**无线测距模块用户手册**

型号：WD-M300

2022-03-20

目录

[1. 前言 3](#_Toc104495146)

[2. 产品图片 3](#_Toc104495147)

[3. 模块接口 3](#_Toc104495148)

[4. 模块尺寸 4](#_Toc104495149)

[5. 模块参数 5](#_Toc104495150)

[6. 数据输出格式 5](#_Toc104495151)

[私有协议 5](#_Toc104495152)

[数据段 5](#_Toc104495153)

[格式说明 5](#_Toc104495154)

[心跳包数据段 6](#_Toc104495155)

[心跳包格式说明 6](#_Toc104495156)

[校验算法 6](#_Toc104495157)

[数据解析示例 7](#_Toc104495158)

[参数读写 8](#_Toc104495159)

[格式说明 8](#_Toc104495160)

[读写地址表 9](#_Toc104495161)

[MODBUS RTU 协议说明 10](#_Toc104495162)

[7. 示例代码 11](#_Toc104495163)

[QT 示列解析数据代码 11](#_Toc104495164)

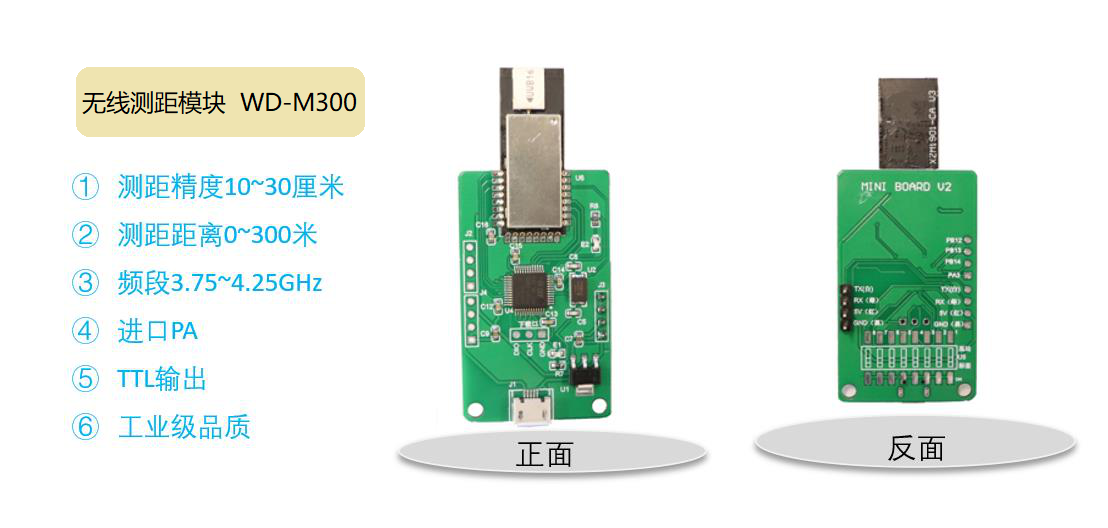
[C语言解析示列代码 12](#_Toc104495165)

[JAVA数据解析源代码示列 14](#_Toc104495166)

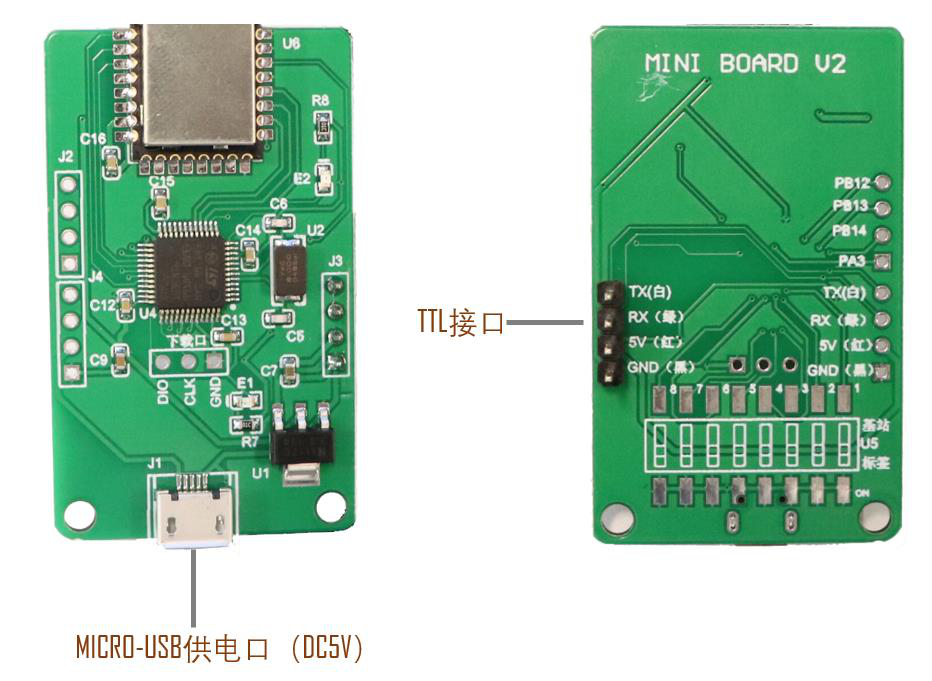
## 1. 前言

无线测距模块WD-M300是我公司自主设计开发的一款小体积便携式的高精度测距模块，该模块测距精度最高可到 10cm，可以实现点对点 、1对1、或者1对多或者多对多的测距模式，在装配定向天线且可视条件下测距距离大于300m，该测距模块核心元单元都采用进口工业级元器件，可满足365天\*24h 连续稳定工作。

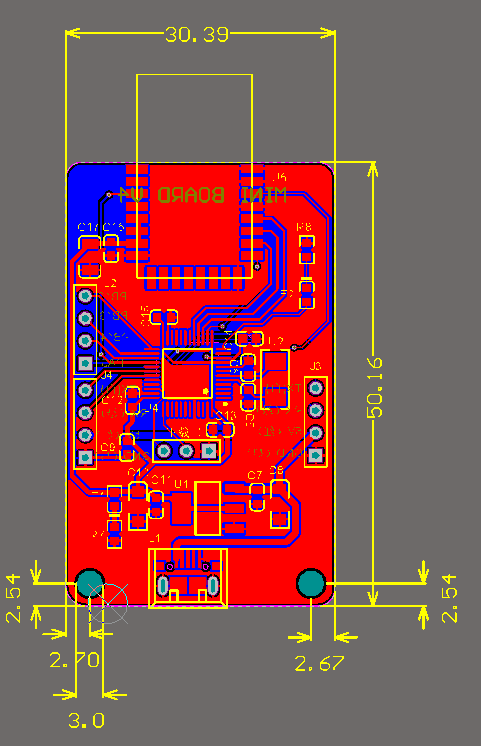
## 2. 产品图片



## 3. 模块接口



## 4. 模块尺寸



## 5. 模块参数

|  |  |
| --- | --- |
| 产品型号 | WD-M300 |
| 输入电压 | DC 5V |
| 接口 | TTL 3.3V |
| 尺寸 | 50.16 x 30.39mm |
| 重量 | 8.2g |
| 工作频率 | 3.75~4.25GHz |
| 功耗 | 1W |
| 测距精度 | 10~30cm |
| 波特率 | 115200 |
| 测距频率 | 1~100Hz（可调） |
| 测距范围 | 0~300m (如需达到最远测距距离，需安装定向天线) |
| 数据输出 | Modbus RTU |

## 6. 数据输出格式

### 私有协议

测距模块支持标准 Modbus\_RTU 协议，也支持私有协议数据输出，我司私有协议数据格式如下所示。

测距基站默认波特率为：115200

数据输出：485

每包数据如下表所示，分为帧头、数据段、校验码 3 个部分。

|  |  |  |  |
| --- | --- | --- | --- |
| 帧头 | 数据段 | Checksum 校验码 | |
| 0X55 0XAA | N | 低 8 位 | 高 8 位 |

### 数据段

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 指令类型 | 数据段长度 | 序号 | 标签ID | 基站ID | 距离 | 电量 | 按键 | 保留 |
| 0x01  （1Byte） | 17  （1Byte） | (1Byte) | (2Byte) | (2Byte) | （4Byte） | (1Byte) | (1Byte) | (4Byte) |

### 格式说明

|  |  |
| --- | --- |
| 名称 | 功能 |
| 指令类型（1Byte） | 0x01: 正常模式  0x02: 心跳包 |
| 数据段长度（1Byte） | 17（0x11） |
| 序号（1Byte） | 测距序号 |
| 标签 ID(2Byte) | 标签 ID |
| 基站 ID(2Byte) | 基站 ID |
| 距离(4Byte) | Uint32 距离信息，单位 cm |
| 电量 (1Byte) | 单位 1% |
| 按键 (1Byte) | 按键信息 |
| 保留(4Byte) |  |

### 心跳包数据段

|  |  |  |  |
| --- | --- | --- | --- |
| 指令类型 | 数据段长度 | 基站 ID | 保留 |
| 0x02（1Byte） | 12（1Byte） | (2Byte) | (8Byte) |

### 心跳包格式说明

|  |  |
| --- | --- |
| 名称 | 功能 |
| 指令类型（1Byte） | 0x01: 正常模式  0x02: 心跳包 |
| 数据段长度（1Byte） | 12（0x0C） |
| 基站 ID(2Byte) | 基站 ID |
| 保留(8Byte) | 保留 |

### 校验算法

uint16\_t Checksum\_u16(uint8\_t\* pdata, uint32\_t len)

{

uint16\_t sum = 0;

uint32\_t i;

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

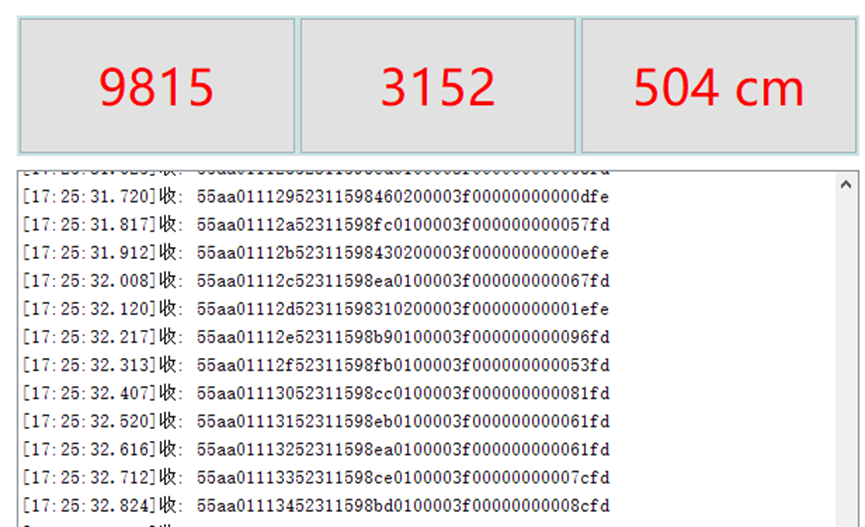
sum += pdata[i];

sum = ~sum;

return sum;

}

### 数据解析示例



[21:43:20.764]收 55 AA 01 11 AC 01 00 05 00 11 02 00 00 64 00 00 00 00 00 C4 FE

[21:43:21.265]收 55 AA 01 11 AD 01 00 05 00 11 02 00 00 64 00 00 00 00 00 C3 FE

[21:43:21.774]收 55 AA 01 11 AE 01 00 05 00 10 02 00 00 64 00 00 00 00 00 C3 FE

[21:43:22.275]收 55 AA 01 11 AF 01 00 05 00 10 02 00 00 64 00 00 00 00 00 C2 FE

示例：55 AA 01 11 AF 01 00 05 00 10 02 00 00 64 00 00 00 00 00 C2 FE

55AA 包头 2 字节，每包数据默认为 55AA 开头。

01 指令类 1 个字节，01 代表正常模式。

11 数据长度 1 个字节；转为 10 进制为 17 代表数据除去包头包尾有 17 个字节就是 34 个字符，加上包头包尾字符串长度就为 42 个字符。

AF 包序 1 字节；0-255 的循环，转为 10 进制为 175，带表第 175 包数据。

0100 标签 ID2 字节 ；低位在前高位在后就是 0001 转为 10 进制标签 ID 为0001。

0500 基站 ID2 字节；低位在前高位在后就是 0005 转为 10 进制标签 ID 为0005。

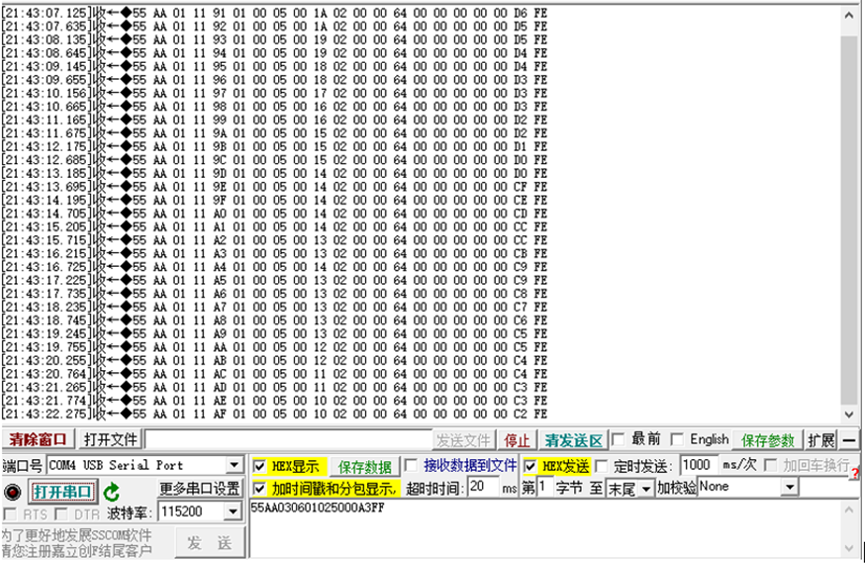
1002 0000 测距距离 4 字节单位厘米，有符号整数，低位在前高位在后为00001002，翻译为 10 进制,距离为 528 cm。

64 电量信息 1 字节，转为 10 进制为 100。

00 按键 1 字节，01 代表按下，00 代表没按下。

00 00 00 00 保留位 4 字节。

C2 FE 为检验码 2 字节。



### 参数读写

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 指令类型 （1BYTE） | 数据段长度 （1BYTE） | 读写类型 （1BYTE） | 读写地址 （1BYTE） | 读写长度 （1BYTE） | 读写数值 |
| 0X 03 | N |  |  |  | （N-5BYTE） |

### 格式说明

|  |  |
| --- | --- |
| 名称 | 功能 |
| 指令类型 | 0x03:参数读写模式 |
| 数据段长度 | 读写长度不同数据段长度不同 |
| 读写类型 | 0x01:读模式  0x02:写模式  0x03:应答模式 |
| 读写地址 | 建读写地址表 |
| 读写长度 | 读写参数的长度 |
| 读写数值 | 写模式下为参数的具体数值，读模式下为 0X00 |

### 读写地址表

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 地址 | 名称 | 长度 | 读/写 | 说明 |
| 0x2 | 版本号（2Byte） |  | 只读 | Xx.xx |
| 0x4 | 模块 ID（2Byte） |  | 读/写 |  |
| 0x6 | 标签通讯间隔（2Byte） |  | 读/写 | 单位毫秒 |
| 0x8 | 单次通讯基站数量上限（2Byte） |  | 读/写 | 一次最多可测距基站 数量 |
| 0xA | 通讯组 ID（2Byte） |  | 读/写 | 默认 0 |
| 0xC | 距离校准值（2Byte） |  | 读/写 | 写模式下为参数具体值  读模式下为 0x00 |
| 0xE | 模块类型（2Byte） |  | 读/写 | 基站 0/标签 1 |
| 0x10 | 基站主动测距（2Byte） |  | 读/写 |  |
| 0x12 | 报警设备（2Byte） |  | 读/写 |  |
| 0x14 | 报警距离 1（2Byte） |  | 读/写 |  |
| 0x16 | 报警距离 2（2Byte） |  | 读/写 |  |
| 0x18 | 报警距离 3（2Byte） |  | 读/写 |  |
| 0x1A | 配对 ID 无作用（2Byte） |  | 读/写 |  |
| 0x1C | 心跳包（2Byte） |  | 读/写 |  |
| 0x1E | Modbus 模式（2Byte） |  | 读/写 |  |
| 0x20 | （2Byte）临近基站数量 |  | 读/写 |  |
| 0x22 | （2Byte）临近基站 1 ID |  | 读/写 |  |
| 0x24 | （2Byte）临近基站 2 ID |  | 读/写 |  |
| 0x26 | （2Byte）临近基站 3 ID |  | 读/写 |  |
| 0x28 | （2Byte）临近基站 4 ID |  | 读/写 |  |
| 0x2a | （2Byte）临近基站 5 ID |  | 读/写 |  |
| 0x2C | （2Byte）临近基站 6 ID |  | 读/写 |  |
| 0x2E | （2Byte）临近基站 7 ID |  | 读/写 |  |
| 0x30 | （2Byte）临近基站 8 ID |  | 读/写 |  |
| 0x32 | （2Byte）临近基站 9 ID |  | 读/写 |  |
| 0x34 | （2Byte）临近基站 10 ID |  | 读/写 |  |
| 0x36 | （2Byte）发射功率 |  | 读/写 |  |
| 0x38 | （2Byte）加速度技灵敏度 |  | 读/写 |  |
| 0x3A | （2Byte）无运动休眠时间 |  | 读/写 |  |
| 0x3C | （2Byte）振动使能 |  | 读/写 |  |
| 0x3E | （2Byte）加速计使能 |  | 读/写 |  |
| 0x40 | （2Byte）滤波参数 |  | 读/写 |  |
| 0x48 | 速度滤波限值 |  | 读/写 |  |
| 0x60 | 模块重启（2Byte） |  | 写 | 写入 1 执行 |
| 0x62 | 恢复出厂设置（2Byte） |  | 写 | 写入 1 执行 |
| 0x64 | （开始升级 2Byte） |  | 写 | 写入 1 执行 |

### MODBUS RTU 协议说明

1. 默认波特率 115200，1 停止位，无校验位;

2. MODBUS 地址为基站模块的 ID;

3. 数据在 03 保持寄存器中，起始地址 40001;

4. 每个距离数据字节为 4 位，占用 2 个寄存器地址;

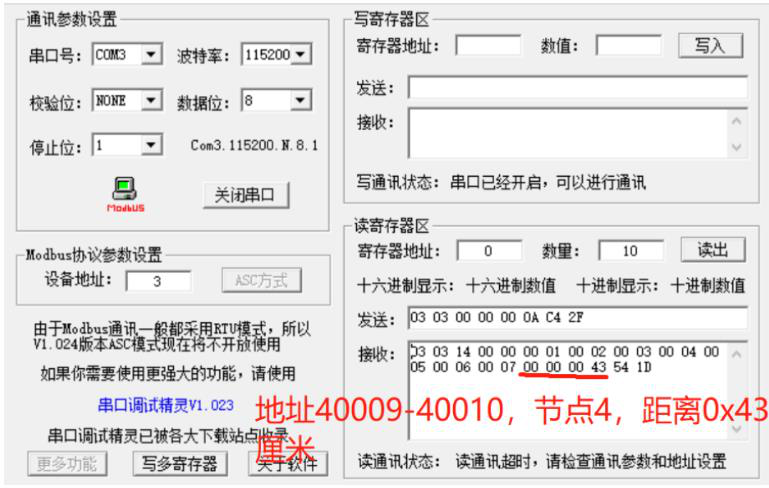
5. 数据高位在前，低位在后，单位 cm。

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| MODBUS 地址 | 4001-4002 | 4003-4004 | 4005-006 | 4007-008 | … |
| 数据 | 节点 0 距离 | 节点 1 距离 | 节点 2 距离 | 节点 3 距离 | … |

节点 N 的地址=40001+N\*2 至 40002+N\*2

示例：

读取基站 3 的 40001-40010 的数据



## 7. 示例代码

### QT 示列解析数据代码

**void** MainWindow::data\_process1(QByteArray &data) //传进来接收到的数据

{

**static** uint8\_t *usart\_state* =0, *data\_pos* =0, *pack\_len* =0, *pack\_cmd* =0;

}

uint16 checksum;

uint8 tmp;

**for**(uint16 i=0; i<data.length(); i++){ //循环处理完所有数据

tmp = (uint8\_t)data[i]; //数据赋值，处理1个字节数据

**switch**(usart\_state){ //串口状态

**case** 0:

**if**(tmp == 0x55) //状态0情况下，等待接收到0x55包头1，然后变成状态1

usart\_state =1;

**break**;

**case** 1:

**if**(tmp == 0xaa){ //状态1情况下，等待接收到0xAA包头2，然后变成状态2

usart\_state =2;

data\_pos =0;

}

**break**;

}

**case** 2:

pack\_cmd = tmp; //收完包头2，接收CMD位，然后变成状态3

frame[data\_pos++] =tmp;

usart\_state =3;

**break**;

**case** 3:

//状态3接收包长度，变状态4

pack\_len = tmp;

frame[data\_pos++] =tmp;

usart\_state =4;

**break**;

**case** 4:

//根据包长度接收足够长的数据

frame[data\_pos++] =tmp;

**if**(data\_pos >= pack\_len+2){

usart\_state = 0;

checksum = Checksum\_u16(frame,pack\_len); //计算校验值

qDebug() << checksum;

qDebug() << (uint16)(frame[pack\_len]|frame[pack\_len+1]<<8);

//对比校验值是否正确，如果正确进入显示模式

**if**(checksum ==(frame[pack\_len]|frame[pack\_len+1]<<8)){

**switch**(pack\_cmd){

**case** 1:

package\_show(1, frame); //测距信息

**break**;

**case** 2:

}

package\_show(2, frame); //心跳包

**break**;

**case** 3:

ui->textEdit->append(QStringLiteral("读取参数成功")); //修改参数

package\_show(3, frame);

**break**;

}

}

}

**break**;

**default**:

qDebug() << "erro";

**break**;

}

### C语言解析示列代码

#define frame\_length 200

uint8\_t frame[frame\_length];

uint8\_t usart\_state =0, data\_pos =0, pack\_len =0, pack\_cmd =0;

uint16\_t packge\_type, heart\_id, tag\_id, anchor\_id, seq ;

int32\_t distance;

void DataProcess(uint8\_t tmp) //将收到的数据一个个字节传入函数中

{ uint16\_t checksum;

switch(usart\_state){

case 0: //解析初始状态，等待接收包头55

if(tmp == 0x55)

usart\_state = 1;

break;

case 1:

if(tmp == 0xaa){ //接收包头AA

usart\_state = 2;

data\_pos = 0;

}else{

usart\_state = 0;

}

break;

case 2: //接收指令类型

pack\_cmd = tmp;

frame[data\_pos++] = tmp;

usart\_state =3;

break;

case 3: //接收包长度

pack\_len = tmp;

frame[data\_pos++] = tmp;

usart\_state =4;

break;

case 4: //接收数据

frame[data\_pos++] =tmp;

if(data\_pos >= pack\_len+2){

usart\_state = 0;

checksum = Checksum\_u16(frame,pack\_len); //计算校验值

if(checksum ==(frame[pack\_len]|frame[pack\_len+1]<<8)){ //比较校验值

switch(pack\_cmd){

case 1: //收到测距信息

memcpy(&distance,&frame[7],4); //距离值

seq = frame[2]; //序号

tag\_id = ( frame[3]|frame[4]<<8);//标签ID

anchor\_id = (frame[5]|frame[6]<<8);//基站ID

packge\_type = 1;

break;

case 2: //收到心跳包

heart\_id = frame[2]|frame[3]<<8;

packge\_type = 2;

break;

}

}

}

break;

default:

break;

}

}

### JAVA数据解析源代码示列

package Model;

import java.util.Vector;

import PublicMethod.DellMessage;

import PublicMethod.Jiaoyan;

public class DellHex implements Runnable {

//收到的原始数据集合，将收到的数据包拆分为2个字符一组放入该集合

static Vector<String> hexvc=new Vector<>();

//处理数据到什么状态了

int usart\_state=0;

//数据位的长度

int pack\_len=0;

//收到的数据类型

String type="";//数据类型

//线程休眠的时间

int sleeptime=0;

//完整的数据包

static StringBuffer pack\_cmd=new StringBuffer();

/\*\*向集合插入一条数据\*/

public static void insert\_adata(String hex) {

hexvc.add(hex);

}

/\*\*用于处理接收到的报文数据\*/

public void dellhex() {

//原始数据集合的长度

int size=hexvc.size();

//如果原始数据集合长度等于0代表没有数据，让线程休眠100毫秒等待集合有数据

if(size==0) {

sleeptime=100;

}else {

sleeptime=0;

String hex=hexvc.get(0).toUpperCase();

//将原始数据集合中的第一条数据取出放入swich循环

switch(usart\_state) {

case 0:

//如果数据是55则执行

if(hex.equals("55")) {

usart\_state=1;

pack\_cmd.append(hex);

}

break;

case 1:

//如果数据是AA则执行

if(hex.equals("AA")) {//状态1情况下，等待接收到AA包头2，然后变成状态2

usart\_state=2;

pack\_cmd.append(hex);

}

break;

case 2:

//数据类型给TYPE

type=hex;

pack\_cmd.append(hex);

usart\_state=3;

break;

case 3:

//报文数据段数据长度

pack\_len=DellMessage.decodeHEX(hex);

pack\_cmd.append(hex);

usart\_state=4;

break;

case 4:

pack\_cmd.append(hex);

if(pack\_cmd.length()==(pack\_len\*2+8)) {

usart\_state=0;

int len=pack\_cmd.length();

//校验

String check=Jiaoyan.check2(pack\_cmd.toString());

String panck\_cmd=pack\_cmd.toString().substring(len-2,len)+pack\_cmd.toString().substring(len-4, len-2);

//如果校验成功

if(check.equals(panck\_cmd)) {

//如果是测距数据

if(type.equals("01")) {

//给到处理测距数据的方法

DellMessage.dell\_info(pack\_cmd.toString());

//0A注册包

}else if(type.equals("0A")) {

//02心跳包

}else if(type.equals("02")) {

//07配置成功返回

}else if(type.equals("07")) {

//返回的配置信息

}else if(type.equals("03")) {

DellMessage.dell\_peizhi\_messge(pack\_cmd.toString());

}

}

pack\_cmd=new StringBuffer("");

}

break;

}

hexvc.removeElement(hexvc.get(0));

}

}

/\*\*启动线程的方法\*/

public void startThread() {

Thread t=new Thread(this);

t.start();

}

public void run() {

while(true) {

try {

dellhex();

Thread.sleep(sleeptime);//休眠时间

} catch (InterruptedException e) {

e.printStackTrace();

}

}

}

}