EH-TR010
MODBUS 手册

V0.21

目录

一、概述

TR010 数字模块（简称模块）支持 Modbus-RTU。本手册详细介绍了 Modbus-RTU 协议及其寄存器映射和高级命令。

模块将全部的功能都映射四种类型的 Modbus 对象（寄存器）上，如果使用主机支持 Modbus-RTU 协议或者使用 Modbus 库进行编程，就可以直接操作这些对象，不用关注协议实现的细节（第二节所述内容）。

对象类型	对象类型	访问类型	作用
离散量	单个比特	只读	读取模块状态标志
线圈	单个比特	读写	控制模块执行功能
输入寄存器	16-比特字	只读	读写参数
保持寄存器	16-比特字	读写	读取重量等信息

二、Modbu-RTU 协议简介

1、帧格式

在 RTU 模式，报文帧由时长至少为 3.5 个字符时间的空闲间隔区分。在后续的部分，这个时间区间被称作 t3.5。

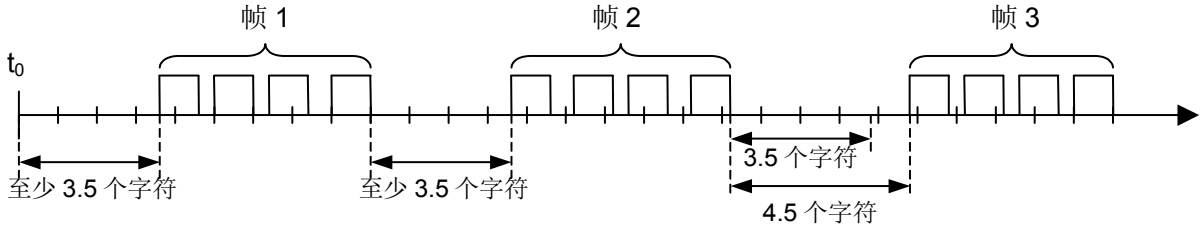
例如：

3.5t（9600） = 1000 * 11 * 3.5 / 9600 ≈ 4ms

3.5t（19200）= 1000 * 11 * 3.5 / 38400 ≈ 2ms

3.5t（>19200）Modbu 协议规定统一为 1.75ms

对于某些主机（如 PC），如果实现不了精确定时，只需将帧间隔判断时间设置为大于 t3.5 即可。



MODBUS 报文

起始	地址	功能代码	数据	CRC 校验	结束
≥ 3.5 字符	8 位	8 位	N x 8 位	16 位	≥ 3.5 字符

2、CRC 校验

在 RTU 模式中，CRC 检验整个报文的内容（包括“地址”、“功能代码”和“数据”）。附加在报文之后时，首先附加低字节，然后是高字节。CRC 高字节为报文发送的最后一个

字节。

CRC 生成 C 语言函数如下：

```
//返回值为 CRC 校验结果，puchMsg 为数据指针，usDataLen 为数据长度
unsigned short CRC16 (unsigned char * puchMsg, unsigned short usDataLen )
{
    unsigned char uchCRCHi = 0xFF ;
    unsigned char uchCRCLo = 0xFF ;
    unsigned uIndex ;

    while (usDataLen--)
    {
        uIndex = uchCRCLo ^ *puchMsgg++ ;
        uchCRCLo = uchCRCHi ^ auchCRCHi[uIndex} ;
        uchCRCHi = auchCRCLo[uIndex] ;
    }
    return (uchCRCHi << 8 | uchCRCLo) ;
}
/* 高位字节的 CRC 值 */
static unsigned char auchCRCHi[] = {
    0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81,
    0x40, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0,
    0x80, 0x41, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x00, 0xC1, 0x81, 0x40, 0x01,
    0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0, 0x80, 0x41,
    0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x00, 0xC1, 0x81,
    0x40, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0,
    0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01,
    0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40,
    0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81,
    0x40, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0,
    0x80, 0x41, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x00, 0xC1, 0x81, 0x40, 0x01,
    0xC0, 0x80, 0x41, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41,
    0x00, 0xC1, 0x81, 0x40, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81,
    0x40, 0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0,
    0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01,
    0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41,
    0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81,
    0x40
};
/* 低位字节的 CRC 值 */
static char auchCRCLo[] = {
    0x00, 0xC0, 0xC1, 0x01, 0xC3, 0x03, 0x02, 0xC2, 0xC6, 0x06, 0x07, 0xC7, 0x05, 0xC5, 0xC4,
    0x04, 0xCC, 0x0C, 0x0D, 0xCD, 0x0F, 0xCF, 0xCE, 0x0E, 0x0A, 0xCA, 0xCB, 0x0B, 0xC9, 0x09,
    0x08, 0xC8, 0xD8, 0x18, 0x19, 0xD9, 0x1B, 0xDB, 0xDA, 0x1A, 0x1E, 0xDE, 0xDF, 0x1F, 0xDD,
    0x1D, 0x1C, 0xDC, 0x14, 0xD4, 0xD5, 0x15, 0xD7, 0x17, 0x16, 0xD6, 0xD2, 0x12, 0x13, 0xD3,
    0x11, 0xD1, 0xD0, 0x10, 0xF0, 0x30, 0x31, 0xF1, 0x33, 0xF3, 0xF2, 0x32, 0x36, 0xF6, 0xF7,
    0x37, 0xF5, 0x35, 0x34, 0xF4, 0x3C, 0xFC, 0xFD, 0x3D, 0xFF, 0x3F, 0x3E, 0xFE, 0xFA, 0x3A,
    0x3B, 0xFB, 0x39, 0xF9, 0xF8, 0x38, 0x28, 0xE8, 0xE9, 0x29, 0xEB, 0x2B, 0x2A, 0xEA, 0xEE,
    0x2E, 0x2F, 0xEF, 0x2D, 0xED, 0xEC, 0x2C, 0xE4, 0x24, 0x25, 0xE5, 0x27, 0xE7, 0xE6, 0x26,
    0x22, 0xE2, 0xE3, 0x23, 0xE1, 0x21, 0x20, 0xE0, 0xA0, 0x60, 0x61, 0xA1, 0x63, 0xA3, 0xA2,
    0x62, 0x66, 0xA6, 0xA7, 0x67, 0xA5, 0x65, 0x64, 0xA4, 0x6C, 0xAC, 0xAD, 0x6D, 0xAF, 0x6F,
    0x6E, 0xAE, 0xAA, 0x6A, 0x6B, 0xAB, 0x69, 0xA9, 0xA8, 0x68, 0x78, 0xB8, 0xB9, 0x79, 0xBB,
    0x7B, 0x7A, 0xBA, 0xBE, 0x7E, 0x7F, 0xBF, 0x7D, 0xBD, 0xBC, 0x7C, 0xB4, 0x74, 0x75, 0xB5,
    0x77, 0xB7, 0xB6, 0x76, 0x72, 0xB2, 0xB3, 0x73, 0xB1, 0x71, 0x70, 0xB0, 0x50, 0x90, 0x91,
    0x51, 0x93, 0x53, 0x52, 0x92, 0x96, 0x56, 0x57, 0x97, 0x55, 0x95, 0x94, 0x54, 0x9C, 0x5C,
    0x5D, 0x9D, 0x5F, 0x9F, 0x9E, 0x5E, 0x5A, 0x9A, 0x9B, 0x5B, 0x99, 0x59, 0x58, 0x98, 0x88,
    0x48, 0x49, 0x89, 0x4B, 0x8B, 0x8A, 0x4A, 0x4E, 0x8E, 0x8F, 0x4F, 0x8D, 0x4D, 0x4C, 0x8C,
    0x44, 0x84, 0x85, 0x45, 0x87, 0x47, 0x46, 0x86, 0x82, 0x42, 0x43, 0x83, 0x41, 0x81, 0x80,
    0x40
};
```

CRC 例子：

地址	功能码	数据				CRC 校验	
0x01	0x04	0x00	0x00	0x00	0x01	0x31	0xCA
0x01	0x03	0x00	0x7B	0x00	0x02	0xB4	0x12
0x0C	0x03	0x00	0xDE	0x00	0x79	0xE5	0x0F

3、地址

地址（即通讯地址）1~247，用于模块的寻址，模块只对地址相符的命令执行动作和应答。在一个总线上，每个模块都有唯一的地址，主机通过不同的地址来区别地访问各个模块。

地址 0 为广播地址，总线上所有模块都会接收并执行动作，但不应答。

4、功能代码

功能代码（简称功能码）决定了请求的类型，也决定了数据字段的含义。不同功能码的请求帧和响应帧中数据字段的含义不尽相同，见下述各个功能码的说明。

访问模式	功能	功能码
比特访问	读输入离散量	02
	读线圈	01
	写单个线圈	05
	写多个线圈	15
16 比特访问	读输入寄存器	04
	读多个寄存器	03
	写单个寄存器	06
	写多个寄存器	16
	读/写多个寄存器	23
	屏蔽写寄存器	22

为了叙述方便，下面各个功能码说明中，只包含功能码字段和数据字段，省略了地址字段和 CRC 校验字段。

5、读线圈（01）

请求 PDU

功能码	1 个字节	**0x01**
起始地址	2 个字节	0x0000 至 0xFFFF
线圈数量	2 个字节	1 至 2000（0x7D0）

响应 PDU

功能码	1 个字节	0x01
字节数	1 个字节	N*
线圈状态	N 个字节	n＝N 或 N+1

*N＝输出数量/8，如果余数不等于 0，那么　N = N+1

如果返回的输出数量不是八的倍数，将用零填充最后数据字节中的剩余比特（一直到字节的高位端）。字节数量域说明了数据的完整字节数。

这是一个请求读线圈输出 20-38 的实例：

请求		响应	
域名	（十六进制）	域名	（十六进制）
功能	01	功能	01
起始地址 Hi	00	字节数	03
起始地址 Lo	13	输出状态 27-20	CD
输出数量 Hi	00	输出状态 35-28	6B
输出数量 Lo	13	输出状态 38-36	05

6、读离散量（02）

请求 PDU

功能码	1 个字节	**0x02**
起始地址	2 个字节	0x0000 至 0xFFFF
输入数量	2 个字节	1 至 2000（0x7D0）

响应 PDU

功能码	1 个字节	0x82
字节数	1 个字节	N*
输入状态	N*×1 个字节	

*N＝输出数量/8，如果余数不等于 0，那么　N = N+1

如果返回的输出数量不是八的倍数，将用零填充最后数据字节中的剩余比特（一直到字节的高位端）。字节数量域说明了数据的完整字节数。

这是一个请求读取离散量输入 197-218 的实例：

请求		响应	
域名	（十六进制）	域名	（十六进制）
功能	02	功能	02
起始地址 Hi	00	字节数	03
起始地址 Lo	C4	输入状态 204-197	AC
输出数量 Hi	00	输入状态 212-205	DB
输出数量 Lo	16	输入状态 218-213	35

7、读保持寄存器（03）

请求

功能码	1 个字节	**0x03**
起始地址	2 个字节	0x0000 至 0xFFFF
寄存器数量	2 个字节	1 至 125（0x7D）

响应

功能码	1 个字节	**0x03**
字节数	1 个字节	2×**N***
寄存器值	**N***×2 个字节	

***N**＝寄存器的数量

这是一个请求读寄存器 108-110 的实例：

请求		响应	
域名	（十六进制）	域名	（十六进制）
功能	03	功能	03
高起始地址	00	字节数	06
低起始地址	6B	寄存器值 Hi（108）	02
高寄存器编号	00	寄存器值 Lo（108）	2B
低寄存器编号	03	寄存器值 Hi（109）	00
		寄存器值 Lo（109）	00
		寄存器值 Hi（110）	00
		寄存器值 Lo（110）	64

将寄存器 108 的内容表示为两个十六进制字节值 02 2B，或十进制 555。将寄存器 109-110 的内容分别表示为十六进制 00 00 和 00 64，或十进制 0 和 100。

8、读输入寄存器（04）

请求

功能码	1 个字节	**0x04**
起始地址	2 个字节	0x0000 至 0xFFFF
输入寄存器数量	2 个字节	0x0001 至 0x007D

响应

功能码	1 个字节	**0x04**
字节数	1 个字节	2×**N***
输入寄存器	N*×2 个字节	

***N**＝输入寄存器的数量

这是一个请求读输入寄存器 9 的实例：

请求		响应	
域名	（十六进制）	域名	（十六进制）
功能	04	功能	04
起始地址 Hi	00	字节数	02
起始地址 Lo	08	输入寄存器 9 Hi	00
输入寄存器数量 Hi	00	输入寄存器 9 Lo	0A
输入寄存器数量 Lo	01		

将输入寄存器 9 的内容表示为两个十六进制字节值 00 0A，或十进制 10。

9、写单个线圈（05）

请求

功能码	1 个字节	**0x05**
输出地址	2 个字节	0x0000 至 0xFFFF
输出值	2 个字节	0x0000 至 0x00

响应

功能码	1 个字节	**0x05**
输出地址	2 个字节	0x0000 至 0xFFFF
输出值	2 个字节	0x0000 至 0xFF00

这是一个请求写线圈 173 为 ON 的实例：

请求		响应	
域名	（十六进制）	域名	（十六进制）
功能	05	功能	05
输出地址 Hi	00	输出地址 Hi	00
输出地址 Lo	AC	输出地址 Lo	AC
输出值 Hi	FF	输出值 Hi	FF
输出值 Lo	00	输出值 Lo	00

１０、写单个保持寄存器（06）

请求

功能码	1 个字节	**0x06**
寄存器地址	2 个字节	0x0000 至 0xFFFF
寄存器值	2 个字节	0x0000 至 0xFFFF

响应

功能码	1 个字节	**0x06**
寄存器地址	2 个字节	0x0000 至 0xFFFF
寄存器值	2 个字节	0x0000 至 0xFFFF

这是一个请求将十六进制 00 03 写入寄存器 2 的实例：

请求		响应	
域名	（十六进制）	域名	（十六进制）
功能	06	功能	06
寄存器地址 Hi	00	输出地址 Hi	00
寄存器地址 Lo	01	输出地址 Lo	01
寄存器值 Hi	00	输出值 Hi	00
寄存器值 Lo	03	输出值 Lo	03

１１、写多个线圈（15）

请求 PDU

功能码	1 个字节	**0x0F**
起始地址	2 个字节	0x0000 至 0xFFFF
输出数量	2 个字节	0x0001 至 0x07B0
字节数	1 个字节	**N***
输出值	**N***×1 个字节	

***N**＝输出数量/8，如果余数不等于 0，那么　N = N+1

响应 PDU

功能码	1 个字节	**0x0F**
起始地址	2 个字节	0x0000 至 0xFFFF
输出数量	2 个字节	0x0001 至 0x07B0

这是一个请求从线圈 20 开始写入 10 个线圈的实例：

请求的数据内容为两个字节：十六进制 CD 01 (二进制 1100 1101 0000 0001)。使用下列方法，二进制比特对应输出。

比特：1　1　0　0　1　1　0　1　0　0　0　0　0　0　0　1

输出：27　26　25　24　23　22　21　20　–　–　–　–　–　–　29　28

传输的第一字节(十六进制 CD)寻址为输出 27-20，在这种设置中，最低有效比特寻址为最低输出（20）。

传输的下一字节(十六进制 01)寻址为输出 29-28，在这种设置中，最低有效比特寻址为最低输出（28）。

应该用零填充最后数据字节中的未使用比特。

请求		响应	
域名	（十六进制）	域名	（十六进制）
功能	0F	功能	0F
起始地址 Hi	00	起始地址 Hi	00
起始地址 Lo	13	起始地址 Lo	13
输出数量 Hi	00	输出数量 Hi	00
输出数量 Lo	0A	输出数量 Lo	0A
字节数	02		
输出值 Hi	CD		
输出值 Lo	01		

１２、写多个保持寄存器（16）

请求 PDU

功能码	1 个字节	**0x10**
起始地址	2 个字节	0x0000 至 0xFFFF
寄存器数量	2 个字节	0x0001 至 0x0078
字节数	1 个字节	2×**N***
寄存器值	**N***×2 个字节	值

***N**＝寄存器数量

响应 PDU

功能码	1 个字节	**0x10**
起始地址	2 个字节	0x0000 至 0xFFFF
寄存器数量	2 个字节	1 至 123（0x7B）

这是一个请求将十六进制 00 0A 和 01 02 写入以 2 开始的两个寄存器的实例：

请求		响应	
域名	（十六进制）	域名	（十六进制）
功能	10	功能	10
起始地址 Hi	00	起始地址 Hi	00
起始地址 Lo	01	起始地址 Lo	01
寄存器数量 Hi	00	寄存器数量 Hi	00
寄存器数量 Lo	02	寄存器数量 Lo	02
字节数	04		
寄存器值 Hi	00		
寄存器值 Lo	0A		
寄存器值 Hi	01		
寄存器值 Lo	02		

1 3、异常响应

当主机向模块发送请求时，可能出现下面四种可能事件之一：

- 如果模块接收到无通信错误的请求，并且可以正常地处理询问，那么模块将返回一个正常响应。
- 如果模块接收到无通信错误的请求，但不能处理这个请求，模块将返回一个异常响应。
- 如果由于通信错误，模块没有接收到请求，那么不能返回响应。主机将超时错误。
- 如果模块收到请求，但是检测到一个通信错误（奇偶校验、CRC、...），那么不能返回响应。主机将超时错误。

异常响应报文有两个与正常响应不同的域：

功能码域：

在正常响应中，响应功能码与最初请求的功能码相同。

在异常响应中，响应功能码 = 最初请求的功能码 + 0x80。

数据域：

在正常响应中，数据域返回请求中要求的数据。

在异常响应中，数据域返回异常码。

异常码的列表：

MODBUS 异常码		
代码	名称	含义
01	非法功能	接收到的功能码是不可允许的操作。
02	非法数据地址	接收到的数据地址是不可允许的地址。
03	非法数据值	询问中包括的值是不可允许的值
04	从站设备故障	模块功能执行失败

1 4、参考资料

详细的协议请参考以下资料：

《Modbus 应用协议》（国标 GB/T19582.1-2008）

《Modbus 协议在串行链路上的实现指南》（国标 GB/T19582.2-2008）

三、输入寄存器

1 、概述

输入寄存器以字（16bit）为单位，只能读取。通过读取输入寄存器可以获得模块的重量数据和状态。

地址	十六进制	字数	功能	默认值	备注
0	0x0000	2	毛重		
2	0x0002	2	皮重		
4	0x0004	2	净重		
6	0x0006	1	状态标志 1		
7	0x0007	1	状态标志 2		
8	0x0008	2	锁定毛量		
10	0x000A	2	锁定净重		
12	0x000C	2	毛重内码		
14	0x000E	2	皮重内码		
16	0x0010	2	净重内码		
18	0x0012	2	累计净重		
20	0x0014	1	累计次数		
21	0x0015	1	模块型号		
22	0x0016	1	模块硬件版本		
23	0x0017	1	模块软件版本		
24	0x0018	10	设备标识字		
34	0x0022	1	软件校验和		
35	0x0023	2	用户标定系数值		定点 6 位小数
37	0x0025	2	厂商重力加速度值		定点 6 位小数
39	0x0027	2	用户重力加速度值		定点 6 位小数
41	0x0029	2	模拟传感器输出（uV）		

2、毛重（0-1）、皮重（2-3）和净重（4-5）

毛重、皮重和净重为模块当前的对应“重量数值”。

“重量数值”为对应重量不包括小数点的数值（小数点位数需要从状态标志 1 获取）。例如毛重为 21kg，小数点为 2 位，那么毛重的重量数据值就是 2100；净重为 123kg，小数点为 3 位，那么净重的重量数值就是 123000。

“重量数值”为 32bit 补码占两个字，高字在前。

注 3-2-1（两个字补码的表示举例）：

数值	高字	低字
123456	0x0001	0xE240
32768	0x0000	0x8000
32767	0x0000	0x7FFF(32767)
12345	0x0000	0x3039(12345)
1	0x0000	0x0000(1)
0	0x0000	0x0000(0)
-1	0xFFFF	0xFFFF(-1)
-12345	0xFFFF	0xCFC7(-12345)
-32768	0xFFFF	0x8000(-32768)
-32769	0xFFFF	0x7FFF
-123456	0xFFFE	0x1DC0

当-32768≤数值≤32767 时，可以忽略高字。

3、状态标志 1（6）

状态标志 1

bit15~12	bit11	bit10~8	bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0
单位	0	小数点	I/O	锁定	异常	开机	超载	去皮	稳定	零位

bit0：零位

当毛重小于等于 0.25 个分度值时，零位标志位为 1。

bit1：稳定

重量变化小于判稳范围 1.6 秒后，稳定标志位为 1。

bit2：去皮

当皮重不为 0 时，皮重标志位为 1。

bit3：超载

当毛重大于满量程加 9 个分度值时，超载标志位为 1。当超载标志为 1 时，主机必须停止显示重量，并做出相应的报警提醒用户，避免压坏传感器。

bit4：开机

模块上电会执行开机置零（如果设置开机置零有效的话），此时开机标志位为 1，输出的重量信息无效（固定为 0）。

bit5：异常

传感器模拟部分损坏或者模块 AD 转换异常，此时异常标志位为 1，输出的重量无效。

bit6：锁定

锁定重量和锁定净重处于锁定状态时，锁定标志位为 1。

bit7：I/O

开关量输入状态，1 表明开关量输入闭合，0 表明开关量输入断开。

bit10~8：小数点

小数点位数占 3 位（0~4），表明所有重量（如毛重、净重、分度值等）相关信息的小数点位数。

小数点位数一般在标定时确定，使用时为一个常量。在主机软件设计时，可以使用固定的小数点位数进行处理，而不需要实时处理。

bit15~12:单位

重量单位占 4 位（0~8），表明所有重量相关信息的单位，对应关系见下表。

数值	0	1	2	3	4	5	6	7	8
含义	kg	g	t	lb	N	kN	mL	L	kL

重量单位一般在标定时确定，使用时为一个常量。在主机软件设计时，可以使用固定的重量单位进行处理，而不需要实时处理。

4、状态标志 2（7）

bit15~bit8	bit7~4	bit3~bit1	bit0
模块温度值	分度值代号	备用(固定为 0)	累计

bit0：累计

当前累计重量不为 0，累计位为 1。

bit7~bit4：分度值代号

分度值代号用 4 位（0~11）表示分度值的不考虑小数点值，见下表：

代号	0	1	2	3	4	5	6	7	8	9	10	11
分度值	1	2	5	10	20	50	100	200	500	1000	2000	5000

bit15~bit8：模块温度值

模块温度值为 1 字节补码（-128~127）= 实际模块温度×2 - 20。（即精度为 0.5℃）

5、锁定毛重（8-9）和锁定净重（10-11）

锁定毛重和锁定净重配合锁定功能使用。

在不锁定时锁定毛重和毛重、锁定净重和净重是完全一致的;锁定时锁定毛重和锁定净重就不会发生变化了，一直到锁定解除。

在使用锁定功能的系统中，用户可以读取锁定毛重或锁定净重作为显示重量。

6、毛重内码（12-13）、皮重内码（14-15）和净重内码（16-17）

内码提供了高分辨率的重量信息，主要在标定调试、生产检定中使用。

内码 = 重量 × 20 ÷ 分度值

例如：净重=6.000kg，分度值=0.002kg，那么净重内码= 6.000 * 20÷0.002=60000

7、累计净重（18-19）和累计次数（20）

累计净重和累计次数配合累计和累清功能使用。

当执行累计功能时，累计净重 = 自身 + 当前净重，累计次数加 1。

当执行累清功能是，累计净重和累计次数都清零。

8、模块型号（21）

本模块固定为 0x4433。其中 0x44 为“D”的 ASC 码，0x33 为“3”的 ASC 码。

9、硬件版本号（22）

硬件版本号主要用于软硬件匹配，同一个硬件版本号的软件都是兼容的，反之则不能使用。在 IAP 升级软件时，首先要核对硬件版本号是否匹配，匹配的话才能升级。

10、软件版本号（23）

软件版本号主要用于判断软件功能。比如说某个功能需要大于某个软件版本才具备。

11、设备标识字（24-33）

设备标识字共 10 个字 20 个字节（高字节在前），它是一个字符串，用于显示用户自己定义的标识（如自己的厂商名称、型号、广告信息等等）。

12、软件校验和（34）

用于校验软件的合法性，有没有被异常改写。一般而言不同的硬件版本号和软件版本号都会对应自己固定的软件校验和。

13、用户标定系数值（35-36）

4 字节补码，固定六位小数，只用于读取。

如用户标定系数=1.123，那么厂商重力加速度=1.123×1000000=1123000。

高字（35）= 0x0011 ，低字（36）= 0x22B8

14、厂商重力加速度值（37-38）和用户重力加速度值（39-40）

4 字节补码，固定六位小数，只用于读取。

15、模拟传感器输出（41-42）

模拟传感器输出为模拟传感器输出信号的电压值，单位为 uV。

相对精度约为±0.1%，绝对精度为±2%。

四、离散量

1、概述

离散量以位为单位，只能读取。其中的内容和输入寄存器状态字 1（6）的含义相同。

地址	十六进制	位数	功能	默认值	备注
0	0x0000	1	零点		
1	0x0001	1	稳定		
2	0x0002	1	去皮		
3	0x0003	1	超载		
4	0x0004	1	开机		
5	0x0005	1	AD 异常		
6	0x0006	1	锁定		
7	0x0007	1	开关量输入		

2、零位（0）

当毛重小于等于 0.25 个分度值时，零位标志位为 1。

3、稳定（1）

重量变化小于判稳范围 1.6 秒后，稳定标志位为 1。

4、去皮（2）

当皮重不为 0 时，皮重标志位为 1。

5、超载（3）

当毛重大于满量程加 9 个分度值时，超载标志位为 1。当超载标志为 1 时，主机必须停止显示重量，并做出相应的报警提醒用户，避免压坏传感器。

6、开机（4）

模块上电会执行开机置零（如果设置开机置零有效的话），此时开机标志位为 1，输出的重量信息无效（固定为 0）。

7、异常（5）

传感器模拟部分损坏或模块 AD 转换异常，异常标志位为 1，输出的重量信息无效。

8、锁定（6）

锁定重量和锁定净重处于锁定状态时，锁定标志位为 1。

9、I/0（7）

开关量输入状态，1 表明开关量输入闭合，0 表明开关量输入断开。

五、线圈

1、概述

线圈以位为单位，可读写。除开关量输出外，都是用于写入 1 来触发某个功能。

地址	十六进制	位数	功能	默认值	备注
0	0x0000	1	开关量输出		
8	0x0008	1	去皮		读取固定为 0
9	0x0009	1	置零		读取固定为 0
10	0x000A	1	重量锁定		读取固定为 0
11	0x000B	1	解除锁定		读取固定为 0
12	0x000C	1	累计		读取固定为 0
13	0x000D	1	累清		读取固定为 0

2、开关量输出（0）

对应模块的开关量输出信号，当模块的开关量输入输出口配置为开关量输出时有效。

写入 1，开关量输出闭合，输出电压约为 0V；

写入 0，开关量输出断开，输出电压约为 5V（未连接外部系统时）。

3、去皮（8）

写入 1，模块执行去皮功能。需要重量稳定，否则返回错误应答。

当毛重大于 0 时，把当前毛重作为皮重，皮重标志位变为 1，净重输出为 0。

当毛重小于等于 0 时，把皮重设置为 0，皮重标志位变为 0。

4、置零（9）

写入 1，模块执行去皮功能。

当毛重处于手动置零范围内时，把当前毛重作为零点，毛重输出为零，否则应答错误。

5、重量锁定（10）

写入 1，锁定标志位变为 1，锁定毛重和锁定净重锁定不再变化。

6、解除锁定（11）

写入 1，锁定标志位变为 0，锁定毛重和毛重、锁定净重和净重同步。

7、累计（12）

写入 1，执行累计功能。

当重量稳定，且净重大于零时，累计净重 = 自身 + 当前净重，累计次数加一。

否则应答错误。

8、累清（13）

写入 1，执行累清功能。累计净重和累计次数清零。

六、保持寄存器

1 、概述

保持以字（16bit）为单位，可读写。主要用于配置参数，和执行某些复杂的指令。

地址	十六进制	字数	功能	默认值	备注
0	0x0000	2	皮重		用于预置皮重
1	0x0001	1	用户寄存器 1		
2	0x0002	1	用户寄存器 2		
3	0x0003	1	用户寄存器 3		
4	0x0004	1	用户寄存器 4		
48	0x0030	1	通讯模式		
49	0x0031	1	波特率		
50	0x0032	1	校验方式		
51	0x0033	1	通讯地址		
52	0x0034	1	通讯发送延时		
53	0x0035	1	I/O 模式		
54	0x0036	2	I/O 参数 1		
56	0x0038	2	I/O 参数 2		
58	0x003A	2	I/O 参数 3		
60	0x003C	2	I/O 参数 4		
62	0x003E	2	备用		
64	0x0040	1	标率组别		只能读
65	0x0041	1	分度值	厂商值	
66	0x0042	1	小数点	厂商值	
67	0x0043	1	重量单位	厂商值	
68	0x0044	2	满量程值	厂商值	
70	0x0046	1	滤波模式	0	
71	0x0047	1	滤波强度	3	
72	0x0048	1	判稳范围	3	
73	0x0049	1	开机置零范围（FULL%）	20	
74	0x004A	1	手动置零范围（FULL%）	4	

75	0x004B	1	零点跟踪模式	0	
76	0x004C	1	零点跟踪范围	0	
77	0x004D	1	零点跟踪速度	1	
78	0x004E	1	补偿模式	0	
80	0x0050	2	用户标定零点		
82	0x0052	2	用户标定系数		
84	0x0054	2	用户重力加速度		
86	0x0056	2	分度值切换点 1		
88	0x0058	2	分度值切换点 2		
4096	0x1000	32	指令读写区		

2、皮重（0-1）

皮重为 4 字节补码（占 2 个字）。

和输入寄存器的皮重不同，保持寄存器的皮重可读可写。

写入皮重，相当于预置皮重的功能。

写入值大于等于 0，写入值作为皮重，去皮标志位变为 1。

写入值小于等于 0，模块退出去皮状态，去皮标志位变为 0。

注意：这里写入的值，也是不考虑小数点的数值。如皮重为 12.30kg，小数点为 2 位，则写入值=1230。

3、用户寄存器 1、2、3、4（1、2、3、4）

用户寄存器为通用寄存器，具备断电保存的能力，用以保存用户自己的参数。

4、通讯模式（48）

值	通信模式
0	Modbus-RTU
1	自定义协议
2	小黄狗定制协议

注意：通讯参数变化后，需要给模块重新上电才会生效。

5、波特率（49）

值	波特率	备注
0	600	Modbus-RTU 不推荐使用
1	1200	Modbus-RTU 不推荐使用
2	2400	Modbus-RTU 不推荐使用
3	4800	
4	9600	
5	14400	
6	19200	
7	28800	
8	38400	
9	57600	
10	76800	
11	96000	
12	115200	
13	160000	
14	200000	
15	250000	

注意：通讯参数变化后，需要给模块重新上电才会生效。

6、校验方式（50）

值	校验方式	停止位	备注
0	无校验	2 位	
1	奇校验	1 位	
2	偶校验	1 位	
3	零校验	1 位	
4	无校验	1 位	Modbus-RTU 不推荐使用

注意：通讯参数变化后，需要给模块重新上电才会生效。

7、通讯地址（51）

用于模块寻址（0~255）。在 Modbus-RTU 模式时，只能设置为 1~247。

8、通讯发送延时（52）

通讯发送延时（0~255），单位为 ms。用于在模块发送应答数据前加入额外的等待时间。

在使用 RS485 半双工模式通讯时，主机发送完数据后，某些“RS232 转 RS485 工具”需要等待一定的时间才能切换为接收状态。如果模块应答的太快可能会造成数据丢失或接收不可靠。此时可以适当的设置通讯发送延时来避免这个错误。

9、I/O 模式（53）

值	I/O 模式	备注
0	不使用	
1	开关量输入	
2	开关量输出	

10、用户参 1、2、3、4（54-55、56-57、58-59、60-61）

用户参数为 4 字节补码（占 2 个字），是通用参数，根据不同的工作模式会有不同的含义。暂时还没有用到这些参数。

11、标率组别（64）

模块支持 3 组标率。标率组别指明了当前使用的是那组参数（后续哪些寄存器）。

标率组别为只读寄存器，写入无效。如果要切换标率组别，则需要使用指令模式（见下节）。

12、分度值（65）

分度值可设置为 1、2、5、10、20、50、100、200、500、1000、2000、5000，它决定了模块最小变化的数值。

它和小数点共同决定了模块的分辨率。例如单位为 kg，如果分度值为 2，小数点为 2 位，它的分辨率就是 0.02kg；如果分度值为 20，小数点为 3 位，它的分辨率就是 0.020kg。两者的分辨率的数值是一样的，但是后者多了一个无效零。

13、小数点（66）

所有重量信息（如毛重、皮重、净重、分度值、满量程等）的小数点位数。

14、单位（67）

重量的单位，含义见下表。

数值	0	1	2	3	4	5	6	7	8
含义	kg	g	t	lb	N	kN	mL	L	kL

它只用于显示，和重量数值无关，模块不会自动根据重量单位的而改变重量数值。

比如说当前输出为 1000kg，切换到 lb 它的输出就变为 1000lb。如果需要实现 kg/lb 的转换，则需要同时修改用户标定系数或者使用标率切换功能。

15、满量程（68-69）

满量程为 4 字节补码（占 2 个字），决定了毛重的最大值，当重量大于满量程 +9×分度值时状态字中的超载标志就置为 1，此时用户系统应做出相应的报警提示，避免压坏秤体。

16、滤波模式（70）

现在固定为 0——滑动滤波。

将来会陆续增加动物秤、动态模式等别的滤波模式。

１７、滤波强度（71）

滤波强度（0~3）数值越大重量数值越稳定，但是加载反应越慢。

１８、判稳范围（72）

数值	0	1	2	3	4	5	6	7	8	9
范围	1d	2d	3d	4d	5d	6d	7d	8d	9d	10d

“d”为分度值。当重量的变化小于设定的判稳范围 1.6 秒后，状态字中稳定位的置位，3.2 秒后进入闪变点抑制状态（如果开了闪变点抑制功能的话）。

１９、开机置零范围（73）

开机置零范围（0~100），单位为 FULL%，例如满量程为 500kg，开机置零范围为 20，那么开机置零的范围值就是±100kg，模块重新上电后，如果毛重小于±100kg，模块就把该重量作为零点，输出毛重为 0。

２０、手动置零范围（74）

手动置零范围（0~100），单位为 FULL%，它影响着置零命令能否成功。如果当前毛重（相对于开机零点）小于手动置零范围值时，置零命令有效，模块把当前毛重作为零点，输出毛重为 0。

２１、零点跟踪模式（75）

有两种零点跟踪模式：毛重为零跟踪和净重为零跟踪。用于修正零点的缓慢变化。

２２、零点跟踪范围（76）

数值	0	1	2	3	4	5	6	7	8	9
范围	不跟踪	0.5d	1d	1.5d	2d	2.5d	3d	3.5d	4d	4.5d

“d”为分度值。当重量处于零点跟踪范围，模块逐步修正零点，使输出趋向于零。

２３、零点跟踪速度（77）

零点跟踪速度（0~3），数值越大零点修正的速度就越快。零点跟踪速度快，可能会造成加载小分量，被跟掉一定的数值，使重量稍微偏轻。

２４、补偿模式（78）

补偿模式参数，用于配置蠕变补偿（bit0）和闪变点抑制（bit1）两个功能是否使用。

蠕变补偿（bit0=0：不使用；bit0=1：使用）

传感器长时间加载，会引起零点的缓慢变化，当加载的重量处于零点跟踪范围外时，这种变化就不能自动的被修正。蠕变补偿就是用于修正这种缓慢变化的。

闪变点抑制（bit0=0：不使用；bit0=1：使用）

当重量处于四舍五入的临界点时，输出可能会上下跳动一个分度，称之为闪变点。如果不希望看到这种跳动，则可以选择闪变点抑制。

２５、用户标定零点（80-81）

用户标定零点为 4 字节补码（占 2 个字），是用户标定零点（见下一小节）时确定的零点内码，参与计算用户标定系数。在平时工作时，仅作为判断开机置零范围的基准。

２６、用户标定系数（82-83）

用户标定系数为 4 字节浮点数（占 2 个字），是用户标定加载点（见下一小节）时得到的系数，用于计算重量的大小，公式如下：

重量 = 原始 AD 码 × 传感器厂商标定系数 × 用户标定系数

它的默认值是 1，即只使用厂商标定系数。例如把用户标定系数变为 1.1，那么输出的重量数值比传感器出厂时要大 10%、

２７、用户重力加速度（84-85）

用户重力加速度为 4 字节浮点数（占 2 个字），是用户重力加速度修正（见下一小节）时输入的重力加速度。不参数重量的计算，只起一个记录的作用，单独修改它是没有意义的。

２８、分度值切换点 1（86-87）和分度值切换点 2（88-89）

分度值切换点为 4 字节补码（占 2 个字）。它是不考虑小数点的数值，例如分度值切换点 1 为 43.20kg，小数点位数为 2 位，那么分度值切换点 1 的值就是 4320。

模块提供两个分度值切换点，当毛重小于分度值切换点 1 时，分度值向下切一档；当毛重大于等于分度值切换点 2 时，分度值向上切一档。当他们等于 0 时，表明不使用。

七、高级命令

1、命令概述

为了实现复杂的功能，我们在保持寄存器中划出一块空间用于高级命令的交互，地址是4096~4127(0x1000~0x101F)，我们称之为“命令交互空间”。

所有高级命令都是通过读写同一块保持寄存器的地址空间（“命令交互空间”）来实现。命令的发起是写“命令交互空间”（必须用连续写寄存器指令一次写入）；命令的应答是读“命令交互空间”。

为了简化 Modbus 的编程，对于立即执行的命令，可以通过 Modbus 的应答来判断命令是否的执行成功。执行成功的话就不需要读取“命令交互空间”来获取应答了。失败的话，如果需要知道错误的原因，才需要读取“命令交互空间”来获取应答。

对于需要一段执行时间的命令，只要符合命令的发起条件，模块总是应答正确的。此时命令是否执行成功，必须通过读取“命令交互空间”来获取应答。

命令格式：

0x00	0x01	0x02	0x03	0x04~0x7D
起始字	命令字	命令字校验	数据字长	命令数据
0x4743	见命令字列表	等于命令字取反	0~28	可选

应答格式：

0x00	0x01	0x02	0x03	0x04~0x7D
起始字	命令字	应答状态	数据字长	应答数据
0x4743	见命令字列表	见应答状态列表	0~28	可选

应答状态列表：

应答状态值	含义
0xFFFF	无命令
0x0000	命令执行中
0x1000	命令执行成功
0x20xx	命令执行失败，xx 为失败标识 xx=0x00：命令错误 xx=0x01：命令参数错误 xx=0x02：未打开写保护开关 xx=0x03：未关闭写保护开关 xx=0x04：重量不稳定 xx=0x05：不符合操作条件 xx=0x80：硬件故障 xx=0x81：参数保存失败

2、高级命令列表

命令字	含义
0x0000	空命令
0x0101	置零命令
0x0202	去皮命令
0x0303	重量锁定命令
0x0404	累计命令
0x3030	零点标定
0x3131	加载点标定
0x3232	重力加速度修正
0x3333	标定系数设定
0x3434	用户参数写保护存储
0x4040	标率切换
0x4141	初始化当前标率

3、空命令（0x0000）

作用：握手命令，无执行功能

命令：

起始字	命令字	命令字校验	数据字长
0x4743	0x0000	0xFFFF	0

4、置零命令（0x0101）

作用：置零

命令：

起始字	命令字	命令字校验	数据字长
0x4743	0x0101	0xFEFE	2

数据 1	数据 2
0x00：不判稳定 0x01：判稳定	**0x00**：保存关机零点 0x01：不保存关机零点

这里的置零命令，比写线圈执行的置零命令有更多的参数选择。写线圈执行的置零相当于高级置零命令使用“不判稳定”和“保存关机零点”参数。

5、去皮命令（0x0202）

作用：去皮

命令：

起始字	命令字	命令字校验	数据字长	数据
0x4743	0x0202	0xFDFD	1	**0x00**：不判稳定 0x01：判稳定

6、重量锁定命令（0x0303）

作用：重量锁定

命令：

起始字	命令字	命令字校验	数据字长	数据
0x4743	0x0303	0xFCFC	1	**0x00**：锁定状态取反 0x01：锁定 0x02：解除锁定

7、累计命令（0x0404）

作用：重量累计

命令：

起始字	命令字	命令字校验	数据字长
0x4743	0x0404	0xFBFB	2

数据 0	数据 1
0x00：累计 0x01: 累清	**0x00**：不判稳定 0x01：判稳定

8、零点标定（0x3030）

作用：标定零点，该命令修改用户零点参数

命令：

起始字	命令字	命令字校验	数据字长
0x4743	0x3030	0xCFCF	1

数据
0：实时响应 1~100，采样 1~100 次取均值

如果参数选择取 1~100 次均值的话，模块需要一定的执行时间，Modbus 立即应答正确，然后再去执行，用户需要反复读取“命令交互空间”获取应答状态，来判断执行结果。

9、加载点标定（0x3131）

作用：标定加载点，该命令修改用户标定系数和用户标定零点

命令：

起始字	命令字	命令字校验	数据字长	数据
0x4743	0x3131	0xCECE	3	目标重量

数据
0：实时响应 1~100，采样 1~100 次取均值

如果参数选择取 1~100 次均值的话，模块需要一定的执行时间，Modbus 立即应答正确，然后再去执行，用户需要反复读取“命令交互空间”获取应答状态，来判断执行结果。

１０、重力加速度修正（0x3232）

作用：重力加速度修正

命令：

起始字	命令字	命令字校验	数据字长	数据
0x4743	0x3232	0xCDCD	2	本地重力加速度值

注：重力加速度值 = 重力加速度×1000000（即定点六位小数）

１１、标定系数设定（0x3333）

作用：标定系数设定

命令：

起始字	命令字	命令字校验	数据字长	数据
0x4743	0x3333	0xCCCC	2	标定系数值

注：标定系数值 = 标定系数×1000000（即定点六位小数）

１２、用户参数写保护存储（0x3434）

作用：无功能，清除上次的应答

命令：

起始字	命令字	命令字校验	数据字长
0x4743	0x3434	0xCBCB	0

１３、标率组别切换（0x4040）

作用：切换标率组别

命令：

起始字	命令字	命令字校验	数据字长	数据 0
0x4743	0x4040	0xBFBF	2	（0~2）标率组别

数据 1
0：使用新标率作为当前标率
1：当前标率不变（用于标率复制）

１４、初始化当前标率（0x4141）

作用：初始化当前标率

命令：

起始字	命令字	命令字校验	数据字长
0x4743	0x4141	0xBEBE	0

1 5 、高级命令举例

在上面的命令描述里面都是描述的往“命令交互空间”（保持寄存器 4096 开始的地址）写入数据的内容，不包含 Modbus 协议的信息。下面我们以置零命令为例，说明一下完整发送的内容。置零命令如下：

起始字	命令字	命令字校验	数据字长
0x4743	0x0101	0xFEFE	2

数据 1	数据 2
0x00：不判稳定 0x01：判稳定	**0x00**：保存关机零点 0x01：不保存关机零点

假设通讯地址是 1，不判稳定，需要保存关机零点，那么我们实际发送的数据帧是：

0x01	0x10	0x10	0x00	0x00	0x06	0x0C	0x47	0x43	0x01	0x01
地址	功能码	保持寄存器地址		寄存器数量		字节数	起始字		命令字	

0xFE	0xFE	0x00	0x02	0x00	0x00	0x00	0x00	0x1D	0xF1
命令字校验		数据字长		数据 1		数据 2		CRC 校验	

1 6 、变长命令

某些带参数的命令中的参数带有默认值（一般为 0x0000），那么在发送命令的时候可以省略这些参数，使用其默认值。在命令描述中参数的默认值，我们以“**粗体字**”表示。

需要注意的是省略的参数只能从最后一个参数开始往前省略，比如有一条命令有 3 个参数依次为 A、B、C，那么我们可以省略 C 或 BC 或 ABC，但是不可以省略 A 或 B 或 AB 或 AC 这样的组合。另外如果一个参数为多字参数（两个字及以上的参数），那么要省略就省略全部字，不能只省略其中的部分字。

变长命令的启用，既可以简化命令的发送，又使得将来升级给命令增加参数时，更容易和老系统兼容。

上一个小节的置零命令，使用的都是默认参数，我们可以简化发送为：

0x01	0x10	0x10	0x00	0x00	0x06	0x0C	0x47	0x43	0x01	0x01
地址	功能码	保持寄存器地址		寄存器数量		字节数	起始字		命令字	

0xFE	0xFE	0x00	0x00	0xD3	0xF1
命令字校验		数据字长		CRC 校验	