



## 成都远向电子科技有限公司产品说明书

产品名称：TOF-LED 激光测距显示传感器

全部资料下载地址：<http://ask.zstel.com:8090>

技术支持服务电话：[028-85583895](tel:028-85583895)

技术支持服务微信：[zskj0009](https://www.zstel.com/)

官网网站：<https://www.zstel.com/>

硬件/软件技术定制热线：[19150158475](tel:19150158475)

## 目录

一、 产品概述 .....	3
1.1 概述 .....	3
1.2 性能特点 .....	3
1.3 技术参数 .....	3
二、 外观尺寸 .....	4
2.1 产品外观 .....	4
2.2 产品尺寸图 .....	4
三、 产品接线图、跳线、指示灯说明 .....	5
3.1 设备接口 .....	5
3.2 接线图 .....	5
3.3 使用测试说明 .....	6
3.4 传感器使用注意事项 .....	7
四、 ModbusRTU 通讯协议地址以及案例说明 .....	9
4.1 通讯协议 .....	9
4.2 寄存器地址 .....	9
4.3 Modbus RTU 功能码 .....	9
4.4 Modbus 通讯实例 .....	9
五、 软件操作 .....	10
5.1 配置软件 .....	10
5.2 配置基本参数 .....	11
5.3 距离数据可视化测试 .....	11
六、 协议详解 .....	11
6.1 功能码描述 .....	12
6.2 错误码描述 .....	18
6.3 CRC 校验算法 .....	18

## 一、产品概述

### 1.1 概述

TOF 传感器是我司推出的一款高性能激光测距产品，其测量距离可达 4 米。它能够通过 485 串口直接输出测量到的距离数据，使用非常方便。广泛应用于工业生产、机器人避障、智能设备/穿戴等场合。

### 1.2 性能特点

- 9~36V 带防反接保护电源
- 支持串口数据主动输出
- 自带法兰盘方便快速安装
- 测量距离可达 4M
- 支持标准 Modbus 通信协议
- 支持 RS485 串口通信
- 高分辨率 1mm

### 1.3 技术参数

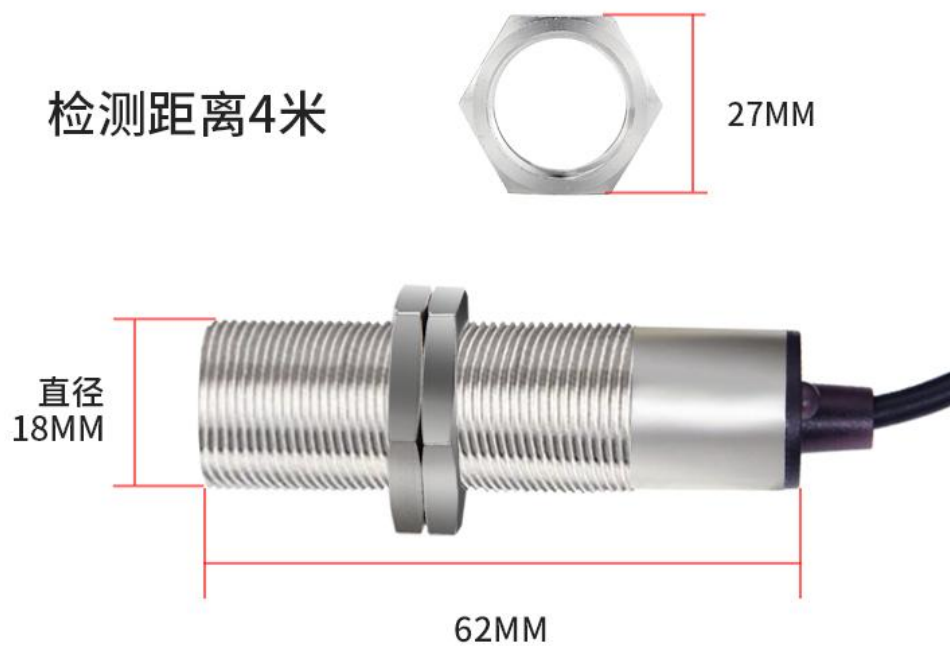
数据接口	通讯接口	RS485
	波特率	1200~115200bps
	数据格式	8N1, 8O1, 8E1, 8N2, 8O2, 8E2
	通讯协议	Modbus-RTU
测量距离	20-4000mm	
传输速率	0.1-30Hz	
精度	±3‰	
电源参数	电源规格	9~36V
	功耗	12V-0.24W
工作环境	工作温度、湿度	-15℃~55℃, 0%RH~95%RH
分辨率	1mm	
视野角	19°	
多连接	支持	
其他	尺寸	62mm*17.75mm (长、直径)

## 二、外观尺寸

### 2.1 产品外观



### 2.2 产品尺寸图



### 三、产品接线图、跳线、指示灯说明

#### 3.1 设备接口



- 底部 4 槽接线位:

VCC: 电源正极/继电器公共端

GND: 电源负极

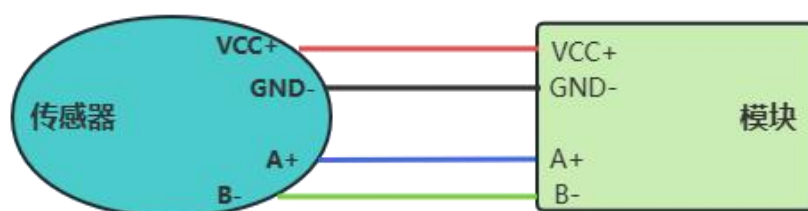
A+: RS485 通讯线 A

B+: RS485 通讯线 B

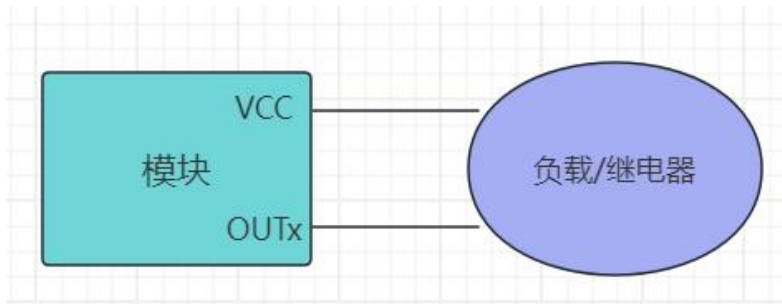
(带有开关量输出的版本, 橙色为输出信号线)

#### 3.2 接线图

##### (1) 传感器与模块接线图



## (2) 开关量输出接线图



## 3.3 使用测试说明

1. 给设备接入电源和 USB 转 485 串口线接入电脑，使用我司提供的参数配置软件进行参数的读取和配置。



2. 设备支持标准 Modbus-RTU 协议，可读取设备 0x0200(512)或者 0x0201(513)地址获取距离数据，具体请查看第四章。

3. 传感器默认支持众山用户脚本，带有开关量输出的型号可以配置脚本进行开关量输出，可根据实际的距离进行一些简单的逻辑判断后输出，比如 0 当检测距离小于 200mm 时输出一个开关量信号，脚本实例如下：

```
@DISP=%d, I1@SET=I2, 200@IF=(I1<I2)@DO1=1@ELSE@DO1=0@ENDIF@D=0.01
```

以上例子便是将 I2 变量赋值成 200，然后通过 IF 判断语句判断 I1 和 I2 的值，I1 变量已经从内部映射成了当前的距离值，所以可直接与 I2 进行判断，当距离小于 200，便使用 @DO1=1 语句进行输出，否则使用 @DO1=0 语句关闭输出，@D=0.01 为延时 10ms，@DISP=%d, I1 语句为显示当前距离值。

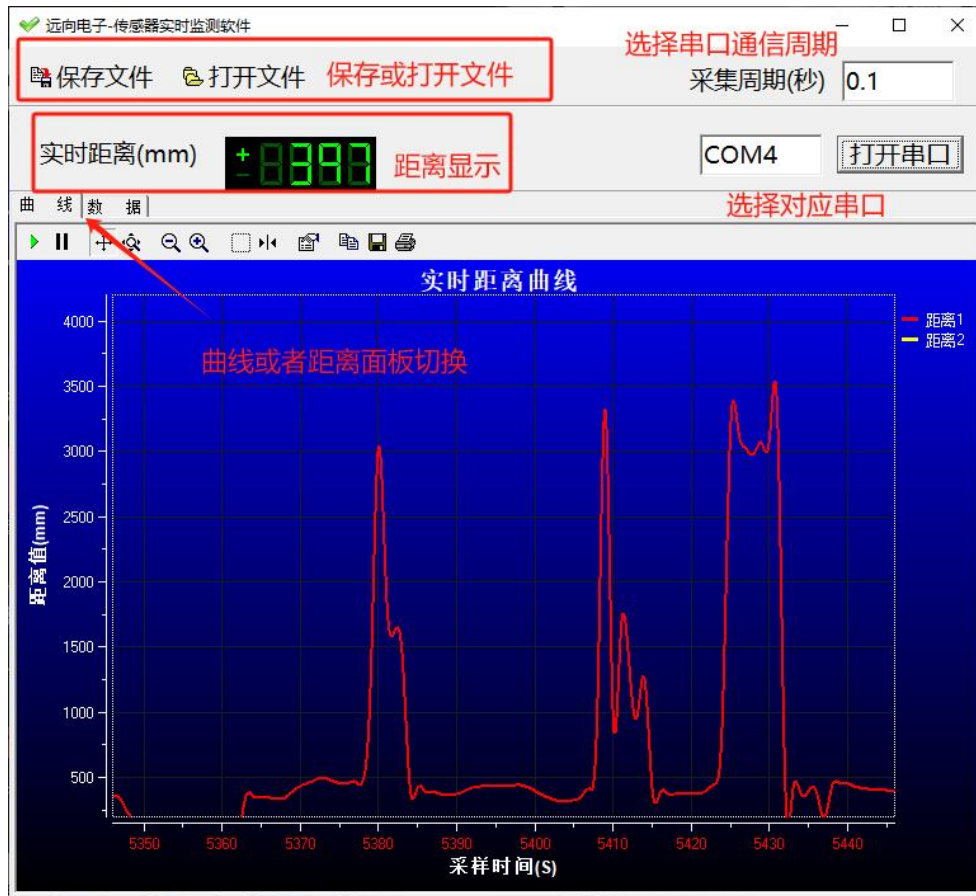
同理，IF 条件判断语句是可以进行嵌套的，例如当距离距离大于 200mm 并且小于 1000 时输出一个开关量信号：

```
@DISP=%d, I1@IF=(I1>200)@IF=(I1<1000)@DO1=1@ELSE@DO1=0@ENDIF@ELSE@DO1=0@ENDIF@D=0.01
```

若要输出 CVS 表格，则需要加入脚本：@PRINT=\$@PRINT=I1@PRINT=;@D=1

注：脚本功能请参考脚本手册，可视化软件请在众山物联电子知识库进行下载

若不想配置脚本让传感器主动往串口输出报文，则可以使用我司提供的 **TOF 测距曲线记录软件** 进行测试，只需打开响应串口，设置好采集周期，便可以进行当前距离的采集，左上角可保存当前距离为 CSV 表格，同理，之前保存过的文件也可以重新使用软件打开来查看曲线。



### 3.4 传感器校准方法

传感器距离参数已映射到脚本变量 I1 中，也就是 0x0201 寄存器，可通过对这个寄存器进行一些操作来校准实际距离：

1: **线性误差**，可直接乘以一个斜率进行校准，例如，实际 100mm，传感器距离为 95mm，并且在 200mm 时传感器变为了 190mm，那么我们可以给 I1 变量乘以 1.0526，也就是  $(100/95)$ ，具体脚本如下：

```
@SET=I1,DIST@MUL=I1,1.0526@D=0.01
```

2: **固定误差**，可以直接加减一个误差数来补充，例如，实际 100mm，传感器为 95mm，并且在 200mm 时传感器为 195mm，那么可以加上这个固定误差 5mm。脚本如下：

```
@SET=I1,DIST@INC=I1,5@D=0.01
```

不同版本的传感器的 @SET=I1,DIST 语句可能有所不同。

**脚本语句可参考远向 DIDO 脚本编程手册，或联系客服获取技术支持！**

### 3.5 传感器使用注意事项

1. 激光测距传感器视野角度（FOV）为  $19^\circ$ （激光发射器+接收器），在测量距离时，被测物体和模块尽量平衡，如果有偏差，偏差角不能大于视野角度。

2. 激光光源发出不可见光束（通过智能手机摄像头可见）。光源发射角之外的任何障碍物都会影响测距效果。

3. 传感器易受自然光干扰。一般来说，较强的自然光影响较大，导致测量距离缩短、精度下降、测量结果波动较大。在光线充足的情况下（例如阳光直射），建议在近距离检测场景中使用该模块。

4. 传感器在使用盖玻片后，因为受盖玻片的反射率等影响，测量的最远距离会略微变短。

5. 传感器为光学器件，应存放在无尘、防潮的环境中。在使用过程中，保持模块表面清洁非常重要，以确保测量结果准确。

6. 激光测距传感器不具备防摔功能，所以要轻拿轻放以免传感器里面的激光头摔坏，而影响使用。

7. 激光测距传感器不能对着人的眼睛直接测量，防止对人的眼睛造成伤害，这是跟激光波长，脉冲宽度，间隙时间、光束的能量、入射角度、受照组织特性等因素有关。



## 四、ModbusRTU 通讯协议地址以及案例说明

### 4.1 通讯协议

本产品支持标准 Modbus RTU 从站协议，能够支持标准 Modbus RTU 组态软件，详细介绍参考本文第六章内容

### 4.2 寄存器地址

寄存器地址	名称	字节数	说明	备注
0x0200(512)	Distance	2	实时测量距离	每个测量通道占 1 个 Modbus 寄存器，2 个字节，格式为整数
0x0201(513)	Distance	2	实时测量距离 (可通过脚本编程)	每个测量通道占 1 个 Modbus 寄存器，2 个字节，格式为整数
0x0014(20)	DO1	2	开关量 DO 输出通道	0x0000 表示断开 0x0001 表示吸合

注：0x0201（513）寄存器已经映射到脚本变量 I1 中，可通过脚本实现自定义操作

### 4.3 Modbus RTU 功能码

功能码	操作	说明
03	读取距离寄存器值	读取距离寄存器值
04	读取距离寄存器值	读取距离寄存器值
06	写单个 DO	0000 为断开 0001 为闭合
10	写多个 DO	详情请参考第六章

详细讲解参照本文第六章内容

### 4.4 Modbus 通讯实例

#### (1) 读取距离：

#### c. 用 03 功能码读取整数距离：

发送：01 03 02 01 00 01 D4 72

接收：01 03 02 01 F0 B9 90

整数读出数值为 496（0x01F0）

#### d. 用 06 功能码操作 DO：

发送：01 06 00 14 00 01 08 0E

接收：01 06 00 14 00 01 08 0E

直接操作 DO1，0000 为断开 0001 为闭合

## 五、软件操作

设备参数配置教程，结合《用户测试文档》即可对设备进行简单测试

### 5.1 配置软件

参数配置软件介绍：



5.1.1 配置软件包含有：

- **功能区：** 包含有配置软件所支持功能, 以及功能那个切换选项
- **参数配置主区域：** 参数配置主要区域, 参数项的读取、写入临时列表
- **串口/命令集区：** 涉及模块的参数读、写、重启等操作
- **串口日志区：** 命令集的操作日志

5.1.2 参数配置准备：

- (1) 用 USB-485 工具连接设备到电脑
- (2) 在串口配置框内配置串口波特率、停止位、校验位、数据位；（默认波特率 9600，数据位 8，停止位 1，校验位 None）
- (3) 选择串口配置框子项“命令集”



- (4) 点击“**读取参数**”命令按钮，读取设备参数（不同设备拥有不同指令集）
- (5) 双击对应参数项的“**参数值**”，然后对参数进行修改
- (6) 修改完参数后需要点击命令集里的“**设置参数**”，写入到模块中
- (7) 写入完成在日志区域会提示成功。



- (8) 通过点击“**重启设备**”按钮，重启模块设备使配置参数生效

## 5.2 配置基本参数

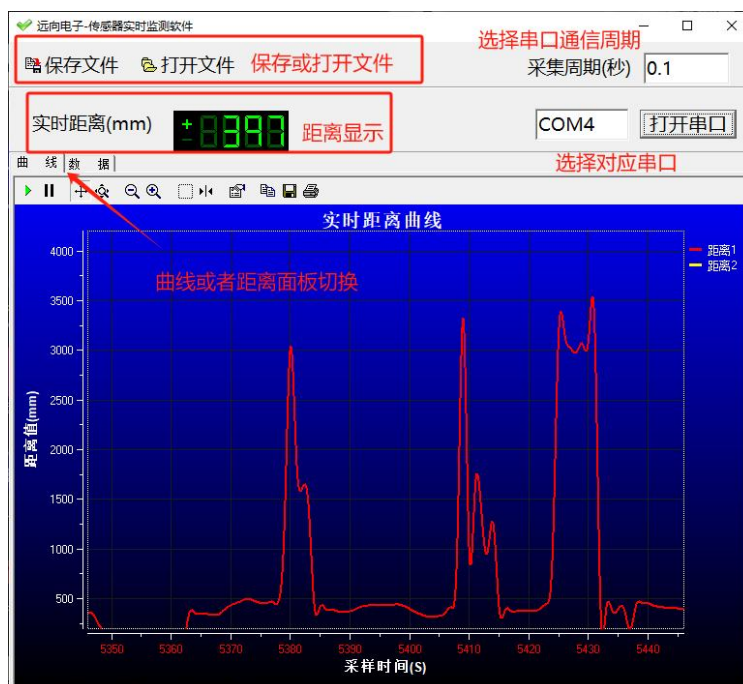
该系列参数涉及到对 485 通讯 Modbus 协议相关配置。

参数名称	参数值	参数说明
<b>&lt;基本参数&gt;</b>		
Modbus地址	<b>双击修改参数</b>	设备的Modbus地址, 1~255
通信模块波特率		与通信模块的波特率一致, 一般设置为9600
通信模块奇偶校验		与通信模块的奇偶校验一致, 一般设置为8N1

- **Modbus 地址**: Modbus 地址参数, 可设置 1~255
- **通讯模块波特率**: 设备 485 通讯波特率 (波特率支持主流的波特率选项)
- **通讯模块就校验**: 设备 485 通讯奇偶校验位, 可配置 8N1, 8E1, 8O1...

## 5.3 距离数据可视化测试

我司提供一个可将距离数据形成折线图的软件, 可实现简单的模块功能测试。操作步骤如下图:



- 打开对应串口
- 设置采集周期
- 根据需要可进行 CSV 文件的保存或打开

**注意：此时界面下方会出现当前的距离数据值。若无上行数据请检查 USB-485 转换器是否正常工作，或尝试模块 485 AB 之间接 120 欧电阻。**  
 详细的测试演示可参考[“用户测试文档”](#)

## 六、协议详解

地址域	功能码	数据	差错检验
-----	-----	----	------

Modbus 使用“big-Endian”（大端模式）表示地址和数据项，这就意味着当发射多个字节时，首先发送最高字节。

例如：寄存器地址为 0x0014，首先发送的是 0x00，然后才是 0x14。

一个正常的 Modbus 响应：响应功能码=请求功能码。

一个 Modbus 的异常响应：响应功能码=请求功能码+0x80，提供一个异常码来指示差错原因。

### 6.1 功能码描述

#### 6.1.1 01 读线圈

可以使用此功能码读取继电器 DOx 的状态。

请求 PDU 详细说明了起始地址，即指定第一个线圈的地址和线圈数量，从零开始寻址线圈，因此寻址线圈 1-N 为 0-(N-1)。

响应 PDU 中 N 个字节的线圈状态的每一个 bit 位代表一个线圈的状态，状态 1=ON，0=OFF。第一个字节的最低位 LSB 代表第 0 号线圈的状态（即起始地址指定的线圈号为 0 号线圈），其他线圈依次类推，一直到这个字节的最高位 MSB 为止，并且后续字节中都是由低到高代表连续的各线圈状态。

如果线圈数量不是 8 的倍数，将用零填充剩余最后数据字节中的剩余比特，字节数量域说明了数据的完整字节数。

请求 PDU

地址	1 个字节	
功能码	1 个字节	0x01
起始地址	2 个字节	0x0014 至 0x0015
线圈数量	2 个字节	n(1 至 n-1)
CRC 校验	2 个字节	

注：线圈状态的字节数 N=线圈数量 n/8，如果余数不等于 0，则 N=n/8+1

错误响应 PDU

地址	1 个字节	
功能码	1 个字节	0x81 （请求功能码+0x80）
异常码	1 个字节	0x01 或 0x02 或 0x03 或 0x04
CRC 校验	2 个字节	

这是一个读离散量 D01 的实例

请求		响应	
地址	01	地址	01
功能码	01	功能码	01
起始地址高 H	00	字节数	01
起始地址低 L	14	D01-D04 状态	01
线圈数量高 H	00	CRC 校验高 H	90
线圈数量低 L	01	CRC 校验低 L	48
CRC 校验高 H	BD		
CRC 校验低 L	CE		

发送: 010100140001BDCE      RTU 响应: 010101019048

D01 的状态字节为 0D, 二进制 00000001, D01 是这个字节的 LSB(第 0 位)为 1 表示闭合, 其他 D0x 是第(x-1)位为 0 表示断开, 用 0 填充未使用位。

### 6.1.2 03 读保持寄存器/04 读输入寄存器

使用该功能码可以读取所有寄存器包括 AI<sub>x</sub>、DO<sub>x</sub>、DI<sub>x</sub> 的状态。

请求 PDU 详细说明了起始寄存器地址和寄存器数量, 从 0 开始寻址寄存器, 因此寻址寄存器 1-N 为 0-(N-1)。

响应报文中的寄存器数据每个寄存器有 2 个字节, 对于每一个寄存器, 第一个字节代表寄存器值的高位, 第二个字节代表寄存器值的低位。字节数为寄存器数量乘以 2。对于 AI, 一个通道占用 2 个寄存器, 4 个字节的值使用浮点数表示, 对于 DO<sub>x</sub>, 2 个字节的值 0000 代表继电器断开, 0001 代表继电器闭合, 对于 DI<sub>x</sub>, 2 个字节的值 0000 代表开关量无输入, 0001 代表有输入。

请求 PDU

地址	1 个字节	
功能码	1 个字节	0x03 或 04
起始地址	2 个字节	0x0000 至 0x0017
寄存器数量	2 个字节	n(1 至 N)
CRC 校验	2 个字节	

响应 PDU

地址	1 个字节	
功能码	1 个字节	0x03 或 0x04
字节数	1 个字节	N=2*n
寄存器值	N 个字节	N=2*n, n 为寄存器数量
CRC 校验	2 个字节	

错误响应 PDU

地址	1 个字节	
----	-------	--

功能码	1 个字节	0x83 或 0x84 (请求功能码+0x80)
异常码	1 个字节	0x01 或 0x02 或 0x03 或 0x04
CRC 校验	2 个字节	

这是一个读模拟量输入 AI1 的实例

请求		响应	
地址	01	地址	01
功能码	03	功能码	03
起始地址高 H	00	字节数	04
起始地址低 L	00	AI1 值	4 个字节
寄存器数量高 H	00	CRC 校验高 H	
寄存器数量低 L	02	CRC 校验低 L	
CRC 校验高 H	C4		
CRC 校验低 L	0B		

发送: 010300000002C40B RTU 响应:0103044019999AD40F

### 6.1.3 05 写单个线圈

可以使用该功能码写单个继电器 DO<sub>x</sub> 为断开或闭合

请求数据域中的常量说明请求的 ON/OFF 状态, 十六进制值 0xFF00 请求输出为 ON(闭合), 十六进制值 0x0000 请求输出为 OFF(断开), 其他所有值都是非法的, 对输出不起作用, RTU 返回错误响应。

请求域中的输出地址规定了要写入线圈的地址。

正常响应是请求的应答, 在写入线圈状态后返回这个正常响应。

请求 PDU

地址	1 个字节	
功能码	1 个字节	0x05
输出地址	2 个字节	0x0014 至 0x0015
输出值	2 个字节	0x0000 或 0xFF00
CRC 校验	2 个字节	

响应 PDU

地址	1 个字节	
功能码	1 个字节	0x05
输出地址	2 个字节	0x0014 至 0x0015
输出值	2 个字节	0x0000 或 0xFF00
CRC 校验	2 个字节	

错误响应 PDU

地址	1 个字节	
功能码	1 个字节	0x85 (请求功能码+0x80)

异常码	1 个字节	0x01 或 0x02 或 0x03 或 0x04
CRC 校验	2 个字节	

这是一个请求写线圈 D01 为 ON(闭合)的实例

请求		响应	
地址	01	地址	01
功能码	05	功能码	05
输出地址高 H	00	输出地址高 H	00
输出地址低 L	14	输出地址低 L	14
输出值高 H	FF	输出值高 H	FF
输出值低 L	00	输出值低 L	00
CRC 校验高 H	CC	CRC 校验高 H	CC
CRC 校验低 L	3E	CRC 校验低 L	3E

发送：01050014FF00CC3E

RTU 响应：01050014FF00CC3E

#### 6.1.4 06 写单个寄存器

可以使用该功能码写单个继电器 D0x 为断开或闭合。

请求数据域中的寄存器值说明请求的 ON/OFF 状态，十六进制值 0001 请求输出为 ON(闭合)，十六进制值 0x0000 请求输出为 OFF(断开)。

请求域中的寄存器地址规定了要写入线圈的地址。

正常响应是请求的应答，在写入线圈状态后返回这个正常响应。

请求 PDU

地址	1 个字节	
功能码	1 个字节	0x06
寄存器地址	2 个字节	0x0014 至 0x0015
寄存器值	2 个字节	0x0000 至 0xFFFF
CRC 校验	2 个字节	

响应 PDU

地址	1 个字节	
功能码	1 个字节	0x06
寄存器地址	2 个字节	0x0014 至 0x0015
寄存器值	2 个字节	0x0000 至 0xFFFF
CRC 校验	2 个字节	

错误响应 PDU

地址	1 个字节	
功能码	1 个字节	0x86 (请求功能码+0x80)
异常码	1 个字节	0x01 或 0x02 或 0x03 或 0x04
CRC 校验	2 个字节	

这是一个请求写线圈 D01 为 ON(闭合)的实例

请求		响应	
地址	01	地址	01
功能码	06	功能码	06
寄存器地址高 H	00	寄存器地址高 H	00
寄存器地址低 L	14	寄存器地址低 L	14
寄存器值高 H	00	寄存器值高 H	00
寄存器值低 L	01	寄存器值低 L	01
CRC 校验高 H	08	CRC 校验高 H	08
CRC 校验低 L	0E	CRC 校验低 L	0E

发送：010600140001080E

RTU 响应：010600140001080E

### 6.1.5 0F 写多个线圈

可以使用此功能码写多个继电器 DOx 为断开或闭合。

请求 PDU 详细说明了起始地址，即指定第一个线圈的地址和线圈数量，从零开始寻址线圈，因此寻址线圈 1-N 为 0-(N-1)。

请求数据域中的内容说明了被请求的 ON/OFF 状态，域比特位中的逻辑“1”请求相应输出为 ON，域比特位中的逻辑“0”请求相应输出为 OFF。从数据域中第一个字节的 bit0 开始到 bit7，然后到第二个字节的 bit0，依次表示第一个线圈到第 n 个线圈的 ON/OFF 值。

正常响应返回功能码、起始地址和线圈数量。

请求 PDU

地址	1 个字节	
功能码	1 个字节	0x0F
起始地址	2 个字节	0x0014 至 0x0015
线圈数量	2 个字节	n(1 至 N)
字节数	1 个字节	$N=n/8$ , 或 $N=n/8+1$
输出值	N 个字节	
CRC 校验	2 个字节	

注：线圈输出字节数  $N=线圈数量 n/8$ ，如果余数不等于 0，则  $N=n/8+1$

响应 PDU

地址	1 个字节	
功能码	1 个字节	0x0F
起始地址	2 个字节	0x0014 至 0x0015
线圈数量	2 个字节	n(1 至 2)
CRC 校验	2 个字节	

错误响应 PDU

地址	1 个字节	
功能码	1 个字节	0x8F (请求功能码+0x80)



异常码	1 个字节	0x01 或 0x02 或 0x03 或 0x04
CRC 校验	2 个字节	

这是一个请求从线圈 D01 开始写入 1 个线圈的实例

请求		响应	
地址	01	地址	01
功能码	0F	功能码	0F
起始地址高 H	00	起始地址高 H	00
起始地址低 L	14	起始地址低 L	14
线圈数量高 H	00	线圈数量高 H	00
线圈数量低 L	01	线圈数量低 L	01
字节数	01	CRC 校验高 H	D4
输出值	01	CRC 校验低 L	0F
CRC 校验高 H	DF		
CRC 校验低 L	54		

**发送：010F0014000201012F51      RTU 响应：010F00140001D40F**

D01 的输出值为 01，二进制 00000001，D01 是这个字节的 LSB(第 0 位)为 0 表示断开，D0x 是第(x-1)位为 1 表示闭合，用 0 填充剩余未使用位。

### 6.1.6 10 写多个寄存器

使用该功能码可以写连续寄存器 D0x 的状态。

请求 PDU 详细说明了起始寄存器地址、寄存器数量、字节数和寄存器值，从零开始寻址寄存器，因此寻址寄存器 1-N 为 0-(N-1)。

寄存器数据中每个寄存器有 2 个字节，对于每一个寄存器，第一个字节代表寄存器值的高位，第二个字节代表寄存器值的低位。字节数为寄存器数量乘以 2，2 个字节的值 0000 代表继电器断开，0001 代表继电器闭合。

正常响应返回功能码、起始地址和被写入寄存器的数量。

请求 PDU

地址	1 个字节	
功能码	1 个字节	0x10
起始地址	2 个字节	0x0014 至 0x0015
寄存器数量	2 个字节	n(1 至 N)
字节数	1 个字节	N=2*n
寄存器值	N 个字节	N=2*n,    n 为寄存器数量
CRC 校验	2 个字节	

响应 PDU

地址	1 个字节	
功能码	1 个字节	0x10
起始地址	2 个字节	0x0014 至 0x0015

寄存器数量	2 个字节	n(1 至 2)
CRC 校验	2 个字节	

错误响应 PDU

地址	1 个字节	
功能码	1 个字节	0x90 (请求功能码+0x80)
异常码	1 个字节	0x01 或 0x02 或 0x03 或 0x04
CRC 校验	2 个字节	

这是一个控制继电器 DOx 的实例

请求		响应	
地址	01	地址	01
功能码	10	功能码	10
起始地址高 H	00	起始地址高 H	00
起始地址低 L	14	起始地址低 L	14
寄存器数量高 H	00	寄存器数量高 H	00
寄存器数量低 L	01	寄存器数量低 L	01
字节数	02	CRC 校验高 H	41
DO1 寄存器值高 H	00	CRC 校验低 L	CD
DO1 寄存器值高 L	01		
CRC 校验高 H	64		
CRC 校验低 L	84		

发送: 0110001400010200016484

RTU 响应: 01100014000141CD

DO1 寄存器值为 0001 表示闭合

## 6.2 错误码描述

错误码含义: 当 DTU 收到错误的 Modbus 指令时, 会返回功能码为请求功能码+0x80, 紧随着一个字节的错误码代表出错原因。

错误码 01: 表示不支持的功能码, 众山 DTU 支持上述 8 种功能码, 除此之外的功能码都会返回错误码为 01 的错误。

错误码 02: 表示起始地址不存在或者起始地址加上寄存器数量后的地址不存在。总的来说表示访问的寄存器不存在。

错误码 03: 表示寄存器数量不符合规范或者寄存器值非法。

错误码 04: 表示读写寄存器错误。

## 6.3 CRC 校验算法

CRC 即[循环冗余校验码](#) (Cyclic Redundancy Check): 是数据通信领域中最常用的一种查错校验码, 其特征是信息字段和校验字段的长度可以任意选定。循环冗余检查 (CRC) 是一种数据传输检错功能, 对数据进行多项式计算, 并将

得到的结果附在帧的后面，接收设备也执行类似的算法，以保证数据传输的正确性和完整性。

ModbusRTU 的 CRC16 计算初值: 0xFFFF

ModbusRTU 的 CRC16 计算多项式 0xA001 (二进制:1010 0000 0000 0001)

附 CRC 校验算法代码:

```
uint16_t mb_crc( uint8_t* snd, uint16_t num )
{
    uint8_t CRC_Lb, CRC_Hb;
    uint8_t CRC_L, CRC_H;
    uint16_t crc;

    CRC_H = 0xFF;
    CRC_L = 0xFF;

    for ( uint16_t i = 0; i < num; i++ ) {
        CRC_L = CRC_L ^ snd[ i ];
        for ( uint16_t j = 0; j < 8; j++ ) {
            CRC_Lb = CRC_L;
            if ( ( CRC_L & 1 ) == 1 ) {
                CRC_L = ( CRC_L - 1 ) / 2;
                CRC_Lb = CRC_L;
                CRC_Hb = CRC_H;
                if ( ( CRC_H & 1 ) == 1 ) {
                    CRC_L = CRC_L + 128;
                    CRC_Lb = CRC_L;
                    CRC_H = ( CRC_H - 1 ) / 2;
                    CRC_Hb = CRC_H;
                } else {
                    CRC_H = CRC_H / 2;
                    CRC_Hb = CRC_H;
                }
            }
            CRC_L = CRC_L ^ 1;
            CRC_Lb = CRC_L;
            CRC_H = CRC_H ^ 0xA0;
            CRC_Hb = CRC_H;
        } else {
            CRC_L = CRC_L / 2;
            CRC_Lb = CRC_L;
            CRC_Hb = CRC_H;
            if ( ( CRC_H & 1 ) == 1 ) {
                CRC_L = CRC_L + 128;
                CRC_Lb = CRC_L;
                CRC_H = ( CRC_H - 1 ) / 2;
                CRC_Hb = CRC_H;
            }
        }
    }
}
```

```
        } else {
            CRC_H = CRC_H / 2;
            CRC_Hb = CRC_H;
        }
    }
}
crc = CRC_L;
crc <<= 8;
crc |= CRC_H;
return crc;
}
```