)
(zvt.contract.schema.ActorMeetTradable attribute)
(zvt.contract.schema.Entity attribute)
(zvt.contract.schema.NormalEntityMixin attribute)
(zvt.contract.schema.Portfolio attribute)
(zvt.contract.schema.PortfolioStock attribute)
(zvt.contract.schema.PortfolioStockHistory attribute)
(zvt.contract.schema.TradableEntity attribute)
(zvt.contract.schema.TradableMeetActor attribute)
(zvt.domain.actor.actor_meta.ActorMeta attribute)
(zvt.domain.actor.stock_actor.StockActorSummary attribute)
(zvt.domain.actor.stock_actor.StockInstitutionalInvestorHolder attribute)
(zvt.domain.actor.stock_actor.StockTopTenFreeHolder attribute)
(zvt.domain.actor.stock_actor.StockTopTenHolder attribute)
(zvt.domain.meta.block_meta.Block attribute)
(zvt.domain.meta.block_meta.BlockStock attribute)
(zvt.domain.meta.country_meta.Country attribute)
(zvt.domain.meta.currency_meta.Currency attribute)
(zvt.domain.meta.etf_meta.Etf attribute)
(zvt.domain.meta.etf_meta.EtfStock attribute)
(zvt.domain.meta.fund_meta.Fund attribute)
(zvt.domain.meta.fund_meta.FundStock attribute)
(zvt.domain.meta.future_meta.Future attribute)
(zvt.domain.meta.index_meta.Index attribute)
(zvt.domain.meta.index_meta.IndexStock attribute)
(zvt.domain.meta.indexus_meta.Indexus attribute)
(zvt.domain.meta.stock_meta.Stock attribute)
(zvt.domain.meta.stock_meta.StockDetail attribute)
(zvt.domain.meta.stockhk_meta.Stockhk attribute)
(zvt.domain.meta.stockus_meta.Stockus attribute)
(zvt.domain.misc.holder.HkHolder attribute)
(zvt.domain.quotes.block.block_1d_kdata.Block1dKdata attribute)
(zvt.domain.quotes.block.block_1mon_kdata.Block1monKdata attribute)
(zvt.domain.quotes.block.block_1wk_kdata.Block1wkKdata attribute)
(zvt.domain.quotes.currency.currency_1d_kdata.Currency1dKdata attribute)
(zvt.domain.quotes.etf.etf_1d_kdata.Etf1dKdata attribute)
(zvt.domain.quotes.future.future_1d_kdata.Future1dKdata attribute)
(zvt.domain.quotes.index.index_1d_kdata.Index1dKdata attribute)

(zvt.domain.quotes.index.index_1wk_kdata.Index1wkKdata attribute)
(zvt.domain.quotes.indexus.indexus_1d_kdata.Indexus1dKdata attribute)
(zvt.domain.quotes.stock.stock_15m_hfq_kdata.Stock15mHfqKdata attribute)
(zvt.domain.quotes.stock.stock_15m_kdata.Stock15mKdata attribute)
(zvt.domain.quotes.stock.stock_1d_hfq_kdata.Stock1dHfqKdata attribute)
(zvt.domain.quotes.stock.stock_1d_kdata.Stock1dKdata attribute)
(zvt.domain.quotes.stock.stock_1h_hfq_kdata.Stock1hHfqKdata attribute)
(zvt.domain.quotes.stock.stock_1h_kdata.Stock1hKdata attribute)
(zvt.domain.quotes.stock.stock_1m_hfq_kdata.Stock1mHfqKdata attribute)
(zvt.domain.quotes.stock.stock_1m_kdata.Stock1mKdata attribute)
(zvt.domain.quotes.stock.stock_1mon_hfq_kdata.Stock1monHfqKdata attribute)
(zvt.domain.quotes.stock.stock_1mon_kdata.Stock1monKdata attribute)
(zvt.domain.quotes.stock.stock_1wk_hfq_kdata.Stock1wkHfqKdata attribute)
(zvt.domain.quotes.stock.stock_1wk_kdata.Stock1wkKdata attribute)
(zvt.domain.quotes.stock.stock_30m_hfq_kdata.Stock30mHfqKdata attribute)
(zvt.domain.quotes.stock.stock_30m_kdata.Stock30mKdata attribute)
(zvt.domain.quotes.stock.stock_4h_hfq_kdata.Stock4hHfqKdata attribute)
(zvt.domain.quotes.stock.stock_4h_kdata.Stock4hKdata attribute)
(zvt.domain.quotes.stock.stock_5m_hfq_kdata.Stock5mHfqKdata attribute)
(zvt.domain.quotes.stock.stock_5m_kdata.Stock5mKdata attribute)
(zvt.domain.quotes.stockhk.stockhk_1d_hfq_kdata.Stockhk1dHfqKdata attribute)
(zvt.domain.quotes.stockhk.stockhk_1d_kdata.Stockhk1dKdata attribute)
(zvt.domain.quotes.stockus.stockus_1d_hfq_kdata.Stockus1dHfqKdata attribute)
(zvt.domain.quotes.stockus.stockus_1d_kdata.Stockus1dKdata attribute)
(zvt.factors.fundamental.finance_factor.FinanceBaseFactor attribute)
(zvt.factors.fundamental.finance_factor.GoodCompanyFactor attribute)
(zvt.factors.ma.ma_factor.CrossMaFactor attribute)
(zvt.factors.ma.ma_factor.CrossMaVolumeFactor attribute)
(zvt.factors.ma.ma_factor.MaFactor attribute)
(zvt.factors.ma.ma_factor.VolumeUpMaFactor attribute)
(zvt.factors.ma.ma_stats_factor.MaStatsFactor attribute)
(zvt.factors.ma.ma_stats_factor.TFactor attribute)
(zvt.factors.ma.top_bottom_factor.TopBottomFactor attribute)
(zvt.factors.macd.macd_factor.BullFactor attribute)
(zvt.factors.macd.macd_factor.GoldCrossFactor attribute)
(zvt.factors.macd.macd_factor.KeepBullFactor attribute)
(zvt.factors.macd.macd_factor.LiveOrDeadFactor attribute)
(zvt.factors.macd.macd_factor.MacdFactor attribute)
(zvt.factors.technical_factor.TechnicalFactor attribute)
(zvt.factors.z.z_factor.ZFactor attribute)
(zvt.recorders.eastmoney.common.EastmoneyMoreDataRecorder attribute)
(zvt.recorders.eastmoney.common.EastmoneyPageabeDataRecorder attribute)
(zvt.recorders.eastmoney.common.EastmoneyTimestampsDataRecorder attribute)
(zvt.recorders.eastmoney.dividend_financing.eastmoney_dividend_detail_recorder.DividendDetailRecord
attribute)
(zvt.recorders.eastmoney.dividend_financing.eastmoney_dividend_financing_recorder.DividendFinancing
attribute)
(zvt.recorders.eastmoney.dividend_financing.eastmoney_rights_issue_detail_recorder.RightsIssueDetailR
attribute)
(zvt.recorders.eastmoney.dividend_financing.eastmoney_spo_detail_recorder.SPODetailRecorder attribut
(zvt.recorders.eastmoney.finance.base_china_stock_finance_recorder.BaseChinaStockFinanceRecorder at
(zvt.recorders.eastmoney.finance.eastmoney_balance_sheet_recorder.ChinaStockBalanceSheetRecorder a
(zvt.recorders.eastmoney.finance.eastmoney_cash_flow_recorder.ChinaStockCashFlowRecorder attribute
(zvt.recorders.eastmoney.finance.eastmoney_finance_factor_recorder.ChinaStockFinanceFactorRecorder_
(zvt.recorders.eastmoney.finance.eastmoney_income_statement_recorder.ChinaStockIncomeStatementRe
attribute)

(zvt.recorders.eastmoney.holder.eastmoney_stock_actor_recorder.EastmoneyActorRecorder attribute)
(zvt.recorders.eastmoney.holder.eastmoney_top_ten_holder_recorder.TopTenHolderRecorder attribute)
(zvt.recorders.eastmoney.holder.eastmoney_top_ten_tradable_holder_recorder.TopTenTradableHolderRec
attribute)
(zvt.recorders.eastmoney.meta.eastmoney_block_meta_recorder.EastmoneyBlockRecorder attribute)
(zvt.recorders.eastmoney.meta.eastmoney_block_meta_recorder.EastmoneyBlockStockRecorder attribute
(zvt.recorders.eastmoney.meta.eastmoney_stock_meta_recorder.EastmoneyStockDetailRecorder attribute
(zvt.recorders.eastmoney.meta.eastmoney_stock_meta_recorder.EastmoneyStockRecorder attribute)
(zvt.recorders.eastmoney.trading.eastmoney_holder_trading_recorder.HolderTradingRecorder attribute)
(zvt.recorders.eastmoney.trading.eastmoney_manager_trading_recorder.ManagerTradingRecorder attribut
(zvt.recorders.em.actor.em_stock_actor_summary_recorder.EMStockActorSummaryRecorder attribute)
(zvt.recorders.em.actor.em_stock_ii_recorder.EMStockIIRecorder attribute)
(zvt.recorders.em.actor.em_stock_top_ten_free_recorder.EMStockTopTenFreeRecorder attribute)
(zvt.recorders.em.actor.em_stock_top_ten_recorder.EMStockTopTenRecorder attribute)
(zvt.recorders.em.macro.em_treasury_yield_recorder.EMTreasuryYieldRecorder attribute)
(zvt.recorders.em.meta.em_block_meta_recorder.EMBlockRecorder attribute)
(zvt.recorders.em.meta.em_block_meta_recorder.EMBlockStockRecorder attribute)
(zvt.recorders.em.meta.em_currency_meta_recorder.EMCurrencyRecorder attribute)
(zvt.recorders.em.meta.em_future_meta_recorder.EMFutureRecorder attribute)
(zvt.recorders.em.meta.em_index_meta_recorder.EMIndexRecorder attribute)
(zvt.recorders.em.meta.em_indexus_meta_recorder.EMIndexusRecorder attribute)
(zvt.recorders.em.meta.em_stock_meta_recorder.EMStockRecorder attribute)
(zvt.recorders.em.meta.em_stockhk_meta_recorder.EMStockhkRecorder attribute)
(zvt.recorders.em.meta.em_stockus_meta_recorder.EMStockusRecorder attribute)
(zvt.recorders.em.misc.em_stock_news_recorder.EMStockNewsRecorder attribute)
(zvt.recorders.em.quotes.em_kdata_recorder.BaseEMStockKdataRecorder attribute)
(zvt.recorders.em.quotes.em_kdata_recorder.EMBlockKdataRecorder attribute)
(zvt.recorders.em.quotes.em_kdata_recorder.EMCurrencyKdataRecorder attribute)
(zvt.recorders.em.quotes.em_kdata_recorder.EMFutureKdataRecorder attribute)
(zvt.recorders.em.quotes.em_kdata_recorder.EMIndexKdataRecorder attribute)
(zvt.recorders.em.quotes.em_kdata_recorder.EMIndexusKdataRecorder attribute)
(zvt.recorders.em.quotes.em_kdata_recorder.EMStockhkKdataRecorder attribute)
(zvt.recorders.em.quotes.em_kdata_recorder.EMStockKdataRecorder attribute)
(zvt.recorders.em.quotes.em_kdata_recorder.EMStockusKdataRecorder attribute)
(zvt.recorders.em.trading.em_dragon_and_tiger_recorder.EMDragonAndTigerRecorder attribute)
(zvt.recorders.exchange.exchange_etf_meta_recorder.ChinaETFListSpider attribute)
(zvt.recorders.exchange.exchange_index_recorder.ExchangeIndexRecorder attribute)
(zvt.recorders.exchange.exchange_index_stock_recorder.ExchangeIndexStockRecorder attribute)
(zvt.recorders.exchange.exchange_stock_meta_recorder.ExchangeStockMetaRecorder attribute)
(zvt.recorders.exchange.exchange_stock_summary_recorder.ExchangeStockSummaryRecorder attribute)
(zvt.recorders.joinquant.fundamental.jq_etf_valuation_recorder.JqChinaEtfValuationRecorder attribute)
(zvt.recorders.joinquant.fundamental.jq_margin_trading_recorder.MarginTradingRecorder attribute)
(zvt.recorders.joinquant.fundamental.jq_stock_valuation_recorder.JqChinaStockValuationRecorder attribu
(zvt.recorders.joinquant.meta.jq_fund_meta_recorder.JqChinaFundRecorder attribute)
(zvt.recorders.joinquant.meta.jq_fund_meta_recorder.JqChinaFundStockRecorder attribute)
(zvt.recorders.joinquant.meta.jq_stock_meta_recorder.BaseJqChinaMetaRecorder attribute)
(zvt.recorders.joinquant.meta.jq_stock_meta_recorder.JqChinaEtfRecorder attribute)
(zvt.recorders.joinquant.meta.jq_stock_meta_recorder.JqChinaStockEtfPortfolioRecorder attribute)
(zvt.recorders.joinquant.meta.jq_stock_meta_recorder.JqChinaStockRecorder attribute)
(zvt.recorders.joinquant.meta.jq_trade_day_recorder.StockTradeDayRecorder attribute)
(zvt.recorders.joinquant.misc.jq_hk_holder_recorder.JoinquantHkHolderRecorder attribute)
(zvt.recorders.joinquant.misc.jq_index_money_flow_recorder.JoinquantIndexMoneyFlowRecorder attribu
(zvt.recorders.joinquant.misc.jq_stock_money_flow_recorder.JoinquantStockMoneyFlowRecorder attribu
(zvt.recorders.joinquant.overall.jq_cross_market_recorder.CrossMarketSummaryRecorder attribute)
(zvt.recorders.joinquant.overall.jq_margin_trading_recorder.MarginTradingSummaryRecorder attribute)
(zvt.recorders.joinquant.overall.jq_stock_summary_recorder.StockSummaryRecorder attribute)

(zvt.recorders.joinquant.quotes.jq_index_kdata_recorder.JqChinaIndexKdataRecorder attribute)
(zvt.recorders.joinquant.quotes.jq_stock_kdata_recorder.JqChinaStockKdataRecorder attribute)
(zvt.recorders.sina.meta.sina_block_recorder.SinaBlockRecorder attribute)
(zvt.recorders.sina.meta.sina_block_recorder.SinaChinaBlockStockRecorder attribute)
(zvt.recorders.sina.money_flow.sina_block_money_flow_recorder.SinaBlockMoneyFlowRecorder attribu
(zvt.recorders.sina.money_flow.sina_stock_money_flow_recorder.SinaStockMoneyFlowRecorder attribut
(zvt.recorders.sina.quotes.sina_etf_kdata_recorder.ChinaETFDayKdataRecorder attribute)
(zvt.recorders.sina.quotes.sina_index_kdata_recorder.ChinaIndexDayKdataRecorder attribute)
(zvt.recorders.wb.wb_country_recorder.WBCountryRecorder attribute)
(zvt.recorders.wb.wb_economy_recorder.WBEconomyRecorder attribute)
(zvt.tag.tag_models.StockTagDetailsModel attribute)
(zvt.tag.tag_schemas.StockSystemTags attribute)
(zvt.tag.tag_service.StockAutoTagger attribute)
(zvt.tag.tagger.StockTagger attribute)
(zvt.tag.tagger.Tagger attribute)
(zvt.trading.trading_models.StockQuoteModel attribute)

O

old_db_to_provider_dir() (in module zvt)
on_bind_field() (zvt.trader.sim_account.AccountDayStatsSchema method)
(zvt.trader.sim_account.PositionSchema method)
on_data_changed() (zvt.contract.factor.Factor method)
(zvt.contract.factor.ScoreFactor method)
(zvt.contract.reader.DataListener method)
(zvt.factors.fundamental.finance_factor.FinanceBaseFactor method)
(zvt.factors.fundamental.finance_factor.GoodCompanyFactor method)
(zvt.factors.ma.ma_factor.CrossMaFactor method)
(zvt.factors.ma.ma_factor.CrossMaVolumeFactor method)
(zvt.factors.ma.ma_factor.MaFactor method)
(zvt.factors.ma.ma_factor.VolumeUpMaFactor method)
(zvt.factors.ma.ma_stats_factor.MaStatsFactor method)
(zvt.factors.ma.ma_stats_factor.TFactor method)
(zvt.factors.ma.top_bottom_factor.TopBottomFactor method)
(zvt.factors.macd.macd_factor.BullFactor method)
(zvt.factors.macd.macd_factor.GoldCrossFactor method)
(zvt.factors.macd.macd_factor.KeepBullFactor method)
(zvt.factors.macd.macd_factor.LiveOrDeadFactor method)
(zvt.factors.macd.macd_factor.MacdFactor method)
(zvt.factors.technical_factor.TechnicalFactor method)
(zvt.factors.z.z_factor.ZFactor method)
on_data_loaded() (zvt.contract.factor.Factor method)
(zvt.contract.factor.ScoreFactor method)
(zvt.contract.reader.DataListener method)
(zvt.factors.fundamental.finance_factor.FinanceBaseFactor method)
(zvt.factors.fundamental.finance_factor.GoodCompanyFactor method)
(zvt.factors.ma.ma_factor.CrossMaFactor method)
(zvt.factors.ma.ma_factor.CrossMaVolumeFactor method)
(zvt.factors.ma.ma_factor.MaFactor method)
(zvt.factors.ma.ma_factor.VolumeUpMaFactor method)
(zvt.factors.ma.ma_stats_factor.MaStatsFactor method)
(zvt.factors.ma.ma_stats_factor.TFactor method)
(zvt.factors.ma.top_bottom_factor.TopBottomFactor method)
(zvt.factors.macd.macd_factor.BullFactor method)
(zvt.factors.macd.macd_factor.GoldCrossFactor method)
operate_mode_id (zvt.doma
operating_costs (zvt.domai
operating_income (zvt.dom
operating_profit (zvt.doma
opposite_change (zvt.facto
(zvt.factors.z.domain.
opposite_interval (zvt.facto
(zvt.factors.z.domain.
opposite_slope (zvt.factors.
(zvt.factors.z.domain.
option (zvt.contract.Tradab
OPTIONS_CLASS (zvt.tra
(zvt.trader.sim_accour
Order (class in zvt.trader.tra
order_amount (zvt.domain.
(zvt.trader.trader_sche
order_price (zvt.trader.trad
order_type (zvt.trader.trade
order_type_color() (in mod
order_type_flag() (in modu
OrderByType (class in zvt.
OrderReader (class in zvt.tr
OrderType (class in zvt.trac
original_page_url (zvt.cont
(zvt.contract.recorder.
(zvt.contract.recorder.
(zvt.contract.recorder.
(zvt.contract.recorder.
(zvt.recorders.eastmor
(zvt.recorders.eastmor
(zvt.recorders.eastmor
(zvt.recorders.eastmor
attribute)
(zvt.recorders.eastmor
attribute)

(zvt.factors.macd.macd_factor.KeepBullFactor method)
(zvt.factors.macd.macd_factor.LiveOrDeadFactor method)
(zvt.factors.macd.macd_factor.MacdFactor method)
(zvt.factors.technical_factor.TechnicalFactor method)
(zvt.factors.z.z_factor.ZFactor method)
on_entity_data_changed() (zvt.contract.factor.Factor method)
(zvt.contract.factor.ScoreFactor method)
(zvt.contract.reader.DataListener method)
(zvt.factors.fundamental.finance_factor.FinanceBaseFactor method)
(zvt.factors.fundamental.finance_factor.GoodCompanyFactor method)
(zvt.factors.ma.ma_factor.CrossMaFactor method)
(zvt.factors.ma.ma_factor.CrossMaVolumeFactor method)
(zvt.factors.ma.ma_factor.MaFactor method)
(zvt.factors.ma.ma_factor.VolumeUpMaFactor method)
(zvt.factors.ma.ma_stats_factor.MaStatsFactor method)
(zvt.factors.ma.ma_stats_factor.TFactor method)
(zvt.factors.ma.top_bottom_factor.TopBottomFactor method)
(zvt.factors.macd.macd_factor.BullFactor method)
(zvt.factors.macd.macd_factor.GoldCrossFactor method)
(zvt.factors.macd.macd_factor.KeepBullFactor method)
(zvt.factors.macd.macd_factor.LiveOrDeadFactor method)
(zvt.factors.macd.macd_factor.MacdFactor method)
(zvt.factors.technical_factor.TechnicalFactor method)
(zvt.factors.z.z_factor.ZFactor method)
on_factor_targets_filtered() (zvt.samples.stock_traders.MyBullTrader method)
(zvt.samples.stock_traders.MyMaTrader method)
(zvt.trader.trader.StockTrader method)
(zvt.trader.trader.Trader method)
on_targets_selected_from_levels() (zvt.samples.stock_traders.MyBullTrader method)
(zvt.samples.stock_traders.MyMaTrader method)
(zvt.trader.trader.StockTrader method)
(zvt.trader.trader.Trader method)
on_time() (zvt.samples.stock_traders.MyBullTrader method)
(zvt.samples.stock_traders.MyMaTrader method)
(zvt.trader.trader.StockTrader method)
(zvt.trader.trader.Trader method)
OneStateService (class in zvt.contract.base_service)
op_cash_flow_ps (zvt.domain.fundamental.finance.FinanceFactor attribute)
op_income_growth_qoq (zvt.domain.fundamental.finance.FinanceFactor attribute)
op_income_growth_yoy (zvt.domain.fundamental.finance.FinanceFactor attribute)
op_net_cash_flow_per_op_income (zvt.domain.fundamental.finance.FinanceFactor attribute)
open (zvt.domain.quotes.block.block_1d_kdata.Block1dKdata attribute)
(zvt.domain.quotes.block.block_1mon_kdata.Block1monKdata attribute)
(zvt.domain.quotes.block.block_1wk_kdata.Block1wkKdata attribute)
(zvt.domain.quotes.currency.currency_1d_kdata.Currency1dKdata attribute)
(zvt.domain.quotes.etf.etf_1d_kdata.Etf1dKdata attribute)
(zvt.domain.quotes.future.future_1d_kdata.Future1dKdata attribute)
(zvt.domain.quotes.index.index_1d_kdata.Index1dKdata attribute)
(zvt.domain.quotes.index.index_1wk_kdata.Index1wkKdata attribute)
(zvt.recorders.eastmor
attribute)
(zvt.recorders.eastmor
(zvt.recorders.eastmor
(zvt.recorders.eastmor
(zvt.recorders.eastmor
(zvt.recorders.eastmor
(zvt.recorders.eastmor
(zvt.recorders.eastmor
attribute)
(zvt.recorders.eastmor
(zvt.recorders.eastmor
(zvt.recorders.eastmor
(zvt.recorders.eastmor
attribute)
(zvt.recorders.eastmor
(zvt.recorders.eastmor
(zvt.recorders.eastmor
(zvt.recorders.eastmor
(zvt.recorders.eastmor
(zvt.recorders.em.acto
(zvt.recorders.em.acto
(zvt.recorders.em.acto
(zvt.recorders.em.acto
(zvt.recorders.em.mac
(zvt.recorders.em.meta
(zvt.recorders.em.meta
(zvt.recorders.em.meta
(zvt.recorders.em.meta
(zvt.recorders.em.meta
(zvt.recorders.em.meta
(zvt.recorders.em.meta
(zvt.recorders.em.meta
(zvt.recorders.em.meta
(zvt.recorders.em.misc
(zvt.recorders.em.quot
(zvt.recorders.em.quot
(zvt.recorders.em.quot
(zvt.recorders.em.quot
(zvt.recorders.em.quot
(zvt.recorders.em.quot
(zvt.recorders.em.quot
(zvt.recorders.em.quot
(zvt.recorders.em.quot
(zvt.recorders.em.trad
(zvt.recorders.exchang
(zvt.recorders.exchang
(zvt.recorders.exchang
(zvt.recorders.exchang
(zvt.recorders.exchang
(zvt.recorders.joinquan
(zvt.recorders.joinquan
(zvt.recorders.joinquan
(zvt.recorders.joinquan
(zvt.recorders.joinquan
(zvt.recorders.joinquan

(zvt.domain.quotes.indexus.indexus_1d_kdata.Indexus1dKdata attribute)
(zvt.domain.quotes.stock.stock_15m_hfq_kdata.Stock15mHfqKdata attribute)
(zvt.domain.quotes.stock.stock_15m_kdata.Stock15mKdata attribute)
(zvt.domain.quotes.stock.stock_1d_hfq_kdata.Stock1dHfqKdata attribute)
(zvt.domain.quotes.stock.stock_1d_kdata.Stock1dKdata attribute)
(zvt.domain.quotes.stock.stock_1h_hfq_kdata.Stock1hHfqKdata attribute)
(zvt.domain.quotes.stock.stock_1h_kdata.Stock1hKdata attribute)
(zvt.domain.quotes.stock.stock_1m_hfq_kdata.Stock1mHfqKdata attribute)
(zvt.domain.quotes.stock.stock_1m_kdata.Stock1mKdata attribute)
(zvt.domain.quotes.stock.stock_1mon_hfq_kdata.Stock1monHfqKdata attribute)
(zvt.domain.quotes.stock.stock_1mon_kdata.Stock1monKdata attribute)
(zvt.domain.quotes.stock.stock_1wk_hfq_kdata.Stock1wkHfqKdata attribute)
(zvt.domain.quotes.stock.stock_1wk_kdata.Stock1wkKdata attribute)
(zvt.domain.quotes.stock.stock_30m_hfq_kdata.Stock30mHfqKdata attribute)
(zvt.domain.quotes.stock.stock_30m_kdata.Stock30mKdata attribute)
(zvt.domain.quotes.stock.stock_4h_hfq_kdata.Stock4hHfqKdata attribute)
(zvt.domain.quotes.stock.stock_4h_kdata.Stock4hKdata attribute)
(zvt.domain.quotes.stock.stock_5m_hfq_kdata.Stock5mHfqKdata attribute)
(zvt.domain.quotes.stock.stock_5m_kdata.Stock5mKdata attribute)
(zvt.domain.quotes.stockhk.stockhk_1d_hfq_kdata.Stockhk1dHfqKdata attribute)
(zvt.domain.quotes.stockhk.stockhk_1d_kdata.Stockhk1dKdata attribute)
(zvt.domain.quotes.stockus.stockus_1d_hfq_kdata.Stockus1dHfqKdata attribute)
(zvt.domain.quotes.stockus.stockus_1d_kdata.Stockus1dKdata attribute)
(zvt.domain.quotes.TickCommon attribute)
(zvt.factors.ma.domain.stock_1d_ma_stats_factor.Stock1dMaStatsFactor attribute)
(zvt.factors.z.domain.stock_1d_z_factor.Stock1dZFactor attribute)
(zvt.factors.z.domain.stock_1wk_z_factor.Stock1wkZFactor attribute)
open_count (zvt.domain.emotion.emotion.LimitUpInfo attribute)
operate_mode (zvt.domain.meta.fund_meta.Fund attribute)

P

parse_sz_etf_underlying_index() (zvt.recorders.exchange.exchange_etf_meta_recorder.ChinaETFListSpider st method)
parse_url_params() (in module zvt.utils.utils)
pb (zvt.domain.fundamental.valuation.EtfValuation attribute)
(zvt.domain.fundamental.valuation.StockValuation attribute)
pb1 (zvt.domain.fundamental.valuation.EtfValuation attribute)
pcf (zvt.domain.fundamental.valuation.EtfValuation attribute)
(zvt.domain.fundamental.valuation.StockValuation attribute)
pcf1 (zvt.domain.fundamental.valuation.EtfValuation attribute)

(zvt.recorders.joinqua
(zvt.recorders.joinqua
(zvt.recorders.joinqua
(zvt.recorders.joinqua
(zvt.recorders.joinqua
(zvt.recorders.joinqua
(zvt.recorders.joinqua
(zvt.recorders.joinqua
(zvt.recorders.joinqua
(zvt.recorders.joinqua
(zvt.recorders.joinqua
(zvt.recorders.sina.me
(zvt.recorders.sina.me
(zvt.recorders.sina.mo
(zvt.recorders.sina.mo
(zvt.recorders.sina.qu
(zvt.recorders.sina.qu
(zvt.recorders.wb.wb_
(zvt.recorders.wb.wb_
other_comprehensive_inco
other_comprehensive_inco
other_comprehensive_inco
other_current_assets (zvt.d
other_current_liabilities (z
other_non_current_assets (
other_non_current_liabiliti
other_payable (zvt.domain.
other_receivables (zvt.dom

pct_to_float() (in module zvt.utils.utils)
pd_is_not_null() (in module zvt.utils.pd_utils)
pe (zvt.domain.fundamental.valuation.EtfValuation attribute)
(zvt.domain.fundamental.valuation.StockValuation attribute)
pe1 (zvt.domain.fundamental.valuation.EtfValuation attribute)
pe_ttm (zvt.domain.fundamental.valuation.EtfValuation attribute)
(zvt.domain.fundamental.valuation.StockValuation attribute)
pe_ttm1 (zvt.domain.fundamental.valuation.EtfValuation attribute)
persist() (zvt.contract.recorder.FixedCycleDataRecorder method)
(zvt.contract.recorder.TimeSeriesDataRecorder method)
(zvt.contract.recorder.TimestampsDataRecorder method)
(zvt.recorders.eastmoney.common.EastmoneyMoreDataRecorder method)
(zvt.recorders.eastmoney.common.EastmoneyPageabeDataRecorder method)
(zvt.recorders.eastmoney.common.EastmoneyTimestampsDataRecorder method)
(zvt.recorders.eastmoney.dividend_financing.eastmoney_dividend_detail_recorder.DividendDetailRecord
method)
(zvt.recorders.eastmoney.dividend_financing.eastmoney_dividend_financing_recorder.DividendFinancing
method)
(zvt.recorders.eastmoney.dividend_financing.eastmoney_rights_issue_detail_recorder.RightsIssueDetailR
method)
(zvt.recorders.eastmoney.dividend_financing.eastmoney_spo_detail_recorder.SPODetailRecorder method
(zvt.recorders.eastmoney.finance.base_china_stock_finance_recorder.BaseChinaStockFinanceRecorder m
(zvt.recorders.eastmoney.finance.eastmoney_balance_sheet_recorder.ChinaStockBalanceSheetRecorder n
(zvt.recorders.eastmoney.finance.eastmoney_cash_flow_recorder.ChinaStockCashFlowRecorder method)
(zvt.recorders.eastmoney.finance.eastmoney_finance_factor_recorder.ChinaStockFinanceFactorRecorder
(zvt.recorders.eastmoney.finance.eastmoney_income_statement_recorder.ChinaStockIncomeStatementRe
method)
(zvt.recorders.eastmoney.holder.eastmoney_top_ten_holder_recorder.TopTenHolderRecorder method)
(zvt.recorders.eastmoney.holder.eastmoney_top_ten_tradable_holder_recorder.TopTenTradableHolderRec
method)
(zvt.recorders.eastmoney.meta.eastmoney_block_meta_recorder.EastmoneyBlockStockRecorder method)
(zvt.recorders.eastmoney.trading.eastmoney_holder_trading_recorder.HolderTradingRecorder method)
(zvt.recorders.eastmoney.trading.eastmoney_manager_trading_recorder.ManagerTradingRecorder method
(zvt.recorders.em.actor.em_stock_actor_summary_recorder.EMStockActorSummaryRecorder method)
(zvt.recorders.em.actor.em_stock_ii_recorder.EMStockIIRecorder method)
(zvt.recorders.em.actor.em_stock_top_ten_free_recorder.EMStockTopTenFreeRecorder method)
(zvt.recorders.em.actor.em_stock_top_ten_recorder.EMStockTopTenRecorder method)
(zvt.recorders.em.macro.em_treasury_yield_recorder.EMTreasuryYieldRecorder method)
(zvt.recorders.em.meta.em_block_meta_recorder.EMBlockStockRecorder method)
(zvt.recorders.em.misc.em_stock_news_recorder.EMStockNewsRecorder method)
(zvt.recorders.em.quotes.em_kdata_recorder.BaseEMStockKdataRecorder method)
(zvt.recorders.em.quotes.em_kdata_recorder.EMBlockKdataRecorder method)
(zvt.recorders.em.quotes.em_kdata_recorder.EMCurrencyKdataRecorder method)
(zvt.recorders.em.quotes.em_kdata_recorder.EMFutureKdataRecorder method)
(zvt.recorders.em.quotes.em_kdata_recorder.EMIndexKdataRecorder method)
(zvt.recorders.em.quotes.em_kdata_recorder.EMIndexusKdataRecorder method)
(zvt.recorders.em.quotes.em_kdata_recorder.EMStockhkKdataRecorder method)
(zvt.recorders.em.quotes.em_kdata_recorder.EMStockKdataRecorder method)
(zvt.recorders.em.quotes.em_kdata_recorder.EMStockusKdataRecorder method)
(zvt.recorders.em.trading.em_dragon_and_tiger_recorder.EMDragonAndTigerRecorder method)
(zvt.recorders.exchange.exchange_index_stock_recorder.ExchangeIndexStockRecorder method)
(zvt.recorders.exchange.exchange_stock_summary_recorder.ExchangeStockSummaryRecorder method)
(zvt.recorders.joinquant.fundamental.jq_etf_valuation_recorder.JqChinaEtfValuationRecorder method)
(zvt.recorders.joinquant.fundamental.jq_margin_trading_recorder.MarginTradingRecorder method)
(zvt.recorders.joinquant.fundamental.jq_stock_valuation_recorder.JqChinaStockValuationRecorder metho
(zvt.recorders.joinquant.meta.jq_fund_meta_recorder.JqChinaFundStockRecorder method)

(zvt.recorders.joinquant.meta.jq_stock_meta_recorder.JqChinaStockEtfPortfolioRecorder method)
(zvt.recorders.joinquant.meta.jq_trade_day_recorder.StockTradeDayRecorder method)
(zvt.recorders.joinquant.misc.jq_hk_holder_recorder.JoinquantHkHolderRecorder method)
(zvt.recorders.joinquant.misc.jq_index_money_flow_recorder.JoinquantIndexMoneyFlowRecorder metho
(zvt.recorders.joinquant.misc.jq_stock_money_flow_recorder.JoinquantStockMoneyFlowRecorder metho
(zvt.recorders.joinquant.overall.jq_cross_market_recorder.CrossMarketSummaryRecorder method)
(zvt.recorders.joinquant.overall.jq_margin_trading_recorder.MarginTradingSummaryRecorder method)
(zvt.recorders.joinquant.overall.jq_stock_summary_recorder.StockSummaryRecorder method)
(zvt.recorders.joinquant.quotes.jq_index_kdata_recorder.JqChinaIndexKdataRecorder method)
(zvt.recorders.joinquant.quotes.jq_stock_kdata_recorder.JqChinaStockKdataRecorder method)
(zvt.recorders.sina.meta.sina_block_recorder.SinaChinaBlockStockRecorder method)
(zvt.recorders.sina.money_flow.sina_block_money_flow_recorder.SinaBlockMoneyFlowRecorder metho
(zvt.recorders.sina.money_flow.sina_stock_money_flow_recorder.SinaStockMoneyFlowRecorder metho
(zvt.recorders.sina.quotes.sina_etf_kdata_recorder.ChinaETFDayKdataRecorder method)
(zvt.recorders.sina.quotes.sina_index_kdata_recorder.ChinaIndexDayKdataRecorder method)
(zvt.recorders.wb.wb_economy_recorder.WBEconomyRecorder method)
pie (zvt.contract.drawer.ChartType attribute)
pipe_df (zvt.contract.factor.Factor attribute)
(zvt.contract.factor.ScoreFactor attribute)
(zvt.factors.fundamental.finance_factor.FinanceBaseFactor attribute)
(zvt.factors.fundamental.finance_factor.GoodCompanyFactor attribute)
(zvt.factors.ma.ma_factor.CrossMaFactor attribute)
(zvt.factors.ma.ma_factor.CrossMaVolumeFactor attribute)
(zvt.factors.ma.ma_factor.MaFactor attribute)
(zvt.factors.ma.ma_factor.VolumeUpMaFactor attribute)
(zvt.factors.ma.ma_stats_factor.MaStatsFactor attribute)
(zvt.factors.ma.ma_stats_factor.TFactor attribute)
(zvt.factors.ma.top_bottom_factor.TopBottomFactor attribute)
(zvt.factors.macd.macd_factor.BullFactor attribute)
(zvt.factors.macd.macd_factor.GoldCrossFactor attribute)
(zvt.factors.macd.macd_factor.KeepBullFactor attribute)
(zvt.factors.macd.macd_factor.LiveOrDeadFactor attribute)
(zvt.factors.macd.macd_factor.MacdFactor attribute)
(zvt.factors.technical_factor.TechnicalFactor attribute)
(zvt.factors.z.z_factor.ZFactor attribute)
point_in_range() (in module zvt.factors.algorithm)
populate_sh_etf_type() (zvt.recorders.exchange.exchange_etf_meta_recorder.ChinaETFListSpider static metho
Portfolio (class in zvt.contract.schema)
portfolio_relate_stock() (in module zvt.api.portfolio)
PortfolioStock (class in zvt.contract.schema)
PortfolioStockHistory (class in zvt.contract.schema)
Position (class in zvt.trader.trader_schemas)
PositionOverflowError
positions (zvt.trader.trader_schemas.AccountStats attribute)
PositionSchema (class in zvt.trader.sim_account)
PositionSchema.Meta (class in zvt.trader.sim_account)
PositionType (class in zvt.common.trading_models)
pre_month() (in module zvt.utils.time_utils)
pre_month_end_date() (in module zvt.utils.time_utils)
pre_month_start_date() (in module zvt.utils.time_utils)
price (zvt.domain.fundamental.trading.BigDealTrading attribute)
(zvt.domain.fundamental.trading.ManagerTrading attribute)
(zvt.domain.meta.stock_meta.StockDetail attribute)
(zvt.domain.quotes.stock.stock_quote.StockQuote attribute)
(zvt.trading.trading_models.StockQuoteModel attribute)
private_equity (zvt.contract.ActorType attribute)

profile (zvt.domain.meta.stock_meta.StockDetail attribute)
profit (zvt.trader.trader_schemas.AccountStats attribute)
(zvt.trader.trader_schemas.Position attribute)
profit_rate (zvt.trader.trader_schemas.AccountStats attribute)
(zvt.trader.trader_schemas.Position attribute)
proportion (zvt.contract.schema.PortfolioStockHistory attribute)
(zvt.domain.meta.etf_meta.EtfStock attribute)
(zvt.domain.meta.fund_meta.FundStock attribute)
(zvt.domain.meta.index_meta.IndexStock attribute)

Q

qfii (zvt.contract.ActorType attribute)
qfq (zvt.contract.AdjustType attribute)
QmtError
QuantileScorer (class in zvt.factors.algorithm)
query_data() (in module zvt.rest.data)
(zvt.contract.schema.ActorEntity class method)
(zvt.contract.schema.ActorMeetTradable class method)
(zvt.contract.schema.Entity class method)
(zvt.contract.schema.Mixin class method)
(zvt.contract.schema.NormalEntityMixin class method)
(zvt.contract.schema.NormalMixin class method)
(zvt.contract.schema.Portfolio class method)
(zvt.contract.schema.PortfolioStock class method)
(zvt.contract.schema.PortfolioStockHistory class method)
(zvt.contract.schema.TradableEntity class method)
(zvt.contract.schema.TradableMeetActor class method)
(zvt.contract.zvt_info.FactorState class method)
(zvt.contract.zvt_info.RecorderState class method)
(zvt.contract.zvt_info.StateMixin class method)
(zvt.contract.zvt_info.TaggerState class method)
(zvt.domain.actor.actor_meta.ActorMeta class method)
(zvt.domain.actor.stock_actor.StockActorSummary class method)
(zvt.domain.actor.stock_actor.StockInstitutionalInvestorHolder class method)
(zvt.domain.actor.stock_actor.StockTopTenFreeHolder class method)
(zvt.domain.actor.stock_actor.StockTopTenHolder class method)
(zvt.domain.emotion.emotion.Emotion class method)
(zvt.domain.emotion.emotion.LimitDownInfo class method)
(zvt.domain.emotion.emotion.LimitUpInfo class method)
(zvt.domain.fundamental.dividend_financing.DividendDetail class method)
(zvt.domain.fundamental.dividend_financing.DividendFinancing class method)
(zvt.domain.fundamental.dividend_financing.RightsIssueDetail class method)
(zvt.domain.fundamental.dividend_financing.SpoDetail class method)
(zvt.domain.fundamental.finance.BalanceSheet class method)
(zvt.domain.fundamental.finance.CashFlowStatement class method)
(zvt.domain.fundamental.finance.FinanceFactor class method)
(zvt.domain.fundamental.finance.IncomeStatement class method)
(zvt.domain.fundamental.trading.BigDealTrading class method)
(zvt.domain.fundamental.trading.DragonAndTiger class method)
(zvt.domain.fundamental.trading.HolderTrading class method)
query_factor_result() (in m
zvt.factors.factor_service)
(in module zvt.rest.fac
query_kdata() (in module
zvt.factors.factor_service)
(in module zvt.rest.fac
query_simple_stock_tags()
zvt.rest.work)
query_stock_quotes() (in m
zvt.trading.trading_service
query_stock_tag_stats() (in
zvt.rest.work)
(in module zvt.tag.tag
query_stock_tags() (in moc
query_tag_quotes() (in moc
zvt.trading.trading_service
query_trading_plan() (in m
zvt.rest.trading)
(in module zvt.trading
QuerySimpleStockTagsMo
zvt.tag.tag_models)
QueryStockQuoteModel (c
zvt.trading.trading_models
QueryStockQuoteSetting (c
zvt.trading.trading_schema
QueryStockQuoteSettingM
zvt.trading.trading_models
QueryStockTagsModel (cla
zvt.tag.tag_models)
QueryStockTagStatsModel
zvt.tag.tag_models)
QueryTagQuoteModel (cla
zvt.trading.trading_models
QueryTradingPlanModel (c
zvt.trading.trading_models
quick_ratio
(zvt.domain.fundamental.fi
attribute)

(zvt.domain.fundamental.trading.ManagerTrading class method)
(zvt.domain.fundamental.trading.MarginTrading class method)
(zvt.domain.fundamental.valuation.EtfValuation class method)
(zvt.domain.fundamental.valuation.StockValuation class method)
(zvt.domain.macro.macro.Economy class method)
(zvt.domain.macro.monetary.TreasuryYield class method)
(zvt.domain.meta.block_meta.Block class method)
(zvt.domain.meta.block_meta.BlockStock class method)
(zvt.domain.meta.country_meta.Country class method)
(zvt.domain.meta.currency_meta.Currency class method)
(zvt.domain.meta.etf_meta.Etf class method)
(zvt.domain.meta.etf_meta.EtfStock class method)
(zvt.domain.meta.fund_meta.Fund class method)
(zvt.domain.meta.fund_meta.FundStock class method)
(zvt.domain.meta.future_meta.Future class method)
(zvt.domain.meta.index_meta.Index class method)
(zvt.domain.meta.index_meta.IndexStock class method)
(zvt.domain.meta.indexus_meta.Indexus class method)
(zvt.domain.meta.stock_meta.Stock class method)
(zvt.domain.meta.stock_meta.StockDetail class method)
(zvt.domain.meta.stockhk_meta.Stockhk class method)
(zvt.domain.meta.stockus_meta.Stockus class method)
(zvt.domain.misc.holder.HkHolder class method)
(zvt.domain.misc.holder.InstitutionalInvestorHolder class method)
(zvt.domain.misc.holder.TopTenHolder class method)
(zvt.domain.misc.holder.TopTenTradableHolder class method)
(zvt.domain.misc.money_flow.BlockMoneyFlow class method)
(zvt.domain.misc.money_flow.IndexMoneyFlow class method)
(zvt.domain.misc.money_flow.StockMoneyFlow class method)
(zvt.domain.misc.overall.CrossMarketSummary class method)
(zvt.domain.misc.overall.MarginTradingSummary class method)
(zvt.domain.misc.overall.StockSummary class method)
(zvt.domain.misc.stock_news.StockNews class method)
(zvt.domain.quotes.block.block_1d_kdata.Block1dKdata class method)
(zvt.domain.quotes.block.block_1mon_kdata.Block1monKdata class method)
(zvt.domain.quotes.block.block_1wk_kdata.Block1wkKdata class method)
(zvt.domain.quotes.BlockKdataCommon class method)
(zvt.domain.quotes.currency.currency_1d_kdata.Currency1dKdata class method)
(zvt.domain.quotes.CurrencyKdataCommon class method)
(zvt.domain.quotes.etf.etf_1d_kdata.Etf1dKdata class method)
(zvt.domain.quotes.EtfKdataCommon class method)
(zvt.domain.quotes.future.future_1d_kdata.Future1dKdata class method)
(zvt.domain.quotes.FutureKdataCommon class method)
(zvt.domain.quotes.index.index_1d_kdata.Index1dKdata class method)
(zvt.domain.quotes.index.index_1wk_kdata.Index1wkKdata class method)
(zvt.domain.quotes.IndexKdataCommon class method)
(zvt.domain.quotes.indexus.indexus_1d_kdata.Indexus1dKdata class method)
(zvt.domain.quotes.IndexusKdataCommon class method)
(zvt.domain.quotes.KdataCommon class method)
(zvt.domain.quotes.stock.stock_15m_hfq_kdata.Stock15mHfqKdata class method)

(zvt.domain.quotes.stock.stock_15m_kdata.Stock15mKdata class method)
(zvt.domain.quotes.stock.stock_1d_hfq_kdata.Stock1dHfqKdata class method)
(zvt.domain.quotes.stock.stock_1d_kdata.Stock1dKdata class method)
(zvt.domain.quotes.stock.stock_1h_hfq_kdata.Stock1hHfqKdata class method)
(zvt.domain.quotes.stock.stock_1h_kdata.Stock1hKdata class method)
(zvt.domain.quotes.stock.stock_1m_hfq_kdata.Stock1mHfqKdata class method)
(zvt.domain.quotes.stock.stock_1m_kdata.Stock1mKdata class method)
(zvt.domain.quotes.stock.stock_1mon_hfq_kdata.Stock1monHfqKdata class method)
(zvt.domain.quotes.stock.stock_1mon_kdata.Stock1monKdata class method)
(zvt.domain.quotes.stock.stock_1wk_hfq_kdata.Stock1wkHfqKdata class method)
(zvt.domain.quotes.stock.stock_1wk_kdata.Stock1wkKdata class method)
(zvt.domain.quotes.stock.stock_30m_hfq_kdata.Stock30mHfqKdata class method)
(zvt.domain.quotes.stock.stock_30m_kdata.Stock30mKdata class method)
(zvt.domain.quotes.stock.stock_4h_hfq_kdata.Stock4hHfqKdata class method)
(zvt.domain.quotes.stock.stock_4h_kdata.Stock4hKdata class method)
(zvt.domain.quotes.stock.stock_5m_hfq_kdata.Stock5mHfqKdata class method)
(zvt.domain.quotes.stock.stock_5m_kdata.Stock5mKdata class method)
(zvt.domain.quotes.stock.stock_quote.StockQuote class method)
(zvt.domain.quotes.stockhk.stockhk_1d_hfq_kdata.Stockhk1dHfqKdata class method)
(zvt.domain.quotes.stockhk.stockhk_1d_kdata.Stockhk1dKdata class method)
(zvt.domain.quotes.StockhkKdataCommon class method)
(zvt.domain.quotes.StockKdataCommon class method)
(zvt.domain.quotes.stockus.stockus_1d_hfq_kdata.Stockus1dHfqKdata class method)
(zvt.domain.quotes.stockus.stockus_1d_kdata.Stockus1dKdata class method)
(zvt.domain.quotes.StockusKdataCommon class method)
(zvt.domain.quotes.TickCommon class method)
(zvt.domain.quotes.trade_day.StockTradeDay class method)
(zvt.factors.ma.domain.common.MaStatsFactorCommon class method)
(zvt.factors.ma.domain.stock_1d_ma_factor.Stock1dMaFactor class method)
(zvt.factors.ma.domain.stock_1d_ma_stats_factor.Stock1dMaStatsFactor class method)
(zvt.factors.top_stocks.TopStocks class method)
(zvt.factors.z.domain.common.ZFactorCommon class method)
(zvt.factors.z.domain.stock_1d_z_factor.Stock1dZFactor class method)
(zvt.factors.z.domain.stock_1wk_z_factor.Stock1wkZFactor class method)
(zvt.tag.tag_schemas.HiddenTagInfo class method)
(zvt.tag.tag_schemas.MainTagInfo class method)
(zvt.tag.tag_schemas.StockPoolInfo class method)

(zvt.tag.tag_schemas.StockPools class method)
(zvt.tag.tag_schemas.StockSystemTags class method)
(zvt.tag.tag_schemas.StockTags class method)
(zvt.tag.tag_schemas.SubTagInfo class method)
(zvt.tag.tag_schemas.TagStats class method)
(zvt.trader.trader_schemas.AccountStats class method)
(zvt.trader.trader_schemas.Order class method)
(zvt.trader.trader_schemas.Position class method)
(zvt.trader.trader_schemas.TraderInfo class method)
(zvt.trading.trading_schemas.QueryStockQuoteSetting class method)
(zvt.trading.trading_schemas.TradingPlan class method)

R

raised_fund (zvt.contract.ActorType attribute)
raising_fund (zvt.domain.meta.stock_meta.StockDetail attribute)
RankScorer (class in zvt.factors.algorithm)
rd_costs (zvt.domain.fundamental.finance.IncomeStatement attribute)
read_csv() (in module zvt.utils.utils)
real_estate_investment (zvt.domain.fundamental.finance.BalanceSheet attribute)
reason (zvt.domain.emotion.LimitUpInfo attribute)
(zvt.domain.fundamental.trading.DragonAndTiger attribute)
receivables_turnover (zvt.domain.fundamental.finance.FinanceFactor attribute)
receivables_turnover_days (zvt.domain.fundamental.finance.FinanceFactor attribute)
recent_acquisition (zvt.tag.tag_models.StockTagDetailsModel attribute)
(zvt.tag.tag_schemas.StockSystemTags attribute)
recent_additional_or_rights_issue (zvt.tag.tag_models.StockTagDetailsModel attribute)
(zvt.tag.tag_schemas.StockSystemTags attribute)
recent_dragon_and_tiger_count (zvt.tag.tag_models.StockTagDetailsModel attribute)
(zvt.tag.tag_schemas.StockSystemTags attribute)
recent_negative_earnings_news (zvt.tag.tag_models.StockTagDetailsModel attribute)
(zvt.tag.tag_schemas.StockSystemTags attribute)
recent_negative_news (zvt.tag.tag_models.StockTagDetailsModel attribute)
(zvt.tag.tag_schemas.StockSystemTags attribute)
recent_news_summary (zvt.tag.tag_models.StockTagDetailsModel attribute)
(zvt.tag.tag_schemas.StockSystemTags attribute)
recent_positive_earnings_news (zvt.tag.tag_models.StockTagDetailsModel attribute)
(zvt.tag.tag_schemas.StockSystemTags attribute)
recent_positive_news (zvt.tag.tag_models.StockTagDetailsModel attribute)
(zvt.tag.tag_schemas.StockSystemTags attribute)
recent_reduction (zvt.tag.tag_models.StockTagDetailsModel attribute)
(zvt.tag.tag_schemas.StockSystemTags attribute)
recent_unlock (zvt.tag.tag_models.StockTagDetailsModel attribute)
(zvt.tag.tag_schemas.StockSystemTags attribute)
recent_violation_alert (zvt.tag.tag_models.StockTagDetailsModel attribute)
(zvt.tag.tag_schemas.StockSystemTags attribute)
recent_year_date() (in module zvt.utils.time_utils)
record() (zvt.contract.recorder.FixedCycleDataRecorder method)
(zvt.contract.recorder.TimeSeriesDataRecorder method)
(zvt.contract.recorder.TimestampsDataRecorder method)
(zvt.recorders.eastmoney.common.EastmoneyMoreDataRecorder method)
(zvt.recorders.eastmoney.common.EastmoneyPageabeDataRecorder method)
(zvt.recorders.eastmoney.common.EastmoneyTimestampsDataRecorder method)
(zvt.recorders.eastmoney.dividend_financing.eastmoney_dividend_detail_recorder.DividendDetailRecord method)

(zvt.recorders.eastmoney.dividend_financing.eastmoney_dividend_financing_recorder.DividendFinancing
method)
(zvt.recorders.eastmoney.dividend_financing.eastmoney_rights_issue_detail_recorder.RightsIssueDetailR
method)
(zvt.recorders.eastmoney.dividend_financing.eastmoney_spo_detail_recorder.SPODetailRecorder method
(zvt.recorders.eastmoney.finance.base_china_stock_finance_recorder.BaseChinaStockFinanceRecorder m
(zvt.recorders.eastmoney.finance.eastmoney_balance_sheet_recorder.ChinaStockBalanceSheetRecorder n
(zvt.recorders.eastmoney.finance.eastmoney_cash_flow_recorder.ChinaStockCashFlowRecorder method)
(zvt.recorders.eastmoney.finance.eastmoney_finance_factor_recorder.ChinaStockFinanceFactorRecorder
(zvt.recorders.eastmoney.finance.eastmoney_income_statement_recorder.ChinaStockIncomeStatementRe
method)
(zvt.recorders.eastmoney.holder.eastmoney_top_ten_holder_recorder.TopTenHolderRecorder method)
(zvt.recorders.eastmoney.holder.eastmoney_top_ten_tradable_holder_recorder.TopTenTradableHolderRec
method)
(zvt.recorders.eastmoney.meta.eastmoney_block_meta_recorder.EastmoneyBlockStockRecorder method)
(zvt.recorders.eastmoney.trading.eastmoney_holder_trading_recorder.HolderTradingRecorder method)
(zvt.recorders.eastmoney.trading.eastmoney_manager_trading_recorder.ManagerTradingRecorder method
(zvt.recorders.em.actor.em_stock_actor_summary_recorder.EMStockActorSummaryRecorder method)
(zvt.recorders.em.actor.em_stock_ii_recorder.EMStockIIRecorder method)
(zvt.recorders.em.actor.em_stock_top_ten_free_recorder.EMStockTopTenFreeRecorder method)
(zvt.recorders.em.actor.em_stock_top_ten_recorder.EMStockTopTenRecorder method)
(zvt.recorders.em.macro.em_treasury_yield_recorder.EMTreasuryYieldRecorder method)
(zvt.recorders.em.meta.em_block_meta_recorder.EMBlockStockRecorder method)
(zvt.recorders.em.misc.em_stock_news_recorder.EMStockNewsRecorder method)
(zvt.recorders.em.quotes.em_kdata_recorder.BaseEMStockKdataRecorder method)
(zvt.recorders.em.quotes.em_kdata_recorder.EMBlockKdataRecorder method)
(zvt.recorders.em.quotes.em_kdata_recorder.EMCurrencyKdataRecorder method)
(zvt.recorders.em.quotes.em_kdata_recorder.EMFutureKdataRecorder method)
(zvt.recorders.em.quotes.em_kdata_recorder.EMIndexKdataRecorder method)
(zvt.recorders.em.quotes.em_kdata_recorder.EMIndexusKdataRecorder method)
(zvt.recorders.em.quotes.em_kdata_recorder.EMStockhkKdataRecorder method)
(zvt.recorders.em.quotes.em_kdata_recorder.EMStockKdataRecorder method)
(zvt.recorders.em.quotes.em_kdata_recorder.EMStockusKdataRecorder method)
(zvt.recorders.em.trading.em_dragon_and_tiger_recorder.EMDragonAndTigerRecorder method)
(zvt.recorders.exchange.exchange_index_stock_recorder.ExchangeIndexStockRecorder method)
(zvt.recorders.exchange.exchange_stock_summary_recorder.ExchangeStockSummaryRecorder method)
(zvt.recorders.joinquant.fundamental.jq_etf_valuation_recorder.JqChinaEtfValuationRecorder method)
(zvt.recorders.joinquant.fundamental.jq_margin_trading_recorder.MarginTradingRecorder method)
(zvt.recorders.joinquant.fundamental.jq_stock_valuation_recorder.JqChinaStockValuationRecorder metho
(zvt.recorders.joinquant.meta.jq_fund_meta_recorder.JqChinaFundStockRecorder method)
(zvt.recorders.joinquant.meta.jq_stock_meta_recorder.JqChinaStockEtfPortfolioRecorder method)
(zvt.recorders.joinquant.meta.jq_trade_day_recorder.StockTradeDayRecorder method)
(zvt.recorders.joinquant.misc.jq_hk_holder_recorder.JoinquantHkHolderRecorder method)
(zvt.recorders.joinquant.misc.jq_index_money_flow_recorder.JoinquantIndexMoneyFlowRecorder metho
(zvt.recorders.joinquant.misc.jq_stock_money_flow_recorder.JoinquantStockMoneyFlowRecorder metho
(zvt.recorders.joinquant.overall.jq_cross_market_recorder.CrossMarketSummaryRecorder method)
(zvt.recorders.joinquant.overall.jq_margin_trading_recorder.MarginTradingSummaryRecorder method)
(zvt.recorders.joinquant.overall.jq_stock_summary_recorder.StockSummaryRecorder method)
(zvt.recorders.joinquant.quotes.jq_index_kdata_recorder.JqChinaIndexKdataRecorder method)
(zvt.recorders.joinquant.quotes.jq_stock_kdata_recorder.JqChinaStockKdataRecorder method)
(zvt.recorders.sina.meta.sina_block_recorder.SinaChinaBlockStockRecorder method)
(zvt.recorders.sina.money_flow.sina_block_money_flow_recorder.SinaBlockMoneyFlowRecorder metho
(zvt.recorders.sina.money_flow.sina_stock_money_flow_recorder.SinaStockMoneyFlowRecorder metho
(zvt.recorders.sina.quotes.sina_etf_kdata_recorder.ChinaETFDayKdataRecorder method)
(zvt.recorders.sina.quotes.sina_index_kdata_recorder.ChinaIndexDayKdataRecorder method)
(zvt.recorders.wb.wb_economy_recorder.WBEconomyRecorder method)

record_data() (zvt.contract.schema.ActorEntity class method)
(zvt.contract.schema.ActorMeetTradable class method)
(zvt.contract.schema.Entity class method)
(zvt.contract.schema.Mixin class method)
(zvt.contract.schema.NormalEntityMixin class method)
(zvt.contract.schema.NormalMixin class method)
(zvt.contract.schema.Portfolio class method)
(zvt.contract.schema.PortfolioStock class method)
(zvt.contract.schema.PortfolioStockHistory class method)
(zvt.contract.schema.TradableEntity class method)
(zvt.contract.schema.TradableMeetActor class method)
(zvt.contract.zvt_info.FactorState class method)
(zvt.contract.zvt_info.RecorderState class method)
(zvt.contract.zvt_info.StateMixin class method)
(zvt.contract.zvt_info.TaggerState class method)
(zvt.domain.actor.actor_meta.ActorMeta class method)
(zvt.domain.actor.stock_actor.StockActorSummary class method)
(zvt.domain.actor.stock_actor.StockInstitutionalInvestorHolder class method)
(zvt.domain.actor.stock_actor.StockTopTenFreeHolder class method)
(zvt.domain.actor.stock_actor.StockTopTenHolder class method)
(zvt.domain.emotion.emotion.Emotion class method)
(zvt.domain.emotion.emotion.LimitDownInfo class method)
(zvt.domain.emotion.emotion.LimitUpInfo class method)
(zvt.domain.fundamental.dividend_financing.DividendDetail class method)
(zvt.domain.fundamental.dividend_financing.DividendFinancing class method)
(zvt.domain.fundamental.dividend_financing.RightsIssueDetail class method)
(zvt.domain.fundamental.dividend_financing.SpoDetail class method)
(zvt.domain.fundamental.finance.BalanceSheet class method)
(zvt.domain.fundamental.finance.CashFlowStatement class method)
(zvt.domain.fundamental.finance.FinanceFactor class method)
(zvt.domain.fundamental.finance.IncomeStatement class method)
(zvt.domain.fundamental.trading.BigDealTrading class method)
(zvt.domain.fundamental.trading.DragonAndTiger class method)
(zvt.domain.fundamental.trading.HolderTrading class method)
(zvt.domain.fundamental.trading.ManagerTrading class method)
(zvt.domain.fundamental.trading.MarginTrading class method)
(zvt.domain.fundamental.valuation.EtfValuation class method)
(zvt.domain.fundamental.valuation.StockValuation class method)
(zvt.domain.macro.macro.Economy class method)
(zvt.domain.macro.monetary.TreasuryYield class method)
(zvt.domain.meta.block_meta.Block class method)
(zvt.domain.meta.block_meta.BlockStock class method)
(zvt.domain.meta.country_meta.Country class method)
(zvt.domain.meta.currency_meta.Currency class method)
(zvt.domain.meta.etf_meta.Etf class method)
(zvt.domain.meta.etf_meta.EtfStock class method)
(zvt.domain.meta.fund_meta.Fund class method)
(zvt.domain.meta.fund_meta.FundStock class method)
(zvt.domain.meta.future_meta.Future class method)
(zvt.domain.meta.index_meta.Index class method)
(zvt.domain.meta.index_meta.IndexStock class method)
(zvt.domain.meta.indexus_meta.Indexus class method)
(zvt.domain.meta.stock_meta.Stock class method)
(zvt.domain.meta.stock_meta.StockDetail class method)
(zvt.domain.meta.stockhk_meta.Stockhk class method)
(zvt.domain.meta.stockus_meta.Stockus class method)

(zvt.domain.misc.holder.HkHolder class method)
(zvt.domain.misc.holder.InstitutionalInvestorHolder class method)
(zvt.domain.misc.holder.TopTenHolder class method)
(zvt.domain.misc.holder.TopTenTradableHolder class method)
(zvt.domain.misc.money_flow.BlockMoneyFlow class method)
(zvt.domain.misc.money_flow.IndexMoneyFlow class method)
(zvt.domain.misc.money_flow.StockMoneyFlow class method)
(zvt.domain.misc.overall.CrossMarketSummary class method)
(zvt.domain.misc.overall.MarginTradingSummary class method)
(zvt.domain.misc.overall.StockSummary class method)
(zvt.domain.misc.stock_news.StockNews class method)
(zvt.domain.quotes.block.block_1d_kdata.Block1dKdata class method)
(zvt.domain.quotes.block.block_1mon_kdata.Block1monKdata class method)
(zvt.domain.quotes.block.block_1wk_kdata.Block1wkKdata class method)
(zvt.domain.quotes.BlockKdataCommon class method)
(zvt.domain.quotes.currency.currency_1d_kdata.Currency1dKdata class method)
(zvt.domain.quotes.CurrencyKdataCommon class method)
(zvt.domain.quotes.etf.etf_1d_kdata.Etf1dKdata class method)
(zvt.domain.quotes.EtfKdataCommon class method)
(zvt.domain.quotes.future.future_1d_kdata.Future1dKdata class method)
(zvt.domain.quotes.FutureKdataCommon class method)
(zvt.domain.quotes.index.index_1d_kdata.Index1dKdata class method)
(zvt.domain.quotes.index.index_1wk_kdata.Index1wkKdata class method)
(zvt.domain.quotes.IndexKdataCommon class method)
(zvt.domain.quotes.indexus.indexus_1d_kdata.Indexus1dKdata class method)
(zvt.domain.quotes.IndexusKdataCommon class method)
(zvt.domain.quotes.KdataCommon class method)
(zvt.domain.quotes.stock.stock_15m_hfq_kdata.Stock15mHfqKdata class method)
(zvt.domain.quotes.stock.stock_15m_kdata.Stock15mKdata class method)
(zvt.domain.quotes.stock.stock_1d_hfq_kdata.Stock1dHfqKdata class method)
(zvt.domain.quotes.stock.stock_1d_kdata.Stock1dKdata class method)
(zvt.domain.quotes.stock.stock_1h_hfq_kdata.Stock1hHfqKdata class method)
(zvt.domain.quotes.stock.stock_1h_kdata.Stock1hKdata class method)
(zvt.domain.quotes.stock.stock_1m_hfq_kdata.Stock1mHfqKdata class method)
(zvt.domain.quotes.stock.stock_1m_kdata.Stock1mKdata class method)
(zvt.domain.quotes.stock.stock_1mon_hfq_kdata.Stock1monHfqKdata class method)
(zvt.domain.quotes.stock.stock_1mon_kdata.Stock1monKdata class method)
(zvt.domain.quotes.stock.stock_1wk_hfq_kdata.Stock1wkHfqKdata class method)
(zvt.domain.quotes.stock.stock_1wk_kdata.Stock1wkKdata class method)
(zvt.domain.quotes.stock.stock_30m_hfq_kdata.Stock30mHfqKdata class method)
(zvt.domain.quotes.stock.stock_30m_kdata.Stock30mKdata class method)
(zvt.domain.quotes.stock.stock_4h_hfq_kdata.Stock4hHfqKdata class method)
(zvt.domain.quotes.stock.stock_4h_kdata.Stock4hKdata class method)
(zvt.domain.quotes.stock.stock_5m_hfq_kdata.Stock5mHfqKdata class method)
(zvt.domain.quotes.stock.stock_5m_kdata.Stock5mKdata class method)
(zvt.domain.quotes.stock.stock_quote.StockQuote class method)
(zvt.domain.quotes.stockhk.stockhk_1d_hfq_kdata.Stockhk1dHfqKdata class method)
(zvt.domain.quotes.stockhk.stockhk_1d_kdata.Stockhk1dKdata class method)
(zvt.domain.quotes.StockhkKdataCommon class method)
(zvt.domain.quotes.StockKdataCommon class method)
(zvt.domain.quotes.stockus.stockus_1d_hfq_kdata.Stockus1dHfqKdata class method)
(zvt.domain.quotes.stockus.stockus_1d_kdata.Stockus1dKdata class method)
(zvt.domain.quotes.StockusKdataCommon class method)
(zvt.domain.quotes.TickCommon class method)
(zvt.domain.quotes.trade_day.StockTradeDay class method)
(zvt.factors.ma.domain.common.MaStatsFactorCommon class method)

(zvt.factors.ma.domain.stock_1d_ma_factor.Stock1dMaFactor class method)
(zvt.factors.ma.domain.stock_1d_ma_stats_factor.Stock1dMaStatsFactor class method)
(zvt.factors.top_stocks.TopStocks class method)
(zvt.factors.z.domain.common.ZFactorCommon class method)
(zvt.factors.z.domain.stock_1d_z_factor.Stock1dZFactor class method)
(zvt.factors.z.domain.stock_1wk_z_factor.Stock1wkZFactor class method)
(zvt.tag.tag_schemas.HiddenTagInfo class method)
(zvt.tag.tag_schemas.MainTagInfo class method)
(zvt.tag.tag_schemas.StockPoolInfo class method)
(zvt.tag.tag_schemas.StockPools class method)
(zvt.tag.tag_schemas.StockSystemTags class method)
(zvt.tag.tag_schemas.StockTags class method)
(zvt.tag.tag_schemas.SubTagInfo class method)
(zvt.tag.tag_schemas.TagStats class method)
(zvt.trader.trader_schemas.AccountStats class method)
(zvt.trader.trader_schemas.Order class method)
(zvt.trader.trader_schemas.Position class method)
(zvt.trader.trader_schemas.TraderInfo class method)
(zvt.trading.trading_schemas.QueryStockQuoteSetting class method)
(zvt.trading.trading_schemas.TradingPlan class method)
record_date (zvt.domain.fundamental.dividend_financing.DividendDetail attribute)
Recorder (class in zvt.contract.recorder)
RecorderState (class in zvt.contract.zvt_info)
Rect (class in zvt.contract.drawer)
rects (zvt.contract.drawer.Drawer attribute)
refresh_all_main_tag_by_sub_tag() (in module zvt.tag.tag_service)
refresh_main_tag_by_sub_tag() (in module zvt.tag.tag_service)
region (zvt.domain.meta.country_meta.Country attribute)
register_entity() (in module zvt.contract.register)
register_provider() (zvt.contract.schema.ActorEntity class method)
(zvt.contract.schema.ActorMeetTradable class method)
(zvt.contract.schema.Entity class method)
(zvt.contract.schema.Mixin class method)
(zvt.contract.schema.NormalEntityMixin class method)
(zvt.contract.schema.NormalMixin class method)
(zvt.contract.schema.Portfolio class method)
(zvt.contract.schema.PortfolioStock class method)
(zvt.contract.schema.PortfolioStockHistory class method)
(zvt.contract.schema.TradableEntity class method)
(zvt.contract.schema.TradableMeetActor class method)
(zvt.contract.zvt_info.FactorState class method)
(zvt.contract.zvt_info.RecorderState class method)
(zvt.contract.zvt_info.StateMixin class method)
(zvt.contract.zvt_info.TaggerState class method)
(zvt.domain.actor.actor_meta.ActorMeta class method)
(zvt.domain.actor.stock_actor.StockActorSummary class method)
(zvt.domain.actor.stock_actor.StockInstitutionalInvestorHolder class method)
(zvt.domain.actor.stock_actor.StockTopTenFreeHolder class method)
(zvt.domain.actor.stock_actor.StockTopTenHolder class method)
(zvt.domain.emotion.emotion.Emotion class method)
(zvt.domain.emotion.emotion.LimitDownInfo class method)
(zvt.domain.emotion.emotion.LimitUpInfo class method)
(zvt.domain.fundamental.dividend_financing.DividendDetail class method)
(zvt.domain.fundamental.dividend_financing.DividendFinancing class method)
(zvt.domain.fundamental.dividend_financing.RightsIssueDetail class method)
(zvt.domain.fundamental.dividend_financing.SpoDetail class method)

(zvt.domain.fundamental.finance.BalanceSheet class method)
(zvt.domain.fundamental.finance.CashFlowStatement class method)
(zvt.domain.fundamental.finance.FinanceFactor class method)
(zvt.domain.fundamental.finance.IncomeStatement class method)
(zvt.domain.fundamental.trading.BigDealTrading class method)
(zvt.domain.fundamental.trading.DragonAndTiger class method)
(zvt.domain.fundamental.trading.HolderTrading class method)
(zvt.domain.fundamental.trading.ManagerTrading class method)
(zvt.domain.fundamental.trading.MarginTrading class method)
(zvt.domain.fundamental.valuation.EtfValuation class method)
(zvt.domain.fundamental.valuation.StockValuation class method)
(zvt.domain.macro.macro.Economy class method)
(zvt.domain.macro.monetary.TreasuryYield class method)
(zvt.domain.meta.block_meta.Block class method)
(zvt.domain.meta.block_meta.BlockStock class method)
(zvt.domain.meta.country_meta.Country class method)
(zvt.domain.meta.currency_meta.Currency class method)
(zvt.domain.meta.etf_meta.Etf class method)
(zvt.domain.meta.etf_meta.EtfStock class method)
(zvt.domain.meta.fund_meta.Fund class method)
(zvt.domain.meta.fund_meta.FundStock class method)
(zvt.domain.meta.future_meta.Future class method)
(zvt.domain.meta.index_meta.Index class method)
(zvt.domain.meta.index_meta.IndexStock class method)
(zvt.domain.meta.indexus_meta.Indexus class method)
(zvt.domain.meta.stock_meta.Stock class method)
(zvt.domain.meta.stock_meta.StockDetail class method)
(zvt.domain.meta.stockhk_meta.Stockhk class method)
(zvt.domain.meta.stockus_meta.Stockus class method)
(zvt.domain.misc.holder.HkHolder class method)
(zvt.domain.misc.holder.InstitutionalInvestorHolder class method)
(zvt.domain.misc.holder.TopTenHolder class method)
(zvt.domain.misc.holder.TopTenTradableHolder class method)
(zvt.domain.misc.money_flow.BlockMoneyFlow class method)
(zvt.domain.misc.money_flow.IndexMoneyFlow class method)
(zvt.domain.misc.money_flow.StockMoneyFlow class method)
(zvt.domain.misc.overall.CrossMarketSummary class method)
(zvt.domain.misc.overall.MarginTradingSummary class method)
(zvt.domain.misc.overall.StockSummary class method)
(zvt.domain.misc.stock_news.StockNews class method)
(zvt.domain.quotes.block.block_1d_kdata.Block1dKdata class method)
(zvt.domain.quotes.block.block_1mon_kdata.Block1monKdata class method)
(zvt.domain.quotes.block.block_1wk_kdata.Block1wkKdata class method)
(zvt.domain.quotes.BlockKdataCommon class method)
(zvt.domain.quotes.currency.currency_1d_kdata.Currency1dKdata class method)
(zvt.domain.quotes.CurrencyKdataCommon class method)
(zvt.domain.quotes.etf.etf_1d_kdata.Etf1dKdata class method)
(zvt.domain.quotes.EtfKdataCommon class method)
(zvt.domain.quotes.future.future_1d_kdata.Future1dKdata class method)
(zvt.domain.quotes.FutureKdataCommon class method)
(zvt.domain.quotes.index.index_1d_kdata.Index1dKdata class method)
(zvt.domain.quotes.index.index_1wk_kdata.Index1wkKdata class method)
(zvt.domain.quotes.IndexKdataCommon class method)
(zvt.domain.quotes.indexus.indexus_1d_kdata.Indexus1dKdata class method)
(zvt.domain.quotes.IndexusKdataCommon class method)
(zvt.domain.quotes.KdataCommon class method)

(zvt.domain.quotes.stock.stock_15m_hfq_kdata.Stock15mHfqKdata class method)
(zvt.domain.quotes.stock.stock_15m_kdata.Stock15mKdata class method)
(zvt.domain.quotes.stock.stock_1d_hfq_kdata.Stock1dHfqKdata class method)
(zvt.domain.quotes.stock.stock_1d_kdata.Stock1dKdata class method)
(zvt.domain.quotes.stock.stock_1h_hfq_kdata.Stock1hHfqKdata class method)
(zvt.domain.quotes.stock.stock_1h_kdata.Stock1hKdata class method)
(zvt.domain.quotes.stock.stock_1m_hfq_kdata.Stock1mHfqKdata class method)
(zvt.domain.quotes.stock.stock_1m_kdata.Stock1mKdata class method)
(zvt.domain.quotes.stock.stock_1mon_hfq_kdata.Stock1monHfqKdata class method)
(zvt.domain.quotes.stock.stock_1mon_kdata.Stock1monKdata class method)
(zvt.domain.quotes.stock.stock_1wk_hfq_kdata.Stock1wkHfqKdata class method)
(zvt.domain.quotes.stock.stock_1wk_kdata.Stock1wkKdata class method)
(zvt.domain.quotes.stock.stock_30m_hfq_kdata.Stock30mHfqKdata class method)
(zvt.domain.quotes.stock.stock_30m_kdata.Stock30mKdata class method)
(zvt.domain.quotes.stock.stock_4h_hfq_kdata.Stock4hHfqKdata class method)
(zvt.domain.quotes.stock.stock_4h_kdata.Stock4hKdata class method)
(zvt.domain.quotes.stock.stock_5m_hfq_kdata.Stock5mHfqKdata class method)
(zvt.domain.quotes.stock.stock_5m_kdata.Stock5mKdata class method)
(zvt.domain.quotes.stock.stock_quote.StockQuote class method)
(zvt.domain.quotes.stockhk.stockhk_1d_hfq_kdata.Stockhk1dHfqKdata class method)
(zvt.domain.quotes.stockhk.stockhk_1d_kdata.Stockhk1dKdata class method)
(zvt.domain.quotes.StockhkKdataCommon class method)
(zvt.domain.quotes.StockKdataCommon class method)
(zvt.domain.quotes.stockus.stockus_1d_hfq_kdata.Stockus1dHfqKdata class method)
(zvt.domain.quotes.stockus.stockus_1d_kdata.Stockus1dKdata class method)
(zvt.domain.quotes.StockusKdataCommon class method)
(zvt.domain.quotes.TickCommon class method)
(zvt.domain.quotes.trade_day.StockTradeDay class method)
(zvt.factors.ma.domain.common.MaStatsFactorCommon class method)
(zvt.factors.ma.domain.stock_1d_ma_factor.Stock1dMaFactor class method)
(zvt.factors.ma.domain.stock_1d_ma_stats_factor.Stock1dMaStatsFactor class method)
(zvt.factors.top_stocks.TopStocks class method)
(zvt.factors.z.domain.common.ZFactorCommon class method)
(zvt.factors.z.domain.stock_1d_z_factor.Stock1dZFactor class method)
(zvt.factors.z.domain.stock_1wk_z_factor.Stock1wkZFactor class method)
(zvt.tag.tag_schemas.HiddenTagInfo class method)
(zvt.tag.tag_schemas.MainTagInfo class method)
(zvt.tag.tag_schemas.StockPoolInfo class method)
(zvt.tag.tag_schemas.StockPools class method)
(zvt.tag.tag_schemas.StockSystemTags class method)
(zvt.tag.tag_schemas.StockTags class method)
(zvt.tag.tag_schemas.SubTagInfo class method)
(zvt.tag.tag_schemas.TagStats class method)
(zvt.trader.trader_schemas.AccountStats class method)
(zvt.trader.trader_schemas.Order class method)
(zvt.trader.trader_schemas.Position class method)
(zvt.trader.trader_schemas.TraderInfo class method)
(zvt.trading.trading_schemas.QueryStockQuoteSetting class method)
(zvt.trading.trading_schemas.TradingPlan class method)

S

sales_costs (zvt.domain.fundamental.finance.IncomeStatement attribute)
sales_net_cash_flow_per_op_income (zvt.domain.fundamental.finance.FinanceFactor attribute)
scatter (zvt.contract.drawer.ChartType attribute)

sched_tasks() (in module zvt.sched.sched)
schemas (zvt.contract.context.Registry attribute)
scope (zvt.domain.IndexCategory attribute)
score() (zvt.contract.factor.Scorer method)
(zvt.factors.algorithm.QuantileScorer method)
(zvt.factors.algorithm.RankScorer method)
ScoreFactor (class in zvt.contract.factor)
Scorer (class in zvt.contract.factor)
sec_refund_value (zvt.domain.fundamental.trading.MarginTrading attribute)
sec_sell_value (zvt.domain.fundamental.trading.MarginTrading attribute)
sec_value (zvt.domain.fundamental.trading.MarginTrading attribute)
second_item_to_float() (in module zvt.utils.utils)
SelectMode (class in zvt.factors.target_selector)
sell() (in module zvt.rest.trading)
sell_broker (zvt.domain.fundamental.trading.BigDealTrading attribute)
sell_stocks() (in module zvt.trading.trading_service)
SellParameter (class in zvt.common.trading_models)
serve_layout() (in module zvt.main)
session (zvt.contract.recorder.EntityEventRecorder attribute)
(zvt.contract.recorder.FixedCycleDataRecorder attribute)
(zvt.contract.recorder.Recorder attribute)
(zvt.contract.recorder.TimeSeriesDataRecorder attribute)
(zvt.contract.recorder.TimestampsDataRecorder attribute)
(zvt.recorders.eastmoney.common.EastmoneyMoreDataRecorder attribute)
(zvt.recorders.eastmoney.common.EastmoneyPageabeDataRecorder attribute)
(zvt.recorders.eastmoney.common.EastmoneyTimestampsDataRecorder attribute)
(zvt.recorders.eastmoney.dividend_financing.eastmoney_dividend_detail_recorder.DividendDetailRecord
attribute)
(zvt.recorders.eastmoney.dividend_financing.eastmoney_dividend_financing_recorder.DividendFinancing
attribute)
(zvt.recorders.eastmoney.dividend_financing.eastmoney_rights_issue_detail_recorder.RightsIssueDetailR
attribute)
(zvt.recorders.eastmoney.dividend_financing.eastmoney_spo_detail_recorder.SPODetailRecorder attribut
(zvt.recorders.eastmoney.finance.base_china_stock_finance_recorder.BaseChinaStockFinanceRecorder at
(zvt.recorders.eastmoney.finance.eastmoney_balance_sheet_recorder.ChinaStockBalanceSheetRecorder a
(zvt.recorders.eastmoney.finance.eastmoney_cash_flow_recorder.ChinaStockCashFlowRecorder attribute
(zvt.recorders.eastmoney.finance.eastmoney_finance_factor_recorder.ChinaStockFinanceFactorRecorder_
(zvt.recorders.eastmoney.finance.eastmoney_income_statement_recorder.ChinaStockIncomeStatementRe
attribute)
(zvt.recorders.eastmoney.holder.eastmoney_stock_actor_recorder.EastmoneyActorRecorder attribute)
(zvt.recorders.eastmoney.holder.eastmoney_top_ten_holder_recorder.TopTenHolderRecorder attribute)
(zvt.recorders.eastmoney.holder.eastmoney_top_ten_tradable_holder_recorder.TopTenTradableHolderRec
attribute)
(zvt.recorders.eastmoney.meta.eastmoney_block_meta_recorder.EastmoneyBlockRecorder attribute)
(zvt.recorders.eastmoney.meta.eastmoney_block_meta_recorder.EastmoneyBlockStockRecorder attribute
(zvt.recorders.eastmoney.meta.eastmoney_stock_meta_recorder.EastmoneyStockDetailRecorder attribute
(zvt.recorders.eastmoney.meta.eastmoney_stock_meta_recorder.EastmoneyStockRecorder attribute)
(zvt.recorders.eastmoney.trading.eastmoney_holder_trading_recorder.HolderTradingRecorder attribute)
(zvt.recorders.eastmoney.trading.eastmoney_manager_trading_recorder.ManagerTradingRecorder attribut
(zvt.recorders.em.actor.em_stock_actor_summary_recorder.EMStockActorSummaryRecorder attribute)
(zvt.recorders.em.actor.em_stock_ii_recorder.EMStockIIRecorder attribute)
(zvt.recorders.em.actor.em_stock_top_ten_free_recorder.EMStockTopTenFreeRecorder attribute)
(zvt.recorders.em.actor.em_stock_top_ten_recorder.EMStockTopTenRecorder attribute)
(zvt.recorders.em.macro.em_treasury_yield_recorder.EMTreasuryYieldRecorder attribute)
(zvt.recorders.em.meta.em_block_meta_recorder.EMBlockRecorder attribute)
(zvt.recorders.em.meta.em_block_meta_recorder.EMBlockStockRecorder attribute)

(zvt.recorders.em.meta.em_currency_meta_recorder.EMCurrencyRecorder attribute)
(zvt.recorders.em.meta.em_future_meta_recorder.EMFutureRecorder attribute)
(zvt.recorders.em.meta.em_index_meta_recorder.EMIndexRecorder attribute)
(zvt.recorders.em.meta.em_indexus_meta_recorder.EMIndexusRecorder attribute)
(zvt.recorders.em.meta.em_stock_meta_recorder.EMStockRecorder attribute)
(zvt.recorders.em.meta.em_stockhk_meta_recorder.EMStockhkRecorder attribute)
(zvt.recorders.em.meta.em_stockus_meta_recorder.EMStockusRecorder attribute)
(zvt.recorders.em.misc.em_stock_news_recorder.EMStockNewsRecorder attribute)
(zvt.recorders.em.quotes.em_kdata_recorder.BaseEMStockKdataRecorder attribute)
(zvt.recorders.em.quotes.em_kdata_recorder.EMBlockKdataRecorder attribute)
(zvt.recorders.em.quotes.em_kdata_recorder.EMCurrencyKdataRecorder attribute)
(zvt.recorders.em.quotes.em_kdata_recorder.EMFutureKdataRecorder attribute)
(zvt.recorders.em.quotes.em_kdata_recorder.EMIndexKdataRecorder attribute)
(zvt.recorders.em.quotes.em_kdata_recorder.EMIndexusKdataRecorder attribute)
(zvt.recorders.em.quotes.em_kdata_recorder.EMStockhkKdataRecorder attribute)
(zvt.recorders.em.quotes.em_kdata_recorder.EMStockKdataRecorder attribute)
(zvt.recorders.em.quotes.em_kdata_recorder.EMStockusKdataRecorder attribute)
(zvt.recorders.em.trading.em_dragon_and_tiger_recorder.EMDragonAndTigerRecorder attribute)
(zvt.recorders.exchange.exchange_etf_meta_recorder.ChinaETFListSpider attribute)
(zvt.recorders.exchange.exchange_index_recorder.ExchangeIndexRecorder attribute)
(zvt.recorders.exchange.exchange_index_stock_recorder.ExchangeIndexStockRecorder attribute)
(zvt.recorders.exchange.exchange_stock_meta_recorder.ExchangeStockMetaRecorder attribute)
(zvt.recorders.exchange.exchange_stock_summary_recorder.ExchangeStockSummaryRecorder attribute)
(zvt.recorders.joinquant.fundamental.jq_etf_valuation_recorder.JqChinaEtfValuationRecorder attribute)
(zvt.recorders.joinquant.fundamental.jq_margin_trading_recorder.MarginTradingRecorder attribute)
(zvt.recorders.joinquant.fundamental.jq_stock_valuation_recorder.JqChinaStockValuationRecorder attribu
(zvt.recorders.joinquant.meta.jq_fund_meta_recorder.JqChinaFundRecorder attribute)
(zvt.recorders.joinquant.meta.jq_fund_meta_recorder.JqChinaFundStockRecorder attribute)
(zvt.recorders.joinquant.meta.jq_stock_meta_recorder.BaseJqChinaMetaRecorder attribute)
(zvt.recorders.joinquant.meta.jq_stock_meta_recorder.JqChinaEtfRecorder attribute)
(zvt.recorders.joinquant.meta.jq_stock_meta_recorder.JqChinaStockEtfPortfolioRecorder attribute)
(zvt.recorders.joinquant.meta.jq_stock_meta_recorder.JqChinaStockRecorder attribute)
(zvt.recorders.joinquant.meta.jq_trade_day_recorder.StockTradeDayRecorder attribute)
(zvt.recorders.joinquant.misc.jq_hk_holder_recorder.JoinquantHkHolderRecorder attribute)
(zvt.recorders.joinquant.misc.jq_index_money_flow_recorder.JoinquantIndexMoneyFlowRecorder attribu
(zvt.recorders.joinquant.misc.jq_stock_money_flow_recorder.JoinquantStockMoneyFlowRecorder attribu
(zvt.recorders.joinquant.overall.jq_cross_market_recorder.CrossMarketSummaryRecorder attribute)
(zvt.recorders.joinquant.overall.jq_margin_trading_recorder.MarginTradingSummaryRecorder attribute)
(zvt.recorders.joinquant.overall.jq_stock_summary_recorder.StockSummaryRecorder attribute)
(zvt.recorders.joinquant.quotes.jq_index_kdata_recorder.JqChinaIndexKdataRecorder attribute)
(zvt.recorders.joinquant.quotes.jq_stock_kdata_recorder.JqChinaStockKdataRecorder attribute)
(zvt.recorders.sina.meta.sina_block_recorder.SinaBlockRecorder attribute)
(zvt.recorders.sina.meta.sina_block_recorder.SinaChinaBlockStockRecorder attribute)
(zvt.recorders.sina.money_flow.sina_block_money_flow_recorder.SinaBlockMoneyFlowRecorder attribu
(zvt.recorders.sina.money_flow.sina_stock_money_flow_recorder.SinaStockMoneyFlowRecorder attribut
(zvt.recorders.sina.quotes.sina_etf_kdata_recorder.ChinaETFDayKdataRecorder attribute)
(zvt.recorders.sina.quotes.sina_index_kdata_recorder.ChinaIndexDayKdataRecorder attribute)
(zvt.recorders.wb.wb_country_recorder.WBCountryRecorder attribute)
(zvt.recorders.wb.wb_economy_recorder.WBEconomyRecorder attribute)
sessions (zvt.contract.context.Registry attribute)
set_class (zvt.trader.sim_account.AccountDayStatsSchema attribute)
(zvt.trader.sim_account.PositionSchema attribute)
set_one_and_only_one() (in module zvt.utils.utils)
set_stock_tags() (in module zvt.rest.work)
SetStockTagsModel (class in zvt.tag.tag_models)
settlement (zvt.domain.quotes.currency.currency_1d_kdata.Currency1dKdata attribute)

(zvt.domain.quotes.CurrencyKdataCommon attribute)
(zvt.domain.quotes.future.future_1d_kdata.Future1dKdata attribute)
(zvt.domain.quotes.FutureKdataCommon attribute)
sh (zvt.contract.Exchange attribute)
share_number (zvt.domain.misc.holder.HkHolder attribute)
share_ratio (zvt.domain.misc.holder.HkHolder attribute)
shareholding_numbers (zvt.domain.misc.holder.InstitutionalInvestorHolder attribute)
(zvt.domain.misc.holder.TopTenHolder attribute)
(zvt.domain.misc.holder.TopTenTradableHolder attribute)
shareholding_ratio (zvt.domain.misc.holder.InstitutionalInvestorHolder attribute)
(zvt.domain.misc.holder.TopTenHolder attribute)
(zvt.domain.misc.holder.TopTenTradableHolder attribute)
shares (zvt.contract.schema.PortfolioStockHistory attribute)
(zvt.domain.meta.etf_meta.EtfStock attribute)
(zvt.domain.meta.fund_meta.FundStock attribute)
(zvt.domain.meta.index_meta.IndexStock attribute)
shfe (zvt.contract.Exchange attribute)
short_amount (zvt.trader.trader_schemas.Position attribute)
short_term_borrowing (zvt.domain.fundamental.finance.BalanceSheet attribute)
short_value (zvt.domain.misc.overall.MarginTradingSummary attribute)
short_volume (zvt.domain.misc.overall.MarginTradingSummary attribute)
show_industry_composition() (in module zvt.api.stats)
show_month_performance() (in module zvt.api.stats)
SimAccountService (class in zvt.trader.sim_account)
SimpleStockTagsModel (class in zvt.tag.tag_models)
SinaBlockMoneyFlowRecorder (class in zvt.recorders.sina.money_flow.sina_block_money_flow_recorder)
SinaBlockRecorder (class in zvt.recorders.sina.meta.sina_block_recorder)
SinaChinaBlockStockRecorder (class in zvt.recorders.sina.meta.sina_block_recorder)
SinaStockMoneyFlowRecorder (class in zvt.recorders.sina.money_flow.sina_stock_money_flow_recorder)
social_security (zvt.contract.ActorType attribute)
south (zvt.domain.meta.stockhk_meta.Stockhk attribute)
special_reserve (zvt.domain.fundamental.finance.BalanceSheet attribute)
SpecificTransformer (class in zvt.factors.transformers)
split_time_interval() (in module zvt.utils.time_utils)
spo_issues (zvt.domain.fundamental.dividend_financing.DividendFinancing attribute)
SpoDetail (class in zvt.domain.fundamental.dividend_financing)
SPODetailRecorder (class in zvt.recorders.eastmoney.dividend_financing.eastmoney_spo_detail_recorder)
StackedDrawer (class in zvt.contract.drawer)
state (zvt.contract.zvt_info.FactorState attribute)
(zvt.contract.zvt_info.RecorderState attribute)
(zvt.contract.zvt_info.StateMixin attribute)
(zvt.contract.zvt_info.TaggerState attribute)
state_name (zvt.contract.zvt_info.FactorState attribute)
(zvt.contract.zvt_info.RecorderState attribute)
(zvt.contract.zvt_info.StateMixin attribute)
(zvt.contract.zvt_info.TaggerState attribute)
state_schema (zvt.contract.base_service.EntityStateService attribute)
(zvt.contract.base_service.OneStateService attribute)
(zvt.contract.base_service.StatefulService attribute)
(zvt.contract.factor.Factor attribute)
(zvt.contract.factor.ScoreFactor attribute)
(zvt.contract.recorder.EntityEventRecorder attribute)
(zvt.contract.recorder.FixedCycleDataRecorder attribute)
(zvt.contract.recorder.Recorder attribute)
(zvt.contract.recorder.TimeSeriesDataRecorder attribute)
(zvt.contract.recorder.TimestampsDataRecorder attribute)

(zvt.factors.fundamental.finance_factor.FinanceBaseFactor attribute)
(zvt.factors.fundamental.finance_factor.GoodCompanyFactor attribute)
(zvt.factors.ma.ma_factor.CrossMaFactor attribute)
(zvt.factors.ma.ma_factor.CrossMaVolumeFactor attribute)
(zvt.factors.ma.ma_factor.MaFactor attribute)
(zvt.factors.ma.ma_factor.VolumeUpMaFactor attribute)
(zvt.factors.ma.ma_stats_factor.MaStatsFactor attribute)
(zvt.factors.ma.ma_stats_factor.TFactor attribute)
(zvt.factors.ma.top_bottom_factor.TopBottomFactor attribute)
(zvt.factors.macd.macd_factor.BullFactor attribute)
(zvt.factors.macd.macd_factor.GoldCrossFactor attribute)
(zvt.factors.macd.macd_factor.KeepBullFactor attribute)
(zvt.factors.macd.macd_factor.LiveOrDeadFactor attribute)
(zvt.factors.macd.macd_factor.MacdFactor attribute)
(zvt.factors.technical_factor.TechnicalFactor attribute)
(zvt.factors.z.z_factor.ZFactor attribute)
(zvt.recorders.eastmoney.common.EastmoneyMoreDataRecorder attribute)
(zvt.recorders.eastmoney.common.EastmoneyPageabeDataRecorder attribute)
(zvt.recorders.eastmoney.common.EastmoneyTimestampsDataRecorder attribute)
(zvt.recorders.eastmoney.dividend_financing.eastmoney_dividend_detail_recorder.DividendDetailRecord
attribute)
(zvt.recorders.eastmoney.dividend_financing.eastmoney_dividend_financing_recorder.DividendFinancing
attribute)
(zvt.recorders.eastmoney.dividend_financing.eastmoney_rights_issue_detail_recorder.RightsIssueDetailR
attribute)
(zvt.recorders.eastmoney.dividend_financing.eastmoney_spo_detail_recorder.SPODetailRecorder attribut
(zvt.recorders.eastmoney.finance.base_china_stock_finance_recorder.BaseChinaStockFinanceRecorder at
(zvt.recorders.eastmoney.finance.eastmoney_balance_sheet_recorder.ChinaStockBalanceSheetRecorder a
(zvt.recorders.eastmoney.finance.eastmoney_cash_flow_recorder.ChinaStockCashFlowRecorder attribute
(zvt.recorders.eastmoney.finance.eastmoney_finance_factor_recorder.ChinaStockFinanceFactorRecorder_
(zvt.recorders.eastmoney.finance.eastmoney_income_statement_recorder.ChinaStockIncomeStatementRe
attribute)
(zvt.recorders.eastmoney.holder.eastmoney_stock_actor_recorder.EastmoneyActorRecorder attribute)
(zvt.recorders.eastmoney.holder.eastmoney_top_ten_holder_recorder.TopTenHolderRecorder attribute)
(zvt.recorders.eastmoney.holder.eastmoney_top_ten_tradable_holder_recorder.TopTenTradableHolderRec
attribute)
(zvt.recorders.eastmoney.meta.eastmoney_block_meta_recorder.EastmoneyBlockRecorder attribute)
(zvt.recorders.eastmoney.meta.eastmoney_block_meta_recorder.EastmoneyBlockStockRecorder attribute
(zvt.recorders.eastmoney.meta.eastmoney_stock_meta_recorder.EastmoneyStockDetailRecorder attribute
(zvt.recorders.eastmoney.meta.eastmoney_stock_meta_recorder.EastmoneyStockRecorder attribute)
(zvt.recorders.eastmoney.trading.eastmoney_holder_trading_recorder.HolderTradingRecorder attribute)
(zvt.recorders.eastmoney.trading.eastmoney_manager_trading_recorder.ManagerTradingRecorder attribut
(zvt.recorders.em.actor.em_stock_actor_summary_recorder.EMStockActorSummaryRecorder attribute)
(zvt.recorders.em.actor.em_stock_ii_recorder.EMStockIIRecorder attribute)
(zvt.recorders.em.actor.em_stock_top_ten_free_recorder.EMStockTopTenFreeRecorder attribute)
(zvt.recorders.em.actor.em_stock_top_ten_recorder.EMStockTopTenRecorder attribute)
(zvt.recorders.em.macro.em_treasury_yield_recorder.EMTreasuryYieldRecorder attribute)
(zvt.recorders.em.meta.em_block_meta_recorder.EMBlockRecorder attribute)
(zvt.recorders.em.meta.em_block_meta_recorder.EMBlockStockRecorder attribute)
(zvt.recorders.em.meta.em_currency_meta_recorder.EMCurrencyRecorder attribute)
(zvt.recorders.em.meta.em_future_meta_recorder.EMFutureRecorder attribute)
(zvt.recorders.em.meta.em_index_meta_recorder.EMIndexRecorder attribute)
(zvt.recorders.em.meta.em_indexus_meta_recorder.EMIndexusRecorder attribute)
(zvt.recorders.em.meta.em_stock_meta_recorder.EMStockRecorder attribute)
(zvt.recorders.em.meta.em_stockhk_meta_recorder.EMStockhkRecorder attribute)
(zvt.recorders.em.meta.em_stockus_meta_recorder.EMStockusRecorder attribute)

(zvt.recorders.em.misc.em_stock_news_recorder.EMStockNewsRecorder attribute)
(zvt.recorders.em.quotes.em_kdata_recorder.BaseEMStockKdataRecorder attribute)
(zvt.recorders.em.quotes.em_kdata_recorder.EMBlockKdataRecorder attribute)
(zvt.recorders.em.quotes.em_kdata_recorder.EMCurrencyKdataRecorder attribute)
(zvt.recorders.em.quotes.em_kdata_recorder.EMFutureKdataRecorder attribute)
(zvt.recorders.em.quotes.em_kdata_recorder.EMIndexKdataRecorder attribute)
(zvt.recorders.em.quotes.em_kdata_recorder.EMIndexusKdataRecorder attribute)
(zvt.recorders.em.quotes.em_kdata_recorder.EMStockhkKdataRecorder attribute)
(zvt.recorders.em.quotes.em_kdata_recorder.EMStockKdataRecorder attribute)
(zvt.recorders.em.quotes.em_kdata_recorder.EMStockusKdataRecorder attribute)
(zvt.recorders.em.trading.em_dragon_and_tiger_recorder.EMDragonAndTigerRecorder attribute)
(zvt.recorders.exchange.exchange_etf_meta_recorder.ChinaETFListSpider attribute)
(zvt.recorders.exchange.exchange_index_recorder.ExchangeIndexRecorder attribute)
(zvt.recorders.exchange.exchange_index_stock_recorder.ExchangeIndexStockRecorder attribute)
(zvt.recorders.exchange.exchange_stock_meta_recorder.ExchangeStockMetaRecorder attribute)
(zvt.recorders.exchange.exchange_stock_summary_recorder.ExchangeStockSummaryRecorder attribute)
(zvt.recorders.joinquant.fundamental.jq_etf_valuation_recorder.JqChinaEtfValuationRecorder attribute)
(zvt.recorders.joinquant.fundamental.jq_margin_trading_recorder.MarginTradingRecorder attribute)
(zvt.recorders.joinquant.fundamental.jq_stock_valuation_recorder.JqChinaStockValuationRecorder attribu
(zvt.recorders.joinquant.meta.jq_fund_meta_recorder.JqChinaFundRecorder attribute)
(zvt.recorders.joinquant.meta.jq_fund_meta_recorder.JqChinaFundStockRecorder attribute)
(zvt.recorders.joinquant.meta.jq_stock_meta_recorder.BaseJqChinaMetaRecorder attribute)
(zvt.recorders.joinquant.meta.jq_stock_meta_recorder.JqChinaEtfRecorder attribute)
(zvt.recorders.joinquant.meta.jq_stock_meta_recorder.JqChinaStockEtfPortfolioRecorder attribute)
(zvt.recorders.joinquant.meta.jq_stock_meta_recorder.JqChinaStockRecorder attribute)
(zvt.recorders.joinquant.meta.jq_trade_day_recorder.StockTradeDayRecorder attribute)
(zvt.recorders.joinquant.misc.jq_hk_holder_recorder.JoinquantHkHolderRecorder attribute)
(zvt.recorders.joinquant.misc.jq_index_money_flow_recorder.JoinquantIndexMoneyFlowRecorder attribu
(zvt.recorders.joinquant.misc.jq_stock_money_flow_recorder.JoinquantStockMoneyFlowRecorder attribu
(zvt.recorders.joinquant.overall.jq_cross_market_recorder.CrossMarketSummaryRecorder attribute)
(zvt.recorders.joinquant.overall.jq_margin_trading_recorder.MarginTradingSummaryRecorder attribute)
(zvt.recorders.joinquant.overall.jq_stock_summary_recorder.StockSummaryRecorder attribute)
(zvt.recorders.joinquant.quotes.jq_index_kdata_recorder.JqChinaIndexKdataRecorder attribute)
(zvt.recorders.joinquant.quotes.jq_stock_kdata_recorder.JqChinaStockKdataRecorder attribute)
(zvt.recorders.sina.meta.sina_block_recorder.SinaBlockRecorder attribute)
(zvt.recorders.sina.meta.sina_block_recorder.SinaChinaBlockStockRecorder attribute)
(zvt.recorders.sina.money_flow.sina_block_money_flow_recorder.SinaBlockMoneyFlowRecorder attribu
(zvt.recorders.sina.money_flow.sina_stock_money_flow_recorder.SinaStockMoneyFlowRecorder attribut
(zvt.recorders.sina.quotes.sina_etf_kdata_recorder.ChinaETFDayKdataRecorder attribute)
(zvt.recorders.sina.quotes.sina_index_kdata_recorder.ChinaIndexDayKdataRecorder attribute)
(zvt.recorders.wb.wb_country_recorder.WBCountryRecorder attribute)
(zvt.recorders.wb.wb_economy_recorder.WBEconomyRecorder attribute)
(zvt.tag.tag_service.StockAutoTagger attribute)
(zvt.tag.tagger.StockTagger attribute)
(zvt.tag.tagger.Tagger attribute)

T

tab_content() (in module zvt.main)
Tagger (class in zvt.tag.tagger)
TaggerState (class in zvt.contract.zvt_info)
TagInfoModel (class in zvt.tag.tag_models)
TagParameter (class in zvt.tag.tag_models)
TagQuoteStatsModel (class in zvt.trading.trading_models)
TagStats (class in zvt.tag.tag_schemas)
to_snake_str() (in module z
to_str() (in module zvt.utils
to_string() (in module zvt.u
to_time_str() (in module zv
to_timestamp() (in module
to_zvt_code() (in module z
today() (in module zvt.utils

TagType (class in zvt.tag.common)
TargetSelector (class in zvt.factors.target_selector)
TargetType (class in zvt.contract.factor)
tax_expense (zvt.domain.fundamental.finance.IncomeStatement attribute)
tax_refund (zvt.domain.fundamental.finance.CashFlowStatement attribute)
taxes_and_surcharges (zvt.domain.fundamental.finance.CashFlowStatement attribute)
taxes_payable (zvt.domain.fundamental.finance.BalanceSheet attribute)
TechnicalFactor (class in zvt.factors.technical_factor)
TFactor (class in zvt.factors.ma.ma_stats_factor)
TickCommon (class in zvt.domain.quotes)
time (zvt.domain.quotes.stock.stock_quote.StockQuote attribute)
(zvt.domain.quotes.TickCommon attribute)
(zvt.trading.trading_models.StockQuoteModel attribute)
TimeRange (class in zvt.common.query_models)
TimeSeriesDataRecorder (class in zvt.contract.recorder)
timestamp (zvt.contract.schema.ActorEntity attribute)
(zvt.contract.schema.ActorMeetTradable attribute)
(zvt.contract.schema.Entity attribute)
(zvt.contract.schema.Mixin attribute)
(zvt.contract.schema.NormalEntityMixin attribute)
(zvt.contract.schema.NormalMixin attribute)
(zvt.contract.schema.Portfolio attribute)
(zvt.contract.schema.PortfolioStock attribute)
(zvt.contract.schema.PortfolioStockHistory attribute)
(zvt.contract.schema.TradableEntity attribute)
(zvt.contract.schema.TradableMeetActor attribute)
(zvt.contract.zvt_info.FactorState attribute)
(zvt.contract.zvt_info.RecorderState attribute)
(zvt.contract.zvt_info.StateMixin attribute)
(zvt.contract.zvt_info.TaggerState attribute)
(zvt.domain.actor.actor_meta.ActorMeta attribute)
(zvt.domain.actor.stock_actor.StockActorSummary attribute)
(zvt.domain.actor.stock_actor.StockInstitutionalInvestorHolder attribute)
(zvt.domain.actor.stock_actor.StockTopTenFreeHolder attribute)
(zvt.domain.actor.stock_actor.StockTopTenHolder attribute)
(zvt.domain.emotion.emotion.Emotion attribute)
(zvt.domain.emotion.emotion.LimitDownInfo attribute)
(zvt.domain.emotion.emotion.LimitUpInfo attribute)
(zvt.domain.fundamental.dividend_financing.DividendDetail attribute)
(zvt.domain.fundamental.dividend_financing.DividendFinancing attribute)
(zvt.domain.fundamental.dividend_financing.RightsIssueDetail attribute)
(zvt.domain.fundamental.dividend_financing.SpoDetail attribute)
(zvt.domain.fundamental.finance.BalanceSheet attribute)
(zvt.domain.fundamental.finance.CashFlowStatement attribute)
(zvt.domain.fundamental.finance.FinanceFactor attribute)
(zvt.domain.fundamental.finance.IncomeStatement attribute)
(zvt.domain.fundamental.trading.BigDealTrading attribute)
(zvt.domain.fundamental.trading.DragonAndTiger attribute)
(zvt.domain.fundamental.trading.HolderTrading attribute)
(zvt.domain.fundamental.trading.ManagerTrading attribute)
(zvt.domain.fundamental.trading.MarginTrading attribute)
(zvt.domain.fundamental.valuation.EtfValuation attribute)
(zvt.domain.fundamental.valuation.StockValuation attribute)
tomorrow_date() (in modul
TopBottomFactor (class in
TopBottomTransformer (cl
TopStocks (class in zvt.fact
TopTenHolder (class in zvt
TopTenHolderRecorder (cl
zvt.recorders.eastmoney.ho
TopTenTradableHolder (cla
TopTenTradableHolderRec
zvt.recorders.eastmoney.ho
TopType (class in zvt.api.st
total_assets (zvt.domain.fu
total_assets_turnover (zvt.c
attribute)
total_assets_turnover_days
attribute)
total_cap (zvt.domain.quot
(zvt.trading.trading_m
total_comprehensive_incor
(zvt.domain.fundamental.fi
total_comprehensive_incor
(zvt.domain.fundamental.fi
total_comprehensive_incor
(zvt.domain.fundamental.fi
total_current_assets (zvt.do
total_current_liabilities (zv
attribute)
total_equity (zvt.domain.fu
total_financing_cash_inflo
(zvt.domain.fundamental.fi
total_financing_cash_outfl
(zvt.domain.fundamental.fi
total_investing_cash_inflo
(zvt.domain.fundamental.fi
total_investing_cash_outfl
(zvt.domain.fundamental.fi
total_liabilities (zvt.domai
total_liabilities_and_equity
attribute)
total_non_current_assets (z
attribute)
total_non_current_liabilitie
attribute)
total_op_cash_inflows (zvt
attribute)
total_op_cash_outflows (zv
attribute)
total_op_income (zvt.doma
total_operating_costs (zvt.c
attribute)
total_profits (zvt.domain.fu
total_value (zvt.domain.mi
tradable_entity_schemas (z
tradable_entity_types (zvt.c
tradable_schema_map (zvt.
TradableEntity (class in zvt

(zvt.domain.macro.macro.Economy attribute)
(zvt.domain.macro.monetary.TreasuryYield attribute)
(zvt.domain.meta.block_meta.Block attribute)
(zvt.domain.meta.block_meta.BlockStock attribute)
(zvt.domain.meta.country_meta.Country attribute)
(zvt.domain.meta.currency_meta.Currency attribute)
(zvt.domain.meta.etf_meta.Etf attribute)
(zvt.domain.meta.etf_meta.EtfStock attribute)
(zvt.domain.meta.fund_meta.Fund attribute)
(zvt.domain.meta.fund_meta.FundStock attribute)
(zvt.domain.meta.future_meta.Future attribute)
(zvt.domain.meta.index_meta.Index attribute)
(zvt.domain.meta.index_meta.IndexStock attribute)
(zvt.domain.meta.indexus_meta.Indexus attribute)
(zvt.domain.meta.stock_meta.Stock attribute)
(zvt.domain.meta.stock_meta.StockDetail attribute)
(zvt.domain.meta.stockhk_meta.Stockhk attribute)
(zvt.domain.meta.stockus_meta.Stockus attribute)
(zvt.domain.misc.holder.HkHolder attribute)
(zvt.domain.misc.holder.InstitutionalInvestorHolder attribute)
(zvt.domain.misc.holder.TopTenHolder attribute)
(zvt.domain.misc.holder.TopTenTradableHolder attribute)
(zvt.domain.misc.money_flow.BlockMoneyFlow attribute)
(zvt.domain.misc.money_flow.IndexMoneyFlow attribute)
(zvt.domain.misc.money_flow.StockMoneyFlow attribute)
(zvt.domain.misc.overall.CrossMarketSummary attribute)
(zvt.domain.misc.overall.MarginTradingSummary attribute)
(zvt.domain.misc.overall.StockSummary attribute)
(zvt.domain.misc.stock_news.StockNews attribute)
(zvt.domain.quotes.block.block_1d_kdata.Block1dKdata attribute)
(zvt.domain.quotes.block.block_1mon_kdata.Block1monKdata attribute)
(zvt.domain.quotes.block.block_1wk_kdata.Block1wkKdata attribute)
(zvt.domain.quotes.BlockKdataCommon attribute)
(zvt.domain.quotes.currency.currency_1d_kdata.Currency1dKdata attribute)
(zvt.domain.quotes.CurrencyKdataCommon attribute)
(zvt.domain.quotes.etf.etf_1d_kdata.Etf1dKdata attribute)
(zvt.domain.quotes.EtfKdataCommon attribute)
(zvt.domain.quotes.future.future_1d_kdata.Future1dKdata attribute)
(zvt.domain.quotes.FutureKdataCommon attribute)
(zvt.domain.quotes.index.index_1d_kdata.Index1dKdata attribute)
(zvt.domain.quotes.index.index_1wk_kdata.Index1wkKdata attribute)
(zvt.domain.quotes.IndexKdataCommon attribute)
(zvt.domain.quotes.indexus.indexus_1d_kdata.Indexus1dKdata attribute)
(zvt.domain.quotes.IndexusKdataCommon attribute)
(zvt.domain.quotes.KdataCommon attribute)
(zvt.domain.quotes.stock.stock_15m_hfq_kdata.Stock15mHfqKdata attribute)
(zvt.domain.quotes.stock.stock_15m_kdata.Stock15mKdata attribute)
(zvt.domain.quotes.stock.stock_1d_hfq_kdata.Stock1dHfqKdata attribute)
(zvt.domain.quotes.stock.stock_1d_kdata.Stock1dKdata attribute)
(zvt.domain.quotes.stock.stock_1h_hfq_kdata.Stock1hHfqKdata attribute)

TradableMeetActor (class i
TradableType (class in zvt.
Trader (class in zvt.trader.t
trader_name (zvt.trader.trad
(zvt.trader.trader_sche
(zvt.trader.trader_sche
(zvt.trader.trader_sche
TraderError, [1]
TraderInfo (class in zvt.trad
TradeType (class in zvt.fact
trading_person (zvt.domain
trading_signal_type_to_ord
trading_t (zvt.trader.trader_
trading_way (zvt.domain.fu
TradingListener (class in zv
TradingPlan (class in zvt.tra
TradingPlanModel (class in
TradingResult (class in zvt.
TradingSignal (class in zvt.
TradingSignalModel (class
TradingSignalType (class i
transaction_num (zvt.doma
transform() (zvt.contract.fa
(zvt.factors.algorithm.
(zvt.factors.algorithm.
(zvt.factors.algorithm.
(zvt.factors.algorithm.
(zvt.factors.ma.top_bo
(zvt.factors.transforme
(zvt.factors.transforme
(zvt.factors.transforme
transform_one() (zvt.contra
(zvt.factors.algorithm.
(zvt.factors.algorithm.
(zvt.factors.algorithm.
(zvt.factors.algorithm.
(zvt.factors.ma.top_bo
(zvt.factors.transforme
(zvt.factors.transforme
(zvt.factors.transforme
Transformer (class in zvt.co
transformer (zvt.contract.fa
(zvt.contract.factor.Sc
(zvt.factors.fundament
(zvt.factors.fundament
(zvt.factors.ma.ma_fa
(zvt.factors.ma.ma_fa
(zvt.factors.ma.ma_fa
(zvt.factors.ma.ma_fa
(zvt.factors.ma.ma_sta
(zvt.factors.ma.ma_sta
(zvt.factors.ma.top_bo
(zvt.factors.macd.mac
(zvt.factors.macd.mac
(zvt.factors.macd.mac
(zvt.factors.macd.mac

(zvt.domain.quotes.stock.stock_1h_kdata.Stock1hKdata attribute)
(zvt.domain.quotes.stock.stock_1m_hfq_kdata.Stock1mHfqKdata attribute)
(zvt.domain.quotes.stock.stock_1m_kdata.Stock1mKdata attribute)
(zvt.domain.quotes.stock.stock_1mon_hfq_kdata.Stock1monHfqKdata attribute)
(zvt.domain.quotes.stock.stock_1mon_kdata.Stock1monKdata attribute)
(zvt.domain.quotes.stock.stock_1wk_hfq_kdata.Stock1wkHfqKdata attribute)
(zvt.domain.quotes.stock.stock_1wk_kdata.Stock1wkKdata attribute)
(zvt.domain.quotes.stock.stock_30m_hfq_kdata.Stock30mHfqKdata attribute)
(zvt.domain.quotes.stock.stock_30m_kdata.Stock30mKdata attribute)
(zvt.domain.quotes.stock.stock_4h_hfq_kdata.Stock4hHfqKdata attribute)
(zvt.domain.quotes.stock.stock_4h_kdata.Stock4hKdata attribute)
(zvt.domain.quotes.stock.stock_5m_hfq_kdata.Stock5mHfqKdata attribute)
(zvt.domain.quotes.stock.stock_5m_kdata.Stock5mKdata attribute)
(zvt.domain.quotes.stock.stock_quote.StockQuote attribute)
(zvt.domain.quotes.stockhk.stockhk_1d_hfq_kdata.Stockhk1dHfqKdata attribute)
(zvt.domain.quotes.stockhk.stockhk_1d_kdata.Stockhk1dKdata attribute)
(zvt.domain.quotes.StockhkKdataCommon attribute)
(zvt.domain.quotes.StockKdataCommon attribute)
(zvt.domain.quotes.stockus.stockus_1d_hfq_kdata.Stockus1dHfqKdata attribute)
(zvt.domain.quotes.stockus.stockus_1d_kdata.Stockus1dKdata attribute)
(zvt.domain.quotes.StockusKdataCommon attribute)
(zvt.domain.quotes.TickCommon attribute)
(zvt.domain.quotes.trade_day.StockTradeDay attribute)
(zvt.factors.ma.domain.common.MaStatsFactorCommon attribute)
(zvt.factors.ma.domain.stock_1d_ma_factor.Stock1dMaFactor attribute)
(zvt.factors.ma.domain.stock_1d_ma_stats_factor.Stock1dMaStatsFactor attribute)
(zvt.factors.top_stocks.TopStocks attribute)
(zvt.factors.z.domain.common.ZFactorCommon attribute)
(zvt.factors.z.domain.stock_1d_z_factor.Stock1dZFactor attribute)
(zvt.factors.z.domain.stock_1wk_z_factor.Stock1wkZFactor attribute)
(zvt.tag.tag_schemas.HiddenTagInfo attribute)
(zvt.tag.tag_schemas.MainTagInfo attribute)
(zvt.tag.tag_schemas.StockPoolInfo attribute)
(zvt.tag.tag_schemas.StockPools attribute)
(zvt.tag.tag_schemas.StockSystemTags attribute)
(zvt.tag.tag_schemas.StockTags attribute)
(zvt.tag.tag_schemas.SubTagInfo attribute)
(zvt.tag.tag_schemas.TagStats attribute)
(zvt.trader.trader_schemas.AccountStats attribute)
(zvt.trader.trader_schemas.Order attribute)
(zvt.trader.trader_schemas.Position attribute)
(zvt.trader.trader_schemas.TraderInfo attribute)
(zvt.trading.trading_schemas.QueryStockQuoteSetting attribute)
(zvt.trading.trading_schemas.TradingPlan attribute)
TimestampsDataRecorder (class in zvt.contract.recorder)
TimeUnit (class in zvt.common.query_models)
(zvt.factors.macd.mac
(zvt.factors.technical_
(zvt.factors.z.z_factor.
TreasuryYield (class in zvt.
trust (zvt.contract.ActorTyp
trustee (zvt.domain.meta.fu
turnover (zvt.domain.funda
(zvt.domain.fundamer
(zvt.domain.quotes.bl
(zvt.domain.quotes.bl
(zvt.domain.quotes.bl
(zvt.domain.quotes.cu
attribute)
(zvt.domain.quotes.etf
(zvt.domain.quotes.fu
(zvt.domain.quotes.in
(zvt.domain.quotes.in
(zvt.domain.quotes.in
(zvt.domain.quotes.st
attribute)
(zvt.domain.quotes.st
(zvt.domain.quotes.st
attribute)
(zvt.domain.quotes.st
(zvt.domain.quotes.st
attribute)
(zvt.domain.quotes.st
(zvt.domain.quotes.st
attribute)
(zvt.domain.quotes.st
(zvt.domain.quotes.st
attribute)
(zvt.domain.quotes.st
(zvt.domain.quotes.st
attribute)
(zvt.domain.quotes.st
(zvt.domain.quotes.st
attribute)
(zvt.domain.quotes.st
(zvt.domain.quotes.st
attribute)
(zvt.domain.quotes.st
(zvt.domain.quotes.st
attribute)
(zvt.domain.quotes.st
(zvt.domain.quotes.st
attribute)
(zvt.domain.quotes.st
(zvt.domain.quotes.st
attribute)
(zvt.domain.quotes.st
(zvt.domain.quotes.Ti
(zvt.factors.ma.domai
attribute)
(zvt.factors.z.domain.

tmp_di (zvt.factors.z.domain.stock_1d_z_factor.Stock1dZFactor attribute)
(zvt.factors.z.domain.stock_1wk_z_factor.Stock1wkZFactor attribute)
tmp_ding (zvt.factors.z.domain.stock_1d_z_factor.Stock1dZFactor attribute)
(zvt.factors.z.domain.stock_1wk_z_factor.Stock1wkZFactor attribute)
tmp_direction (zvt.factors.z.domain.stock_1d_z_factor.Stock1dZFactor attribute)
(zvt.factors.z.domain.stock_1wk_z_factor.Stock1wkZFactor attribute)
to_camel_str() (in module zvt.utils.str_utils)
to_datetime() (zvt.zhdate.zhdate.ZhDate method)
to_em_entity_flag() (in module zvt.recorders.em.em_api)
to_em_fc() (in module zvt.recorders.em.em_api)
to_em_fq_flag() (in module zvt.recorders.em.em_api)
to_em_level_flag() (in module zvt.recorders.em.em_api)
to_em_sec_id() (in module zvt.recorders.em.em_api)
to_entity_id() (in module zvt.recorders.joinquant.common)
to_float() (in module zvt.utils.utils)
to_high_level_kdata() (in module zvt.api.kdata)
to_jq_entity_id() (in module zvt.recorders.joinquant.common)
to_jq_report_period() (in module zvt.recorders.eastmoney.finance.base_china_stock_finance_recorder)
to_jq_trading_level() (in module zvt.recorders.joinquant.common)
to_ms() (zvt.contract.IntervalLevel method)
to_pd_timestamp() (in module zvt.utils.time_utils)
to_positive_number() (in module zvt.utils.utils)
to_report_period_type() (in module zvt.api.utils)

U

underlying_asset_type (zvt.domain.meta.fund_meta.Fund attribute)
underlying_asset_type_id (zvt.domain.meta.fund_meta.Fund attribute)

(zvt.factors.z.domain.s
(zvt.trading.trading_m
(zvt.trading.trading_m
turnover_rate (zvt.domain.e
(zvt.domain.emotion.e
(zvt.domain.quotes.bl
(zvt.domain.quotes.bl
(zvt.domain.quotes.bl
(zvt.domain.quotes.cu
attribute)
(zvt.domain.quotes.etf
(zvt.domain.quotes.fu
(zvt.domain.quotes.in
(zvt.domain.quotes.in
(zvt.domain.quotes.in
(zvt.domain.quotes.st
attribute)
(zvt.domain.quotes.st
(zvt.domain.quotes.st
attribute)
(zvt.domain.quotes.st
(zvt.domain.quotes.st
attribute)
(zvt.domain.quotes.st
(zvt.domain.quotes.st
attribute)
(zvt.domain.quotes.st
(zvt.domain.quotes.st
attribute)
(zvt.domain.quotes.st
(zvt.domain.quotes.st
attribute)
(zvt.domain.quotes.st
(zvt.domain.quotes.st
attribute)
(zvt.domain.quotes.st
(zvt.domain.quotes.st
attribute)
(zvt.domain.quotes.st
(zvt.domain.quotes.st
attribute)
(zvt.domain.quotes.st
(zvt.domain.quotes.st
attribute)
(zvt.domain.quotes.st
turnover_ratio (zvt.domain.

undistributed_profit_ps (zvt.domain.fundamental.finance.FinanceFactor attribute)
undistributed_profits (zvt.domain.fundamental.finance.BalanceSheet attribute)
unzip() (in module zvt.utils.zip_utils)
up_count (zvt.trading.trading_models.StockQuoteStatsModel attribute)
(zvt.trading.trading_models.TagQuoteStatsModel attribute)
update_model() (in module zvt.utils.model_utils)
update_position() (zvt.trader.sim_account.SimAccountService method)
update_targets_by_level() (zvt.samples.stock_traders.MyBullTrader method)
(zvt.samples.stock_traders.MyMaTrader method)
(zvt.trader.trader.StockTrader method)
(zvt.trader.trader.Trader method)
update_vol_up() (in module zvt.factors.top_stocks)
update_with_limit_up() (in module zvt.factors.top_stocks)
updated_timestamp (zvt.contract.schema.NormalEntityMixin attribute)
(zvt.contract.schema.NormalMixin attribute)
url (zvt.contract.recorder.EntityEventRecorder attribute)
(zvt.contract.recorder.FixedCycleDataRecorder attribute)
(zvt.contract.recorder.Recorder attribute)
(zvt.contract.recorder.TimeSeriesDataRecorder attribute)
(zvt.contract.recorder.TimestampsDataRecorder attribute)
(zvt.recorders.eastmoney.common.EastmoneyMoreDataRecorder attribute)
(zvt.recorders.eastmoney.common.EastmoneyPageabeDataRecorder attribute)
(zvt.recorders.eastmoney.common.EastmoneyTimestampsDataRecorder attribute)
(zvt.recorders.eastmoney.dividend_financing.eastmoney_dividend_detail_recorder.DividendDetailRecord
attribute)
(zvt.recorders.eastmoney.dividend_financing.eastmoney_dividend_financing_recorder.DividendFinancing
attribute)
(zvt.recorders.eastmoney.dividend_financing.eastmoney_rights_issue_detail_recorder.RightsIssueDetailR
attribute)
(zvt.recorders.eastmoney.dividend_financing.eastmoney_spo_detail_recorder.SPODetailRecorder attribut
(zvt.recorders.eastmoney.finance.base_china_stock_finance_recorder.BaseChinaStockFinanceRecorder at
(zvt.recorders.eastmoney.finance.eastmoney_balance_sheet_recorder.ChinaStockBalanceSheetRecorder a
(zvt.recorders.eastmoney.finance.eastmoney_cash_flow_recorder.ChinaStockCashFlowRecorder attribute
(zvt.recorders.eastmoney.finance.eastmoney_finance_factor_recorder.ChinaStockFinanceFactorRecorder_
(zvt.recorders.eastmoney.finance.eastmoney_income_statement_recorder.ChinaStockIncomeStatementRe
attribute)
(zvt.recorders.eastmoney.holder.eastmoney_stock_actor_recorder.EastmoneyActorRecorder attribute)
(zvt.recorders.eastmoney.holder.eastmoney_top_ten_holder_recorder.TopTenHolderRecorder attribute)
(zvt.recorders.eastmoney.holder.eastmoney_top_ten_tradable_holder_recorder.TopTenTradableHolderRec
attribute)
(zvt.recorders.eastmoney.meta.eastmoney_block_meta_recorder.EastmoneyBlockRecorder attribute)
(zvt.recorders.eastmoney.meta.eastmoney_block_meta_recorder.EastmoneyBlockStockRecorder attribute
(zvt.recorders.eastmoney.meta.eastmoney_stock_meta_recorder.EastmoneyStockDetailRecorder attribute
(zvt.recorders.eastmoney.meta.eastmoney_stock_meta_recorder.EastmoneyStockRecorder attribute)
(zvt.recorders.eastmoney.trading.eastmoney_holder_trading_recorder.HolderTradingRecorder attribute)
(zvt.recorders.eastmoney.trading.eastmoney_manager_trading_recorder.ManagerTradingRecorder attribut
(zvt.recorders.em.actor.em_stock_actor_summary_recorder.EMStockActorSummaryRecorder attribute)
(zvt.recorders.em.actor.em_stock_ii_recorder.EMStockIIRecorder attribute)
(zvt.recorders.em.actor.em_stock_top_ten_free_recorder.EMStockTopTenFreeRecorder attribute)
(zvt.recorders.em.actor.em_stock_top_ten_recorder.EMStockTopTenRecorder attribute)
(zvt.recorders.em.macro.em_treasury_yield_recorder.EMTreasuryYieldRecorder attribute)
(zvt.recorders.em.meta.em_block_meta_recorder.EMBlockRecorder attribute)
(zvt.recorders.em.meta.em_block_meta_recorder.EMBlockStockRecorder attribute)
(zvt.recorders.em.meta.em_currency_meta_recorder.EMCurrencyRecorder attribute)
(zvt.recorders.em.meta.em_future_meta_recorder.EMFutureRecorder attribute)
(zvt.recorders.em.meta.em_index_meta_recorder.EMIndexRecorder attribute)

(zvt.recorders.em.meta.em_indexus_meta_recorder.EMIndexusRecorder attribute)
(zvt.recorders.em.meta.em_stock_meta_recorder.EMStockRecorder attribute)
(zvt.recorders.em.meta.em_stockhk_meta_recorder.EMStockhkRecorder attribute)
(zvt.recorders.em.meta.em_stockus_meta_recorder.EMStockusRecorder attribute)
(zvt.recorders.em.misc.em_stock_news_recorder.EMStockNewsRecorder attribute)
(zvt.recorders.em.quotes.em_kdata_recorder.BaseEMStockKdataRecorder attribute)
(zvt.recorders.em.quotes.em_kdata_recorder.EMBlockKdataRecorder attribute)
(zvt.recorders.em.quotes.em_kdata_recorder.EMCurrencyKdataRecorder attribute)
(zvt.recorders.em.quotes.em_kdata_recorder.EMFutureKdataRecorder attribute)
(zvt.recorders.em.quotes.em_kdata_recorder.EMIndexKdataRecorder attribute)
(zvt.recorders.em.quotes.em_kdata_recorder.EMIndexusKdataRecorder attribute)
(zvt.recorders.em.quotes.em_kdata_recorder.EMStockhkKdataRecorder attribute)
(zvt.recorders.em.quotes.em_kdata_recorder.EMStockKdataRecorder attribute)
(zvt.recorders.em.quotes.em_kdata_recorder.EMStockusKdataRecorder attribute)
(zvt.recorders.em.trading.em_dragon_and_tiger_recorder.EMDragonAndTigerRecorder attribute)
(zvt.recorders.exchange.exchange_etf_meta_recorder.ChinaETFListSpider attribute)
(zvt.recorders.exchange.exchange_index_recorder.ExchangeIndexRecorder attribute)
(zvt.recorders.exchange.exchange_index_stock_recorder.ExchangeIndexStockRecorder attribute)
(zvt.recorders.exchange.exchange_stock_meta_recorder.ExchangeStockMetaRecorder attribute)
(zvt.recorders.exchange.exchange_stock_summary_recorder.ExchangeStockSummaryRecorder attribute)
(zvt.recorders.joinquant.fundamental.jq_etf_valuation_recorder.JqChinaEtfValuationRecorder attribute)
(zvt.recorders.joinquant.fundamental.jq_margin_trading_recorder.MarginTradingRecorder attribute)
(zvt.recorders.joinquant.fundamental.jq_stock_valuation_recorder.JqChinaStockValuationRecorder attribu
(zvt.recorders.joinquant.meta.jq_fund_meta_recorder.JqChinaFundRecorder attribute)
(zvt.recorders.joinquant.meta.jq_fund_meta_recorder.JqChinaFundStockRecorder attribute)
(zvt.recorders.joinquant.meta.jq_stock_meta_recorder.BaseJqChinaMetaRecorder attribute)
(zvt.recorders.joinquant.meta.jq_stock_meta_recorder.JqChinaEtfRecorder attribute)
(zvt.recorders.joinquant.meta.jq_stock_meta_recorder.JqChinaStockEtfPortfolioRecorder attribute)
(zvt.recorders.joinquant.meta.jq_stock_meta_recorder.JqChinaStockRecorder attribute)
(zvt.recorders.joinquant.meta.jq_trade_day_recorder.StockTradeDayRecorder attribute)
(zvt.recorders.joinquant.misc.jq_hk_holder_recorder.JoinquantHkHolderRecorder attribute)
(zvt.recorders.joinquant.misc.jq_index_money_flow_recorder.JoinquantIndexMoneyFlowRecorder attribu
(zvt.recorders.joinquant.misc.jq_stock_money_flow_recorder.JoinquantStockMoneyFlowRecorder attribu
(zvt.recorders.joinquant.overall.jq_cross_market_recorder.CrossMarketSummaryRecorder attribute)
(zvt.recorders.joinquant.overall.jq_margin_trading_recorder.MarginTradingSummaryRecorder attribute)
(zvt.recorders.joinquant.overall.jq_stock_summary_recorder.StockSummaryRecorder attribute)
(zvt.recorders.joinquant.quotes.jq_index_kdata_recorder.JqChinaIndexKdataRecorder attribute)
(zvt.recorders.joinquant.quotes.jq_stock_kdata_recorder.JqChinaStockKdataRecorder attribute)
(zvt.recorders.sina.meta.sina_block_recorder.SinaBlockRecorder attribute)
(zvt.recorders.sina.meta.sina_block_recorder.SinaChinaBlockStockRecorder attribute)
(zvt.recorders.sina.money_flow.sina_block_money_flow_recorder.SinaBlockMoneyFlowRecorder attribu
(zvt.recorders.sina.money_flow.sina_stock_money_flow_recorder.SinaStockMoneyFlowRecorder attribut
(zvt.recorders.sina.quotes.sina_etf_kdata_recorder.ChinaETFDayKdataRecorder attribute)
(zvt.recorders.sina.quotes.sina_index_kdata_recorder.ChinaIndexDayKdataRecorder attribute)
(zvt.recorders.wb.wb_country_recorder.WBCountryRecorder attribute)
(zvt.recorders.wb.wb_economy_recorder.WBEconomyRecorder attribute)

V

validate() (zvt.trader.sim_account.AccountDayStatsSchema method)
(zvt.trader.sim_account.PositionSchema method)
(zvt.zhdate.zhdate.ZhDate static method)
value (zvt.trader.trader_schemas.AccountStats attribute)
(zvt.trader.trader_schemas.Position attribute)
value_multiply() (in module zvt.api.utils)
VolumeUpMaFactor (class in zvt.factors.ma.ma_factor)

value_to_pct() (in module zvt.api.utils)
vol_ma30 (zvt.factors.ma.domain.stock_1d_ma_stats_factor.Stock1dMaStatsFactor attribute)
volume (zvt.domain.fundamental.trading.HolderTrading attribute)
(zvt.domain.fundamental.trading.ManagerTrading attribute)
(zvt.domain.quotes.block.block_1d_kdata.Block1dKdata attribute)
(zvt.domain.quotes.block.block_1mon_kdata.Block1monKdata attribute)
(zvt.domain.quotes.block.block_1wk_kdata.Block1wkKdata attribute)
(zvt.domain.quotes.currency.currency_1d_kdata.Currency1dKdata attribute)
(zvt.domain.quotes.etf.etf_1d_kdata.Etf1dKdata attribute)
(zvt.domain.quotes.future.future_1d_kdata.Future1dKdata attribute)
(zvt.domain.quotes.index.index_1d_kdata.Index1dKdata attribute)
(zvt.domain.quotes.index.index_1wk_kdata.Index1wkKdata attribute)
(zvt.domain.quotes.indexus.indexus_1d_kdata.Indexus1dKdata attribute)
(zvt.domain.quotes.stock.stock_15m_hfq_kdata.Stock15mHfqKdata attribute)
(zvt.domain.quotes.stock.stock_15m_kdata.Stock15mKdata attribute)
(zvt.domain.quotes.stock.stock_1d_hfq_kdata.Stock1dHfqKdata attribute)
(zvt.domain.quotes.stock.stock_1d_kdata.Stock1dKdata attribute)
(zvt.domain.quotes.stock.stock_1h_hfq_kdata.Stock1hHfqKdata attribute)
(zvt.domain.quotes.stock.stock_1h_kdata.Stock1hKdata attribute)
(zvt.domain.quotes.stock.stock_1m_hfq_kdata.Stock1mHfqKdata attribute)
(zvt.domain.quotes.stock.stock_1m_kdata.Stock1mKdata attribute)
(zvt.domain.quotes.stock.stock_1mon_hfq_kdata.Stock1monHfqKdata attribute)
(zvt.domain.quotes.stock.stock_1mon_kdata.Stock1monKdata attribute)
(zvt.domain.quotes.stock.stock_1wk_hfq_kdata.Stock1wkHfqKdata attribute)
(zvt.domain.quotes.stock.stock_1wk_kdata.Stock1wkKdata attribute)
(zvt.domain.quotes.stock.stock_30m_hfq_kdata.Stock30mHfqKdata attribute)
(zvt.domain.quotes.stock.stock_30m_kdata.Stock30mKdata attribute)
(zvt.domain.quotes.stock.stock_4h_hfq_kdata.Stock4hHfqKdata attribute)
(zvt.domain.quotes.stock.stock_4h_kdata.Stock4hKdata attribute)
(zvt.domain.quotes.stock.stock_5m_hfq_kdata.Stock5mHfqKdata attribute)
(zvt.domain.quotes.stock.stock_5m_kdata.Stock5mKdata attribute)
(zvt.domain.quotes.stockhk.stockhk_1d_hfq_kdata.Stockhk1dHfqKdata attribute)
(zvt.domain.quotes.stockhk.stockhk_1d_kdata.Stockhk1dKdata attribute)
(zvt.domain.quotes.stockus.stockus_1d_hfq_kdata.Stockus1dHfqKdata attribute)
(zvt.domain.quotes.stockus.stockus_1d_kdata.Stockus1dKdata attribute)
(zvt.domain.quotes.TickCommon attribute)
(zvt.factors.z.domain.stock_1d_z_factor.Stock1dZFactor attribute)

(zvt.factors.z.domain.stock_1wk_z_factor.Stock1wkZFactor attribute)

W

WBCountryRecorder (class in zvt.recorders.wb.wb_country_recorder)
WBEconomyRecorder (class in zvt.recorders.wb.wb_economy_recorder)
WechatInformer (class in zvt.informer.informer)
WindowMethod (class in zvt.api.stats)
WrongKdataError

X

x0 (zvt.contract.drawer.Rect attribute)
x1 (zvt.contract.drawer.Rect attribute)

Y

y0 (zvt.contract.drawer.Rect attribute)
y1 (zvt.contract.drawer.Rect attribute)

Z

ZAccumulator (class in zvt.factors.z.z_factor)
ZFactor (class in zvt.factors.z.z_factor)
ZFactorCommon (class in zvt.factors.z.domain.common)
ZhDate (class in zvt.zhdate.zhdate)
zhongshu
(zvt.factors.z.domain.stock_1d_z_factor.Stock1dZFactor attribute)
(zvt.factors.z.domain.stock_1wk_z_factor.Stock1wkZFactor attribute)
zhongshu_change
(zvt.factors.z.domain.stock_1d_z_factor.Stock1dZFactor attribute)
(zvt.factors.z.domain.stock_1wk_z_factor.Stock1wkZFactor attribute)
zip_dir() (in module zvt.utils.zip_utils)
ZState (class in zvt.factors.z.z_factor)
zvt
module
zvt.api
module
zvt.api.intent
module
zvt.api.kdata
module
zvt.api.portfolio
module
zvt.api.selector
module
zvt.api.stats
module
zvt.api.utils
module
zvt.autocode
module
zvt.recorders
module
zvt.recorders.consts
module
zvt.recorders.eastmoney
module
zvt.recorders.eastmoney.common
module
zvt.recorders.eastmoney.dividend_finan
module
zvt.recorders.eastmoney.dividend_finan
module
zvt.recorders.eastmoney.dividend_finan
module
zvt.recorders.eastmoney.dividend_finan
module
zvt.recorders.eastmoney.dividend_finan
module
zvt.recorders.eastmoney.finance
module
zvt.recorders.eastmoney.finance.base_cl
module
zvt.recorders.eastmoney.finance.eastmo
module
zvt.recorders.eastmoney.finance.eastmo
module
zvt.recorders.eastmoney.finance.eastmo
module
zvt.recorders.eastmoney.finance.eastmo
module
zvt.recorders.eastmoney.holder
module
zvt.recorders.eastmoney.holder.eastmon
module

zvt.autocode.generator
module
zvt.autocode.templates
module
zvt.broker
module
zvt.broker.qmt
module
zvt.broker.qmt.errors
module
zvt.common
module
zvt.common.query_models
module
zvt.common.trading_models
module
zvt.consts
module
zvt.contract
module
zvt.contract.api
module
zvt.contract.base_service
module
zvt.contract.context
module
zvt.contract.data_type
module
zvt.contract.drawer
module
zvt.contract.factor
module
zvt.contract.model
module
zvt.contract.normal_data
module
zvt.contract.reader
module
zvt.contract.recorder
module
zvt.contract.register
module
zvt.contract.schema
module
zvt.contract.zvt_info
module
zvt.domain
module
zvt.domain.actor
module
zvt.domain.actor.actor_meta
module
zvt.domain.actor.stock_actor
module
zvt.domain.emotion
module

zvt.recorders.eastmoney.holder.eastmon
module
zvt.recorders.eastmoney.holder.eastmon
module
zvt.recorders.eastmoney.meta
module
zvt.recorders.eastmoney.meta.eastmone
module
zvt.recorders.eastmoney.meta.eastmone
module
zvt.recorders.eastmoney.trading
module
zvt.recorders.eastmoney.trading.eastmo
module
zvt.recorders.eastmoney.trading.eastmo
module
zvt.recorders.em
module
zvt.recorders.em.actor
module
zvt.recorders.em.actor.em_stock_actor_
module
zvt.recorders.em.actor.em_stock_ii_rec
module
zvt.recorders.em.actor.em_stock_top_te
module
zvt.recorders.em.actor.em_stock_top_te
module
zvt.recorders.em.em_api
module
zvt.recorders.em.macro
module
zvt.recorders.em.macro.em_treasury_yi
module
zvt.recorders.em.meta
module
zvt.recorders.em.meta.em_block_meta_
module
zvt.recorders.em.meta.em_currency_me
module
zvt.recorders.em.meta.em_future_meta_
module
zvt.recorders.em.meta.em_index_meta_
module
zvt.recorders.em.meta.em_indexus_met
module
zvt.recorders.em.meta.em_stock_meta_
module
zvt.recorders.em.meta.em_stockhk_met
module
zvt.recorders.em.meta.em_stockus_met
module
zvt.recorders.em.misc
module
zvt.recorders.em.misc.em_stock_news_
module

zvt.domain.emotion.emotion
module
zvt.domain.fundamental
module
zvt.domain.fundamental.dividend_financing
module
zvt.domain.fundamental.finance
module
zvt.domain.fundamental.trading
module
zvt.domain.fundamental.valuation
module
zvt.domain.macro
module
zvt.domain.macro.macro
module
zvt.domain.macro.monetary
module
zvt.domain.meta
module
zvt.domain.meta.block_meta
module
zvt.domain.meta.country_meta
module
zvt.domain.meta.currency_meta
module
zvt.domain.meta.etf_meta
module
zvt.domain.meta.fund_meta
module
zvt.domain.meta.future_meta
module
zvt.domain.meta.index_meta
module
zvt.domain.meta.indexus_meta
module
zvt.domain.meta.stock_meta
module
zvt.domain.meta.stockhk_meta
module
zvt.domain.meta.stockus_meta
module
zvt.domain.misc
module
zvt.domain.misc.holder
module
zvt.domain.misc.money_flow
module
zvt.domain.misc.overall
module
zvt.domain.misc.stock_news
module
zvt.domain.quotes
module
zvt.domain.quotes.block
module

zvt.recorders.em.quotes
module
zvt.recorders.em.quotes.em_kdata_reco
module
zvt.recorders.em.trading
module
zvt.recorders.em.trading.em_dragon_an
module
zvt.recorders.exchange
module
zvt.recorders.exchange.api
module
zvt.recorders.exchange.api.cn_index_ap
module
zvt.recorders.exchange.api.cn_index_st
module
zvt.recorders.exchange.api.cs_index_ap
module
zvt.recorders.exchange.api.cs_index_st
module
zvt.recorders.exchange.exchange_etf_m
module
zvt.recorders.exchange.exchange_index
module
zvt.recorders.exchange.exchange_index
module
zvt.recorders.exchange.exchange_stock
module
zvt.recorders.exchange.exchange_stock
module
zvt.recorders.joinquant
module
zvt.recorders.joinquant.common
module
zvt.recorders.joinquant.fundamental
module
zvt.recorders.joinquant.fundamental.jq_
module
zvt.recorders.joinquant.fundamental.jq_
module
zvt.recorders.joinquant.fundamental.jq_
module
zvt.recorders.joinquant.meta
module
zvt.recorders.joinquant.meta.jq_fund_m
module
zvt.recorders.joinquant.meta.jq_stock_n
module
zvt.recorders.joinquant.meta.jq_trade_d
module
zvt.recorders.joinquant.misc
module
zvt.recorders.joinquant.misc.jq_hk_hold
module
zvt.recorders.joinquant.misc.jq_index_r
module

zvt.domain.quotes.block.block_1d_kdata
module
zvt.domain.quotes.block.block_1mon_kdata
module
zvt.domain.quotes.block.block_1wk_kdata
module
zvt.domain.quotes.currency
module
zvt.domain.quotes.currency.currency_1d_kdata
module
zvt.domain.quotes.etf
module
zvt.domain.quotes.etf.etf_1d_kdata
module
zvt.domain.quotes.future
module
zvt.domain.quotes.future.future_1d_kdata
module
zvt.domain.quotes.index
module
zvt.domain.quotes.index.index_1d_kdata
module
zvt.domain.quotes.index.index_1wk_kdata
module
zvt.domain.quotes.indexus
module
zvt.domain.quotes.indexus.indexus_1d_kdata
module
zvt.domain.quotes.stock
module
zvt.domain.quotes.stock.stock_15m_hfq_kdata
module
zvt.domain.quotes.stock.stock_15m_kdata
module
zvt.domain.quotes.stock.stock_1d_hfq_kdata
module
zvt.domain.quotes.stock.stock_1d_kdata
module
zvt.domain.quotes.stock.stock_1h_hfq_kdata
module
zvt.domain.quotes.stock.stock_1h_kdata
module
zvt.domain.quotes.stock.stock_1m_hfq_kdata
module
zvt.domain.quotes.stock.stock_1m_kdata
module
zvt.domain.quotes.stock.stock_1mon_hfq_kdata
module
zvt.domain.quotes.stock.stock_1mon_kdata
module
zvt.domain.quotes.stock.stock_1wk_hfq_kdata
module
zvt.domain.quotes.stock.stock_1wk_kdata
module
zvt.domain.quotes.stock.stock_30m_hfq_kdata
module

zvt.recorders.joinquant.misc.jq_stock_n
module
zvt.recorders.joinquant.overall
module
zvt.recorders.joinquant.overall.jq_cross_
module
zvt.recorders.joinquant.overall.jq_margi
module
zvt.recorders.joinquant.overall.jq_stock_
module
zvt.recorders.joinquant.quotes
module
zvt.recorders.joinquant.quotes.jq_index_
module
zvt.recorders.joinquant.quotes.jq_stock_
module
zvt.recorders.jqka
module
zvt.recorders.jqka.emotion
module
zvt.recorders.jqka.emotion.JqkaEmotion
module
zvt.recorders.jqka.jqka_api
module
zvt.recorders.qmt
module
zvt.recorders.qmt.meta
module
zvt.recorders.sina
module
zvt.recorders.sina.meta
module
zvt.recorders.sina.meta.sina_block_reco
module
zvt.recorders.sina.money_flow
module
zvt.recorders.sina.money_flow.sina_blo
module
zvt.recorders.sina.money_flow.sina_sto
module
zvt.recorders.sina.quotes
module
zvt.recorders.sina.quotes.sina_etf_kdata
module
zvt.recorders.sina.quotes.sina_index_kd
module
zvt.recorders.wb
module
zvt.recorders.wb.wb_api
module
zvt.recorders.wb.wb_country_recorder
module
zvt.recorders.wb.wb_economy_recorder
module
zvt.rest
module

zvt.domain.quotes.stock.stock_30m_kdata
module
zvt.domain.quotes.stock.stock_4h_hfq_kdata
module
zvt.domain.quotes.stock.stock_4h_kdata
module
zvt.domain.quotes.stock.stock_5m_hfq_kdata
module
zvt.domain.quotes.stock.stock_5m_kdata
module
zvt.domain.quotes.stock.stock_quote
module
zvt.domain.quotes.stockhk
module
zvt.domain.quotes.stockhk.stockhk_1d_hfq_kdata
module
zvt.domain.quotes.stockhk.stockhk_1d_kdata
module
zvt.domain.quotes.stockus
module
zvt.domain.quotes.stockus.stockus_1d_hfq_kdata
module
zvt.domain.quotes.stockus.stockus_1d_kdata
module
zvt.domain.quotes.trade_day
module
zvt.factors
module
zvt.factors.algorithm
module
zvt.factors.factor_models
module
zvt.factors.factor_service
module
zvt.factors.fundamental
module
zvt.factors.fundamental.finance_factor
module
zvt.factors.ma
module
zvt.factors.ma.domain
module
zvt.factors.ma.domain.common
module
zvt.factors.ma.domain.stock_1d_ma_factor
module
zvt.factors.ma.domain.stock_1d_ma_stats_factor
module
zvt.factors.ma.ma_factor
module
zvt.factors.ma.ma_stats_factor
module
zvt.factors.ma.top_bottom_factor
module
zvt.factors.macd
module
zvt.rest.data
module
zvt.rest.factor
module
zvt.rest.trading
module
zvt.rest.work
module
zvt.samples
module
zvt.samples.stock_traders
module
zvt.sched
module
zvt.sched.sched
module
zvt.tag
module
zvt.tag.common
module
zvt.tag.tag_models
module
zvt.tag.tag_schemas
module
zvt.tag.tag_service
module
zvt.tag.tag_stats
module
zvt.tag.tag_utils
module
zvt.tag.tagger
module
zvt.trader
module
zvt.trader.errors
module
zvt.trader.sim_account
module
zvt.trader.trader
module
zvt.trader.trader_info_api
module
zvt.trader.trader_schemas
module
zvt.trading
module
zvt.trading.common
module
zvt.trading.trading_models
module
zvt.trading.trading_schemas
module
zvt.trading.trading_service
module
zvt.ui
module

zvt.factors.macd.macd_factor
module
zvt.factors.shape
module
zvt.factors.target_selector
module
zvt.factors.technical_factor
module
zvt.factors.top_stocks
module
zvt.factors.transformers
module
zvt.factors.z
module
zvt.factors.z.domain
module
zvt.factors.z.domain.common
module
zvt.factors.z.domain.stock_1d_z_factor
module
zvt.factors.z.domain.stock_1wk_z_factor
module
zvt.factors.z.z_factor
module
zvt.fill_project
module
zvt.informer
module
zvt.informer.inform_utils
module
zvt.informer.informer
module
zvt.main
module
zvt.ml
module
zvt.ml.lables
module
zvt.ml.ml
module
zvt.plugin
module
zvt.ui.assets
module
zvt.ui.components
module
zvt.ui.components.dcc_components
module
zvt.utils
module
zvt.utils.decorator
module
zvt.utils.file_utils
module
zvt.utils.git_utils
module
zvt.utils.model_utils
module
zvt.utils.pd_utils
module
zvt.utils.recorder_utils
module
zvt.utils.str_utils
module
zvt.utils.time_utils
module
zvt.utils.utils
module
zvt.utils.zip_utils
module
zvt.zhdate
module
zvt.zhdate.constants
module
zvt.zhdate.zhdate
module
zvt.zhdate.ztime
module
zvt_context (in module zvt.contract.con

Core concepts

Entity
: the existence described by itself, classification of existential concepts.

EntityEvent
: the event happened on the entity.

TradableEntity
: the Entity could be traded, e.g, Stock, Future.

ActorEntity
: the Entity acting in market, e.g, Fund, Individual, Government.

IntervalLevel
: repeated fixed time interval, e.g, 5m, 1d.

Schema
: the data structure with fields, one schema could have multiple storage with different Providers.

Kdata(Quote)
: the candlestick data with OHLC.

Provider
: the data provider.

Storage
: the sql database, e.g, sqlite, mysql.

Recorder
: class for recording data for Schema.

Factor
: data describing market.It's computed from from Schema, and save as new Schema if need.

TargetSelector
: the class select targets according to Factor.

Trader
: the backtest engine using TargetSelector, MLMachine or free style.

Tagger
: classify TradableEntity by different dimensions, could be used as ml category feature.

MLMachine
: the ml engine.

TradingSignal
: the signal contains information about how to trade.

Drawer
: the class for draw charts.

Intent
: what do you want to do, e.g. compare, distribute, composite.

Normal data
: the pandas dataframe with multiple index which level 0 is entity_id and level 1 is timestamp:

entity_id	**timestamp**	**col1**	**col2**	**col3**	**col4**
stock_sz_000338	2020-05-05	1.2	0.5	0.3	a
…	2020-05-06	1.0	0.7	0.2	b
stock_sz_000778	2020-05-05	1.2	0.5	0.3	a

entity_id	timestamp	col1	col2	col3	col4
…	2020-05-06	1.0	0.7	0.2	b

Usage

Here show some use cases of zvt.

Data

Factor

Backtesting

Machine learning

Draw by intent

Informer