上海期货 信息技术有限公司 Shanghai Futures Information Technology Co.,Ltd.	Mini 项目组	编 号	
		版 号	V1.3

CTPIImini API 说明手册

修订记录、审核记录和审批记录

修订记录

版本编号	修订日期	修订人	主要修订摘要
V0.1.0	2016/7/1		第一版
V0.1.1	2016/1/5	陈子君	排版并增加持仓明细，持仓组合明细查询
V0.1.2	2017/9/5	陈子君	增加 API 不休眠功能
V0.1.3	2017/12/8	陈子君	增加 API 加密登录
V1.0	2018/9/19	郑学元	增加自对冲功能
V1.2	2019/2/25	郑学元	大商所期权六期接口
V1.3	2019/4/19	陈子君	删除不支持的 product 接口

目录

一、 概述

CTP2mini 的 api 接口与 mini1 是一致的，只是 api 内部采用了 UTP 协议与后台进行通讯,所以对于原有的客户端程序，直接替换 CTP2mini api 的 dll 和头文件即可成功编译。

二、 以下是已经实现的接口列表(未列出的接口一律不支持)：

```
///创建TraderApi
    ///@param pszFlowPath 存贮订阅信息文件的目录，默认为当前目录
    ///@return 创建出的UserApi
    static CThostFtdcTraderApi *CreateFtdcTraderApi(const char
*pszFlowPath = "");

    ///删除接口对象本身
    ///@remark 不再使用本接口对象时,调用该函数删除接口对象
    virtual void Release() = 0;

    ///初始化
    ///@remark 初始化运行环境,只有调用后,接口才开始工作，如果需要API
不休眠则填写Init(1)
    virtual void Init() = 0;

    ///等待接口线程结束运行
    ///@return 线程退出代码
    virtual int Join() = 0;

    ///注册前置机网络地址
    ///@param pszFrontAddress：前置机网络地址。
    ///@remark 网络地址的格式为：“protocol://ipaddress:port”，
如：”tcp://127.0.0.1:17001”。
    ///@remark “tcp”代表传输协议，“127.0.0.1”代表服务器地
址。”17001”代表服务器端口号。
    virtual void RegisterFront(char *pszFrontAddress) = 0;

    ///注册回调接口
    ///@param pSpi 派生自回调接口类的实例
    virtual void RegisterSpi(CThostFtdcTraderSpi *pSpi) = 0;

    ///订阅私有流。
```

```
    ///@param nResumeType 私有流重传方式
    ///        THOST_TERT_RESTART:从本交易日开始重传
    ///        THOST_TERT_RESUME:从上次收到的续传
    ///        THOST_TERT_QUICK:只传送登录后私有流的内容
    ///@remark 该方法要在Init方法前调用。若不调用则不会收到私有流的数据。
    virtual void SubscribePrivateTopic(THOST_TE_RESUME_TYPE nResumeType) = 0;

    ///订阅公共流。
    ///@param nResumeType 公共流重传方式
    ///        THOST_TERT_RESTART:从本交易日开始重传
    ///        THOST_TERT_RESUME:从上次收到的续传
    ///        THOST_TERT_QUICK:只传送登录后公共流的内容
    ///@remark 该方法要在Init方法前调用。若不调用则不会收到公共流的数据。
    virtual void SubscribePublicTopic(THOST_TE_RESUME_TYPE nResumeType) = 0;

///用户登录请求
    virtual int ReqUserLogin(CThostFtdcReqUserLoginField *pReqUserLoginField, int nRequestID)

///用户加密登录请求
   virtual int ReqUserLoginEncrypt(CThostFtdcReqUserLoginField *pReqUserLoginField, int nRequestID) = 0;


///登出请求
    virtual int ReqUserLogout(CThostFtdcUserLogoutField *pUserLogout, int nRequestID)

///报单录入请求
    virtual int ReqOrderInsert(CThostFtdcInputOrderField *pInputOrder, int nRequestID)

///报单录入请求CTP报错时的响应
    virtual      void      OnRspOrderInsert(CThostFtdcInputOrderField *pInputOrder, CThostFtdcRspInfoField *pRspInfo, int nRequestID, bool bIsLast)

///报单录入错误回报,不再使用
    virtual void OnErrRtnOrderInsert(CThostFtdcInputOrderField *pInputOrder, CThostFtdcRspInfoField *pRspInfo)
```

```
///报单通知
    virtual void OnRtnOrder(CThostFtdcOrderField *pOrder)

///成交通知
    virtual void OnRtnTrade(CThostFtdcTradeField *pTrade)

///报单操作请求
    virtual     int     ReqOrderAction(CThostFtdcInputOrderActionField
*pInputOrderAction, int nRequestID)

///报单操作请求CTP报错时响应
    virtual    void    OnRspOrderAction(CThostFtdcInputOrderActionField
*pInputOrderAction, CThostFtdcRspInfoField *pRspInfo, int nRequestID,
bool bIsLast)

///报单操作交易所报错时的回报
    virtual     void     OnErrRtnOrderAction(CThostFtdcOrderActionField
*pOrderAction, CThostFtdcRspInfoField *pRspInfo)

///批量报单操作请求
    virtual                                                    int
ReqBatchOrderAction(CThostFtdcInputBatchOrderActionField
*pInputBatchOrderAction, int nRequestID) = 0;

///批量报单操作请求CTP报错时响应
    virtual                                                   void
OnRspBatchOrderAction(CThostFtdcInputBatchOrderActionField
*pInputBatchOrderAction,    CThostFtdcRspInfoField    *pRspInfo,    int
nRequestID, bool bIsLast) {};

///批量报单操作交易所报错回报
    virtual                                                   void
OnErrRtnBatchOrderAction(CThostFtdcBatchOrderActionField
*pBatchOrderAction, CThostFtdcRspInfoField *pRspInfo) {};

///执行宣告录入请求
    virtual int ReqExecOrderInsert(CThostFtdcInputExecOrderField
*pInputExecOrder, int nRequestID) = 0;

///执行宣告录入请求CTP报错时响应
    virtual void OnRspExecOrderInsert(CThostFtdcInputExecOrderField
*pInputExecOrder, CThostFtdcRspInfoField *pRspInfo, int nRequestID,
bool bIsLast) {};
```

```
///执行宣告录入错误回报,不再使用
    virtual void
OnErrRtnExecOrderInsert(CThostFtdcInputExecOrderField
*pInputExecOrder, CThostFtdcRspInfoField *pRspInfo) {};

///执行宣告通知
    virtual void OnRtnExecOrder(CThostFtdcExecOrderField *pExecOrder)
{};

///执行宣告操作请求
    virtual int
ReqExecOrderAction(CThostFtdcInputExecOrderActionField
*pInputExecOrderAction, int nRequestID) = 0;

    ///执行宣告操作请求CTP报错时响应
    virtual void
OnRspExecOrderAction(CThostFtdcInputExecOrderActionField
*pInputExecOrderAction, CThostFtdcRspInfoField *pRspInfo, int
nRequestID, bool bIsLast) {};

///执行宣告操作交易所报错回报
    virtual void
OnErrRtnExecOrderAction(CThostFtdcExecOrderActionField
*pExecOrderAction, CThostFtdcRspInfoField *pRspInfo) {};

///询价录入请求
    virtual int ReqForQuoteInsert(CThostFtdcInputForQuoteField
*pInputForQuote, int nRequestID) = 0;

    ///询价录入请求CTP报错时响应
    virtual void OnRspForQuoteInsert(CThostFtdcInputForQuoteField
*pInputForQuote, CThostFtdcRspInfoField *pRspInfo, int nRequestID,
bool bIsLast) {};

///询价录入交易所报错回报
    virtual void OnErrRtnForQuoteInsert(CThostFtdcInputForQuoteField
*pInputForQuote, CThostFtdcRspInfoField *pRspInfo) {};

///报价录入请求
    virtual int ReqQuoteInsert(CThostFtdcInputQuoteField
*pInputQuote, int nRequestID) = 0;

    ///报价录入请求CTP报错时响应
```

```
    virtual void OnRspQuoteInsert(CThostFtdcInputQuoteField
*pInputQuote, CThostFtdcRspInfoField *pRspInfo, int nRequestID, bool
bIsLast) {};

///报价录入错误回报,不再使用
    virtual void OnErrRtnQuoteInsert(CThostFtdcInputQuoteField
*pInputQuote, CThostFtdcRspInfoField *pRspInfo) {};

///报价通知
    virtual void OnRtnQuote(CThostFtdcQuoteField *pQuote) {};

///报价操作请求
    virtual     int     ReqQuoteAction(CThostFtdcInputQuoteActionField
*pInputQuoteAction, int nRequestID) = 0;

///报价操作请求CTP报错时响应
    virtual void OnRspQuoteAction(CThostFtdcInputQuoteActionField
*pInputQuoteAction, CThostFtdcRspInfoField *pRspInfo, int nRequestID,
bool bIsLast) {};

///报价操作交易所报错回报
    virtual void OnErrRtnQuoteAction(CThostFtdcQuoteActionField
*pQuoteAction, CThostFtdcRspInfoField *pRspInfo) {};

///期权自对冲录入请求
    virtual int
ReqOptionSelfCloseInsert(CThostFtdcInputOptionSelfCloseField
*pInputOptionSelfClose, int nRequestID);

///期权自对冲操作请求
    virtual int
ReqOptionSelfCloseAction(CThostFtdcInputOptionSelfCloseActionField
*pInputOptionSelfCloseAction, int nRequestID);
```

大商所期权对冲和履约对冲也使用本接口，区别是履约对冲不需要合约号。大商所的期权自对冲参考上期所的期权自对冲，大商所的履约对冲参考上期所期权卖方期货自对冲。具体使用方法请参考CTP1接口。

```
///期权自对冲录入请求响应
    virtual void
OnRspOptionSelfCloseInsert(CThostFtdcInputOptionSelfCloseField
*pInputOptionSelfClose, CThostFtdcRspInfoField *pRspInfo, int
nRequestID, bool bIsLast);
```

```
///期权自对冲操作请求响应
    virtual void
OnRspOptionSelfCloseAction(CThostFtdcInputOptionSelfCloseActionField
*pInputOptionSelfCloseAction, CThostFtdcRspInfoField *pRspInfo, int
nRequestID, bool bIsLast);

///期权自对冲通知
    virtual void OnRtnOptionSelfClose(CThostFtdcOptionSelfCloseField
*pOptionSelfClose);

///期权自对冲操作错误回报
    virtual void
OnErrRtnOptionSelfCloseAction(CThostFtdcOptionSelfCloseActionField
*pOptionSelfCloseAction, CThostFtdcRspInfoField *pRspInfo);
```

三、 以下是已实现的查询接口

```
///请求查询报单
virtual int ReqQryOrder(CThostFtdcQryOrderField *pQryOrder, int
nRequestID) = 0;

    ///请求查询成交
virtual int ReqQryTrade(CThostFtdcQryTradeField *pQryTrade, int
nRequestID) = 0;

    ///请求查询投资者持仓
 virtual int
 ReqQryInvestorPosition(CThostFtdcQryInvestorPositionField
 *pQryInvestorPosition, int nRequestID) = 0;

///请求查询投资者持仓明细
 virtual  int  ReqQryInvestorPositionDetail(CThostFtdcQryInvestorPos
 itionDetailField  *pQryInvestorPositionDetail,  int  nRequestID)  =
   0;
```

目前只支持大商所

```
///请求查询投资者组合持仓明细
virtual int ReqQryInvestorPositionCombineDetail(CThostFtdcQryInvestorPositionCombineDetailField *pQryInvestorPositionCombineDetail, int nRequestID) = 0;
```

目前只支持大商所

```
    ///请求查询资金账户
    virtual int ReqQryTradingAccount(CThostFtdcQryTradingAccountField *pQryTradingAccount, int nRequestID) = 0;

    ///请求查询交易编码
    virtual int ReqQryTradingCode(CThostFtdcQryTradingCodeField *pQryTradingCode, int nRequestID) = 0;

    ///请求查询合约保证金率
    virtual int ReqQryInstrumentMarginRate(CThostFtdcQryInstrumentMarginRateField *pQryInstrumentMarginRate, int nRequestID) = 0;

    ///请求查询合约手续费率
    virtual int ReqQryInstrumentCommissionRate(CThostFtdcQryInstrumentCommissionRateField *pQryInstrumentCommissionRate, int nRequestID) = 0;

///请求查询行情
    virtual int ReqQryDepthMarketData(CThostFtdcQryDepthMarketDataField *pQryDepthMarketData, int nRequestID) = 0;

///请求查询合约
    virtual int ReqQryInstrument(CThostFtdcQryInstrumentField *pQryInstrument, int nRequestID) = 0;

///请求查询投资者
    virtual int ReqQryInvestor(CThostFtdcQryInvestorField *pQryInvestor, int nRequestID) = 0;

///请求查询交易所
    virtual int ReqQryExchange(CThostFtdcQryExchangeField *pQryExchange, int nRequestID) = 0;


///请求查询合约状态
    virtual int
```

```
ReqQryInstrumentStatus(CThostFtdcQryInstrumentStatusField
*pQryInstrumentStatus, int nRequestID) = 0;

///请求查询期权交易成本
    virtual int
ReqQryOptionInstrTradeCost(CThostFtdcQryOptionInstrTradeCostField
*pQryOptionInstrTradeCost, int nRequestID) = 0;

///请求查询期权合约手续费
    virtual int
ReqQryOptionInstrCommRate(CThostFtdcQryOptionInstrCommRateField
*pQryOptionInstrCommRate, int nRequestID) = 0;

///请求查询执行宣告
    virtual int ReqQryExecOrder(CThostFtdcQryExecOrderField
*pQryExecOrder, int nRequestID) = 0;

///请求查询询价
    virtual int ReqQryForQuote(CThostFtdcQryForQuoteField
*pQryForQuote, int nRequestID) = 0;

///请求查询报价
    virtual int ReqQryQuote(CThostFtdcQryQuoteField *pQryQuote, int
nRequestID) = 0;
```

四、 业务算法的修改：

(a).资金表(TradingAccount),仅计算以下字段：

FrozenMargin:冻结保证金---期货：限价单以开仓价计算，其余以涨停价计算；期权：一律以昨结算价计算

FrozenCash:冻结权利金

CurrMargin:保证金---期货：昨仓以昨结算价计算，今仓以开仓价计算；平仓后减少的保证金为持仓均价*平仓数量;支持按产品组收取大边保证金

期权：一律以昨结算价计算

Commission:手续费,分开仓，平今，平昨计算

CloseProfit:平仓盈亏，由于不使用持仓明细计算，所以平仓盈亏=(成交价-持仓均价)*平仓数量

CashIn:权利金

Available:可用资金，随且仅随着以上字段的变化而变化

(b).持仓表(InvestorPosition),初始化数据需要这张表(不再使用原来的持仓明细表)；仅计算以下字段:

LongFrozen:买冻结,平空头仓时会更新该字段

ShortFrozen:卖冻结,平多头仓时会更新该字段
Position:总仓
TodayPosition:今仓
PositionCost：持仓成本
CloseProfit:平仓盈亏