

项目		参数					
1	通讯接口	接口	RS485、异步串行通讯				
2		波特率	300、600、1200、2400、4800、9600、19200、38400、57600、76800、115200			默认：9600	
3		数据格式	1起始位、8数据位、无校验、1停止位(或 2停止位)				
4		物理连接	根据传感器输出接口的电气定义：电压正、电源负、RS485+、RS485-；有不同的接插件、直出线或其他方式。				
5	通讯协议	协议	Modbus-RTU				
6		设备地址号	1~247； 为广播地址为：0； 设备出厂默认地址号为：1； 注：多个设备组网使用前需要把设备地址修改为一组连续的地址，便于主机分别巡检传感器数据。				
7		功能码	03(读保持寄存器)、04(读输入寄存器)、05(写单个线圈)、06(写单个保持寄存器)				
8		保持寄存器	可读写，在此应用中，用于保存传感器（或变送器）的设置参数、校准参数、以及历史异常记录等； 最大支持254个寄存器(1个16bit的寄存器占用2个字节，即：508个字节)				
9		输入寄存器	只读，在此应用中，用于主机查询传感器的检测数据、中间计算数据、调试信息、运行状态、突发异常等；根据具体应用，将定义不同寄存器值的物理意义。				
10		线圈	可读写，一般用于开关量的操作、使能或禁用。 在此应用中，加入了一些类似于开关量操作的功能性命令，比如：保存参数、复位MCU、清零、恢复默认参数、切换校准模式…等。				
11	主机发送命令帧格式	域	从设备地址 (或广播地址:0)	功能码	寄存器地址	写寄存器：值 读寄存器：个数	CRC16
		字节数	1	1	2	2	2
12	主机接收消息帧格式	域	从设备地址	功能码	数据域的字节数	数据	CRC16
		字节数	1	1	1	2~16	2

保持寄存器列表					
地址	定义	数据类型	数值设定范围	说明	出厂值
0000H	设备通讯地址	16位无符号整数	1~247	从机地址编号	1
0001H	通讯协议	16位无符号整数	0	Modbus-RTU 从机模式	0
			1	(无效, 预留选项)	
			2	(无效, 预留选项)	
0002H	通讯波特率	16位无符号整数	0~10	0(300bps)1(600bps)2(1200bps)3(2400bps)4(4800bps) 5(9600bps)6(19200bps)7(38400bps)8(57600bps) 9(76800bps)10(115200bps)	5
0003H	数据位	16位无符号整数	0~2	0:7位; 1:8位; 2:9位	1
0004H	停止位	16位无符号整数	0~1	0: 1位; 1: 2位	0
0005H	校验位	16位无符号整数	0~2	0(无校验)1(奇校验)2(偶校验)	0
0006H	自动发送	16位无符号整数	0~1	0-禁止; 1-开启	0
0007H	自动发送周期	16位无符号整数	10~65535	每隔一个时间段,自动发送显示数据10~65535毫秒	200
0008H	用户单位选择(低16位)	32位无符号整数	0~2	0 --- 校准后初始计量单位 1、2 --- 用户设置单位1、2	0
0009H	用户单位选择(高16位)				
000AH	用户零点1(低16位)	32位有符号整数	±2147483647	显示值= (测量值 - 零点偏置) * 比例系数/10000 用于用户切换显示值的计量单位, 通过用户单位选择计量单位1、2	0
000BH	用户零点1(高16位)				
000CH	用户零点2(低16位)	32位有符号整数	±2147483647		0
000DH	用户零点2(高16位)				
000EH	用户系数1 (x/1w) (低16位)	32位有符号整数	±2147483647	显示值= (测量值 - 零点偏置) * 比例系数/10000 用于用户切换显示值的计量单位, 通过用户单位选择计量单位1、2	10000
000FH	用户系数1 (x/1w) (高16位)				
0010H	用户系数2 (x/1w) (低16位)	32位有符号整数	±2147483647		10000
0011H	用户系数2 (x/1w) (高16位)				

保持寄存器列表

地址	定义	数据类型	数值设定范围	说明	出厂值
0012H	清零方式	16位无符号整数	0~3	0 禁用 1 普通方式：发送清零指令启动清零,再发送一次恢复。 2 触发方式：当测量值小于一定值（清零阈值设置的值）时，每次发送清零指令都清零。 3 自动零点跟踪：当测量值小于一定值（清零阈值设置的值）时，将启动零点跟踪。	1
0014H	清零阈值(低16位)	32位有符号整数	0~9999	0 ± x 范围内允许清零	20
0015H	清零阈值(高16位)				
0016H	自动清零步长值(低16位)	32位有符号整数	0~9999	当满足自动清零条件时，清零幅度限制在一定范围内。	5
0017H	自动清零步长值(高16位)				
0018H	配置参数，略……请勿修改!!!				
0019H					
001AH					
001BH					
0034H					
0035H					
0036H					
0037H					

输入寄存器列表

地址	定义	数据类型	说明
0000H	运行状态寄存器(低8位)	8位无符号整数	备用
0000H	参数状态寄存器(高8位)	8位无符号整数	备用
0001H	预留	16位无符号整数	备用
0002H	ADC转换原始值(低16位)	32位有符号整数	±2147483647
0003H	ADC转换原始值(高16位)		
0004H	MCU内部温度(低16位)	32位有符号整数	±2147483647
0005H	MCU内部温度(高16位)		
0006H	数字量输出值(低16位)	32位有符号整数	±2147483647
0007H	数字量输出值(高16位)		
0008H	模拟量(DAC)输出值(低16位)	32位有符号整数	±2147483647
0009H	模拟量(DAC)输出值(高16位)		
000AH~0014H, 细节省略, 存放校准后、单位换算后的数值和软件、硬件的版本号等信息。			

线圈(操作命令)寄存器列表

地址	定义	说明
0000H	无	预留
0001H	读保持寄存器的CRC16操作	<p>当使用功能码(03H读保持寄存器)连续发送命令来读取多个寄存器时，MCU内部将对这些寄存器进行CRC16计算。</p> <p>往此地址的线圈写入0000H，将复位此CRC16计算模块</p> <p>往此地址的线圈写入FF00H，将向主机返回 上几次读取寄存器计算的CRC16值</p> <p>此功能主要用于：读取大量的寄存器数据后，对这些数据进行再次校验，以保证数据的正确性</p>
0002H	无	预留
0003H	零点校准命令	写入FF00H，传感器将把当前测量的物理量当作零位值，并对后续的测量数据进行新的零位校准
0004H	满载校准命令	写入FF00H，传感器将把当前测量的物理量当作满载值，并对后续的测量数据进行新的灵敏度校准
0005H	清除线性表数据	写入FF00H，传感器将把清除线性表数据。
0006H	保存所有参数	写入FF00H，传感器将把当前RAM中的参数，保持至FLASH中。重新上电后，将从FLASH中加载参数
0007H	复位MCU	写入FF00H，将立即复位MCU
0008H	恢复默认值	写入FF00H，将把所有参数恢复至产品组装、调试前的默认值。
001BH	清零指令	写入FF00H，对应保持寄存器清零方式
0009H~00FFH 预留，以后增加功能用		

举例											
1	更改设备地址编号	Modbus主机发送命令	域	从设备地址	功能码	寄存器地址		数据		CRC16	
		值	00H	06H	00H	00H	00H	05H	48H	18H	
	Modbus主机收到返回的信息	域	从设备地址	功能码	寄存器地址		数据		CRC16		
		值	02H	06H	00H	00H	00H	05H	49H	FAH	
<p>说明：此操作用于修改从机地址编号，便于多台从设备组网使用。在未知从设备地址编号情况下，主机可以使用广播地址(00H)对单个从设备进行更改地址编号。且从设备在返回的所有信息中将带上自己的实际地址编号，包括所有的读写寄存器操作。</p> <p>注意1：当使用广播地址(00H)时，其他从设备必须断电，或者与此网络断开，否则将会把所有的传感器改成同一地址编号。</p> <p>注意2：所有修改的保持寄存器的参数只存储在RAM中，断电后将会丢失，需要立即发送【保存参数】命令，以永久保存修改后数据。</p> <p>注意3：新地址不会立即生效，需要复位MCU 或者重新上电后才会启用新地址编号。</p>											
2	保持参数	Modbus主机发送命令	域	从设备地址	功能码	寄存器地址		数据		CRC16	
		值	02H	05H	00H	06H	FFH	00H	6CH	08H	
	Modbus主机收到返回的信息	域	从设备地址	功能码	寄存器地址		数据		CRC16		
		值	02H	05H	00H	06H	FFH	00H	6CH	08H	
<p>说明：此操作将把RAM中所有的参数 保存至 FLASH中，MCU复位或重新上电后，将把FLASH中保存的参数加载至RAM中使用。任何对参数修改后，都需要使用此命令来永久保存，否则断电或复位后，参数将丢失。</p>											
3	读取传感器的值	Modbus主机发送命令	域	从设备地址	功能码	寄存器地址		寄存器个数		CRC16	
		值	05H	04H	00H	06H	00H	02H	90H	4EH	
	Modbus主机收到返回的信息	域	从设备地址	功能码	字节个数	数据 1、数据 2、数据 3、数据 4				CRC16	
		值	05H	04H	04H	寄存器06H (高8位)	寄存器06H (低8位)	寄存器07H (高8位)	寄存器07H (低8位)	CRC低8位	CRC高8位
<p>说明：读取输入寄存器中的0006H和0007H的值，即经过校准后的传感器输出的数字量。也可以分2次分别读取寄存器0006H 和 0007H</p> <p>传感器输出的数字量 = 寄存器(0007H) * 10000H + 寄存器(0006H) = 数据1*100H + 数据2 + 数据3*100000H + 数据4*10000H</p>											

举例

4 读取传感器的值后，怎样转换成实际数据？ -- C语言

域	从设备地址	功能码	寄存器地址		寄存器个数		CRC16		Modbus主机 发送命令	
值	05H	04H	00H	06H	00H	02H	90H	4EH		
域	从设备地址	功能码	字节个数	数据 1、数据 2、数据 3、数据 4				CRC16		Modbus主机 收到返回的信息
值	05H	04H	04H	寄存器 06H (高8位)	寄存器 06H (低8位)	寄存器 07H (高8位)	寄存器 07H (低8位)	CRC 低8位	CRC 高8位	

```
uint32_t GetValFromRxBuf(uint8_t* COM_rxBuf, uint32_t len, int32_t* DVal)
{
//输入参数 COM_rxBuf 为串口接收的数据所存放的缓存首指针
//输入参数 len 为串口接收的数据字节个数
//输出参数 DVal 为输出获得的值

    int32_t V; // 然后定义一个变量，用于获取实际数据
    uint8_t* p = (uint8_t*)(&V); //以及定义一个指针，用于强制转换用的：
/*
当收到一串消息帧时，首先对缓存中的数据进行CRC16校验，如果校验通过的话，表示接受到的数据没有问题。才能进行下一步操作
CRC16校验计算 略...
*/
    if (CRC16_Check(COM_rxBuf, len) == 0)
    {
        *p++ = COM_rxBuf[4];
        *p++ = COM_rxBuf[3];
        *p++ = COM_rxBuf[6];
        *p++ = COM_rxBuf[5];
        *DVal = V;
        return BL_TRUE; // 返回成功标志
    }
    else
    { return BL_FALSE; } // 返回失败标志
}
```

举例

5 读取传感器的值后，怎样转换成实际数据？ --- Delphi语言

域	从设备地址	功能码	寄存器地址		寄存器个数			CRC16		Modbus主机 发送命令
值	05H	04H	00H	06H	00H	02H		90H	4EH	
域	从设备地址	功能码	字节个数	数据 1、数据 2、数据 3、数据 4				CRC16		Modbus主机 收到返回的信息
值	05H	04H	04H	寄存器 06H (高8位)	寄存器 06H (低8位)	寄存器 07H (高8位)	寄存器 07H (低8位)	CRC 低8位	CRC 高8位	

```
function GetValFromRxBuf(COM_rxBuf: Pointer; len:Integer; var DVal:Integer):Boolean;
var
    V: Integer; // 然后定义一个变量，用于获取实际数据
    pval, psrc: ^Byte; //以及定义一个指针，用于强制转换用的
begin
    //输入参数 COM_rxBuf 为串口接收的数据所存放的缓存首指针
    //输入参数 len 为串口接收的数据字节个数
    //输出参数 DVal 为输出获得的值
    V := 0; pval := @V; // 初始化参数
    { 当收到一串消息帧时，首先对缓存中的数据进行CRC16校验，如果校验通过的话，
    表示接受到的数据没有问题。才能进行下一步操作 CRC16校验计算 略... }
    if (CRC16_Check(COM_rxBuf, len) = True) then
    begin
        psrc := COM_rxBuf; Inc(psrc, 4); pval^ := psrc^; Inc(pval);
        psrc := COM_rxBuf; Inc(psrc, 3); pval^ := psrc^; Inc(pval);
        psrc := COM_rxBuf; Inc(psrc, 6); pval^ := psrc^; Inc(pval);
        psrc := COM_rxBuf; Inc(psrc, 5); pval^ := psrc^;

        DVal := V;
        result := True; // 返回成功标志
    end
    else begin
        result := False; // 返回失败标志
    end;
end;
```