国泰君安期货 CTP 接口开发指南

一 CTP 介绍

背景

✧ 国内程序化发展

- 自从2010年4月推出股指期货之后，大量的程序化套利策略纷纷出炉并创造出惊人的交易量
- 据市场人士估计，股指期货上市交易三年来，程序化交易在股指期货交易中所占比重由2%提升至8%，去年则超过20%；从收益情况看，2010年程序化交易收益普遍在20%以上，资金容量也很大
- 据数据统计显示，借助程序化交易系统提示交易信号进行手动交易或进行半自动交易的投资者数量占投资者总量的5%—10%，而通过程序化交易完成的交易量占市场总交易量的20%—30%。
- 国泰君安期货提供一流的程序化交易服务，为您的程序化交易武装到牙齿，我们同时也提供交易策略供用户选择，国泰君安期货CTP主席平台部署在上期技术张江机房，通过千兆局域网接入中金所和上期所 交易系统。2013年8月16日，国泰君安期货CTP主席系统上线。

综合交易平台

- 做为一个开放、快速、稳定、安全的期货交易、结算系统解决方案，在期货界也获得了越来越普遍的认同。综合交易平台开放的接口、优异的性能、集中部署的创新模式以及经验丰富的技术背景都为程序化交易在国内的快速发展提供了最为优异的平台。综合交易平台现有的程序化交易客户对综合交易平台的解决方案给了很高的评价，其交易量也不断攀升。
- 综合交易平台8000笔/秒处理速度的交易引擎，整套系统在0.5毫秒以内处理完成报单、成交全过程的资金持仓计算的能力，无单点故障并实现负载均衡的交易系统体系架构构成了综合交易平台的高性能体验。
- 综合交易平台目前的系统配置拥有2万个客户同时在线的处理能力，还可以通过扩展前置机群进一步提升系统对更多客户在线的处理能力。
- 综合交易平台通过千兆局域网接入中金所和上期所交易系统，通过三所联网主干接入大商所和郑商所。 投资者在综合交易平台的报单直接进入综合交易平台的前置机，经过交易后台高速的资金持仓计算后再经局域网报到中金所和上期所，通过三所联网主干报到大商所和郑商所。行情服务器直连交易所并在同一个进程实现分发到行情前置，接收和分发完全在内存中完成，网络迟延也被压缩到了极点。 托管于上期技术的程序化交易终端，因为通过局域网接入综合交易平台，其报单和行情速度处于目前业内最快水平。

- 期货交易数据交换协议(FTD)Futures trading data exchange protocol
  - 2005年证监会发布

二、 接口通用规则

- 命名规则
  - 请求指令:**Req**XXX 如ReqUserLogin
  - 请求响应:On**Rsp**XXX 如OnRspUserLogin
  - 查询指令:**Req**QryXXX 如ReqQryInstrument
  - 查询响应:On**Rsp**QryXXX 如OnRspQryInstrument
  - 回报响应:On**Rtn**XXX 如OnRtnOrder
  - 错误响应:On**ErrRtn**XXX 如OnErrRtnOrderInsert
- 通讯模式:
  - 对话通讯模式:由客户端主动发起请求。Thost收到请求、处理请求后，返回1条或者多条响应纪录。例如登入、各项查询、报单、撤单等操作。
  - 私有通讯模式:由Thost主动向客户端发出的相关信息。例如委托回报、成交回报、错单回报等
  - 广播通讯模式:由Thost主动向所有客户端发出的公共信息，例如行情等
- 数据流重传方式
  - TERT_RESTART:从本交易日开始重传
  - TERT_RESUME:从上次收到的续传（本地数据落地）
  - TERT_QUICK:只传送登录后私有流的内容
  - 开发过程中重传模式定义如下:

```
enum TE_RESUME_TYPE
{
    TERT_RESTART = 0,
    TERT_RESUME,
    TERT_QUICK
};
```

- 数据流重传方式注意事项
  - 通常使用Restart模式较为方便
  - 本地数据落地可用Resume模式，采用这种模式需要将上次流水的最后的流水号保存在本地，在重新连接时候需要读取本地的流水号作为参数传给订阅函数，以保证流水从上次断开处开始重传。可以参考一下例子

```
case __evtBizNotify: //CTP业务通知
    {
        char* p = (char*)item.data;
        CShfeFtdcSeqBizNoticeField* pSeqBizNotice = reinterpret_cast<CShfeFtdcSeqBizNoticeField*>(p);
        g_lastBizNotifySeqNum = pSeqBizNotice->UniqSequenceNo;
        g_memMgr.Release(p);
        PutFile("mycon/BizNotify.mcon",&g_lastBizNotifySeqNum,sizeof(g_lastBizNotifySeqNum));
    }
    break;
```

在本地保存最后一次流水号，用来下次重连时使用

将最后一次流水号保存在本地

```
ReadFile("mycon/Trade.mcon",&g_lastTradeSeqNum,sizeof(g_lastTradeSeqNum));
riskreq.SequenceNo = g_lastBizNotifySeqNum;
g_riskUserApi->ReqSubSeqData(&riskreq,g_requestId++); //订阅业务通知
```

在每次重连时如果采用Resume模式，将要读取本地保存的上次的最后 流水号，作为订阅业务通知的参数，系统将从上次最后一次流水开始推送流水

读取本地流水并且订阅

- 问：程序使用TradeApi和MdApi，并且把这2个dll 放在同一个目录下。程序再次启动后，如果某个api采用Resume模式订阅公有流/私有流，就会去参考相关的本地流文件。可能会导致数据异常？
- 答：相同目录下的2个dll 会把数据写入相同本地流文件，导致2个dll 不断的覆盖对方写下的流文件。程序再次启动时，TradeApi可能去参考MdApi写下的流文件，所以导致数据流不连续。
- 解决方法：如果一定要把2个dll 放在相同的目录下，可以在创建api时指定流文件的路径。使得不同的dll 写入不同流文件

- 交易接口流控
  - 查询1笔/秒
    - 问：现在对每秒发送查询数量的限制是多少？
    - 答：综合交易平台仅对查询进行流量限制，对交易指令没有限制。如果有在途的查询，不允许发新的查询。1秒钟最多允许发送1 个查询。返回值“-2”表示“未处理请求超过许可数”，“-3”表示“每秒发送请求数超过许可数”
  - 指令(报单/撤单/查询):每客户每连接6笔/秒,超过部分将排队
  - 同一帐户连接最大前置数:6个

- XXXSpi与XXXApi 接口
  - XXXSpi:响应回调函数,需要继承并重写回调函数来处理后台服务的响应，

如果处理服务器的响应取决于您怎么处理，此处的回调函数处理时间不宜过程，否则会阻塞其他流水，在重写的回调函数中，通常应迅速返回，可以利用将数据放入缓冲区或通过Windows的消息机制来实现。否则会阻塞流水，通讯停止。

- ✧ XXXApi:指令函数，不需要继承也不需要重写，客户端应用程序只需要直接调用Api 接口来发出操作请求

- 接口通用参数解释
  - ✧ nRequestID
    - 发送请求时需要设定RequestID,TraderApi返回响应时返回相关请求的RequestID。因为TraderApi是异步实现的，终端程序可能连续发出多个请求和查询指令。RequestID可以把请求/查询指令和相关的回报关联起来
  - ✧ IsLast
    - 无论是否有查询响应数据，只要查询响应结束，IsLast为true
  - ✧ 响应信息RspInfo
    - 如果RspInfo为空，或者RspInfo的错误代码为0，说明查询成功。
    - 否则RspInfo中会保存错误编码和错误信息。
    - 如下图 所示函数为对返回的响应信息进行分析的函数，根据分析的结果判断是否查询成功。

```
bool CTraderSpi::IsErrorRspInfo(CThostFtdcRspInfoField *pRspInfo)
{
    // 如果ErrorID != 0, 说明收到了错误的响应
    bool bResult = ((pRspInfo) && (pRspInfo->ErrorID != 0));
    if (bResult)
        cerr << "---->>> ErrorID=" << pRspInfo->ErrorID << ", ErrorMsg=" << pRspInfo->ErrorMsg << endl;
    return bResult;
}
```

- 查询响应数据
  - ✧ 查询响应方法每次返回1条记录。如果没有查询结果，就返回空指针

三、 开发准备

- C++开发环境
  - 跨平台的CodeBlocks 或者Microsoft Visual Studio 2010
  - wxWidgets
  - Boost

四、接口使用流程大致介

- 通讯的规则，过程和步骤

客户端和交易托管系统的通讯过程分为2个阶段：初始化阶段和功能调用阶段。

在初始化阶段，程序必须完成如下步骤（具体代码请参考开发实例）：

- ✧ 产生一个XXXApi实例
- ✧ 产生一个事件处理的实例(XXXSpi实例)
- ✧ 注册一个事件处理的实例（向XXXApi实例注册XXXspi实例用以处理返回的响应）

- ✧ 订阅私有流
- ✧ 订阅公共流
- ✧ 设置交易托管服务的地址
  - ● 在功能调用阶段，程序可以任意调用交易接口中的请求方法，如ReqOrderInsert 等。同时按照需要响应回调接口中的。
  - ● 其他注意事项：
    - ➢ API请求的输入参数不能为NULL。
    - ➢ API请求的返回参数，0表示正确，其他表示错误，详细错误编码请查表。

- 首先需要继承Spi 并实现虚方法用以处理响应数据
- 连接 的步骤
  - ✧ CreatApi 创建一个XXXApi实例和一个Spi实例用以处理响应数据
  - ✧ RegisterSpi 注册回调函数
  - ✧ RegisterFront 注册前置
  - ✧ (Trade)SubscribePrivateTopic 订阅私有流
  - ✧ (Trade)SubscribePublicTopic 订阅公有流
  - ✧ Init 初始化链接
  - ✧ //Join
  - ✧ OnFrontConnected 连接成功回调函数

下图是初始化和连接的实例

```
void main(void)
{
    // 初始化UserApi
    pUserApi = CThostFtdcTraderApi::CreateFtdcTraderApi();          // 创建UserApi
    CTraderSpi* pUserSpi = new CTraderSpi();
    pUserApi->RegisterSpi((CThostFtdcTraderSpi*)pUserSpi);          // 注册事件类
    pUserApi->SubscribePublicTopic(TERT_QUICK);                  // 注册公有流
    pUserApi->SubscribePrivateTopic(TERT_QUICK);                    // 注册私有流
    pUserApi->RegisterFront(FRONT_ADDR);                            // connect
    pUserApi->Init();

    pUserApi->Join();
//  pUserApi->Release();
```

在Init 初始化链接之后，如果连接成功OnFrontConnected回调函数将会被调用，这个函数的调用表示连接已经成功了，用户可以登录了，所以在这个回调函数回调函数中发起用户请求。

下图是nFrontConnected回调函数实例

```
//连接成功
void CtpRiskSpi::OnFrontConnected()
{
    cerr<<"连接交易前置...成功"<<endl;
    g_evtQueue.Release(EvtItem(__evtConnected));
}
```

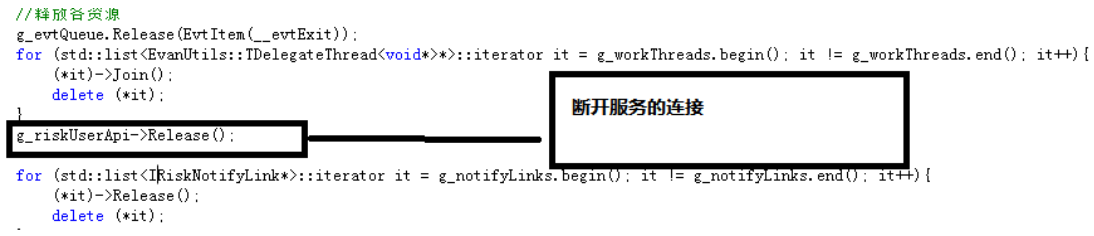
连接成功，触发消息处理线程。

在登录成功之后回调函数OnRspUserLogin将会被调用，在登录成功之后可以发起结算确认请求以及其他查询请求，下面是登录回调函数OnRspUserLogin实例。

```
void CTraderSpi::OnRspUserLogin(CThostFtdcRspUserLoginField *pRspUserLogin,
        CThostFtdcRspInfoField *pRspInfo, int nRequestID, bool bIsLast)
{
    cerr << "--->>> " << "OnRspUserLogin" << endl;
    if (bIsLast && !IsErrorRspInfo(pRspInfo))
    {
        // 保存会话参数
        FRONT_ID = pRspUserLogin->FrontID;
        SESSION_ID = pRspUserLogin->SessionID;
        int iNextOrderRef = atoi(pRspUserLogin->MaxOrderRef);
        iNextOrderRef++;
        sprintf(ORDER_REF, "%d", iNextOrderRef);
        ///获取当前交易日
        cerr << "--->>> 获取当前交易日 = " << pUserApi->GetTradingDay() << endl;
        ///投资者结算结果确认
        ReqSettlementInfoConfirm();
    }
}
```

- Release 断开连接的函数

  断开连接实例



注意事项：

✧ 问：测试时发现XXXSpi有个比较严重的问题，就是使用Release()退出清理对象时会出现死机，并且频率很高，怎样解决？

✧ 答：请参考以下代码的释放顺序。

```
template <class TUserApi>
void CUserApiEnv<TUserApi>::UnInitialUserApi()
{
```

```
  // 释放UserApi
     if (m_pUserApi)
     {
         m_pUserApi->RegisterSpi(NULL);
         m_pUserApi->Release();
         m_pUserApi = NULL;
     }
 // 释放UserSpi实例
   if (m_pUserSpiImpl)
   {
        delete m_pUserSpiImpl;
       m_pUserSpiImpl = NULL;
   }
}
```

- OnFrontDisconnected 连接断开回调函数，当客户和服务器断开连接，这个回调函数竟会被调用。
  - ✧ 返回代码及其的意义
    - 0x1001 网络读失败 => 4097
    - 0x1002 网络写失败
    - 0x2001 接收心跳超时
    - 0x2002 发送心跳失败
    - 0x2003 收到错误报文

```
///连接断开
void CtpRiskSpi::OnFrontDisconnected(int nReason)
{
  cerr<<" 响应 | 连接中断..."
    << " reason=" << nReason << endl;
}
```

回调实例

- ✧ 注意事项：当客户端与交易托管系统通信连接断开时，该方法被调用。当发生这个情况后，API会自动重新连接，客户端可不做处理。自动重连地址，可能是原来注册的地址，也可能是系统支持的其它可用的通信地址，它由程序自动选择。

五、细分的各功能点介绍

- ReqUserLogin 登录函数
  - ✧ OnRspUserLogin 登录回调函数
    - 非法登录
    - 未初始化
  - ✧ OnRspUserLogin 函数实例如下图

```
void CTraderSpi::OnRspUserLogin(CThostFtdcRspUserLoginField *pRspUserLogin,
        CThostFtdcRspInfoField *pRspInfo, int nRequestID, bool bIsLast)
{
    cerr << "--->>> " << "OnRspUserLogin" << endl;
    if (bIsLast && !IsErrorRspInfo(pRspInfo))
    {
        // 保存会话参数
        FRONT_ID = pRspUserLogin->FrontID;
        SESSION_ID = pRspUserLogin->SessionID;
        int iNextOrderRef = atoi(pRspUserLogin->MaxOrderRef);
        iNextOrderRef++;
        sprintf(ORDER_REF, "%d", iNextOrderRef);
        ///获取当前交易日
        cerr << "--->>> 获取当前交易日 = " << pUserApi->GetTradingDay() << endl;
        ///投资者结算结果确认
        ReqSettlementInfoConfirm();
    }
}
```

- ReqSettlementInfoConfirm 确认结算
  - ✧ OnRspSettlementInfoConfirm

```
void CTraderSpi::ReqSettlementInfoConfirm()
{
    CThostFtdcSettlementInfoConfirmField req;
    memset(&req, 0, sizeof(req));
    strcpy(req.BrokerID, BROKER_ID);
    strcpy(req.InvestorID, INVESTOR_ID);
    int iResult = pUserApi->ReqSettlementInfoConfirm(&req, ++iRequestID);
    cerr << "--->>> 投资者结算结果确认: " << ((iResult == 0) ? "成功" : "失败") << endl;
}
```

结算信息确认初始化参数，包括投资者资产号，和经纪人代码

发起投资者确认请求

结算确认请求实例

```
void CTraderSpi::OnRspSettlementInfoConfirm(CThostFtdcSettlementInfoConfirmField *pSettlementInfoConfirm,
    CThostFtdcRspInfoField *pRspInfo, int nRequestID, bool bIsLast)
{
    cerr << "--->>> " << "OnRspSettlementInfoConfirm" << endl;
    if (bIsLast && !IsErrorRspInfo(pRspInfo))
    {
        ///请求查询合约
        ReqQryInstrument();
    }
}
```

结算确认请求成功之后，回调函数会被调用，上图为回调函数实例，在结算确认之后可以查询合约和交易等操作

- ✧ 注意事项
  - 问：请问投资者结算结果确认是什么意思？有什么用？
  - 答：投资者在登录后首先需要确认自己的结算单（即账单），结算单确认后才可以进行交易操作。终端可以使ReqSettlementInfoConfirm请求确认结算单，请求时只需要填写经纪公司代码和投资者代码。查询结算单使用ReqQrySettlementInfo，不填日期，表示取上一交易日结算单。使用ReqQrySettlementInfoConfirm可以查询当天客户结算单确认情况，无记录返回表示当天未确认结算单，为避免客户当天多

次登陆多次重复确认结算单，建议在确认前先查询当天是否已经确认，如果客户已经确认过则不需要再次重复确认。

- 相关(当日首次登录必须确认结算)
  - ReqQrySettlementInfoConfirm
    - OnRspQrySettlementInfoConfirm(是否当日结算)
  - ReqQrySettlementInfo(查结算信息)
    - OnRspQrySettlementInfo
  - ReqCFMMCTradingAccountKey(查保证金监控中心密钥)
    - “https://investorservice.cfmmc.com/loginByKey.do?companyID=“ + f.ParticipantID + "&userid=" + f.AccountID + "&keyid=" + f.KeyID + "&passwd=" + f.CurrentKey
- □ReqQryInstrument 查合约
  - OnRspQryInstrument
    - 套利合约：SPxxx&xx(大商所跨期)/SPC(大商所跨品种)/SPD(郑商所跨期)

```
void CTraderSpi::ReqQryInstrument()
{
    CThostFtdcQryInstrumentField req;
    memset(&req, 0, sizeof(req));
    strcpy(req.InstrumentID, INSTRUMENT_ID);
    int iResult = pUserApi->ReqQryInstrument(&req, ++iRequestID);
    cerr << "--->>> 请求查询合约: " << ((iResult == 0) ? "成功" : "失败") << endl;
}
```

初始化查询合约的代码

发起查询合约的请求

查询合约的实例

```
void CTraderSpi::OnRspQryInstrument(CThostFtdcInstrumentField *pInstrument, CThostFtdcRspInfoField *pRspInfo,
    int nRequestID, bool bIsLast)
{
    cerr << "--->>> " << "OnRspQryInstrument" << endl;
    if (bIsLast && !IsErrorRspInfo(pRspInfo))
    {
        ///请求查询合约
        ReqQryTradingAccount();
    }
}
```

查询合约之后的成功之后该回调函数会被调用，

- ReqUserLogout登出
  - OnRspUserLogout

十、 行情

- SubscribeMarketData
  - 合约可重复订阅,对响应无影响，需要行情之前必须提前订阅
  - OnRspSubMarketData 行情订阅
  - 注册错误

```
void Quote::OnRspUserLogin(CThostFtdcRspUserLoginField* pRspUserLogin,
                           CThostFtdcRspInfoField* pRspInfo, int nRequestID, bool bIsLast)
{
    show("行情登录成功，订阅行情！");

    char* tmp[1] = {new char[256]};
    BOOST_FOREACH(mapInstrument::value_type i, dicInstrument)
    {
        strcpy(tmp[0], i.first.c_str());
        pUserApi->SubscribeMarketData(tmp, 1);
    }
}
```

订阅行情数据，如果需要获取行情必须提前订阅，订阅的实际在登录成功之后，如果订阅成功，服务器按照固定的时间频率推送行情数据过来

订阅行情实例

```
void Quote::OnRtnDepthMarketData(CThostFtdcDepthMarketDataField* pDepthMarketData)
{
    CThostFtdcDepthMarketDataField field;
    memset(&field, 0, sizeof(field));
    field = *pDepthMarketData;

    boost::thread tExecTick(&tickToFile, field);    //启线程
}
```

订阅之后，行情数据在不停的推送

新建一个线程处理不断推送过来的行情数据

行情接收实例，在订阅完之后，在这个回调函数中就能不断的接收到来自服务器推送的行情数据

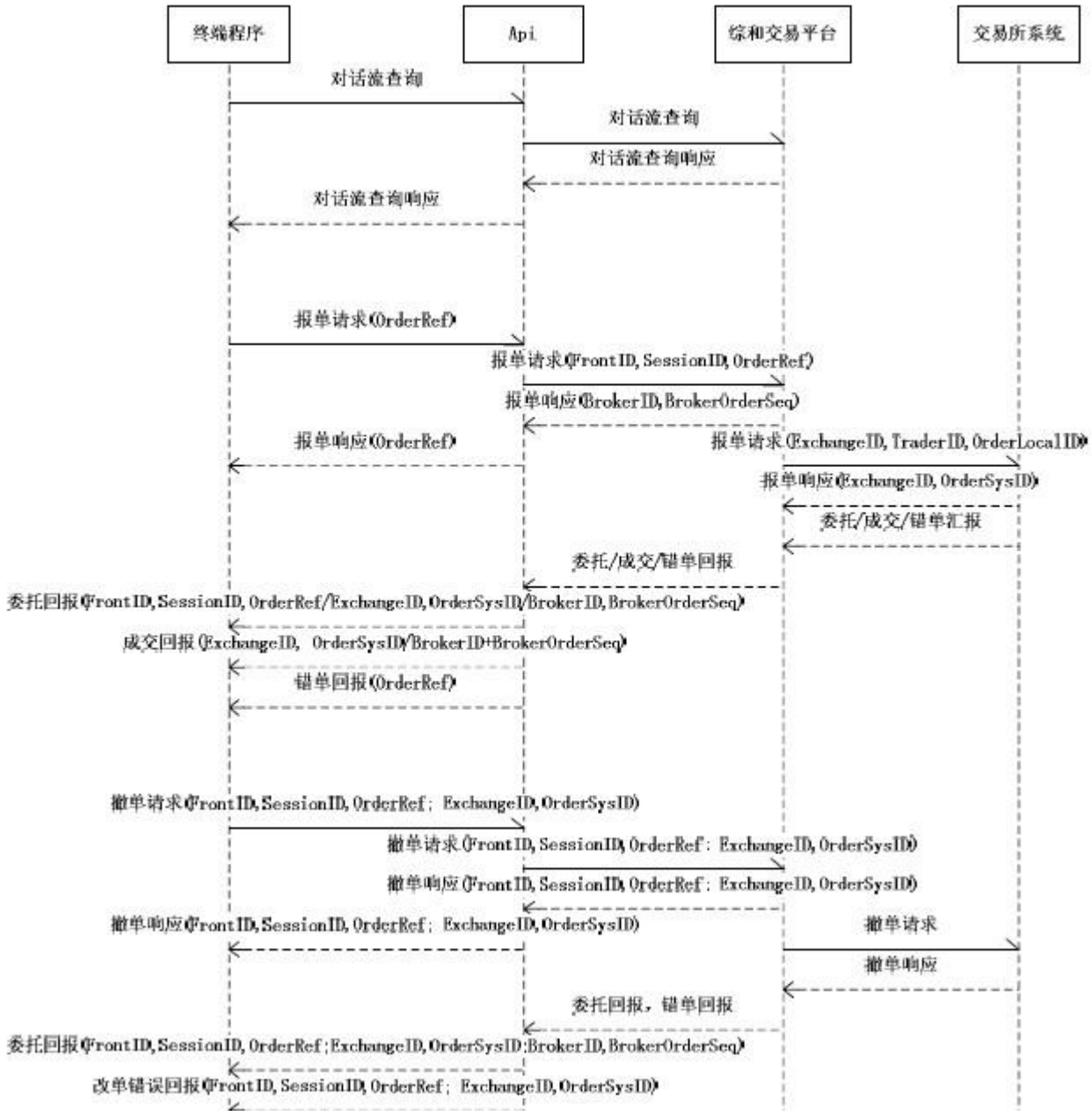
```
void tickToFile(CThostFtdcDepthMarketDataField f)
{
    try
    {
        //合法合约//合法数据
        if(dicInstrument.find(f.InstrumentID) != dicInstrument.end() && f.LastPrice < f.UpperLimitPrice)
        {
            CThostFtdcInstrumentField instField;
            memset(&instField, 0, sizeof(instField));
            instField = dicInstrument[f.InstrumentID];  //合约信息

            strcpy(f.TradingDay, tradingDay);           //日期
            strcpy(f.ExchangeID, instField.ExchangeID); //交易所

            if(f.AskPrice1 > f.UpperLimitPrice) //单边有挂边的情况
                f.AskPrice1 = f.LastPrice;
            if(f.BidPrice1 > f.UpperLimitPrice)
                f.BidPrice1 = f.LastPrice;

            if(instField.ExchangeID == "CZCE") //成交额与均价
                f.Turnover *= instField.VolumeMultiple;
            else
                f.AveragePrice /= instField.VolumeMultiple;

            dicTick[f.InstrumentID] = f;     //更新Tick
```

行情数据处理线程

行情数据处理线程

行情数据处理线程

- OnRtnDepthMarketData
  - Tick推送
  - 行情响应中处理延时,可能导致接口断开.
  - 日期字段为NULL
  - 交易所字段为NULL
  - 非法数据
    - LastPrice为最大值
    - 单边挂单(封板)
- 成交额与均价(CZCE)
- 注意事项:
  - 问：综合交易平台能显示买卖价的深度行情吗？比如说10 档买卖价？
  - 答：综合交易平台行情是从期货公司的远程席位获得，交易所分配给各期货公司的远程席位能够取得数据,综合交易平台都能提供给期货公司的客户。综合交易平台系统使用期货公司的远程席位登录交易所系统，对交易及数据的操作权限完全来源于期货公司。上期所也没有给我们提供任何市场之外的帮助或默许,但由于综合交易平台强大的数据处理能力,为期货投资者提供上期所早已公开提供的无限深度行情数据也有了技术上的可行性。
  - 问. 请问没有历史行情，实际交易时需要取历史数据做相应计算，比如atr(30)等，如何处理？是否只能终端通过别的接口自己补数据？
  - 答：历史数据需要通过行情商解决。对于未及时登录及断线造成的行情数据丢失，综合交易平台也不提供行情回补机制，因为行情的实时性对CTP 的系统延时要求非常高,行情数据的回补逻辑增加的系统延时以及网络资源的消耗限制了其在高速系统内部实现的空间。程序化交易终端可以通过多路连接的方式降低断线风险，或是托管策略服务器的方式以提高到CTP 连接的保障级别。



报单指令图

因为 TraderApi 是异步处理的，所以交易指令都是分2阶段提交。首先，Thost收到交易指令后，客户端会收到报单响应，确认收到客户端的交易指令。同时，Thost把交易指令转发到交易所。之后，Thost收到来自交易所的响应和回报，通过TraderApi 的回报事件通知给客户端。在报单交易过程中，会产生如下几组交易序列号：

- FrontID 、SessionID 、OrderRef

用户使用这组交易序列号可以按照自己的方式来唯一标示发出的任何一笔托。用户登入成功后，会收到前置机编号FrontID，会话编号SessionID 和最大 报单引用MaxOrderRef。用户在报单时设定报单引用OrderRef。OrderRef可以从MaxOrderRef开始递增。如果用户没有设定OrderRef，在报单响应中，Thost会为用户设置一个的OrderRef。使得每个报单的这组序列号保持唯一。因为这组交易

序列号是由用户设定的。所以在没有得到报单响应前，就可以使用这组交易序列号进行撤单操作。

- BrokerID、BrokerOrderSeq
  Thost收到用户报单后，为每个经纪公司的报单生成1组交易序列号。
- exchangeID 、traderID 、OrderLocalID
  交易席位在向交易所报单时，产生这组交易序列号，标示每一笔发往交易所的报单。
- exchangeID、OrderSysID
  交易所接受了投资者报单，产生这组交易序列号，标示每一笔收到的报单。

✧ 报单响应和回报

✧ Thost 收到报单指令，如果没有通过参数校验，拒绝接受报单指令。用户就会收到OnRspOrderInsert消息，其中包含了错误编码和错误消息。

- ReqOrderInsert 报单指令
  - ✧ OnRspOrderInsert
    - Thost收到报单指令，如果没有通过参数校验，拒绝接受报单指令回调函数
  - ✧ OnErrRtnOrderInsert
    - 交易所收到报单后认为报单错误的回调函数
  - ✧ OnRtnOrder
    - 委托回报，获取委托状态
  - ✧ OnRtnTrade
    - 与委托对应的成交回报

```
/////////////////////////////////////////////////////////////////////////
///TFtdcOrderStatusType是一个报单状态类型
/////////////////////////////////////////////////////////////////////////
///全部成交
#define THOST_FTDC_OST_AllTraded '0'
///部分成交还在队列中
#define THOST_FTDC_OST_PartTradedQueueing '1'
///部分成交不在队列中
#define THOST_FTDC_OST_PartTradedNotQueueing '2'
///未成交还在队列中
#define THOST_FTDC_OST_NoTradeQueueing '3'
///未成交不在队列中
#define THOST_FTDC_OST_NoTradeNotQueueing '4'
///撤单
#define THOST_FTDC_OST_Canceled '5'
///未知
#define THOST_FTDC_OST_Unknown 'a'
///尚未触发
#define THOST_FTDC_OST_NotTouched 'b'
```

委托状态图

- 限价单 实例

```
//报单-限价
void OrderInsert(const char* instrument, double price, int director, int offset, int volume)
{
    CThostFtdcInputOrderField f;
    memset(&f, 0, sizeof(f));
    f.CombHedgeFlag[0] = THOST_FTDC_HF_Speculation; //1投机
    f.ContingentCondition = THOST_FTDC_CC_Immediately;//立即触发
    f.ForceCloseReason = THOST_FTDC_FCC_NotForceClose;
    f.IsAutoSuspend = 0;
    f.OrderPriceType = THOST_FTDC_OPT_LimitPrice;
    f.TimeCondition = THOST_FTDC_TC_GFD;          //***立即完成1  当日有效3***
    f.VolumeCondition = THOST_FTDC_VC_AV;    //任意数量1  最小数量2  全部成交3
    f.MinVolume = 1;

    strcpy(f.InvestorID, trade->investor);
    strcpy(f.UserID, trade->investor);
    strcpy(f.BrokerID, trade->broker);

    strcpy(f.InstrumentID, instrument); //合约

    f.LimitPrice = price;

    if(director == 0)
        f.Direction = THOST_FTDC_D_Buy;        //买
    else
        f.Direction = THOST_FTDC_D_Sell;       //卖

    if(offset == 0)
        f.CombOffsetFlag[0] = THOST_FTDC_OF_Open;//开仓
    else if(offset == 1)
        f.CombOffsetFlag[0] = THOST_FTDC_OF_Close;  //平仓
    else
        f.CombOffsetFlag[0] = THOST_FTDC_OF_CloseToday; //平今

    f.VolumeTotalOriginal = volume;//数量

    sprintf(f.OrderRef, "%d", ++orderRef);//OrderRef

    ptime = boost::posix_time::microsec_clock::local_time();     //计时
    show(str(boost::format("报单(编号%1%).....") %f.OrderRef), 1);

    trade->pUserApi->ReqOrderInsert(&f, trade->iReqID++);     //报单
}
```

价格类型限价

限定的价格

买卖方向

下单

下单实例图

- 市价单 实例
  - ✧ 任意价格
  - ✧ 价格为0

```
void ReqOrderInsert(const char* instrument, int director, int offset, int volume)
{
    CThostFtdcInputOrderField f;
    memset(&f, 0, sizeof(f));
    f.CombHedgeFlag[0] = THOST_FTDC_HF_Speculation; //1投机
    f.ContingentCondition = THOST_FTDC_CC_Immediately;//立即触发
    f.ForceCloseReason = THOST_FTDC_FCC_NotForceClose;
    f.IsAutoSuspend = 0;
    f.OrderPriceType = THOST_FTDC_OPT_AnyPrice;
    f.TimeCondition =  THOST_FTDC_TC_IOC;
    f.VolumeCondition = THOST_FTDC_VC_AV;    //任意数量1  最小数量2  全部成交3
    f.MinVolume = 1;

    strcpy(f.InvestorID, trade->investor);
    strcpy(f.UserID, trade->investor);
    strcpy(f.BrokerID, trade->broker);

    strcpy(f.InstrumentID, instrument); //合约

    f.LimitPrice = 0;

    if(director == 0)
        f.Direction = THOST_FTDC_D_Buy;          //买
    else
        f.Direction = THOST_FTDC_D_Sell;         //卖
```

市价单以任意价格成交

限定价格为0

市价单实例

- 触发单

```
///触发条件：用户设定
ContingentCondition = ……;
/// 止损价：用户设定
StopPrice = ……;
/// 报单价格条件类型：限价
OrderPriceType = THOST_FTDC_OPT_LimitPrice;
/// 价格：用户设定
LimitPrice = ……;
/// 有效期类型类型：当日有效
TimeCondition = THOST_FTDC_TC_GFD;
```

用户限定触发条件

价格类型，限价

触发单实例

```
///报单通知
void CTraderSpi::OnRtnOrder(CThostFtdcOrderField *pOrder)
{
    cerr << "--->>> " << "OnRtnOrder"  << endl;
    if (IsMyOrder(pOrder))
    {
        if (IsTradingOrder(pOrder))
            ReqOrderAction(pOrder);
        else
    }
}
```

void CTraderSpi::ReqOrderAction(CThostFtdcOrderField *pOrder)

在报单进入交易所处理后，交易所对报单的委托回报

委托回报实例

```
///成交通知
void CTraderSpi::OnRtnTrade(CThostFtdcTradeField *pTrade)
{
    cerr << "--->>> " << "OnRtnTrade"  << endl;
}
```

成交回报实例

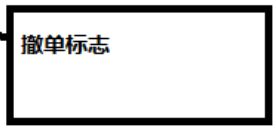
成交回报实例

- 关于平仓
  - ✧ 上期所区分昨仓和今仓。
    - ● 平昨仓时，开平标志类型设置为平仓THOST_FTDC_OF_Close
    - ● 平今仓时，开平标志类型设置为平今THOST_FTDC_OF_CloseToday
  - ✧ 其他交易所不区分昨仓和今仓。
  - ✧ 开平标志类型统一设置为平仓THOST_FTDC_OF_Close
  - ✧ 注意事项
    - ● 问：平今仓的时候，对大连或者郑州使用CloseToday是否有问题？
    - ● 答：后台有对应的转换，对DCE 和CZCE 的仓位使用平今或平昨都转换为平仓.

- ReqOrderAction 撤单
  - ✧ OnRspOrderAction
    - ● Thost收到撤单指令，如果没有通过参数校验，拒绝接受撤单指令
  - ✧ OnErrRtnOrderAction
    - ● 如果交易所认为报单错误
  - ✧ OnRtnOrder
    - ● 报单状态THOST_FTDC_OST_Canceled

```
//撤单
void ReqOrderAction(const char* instrument, int session, int frontid, const char* orderref)
{
    CThostFtdcInputOrderActionField f;
    memset(&f, 0, sizeof(f));
    f.ActionFlag = THOST_FTDC_AF_Delete;
    strcpy(f.BrokerID, trade->broker);
    strcpy(f.InvestorID, trade->investor);

    strcpy(f.InstrumentID, instrument);
    f.SessionID = session;
    f.FrontID = frontid;
    strcpy(f.OrderRef, orderref);
    trade->pUserApi->ReqOrderAction(&f, ++trade->iReqID);
}
```

撤单实例



撤单标志

  - ✧ 注意事项
    - ● 问：还有就是OnRtnOrder有重复推送的问题。比如发一个单，OnRtnOrder推了一个“已报”状态回来。然后我开始撤单，撤单一报入，OnRtnOrder 首先又推一个“已报”的状态回来，然后才

是“撤消”的状态。重复推送当然不会出错，不过会影响效率。

- 答：投资者发出1 个操作，都会收到2 条回报。具体说，1，发出1 笔委托，2，CTP发出“已提交”状态回报，3，CTP 转发交易所的“未成交”状态回报。下一个动作：1，用户发出撤单；2，CTP修改active User，再发出“未成交”状态回报；3，CTP 转发交易所的“已撤单”状态回报。就是CTP 应答一下，然后交易所又应答一下。都是把对应的委托状态从OnRtnOrder推回来。

- ReqQryInvestorPosition查持仓
  - OnRspQryInvestorPosition
    - 合约+多空+今昨
- 持仓为0
  - 报单未成交
  - 持仓全部平掉
  - 注意事项
    - 问：持仓查询记录中的昨持仓是今天开盘前的一个初始值，不会因为平昨或者平仓而减少。
    - 答：当前时侯的昨持仓=总持仓-今持仓。YdPosition := Position - TodayPosition。
  - 查持仓实例

```
void Trade::OnRspQryInvestorPosition(CThostFtdcInvestorPositionField* pInvestorPosition,
                                     CThostFtdcRspInfoField* pRspInfo, int nRequestID, bool bIsLast)
{
    string positionID = str(boost::format("%1%|%2%|%3%") %pInvestorPosition->InstrumentID
                            %pInvestorPosition->PosiDirection %pInvestorPosition->PositionDate);
    dicPosition[positionID] = *pInvestorPosition;

    if(bIsLast)
    {
        string msg = str(boost::format("\n%|1$+12|%|2$+12|%|3$+12|%|4$+12|%|5$+12|%|6$+12|\n")
                         %"合约" %"买卖" %"今昨" %"持仓" %"成本" %"持仓盈亏");
        BOOST_FOREACH(mapPosition::value_type i, dicPosition)
        {
            CThostFtdcInvestorPositionField f = i.second;
            msg += str(boost::format("%|1$+11|%|2$+11|%|3$+11|%|4$+11|%|5$+11|%|6$+11|\n")
                       %f.InstrumentID %(f.PosiDirection == THOST_FTDC_PD_Long?"多":"空")
                       %(f.PositionDate == THOST_FTDC_PSD_History?"昨仓":"今仓") %f.Position
                       %(f.PositionCost / dicInstrument[string(pInvestorPosition->InstrumentID)].VolumeMultiple)
                       %f.PositionProfit);
        }
        show(msg, 1);
    }
}
```

- ReqQryInvestorPositionDetail (推荐)
  - OnRspQryInvestorPositionDetail
  - 合成为InvestorPositionField

```
            CThostFtdcInvestorPositionField pf;
            if(iter == dicPosition.end())
            {
                memset(&pf, 0, sizeof(pf
                strcpy(pf.InstrumentID, f.InstrumentID);
                pf.PosiDirection = (f.Direction == THOST_FTDC_D_Buy ? THOST_FTDC_PD_Long: THOST_FTDC_PD_Short);
                pf.PositionDate = (strcmp(f.OpenDate, tradingDay) == 0? THOST_FTDC_PSD_Today : THOST_FTDC_PSD_History);
                pf.Position = 0;//f.Volume;
                pf.PositionCost = 0;//f.OpenPrice;
                pf.PositionProfit = 0;//f.PositionProfitByDate
                pf.TodayPosition = 0
                pf.YdPosition = 0;

            }
            else
            {
                pf = dicPosition[positionID];
            }

            pf.Position  += f.Volume;
            if(pf.PositionDate == THOST_FTDC_PSD_History)
            {
                pf.YdPosition += f.Volume;
                pf.PositionCost += f.Volume * f.LastSettlementPrice * multi;
                //持仓盈亏需要自己运算detail始终为0
                pf.PositionProfit += (f.Direction == THOST_FTDC_PD_Long? 1 : -1) *
                                     (dicTick[f.InstrumentID].LastPrice - f.LastSettlementPrice) * f.Volume * multi;
            }
```

```
            else
            {
                pf.TodayPosition += f.Volume;
                pf.PositionCost += f.Volume * f.OpenPrice * multi;
                //持仓盈亏需要自己运算detail始终为0
                pf.PositionProfit += (f.Direction == THOST_FTDC_PD_Long? 1 : -1) *
                                     (dicTick[f.InstrumentID].LastPrice - f.OpenPrice) * f.Volume * multi;
            }

            dicPosition[positionID] = pf;
        }
        show(msg, 1);

        //显示持仓汇总
        msg = str(boost::format("\n%|1$+12|%|2$+12|%|3$+12|%|4$+12|%|5$+12|%|6$+12|\n")
                  %"合约" %"买卖" %"今昨" %"持仓" %"成本" %"持仓盈亏");
        BOOST_FOREACH(mapPosition::value_type i, dicPosition)
        {
            CThostFtdcInvestorPositionField f = i.second;
            msg += str(boost::format("%|1$+11|%|2$+11|%|3$+11|%|4$+11|%|5$+11|%|6$+11|\n")
                       %f.InstrumentID %(f.PosiDirection == THOST_FTDC_PD_Long?"多":"空")
                       %(f.PositionDate == THOST_FTDC_PSD_History?"昨仓":"今仓") %f.Position
                       %(f.PositionCost / f.Position / dicInstrument[string(f.InstrumentID)].VolumeMultiple) %f.PositionProfit)
        }
        show(msg, 1);
    }
}
```

查持仓实例

- ReqQryTradingAccount查资金
  - ✧ OnRspQryTradingAccount
    - 静态权益=上日结算-出金金额+入金金额 account.PreBalance - account.Withdraw + account.Deposit
    - 动态权益=静态权益+ 平仓盈亏+ 持仓盈亏- 手续费 静态权益+ account.CloseProfit + account.PositionProfit - account.Commission
    - 可用资金(动态权益-占用保证金- 冻结保证金- 冻结手续费- 交割保证金) account.Available
  - ✧ 查询资金实例

```
//查资金
void Trade::OnRspQryTradingAccount(CThostFtdcTradingAccountField* pTradingAccount,
                                   CThostFtdcRspInfoField* pRspInfo, int nRequestID, bool bIsLast)
{
    if(bIsLast)
    {
        //静态权益=上日结算-出金金额+入金金额
        double preBalance = pTradingAccount->PreBalance - pTradingAccount->Withdraw + pTradingAccount->Deposit;
        //动态权益=静态权益+ 平仓盈亏+ 持仓盈亏- 手续费
        double current = preBalance
                         + pTradingAccount->CloseProfit + pTradingAccount->PositionProfit - pTradingAccount->Commission;
        string msg = str(boost::format("\n%|1$15|%|2$15|%|3$15|%|4$15|%|5$15|%|6$15|%|7$15|%|8$15|\n")
                         %"可用资金" %"平仓盈亏" %"持仓盈亏" %"静态权益" %"动态权益" %"占用保证金" %"冻结资金" %"风险度");
        msg += str(boost::format("%|1$14.2f|%|2$14.2f|%|3$14.2f|%|4$14.2f|%|5$14.2f|%|6$14.2f|%|7$14.2f|%|8$14.2f|%%")
                   %pTradingAccount->Available %pTradingAccount->CloseProfit %pTradingAccount->PositionProfit
                   %preBalance %current %pTradingAccount->CurrMargin
                   %(pTradingAccount->FrozenMargin + pTradingAccount->FrozenCommission)
                   %(pTradingAccount->CurrMargin / current * 100));
        show(msg, 1);
    }
}
```

查询资金之后的回调函数实例，返回资金信息

- ReqQryInstrumentCommissionRate查手续费
  - ✧ OnRspQryInstrumentCommissionRate
    - 按合约查询,以品种响应
- ReqQryInstrumentMarginRate查保证金
  - ✧ OnRspQryInstrumentMarginRate
    - 只能按合约查询
  - ✧ 注意事项：
    - 问：可以在综合交易平台上面设置保证金的算法--结算价/昨结算价/成交均价/开仓价四种算法分别都是什么意思？
    - 答：保证金算法：历史仓用昨结算价计算，今仓可以选择（成交均价/开仓价/结算价），这里的结算价是指最新价，该项配置由期货公司管理人员在后台进行配置，终端可以通过ＡＰＩ查到该配置的内容（v4.1 版本将支持该项查询，所以现在快期是在前端自己做配置）。
- ReqQryAccountregister(查签约银行) 银期转帐
  - ✧ OnRspQryAccountregister
  - ✧ 银行卡号

```
//签约银行
void Trade::OnRspQryAccountregister(CThostFtdcAccountregisterField* pAccountregister,
                                    CThostFtdcRspInfoField* pRspInfo, int nRequestID, bool bIsLast)
{
    if(IsErrorRspInfo(pRspInfo))
        show(str(boost::format("查签约银行错误:%1%") %pRspInfo->ErrorMsg));
    else
    {
        string bank = "";
        if(pAccountregister->BankID[0] == THOST_FTDC_BF_ABC)
            bank = "农业银行";
        else if(pAccountregister->BankID[0] == THOST_FTDC_BF_BC)
            bank = "中国银行";
        else if(pAccountregister->BankID[0] == THOST_FTDC_BF_BOC)
            bank = "交通银行";
        else if(pAccountregister->BankID[0] == THOST_FTDC_BF_CBC)
            bank = "建设银行";
        else if(pAccountregister->BankID[0] == THOST_FTDC_BF_ICBC)
            bank = "工商银行";
        else if(pAccountregister->BankID[0] == THOST_FTDC_BF_Other)
            bank = "其他银行";

        string bankID = string(pAccountregister->BankAccount);
        bankID = bankID.substr(strlen(pAccountregister->BankAccount)-4, 4);
        show(str(boost::format("%1%.%2%(****%3%)") %pAccountregister->BankID[0] % bank
                 %bankID), 3);
    }
}
```

查询签约银行信息回调实例

  - ✧ 注意事项
    - 问：银期转账里，业务功能码，应该填什么？

- 答：101001 银行发起转帐开户
  101002 银行发起签约销户
  101003 银行发起银行帐号变更
  102001 银行发起银行资金转期货
  102002 银行发起期货资金转银行
  103001 银行发起冲正银行转期货
  103002 银行发起冲正期货转银行
  104001 银行发起查询资金帐户余额
  104003 银行发起查询账户对应关系
  104005 完成银行向期货公司发起验证期货投资账号密码交易，并可查询账户
  104006 银行发起查询期货公司系统状态
  105099 银行下传对帐明细文件
  202001 期货发起银行资金转期货
  202002 期货发起期货资金转银行
  203001 期货发起冲正银行转期货
  203002 期货发起冲正期货转银行
  204002 期货发起查询银行余额
  204004 期货发起个人客户查询银期直通车开通情况
  204005 期货端发起查询转帐明细
  204006 期货公司发起查询银行系统状态
  204999 期货端发起查询客户平台当日流水
  206001 期货发起银行资金转期货资金（入金）通知
  206002 期货发起期货资金转银行资金（出金）通知
  901001 个人开通银期直通车
  901002 个人解除银期直通车
  905001 平台发起期商签到
  905002 平台发起期商签退
  905003 期货发起同步密钥

查询签约银行实例

- ReqFromBankToFutureByFuture(银转期)
  - OnRspFromBankToFutureByFuture
    - 错误响应
  - OnRtnFromBankToFutureByFuture
    - 成功
- ReqFromFutureToBankByFuture(期转银)
  - OnRspFromFutureToBankByFuture
    - 错误响应
  - OnRtnFromFutureToBankByFuture
    - 成功
- 要点
  - 涉及农行/中行的指令需要输入银行密码
  - BankBranchID=”0000”

```
//银转功能
void ReqTransferByFuture(const char* bankID, const char* bankPWD, const char* accountPWD, double amount, bool f2B)
{
    CThostFtdcReqTransferField f;
    memset(&f, 0, sizeof(f));
                    class CThostFtdcReqTransferField {...}
    strcpy(f.BankID, bankID);
    strcpy(f.BankBranchID, "0000");      //必须有
    strcpy(f.BankPassWord, bankPWD);

    strcpy(f.BrokerID, trade->broker);
    strcpy(f.AccountID, trade->investor);
    strcpy(f.Password, accountPWD);
    f.SecuPwdFlag = THOST_FTDC_BPWDF_BlankCheck;     //明文核对
    strcpy(f.CurrencyID, "RMB");                     //币种：RMB-人民币 USD-美圆 HKD-港元

    f.TradeAmount = amount;
    if(f2B) //期转银
        trade->pUserApi->ReqFromFutureToBankByFuture(&f, ++trade->iReqID);
    else    //银转期
        trade->pUserApi->ReqFromBankToFutureByFuture(&f, ++trade->iReqID);
}
```

查询银期转账实例

查询银期转账实例

六、常见问题

问:出现编译错误
答:在windows 环境中,要使用VC++编译器,否则会出现上面的提示.
问:OrderRef 报单失败
答:大商所与郑商所在报单未到达交易所时,用OrderRef 撤单会失败.
问:中金所套利编码
答: OrderInsert 中的CombHedgeFlag 字段设置为:THOST_FTDC_HF_Arbitrage

七、 模拟测试

- 国泰君安期货为客户提供综合交易平台模拟测试环境，供大家开发、测试及交易试用：
  - ✧ 交易前置：tcp://mn101.ctp.gtja-futures.com:41205
  - ✧ 行情前置：tcp://mn101.ctp.gtja-futures.com:41213
  - ✧ 经纪公司代码（BrokerId）1038
  - ✧ 测试账号(以下账号任选其一)：
    - 用户账号：00000041
    - 密码:123456
    - 用户账号：00000042
    - 密码:123456
    - 用户账号：00000043
    - 密码:123456
    - 用户账号：00000044
    - 密码:123456
    - 用户账号：00000045
    - 密码:123456
    - 用户账号：00000046
    - 密码:123456
    - 用户账号：00000047
    - 密码:123456
    - 用户账号：00000048

- 密码:123456
- 用户账号：00000049
- 密码:123456
- 用户账号：00000050
- 密码:123456