**中软万维证券交易网关说明**

# 交易协议说明

顶点柜台

监控连接数

监控代理，ssl short read， 长连接

版本管理

## 请求

cssweb\_SysNo系统编号

cssweb\_BusiType业务类型

cssweb\_SysVer客户端类型说明

cssweb\_appversion

cssweb\_FuncId业务功能号

cssweb\_account登录标识

cssweb\_hardinfo硬件特征码

cssweb\_route恒生路由

## 返回

Cssweb\_code返回码

Cssweb\_msg 返回消息

Cssweb\_gwInfo网关信息

Cssweb\_counter柜台信息

# 自定义业务功能说明

## 查询状态数据

## 心跳功能

cssweb\_funcid=999999

# 监控功能

# 本地日志说明

## 通用日志

证券网关日志.log

## 业务日志log\业务

系统编号\客户端类型\业务类型\日期\客户号\_功能号\_success.log

系统编号\客户端类型\业务类型\日期\客户号\_功能号\_error.log

Gtja\_yht\iphone\信用交易\日志\12345678\_400000\_success.log

Gtja\_yht\iphone\信用交易\日志\12345678\_400000\_error.log

## 柜台日志log\柜台

## 网络层日志log\网络

# 日志格式说明

需要记录IP和端口

# 错误代码及说明

1000缺少参数或参数数据错误

1001

# 客户端版本管理

# 安全控制

## 通信安全

协议采用TLS1.2，禁止SSL3.0

## 访问控制

同一账户同时登录次数超过规定限制。

同一ip地址单位时间内超过规定次数。

# 需要解决的问题

## 柜台连接中断如何判断？

## 如何控制连接池最大大小？

采用线程安全vector保存

不再使用队列，

## 日志去掉copy操作

心跳功能不记录日志

## 压缩功能移到发送线程

## 请求的字符集编码问题

前端必须传入GBK编码

## Dogskin引起的崩溃

# 暂末实现的功能

## 分布式日志

## 统一的交易协议

## 金证柜台有用的代码

/\*

if (type == "flash")

{

if (funcid == "410302")

{

//修改密码

trdpwd = GetOtherPWD(isSafe, value);

std::string encrypt\_pwd = g\_Encrypt.EncryptPWD(trdpwd);

KCBPCLI\_SetValue(m\_pConn->handle, (char\*)key.c\_str(), (char\*)encrypt\_pwd.c\_str());

}

else

{

// 需要先解密

trdpwd = GetTradePWD(isSafe, value);

std::string encrypt\_pwd = g\_Encrypt.EncryptPWD(trdpwd);

KCBPCLI\_SetValue(m\_pConn->handle, (char\*)key.c\_str(), (char\*)encrypt\_pwd.c\_str());

OutputDebugString("送到柜台的交易密码");

OutputDebugString(encrypt\_pwd.c\_str());

OutputDebugString("\n");

}

}

else

{

}

continue;

\*/

/\*

char szEncryptPwd[33];

memset(szEncryptPwd, 0, sizeof(szEncryptPwd));

std::string enc\_key = reqmap["custid"];

if (type == "web")

{

if (funcid == "410301")

{

KDEncode(KDCOMPLEX\_ENCODE, (unsigned char\*)value.c\_str(), value.length(), (unsigned char \*)szEncryptPwd, sizeof(szEncryptPwd)-1, "410301", 6);

}

else

{

KDEncode(KDCOMPLEX\_ENCODE, (unsigned char\*)value.c\_str(), value.length(), (unsigned char \*)szEncryptPwd, sizeof(szEncryptPwd)-1, (void\*)enc\_key.c\_str(), enc\_key.length());

}

}

else

{

std::string tradepwd\_plain = GetTradePWD(isSafe, value);

if (funcid == "410301")

{

KDEncode(KDCOMPLEX\_ENCODE, (unsigned char\*)tradepwd\_plain.c\_str(), tradepwd\_plain.length(), (unsigned char \*)szEncryptPwd, sizeof(szEncryptPwd)-1, "410301", 6);

}

else

{

KDEncode(KDCOMPLEX\_ENCODE, (unsigned char\*)tradepwd\_plain.c\_str(), tradepwd\_plain.length(), (unsigned char \*)szEncryptPwd, sizeof(szEncryptPwd)-1, (void\*)enc\_key.c\_str(), enc\_key.length());

}

}

std::string encrypt\_pwd = g\_Encrypt.EncryptPWD("456456");

//KCBPCLI\_SetValue(m\_pConn->handle, (char\*)key.c\_str(), szEncryptPwd);

//KCBPCLI\_SetValue(m\_pConn->handle, (char\*)key.c\_str(), "");

TRACE("交易密码...........................%s\n", szEncryptPwd);

//7Ketq+ncv0rD+iX0Sgq8/A==

\*//\*

if (type == "flash")

{

//解密前端

newpwd = GetOtherPWD(isSafe, value);

//用柜台加密

std::string encrypt\_pwd = g\_Encrypt.EncryptPWD(newpwd);

KCBPCLI\_SetValue(m\_pConn->handle, (char\*)key.c\_str(), (char\*)encrypt\_pwd.c\_str());

// 标准登录，修改密码成功，需要加密返回给前端，下次传入加密后的新密码

// 加强登录, 修改密码成功，前端自己保留新密码

std::string aeskey = "23dpasd23d-23l;df9302hzv/3lvjh\*5";

char cipher[50];

memset(cipher, 0x00, sizeof(cipher));

int outlen;

bool bRet = g\_MyBotan.AESEncrypt("AES-256/ECB/PKCS7", aeskey, newpwd, cipher, &outlen);

//char \* out = aes\_encrypt\_encode(aeskey.c\_str(), newpwd.c\_str());

if (bRet)

{

newpwd\_enc = cipher;

boost::algorithm::replace\_all(newpwd\_enc, "=", "$");

//free(out);

}

}

else

{

newpwd = value;

//用柜台加密

std::string encrypt\_pwd = g\_Encrypt.EncryptPWD(newpwd);

KCBPCLI\_SetValue(m\_pConn->handle, (char\*)key.c\_str(), (char\*)encrypt\_pwd.c\_str());

}

continue;

\*/

/\*

char szEncryptPwd[33];

memset(szEncryptPwd, 0, sizeof(szEncryptPwd));

std::string enc\_key = reqmap["custid"];

if (type == "web")

{

KDEncode(KDCOMPLEX\_ENCODE, (unsigned char\*)value.c\_str(), value.length(), (unsigned char \*)szEncryptPwd, sizeof(szEncryptPwd)-1, (void\*)enc\_key.c\_str(), enc\_key.length());

}

else

{

std::string tradepwd\_plain = GetTradePWD(isSafe, value);

KDEncode(KDCOMPLEX\_ENCODE, (unsigned char\*)tradepwd\_plain.c\_str(), tradepwd\_plain.length(), (unsigned char \*)szEncryptPwd, sizeof(szEncryptPwd)-1, (void\*)enc\_key.c\_str(), enc\_key.length());

}

KCBPCLI\_SetValue(m\_pConn->handle, (char\*)key.c\_str(), szEncryptPwd);

continue;

\*//\*

if (type == "flash")

{

//解密前端

std::string sPwdPlain = GetOtherPWD(isSafe, value);

//用柜台加密

std::string encrypt\_pwd = g\_Encrypt.EncryptPWD(sPwdPlain);

KCBPCLI\_SetValue(m\_pConn->handle, (char\*)key.c\_str(), (char\*)encrypt\_pwd.c\_str());

}

else

{

//用柜台加密

}

continue;

\*/

/\*

int KDEncode(int nEncode\_Level,

unsigned char \*pSrcData, int nSrcDataLen,

unsigned char \*pDestData, int nDestDataBufLen,

void \*pKey, int nKeyLen);

char szEncryptPwd[33];

memset(szEncryptPwd, 0, sizeof(szEncryptPwd));

std::string enc\_key = reqmap["fundid"];

if (type == "web")

{

KDEncode(KDCOMPLEX\_ENCODE, (unsigned char\*)value.c\_str(), value.length(), (unsigned char \*)szEncryptPwd, sizeof(szEncryptPwd)-1, (void\*)enc\_key.c\_str(), enc\_key.length());

}

else

{

std::string tradepwd\_plain = GetTradePWD(isSafe, value);

KDEncode(KDCOMPLEX\_ENCODE, (unsigned char\*)tradepwd\_plain.c\_str(), tradepwd\_plain.length(), (unsigned char \*)szEncryptPwd, sizeof(szEncryptPwd)-1, (void\*)enc\_key.c\_str(), enc\_key.length());

}

\*/// else if(key == "ddasigndata")

// {

// if (type == "flash")

// {

//flash交易专用

// boost::algorithm::replace\_all(value, "$", "=");

// KCBPCLI\_SetValue(m\_pConn->handle, (char\*)key.c\_str(), (char\*)value.c\_str());

// }

// continue;

// }/gFileLog::instance().Log(funcid + " KCBPCLI\_SQLExecute");

//if (nRet == 2003 || nRet == 2004 || nRet == 2055 || nRet == 2054)

//重连

//int nErrCode = 0;

//char szErrMsg[2048] = {0};

//KCBPCLI\_GetErr(m\_pConn->handle, &nErrCode, szErrMsg);

//if (nErrCode == 0)

//{

// RetNoRecordRes(response);

// goto FINISH;

//}