文档标识

文档名称	看穿式监管金仕达 API 使用说明
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	2018-12-25	创建	胡闻涛

目录

第一章 看穿式监管设计方案

1.1 终端认证方案

1.1.1 背景条件

每个期货终端软件需要向期货公司申请自己的 appid

每个中继服务器软件需要向期货公司申请自己的 relayappid

1.1.2 appid 对应的授权码分发流程

期货公司确认终端软件集成了正确的数据采集模块后，为该 appid 的终端软件分配授权码。终端软件需要保护好自己的 appid 和授权码，防止被其他软件盗用。

1.1.2.1 直连终端认证流程

对于直连期货公司交易柜台的交易终端软件，期货公司确认终端软件集成了正确的数据采集模块后。给该终端软件（根据 appid）分配一个授权码。这个授权码和 appid 是绑定的，当终端试图登录期货公司交易软件的时候，交易后台会验证该终端是否持有合法的 appid 和授权码。

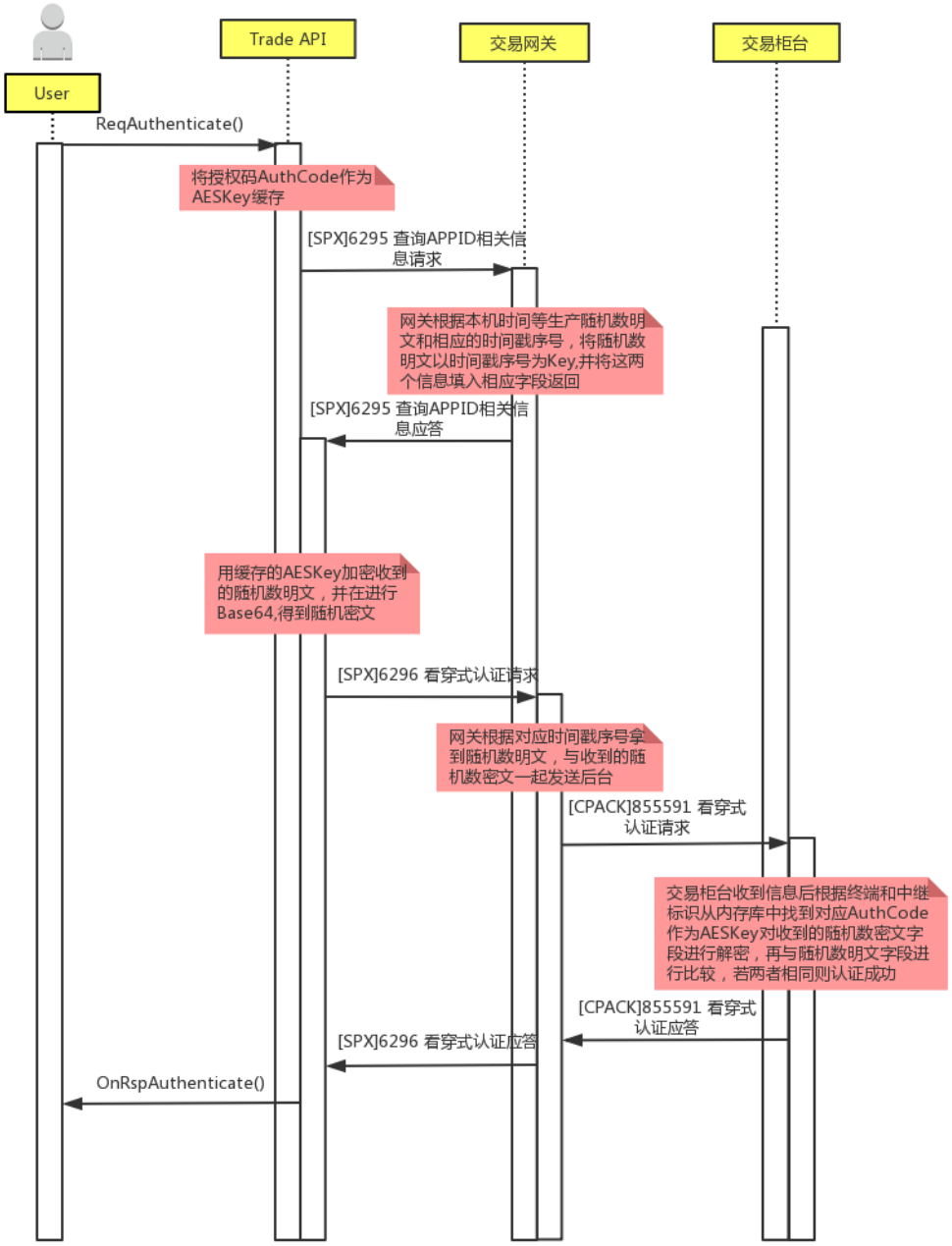
1.1.2.2 中继和中继下属终端的认证流程

对于使用中继服务器连接期货公司交易柜台的终端，期货公司确认终端软件集成了正确的数据采集模块和确认中继可以正常报送终端信息后。期货公司给该终端软件（根据 appid）分配一个授权码，给中继服务器（根据 relayappid） 分配一个授权码。当终端登录中继服务器时，中继服务器负责验证终端的合法性；当中继服务器登录期货公司交易软件的时候，交易后台会验证该中继服务器是否持有合法的 relayappid 和授权码。

1.1.3 登录前认证

当客户直接使用金仕达交易 API 时，客户的终端软件（或者中继服务器）必须存有期货公司分配的授权码，在调用 ReqAuthenticate()时填入 appid 和对应的授权码，交易 API 将该授权码（暂定 32 字节）缓存下来作为加密的 AES_KEY，以对后续信息进行 AES 加密。

认证流程图如下：



步骤：

1. 在终端登录之前，用户通过交易 API 发起终端认证请求 ReqAuthenticate。需要用户填写 appid（relayappid）和授权码（authcode），该授权码只会缓存交易 API 中，不会在网络中直接传输授权码。
2. 交易网关生成随机数作为 app 认证随机明文，交易 API 通过[SPX]6295 查询 appid 相关信息接口获取到该字段。
3. 交易 API 收到[SPX]6295 回调信息后，使用 AES_KEY 加密随机数明文字段，并将其赋值到随机数密文字段，之后发起[SPX]6296 看穿式认证请求。

4. 网关收到[SPX]6296 看穿式认证请求之后，将之前保存的随机数明文赋值到请求里面的随机数明文字段中。将认证消息发送给交易核心
5. 交易核心使用内存数据库中终端信息对应的授权码解密随机数密文字段，并将解密结果与随机数明文比较。如果相同设置当前终端为已经认证。并返回看穿式认证请求成功结果给网关。
6. 网关通过 API 回调，将认证结果返回给用户

对于认证失败的连接，不允许进行登录。

第二章 使用金仕达交易 API 进行终端信息采集

用户可以直接使用金仕达交易 API 进行交易，也可以通过中继服务器间接调用交易 API 进行交易。这就需要将信息采集动态库和金仕达的交易 API 分离开来，因此信息的采集和上报有可能需要分为两步，**API 需要增加新的交易 API 接口（SubmitUserSystemInfo）让用户终端填写上报终端系统信息。**

为了保证上报的信息是来自信息采集动态库，金仕达在数据采集动态库中对已经公钥加密的信息进行二次加密。另外，在直接使用金仕达交易 API 进行交易时（直连模式），用户不需要调用 SubmitUserSystemInfo 交易 API 接口上报系统信息。但是如果用户调用了 SubmitUserSystemInfo 交易 API 接口 ，根据登录之前终端认证通过的 appid 或者 relayappid 可以区分该终端是直连终端、多对一中继服务器还是多对多的中继服务器。对于直连的终端类型调用 SubmitUserSystemInfo 接口，直接返回失败。在调用登录接口时仍会自动采集本机的终端信息。

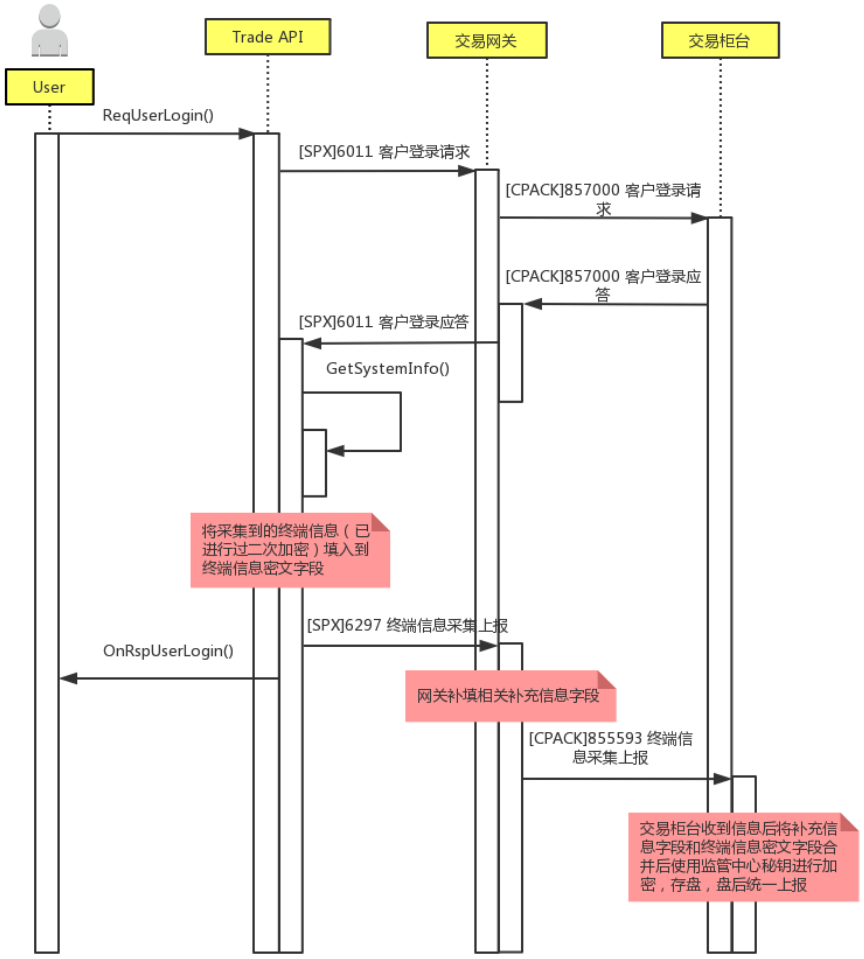
2.1 直接使用金仕达交易 API 直连模式(安卓使用方式详见安卓使用文档)

直接使用交易 API 进行交易时，在登录成功之后，API 会直接调用 KingStar_GetSystemInfo ()采集终端信息，并将信息填入终端采集信息密文字段，通过[SPX]6297 终端信息采集上报接口将采集到的信息送给网关。网关收到交易 API 的[SPX]6297 终端信息采集上报请求后采集客户端的公网 IP，端口号，终端登录时间，客户登录的 session，并将该信息填写到[SPX]6297 终端信息采集上报请求的相应字段内，然后将终端信息采集上

报请求发送给交易核心。

交易柜台收到终端信息采集上报请求请求后，用监控中心发布的公钥对终端信息采集上报请求增加字段的信息进行加密，然后将该信息与其它信息一起写入数据库，盘后统一上报给监控中心。

流程图如下：



具体步骤：

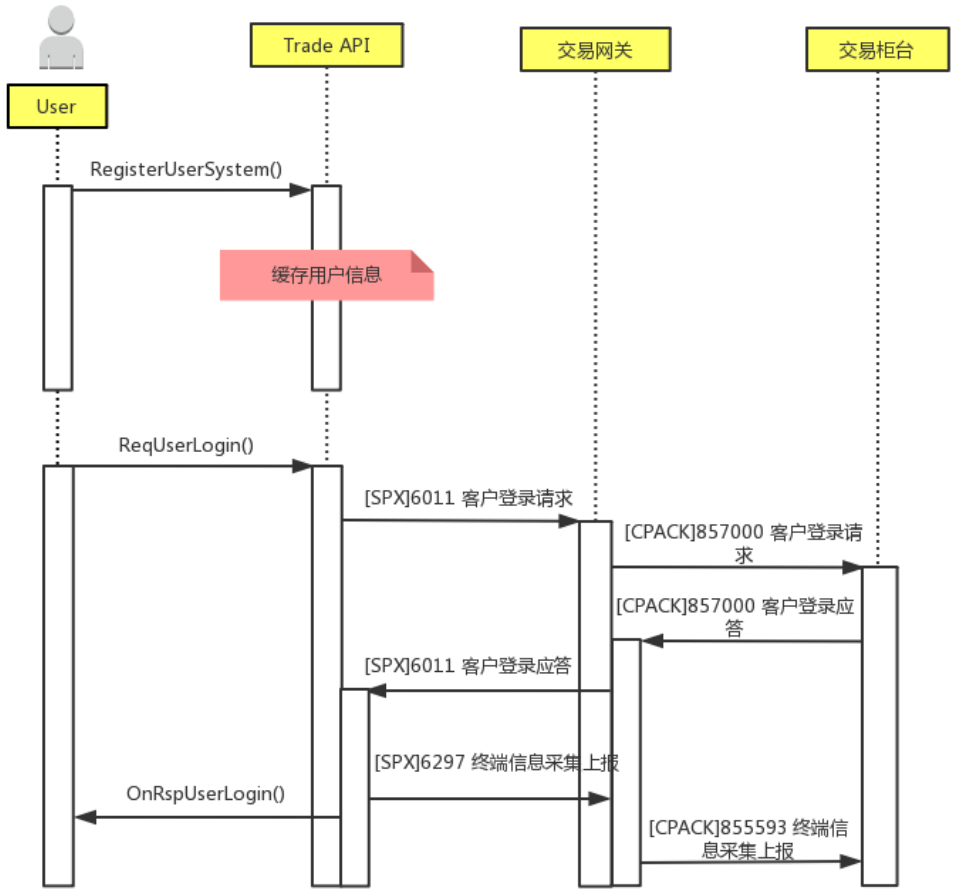
1. 当终端软件通过交易 API 发起登录请求成功后，交易 API 调用 KingStar_GetSystemInfo ()采集终端信息，将该信息填写到终端信息采集上报请求的终端采集信息密文字段中。上报用户系统信息给网关。
2. 网关收到用户系统信息上报后，采集终端的公网 IP，端口号，终端登录时间，客户登录的 session 填入相应字段，缓存该用户的系统信息，每个用户只缓存一条信息。
3. 网关将终端信息采集上报请求发送给交易核心，交易柜台用监控中心的公钥加

密网关补填信息，然后将所有的用户系统信息回写到物理数据库中。

4. 结算系统读取物理数据库中的信息，每日汇总所有的采集信息，将信息报送给保证金监控中心

2.2 使用中继服务器多对多登录模式

流程图如下：

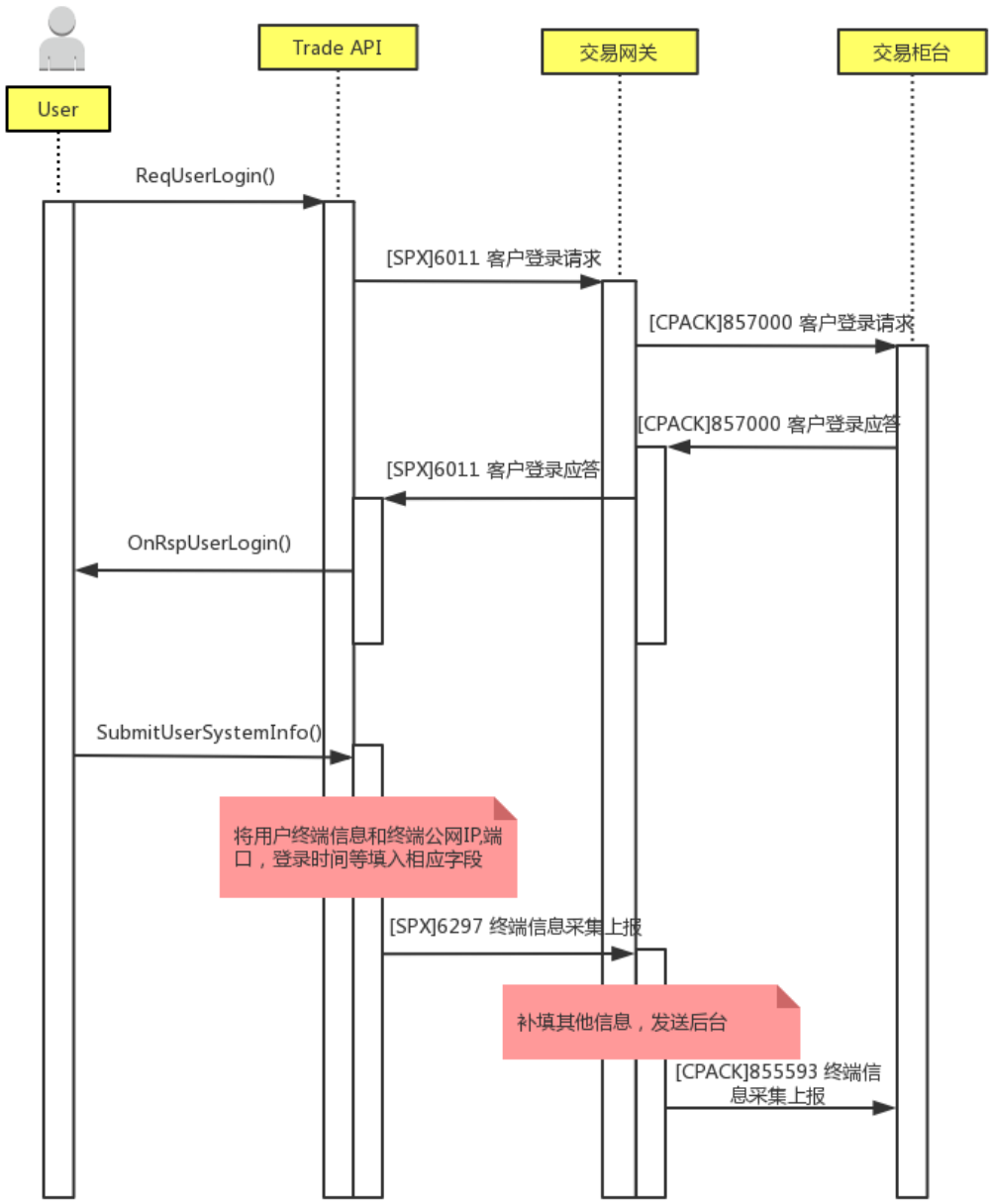


具体步骤：

1. 中继服务器先为用户调用交易 API 的 RegisterUserSystemInfo()接口（填写信息包含客户终端采集的信息、终端的 appid、终端的登录时间和公网 IP、中继服务器的 relayappid 等）。
2. 其余过程与用户直接连时相同。RegisterUserSystemInfo 交易 API 接口只对多对多类型的中继服务器开放调用，其他类型的终端或者中继调用会直接返回失败。

2.3 使用中继服务器多对一登录模式

流程图如下：



具体步骤：

1. 中继服务器先调用交易 API 登录金仕达系统，登录时采集信息的方式与用户直接连接时相同。
2. 中继服务器须采集客户端的信息，然后调用 SubmitUserSystemInfo()上报终端信息（包含客户终端采集的信息、终端的 appid、终端的登录时间和公网 IP、中继服务器的 relayappid 等）。

3. 网关收到报送信息后补填 session 等字段，之后报送给交易柜台。
4. 交易柜台收到信息后用监控中心秘钥对补充信息字段进行加密，再将加密后的信息和终端采集信息写入数据库，结算系统读取数据库，将信息上报监管中心。

第三章 看穿式 API 使用说明

为了实现看穿式监管方案，金仕达会发布两个 API，TraderAPI 和信息采集加密 API。下面是新增或修改的 API 的说明。

3.1 新增 API 使用说明

3.1.1 采集加密 API 说明

```
//客户端信息采集接口
//输出参数：
//@pSystemInfo  出参  空间需要调用者自己分配  至少520个字节
//@nLen  出参  获取到的采集信息的长度
KCC_API int KingStar_GetSystemInfo(char* pSystemInfo, int& nLen);
```

中继类型终端使用该函数获取终端信息数据，现支持windows,linux,android,ios。

3.2 TraderApi

3.2.1 看穿式监管涉及到的 API

```
1.///客户端认证请求
virtual int ReqAuthenticate(CThostFtdcReqAuthenticateField *pReqAuthenticateField, int
nRequestID) = 0;
///客户端认证请求
struct CThostFtdcReqAuthenticateField
{
    ///经纪公司代码
    TThostFtdcBrokerIDType  BrokerID;
    ///用户代码
    TThostFtdcUserIDType    UserID;
    ///用户端产品信息
    TThostFtdcProductInfoType    UserProductInfo;
    ///认证码
    TThostFtdcAuthCodeType AuthCode;
```

```
    ///App代码
    TThostFtdcAppIDType     AppID;   ///需要按照规则定义
};


2.///客户端认证响应
virtual void OnRspAuthenticate(CThostFtdcRspAuthenticateField *pRspAuthenticateField,
CThostFtdcRspInfoField *pRspInfo, int nRequestID, bool bIsLast) {};
///客户端认证响应
struct CThostFtdcRspAuthenticateField
{
    ///经纪公司代码
    TThostFtdcBrokerIDType  BrokerID;
    ///用户代码
    TThostFtdcUserIDType    UserID;
    ///用户端产品信息
    TThostFtdcProductInfoType       UserProductInfo;
    ///App代码
    TThostFtdcAppIDType     AppID;
    ///App类型
    TThostFtdcAppTypeType   AppType;
};
3.///上报用户终端信息，用于中继服务器登录模式
///中继登录后，可以多次调用该接口上报客户信息
virtual int SubmitUserSystemInfo(CThostFtdcUserSystemInfoField *pUserSystemInfo) = 0;

4.///注册用户终端信息，用于中继服务器多连接模式
///需要在终端认证成功后，用户登录前调用该接口
virtual int RegisterUserSystemInfo(CThostFtdcUserSystemInfoField *pUserSystemInfo) = 0;
///用户系统信息
struct CThostFtdcUserSystemInfoField
{
    ///经纪公司代码
    TThostFtdcBrokerIDType    BrokerID;
    ///用户代码
    TThostFtdcUserIDType    UserID;
    ///用户端系统内部信息长度
    TThostFtdcSystemInfoLenType  ClientSystemInfoLen;
    ///用户端系统内部信息
    TThostFtdcClientSystemInfoType      ClientSystemInfo;
    ///用户公网IP
    TThostFtdcIPAddressType ClientPublicIP;
    ///终端IP端口
    TThostFtdcIPPortType    ClientIPPort;
```

```
    ///登录成功时间
    TThostFtdcTimeTypeClientLoginTime;
    ///App代码
    TThostFtdcAppIDType     ClientAppID;
};
```

ClientSystemInfoLen 存储的为加密后的用户终端系统内部信息的长度

ClientSystemInfo 存储的为加密后的用户终端系统内部信息。

ClientPublicIP 存储的为用户终端 IP，由中继服务器采集和填写

ClientLoginTime 存储的为用户登录中继时间，由中继服务器采集和填写

ClientAppid 存储的为用户终端的 appid，由中继服务器采集和填写

3.2.2 客户使用流程

3.2.2.1 直连终端使用流程

```
///API连接后发起认证
virtual void OnFrontConnected()
{
    CThostFtdcReqAuthenticateField    reqAuthenticateField;
    memset(&reqAuthenticateField,0,sizeof(reqAuthenticateField));
    strcpy(reqAuthenticateField.AppID,"ks3");
    strcpy(reqAuthenticateField.AuthCode,"ks3kingstarsoft30.32018120514380");

    m_pUserApi->ReqAuthenticate(&reqAuthenticateField,m_nRequestID++);
}

///认证成功后发起登录
virtual void OnRspAuthenticate(CThostFtdcRspAuthenticateField *pRspAuthenticateField,
    CThostFtdcRspInfoField *pRspInfo, int nRequestID, bool bIsLast)
{
    if(pRspInfo->ErrorID != 0)
    {
        printf("ErrorCode=[%d],ErrorMsg=[%s]\n",pRspInfo->ErrorID,
            pRspInfo->ErrorMsg);
        printf("RequestID=[%d], Chain=[%d]\n", nRequestID, bIsLast);
    }
    else
    {
        CThostFtdcReqUserLoginField reqUserLogin;
        memset(&reqUserLogin, 0, sizeof(reqUserLogin));
        // set BrokerID
```

```
            strcpy(reqUserLogin. BrokerID, m_chBrokerID);

            // set user id
            strcpy(reqUserLogin.UserID, m_chUserID);

            // set password
            strcpy(reqUserLogin.Password, m_chPassword);

            // send the login request
            m_pUserApi->ReqUserLogin(&reqUserLogin, m_nRequestID++ );
        }
}
```

3.2.2.2 多对多中继终端使用流程

```
///中继收到终端登录请求后发起认证
virtual void OnFrontConnected()
{
    CThostFtdcReqAuthenticateField    reqAuthenticateField;
    memset(&reqAuthenticateField,0,sizeof(reqAuthenticateField));
    strcpy(reqAuthenticateField.AppID,"ks3");
    strcpy(reqAuthenticateField.AuthCode,"ks3kingstarsoft30.32018120514380");

    m_pUserApi->ReqAuthenticate(&reqAuthenticateField,m_nRequestID++);
}

///认证成功后发起登录
virtual void OnRspAuthenticate(CThostFtdcRspAuthenticateField
    *pRspAuthenticateField,CThostFtdcRspInfoField *pRspInfo, int nRequestID, bool bIsLast)
{
    if(pRspInfo->ErrorID != 0)
    {
        printf("ErrorCode=[%d],ErrorMsg=[%s]\n",pRspInfo->ErrorID,
            pRspInfo->ErrorMsg);
        printf("RequestID=[%d], Chain=[%d]\n", nRequestID, bIsLast);
    }
    else
    {
        RegSystemInfo();
        CThostFtdcReqUserLoginField reqUserLogin;
        memset(&reqUserLogin, 0, sizeof(reqUserLogin));
        // set BrokerID
        strcpy(reqUserLogin. BrokerID, m_chBrokerID);
```

```
            // set user id
            strcpy(reqUserLogin.UserID, m_chUserID);

            // set password
            strcpy(reqUserLogin.Password, m_chPassword);

            // send the login request
            m_pUserApi->ReqUserLogin(&reqUserLogin, m_nRequestID++ );
        }
    }

    ///终端信息上传
    void RegSystemInfo()
    {
        char pSystemInfo[344];
        int len;
        ////将从终端得到的信息赋值给下面结构体

        CThostFtdcUserSystemInfoField UserSystemInfo;
        memset(&UserSystemInfo, 0, sizeof(UserSystemInfo));
        strcpy(UserSystemInfo.BrokerID, m_chBrokerID);
        strcpy(UserSystemInfo.UserID, m_chUserID);
        //strcpy(UserSystemInfo.ClientSystemInfo, pSystemInfo);
        memcpy(UserSystemInfo.ClientSystemInfo, pSystemInfo, len);
        UserSystemInfo.ClientSystemInfoLen = len;
        strcpy(UserSystemInfo.ClientPublicIP, "198.4.4.124");
        UserSystemInfo.ClientIPPort = 65535;
        strcpy(UserSystemInfo.ClientLoginTime, "11:28:28");
        strcpy(UserSystemInfo.ClientAppID, "Q7");
        int ret = m_pUserApi->RegisterUserSystemInfo(&UserSystemInfo);
        cout << "retd = " << ret << endl;
    }
```

3.2.2.3 多对一中继终端使用流程

```
    ///中继收到终端登录请求后发起认证
    virtual void OnFrontConnected()
    {
        CThostFtdcReqAuthenticateField    reqAuthenticateField;
        memset(&reqAuthenticateField,0,sizeof(reqAuthenticateField));
        strcpy(reqAuthenticateField.AppID,"ks3");
        strcpy(reqAuthenticateField.AuthCode,"ks3kingstarsoft30.32018120514380");

        m_pUserApi->ReqAuthenticate(&reqAuthenticateField,m_nRequestID++);
```

```
}

///认证成功后发起登录
virtual void OnRspAuthenticate(CThostFtdcRspAuthenticateField
    *pRspAuthenticateField,CThostFtdcRspInfoField *pRspInfo, int nRequestID, bool bIsLast)
{
    if(pRspInfo->ErrorID != 0)
    {
        printf("ErrorCode=[%d],ErrorMsg=[%s]\n",pRspInfo->ErrorID,
            pRspInfo->ErrorMsg);
        printf("RequestID=[%d], Chain=[%d]\n", nRequestID, bIsLast);
    }
    else
    {
        CThostFtdcReqUserLoginField reqUserLogin;
        memset(&reqUserLogin, 0, sizeof(reqUserLogin));
        // set BrokerID
        strcpy(reqUserLogin. BrokerID, m_chBrokerID);

        // set user id
        strcpy(reqUserLogin.UserID, m_chUserID);

        // set password
        strcpy(reqUserLogin.Password, m_chPassword);

        // send the login request
        m_pUserApi->ReqUserLogin(&reqUserLogin, m_nRequestID++ );
    }
}
////中继登录金仕达之后
///终端登录中继时，  中继发起终端信息的上报
void SubSystemInfo()
{
    char pSystemInfo[344];
    int len;
    KingStar_GetSystemInfo (pSystemInfo, len);
    //cout << "KingStar_GetSystemInfo" << endl;
    cout << "SubSystemInfo 1" << endl;
    CThostFtdcUserSystemInfoField field;

    memset(&field, 0, sizeof(field));
    strcpy(field.BrokerID, "8000");
    strcpy(field.UserID, "001888");
```

```
    //strcpy(field.ClientSystemInfo, pSystemInfo);
    memcpy(field.ClientSystemInfo, pSystemInfo, len);
    field.ClientSystemInfoLen = len;
    strcpy(field.ClientPublicIP, "198.114.114.124");
    field.ClientIPPort = 65535;
    strcpy(field.ClientLoginTime, "11:28:28");
    strcpy(field.ClientAppID, "Q7");

    int retx = m_pUserApi->SubmitUserSystemInfo(&field);
    cout << "ret = " << retx << endl;

    CThostFtdcUserSystemInfoField field1;

    memset(&field1, 0, sizeof(field1));
    strcpy(field1.BrokerID, "8000");
    strcpy(field1.UserID, "001888");

    //strcpy(field.ClientSystemInfo, pSystemInfo);
    memcpy(field1.ClientSystemInfo, pSystemInfo, len);
    field1.ClientSystemInfoLen = len;
    strcpy(field1.ClientPublicIP, "198.4.4.124");
    field1.ClientIPPort = 65532;
    strcpy(field1.ClientLoginTime, "11:28:28");
    strcpy(field1.ClientAppID, "Q7");

    m_pUserApi->SubmitUserSystemInfo(&field1);
}
```