



# FS/MF4000系列气体质量流量计 Modbus通讯协议 (VerA.2)

## 矽翔微机电系统有限公司（Siargo Ltd.）

3100 De La Cruz Boulevard, Suite 210, Santa Clara, CA 95054	+1-408-9690368
上海市闵行区万源路2158号泓毅大厦410室	021-54265998
成都市高新区科园南二路1号4栋	028-85139315
北京市朝阳区安立路101号名人广场32F	010-58296058
广东省深圳市南山区前海路3101号5栋15D	0755-22673459

[www.siargo.com](http://www.siargo.com); [www.siargo.com.cn](http://www.siargo.com.cn)

目 录

<b>1</b>	<b>适用范围.....</b>	<b>1</b>
<b>2</b>	<b>Modbus 协议.....</b>	<b>1</b>
2.1	协议简介 .....	1
2.2	通讯参数 .....	2
2.3	Modbus 消息帧 .....	3
2.3.1	RTU 模式 .....	3
2.3.2	设备地址域.....	3
2.3.3	功能代码域.....	4
2.3.4	数据域.....	5
2.3.5	错误检测域.....	5
2.4	错误检测方法 .....	6
2.4.1	奇偶校验 .....	6
2.4.2	CRC 检测 .....	6
<b>3</b>	<b>功能描述.....</b>	<b>10</b>
3.1	Modbus 功能格式 .....	10
3.1.1	数字值的表达 .....	10
3.1.2	消息帧中的数据地址 .....	10
3.1.3	消息帧中的域内容 .....	10
3.1.4	字节个数域的使用 .....	11
3.1.5	异常响应 .....	11
3.2	读保持寄存器（功能码=03） .....	13
3.2.1	功能描述 .....	13
3.2.2	消息帧格式 .....	13
3.3	预置单寄存器（功能码=06） .....	14
3.3.1	功能描述 .....	14
3.3.2	消息帧格式 .....	14
3.4	错误诊断查询（功能码=08） .....	15
3.4.1	功能描述 .....	15
3.4.2	消息帧格式 .....	15
3.5	预置多寄存器（功能码=16） .....	16
3.5.1	功能描述 .....	16
3.5.2	查询消息帧 .....	16
3.5.3	响应消息帧 .....	16
<b>附录 A</b>	<b>参数说明.....</b>	<b>17</b>
A.1	参数限制 .....	17
A.2	参数概述 .....	17
A.3	参数详述 .....	17
A.3.1	约定 .....	17
A.3.2	参数详细说明 .....	18

# 1 适用范围

本协议适用于具有 RS-485 通讯接口的 FS/MF4000 气体质量流量计，其软件版本为 V4.2 及以上。

## 2 Modbus 协议

### 2.1 协议简介

该协议定义了 Modbus 总线 Master（主设备）与 Slave（从设备）之间的通讯报文格式，对于主设备来说，Modbus 协议是联系上位机（如 PC、PLC、HMI 等）的接口，而且所有的通讯都是“透明的”。控制器通信使用主-从技术，即仅有一个设备（主设备）能初始化传输（查询），其它设备（从设备）根据主设备查询提供的数据作出相应反应。

主设备可单独和从设备通信，也能以广播方式和所有从设备通信（仅限于点对点通讯）。如果单独通信，从设备返回一帧消息作为回应；如果是广播方式查询的，则不作任何回应。Modbus 协议建立了主设备查询的格式：设备（或广播）地址、功能代码、所有要发送的数据和错误检测域。

从设备回应消息帧也由 Modbus 协议构成，包括确认要产生动作的域、任何要返回的数据和错误检测域。如果在消息接收过程中发生错误，或从设备不能执行其命令，从设备将建立一错误消息帧并把它作为回应发送出去。

Modbus 协议是工业最常用的 PLC 通信协议，包含 RTU（远程终端单元）及 ASCII（美国标准信息交换代码）两种模式，两种模式的报文各字段解释是相同的，他们之间最大的差别是其执行错误核对的方式与字符的格式不同。

控制器能设置为两种传输模式（ASCII 或 RTU）中的任何一种在标准的 Modbus 网络进行通信，用户选择想要的模式，包括串口通信参数（波特率、奇偶校验方式等），在配置每个控制器的时候，在一个 Modbus 网络上的所有设备都必须选择相同的传输模式和串口参数。所选的 ASCII 或 RTU 方式仅适用于标准的 Modbus 网络，它定义了在这些网络上连续传输的消息段的每一位，以及决定怎样将信息打包成消息域和如何解码。

## 2.2 通讯参数

Modbus 使用 RS-232、RS-485 或 RS-422 接口作为硬件载体。MF 系列工业用气体质量流量计采用以下通讯参数：

表 2.1 Modbus 通讯参数

通讯参数	协议格式
	RTU
通讯速率	4800, 9600bps, 19200, 38400
起始位	1 位
数据位	8 位
停止位	1 位
奇偶校验	无
最大缓冲区长度（数据）	20
最大节点数	247

每个字符的发送和接收格式如下（数据的最低有效位 D0 在前）：

RTU 模式（10 位）：

ST	D0	D1	D2	D3	D4	D5	D6	D7	SP
起始位	8 位数据位								停止位

## 2.3 Modbus 消息帧

传输设备可以将 Modbus 消息转为有起点和终点的帧，这就允许接收的设备在消息起始处开始工作，读地址分配信息，判断哪一个设备被选中（广播方式则传给所有设备），判知何时消息已完成。接收设备也能侦测到部分消息是否有错误，如果有则通过响应消息帧将错误信息返回给主控制器。

### 2.3.1 RTU 模式

当控制器设为在 Modbus 网络上以 RTU 模式通信时，在消息中的每个 8Bit 字节（包含两个 4Bits 的 16 进制字符）作为一个 8Bit 字符发送，数据校验方式采用 CRC（循环冗余校验），这样可以最大限度地利用每个数据位的空间，提高通信效率。RTU 模式的主要优点是：在波特率相同的条件下，可比 ASCII 码方式传送更多的数据。

使用 RTU 模式，消息发送至少要以 3.5 个字符时间（这里的字符时间指的是表 1 中所列的字符时间，不是单纯的以 7 位或 8 位数据构成的字符传输的时间，下同）的停顿间隔开始。在网络波特率下多样的字符时间，这是最容易实现的（如下图的 T1-T2-T3-T4 所示，相当于四个字节的传输延迟）。传输的第一个域是设备地址。可以使用的传输字符是十六进制的 0~9，A~F。网络设备不断侦测网络总线，包括停顿间隔时间在内。当接收到第一个域（地址域）以后，每个设备都进行解码以判断是否是发往自己的。在最后一个传输字符之后，一个至少 3.5 个字符时间的停顿确定了消息的结束。一个新的消息可在此停顿后开始。

Modbus RTU 通信过程中，必须把整个消息帧作为一个连续的数据流转输。如果在帧完成之前有超过 1.5 个字符时间的停顿时间，接收设备将刷新不完整的消息并假定下一字节是一个新消息的地址域（当前消息帧传输中断）。同样地，如果一个新消息在小于 3.5 个字符时间内接着前一个消息开始，接收的设备将认为它是前一消息的延续。这将导致一个错误，因为在最后的 CRC 域的值不可能是正确的。一个典型的消息帧如下所示：

起始位	设备地址	功能代码	数据	CRC 校验	结束符
T1-T2-T3-T4	8Bit	8Bit	n 个 8Bit ( $20 \geq n \geq 0$ )	16Bit	T1-T2-T3-T4

### 2.3.2 设备地址域

消息帧的设备地址域包含 8Bit（RTU）。可能的从设备地址是 0~247（十进制），单个设备的地址范围是 1~247，地址 0 用作广播地址，以使所有的从设备都能认识。主设备通过将要联络的从设备的地址放入消息中的地址域来选通从设备。当从设备发送回应消息时，它把自己的地址放入回应的地址域中，以便主设备知道是哪一个设备作出回应。

### 2.3.3 功能代码域

消息帧中的功能代码域包含了 8Bits (RTU)，可能的代码范围是十进制的 1~255。当然，有些代码是适用于所有控制器的，有些则只应用于某种特定的控制器，还有些保留以备后用。

当消息从主设备发往从设备时，功能代码域将告之从设备需要执行哪些行为。例如去读取输入的开关状态，读一组寄存器的数据内容，读从设备的诊断状态，允许调入、记录、校验在从设备中的程序等。

当从设备回应时，它使用功能代码域来指示是正常回应（无误）还是有某种错误发生（称作异议回应）。对正常回应，从设备仅回应相应的功能代码。对异议回应，从设备返回一等同于正常代码的代码，但最高位为逻辑 1（正常功能代码+0x80）。

例如：一从主设备发往从设备的消息要求读一组保持寄存器，将产生如下功能代码：

0 0 0 0 0 0 1 1 （十六进制 0x03）

对正常回应，从设备仅回应同样的功能代码。对异议回应，它返回：

1 0 0 0 0 0 1 1 （十六进制 0x83）

除功能代码因异议错误作了修改外，从设备将一独特的错误代码放到回应消息的数据域中，这能告诉主设备发生了什么错误。

主设备应用程序得到异议的回应后，典型的处理过程是重发消息，或者诊断发给从设备的消息并报告给操作员。

FS/MF4000 气体质量流量计所使用的功能代码是 Modbus 标准通讯协议所规定的功能代码的一个子集，主要包括表 2.2 所列的几种（功能码用十进制数表示，详细的说明见下一章的描述）。

表 2.2 MF 系列工业用气体质量流量计使用的 Modbus 功能码

功能码	名 称	数据类型	作 用
03	读保持寄存器	整型、字符型、状态字、浮点型	读取一个或多个连续的保持寄存器的值
06	预置单寄存器	整型、字符型、状态字、浮点型	把具体二进制值装入一个保持寄存器
08	错误诊断查询	整型	检查主设备与流量计之间的通讯是否正常
16	预置多寄存器	整型、字符型、状态字、浮点型	把具体二进制值装入多个连续的保持寄存器

## 2.3.4 数据域

数据域由两个十六进制数的集合构成，范围为 0x00~0xFF。根据不同的网络传输模式，这可以由一对 ASCII 字符组成或由一个 RTU 字符组成。

从主设备发给从设备消息的数据域包含附加的信息，即从设备必须用于执行由功能代码所定义的行为。这包括了像不连续的寄存器地址，要处理项的数目，域中实际数据字节数等。

例如，如果主设备需要从设备读取一组保持寄存器（功能代码 0x03），数据域指定了起始寄存器地址以及要读的寄存器数量。如果主设备写一组从设备的寄存器（功能代码 0x10），数据域则指明了要写的起始寄存器地址以及要写的寄存器数量，数据域的数据字节数，要写入寄存器的数据。

如果没有错误发生，从从设备返回的数据域包含请求的数据。如果有错误发生，此域包含一异议代码，主设备应用程序可以用来判断采取下一步行动。

在某种消息中数据域可以是不存在的（0 长度）。例如，主设备要求从设备回应通信事件记录（功能代码 0x0B），从设备不需任何附加的信息。

## 2.3.5 错误检测域

当选用 RTU 模式作字符帧时，错误检测域包含一个 16Bits 值（用两个 8Bit 的字符来实现）。错误检测域的内容是通过对消息内容进行 CRC（Cyclic Redundant Check，循环冗余校验）方法得出的。CRC 域附加在消息的最后，添加时先是低字节然后是高字节。故 CRC 的高位字节是发送消息帧的最后一个字节。

有关 CRC 的算法介绍，请参见下一节有关错误检测方法的讨论。

## 2.4 错误检测方法

标准的 Modbus 网络有两种错误检测方法：奇偶校验和帧检测（即错误检测域）。奇偶校验对每个字符都可用，帧检测（LRC 或 CRC）应用于整个消息。它们都是在消息发送前由主设备产生的，从设备在接收过程中检测每个字符和整个消息帧。

用户要给主设备配置一预先定义的超时时间间隔，这个时间间隔要足够长，以使任何从设备都能作出正常反应。如果从设备检测到一个传输错误，消息将不会接收，也不会向主设备作出回应。这样超时事件将触发主设备来处理错误。发往不存在的从设备的地址也会产生超时。

### 2.4.1 奇偶校验

用户可以配置控制器是奇校验（odd）、偶校验（even）还是无校验（none）。这将决定每个字符中的奇偶校验位以及停止位是如何设置的。

如果指定了奇校验或偶校验，“1”的位数将算到每个字符的位数中（ASCII 模式 7 个数据位，RTU 中 8 个数据位）。例如 RTU 字符帧中包含以下 8 个数据位：

1 1 0 0 0 1 0 1

整个“1”的数目是 4 个。如果使用了偶校验，帧的奇偶校验位将是 0，使得整个“1”的个数仍是 4 个（偶数）。如果使用了奇校验，帧的奇偶校验位将是 1，使得整个“1”的个数是 5 个（奇数）。

如果没有指定奇偶校验位，传输时就没有校验位，也不进行校验检测。代替它的一位附加的停止位将填充至要传输的字符帧中。

### 2.4.2 CRC 检测

使用 RTU 模式，消息帧包括了一个基于 CRC 方法的错误检测域。CRC 域是两个字节，包含一个 16 位的二进制值，它检测了整个消息帧的内容（不含 CRC 域本身）。它由传输设备计算后加入到消息中，接收设备重新计算收到消息的 CRC，并与接收到的 CRC 域中的值比较，如果两值不同，则有误。

CRC 是先调入一个值为全“1”的 16 位寄存器，然后调用一个过程来处理消息帧中连续的字节。只有每个字符中的 8Bit 字节数据参与 CRC 计算，而起始位、停止位以及奇偶校验位均不参与计算。

CRC 产生过程中，每个 8 位字符都单独和 16 位寄存器中的内容相或（OR），结果向最低有效位（LSB）方向移动，最高有效位（MSB）以 0 填充。LSB 被提取出来进行测试，



如果 LSB 为 1，寄存器单独和预置的固定值或（OR）一下；否则，则不作任何操作。整个过程要重复 8 次。在最后一位（第 8 位）完成后，下一个 8 位字节又单独和寄存器的当前值相或。最终寄存器中的值，是消息中所有的字节（不含 CRC 域本身）都参与计算之后的 CRC 值。

采用下面的方法可以计算 CRC 错误检测域的值：

- a) 初始化一个 16 位的寄存器 CRC，使其值为 0xFFFF（全为 1）；
- b) 对 CRC 寄存器的低 8 位与消息帧中的每个 8Bit 字节执行或（OR）操作，并把结果放回 CRC 寄存器中；
- c) 将 CRC 寄存器向右移动一位，最高有效位（MSB）补 0，然后准备对移出的最低有效位（LSB）进行测试；
- d) 如果 LSB 等于 0，则重复 c) 步操作；如果 LSB 等于 1，则对 CRC 寄存器与一个多项式的值 0xA001 执行或（OR）操作；
- e) 重复 c) 和 d) 步操作，直到 8 次移位操作完成。这时，一个完整的 8Bit 字节就处理完了；
- f) 重复 b) 到 e) 步操作处理消息帧中的下一个字节，直到消息帧中所有的字节处理完毕；
- g) 最终的 CRC 寄存器中的值就是 CRC 错误检测域的值。

以下是采用 C 语言编写的 CRC 生成示例（示例 2 为直接算法，适用于运算能力较强及存储空间有限的微处理器或微控制器，示例 3 为查表法，适用于运算能力较弱且存储空间有富余的微控制器）：

## 示例 2：基于直接算法的 CRC 的 C 语言实现

// 为简化计算定义组合结构

**typedef union**

{

    UINT16 val;

**struct**

    {

        UINT16 bit0 : 1;

        UINT16 bit1 : 1;

        UINT16 bit2 : 1;

        UINT16 bit3 : 1;

        UINT16 bit4 : 1;

        UINT16 bit5 : 1;

        UINT16 bit6 : 1;

        UINT16 bit7 : 1;

        UINT16 bit8 : 1;

        UINT16 bit9 : 1;

        UINT16 bit10 : 1;

        UINT16 bit11 : 1;

```

    UINT16 bit12: 1;
    UINT16 bit13: 1;
    UINT16 bit14: 1;
    UINT16 bit15: 1;
} bits;
} TCRCRegs;
TCRCRegs regs;

// pBuf: 要参与计算的消息缓冲区
// nDataLen: CRC 要处理的字节的数量（消息缓冲区长度）
UINT16 CRC1(UINT8 *pBuf, UINT16 nDataLen)
{
    int i;
    UINT8 j,nTest;

    regs.val = 0xFFFF;
    for (i = 0; i < nDataLen; i++)
    {
        regs.val ^= *pBuf++;
        for (j = 0; j < 8; j++)
        {
            nTest = (regs.bits.bit0) ? 0x01:0x00;
            regs.val >>= 1;
            if (nTest == 1)
                regs.val ^= 0xA001;
        }
    }
    return regs.val;
}

```

### 示例 3：基于查表法的 CRC 的 C 语言实现

```

// 为简化计算定义组合结构
typedef union
{
    UINT16 nCRC16;
    struct
    {
        UINT8 nCRCHi;
        UINT8 nCRCLo;
    }
} TCRCRegs;

// pBuf: 要参与计算的消息缓冲区
// nDataLen: CRC 要处理的字节的数量（消息缓冲区长度）
UINT16 CRC2(UINT8 *pBuf, UINT16 nDataLen)

```

```

{
    TCRCRegs nCRC;
    UINT16 nIndex;                                // CRC 查找表索引定义

    nCRC.nCRC16 = 0xFFFF;                        // CRC 初始化
    while (uDataLen--)                            // 遍历消息缓冲区
    {
        nIndex = nCRC.nCRCHi ^ *pBuf++;          // 计算 CRC
        nCRC.nCRCHi = nCRC.nCRCLo ^ CRCHi[nIndex];
        nCRC.nCRCLo = CRCLo[nIndex];
    }
    return (nCRC.nCRC16);
}

/* CRC 的高 8 位查找表 */
static UINT8 CRCHi[] =
{
    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, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41,
    0x01, 0xC0, 0x80, 0x41, 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
};

/* CRC 的低 8 位查找表 */
static INT8 CRCLo[] =
{
    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
};

```

## 3 功能描述

### 3.1 Modbus 功能格式

#### 3.1.1 数字值的表达

除非特指，否则本节所述的数字值（如地址、功能代码或数据）均以 10 进制表示，但在要发送或接收的消息帧中则以 16 进制表示。

#### 3.1.2 消息帧中的数据地址

所有 Modbus 消息帧中的数据地址都是相对于 0 的，数据序列的第一个数据总是从 0 开始。比如：

- a) 从设备中的“1 号线圈”在 Modbus 消息帧的数据地址域中的地址为 0x0000，“127 号线圈”的地址为 0x007E（即 126）；
- b) “40001 号保持寄存器”在 Modbus 消息帧的数据地址域中的地址为 0x0000，因为其功能代码已经指定了是对保持寄存器进行操作，因此“4xxxx”已暗指了保持寄存器。比如，保持寄存器 40108 的地址为 0x006B（即 107）。

#### 3.1.3 消息帧中的域内容

表 3.1 显示了一个 Modbus 查询消息的示例，表 3.2 是一个正常响应的示例。这两个示例均以 16 进制数据显示了域的内容，以及消息怎样以 ASCII 或 RTU 模式组织成消息帧。

表 3.1 查询消息帧格式

域名	实例	RTU
帧头		3.5T 周期超时
从设备地址	6	0000 0110
功能码	0x03	0000 0011
起始地址高（字节）	0x00	0000 0000
起始地址低（字节）	0x6b	0110 1011
寄存器数量高（字节）	0x00	0000 0000
寄存器数量低（字节）	0x03	0000 0011
校验和		crc
帧尾		3.5T 周期超时

表 3.2 响应消息帧格式

域 名	示 例	RTU
帧头		无
从设备地址	0x06	0000 0110
功能码	0x03	0000 0011

字节数	0x06	0000 0110
数据高（字节）	0x02	0000 0010
数据低（字节）	0x2B	0010 1011
数据高（字节）	0x00	0000 0000
数据低（字节）	0x00	0000 0000
数据高（字节）	0x00	0000 0000
数据低（字节）	0x63	0110 0011
校验和		CRC（16 位）
帧尾		无

主设备查询是一个读 06 号从设备的保持寄存器消息帧。该消息请求读取保持寄存器 40108 到 40110 中的数据，起始寄存器的地址为 0x006B（即 107）。

从设备响应同样的功能码，指示这是一个正常的响应。字节个数域指出响应消息帧中有多少个 8Bit 的数据将要返回给主设备。

### 3.1.4 字节个数域的使用

当构造一个响应缓冲区时，需要使用字节个数域来标识一帧消息中数据域的字节个数。该值仅指示数据域的字节个数而不含其它域（包括该域自身在内）。

### 3.1.5 异常响应

当主设备发送一帧查询消息至从设备时，主设备期望得到一帧正常的响应消息，但此时可能因为某种原因导致发生异常。主设备执行查询时可能引发以下四种事件之一：

- 如果从设备接收到查询消息且未发生通讯错误，则它能够正常处理查询并返回一个正常的响应消息；
- 如果因为通讯错误而导致从设备未接收到查询消息，则没有响应消息返回。主设备最终将以一个超时事件来处理这种情况；
- 如果从设备接收到了查询消息，但是它检测到有通讯错误发生（如奇偶校验错、LRC 或 CRC 错误等），则从设备不返回响应消息。主设备最终将以一个超时事件来处理这种情况；
- 如果从设备正确接收到了查询消息（没有发生通讯错误），但不能正常处理该消息（比如，请求要读取的输出继电器线圈或保持寄存器不存在），则从设备会返回一个异常响应信息，通知主设备发生错误的类型。

异常响应消息有两个域不同于正常响应：

#### a) 功能代码域

在正常响应消息中，从设备回显原始查询消息中的功能代码。所有功能代码的最高有效位 MSB 都为 0（其值都小于 0x80）。而在异常响应消息中，从设备将功能代码

的最高有效位 MSB 置为 1，这使得功能代码值都大于 0x80。

通过将功能代码的最高有效位置为 1，主设备的应用程序可以识别响应消息有异常发生。

b) 数据域

在正常响应消息中，从设备在其数据域中可能返回数据或统计信息。在异常响应消息中，从设备在数据域中返回一个标识引起错误发生的错误代码。

下表给出了流量计在 Modbus 通讯中可能产生的错误代码。

表 3.3 从设备异常响应时的错误代码

错误代码	名 称	含 义
1	ILLEGAL_FUNCTION	在收到的查询消息中的功能代码对于当前从设备而言是非法（不允许出现）的。
2	ILLEGAL_DATA_ADDRESS	在收到的查询消息中的数据地址对于当前从设备而言是非法（不允许出现）的。
3	ILLEGAL_DATA_VALUE	在收到的查询消息数据域中的值对于当前从设备而言是非法（不允许出现）的。
4	SLAVE_DEVICE_FAILURE	当从设备尝试执行请求的动作时发生了一个不可恢复的错误。
5	ACKNOWLEDGE	从设备已经收到了一个请求且正在处理当中，但该操作所需花费的时间较长。为了防止主设备发生超时错误，可以向主设备发送 ACKNOWLEDGE 响应。
6	SLAVE_DEVICE_BUSY	从设备正在处理一个耗时的程序命令，主设备应该在从设备不忙的时候再重新发送查询消息。

下表给出了一个由于写只读寄存器地址而引起 Modbus 异常响应的查询消息帧及异常响应帧示例。

表 3.4 异常响应帧格式示例

查询消息帧		异常响应消息帧	
消息帧组成	数据	消息帧组成	数据
从设备地址	0x11	从设备地址	0x11
功能码	0x06	功能码	0x86
寄存器地址高	0x00	错误码	0x02
寄存器地址低	0x00	校验和	0xC2
预置数据高	0x01		
预置数据低	0x03		
校验和	0xCA		0x64
	0xCB		
说 明	示例尝试修改 17 号流量计内部只读保持寄存器 40000 中的值，导致了异常响应（错误码 = ILLEGAL_DATA_ADDRESS = 0x02）		

## 3.2 读保持寄存器（功能码=03）

### 3.2.1 功能描述

读取流量计内部保持寄存器（以 4 打头）的值，读取的是一个或多个 16 位整数或者无符号整数。

数据起始地址及每次允许读取的寄存器个数请参见附录 B。

### 3.2.2 消息帧格式

查询消息帧（主设备发送）由设备地址域、功能码、寄存器地址域、数据域和校验和五部分组成。其中，功能码固定为 0x03，代表读保持寄存器功能；寄存器地址域指定本次操作所要读取的保持寄存器的首地址（寄存器地址从 0x0000 开始，寄存器 1~16 的地址为 0x00000~0x000F），由两个字节组成；数据域指定本次操作所要读取的保持寄存器个数，由两个字节组成。

响应消息帧（流量计发送）由设备地址域、功能码、字节数、数据域和校验和五部分组成。其中，功能码固定为 0x03，代表返回的是读保持寄存器功能响应；字节数指示返回的数据域所包含的数据字节个数；数据域由一系列连续的双字节数据组成，代表要读取的保持寄存器中的数据，每个寄存器中的数据用两个字节表示，寄存器地址按照由低到高的发送顺序递增。

示例（RTU 格式）如下：

表 3.5 帧格式示例

查询消息帧		响应消息帧	
消息帧组成	数据	消息帧组成	数据
从设备地址	0x11	从设备地址	0x11
功能码	0x03	功能码	0x03
起始地址高	0x00	字节数	0x06
起始地址低	0x6B	数据高（寄存器 40108）	0x02
读取点数高	0x00	数据低（寄存器 40108）	0x2B
读取点数低	0x03	数据高（寄存器 40109）	0x00
校验和	0x76  0x87	数据低（寄存器 40109）	0x00
		数据高（寄存器 40110）	0x00
		数据低（寄存器 40110）	0x64
		校验和	0xC8 0xBA
说 明	示例读取 17 号流量计保持寄存器 40108~40110 中的值		

寄存器 40108 的值用 0x02 和 0x2B 两个字节表示，即十进制的 555。寄存器 40109~40110 的值是 0x00、0x00 和 0x00、0x65，即十进制的 0 和 100。

### 3.3 预置单寄存器（功能码=06）

#### 3.3.1 功能描述

预置一个值到流量计的指定保持寄存器（以 4 打头）中。在广播模式下，该命令将同时使所有连接在一起的流量计的相同保持寄存器具有相同的预置值。

流量计保持寄存器的值也可能根据内部的控制流程而被流量计自身修改。

数据起始地址及数据长度请参见附录 B。

#### 3.3.2 消息帧格式

查询消息帧（主设备发送）和响应消息帧的格式完全相同，都由设备地址域、功能码、寄存器地址域、数据域和校验和五部分组成。其中，功能码固定为 0x06，代表预置单寄存器功能；寄存器地址域指定要预置的寄存器地址（寄存器的地址从 0x0000 开始，即寄存器 1 的地址为 0x0000），由两个字节组成；数据域指定要在指定寄存器地址处预置的数据，由两个字节组成。

在非广播模式下，正常响应为查询消息帧的回显（即和查询消息帧内容完全相同），它在指定的寄存器被预置后返回；在广播模式下，流量计不会返回响应消息帧。

举例（RTU 格式），见下表。

表 3.6 帧格式示例

查询消息帧		响应消息帧	
消息帧组成	数据	消息帧组成	数据
从设备地址	0x11	从设备地址	0x11
功能码	0x06	功能码	0x06
寄存器地址高	0x00	寄存器地址高	0x00
寄存器地址低	0x01	寄存器地址低	0x01
预置数据高	0x00	预置数据高	0x00
预置数据低	0x03	预置数据低	0x03
校验和	0x9A	校验和	0x9A
	0x9B		0x9B
说 明	示例预置 17 号流量计的寄存器 40002 的值为 0x0003		



### 3.4 错误诊断查询（功能码=08）

#### 3.4.1 功能描述

利用 Modbus 协议提供的 08 号诊断功能，可以对上位机与流量计之间的通讯进行测试以检查其通讯功能是否正常。该功能不支持广播工作模式。

错误诊断查询功能在查询消息帧中使用一个双字节的子功能代码域来定义要执行的测试操作的类型。本版协议仅支持标准诊断子功能代码中的一种，即返回查询数据（子功能代码：00），其余子功能代码均保留将来使用，用户若使用将返回一个错误响应。数据域中的数据为一个任意的双字节数据，

如果通讯正常，则流量计（从设备）将把主设备的查询消息帧不做修改地作为响应帧回传给主设备。

#### 3.4.2 消息帧格式

查询消息帧（主设备发送）和响应消息帧的格式完全相同，都由从设备地址、功能码、子功能代码、诊断数据和校验和五部分组成。其中，功能码固定为 0x08，代表设备诊断功能；子功能代码固定为 0x0000，由两个字节组成，代表回显查询数据子功能；诊断数据由两个字节组成，可以为任意数值。

下面是一个应用诊断功能的示例：

表 3.7 帧格式示例

查询消息帧		响应消息帧	
消息帧组成	数据	消息帧组成	数据
从设备地址	0x11	从设备地址	0x11
功能码	0x08	功能码	0x08
子功能高	0x00	子功能高	0x00
子功能低	0x00	子功能低	0x00
数据高	0xA5	数据高	0xA5
数据低	0x37	数据低	0x37
校验和	0x36	校验和	0x36
	0xF9		0xF9
说 明		示例检查 17 号流量计与上位机的通讯是否正常（正常时回显）	

### 3.5 预置多寄存器（功能码=16）

#### 3.5.1 功能描述

预置多个值到流量计的多个保持寄存器（以 4 打头）中。在广播模式下，该命令将同时使所有连接在一起的流量计的相同保持寄存器具有相同的预置值。

流量计保持寄存器的值也可能根据内部的控制流程而被流量计自身修改。

数据起始地址及数据长度请参见附录 B。

#### 3.5.2 查询消息帧

查询消息帧（主设备发送）指定要预置的寄存器及要预置的数据。寄存器的地址从 0 开始，即寄存器 1 的地址为 0。

请求的预置值在查询消息帧的数据域中指定。每个寄存器的值由 2 个字节表示（共 16 位）。

#### 3.5.3 响应消息帧

正常的响应将返回从设备地址、功能码、开始地址和被预置的寄存器个数，它在所有寄存器被强置后返回。

举例（RTU 格式），见下表。

表 3.8 帧格式示例

查询消息帧		响应消息帧	
消息帧组成	数据	消息帧组成	数据
从设备地址	0x11	从设备地址	0x11
功能码	0x10	功能码	0x10
开始地址高	0x00	起始地址高	0x00
开始地址低	0x01	起始地址低	0x01
寄存器数高	0x00	寄存器数高	0x00
寄存器数低	0x02	寄存器数低	0x02
字节数	0x04	校验和	0x12
预置数据高	0x00		
预置数据低	0x0A		
预置数据高	0x01		0x98
预置数据低	0x02		
校验和	0xC6		
	0xF0		
说 明	示例预置 17 号流量计以 40002 开始的寄存器的值为 0x000A 和 0x0102		

## 附录 A 参数说明

### A.1 参数限制

表 A.1 给出了主设备在主查询中可以请求或发送的数据最大个数，或者由从设备（流量计）在响应消息中返回的数据最大个数，所有数据均用 10 进制数表示。

表 A.1 最大查询/响应参数个数

功能码	功能描述	查询消息帧	响应消息帧	备注
03	读保持寄存器	8 个寄存器	8 个寄存器	
06	预置单寄存器	1 个寄存器	1 个寄存器	
08	错误诊断查询	N/A	N/A	
16	预置多寄存器	8 个寄存器	8 个寄存器	

### A.2 参数概述

参见 A.3.2 说明。

### A.3 参数详述

#### A.3.1 约定

数据类型：

数据类型	描 述	长度（字节数）
UINT8	unsigned char, 8 bits	1
UINT16	unsigned int, 16 bits	2
UINT32	unsigned long int, 32 bits	4
STRING	ASCII 码组成的字符串	
组合型	由 UINT8、UINT16 或 UINT32 组合构成	

其它：

- 1) 参数的读写属性指明了参数是否可读写，其中，R 代表只读，W 代表只写，RW 代表可读可写；
- 2) (H) 代表取指定地址处一个字变量的高 8 位，(L) 代表取指定地址处一个字变量的低 8 位，比如，0x0000 地址中存储的变量值等于 0x1234，则 0x0000(H)=0x12，而 0x0000(L)=0x34。

### A.3.2 参数详细说明

本机地址	0x0001	修改	允许
		读取	允许
参数描述	本机设备地址		
数据类型	UINT16		
数据表示	从 1 至 247 的任意数值。0 为广播地址，不可将本机地址设置为 0。		
当前流量	0x0002~0x0003	修改	不允许
		读取	允许
参数描述	当前的气体流量。		
数据类型	UINT32		
数据表示	<p>0x0002~0x0003 构成一个 UINT16 无符号整型数，代表当前气体流量；            流量 <math>V = \text{value}(0x0002) * 65536 + \text{value}(0x0003)</math>  <math>V</math> 的数值为 流量（基本单位）*1000            例：目前单位为 L/min 流量数值为 20.34，则通过 modbus 获得的数值为 <math>20.34 * 1000 = 20340</math></p>		
当前总流量	0x0004~0x0006	修改	允许
		读取	允许
参数描述	自最近一次总量清零至当前，仪表记录到的累积总流量		
数据类型	UINT32 + UINT16		
数据表示	<p><math>V1 = \text{value}(0x0004) * 65536 + \text{value}(0x0005)</math> ;  <math>V2 = \text{value}(0x0006)</math>            总量 <math>V = V1 * 1000 + V2</math>;            其中 <math>V1</math> 为 32 位，代表当前总流量的整数部分；<math>V2</math> 为 16 位，代表当前总流量的小数部分。            例：当前总量为 3452.245 方，则通过 modbus 获取的值为 <math>3452 * 1000 + 245 = 3452245</math></p>		
波特率	0x0015	修改	允许
		读取	允许
参数描述	仪表当前波特率对应的索引		
数据类型	UINT16		
数据表示	<p>仪表当前波特率对应的索引关系：            0: 4800, 1: 9600, 2: 19200, 3: 38400            例：当仪表当前波特率为 38400 时，则通过 modbus 协议获取的值为 3</p>		
GDCF	0x0016	修改	允许*
		读取	允许
参数描述	流量的一次修正系数		

数据类型	UINT16		
数据表示	例：如一次修正系数 1000 时，则通过 modbus 协议获取的值为 1000		
响应时间	0x0017	修改	允许*
		读取	允许
参数描述	保存传感器响应时间的索引		
数据类型	UINT16		
数据表示	0:10ms, 1:20ms, 2:50ms, 3:100ms, 4:200ms, 5:500ms, 6:1000ms 例：如响应时间是 10ms 时，则通过 modbus 协议获取的值为 0		
滤波深度	0x0018	修改	允许*
		读取	允许
参数描述	滑动平均滤波的深度（滤波缓冲区长度）		
数据类型	UINT16		
数据表示	例：如滤波深度为 8 时，则通过 modbus 协议获取的值为 8		
仪表版本号	0x0019	修改	不允许
		读取	允许
参数描述	软件程序的版本号		
数据类型	HEX		
数据表示	例：如产品软件版本为 V4201 则通过 modbus 协议获取的值为 0x4001		
仪表序列号	0x001A~0x001F	修改	不允许
		读取	允许
参数描述	产品序列号（长度 = 12 字节）		
数据类型	STRING		
数据表示	例：如产品序列号为**C2E01023**时，则通过 modbus 协议获取的值为**C2E01023**		
设置零点内码值	0x0020	修改	不允许
		读取	允许
参数描述	零流量时的内码值		
数据类型	UINT16		
数据表示	例：如零点内码值为 32765 时，则通过 modbus 协议获得的值为 32765		
仪表最小流量	0x0021~0x0022	修改	不允许
		读取	允许
参数描述	仪表的最小输出流量（扩大 1000 倍）		
数据类型	UINT32		
数据表示	例：如仪表的最小输出流量为 0SLPM 时，则通过 modbus 协议获得的值为 0		
仪表最大流量	0x0023~0x0024	修改	不允许
		读取	允许

参数描述	仪表的最大输出流量（扩大 1000 倍）		
数据类型	UINT32		
数据表示	例：如仪表的最大输出流量为 5SLPM 时，则通过 modbus 协议获得的值为 5000		
仪表最小输出电压 (mV)	0x0025	修改	不允许
		读取	允许
参数描述	仪表的最小输出电压（mV）		
数据类型	UINT16		
数据表示	例：如最小输出电压为 500mV 时，则通过 modbus 协议获得的值为 500		
仪表最大输出电压 (mV)	0x0026	修改	不允许
		读取	允许
参数描述	仪表的最大输出电压（mV）		
数据类型	UINT16		
数据表示	例：如最大输出电压为 500mV 时，则通过 modbus 协议获得的值为 500		
自动校零	0x0027	修改	允许
		读取	不允许
参数描述	本命令强制仪表执行一次自动校零操作。需要注意的是，执行此操作前应确保传感器管道中的气流处于静止状态。		
数据类型	指定数据 0xAA55		
数据表示	例：向寄存器中写入指定的 0xAA55 即可完成自动校零		
写保护寄存器	0x0014	修改	允许
		读取	不允许
参数描述	有些寄存器的写入是受到保护的，防止误写对仪表参数造成错误，要修改这些参数是需要先操作写保护寄存器，一次有效，下一次修改时候仍需要再次写入，以上修改有星号标识的寄存器需要操作写保护寄存器		
数据类型	指定数据 0xAA55		
数据表示	例：修改 GDCF 时先在写保护寄存器中写入 0xAA55 才会修改 GDCF 成功		