**通讯协议：**

* tcp 和 串口 传输数据 类似的， 属于流式的，一方发送一个数据包，对方可能一次就刚好接收到一个完整的包，也可能分多次才接收完毕一个完整的数据包。当数据包尺寸比较小，通讯线路比较空闲的时候，可能对方一次性收到一个完整的包，当数据包比较大，或者线路比较忙的时候， 往往会出现一个包被分成了几个部分 ，所以接受方，需要组包，来还原数据包 ，所以组包是个关键算法。

UDP虽然是可以保证每次收发的都是一个完整的数据包，通过实测， 本转换网关，因为牵扯到串口通讯，所以，即使使用UDP的通讯方式，依然会出现分包的情形。

本方案我们采用tcp 模式， 不考虑udp。

* 上下位机之间的数据通讯采用的是二进制格式， 而不是ASCII码字符， 这个一定要注意，这个是因为我们传输的是二进制数据， 不是普通的聊天程序那样传输的文本信息 。
* 数据包格式

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **2 byte** | **2 byte** | **2 byte** | **2 byte** | **Data length byte** | **2 byte** |
| **0xAABB** | **command** | **Serial No** | **Data length** | **data** | **Check sum** |
| **unsigned short** | **unsigned short** | **unsigned short** | **unsigned short** | **char** | **unsigned short** |

* 校验码为数据包中除校验码之外的所有字段的累加和，对65535求余后的余数， 该字段为可选的，根据数据可靠性的要求，决定是否启用校验码，不启用校验码的，该字段可以填入任意数值， 对于下位机上报的普通的信息, 对于不需要对方确认的命令，都可以不做和校验 ，以便提高程序效率。

校验码是可选的，即使不不做校验， 那个校验码位置一定要填充2字节的数据。为了安全因素，上位机下发给下位机的 命令，建议要带校验码， 比如上位机发送的控制命令， 下发的更新系统的命令 等。

校验是需要花费一定的计算工作的，请权衡利弊。

* cmmd 定义 ：

#define CMMD\_NORMAL\_DATA 0 //下位机上传正常的数据

#define CMMD\_CONTROL\_UPGRADE 1 //上位机请求更新指定设备的系统，上位机给下位机下发文件

#define CMMD\_CONTROL\_ON 2 //上位机请求开启指定的设备

#define CMMD\_CONTROL\_OFF 3 //上位机请求关闭指定的设备

#define CMMD\_CONTROL\_SYNC\_TIME 4 //上位机请求同步下位机时间，用于矫正下位机时间

#define CMMD\_HEART\_BEAT 5 //上报设备状态， 设备心跳包，上报数据为上报时间（格式字符串 YYYY-MM-DD HH-MM-SS），zigebee网络的panid（十六进制数值） 和 zigbee网络本机地址（十六进制数值）； 定时上报，时间暂定为一分钟； 本命令不需要上位机回应； 上位机将心跳数据存放与数据库，上位机程序定时或者手动检查设备的状态

|  |  |  |  |
| --- | --- | --- | --- |
| 字段名称 | 数据类型 | 长度 byte | 字段说明 |
| **起始码** | unsigned short | 2 | 固定为2字节的 0xAABB |
| **command** | unsigned short | 2 | 0 --下位机 向上位机发送的普通数据（netId,netAddr,event,steps），这个指令不需要对方回应,  1--上位机请求下位机更新系统，下位机需要回应此命令， 该命令的data字段为配置数据，可以从本地电脑文件中读取  2--上位机请求控制下位机设备的请求，开机， 下位机需要回应命令  3--上位机请求控制下位机设备的请求，关机， 下位机需要回应命令  4--... |
| **Serial No** | unsigned short | 2 | 上下位机需要对命令进行回应确认的，需要设置序列号， 其他的情形，该字段忽略，可以任意设置没有实际意义  请求包和回应包，使用同一个序列号 |
| **Data length** | unsigned short | 2 | 数据长度 ，即data字段的长度 |
| **data** | char | data length | 数据 |
| **Check sum** | unsigned short | 2 | 起始码**0xAABB,** 不参与校验 !  校验码， 从起始码之后command字段，到data的 所有字段的 校验和，算法为，单字节求和再除以65535的余数，，该项为选项，建议上位机下发到下位机的命令使用校验，不使用校验的场合该字段可以填充该字段范围内的任意值 |

**下位机上报的普通数据**

|  |  |  |  |
| --- | --- | --- | --- |
| 字段名称 | 数据类型 | 长度  byte | 说明 |
| **netId** | unsigned short | 2 | zigbee网络号 |
| **netAddr** | unsigned short | 2 | zigbee网址 |
| **event** | unsigned short | 2 | 事件类型  1 开机，  2 关机，  3剪线，  4 倒缝，  5 断线 |
| **time** | Char[19] | 19 | 时间字符串：2016-11-29 10:50:20 |
| **steps** | unsigned short | 2 | 针数 |
| **EmployeeNo.** | Char[10] | 10 | 员工工号：0100101001 |
| **ProcessNo.** | Char[24] | 24 | 工序号：301302303304305306307308 |
| **ModelNo.** | Char[8] | 8 | 款号：3902.601 |

示例数据 ：

unsigned short start = 0xaabb;

unsigned short cmmd = CMMD\_NORMAL\_DATA; // 0x0000;

unsigned short serialNo = 1;

unsigned short dataLen = 69;

cd.netId = 0x1001;

cd.netAddr = 0x2002;

cd.event = 0x0000;

memcpy(cd.time, "2016-11-30 11:06:18", 19);

cd.steps = 0x0100;

memcpy(cd.employeeNo, "0100101001" , 10); //[10]; //员工编号

memcpy(cd.processNo, "301302303304305306307308", 24); //[24]; //工序号

memcpy(cd.modelNo, "3902.601" ,8); //[8]; 型号

checksum = 0000;

aabb000000010045100120020000323031362d31312d33302031313a30363a3138000130313030313031303031333031333032333033333034333035333036333037333038333930322e3630310000

* **上位机对上传普通报文的回应包的data部分的格式**

|  |  |  |  |
| --- | --- | --- | --- |
| 字段名称 | 数据类型 | 长度  byte | 说明 |
| **netId** | unsigned short | 2 | zigbee网络号 |
| **netAddr** | unsigned short | 2 | zigbee网址 |

* 更新系统文件指令的数据结构

struct TagUpgradeDataStruct

{

unsigned short netId;

unsigned short netAddr;

//////////////////////////////

unsigned short totalPack; //总的pack数量

unsigned short packNo; //pack编号

unsigned short packLen; // current pack length , 此处为当前pack的长度，标准pack的长度为MAX\_FILE\_PACK\_LEN 1024byte，

char fileName[30]; // 不足30的 空余地方填充0

// above sum up to 36 bytes

char \* packData; //packData[ current pack Len];

};

struct TagUpgradeResDataStruct //下位机到上位机

{

unsigned short netId;

unsigned short netAddr;

char fileName[30]; // 不足30的 空余地方填充0

};

* **更新系统文件指令的数据格式（上位机到下位机）**

|  |  |  |  |
| --- | --- | --- | --- |
| 字段名称 | 数据类型 | 长度byte | 说明 |
| **netId** | unsigned short | 2 | zigbee网络号 |
| **netAddr** | unsigned short | 2 | zigbee网址 |
| totalPack | unsigned short | 2 | pack总的数量 |
| packNo | unsigned short | 2 | pack编号 |
| packLen | unsigned short | 2 | 此处为当前pack的长度，标准pack的长度为MAX\_FILE\_PACK\_LEN 1024byte， |
| fileName | char | 30 | 不足30的 空余地方填充0 |
| packData | char | packLen | pack数据 |

* **更新系统文件响应指令的数据格式（下位机到上位机）**

|  |  |  |  |
| --- | --- | --- | --- |
| 字段名称 | 数据类型 | 长度byte | 说明 |
| **netId** | unsigned short | 2 | zigbee网络号 |
| **netAddr** | unsigned short | 2 | zigbee网址 |
| fileName | char | 30 | 不足30的 空余地方填充0 |

/\* test data : 回应数据示例

sample:

起始两个字节 0xAA 0xBB

cmmd = 0x0001

serialNo = 0x0006

dataLength =0x0022 //34

data指向一个结构体 TagUpgradeResDataStruct， 赋值如下 ：

netId=0x1001

netAddr=0x2002

fileName[30]="system" //0x73 0x79 0x73 0x74 0x65 0x6d 0x0000.......

checkSum = 0x0000 //这里可以任意写一个2字节的校验码

该完整的数据包是

0xaa 0xbb 0x0001 0x0006 0x0022 0x1001 0x2002 0x73 0x79 0x73 0x74 0x65 0x6d ....填充24个 0x0000..... 校验码 0x0000

去掉前缀0x 后 拷贝到串口调试助手，按十六进制 发送 即可

aa bb 0001 0006 0022 1001 2002 73 79 73 74 65 6d 0000.... 0000

aabb0001000600221001200273797374656d0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000

\*/

* **控制开关机的指令数据部分格式**

|  |  |  |  |
| --- | --- | --- | --- |
| 字段名称 | 数据类型 | 长度byte | 说明 |
| **netId** | unsigned short | 2 | zigbee网络号 |
| **netAddr** | unsigned short | 2 | zigbee网址 |

同步下位机时间的指令 ， 数据是上位机当前本机的时间

|  |  |  |  |
| --- | --- | --- | --- |
| 字段名称 | 数据类型 | 长度byte | 说明 |
| **netId** | unsigned short | 2 | zigbee网络号 |
| **netAddr** | unsigned short | 2 | zigbee网址 |
| Time | char[19] | 19 | 同步时间 |

示例 数据

AA BB 00 04 00 01 00 17 00 00 00 00 32 30 31 36 2D 31 32 2D 30 31 20 31 31 3A 33 36 3A 33 39 03 CE

* 校验和算法示例， 根据具体情况选用自己的算法：

注意网络字节顺序和本机字节顺序， 高低位字节顺序是相反的

unsigned short CProtocol::CheckSum(char \* buffer, int len)

{

unsigned short sum = 0;

for (int i = 0; i < len; i++)

{

sum = (sum+buffer[i]) % 65535;

}

return sum;

}

// 注意, 起始位不参与校验

unsigned short CProtocol::CheckSum( unsigned short cmmd, unsigned short serial,unsigned short length, char\* data)

{

char head[6];

char \* pHead = head;

unsigned short sum = 0;

\*((int\*)pHead) = cmmd;

pHead += sizeof(unsigned short);

\*((int\*)pHead) = serial;

pHead += sizeof(unsigned short);

\*((int\*)pHead) = length;

for(int i=0; i<sizeof(head); i++)

sum = (sum+ head[i] ) % 65535;;

for(int i = 0; i<length; i++)

sum = (sum+data[i]) % 65535;

return sum;

}

unsigned short CProtocol::CheckSum()

{

unsigned short sum = 0;

for (int i = 0; i<sizeof(m\_packHead); i++)

sum =(sum+ m\_packHead[i]) % 65535;

for (int i = 0; i<m\_PacketStruct.dataLength; i++)

sum = (sum+ m\_PacketStruct.data[i]) % 65535;

return sum;

}

* 示例

start= 0xAABB

cmmd = 0x0000

serialNo = 0x0006

dataLength =0x0008

data指向一个结构体 TagClientDataStruct， 赋值如下 ：

netId=0x1001

netAddr=0x2002

event =0x0000 ;

steps =0x0100

该完整的数据包是

0xaa 0xbb 0x0000 0x0006 0x0008 0 x1001 0x2002 0x0000 0x0100

去掉前缀0x 后 拷贝到串口调试助手，按十六进制 发送 即可

aa bb 0000 0006 0008 1001 2002 0000 0100

================checksum 计算方法=================================

TagClientDataStruct dataSt ;

dataSt .netId=0x1001

dataSt .netAddr=0x2002

dataSt .event =0x0000 ;

dataSt .steps =0x0100

data = (char\*) & dataSt;

checksum= CheckSum( cmmd, serial, length, data) = 0x0042

* 生成序列号算法

static unsigned short GetSerialNo()

{

if (false)

{

static unsigned short num = 0;

num = (num + 1) % 65535;

return num;

}

else

return 1; //for test purpose，， 测试初期暂时先固定为1 ， 以方便调试报文,

}

* **上位机生成控制命令缓冲区buffer的示例算法代码：**

WSABUF wbuf;

//OVERLAPPED ol;

//ZeroMemory(&ol, sizeof(OVERLAPPED));

WSABUF \*p\_wbuf = &wbuf;

//OVERLAPPED \*p\_ol = &ol;

DWORD dwBytes = 0, dwFlags = 0;

char buffer[512];

char \* pbuf = p\_wbuf->buf = buffer;

p\_wbuf->len = 0;

\*(unsigned short\*)pbuf = htons(0xAABB);

p\_wbuf->len += 2;

pbuf += 2;

controlCode = pcs->controlCode == 1 ? CMMD\_CONTROL\_ON : CMMD\_CONTROL\_OFF;

\*(unsigned short\*)pbuf = htons(controlCode); //cmmd

p\_wbuf->len += 2;

pbuf += 2;

\*(unsigned short\*)pbuf = htons(CProtocol::GetSerialNo());

// GetSerialNo初始测试期间，暂定为1 ，

p\_wbuf->len += 2; pbuf += 2;

\*(unsigned short\*)pbuf = htons(4); // data length

p\_wbuf->len += 2; pbuf += 2;

\*(unsigned short\*)pbuf = htons(pcs->netId);//0x2001

p\_wbuf->len += 2; pbuf += 2;

\*(unsigned short\*)pbuf = htons(pcs->netAddr); //0x2002

p\_wbuf->len += 2; pbuf += 2;

//unsigned int check = htons(pSocketOlp->m\_cp.CheckSum(pfs.cmmd, pfs.serialNo, pfs.dataLength, pfs.data));

unsigned int check = htons(pSocketOlp->m\_cp.CheckSum(p\_wbuf->buf+2, p\_wbuf->len-2));

\*(unsigned short\*)pbuf = check;

p\_wbuf->len += 2; //pbuf += 2;

// on---- AA BB 00 02 00 01 00 04 20 01 20 02 00 4A

//off--- AA BB 00 03 00 01 00 04 20 01 20 02 00 4B

//int nBytesSent = Send(pSocketOlp->m\_socket, p\_wbuf, 1, &dwBytes, dwFlags, p\_ol, NULL);

**下位机上报的心跳数据包**

|  |  |  |  |
| --- | --- | --- | --- |
| 字段名称 | 数据类型 | 长度  byte | 说明 |
| **netId** | unsigned short | 2 | zigbee网络号 |
| **netAddr** | unsigned short | 2 | zigbee网址 |
| **time** | Char[19] | 19 | 时间字符串：2016-11-29 10:50:20 |