Websocket 接口使用说明

1.接口使用方法：

1.1

websocketClient.so 使用该动态库前应将该动态库拷贝到/usr/lib 目录下，并安装 asio 和 openssl 库

1.2

写好 c 程序后编译命令：*gcc test.c -o test -lssl -lcrypto -lwss_client*

2.接口详细介绍：

2.1 要建立一个完整的连接，需要包含以下 4 个步骤：

2.1.1

clientptr 是一个客户端类型的指针类，在建立连接前应先定义一个 clientptr 型变量，并通过 getclientptr()函数获取一个客户端对象指针。

例：clientptr c_ptr=getclientptr()；

c_ptr 指向一个客户端对象，后面客户端的一切操作都需要该客户端指针作为参数。

2.1.2

设置 uri。

2.1.3

ws_set_headers(c_ptr, key, val);//设置客户端请求头信息。

ws_set_open_callback()//设置收到建立连接成功时的回调

ws_set_fail_callback()//设置收到建立连接失败时的回调

ws_set_close_callback()//设置收到关闭连接时的回调

ws_set_binaryMessage_callback(c_ptr, on_binaryMessage);//设置收到 binary 消息的回调

ws_set_textMessage_callback(c_ptr, on_textMessage);//设置收到 text 消息的回调函数

ws_set_pong_callback(c_ptr, on_pong);//设置收到 pong 时的回调函数

ws_set_ping_callback(c_ptr, on_ping);//设置收到 ping 时的回调函数

2.1.4

int id = ws_connect(c_ptr, uri);

发起连接请求，返回一个 id。id 用于标记这连接。因为一个客户端可以连接多个服务端，因此需要 id 作为标识，后面对某一服务端 send、ping、pong 等操作都需要传递这个 id 作为参数，用来区分是对那个服务端进行操作。

2.2 回调函数格式：

on_textMessage,on_ping,on_pong 三个回调函数格式一致，具体格式如下：

```
/*************************************************************
   *    void on_callback(const char * const msg,const size_t len,const int id){}
   * @param       msg 为收到的文本类型消息,len 为消息长度,id 为对端标识
 *************************************************************/
```

收到二进制消息的回调函数格式：

```
/*************************************************************
   *    void binary_callback(const char * const msg,const size_t len,const int id){}
   * @param       msg 为收到的二进制类型消息,len 为消息长度,id 为对端标识
 *************************************************************/
```

on_open,on_close,on_fail 三个回调函数具体格式如下：

```
/*************************************************************
   *    void on_open(const int id)
   *    void on_close(const int id)
   *    void on_fail(const int id)
   *    @param         id 为触发回调函数的 id
 *************************************************************/
```

2.3 全部接口：

```
//定义回调函数类型，callback 类型函数用于 on_textMessage,on_ping,on_pong
typedef void (*callback)(const char *const, const size_t, const int);

//binary_callback 类型函数用于 on_binaryMessage 回调
typedef void (*binary_callback)(const void *const, const size_t, const int);

//connectCallback 类型用于 on_open,on_close,on_fail
typedef void (*connectCallback)(const int);

//获取客户端对象指针
clientptr getclientptr();

//删除客户端对象指针
void delete_clientptr(clientptr c_ptr);

//通过 id 获取此连接状态，状态有 Connecting、Open 两种状态,状态通过 status 指针
```

```
//返回,刚发起 connect 请求时此 id 状态为 connecting，当建立连接成功后，状态为
//Open
void ws_getStatus(clientptr c_ptr, const int id, char *status);

//发起连接请求，返回一个 id 用来标识此连接，发起连接请求时此连接的状
//Connectin,连接建立成功后为 Open 状态
int ws_connect(clientptr c_ptr, const char *const uri, const int len);

//关闭标识号为 id 的这个连接，关闭的同时删除 id。
void ws_close(clientptr c_ptr, const int id);

//发送 text 类型信息，信息长度为 len
void ws_send_text(clientptr c_ptr, const int id, const char *const str, const size_t len);

//发送 binary 类型信息，信息长度为 len
void ws_send_binary(clientptr c_ptr, const int id, const void *const str, const size_t len);

//发送 ping，payload 为附带 text 信息，长度为 len
void ws_ping(clientptr c_ptr, const int id, const char *const payload, const size_t len);

//发送 pong，payload 为附带 text 信息，长度为 len
void ws_pong(clientptr c_ptr, const int id, const char *const payload, const size_t len);

//设置请求 headers，键为 key，长度为 lenk，值为 val，长度为 lenv
void ws_set_headers(clientptr c_ptr, const char *const key, const size_t lenk, const char
   *const val, const size_t lenv);

//设置收到 binary 类型消息时的回调函数
void ws_set_binaryMessage_callback(clientptr c_ptr, binary_callback msgfun);

//设置收到 text 类型消息时的回调函数
void ws_set_textMessage_callback(clientptr c_ptr, callback msgfun);

//设置 ping 回调
void ws_set_ping_callback(clientptr c_ptr, callback pingfun);

//设置 pong 回调
void ws_set_pong_callback(clientptr c_ptr, callback pongfun);

//设置连接成功时 open 回调函数
void ws_set_open_callback(clientptr c_ptr, connectCallback openfun);

//设置关闭连接时 close 回调函数
void ws_set_close_callback(clientptr c_ptr, connectCallback closefun);
```

```
//设置连接失败时 fail 回调函数
void ws_set_fail_callback(clientptr c_ptr, connectCallback failfun);

//设置当前收到消息的缓存大小
void ws_set_msgbuffersize(clientptr c_ptr, const size_t size);

//开启事件循环
void ws_run(clientptr c_ptr);

//关闭 ws_run;
void ws_stop(clientptr c_ptr);
```

3.程序示例：

```
#include <stdio.h>

#include <string.h>
#include <unistd.h>
#include <stdlib.h>
#include "wss_client_api.h"

clientptr c_ptr; // clientptr  是一个
void message(const char *const msg, const size_t len, const int id)
{

    printf("发送成功并接受到回应%s:", msg);
    printf("id:%d", id);
    ws_send_text(c_ptr, id, msg, strlen(msg));
}

void on_message_binary(const void *const msg, const size_t len, const int id)
{
    printf("收到一个二进制消息%d\n", id);
}

void onpongf(const char *const msg, const size_t len, const int id)
{
    printf("收到一个 pongid%d\n", id);
    // ws_send(c_ptr,id,msg,strlen(msg));
}
void onpingf(const char *const msg, const size_t len, const int id)
{
```

```
    printf("收到一个 pingid%d", id);
}

void onopenf(const int id)
{
    printf("连接建立成功 id:%d\n", id);
    printf("自设 open 回调\n");
}

void onfailf(const int id)
{
    printf("连接建立失败\n");
}

void onclosef(const int id)
{
    printf("触发自设 close 回调\n");
}

int main()
{
    c_ptr = getclientptr();

    char uri[] = "wss://170s2247n7.51mypc.cn/ws/cs/YK0122110001";

    char key[] = "Sec-WebSocket-Protocol";
    char val[] = "ocpp1.6";

    ws_set_open_callback(c_ptr, onopenf);
    ws_set_close_callback(c_ptr, onclosef);
    ws_set_fail_callback(c_ptr, onfailf);
    ws_set_headers(c_ptr, key, strlen(key), val, strlen(val));
    ws_set_textMessage_callback(c_ptr, message);
    ws_set_binaryMessage_callback(c_ptr, on_message_binary);
    ws_set_pong_callback(c_ptr, onpongf);
    ws_set_ping_callback(c_ptr, onpingf);

    int id = ws_connect(c_ptr, uri, strlen(uri));
    int done = 1;
    while (done)
    {
        printf("Enter Command:\n");
        scanf("%s", input);
        if (strncmp(input, "help", 4) == 0)
```

```
{
    printf("Command List:\n"
            "send text <message>\n"
            "send binary <message>\n"
            "close <connection id> [<close code:default=1000>] [<close reason>]\n"
            "help: Display this help text\n"
            "ping<payload>\n"
            "getStatus<connection id>\n"
            "quit: Exit the program\n");
}
else if (strncmp(input, "send text", 9) == 0)
{
    input += 10;
    ws_send_text(c_ptr, id, input, strlen(input) - 10);
    input -= 10;
}

else if (strncmp(input, "send binary", 11) == 0)
{
    input += 12;
    ws_send_binary(c_ptr, id, input, strlen(input) - 12);
    input -= 12;
}

else if (strncmp(input, "close", 5) == 0)
{
    ws_close(c_ptr, id);
}

else if (strncmp(input, "ping", 4) == 0)
{
    input += 5;
    ws_ping(c_ptr, id, input, strlen(input) - 5);
    input -= 5;
}
else if (strncmp(input, "get status", 10) == 0)
{
    input += 11;
    char *status = malloc(10);
    ws_get_Status(id,status);
    input -= 11;
    printf("此链接的状态为:%s", status);
}
```

```
        else if (strncmp(input, "quit", 4) == 0)
        {
            done = 0;
            delete_clientptr(c_ptr);
        }
    }
    return 0;
}
```