金仕达

KSFT_API4Ard
特别说明

文档标识

文档名称	KSFT_API4Ard 特别说明
版本号	<V1.0>
状况	◉ 草案 ○ 评审过的 ○ 更新过的 ○ 定为基线的

文档修订历史

版本	日期	描述	修订者
V1.0	<2016-05-27>	创建文件	IRDG API 开发组

正式核准

姓名	签字	日期

分发控制

副本	接受人	机构

目 录

1. API 文件

Android 平台 API 包含文件如下：

库文件名	库文件描述
libkstradeapi.so	Android 平台交易 API 动态链接库
libkstradeapi_wrap.so	实现 C++与 Java 转换的交易动态链接库
libksmarketdataapi.so	Android 平台行情 API 动态链接库
libksmarketdataapi_wrap.so	实现 C++与 Java 转换的行情动态链接库
kstradeapi.jar	Android java 接口交易 jar 包
ksmarketdataapi.jar	Android java 接口行情 jar 包
KSInterB2C.lkc	客户端授权文件

2. 使用方法

2.1 库文件及 jar 包

以使用交易 api 及 Eclipse IDE 为例，armeabi 文件夹包含两个库文件，分别是 libkstradeapi.so 和 libkstradeapi_wrap.so。请将发布包中 armeabi 文件夹放在工程中 libs 目录下。

jar 包添加可选择工程--右键--Properties--Java Build Path--Libraries--Add External JARs，选择 kstradeapi.jar 进行添加。

2.2 授权文件

授权文件放在用户目录下：getApplicationContext().getFilesDir().getAbsolutePath()。

2.3 注意事项

使用交易 API 前先导入开发包：

```
import com.kingstar.kstradeapi.*;
```

然后在 MainActivity 中添加对库文件的引用（因*_wrap.so 依赖于*.so，故需先加载 libkstradeapi.so 再 libkstradeapi _wrap.so）：

```
static {
    try {
        System.loadLibrary("kstradeapi");
        System.loadLibrary("kstradeapi_wrap");
    } catch(Exception e) {
        e.printStackTrace(System.out);
    }
}
```

CTP 交易接口详见：说明文档\doc\com\kingstar\kstradeapi\CThostFtdcTraderApi.html 和 CThostFtdcTraderSpi.html。
KS 条件单 Cos 接口详见：说明文档\doc\com\kingstar\kstradeapi\CKSCosApi.html 和 CKSCosSpi.html。
KS 期权 Option 接口详见：说明文档\doc\com\kingstar\kstradeapi\CKSOptionApi.html 和 CKSOptionSpi.html。
KS 客户需求 Voc 接口详见：说明文档\doc\com\kingstar\kstradeapi\CKSVocApi.html 和 CKSVocSpi.html。
Cos（Option、Voc）接口使用时先调用 LoadCosApi（LoadOptApi、LoadVocApi）得到对应的 Api，再调用其中的方法。

3. 示例代码

```
package com.kingstar.ksfapidemo;
import android.os.Bundle;
import android.os.Looper;
import android.os.Message;
import android.os.Handler;
import android.app.Activity;
import android.text.method.ScrollingMovementMethod;
import android.view.Menu;
import android.view.View;
import android.widget.Button;
```

```
import android.widget.ScrollView;
import android.widget.TextView;

import com.kingstar.kstradeapi.*;
//以下写法为兼容 CTP 接口的调用方法
public class MainActivity extends Activity {
    private static final int INFO_FRONT_SUCCESS = 0;// 连接前置成功
    private static final int INFO_FRONT_DISCONNECT = 1;// 连接前置断开
    private static final int INFO_LOGIN_SUCCESS = 2;// 登录成功
    private static final int INFO_LOGIN_FAILURE = 3;// 登录失败

    private static final int INFO_REQ_INSTRUMENT_FINSH = 4; // 合约查询返回
    private static final int INFO_REQ_ACCOUNT_FINSH = 5;    // 资金查询返回
    private static final int INFO_REQ_POSITION_FINSH = 6;   // 持仓查询返回
    private static final int INFO_REQ_POSITIONDETAIL_FINSH = 7;   // 持仓明细查询返回
    private static final int INFO_ORDERINSERT_OK = 8;
    private static final int INFO_RTNORDER = 9;

    private static final String FrontAddress   = "tcp://10.253.117.107:13163";
    private static final String BrokerID  = "31000853";
    private static final String UserID = "80001";
    private static final String InvestorID = "80001";
    private static final String InstrumentID = "IF1611";
    private static final String Password = "123456";

    CThostFtdcTraderApi api = null;
    TextView outputText = null;
    ScrollView scroller = null;
    Button Connect = null;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        outputText = (TextView)findViewById(R.id.OutputText);
        outputText.setText("Press 'Connect' button to Connect Front.\n");
        outputText.setMovementMethod(new ScrollingMovementMethod());
        scroller = (ScrollView)findViewById(R.id.Scroller);
        Connect = (Button)findViewById(R.id.connect);
    }

    @Override
```

```
public boolean onCreateOptionsMenu(Menu menu) {
    // Inflate the menu; this adds items to the action bar if it is present.
    getMenuInflater().inflate(R.menu.main, menu);
    return true;
}

class TraderHandler extends Handler
{
    public TraderHandler(Looper looper)
    {
        super(looper);
    }
    public void handleMessage(Message msg)
    {
        switch (msg.what)
        {
            case INFO_FRONT_SUCCESS:
                outputText.append("前置已经连接，发送登录\n");

                CThostFtdcReqUserLoginField ReqUserLogin = new CThostFtdcReqUserLoginField();

                ReqUserLogin.setBrokerID(BrokerID);
                ReqUserLogin.setUserID(UserID);
                ReqUserLogin.setPassword(Password);
                ReqUserLogin.setUserProductInfo("KSFTard");
                api.ReqUserLogin(ReqUserLogin, 0);
                break;

            case INFO_FRONT_DISCONNECT:
                outputText.append("前置断开\n");
                Connect.setEnabled(true);

                break;

            case INFO_LOGIN_SUCCESS:
                outputText.append("登入成功，查询合约");
                CThostFtdcQryInstrumentField QryInstrument = new
CThostFtdcQryInstrumentField();
                QryInstrument.setInstrumentID(InstrumentID);
                QryInstrument.setExchangeID("CFFEX");
                api.ReqQryInstrument(QryInstrument, 1);
```

```
                    break;
                case INFO_REQ_INSTRUMENT_FINSH:
                    outputText.append(msg.toString());

                    try {
                        Thread.sleep(1000);
                    }
                    catch (InterruptedException e) {

                    }
                    outputText.append("\n 查询资金");
                    CThostFtdcQryTradingAccountField QryTradingAccount= new
CThostFtdcQryTradingAccountField();
                    QryTradingAccount.setInvestorID(InvestorID);
                    QryTradingAccount.setBrokerID(BrokerID);

                    api.ReqQryTradingAccount(QryTradingAccount,2);

                    break;

                case INFO_REQ_ACCOUNT_FINSH:
                    outputText.append(msg.toString());
                    outputText.append("\n 查询持仓");
                    try {
                        Thread.sleep(1000);
                    }
                    catch (InterruptedException e) {

                    }
                    CThostFtdcQryInvestorPositionField QryInvestorPosition = new
CThostFtdcQryInvestorPositionField();
                    QryInvestorPosition.setBrokerID(BrokerID);
                    QryInvestorPosition.setInvestorID(InvestorID);
                    QryInvestorPosition.setInstrumentID(InstrumentID);

                    api.ReqQryInvestorPosition(QryInvestorPosition, 3);
                    break;

                case INFO_REQ_POSITION_FINSH:
```

```
                    outputText.append(msg.toString());

                    outputText.append("\n 查询持仓明细");
                    try {
                        Thread.sleep(1000);
                    }
                    catch (InterruptedException e) {

                    }
                    CThostFtdcQryInvestorPositionDetailField QryInvestorPositionDetail = new
CThostFtdcQryInvestorPositionDetailField();
                    QryInvestorPositionDetail.setBrokerID(BrokerID);
                    QryInvestorPositionDetail.setInvestorID(InvestorID);

                    api.ReqQryInvestorPositionDetail(QryInvestorPositionDetail, 4);

                    break;

                case INFO_REQ_POSITIONDETAIL_FINSH:
                    outputText.append(msg.toString());

                    try {
                        Thread.sleep(1000);
                    }
                    catch (InterruptedException e) {

                    }
                    outputText.append("\n 发送报单");

                    CThostFtdcInputOrderField InputOrder = new CThostFtdcInputOrderField();
                    InputOrder.setBrokerID(BrokerID);
                    InputOrder.setInvestorID(InvestorID);
                    InputOrder.setInstrumentID(InstrumentID);
                    InputOrder.setOrderRef("");
                    InputOrder.setUserID(UserID);
                    InputOrder.setOrderPriceType('2');
                    InputOrder.setDirection(kstradeapiConstants.THOST_FTDC_D_Buy);
                    InputOrder.setCombOffsetFlag("0");
                    InputOrder.setCombHedgeFlag("1");
                    InputOrder.setLimitPrice(5000);
                    InputOrder.setVolumeTotalOriginal(1);
```

```
                    InputOrder.setTimeCondition(kstradeapiConstants.THOST_FTDC_TC_GFD);
                    InputOrder.setGTDDate("");
                    InputOrder.setVolumeCondition(kstradeapiConstants.THOST_FTDC_VC_AV);
                    InputOrder.setMinVolume(1);

InputOrder.setContingentCondition(kstradeapiConstants.THOST_FTDC_CC_Immediately);

InputOrder.setForceCloseReason(kstradeapiConstants.THOST_FTDC_FCC_NotForceClose);
                    InputOrder.setIsAutoSuspend(0);
                    InputOrder.setBusinessUnit("");
                    InputOrder.setUserForceClose(0);
                    InputOrder.setIsSwapOrder(0);

                    api.ReqOrderInsert(InputOrder, 5);
                    break;
                case INFO_ORDERINSERT_OK:
                    outputText.append(msg.toString());
                    outputText.append("\n 测试完成");
                    break;

                case INFO_RTNORDER:
                    outputText.append(msg.toString()+"\n");
                    outputText.append("\n 测试完成");
                    break;
            }
        }
    }


    public void onConnectClick(View view)
    {
        outputText.append("\nStart to connect Front...\n");

        new apiThread().start();
        Connect.setEnabled(false);
        // Ensure scroll to end of text

        scroller.post(new Runnable()
        {
            public void run() {
                scroller.fullScroll(ScrollView.FOCUS_DOWN);
```

```
        }
    });
}

public static String getBrokerID() {
    return BrokerID;
}

public static String getUserID() {
    return UserID;
}

public static String getFrontAddress() {
    return FrontAddress;
}

public static int getMsgReqInstrumentFinsh() {
    return INFO_REQ_INSTRUMENT_FINSH;
}

public static String getInvestorID() {
    return InvestorID;
}

public static String getInstrumentID() {
    return InstrumentID;
}


//加了一个线程类
class apiThread extends Thread
{
    public void run()
    {
        String apppath = getApplicationContext().getFilesDir().getAbsolutePath();
        apppath += "/";
        try
        {
            api = CThostFtdcTraderApi.CreateFtdcTraderApi(apppath);
        }
        catch(Exception e)
```

```
            {
                e.printStackTrace(System.out);
            }

            TraderApiSample testspi = new TraderApiSample();
            testspi.initspi(api);

            api.RegisterSpi(testspi);

            api.RegisterFront(FrontAddress);

            api.SubscribePrivateTopic(THOST_TE_RESUME_TYPE.THOST_TERT_QUICK);
            api.Init();
            api.Join();
        }
    }

    class TraderApiSample extends CThostFtdcTraderSpi
    {
        Looper looper = null;
        TraderHandler myhandler =null;
        public void initspi(CThostFtdcTraderApi api)
        {
            //主线程的 Looper 对象
            looper  =  Looper.getMainLooper();
            //这里以主线程的 Looper 对象创建了 handler，
            //所以，这个 handler 发送的 Message 会被传递给主线程的 MessageQueue。
            myhandler = new TraderHandler(looper);
        }


        @Override
        public void OnFrontConnected()
        {
            System.out.println("front connected");
            //构建 Message 对象
            //第一个参数：是自己指定的 message 代号，方便在 handler 选择性地接收
            //第二三个参数没有什么意义
            //第四个参数需要封装的对象
            Message msg = myhandler.obtainMessage(INFO_FRONT_SUCCESS,1,1,"前置已经连接\n");
            myhandler.sendMessage(msg); //发送消息
```

```
        }
        @Override
        public void OnFrontDisconnected(int nReason)
        {
            Message msg = myhandler.obtainMessage(INFO_FRONT_DISCONNECT,1,1,nReason);
            myhandler.sendMessage(msg); //发送消息
        }

        @Override
        public void OnRspUserLogin(CThostFtdcRspUserLoginField pRspUserLogin,
CThostFtdcRspInfoField pRspInfo, int nRequestID, boolean bIsLast)
        {
            if(pRspInfo.getErrorID() == 0)
            {
                Message msg = myhandler.obtainMessage(INFO_LOGIN_SUCCESS,1,1,"登录成功");
                myhandler.sendMessage(msg); //发送消息
            }
            else
            {
                Message msg = myhandler.obtainMessage(INFO_LOGIN_FAILURE,1,1,"登录失败\n");
                myhandler.sendMessage(msg); //发送消息
            }
        }

        @Override
        public void OnRspQryInstrument(CThostFtdcInstrumentField pInstrument,
CThostFtdcRspInfoField pRspInfo, int nRequestID, boolean bIsLast)
        {
            if(bIsLast==true)
            {
                Message msg =
myhandler.obtainMessage(INFO_REQ_INSTRUMENT_FINSH,1,1,pInstrument.getInstrumentID() + "的交易所
代码:" + pInstrument.getExchangeID());
                myhandler.sendMessage(msg); //发送消息
            }
        }

        @Override
        public void OnRspQryTradingAccount(CThostFtdcTradingAccountField pTradingAccount,
CThostFtdcRspInfoField pRspInfo, int nRequestID, boolean bIsLast)
        {
```

```
            if(bIsLast==true)
            {
                Message msg =
myhandler.obtainMessage(INFO_REQ_ACCOUNT_FINSH,1,1,pTradingAccount.getAccountID() + "的可用资
金:" + pTradingAccount.getAvailable());
                myhandler.sendMessage(msg); //发送消息
            }
        }

        @Override
        public void OnRspQryInvestorPosition(CThostFtdcInvestorPositionField pInvestorPosition,
CThostFtdcRspInfoField pRspInfo, int nRequestID, boolean bIsLast)
        {
            if(bIsLast==true)
            {
                Message msg =
myhandler.obtainMessage(INFO_REQ_POSITION_FINSH,1,1,pInvestorPosition.getInvestorID() +"的合约"
+ pInvestorPosition.getInstrumentID()+"总持仓为:"+pInvestorPosition.getPosition());
                myhandler.sendMessage(msg); //发送消息
            }

        }

        @Override
        public void OnRspQryInvestorPositionDetail(CThostFtdcInvestorPositionDetailField
pInvestorPositionDetail, CThostFtdcRspInfoField pRspInfo, int nRequestID, boolean bIsLast)
        {
            if(bIsLast==true)
            {
                Message msg =
myhandler.obtainMessage(INFO_REQ_POSITIONDETAIL_FINSH,1,1,pInvestorPositionDetail.getInvestorID
() + "的合约:" + pInvestorPositionDetail.getInstrumentID()+"投资者保证
金:"+pInvestorPositionDetail.getMargin());
                myhandler.sendMessage(msg); //发送消息
            }
        }

        @Override
        public void OnRspOrderInsert(CThostFtdcInputOrderField pInputOrder,
CThostFtdcRspInfoField pRspInfo, int nRequestID, boolean bIsLast)
        {
```

```
            if(bIsLast == true)
            {
                Message msg =
myhandler.obtainMessage(INFO_ORDERINSERT_OK,1,1,pRspInfo.getErrorMsg());
                myhandler.sendMessage(msg); //发送消息
            }

        }

        @Override
        public void OnRtnOrder(CThostFtdcOrderField pOrder)
        {
            Message msg = myhandler.obtainMessage(INFO_RTNORDER,1,1,"委托回报:" +
pOrder.getOrderStatus());
            myhandler.sendMessage(msg); //发送消息
        }

    }


    static {
        try {
            System.loadLibrary("kstradeapi");
            System.loadLibrary("kstradeapi_wrap");

        } catch(Exception e) {
            e.printStackTrace(System.out);
        }
    }
}
```