/\*

\*

\* 控制类消息的TCP线程 主要功能是接收和处理tcp控制类消息

\* 上位机的通讯部分

\* 单元机的通讯部分

\* 从服务端角度出发，接收客户端连接请求后，需要知道的客户端的消息，如下

\* 1.TCP中客户端的fd，后续下发指令需要用到

\* 2.客户端的IP/MAC地址，主要是识别具体的单元机

\*

\* 在TCP控制模块主要处理的工作，如下，

\* 1.逻辑控制类管理，如单元机关机

\* 2.会议类管理，扫描，ID设置，席别设置，签到

\*

\* 综上，服务端需要绑定fd和对应的ID（MAC）,控制消息才能准确下发

\*

\* Created on: 2016年9月29日

\* Author: leon

\*/

#include "tcp\_ctrl\_server.h"

#include "tcp\_ctrl\_data\_compose.h"

#include "tcp\_ctrl\_data\_process.h"

#include "tcp\_ctrl\_device\_status.h"

#include "tcp\_ctrl\_device\_manage.h"

#include "tcp\_ctrl\_list.h"

#include "tcp\_ctrl\_queue.h"

#include "tcp\_ctrl\_api.h"

#include "scanf\_md\_udp.h"

#include "sys\_uart\_init.h"

//互斥锁和信号量

sys\_info sys\_in;

//链表和结点

Pglobal\_info node\_queue;

/\*

\* tcp\_ctrl\_local\_addr\_init.c

\* 服务端socket初始化

\*

\* 返回值

\* 错误码和成功后的端口号

\*

\*/

static int tcp\_ctrl\_local\_addr\_init(int port)

{

struct sockaddr\_in sin;

int sock\_fd;

int ret;

bzero(&sin, sizeof(sin));

sin.sin\_family = AF\_INET;

sin.sin\_addr.s\_addr = INADDR\_ANY;

sin.sin\_port = htons(port);

sock\_fd = socket(AF\_INET, SOCK\_STREAM, 0);

if (-1 == sock\_fd)

{

perror("call to socket");

return SOCKERRO;

}

//reuse socket

int opt=1;

setsockopt(sock\_fd,SOL\_SOCKET,SO\_REUSEADDR,&opt,sizeof(opt));

ret = bind(sock\_fd, (struct sockaddr \*)&sin, sizeof(sin));

if (-1 == ret)

{

perror("call to bind");

return BINDERRO;

}

ret = listen(sock\_fd, 20);

if (-1 == ret)

{

perror("call to listen");

return LISNERRO;

}

return sock\_fd;

}

/\*

\* tc\_set\_noblock.c

\* 设置socket\_fd为非阻塞模式

\*/

static int tcp\_ctrl\_set\_noblock(int fd)

{

int fl=fcntl(fd,F\_GETFL);

if(fl<0)

{

perror("fcntl");

return SETOPTERRO;

}

if(fcntl(fd,F\_SETFL,fl|O\_NONBLOCK))

{

perror("fcntl");

return SETOPTERRO;

}

return SUCCESS;

}

/\*

\* tcp\_ctrl\_process\_recv\_msg

\* 接收消息的初步处理

\* 分类：

\* 上位机通讯部分

\* 单元机通讯部分

\*

\* 参数：

\* 客户端FD

\* 接收到的消息

\*

\*

\*/

static int tcp\_ctrl\_process\_recv\_msg(int\* cli\_fd, unsigned char\* value, int\* length)

{

/\*

\* 检查数据包是否是完整的数据包

\* 数据头的检测

\* 数据长度的检测

\* 校验和的检测

\*/

Pframe\_type tmp\_type;

int ret = 0;

int status = 0;

unsigned char\* handlbuf = NULL;

tmp\_type = (Pframe\_type)malloc(sizeof(frame\_type));

memset(tmp\_type,0,sizeof(frame\_type));

ret = tcp\_ctrl\_frame\_analysis(cli\_fd,value,length,tmp\_type,&handlbuf);

#if 0

printf("type->msg\_type = %d\n"

"type->data\_type = %d\n"

"type->dev\_type = %d\n",tmp\_type->msg\_type,tmp\_type->data\_type,tmp\_type->dev\_type);

#endif

if(ret != SUCCESS)

{

printf("tcp\_ctrl\_frame\_analysis failed\n");

free(tmp\_type);

free(handlbuf);

return ret;

}

/\*

\* 在连接信息链表中检查端口合法性

\*/

if(tmp\_type->msg\_type != ONLINE\_REQ)

{

status = conf\_status\_check\_client\_connect\_legal(tmp\_type);

if(status < 1)

{

printf("client is not legal connect\n");

free(tmp\_type);

free(handlbuf);

return CONECT\_NOT\_LEGAL;

}

}

/\*

\* 设备类型分类

\*/

switch(tmp\_type->dev\_type)

{

case PC\_CTRL:

tcp\_ctrl\_from\_pc(handlbuf,tmp\_type);

break;

case UNIT\_CTRL:

tcp\_ctrl\_from\_unit(handlbuf,tmp\_type);

break;

}

free(tmp\_type);

free(handlbuf);

return SUCCESS;

}

/\*

\* wifi\_sys\_ctrl\_tcp\_recv

\* 设备控制模块TCP数据接收模块

\*

\* 采用epoll模式，维护多终端

\* 接收客户端的通讯数据

\*

\*/

void\* wifi\_sys\_ctrl\_tcp\_recv(void\* p)

{

struct epoll\_event event;

struct epoll\_event\* wait\_event;

struct sockaddr\_in cli\_addr;

int sockfd;

int epoll\_fd;

int clilen = sizeof(cli\_addr);

int n;

int newfd;

int ctrl\_ret,wait\_ret;

int maxi = 0;

int len = 0;

int ret = 0;

unsigned char buf[1024] = {0};

printf("%s,%d\n",\_\_func\_\_,\_\_LINE\_\_);

pthread\_detach(pthread\_self());

/\*

\* 控制模块接收服务器套接字初始化

\*/

sockfd = tcp\_ctrl\_local\_addr\_init(CTRL\_TCP\_PORT);

if(sockfd < 0)

{

printf("wifi\_sys\_ctrl\_tcp\_recv failed\n");

pthread\_exit(0);

}

ret = tcp\_ctrl\_set\_noblock(sockfd);

if(ret < 0)

{

printf("tcp\_ctrl\_set\_noblock failed\n");

pthread\_exit(0);

}

/\*

\* epoll event init

\* 除了参数size被忽略外,此函数和epoll\_create完全相同

\*/

epoll\_fd = epoll\_create1(0);

if(epoll\_fd < 0){

perror ("epoll\_create");

pthread\_exit(0);

}

/\*

\* epoll监听套接字

\* epoll事件参数设置

\*/

event.data.fd = sockfd;

event.events = EPOLLIN | EPOLLET;

//事件注册函数，将监听套接字描述符 sockfd 加入监听事件

ctrl\_ret = epoll\_ctl(epoll\_fd, EPOLL\_CTL\_ADD, sockfd, &event);

if(-1 == ctrl\_ret){

perror("epoll\_ctl");

pthread\_exit(0);

}

wait\_event = calloc(255,sizeof(struct epoll\_event));

while(1)

{

// printf("epoll\_wait...\n");

// 监视并等待多个文件（标准输入，udp套接字）描述符的属性变化（是否可读）

// 没有属性变化，这个函数会阻塞，直到有变化才往下执行，这里没有设置超时

wait\_ret = epoll\_wait(epoll\_fd, wait\_event, maxi+1, -1);

if(wait\_ret == -1)

{

perror("epoll\_wait");

break;

}

for (n = 0; n < wait\_ret; ++n)

{

if (wait\_event[n].data.fd == sockfd

&& (EPOLLIN == (wait\_event[n].events & (EPOLLIN|EPOLLERR))))

{

pthread\_mutex\_lock(&sys\_in.sys\_mutex[CTRL\_TCP\_MUTEX]);

newfd = accept(sockfd, (struct sockaddr \*) &cli\_addr,

(socklen\_t\*)&clilen);

if (newfd < 0) {

perror("accept");

continue;

}

else

{

ret = tcp\_ctrl\_set\_noblock(newfd);

if(ret < 0)

{

printf("tcp\_ctrl\_set\_noblock failed\n");

pthread\_exit(0);

}

// 打印客户端的 ip 和端口

printf("%d---------------------------\n",newfd);

printf("client ip=%s,port=%d\n", inet\_ntoa(cli\_addr.sin\_addr),

ntohs(cli\_addr.sin\_port));

event.events = EPOLLIN | EPOLLET;

event.data.fd = newfd;

if (epoll\_ctl(epoll\_fd, EPOLL\_CTL\_ADD, newfd, &event) < 0) {

fprintf(stderr, "add socket '%d' to epoll failed %s\n",

newfd, strerror(errno));

pthread\_exit(0);

}

maxi++;

pthread\_mutex\_unlock(&sys\_in.sys\_mutex[CTRL\_TCP\_MUTEX]);

}

}else {

pthread\_mutex\_lock(&sys\_in.sys\_mutex[CTRL\_TCP\_MUTEX]);

len = recv(wait\_event[n].data.fd, buf, sizeof(buf), 0);

pthread\_mutex\_unlock(&sys\_in.sys\_mutex[CTRL\_TCP\_MUTEX]);

//客户端关闭连接

if(len < 0)// && errno != EAGAIN)

{

printf("wait\_event[n].data.fd--%d offline\n",wait\_event[n].data.fd);

ctrl\_ret = epoll\_ctl(epoll\_fd, EPOLL\_CTL\_DEL,

wait\_event[n].data.fd,&event);

if( ctrl\_ret < 0)

{

fprintf(stderr, "delete socket '%d' from epoll failed! %s\n",

wait\_event[n].data.fd, strerror(errno));

}

/\*

\* 删除链表中节点信息与文本中的信息

\* fd参数

\*/

pthread\_mutex\_lock(&sys\_in.sys\_mutex[LIST\_MUTEX]);

dmanage\_delete\_info(wait\_event[n].data.fd);

pthread\_mutex\_unlock(&sys\_in.sys\_mutex[LIST\_MUTEX]);

printf("size--%d\n",node\_queue->sys\_list[CONNECT\_LIST]->size);

close(wait\_event[n].data.fd);

maxi--;

}

else if(len == 0)//客户端关闭连接

{

printf("client %d offline\n",wait\_event[n].data.fd);

ctrl\_ret = epoll\_ctl(epoll\_fd, EPOLL\_CTL\_DEL,

wait\_event[n].data.fd,&event);

if( ctrl\_ret < 0)

{

fprintf(stderr, "delete socket '%d' from epoll failed! %s\n",

wait\_event[n].data.fd, strerror(errno));

}

pthread\_mutex\_lock(&sys\_in.sys\_mutex[LIST\_MUTEX]);

dmanage\_delete\_info(wait\_event[n].data.fd);

pthread\_mutex\_unlock(&sys\_in.sys\_mutex[LIST\_MUTEX]);

printf("size--%d\n",node\_queue->sys\_list[CONNECT\_LIST]->size);

close(wait\_event[n].data.fd);

maxi--;

}

else

{

// getpeername(wait\_event[n].data.fd,(struct sockaddr\*)&cli\_addr,

// &clilen);

// printf("client %s:%d\n",inet\_ntoa(cli\_addr.sin\_addr),

// ntohs(cli\_addr.sin\_port));

//通过返回值进行处理

pthread\_mutex\_lock(&sys\_in.sys\_mutex[CTRL\_TCP\_RQUEUE\_MUTEX]);

tcp\_ctrl\_tprecv\_enqueue(&wait\_event[n].data.fd,buf,&len);

pthread\_mutex\_unlock(&sys\_in.sys\_mutex[CTRL\_TCP\_RQUEUE\_MUTEX]);

}

}//end if

}//end for

}//end while

free(wait\_event);

close(sockfd);

free(node\_queue->con\_status);

free(node\_queue);

pthread\_exit(0);

}

/\*

\* wifi\_sys\_ctrl\_tcp\_send

\* 设备控制模块TCP数据下发线程

\*

\* 发送消息进入发送队列，进行数据下发

\*

\*/

void\* wifi\_sys\_ctrl\_tcp\_send(void\* p)

{

int i,ret;

Plinknode node;

Pctrl\_tcp\_rsqueue tmp;

pthread\_detach(pthread\_self());

while(1)

{

sem\_wait(&sys\_in.sys\_sem[CTRL\_TCP\_SEND\_SEM]);

pthread\_mutex\_lock(&sys\_in.sys\_mutex[CTRL\_TCP\_SQUEUE\_MUTEX]);

ret = out\_queue(node\_queue->sys\_queue[CTRL\_TCP\_SEND\_QUEUE],&node);

if(ret == 0)

{

tmp = node->data;

if(tmp->msg[4] != 0x87)

{

printf("%s-tmp->socket\_fd[%d] : \n",

\_\_func\_\_,tmp->socket\_fd);

for(i=0;i<tmp->len;i++)

{

printf("%x ",tmp->msg[i]);

}

printf("\n");

}

pthread\_mutex\_lock(&sys\_in.sys\_mutex[CTRL\_TCP\_MUTEX]);

write(tmp->socket\_fd,tmp->msg, tmp->len);

pthread\_mutex\_unlock(&sys\_in.sys\_mutex[CTRL\_TCP\_MUTEX]);

free(tmp->msg);

free(tmp);

free(node);

}else{

printf("%s dequeue error\n",\_\_func\_\_);

}

pthread\_mutex\_unlock(&sys\_in.sys\_mutex[CTRL\_TCP\_SQUEUE\_MUTEX]);

}

free(tmp);

}

/\*

\* wifi\_sys\_ctrl\_tcp\_procs\_data

\* 设备控制模块TCP接收数据处理线程

\*

\* 将消息从队列中读取，进行处理后发送

\*

\*/

void\* wifi\_sys\_ctrl\_tcp\_procs\_data(void\* p)

{

Plinknode node;

Pctrl\_tcp\_rsqueue tmp;

int ret;

pthread\_detach(pthread\_self());

while(1)

{

sem\_wait(&sys\_in.sys\_sem[CTRL\_TCP\_RECV\_SEM]);

// printf("%s-%s-%d\n",\_\_FILE\_\_,\_\_func\_\_,\_\_LINE\_\_);

pthread\_mutex\_lock(&sys\_in.sys\_mutex[CTRL\_TCP\_RQUEUE\_MUTEX]);

ret = out\_queue(node\_queue->sys\_queue[CTRL\_TCP\_RECV\_QUEUE],&node);

if(ret == 0)

{

tmp = node->data;

ret = tcp\_ctrl\_process\_recv\_msg(&tmp->socket\_fd,tmp->msg,&tmp->len);

free(tmp->msg);

free(tmp);

free(node);

}else{

printf("%s dequeue error\n",\_\_func\_\_);

}

pthread\_mutex\_unlock(&sys\_in.sys\_mutex[CTRL\_TCP\_RQUEUE\_MUTEX]);

}

free(tmp);

}

/\*

\* wifi\_sys\_ctrl\_tcp\_heart\_sta

\* 设备管理，心跳检测

\* 1、轮询查找当前设备连接信息中的心跳状态

\* 2.判断连接信息，如果状态为0，则表示端口异常，需要关闭

\*/

void\* wifi\_sys\_ctrl\_tcp\_heart\_state(void\* p)

{

frame\_type type;

memset(&type,0,sizeof(frame\_type));

pthread\_detach(pthread\_self());

while(1)

{

if(!conf\_status\_get\_connected\_len())

{

sleep(1);

continue;

}else{

sleep(5);

dmanage\_get\_communication\_heart(&type);

if(type.fd)

{

printf("%s-%s-%d client connect err,close it\n",\_\_FILE\_\_,\_\_func\_\_,\_\_LINE\_\_);

pthread\_mutex\_lock(&sys\_in.sys\_mutex[LIST\_MUTEX]);

dmanage\_delete\_info(type.fd);

pthread\_mutex\_unlock(&sys\_in.sys\_mutex[LIST\_MUTEX]);

close(type.fd);

type.fd = 0;

}

}

}

}

void read\_file(char\* fname)

{

Pclient\_info pinfo;

int ret;

FILE\* file;

pinfo = malloc(sizeof(client\_info));

file = fopen(fname,"r");

while(1)

{

ret = fread(pinfo,sizeof(client\_info),1,file);

perror("fread");

if(ret ==0)

return;

if(pinfo == NULL)

break;

printf("fd:%d,ip:%s-ip-%s,client:%d\n",pinfo->client\_fd,

inet\_ntoa(pinfo->cli\_addr.sin\_addr),pinfo->ip,

pinfo->client\_name);

printf("ip-%s,client:%x\n",

inet\_ntoa(pinfo->cli\_addr.sin\_addr),pinfo->cli\_addr.sin\_addr.s\_addr);

usleep(100000);

}

free(pinfo);

fclose(file);

}

int print\_conference\_list()

{

pclient\_node tmp = NULL;

Pconference\_list info;

tmp = node\_queue->sys\_list[CONFERENCE\_LIST]->next;

while(tmp != NULL)

{

info = tmp->data;

if(info->fd > 0)

{

printf("fd--%d,id--%d,seat--%d,name--%s conf\_name-%s\n",info->fd,

info->con\_data.id,info->con\_data.seat,info->con\_data.name,info->con\_data.conf\_name);

}

tmp = tmp->next;

}

return SUCCESS;

}

void\* control\_tcp\_send(void\* p)

{

int ret;

/\*

\* 设备连接后，发送一个获取mac的数据单包

\* 控制类，事件型，主机发送数据

\*/

int s,fd,id,seat,value;

host\_info list;

memset(&list,0,sizeof(host\_info));

char file\_name[32] = {0};

while(1)

{

/\*

\* 模拟对单元机的控制

\* ‘S’扫描单元机,0是获取mac

\* ‘I’ID设置

\* ‘X’席别

\*

\*/

printf("######################################\n");

printf("1 扫描单元机\n");

printf("2 设置会议ID参数\n");

printf("3 获取单元机会议参数\n");

printf("4 设置最大发言人数\n");

printf("5 获取最大发言人数\n");

printf("6 设置话筒模式\n");

printf("7 获取话筒模式\n");

printf("8 开始签到\n");

printf("9 结束签到\n");

printf("10 恢复出厂设置\n");

printf("11 网络信息\n");

printf("12 关于本机\n");

printf("13 关机\n");

printf("14 摄像跟踪\n");

printf("######################################\n");

scanf("%d",&s);

switch(s)

{

case 1:

ret=unit\_info\_get\_connected\_info(file\_name);

read\_file(file\_name);

print\_conference\_list();

printf("file name \"%s\",scanf\_client size--%d\n",file\_name,ret);

break;

case 2:

printf("s=%d,input the fd,id,seat\n",s);

scanf("%d",&fd);

scanf("%d",&id);

scanf("%d",&seat);

conf\_info\_set\_conference\_params(fd,id,seat,0,0);

break;

case 3:

printf("s=%d,input the fd\n",s);

scanf("%d",&fd);

conf\_info\_get\_the\_conference\_params(fd);

break;

case 4:

printf("s=%d,input the spk numt\n",s);

scanf("%d",&value);

conf\_info\_set\_spk\_num(value);

break;

case 5:

ret = conf\_info\_get\_spk\_num();

printf("conf\_info\_get\_spk\_num value=%d\n",ret);

break;

case 6:

printf("s=%d,input the mic mode\n",s);

scanf("%d",&value);

conf\_info\_set\_mic\_mode(value);

break;

case 7:

conf\_info\_get\_mic\_mode();

break;

case 8:

break;

case 9:

break;

case 10:

host\_info\_reset\_factory\_mode();

break;

case 11:

host\_info\_get\_network\_info(&list);

printf("\tmac: %s\n",list.mac\_addr );

printf("\tIP: %s\n", list.ip\_addr);

printf("\tNetmask: %s\n",list.netmask );

break;

case 12:

host\_info\_get\_factory\_info(&list);

printf("\tversion: %s\n",list.version );

printf("\thost\_model: %s\n",list.host\_model );

printf("\tfactory\_information: %s\n",list.factory\_info);

break;

case 13:

unit\_info\_set\_device\_power\_off();

break;

case 14:

printf("sys\_uart\_video\_set\n");

printf("s=%d,input the value\n",s);

scanf("%d",&value);

scanf("%d",&id);

sys\_uart\_video\_set(id,value);

break;

case 15:

printf("subject set\n");

conf\_info\_set\_conference\_sub\_params();

break;

default:

break;

}

}

}

void\* control\_tcp\_queue(void\* p)

{

Prun\_status event\_tmp;

event\_tmp = (Prun\_status)malloc(sizeof(run\_status));

while(1)

{

unit\_info\_get\_running\_status(event\_tmp);

printf("%s-%s-fd:%d,id:%d,seat:%d,value:%d\n",

\_\_FILE\_\_,\_\_func\_\_,event\_tmp->socket\_fd,

event\_tmp->id,event\_tmp->seat,event\_tmp->value);

}

free(event\_tmp);

}

/\*

\* tcp\_ctrl\_data\_process.c

\*

\* Created on: 2016年10月14日

\* Author: leon

\*/

#include "tcp\_ctrl\_device\_status.h"

#include "tcp\_ctrl\_data\_compose.h"

#include "tcp\_ctrl\_device\_manage.h"

#include "sys\_uart\_init.h"

/\*

\* 定义两个结构体

\* 存取前状态，存储当前状态

\*

\* 创建查询线程，轮训比对，有差异，就返回新状态

\*/

extern Pglobal\_info node\_queue;

extern sys\_info sys\_in;

/\*

\* tcp\_ctrl\_source\_dest\_setting

\* tcp控制模块数据源地址和目标地址设置

\*

\* in：

\* @s\_fd 源套接字号

\* @d\_fd 目标套接字号

\* out：

\* @Pframe\_type

\*

\* return：

\* @error

\* @success

\*/

int tcp\_ctrl\_source\_dest\_setting(int s\_fd,int d\_fd,Pframe\_type type)

{

pclient\_node tmp = NULL;

Pclient\_info cinfo;

Pconference\_list info;

int i,j;

i = j = 0;

type->d\_id = type->s\_id = 0;

tmp = node\_queue->sys\_list[CONFERENCE\_LIST]->next;

while(tmp!=NULL)

{

info = tmp->data;

if(info->fd == d\_fd)

{

i++;

type->d\_id = info->con\_data.id;

}

if(info->fd == s\_fd)

{

j++;

type->s\_id = info->con\_data.id;

}

if(i>0 && j>0)

break;

tmp = tmp->next;

}

/\*

\* 查找上位机

\* 如果有，将地址信息赋值

\*/

tmp = node\_queue->sys\_list[CONNECT\_LIST]->next;

while(tmp!=NULL)

{

cinfo = tmp->data;

if(cinfo->client\_fd == d\_fd

&& cinfo->client\_name == PC\_CTRL)

{

i++;

type->d\_id = PC\_ID;

}

if(cinfo->client\_fd == s\_fd

&& cinfo->client\_name == PC\_CTRL)

{

j++;

type->s\_id = PC\_ID;

}

if(i>0 || j>0)

break;

tmp = tmp->next;

}

return SUCCESS;

}

/\*

\* tcp\_ctrl\_data\_char\_to\_int

\* tcp控制模块数据源地址和目标地址设置

\*

\* in：

\* @s\_fd 源套接字号

\* @d\_fd 目标套接字号

\* out：

\* @Pframe\_type

\*

\* return：

\* @error

\* @success

\*/

int tcp\_ctrl\_data\_char2short(unsigned short\* value,unsigned char\* buf)

{

unsigned short tmp = 0;

tmp = (buf[0] & 0xff) << 8 ;

tmp = tmp + (buf[1] & 0xff);

\*value = tmp;

return SUCCESS;

}

/\*

\* tcp\_ctrl\_msg\_send\_to

\* 处理后的消息进行何种处理，是发送给上位机还是在主机进行显示

\* 通过判断是否有上位机进行分别处理，有上位机的情况，主机就不进行显示提示

\*

\*/

int tcp\_ctrl\_msg\_send\_to(Pframe\_type type,const unsigned char\* msg,int value)

{

pclient\_node tmp = NULL;

Pclient\_info pinfo;

struct sockaddr\_in cli\_addr;

int clilen = sizeof(cli\_addr);

int status = 0;

int tmp\_fd = type->fd;

//本地状态上报qt

tcp\_ctrl\_report\_enqueue(type,value);

/\*

\* 查询是否有上位机连接，若有，将信息上报给上位机

\* 单元机上线后，需要上报上位机，上报内容为fd和ip地址

\* msg重新组包

\*/

if(type->dev\_type == UNIT\_CTRL && conf\_status\_get\_pc\_staus() > 0)

{

/\*

\* 单元机上线和下线再有上位机的情况下

\* 需要上报fd、id、ip、席别、电量

\*/

if(type->msg\_type == ONLINE\_REQ ||

type->msg\_type == OFFLINE\_REQ)

{

getpeername(type->fd,(struct sockaddr\*)&cli\_addr,

(socklen\_t\*)&clilen);

type->name\_type[0] = WIFI\_MEETING\_EVT\_UNIT\_ONOFF\_LINE;

type->code\_type[0] = WIFI\_MEETING\_STRING;

type->evt\_data.unet\_info.sockfd = type->fd;

type->evt\_data.unet\_info.ip = cli\_addr.sin\_addr.s\_addr;

status++;

}

tmp=node\_queue->sys\_list[CONNECT\_LIST]->next;;

while(tmp != NULL)

{

pinfo = tmp->data;

if((pinfo->client\_name == PC\_CTRL) &&

(conf\_status\_get\_pc\_staus() == pinfo->client\_fd))

{

type->evt\_data.status = WIFI\_MEETING\_HOST\_REP\_TO\_PC;

//上报上位机的源地址统一为sockfd

type->d\_id = PC\_ID;

type->fd = pinfo->client\_fd;

if(status > 0)

{

tcp\_ctrl\_module\_edit\_info(type,NULL);

}else{

type->s\_id = tmp\_fd;

tcp\_ctrl\_module\_edit\_info(type,msg);

}

break;

}

tmp = tmp->next;

}

}

return SUCCESS;

}

/\*

\* tcp\_ctrl\_uevent\_request\_pwr

\* 电源管理，主席单元下发一键关闭单元机功能

\*

\* 主机将请求消息发送给主机， 主机下发关机指令

\*

\* 主机界面提示关闭电源消息

\*

\*/

static int tcp\_ctrl\_uevent\_request\_pwr(Pframe\_type frame\_type,const unsigned char\* msg)

{

int tmp\_fd = 0;

char tmp\_msg = 0;

int pos = 0;

int value = 0;

if(sys\_debug\_get\_switch())

{

printf("%s-%s-%d,value=%d\n",\_\_FILE\_\_,\_\_func\_\_,\_\_LINE\_\_,

frame\_type->evt\_data.value);

}

tmp\_fd = frame\_type->fd;

tmp\_msg = frame\_type->msg\_type;

switch(frame\_type->evt\_data.value)

{

case WIFI\_MEETING\_EVT\_PWR\_OFF\_ALL:

/\*

\* 检查是否是主席单元下发的指令

\*/

pos = conf\_status\_check\_chairman\_legal(frame\_type);

if(pos > 0)

{

value = WIFI\_MEETING\_EVENT\_POWER\_OFF\_ALL;

//变换为控制类消息下个给单元机

frame\_type->msg\_type = WRITE\_MSG;

frame\_type->evt\_data.status = value;

tcp\_ctrl\_module\_edit\_info(frame\_type,msg);

}else{

printf("%s-%s-%d not the chariman command\n",\_\_FILE\_\_,\_\_func\_\_,\_\_LINE\_\_);

return ERROR;

}

break;

default:

printf("%s-%s-%d there is not legal value\n",\_\_FILE\_\_,\_\_func\_\_,\_\_LINE\_\_);

return ERROR;

}

frame\_type->fd = tmp\_fd;

frame\_type->msg\_type = tmp\_msg;

if(value)

{

tcp\_ctrl\_msg\_send\_to(frame\_type,msg,value);

}

return SUCCESS;

}

/\*

\* tcp\_ctrl\_uevent\_spk\_port

\* 音频端口管理

\* 主席单元固定为第一号端口，列席单元为后续的端口号

\*

\* @Pframe\_type

\* @value

\*

\*/

int tcp\_ctrl\_uevent\_spk\_port(Pframe\_type frame\_type)

{

int tmp = 0;

int tmp\_num,tmp\_port = 0;

int tmp\_fd = frame\_type->fd;

frame\_type->msg\_type = WRITE\_MSG;

frame\_type->dev\_type = HOST\_CTRL;

frame\_type->name\_type[0] = WIFI\_MEETING\_EVT\_AD\_PORT;

frame\_type->code\_type[0] = WIFI\_MEETING\_CHAR;

switch(frame\_type->evt\_data.status)

{

case WIFI\_MEETING\_EVENT\_SPK\_ALLOW:

case WIFI\_MEETING\_EVENT\_SPK\_REQ\_SPK:

{

if(frame\_type->con\_data.seat == WIFI\_MEETING\_CON\_SE\_CHAIRMAN)

{

tmp\_port = AUDIO\_RECV\_PORT;

/\*

\* 判断当前发言人数是否饱和，如果饱和，则需要进行处理

\* 人数饱满，则需要关闭最后上线的设备

\*/

if(conf\_status\_get\_cspk\_num() == conf\_status\_get\_spk\_num())

{

conf\_status\_close\_last\_spk\_client(frame\_type);

}else{

tmp = conf\_status\_get\_cspk\_num();

tmp++;

conf\_status\_set\_cspk\_num(tmp);

}

conf\_status\_set\_cmspk(WIFI\_MEETING\_CON\_SE\_CHAIRMAN);

frame\_type->fd = tmp\_fd;

frame\_type->name\_type[0] = WIFI\_MEETING\_EVT\_AD\_PORT;

}else if(frame\_type->con\_data.seat == WIFI\_MEETING\_CON\_SE\_GUEST

|| frame\_type->con\_data.seat == WIFI\_MEETING\_CON\_SE\_ATTEND)

{

/\*

\* 判断当前发言人数是否饱和，如果饱和，则需要进行处理

\* 已达到上限，则需要关闭第一个单元

\* 轮询发言链表中的设备，比较时间戳， 时间戳最小的，表示最早加入设备，需要关闭

\*/

if(conf\_status\_get\_cspk\_num() == conf\_status\_get\_spk\_num())

{

/\*

\* 人数已满，需要进行管理，关闭最早上线单元

\*/

conf\_status\_close\_first\_spk\_client(frame\_type);

/\*

\* 更新发言链表，下发端口信息

\*/

tmp\_port = frame\_type->spk\_port;

frame\_type->fd = tmp\_fd;

frame\_type->name\_type[0] = WIFI\_MEETING\_EVT\_AD\_PORT;

}else{

//列席单元是从9002端口开始

tmp\_port = AUDIO\_RECV\_PORT+2;

/\*

\* 查找申请者是否重复申请，若是，则直接下发原来原来的端口，并删除原来信息

\*/

tmp\_num = conf\_status\_get\_cspk\_num();

if(conf\_status\_get\_cmspk() == WIFI\_MEETING\_CON\_SE\_CHAIRMAN)

{

tmp\_num--;

}

switch(tmp\_num)

{

case 0:

case 1:

case 2:

case 3:

case 4:

case 5:

case 6:

case 7:

tmp\_port = tmp\_port+tmp\_num\*2;

//当前发言人数需要告知全局变量

if((tmp=conf\_status\_get\_cspk\_num()) < conf\_status\_get\_spk\_num())

{

tmp++;

conf\_status\_set\_cspk\_num(tmp);

}

break;

default:

{

frame\_type->name\_type[0] = WIFI\_MEETING\_EVT\_SPK;

frame\_type->evt\_data.value = WIFI\_MEETING\_EVT\_SPK\_VETO;

tcp\_ctrl\_source\_dest\_setting(-1,frame\_type->fd,frame\_type);

tcp\_ctrl\_module\_edit\_info(frame\_type,NULL);

printf("%s-%s-%d error\n",\_\_FILE\_\_,\_\_func\_\_,\_\_LINE\_\_);

return ERROR;

}

}

}

}

tcp\_ctrl\_source\_dest\_setting(-1,frame\_type->fd,frame\_type);

/\*

\* 下发给摄像

\*/

sys\_uart\_video\_set(frame\_type->d\_id,1);

/\*

\* 更新发言管理链表

\*/

frame\_type->spk\_port = tmp\_port;

conf\_status\_refresh\_spk\_node(frame\_type);

tcp\_ctrl\_module\_edit\_info(frame\_type,NULL);

break;

}

case WIFI\_MEETING\_EVENT\_SPK\_VETO:

{

frame\_type->name\_type[0] = WIFI\_MEETING\_EVT\_SPK;

frame\_type->evt\_data.value = WIFI\_MEETING\_EVT\_SPK\_VETO;

tcp\_ctrl\_source\_dest\_setting(-1,frame\_type->fd,frame\_type);

tcp\_ctrl\_module\_edit\_info(frame\_type,NULL);

break;

}

case WIFI\_MEETING\_EVENT\_SPK\_REQ\_SND:

case WIFI\_MEETING\_EVENT\_SPK\_ALOW\_SND:

{

conf\_status\_set\_snd\_brd(WIFI\_MEETING\_EVENT\_SPK\_REQ\_SND);

frame\_type->brd\_port = AUDIO\_SEND\_PORT;

tcp\_ctrl\_module\_edit\_info(frame\_type,NULL);

break;

}

case WIFI\_MEETING\_EVENT\_SPK\_CLOSE\_MIC:

tcp\_ctrl\_source\_dest\_setting(-1,frame\_type->fd,frame\_type);

sys\_uart\_video\_set(frame\_type->d\_id,0);

tmp=conf\_status\_get\_cspk\_num();

if(tmp)

{

tmp--;

conf\_status\_set\_cspk\_num(tmp);

}

if(frame\_type->con\_data.seat == WIFI\_MEETING\_CON\_SE\_CHAIRMAN)

{

conf\_status\_set\_cmspk(WIFI\_MEETING\_CON\_SE\_GUEST);

}

conf\_status\_refresh\_spk\_node(frame\_type);

break;

case WIFI\_MEETING\_EVENT\_SPK\_CHAIRMAN\_ONLY:

{

conf\_status\_close\_guest\_spk\_client(frame\_type);

break;

}

}

return SUCCESS;

}

/\*

\* tcp\_ctrl\_uevent\_spk\_manage

\* 发言管理

\* 1、处理单元机的请求情况，判断当前发言人数是否达到上限，达到则告知主席，否则直接下发端口号

\* 2、主席反馈的结果后，处理允许消息，并下发音频端口号

\*

\* @Pframe\_type

\* @value

\*

\*/

int tcp\_ctrl\_uevent\_spk\_manage(Pframe\_type frame\_type,const unsigned char\* msg)

{

int pos = 0;

int tmp\_fd = frame\_type->fd;

/\*

\* 判断当前请求单元机的席别属性

\* 主席就直接分配固定端口号

\* 列席就需要判断当前发言人数和设置的最大发言人数

\* 旁听则不允许申请发言

\*/

switch(frame\_type->evt\_data.status)

{

case WIFI\_MEETING\_EVENT\_SPK\_REQ\_SPK:

{

/\*

\* 判断申请发言单元的席别，如果是主席，则直接进行处理

\* 如果是单元，则需要判断发言人数限制和话筒模式

\*/

if(frame\_type->con\_data.seat == WIFI\_MEETING\_CON\_SE\_CHAIRMAN){

tcp\_ctrl\_uevent\_spk\_port(frame\_type);

}else{

/\*

\* 判断发言人数，如果发言人数为1，则表示主席独占模式

\* 发言人数为1 ，则直接拒绝单元申请

\* 发言人数大于1，则需要判断话筒模式和发言人数

\* FIFO模式，则直接关闭最先发言单元，新单元加入

\* 其他模式，先判断人，饱和则需要通知主席

\*/

if(conf\_status\_get\_spk\_num() == WIFI\_MEETING\_EVT\_MIC\_CHAIRMAN)

{

//告知单元，拒绝其发言

printf("%s-%s-%d only chairman mode\n",\_\_FILE\_\_,\_\_func\_\_,\_\_LINE\_\_);

frame\_type->msg\_type = WRITE\_MSG;

frame\_type->evt\_data.value = WIFI\_MEETING\_EVT\_SPK\_VETO;

tcp\_ctrl\_source\_dest\_setting(-1,frame\_type->fd,frame\_type);

tcp\_ctrl\_module\_edit\_info(frame\_type,NULL);

}else{

/\*

\* 判断发言话筒模式，如果是FIFO模式，则直接进行处理，不需要转发主席

\*/

if(conf\_status\_get\_mic\_mode() == WIFI\_MEETING\_EVT\_MIC\_FIFO)

{

tcp\_ctrl\_uevent\_spk\_port(frame\_type);

}else{

/\*

\* 判断发言人数是否饱和，如果达到上限，则需要请求主席授权

\*/

if(conf\_status\_get\_cspk\_num() < conf\_status\_get\_spk\_num())

{

tcp\_ctrl\_uevent\_spk\_port(frame\_type);

}else{

pos = conf\_status\_find\_chairman\_sockfd(frame\_type);

if(pos > 0){

//源地址为请求单元机id，目标地址修改为主机单元id

tcp\_ctrl\_source\_dest\_setting(tmp\_fd,frame\_type->fd,frame\_type);

tcp\_ctrl\_module\_edit\_info(frame\_type,msg);

}else{

//没有主席，告知单元，拒绝其发言

printf("%s-%s-%d not find chairman\n",\_\_FILE\_\_,\_\_func\_\_,\_\_LINE\_\_);

frame\_type->msg\_type = WRITE\_MSG;

frame\_type->evt\_data.value = WIFI\_MEETING\_EVT\_SPK\_VETO;

tcp\_ctrl\_source\_dest\_setting(-1,frame\_type->fd,frame\_type);

tcp\_ctrl\_module\_edit\_info(frame\_type,NULL);

}

}

}

}

}

break;

}

case WIFI\_MEETING\_EVENT\_SPK\_REQ\_SND:

{

tcp\_ctrl\_uevent\_spk\_port(frame\_type);

break;

}

case WIFI\_MEETING\_EVENT\_SPK\_ALLOW:

case WIFI\_MEETING\_EVENT\_SPK\_VETO:

{

pos = conf\_status\_check\_chairman\_legal(frame\_type);

if(pos > 0){

pos = 0;

pos = conf\_status\_find\_did\_sockfd\_id(frame\_type);

if(pos > 0)

{

printf("reply request\n");

//变换为控制类消息下个给单元机

frame\_type->msg\_type = WRITE\_MSG;

tcp\_ctrl\_source\_dest\_setting(-1,frame\_type->fd,frame\_type);

tcp\_ctrl\_module\_edit\_info(frame\_type,msg);

//设置单元音频端口信息

if(conf\_status\_check\_client\_connect\_legal(frame\_type)

&& frame\_type->evt\_data.status == WIFI\_MEETING\_EVENT\_SPK\_ALLOW)

{

tcp\_ctrl\_uevent\_spk\_port(frame\_type);

}

}else{

printf("%s-%s-%d not the find the unit device\n",\_\_FILE\_\_,\_\_func\_\_,\_\_LINE\_\_);

return ERROR;

}

}else{

printf("%s-%s-%d not the chariman command\n",\_\_FILE\_\_,\_\_func\_\_,\_\_LINE\_\_);

return ERROR;

}

break;

}

case WIFI\_MEETING\_EVENT\_SPK\_ALOW\_SND:

case WIFI\_MEETING\_EVENT\_SPK\_VETO\_SND:

{

pos = conf\_status\_check\_chairman\_legal(frame\_type);

if(pos > 0){

pos = 0;

pos = conf\_status\_find\_did\_sockfd\_id(frame\_type);

if(pos > 0)

{

printf("reply sound request\n");

//变换为控制类消息下个给单元机

frame\_type->msg\_type = WRITE\_MSG;

tcp\_ctrl\_source\_dest\_setting(-1,frame\_type->fd,frame\_type);

tcp\_ctrl\_module\_edit\_info(frame\_type,msg);

//设置单元音频端口信息

if(frame\_type->evt\_data.status == WIFI\_MEETING\_EVENT\_SPK\_ALOW\_SND)

tcp\_ctrl\_uevent\_spk\_port(frame\_type);

}else{

printf("%s-%s-%d not the find the unit device\n",\_\_FILE\_\_,\_\_func\_\_,\_\_LINE\_\_);

return ERROR;

}

}else{

printf("%s-%s-%d not the chariman command\n",\_\_FILE\_\_,\_\_func\_\_,\_\_LINE\_\_);

return ERROR;

}

break;

}

case WIFI\_MEETING\_EVENT\_SPK\_CLOSE\_MIC:

case WIFI\_MEETING\_EVENT\_SPK\_CHAIRMAN\_ONLY:

{

tcp\_ctrl\_uevent\_spk\_port(frame\_type);

break;

}

/\*

\* 单元请求发言手动关闭

\*/

case WIFI\_MEETING\_EVENT\_SPK\_CLOSE\_REQ:

{

pos = conf\_status\_find\_chairman\_sockfd(frame\_type);

if(pos > 0){

//源地址为请求单元机id，目标地址修改为主机单元id

tcp\_ctrl\_source\_dest\_setting(tmp\_fd,frame\_type->fd,frame\_type);

tcp\_ctrl\_module\_edit\_info(frame\_type,msg);

}else{

printf("%s-%s-%d not have chariman \n",\_\_FILE\_\_,\_\_func\_\_,\_\_LINE\_\_);

}

break;

}

case WIFI\_MEETING\_EVENT\_SPK\_CLOSE\_SND:

conf\_status\_set\_snd\_brd(WIFI\_MEETING\_EVENT\_SPK\_CLOSE\_SND);

break;

}

return SUCCESS;

}

/\*

\* tcp\_ctrl\_uevent\_request\_spk

\* 发言管理函数

\*

\* 主机将请求消息发送给主席单元，主席单元显示几号单元机请求发言

\*

\* 主机界面显示请求发言，如果主席单元处理请求后，显示关闭

\*

\*/

int tcp\_ctrl\_uevent\_request\_spk(Pframe\_type frame\_type,const unsigned char\* msg)

{

int value = 0;

char tmp\_msg = 0;

int tmp\_fd = 0;

if(sys\_debug\_get\_switch())

{

printf("%s-%s-%d,value=%d\n",\_\_FILE\_\_,\_\_func\_\_,\_\_LINE\_\_,

frame\_type->evt\_data.value);

}

//保存初始源地址信息

tmp\_fd = frame\_type->fd;

tmp\_msg = frame\_type->msg\_type;

switch(frame\_type->evt\_data.value)

{

/\*

\* 主机收到主席单元的允许指令，目标地址为具体允许的单元机ID，源地址为主席单元

\* 主机下发允许指令，修改源地址为主机，目标地址不变

\*/

case WIFI\_MEETING\_EVT\_SPK\_ALLOW:

value = WIFI\_MEETING\_EVENT\_SPK\_ALLOW;

break;

case WIFI\_MEETING\_EVT\_SPK\_VETO:

value = WIFI\_MEETING\_EVENT\_SPK\_VETO;

break;

case WIFI\_MEETING\_EVT\_SPK\_ALOW\_SND:

value = WIFI\_MEETING\_EVENT\_SPK\_ALOW\_SND;

break;

case WIFI\_MEETING\_EVT\_SPK\_VETO\_SND:

value = WIFI\_MEETING\_EVENT\_SPK\_VETO\_SND;

break;

case WIFI\_MEETING\_EVT\_SPK\_REQ\_SND:

value = WIFI\_MEETING\_EVENT\_SPK\_REQ\_SND;

break;

case WIFI\_MEETING\_EVT\_SPK\_REQ\_SPK:

value = WIFI\_MEETING\_EVENT\_SPK\_REQ\_SPK;

break;

case WIFI\_MEETING\_EVT\_SPK\_CLOSE\_MIC:

value = WIFI\_MEETING\_EVENT\_SPK\_CLOSE\_MIC;

break;

case WIFI\_MEETING\_EVT\_SPK\_CLOSE\_REQ:

value = WIFI\_MEETING\_EVENT\_SPK\_CLOSE\_REQ;

break;

case WIFI\_MEETING\_EVT\_SPK\_CHAIRMAN\_ONLY:

value = WIFI\_MEETING\_EVENT\_SPK\_CHAIRMAN\_ONLY;

break;

default:

printf("%s-%s-%d there is not legal value\n",\_\_FILE\_\_,\_\_func\_\_,\_\_LINE\_\_);

return ERROR;

}

frame\_type->evt\_data.status = value;

tcp\_ctrl\_uevent\_spk\_manage(frame\_type,msg);

/\*

\* 将事件信息发送至消息队列

\* 告知应用

\*/

if(value)

{

frame\_type->fd = tmp\_fd;

frame\_type->msg\_type = tmp\_msg;

tcp\_ctrl\_msg\_send\_to(frame\_type,msg,value);

}

return SUCCESS;

}

/\*

\* tcp\_ctrl\_uevent\_request\_subject

\* 会议议题控制管理

\*

\* 会议中主席单元会选择进行第几个议题，主机接收到具体的议题号，下发给单元机和上报给上位机

\* 主机单元通过全局变量保存当前议题号

\*

\*/

int tcp\_ctrl\_uevent\_request\_subject(Pframe\_type frame\_type,const unsigned char\* msg)

{

int value = 0;

int pos = 0;

char tmp\_msgt = 0;

int tmp\_fd = 0;

tmp\_fd = frame\_type->fd;

tmp\_msgt = frame\_type->msg\_type;

printf("%s-%s-%d,value=%d\n",\_\_FILE\_\_,\_\_func\_\_,\_\_LINE\_\_,

frame\_type->evt\_data.value);

/\*

\* 判断消息是否是主席单元发送

\*/

pos = conf\_status\_check\_chairman\_legal(frame\_type);

if(pos > 0)

{

frame\_type->msg\_type = WRITE\_MSG;

value = frame\_type->evt\_data.value;

//议题号保存到全局变量

conf\_status\_set\_current\_subject(value);

frame\_type->evt\_data.status = WIFI\_MEETING\_EVENT\_SUBJECT\_OFFSET;

tcp\_ctrl\_module\_edit\_info(frame\_type,msg);

}else{

printf("the subject command is not chariman send\n");

return ERROR;

}

/\*

\* 判断议题号

\*/

value += WIFI\_MEETING\_EVENT\_SUBJECT\_OFFSET;

/\*

\* 请求消息发送给上位机和主机显示

\*/

frame\_type->fd = tmp\_fd;

frame\_type->msg\_type = tmp\_msgt;

if(value > 0)

{

tcp\_ctrl\_msg\_send\_to(frame\_type,msg,value);

}

return SUCCESS;

}

/\*

\* tcp\_ctrl\_uevent\_request\_service

\* 服务请求

\* 主机接收端服务请求，将请求在主机界面显示或上传上位机

\*

\*/

int tcp\_ctrl\_uevent\_request\_service(Pframe\_type frame\_type,const unsigned char\* msg)

{

int value = 0;

printf("%s-%s-%d,value=%d\n",\_\_FILE\_\_,\_\_func\_\_,\_\_LINE\_\_,

frame\_type->evt\_data.value);

switch(frame\_type->evt\_data.value)

{

case WIFI\_MEETING\_EVT\_SER\_WATER:

value = WIFI\_MEETING\_EVENT\_SERVICE\_WATER;

break;

case WIFI\_MEETING\_EVT\_SER\_PEN:

value = WIFI\_MEETING\_EVENT\_SERVICE\_PEN;

break;

case WIFI\_MEETING\_EVT\_SER\_PAPER:

value = WIFI\_MEETING\_EVENT\_SERVICE\_PAPER;

break;

default:

printf("there is not legal value\n");

return ERROR;

}

tcp\_ctrl\_msg\_send\_to(frame\_type,msg,value);

return SUCCESS;

}

/\*

\* tcp\_ctrl\_uevent\_request\_checkin

\* 处理单元机签到功能

\* 单元机请求已经签到的信息

\*

\* 将数据信息上报给上位机

\*

\*/

int tcp\_ctrl\_uevent\_request\_checkin(Pframe\_type frame\_type,const unsigned char\* msg)

{

// pclient\_node tmp = NULL;

// Pconference\_list con\_list;

int value = 0;

int pos = 0;

int tmp\_fd = 0;

char tmp\_msgt = 0;

tmp\_fd = frame\_type->fd;

tmp\_msgt = frame\_type->msg\_type;

printf("%s-%s-%d,value=%d\n",\_\_FILE\_\_,\_\_func\_\_,\_\_LINE\_\_,

frame\_type->evt\_data.value);

/\*

\* 开始签到和结束签到的指令需要下发给所有单元

\*/

switch(frame\_type->evt\_data.value)

{

case WIFI\_MEETING\_EVT\_CHECKIN\_START:

value = WIFI\_MEETING\_EVENT\_CHECKIN\_START;

break;

case WIFI\_MEETING\_EVT\_CHECKIN\_END:

value = WIFI\_MEETING\_EVENT\_CHECKIN\_END;

break;

case WIFI\_MEETING\_EVT\_CHECKIN\_SELECT:

//保存签到信息

value = WIFI\_MEETING\_EVENT\_CHECKIN\_SELECT;

break;

default:

printf("there is not legal value\n");

return ERROR;

}

//开始和结束的时候需要告知列席

if(value == WIFI\_MEETING\_EVENT\_CHECKIN\_START ||

value == WIFI\_MEETING\_EVENT\_CHECKIN\_END)

{

/\*

\* 验证指令是否是主席发送

\*/

pos = conf\_status\_check\_chairman\_legal(frame\_type);

if(pos > 0)

{

frame\_type->msg\_type = WRITE\_MSG;

frame\_type->dev\_type = HOST\_CTRL;

frame\_type->evt\_data.status = value;

tcp\_ctrl\_module\_edit\_info(frame\_type,msg);

}else{

printf("the checkin command is not chariman send\n");

return ERROR;

}

}

frame\_type->fd = tmp\_fd;

frame\_type->msg\_type = tmp\_msgt;

if(value > 0)

{

tcp\_ctrl\_msg\_send\_to(frame\_type,msg,value);

}

return SUCCESS;

}

/\*

\* tcp\_ctrl\_uevent\_request\_vote

\* 处理单元机投票实时信息

\* 开始投票，结束投票

\* 赞成，反对，弃权，超时

\*

\* 将数据信息上报给上位机

\*

\*/

int tcp\_ctrl\_uevent\_request\_vote(Pframe\_type frame\_type,const unsigned char\* msg)

{

int value = 0;

int pos = 0;

int tmp\_fd = 0;

int tmp\_msgt = 0;

printf("%s-%s-%d,value=%d\n",\_\_FILE\_\_,\_\_func\_\_,\_\_LINE\_\_,

frame\_type->evt\_data.value);

tmp\_fd = frame\_type->fd;

tmp\_msgt = frame\_type->msg\_type;

/\*

\* 开始投票和结束投票的指令需要下发给所有单元

\*/

switch(frame\_type->evt\_data.value)

{

case WIFI\_MEETING\_EVT\_VOT\_START:

value = WIFI\_MEETING\_EVENT\_VOTE\_START;

break;

case WIFI\_MEETING\_EVT\_VOT\_END:

value = WIFI\_MEETING\_EVENT\_VOTE\_END;

break;

case WIFI\_MEETING\_EVT\_VOT\_RESULT:

value = WIFI\_MEETING\_EVENT\_VOTE\_RESULT;

break;

case WIFI\_MEETING\_EVT\_VOT\_ASSENT:

value = WIFI\_MEETING\_EVENT\_VOTE\_ASSENT;

break;

case WIFI\_MEETING\_EVT\_VOT\_NAY:

value = WIFI\_MEETING\_EVENT\_VOTE\_NAY;

break;

case WIFI\_MEETING\_EVT\_VOT\_WAIVER:

value = WIFI\_MEETING\_EVENT\_VOTE\_WAIVER;

break;

case WIFI\_MEETING\_EVT\_VOT\_TOUT:

value = WIFI\_MEETING\_EVENT\_VOTE\_TIMEOUT;

break;

default:

printf("there is not legal value\n");

return ERROR;

}

//开始和结束的时候需要告知列席

if(value == WIFI\_MEETING\_EVENT\_VOTE\_START || value == WIFI\_MEETING\_EVENT\_VOTE\_END)

{

/\*

\* 检查是否是主席单元下发的指令

\*/

pos = conf\_status\_check\_chairman\_legal(frame\_type);

if(pos > 0)

{

//变换为控制类消息下个给单元机

frame\_type->msg\_type = WRITE\_MSG;

frame\_type->evt\_data.status = value;

tcp\_ctrl\_module\_edit\_info(frame\_type,msg);

}else{

printf("%s-%s-%d not the chariman command\n",\_\_FILE\_\_,\_\_func\_\_,\_\_LINE\_\_);

return ERROR;

}

}else if(value == WIFI\_MEETING\_EVENT\_VOTE\_RESULT)

{

conf\_status\_send\_vote\_result();

}else if(value > WIFI\_MEETING\_EVENT\_VOTE\_RESULT){

//进行投票结果处理

conf\_status\_save\_vote\_result(value);

}

frame\_type->fd = tmp\_fd;

frame\_type->msg\_type = tmp\_msgt;

if(value > 0)

{

tcp\_ctrl\_msg\_send\_to(frame\_type,msg,value);

}

return SUCCESS;

}

/\*

\* tcp\_ctrl\_uevent\_request\_election

\* 选举管理

\*

\* 单元机请求消息选举

\* 将数据信息上报给上位机

\* 通过选举人编号，

\*

\*/

int tcp\_ctrl\_uevent\_request\_election(Pframe\_type frame\_type,const unsigned char\* msg)

{

int tmp\_fd = 0;

int tmp\_msgt = 0;

int pos = 0;

int value = 0;

printf("%s-%s-%d,value=%d\n",\_\_FILE\_\_,\_\_func\_\_,\_\_LINE\_\_,

frame\_type->evt\_data.value);

tmp\_fd = frame\_type->fd;

tmp\_msgt = frame\_type->msg\_type;

switch(frame\_type->evt\_data.value)

{

case WIFI\_MEETING\_EVT\_ELECTION\_START:

value = WIFI\_MEETING\_CONF\_ELECTION\_START;

break;

case WIFI\_MEETING\_EVT\_ELECTION\_END:

value = WIFI\_MEETING\_CONF\_ELECTION\_END;

break;

case WIFI\_MEETING\_EVT\_ELECTION\_RESULT:

value = WIFI\_MEETING\_CONF\_ELECTION\_RESULT;

break;

default:

//fixme 选举人编号，设置对应编号值累加

if((frame\_type->evt\_data.value - WIFI\_MEETING\_EVT\_ELECTION\_RESULT)

< conf\_status\_get\_elec\_totalp(0))

{

value = frame\_type->evt\_data.value - WIFI\_MEETING\_EVT\_ELECTION\_RESULT;

conf\_status\_save\_elec\_result(value);

value = WIFI\_MEETING\_CONF\_ELECTION\_UNDERWAY;

}

break;

}

//开始和结束的时候需要告知列席单元

if(value == WIFI\_MEETING\_CONF\_ELECTION\_START || value == WIFI\_MEETING\_CONF\_ELECTION\_END)

{

/\*

\* 检查是否是主席单元下发的指令

\*/

pos = conf\_status\_check\_chairman\_legal(frame\_type);

if(pos > 0)

{

//变换为控制类消息下个给单元机

frame\_type->msg\_type = WRITE\_MSG;

frame\_type->evt\_data.status = value;

tcp\_ctrl\_module\_edit\_info(frame\_type,msg);

}else{

printf("%s-%s-%d not the chariman command\n",\_\_FILE\_\_,\_\_func\_\_,\_\_LINE\_\_);

return ERROR;

}

}else if( value == WIFI\_MEETING\_CONF\_ELECTION\_RESULT)

{

//收到结束指令后，延时1ms下发选举结果

conf\_status\_send\_elec\_result();

}

frame\_type->fd = tmp\_fd;

frame\_type->msg\_type = tmp\_msgt;

if(value > 0)

{

tcp\_ctrl\_msg\_send\_to(frame\_type,msg,value);

}

return SUCCESS;

}

/\*

\* tcp\_ctrl\_uevent\_request\_score

\* 评分管理

\* 单元机请求消息计分

\*

\* 将数据信息上报给上位机

\*

\*/

int tcp\_ctrl\_uevent\_request\_score(Pframe\_type frame\_type,const unsigned char\* msg)

{

int tmp\_fd = 0;

int tmp\_msgt = 0;

int pos = 0;

int value = 0;

printf("%s-%s-%d,value=%d\n",\_\_FILE\_\_,\_\_func\_\_,\_\_LINE\_\_,

frame\_type->evt\_data.value);

tmp\_fd = frame\_type->fd;

tmp\_msgt = frame\_type->msg\_type;

switch(frame\_type->evt\_data.value)

{

case WIFI\_MEETING\_EVT\_SCORE\_START:

value = WIFI\_MEETING\_CONF\_SCORE\_START;

break;

case WIFI\_MEETING\_EVT\_SCORE\_END:

value = WIFI\_MEETING\_CONF\_SCORE\_END;

break;

case WIFI\_MEETING\_EVT\_SCORE\_RESULT:

value = WIFI\_MEETING\_CONF\_SCORE\_RESULT;

break;

default:

//fixme 计分议题,不能超过编号人数

if(conf\_status\_get\_score\_totalp() < node\_queue->sys\_list[CONFERENCE\_LIST]->size)

{

conf\_status\_save\_score\_result(frame\_type->evt\_data.value);

value = WIFI\_MEETING\_CONF\_SCORE\_UNDERWAY;

}

break;

}

if(value == WIFI\_MEETING\_CONF\_SCORE\_START || value == WIFI\_MEETING\_CONF\_SCORE\_END)

{

/\*

\* 检查是否是主席单元下发的指令

\*/

pos = conf\_status\_check\_chairman\_legal(frame\_type);

if(pos > 0)

{

//变换为控制类消息下个给单元机

frame\_type->msg\_type = WRITE\_MSG;

frame\_type->evt\_data.status = value;

tcp\_ctrl\_module\_edit\_info(frame\_type,msg);

}else{

printf("%s-%s-%d not the chariman command\n",\_\_FILE\_\_,\_\_func\_\_,\_\_LINE\_\_);

return ERROR;

}

}else if(value == WIFI\_MEETING\_CONF\_SCORE\_RESULT)

{

conf\_status\_send\_score\_result();

}

frame\_type->fd = tmp\_fd;

frame\_type->msg\_type = tmp\_msgt;

if(value > 0)

{

tcp\_ctrl\_msg\_send\_to(frame\_type,msg,value);

}

return SUCCESS;

}

/\*

\* tcp\_ctrl\_uevent\_request\_conf\_manage

\* 会议管理

\*

\* 开始会议和结束会议

\*/

int tcp\_ctrl\_uevent\_request\_conf\_manage(Pframe\_type frame\_type,const unsigned char\* msg)

{

int pos = 0;

int value = 0;

char tmp\_msgt = 0;

int tmp\_fd = 0;

printf("%s-%s-%d,value=%d\n",\_\_FILE\_\_,\_\_func\_\_,\_\_LINE\_\_,

frame\_type->evt\_data.value);

//保存初始源地址信息

tmp\_fd = frame\_type->fd;

tmp\_msgt = frame\_type->msg\_type;

switch(frame\_type->evt\_data.value)

{

case WIFI\_MEETING\_EVT\_CON\_MAG\_START:

value = WIFI\_MEETING\_EVENT\_CON\_MAG\_START;

break;

case WIFI\_MEETING\_EVT\_CON\_MAG\_END:

value = WIFI\_MEETING\_EVENT\_CON\_MAG\_END;

break;

}

if(value == WIFI\_MEETING\_EVENT\_CON\_MAG\_END ||

value == WIFI\_MEETING\_EVENT\_CON\_MAG\_START)

{

pos = conf\_status\_check\_chairman\_legal(frame\_type);

if(pos > 0)

{

//变换为控制类消息下个给单元机

frame\_type->msg\_type = WRITE\_MSG;

frame\_type->dev\_type = HOST\_CTRL;

frame\_type->evt\_data.status = value;

tcp\_ctrl\_module\_edit\_info(frame\_type,msg);

conf\_status\_set\_conf\_staus(value);

}else{

printf("%s-%s-%d not the chariman command\n",\_\_FILE\_\_,\_\_func\_\_,\_\_LINE\_\_);

return ERROR;

}

}

frame\_type->fd = tmp\_fd;

frame\_type->msg\_type = tmp\_msgt;

if(value > 0)

{

tcp\_ctrl\_msg\_send\_to(frame\_type,msg,value);

}

return SUCCESS;

}

/\*

\* tcp\_ctrl\_uevent\_request\_conf\_manage

\* 电量状态请求，主机需要转发上位机

\*

\* 开始会议和结束会议

\*/

int tcp\_ctrl\_uevent\_request\_electricity(Pframe\_type frame\_type,const unsigned char\* msg)

{

int value = 0;

value = WIFI\_MEETING\_EVENT\_UNIT\_ELECTRICITY;

tcp\_ctrl\_msg\_send\_to(frame\_type,msg,value);

return SUCCESS;

}

/\*

\* tcp\_ctrl\_unit\_request\_msg

\* 单元机请求消息处理函数

\* 请求消息再细分为事件类和会议类

\*

\* 处理接收到的消息，将消息内容进行解析，存取到全局结构体中@new\_uint\_data

\*

\* 事件类已增加发言，投票，签到，服务

\*

\* 会议类暂时没有

\* 消息格式

\* char | char | char

\* 内容 编码 数值

\*

\* in:

\* @msg数据内容

\* @Pframe\_type帧信息

\*

\* 处理请求类消息有两种情况：

\* 1/有上位机的情况，这个时候，系统初始化时单元机上线，不做特殊处理，如果是会议中途加入则需要自动对其编号

\* 2/没有上位机情况，系统初始化时，不做任何处理,会议过程中，则需要自动对新上线设备编号

\*

\*/

int tcp\_ctrl\_unit\_request\_msg(const unsigned char\* msg,Pframe\_type type)

{

int pos = 0;

/\*

\* 检查设备的合法性

\* 赋值id和席别,上报qt

\* 发言管理功能，是特殊情况，任何时间单元机上线后，均有可能请求音频功能

\*/

pos = conf\_status\_check\_client\_conf\_legal(type);

if(pos > 0){

/\*

\* 判读是事件型还是会议型

\*/

switch(type->data\_type)

{

case EVENT\_DATA:

{

type->name\_type[0] = msg[0];

type->code\_type[0] = msg[1];

type->evt\_data.value = msg[2];

switch(type->name\_type[0])

{

/\*

\* 电源管理

\*/

case WIFI\_MEETING\_EVT\_PWR:

tcp\_ctrl\_uevent\_request\_pwr(type,msg);

break;

/\*

\* 发言管理

\*/

case WIFI\_MEETING\_EVT\_SPK:

tcp\_ctrl\_uevent\_request\_spk(type,msg);

break;

/\*

\* 投票管理

\*/

case WIFI\_MEETING\_EVT\_VOT:

tcp\_ctrl\_uevent\_request\_vote(type,msg);

break;

/\*

\* 议题管理

\*/

case WIFI\_MEETING\_EVT\_SUB:

tcp\_ctrl\_uevent\_request\_subject(type,msg);

break;

/\*

\* 服务请求

\*/

case WIFI\_MEETING\_EVT\_SER:

tcp\_ctrl\_uevent\_request\_service(type,msg);

break;

/\*

\* 签到管理

\*/

case WIFI\_MEETING\_EVT\_CHECKIN:

tcp\_ctrl\_uevent\_request\_checkin(type,msg);

break;

/\*

\* 选举管理

\*/

case WIFI\_MEETING\_EVT\_ELECTION:

tcp\_ctrl\_uevent\_request\_election(type,msg);

break;

/\*

\* 评分管理

\*/

case WIFI\_MEETING\_EVT\_SCORE:

tcp\_ctrl\_uevent\_request\_score(type,msg);

break;

/\*

\* 会议管理

\*/

case WIFI\_MEETING\_EVT\_CON\_MAG:

tcp\_ctrl\_uevent\_request\_conf\_manage(type,msg);

break;

/\*

\* 单元电量

\*/

case WIFI\_MEETING\_EVT\_UNIT\_ELECTRICITY:

tcp\_ctrl\_uevent\_request\_electricity(type,msg);

break;

default:

printf("%s-%s-%d not legal event\n",\_\_FILE\_\_,\_\_func\_\_,\_\_LINE\_\_);

return ERROR;

}

}

break;

case CONFERENCE\_DATA:

break;

default:

printf("%s-%s-%d not legal event\n",\_\_FILE\_\_,\_\_func\_\_,\_\_LINE\_\_);

return ERROR;

}

}else{

printf("%s-%s-%d the client not in conference list\n",\_\_FILE\_\_,\_\_func\_\_,\_\_LINE\_\_);

return ERROR;

}

return SUCCESS;

}

/\*

\* tcp\_ctrl\_unit\_reply\_conference

\* 单元机会议类应答消息处理函数

\* 消息分为控制类应答和查询类应答

\* 消息会首先在主机做出处理，如果有上位机，还需通过上位机进行处理

\*

\* 处理接收到的消息，将消息内容进行解析

\*

\* in:

\* @msg数据内容

\* @Pframe\_type帧信息

\*

\*/

int tcp\_ctrl\_unit\_reply\_conference(const unsigned char\* msg,Pframe\_type frame\_type)

{

pclient\_node tmp = NULL;

Pconference\_list con\_list;

Pconference\_list confer\_info;

int tc\_index = 0;

int ret = 0;

int str\_len = 0;

int value = 0;

printf("%s-%s-%d ",\_\_FILE\_\_,\_\_func\_\_,\_\_LINE\_\_);

int i;

for(i=0;i<frame\_type->data\_len;i++)

{

printf("0x%02x ",msg[i]);

}

printf("\n");

/\*

\* 赋值id和席别

\* qt上报

\*/

tmp = node\_queue->sys\_list[CONFERENCE\_LIST]->next;

while(tmp!=NULL)

{

con\_list = tmp->data;

if(con\_list->fd == frame\_type->fd)

{

frame\_type->con\_data.id = frame\_type->s\_id;

frame\_type->con\_data.seat=con\_list->con\_data.seat;

break;

}

tmp=tmp->next;

}

switch(frame\_type->msg\_type)

{

/\*

\* 控制类应答是设置失败，单元机返回的

\* 所以，在接收到此消息后，主机要进行显示，还要将信息上报给上位机

\*/

case W\_REPLY\_MSG:

{

frame\_type->name\_type[0] = msg[tc\_index++];

/\*

\* 会议类控制应答，单元机只返回失败情况0xe3

\* 上报消息进入队列

\*/

if(msg[tc\_index++] == WIFI\_MEETING\_ERROR)

{

switch(msg[tc\_index++])

{

case TCP\_C\_ERR\_SUCCESS:

value = WIFI\_MEETING\_CONF\_WREP\_SUCCESS;

break;

case TCP\_C\_ERR\_HEAD:

case TCP\_C\_ERR\_LENGTH:

case TCP\_C\_ERR\_CHECKSUM:

value = WIFI\_MEETING\_CONF\_WREP\_ERR;

break;

}

}

break;

}

case R\_REPLY\_MSG:

{

/\*

\* 会议查询类应答，单元机应答全状态

\* 对于单主机情况，主机只关心ID 和席别

\* 有上位机情况，将应答信息转发给上位机

\*/

if(msg[tc\_index++] == WIFI\_MEETING\_CON\_ID){

--tc\_index;

frame\_type->name\_type[0] = msg[tc\_index++];

frame\_type->code\_type[0] = msg[tc\_index++];

frame\_type->con\_data.id = (msg[tc\_index++] << 8);

frame\_type->con\_data.id = frame\_type->con\_data.id+(msg[tc\_index++] << 0);

value = WIFI\_MEETING\_CONF\_RREP;

}

if(msg[tc\_index++] == WIFI\_MEETING\_CON\_SEAT){

--tc\_index;

frame\_type->name\_type[0] = msg[tc\_index++];

frame\_type->code\_type[0] = msg[tc\_index++];

frame\_type->con\_data.seat = msg[tc\_index++];

}

if(msg[tc\_index++] == WIFI\_MEETING\_CON\_NAME){

--tc\_index;

frame\_type->name\_type[0] = msg[tc\_index++];

frame\_type->code\_type[0] = msg[tc\_index++];

str\_len = msg[tc\_index++];

memcpy(frame\_type->con\_data.name,&msg[tc\_index],str\_len);

tc\_index = tc\_index+str\_len;

}

if(msg[tc\_index++] == WIFI\_MEETING\_CON\_CNAME){

--tc\_index;

frame\_type->name\_type[0] = msg[tc\_index++];

frame\_type->code\_type[0] = msg[tc\_index++];

str\_len = msg[tc\_index++];

memcpy(frame\_type->con\_data.conf\_name,&msg[tc\_index],str\_len);

tc\_index = tc\_index+str\_len;

}

if(value == WIFI\_MEETING\_CONF\_RREP)

{

printf("id=%d,seat=%d,name=%s,cname=%s\n",

frame\_type->con\_data.id,frame\_type->con\_data.seat,

frame\_type->con\_data.name,frame\_type->con\_data.conf\_name);

/\*

\* 更新conference\_head链表的内容

\*/

confer\_info = (Pconference\_list)malloc(sizeof(conference\_list));

memset(confer\_info,0,sizeof(conference\_list));

confer\_info->fd = frame\_type->fd;

confer\_info->con\_data.id = frame\_type->con\_data.id;

confer\_info->con\_data.seat = frame\_type->con\_data.seat;

if(strlen(frame\_type->con\_data.name) > 0)

{

memcpy(confer\_info->con\_data.name,frame\_type->con\_data.name,strlen(frame\_type->con\_data.name));

memcpy(confer\_info->con\_data.conf\_name,frame\_type->con\_data.conf\_name,strlen(frame\_type->con\_data.conf\_name));

}

/\*

\* 更新到会议信息链表中

\*/

ret = dmanage\_refresh\_conference\_list(confer\_info);

if(ret)

{

free(confer\_info);

return ERROR;

}

}

break;

}

}

/\*

\* 消息进行处理

\* 如果有PC就直接将数据传递给PC

\* 单主机的话，就需要上报给应用

\*/

if(value > 0)

{

tcp\_ctrl\_msg\_send\_to(frame\_type,msg,value);

}

return SUCCESS;

}

/\*

\* tcp\_ctrl\_unit\_reply\_event

\* 事件类应答消息处理

\*

\* 事件类应答不用区分应答类型，单元机应答的内容为固定格式

\* 消息内容为name+code+ID

\*

\*/

int tcp\_ctrl\_unit\_reply\_event(const unsigned char\* msg,Pframe\_type frame\_type)

{

pclient\_node tmp = NULL;

Pconference\_list tmp\_type;

int value = 0;

frame\_type->name\_type[0] = msg[0];

frame\_type->code\_type[0] = msg[1];

frame\_type->evt\_data.value = msg[2];

printf("%s-%s-%d,value=%d\n",\_\_FILE\_\_,\_\_func\_\_,\_\_LINE\_\_,

frame\_type->evt\_data.value);

/\*

\* 赋值id和席别

\* 在上报qt时需要

\*/

frame\_type->con\_data.id = frame\_type->s\_id;

//席别

tmp = node\_queue->sys\_list[CONFERENCE\_LIST]->next;

while(tmp!=NULL)

{

tmp\_type = tmp->data;

if(tmp\_type->con\_data.id == frame\_type->s\_id)

{

frame\_type->con\_data.seat=tmp\_type->con\_data.seat;

break;

}

tmp=tmp->next;

}

switch(frame\_type->name\_type[0])

{

/\*

\* 单元机电源

\*/

case WIFI\_MEETING\_EVT\_PWR:

{

switch(frame\_type->evt\_data.value)

{

case WIFI\_MEETING\_EVT\_PWR\_ON:

value = WIFI\_MEETING\_EVENT\_POWER\_ON;

break;

case WIFI\_MEETING\_EVT\_PWR\_OFF:

case WIFI\_MEETING\_EVT\_PWR\_OFF\_ALL:

value = WIFI\_MEETING\_EVENT\_POWER\_OFF;

break;

}

break;

}

/\*

\* 话筒应答

\*/

case WIFI\_MEETING\_EVT\_MIC:

{

switch(frame\_type->evt\_data.value)

{

case WIFI\_MEETING\_EVT\_MIC\_FIFO:

value = WIFI\_MEETING\_EVENT\_MIC\_FIFO;

break;

case WIFI\_MEETING\_EVT\_MIC\_STAD:

value = WIFI\_MEETING\_EVENT\_MIC\_STAD;

break;

case WIFI\_MEETING\_EVT\_MIC\_FREE:

value = WIFI\_MEETING\_EVENT\_MIC\_FREE;

break;

}

break;

}

/\*

\* 签到应答

\*/

case WIFI\_MEETING\_EVT\_CHECKIN:

{

switch(frame\_type->evt\_data.value)

{

case WIFI\_MEETING\_EVT\_CHECKIN\_START:

value = WIFI\_MEETING\_EVENT\_CHECKIN\_START;

break;

case WIFI\_MEETING\_EVT\_CHECKIN\_END:

value = WIFI\_MEETING\_EVENT\_CHECKIN\_END;

break;

}

break;

}

/\*

\* 发言管理应答

\*/

case WIFI\_MEETING\_EVT\_SPK:

{

switch(frame\_type->evt\_data.value)

{

case WIFI\_MEETING\_EVT\_SPK\_ALLOW:

value = WIFI\_MEETING\_EVENT\_SPK\_ALLOW;

break;

case WIFI\_MEETING\_EVT\_SPK\_VETO:

value = WIFI\_MEETING\_EVENT\_SPK\_VETO;

break;

case WIFI\_MEETING\_EVT\_SPK\_ALOW\_SND:

value = WIFI\_MEETING\_EVENT\_SPK\_ALOW\_SND;

break;

case WIFI\_MEETING\_EVT\_SPK\_VETO\_SND:

value = WIFI\_MEETING\_EVENT\_SPK\_VETO\_SND;

break;

case WIFI\_MEETING\_EVT\_SPK\_REQ\_SPK:

value = WIFI\_MEETING\_EVENT\_SPK\_REQ\_SPK;

break;

}

break;

}

/\*

\* 投票管理应答

\*/

case WIFI\_MEETING\_EVT\_VOT:

{

switch(frame\_type->evt\_data.value)

{

case WIFI\_MEETING\_EVT\_VOT\_START:

value = WIFI\_MEETING\_EVENT\_VOTE\_START;

break;

case WIFI\_MEETING\_EVT\_VOT\_END:

value = WIFI\_MEETING\_EVENT\_VOTE\_END;

break;

case WIFI\_MEETING\_EVT\_VOT\_ASSENT:

value = WIFI\_MEETING\_EVENT\_VOTE\_ASSENT;

break;

case WIFI\_MEETING\_EVT\_VOT\_NAY:

value = WIFI\_MEETING\_EVENT\_VOTE\_NAY;

break;

case WIFI\_MEETING\_EVT\_VOT\_WAIVER:

value = WIFI\_MEETING\_EVENT\_VOTE\_WAIVER;

break;

case WIFI\_MEETING\_EVT\_VOT\_TOUT:

value = WIFI\_MEETING\_EVENT\_VOTE\_TIMEOUT;

break;

}

break;

}

/\*

\* 议题管理应答

\*/

case WIFI\_MEETING\_EVT\_SUB:

{

value = frame\_type->evt\_data.value + SUBJECT\_OFFSET;

break;

}

/\*

\* ssid和key管理应答

\*/

case WIFI\_MEETING\_EVT\_SSID:

case WIFI\_MEETING\_EVT\_KEY:

{

value = WIFI\_MEETING\_EVENT\_SSID\_KEY;

break;

}

/\*

\* 选举管理应答

\*/

case WIFI\_MEETING\_EVT\_ELECTION:

{

switch(frame\_type->evt\_data.value)

{

case WIFI\_MEETING\_EVT\_ELECTION\_START:

value = WIFI\_MEETING\_CONF\_ELECTION\_START;

break;

case WIFI\_MEETING\_EVT\_ELECTION\_END:

value = WIFI\_MEETING\_CONF\_ELECTION\_END;

break;

}

break;

}

/\*

\* 计分管理应答

\*/

case WIFI\_MEETING\_EVT\_SCORE:

{

switch(frame\_type->evt\_data.value)

{

case WIFI\_MEETING\_EVT\_SCORE\_START:

value = WIFI\_MEETING\_CONF\_SCORE\_START;

break;

case WIFI\_MEETING\_EVT\_SCORE\_END:

value = WIFI\_MEETING\_CONF\_SCORE\_END;

break;

}

break;

}

/\*

\* 会议管理应答

\*/

case WIFI\_MEETING\_EVT\_CON\_MAG:

{

switch(frame\_type->evt\_data.value)

{

case WIFI\_MEETING\_EVT\_CON\_MAG\_START:

value = WIFI\_MEETING\_EVENT\_CON\_MAG\_START;

break;

case WIFI\_MEETING\_EVT\_CON\_MAG\_END:

value = WIFI\_MEETING\_EVENT\_CON\_MAG\_END;

break;

}

break;

}

}

if(value > 0)

{

tcp\_ctrl\_msg\_send\_to(frame\_type,msg,value);

}

return SUCCESS;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*

\* UNIT PART END

\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*

\* tcp\_ctrl\_device\_status.c

\*

\* Created on: 2016年10月20日

\* Author: leon

\*/

#include "tcp\_ctrl\_device\_status.h"

#include "tcp\_ctrl\_data\_compose.h"

extern sys\_info sys\_in;

extern Pglobal\_info node\_queue;

/\*

\* 系统调试开关设置

\*/

int sys\_debug\_set\_switch(unsigned char value)

{

node\_queue->con\_status->debug\_sw = value;

return SUCCESS;

}

/\*

\* 系统调试开关状态获取

\*/

int sys\_debug\_get\_switch()

{

return node\_queue->con\_status->debug\_sw;

}

/\*

\* tcp\_ctrl\_tprecv\_enqueue

\* tcp控制模块的数据发送队列

\* 将消息数据送入发送队列等待发送

\*

\* in:

\* @Pframe\_type 数据信息结构体

\* @msg 组包后的数据信息

\*

\* out:

\* @NULL

\*

\* return：

\* @ERROR

\* @SUCCESS

\*/

int tcp\_ctrl\_tprecv\_enqueue(int\* fd,unsigned char\* msg,int\* len)

{

Pctrl\_tcp\_rsqueue tmp;

// printf("%s-%s-%d\n",\_\_FILE\_\_,\_\_func\_\_,\_\_LINE\_\_);

tmp = (Pctrl\_tcp\_rsqueue)malloc(sizeof(ctrl\_tcp\_rsqueue));

memset(tmp,0,sizeof(ctrl\_tcp\_rsqueue));

tmp->msg = malloc(\*len);

memset(tmp->msg,0,\*len);

/\*

\* 单元机固有属性

\*/

tmp->socket\_fd = \*fd;

tmp->len = \*len;

memcpy(tmp->msg,msg,tmp->len);

enter\_queue(node\_queue->sys\_queue[CTRL\_TCP\_RECV\_QUEUE],tmp);

sem\_post(&sys\_in.sys\_sem[CTRL\_TCP\_RECV\_SEM]);

return SUCCESS;

}

/\*

\* tcp\_ctrl\_tprecv\_dequeue

\* tcp控制模块的接收的数据处队列

\*

\* in:

\* @Pframe\_type 数据信息结构体

\* @msg 组包后的数据信息

\*

\*

\* in/out:

\* @Pctrl\_tcp\_rsqueue

\*

\* return：

\* @ERROR

\* @SUCCESS

\*/

int tcp\_ctrl\_tprecv\_dequeue(Pctrl\_tcp\_rsqueue msg\_tmp)

{

int ret;

Plinknode node;

Pctrl\_tcp\_rsqueue tmp;

sem\_wait(&sys\_in.sys\_sem[CTRL\_TCP\_RECV\_SEM]);

// printf("%s-%s-%d\n",\_\_FILE\_\_,\_\_func\_\_,\_\_LINE\_\_);

ret = out\_queue(node\_queue->sys\_queue[CTRL\_TCP\_RECV\_QUEUE],&node);

if(ret == 0)

{

tmp = node->data;

msg\_tmp->socket\_fd = tmp->socket\_fd;

msg\_tmp->len = tmp->len;

memcpy(msg\_tmp->msg,tmp->msg,tmp->len);

free(tmp->msg);

free(tmp);

free(node);

}else{

return ERROR;

}

return SUCCESS;

}

/\*

\* tcp\_ctrl\_tcp\_send\_enqueue

\* tcp控制模块的数据发送队列

\* 将消息数据送入发送队列等待发送

\*

\* in:

\* @Pframe\_type 数据信息结构体

\* @msg 组包后的数据信息

\*

\* out:

\* @NULL

\*

\* return：

\* @ERROR

\* @SUCCESS

\*/

int tcp\_ctrl\_tpsend\_enqueue(Pframe\_type frame\_type,unsigned char\* msg)

{

Pctrl\_tcp\_rsqueue tmp;

tmp = (Pctrl\_tcp\_rsqueue)malloc(sizeof(ctrl\_tcp\_rsqueue));

memset(tmp,0,sizeof(ctrl\_tcp\_rsqueue));

tmp->msg = malloc(frame\_type->frame\_len);

memset(tmp->msg,0,frame\_type->frame\_len);

// printf("%s-%s-%d\n",\_\_FILE\_\_,\_\_func\_\_,\_\_LINE\_\_);

#if TCP\_DBG

printf("tcp\_ctrl\_tpsend\_enqueue the queue..\n");

#endif

/\*

\* 单元机固有属性

\*/

tmp->socket\_fd = frame\_type->fd;

tmp->len = frame\_type->frame\_len;

memcpy(tmp->msg,msg,frame\_type->frame\_len);

enter\_queue(node\_queue->sys\_queue[CTRL\_TCP\_SEND\_QUEUE],tmp);

sem\_post(&sys\_in.sys\_sem[CTRL\_TCP\_SEND\_SEM]);

return SUCCESS;

}

/\*

\* tcp\_ctrl\_tpsend\_dequeue

\* tcp控制模块的数据出队列

\* 将消息数据送入发送队列等待发送

\*

\* in/out:

\* @Ptcp\_send 数据信息结构体

\*

\* return：

\* @ERROR

\* @SUCCESS

\*/

int tcp\_ctrl\_tpsend\_dequeue(Pctrl\_tcp\_rsqueue event\_tmp)

{

int ret;

Plinknode node;

Pctrl\_tcp\_rsqueue tmp;

sem\_wait(&sys\_in.sys\_sem[CTRL\_TCP\_SEND\_SEM]);

#if TCP\_DBG

printf("get the value from tcp\_ctrl\_tpsend\_outqueue queue\n");

#endif

ret = out\_queue(node\_queue->sys\_queue[CTRL\_TCP\_SEND\_QUEUE],&node);

if(ret == 0)

{

tmp = node->data;

event\_tmp->socket\_fd = tmp->socket\_fd;

event\_tmp->len = tmp->len;

memcpy(event\_tmp->msg,tmp->msg,tmp->len);

free(tmp->msg);

free(tmp);

free(node);

}else{

return ERROR;

}

return SUCCESS;

}

/\*

\* tcp\_ctrl\_report\_enqueue

\* 上报数据数据发送队列

\* 将消息数据送入发送队列等待发送

\*

\* in:

\* @Pframe\_type 数据信息结构体

\* @value 事件信息

\*

\* out:

\* @NULL

\*

\* return：

\* @ERROR

\* @SUCCESS

\*/

int tcp\_ctrl\_report\_enqueue(Pframe\_type type,int value)

{

Prun\_status tmp;

tmp = (Prun\_status)malloc(sizeof(run\_status));

memset(tmp,0,sizeof(run\_status));

// printf("%s-%s-%d,value:%d\n",\_\_FILE\_\_,\_\_func\_\_,

// \_\_LINE\_\_,value);

/\*

\* 单元机固有属性

\*/

tmp->socket\_fd = type->fd;

tmp->id = type->con\_data.id;

tmp->seat = type->con\_data.seat;

tmp->value = value;

#if TCP\_DBG

printf("enter the tcp\_ctrl\_report\_enqueue..\n");

#endif

pthread\_mutex\_lock(&sys\_in.sys\_mutex[LOCAL\_REP\_MUTEX]);

enter\_queue(node\_queue->sys\_queue[LOCAL\_REP\_QUEUE],tmp);

pthread\_mutex\_unlock(&sys\_in.sys\_mutex[LOCAL\_REP\_MUTEX]);

sem\_post(&sys\_in.sys\_sem[LOCAL\_REP\_SEM]);

return SUCCESS;

}

int tcp\_ctrl\_report\_dequeue(Prun\_status event\_tmp)

{

int ret;

Plinknode node;

Prun\_status tmp;

sem\_wait(&sys\_in.sys\_sem[LOCAL\_REP\_SEM]);

pthread\_mutex\_lock(&sys\_in.sys\_mutex[LOCAL\_REP\_MUTEX]);

ret = out\_queue(node\_queue->sys\_queue[LOCAL\_REP\_QUEUE],&node);

pthread\_mutex\_unlock(&sys\_in.sys\_mutex[LOCAL\_REP\_MUTEX]);

if(ret == 0)

{

tmp = node->data;

memcpy(event\_tmp,tmp,sizeof(run\_status));

free(tmp);

free(node);

}else{

return ERROR;

}

return SUCCESS;

}

/\*

\* conf\_status\_get\_connected\_len

\* 获取会议中连接设备的数量

\*

\*/

int conf\_status\_get\_connected\_len()

{

return node\_queue->sys\_list[CONNECT\_LIST]->size;

}

/\*

\* conf\_status\_get\_conference\_len

\* 获取会议中参会设备数量

\*

\*/

int conf\_status\_get\_conference\_len()

{

return node\_queue->sys\_list[CONFERENCE\_LIST]->size;

}

/\*

\* conf\_status\_check\_client\_legal

\* 检查数据包来源合法性，会议表中是否有此设备

\* 并将此设备的席位赋值给临时变量

\*

\* @Pframe\_type

\*

\*/

int conf\_status\_check\_client\_conf\_legal(Pframe\_type frame\_type)

{

pclient\_node tmp = NULL;

Pconference\_list con\_list;

int pos = 0;

tmp = node\_queue->sys\_list[CONFERENCE\_LIST]->next;

while(tmp!=NULL)

{

con\_list = tmp->data;

if(con\_list->fd == frame\_type->fd

&& frame\_type->s\_id == con\_list->con\_data.id)

{

frame\_type->con\_data.seat = con\_list->con\_data.seat;

frame\_type->con\_data.id = con\_list->con\_data.id;

pos++;

break;

}

tmp=tmp->next;

}

return pos;

}

/\*

\* conf\_status\_check\_connect\_legal

\* 检查连接信息中，设备的合法性

\*

\* @Pframe\_type

\*

\*/

int conf\_status\_check\_client\_connect\_legal(Pframe\_type frame\_type)

{

pclient\_node tmp = NULL;

Pclient\_info cnet\_list;

int pos = 0;

tmp = node\_queue->sys\_list[CONNECT\_LIST]->next;

while(tmp!=NULL)

{

cnet\_list = tmp->data;

if(cnet\_list->client\_fd == frame\_type->fd)

{

frame\_type->con\_data.seat = cnet\_list->seat;

pos++;

break;

}

tmp=tmp->next;

}

return pos;

}

/\*

\* conf\_status\_check\_chariman\_legal

\* 检查数据包来源合法性，是否是主席单元发送的数据

\*

\* @Pframe\_type

\*

\*/

int conf\_status\_check\_chairman\_legal(Pframe\_type frame\_type)

{

pclient\_node tmp = NULL;

Pconference\_list con\_list;

int pos = 0;

/\*

\* 判断消息是否是主席单元发送

\*/

tmp = node\_queue->sys\_list[CONFERENCE\_LIST]->next;

while(tmp!=NULL)

{

con\_list = tmp->data;

if(con\_list->fd == frame\_type->fd)

{

if(con\_list->con\_data.seat == WIFI\_MEETING\_CON\_SE\_CHAIRMAN)

{

pos++;

break;

}

}

tmp=tmp->next;

}

return pos;

}

/\*

\* conf\_status\_check\_chariman\_staus

\* 检查会议中是否有主席单元存在

\*

\* 返回值

\* @主席状态0无1有

\*

\*/

int conf\_status\_check\_chariman\_staus()

{

pclient\_node tmp = NULL;

Pconference\_list con\_list;

int pos = 0;

/\*

\* 判断消息是否是主席单元发送

\*/

tmp = node\_queue->sys\_list[CONFERENCE\_LIST]->next;

while(tmp!=NULL)

{

con\_list = tmp->data;

if(con\_list->con\_data.seat == WIFI\_MEETING\_CON\_SE\_CHAIRMAN)

{

pos++;

break;

}

tmp=tmp->next;

}

return pos;

}

/\*

\* conf\_status\_find\_did\_sockfd

\* 检测临时变量中目标地址对应的客户端的sockfd

\*

\* @Pframe\_type

\*

\*/

int conf\_status\_find\_did\_sockfd\_id(Pframe\_type frame\_type)

{

pclient\_node tmp = NULL;

Pclient\_info cnet\_list;

int pos = 0;

tmp = node\_queue->sys\_list[CONNECT\_LIST]->next;

while(tmp!=NULL)

{

cnet\_list = tmp->data;

if(cnet\_list->id == frame\_type->d\_id)

{

frame\_type->fd=cnet\_list->client\_fd;

pos++;

break;

}

tmp=tmp->next;

}

printf("%s-%s-%d，did=%d,pos=%d\n",\_\_FILE\_\_,\_\_func\_\_,\_\_LINE\_\_,frame\_type->d\_id,pos);

return pos;

}

/\*

\* conf\_status\_get\_did\_sockfd

\* 检测临时变量中目标地址对应的客户端的sockfd

\*

\* @Pframe\_type

\*

\*/

int conf\_status\_find\_did\_sockfd\_sock(Pframe\_type frame\_type)

{

pclient\_node tmp = NULL;

Pclient\_info cnet\_list;

int pos = 0;

tmp = node\_queue->sys\_list[CONNECT\_LIST]->next;

while(tmp!=NULL)

{

cnet\_list = tmp->data;

if(cnet\_list->client\_fd == frame\_type->d\_id)

{

frame\_type->fd=cnet\_list->client\_fd;

pos++;

break;

}

tmp=tmp->next;

}

printf("%s-%s-%d，did=%d,pos=%d\n",\_\_FILE\_\_,\_\_func\_\_,\_\_LINE\_\_,frame\_type->d\_id,pos);

return pos;

}

/\*

\* conf\_status\_find\_chariman\_sockfd

\* 检测链表中，主席单元的sockfd

\*

\* @Pframe\_type

\*

\*/

int conf\_status\_find\_chairman\_sockfd(Pframe\_type frame\_type)

{

pclient\_node tmp = NULL;

Pconference\_list con\_list;

int pos = 0;

tmp = node\_queue->sys\_list[CONFERENCE\_LIST]->next;

while(tmp!=NULL)

{

con\_list = tmp->data;

if(con\_list->con\_data.seat == WIFI\_MEETING\_CON\_SE\_CHAIRMAN)

{

printf("find the chairman\n");

frame\_type->fd=con\_list->fd;

pos++;

break;

}

tmp=tmp->next;

}

return pos;

}

/\*

\* conf\_status\_find\_max\_id

\* 检测链表中，ID最大值

\*

\*/

int conf\_status\_find\_max\_id()

{

pclient\_node tmp = NULL;

Pconference\_list con\_list;

int max\_id = 0;

tmp = node\_queue->sys\_list[CONFERENCE\_LIST]->next;

while(tmp!=NULL)

{

con\_list = tmp->data;

if(con\_list->con\_data.id)

{

if(con\_list->con\_data.id > max\_id)

{

max\_id = con\_list->con\_data.id;

}

}

tmp=tmp->next;

}

return max\_id+1;

}

/\*

\* conf\_status\_compare\_id

\* 比较链表中id号

\*

\*/

int conf\_status\_compare\_id(int value)

{

pclient\_node tmp = NULL;

Pconference\_list con\_list;

int pos = 0;

tmp = node\_queue->sys\_list[CONFERENCE\_LIST]->next;

while(tmp!=NULL)

{

con\_list = tmp->data;

if(con\_list->con\_data.id == value)

{

pos++;

break;

}

tmp=tmp->next;

}

return pos;

}

/\*

\* conf\_status\_set\_current\_subject

\* 会议中投票议题的状态

\*

\*/

int conf\_status\_set\_current\_subject(unsigned char num)

{

node\_queue->con\_status->sub\_num = num;

printf("%s-%s-%d,value=%d\n",\_\_FILE\_\_,\_\_func\_\_,\_\_LINE\_\_,

node\_queue->con\_status->sub\_num);

return SUCCESS;

}

/\*

\* conf\_status\_get\_subject\_property

\* 会议中投票议题的状态

\*

\*

\*/

int conf\_status\_get\_current\_subject()

{

return node\_queue->con\_status->sub\_num;

}

/\*

\* conf\_status\_set\_subject\_property

\* 会议中投票议题的状态

\*

\*/

int conf\_status\_set\_subject\_property(unsigned char num,unsigned char prop)

{

printf("%s-%s-%d,num=%d,prop=%d\n",\_\_FILE\_\_,\_\_func\_\_,\_\_LINE\_\_,num,prop);

node\_queue->con\_status->sub\_list[num].subj\_prop = prop;

return SUCCESS;

}

/\*

\* conf\_status\_get\_subject\_property

\* 会议中投票议题的状态

\*

\*

\*/

int conf\_status\_get\_subject\_property(unsigned char num)

{

unsigned char sub\_num;

unsigned char prop = 0;

if(num == 0)

{

sub\_num = conf\_status\_get\_current\_subject();

printf("%s-%s-%d,prop=%d\n",\_\_FILE\_\_,\_\_func\_\_,\_\_LINE\_\_,

node\_queue->con\_status->sub\_list[sub\_num].subj\_prop);

prop = node\_queue->con\_status->sub\_list[sub\_num].subj\_prop;

}

else

{

printf("%s-%s-%d,prop=%d\n",\_\_FILE\_\_,\_\_func\_\_,\_\_LINE\_\_,

node\_queue->con\_status->sub\_list[num].subj\_prop);

prop = node\_queue->con\_status->sub\_list[num].subj\_prop;

}

return prop;

}

/\*

\* conf\_status\_save\_vote\_result

\* 会议中投票议题的状态

\*

\* @value投票结果

\*

\*/

int conf\_status\_save\_vote\_result(int value)

{

unsigned char sub\_num;

int sub\_prop = 0;

sub\_num = conf\_status\_get\_current\_subject();

printf("%s-%s-%d,sub\_num=%d\n",\_\_FILE\_\_,\_\_func\_\_,\_\_LINE\_\_,sub\_num);

//判断当前议题的属性

sub\_prop = conf\_status\_get\_subject\_property(sub\_num);

if(sub\_prop == WIFI\_MEETING\_CON\_SUBJ\_VOTE)

{

switch(value)

{

case WIFI\_MEETING\_EVENT\_VOTE\_ASSENT:

node\_queue->con\_status->sub\_list[sub\_num].v\_result.assent++;

break;

case WIFI\_MEETING\_EVENT\_VOTE\_NAY:

node\_queue->con\_status->sub\_list[sub\_num].v\_result.nay++;

break;

case WIFI\_MEETING\_EVENT\_VOTE\_WAIVER:

node\_queue->con\_status->sub\_list[sub\_num].v\_result.waiver++;

break;

case WIFI\_MEETING\_EVENT\_VOTE\_TIMEOUT:

node\_queue->con\_status->sub\_list[sub\_num].v\_result.timeout++;

break;

}

}else{

printf("%s-%s-%d the subject is not vote subject\n",\_\_FILE\_\_,\_\_func\_\_,\_\_LINE\_\_);

return ERROR;

}

return SUCCESS;

}

/\*

\* conf\_status\_get\_vote\_result

\* 会议中投票议题的状态

\*

\*

\*/

int conf\_status\_get\_vote\_result(unsigned char num,unsigned short\* value)

{

int c\_num = 0;

if(num == 0)

{

c\_num = conf\_status\_get\_current\_subject();

value[0] = node\_queue->con\_status->sub\_list[c\_num].v\_result.assent;

value[1] = node\_queue->con\_status->sub\_list[c\_num].v\_result.nay;

value[2] = node\_queue->con\_status->sub\_list[c\_num].v\_result.waiver;

value[3] = node\_queue->con\_status->sub\_list[c\_num].v\_result.timeout;

}else{

value[0] = node\_queue->con\_status->sub\_list[num].v\_result.assent;

value[1] = node\_queue->con\_status->sub\_list[num].v\_result.nay;

value[2] = node\_queue->con\_status->sub\_list[num].v\_result.waiver;

value[3] = node\_queue->con\_status->sub\_list[num].v\_result.timeout;

}

return SUCCESS;

}

/\*

\* conf\_status\_save\_elec\_result

\* 会议中投票议题的状态

\*

\* @value投票结果

\*

\*/

int conf\_status\_save\_elec\_result(unsigned short value)

{

unsigned char sub\_num = conf\_status\_get\_current\_subject();

int sub\_prop = 0;

//判断当前议题的属性

sub\_prop = conf\_status\_get\_subject\_property(sub\_num);

if(sub\_prop == WIFI\_MEETING\_CON\_SUBJ\_ELE)

{

node\_queue->con\_status->sub\_list[sub\_num].ele\_result.ele\_id[value]++;

}else{

printf("%s-%s-%d the subject is not ele subject\n",\_\_FILE\_\_,\_\_func\_\_,\_\_LINE\_\_);

return ERROR;

}

return SUCCESS;

}

/\*

\* conf\_status\_save\_vote\_result

\* 会议中投票议题的状态

\*

\* @value投票结果

\*

\*/

int conf\_status\_get\_elec\_result(unsigned char num,unsigned short value)

{

int sub\_num ;

if(num == 0){

sub\_num = conf\_status\_get\_current\_subject();

}else{

sub\_num = num;

}

return node\_queue->con\_status->sub\_list[sub\_num].ele\_result.ele\_id[value];

}

/\*

\* conf\_status\_set\_elec\_totalp

\* 议题中被选举总人数

\*

\* @value投票结果

\*

\*/

int conf\_status\_set\_elec\_totalp(unsigned char num,unsigned char pep)

{

node\_queue->con\_status->sub\_list[num].ele\_result.ele\_total = pep;

return SUCCESS;

}

/\*

\* conf\_status\_get\_elec\_totalp

\* 议题中被选举总人数

\*

\* @value投票结果

\*

\*/

int conf\_status\_get\_elec\_totalp(unsigned char num)

{

unsigned char sub\_num ;

int sub\_prop = 0;

if(num == 0){

sub\_num = conf\_status\_get\_current\_subject();

}else{

sub\_num = num;

}

//判断当前议题的属性

sub\_prop = conf\_status\_get\_subject\_property(sub\_num);

if(sub\_prop == WIFI\_MEETING\_CON\_SUBJ\_ELE)

{

return node\_queue->con\_status->sub\_list[sub\_num].ele\_result.ele\_total;

}else

{

printf("%s-%s-%d the subject is not elec subject\n",\_\_FILE\_\_,\_\_func\_\_,\_\_LINE\_\_);

return ERROR;

}

}

/\*

\* conf\_status\_save\_score\_result

\* 会议中投票议题的状态

\*

\* @value投票结果

\*

\*/

int conf\_status\_save\_score\_result(unsigned char value)

{

unsigned char sub\_num = conf\_status\_get\_current\_subject();

int sub\_prop = 0;

//判断当前议题的属性

sub\_prop = conf\_status\_get\_subject\_property(sub\_num);

if(sub\_prop == WIFI\_MEETING\_CON\_SUBJ\_SCORE)

{

node\_queue->con\_status->sub\_list[sub\_num].scr\_result.score += value;

node\_queue->con\_status->sub\_list[sub\_num].scr\_result.num\_peop++;

}else{

printf("%s-%s-%d the subject is not score subject\n",\_\_FILE\_\_,\_\_func\_\_,\_\_LINE\_\_);

return ERROR;

}

return SUCCESS;

}

/\*

\* conf\_status\_calc\_score\_result

\* 计算积分结果

\*

\* @value投票结果

\*

\*/

int conf\_status\_calc\_score\_result()

{

unsigned char value = 0;

int sub\_num = conf\_status\_get\_current\_subject();

value = node\_queue->con\_status->sub\_list[sub\_num].scr\_result.score /

node\_queue->con\_status->sub\_list[sub\_num].scr\_result.num\_peop;

node\_queue->con\_status->sub\_list[sub\_num].scr\_result.score\_r = value;

return SUCCESS;

}

/\*

\* conf\_status\_get\_score\_result

\* 获取计分结果

\*

\* @value投票结果

\*

\*/

int conf\_status\_get\_score\_result(Pscore\_result result)

{

int sub\_num = conf\_status\_get\_current\_subject();

result->num\_peop = conf\_status\_get\_score\_totalp();

result->score = node\_queue->con\_status->sub\_list[sub\_num].scr\_result.score;

result->score\_r = node\_queue->con\_status->sub\_list[sub\_num].scr\_result.score\_r;

return SUCCESS;

}

/\*

\* conf\_status\_get\_score\_totalp

\* 进行计分总人数

\*

\* @value投票结果

\*

\*/

int conf\_status\_get\_score\_totalp()

{

int sub\_num = conf\_status\_get\_current\_subject();

return node\_queue->con\_status->sub\_list[sub\_num].scr\_result.num\_peop;

}

/\*

\* TODO

\* 1/发言管理

\* 2/话筒管理

\* 3/音效管理

\* 4/会议管理

\*

\*/

/\*

\* conf\_status\_set\_cmspk

\* 会议中主席的发言状态

\*

\* @value

\*

\* 返回值：

\* @ERROR

\* @SUCCESS

\*/

int conf\_status\_set\_cmspk(int value){

node\_queue->con\_status->chirman\_t = value;

return SUCCESS;

}

/\*

\* conf\_status\_get\_cmspk

\* 会议中主席的发言状态

\*

\* 返回值：

\* @status

\*/

int conf\_status\_get\_cmspk(){

return node\_queue->con\_status->chirman\_t;

}

/\*

\* conf\_status\_set\_mic\_mode

\* 设置会议话筒模式

\*

\* @value FIFO模式(1)、标准模式(2)、自由模式(3)

\*

\*/

int conf\_status\_set\_mic\_mode(int value)

{

printf("%s-%s-%d,value=%d\n",\_\_FILE\_\_,\_\_func\_\_,\_\_LINE\_\_,

value);

value = value+WIFI\_MEETING\_EVT\_MIC\_CHAIRMAN;

node\_queue->con\_status->mic\_mode = value;

return SUCCESS;

}

/\*

\* conf\_status\_get\_mic\_mode

\* 获取话筒模式

\*

\* @value FIFO模式(1)、标准模式(2)、自由模式(3)

\*

\*/

int conf\_status\_get\_mic\_mode()

{

int mode;

mode = node\_queue->con\_status->mic\_mode;

printf("%s-%s-%d,value=%d\n",\_\_FILE\_\_,\_\_func\_\_,\_\_LINE\_\_,

mode);

return mode;

}

/\*

\* conf\_status\_set\_spk\_num

\* 会议最大发言人数设置

\*

\* @value投票结果

\*

\*/

int conf\_status\_set\_spk\_num(int value)

{

printf("%s-%s-%d,value=%d\n",\_\_FILE\_\_,\_\_func\_\_,\_\_LINE\_\_,

value);

node\_queue->con\_status->spk\_number = value;

return SUCCESS;

}

/\*

\* conf\_status\_get\_spk\_num

\* 会议最大发言人数获取

\*

\* @value投票结果

\*

\*/

int conf\_status\_get\_spk\_num()

{

return node\_queue->con\_status->spk\_number;

}

/\*

\* conf\_status\_set\_cspk\_num

\* 会议当前发言人数设置

\*

\* @value投票结果

\*

\*/

int conf\_status\_set\_cspk\_num(int value)

{

printf("%s-%s-%d,value=%d\n",\_\_FILE\_\_,\_\_func\_\_,\_\_LINE\_\_,

value);

node\_queue->con\_status->current\_spk = value;

return SUCCESS;

}

/\*

\* conf\_status\_get\_cspk\_num

\* 会议当前发言人数获取

\*

\*

\*/

int conf\_status\_get\_cspk\_num()

{

printf("%s-%s-%d,value=%d\n",\_\_FILE\_\_,\_\_func\_\_,\_\_LINE\_\_,

node\_queue->con\_status->current\_spk);

return node\_queue->con\_status->current\_spk;

}

/\*

\* conf\_status\_set\_snd\_brd

\* 会议当前音频下发状态

\*

\* @value投票结果

\*

\*/

int conf\_status\_set\_snd\_brd(int value)

{

printf("%s-%s-%d,value=%d\n",\_\_FILE\_\_,\_\_func\_\_,\_\_LINE\_\_,

value);

node\_queue->con\_status->snd\_brdcast = value;

return SUCCESS;

}

/\*

\* conf\_status\_get\_snd\_brd

\* 会议当前发言人数获取

\*

\*/

int conf\_status\_get\_snd\_brd()

{

return node\_queue->con\_status->snd\_brdcast;

}

/\*

\* conf\_status\_set\_snd\_effect

\* DSP音效设置

\* 采用为管理分别从bit[0-2]表示状态

\* bit 0 1 2

\* AFC ANC0 ANC1

\* ANC2

\* bit[0-2] 共有8个状态

\*

\* @value AFC(0/1)，ANC(0/1/2/3)

\*

\*

\* 返回值：

\* @ERROR

\* @SUCCESS

\*/

int conf\_status\_set\_snd\_effect(int value)

{

printf("%s-%s-%d,value=0x%02x\n",\_\_FILE\_\_,\_\_func\_\_,\_\_LINE\_\_,

value);

node\_queue->con\_status->snd\_effect = value;

uart\_snd\_effect\_set(value);

return SUCCESS;

}

/\*

\* conf\_info\_get\_snd\_effect

\* DSP音效获取

\*

\* @value AFC(0/1)，ANC(0/1/2/3)

\*

\* 返回值：

\* @ERROR

\* @SUCCESS

\*/

int conf\_status\_get\_snd\_effect()

{

printf("%s-%s-%d,value=0x%02x\n",\_\_FILE\_\_,\_\_func\_\_,\_\_LINE\_\_,

node\_queue->con\_status->snd\_effect);

return node\_queue->con\_status->snd\_effect;

}

/\*

\* conf\_status\_set\_camera\_track

\* 设置摄像跟踪设置

\*/

int conf\_status\_set\_camera\_track(int value)

{

node\_queue->con\_status->camera\_track = value;

return SUCCESS;

}

/\*

\* conf\_status\_get\_camera\_track

\* 摄像跟踪获取

\*/

int conf\_status\_get\_camera\_track()

{

return node\_queue->con\_status->camera\_track;

}

/\*

\* conf\_status\_set\_conf\_staus

\* 设置会议进程状态

\*

\* @value

\*

\*/

int conf\_status\_set\_conf\_staus(int value)

{

printf("%s-%s-%d,value=%d\n",\_\_FILE\_\_,\_\_func\_\_,\_\_LINE\_\_,

value);

node\_queue->con\_status->confer\_status = value;

return SUCCESS;

}

/\*

\* conf\_status\_get\_conf\_staus

\* 获取会议进程状态

\*

\* @value

\*

\*/

int conf\_status\_get\_conf\_staus()

{

printf("%s-%s-%d,value=%d\n",\_\_FILE\_\_,\_\_func\_\_,\_\_LINE\_\_,

node\_queue->con\_status->confer\_status);

return node\_queue->con\_status->confer\_status;

}

/\*

\* conf\_status\_set\_pc\_staus

\* 设置会议中上位机的连接状态

\*

\* @value

\*

\*/

int conf\_status\_set\_pc\_staus(int value)

{

printf("%s-%s-%d,value=%d\n",\_\_FILE\_\_,\_\_func\_\_,\_\_LINE\_\_,

value);

node\_queue->con\_status->pc\_status = value;

return SUCCESS;

}

/\*

\* conf\_status\_get\_pc\_staus

\* 获取会议进程中上位机的连接状态

\*

\* @value

\*

\*/

int conf\_status\_get\_pc\_staus()

{

// printf("%s-%s-%d,value=%d\n",\_\_FILE\_\_,\_\_func\_\_,\_\_LINE\_\_,

// node\_queue->con\_status->pc\_status);

return node\_queue->con\_status->pc\_status;

}

int conf\_status\_send\_vote\_result()

{

frame\_type data\_info;

memset(&data\_info,0,sizeof(frame\_type));

unsigned short result[4] = {0};

printf("%s-%s-%d\n",\_\_FILE\_\_,\_\_func\_\_,\_\_LINE\_\_);

/\*

\* 判断当前议题的属性，符合就下发，不是则返回错误码

\*/

if(conf\_status\_get\_subject\_property(0) == WIFI\_MEETING\_CON\_SUBJ\_VOTE)

{

data\_info.name\_type[0] = WIFI\_MEETING\_CON\_VOTE;

data\_info.code\_type[0] = WIFI\_MEETING\_STRING;

conf\_status\_get\_vote\_result(0,result);

data\_info.con\_data.v\_result.assent = result[0];

data\_info.con\_data.v\_result.nay = result[1];

data\_info.con\_data.v\_result.waiver = result[2];

data\_info.con\_data.v\_result.timeout = result[3];

data\_info.msg\_type = WRITE\_MSG;

data\_info.data\_type = CONFERENCE\_DATA;

data\_info.dev\_type = HOST\_CTRL;

tcp\_ctrl\_module\_edit\_info(&data\_info,NULL);

}else{

printf("%s-%s-%d not vote subject\n",\_\_FILE\_\_,\_\_func\_\_,\_\_LINE\_\_);

return ERROR;

}

return SUCCESS;

}

int conf\_status\_send\_elec\_result()

{

int i;

frame\_type data\_info;

memset(&data\_info,0,sizeof(frame\_type));

printf("%s-%s-%d\n",\_\_FILE\_\_,\_\_func\_\_,\_\_LINE\_\_);

/\*

\* 判断当前议题的属性，符合就下发，不是则返回错误码

\*/

if(conf\_status\_get\_subject\_property(0) == WIFI\_MEETING\_CON\_SUBJ\_ELE)

{

data\_info.name\_type[0] = WIFI\_MEETING\_CON\_ELEC;

data\_info.code\_type[0] = WIFI\_MEETING\_STRING;

data\_info.msg\_type = WRITE\_MSG;

data\_info.data\_type = CONFERENCE\_DATA;

data\_info.dev\_type = HOST\_CTRL;

//将全局变量中的数据保存到

for(i=1;i<=conf\_status\_get\_elec\_totalp(0);i++)

{

data\_info.con\_data.elec\_rsult.ele\_id[i] = conf\_status\_get\_elec\_result(0,i);

}

}else{

printf("%s-%s-%d not election subject\n",\_\_FILE\_\_,\_\_func\_\_,\_\_LINE\_\_);

return ERROR;

}

tcp\_ctrl\_module\_edit\_info(&data\_info,NULL);

return SUCCESS;

}

int conf\_status\_send\_score\_result()

{

score\_result result;

frame\_type data\_info;

memset(&data\_info,0,sizeof(frame\_type));

printf("%s-%s-%d\n",\_\_FILE\_\_,\_\_func\_\_,\_\_LINE\_\_);

/\*

\* 判断当前议题的属性，符合就下发，不是则返回错误码

\*/

if(conf\_status\_get\_subject\_property(0) == WIFI\_MEETING\_CON\_SUBJ\_SCORE)

{

conf\_status\_get\_score\_result(&result);

data\_info.name\_type[0] = WIFI\_MEETING\_CON\_SCORE;

data\_info.code\_type[0] = WIFI\_MEETING\_STRING;

data\_info.msg\_type = WRITE\_MSG;

data\_info.data\_type = CONFERENCE\_DATA;

data\_info.dev\_type = HOST\_CTRL;

data\_info.con\_data.src\_result.score\_r = result.score\_r;

}else{

printf("%s-%s-%d not score subject\n",\_\_FILE\_\_,\_\_func\_\_,\_\_LINE\_\_);

return ERROR;

}

tcp\_ctrl\_module\_edit\_info(&data\_info,NULL);

return SUCCESS;

}

/\*

\* tcp\_ctrl\_device\_manage.c

\*

\* Created on: 2016年12月21日

\* Author: leon

\*/

#include "tcp\_ctrl\_device\_status.h"

#include "tcp\_ctrl\_data\_process.h"

#include "tcp\_ctrl\_data\_compose.h"

#include "tcp\_ctrl\_device\_manage.h"

#include "sys\_uart\_init.h"

#include "audio\_ring\_buf.h"

extern Pglobal\_info node\_queue;

extern Paudio\_queue\* rqueue;

unsigned int spk\_ts = 0;

/\*

\* 关闭最后发言的设置

\*/

int conf\_status\_close\_last\_spk\_client(Pframe\_type type)

{

printf("%s-%s-%d\n",\_\_FILE\_\_,\_\_func\_\_,\_\_LINE\_\_);

/\*

\* 查询会议中排位，关闭时间戳最大的单元

\* 将端口下发给新申请的单元

\*

\*/

conf\_status\_search\_last\_spk\_node(type);

conf\_status\_delete\_spk\_node(type->fd);

type->name\_type[0] = WIFI\_MEETING\_EVT\_SPK;

type->evt\_data.value = WIFI\_MEETING\_EVT\_SPK\_VETO;

tcp\_ctrl\_source\_dest\_setting(-1,type->fd,type);

tcp\_ctrl\_module\_edit\_info(type,NULL);

return SUCCESS;

}

int conf\_status\_close\_first\_spk\_client(Pframe\_type type)

{

printf("%s-%s-%d\n",\_\_FILE\_\_,\_\_func\_\_,\_\_LINE\_\_);

/\*

\* 查询会议中排位，关闭时间戳最小的单元

\* 将端口下发给新申请的单元

\*

\*/

conf\_status\_search\_first\_spk\_node(type);

conf\_status\_delete\_spk\_node(type->fd);

type->name\_type[0] = WIFI\_MEETING\_EVT\_SPK;

type->evt\_data.value = WIFI\_MEETING\_EVT\_SPK\_VETO;

tcp\_ctrl\_source\_dest\_setting(-1,type->fd,type);

tcp\_ctrl\_module\_edit\_info(type,NULL);

return SUCCESS;

}

int conf\_status\_close\_guest\_spk\_client(Pframe\_type type)

{

pclient\_node tmp\_node;

Pas\_port sinfo;

type->name\_type[0] = WIFI\_MEETING\_EVT\_SPK;

type->evt\_data.value = WIFI\_MEETING\_EVT\_SPK\_VETO;

/\*

\* 查找连接信息中，是否有该设备存在

\*/

tmp\_node = node\_queue->sys\_list[CONFERENCE\_SPK]->next;

while(tmp\_node!=NULL)

{

sinfo = tmp\_node->data;

if(sinfo->seat != WIFI\_MEETING\_CON\_SE\_CHAIRMAN)

{

tcp\_ctrl\_source\_dest\_setting(-1,sinfo->sockfd,type);

tcp\_ctrl\_module\_edit\_info(type,NULL);

conf\_status\_delete\_spk\_node(sinfo->sockfd);

msleep(1);

}

tmp\_node=tmp\_node->next;

}

return SUCCESS;

}

int conf\_status\_delete\_spk\_node(int fd)

{

pclient\_node tmp\_node;

pclient\_node del;

Pas\_port sinfo;

int pos,status;

int num = 0;

pos = status =0;

/\*

\* 查找连接信息中，是否有该设备存在

\*/

tmp\_node = node\_queue->sys\_list[CONFERENCE\_SPK]->next;

while(tmp\_node!=NULL)

{

sinfo = tmp\_node->data;

if(sinfo->sockfd == fd)

{

/\*

\* 复位音频接收队列

\*/

num = (sinfo->asport -AUDIO\_RECV\_PORT)/2 + 1;

audio\_queue\_reset(rqueue[num]);

status++;

break;

}

tmp\_node=tmp\_node->next;

pos++;

}

if(status)

{

list\_delete(node\_queue->sys\_list[CONFERENCE\_SPK],pos,&del);

sinfo = del->data;

printf("%s-%s-%d,remove %d in spk list\n",\_\_FILE\_\_,\_\_func\_\_,\_\_LINE\_\_,

sinfo->sockfd);

free(sinfo);

free(del);

}else{

printf("%s-%s-%d,not have spk client\n",\_\_FILE\_\_,\_\_func\_\_,\_\_LINE\_\_);

return ERROR;

}

return SUCCESS;

}

int conf\_status\_add\_spk\_node(Pframe\_type type)

{

Pas\_port sinfo;

sinfo = malloc(sizeof(as\_port));

spk\_ts++;

sinfo->sockfd = type->fd;

sinfo->seat = type->con\_data.seat;

sinfo->asport = type->spk\_port;

sinfo->ts = spk\_ts;

printf("%s-%s-%d fd:%d,asport：%d,ts:%d\n",\_\_FILE\_\_,\_\_func\_\_,\_\_LINE\_\_,

sinfo->sockfd,sinfo->asport,sinfo->ts);

/\*

\* 发言信息存入

\*/

list\_add(node\_queue->sys\_list[CONFERENCE\_SPK],sinfo);

return SUCCESS;

}

/\*

\* conf\_status\_search\_spk\_node

\* 查找发言链表中的设备信息，时间戳最小的，将会返回fd，用于发送关闭

\*

\* 返回值：

\* @ERROR

\* @SUCCESS

\*

\*/

int conf\_status\_search\_first\_spk\_node(Pframe\_type type)

{

pclient\_node tmp\_node;

Pas\_port sinfo;

unsigned int ts = 0xFFFFFFFF;

/\*

\* 找出信息中的最小时间戳

\*/

tmp\_node = node\_queue->sys\_list[CONFERENCE\_SPK]->next;

while(tmp\_node!=NULL)

{

sinfo = tmp\_node->data;

if(sinfo->sockfd)

{

if((sinfo->ts < ts) &&

(sinfo->seat != WIFI\_MEETING\_CON\_SE\_CHAIRMAN))

{

ts = sinfo->ts;

}

}

tmp\_node=tmp\_node->next;

}

if(ts < 0xffffffff)

{

/\*

\* 找到最小时间戳的配置信息

\*/

tmp\_node = node\_queue->sys\_list[CONFERENCE\_SPK]->next;

while(tmp\_node!=NULL)

{

sinfo = tmp\_node->data;

if(sinfo->sockfd)

{

if(sinfo->ts == ts)

{

type->fd = sinfo->sockfd;

type->spk\_port = sinfo->asport;

break;

}

}

tmp\_node=tmp\_node->next;

}

}

return SUCCESS;

}

/\*

\* 查找发言中，最后上线的设备

\*

\*/

int conf\_status\_search\_last\_spk\_node(Pframe\_type type)

{

pclient\_node tmp\_node;

Pas\_port sinfo;

unsigned int ts = 0;

/\*

\* 找出信息中的最大时间戳

\*/

tmp\_node = node\_queue->sys\_list[CONFERENCE\_SPK]->next;

while(tmp\_node!=NULL)

{

sinfo = tmp\_node->data;

if(sinfo->sockfd)

{

if((sinfo->ts > ts) &&

(sinfo->seat != WIFI\_MEETING\_CON\_SE\_CHAIRMAN))

{

ts = sinfo->ts;

}

}

tmp\_node=tmp\_node->next;

}

if(ts > 0)

{

/\*

\* 找到最大设备的配置信息

\*/

tmp\_node = node\_queue->sys\_list[CONFERENCE\_SPK]->next;

while(tmp\_node!=NULL)

{

sinfo = tmp\_node->data;

if(sinfo->sockfd)

{

if(sinfo->ts == ts)

{

type->fd = sinfo->sockfd;

type->spk\_port = sinfo->asport;

break;

}

}

tmp\_node=tmp\_node->next;

}

}

return SUCCESS;

}

/\*

\* conf\_status\_refresh\_spk\_node

\* 更新会议链表中的发言管理设备信息

\*

\* 返回值：

\* @ERROR

\* @SUCCESS

\*

\*/

int conf\_status\_refresh\_spk\_node(Pframe\_type type)

{

int ret = 0;

switch(type->evt\_data.status)

{

case WIFI\_MEETING\_EVENT\_SPK\_ALLOW:

case WIFI\_MEETING\_EVENT\_SPK\_REQ\_SPK:

ret = conf\_status\_add\_spk\_node(type);

break;

case WIFI\_MEETING\_EVENT\_SPK\_CLOSE\_MIC:

ret = conf\_status\_delete\_spk\_node(type->fd);

break;

}

return ret;

}

int dmanage\_delete\_heart\_list(int fd)

{

pclient\_node tmp\_node;

pclient\_node del;

Pconnect\_heart hinfo;

int pos,status;

pos = status =0;

/\*

\* 查找连接信息中，是否有该设备存在

\*/

tmp\_node = node\_queue->sys\_list[CONNECT\_HEART]->next;

while(tmp\_node!=NULL)

{

hinfo = tmp\_node->data;

if(hinfo->sockfd == fd)

{

status++;

break;

}

tmp\_node=tmp\_node->next;

pos++;

}

if(status)

{

list\_delete(node\_queue->sys\_list[CONNECT\_HEART],pos,&del);

hinfo = del->data;

printf("%s-%s-%d,remove %d in heart list\n",\_\_FILE\_\_,\_\_func\_\_,\_\_LINE\_\_,hinfo->sockfd);

free(hinfo);

free(del);

}else{

printf("%s-%s-%d,not have heart client\n",\_\_FILE\_\_,\_\_func\_\_,\_\_LINE\_\_);

}

return SUCCESS;

}

/\*

\* dmanage\_set\_communication\_heart

\* 为连接设备增加心跳属性

\* 设置设备心跳阈值

\*/

int dmanage\_set\_communication\_heart(Pframe\_type type)

{

Pconnect\_heart hinfo;

hinfo = malloc(sizeof(connect\_heart));

hinfo->sockfd = type->fd;

hinfo->status = DEVICE\_HEART;

// printf("%s-%s-%d fd:%d,status：%d\n",\_\_FILE\_\_,\_\_func\_\_,\_\_LINE\_\_,

// hinfo->sockfd,hinfo->status);

/\*

\* 连接信息存入链表

\*/

list\_add(node\_queue->sys\_list[CONNECT\_HEART],hinfo);

return SUCCESS;

}

/\*

\* dmanage\_get\_communication\_heart

\* 获取设备心跳属性

\*

\* 获取一次，将心跳属性减一，心跳为0则需要返回此设备sockfd

\*/

int dmanage\_get\_communication\_heart(Pframe\_type type)

{

pclient\_node tmp\_node;

Pconnect\_heart hinfo;

int ret = 0;

tmp\_node = node\_queue->sys\_list[CONNECT\_HEART]->next;

while(tmp\_node!=NULL)

{

hinfo = tmp\_node->data;

if(hinfo->sockfd)

{

hinfo->status--;

if(!hinfo->status)

{

type->fd = hinfo->sockfd;

ret = conf\_status\_check\_chairman\_legal(type);

if(ret)

type->con\_data.seat = WIFI\_MEETING\_CON\_SE\_CHAIRMAN;

break;

}

}

tmp\_node=tmp\_node->next;

}

return SUCCESS;

}

/\*

\* dmanage\_process\_communication\_heart

\* 处理设备心跳请求

\* 1、在接收到终端消息就表示心跳正常

\* 2、收到消息类型为心跳，则需要对终端进行回复

\*/

int dmanage\_process\_communication\_heart(const unsigned char\* msg,Pframe\_type type)

{

pclient\_node tmp\_node;

pclient\_node del;

Pconnect\_heart hinfo;

int ret,pos,status;

pos = status = 0;

switch(type->msg\_type)

{

case REQUEST\_MSG:

case W\_REPLY\_MSG:

case R\_REPLY\_MSG:

case ONLINE\_HEART:

/\*

\* 查找连接信息中，是否有该设备存在

\*/

ret = conf\_status\_check\_client\_connect\_legal(type);

if(ret)

{

tmp\_node = node\_queue->sys\_list[CONNECT\_HEART]->next;

while(tmp\_node!=NULL)

{

hinfo = tmp\_node->data;

if(hinfo->sockfd == type->fd)

{

status++;

break;

}

tmp\_node=tmp\_node->next;

pos++;

}

}else{

printf("%s-%s-%d,not have heart client\n",\_\_FILE\_\_,\_\_func\_\_,\_\_LINE\_\_);

return ERROR;

}

break;

}

if(status)

{

list\_delete(node\_queue->sys\_list[CONNECT\_HEART],pos,&del);

hinfo = del->data;

free(hinfo);

free(del);

dmanage\_set\_communication\_heart(type);

if(type->msg\_type == ONLINE\_HEART)

{

type->d\_id = type->s\_id;

type->s\_id = 0;

type->evt\_data.status = WIFI\_MEETING\_EVENT\_DEVICE\_HEART;

tcp\_ctrl\_module\_edit\_info(type,msg);

}

}

return SUCCESS;

}

int dmanage\_delete\_connected\_list(int fd)

{

pclient\_node tmp = NULL;

pclient\_node del = NULL;

Pclient\_info pinfo;

Pclient\_info cinfo;

FILE\* file;

int ret;

int status = 0;

int pos = 0;

/\*

\* 删除链接信息链表中的客户端

\*/

tmp = node\_queue->sys\_list[CONNECT\_LIST]->next;

while(tmp != NULL)

{

pinfo = tmp->data;

if(pinfo->client\_fd == fd)

{

/\*

\* 上位机下线

\*/

if(pinfo->client\_name == PC\_CTRL)

{

conf\_status\_set\_pc\_staus(ERROR);

}

status++;

break;

}

pos++;

tmp = tmp->next;

}

if(status > 0)

{

list\_delete(node\_queue->sys\_list[CONNECT\_LIST],pos,&del);

pinfo = del->data;

printf("%s-%s-%d,remove %s in connected list\n",\_\_FILE\_\_,\_\_func\_\_,\_\_LINE\_\_,

inet\_ntoa(pinfo->cli\_addr.sin\_addr));

free(pinfo);

free(del);

}else{

printf("%s-%s-%d,there is no data in connected list\n",\_\_FILE\_\_,\_\_func\_\_,\_\_LINE\_\_);

return ERROR;

}

/\*

\* 更新文本文件

\*/

file = fopen(CONNECT\_FILE,"w+");

tmp = node\_queue->sys\_list[CONNECT\_LIST]->next;

while(tmp != NULL)

{

cinfo = tmp->data;

if(cinfo->client\_fd > 0)

{

printf("%s-%s-%d,add %s in txt\nn",\_\_FILE\_\_,\_\_func\_\_,\_\_LINE\_\_,

inet\_ntoa(cinfo->cli\_addr.sin\_addr));

ret = fwrite(cinfo,sizeof(client\_info),1,file);

perror("fwrite");

if(ret != 1)

return ERROR;

}

tmp = tmp->next;

usleep(1000);

}

fclose(file);

return SUCCESS;

}

int dmanage\_delete\_conference\_list(int fd)

{

pclient\_node tmp = NULL;

pclient\_node del = NULL;

Pconference\_list cinfo;

int status = 0;

int pos = 0;

/\*

\* 删除会议信息链表中的数据

\*/

tmp = node\_queue->sys\_list[CONFERENCE\_LIST]->next;

while(tmp != NULL)

{

cinfo = tmp->data;

if(cinfo->fd == fd)

{

status++;

break;

}

pos++;

tmp = tmp->next;

}

if(status > 0)

{

list\_delete(node\_queue->sys\_list[CONFERENCE\_LIST],pos,&del);

cinfo = del->data;

printf("%s-%s-%d remove %d in conference list\n",\_\_FILE\_\_,\_\_func\_\_,\_\_LINE\_\_,cinfo->fd);

free(cinfo);

free(del);

}else{

printf("%s-%s-%d there is no data in conference list\n",\_\_FILE\_\_,\_\_func\_\_,\_\_LINE\_\_);

return ERROR;

}

return SUCCESS;

}

/\*

\* dmanage\_delete\_info

\* 客户端 删除函数

\* 删除链表中的连接信息

\* 先通过查找链表，定位离线客户端的位置，在调用链表删除函数，删除链表中的设备

\*

\* 更新文本文件

\*/

int dmanage\_delete\_info(int fd)

{

int tmp = 0;

int ret = 0;

if(conf\_status\_get\_pc\_staus() != fd)

{

/\*

\* 下线消息告知主机或上位机

\*/

//上报上位机部分

frame\_type tmp\_type;

memset(&tmp\_type,0,sizeof(frame\_type));

tmp\_type.msg\_type = OFFLINE\_REQ;

tmp\_type.dev\_type = UNIT\_CTRL;

tmp\_type.data\_type = EVENT\_DATA;

tmp\_type.fd = fd;

tmp\_type.evt\_data.status = WIFI\_MEETING\_EVENT\_SPK\_CLOSE\_MIC;

ret = conf\_status\_refresh\_spk\_node(&tmp\_type);

if(!ret)

{

tmp=conf\_status\_get\_cspk\_num();

if(tmp)

{

tmp--;

conf\_status\_set\_cspk\_num(tmp);

}

if(conf\_status\_get\_cmspk() == WIFI\_MEETING\_CON\_SE\_CHAIRMAN)

{

conf\_status\_set\_cmspk(WIFI\_MEETING\_CON\_SE\_GUEST);

}

}

tcp\_ctrl\_msg\_send\_to(&tmp\_type,NULL,WIFI\_MEETING\_EVENT\_OFFLINE\_REQ);

dmanage\_delete\_heart\_list(fd);

dmanage\_delete\_conference\_list(fd);

}

dmanage\_delete\_connected\_list(fd);

return SUCCESS;

}

/\*

\* tcp\_ctrl\_add\_client

\* 终端信息录入函数

\*

\* 生成设备连接信息文本文件，采用结构体方式进行存储

\* 应用在设置参数时，需要读取文本文件中的fd信息，进行参数设置

\*

\*/

int dmanage\_add\_connected\_info(Pclient\_info value)

{

FILE\* file;

int ret;

/\*

\* 连接信息存入链表

\*/

list\_add(node\_queue->sys\_list[CONNECT\_LIST],value);

/\*

\* 存入到文本文件

\* 更新文本信息

\*/

file = fopen(CONNECT\_FILE,"a+");

printf("%s-%s-%d fd:%d,ip:%s,machine:%d\n",\_\_FILE\_\_,\_\_func\_\_,\_\_LINE\_\_,value->client\_fd,

inet\_ntoa(value->cli\_addr.sin\_addr),value->client\_name);

ret = fwrite(value,sizeof(client\_info),1,file);

perror("fwrite");

if(ret != 1)

return ERROR;

fclose(file);

return SUCCESS;

}

/\*

\* dmanage\_add\_info

\* 删除

\*

\*/

int dmanage\_add\_info(const unsigned char\* msg,Pframe\_type type)

{

/\*

\* 终端信息录入结构体

\*/

pclient\_node tmp\_node = NULL;

Pclient\_info info;

Pclient\_info pinfo;

Pconference\_list confer\_info;

struct sockaddr\_in cli\_addr;

int clilen = sizeof(cli\_addr);

int state = 0;

/\*

\* 宣告上线消息类型

\*/

// if(type->msg\_type == ONLINE\_REQ)

// {

/\*

\* 将新连接终端信息录入结构体中

\* 这里可以将fd和IP信息存入到本地链表和文件中

\*

\* 区分上线的设备类型，单元机还是主机

\*

\*/

info = (Pclient\_info)malloc(sizeof(client\_info));

memset(info,0,sizeof(client\_info));

info->client\_fd = type->fd;

getpeername(info->client\_fd,(struct sockaddr\*)&cli\_addr,(socklen\_t\*)&clilen);

info->cli\_addr = cli\_addr;

info->clilen = clilen;

sprintf(info->ip,"%s",inet\_ntoa(info->cli\_addr.sin\_addr));

if(type->dev\_type == PC\_CTRL){

info->client\_name = PC\_CTRL;

conf\_status\_set\_pc\_staus(type->fd);

//上位机ID号位0xffff，无席别

if(type->s\_id > 0)

{

info->id = type->s\_id;

}

}else{

info->client\_name = UNIT\_CTRL;

info->seat = type->con\_data.seat = msg[0];

info->id = type->s\_id;

//增加电量信息

info->client\_power = type->evt\_data.electricity = msg[1];

}

/\*

\* 检查该客户端是否已经存在

\*

\* 链表如果为空，则不需要进行检查，直接存储

\* 链表不为空，则通过读取再比对，进行存储

\*

\*/

tmp\_node = node\_queue->sys\_list[CONNECT\_LIST]->next;

do

{

if(tmp\_node == NULL)

{

state++;

dmanage\_add\_connected\_info(info);

break;

}else{

pinfo = tmp\_node->data;

if(pinfo->client\_fd == type->fd)

{

state++;

printf("the client is exist\n");

free(info);

return ERROR;

}

}

tmp\_node = tmp\_node->next;

}while(tmp\_node != NULL);

if(state == 0)

{

dmanage\_add\_connected\_info(info);

}

/\*

\* 添加心跳链表设备

\*/

if(type->dev\_type != PC\_CTRL){

dmanage\_set\_communication\_heart(type);

}

/\*

\* 保存至会议信息链表

\* 如果上线单元机有id号则将上线单元机信息进行存储

\*

\* 会议中上线的单元机，主机会自动给其分配ID号，用于管理

\*/

if(type->dev\_type != PC\_CTRL)

{

confer\_info = (Pconference\_list)malloc(sizeof(conference\_list));

memset(confer\_info,0,sizeof(conference\_list));

confer\_info->fd = type->fd;

confer\_info->con\_data.id = type->con\_data.id = type->s\_id;

confer\_info->con\_data.seat = type->con\_data.seat;

//判断当前是否有会议主席，如果有则上线的单元设置为客席，会议链表中只能存在一个主席

if(conf\_status\_check\_chariman\_staus())

{

confer\_info->con\_data.seat = type->con\_data.seat = WIFI\_MEETING\_CON\_SE\_GUEST;

}

/\*

\* 1、判断源地址是否存在，存在则判断是否与现有设备冲突，冲突则设置为最大ID并保存到链表

\* 2、如果源地址为0.且在会议中加入，则需要自动对其进行编号和席别设置

\*/

if(type->s\_id > 0)

{

if(conf\_status\_compare\_id(type->s\_id))

{

type->s\_id = conf\_status\_find\_max\_id();

confer\_info->con\_data.id = type->con\_data.id = type->s\_id;

//将修改后的会议参数下发给单元机

conf\_info\_set\_conference\_params(type->fd,type->con\_data.id ,

type->con\_data.seat,NULL,NULL);

}

dmanage\_refresh\_conference\_list(confer\_info);

}else if(type->s\_id == 0)

{

if((conf\_status\_get\_conf\_staus() ==

WIFI\_MEETING\_EVENT\_CON\_MAG\_START))

{

//会议过程中上线的设备，需要对其进行编号

type->s\_id = conf\_status\_find\_max\_id();

confer\_info->con\_data.id = type->con\_data.id = type->s\_id;

//将修改后的会议参数下发给单元机

conf\_info\_set\_conference\_params(type->fd,type->con\_data.id,

type->con\_data.seat,NULL,NULL);

dmanage\_refresh\_conference\_list(confer\_info);

}

}

/\*

\* 发送至本地上报队列

\*/

tcp\_ctrl\_msg\_send\_to(type,msg,WIFI\_MEETING\_EVENT\_ONLINE\_REQ);

}

// }

return SUCCESS;

}

/\*

\* dmanage\_refresh\_connected\_list

\* 更新设备连接信息链表

\*

\*/

int dmanage\_refresh\_connected\_list(Pconference\_list data\_info)

{

pclient\_node tmp = NULL;

pclient\_node del = NULL;

Pclient\_info cinfo;

Pclient\_info tcinfo;

Pclient\_info newinfo;

int pos = 0;

int status = 0;

FILE\* cfile;

int ret = 0;

/\*

\* 首先在连接信息中搜寻socke\_fd

\* 判断是否有此设备连接，删除存在结点，重新插入结点

\* 更新

\*/

tmp = node\_queue->sys\_list[CONNECT\_LIST]->next;

while(tmp!=NULL)

{

cinfo = tmp->data;

if(cinfo->client\_fd == data\_info->fd)

{

status++;

break;

}

pos++;

tmp = tmp->next;

}

if(status > 0)

{

tcinfo = (Pclient\_info)malloc(sizeof(client\_info));

memset(tcinfo,0,sizeof(client\_info));

list\_delete(node\_queue->sys\_list[CONNECT\_LIST],pos,&del);

cinfo = del->data;

memcpy(tcinfo,cinfo,sizeof(client\_info));

free(cinfo);

free(del);

printf("%s-%s-%d,delete data in the connection list,then add it\n",

\_\_FILE\_\_,\_\_func\_\_,\_\_LINE\_\_);

tcinfo->id = data\_info->con\_data.id;

tcinfo->seat = data\_info->con\_data.seat;

list\_add(node\_queue->sys\_list[CONNECT\_LIST],tcinfo);

}else{

printf("%s-%s-%d,there is no client in the connection list\n",

\_\_FILE\_\_,\_\_func\_\_,\_\_LINE\_\_);

return ERROR;

}

if(status > 0)

{

cfile = fopen(CONNECT\_FILE,"w+");

/\*

\* 更新文本文件

\*/

tmp = node\_queue->sys\_list[CONNECT\_LIST]->next;

while(tmp != NULL)

{

newinfo = tmp->data;

if(newinfo->client\_fd > 0)

{

printf("%s-%s-%d,fd=%d,id=%d,seat=%d\n",\_\_FILE\_\_,\_\_func\_\_,

\_\_LINE\_\_,newinfo->client\_fd, newinfo->id,newinfo->seat);

ret = fwrite(newinfo,sizeof(client\_info),1,cfile);

perror("fwrite");

if(ret != 1)

return ERROR;

}

tmp = tmp->next;

usleep(100);

}

fclose(cfile);

}

//end

return SUCCESS;

}

/\*

\* dmanage\_refresh\_conference\_list

\* 更新会议信息链表

\*

\*/

int dmanage\_refresh\_conference\_list(Pconference\_list data\_info)

{

pclient\_node tmp = NULL;

pclient\_node del = NULL;

Pclient\_info cinfo;

Pconference\_list finfo;

int pos = 0;

int status = 0;

/\*

\* 设置会议参数，需要判断是否当前有设备在连接信息链表中

\*/

tmp = node\_queue->sys\_list[CONNECT\_LIST]->next;

while(tmp!=NULL)

{

cinfo = tmp->data;

if(cinfo->client\_fd == data\_info->fd)

{

status++;

break;

}

tmp = tmp->next;

}

if(status > 0)

{

status = 0;

}else{

printf("%s-%s-%d,there is no client in the connected list\n",

\_\_FILE\_\_,\_\_func\_\_,\_\_LINE\_\_);

return ERROR;

}

/\*

\* 判断会议信息中的设备参数socke\_fd

\* 判断是否有此设备连接，删除存在结点，重新插入结点

\*/

tmp = node\_queue->sys\_list[CONFERENCE\_LIST]->next;;

while(tmp != NULL)

{

finfo = tmp->data;

if(finfo->fd == data\_info->fd)

{

status++;

break;

}

pos++;

tmp = tmp->next;

}

if(status > 0)

{

list\_delete(node\_queue->sys\_list[CONFERENCE\_LIST],pos,&del);

finfo = del->data;

status = 0;

free(finfo);

free(del);

printf("%s-%s-%d,delete data in the conference list,then add it\n",

\_\_FILE\_\_,\_\_func\_\_,\_\_LINE\_\_);

}else{

printf("%s-%s-%d,there is no data in the conference list,add it\n",

\_\_FILE\_\_,\_\_func\_\_,\_\_LINE\_\_);

}

list\_add(node\_queue->sys\_list[CONFERENCE\_LIST],data\_info);

//end

return SUCCESS;

}

/\*

\* dmanage\_refresh\_info

\* 更新设备信息

\* 1、连接链表信息，文本文件

\* 2、会议链表信息

\*

\*/

int dmanage\_refresh\_info(const Pframe\_type data\_info)

{

Pconference\_list confer\_info = NULL;

int ret = 0;

/\*

\* 更新到连接信息链表中

\*/

ret = conf\_status\_refresh\_spk\_node(data\_info);

if(ret)

{

printf("%s-%s-%d,dmanage\_refresh\_connected\_list err\n",\_\_FILE\_\_,\_\_func\_\_,\_\_LINE\_\_);

free(confer\_info);

return ERROR;

}

/\*

\* 会议信息链表

\*/

confer\_info = (Pconference\_list)malloc(sizeof(conference\_list));

memset(confer\_info,0,sizeof(conference\_list));

confer\_info->fd = data\_info->fd;

confer\_info->con\_data.id = data\_info->con\_data.id;

confer\_info->con\_data.seat = data\_info->con\_data.seat;

memcpy(confer\_info->con\_data.name,data\_info->con\_data.name,strlen(data\_info->con\_data.name));

memcpy(confer\_info->con\_data.conf\_name,data\_info->con\_data.conf\_name,strlen(data\_info->con\_data.conf\_name));

/\*

\* 更新到连接信息链表中

\*/

ret = dmanage\_refresh\_connected\_list(confer\_info);

if(ret)

{

printf("%s-%s-%d,dmanage\_refresh\_connected\_list err\n",\_\_FILE\_\_,\_\_func\_\_,\_\_LINE\_\_);

free(confer\_info);

return ERROR;

}

/\*

\* 更新到会议信息链表中

\*/

ret = dmanage\_refresh\_conference\_list(confer\_info);

if(ret)

{

printf("%s-%s-%d,dmanage\_refresh\_conference\_list err\n",\_\_FILE\_\_,\_\_func\_\_,\_\_LINE\_\_);

free(confer\_info);

return ERROR;

}

return SUCCESS;

}