

振动送料控制器 SDVC

Modbus 协议（标准版）

型号：

SDVC40XS

日期：2022-3-3

南京创优科技有限责任公司

目录

第一章、[控制器Modbus通讯协议](#)

1.1 [通信协议概述](#)

第二章、[Modbus数据链路层](#)

- 2.1 [Modbus主站/从站协议原理](#)
- 2.2 [Modbus从站（子节点）状态图解](#)
- 2.3 [Modbus地址规则](#)
- 2.4 [Modbus帧描述](#)
- 2.5 [Modbus数据传输方式](#)

第三章、[应用层协议](#)

- 3.1 [协议描述](#)
- 3.2 [数据编码传输方式](#)
- 3.3 [Modbus事务处理流程](#)

第四章、[功能码总述](#)

- 4.1 [功能码分类](#)
- 4.2 [功能码描述](#)
 - 4.2.1 [读控制器可设定状态位命令（0x01）](#)
 - 4.2.1.1 [控制器线圈比特位状态表](#)
 - 4.2.2 [写单个控制器可设定状态位命令（0x05）](#)
 - 4.2.3 [写多个控制器可设定状态位命令（0x0F）](#)
 - 4.2.4 [读控制器运行状态位命令（0x02）](#)
 - 4.2.4.1 [控制器运行状态位总表](#)
 - 4.2.5 [读控制器运行状态参数值命令（0x04）](#)
 - 4.2.5.1 [控制器输入数据总表](#)
 - 4.2.6 [读控制器可调参数命令（0x03）](#)
 - 4.2.6.1 [控制器可调参数对应地址表](#)
 - 4.2.7 [写单个控制器可调参数命令（0x06）](#)
 - 4.2.8 [写多个控制器可调参数命令（0x10）](#)

第一章、[控制器Modbus通信协议](#)

1.1 通信协议概述

根据标准Modbus通信协议该文档描述了本公司振动送料控制器（以下简称控制器）所使用的Modbus协议，包括串行链路协议和应用层协议。

Modbus串行链路协议是一个主/从协议。该协议位于OSI模型的第二层。

位于OSI模型第7层上的应用层报文传输协议，它提供了在不同类型总线或网络上的客户机和服务器之间的通信。

本控制器物理层采用RS485协议，异步通讯模式，波特率为300bps~115200bps可调，并做为服务器（子节点）在总线上通信。

本手册所提及的子节点/服务器，都可以认为是挂接在总线上的某一个控制器，主节点/客户机，都可以认为是具体的一个主动发送通信数据的设备。

第二章、Modbus数据链路层

2.1 Modbus主站/从站协议原理

Modbus 串行链路协议是一个主/从协议。在同一时刻，只有一个主节点连接于总线，一个或多个子节点(子节点地址范围为1到31)连接于同一个串行总线。Modbus 通信总是由主节点发起。子节点在没有收到来自主节点的请求时，从不会发送数据。子节点之间从不会互相通信。主节点在同一时刻只会发起一个 Modbus事务处理。

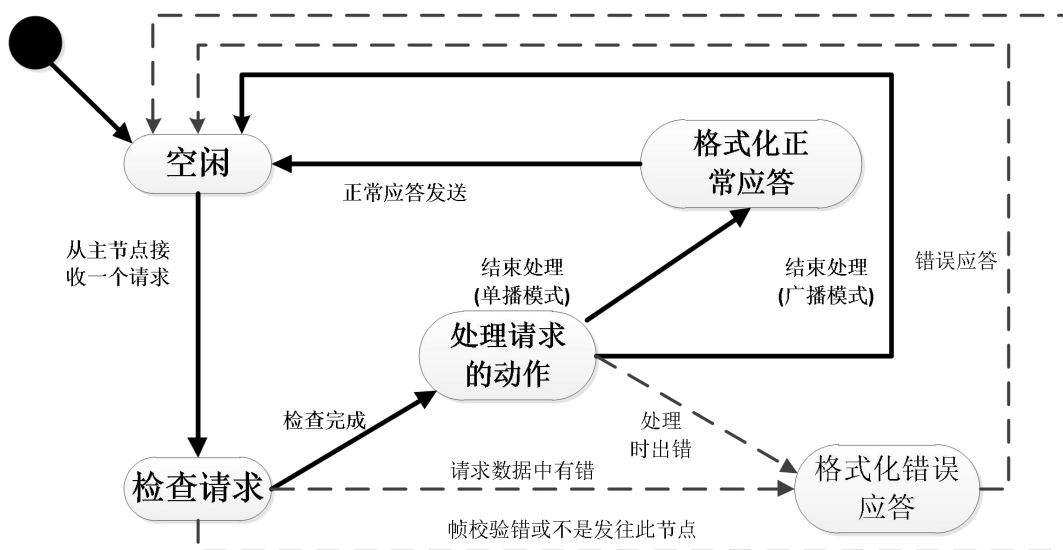
主节点以两种模式对子节点发出 Modbus 请求：

- a) 在 单播模式下，主节点以特定地址访问某个子节点，子节点接到并处理完请求后，子节点向主节点返回一个报文(‘应答’)。
- b) 在 广播模式下，主节点向所有的子节点发送请求。
对于主节点广播的请求没有应答返回。广播请求一般用于写命令。所有设备必须接受广播模式的写功能。地址 0是专门用于表示广播数据的。

2.2 Modbus从站（子节点）状态图解

(实线表示正常处理过程，虚线表示通信错误的处理过程)

下图描述了子节点的状态特征：



子节点状态图

说明：

- a) 状态“空闲”表示子节点没有接收到请求。这是控制器上电后的初始状态。
- b) 当收到一个请求时，子节点在处理请求中要求的动作前先检验报文包。可能的错误有：非法地址，无该功能码，非法数据…… 当检测到错误时，必须向主节点发送应答，指明何种错误。
- c) 当要求的动作完成后，单播报文要求必须格式化一个应答并发往主节点。
- d) 如果子节点在接收到的帧中检测到错误，则没有响应返回到主节点。

2.3 Modbus地址规则

- 0 为广播地址。所有子节点必须识别广播地址。
- 1-31 子节点地址，子节点必须有一个在总线上唯一的地址。
- 主节点没有地址。

2.4 Modbus帧描述

MODBUS 协议定义了一个与基础通信层无关的简单协议数据单元（PDU）。在PDU上附加地址域和校验域即构成应用数据单元（ADU）。在应用数据单元（ADU）上引入一些附加域，MODBUS 协议就能够映射在其他特定总线或网络上。



通用MODBUS帧

- 地址域 只含有子节点地址（1-31）。
主节点通过将子节点的地址放到报文的地址域对子节点进行寻址。当子节点返回应答时，它将自己的地址放到应答报文的地址域以让主节点知道哪个子节点在应答。
- 功能码 指明子节点要执行的动作。功能码后面可跟表示含有请求和响应参数的数据域（数据域也可能没有）。
- 差错检验域 是对报文内容执行“冗余校验”的计算结果。本控制器采用ASCII和RTU两种传输模式，所以使用纵向冗余校验（LRC）和循环冗余校验（CRC）的计算方法。（参见后文如何计算LRC和CRC）

2.5 Modbus数据传输方式

本控制器Modbus数据传输方式，采用ASCII字符传输和RTU传输两种传输模式，在发送的报文中，每两个 ASCII 字符代表一个8位的十六进制数。

1. ASCII

例如：报文中的连续2个ASCII字符【0x35】和【0x42】代表的十六进制数为0x5B

十六进制数与 ASCII 字符的对应关系如下表所示

十六进制数	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
ASCII 字符	30	31	32	33	34	35	36	37	38	39	41	42	43	44	45	46

ASCII 字符传输模式：

每个ASCII 字符在传输时，由10个位组成，它的格式为：

名称	起始位	数据位							停止位	停止位
	Bit0	Bit0	Bit1	Bit2	Bit3	Bit4	Bit5	Bit6	Bit0	Bit0
位序号	1	2	3	4	5	6	7	8	9	10

由上表可以看出ASCII 传输模式中的每个字节由3个部分组成：

- a) 1个 起始位
- b) 7个数据位（首先发送最低有效位）
- c) 2个停止位

Modbus ASCII报文帧

典型的报文帧结构如下所示（十六进制表示）：

名称	起始	地址		功能		数据1				...	数据n (n≥0)				LRC		结束	
	:	高 字 符	低 字 符	高 字 符	低 字 符	高字 节		低字 节		...	高字节		低字节		高 字 符	低 字 符	CR	LF
						高 字 符	低 字 符	高 字 符	低 字 符		高 字 符	低 字 符						
序号	1	2	3	4	5	6	7	8	9	...	10+ 2n	11+ 2n	12+ 2n	13+ 2n	14+ 2n	15+ 2n	16+ 2n	17+ 2n

报文帧的地址域含有两个字符。

在 ASCII 模式，报文用特殊的字符区分帧起始和帧结束。一个报文必须以一个‘冒号’（：）(ASCII十六进制 3A)起始，以‘回车-换行’(CR LF)对 (ASCII 十六进制 0D 和 0A)结束。

对于所有的域，允许传送的字符为十六进制 0 - 9, A - F (ASCII 编码)。设备连续的监视总线上的‘冒号’ 字符。当收到这个字符后，每个设备解码后续的字符一直到检测到‘回车-换行’对，表示帧结束。

报文中字符间的时间间隔可以达一秒。如果有更大的间隔，则接受设备认为发生了错误，不做响应。

其中数据n的长度范围为0到2*252字符

LRC校验

在 ASCII 模式，包含一个对全部报文内容执行的，基于纵向冗余校验（LRC - Longitudinal Redundancy Checking）算法的错误检验域。LRC 域检验不包括起始“冒号”和结尾 CR LF对的整个报文的内容。不管报文有无奇偶校验，均执行此检验。

纵向冗余校验(LRC)为一个字节，含有 8位二进制值。LRC由发送设备计算，并附加 LRC到报文。接收设备在接收报文时计算LRC，并将计算的结果与接收到的实际值相比较，如果两个值不相等，则结果为错。

LRC的计算，对报文中的所有连续8位字节相加，忽略任何进位，然后求出其二进制补码。LRC 为一个8位域，那么每个会导致值大于255的相加只是简单的将域的值在零”回绕”。因为没有第 9 位，进位被自动放弃。

生成一个 LRC 的过程为：

1. 不包括起始“冒号”和结束CR LF的报文中的所有字节相加到一个8位域，故此进位被丢弃。
2. 从0xFF(全1)减去域的最终值，产生二进制反码。
3. 加 1 产生二进制补码。

注：LRC计算并不是对转换为ASCII后的报文进行计算。

2. RTU

RTU传输模式：

RTU传输过程中通过十六位进制数据进行传输，由10个位组成，它的格式为：

名称	地址	功能	数据				CRC	
序号	1	2	3	4	5	6	7	8

报文帧的地址域含有一个十六进制。

在RTU模式下，报文的起始帧为从机地址，且没有起始位和结束位，在信息开始至少需要有3.5个字符时间的静止时间，依据使用的波特率可算出三个字符时间的长度。

各个区允许发送的字符均为16进制的0-9，A-F。

网络上的设备连续检测网络上的信息，包括静止时间。当接收第一个地址数据时，每台设备立即对它解码，以决定是否为自己的地址。发送完最后一个字符后，也有3.5个字符的静止时间，然后才能发送一个新的信息。

整个信息必须连续发送。如果在发送帧信息期间，出现大于1.5个字符的静止时间时，则设备刷新不完整的信息，并假设下一个地址数据。

CRC校验

在 RTU 模式，包含一个对全部报文内容执行的，基于循环冗余校验算法的错误检验域。

循环冗余校验（CRC）为两个字节。CRC由发送设备计算，并附加 CRC 到报文。接收设备在接收报文时计算CRC，并将计算的结果与接收到的实际值相比较，如果两个值不相等，则结果为错。

CRC的计算主要将需要发送的十六位进制数据转换成二进制数据后面补齐16个零再与多项式 $G(x) = X^{16} + X^{15} + X^2 + 1$ 的二进制比特串1100000000000101进行“模2除法”，将得到的二进制余数转换成十六进制，这时候的到的两个十六进制数据为发送数据的CRC

生成一个 CRC 的过程为：

1. 先将需要发送的数据转换成二进制，多项式 $G(x) = X^{16} + X^{15} + X^2 + 1$ 转换成二进制比特串
2. 在得到的二进制数据后面补齐16个零。
3. 将得到的二进制多项式 $G(x) = X^{16} + X^{15} + X^2 + 1$ 的二进制比特串进行“模2除法”。
4. 将得到的余数转换成十六进制

第三章、应用层协议

3.1 协议描述

MODBUS应用层是OSI模型第7层上的应用层报文传输协议，它提供了在不同类型总线或网络上的客户机和服务器之间的通信。



通用MODBUS帧

在客户机发起MODBUS事务处理时会创建一个MODBUS应用数据单元。其中功能码指示服务器执行哪种操作。MODBUS 协议规定了客户机启动一个请求时的帧格式。

用一个字节编码 MODBUS 数据单元的功能码域。有效的功能码范围是：1-255（十进制）（128-255 为异常响应保留）。当从客户机向服务器设备发送报文时，功能码通知服务器执行哪种操作。功能码0为无效功能码。

从客户机向服务器设备发送的报文数据域包括附加信息，服务器使用这个信息执行功能码定义的操作。这个域还包括离散项目和寄存器地址、处理的项目数量以及域中的实际数据字节数。

在某种请求中，数据域可以是不存在的（0长度），在此情况下服务器不需要任何附加信息。功能码仅说明操作。

如果在一个正确接收的MODBUS ADU中，不出现与请求MODBUS 功能有关的差错，那么服务器至客户机的响应数据域包括请求数据。如果出现与请求 MODBUS 功能有关的差错，那么域包括一个异常码，服务器应用能够使用这个域确定下一个执行的操作。

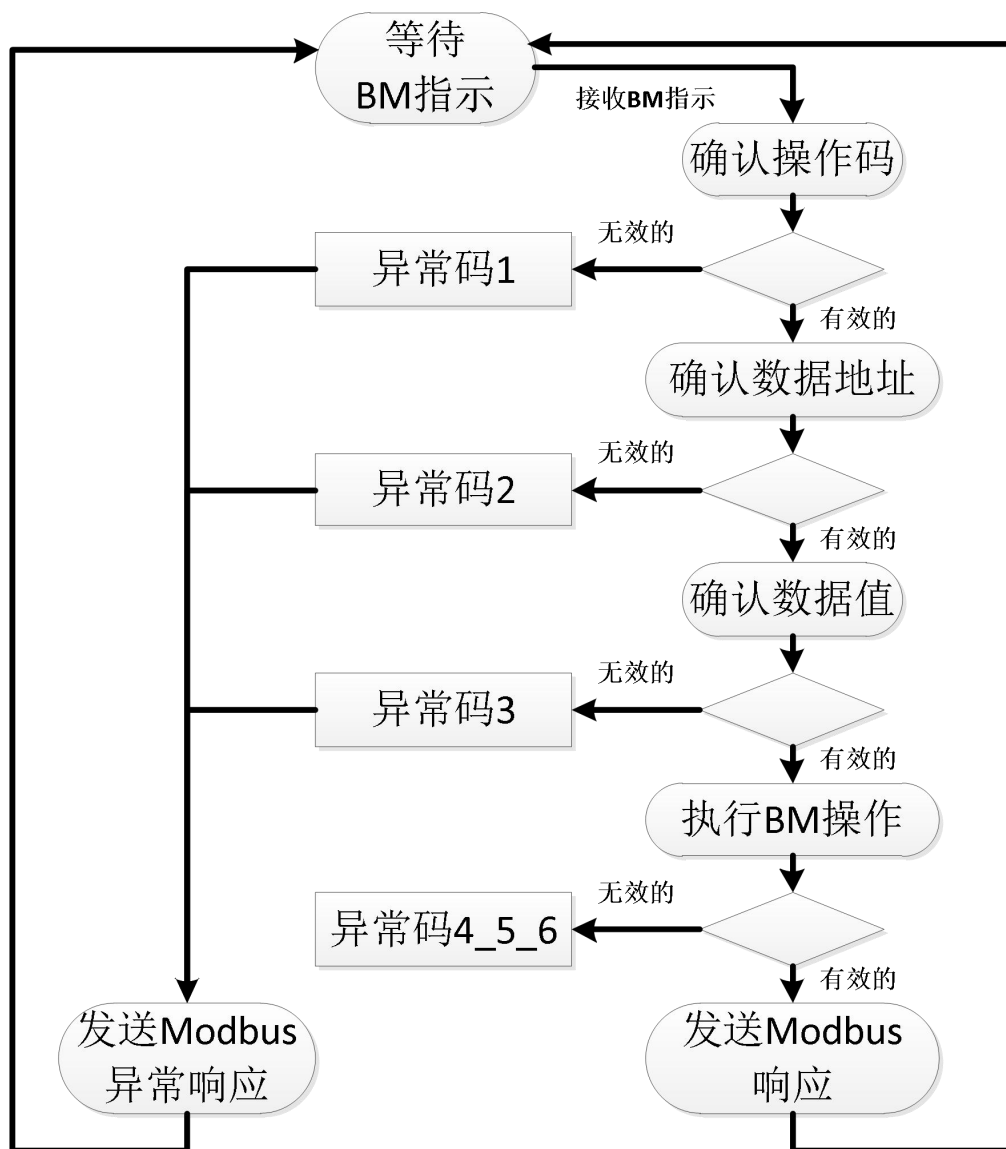
3.2 数据编码传输方式

MODBUS 使用‘big-Endian’方式表示地址和数据项。这意味着当发送多个字节时，首先发送最高有效位。例如：

寄存器大小	值	
16 - 比特	0x1234	发送的第一字节为 0x12 然后发送0x34

3.3 Modbus事务处理流程

下述状态图描述了在服务器侧 MODBUS 事务处理的一般处理过程。



一旦服务器处理请求，就使用合适的 MODBUS服务器事务建立MODBUS 响应。
根据处理结果，可以建立两种类型响应：

c) 一个正常的 MODBUS 响应：

响应功能码 = 请求功能码

d) 一个 MODBUS 异常响应：

用来为客户机提供处理过程中与被发现的差错相关的信息；

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。

第四章、功能码总述

4.1 功能码分类

本控制器实现的功能码如下：

数据访问方式			功能码（十六进制）
	物理离散量输入	读控制器运行状态位命令	0x02
	物理输入寄存器	读控制器运行状态参数值命令	0x04
		读控制器可调参数命令	0x03
		写单个控制器可调参数命令	0x06
		写多个控制器可调参数命令	0x10
	内部比特或物理线圈	读控制器可设定状态位	0x01
		写单个控制器可设定状态位	0x05
		写多个控制器可设定状态位	0x0F

4.2 功能码描述

4.2.1 读控制器可设定状态位命令（0x01）

应用举例ASCII

请求PDU：

可使用该功能码读取地址为0x01的控制器的一组连续的控制标志位，控制标志位的起始地址从0x0000开始，连续读出3个控制标志位的请求PDU如下：

名称	起始	地址		功能		起始地址				位数				LRC		结束	
		高	低	高	低	高字节	低字节	高字节	低字节	高字节	低字节	高字节	低字节	高	低	高	低
	1	1	2	1	2	1	2	1	2	1	2	1	2	1	2	1	2
序号	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
数据	3A	30	31	30	31	30	30	30	30	30	30	30	31	46	44	0D	0A

起始： 3A 请求 PDU 帧头

地址： 30 31 ASCII 码表示的目标从机地址 0x01。

功能： 30 31 ASCII 码表示的“读线圈命令”指令代码 0x01。

起始地址：30 30 30 30 ASCII 码表示的“读线圈命令”的起始地址 0x0000。

位数： 30 30 30 31 ASCII 码表示的“读线圈命令”的位数 0x0001，十进制数为 1。
 读出地址从 0x0000 开始的 1 个位。本控制器读线圈（控制标志位）的最大位数为 3，所以在实际应用中此处的数据不应大过 0x0003。

LRC： 46 44 ASCII 码表示的校验值 0xFD。

具体的计算方法是：

1、从连续的 2 个 ASCII 字符中提取出 1 个十六进制数

本例是从 30 31 30 31 30 30 30 30 30 30 30 31 中提取出

十六进制数 0x01, 0x01, 0x00, 0x00, 0x00, 0x01

2、0x01+0x01+0x00+0x00+0x00+0x01=0x0003

3、将累加和的高字节丢弃，只保留低字节 0x03

4、校验值：0xff-0x03+0x01=0xFD

5、将校验值转换成 ASCII 码的校验码：46 44

结束： 0D 0A 代表为帧结束。

响应PDU：

当地址为0x01的子节点上的控制器正确的接收到请求PDU后，会根据“读线圈命令”0x01和起始地址0x0000以及读命令的位数0x0001做出正确的响应。

例如本例的响应PDU 如下：

名称	起始	地址		功能		字节数		数据				LRC		结束	
								数据1		数据2					
		高	低	高	低	高	低	高	低	高	低	高	低	高	低
	1	1	2	1	2	1	2	1	2	1	2	1	2	1	2
序号	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
数据	3A	30	31	30	31	30	31	30	30	30	30	46	44	0D	0A

起始： 3A 响应 PDU 帧头

地址： 30 31 ASCII 码表示的目标从机地址 0x01。

功能： 30 31 ASCII 码表示的“读线圈命令”指令代码 0x01。

字节数：30 31 ASCII 码表示的数据段字节数 0x01，十进制数为 1。

说明在紧跟 30 32 之后有 2 个字节的数据：30 35 30 30。

数据： 30 30 30 30

ASCII 码表示的 16 进制数 0x00,0x00 这两个字节表示 16 个位状态。在实际应用中即使要求读 1 个位的状态，返回的数据字节数任然会是 2，因为返回的数据是以字节为最小单位的（一个字节是 8 位），所以必须用 0（位数据）填满不足的空位，这些填充的位是没有意义的。

LRC： 46 44 ASCII 码表示的校验值 0xFD。

其计算方法上面已经说过，需要注意的是这次计算的对象为：

30 31 30 31 30 31 30 30 30 30

结束： 0D 0A 帧结束。

应用举例RTU

请求PDU：

可使用该功能码读取地址为0x01的控制器的一组连续的控制标志位，控制标志位的起始地址从0x0000开始，连续读出3个控制标志位的请求PDU如下：

名称	地址	功能	起始地址		位数		CRC	
序号	1	2	3	4	5	6	7	8
数据	01	01	00	00	00	01	FD	CA

地址： 01 表示的目标从机地址 0x01。
 功能： 01 表示的“读线圈命令”指令代码 0x01。
 起始地址:00 00 表示的“读线圈命令”的起始地址 0x0000。

位数： 00 01 表示的“读线圈命令”的位数 0x0001，十进制数为 1。
 读出地址从 0x0000 开始的 1 个位。本控制器读线圈（控制标志位）的
最大位数为 3，所以在实际应用中此处的数据不应大过 0x0003。

CRC： FD CA 表示的校验值 0xFD 0xCA。
 具体的计算方法是：
 1、将多项式 $G(x) = x^{16} + x^{15} + x^2 + 1$ ，转化为二进制比特串为 11000000000000101，将数据 01 01 01 00 00 01 转换成二进制表示为 1 0000 0001 0000 0001 0000 0000 0000 0000 0001。
 2、在上面得到的二进制添加 16 个零后与多项式转换成的比特串进行“模二除法”
 3、得到结果为 1111110111001010，将二进制转换成十六进制 FD CA。
 4、校验值：FD CA

响应PDU：

当地址为0x01的子节点上的控制器正确的接收到请求PDU后，会根据“读线圈命令”0x01和起始地址0x0000以及读命令的位数0x0001做出正确的响应。

例如本例的响应PDU 如下：

名称	地址	功能	字节数	数据	CRC	
序号	1	2	3	4	5	6
数据	01	01	01	00	51	88

地址： 01 表示的目标从机地址 0x01。
 功能： 01 表示的“读线圈命令”指令代码 0x01。
 字节数:01 表示的数据段字节数 0x01，十进制数为 1。
 说明在紧跟 30 32 之后有 2 个字节的数据：00 。

数据： 00
 表示的 16 进制数 0x00 这个字节表示 1 个位状态。在实际应用中即使要求读 1 个位的状态，返回的数据字节数任然会是 2，因为返回的数据是以字节为最小单位的（一个字节是 8 位），所以必须用 0（位数据）填满不足的空位，这些填充的位是没有意义的。

LRC： 51 88 表示的校验值 0x51 0x88。
 其计算方法上面已经说过，需要注意的是这次计算的对象为：
01 01 01 00

4.2.1. 1 控制器线圈比特位状态表

地址	状态位	说明
0x00	恢复出厂设置	1: 恢复出厂设置
0x01	保存参数	1: 保存参数
0x02	运行状态	1: 停止输出 0: 开启输出

4.2.2 写单个控制器可设定状态位命令（0x05）

应用举例ASCII

请求PDU:

使用该功能码将地址为0x01的控制器上控制标志位的地址为0x0002处置1（停止输出）。

请求PDU如下:

名称	起始	地址		功能		起始地址				数据				LRC		结束	
		高	低	高	低	高字节		低字节		数据1		数据2		高	低	高	低
	1	1	2	1	2	1	2	1	2	1	2	1	2	1	2	1	2
序号	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
数据	3A	30	31	30	35	30	30	30	32	46	46	30	30	46	39	0D	0A

起始: 3A 请求 PDU 帧头

地址: 30 31 ASCII 码表示的目标从机地址 0x01。

功能: 30 35 ASCII 码表示的“写单个线圈”指令代码 0x05。

起始地址: 30 30 30 32 ASCII 码表示的“写单个线圈”的地址 0x0002。

数据: 46 46 30 30 ASCII 码表示的 0x0001(注意字节高地位顺序)。

需要说明的是将某一位位置 1 发送的数据是 0x00ff, 而将某一位清 0 发送的数据是 0x0000。

LRC: 46 39 ASCII 码表示的校验值 0xf9。

具体的计算方法是:

1、从连续的 2 个 ASCII 字符中提取出 1 个十六进制数

本例是从 30 31 30 35 30 30 30 32 46 46 30 30 中提取出十六进制数 0x01, 0x05, 0x00, 0x00, 0x02, 0xff, 0x00

2、 $0x01+0x05+0x00+0x02+0xff+0x00=0x0107$

3、将累加和的高字节丢弃, 只保留低字节 0x07

4、校验值: $0xff-0x07+0x01=0xf9$

5、将校验值转换成 ASCII 码的校验码: 46 39

结束: 0D 0A 代表为帧结束。

响应PDU：

当地址为0x01的子节点上的控制器正确的接收到请求PDU后，会根据“写单个线圈命令”0x05和地址0x0002做出正确的响应。

例如本例的响应PDU 如下：

名称	起始	地址		功能		起始地址				数据				LRC		结束	
										数据1		数据2					
		高	低	高	低	高字节	低字节	高	低	高	低	高	低	高	低		
	1	1	2	1	2	1	2	1	2	1	2	1	2	1	2	1	2
序号	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
数据	3A	30	31	30	35	30	30	30	32	46	46	30	30	46	39	0D	0A

可以看出写单个线圈命令的响应PDU格式以及返回的数据内容和请求PDU一样。

应用举例RTU

请求PDU：

使用该功能码将地址为0x01的控制器上控制标志位的地址为0x0002处置1（停止输出）。

请求PDU如下：

名称	地址	功能	起始地址		数据		CRC	
序号	1	2	3	4	5	6	7	8
数据	01	05	00	02	FF	00	2D	FA

地址： 01 表示的目标从机地址 0x01。
 功能： 05 表示的“写单个线圈”指令代码 0x05。
 起始地址：00 02 表示的“写单个线圈”的地址 0x0002。

数据： FF 00 表示的 0x0001(注意字节高地位顺序)。
 需要说明的是将某一位置 1 发送的数据是 0x00ff，而将某一位清 0 发送的数据是 0x0000。

CRC： 2D FA 表示的校验值 0x2D 0xFA。

具体的计算方法是：

1、将多项式 $G(x) = X^{16} + X^{15} + X^2 + 1$ ，转化为二进制比特串为 11000000000000101，将数据 01 05 00 02 FF 00 转换成二进制表示为 1 0000 0101 0000 0000 0000 0010 1111 1111 0000 0000。

2、在上面得到的二进制添加 16 个零后与多项式转换成的比特串进行“模二除法”

3、得到结果为 10110111111010，将二进制转换成十六进制 2D FA。

4、校验值：2D FA

响应PDU：

当地址为0x01的子节点上的控制器正确的接收到请求PDU后，会根据“写单个线圈命令”0x05和地址0x0002做出正确的响应。

例如本例的响应PDU 如下：

名称	地址	功能	起始地址		数据		CRC	
序号	1	2	3	4	5	6	7	8
数据	01	05	00	02	FF	00	2D	FA

可以看出写单个线圈命令的响应PDU格式以及返回的数据内容和请求PDU一样。

4.2.3 写多个控制器可设定状态位命令（0x0F）

应用举例ASCII

请求PDU：

使用该功能码可一次写地址为0x01的控制器多个线圈的值，线圈起始地址从0x00开始的32个控制标志位。

请求PDU如下：

名称	起始	地址		功能		起始地址				位数				字节数		数据	
	1	高	低	高	低	高字节		低字节		高字节		低字节		高	低	数据1	
		1	2	1	2	1	2	1	2	1	2	1	2	1	2	1	2
序号	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
数据	3A	30	31	30	46	30	30	30	30	30	30	32	30	30	34	30	35

接上表

数据						LRC		结束	
数据2		数据3		数据4					
高	低	高	低	高	低	高	低	高	低
1	2	1	2	1	2	1	2	1	2
18	19	20	21	22	23	24	25	26	27
30	30	38	30	30	30	34	37	0D	0A

起始： 3A 请求 PDU 帧头

地址： 30 31 ASCII 码表示的目标从机地址 0x01。

功能： 30 46 ASCII 码表示的“写多个线圈”指令代码 0x0f。

起始地址：30 30 30 30 ASCII 码表示的“写多个线圈”的地址 0x0000。

位数： 30 30 32 30 ASCII 码表示的“写多个线圈”的位数 0x0020，十进制数为 32。
写地址从 0x0000 开始的 32 个位。本控制器写线圈（控制标志位）的
最大位数为 32，所以在实际应用中此处的数据不应大过 0x0020。

字节数： 30 34 ASCII 码表示的数据段字节数 0x04，十进制数为 4。
即使要求写入 31 个位，此处的字节数任然应该是 4，因为数据是以字
节为最小单位（一个字节是 8 位）发送的，所以必须用 0（位数据）
填满不足的空位，这些填充的位是没有意义的。

数据： 30 35 30 30 38 30 30 30
ASCII 码表示的 0x0080,0x0005 (注意字节高地位顺序)。

LRC： 34 37 ASCII 码表示的校验值 0x47。
具体的计算方法是：
1、从连续的 2 个 ASCII 字符中提取出 1 个十六进制数
本例是从 30 31 30 46 30 30 30 30 30 30 32 30 30 34 30 35 30
30 38 30 30 30 中 提取出 十六进制数 0x01, 0x0f, 0x00, 0x00,
0x00, 0x20, 0x04, 0x05, 0x00, 0x80, 0x00,
2、 $0x01+0x0f+0x00+0x00+0x00+0x20+0x04+0x05+0x00+0x80+0x00=0x00b9$
3、将累加和的高字节丢弃，只保留低字节 0xb9
4、校验值： $0xff-0xb9+0x01=0x47$
5、将校验值转换成 ASCII 码的校验码：34 37

结束： 0D 0A 代表为帧结束。
对应 4.2.2.1 中的控制器控制标志位总表可以知道各个控制标志位的意义，在收到请求
PDU 后，控制器的当前控制状态位情况如下图表所示：

控制位地址	0x00	0x01	0x02	0x03	0x04	0x05	0x06	0x07
位状态	1	0	1	0	0	0	0	0

控制位地址	0x08	0x09	0x0A	0x0B	0x0C	0x0D	0x0E	0x0F
位状态	0	0	0	0	0	0	0	0

控制位地址	0x10	0x11	0x12	0x13	0x14	0x15	0x16	0x17
位状态	0	0	0	0	0	0	0	1
控制位地址	0x18	0x19	0x1A	0x1B	0x1C	0x1D	0x1E	0x1F
位状态	0	0	0	0	0	0	0	0

响应 PDU：

当地址为 0x01 的子节点上的控制器正确的接收到请求 PDU 后，会根据“写多个线圈命令”
0x0f 和起始地址 0x0000，写控制状态的位数 0x0020 以及数据 0x0500, 0x8000 做出正确的响应。
例如本例的响应 PDU 如下：

名称	起始	地址		功能		起始地址				位数				LRC		结束	
		高	低	高	低	高字节		低字节		高字节		低字节		高	低	高	低
	1	1	2	1	2	1	2	1	2	1	2	1	2	1	2	1	2
序号	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
数据	3A	30	31	30	46	30	30	30	30	30	30	32	30	44	30	0D	0A

起始： 3A 响应 PDU 帧头
 地址： 30 31 ASCII 码表示的目标从机地址 0x01。
 功能： 30 46 ASCII 码表示的“写多个线圈”指令代码 0x0f。
 起始地址：30 30 30 30 ASCII 码表示的“写多个线圈”的起始地址 0x0000。
 位数： 30 30 32 30 ASCII 码表示的“写多个线圈”的位数 0x0020，十进制数为 32。
 LRC : 44 30 ASCII 码表示的校验值 0xd0。
 其计算方法上面已经说过，需要注意的是这次计算的对象为：
 30 31 30 46 30 30 30 30 30 30 32 30

结束： 0D 0A 帧结束

应用举例 RTU

请求 PDU:

使用该功能码可一次写地址为 0x01 的控制器多个线圈的值，线圈起始地址从 0x00 开始的 32 个控制标志位。

请求 PDU 如下:

名称	地址	功能	起始地址		位数		字节数	数据				CRC	
序号	1	2	3	4	5	6	7	8	9	10	11	12	13
数据	01	0f	00	00	00	20	04	05	00	80	00	A5	84

地址： 01 表示的目标从机地址 0x01。
 功能： 0f 表示的“写多个线圈”指令代码 0x0f。
 起始地址：00 00 表示的“写多个线圈”的地址 0x0000。
 位数： 00 20 表示的“写多个线圈”的位数 0x0020，十进制数为 32。
 写地址从 0x0000 开始的 32 个位。本控制器写线圈（控制标志位）的
 最大位数为 32，所以在实际应用中此处的数据不应过大 0x0020。
 字节数： 04 ASCII 码表示的数据段字节数 0x04，十进制数为 4。
 即使要求写入 31 个位，此处的字节数任然应该是 4，因为数据是以字节为最小单位（一个字节是 8 位）发送的，所以必须用 0（位数据）
 填满不足的空位，这些填充的位是没有意义的。

数据： 05 00 80 00

ASCII 码表示的 0x0500,0x8000 (注意字节高地位顺序)。

CRC： A5 84 ASCII 码表示的校验值 0xA5 0x84。

具体的计算方法是：

1、将多项式 $G(x) = x^{16} + x^{15} + x^2 + 1$ ，转化为二进制比特串为 11000000000000101，将数据 01 0F 00 00 00 20 04 05 00 80 00 转换成二进制表示为 1 0000 1111 0000 0000 0000 0010 0000 0000 0100 0000 0101 0000 1000 0000。

2、在上面得到的二进制添加 16 个零后与多项式转换成的比特串进行“模二除法”

3、得到结果为 1010010110000100，将二进制转换成十六进制 A5 84。

4、校验值：A5 84

响应PDU：

当地址为0x01的子节点上的控制器正确的接收到请求PDU后，会根据“写多个线圈命令”0x0f和起始地址0x0000，写控制状态的位数0x0020以及数据0x0500,0x8000做出正确的响应。例如本例的响应PDU 如下：

名称	地址	功能	起始地址		位数		CRC	
序号	1	2	3	4	5	6	7	8
数据	01	0F	00	00	00	20	54	13

地址： 01 表示的目标从机地址 0x01。

功能： 00 0F 表示的“写多个线圈”指令代码 0x0f。

起始地址：00 00 表示的“写多个线圈”的起始地址 0x0000。

位数： 00 20 表示的“写多个线圈”的位数0x0020，十进制数为32。

CRC： 54 13 表示的校验值 0x54 0x13。

其计算方法上面已经说过，需要注意的是这次计算的对象为：

01 0F 00 00 00 20

4.2.4 读控制器运行状态位命令 (0x02)

应用举例ASCII

请求PDU:

可使用该功能码读取地址为0x01的控制器的一组连续状态位，状态位的起始地址从0x0000开始，连续读出32个状态位的请求PDU如下：

名称	起始	地址		功能		起始地址				位数				LRC		结束	
	1	高	低	高	低	高字节	低字节	高字节	低字节	高字节	低字节	高字节	低字节	高	低	高	低
	1	1	2	1	2	1	2	1	2	1	2	1	2	1	2	1	2
序号	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
数据	3A	30	31	30	32	30	30	30	30	30	30	32	30	44	44	0D	0A

起始： 3A 请求 PDU 帧头

地址： 30 31 ASCII 码表示的目标从机地址 0x01。

功能： 30 32 ASCII 码表示的“读离散量”指令代码 0x02。

起始地址:30 30 30 30 ASCII 码表示的“读离散量”的起始地址 0x0000。

位数： 30 30 32 30 ASCII 码表示的“读离散量”的位数 0x0020，其十进制数为 32。

读出地址从 0x0000 开始的 32 个位。本控制器输入离散量(运行状态)的最大位数为 32，所以在实际应用中此处的数据不应大过 0x0020。

LRC： 44 44 ASCII 码表示的校验值 0xDD。

具体的计算方法是：

1、从连续的 2 个 ASCII 字符中提取出 1 个十六进制数

本例是从 30 31 30 32 30 30 30 30 30 30 32 30 中提取出十六进制数 0x01, 0x02, 0x00, 0x00, 0x00, 0x20

2、 $0x01+0x02+0x00+0x00+0x00+0x20=0x0023$

3、将累加和的高字节丢弃，只保留低字节 0x23

4、校验值： $0xff-0x23+0x01=0xdd$

5、将校验值转换成 ASCII 码的校验码：44 44

结束： 0D 0A 帧结束。

响应PDU：

当地址为0x01的子节点上的控制器正确的接收到请求PDU后，会根据“读离散量”0x02和起始地址0x0000以及读命令的位数0x0020做出正确的响应。

例如本例的响应PDU 如下：

名称	起始	地址		功能		字节数		数据								LRC		结束	
								数据1		数据2		数据3		数据4					
		高	低	高	低	高	低	高	低	高	低	高	低	高	低	高	低		
	1	1	2	1	2	1	2	1	2	1	2	1	2	1	2	1	2	1	2
序号	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
数据	3A	30	31	30	32	30	34	30	30	46	46	34	30	30	30	42	41	0D	0A

起始： 3A 响应 PDU 帧头

地址： 30 31 ASCII 码表示的目标从机地址 0x01。

功能： 30 32 ASCII 码表示的“读离散量”指令代码 0x02。

字节数：30 34 ASCII 码表示的数据段字节数 0x04，十进制数为 4。

说明在紧跟 30 34 之后有 4 个字节的数据：30 30 46 46 34 30 30 30。

数据： 30 30 46 46 34 30 30 30

ASCII 码表示的 16 进制数 0x00,0xff,0x40,0x00 这四个字节表示 32 个位状态。在实际应用中即使只要求读 31 个位的状态，返回的数据字节数任然是 4，因为返回的数据是以字节为最小单位的（一个字节是 8 位），所以必须用 0（位数据）填满不足的空位，这些填充的位是没有意义的。

LRC : 42 41 ASCII 码表示的校验值 0xBA。

其计算方法上面已经说过，需要注意的是这次计算的对象为：

30 31 30 32 30 34 30 30 46 46 34 30 30 30

结束： 0D 0A 帧结束。

应用举例RTU

请求PDU:

可使用该功能码读取地址为0x01的控制器的一组连续状态位，状态位的起始地址从0x0000开始，连续读出32个状态位的请求PDU如下：

名称	地址	功能	起始地址		位数		CRC	
序号	1	2	3	4	5	6	7	8
数据	01	02	00	00	00	20	79	D2

地址： 01 表示的目标从机地址 0x01。

功能： 02 表示的“读离散量”指令代码 0x02。

起始地址：00 00 表示的“读离散量”的起始地址 0x0000。

位数： 00 20 表示的“读离散量”的位数 0x0020，其十进制数为 32。
读出地址从 0x0000 开始的 32 个位。本控制器输入离散量(运行状态)的最大位数为 32，所以在实际应用中此处的数据不应大过 0x0020。

CRC： 79 D2 表示的校验值 0x79 0xD2 。

具体的计算方法是：

- 1、将多项式 $G(x) = x^{16} + x^{15} + x^2 + 1$ ，转化为二进制比特串为 11000000000000101，将数据 01 02 00 00 00 20 转换成二进制表示为 1 0000 0010 0000 0000 0000 0000 0010 0000。
- 2、在上面得到的二进制添加 16 个零后与多项式转换成的比特串进行“模二除法”
- 3、得到结果为 1111001 11010010，将二进制转换成十六进制 79 D2。
- 4、校验值：79 D2

响应PDU：

当地址为0x01的子节点上的控制器正确的接收到请求PDU后，会根据“读离散量”0x02和起始地址0x0000以及读命令的位数0x0020做出正确的响应。

例如本例的响应PDU 如下：

名称	地址	功能	字节数	数据				CRC	
序号	1	2	3	4	5	6	7	8	9
数据	01	02	04	00	FF	40	00	FA	12

地址： 01 表示的目标从机地址 0x01。

功能： 02 表示的“读离散量”指令代码 0x02。

字节数： 04 表示的数据段字节数 0x04，十进制数为 4。
说明在紧跟 04 之后有 4 个字节的数据：00 00 00 00。

数据： 00 FF 40 00

表示的 16 进制数 0x00,0xFF,0x40,0x00 这四个字节表示 32 个位状态。在实际应用中即使只要求读 31 个位的状态，返回的数据字节数任然是 4，因为返回的数据是以字节为最小单位的（一个字节是 8 位），所以必须用 0（位数据）填满不足的空位，这些填充的位是没有意义的。

CRC： FA 12 表示的校验值 0xFA 0x12。

其计算方法上面已经说过，需要注意的是这次计算的對象为：

01 02 04 00 FF 40 00

本例中的 32 个状态位由四个十六进制数 0x00,0xFF,0x40,0x00 组成。

名称	数据			
	数据1	数据2	数据3	数据4
十六进制	0x00	0x00	0x00	0x00
二进制	0b00000000	0b11111111	0b01000000	0b00000000
字节顺序	最低字节(LSB)	次低字节	次高字节	最高字节(MSB)
	Byte0	Byte1	Byte2	Byte3

在上表中可以看出数据的字节顺序是 最低字节的数据排在高字节数据的前面。

而对于每一字节的数据（例如数据 3）它的顺序如下图所示

字节数据顺序	Bit0	Bit1	Bit2	Bit3	Bit4	Bit5	Bit6	Bit7
位数据	0 (最低位LSB)	0	0	0	0	0	0	0 (最高位MSB)

综上所述这 32 个位的顺序表示为：

名称	数据 1								数据 2							
位状态	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
位顺序	Bit 0	Bit 1	Bit 2	Bit 3	Bit 4	Bit 5	Bit 6	Bit 7	Bit 8	Bit 9	Bit 0	Bit 1	Bit 2	Bit 3	Bit 4	Bit 5

续上：

名称	数据 3								数据 4							
位状态	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
位顺序	Bit 16	Bit 17	Bit 18	Bit 19	Bit 20	Bit 21	Bit 22	Bit 23	Bit 24	Bit 25	Bit 26	Bit 27	Bit 28	Bit 29	Bit 30	Bit 31

对照 4.2.1.1 节中的控制器状态位总表可以读出地址从 0x00 开始到 0x1F 的总共 32 个状态位的信息。

4.2.4.1 控制器运行状态位总表

状态位地址从 0 开始的每个位的定义如下：

总共 64 个状态位（离散量输入，只读）

地址(序号)	状态位	说明
0x00 (0)	锁定状态位	1：锁定 0：未锁定
0x01 (1)	料满停机操作标志	1：关闭输出 0：允许输出
0x02 (2)	料空停机操作标志位	1：关闭输出 0：允许输出
0x03 (3)	逻辑结果操作标志位	1：关闭输出 0：允许输出
0x04 (4)	料满停机输入信号	1：有输入信号 0：无输入信号
0x05 (5)	逻辑分离后的控制输出状态	1：控制输出无输出

		0: 控制输出有输出
0x06~0x07 (6, 7)	补齐	
0x08 (8)	主输出A状态	1: 关闭 0: 运行
0x09 (9)	主输出B状态	1: 关闭 0: 运行
0x0A (10)	主输出C状态	1: 关闭 0: 运行
0x0B (11)	主输出D状态	1: 关闭 0: 运行
0x0C (12)	控制输出A状态	1: 关闭 0: 运行
0x0D (13)	控制输出B状态	1: 关闭 0: 运行
0x0E (14)	控制输出C状态	1: 关闭 0: 运行
0x0F (15)	控制输出D状态	1: 关闭 0: 运行
0x10 (16)	端口A过流	1: 端口A过流 0: 端口A未过流
0x11 (17)	端口B过流	1: 端口A过流 0: 端口A未过流
0x12 (18)	端口C过流	1: 端口A过流 0: 端口A未过流
0x13 (19)	端口D过流	1: 端口A过流 0: 端口A未过流
0x14 (20)	端口过热状态	1: 端口过热 0: 端口温度正常
0x15 (21)	温度传感器过热	1: 温度传感器过热 0: 温度传感器正常
0x16 (22)	电源开关状态	1: 关闭 0: 运行
0x17 (23)	锁定状态	1: 锁定 0: 正常
0x18 (24)	控制输出A短路	1: 控制输出A短路 0: 控制输出A未短路
0x19 (25)	控制输出B短路	1: 控制输出B短路 0: 控制输出B未短路
0x1A (26)	控制输出C短路	1: 控制输出C短路 0: 控制输出C未短路
0x1B (27)	控制输出D短路	1: 控制输出D短路 0: 控制输出D未短路
0x1C~0x1F (28~31)	补齐	

4.2.5 读控制器运行状态参数值命令（0x04）

应用举例ASCII

请求PDU:

使用该功能码从地址为0x01的控制器，读取连续的输入寄存器块内容，输入寄存器块的起始地址为0x0000，个数为2。

请求PDU如下：

名称	起始	地址		功能		起始地址				位数				LRC		结束	
		高	低	高	低	高字节		低字节		高字节		低字节		高	低	高	低
	1	1	2	1	2	1	2	1	2	1	2	1	2	1	2	1	2
序号	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
数据	3A	30	31	30	34	30	30	30	30	30	30	30	32	46	39	0D	0A

起始： 3A 请求 PDU 帧头

地址： 30 31 ASCII 码表示的目标从机地址 0x01。

功能： 30 34 ASCII 码表示的“读输入寄存器”指令代码 0x04。

起始地址：30 30 30 30 ASCII 码表示的“读输入寄存器”的地址 0x0000。

位数： 30 30 30 32 ASCII 码表示的“读输入寄存器”的个数 0x0002，十进制数为 2。

读地址从 0x0000 开始的 2 个寄存器。本控制器读输入寄存器（控制器运行状态值）的最大个数为 6，所以在实际应用中此处的数据不应大过 0x0006。

LRC： 46 39 ASCII 码表示的校验值 0xf9。

具体的计算方法是：

1、从连续的 2 个 ASCII 字符中提取出 1 个十六进制数

本例是从 30 31 30 34 30 30 30 30 30 30 30 32 中提取出十六进制数 0x01, 0x04, 0x00, 0x00, 0x00, 0x02,

2、 $0x01+0x04+0x00+0x00+0x00+0x02=0x0007$

3、将累加和的高字节丢弃，只保留低字节 0x07

4、校验值： $0xff-0x07+0x01=0xf9$

5、将校验值转换成 ASCII 码的校验码：46 39

结束： 0D 0A 代表为帧结束。

响应PDU：

当地址为0x01的子节点上的控制器正确的接收到请求PDU后，会根据“读输入寄存器命令”0x04和起始地址0x0000，以及读寄存器的个数0x0002做出正确的响应。

例如本例的响应PDU 如下：

名称	起始	地址		功能		字节数		数据							
								数据1				数据2			
		高	低	高	低	高字节		高字节		低字节		高字节		低字节	
	1	1	2	1	2	1	2	1	2	1	2	1	2	1	2
序号	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
数据	3A	30	31	30	34	30	34	30	31	37	37	30	30	30	30

LRC		结束	
高	低	高	低
1	2	1	2
16	17	18	19
37	46	0D	0A

起始： 3A 响应 PDU 帧头
 地址： 30 31 ASCII 码表示的目标从机地址 0x01。
 功能： 30 34 ASCII 码表示的“读输入寄存器”指令代码 0x04。
 字节数： 30 34 ASCII 码表示的数据段字节数 0x04，十进制数为 4。
 数据： 30 31 37 37 30 30 30 30
 ASCII 码表示的 0x0177,0x0000 (注意字节高地位顺序)。
 对应“4.2.2.1 控制器输入数据总表”这 2 个数据分别表示了温度 37.5℃
 和遥控调速 A 端口电压为 0 V。
 LRC : 37 46 ASCII 码表示的校验值 0x7F。
 其计算方法上面已经说过，需要注意的是这次计算的对象为：
 30 31 30 34 30 34 30 31 37 37 30 30 30 30

 结束： 0D 0A 帧结束。

应用举例RTU

请求PDU:

使用该功能码从地址为0x01的控制器，读取连续的输入寄存器块内容，输入寄存器块的起始地址为0x0000，个数为2。

请求PDU如下：

名称	地址	功能	起始地址		位数		CRC	
序号	1	2	3	4	5	6	7	8
数据	01	04	00	00	00	02	71	CB

地址： 01 表示的目标从机地址 0x01。
 功能： 04 表示的“读输入寄存器”指令代码 0x04。
 起始地址： 00 00 表示的“读输入寄存器”的地址 0x0000。
 位数： 00 02 表示的“读输入寄存器”的个数 0x0002，十进制数为 2。
 读地址从 0x0000 开始的 2 个寄存器。本控制器读输入寄存器（控制器运行状态值）的最大个数为 6，所以在实际应用中此处的数据不应大过 0x0006。

CRC： 71 CB ASCII 码表示的校验值 0x71 0xCB。

具体的计算方法是：

- 1、将多项式 $G(x) = x^{16} + x^{15} + x^2 + 1$ ，转化为二进制比特串为 11000000000000101，将数据 01 04 00 00 00 02 转换成二进制表示为 1 0000 0100 0000 0000 0000 0000 0000 0010。
- 2、在上面得到的二进制添加 16 个零后与多项式转换成的比特串进行“模二除法”
- 3、得到结果为 111000111001011，将二进制转换成十六进制 71 CB。
- 4、校验值：71 CB

响应PDU：

当地址为0x01的子节点上的控制器正确的接收到请求PDU后，会根据“读输入寄存器命令”0x04和起始地址0x0000，以及读寄存器的个数0x0002做出正确的响应。

例如本例的响应PDU 如下：

名称	地址	功能	字节数	数据				CRC	
序号	1	2	3	3	4	5	6	7	8
数据	01	04	04	01	77	00	00	4A	62

地址： 01 表示的目标从机地址 0x01。
 功能： 04 表示的“读输入寄存器”指令代码 0x04。
 字节数： 04 表示的数据段字节数 0x04，十进制数为 4。
 数据： 01 77 00 00
 表示的 0x0177,0x0000 (注意字节高地位顺序)。
 对应“4.2.2.1 控制器输入数据总表”这 2 个数据分别表示了温度 37.5℃ 和遥控调速 A 端口电压为 0 V。

CRC： 4A 62 ASCII 码表示的校验值 0x4A 0x62。

其计算方法上面已经说过，需要注意的是这次计算的对象为：

01 04 04 01 77 00 00

4.2.5.1 控制器输入数据总表

地址（序号）	状态位	说明
0x01	温度	【0, 65】℃
0x02	遥控调速A端口电压	【0, 5】V
0x03	遥控调速B端口电压	【0, 5】V
0x04	遥控调速C端口电压	【0, 5】V
0x05	遥控调速D端口电压	【0, 5】V
0x06	24V端口电压	【0, 24】V

4.2.6 读控制器可调参数命令（0x03）

应用举例ASCII

请求PDU:

使用该功能码从地址为0x01的控制器，读取连续的保持寄存器块内容，保持寄存器块的起始地址为0x0000，个数为10。

请求PDU如下：

名称	起始	地址		功能		起始地址				位数				LRC		结束	
		高	低	高	低	高字节	低字节	高字节	低字节	高字节	低字节	高字节	低字节	高	低	高	低
	1	1	2	1	2	1	2	1	2	1	2	1	2	1	2	1	2
序号	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
数据	3A	30	31	30	33	30	30	30	30	30	30	30	41	46	32	0D	0A

起始： 3A 请求 PDU 帧头

地址： 30 31 ASCII 码表示的目标从机地址 0x01。

功能： 30 33 ASCII 码表示的“读保持寄存器”指令代码 0x03。

起始地址：30 30 30 30 ASCII 码表示的“读保持寄存器”的地址 0x0000。

位数： 30 30 30 41 ASCII 码表示的“读保持寄存器”的个数 0x000A，十进制数为 10。

读地址从 0x0000 开始的 10 个寄存器。本控制器读保持寄存器（控制器设定参数值）的最大个数为 122，所以在实际应用中此处的数据不应大过 0x007A。

LRC： 46 32 ASCII 码表示的校验值 0xf2。

具体的计算方法是：

1、从连续的 2 个 ASCII 字符中提取出 1 个十六进制数

本例是从 30 31 30 33 30 30 30 30 30 30 30 41 中提取出十六进制数 0x01, 0x03, 0x00, 0x00, 0x00, 0x0a,

2、0x01+0x03+0x00+0x00+0x00+0x0a=0x000e

3、将累加和的高字节丢弃，只保留低字节 0x0e

4、校验值：0xff-0x0e+0x01=0xf2

5、将校验值转换成 ASCII 码的校验码：46 32

结束： 0D 0A 代表为帧结束。

响应PDU：

当地址为0x01的子节点上的控制器正确的接收到请求PDU后，会根据“读保持寄存器命令” 0x03和起始地址0x0000，以及读寄存器的个数0x000A做出正确的响应。

例如本例的响应PDU 如下：

名称	起始	地址		功能		字节数		数据							
		高	低	高	低	高	低	数据1				数据2			
	1	1	2	1	2	1	2	1	2	1	2	1	2	1	2
序号	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
数据	3A	30	31	30	33	31	34	30	31	46	34	30	30	30	32

数据															
数据3				数据4				数据5				数据6			
高字节	低字节	高字节	低字节	高字节	低字节	高字节	低字节	高字节	低字节	高字节	低字节	高字节	低字节	高字节	低字节
1	2	1	2	1	2	1	2	1	2	1	2	1	2	1	2
16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
30	30	30	32	30	30	30	30	30	30	39	36	30	30	30	35

数据															
数据7				数据8				数据9				数据10			
高字节	低字节	高字节	低字节	高字节	低字节	高字节	低字节	高字节	低字节	高字节	低字节	高字节	低字节	高字节	低字节
1	2	1	2	1	2	1	2	1	2	1	2	1	2	1	2
32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
30	30	30	30	30	30	30	32	30	30	30	30	30	30	30	30

LRC		结束	
高	低	高	低
1	2	1	2
48	49	50	51
35	32	0D	0A

起始： 3A 响应 PDU 帧头
 地址： 30 31 ASCII 码表示的目标从机地址 0x01。
 功能： 30 33 ASCII 码表示的“读保持寄存器”指令代码 0x03。

字节数： 31 34 ASCII 码表示的数据段字节数 0x14，十进制数为 20。

数据： 30 31 46 34 30 30 30 32 30 30 30 32 30 30 30 30 30 30 39 36 30 30 30 35
 30 30 30 30 30 30 30 32 30 30 30 30 30 30 30 30
 ASCII 码表示的 0x01F4,0x0002, 0x0002, 0x0000, 0x0096, 0x0005, 0x0000, 0x0002, 0x0000, 0x0000 (注意字节高地位顺序)。
 对应“4.2.6.1 控制器可调参数对应地址表”这 10 个数据分别表示了：
 1、主输出 A 输出频率 50Hz
 2、主输出 A 信号开延时 0.2s
 3、主输出 A 信号关延时 0.2s
 4、主输出 A 相位差 0
 5、主输出 A 输出电压 150V
 6、主输出 A 缓启动时间 0.5s
 7、主端口 A 源 1 为 0
 8、主端口 A 源 2 为 A 口
 9、主输出 A 开延时时间 0s
 10、主输出 A 关延时时间 0s

LRC : 35 32 ASCII 码表示的校验值 0x52。
 其计算方法上面已经说过，需要注意的是这次计算的对象为：
 30 31 30 33 31 34 30 31 46 34 30 30 30 32 30 30 30 32 30 30 30
 30 30 30 39 36 30 30 30 35 30 30 30 30 30 30 30 32 30 30 30
 30 30 30 30 30

结束： 0D 0A 帧结束。

应用举例RTU

请求PDU:

使用该功能码从地址为0x01的控制器，读取连续的保持寄存器块内容，保持寄存器块的起始地址为0x0000，个数为10。

请求PDU如下：

名称	地址	功能	起始地址		位数		CRC	
序号	1	2	3	4	5	6	7	8
数据	01	03	00	00	00	0A	C5	CD

地址： 01 表示的目标从机地址 0x01。

功能： 03 表示的“读保持寄存器”指令代码 0x03。

起始地址:00 00 表示的“读保持寄存器”的地址 0x0000。

位数： 00 0A 表示的“读保持寄存器”的个数 0x000A，十进制数为 10。
 读地址从 0x0000 开始的 10 个寄存器。本控制器读保持寄存器（控制器设定参数值）的最大个数为 122，所以在实际应用中此处的数据不应大过 0x007A。

CRC： C5 CD 表示的校验值 0xC5 0xCD。
 具体的计算方法是：
 1、将多项式 $G(x) = X^{16} + X^{15} + X^2 + 1$ ，转化为二进制比特串为

11000000000000101,将数据 01 03 00 00 00 0A 转换成二进制表示为 1 0000 0011 0000 0000 0000 0000 0000 1010。

2、在上面得到的二进制添加 16 个零后与多项式转换成的比特串进行“模二除法”

3、得到结果为 1100010111001101, 将二进制转换成十六进制 C5 CD。

4、校验值: C5 CD

响应PDU :

当地址为0x01的子节点上的控制器正确的接收到请求PDU后, 会根据“读保持寄存器命令” 0x03和起始地址0x0000, 以及读寄存器的个数0x000A做出正确的响应。

例如本例的响应PDU 如下:

名称	地址	功能	字节数	数据									
序号	1	2	3	4	5	6	7	8	9	10	11	12	13
数据	01	03	14	01	F4	00	02	00	02	00	00	00	96

数据										CRC	
14	15	16	17	18	19	20	21	22	23	24	25
00	05	00	00	00	02	00	00	00	00	9A	E8

地址: 01 表示的目标从机地址 0x01。

功能: 03 表示的“读保持寄存器”指令代码 0x03。

字节数: 14 表示的数据段字节数 0x14, 十进制数为 20。

数据: 01 F4 00 02 00 02 00 00 00 96 00 05 00 00 00 02 00 00 00 00

表示的 0x01F4, 0x0002, 0x0002, 0x0000, 0x0096, 0x0005, 0x0000, 0x0002, 0x0000, 0x0000 (注意字节高地位顺序)。

对应“4.2.6.1 控制器可调参数对应地址表”这 10 个数据分别表示了:

- 1、主输出 A 输出频率 50Hz
- 2、主输出 A 信号开延时 0.2s
- 3、主输出 A 信号关延时 0.2s
- 4、主输出 A 相位差 0
- 5、主输出 A 输出电压 150V
- 6、主输出 A 缓启动时间 0.5s
- 7、主端口 A 源 1 为 0
- 8、主端口 A 源 2 为 A 口
- 9、主输出 A 开延时时间 0s
- 10、主输出 A 关延时时间 0s

CRC : 9A E8 表示的校验值 0x9A 0xE8。

其计算方法上面已经说过, 需要注意的是这次计算的对象为:

4.2.6.1 控制器可调参数对应地址表

地址(序号)	数据	单位	范围	备注
0x00 (0)	端口A输出频率	Hz	【40.0, 400.0】	
0x01 (1)	端口A信号开延迟	s	【0, 20.0】	
0x02 (2)	端口A信号关延时	s	【0, 20.0】	
0x03 (3)	端口A相位差		【-180, 180】	
0x04 (4)	端口A输出电压	V	【0, 220】	
0x05 (5)	端口A缓启动时间	s	【0.1, 10.0】	
0x06 (6)	主端口A源1		【0, 17】	0: 0 1: 1 2: A口 3: b口 4: C口 5: d口 6: 0A口 7: 0b口 8: 0C口 9: 0D口 10: -A口 11: -b口 12: -C口 13: -D口 14: -0A口 15: -0b口 16: -0C口 17: -0D口
0x07 (7)	主端口A源2		【0, 17】	0: 0 1: 1 2: A口 3: b口 4: C口 5: d口 6: 0A口 7: 0b口 8: 0C口

				9: 0D口 10: -A口 11: -b口 12: -C口 13: -D口 14: -OA口 15: -Ob口 16: -OC口 17: -OD口
0x08 (8)	主输出A开延时时间	s	【0, 20.0】	
0x09 (9)	主输出A关延时时间	s	【0, 20.0】	
0x0A (10)	主逻辑A		【0, 3】	0: 与 1: 或 2: 异或 3: rs触发器
0x0B (11)	主输出 A 模式		【0, 1】	0: 延时模式 1: 保持模式
0x0C (12)	A控制输出端口源1		【0, 17】	0: 0 1: 1 2: A口 3: b口 4: C口 5: d口 6: OA口 7: Ob口 8: OC口 9: OD口 10: -A口 11: -b口 12: -C口 13: -D口 14: -OA口 15: -Ob口 16: -OC口 17: -OD口
0x0D (13)	A控制输出端口源2		【0, 17】	0: 0 1: 1 2: A口 3: b口 4: C口 5: d口 6: OA口

				7: 0b口 8: 0C口 9: 0D口 10: -A口 11: -b口 12: -C口 13: -D口 14: -0A口 15: -0b口 16: -0C口 17: -0D口
0x0E (14)	A控制输出 开延时	s	【0, 20.0】	
0x0F (15)	A控制输出关 延时	s	【0, 20.0】	
0x10 (16)	A控制输出逻辑		【0, 3】	0: 与 1: 或 2: 异或 3: rs触发器
0x11 (17)	A控制输出模式		【0, 1】	0: 延时模式 1: 保持模式
0x12 (18)	A最大输出电压	V	【0, 220】	
0x13 (19)	开机运行状态		【0, 2】	0: 开机运行 1: 开机停止 2: 开机跟随
0x14 (20)	开关传感器		【0, 2】	0: NPN 1: Auto 2: PNP
0x15 (21)	控制输出类型		【0, 2】	0: NPN 1: PNP 2: 推挽
0x16 (22)	RS485协议类型		【0, 1】	0: ASCII 1: RTU
0x17 (23)	RS485地址		【1, 31】	
0x18 (23)	RS485波特率	Kbps	【0, 6】	0: 0.3 1: 1.2 2: 2.4 3: 9.6 4: 19.2 5: 57.6 6: 115.2
0x19 (24)	保留			

0x1A (25)	A缓关闭事件	s	【0.1, 10.0】	
0x1B~0x1F (27, 31)	保留			
0x20 (32)	端口B输出频率	Hz	【40.0, 400.0】	
0x21 (33)	端口B信号开延迟	s	【0, 20.0】	
0x22 (34)	端口B信号关延时	s	【0, 20.0】	
0x23 (35)	端口B相位差		【-180, 180】	
0x24 (36)	端口B输出电压	V	【0, 220】	
0x25 (37)	端口B缓启动时间	s	【0.1, 10.0】	
0x26 (38)	主端口B源1		【0, 17】	0: 0 1: 1 2: A□ 3: b□ 4: C□ 5: d□ 6: 0A□ 7: 0b□ 8: 0C□ 9: 0D□ 10: -A□ 11: -b□ 12: -C□ 13: -D□ 14: -0A□ 15: -0b□ 16: -0C□ 17: -0D□
0x27 (39)	主端口B源2		【0, 17】	0: 0 1: 1 2: A□ 3: b□ 4: C□ 5: d□ 6: 0A□ 7: 0b□ 8: 0C□ 9: 0D□ 10: -A□ 11: -b□

				12: -C口 13: -D口 14: -OA口 15: -Ob口 16: -OC口 17: -OD口
0x28 (40)	主输出B开延时时间	s	【0, 20.0】	
0x29 (41)	主输出B关延时时间	s	【0, 20.0】	
0x2A (42)	主逻辑B		【0, 3】	0: 与 1: 或 2: 异或 3: rs触发器
0x2B (43)	主输出B模式		【0, 1】	0: 延时模式 1: 保持模式
0x2C (44)	B控制输出端口源1		【0, 17】	0: 0 1: 1 2: A口 3: b口 4: C口 5: d口 6: OA口 7: Ob口 8: OC口 9: OD口 10: -A口 11: -b口 12: -C口 13: -D口 14: -OA口 15: -Ob口 16: -OC口 17: -OD口
0x2D (45)	B控制输出端口源2		【0, 17】	0: 0 1: 1 2: A口 3: b口 4: C口 5: d口 6: OA口 7: Ob口 8: OC口 9: OD口

				10: -A口 11: -b口 12: -C口 13: -D口 14: -0A口 15: -0b口 16: -0C口 17: -0D口
0x2E (46)	B控制输出开 延时	s	【0, 20.0】	
0x2F (47)	B控制输出关 延时	s	【0, 20.0】	
0x30 (48)	B控制输出逻辑		【0, 3】	0: 与 1: 或 2: 异或 3: rs触发器
0x31 (49)	B控制输出模式		【0, 1】	0: 延时模式 1: 保持模式
0x32 (50)	B最大输出电压	V	【0, 220】	
0x33~0x39 (51, 58)	保留			
0x3A (58)	B缓关闭事件	s	【0.1, 10.0】	
0x3B~0x3C (59, 63)	保留			
0x40 (64)	端口C输出频率	Hz	【40.0, 400.0】	
0x41 (65)	端口C信号开 延迟	s	【0, 20.0】	
0x42 (66)	端口C信号关 延时	s	【0, 20.0】	
0x43 (67)	端口C相位差		【-180, 180】	
0x44 (68)	端口C输出电压	V	【0, 220】	
0x45 (69)	端口C缓启动 时间	s	【0.1, 10.0】	
0x46 (70)	主端口C源1		【0, 17】	0: 0 1: 1 2: A口 3: b口 4: C口 5: d口

				6: 0A口 7: 0b口 8: 0C口 9: 0D口 10: -A口 11: -b口 12: -C口 13: -D口 14: -0A口 15: -0b口 16: -0C口 17: -0D口
0x47 (71)	主端口C源2		【0, 17】	0: 0 1: 1 2: A口 3: b口 4: C口 5: d口 6: 0A口 7: 0b口 8: 0C口 9: 0D口 10: -A口 11: -b口 12: -C口 13: -D口 14: -0A口 15: -0b口 16: -0C口 17: -0D口
0x48 (72)	主输出C开延 时时间	s	【0, 20.0】	
0x49 (73)	主输出C关延 时时间	s	【0, 20.0】	
0x4A (74)	主逻辑C		【0, 3】	0: 与 1: 或 2: 异或 3: rs触发器
0x4B (75)	主输出C模式		【0, 1】	0: 延时模式 1: 保持模式
0x4C (76)	C控制输出端 口源1		【0, 17】	0: 0 1: 1 2: A口 3: b口

				4: C口 5: d口 6: 0A口 7: 0b口 8: 0C口 9: 0D口 10: -A口 11: -b口 12: -C口 13: -D口 14: -0A口 15: -0b口 16: -0C口 17: -0D口
0x4D (77)	C控制输出端口源2		【0, 17】	0: 0 1: 1 2: A口 3: b口 4: C口 5: d口 6: 0A口 7: 0b口 8: 0C口 9: 0D口 10: -A口 11: -b口 12: -C口 13: -D口 14: -0A口 15: -0b口 16: -0C口 17: -0D口
0x4E (78)	C控制输出开延时	s	【0, 20.0】	
0x4F (79)	C控制输出关延时	s	【0, 20.0】	
0x50 (80)	C控制输出逻辑		【0, 3】	0: 与 1: 或 2: 异或 3: rs触发器
0x51 (81)	C控制输出式		【0, 1】	0: 延时模式 1: 保持模式
0x52 (82)	C最大输出电压	V	【0, 220】	

0x53~0x59 (83, 89)	保留			
0x5A (90)	C 缓关闭事件	s	【0. 1, 10. 0】	
0x5B~0x5F (91, 95)	保留			
0x60(96)	端口 D 输出频率	Hz	【40. 0, 400. 0】	
0x61(97)	端口 D 信号开延迟	s	【0, 20. 0】	
0x62(98)	端口 D 信号关延时	s	【0, 20. 0】	
0x63(99)	端口 D 相位差		【-180, 180】	
0x64(100)	端口 D 输出电压	V	【0, 220】	
0x65(101)	端口 D 缓启动时间	s	【0. 1, 10. 0】	
0x66(102)	主端口 D 源1		【0, 17】	0: 0 1: 1 2: A□ 3: b□ 4: C□ 5: d□ 6: 0A□ 7: 0b□ 8: 0C□ 9: 0D□ 10: -A□ 11: -b□ 12: -C□ 13: -D□ 14: -0A□ 15: -0b□ 16: -0C□ 17: -0D□
0x67(103)	主端口 D 源2		【0, 17】	0: 0 1: 1 2: A□ 3: b□ 4: C□ 5: d□ 6: 0A□ 7: 0b□

				8: OC口 9: OD口 10: -A口 11: -b口 12: -C口 13: -D口 14: -OA口 15: -Ob口 16: -OC口 17: -OD口
0x68 (104)	主输出D开延时时间	s	【0, 20.0】	
0x69 (105)	主输出D关延时时间	s	【0, 20.0】	
0x6A (106)	主逻辑D		【0, 3】	0: 与 1: 或 2: 异或 3: rs触发器
0x6B (107)	主输出D模式		【0, 1】	0: 延时模式 1: 保持模式
0x6C (108)	D控制输出端口源1		【0, 17】	0: 0 1: 1 2: A口 3: b口 4: C口 5: d口 6: OA口 7: Ob口 8: OC口 9: OD口 10: -A口 11: -b口 12: -C口 13: -D口 14: -OA口 15: -Ob口 16: -OC口 17: -OD口
0x6D (109)	D控制输出端口源2		【0, 17】	0: 0 1: 1 2: A口 3: b口 4: C口 5: d口

				6: 0A口 7: 0b口 8: 0C口 9: 0D口 10: -A口 11: -b口 12: -C口 13: -D口 14: -0A口 15: -0b口 16: -0C口 17: -0D口
0x6E (110)	D控制输出开 延时	s	【0, 20.0】	
0x6F (111)	D控制输出关 延时	s	【0, 20.0】	
0x70 (112)	D控制输出逻辑		【0, 3】	0: 与 1: 或 2: 异或 3: rs触发器
0x71 (113)	D控制输出模式		【0, 1】	0: 延时模式 1: 保持模式
0x72 (114)	D最大输出电压	V	【0, 220】	
0x73~0x79 (115, 121)	保留			
0x7A (122)	D缓关闭事件	s	【0.1, 10.0】	

4.2.7 写单个控制器可调参数命令 (0x06)

应用举例ASCII

请求PDU:

使用该功能码对地址为0x01的控制器，进行写单个寄存器操作，寄存器的地址为0x0001写入的数据为2。

请求PDU如下：

名称	起始	地址		功能		寄存器地址				数据				LRC		结束	
		高	低	高	低	高字节		低字节		高字节		低字节		高	低	高	低
	1	1	2	1	2	1	2	1	2	1	2	1	2	1	2	1	2
序号	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17

数据	3A	30	31	30	36	30	30	30	31	30	30	30	32	46	36	0D	0A
----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----

起始： 3A 请求 PDU 帧头

地址： 30 31 ASCII 码表示的目标从机地址 0x01。

功能： 30 36 ASCII 码表示的“写单个寄存器”指令代码 0x06。

起始地址：30 30 30 31 ASCII 码表示的“写单个寄存器”的地址 0x0001。

数据： 30 30 30 32 ASCII 码表示的“写单个寄存器”的数据 0x0002，十进制数为 2。
对应“4.2.3.1 控制器可调参数对应地址表”可以找到对应的地址所代表的参数名称和意义

LRC： 46 36 ASCII 码表示的校验值 0xF6。
具体的计算方法是：

- 1、从连续的 2 个 ASCII 字符中提取出 1 个十六进制数
本例是从 30 31 30 36 30 30 30 31 30 30 30 32 中提取出十六进制数 0x01, 0x06, 0x00, 0x01, 0x00, 0x02,
- 2、 $0x01+0x06+0x00+0x01+0x00+0x02=0x000A$
- 3、将累加和的高字节丢弃，只保留低字节 0x0A
- 4、校验值： $0xFF-0x0A+0x01=0xF6$
- 5、将校验值转换成 ASCII 码的校验码：46 36

结束： 0D 0A 代表为帧结束。

响应PDU：

当地址为0x01的子节点上的控制器正确的接收到请求PDU后，会根据“写单个寄存器”0x06和寄存器地址0x0001，以及待写入的数据0x0002做出正确的响应。
例如本例的响应PDU 如下：

名称	起始	地址		功能		寄存器地址				数据				LRC		结束	
		高	低	高	低	高字节	低字节	高字节	低字节	高字节	低字节	高字节	低字节	高	低	高	低
	1	1	2	1	2	1	2	1	2	1	2	1	2	1	2	1	2
序号	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
数据	3A	30	31	30	36	30	30	30	31	30	30	30	32	46	36	0D	0A

可以看出写写单个寄存器命令的响应PDU格式以及返回的数据内容和请求PDU一样。

应用举例RTU

请求PDU：

使用该功能码对地址为0x01的控制器，进行写单个寄存器操作，寄存器的地址为0x0001写入的数据为2。

请求PDU如下：

名称	地	功	寄存器地址				数据				CRC			
----	---	---	-------	--	--	--	----	--	--	--	-----	--	--	--

	址	能						
序号	1	2	3	4	5	6	7	8
数据	01	06	00	01	00	02	59	CB

地址： 01 表示的目标从机地址 0x01。

功能： 06 表示的“写单个寄存器”指令代码 0x06。

起始地址：00 01 表示的“写单个寄存器”的地址 0x0001。

数据： 00 02 表示的“写单个寄存器”的数据 0x0002，十进制数为 2。
对应“4.2.3.1 控制器可调参数对应地址表”可以找到对应的地址所代表的参数名称和意义

CRC： 59 CB 表示的校验值 0x59 0xCB。
具体的计算方法是：
1、将多项式 $G(x) = x^{16} + x^{15} + x^2 + 1$ ，转化为二进制比特串为 11000000000000101，将数据 01 06 00 00 00 02 转换成二进制表示为 1 0000 0110 0000 0000 0000 0000 0000 0010。
2、在上面得到的二进制添加 16 个零后与多项式转换成的比特串进行“模二除法”
3、得到结果为 101100111001011，将二进制转换成十六进制 59 CB。
4、校验值：59 CB

响应PDU：

当地址为0x01的子节点上的控制器正确的接收到请求PDU后，会根据“写单个寄存器”0x06和寄存器地址0x0001，以及待写入的数据0x0002做出正确的响应。
例如本例的响应PDU 如下：

名称	地址	功能	寄存器地址		数据		CRC	
序号	1	2	3	4	5	6	7	8
数据	01	06	00	01	00	02	59	CB

可以看出写写单个寄存器命令的响应PDU格式以及返回的数据内容和请求PDU一样。

4.2.8 写多个控制器可调参数命令 (0x10)

应用举例ASCII

请求PDU：

使用该功能码对地址为0x01的控制器，进行写多个寄存器操作，寄存器的地址从0x0000开始的连续3个寄存器块，写入的数据为500, 7, 10。

请求PDU如下：

名称	起始	地址		功能		起始地址				结束地址				字节数	
		高	低	高	低	高字节		低字节		高字节		低字节		高	低
	1	1	2	1	2	1	2	1	2	1	2	1	2	1	2
序号	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
数据	3A	30	31	31	30	30	30	30	30	30	30	30	33	30	36

数据												LRC		结束	
数据1				数据2				数据3							
高字节		低字节		高字节		低字节		高字节		低字节		高	低	高	低
1	2	1	2	1	2	1	2	1	2	1	2	1	2	1	2
16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
30	31	46	34	30	30	30	37	30	30	30	41	45	30	0D	0A

起始： 3A 请求 PDU 帧头
 地址： 30 31 ASCII 码表示的目标从机地址 0x01。
 功能： 31 30 ASCII 码表示的“写多个寄存器”指令代码 0x10。
 起始地址：30 30 30 30 ASCII 码表示的“写多个寄存器”的地址 0x0000。
 结束地址：30 30 30 33 ASCII 码表示的“写多个寄存器”的结束地址 0x0003。
 数据： 30 31 46 34 30 30 30 37 30 30 30 41

ASCII 码表示的“写多个寄存器”的数据 0x01F4, 0x0007, 0x000A 十进制数为 500, 7, 10。对应“4.2.3.1 控制器可调参数对应地址表”可以找到对应的地址所代表的参数名称和意义

LRC： 45 30 ASCII 码表示的校验值 0xE0。

具体的计算方法是：

1、从连续的 2 个 ASCII 字符中提取出 1 个十六进制数

本例是从 30 31 31 30 30 30 30 30 30 30 33 30 36 30 31 46 34 30 30 30 37 30 30 30 41 中提取出十六进制数 0x01, 0x10, 0x00, 0x00, 0x00, 0x03, 0x06, 0x01, 0xF4, 0x00, 0x07, 0x00, 0x0A

2、 $0x01+0x10+0x00+0x00+0x00+0x03+0x06+0x01+0xF4+0x00+0x07+0x00+0x0A=0x120$

3、将累加和的高字节丢弃，只保留低字节 0x20

4、校验值： $0xFF-0x20+0x01=0xE0$

5、将校验值转换成 ASCII 码的校验码：45 30

结束： 0D 0A 代表为帧结束。

响应PDU：

当地址为0x01的子节点上的控制器正确的接收到请求PDU后，会根据“写多个寄存器”命令0x10和寄存器起始地址0x0000，寄存器结束地址0x0003，以及字节数据0x06和待写入的数据0x0099，0x0320，0x0005做出正确的响应。

例如本例的响应PDU 如下：

名称	起始	地址		功能		起始地址				结束地址				LRC		结束	
		高	低	高	低	高字节		低字节		高字节		低字节		高	低	高	低
	1	1	2	1	2	1	2	1	2	1	2	1	2	1	2	1	2
序号	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
数据	3A	30	31	31	30	30	30	30	30	30	30	30	33	45	43	0D	0A

起始： 3A 响应 PDU 帧头

地址： 30 31 ASCII 码表示的目标从机地址 0x01。

功能： 31 30 ASCII 码表示的“写多个寄存器”指令代码 0x10。

起始地址:30 30 30 30 ASCII 码表示的“写多个寄存器”的地址 0x0000。

结束地址:30 30 30 33 ASCII 码表示的“写多个寄存器”的结束地址 0x0003。

LRC： 45 43 ASCII 码表示的校验值 0xe3。

其计算方法上面已经说过，需要注意的是这次计算的对象为：

30 31 31 30 30 30 30 30 30 30 33

结束： 0D 0A 代表为帧结束。

应用举例RTU

请求PDU：

使用该功能码对地址为0x01的控制器，进行写多个寄存器操作，寄存器的地址从0x0000开始的连续3个寄存器块，写入的数据为500,7,10。

请求PDU如下：

名称	地址	功能	起始地址		结束地址		字节数	数据						CRC	
序号	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
数据	01	10	00	00	00	03	06	01	F4	00	07	00	0A	67	42

地址： 01 表示的目标从机地址 0x01。

功能： 01 表示的“写多个寄存器”指令代码 0x10。

起始地址:00 00 表示的“写多个寄存器”的地址 0x0000。

结束地址: 00 03 表示的“写多个寄存器”的结束地址 0x0003。

数据: 01 F4 00 07 00 0A

表示的“写多个寄存器”的数据 0x01F4, 0x0007, 0x000A 十进制数为 500, 7, 10。对应“4.2.3.1 控制器可调参数对应地址表”可以找到对应的地址所代表的参数名称和意义

CRC: 67 42 表示的校验值 0x67 0x42。

具体的计算方法是:

1、将多项式 $G(x) = x^{16} + x^{15} + x^2 + 1$, 转化为二进制比特串为 11000000000000101, 将数据 01 10 00 00 00 03 06 00 02 00 00 00 02 转换成二进制表示为 1 0001 0000 0000 0000 0000 0000 0000 0000 0000 0011 0000 0110 0000 0000 0000 0010 0000 0000 0000 0000 0000 0000 0010

2、在上面得到的二进制添加 16 个零后与多项式转换成的比特串进行“模二除法”

3、得到结果为 110011101000010, 将二进制转换成十六进制 67 42。

4、校验值: 67 42

响应PDU :

当地址为0x01的子节点上的控制器正确的接收到请求PDU后, 会根据“写多个寄存器”命令0x10和寄存器起始地址0x0000, 寄存器结束地址0x0003, 以及字节数据0x06和待写入的数据0x01F4, 0x0007, 0x000A做出正确的响应。

例如本例的响应PDU 如下:

名称	地址	功能	起始地址		结束地址		CRC	
序号	1	2	3	4	5	6	7	8
数据	01	10	00	00	00	03	80	08

地址: 01 表示的目标从机地址 0x01。

功能: 01 表示的“写多个寄存器”指令代码 0x10。

起始地址: 00 00 表示的“写多个寄存器”的地址 0x0000。

结束地址: 00 03 表示的“写多个寄存器”的结束地址 0x0003。

CRC: 80 08 表示的校验值 0x80 0x08。

其计算方法上面已经说过, 需要注意的是这次计算的对象为:

01 10 00 00 00 03

参考资料:

1. MODBUS APPLICATION PROTOCOL SPECIFICATION

Modbus_Application_Protocol_V1_1b.pdf

2. MODBUS over Serial Line Specification and Implementation Guide

Modbus_over_serial_line_V1_02.pdf