BackTestApi股票回测工具使用指南

目录

[BackTestApi股票回测工具使用指南 1](#_Toc39745521)

[1 BackTestApi介绍 2](#_Toc39745522)

[1.1 概述 2](#_Toc39745523)

[1.2 BackTestApi接口文件 2](#_Toc39745524)

[1.3 BackTestApi接口类 3](#_Toc39745525)

[2 回测前必须知道的信息 4](#_Toc39745526)

[2.1 BackTestApi与奇点生产API有哪些不同 5](#_Toc39745527)

[3 如何快速开始回测（C++） 6](#_Toc39745528)

[3.1 开发环境准备 7](#_Toc39745529)

[3.2 设置回测初始参数 7](#_Toc39745530)

[3.3 如何编写回测程序 8](#_Toc39745531)

[3.3.1 初始化TraderAPI 8](#_Toc39745532)

[3.3.2 初始化MdAPI 9](#_Toc39745533)

[3.3.3 连续回测多个交易 10](#_Toc39745534)

[3.3.4 登录Login 11](#_Toc39745535)

[3.3.5 报单申报 16](#_Toc39745536)

[3.3.6 如何运行回测实例 17](#_Toc39745537)

[3.3.7 查询回测结果 17](#_Toc39745538)

[4 如何快速开始回测（Python） 19](#_Toc39745539)

[4.1 开发环境准备 20](#_Toc39745540)

[4.2 如何运行回测程序 20](#_Toc39745541)

[5 回测工具支持哪些证券品种 21](#_Toc39745542)

[6 回测如何撮合成交 21](#_Toc39745543)

[7 BackTestApi如何使用详解 22](#_Toc39745544)

[7.1 交易接口和行情接口的初始化 22](#_Toc39745545)

[7.2 登录&订阅行情 23](#_Toc39745546)

[7.3 行情推送 25](#_Toc39745547)

[7.4 报单 25](#_Toc39745548)

[7.4.1 报单录入 25](#_Toc39745549)

[7.4.2 报单录入响应 27](#_Toc39745550)

[7.4.3 报单状态推送 28](#_Toc39745551)

[7.4.4 成交回报 29](#_Toc39745552)

[7.4.5 报单序列号 29](#_Toc39745553)

[7.4.6 撤单 30](#_Toc39745554)

[8 回测BackTestApi支持生产API的哪些接口 31](#_Toc39745555)

[8.1 支持生产mdapi哪些接口 31](#_Toc39745556)

[8.2 支持生产traderapi哪些接口 34](#_Toc39745557)

[9 回测BackTestApi支持查询股票基本面数据 37](#_Toc39745558)

# BackTestApi介绍

## 概述

感谢您使用BackTestApi股票回测工具，本文主要介绍了BackTestApi的使用方法。

奇点快速股票交易系统（简称奇点交易系统）是一款快速证券交易柜台系统，并提供交易API和行情API（简称奇点Api）接口来编写策略程序。

BackTestApi股票回测工具支持对“基于奇点交易系统API编写的策略”进行回测。

BackTestApi采用与“奇点交易系统”的完全一致的API接口，无缝对接实盘。因此，BackTestApi接口编写的策略程序经过回测后，可以直接用“奇点交易系统API”替换掉BackTestApi就可以进行实盘交易。

需要注意的是：BackTestApi只能用来做回测，不能用来连接真实的奇点交易系统。

## BackTestApi接口文件

BackTestApi采用与“奇点交易系统”的完全一致的API接口，并且接口文件的名称也完全相同。

目前，BackTestAPI支持平台和编程语言接口：

|  |  |  |
| --- | --- | --- |
| 操作系统 | C++ | Python（**只支持*3.6*版本**） |
| 64位Linux | √ | √ |
| 32位Windows | √ | √ |
| 64位Windows | √ | √ |

C++接口文件列表如下：

|  |  |
| --- | --- |
| 文件名 | 详情 |
| TORATstpTraderApi.h | 包含交易相关的请求和回调接口 |
| TORATstpMdApi.h | 包含获取行情相关的请求和回调接口 |
| TORATstpUserApiStruct.h | 包含了所有用到的数据结构。 |
| TORATstpUserApiDataType.h | 包含了所有用到的数据类型。 |
| traderapi.dll, traderapi.lib (windows)  libtraderapi.so (linux) | 交易部分的动态链接库和静态链接库 |
| mdapi.dll, mdapi.lib (windows)  libmdapi.so (linux) | 行情部分的动态链接库和静态链接库 |

Python接口文件列表如下：

|  |  |
| --- | --- |
| 文件名 | 详情 |
| mdapi.py | 行情相关Python接口 |
| \_mdapi.pyd(windows)  \_mdapi.so(linux) | 行情Python转换C++接口组件 |
| traderapi.py | 交易相关Python接口 |
| \_traderapi.pyd(windows)  \_traderapi.so(linux) | 交易Python转换C++接口组件 |
| traderapi.dll(windows)  libtraderapi.so (linux) | 交易部分的动态链接库 |
| mdapi.dll(windows)  libmdapi.so (linux) | 行情部分的动态链接库 |

## BackTestApi接口类

CTORATstpTraderApi是交易API，负责主动发起请求查询和订阅交易回报信息的接口函数；CTORATstpTraderSpi含有对应所有的响应和回报函数，用于接收证券柜台交易系统发送的请求响应和回报信息，开发者需要继承该接口类，并实现其中相应的虚函数。

CTORATstpMdApi是行情API，负责主动发起查询基本面数据和订阅行情信息的接口函数；CTORATstpMdSpi含有对应所有的响应和回报函数，用于接收证券柜台交易系统发送的行情信息和查询结果，开发者需要继承该接口类，并实现其中相应的虚函数。

接口函数主要有请求，响应，回报，错误响应这四种。除了请求类型的接口函数属于请求CTORATstpTraderApi或者CTORATstpMdApi类，其余都属于CTORATstpTraderSpi或者CTORATstpMdSpi。

|  |  |  |  |
| --- | --- | --- | --- |
| 消息 | 格式 | 示例 | 所属类 |
| 请求 | Req\*\*\* | ReqUserLogin  ReqQrySecurity | CTORATstpTraderApi或者CTORATstpMdApi |
| 响应 | OnRsp\*\*\* | OnRspUserLogin  OnRspQrySecurity | CTORATstpTraderSpi或者CTORATstpMdSpi |
| 回报 | OnRtn\*\*\* | OnRtnOrder | CTORATstpTraderSpi或者CTORATstpMdSpi |
| 错误回报 | OnErrRtn\*\*\* | OnErrRtnOrderInsert | CTORATstpTraderSpi或者CTORATstpMdSpi |

# 回测前必须知道的信息

开始回测前，必须要知道以下信息

|  |  |
| --- | --- |
| * BackTestApi只能用来做回测，不能代替生产的API去连接生产交易系统。 * 编写回测程序时，必须同时使用BackTestApi的mdapi.dll，traderapi.dll去完成登录交易引擎和行情引擎，不能单独使用其中一个。 * 回测服务器的地址是**tcp://210.14.72.18:26666；**用户必须使用回测traderapi的RegisterFront函数来注册此地址去连接上回测服务器。 * 回测服务的历史行情是从2008年1月开始至今的不复权行情。 * 目前回测服务允许从2008年1月开始至今的时间段，其他时间段目前暂不支持。因此配置的回测时间段必须在规定的时间段内，且回测开始日期和结束日期都需是有效交易日。 * BackTestApi不支持在开盘和收盘集合竞价时段报单。 * 如果只进行单个交易日的回测，则收到的行情为秒级行情；如果进行多个交易日的回测，则收到的行情为分钟级行情。 * 每个用户都有自己特有的回测交易账号和密码，用户必须通过BackTestApi中的traderapi的ReqUserLogin接口输入交易账号和密码才能成功登录到回测服务器上。获取自己的交易账号和密码，需要登录N视界网站<http://www.n-sight.com.cn>，点击个人中心按钮去进入个人信息页面，然后在个人信息页面中点击交易密码显示按钮获取账号密码（如下图所示）。  |  | | --- | |  | |

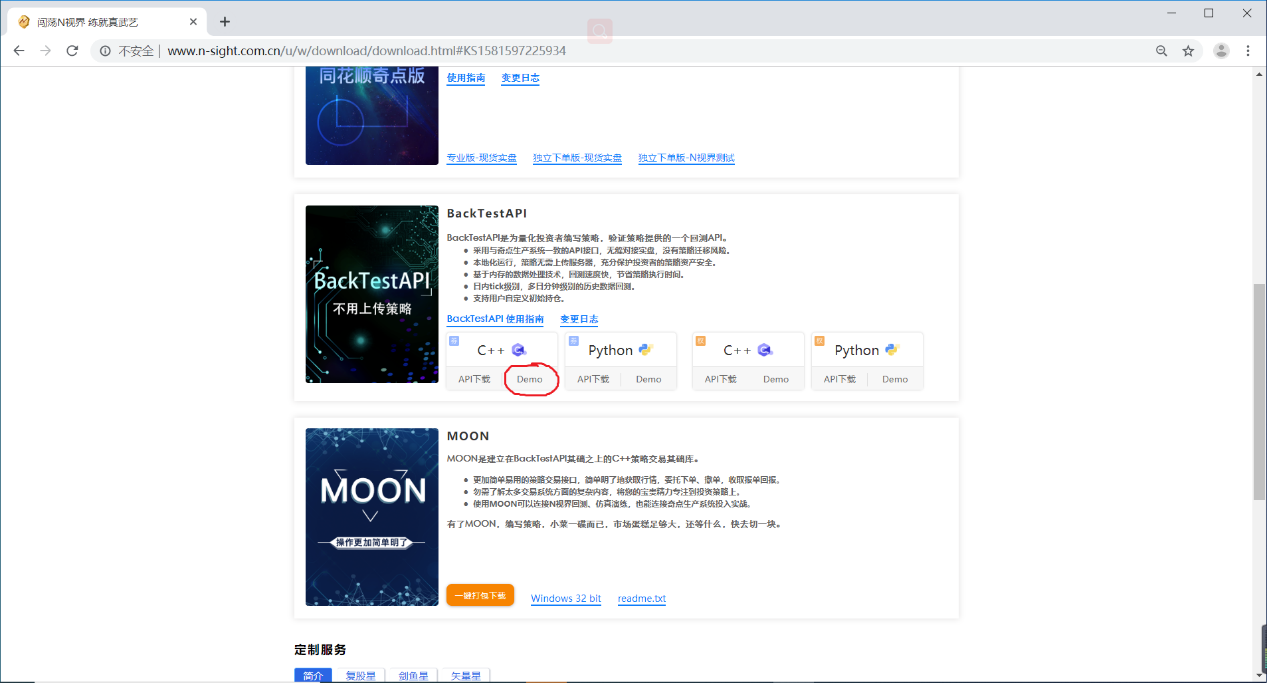
## BackTestApi与奇点生产API有哪些不同

|  |
| --- |
| * BackTestApi只能用来做回测，不能代替生产的API去连接生产交易系统。 * 编写回测程序时，必须同时使用BackTestApi的mdapi.dll，traderapi.dll去完成登录交易引擎和行情引擎，不能单独使用其中一个；而奇点生产API则可以单独使用其中一个API。 * 需要在traderapi.ini配置文件中设置回测的初始化信息；而奇点生产API不需要配置文件。 * BackTestApi回测多个交易日，那么在每个交易日开始时，必需重新调用mdapi和traderapi的init函数；而当每个交易日结束时，调用的mdapi和traderapi的join函数会立即退出，且如果所有交易日都完成则join函数返回整数值0，否则返回1。 * 调用traderapi的ReqUserLogin函数去登录时，可以在登录请求报文中的TerminalInfo字段里填写本次回测的名称，还可以用登录请求报文中的UserProductInfo字段来设置回测的历史行情起始日期（traderapi.ini中的SubMD\_Begin\_date字段会失效）。 * 由于回测交易是由行情来驱动的，因此BackTestApi的mdapi在登录成功的OnRspUserLogin回调必须立刻订阅行情；而奇点生产的mdapi没有这个要求，只要在登录成功后随时都可以订阅行情。 * 虽然BACKTESTAPI和奇点生产API的接口相同，但是BACKTESTAPI只支持部分的奇点生产API接口。详情请查看第9章节。 |

# 如何快速开始回测（C++）

为了快速开始上手回测，本节将会介绍一个windows平台上的简单示例（BackTestAPI\_demo\_c.zip），来演示如何使用BackTestApi去编写和执行一个C++语言编写的回测程序。

示例下载，可以通过登录N视界网站<http://www.n-sight.com.cn>，并访问百宝箱页面<http://www.n-sight.com.cn/u/w/download/download.html>，然后点击BackTestApi的C++ Demo按钮去下载（如下图所示）。



## 开发环境准备

* 需要安装visual studio 2013以上版本
* BackTestApi C++组件traderapi.dll，mdapi.dll，traderapi.lib，mdapi.lib，以及配置文件traderapi.ini都需要放置在执行程序目录下面。

## 设置回测初始参数

首先需要在traderapi.ini配置文件中设置回测的初始化参数。

在本例子的traderapi.ini文件，可以看到设置回测时间段为20190102到20190108，初始资金设置为10000000，使用缓存行情。使用缓存行情机制可以避免重复下载同一个股票的相同行情。

|  |
| --- |
| #订阅历史行情的起始日期  SubMD\_Begin\_date=20190102  #订阅历史行情的结束日期  SubMD\_End\_date=20190108  #用户设置初始资金。如果不设置则使用默认金额10000000  InitMoney=10000000  #是否使用缓存行情，如果不设置则默认使用  UseMDCache=1  #股票佣金费率，如果不设置默认0.0003（万三）  StockFeeRatio=0.0003  #股票最低佣金金额，如果不设置默认0  StockFeeMin=0  #是否上传回测结果到回测服务器，如果不设置则默认上传  IsToUploadResult=1  #初始持仓需要在InitPosition配置项里设置：设置格式 (交易所代码,股票代码,数量),如果#设置多个持仓请用分号分隔，交易所代码1是上海2是深圳。  #例如设置初始持仓浦发5000股，万科6000股。  InitPosition=(1,600000,5000);(2,000002,6000) |

在traderapi.ini配置文件中，必须要配置行情起始日期“SubMD\_Begin\_date”和结束日期“SubMD\_End\_date”，这两项配置决定了回测的起止时间段。目前回测服务允许从20180102开始至今的时间段，其他时间段目前暂不支持，因此配置的回测起止时间段必须符合规定的时间段内，并且起始日期和结束日期都需是有效交易日。

## 如何编写回测程序

在本节，将会以示例程序为例（BackTestAPI\_demo\_c.zip），演示一下如何编写一个简单回测程序。此程序由simspi.h，simspi.cpp，demo.cpp三个源代码文件组成。

### 初始化TraderAPI

建立**SimpleTraderSpi**类继承交易响应回调基类**CTORATstpTraderSpi**，并且实现其中所需的回调函数，具体实现请查看**simspi.h**文件。

|  |  |
| --- | --- |
| OnFrontConnected | 当客户端与交易后台建立起通信连接时（还未登录前），该方法被调用。 |
| OnFrontDisconnected | 当客户端与交易后台通信连接断开时，该方法被调用。 |
| OnRspUserLogin | 登录应答 |
| OnRspOrderInsert | 报单录入应答 |
| OnRspOrderAction | 撤单申报应答 |
| OnRtnOrder | 报单回报通知推送应答，每次投资者报单状态变化都会推送报单回报。 |
| OnRtnTrade | 成交回报通知推送应答，每次投资者申报报单有成交就会推送成交回报。 |
| OnRspQryShareholderAccount | 查询股东账户结果应答 |
| OnRspQryTradingAccount | 查询资金账户结果应答 |
| OnRspQryPosition | 查询持仓结果应答 |
| OnRspQryOrder | 查询报单结果应答 |
| OnRspQryOrderAction | 查询撤单记录结果应答 |
| OnRspQryTrade | 查询成交结果应答 |
| OnRspQryTradingFee | 查询交易所基础税费结果应答 |
| OnRspQryInvestorTradingFee | 查询投资者佣金费率结果应答 |

初始化TraderApi实例，创建自定义的交易回调类实例，设置私有流和公有流的订阅方式，并且注册回测服务器的地址。

|  |
| --- |
| CTORATstpTraderApi\* tradeapi = CTORATstpTraderApi::CreateTstpTraderApi();  SimpleTraderSpi\* traderspi = new SimpleTraderSpi(tradeapi);  tradeapi->SubscribePrivateTopic(TORA\_TERT\_RESUME);  tradeapi->SubscribePublicTopic(TORA\_TERT\_RESUME);  tradeapi->RegisterSpi(traderspi);  tradeapi->RegisterFront(g\_address); |

### 初始化MdAPI

建立**SimpleMdSpi**类继承行情响应回调基类**CTORATstpMdSpi**，并且实现其中所需的回调函数，具体实现请查看**simspi.h**文件。

|  |  |
| --- | --- |
| OnFrontConnected | 当客户端与行情后台建立起通信连接时（还未登录前），该方法被调用。 |
| OnFrontDisconnected | 当客户端与行情后台通信连接断开时，该方法被调用。 |
| OnRspUserLogin | 登录应答 |
| OnRspSubMarketData | 行情订阅应答 |
| OnRtnDepthMarketData | 行情通知推送应答，有行情更新就会推送行情。 |
| OnRspInquiryFinancialIndicator | 查询财务指标数据应答。 |

初始化MdApi实例，创建自定义的行情回调类实例，并且注册回测服务器的地址。

|  |
| --- |
| CTORATstpMdApi\* mdapi = CTORATstpMdApi::CreateTstpMdApi();  DemoMdSpi\* mdspi = new DemoMdSpi(mdapi, traderspi, submd\_sse\_securities, sse\_submd\_num, submd\_szse\_securities, szse\_submd\_num);  mdapi->RegisterSpi(mdspi);  mdapi->RegisterFront(g\_address); |

### 连续回测多个交易

如果要回测多个交易日，那么在每个交易日开始时，必需重新调用MdApi和TraderApi的init函数，去连接上回测引擎，并会产生OnFrontConnected回调函数；而当每个交易日结束时，回测引擎会断开所有的MdApi和TraderApi，会产生相应OnFrontDisconnected回调函数，而调用的MdApi和TraderApi的join函数会立即退出，且如果所有交易日都完成则join函数返回整数值0，否则返回1。

|  |
| --- |
| while (true)  { /////////////////////////////////////////////////////////////////////////////////////////////////////////////////  //每个交易日回测开始时，首先需调用mdapi和traderapi的init函数，去连接上回测引擎，  //并会产生OnFrontConnected回调函数；接着再进行登录报单等操作  //当上一个交易日结束，要回测下一个交易日之前，必需重新调用mdapi和traderapi的init函数。 //////////////////////////////////////////////////////////////////////////////////////////////////////////////////  tradeapi->Init();  mdapi->Init(); /////////////////////////////////////////////////////////////////////////////////////////////////////////////////  //当每个交易日回测结束时，回测引擎会断开所有的mdapi和traderapi，会产生相应OnFrontDisconnected回调函数；  //同时如果调用了mdapi和traderapi的join函数，则join函数会立即退出:  //并且如果还有下一个交易日需要回测则join返回整数值1，如果回测结束没有下一个交易日需要回测则join返回整数值0。  /////////////////////////////////////////////////////////////////////////////////////////////////////////////////  int ret1 = mdapi->Join();  int ret2 = tradeapi->Join();  if (1 == ret1 && 1 == ret2)  {  *printf*("Join exit because one TradingDay is finished\n");  continue;  }  else if (0 == ret1 && 0 == ret2)  {  *printf*("Join exit because the whole test is finished\n");  *printf*("please press any key to exit\n");  *getchar*();  break;  }  else  {  break;  }  } |

### 登录Login

回测程序必须同时使用BackTestApi的MdApi，TraderApi去完成登录交易引擎和行情引擎，两者都登录成功后，才能开始进行回测。

在回测TraderApi的OnFrontConnected回调函数，需调用ReqUserLogin函数发送登录请求，并且当调用TraderApi的ReqUserLogin函数去登录时，可以在登录请求报文中的TerminalInfo字段里填写本次回测的名称，还可以用登录请求报文中的UserProductInfo字段来设置回测的历史行情起始日期（traderapi.ini中的SubMD\_Begin\_date字段会失效）。在登录成功后，需要获取股东账号和资金账号等必需信息，在OnRspUserLogin回调函数中，发送查询股东账户请求，查询股东账户完成后，接着去查询资金账户请求，最后再查询持仓。查询过程必须是一次查询结束，再进行下一次查询，不能同时发送多次查询。

|  |
| --- |
| virtual void OnFrontConnected()  {  LOG("trade OnFrontConnected\n");  ReqUserLogin();  }  virtual void OnFrontDisconnected(int nReason)  {  LOG("trade OnFrontDisconnected\n");  }  ////发送登录请求  void ReqUserLogin()  {  /////连接成功后直接去登录  /////调用traderapi的ReqUserLogin函数去登录时，可以在登录请求报文中的TerminalInfo字段里填写本次回测的名称。  CTORATstpReqUserLoginField ReqUserLoginField;  *memset*(&ReqUserLoginField, 0, sizeof(ReqUserLoginField));  *strcpy*(ReqUserLoginField.LogInAccount, g\_userid);  ReqUserLoginField.LogInAccountType = TORA\_TSTP\_LACT\_UserID;  *strcpy*(ReqUserLoginField.Password, g\_passwd);  *strcpy*(ReqUserLoginField.TerminalInfo, g\_testname);  m\_api->ReqUserLogin(&ReqUserLoginField, g\_requestid++);  }  //登录请求响应  virtual void OnRspUserLogin(CTORATstpRspUserLoginField \*pRspUserLoginField, CTORATstpRspInfoField \*pRspInfo, int nRequestID, bool bIsLast)  {  LOG("OnRspUserLogin trader TradingDay=%s LogInAccount=%sErrorID=%d ErrorMsg=%s\n",  pRspUserLoginField->TradingDay, pRspUserLoginField->LogInAccount, pRspInfo->ErrorID, pRspInfo->ErrorMsg);  if (0==pRspInfo->ErrorID)  {  //登录成功后，查询股东账号  ReqQryShareholderAccount();  }  }  ///发送查询股东账户请求  void ReqQryShareholderAccount()  {  CTORATstpQryShareholderAccountField QryShareholderAccountField;  *memset*(&QryShareholderAccountField, 0, sizeof(QryShareholderAccountField));  m\_api->ReqQryShareholderAccount(&QryShareholderAccountField, g\_requestid++);  }  ///股东账号查询  virtual void OnRspQryShareholderAccount(CTORATstpShareholderAccountField \*pShareholderAccount, CTORATstpRspInfoField \*pRspInfo, int nRequestID, bool bIsLast)  {  if (pShareholderAccount)  {  if (TORA\_TSTP\_EXD\_SSE == pShareholderAccount->ExchangeID)  {  *strcpy*(g\_sseshareholderID, pShareholderAccount->ShareholderID);  }  else if (TORA\_TSTP\_EXD\_SZSE == pShareholderAccount->ExchangeID)  {  *strcpy*(g\_szseshareholderID, pShareholderAccount->ShareholderID);  }  }  if (bIsLast)  {  ////完成股东账户查询后，再查询资金账号  ReqQryTradingAccount();  }  }  ////发送查询资金账户请求  void ReqQryTradingAccount()  {  CTORATstpQryTradingAccountField QryTradingAccountField;  *memset*(&QryTradingAccountField, 0, sizeof(QryTradingAccountField));  m\_api->ReqQryTradingAccount(&QryTradingAccountField, g\_requestid++);  }  //资金账户查询  virtual void OnRspQryTradingAccount(CTORATstpTradingAccountField \*pTradingAccount, CTORATstpRspInfoField \*pRspInfo, int nRequestID, bool bIsLast)  {  if (pTradingAccount)  {  *strcpy*(g\_accountid, pTradingAccount->AccountID);  *strcpy*(g\_investorid, pTradingAccount->AccountOwner);  g\_have\_got\_tradingaccount\_shareholderID = true;  }  if (bIsLast)  {  ReqQryPosition();  }  }  ////发送查询持仓请求  void ReqQryPosition()  {  CTORATstpQryPositionField QryPositionField;  *memset*(&QryPositionField, 0, sizeof(QryPositionField));  m\_api->ReqQryPosition(&QryPositionField, g\_requestid++);  }  //请求查询投资者持仓响应  virtual void OnRspQryPosition(CTORATstpPositionField \*pPosition, CTORATstpRspInfoField \*pRspInfo, int nRequestID, bool bIsLast)  {  if (pPosition)  {  LOG("OnRspQryPosition InvestorID=%s TradingDay=%s ExchangeID=%c SecurityID=%s AvailablePosition=%d CurrentPosition=%d PrePosition=%d HistoryPos=%d HistoryPosFrozen=%d TodayBSPos=%d TodayBSFrozen=%d LastPrice=%f TotalPosCos=%f \n",  pPosition->InvestorID, pPosition->TradingDay, pPosition->ExchangeID, pPosition->SecurityID,  pPosition->AvailablePosition, pPosition->CurrentPosition, pPosition->PrePosition,  pPosition->HistoryPos, pPosition->HistoryPosFrozen, pPosition->TodayBSPos, pPosition->TodayBSFrozen,  pPosition->LastPrice, pPosition->TotalPosCost);  }  } |

由于回测交易是由行情来驱动的，因此MdApi在登录成功的OnRspUserLogin回调必须立刻订阅行情。

|  |
| --- |
| virtual void OnFrontConnected()  {  LOG("MD OnFrontConnected\n");  ////连接上后去登录  ReqUserLogin();  }  virtual void OnFrontDisconnected(int nReason)  {  LOG("MD OnFrontDisconnected\n");  }  ////发送登录请求  void ReqUserLogin()  {  CTORATstpReqUserLoginField ReqUserLoginField;  *memset*(&ReqUserLoginField, 0, sizeof(ReqUserLoginField));  *strcpy*(ReqUserLoginField.LogInAccount, g\_userid);  ReqUserLoginField.LogInAccountType = TORA\_TSTP\_LACT\_UserID;  *strcpy*(ReqUserLoginField.Password, g\_passwd);  m\_api->ReqUserLogin(&ReqUserLoginField, g\_requestid++);  }  ///登录请求响应  virtual void OnRspUserLogin(CTORATstpRspUserLoginField \*pRspUserLogin, CTORATstpRspInfoField \*pRspInfo, int nRequestID, bool bIsLast)  {  LOG("OnRspUserLogin MD LogInAccount=%s ErrorID=%d ErrorMsg=%s\n",  pRspUserLogin->LogInAccount, pRspInfo->ErrorID, pRspInfo->ErrorMsg);  if (pRspInfo->ErrorID == 0)  {  ////登录成功后直接订阅行情  SubscribeMarketData();  }  }  ///发送订阅行情请求  void SubscribeMarketData()  {  if (m\_sse\_submd\_num>=1)  {  m\_api->SubscribeMarketData(m\_submd\_sse\_securities, m\_sse\_submd\_num, TORA\_TSTP\_EXD\_SSE);  }  if (m\_szse\_submd\_num >= 1)  {  m\_api->SubscribeMarketData(m\_submd\_szse\_securities, m\_szse\_submd\_num, TORA\_TSTP\_EXD\_SZSE);  }  }  ///订阅行情应答  virtual void OnRspSubMarketData(CTORATstpSpecificSecurityField \*pSpecificSecurity, CTORATstpRspInfoField \*pRspInfo, int nRequestID, bool bIsLast)  {  LOG("OnRspSubMarketData ErrorID=%d ErrorMsg=%s\n", pRspInfo->ErrorID, pRspInfo->ErrorMsg);  } |

### 报单申报

由于回测交易是由历史行情来驱动的，因此建议务必在继承实现**CTORATstpMdSpi::OnRtnDepthMarketData**回调函数中发送报单请求。

|  |
| --- |
| virtual void OnRtnDepthMarketData(CTORATstpMarketDataField \*pDepthMarketData)  {  if (!g\_have\_got\_tradingaccount\_shareholderID)  {  /////没有获取到股东账户和资金账号之前，不能报单  return;  }  if (*strcmp*(pDepthMarketData->UpdateTime, "09:30:00") >= 0  && pDepthMarketData->AskPrice1 > 0.000001 && pDepthMarketData->AskVolume1 >= 1)  { ////////////////////////////////////////////////////////////////////////////////////////////////////  //////如果接收到每个交易日的9:30:00以后的第一笔行情，立即以卖一价格买入。  ////在本例中，每个交易日，单个股票就进行了一次买入操作  ////判断此股票当前交易日是否已经买入过  if (!IsHaveInsertOrder(pDepthMarketData->TradingDay, pDepthMarketData->SecurityID))  {  ////申报买入股票数量必须是100股的整数倍  int VolumeTotalOriginal = 100;  if (pDepthMarketData->AskVolume1>=100)  {  VolumeTotalOriginal=*std*::*min*<int>(10000, (100 \* (pDepthMarketData->AskVolume1 / 100)));  }  ////以卖一价格限价报单买入指定股票  m\_traderspi->ReqOrderInsert\_Limit(pDepthMarketData->SecurityID, pDepthMarketData->ExchangeID,  pDepthMarketData->AskPrice1, VolumeTotalOriginal, TORA\_TSTP\_D\_Buy);  ////设置此股票在当前交易日已经买入  SetHaveInsertOrder(pDepthMarketData->TradingDay, pDepthMarketData->SecurityID);  }  } |

### 如何运行回测实例

使用回测时，mdapi.dll，traderapi.dll和traderapi.ini配置文件，都必须和执行程序demo.exe放在同一目录下。

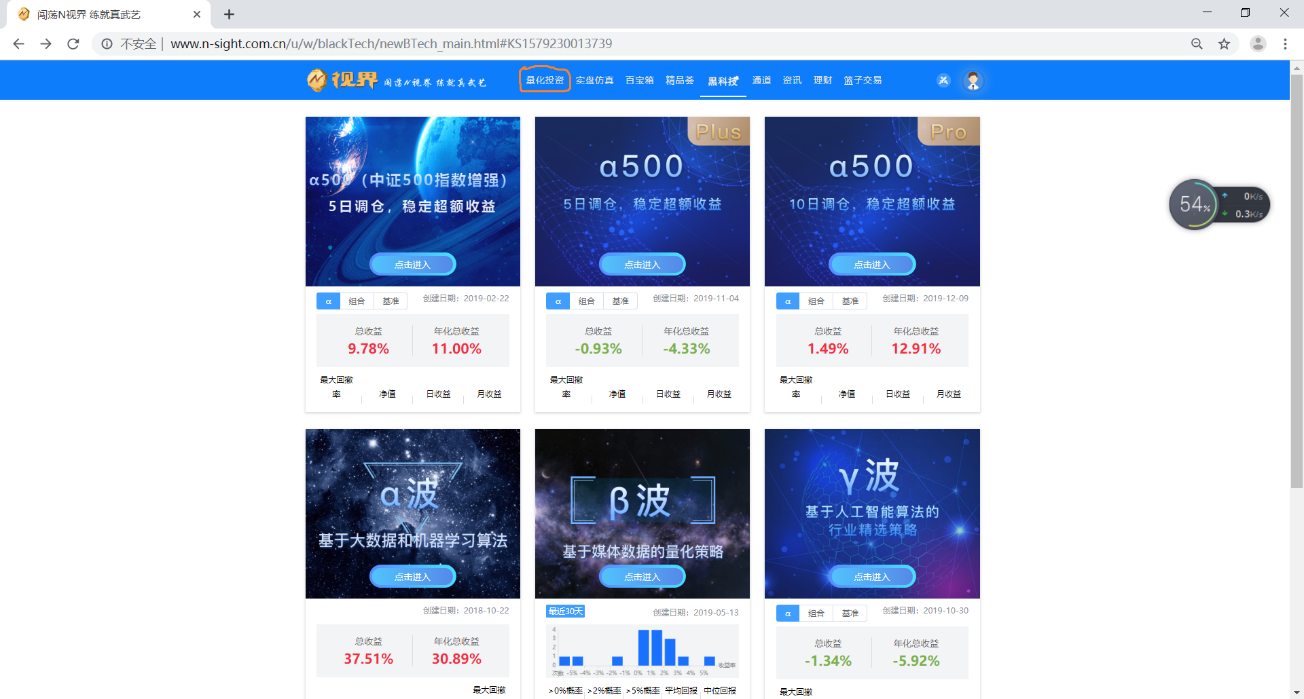
启动命令如下:

demo.exe tcp://210.14.72.18:26666 user passwd

### 查询回测结果

运行完demo回测程序，可以通过登录N视界网站<http://www.n-sight.com.cn>，来查询本次回测结果。如下面五张图，来演示如何查询回测结果。

1. 登录N视界网站成功后，在主页上点击“量化投资“按钮。



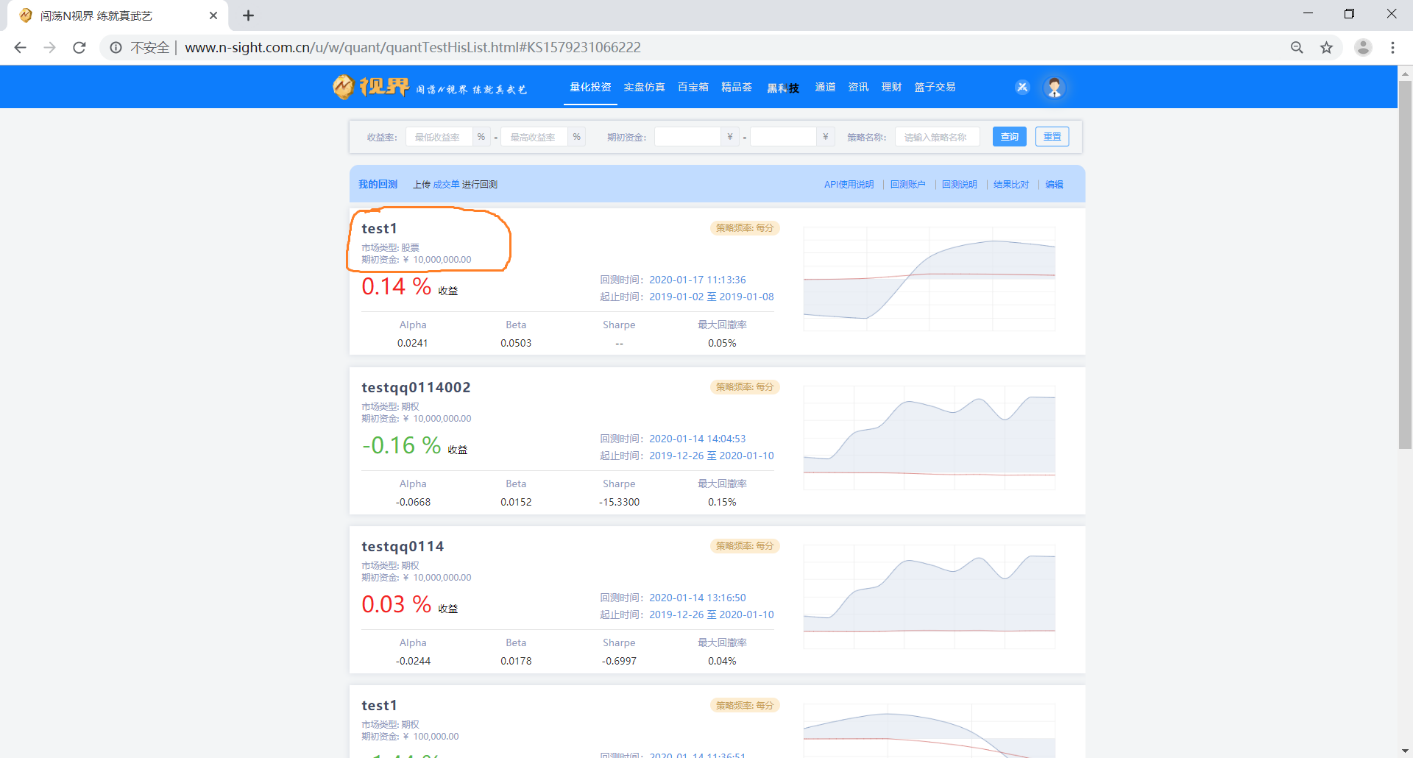
1. 点击“编程式策略“按钮。

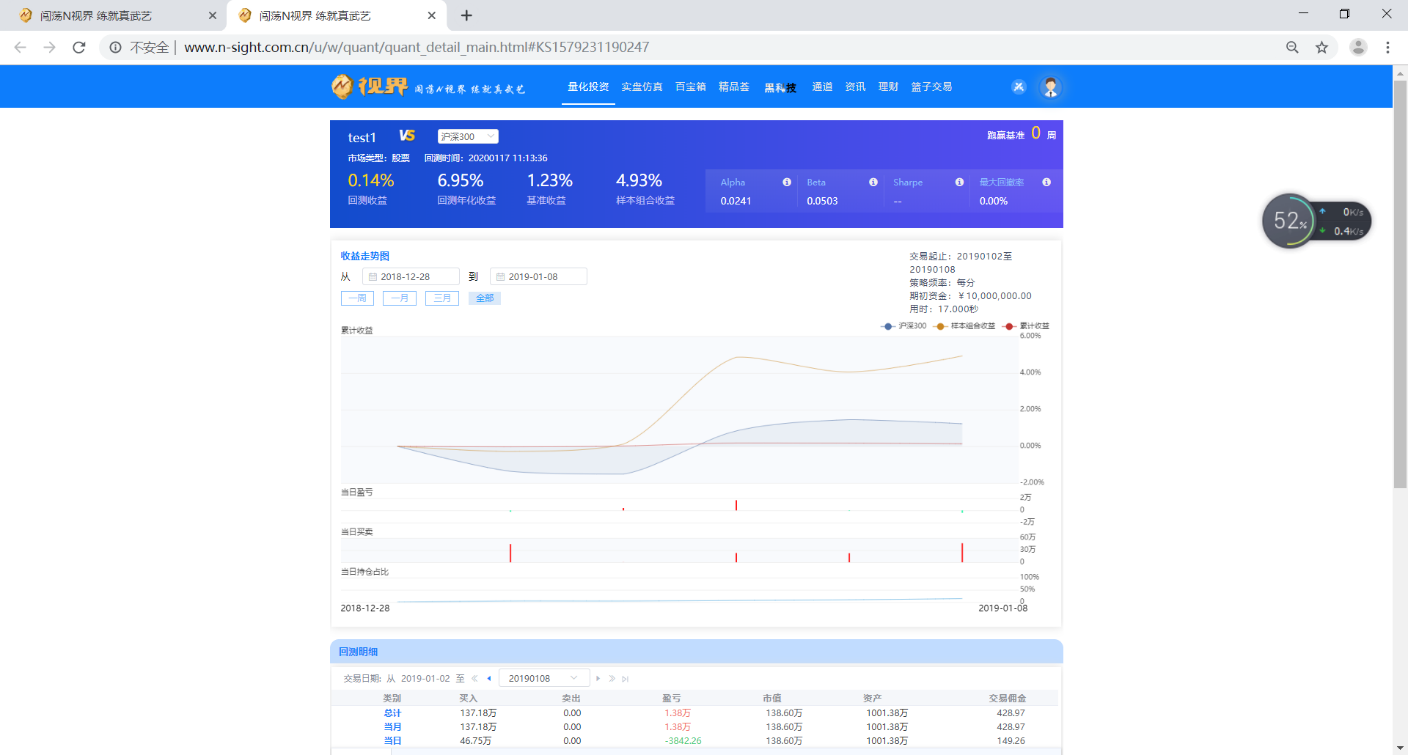


1. 点击“进入我的回测“按钮。



1. 点击某次回测来查看回测结果的详细信息。





# 如何快速开始回测（Python）

本节内容将演示用BackTestApi的python接口来重新编写上一节的C++ 示例。

示例下载（BackTestAPI\_demo\_python.zip），可以通过登录N视界网站<http://www.n-sight.com.cn>，并访问百宝箱页面<http://www.n-sight.com.cn/u/w/download/download.html>，然后点击BackTestApi的Python Demo按钮去下载（如下图所示）。



## 开发环境准备

* 需要安装Python3.6版本
* 安装Python开发IDE，推荐PyCharm

## 如何运行回测程序

traderapi.ini配置文件与C++示例的配置文件相同，查看回测结果方式也与C++示例相同。

使用回测时，traderapi.dll，mdapi.dll，traderapi.py，mdapi.py，\_traderapi.pyd，\_mdapi.pyd和traderapi.ini配置文件，都必须和执行程序demo.py放在同一目录下。demo.py是回测程序的源代码。

启动命令如下:

python demo.py tcp://210.14.72.18:26666 user passwd

# 回测工具支持哪些证券品种

目前回测工具支持上证深证的A股的竞价交易业务，支持上证和深证所有的竞价交易报单类型：

|  |
| --- |
| 限价单 |
| 五档即时成交剩余撤销 |
| 五档即时成交剩余转限价单(上证) |
| 本方最优(深证) |
| 对手方最优剩余转限价(深证) |
| 市价立即成交剩余撤销(深证) |
| 市价全额成交或撤销(深证) |

# 回测如何撮合成交

在回测过程中，报单撮合成交是由行情来驱动。根据行情对手方的五档价格来对价成交，按照一档到五档价格的顺序来依次成交，并且每次成交的数量都会消耗掉该行情档位价格的挂单剩余量。

* 买入时：如果委托价大于等于行情某档卖价格，则成交，成交价为此档位价格，如果委托价小于此档位价格，不能成交，等待更优的行情才能成交；
* 卖出时：如果委托价小于等于行情某档买价格，则成交，成交价为此档位价格，如果委托价大于此档位价格，不能成交，等待更优的行情才能成交。

# BackTestApi如何使用详解

## 交易接口和行情接口的初始化

编写回测程序时，必须同时初始化交易接口traderapi和行情接口mdapi，不能单独只使用其中一个。并且使用traderapi的RegisterFront函数来注册回测服务器地址（**tcp://210.14.72.18:26666**）去连接上回测服务器。

当回测多个交易日，那么在每个交易日开始时，必需重新调用mdapi和traderapi的init函数；而当每个交易日结束时，调用的mdapi和traderapi的join函数会立即退出，且如果所有交易日都完成则join函数返回整数值0，否则返回1。

|  |
| --- |
| CTORATstpTraderApi\* tradeapi = CTORATstpTraderApi::CreateTstpTraderApi();  SimpleTraderSpi\* traderspi = new SimpleTraderSpi(tradeapi);  tradeapi->SubscribePrivateTopic(TORA\_TERT\_RESUME);  tradeapi->SubscribePublicTopic(TORA\_TERT\_RESUME);  tradeapi->RegisterSpi(traderspi);  tradeapi->RegisterFront(g\_address);  CTORATstpMdApi\* mdapi = CTORATstpMdApi::CreateTstpMdApi();  SimpleMdSpi\* mdspi = new SimpleMdSpi(mdapi, tradeapi);  mdapi->RegisterSpi(mdspi);  while (true)  {  tradeapi->Init();  mdapi->Init();  int ret1 = mdapi->Join();  int ret2 = tradeapi->Join();  if (1 == ret1 && 1 == ret2)  {  continue;  }  else if (0 == ret1 && 0 == ret2)  {  break;  }  else  {  break;  }  } |

## 登录&订阅行情

* 调用traderapi的init函数之后，交易连接立即建立后对应交易的回调函数OnFrontConnected会被调用；而在每个交易日回测结束时，交易连接会断开，则对应交易的回调函数OnFrontDisconnected会被调用。

当交易连接建立时，在对应交易的回调函数OnFrontConnected中，即可使用函数 ReqUserLogin 请求登录证券柜台交易系统。发送登录请求之后，traderapi会接收到登录请求响应，对应交易的回调函数OnRspUserLogin会被调用，OnRspUserLogin的第二个参数pRspInfo应答信息的ErrorID值若为0，则表示登录成功，否则登录失败。

|  |
| --- |
| class SimpleTraderSpi : public CTORATstpTraderSpi  {  virtual void OnFrontConnected()  {  *printf*("trade OnFrontConnected\n");  /////连接成功后直接去登录  CTORATstpReqUserLoginField ReqUserLoginField;  *memset*(&ReqUserLoginField, 0, sizeof(ReqUserLoginField));  *strcpy*(ReqUserLoginField.LogInAccount, g\_userid);  ReqUserLoginField.LogInAccountType = TORA\_TSTP\_LACT\_UserID;  *strcpy*(ReqUserLoginField.Password, g\_passwd);  *strcpy*(ReqUserLoginField.TerminalInfo, g\_testname);  m\_api->ReqUserLogin(&ReqUserLoginField, g\_requestid++);  }  //登录请求响应  virtual void OnRspUserLogin(CTORATstpRspUserLoginField \*pRspUserLoginField, CTORATstpRspInfoField \*pRspInfo, int nRequestID, bool bIsLast)  {  if (pRspInfo->ErrorID == 0)  {  //登录成功后，查询股东账号  CTORATstpQryShareholderAccountField QryShareholderAccountField;  *memset*(&QryShareholderAccountField, 0, sizeof(QryShareholderAccountField));  m\_api->ReqQryShareholderAccount(&QryShareholderAccountField, g\_requestid++);  }  }  }; |

* 调用mdapi的init函数之后，行情连接立即建立后对应行情的回调函数OnFrontConnected会被调用；而在每个交易日回测结束时，行情连接也会断开，则对应行情的回调函数OnFrontDisconnected会被调用。

当行情连接建立时，在对应行情的回调函数OnFrontConnected中，即可使用函数 ReqUserLogin 请求登录证券柜台交易系统。发送登录请求之后，mdapi会接收到登录请求响应，对应行情的回调函数OnRspUserLogin会被调用，OnRspUserLogin的第二个参数pRspInfo应答信息的ErrorID值若为0，则表示登录成功，否则登录失败。在行情登录成功之后，在OnRspUserLogin回调函数中即可使用SubscribeMarketData函数去订阅所关注的证券行情。订阅行情成功后，OnRspSubMarketData回调函数会被调用。

|  |
| --- |
| class SimpleMdSpi : public CTORATstpMdSpi  {  virtual void OnFrontConnected()  {  *printf*("MD OnFrontConnected\n");  ////连接上后去登录  CTORATstpReqUserLoginField ReqUserLoginField;  *memset*(&ReqUserLoginField, 0, sizeof(ReqUserLoginField));  *strcpy*(ReqUserLoginField.LogInAccount, g\_userid);  ReqUserLoginField.LogInAccountType = TORA\_TSTP\_LACT\_UserID;  *strcpy*(ReqUserLoginField.Password, g\_passwd);  m\_api->ReqUserLogin(&ReqUserLoginField, g\_requestid++);  }  ///登录请求响应  virtual void OnRspUserLogin(CTORATstpRspUserLoginField \*pRspUserLogin, CTORATstpRspInfoField \*pRspInfo, int nRequestID, bool bIsLast)  {  if (pRspInfo->ErrorID == 0)  {  ////登录成功后直接订阅行情  char\*\* sse\_securities = new char\*[1];  sse\_securities[0] = new char[64];  *memset*(sse\_securities[0], 0, 64);  *strcpy*(sse\_securities[0], "600000");  char\*\* szse\_securities = new char\*[1];  szse\_securities[0] = new char[64];  *memset*(szse\_securities[0], 0, 64);  *strcpy*(szse\_securities[0], "000002");  m\_api->SubscribeMarketData(sse\_securities, 1, TORA\_TSTP\_EXD\_SSE);  m\_api->SubscribeMarketData(szse\_securities, 1, TORA\_TSTP\_EXD\_SZSE);  }  }  ///订阅行情应答  virtual void OnRspSubMarketData(CTORATstpSpecificSecurityField \*pSpecificSecurity, CTORATstpRspInfoField \*pRspInfo, int nRequestID, bool bIsLast)  {  *printf*("OnRspSubMarketData ErrorID=%d ErrorMsg=%s\n", pRspInfo->ErrorID, pRspInfo->ErrorMsg);  }  }; |

## 行情推送

当行情推送给BackTestApi的mdapi，行情对应的回调函数OnRtnDepthMarketData会被调用。由于回测交易是由行情来驱动的，因此建议务必在OnRtnDepthMarketData回调函数中去发送报单请求。

## 报单

### 报单录入

录入报单委托使用的函数是 ReqOrderInsert ，其中的核心数据结构是CTORATstpInputOrderField，通用的报单录入代码如下：

|  |
| --- |
| CTORATstpInputOrderField inputorder;  *memset*(&inputorder, 0, sizeof(inputorder));  *strcpy*(inputorder.UserID, UserID);  *strcpy*(inputorder.InvestorID, InvestorID);  *strcpy*(inputorder.AccountID, AccountId);  *strcpy*(inputorder.ShareholderID, ShareholderID);  inputorder.ExchangeID = TORA\_TSTP\_EXD\_SSE;  *strcpy*(inputorder.SecurityID, "600000");  inputorder.Direction = TORA\_TSTP\_D\_Buy;  inputorder.LimitPrice = 10.0;  inputorder.VolumeTotalOriginal = 1000;  inputorder.OrderPriceType = TORA\_TSTP\_OPT\_LimitPrice;  inputorder.TimeCondition = TORA\_TSTP\_TC\_GFD;  inputorder.VolumeCondition = TORA\_TSTP\_VC\_AV;  inputorder.MinVolume = 0;  inputorder.CombOffsetFlag[0] = TORA\_TSTP\_OF\_Open;  inputorder.CombHedgeFlag[0] = TORA\_TSTP\_HF\_Speculation;  inputorder.ForceCloseReason = TORA\_TSTP\_FCC\_NotForceClose;  inputorder.Operway = TORA\_TSTP\_OPERW\_PCClient;  tradeapi->ReqOrderInsert(&inputorder, 1); |

OrderPriceType，TimeCondition和VolumeCondition三个字段决定报单类型:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **交易所** | **OrderPriceType** | **TimeCondition** | **VolumeCondition** | **说明** |
| 上交所 | TORA\_TSTP\_OPT\_LimitPrice | TORA\_TSTP\_TC\_GFD | TORA\_TSTP\_VC\_AV | 限价 |
| 上交所 | TORA\_TSTP\_OPT\_FiveLevelPrice | TORA\_TSTP\_TC\_IOC | TORA\_TSTP\_VC\_AV | 最优五档成交剩余转撤销 |
| 上交所 | TORA\_TSTP\_OPT\_FiveLevelPrice | TORA\_TSTP\_TC\_GFD | TORA\_TSTP\_VC\_AV | 最优五档成交剩余转限价 |
| 上交所（科创板） | TORA\_TSTP\_OPT\_HomeBestPrice | TORA\_TSTP\_TC\_GFD | TORA\_TSTP\_VC\_AV | 本方最优 |
| 上交所（科创板） | TORA\_TSTP\_OPT\_BestPrice | TORA\_TSTP\_TC\_GFD | TORA\_TSTP\_VC\_AV | 对手方最优 |
| 上交所（科创板） | TORA\_TSTP\_OPT\_FixPrice | TORA\_TSTP\_TC\_GFD | TORA\_TSTP\_VC\_AV | 盘后定价交易 |
| 深交所 | TORA\_TSTP\_OPT\_LimitPrice | TORA\_TSTP\_TC\_GFD | TORA\_TSTP\_VC\_AV | 限价 |
| 深交所 | TORA\_TSTP\_OPT\_BestPrice | TORA\_TSTP\_TC\_GFD | TORA\_TSTP\_VC\_AV | 对手方最优 |
| 深交所 | TORA\_TSTP\_OPT\_HomeBestPrice | TORA\_TSTP\_TC\_GFD | TORA\_TSTP\_VC\_AV | 本方最优 |
| 深交所 | TORA\_TSTP\_OPT\_AnyPrice | TORA\_TSTP\_TC\_IOC | TORA\_TSTP\_VC\_AV | 立即成交剩余撤销 |
| 深交所 | TORA\_TSTP\_OPT\_FiveLevelPrice | TORA\_TSTP\_TC\_IOC | TORA\_TSTP\_VC\_AV | 最优五档成交剩余转撤销 |
| 深交所 | TORA\_TSTP\_OPT\_AnyPrice | TORA\_TSTP\_TC\_IOC | TORA\_TSTP\_VC\_CV | 全额成交或撤销 |
| 深港通 | TORA\_TSTP\_OPT\_LimitPrice | TORA\_TSTP\_TC\_GFA | TORA\_TSTP\_VC\_AV | 竞价限价盘 |
| 深港通 | TORA\_TSTP\_OPT\_LimitPrice | TORA\_TSTP\_TC\_GFD | TORA\_TSTP\_VC\_AV | 增强限价盘 |

### 报单录入响应

报单录入之后，证券柜台系统会返回报单录入响应，OnRspOrderInsert回调函数会被调用：若报单成功录入证券柜台系统，则OnRspOrderInsert的第二个参数pRspInfo响应信息的ErrorID值为0；否则ErrorID为非0值。

|  |
| --- |
| //报单录入请求响应  virtual void OnRspOrderInsert(CTORATstpInputOrderField \*pInputOrderField, CTORATstpRspInfoField \*pRspInfo, int nRequestID, bool bIsLast)  {  if (pRspInfo->ErrorID==0)  {  *printf*("ReqOrderInsert ok\n");  }  else  {  *printf*("ReqOrderInsert fail\n");  }  } |

报单录入之后，如果报单直接被证券柜台系统拒绝，则OnErrRtnOrderInsert回调函数会被调用，而且同时OnRspOrderInsert回调函数也会被调用且pRspInfo的ErrorID为非0值。

|  |
| --- |
| virtual void OnErrRtnOrderInsert(CTORATstpInputOrderField \*pInputOrder, CTORATstpRspInfoField \*pRspInfo)  {  *printf*("ReqOrderInsert fail\n");  } |

### 报单状态推送

每次报单状态发生变化，证券柜台系统都会通知traderapi，OnRtnOrder回调函数会被调用。一次报单过程中OnRtnOrder会被调用数次：报单成功录入证券柜台系统时，交易所撤销或接受该报单时，该报单成交时，报单被交易所拒绝时。报单回报主要作用是通知traderapi该报单的最新状态，如全部成交，已撤销，部分成交，未成交等。每次报单状态有变化，该函数都会被调用一次。如果报单是分笔成交，则每次成交都会有一次 OnRtnOrder 返回。

OnRtnOrder回调函数只有一个参数Order报单信息，参数的数据结构为CTORATstpOrderField，Order报单信息的OrderStatus字段取值如下：

|  |  |
| --- | --- |
| **OrderStatus** | 含义 |
| 0 | 全部成交 |
| 1 | 部分成交 |
| 3 | 未成交，报单还在交易所撮合队列中 |
| 5 | 已撤销 |
| a | 未知：报单已提交交易所，但未从交易所收到确认信息 |
| d | 预埋：报单成功录入证券柜台系统，但是还未发往交易所 |

|  |
| --- |
| //报单回报  virtual void OnRtnOrder(CTORATstpOrderField \*pOrder)  {  if (TORA\_TSTP\_OST\_AllTraded == pOrder->OrderStatus)  {  printf("报单全部成交\n");  }  else if (TORA\_TSTP\_OST\_PartTradedQueueing == pOrder->OrderStatus)  {  printf("报单部分成交\n");  }  else if (TORA\_TSTP\_OST\_NoTradeQueueing == pOrder->OrderStatus)  {  printf("报单未成交\n");  }  else if (TORA\_TSTP\_OST\_Canceled == pOrder->OrderStatus)  {  printf("报单已撤单\n");  }  else if (TORA\_TSTP\_OST\_Unknown == pOrder->OrderStatus)  {  printf("报单状态未知:报单已提交交易所，但未从交易所收到确认信息\n");  }  else if (TORA\_TSTP\_OST\_Cached == pOrder->OrderStatus)  {  printf("报单已预埋\n");  }  } |

### 成交回报

如果一笔报单成交之后，一个报单回报（OnRtnOrder）和一个成交回报（OnRtnTrade）会被发送到traderapi，OnRtnOrder报单回报中OrderStatus会为“已成交”或“部分成交”。

|  |
| --- |
| //成交回报  virtual void OnRtnTrade(CTORATstpTradeField \*pTrade)  {  *printf*("OnRtnTrade TradeID=%s Direction=%c Volume=%d Price=%f OrderSysID=%s\n",  pTrade->TradeID, pTrade->Direction, pTrade->Volume, pTrade->Price, pTrade->OrderSysID);  } |

### 报单序列号

在证券柜台交易系统中，每笔报单都有 3 组唯一序列号，保证其与其他报单是不重复的。

* **FrontID + SessionID + OrderRef**
* **ExchangeID + PBUID+ OrderLocalID**
* **ExchangeID + OrderSysID**

### 撤单

撤单使用的函数是 ReqOrderAction，核心的数据结构是 CTORATstpInputOrderActionField。撤单操作与报单操作很类似，但需要注意以下几个地方。

* 字段 **ActionFlag：**由于国内的交易所目前只支持撤单，不支持改单操作，因此 ActionFlag的赋值目前只能是 TORA\_TSTP\_AF\_Delete。
* 字段 **LimitPrice:** LimitPrice字段必须和被撤报单的申报价格相同。
* 指定被撤报单的序列号：可用(**FrontID + SessionID + OrderRef**)或者(**ExchangeID + OrderSysID**)方式来指定被撤报单。
* 如果报单成功被撤销，OnRspOrderAction回调函数会被调用，且第二个参数pRspInfo的ErrorID值为0；还会收到OnRtnOrder报单回报报单状态为“撤单“。
* 如果报单撤单操作被拒绝，OnRspOrderAction回调函数也会被调用，但第二个参数pRspInfo的ErrorID值为非0值；还会收到OnErrRtnOrderAction撤单错误回报。

|  |
| --- |
| CTORATstpInputOrderActionField action\_order;  *memset*(&action\_order, 0, sizeof(action\_order));  action\_order.ExchangeID = TORA\_TSTP\_EXD\_SSE;  *strcpy*(action\_order.SecurityID, "600000");  action\_order.ActionFlag = TORA\_TSTP\_AF\_Delete;  action\_order.LimitPrice = LimitPrice;  action\_order.FrontID = FrontID;  action\_order.SessionID = SessionID;  *strcpy*(action\_order.InvestorID, InvestorID);  *strcpy*(action\_order.OrderRef, OrderRef);  *strcpy*(action\_order.IPAddress, IPAddress);  *strcpy*(action\_order.MacAddress, MacAddress);  *strcpy*(action\_order.OrderSysID, OrderSysID);  TraderApi->ReqOrderAction(&action\_order, 1); |

# 回测BackTestApi支持生产API的哪些接口

虽然BACKTESTAPI和奇点生产API的接口相同，但是BACKTESTAPI只支持部分的奇点生产API接口，其他未支持的接口BACKTESTAPI内部实现为空。

## 支持生产mdapi哪些接口

BackTestApi的mdapi.dll和生产行情mdapi的接口完全相同，但只支持部分生产行情mdapi的接口。BackTestApi支持的mdapi的接口如下：

CTORATstpMdApi类支持如下请求接口，其他未支持的接口内部实现为空。

|  |
| --- |
| CreateTstpMdApi |
| GetApiVersion |
| RegisterFront |
| Release |
| Init |
| Join |
| RegisterSpi |
| SubscribeMarketData |
| UnSubscribeMarketData |
| SubscribeFundsFlowMarketData |
| UnSubscribeFundsFlowMarketData |
| ReqUserLogin |
| ReqUserLogout |
| ReqQryRightsAdjustmentInfo |
| ReqQryHistoryFundsFlowInfo |
| ReqQryFinancialIndicatorInfo |
| ReqQryDividendInfo |
| ReqQryRightIssueInfo |
| ReqQryCompanyDescriptionInfo |
| ReqQrySalesSegmentInfo |
| ReqQryEquityStructureInfo |
| ReqQryTopTenHoldersInfo |
| ReqQryTopTenFloatHoldersInfo |
| ReqQryIndustryInfo |
| ReqQryConceptionInfo |
| ReqQryRegionInfo |
| ReqQryIndexDescriptionInfo |
| ReqQryIndustryConstituentsInfo |
| ReqQryConceptionConstituentsInfo |
| ReqQryRegionConstituentsInfo |
| ReqQryIndustryCodeList |
| ReqQryConceptionCodeList |
| ReqQryRegionCodeList |
| ReqQryFreeFloatSharesInfo |

CTORATstpMdSpi类支持如下回调接口，其他未支持的回调接口不会被调用。

|  |
| --- |
| OnFrontConnected |
| OnFrontDisconnected |
| OnRspUserLogin |
| OnRspUserLogout |
| OnRspSubMarketData |
| OnRspUnSubMarketData |
| OnRspSubFundsFlowMarketData |
| OnRspUnSubFundsFlowMarketData |
| OnRtnDepthMarketData |
| OnRtnEffectPriceMarketData |
| OnRtnEffectVolumeMarketData |
| OnRtnFundsFlowMarketData |
| OnRspInquiryRightsAdjustment |
| OnRspInquiryHistoryFundsFlow |
| OnRspInquiryFinancialIndicator |
| OnRspInquiryDividend |
| OnRspInquiryRightIssue |
| OnRspInquiryCompanyDescription |
| OnRspInquirySalesSegment |
| OnRspInquiryEquityStructure |
| OnRspInquiryTopTenHolders |
| OnRspInquiryTopTenFloatHolders |
| OnRspInquiryIndustry |
| OnRspInquiryConception |
| OnRspInquiryRegion |
| OnRspInquiryIndexDescription |
| OnRspInquiryIndustryConstituents |
| OnRspInquiryConceptionConstituents |
| OnRspInquiryRegionConstituents |
| OnRspInquiryIndustryCodeList |
| OnRspInquiryConceptionCodeList |
| OnRspInquiryRegionCodeList |
| OnRspInquiryFreeFloatShares |

## 支持生产traderapi哪些接口

BackTestApi的traderapi.dll和生产交易traderapi的接口完全相同，但只支持部分生产traderapi的接口。BackTestApi支持的traderapi的接口如下：

CTORATstpTraderApi类支持如下请求接口，其他未支持的接口内部实现为空。

|  |
| --- |
| CreateTstpTraderApi |
| GetApiVersion |
| Release |
| RegisterFront 用户可使用RegisterFront函数来注册回测服务器地址去连接上该服务器 |
| Init |
| Join |
| RegisterSpi |
| SubscribePrivateTopic支持所有的三种私有流重传方式  TORA\_TERT\_RESTART:从本交易日开始重传  TORA\_TERT\_RESUME:从上次收到的续传  TORA\_TERT\_QUICK:只传送登录后私有流的内容 |
| ReqUserLogin 使用用户名和密码通过ReqUserLogin函数去登录上回测服务器 |
| ReqUserLogout |
| ReqOrderInsert |
| ReqOrderAction |
| ReqQryInvestorTradingFee |
| ReqQryBUProxy |
| ReqQryTradingAccount |
| ReqQryOrderAction |
| ReqQryUser |
| ReqQryInvestor |
| ReqQryTradingFee |
| ReqQryPosition |
| ReqQrySecurity |
| ReqQryOrder |
| ReqQryTrade |
| ReqQryShareholderAccount |
| ReqQryMarketData |
| ReqQryOrderFundDetail |
| ReqInputRemarkEvent |
| ReqInquiryHistoryRemarkEvent |
| ReqUpdateRemarkEvent |
| ReqDeleteRemarkEvent |

CTORATstpTraderSpi类支持如下回调接口，其他未支持的回调接口不会被调用。

|  |
| --- |
| OnFrontConnected |
| OnFrontDisconnected |
| OnRspUserLogin |
| OnRspUserLogout |
| OnRspOrderInsert |
| OnRtnOrder |
| OnErrRtnOrderInsert |
| OnRspOrderAction |
| OnErrRtnOrderAction |
| OnRtnTrade |
| OnRspQryInvestorTradingFee |
| OnRspQryBUProxy |
| OnRspQryTradingAccount |
| OnRspQryOrderAction |
| OnRspQryUser |
| OnRspQryInvestor |
| OnRspQryTradingFee |
| OnRspQryPosition |
| OnRspQrySecurity |
| OnRspQryOrder |
| OnRspQryTrade |
| OnRspQryShareholderAccount |
| OnRspQryMarketData |
| OnRspQryOrderFundDetail |
| OnRspInputRemarkEvent |
| OnRspInquiryHistoryRemarkEvent |
| OnRspUpdateRemarkEvent |
| OnRspDeleteRemarkEvent |

# 回测BackTestApi支持查询股票基本面数据

在生产mdapi中有一批查询股票基本面数据的接口，比如查询十大股东数据信息” ReqQryTopTenHoldersInfo”，查询财务指标信息“ReqQryFinancialIndicatorInfo”等接口。回测BACKTESTAPI也同样支持这些查询基本面数据的接口，并且会根据回测交易日的不同而查询到基于当前回测交易日的最新数据结果，例如运行20180103到20181228的回测策略，在回测交易日20180103时候去查询某个股票的财务指标信息是基于20180103的最新财务指标信息，而在回测交易日20181228时候去查询财务指标信息是基于20181228的最新财务指标信息，两个不同回测交易日查询到结果是不一样的。

BackTestApi支持mdapi中查询基本面数据接口如下：

BackTestApi支持CTORATstpTraderApi类中查询基本面数据的请求接口：

|  |
| --- |
| ReqQryRightsAdjustmentInfo |
| ReqQryHistoryFundsFlowInfo |
| ReqQryFinancialIndicatorInfo |
| ReqQryDividendInfo |
| ReqQryRightIssueInfo |
| ReqQryCompanyDescriptionInfo |
| ReqQrySalesSegmentInfo |
| ReqQryEquityStructureInfo |
| ReqQryTopTenHoldersInfo |
| ReqQryTopTenFloatHoldersInfo |
| ReqQryIndustryInfo |
| ReqQryConceptionInfo |
| ReqQryRegionInfo |
| ReqQryIndexDescriptionInfo |
| ReqQryIndustryConstituentsInfo |
| ReqQryConceptionConstituentsInfo |
| ReqQryFreeFloatSharesInfo |
| ReqQryRegionConstituentsInfo |
| ReqQryIndustryCodeList |
| ReqQryConceptionCodeList |
| ReqQryRegionCodeList |

BackTestApi支持CTORATstpMdSpi类中查询基本面数据的回调接口：

|  |
| --- |
| OnRspInquiryRightsAdjustment |
| OnRspInquiryHistoryFundsFlow |
| OnRspInquiryFinancialIndicator |
| OnRspInquiryDividend |
| OnRspInquiryRightIssue |
| OnRspInquiryCompanyDescription |
| OnRspInquirySalesSegment |
| OnRspInquiryEquityStructure |
| OnRspInquiryTopTenHolders |
| OnRspInquiryTopTenFloatHolders |
| OnRspInquiryIndustry |
| OnRspInquiryConception |
| OnRspInquiryRegion |
| OnRspInquiryIndexDescription |
| OnRspInquiryIndustryConstituents |
| OnRspInquiryConceptionConstituents |
| OnRspInquiryFreeFloatShares |
| OnRspInquiryRegionConstituents |
| OnRspInquiryIndustryCodeList |
| OnRspInquiryConceptionCodeList |
| OnRspInquiryRegionCodeList |