**上海黄金交易所**

**核心交易系统接口API**

**说明书**

上海黄金交易所国际中心

2016年02月

**版本历史**

|  |  |  |  |
| --- | --- | --- | --- |
| 版本号 | 修改日期 | 描述 | 修订人 |
| V0.1 | 2015-10-10 | 初始版本编写 | 核心交易组 |
| V0.2 | 2015-10-12 | 内容编写和更新 | 核心交易组 |
| V0.3 | 2015-10-15 | 修改系统概述文字和图等, | 核心交易组 |
| V0.4 | 2015-10-16 | 修正回调函数接口，  修正api初始化函数接口等 | 核心交易组 |
| V1.0 | 2015-10-31 | 正式发布 | 核心交易组 |
| V1.4 | 2015-12-31 | 配合Beta版本发布  分离安全模块 | 核心交易组 |
| V1.5 | 2016-01-08 | 根据会员（中行开发人员）反馈的问题，修正一下错误说明；添加了一些详细的说明。 | 核心交易组—程论 |
| V1.6 | 2016-02-01 | 文件清单分离为linux和windows平台；  回调函数返回值全部设置为void;  所有api函数都需要传入api\_t对象;  更新和优化demo代码，使demo代码支持linux和windows平台; | 核心交易组—程论 |
| V1.7 | 2016-02-02 | 增加合并编译GTP API 和2.5代API的方法章节 | 核心交易组—程论 |
| V1.7.1 | 2016-04-15 | 1. 增加系统概述中api的在windows和linux下的编译环境信息 2. 增加连接管理简介中：多接入服务器连接管理描述 3. 更新文件清单 4. 增加两个api函数，api\_set\_new\_endpoint   api\_is\_running   1. 修改demo程序使用 api\_is\_running 2. 修改api\_safe模块的并发使用提示 3. 增加附录C Linux下安全模块依赖so安装 | 核心交易组—程论 |
| V1.7.2 | 2016-05-18 | 1. 补充编写api\_disconnect接口说明，api头文件中早已存在，文档中未说明，居然这么久了，也没会员提到这个问题。 2. 更新windows的文件清单 | 核心交易组—程论 |
| V1.7.3 | 2016-05-24 | 1.    增加两个获取api信息的函数 api\_get\_ip && api\_get\_port  2.    修正api连接成功后，挂掉的bug | 核心交易组—程论 |
| V1.7.4 | 2016-08-03 | 增加 api\_stop函数 | 核心交易组—程论 |
| V1.7.5 | 2016-09-23 | 1. 增加Windows Server2008 支持的说明 2. 修改 api\_send\_msg, api\_send\_msg\_no\_enc,   api\_recv\_msg  三个函数中，表示消息长度的参数类型为 unsigned int   1. 修改sge\_safe\_encrypt函数的cap参数说明：当明文长度为8的倍数时： cap必须大于等于len+8字节   当明文长度不为8的倍数时： cap必须大于等于 len 8字节向上对齐，再加8字节： (8\*(len/8) + 8) + 8; | 核心交易组-程论 |

目录

[1. 简介 1](#_Toc462407727)

[2. 系统概述 1](#_Toc462407728)

[**2.1.** **逻辑结构** 1](#_Toc462407729)

[**2.2.** **功能说明** 2](#_Toc462407730)

[**2.3.** **连接管理简介** 2](#_Toc462407731)

[**2.4.** **文件清单** 3](#_Toc462407732)

[**1.** **Linux 平台API文件清单** 4](#_Toc462407733)

[**2.** **Windows平台API 文件清单** 5](#_Toc462407734)

[3. 开发接口 5](#_Toc462407735)

[**3.1.** **统一规则说明** 5](#_Toc462407736)

[**3.2.** **API系统类** 5](#_Toc462407737)

[**3.2.1.** **api初始化-api\_init** 5](#_Toc462407738)

[**3.2.2.** **等待api线程结束-api\_wait** 6](#_Toc462407739)

[**3.2.3.** **释放api资源-api\_free** 6](#_Toc462407740)

[**3.2.4.** **获取api运行状态- api\_is\_running** 6](#_Toc462407741)

[**3.2.5.** **设置新的连接点-api\_set\_new\_endpoint** 7](#_Toc462407742)

[**3.2.6.** **停止api -api\_stop** 7](#_Toc462407743)

[**3.2.7.** **断开api连接-** **api\_disconnect** 7](#_Toc462407744)

[**3.2.8.** 网络连接成功时的回调函数-**api\_cb\_connected** 7](#_Toc462407745)

[**3.2.9.** 网络连接断开时的回调函数**-api\_cb\_disconnected** 8](#_Toc462407746)

[**3.2.10.** api内部写日志（开始）**-api\_thread\_begin** 8](#_Toc462407747)

[**3.2.11.** api内部写日志（结束）**-api\_thread\_end** 8](#_Toc462407748)

[**3.2.12.** 获取api的端口**-api\_get\_port** 8](#_Toc462407749)

[**3.2.13.** 获取api的IP地址**-api\_get\_ip** 9](#_Toc462407750)

[**3.2.14.** 获取api的版本号**-api\_get\_version** 9](#_Toc462407751)

[**3.3.** **对话流报文** 9](#_Toc462407752)

[**3.3.1.** **发送登录请求- api\_send\_msg\_no\_enc** 9](#_Toc462407753)

[**3.3.2.** **发送加密消息请求- api\_send\_msg** 10](#_Toc462407754)

[**3.3.3.** **请求应答及通知流报文- api\_recv\_msg\_** 10](#_Toc462407755)

[4. API开发示例 10](#_Toc462407756)

[5. 安全认证和加密 11](#_Toc462407757)

[**5.1.** **认证原理** 11](#_Toc462407758)

[**5.2.** **证书私钥的文件** 11](#_Toc462407759)

[**5.3.** **安全模块接口** 11](#_Toc462407760)

[5.3.1初始化接口-api\_safe\_init 11](#_Toc462407761)

[5.3.2结束接口- api\_safe\_deinit 12](#_Toc462407762)

[5.3.3创建客户端安全信息接口 12](#_Toc462407763)

[5.3.4验证服务器端安全信息接口- api\_safe\_check\_server\_info 12](#_Toc462407764)

[5.3.5对称加密接口- api\_safe\_encrypt 13](#_Toc462407765)

[5.3.6对称解密接口- api\_safe\_decrypt 13](#_Toc462407766)

[5.3.7获取版本接口- api\_safe\_get\_version 13](#_Toc462407767)

[5.4 api与safe接口调用关系简述 14](#_Toc462407768)

[6. API错误码对照表 16](#_Toc462407769)

[7. 附录A： Demo程序演示 16](#_Toc462407770)

[8. 附录 B 合并编译GTP API 和2.5代API方法 20](#_Toc462407771)

[**8.1.** **Windows版本 C++编译过程** 20](#_Toc462407772)

[**1.** **编译准备** 21](#_Toc462407773)

[**2.** **增加2.5代API的头文件** 21](#_Toc462407774)

[**3.** **增加2.5代API 链接信息（路径和静态库文件）** 22](#_Toc462407775)

[**8.2.** **Linux版本 C++编译过程** 23](#_Toc462407776)

[9. 附录 C Linux版本 安全模块依赖CA库部署 23](#_Toc462407777)

1. **简介**

API为会员二级系统与黄金交易所核心交易系统间交互，而开发的统一功能和数据接口。

本接口说明书的编写的目的是说明API在整个系统中的位置，API所提供的功能及使用方法。

本文档目标读者：会员二级系统开发人员，黄马甲，红马甲开发人员，测试人员。

**该文档主要针对基于GTP协议的API使用；**

**2.5代XTP协议API使用手册请各会员参考已有的API使用手册；**

**涵盖交易API， ETF API，保证金API，账户卡API和行情API；**

**五个API中GTP协议模块使用方式相同。**

**注意：各位会员开发人员请详细阅读，文档最后的demo示例代码，该示例将api模块和安全模块结合，有助于理解api和安全模块的使用。**

1. **系统概述**

会员API支持Linux/Windows系统

Windows平台 X86-32bit

编译器：VS2010, 其中pthread(VS2010)

**操作系统： Windows-7 , Windows Server 2008及以上**

Linux 平台 X86-64bit ，

编译器：gcc (GCC) 4.4.7 20120313 (Red Hat 4.4.7-3)，

操作系统Red Hat Enterprise Linux Server release 6.4 (Santiago)

* 1. **逻辑结构**



* 1. **功能说明**

API主要为系统提供应用接口，借助API，会员二级系统可以将包含数据项的数据域传给核心交易系统前置机，也可以接收来自核心交易系统前置机的响应数据。

API内部有多个线程：

1个业务处理线程：

用于处理用户注册的回调报文处理函数。

1个管理线程

用于管理读线程和写线程以及业务报文的关系。

1个读线程

读取socket中的报文信息

1个写线程

写入socket的报文信息

API只支持单进程。

* 1. **连接管理简介**

API会自动重连

API如果接收超时，会主动断开连接；

API如果长时间没发送心跳，也会被交易所前置断开；

开发人员根据需求可以主动断开连接（调用API函数），但是只是闪断，断开后，API自动重连。

开发人员如果需要完全断开连接，请关闭进程。

**多接入服务器连接管理：**

交易所主板提供两台接入服务器，当会员由于某些原因无法连接其中一台时，可以通过api的对应接口设置新的IP和端口后，主动断开连接。API会自动重新连接新的接入服务器。

* 1. **文件清单**
     + 1. **Linux 平台API文件清单**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **序号** | **目录名** | **文件名** | **类型** | **描述** | **平台** |
| 1 | cert\_file | RootCert.der | 安全证书 | 证书文件 | Linux |
| 2 | server.der | 安全证书 | 证书文件 |
| 3 | UserCert.der | 安全证书 | 证书文件 |
| 4 | UserKey.key | 安全证书 | 证书文件 |
| 5 | include | api.h | 头文件 | API 相关函数 |
| 6 | sge\_api\_safe\_interface.h | 头文件 | API安全模块相关函数 |
| 7 | lib | libapi\_safe.so | 库文件 | API相关库 |
| 8 | libapi.so | 库文件 | API相关库 |
| 9 | libeasy.so | 库文件 | API安全相关库 |
| 10 | ca\_cert\_lib/ | 目录 | 安全相关目录 |

* + - 1. **Windows平台API 文件清单**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **序号** | **目录名** | **文件名** | **类型** | **描述** | **平台** |
| 1 | cert\_file | RootCert.der | 安全证书 | 证书文件 | Windows |
| 2 | server.der | 安全证书 | 证书文件 |
| 3 | UserCert.der | 安全证书 | 证书文件 |
| 4 | UserKey.key | 安全证书 | 证书文件 |
| 5 | include | api.h | 头文件 | API 相关函数 |
| 6 | sge\_api\_safe\_interface.h | 头文件 | API安全模块相关函数 |
| 7 | lib | libapi.dll | 动态库文件 | API相关库 |
| 8 | libapi.lib | 链接文件 | API相关库 |
| 9 | libEasy.dll | 动态库文件 | API相关库 |
| 11 | libapi\_safe.dll | 动态库文件 | API安全相关库 |
| 12 | libapi\_safe.lib | 链接文件 | API安全相关库 |
| 13 | msvcp100d.dll | 库文件 | API相关库 |
| 14 | msvcr100.dll | 库文件 | API相关库 |
|  | msvcr100d.dll | 库文件 | API相关库 |
| 15 | pthreadVC2.dll | 库文件 | API相关库 |
| 16 | pthreadVC2.lib | 链接文件 | API相关库 |
| 17 | SafeEngine.dll | 库文件 | API安全相关库 |
| 18 | SE\_Crypt.dll | 库文件 | API安全相关库 |

1. **开发接口**

api.h定义了所有功能接口，其中有发送GTP报文和接收报文的功能接口。

* 1. **统一规则说明**

1. 字符串的左右不允许出现空格;

* 1. **API系统类**
     1. **api初始化-api\_init**

Api初始化。成功后会启动api所需要使用的读、写和处理线程。

**函数原型**

api\_t \* api\_init(const char \* ip, int port, api\_cb\_t \* cb, void \* ctx)

**参数：**

ip 输入参数，前置ip地址

port 输入参数，前置端口

cb 输入参数，接收处理函数，，如上三个回调函数，api会主动传入

类型定义为：

typedef struct {

void (\* api\_recv\_msg\_)(const char \* msg, int type, unsigned short con\_len, void \* ctx);

void (\* api\_cb\_connected\_)(api\_connection\_state\_t \* cs, void \* ctx);

void (\* api\_cb\_disconnected\_)(api\_connection\_state\_t \* cs, void \* ctx);

} api\_cb\_t;

api\_recv\_msg\_ 接收消息后的回调

api\_cb\_connected\_ 连接成功后的回调

api\_cb\_disconnected\_ 断开连接后的回调

详细说明参考后续章节对应的说明

ctx 输入参数，上下文参数，在api\_init传入。

**返回值：**

api\_t\*: 成功

NULL: 失败

* + 1. **等待api线程结束-api\_wait**

api启动成功后，等待api线程的结束。

**函数原型**

void api\_wait(api\_t \* api)

**参数：**

api 输入参数，api对象指针（以下函数的api\_t\*类型的参数同此解释。）

**返回值：**

无

* + 1. **释放api资源-api\_free**

api结束后释放其所分配的资源。

**函数原型**

void api\_free(api\_t \* api)

**参数：**

api 输入参数，api对象指针

**返回值：**

无

* + 1. **获取api运行状态- api\_is\_running**

获取api的运行状态。

**函数原型**

int api\_is\_running(api\_t \* api)

**参数：**

api 输入参数，api对象指针

**返回值：**

0: api已经处于退出过程中

非0：api还在运行过程中

* + 1. **设置新的连接点-api\_set\_new\_endpoint**

设置api下次重连时的连接点信息（IP+Port）

**函数原型**

int api\_set\_new\_endpoint(api\_t \* api, const char \* remote\_ip , int remote\_port);

**参数：**

api 输入参数，api对象指针

remote\_ip 输入参数，连接点 IP 地址

port 输入参数，连接点端口

**返回值：**

0: 成功

非0: 失败

* + 1. **停止api -api\_stop**

停止api， 可以应用于自动退出api;

**函数原型**

void api\_stop(api\_t \* api)

**参数：**

api 输入参数，api对象指针

**返回值：**

无

* + 1. **断开api连接-** **api\_disconnect**

闪断连接，api会自动连接前置

**函数原型**

int api\_disconnect(api\_t \* api)

**参数：**

api 输入参数，api对象指针

**返回值：**

0: 成功

非0: 失败

* + 1. 网络连接成功时的回调函数-**api\_cb\_connected**

api连接前置成功后会回调这一函数。

**函数原型**

void (\* api\_cb\_connected\_)(api\_connection\_state\_t \* cs, void \* ctx)

**参数：**

cs 输出参数，连接的信息和状态

ctx 输出参数，调用api\_init时的第四个参数ctx

**返回值：**

无

* + 1. 网络连接断开时的回调函数**-api\_cb\_disconnected**

在api与前置的连接断开后会回调这一函数。

**函数原型**

void (\* api\_cb\_disconnected\_)(api\_connection\_state\_t \* cs, void \* ctx)

**参数：**

cs 输出参数，连接的信息和状态

ctx 输出参数，调用api\_init时的第四个参数ctx

**返回值：**

无

* + 1. api内部写日志（开始）**-api\_thread\_begin**

需要api写日志时，会员系统在新建的线程开始处调用。

**函数原型**

int api\_thread\_begin(api\_t \* api, const char \* tname)

**参数：**

api 输入参数

tname 输入参数，日志文件名的一部分

**返回值：**

0: 成功

-1：输入参数错误

* + 1. api内部写日志（结束）**-api\_thread\_end**

需要api写日志时，会员系统在新建的线程结束处调用。

**函数原型**

int api\_thread\_end(api\_t \* api)

**参数：**

api 输入参数

**返回值：**

无

* + 1. 获取api的端口**-api\_get\_port**

获取api的端口

**函数原型**

int api\_get\_port(api\_t \* api);

**参数：**

api 输入参数

**返回值：**

>=0 : 当前设置端口号

-1: 参数为NULL

* + 1. 获取api的IP地址**-api\_get\_ip**

获取api当前的ip地址

**函数原型**

char \* api\_get\_version(api\_t \* api)

**参数：**

api 输入参数

**返回值：**

char\*: 成功

NULL: 失败

* + 1. 获取api的版本号**-api\_get\_version**

获取api的版本号

**函数原型**

char \* api\_get\_version(api\_t \* api)

**参数：**

api 输入参数

**返回值：**

char\*: 成功

NULL: 失败

* 1. **对话流报文**

对话流报文即请求应答类报文，会员系统调用API发送一个请求报文到会员前置，会员前置返回一个应答给会员系统，表示收到请求且处理后的结果。如交易员登录，报单等。

* + 1. **发送登录请求- api\_send\_msg\_no\_enc**

会员系统通过这一接口发送登录请求至前置，登陆请求除安全信息外，都不加密。

**函数原型**

int api\_send\_msg\_no\_enc(api\_t \* api, const char \* plain\_gtp\_msg, unsigned int len)

**参数：**

api 输入参数

plain\_gtp\_msg 输入参数，登录请求报文

len 输入参数，登录请求报文长度

**返回值：**

0 : 成功

-1: 参数为NULL

-2: 内存分配失败

-4：尚未与前置连接

* + 1. **发送加密消息请求- api\_send\_msg**

获取api当前的ip地址

**函数原型**

int api\_send\_msg(api\_t \* api, const char \* encrypted\_msg, unsigned int len);

**参数：**

api 输入参数

encrypted\_msg 接收消息

len 消息实际长度

**返回值：**

0 : 成功

-1: 参数为NULL

-2: 内存分配失败

-4：尚未与前置连接

* + 1. **请求应答及通知流报文- api\_recv\_msg\_**

在接收到前置返回的应答或者推送的通知流报文时，会调用此函数进行相应的处理。

**函数原型**

void (\* api\_recv\_msg\_)(const char \* msg, int type, unsigned int con\_len, void \* ctx)

**参数：**

msg, 输出参数，前置返回的报文

type, 输出参数，前置返回的报文的类型 ，标识消息是否加密，

取值：

#define GTP\_TYPE\_STRING 1

#define GTP\_TYPE\_ENCRYPT 4

con\_len, 输出参数，前置返回的报文的长度

ctx, 输出参数，调用api\_init时的第四个参数ctx

**返回值：**

无

1. **API开发示例**

参考附录A Demo程序演示

1. **安全认证和加密**
   1. **认证原理**

API在每次登录时通过根证书和用户私钥建立一个加密会话，通过会话生成一个随机数R1，用服务器证书加密后随登录请求发送到会员前置。随后API等待并接收会员前置的应答报文，通过加密会话和用户证书对应答报文的安全认证部分数据进行解密，并读取会员前置发来的随机数R2。通过计算R1和R2生成一组对称密钥，用于加解密后续跟会员前置的通讯报文，如操作员登录，报单，查询，通知等。

服务器在初始化时，针对每一个用户，通过根证书和用户私钥建立一个加密会话。在收到某个API的登录请求时，服务器通过加密会话和服务器证书对请求报文的安全认证部分数据进行解密，读取随机数R1，并自己生成随机数R2，通过计算R1和R2生成一组对称密钥。随后服务器将R2用用户证书加密后随应答报文发送给API。

* 1. **证书私钥的文件**
* 每套证书包含下列文件：

1.UserCert.der：用户证书。

2.UserKey.key：用户私钥。

3.server.der: 服务器证书。

4.RootCert.der: 根证书。

* 1. **安全模块接口**

由于不了解上海CA的安全库的内部实现，所以，并发条件下，请谨慎使用安全模块接口。

多个线程环境下，如果并发调用除了加解密两个函数外的其他函数，建议加锁。

通过编写测试，加解密两个函数可以并发调用。测试场景为：使用同一个安全模块句柄，一个线程加密，一个线程解密。

### 5.3.1初始化接口-api\_safe\_init

初始化接口创建安全模块句柄。

**函数原型**

api\_safe\_handle\_t api\_safe\_init(const char \*key\_path, const char \*rootcert\_path, const char \*svr\_cert\_path, const char \*cli\_cert\_path, const char \*passwd);

参数

key\_path： UserKey.key文件路径

rootcert\_path： RootCert.der文件路径

svr\_cert\_path： server.der文件路径

cli\_cert\_path： UserCert.der文件路径

passwd： 证书密码， 默认123412

**返回值**

api\_safe\_handle\_t handle： 成功

NULL： 失败

### 5.3.2结束接口- api\_safe\_deinit

结束接口释放安全模块句柄。

**函数原型**

int api\_safe\_deinit(api\_safe\_handle\_t handle);

**参数**

handle: 安全模块句柄

**返回值**

0： 成功

-1： 失败

### 5.3.3创建客户端安全信息接口

创建客户端的安全信息域并通过参数输出，用于生成交易员登录报文中的safe\_info字段；接口返回安全信息域的实际长度，用于生成交易员登录报文中的safe\_info\_len。

函数原型

ssize\_t api\_safe\_create\_client\_info(api\_safe\_handle\_t handle, char \*output, size\_t cap);

**参数**

handle： 安全模块句柄

output: 用于输出安全信息的buffer。请注意，output输出时，已经使用base64编码，调用者无需再对输出编码

cap： 输出buffer的容量，必须大于等于2048

**返回值**

一个>0的数值： 输出安全信息的实际长度

-1： 失败

### 5.3.4验证服务器端安全信息接口- api\_safe\_check\_server\_info

对服务器返回的报文安全信息数据进行验证，参数input用于验证交易员登录返回报文中的safe\_info域，参数real\_en\_len用于验证交易员登录返回报文中的safe\_info\_len域。

**函数原型**

int api\_safe\_check\_server\_info(api\_safe\_handle\_t handle, const char \*input, const size\_t real\_en\_len);

**参数**

handle: 安全模块句柄

input: 输入安全信息数据。请注意，input输入时，必须已经经过base64编码

real\_en\_len: 安全信息数据实际长度

**返回值**

0： 成功

-1：失败

### 5.3.5对称加密接口- api\_safe\_encrypt

对明文报文进行对称加密

**函数原型**

SGE\_API int api\_safe\_encrypt(api\_safe\_handle\_t handle, const unsigned char \*src, const size\_t len, unsigned char \*dst, size\_t \*cap);

**参数**

handle: 安全模块句柄

src: 明文报文

len: 明文报文长度

dst: 输出的加密报文

cap: 输出加密报文的容量，

当明文长度为8的倍数时： cap必须大于等于len+8字节

当明文长度不为8的倍数时： cap必须大于等于 len 8字节向上对齐，再加8字节： (8\*(len/8) + 8) + 8;

**返回值**

0：成功

-1：失败

### 5.3.6对称解密接口- api\_safe\_decrypt

对密文报文进行对称解密

**函数原型**

int api\_safe\_decrypt(api\_safe\_handle\_t handle, const unsigned char \*src, const size\_t len, unsigned char \*dst, size\_t \*cap);

**参数**

handle: 安全模块句柄

src: 密文报文

len: 密文报文长度

dst: 输出的解密报文

cap: 输出解密报文的长度，cap必须大于等于len

**返回值**

0： 成功

-1： 失败

### 5.3.7获取版本接口- api\_safe\_get\_version

获取当前安全模块的版本号

**函数原型**

const char \* api\_safe\_get\_version(api\_safe\_handle\_t handle)；

**参数**

handle: 安全模块句柄

**返回值**

版本号

## 5.4 api与safe接口调用关系简述

1. api初始化：启动api相关线程；
2. 安全模块初始化：生成安全模块句柄；
3. 创建客户安全信息：用户调用函数后通过参数得到安全信息的数据，通过函数返回值得到安全信息的长度；
4. 组登陆报文： 将api相关信息，安全信息域的数据和长度放入交易员登陆请求报文的相应字段中。具体字段请参见《上海黄金交易所GEMS-2会员二级系统竞价交易接口规范》；
5. 发送交易员登陆请求报文；
6. 接收返回报文；
7. 验证服务器端安全信息：从gtp报文中读取安全信息域和长度字段，作为函数参数输入。具体字段请参见《上海黄金交易所GEMS-2会员二级系统竞价交易接口规范》；
8. 加密所有发送报文；
9. 解密所有接收报文。



1. **API错误码对照表**

-1 参数为NULL

-2 内存分配失败

-3 创建安全认证相关信息失败

-4 尚未与前置连接

0 操作成功

1 系统初始化失败

2 系统尚未初始化

1. 系统错误

1. **附录A： Demo程序演示**

安全和API 结合使用示例；

如下代码同时支持Windows和Linux平台。

发布了该Demo的Windows和Linux的项目代码，详细的环境和编译配置请参考对应的代码工程。

注意：demo代码只是演示如何使用api和api\_safe模块。缺失许多错误检查和合理的结构管理等，会员请自行根据自己的需求和业务增加缺失部分。

#include <pthread.h>

#include <stdio.h>

#include <string.h>

#include <stdlib.h>

#include "sge\_api\_safe\_interface.h"

#include "api.h"

#ifdef WIN32

#include <windows.h>

#define STRNCPY strncpy\_s

#define SNPRINTF \_snprintf\_s

#define SLEEP(sec) Sleep((sec)\*1000)

#else

#include <unistd.h>

#define STRNCPY strncpy

#define SNPRINTF snprintf

#define SLEEP(sec) sleep(sec)

#endif

#define MAX\_GTP\_MSG\_LENGTH (4096\*2)

typedef struct ctx{

api\_t \* api\_;

api\_cb\_t cb\_;

api\_safe\_handle\_t safe\_handle\_;

volatile unsigned int recv\_count\_;

volatile unsigned int connected\_count\_;

}ctx\_t;

static void \_\_recv\_msg(const char \* msg, int type, unsigned short con\_len, void \* ctx) {

ctx\_t \* pctx = (ctx\_t\*)ctx;

++(pctx->recv\_count\_);

if(type == GTP\_TYPE\_STRING){ // is the no\_enc msg

char gtp\_msg[MAX\_GTP\_MSG\_LENGTH] = { 0 };

STRNCPY(gtp\_msg, msg, MAX\_GTP\_MSG\_LENGTH);

printf("recv msg: %s type: %d con\_len: %d\n", gtp\_msg, type, con\_len);

}

else if(type == GTP\_TYPE\_ENCRYPT){

unsigned char recv\_msg[MAX\_GTP\_MSG\_LENGTH] = {0};

size\_t recv\_msg\_len = MAX\_GTP\_MSG\_LENGTH;

int ret = api\_safe\_decrypt(pctx->api\_, (unsigned char\*)msg, (size\_t)con\_len, recv\_msg, &recv\_msg\_len);

if(ret != 0){

printf("decrypt Error!\n");

return ;

}

}

else{

printf("this message is no handled,msg: %s", msg);

}

}

static void \_\_cb\_connected(api\_connection\_state\_t \* cs, void \* ctx) {

ctx\_t \* pctx = (ctx\_t\*)ctx;

if(cs->state == IS\_CONNECTED){

++(pctx->connected\_count\_);

}

printf("connected to acsvr. ip: %s, port: %d, state: %d\n", cs->ip, cs->port, cs->state);

}

static void \_\_cb\_disconnect(api\_connection\_state\_t \* cs, void \* ctx) {

ctx\_t \* pctx = (ctx\_t\*)ctx;

if(cs->state == IS\_DISCONNECTED){

--(pctx->connected\_count\_);

}

printf("disconnect with acsvr. ip: %s, port: %d, state: %d\n", cs->ip, cs->port, cs->state);

}

static void\* start(void \* ctx){

ctx\_t \* pctx = (ctx\_t\*)ctx;

//(pctx != NULL && pctx->api\_ != NULL);

api\_thread\_begin(pctx->api\_, ".member\_id.req");

//0. wait the session connected

while(pctx->connected\_count\_ <= 0 && api\_is\_running(pctx->api\_)){

SLEEP(1);

printf("wait the session connected! pctx->connected\_count\_ %d\n", pctx->connected\_count\_);

}

if(api\_is\_running(pctx->api\_) == 0){

printf("thread exit!\n");

api\_thread\_end(pctx->api\_);

return NULL;

}

//1. create clien safe\_info

int ret = 0;

char \*en\_cli\_ctx = (char \*)calloc(2048, 1);

size\_t en\_cli\_ctx\_len = 2048;

ret = api\_safe\_create\_client\_info(pctx->safe\_handle\_, en\_cli\_ctx, en\_cli\_ctx\_len);

//TODO: error check

//2.prepare the login msg

char \* trader\_login = "gtp\_version=GTP1.0,msg\_type=A100,seq\_series\_no=1,seq\_no=0,chain\_flag=l,root\_id=gtp test,sender\_id=api\_test,receiver\_id=0000,cert\_no=310004198303234345,flow\_brkpt=[{flow\_type=2,flow\_seq=0},{flow\_type=3,flow\_seq=9},{flow\_type=4,flow\_seq=9}],ip\_address=180.3.10.99,is\_login=1,local\_order\_no=999999999,machine\_id=123456,member\_id=0003,password=123,safe\_info=%s,safe\_info\_len=%d,seat\_id=000301,state=1,trade\_date=20140101,trader\_id=gmx6003,version=v1.1.0";

char req[MAX\_GTP\_MSG\_LENGTH] = {0};

size\_t req\_len = SNPRINTF(req, sizeof(req), trader\_login, en\_cli\_ctx, ret);

//TODO: error check

//3. send it with no\_enc msg

printf("begin to send trader login request.\n");

ret = api\_send\_msg\_no\_enc(pctx->api\_, req, (int)req\_len);

//error check

printf("Return value of api\_send\_msg\_no\_nec: %d\n", ret);

//4. send non-login msg

char \* order\_apply = "gtp\_version=GTP1.0,msg\_type=T000,seq\_series\_no=1,seq\_no=0,chain\_flag=l,root\_id=gtp test,sender\_id=api\_test,receiver\_id=0000,buy\_sale\_flag=b,client\_id=0000000003,inst\_id=Au(T+D),local\_order\_no=00000001,market\_id=02,match\_type=1,member\_id=0003,offset\_flag=1,order\_type=0,price=25200,trader\_id=gmx6003,volume=1";

//4.1 enc the req

memset(req, 0 ,sizeof(req));

req\_len = MAX\_GTP\_MSG\_LENGTH;

ret = api\_safe\_encrypt(pctx->safe\_handle\_, (const unsigned char \*)order\_apply, strlen(order\_apply), (unsigned char \*)req, &req\_len);

//TODO: error check

//4.2 send it with send\_msg with enc

ret = api\_send\_msg(pctx->api\_, req, req\_len);

//TODO: error check

printf("Return value of api\_send\_msg with enc req: %d\n", ret);

printf("thread exit!\n");

api\_thread\_end(pctx->api\_);

return NULL;

}

int main(int argc, char \*\*argv) {

int ret = 0;

//init ctx

ctx\_t ctx;

memset(&ctx, 0, sizeof(ctx\_t));

ctx.cb\_.api\_recv\_msg\_ = \_\_recv\_msg;

ctx.cb\_.api\_cb\_connected\_ = \_\_cb\_connected;

ctx.cb\_.api\_cb\_disconnected\_ = \_\_cb\_disconnect;

// init api

ctx.api\_ = api\_init("127.0.0.1", 12344, &(ctx.cb\_), &ctx);

if(ctx.api\_ == NULL) {

printf("api init fail\n");

ret = -1;

goto err;

}

// init the safe handler

char \* key\_path = "./cert\_file/UserKey.key";

char \* cert\_chain\_path = "./cert\_file/RootCert.der";

char \* cli\_cert\_path = "./cert\_file/UserCert.der";

char \* svr\_cert\_path = "./cert\_file/server.der";

ctx.safe\_handle\_ = API\_SAFE\_EASY\_INIT(key\_path, cert\_chain\_path, svr\_cert\_path, cli\_cert\_path);

if(ctx.safe\_handle\_ == NULL){

printf("ERROR: init safe api failed\n");

ret = -1;

goto err\_to\_free\_api;

}

pthread\_t tid;

if (pthread\_create(&tid, NULL, start, &ctx) != 0) {

printf("pthread create fail.\n");

ret = -1;

goto err\_to\_free\_api\_safe;

}

pthread\_join(tid, NULL);

api\_wait(ctx.api\_);

err\_to\_free\_api\_safe:

api\_safe\_deinit(ctx.safe\_handle\_);

err\_to\_free\_api:

api\_free(ctx.api\_);

err:

return ret;

}

1. **附录 B 合并编译GTP API 和2.5代API方法**
   1. **Windows版本 C++编译过程**

如下以api\_demo的windows版本为例子，描述如何合并2.5代API，其中api\_demo的编译配置，请查看已发布的api\_demo项目配置。

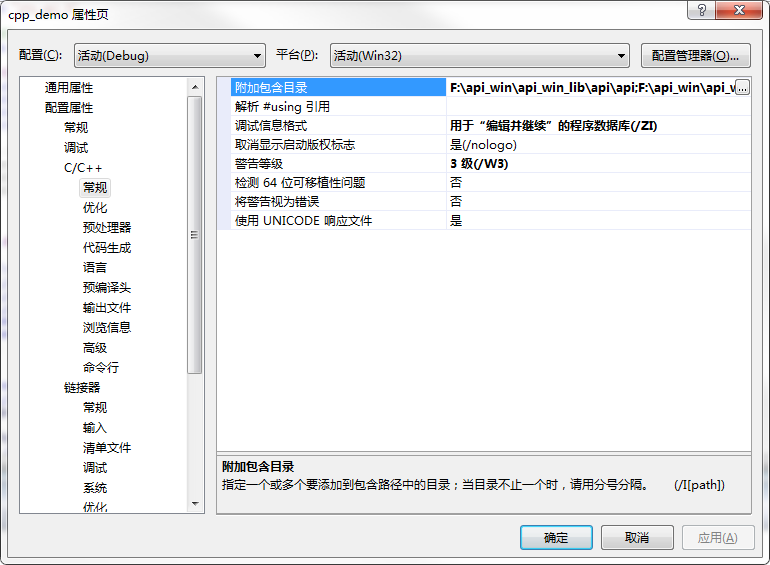
* + - 1. **编译准备**

修改GTP API lib中头文件api.h为 gtp\_api.h；

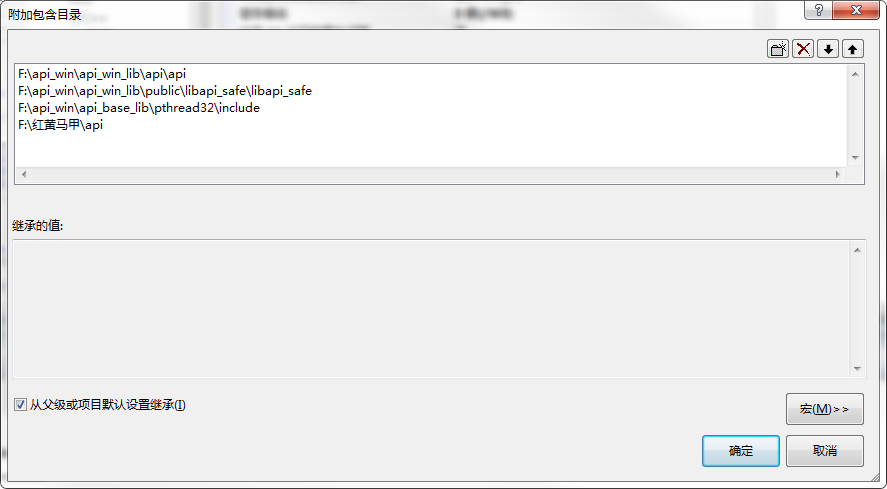
保持2.5代API为api.h （windows平台不区分大小写）

* + - 1. **增加2.5代API的头文件**

在Visual Studio 编译器中，右键项目属性->配置属性->C/C++->常规->附加包含目录



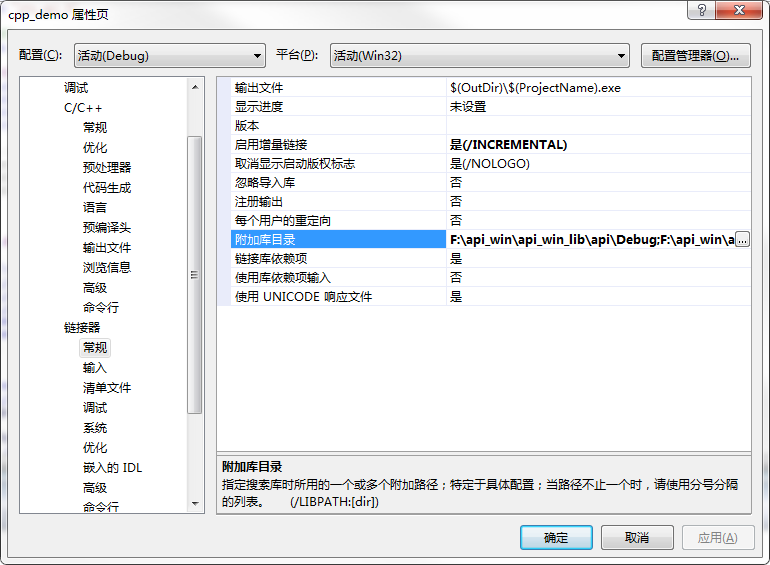
单击附加包含目录



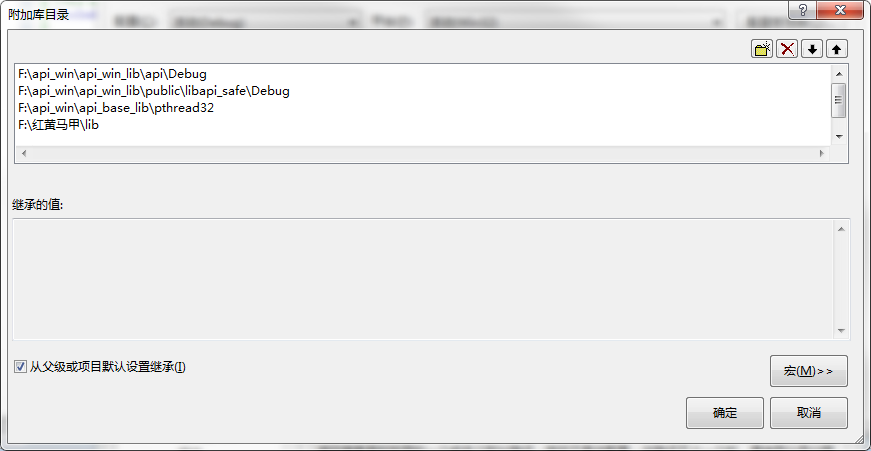
增加 2.5代API的头文件目录 F:\红黄马甲\api

* + - 1. **增加2.5代API 链接信息（路径和静态库文件）**

右键项目属性->配置属性->链接器->常规->附加库目录

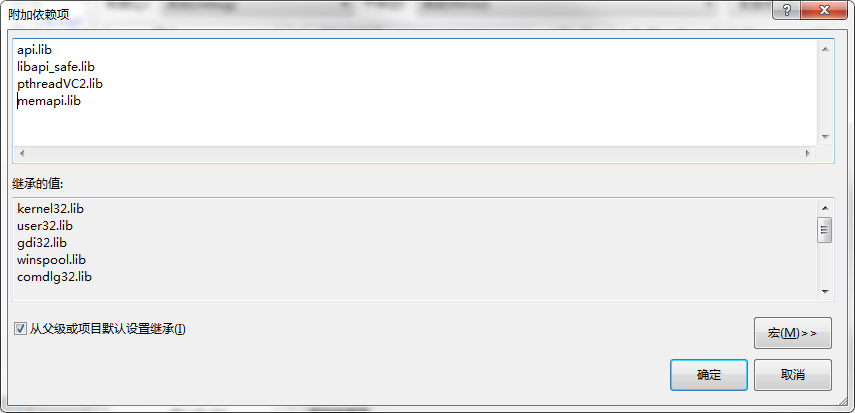


单击附加目录库按钮，增加2.5代API库所在的目录 F:\红黄马甲\lib



右键项目属性->配置属性->链接器->常规->附加依赖项

增加依赖项memapi.lib



* 1. **Linux版本 C++编译过程**

由于2.5代API不支持Linux系统，无法支持2.5代和新GTP API的合并编译。

Linux的GTP API编译选项可以参考Linux版本的GTP API demo文件。

1. **附录 C Linux版本 安全模块依赖CA库部署**
   * + - 1. 检查/usr/lib下是否有对应的CA安全库重名的so库，如果确认是旧版本，请删除；

so名称请查看api\_lib\_linux\_x86\_64/lib/ca\_cert\_lib/lib下的所有so文件名称。

* + - * 1. 复制lib文件下的so文件至/usr/lib
        2. 指定环境变量LD\_LIBRARY\_PATH="/usr/lib"
        3. 复制ini文件夹下的所有文件至/etc
        4. 运行test文件夹下的test确认能否正常运行

摘抄自 api\_lib\_linux\_x86\_64/lib/ca\_cert\_lib/readme.txt