§. 带格式数据包的读取及注意事项

1. 数据格式的定义

★ 采用定长头部+可变数据部分组成

例：

▪1. 报道

- 功能：用户连接服务器后，用户向服务器进行报道，服务器给予允许/拒绝的应答
- 报道请求：用户=>服务器（共 32 个字节）

0	7 8	15 16	23 24	31
0x11	0x0	0x0020：代表报文的长度（32 字节）		
最长 28 字节的用户名，如果不满 28 字节，则用\0 填充剩余部分				

- 报道应答：服务器=>用户（共 4 个字节）

0	7 8	15 16	23 24	31
0x71	应答类型	0x0004：代表报文的长度（4 字节）		

【应答类型】 0x00：检查错误，拒绝报道 / 0x01：检查正确，接受报道

- 备注：
  - 这是 TCP 连接成功后发的第一个报文
  - 在 Client 端收到“报道应答”之前，不处理服务器来的报文，也不向服务器发任何报文
  - 如果“报道应答”为拒绝报道，则关闭 TCP 连接，延时后再次重连
  - 所有 2/4 字节数据均是网络序（高位在前，低位在后），下同

§. 带格式数据包的读取及注意事项

1. 数据格式的定义

★ 采用定长头部+可变数据部分组成

例：

▪2. 文本信息发送

- 功能：用户向服务器发送数据
- 发送请求：用户=>服务器（长度不定，但不超过 1024 字节）

0	7 8	15 16	23 24	31
0x12	0x0	0x****：代表报文的长度（**字节）		
最长 1020 字节的文本信息，要求为不含尾零的文本信息串，中英文均可 信息的最开始以@all / @单个用户名开始，跟冒号（全半角均可），冒号后面是具体的信息 例：@all:大家好 @张三：你现在在哪里 其它格式均认为是错误的				

- 发送应答：服务器=>用户（共 4 个字节）

0	7 8	15 16	23 24	31
0x72	应答类型	0x0004：代表报文的长度（4 字节）		

【应答类型】 0x00：检查正确，已转发

0xFF：发送的信息语法有误，拒绝转发

0xFE：如果是发给单个好友的信息，该好友的名称不存在

0xFD：如果是发给单个好友的信息，该好友未上线

§.带格式数据包的读取及注意事项

1.数据格式的定义

★ 数据格式尽量采用结构体方式

例：

```
/* 所有报文共同的头结构（4字节） */
typedef struct {
    u_char  type;
    u_char  ack;
    u_short len;
} IM_HEAD;

/* 报道 - 第一个报文 */
typedef struct {
    IM_HEAD pkg_head;
    u_char  username[28];
} IM_USER_REG;

/* 文本信息发送 - 第二个报文 */
typedef struct {
    IM_HEAD pkg_head;
    u_short value[1020];
} IM_TXTINFO;
```

§.带格式数据包的读取及注意事项

1.数据格式的定义

★ 包类型尽量采用宏定义

例：

```
#ifndef _IM_PACKET_TYPE_H_
#define _IM_PACKET_TYPE_H_

/* 定义服务器与设备之间的报文格式 */
#define IM_TYPE_REGISTER_REQ                0x11  //Client=>Server 报到
#define IM_TYPE_TXTINFO_REQ                 0x12  //                文本
...

#define IM_TYPE_REGISTER_ACK                0x71  //Server=>Client 报到应答
#define IM_TYPE_TXTINFO_ACK                 0x72  //                文本应答
...

#endif
```

§.带格式数据包的读取及注意事项

1.数据格式的定义

★ 定义结构体时注意严格四字节对齐，否则会有填充字节出现(也可以用紧凑定义，但不推荐)

```
#include <stdio.h>

struct a {
    int a;
    char b;
};

struct b {
    char b;
    int a;
};

int main()
{
    printf("%d %d\n",
           sizeof(struct a),
               sizeof(struct b));

    return 0;
}
//结果都是8
```

若定义时存在填充字节，建议直接写出来，方便后面理解

```
#include <stdio.h>

struct a {
    int a;
    char b;
    char pad[3];
};

struct b {
    char b;
    char pad[3];
    int a;
};
```

§.带格式数据包的读取及注意事项

2.数据的收发与解析（实例）

2.1 Socket结构体的定义

```
#ifndef _IM_SOCKET_H_
#define _IM_SOCKET_H_

#define IM_SOCKET_SENDBUF_LEN                   2048        //C向S发送的内容（值根据具体情况定义，可变）
#define IM_SOCKET_RECVBUF_LEN                   4096        //S向C发送的内容（值根据具体情况定义，可变）

/* 定义SOCKET结构，适用链表，对于非链表形式，不处理next即可 */
typedef struct _socket_ {
    struct _socket_ *next;

    /* 基本信息 */
    int     sock;
    u_int   ip;                                             //对端IP地址
    u_short port;                                           //对端端口号
    u_char  type;                                           //SOCKET的类型（设备、客户端或其他）
    u_char  pad;

    /* 收发缓冲区 */
    short   sendbuf_len;                                    //发送缓冲区中现有信息的长度
    short   recvbuf_len;                                    //接收缓冲区中现有信息的长度
    u_char  sendbuf[IM_SOCKET_SENDBUF_LEN];                 //发送缓冲区
    u_char  recvbuf[IM_SOCKET_RECVBUF_LEN];                 //接收缓冲区

    /* 管理信息（可以根据需要增减） */
    time_t  connect_time;                                   //socket连接上的时间
    int     userid;                                         //代表该连接的用户名id
    int     keepalive_count;                                //该连接的心跳记数
} SOCKET;

#endif
```

§.带格式数据包的读取及注意事项

2.数据的收发与解析（实例）

2.2 Socket结构体的初始化（假设为Server端accept）

```
void im_accept_new_socket(int listensock, SOCKET *head) //设listensock有连接时调用，head为SOCKET链表的头指针
{
    SOCKET   *new_tmp;
    int       new_sock;
    ...//其它需要定义的变量

    ...//new_sock为从listensock中accept的描述符，若accept返回-1则直接返回（限于篇幅，具体略）
    ...//将new_sock设为非阻塞方式（略）

    /* 申请新的SOCKET结点 */
    if ((new_tmp = (SOCKET *)malloc(sizeof(SOCKET)) == NULL) {
        ...打印错误信息（具体略）
        close(new_sock);
        return;
        }
    /* 填充SOCKET结构体 */
    new_tmp->next            = NULL;
    new_tmp->sock            = new_sock;
    new_tmp->sendbuf_len     = 0;
    new_tmp->recvbuf_len     = 0;
    memset(new_tmp->sendbuf, 0, IM_SOCKET_SENDBUF_LEN);      //其实是没有必要的，因为操作时判断长度而不是内容
    memset(new_tmp->recvbuf, 0, IM_SOCKET_RECVBUF_LEN);      //其实是没有必要的，因为操作时判断长度而不是内容

    //填充SOCKET结构体的其它需要信息（可多项，略）
    new_tmp->connect_time    = time(0);//连接时，该时间是连接上的时间

    ...//加入链表尾部（略）
    ...//给出连接成功的提示信息
    return;
}
```

§.带格式数据包的读取及注意事项

2.数据的收发与解析（实例）

2.3 将Socket结构体置入读/写集，判断select的返回值

```
int socket_add_to_rwset(SOCKET *head, fd_set *rfd, fd_set *wfd, int maxfd)
{
    SOCKET *node = head->next; //假设head为带头结点的单链表的头指针
    while(node) {
        /* 只有当接收缓冲区还有空的时候，才置读集 */
        if (node->recvbuf_len < IM_SOCKET_RECVBUF_LEN)
            FD_SET(node->sock, rfd);

        /* 只有当发送缓冲区有内容时，才置写集 */
        if (node->sendbuf_len > 0)
            FD_SET(node->sock, wfd);

        /* 扩大文件句柄集合 */
        if (node->sock > maxfd)
            maxfd  = node->sock;

        node = node->next;
        } //end of while

    return maxfd;
}
```

```
void select_loop(int listensock, SOCKET *head, 其它想加入的fd)
{
    fd_set rfd, wfd;
    int    maxfd;
    int    sel;

    for (;;) {
        FD_ZERO(&rfd);
        FD_ZERO(&wfd);
        maxfd = 加入listensock及其它需要的fd
        maxfd = socket_add_to_rwset(head, &rfd, &wfd, maxfd);

        sel = select(maxfd+1, &rfd, &wfd, NULL, NULL); //阻塞轮询
        if (sel>0) {
            //判断读写集，调用gn_accept_new_socket、
            //gn_dev_socket_read、gn_dev_socket_write等函数
            }
        //select <0 / ==0 等情况
        }//end of for(;;)
}
```

§.带格式数据包的读取及注意事项

2.数据的收发与解析（实例）

2.4 读Socket的内容，放入recvbuf

```
void im_socket_read(SOCKET *head, SOCKET *tsock) //链表中某结点的socket置读
{
    char *p = (char *)(tsock->recvbuf+tsock->recvbuf_len);
    int    len;
    ...//其它需要定义的变量

    //读的最大长度为缓冲区剩余大小
    len = recv(tsock->sock, p, IM_SOCKET_RECVBUF_LEN - tsock->recvbuf_len, 0);
    if (len>0) {
        ...//给出提示信息等其它处理工作

        tsock->recvbuf_len += len; //缓冲区长度增加

        // 进行数据包的解析处理
        im_process_socket_readbuf(tsock);
        }
    else {
        ...//处理recv返回0、-1的情况，给出提示信息，关闭socket，从链表中移除等
        }

    return;
}
```

§.带格式数据包的读取及注意事项

2.数据的收发与解析（实例）

2.5 处理recvbuf中收到的数据包（自定义的格式解析）

```
static int im_process_socket_readbuf(SOCKET *tsock) //处理socket读到的数据包
{
    u_char      *p     = tsock->recvbuf;
    IM_HEAD     *head;

    /* 缓冲区长度至少应该比数据包的头长要长才能继续循环 */
    while (tsock->recvbuf_len>=sizeof(IM_HEAD)) {
            head = (IM_HEAD *)p;  //强制类型转换，熟练使用会带来很大好处!!!

            ...// 检查收到的报文头的信息是否正确，出错则清空缓冲区并返回

        /* 若缓冲区中的数据<包长（不够一个包），移动到前面，等候下次再读 */
        if (tsock->recvbuf_len < ntohs(head->len))
            break;

        switch (head->type) {
            ...//根据数据包类型处理各数据包（此处可能很长）
            }

        /* 一个数据包处理完，指向socket中的下一个包 */
        tsock->recvbuf_len -= ntohs(head->len);
        p                  += ntohs(head->len);
        } /* end of while */

    /* 如果还有剩余数据（接收的数据不满一个包的要求长度），则移动到前面 */
    if (tsock->recvbuf_len>0)
        memmove(tsock->recvbuf, p, tsock->recvbuf_len);

    return 0;
}
```

§.带格式数据包的读取及注意事项

2.数据的收发与解析（实例）

2.6 向sendbuf中填写数据（自定义的格式，注：有报文后才会被置select写集并发送）

```
int im_send_packet(SOCKET *tsock, u_char type, 其它需要的信息)  //向发送缓冲区中填写要发送的报文
{
    u_char      *p     = tsock->sendbuf+tsock->sendbuf_len;
    GN_HEAD     *head  = (IM_HEAD *)p; //强制类型转换，熟练使用会带来很大好处!!!
    u_short      len;
    u_char       ack;

    /* 1、根据要发送报文的类型及各种情况，决定ack及len字段的值 */
    switch(type) {
        ack = ***;
        len = ***; //此处应该是正常长度（主机序）
        }

    /* 2、判断发送缓冲区的长度是否够容纳下新包 */
    if (tsock->sendbuf_len + len > IM_SOCKET_SENDBUF_LEN)
        return -1;

    /* 3、填写报文头 */
    head->type = type;
    head->ack  = ack;
    head->len  = ntohs(len);

    /* 4、根据需要附加内容 */
    switch(type) {
        ...//某些报文除头结构外，还需要附加信息
        }

    /* 5、发送缓冲区长度增加 */
    tsock->sendbuf_len += len;

    return 0;
}
```

```
/* 假设是第一个报文 ，则 */
IM_USER_REG *preg = p;  //和head重复，没关系

/* 填充除head外的其它字段 */
strcpy(preg->username, ***);
...

只要保证len = 32即正确
若需要复制整个报文，则memcpy(dst, preg, sizeof(IM_USER_REG));
```

```
/* 假设是第二个报文，则 */
IM_TXTINFO *ptxt = p;  //和head重复，没问题

/* 填充若干温度值（假设30个） */
memcpy(ptxt->value, 内容, 长度);

只要保证 len = head->len 即正确
```

§.带格式数据包的读取及注意事项

2.数据的收发与解析（实例）

2.7 将sendbuf的内容写入socket

```
void im_socket_write(SOCKET *head, SOCKET *tsock) //链表中某结点的socket置写
{
    int len;
     ...//其它需要定义的变量

    //写的最大长度为缓冲区大小（从[0]开始写，注意可能不是一个数据包的开头）
    len = send(tsock->sock, tsock->sendbuf, tsock->sendbuf_len, 0);
    if (len>0) {
        ...//给出提示信息等其它处理工作

        tsock->sendbuf_len -= len; //缓冲区长度减少

        /* 如果还有剩余数据，则剩余部分移动到前面 （注：用循环队列方式更好）
          （下层缓冲区满导致本次数据[可能是多个包]没写完，注意，最后一个数据包可能部分写入）*/
        if (tsock->sendbuf_len > 0)
            memmove(tsock->sendbuf, tsock->sendbuf+len, tsock->sendbuf_len);
        }
    else {
        ...//处理send返回0、-1的情况，给出提示信息，关闭socket，从链表中移除等
        }

    return;
}
```