

python api 文档

python api 文档	1
快速开始	3
安装Python	3
Windows	3
mac	6
下载sdk	8
安装配置	8
我的第一个策略	8
方式1 直接使用例子运行	10
方式2 创建自己的策略	10
Python API 范例	10
数据接口范例	10
历史数据提取	11
行情订阅	12
策略接口范例	12
获取用户持仓	12
查询用户的委托的订单列表	13
策略卖出	13
策略下单	14
综合范例	15
第一部分 定义策略类	15
第二部分 方法调用	16
Python api接口	17
常量定义	17
连接类型常量值	17
期货对冲常量值	17
下单方向	17
订单对持仓的影响	17
交易所	18
数据类型	18

订单状态	18
价格类型常量值	19
策略初始化	20
基础策略类	20
日志函数	20
简单策略类	21
策略开始方法	21
策略验证方法	21
数据提取方法	22
历史bar数据方法	22
最新bar数据获取方法	22
最新一天bar数据获取方法	23
指定一天bar数据获取方法	23
最新tick数据获取方法	24
历史tick数据获取方法	24
最新tick数据获取方法	25
用户操作方法	26
一日帐号详情获取方法	26
查询用户持仓详情	26
查询用户的委托的订单列表	26
订阅数据	27
用户下单	27
其他类	28
StrategyPosition	28
StrategyOrderDetail	29
StrategyAccountDetail	30
KlineBean	31
DailyKLineBean	32
TickBean	33

快速开始

安装Python

(下载地址:<https://www.python.org/downloads/release/python-2712/>)。

Windows

1.使用浏览器登录<https://www.python.org/downloads/release/python-2712/>前往下载Windows对应安装软件。

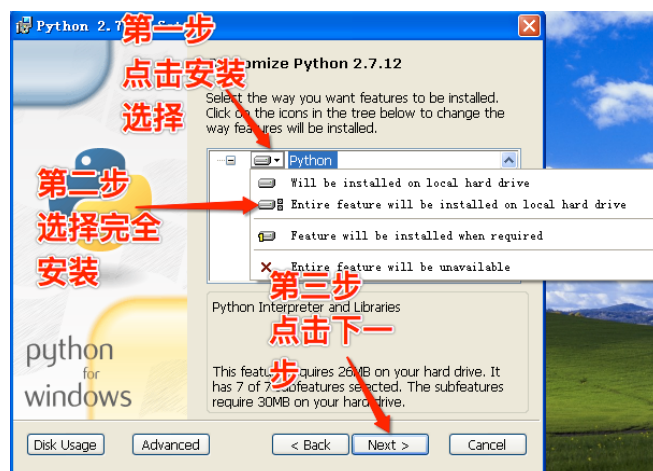
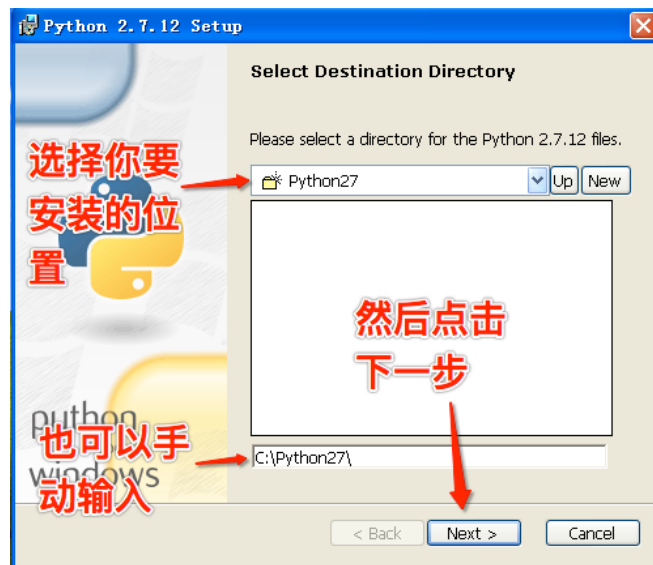
Version	Operating System	Description	MD5 Sum
Gzipped source tarball	Source release		88d61f82e3616
XZ compressed source tarball	Source release		57dffcee9cee8
Mac OS X 32-bit i386/PPC installer	Mac OS X	for Mac OS X 10.5 and later	3adbedcc935a
Mac OS X 64-bit/32-bit installer	Mac OS X	for Mac OS X 10.6 and later	86bedde2becc
Windows debug information files			1751598e1643
Windows debug information files for 64-bit binaries			c5433a7fca9ec
Windows help file	Windows		7bc4e15ecae8
Windows x86-64 MSI installer	Windows	64, not Itanium	8fa13925db87
Windows x86 MSI installer	Windows		fe0ef5b8fd027

windows 64位安装软件

windows 32位安装软件

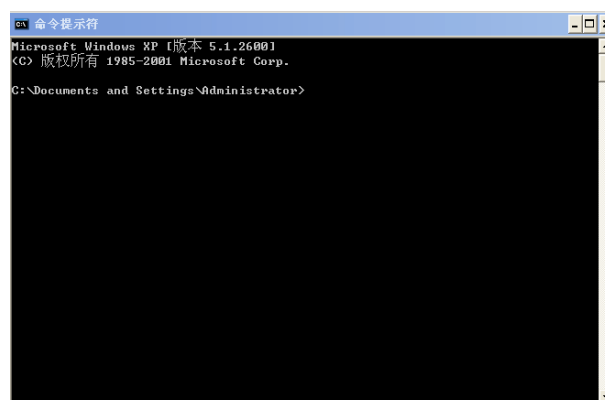
2.点击打开下载好的安装包

3进行如下步骤：





4. 验证安装是否成功



使用“Win+R”快捷键召唤出运行窗口，再在运行中输入cmd
出现黑色框体

在黑色框体中输入python
出现以下情况安装成功

```
C:\Documents and Settings\Administrator>python
Python 2.7.12 (v2.7.12:d33e0cf91556, Jun 27 2016, 15:19:22) [MSC v.1500 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

mac

1.使用浏览器登录<https://www.python.org/downloads/release/python-2712/>
前往下载mac os x 对应安装软件。

Files		
Version	Operating System	Description
Gzipped source tarball	Source release	
XZ compressed source tarball	Source release	
Mac OS X 32-bit i386/PPC installer	Mac OS X	for Mac OS X 10.5 and later
Mac OS X 64-bit/32-bit installer	Mac OS X	for Mac OS X 10.6 and later
Windows debug information files	Windows	
Windows debug information files for 64-bit binaries	Windows	
Windows help file	Windows	
Windows x86-64 MSI installer	Windows	for AMD64/EM64T/x64, not Itanium processors
Windows x86 MSI installer	Windows	

2.点击下载后软件





最后完成安装

3.打开应用程序文件夹,实用工具, 终端
输入Python

```
Python 2.7.11 (v2.7.11:6d1b6a68f775, Dec 5 2015, 12:54:16)
[GCC 4.2.1 (Apple Inc. build 5666) (dot 3)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> 
```

出现上图所示表示安装成功。

下载sdk

目前提供的sdk支持Windows、mac、linux下Python2.7(下载地址:<https://github.com/uxmc/sdk/archive/master.zip>)。

安装配置

1. 解压已经下载后的sdk压缩文件，放置到任意位置。
2. 使用快捷键win+R 打开运行窗口 输入cmd 打开dos黑框(windows)
打开应用程序文件夹,实用工具，终端(mac)
3. 切换到sdk文件夹下py2文件夹中
4. 输入python setup.py install 安装程序。
5. 验证安装，输入Python 进入 python command line 输入import sd 没有报错 则为安装成功

我的第一个策略

首先需前往网站(<http://mock.trade.uxmc.cn/quant-trade/login.html>)注册账号。



登录账户获取mock资金



绑定我的股票帐号

选择期货模拟或股票模拟

帐号种类 股票 经纪公司

请选择开户的经纪公司
上期技术
期货模拟
股票模拟

注意: 通过您指定的交易密码, 我们的后台系统将以获取帐号内现金和持仓信息。
我们将不会人工使用您的交易密码, 并且对交易密码进行加密保存。

确定 取消

点击



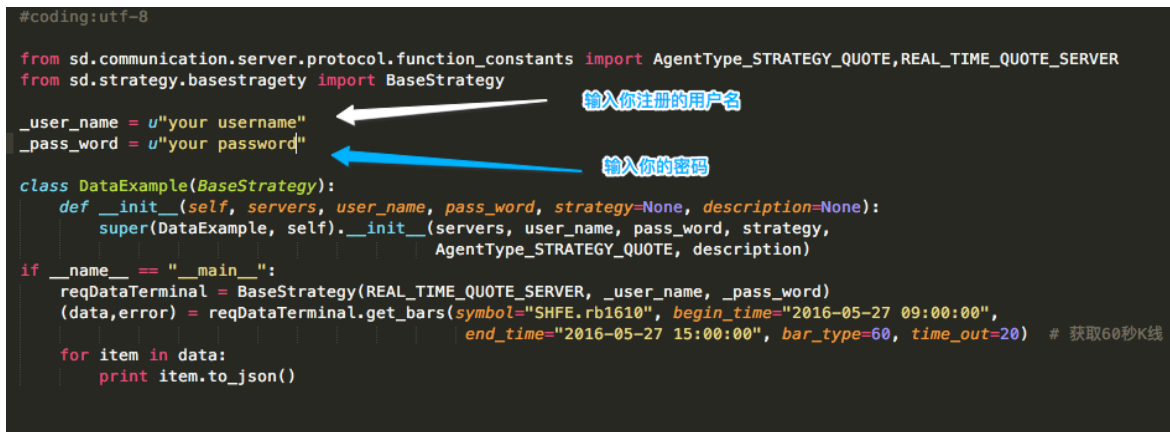
然后进行以下步骤

方式1 直接使用例子运行

使用sdk包中 py2/sd/example中的项目例子。

方式2 创建自己的策略

以下是一个完整的策略代码示例



```
#coding:utf-8
from sd.communication.server.protocol.function_constants import AgentType_STRATEGY_QUOTE, REAL_TIME_QUOTE_SERVER
from sd.strategy.basestrategy import BaseStrategy

_user_name = u"your username"
_pass_word = u"your password"

class DataExample(BaseStrategy):
    def __init__(self, servers, user_name, pass_word, strategy=None, description=None):
        super(DataExample, self).__init__(servers, user_name, pass_word, strategy,
                                          AgentType_STRATEGY_QUOTE, description)

if __name__ == "__main__":
    reqDataTerminal = BaseStrategy(REAL_TIME_QUOTE_SERVER, _user_name, _pass_word)
    (data,error) = reqDataTerminal.get_bars(symbol="SHFE.rb1610", begin_time="2016-05-27 09:00:00",
                                          end_time="2016-05-27 15:00:00", bar_type=60, time_out=20) # 获取60秒K线
    for item in data:
        print item.to_json()
```

编译策略并运行

以上代码是获取SHFE.RB1610 2016年5月27日从上午九点到下午3点 60秒K线

参数

symbol 证券代码 begin_time 起始时间 end_time 结束时间

bar_type bar周期 即 每多少秒的bar time_out 错误时退出时间

Python API 范例

数据接口范例

历史数据提取

1. 获取半小时K线

示例：提取上交所浦发银行2016年5月27日上午九点到下午三点半小时K线。

```
from sd.strategy.basestrategy import BaseStrategy
from sd.communication.server.protocol.function_constants import REAL_TIME_QUOTE_SERVER

_user_name = u"your username"
_pass_word = u"your password"

reqDataTerminal = BaseStrategy(REAL_TIME_QUOTE_SERVER, _user_name, _pass_word)

(data,error) = reqDataTerminal.get_bars(symbol="SHSE.600000", begin_time="2016-05-27 09:00:00",
                                       end_time="2016-05-27 15:00:00", bar_type=60 * 30, time_out=20)
for item in data:
    print item.to_json()
```

参数

symbol 证券代码 begin_time 起始时间 end_time 结束时间
bar_type bar周期 即 没多少秒的bar time_out 错误时退出时间

2. 获取60秒K线

示例：提取上交所浦发银行2016年5月27日上午九点到下午三点60秒K线。

```
from sd.strategy.basestrategy import BaseStrategy
from sd.communication.server.protocol.function_constants import REAL_TIME_QUOTE_SERVER

_user_name = u"your username"
_pass_word = u"your password"

reqDataTerminal = BaseStrategy(REAL_TIME_QUOTE_SERVER, _user_name, _pass_word)

(data,error) = reqDataTerminal.get_bars(symbol="SHSE.600000", begin_time="2016-05-27 09:00:00",
                                       end_time="2016-05-27 15:00:00", bar_type=60, time_out=20)
for item in data:
    print item.to_json()
```

参数

symbol 证券代码 begin_time 起始时间 end_time 结束时间
bar_type bar周期 即 每多少秒的bar time_out 错误时退出时间

3. 获取最新的10根半小时K线

示例：提取上交所张江高科最新的10根半小时K线。

```
from sd.strategy.basestrategy import BaseStrategy
from sd.communication.server.protocol.function_constants import REAL_TIME_QUOTE_SERVER

_user_name = u"your username"
_pass_word = u"your password"

reqDataTerminal = BaseStrategy(REAL_TIME_QUOTE_SERVER, _user_name, _pass_word)

(data,error) = reqDataTerminal.get_last_n_bars(symbol="SHSE.600895", n=10, bar_type=60 * 30,
                                              time_out=20)
for item in data:
    print item.to_json()
```

参数

symbol 证券代码 n 提取数据条数

bar_type bar周期 即 每多少秒的bar time_out 错误时退出时间

4. 获取指定时间范围的tick数据。

示例：提取上交所浦发银行2016年5月27日上午九点到下午三点tick数据。

```
from sd.strategy.basestrategy import BaseStrategy
from sd.communication.server.protocol.function_constants import REAL_TIME_QUOTE_SERVER

_user_name = u"your username"
_pass_word = u"your password"

reqDataTerminal = BaseStrategy(REAL_TIME_QUOTE_SERVER, _user_name, _pass_word)

(data,error) = reqDataTerminal.get_ticks(symbol="SHSE.600000", begin_time="2016-05-27 09:00:00",
                                         end_time="2016-05-27 15:00:00", time_out=20)
for item in data:
    print item.to_json()
```

参数

symbol 证券代码 begin_time 起始时间 end_time 结束时间

time_out 错误时退出时间

行情订阅

订阅浦发银行tick值。

ret = dataExample.req_subscribe(TICK, "SHSE.600000")

策略接口范例

获取用户持仓

```
from sd.communication.server.protocol.function_constants import *
from sd.configuration.logconfig import configure_log
from sd.strategy.simplestrategy import SimpleStrategy

_user_name = u"your username"
_pass_word = u"your password"

class DemoStrategy(SimpleStrategy):
    def __init__(self, trade_servers, quote_servers, user_name, pass_word, strategy=None, description=None):
        super(DemoStrategy, self).__init__(trade_servers, quote_servers, user_name, pass_word,
                                           strategy=strategy,
                                           description=description)

simple_strategy = DemoStrategy(TRADE_SERVER, REAL_TIME_QUOTE_SERVER, _user_name, _pass_word,
                              description="my demo strategy")

err = simple_strategy.init()
if err.errorId != 0:
    print err
    exit(1)

(result, error) = simple_strategy.req_position()
for item in result:
    print item.to_json()
```

req_position() 获取用户持仓

查询用户的委托的订单列表

```
from sd.communication.server.protocol.function_constants import *
from sd.configuration.logconfig import configure_log
from sd.strategy.simplestragety import SimpleStrategy

_user_name = u"your username"
_pass_word = u"your password"

class DemoStrategy(SimpleStrategy):
    def __init__(self, trade_servers, quote_servers, user_name, pass_word, strategy=None, description=None):
        super(DemoStrategy, self).__init__(trade_servers, quote_servers, user_name, pass_word,
                                           strategy=strategy,
                                           description=description)

simple_strategy = DemoStrategy(TRADE_SERVER, REAL_TIME_QUOTE_SERVER, _user_name, _pass_word,
                              description="my demo strategy")

err = simple_strategy.init()
if err.errorId != 0:
    print err
    exit(1)

(result, error) = simple_strategy.req_order_list()

for item in result:
    print item.to_json()
```

req_order_list()查询用户委托订单列表

策略卖出

[illegible]

send_order(证券代码,bs_flag = 下单方向,position_effect = 对持仓的影响,price_type = 价格的类型,
price = 委托价, volume = 委托量)

策略下单

```
from sd.communication.server.protocol.function_constants import *
from sd.configuration.logconfig import configure_log
from sd.strategy.simplestragetey import SimpleStrategy

_user_name = u"your username"
_pass_word = u"your password"

class DemoStrategy(SimpleStrategy):
    def __init__(self, trade_servers, quote_servers, user_name, pass_word, strategy=None, description=None):
        super(DemoStrategy, self).__init__(trade_servers, quote_servers, user_name, pass_word,
                                            strategy=strategy,
                                            description=description)

simple_strategy = DemoStrategy(TRADE_SERVER, REAL_TIME_QUOTE_SERVER, _user_name, _pass_word,
                              description="my demo strategy")
err = simple_strategy.init()
if err.errorId != 0:
    print err
    exit(1)
(order_id, error) = simple_strategy.send_order("SHFE.rb1701", bs_flag=BSFlag_BUY,
                                              position_effect=PositionEffect_OPEN,
                                              price_type=PRTP_FIX,
                                              price=2089, volume=4)
```

send_order(证券代码,bs_flag = 下单方向,position_effect = 对持仓的影响,price_type = 价格的类型,
price = 委托价, volume = 委托量)

综合范例

第一部分 定义策略类

```
import logging #引用python日志系统
from sd.communication.server.protocol.function_constants import * #引用所有的变量
from sd.configuration.logconfig import configure_log #引用提供的日志
from sd.strategy.simplestragety import SimpleStrategy #引用提供的策略

_user_name = u"your-user-name"#定义你的账号
_pass_word = u"your-pass-word"#定义你的密码
_strategy_id = "your-strategy-name"#定义你的策略名

class DemoStrategy(SimpleStrategy):#定义你的策略,继承自引用策略

    '''定义成员变量
    分别含义
    trade_servers 交易服务端      quote_servers 数据服务端      user_name 账号
    pass_word 密码      strategy 策略      description 描述
    '''

    def __init__(self, trade_servers, quote_servers, user_name, pass_word, strategy=None, description=None):
        super(DemoStrategy, self).__init__(trade_servers, quote_servers, user_name, pass_word,
                                             strategy=strategy,description=description)

    '''
    下单通知的订单交易最新情况,接收后更新状态,字段参考class StrategyOrderDetail, 判断订单执行情况
    '''

    def on_notify_quote(self, data_type, data):
        logging.info("%s", data.to_json())

    def on_notify_order_detail(self, order_detail):
        logging.info("[on_notify_order_detail]:%s", order_detail.to_json())

    def on_notify_connection_error(self, reason=""):
        logging.warn("on disconnection:" + reason)

    def on_command_strategy_exit(self, reason=None):
        logging.warn("on command exit")
```

1 引用所需python自带包、sdk提供变量、函数、策略类

2 定义所需变量，如账号、密码、策略名

3 定义你的策略类、策略方法、日志输出类型

第二部分 方法调用

```
if __name__ == "__main__":
    configure_log("DemoStrategy.log", logging.DEBUG) # 日志打印地址
    simple_strategy = DemoStrategy(TRADE_SERVER, REAL_TIME_QUOTE_SERVER, _user_name, \
        _pass_word, _strategy_id, description="my demo strategy") # 实例化你的策略

    ''' 验证是否连接成功 '''
    err = simple_strategy.init()
    if err.errorId != 0:
        logging.error("Failed to login, exit:%s", err.to_json())
        exit(1)

    ''' 查询用户的持仓详情 '''
    (result, error) = simple_strategy.req_position()
    if error.errorId != 0:
        logging.error(error.to_json())
    else:
        for item in result:
            logging.info(item.to_json())

    ''' 获取半小时K线 '''
    (data, error) = simple_strategy.get_bars(symbol="SHSE.600895", begin_time="2016-05-27 09:00:00", \
        end_time="2016-05-27 15:00:00", bar_type=60 * 30, time_out=20)
    if data is not None:
        for item in data:
            print item.to_dict()

    ''' 订阅数据 '''
    error = simple_strategy.req_subscribe(TICK, "SHFE.rb1610")
    if error.errorId != 0:
        logging.error(error.to_json())

    ''' 买入下单 '''
    (order_id, error) = simple_strategy.send_order("SHFE.rb1701", bs_flag=BSFlag_BUY,
        position_effect=PositionEffect_OPEN,
        price_type=PRTP_FIX,
        price=2089, volume=4)

    simple_strategy.run() # 策略运行
```

- 1 设置打印日志
- 2 验证连接结果
- 3 数据获取订阅
- 4 订单买入卖出
- 5 策略运行

Python api接口

常量定义

连接类型常量值

(import sd.communication.server.protocol.function_constants)

AgentType_STRATEGY_TRADE	交易策略
AgentType_STRATEGY_QUOTE	数据策略
REAL_TIME_QUOTE_SERVER	实时数据服务器
HISTORY_QUOTE_SERVER	历史数据服务器
TRADE_SERVER	交易服务连接

期货对冲常量值

(import sd.communication.server.protocol.function_constants)

HEDGE_FLAG_SPECULATION
HEDGE_FLAG_ARBITRAGE
HEDGE_FLAG_HEDGE

下单方向

(import sd.communication.server.protocol.function_constants)

BSFlag_BUY	卖出
BSFlag_SELL	买入

订单对持仓的影响

(import sd.communication.server.protocol.function_constants)

PositionEffect_OPEN	开仓
PositionEffect_CLOSE	平仓
PositionEffect_CLOSE_TODAY	平今，对上期所有效
PositionEffect_CLOSE_YESTERDAY	平昨，对上期所有效

交易所

(import sd.communication.server.protocol.function_constants)

EXCHANGEID_SHFE	上期所
EXCHANGEID_DCE	大商所
EXCHANGEID_CFFEX	中金所
EXCHANGEID_CZCE	郑商所
EXCHANGEID_SHSE	上海股票交易所
EXCHANGEID_SZSE	深圳股票交易所

数据类型

(import sd.communication.server.protocol.function_constants)

TICK	tick数据
BAR	K线数据
DAILY_BAR	日线数据

订单状态

(import sd.communication.server.protocol.function_constants)

ORDER_STATUS_CREATED	创建订单
ORDER_STATUS_UNREPORTED	订单待确认
ORDER_STATUS_REPORTED	订单已确认
ORDER_STATUS_NOT_TRADE	未成交
ORDER_STATUS_PART_SUCC	部分成交
ORDER_STATUS_SUCCEEDED	交易成交
ORDER_STATUS_UNREPORTED_CANCEL	取消订单待确认
ORDER_STATUS_REPORTED_CANCEL	取消订单已确认
ORDER_STATUS_INVALID	废单
ORDER_STATUS_CANCELED	取消成功

价格类型常量值

(import sd.communication.server.protocol.function_constants)

PRTP_FIX	限价单
PRTP_MARKET	市价单
PRTP_OPPOSIT	对手价格

策略初始化

基础策略类

(from sd.strategy.simplestragety import BaseStrategy)

BaseStrategy(servers, user_name, pass_word, strategy=None, description=None)

参数	类型	说明
servers	常量值	需要连接的服务器(引用常量)
user_name	string	账号
pass_word	string	密码
strategy	string	策略名称
description	string	策略描述

日志函数

(from sd.configuration.logconfig import configure_log)

configure_log(filename, level)

参数	类型	说明
filename	string	日志名
level	logging.{value}	日志级别

logging 模块日志级别

CRITICAL 、 ERROR 、 WARNING 、 INFO 、 DEBUG 、 NOTSET

简单策略类

(from sd.strategy.simplestrategy import SimpleStrategy)

BaseStrategy(servers, user_name, pass_word, strategy, usage_type, description, mode=1, start_time, end_time, base_money, symbol, commission_ratio, adjust_price_type, history_data_server)

参数	类型	说明
servers	string	需要连接的服务器(一般引用常量)
user_name	string	账号
pass_word	string	密码
strategy	string	策略名称
description	string	策略描述
mode	int	策略模式, 1表示实盘, 2表示回测
start_time	string	起始时间,如2016-05-27 09:00:00
end_time	string	结束时间,如2016-05-27 09:00:00
base_money	int	回测初始资金
symbol	string	回测用到的数据类型
commission_ratio	float	佣金比例
adjust_price_type	int	调价类型
history_data_server	常量值	历史数据服务端

灰色字段为回测使用字段

策略开始方法

run()

策略验证方法

init()

数据提取方法

历史bar数据方法

get_bars(symbol, begin_time, end_time, bar_type, time_out)

参数	类型	说明
symbol	string	证券代码
begin_time	string	起始时间,如2016-05-27 09:00:00
end_time	string	结束时间,如2016-05-27 09:00:00
bar_type	int	bar周期，以秒为单位，比如60即1分钟bar
time_out	int	错误退出时间

返回值:

查询数据,错误消息

最新bar数据获取方法

get_last_n_bars(symbol, bar_type, n, time_out)

参数	类型	说明
symbol	string	证券代码
n	int	提取的数据条数
bar_type	int	bar周期，以秒为单位，比如60即1分钟bar
time_out	int	错误退出时间

返回值:

查询数据,错误消息

最新一天bar数据获取方法

get_last_n_daily_bars(symbol, n, time_out)

参数	类型	说明
symbol	string	证券代码
n	int	提取的数据条数
time_out	int	错误退出时间

返回值:

查询数据,错误消息

指定一天bar数据获取方法

get_daily_bars(symbol, begin_time, end_time, time_out)

参数	类型	说明
symbol	string	证券代码
begin_time	string	起始时间,如2016-05-27 09:00:00
end_time	string	结束时间,如2016-05-27 09:00:00
bar_type	int	bar周期, 以秒为单位, 比如60即1分钟bar
time_out	int	错误退出时间

返回值:

查询数据,错误消息

最新tick数据获取方法

get_last_n_ticks(symbol, n, time_out)

参数	类型	说明
symbol	string	证券代码
n	int	提取的数据条数
time_out	int	错误退出时间

返回值:

查询数据,错误消息

历史tick数据获取方法

get_ticks(symbol, begin_time, end_time,time_out)

参数	类型	说明
symbol	string	证券代码
begin_time	string	起始时间,如2016-05-26
end_time	string	结束时间,如2016-05-27
time_out	int	错误退出时间

返回值:

查询数据,错误消息

最新tick数据获取方法

get_last_ticks(symbol, time_out)

参数	类型	说明
symbol	string	证券代码
time_out	int	错误退出时间

返回值:

查询数据,错误消息

用户操作方法

一日帐号详情获取方法

req_account_detail(trading_day)

参数	类型	说明
trading_day	string	需要获取的帐号详情日期,如: 20160827,不指定获取当前数据

返回值:

查询数据

查询用户持仓详情

req_position(strategy_id)

参数	类型	说明
strategy_id	string	策略id, 默认为None查询用户所有数据

返回值:

查询数据

查询用户的委托的订单列表

req_order_list(strategy_id=None)

参数	类型	说明
strategy_id	string	策略id,默认为None查询用户所有数据

返回值:

查询数据

订阅数据

req_subscribe(data_type, symbol)

参数	类型	说明
data_type	常量值	需要返回数据类型,使用定义常量,如:TICK,BAR
symbol	string	证券代码

返回值:

错误提示

用户下单

send_order(symbol, bs_flag, position_effect, price_type, price, volume, strategy, hedge_type, order_time)

参数	类型	说明
symbol	string	证券代码
bs_flag	常量值	下单方向
position_effect	常量值	订单对持仓的影响
price_type	常量值	价格类型
price	float	委托价
volume	float	委托量
strategy	string	策略名称
hedge_type	常量值	对冲类型
order_time	string	下单时间,只用于回测

返回值:

订单id,错误提示

其他类

StrategyPosition

(from sd.strategy.bean.bean import StrategyPosition)

参数	类型	说明
userName	string	用户名
userId	int	用户id
tradingDay	string	交易日,不指定获取当前数据
exchange	string	交易所
exchangeName	string	交易所名称
strategyId	string	策略id
strategyName	string	策略名
bsFlag	常量	方向
product	string	产品名
instrument	string	合约,如:SHFE.rb1610
instrumentName	string	合约名
yesterdayPosition	float	昨仓
todayPosition	float	今仓
closeProfit	float	平仓盈亏
floatProfit	float	浮动盈亏
avgPrice	float	均价
settlementPrice	float	结算价
lastSettlePrice	float	昨日结算价
usedCommission	float	手续费
usedMargin	float	保证金
marketValue	float	市价
assertType	int	资产类型
createTime	string	创建市价
updateTime	string	更新市价
frozenVolume	float	冻结手数目,上期所表示冻结的昨仓
frozenTodayVolume	float	冻结今仓手数目,上期所使用

StrategyOrderDetail

(from sd.strategy.bean.bean import StrategyOrderDetail)

参数	类型	说明
orderId	long	订单id
userId	int	用户id
userName	string	用户名
strategyId	string	策略id
strategyName	string	合约id,如:SHFE.rb1610
instrumentName	string	合约名
orderPrice	float	下单价格
orderPriceType	int	下单价格类型
volumeOriginal	int	下单手数
bsFlag	常量	下单方向
positionEffect	常量	期货开平表示
hedgeFlag	常量	期货对冲标示
exchange	string	交易所
exchangeOrderId	string	交易所订单id
tradePrice	float	成交均价
tradedVolume	int	成交手数
tradedAmount	float	成交金额
orderStatus	int	订单状态
tradingDay	string	交易日
createTime	string	订单创建时间
errorMsg	string	错误信息
orderType	int	订单类型
extentOrderInfo	string/json	扩展订单信息

StrategyAccountDetail

(from sd.strategy.bean.bean import StrategyAccountDetail)

参数	类型	说明
brokerId	string	服务id
accountName	string	账号
userName	string	用户名
userId	int	用户id
withdrawQuota	float	可取资金
available	float	可用资金
closeProfit	float	平仓盈亏
positionProfit	float	持仓盈亏费
deposit	float	入金
withdraw	float	出金
preBalance	float	期初权益
frozenMargin	float	冻结保证金
frozenCommission	float	冻结手续费
currMargin	float	当前占用保证金
balance	float	动态总权益
createTime	string	创建日
updateTime	string	更新时间
tradingDay	string	交易日

KlineBean

(from sd.strategy.bean.databean import KlineBean)

参数	类型	说明
exchange	string	交易所
instrumentID	string	证券ID
barType	int	bar类型，以秒为单位，比如1分钟 bar = bar_type=60
strTime	string	可视化时间
updateTime	string	行情时间戳
openPrice	float	开盘价
highPrice	float	最高价
lowPrice	float	最低价
closePrice	float	收盘价
volume	int	委托量
turnover	float	成交量
preClosePrice	float	前一天收盘价
position	float	仓位
adjFactor	float	复权因子
flag	int	除权出息标记

DailyKLineBean

(from sd.strategy.bean.databean import DailyKLineBean)

参数	类型	说明
exchange	string	交易所
instrumentID	string	证券ID
barType	int	bar类型，以秒为单位，比如1分钟 bar = bar_type=60
strTime	string	可视化时间
updateTime	string	行情时间戳
openPrice	float	开盘价
highPrice	float	最高价
lowPrice	float	最低价
closePrice	float	收盘价
volume	int	委托量
turnover	float	成交量
position	float	仓位
settlePrice	float	结算价
upperLimitPrice	float	涨停价
lowerLimitPrice	float	前一天收盘价
adjFactor	float	复权因子
flag	float	除权出息标记

TickBean

(from sd.strategy.bean.databean import TickBean)

参数	类型	说明
exchange	string	交易所
instrumentID	string	证券ID
strTime	string	可视化时间
updateTime	string	行情时间戳
openPrice	float	开盘价
highPrice	float	最高价
lowPrice	float	最低价
closePrice	float	收盘价
volume	int	委托量
turnover	float	成交量
position	float	仓位
lastPrice	float	昨日收盘价
upperLimitPrice	float	涨停价
lowerLimitPrice	float	跌停价
settlePrice	float	结算价
preClosePrice	float	昨日收盘价
askPrice	float	卖方价
askVolume	float	卖方价成交量
bidPrice	float	买方价
bidVolume	float	买方价量