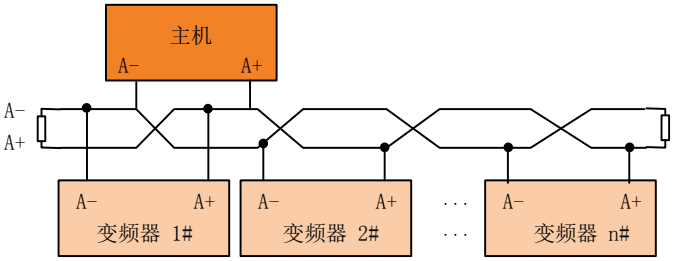
MODBUS 通讯协议

1.1 适用范围

1、 适用系列：A90 系列

2、 适用网络：支持 MODBUS-RTU 协议格式，具备 RS-485 总线的“单主多从”通讯网络。



1.2 接口方式

RS-485 异步半双工通讯模式，最低有效位优先发送；

RS-485 网络地址：1～247 可设，0 为广播地址；

RS-485 端子默认数据格式：1-8-N-1[1]（1-8-E-1，1-8-O-1，1-8-N-2，1-8-E-2 和 1-8-O-2 可选）；

RS-485 端子默认波特率：9600bps（4800bps、19200bps、38400bps、57600bps 和 115200bps 可选）；

推荐使用双绞屏蔽线作为通讯线，以降低外部干扰对通讯的影响。

[1]：*1-8-N-1，表示 1 起始位-每字节数据 8 个字符-无奇偶校验-1 停止位。E，偶校验。O，奇校验。*

1.3 协议格式

1.3.1 报文格式

如图 1 所示，一个标准的 MODBUS 报文包括起始标记、RTU 报文（Remote Terminal Unit，远程终端装置）和结束标记。

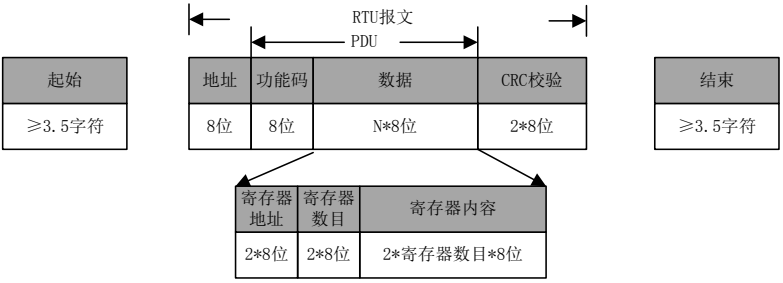


图 1 RTU 模式报文帧示意图

其中 RTU 报文包括地址码、PDU（Protocol DataUnit，协议数据单元）和 CRC[2]校验。PDU 包括功能码和数据部分（主要包括寄存器地址、寄存器数目和寄存器内容等，各功能码其详细定义各不相同，详见 1.3.3 功能码。）

[2]:*CRC 校验低字节在前，高字节在后*

1.3.2 地址码

地址范围	用途
1～247	从机
0	广播

1.3.3 功能码

MODBUS 功能码分类图 2 所示。

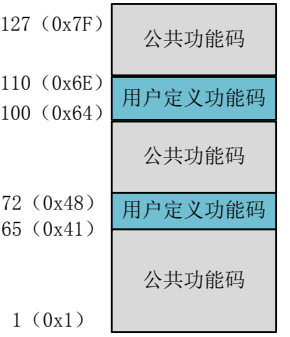


图 2 MODBUS 功能码分类

如表 1 所示，A90 系列产品主要涉及**公共类功能码**。如 0x03 读多个寄存器或状态字功能码、0x06 写单个寄存器或命令功能码、0x10 写多个寄存器或命令功能码和 0x08 诊断功能码。

另外，为了完成一些特定的功能，如写寄存器（RAM）但不存 EEPROM，在**用户定义功能码**中自定义了 0x41 写单个寄存器或命令功能码（不保存）和 0x42 写多个寄存器或命令功能码（不保存）。

当从设备接收到异常有效数据时，会返回相关异常信息（详见 1.3.7 异常信息响应）。为与正常通讯数据区分，特定义异常功能码。与正常请求功能码相对应，**异常功能码 = 请求功能码 + 0x80**。

表 1 A90 系列产品定义功能码

功能码	异常功能码	功能
03	83	读多个寄存器或状态字功能码
41	C1	写单个寄存器或命令功能码，不保存
42	C2	写多个寄存器或命令功能码，不保存
08	88	诊断功能码
06	86	写单个寄存器或命令功能码
10	90	写多个寄存器或命令功能码

以下几节针对因功能而各异的 PDU 部分做详细说明。

1.3.3.1 0x03 读多个寄存器或状态字功能码

在一个远程设备中，使用该功能码读取保持寄存器连续块的内容。请求 PDU 说明了起始寄存器地址和寄存器数量。

将响应报文中的寄存器数据分成每个寄存器有两字节，对于每个寄存器，第一个字节包括高位比特，第二个字节包括低位比特。

- 请求 PDU

功能码	1 个字节	**0x03**
起始地址	2 个字节	0x0000～0xFFFF
寄存器数量	2 个字节	1～16

- 响应 PDU

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*=寄存器数量

- 错误 PDU

差错码	1 个字节	**0x83**
异常码	1 个字节	01 或 02 或 03 或 04

以下是一个请求读寄存器 F19.00～F19.05（最近一次故障相关信息）的实例：

请求		响应			
域名	（0x）	域名(正常)	（0x）	域名(异常)	（0x）
功能码	03	功能码	03	功能	83
起始地址 Hi	13	字节数	0C	异常码	03(例，下同)
起始地址 Lo	00	寄存器值 Hi（F19.00）	00		
寄存器数量 Hi	00	寄存器值 Lo（F19.00）	11		
寄存器数量 Lo	06	寄存器值 Hi（F19.01）	00		
		寄存器值 Lo（F19.01）	00		
		寄存器值 Hi（F19.02）	00		
		寄存器值 Lo（F19.02）	00		
		寄存器值 Hi（F19.03）	01		
		寄存器值 Lo（F19.03）	2C		
		寄存器值 Hi（F19.04）	00		
		寄存器值 Lo（F19.04）	00		
		寄存器值 Hi（F19.05）	00		
		寄存器值 Lo（F19.05）	00		

由返回数据可知，之前变频器发生 17（0011H）：温度传感器异常故障，当时输出频率为 0.00Hz、输出电流为 0.00A、母线电压为 300V（012CH）、加减速状态为待机和工作时间为 0hour。

★： 目前 MODBUS 协议 0x03 功能码支持跨组读取多个功能码，但建议客户若无特殊需求，不要跨组读取，以便于在我司产品升级后客户的软件程序不用升级。

1.3.3.2 0x41 写单个寄存器或命令功能码（不保存）

在一个远程设备中，使用该功能码写单个非保持寄存器。

请求 PDU 说明了被写入寄存器的地址。

正常响应是请求的应答，在写入寄存器内容之后返回这个正常响应。

- 请求 PDU

功能码	1 个字节	**0x41**
寄存器地址	2 个字节	0x0000～0xFFFF
寄存器值	2 个字节	0x0000～0xFFFF

- 响应 PDU

功能码	1 个字节	**0x41**
寄存器地址	2 个字节	0x0000～0xFFFF
寄存器值	2 个字节	0x0000～0xFFFF

- 错误 PDU

差错码	1 个字节	**0xC1**
异常码	1 个字节	见表 4

以下是一个请求将主频率源 A（7001H）改为“-50.00%”的实例：

请求		响应			
域名	（0x）	域名（正常）	（0x）	域名(异常)	（0x）
功能	41	功能	41	功能	C1
寄存器地址 Hi	70	寄存器地址 Hi	70	异常码	03
寄存器地址 Lo	01	寄存器地址 Lo	01		
寄存器值 Hi	EC	寄存器值 Hi	EC		
寄存器值 Lo	78	寄存器值 Lo	78		

★ 用此功能码不能对“○”属性（运行时不可改）参数进行操作，即只能修改“●”属性（运行时可修改）参数进行操作，否则，返回错误码 1。

1.3.3.3 0x42 写多个寄存器或命令功能码（不保存）

在一个远程设备中，使用该功能码写连续非保持寄存器块(1 至 16 个寄存器)。

在请求数据域中说明了请求写入的值。每个寄存器将数据分成两字节。

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

功能码	1 个字节	**0x42**
起始地址	2 个字节	0x0000～0xFFFF
寄存器数量	2 个字节	1～16
字节数	1 个字节	2×N*
寄存器值	N*×2 个字节	

N*=寄存器数量

- 响应 PDU

功能码	1 个字节	**0x42**
起始地址	2 个字节	0x0000～0xFFFF
寄存器数量	2 个字节	1～16

- 错误 PDU

差错码	1 个字节	0xC2
异常码	1 个字节	见表 4

以下是一个请求将加速时间 1（F00.14）设为 5.00，减速时间 1（F00.15）设为 6.00 的实例：

请求		响应			
域名	（0x）	域名（正常）	（0x）	域名(异常)	（0x）
功能	42	功能	42	功能	C2
起始地址 Hi	00	起始地址 Hi	00	异常码	03
起始地址 Lo	0E	起始地址 Lo	0E		
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（F00.14）	01				
寄存器值 Lo（F00.14）	F4				
寄存器值 Hi（F00.15）	02				
寄存器值 Lo（F00.15）	58				

★ 用此功能码不能对“○”属性（运行时不可改）参数进行操作，即只能修改“●”属性（运行时可修改）参数进行操作，否则，返回错误码 1。

1.3.3.4 0x08 诊断功能码

Modbus 功能码 08 提供一系列测试，用于检查客户机（主站）设备与服务器（从站）之间的通信系统，或服务器中的各种内部差错状态。

这个功能使用询问中的 2 个字节的子功能码域来定义所执行的测试类型。服务器在正常的响应中

复制功能码和子功能码。一些诊断会导致远程设备通过正常响应的数据域返回相应数据。

通常，向远程设备发送诊断功能，不影响远程设备中的用户程序运行。诊断不能访问用户逻辑，例如：离散量和寄存器。某些功能可以任意地复位远程设备中的差错计数器。

我司所用诊断功能主要为线路诊断（0000），用于测试主从机是否能正常通讯。对返回询问数据请求的正常响应是回送相同的数据。同时还复制功能码和子功能码。

- 请求 PDU

功能码	1 个字节	0x08
子功能码	2 个字节	0x0000～0xFFFF

数据	2 个字节	0x0000～0xFFFF

- 响应 PDU

功能码	1 个字节	**0x08**
子功能码	2 个字节	0x0000～0xFFFF
数据	2 个字节	0x0000～0xFFFF

- 错误 PDU

差错码	1 个字节	**0x88**
异常码	1 个字节	见表 4

- 子功能码

子功能	含义	数据域（请求）	数据域（响应）
0000	返回询问数据	任意	复制请求数据
…			

0000：在响应中返回请求数据域中传递的数据。全部报文应该与请求报文一致。

下表是一个请求远程设备返回询问数据的实例。它使用子功能码 0000。用两个字节数据域（0xA537)发送返回的数据。

请求		响应			
域名	（0x）	域名（正常）	（0x）	域名(异常)	（0x）
功能	08	功能	08	功能	88
子功能码 Hi	00	子功能码 Hi	00	异常码	03
子功能码 Lo	00	子功能码 Lo	00		
数据 Hi	A5	数据 Hi	A5		
数据 Lo	37	数据 Lo	37		

1.3.3.5 0x06 写单个寄存器或命令功能码

在一个远程设备中，使用该功能码写单个保持寄存器。

请求 PDU 说明了被写入寄存器的地址。

正常响应是请求的应答，在写入寄存器内容之后返回这个正常响应。

- 请求 PDU

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

- 响应 PDU

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

寄存器值	2 个字节	0x0000～0xFFFF

- 错误 PDU

差错码	1 个字节	**0x86**
异常码	1 个字节	见表 4

以下是一个请求将电机 1 驱动控制方式（F00.01）改为“1：SVC”的实例：

请求		响应			
域名	（0x）	域名（正常）	（0x）	域名(异常)	（0x）
功能	06	功能	06	功能	86
寄存器地址 Hi	00	寄存器地址 Hi	00	异常码	03
寄存器地址 Lo	01	寄存器地址 Lo	01		
寄存器值 Hi	00	寄存器值 Hi	00		
寄存器值 Lo	01	寄存器值 Lo	01		

★ 经常修改的变频器功能代码不能用 0x06 完成，以免损坏变频器。

0x41“只改不存”用户自定义功能码对应 0x06 标准公共功能码——其功能码定义与相对应标准功能码相同（请求、应答与错误 PDU 均相同），不同之处为从机响应此用户自定义功能码时，只修改 RAM 对应值，而不保存至 EEPROM（保持寄存器）。

针对 F00.07 类经常修改功能码，建议用 0x41 功能码完成（修改主频率源 A 也可直接操作 7001H，详见章节 1.3.3.2 和 1.3.4），避免损坏变频器。具体操作如下所述。

请求		响应	
域名	（0x）	域名（正常）	（0x）
功能	41	功能	41
寄存器地址 Hi	00	寄存器地址 Hi	00
寄存器地址 Lo	07	寄存器地址 Lo	07
寄存器值 Hi	13	寄存器值 Hi	13
寄存器值 Lo	88	寄存器值 Lo	88

以上数据表示把给定频率（F00.07）改为 50.00Hz，即刻生效，但不存入 EEPROM。即改写后，变频器以 50.00Hz 运行，但重新上电后以修改之前频率运行。

1.3.3.6 0x10 写多个寄存器或命令功能码

在一个远程设备中，使用该功能码写连续寄存器块(1 至 16 个寄存器)。

在请求数据域中说明了请求写入的值。每个寄存器将数据分成两字节。

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

功能码	1 个字节	**0x10**

起始地址	2 个字节	0x0000～0xFFFF
寄存器数量	2 个字节	1～16
字节数	1 个字节	2×N*
寄存器值	N*×2 个字节	

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	0x0000～0xFFFF
寄存器数量	2 个字节	1～16

- 错误 PDU

差错码	1 个字节	**0x90**
异常码	1 个字节	见表 4

以下是一个请求将 00 01 和 00 03 写入 F03.00 开始的 2 个寄存器（即设置 Y1 和 Y2 输出端子功能）的实例：

请求		响应			
域名	（0x）	域名（正常）	（0x）	域名(异常)	（0x）
功能	10	功能	10	功能	90
起始地址 Hi	03	起始地址 Hi	03	异常码	03
起始地址 Lo	00	起始地址 Lo	00		
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（F03.00）	00				
寄存器值 Lo（F03.00）	01				
寄存器值 Hi（F03.01）	00				
寄存器值 Lo（F03.01）	03				

★ 经常修改的变频器功能代码不能用 0x10 完成，以免损坏变频器，详见章节 1.3.3.5 说明。

1.3.4 寄存器地址分布

表 2 MODBUS 协议寄存器地址定义详解

地址空间	说明
功能码 0000H～6F63H	针对功能码 FXX.YY，其地址高位为 XX 的十六进制，地址低位为 YY 的十六进制。如 F00.14，其地址为 000EH（00D=00H，14D=0EH）。
功能码（掉电不存）	用 0x06 或 0x10 功能码设置参数时，可通过“原地址+8000H”

8000H～EF63H		方式实现“设置值立即生效、掉电不存”功能。如 F00.14 对应地址为 800EH（=000EH+8000H）。	
控制命令（**只写**） 7000H ～ 71FFH	7000H 控制字	0000H	无效指令
		0001H	正转运行
		0002H	反转运行
		0003H	JOG 正转
		0004H	JOG 反转
		0005H	减速停车
		0006H	快速停车
		0007H	自由停车
		0008H	故障复位
		0009H	+/-输入切换
		000BH	JOG 停车
		其它～00FFH	保留
	7001H	主通道频率 A 通讯百分比给定	-100.00%～100.00%（100%=最大频率）
	7002H	辅通道频率 B 通讯百分比给定	-100.00%～100.00%（100%=最大频率）
	7003H	转矩通讯给定	-200.00%～200.00%(100%=数字转矩给定)
	7004H	过程 PID 给定通讯给定	-100.00%～100.00%
	7005H	过程 PID 反馈通讯给定	-100.00%～100.00%
	7006H	VF 分离模式电压给定	0.00%～100.00%（数字给定基准）
	7007H～7009H	保留	
	700AH	上限频率通讯百分比给定	0.00%～200.00%（数字给定基准）
	700BH	转矩控制的上限频率通讯百分比给定	0.00%～200.00%（数字给定基准）
	700CH	惯量补偿线速度输入	0.00%～100.00%（数字给定基准）
	700DH～700EH	保留	
	700FH	主从通讯给定	-100.00%～100.00%（最大值基准）
	7010H～7013H	保留	
	7014H	外部故障	外部设备（包括选件卡）故障输入
	7015H	主通道频率 A 通讯给定	0.00～最大频率
	7016H	辅通道频率 B 通讯给定	0.00～最大频率
	7017H	上限频率通讯给定	0.00～最大频率

	7018H	转矩控制的上限频率通讯给定	0.00～最大频率	
	7019H	速度控制的转矩上限通讯给定	0.0～250.0%（按 100.0%或直接发送均可）	
	7019H～71FFH	保留		
工作状态 7200H～73FFH	7200H 状态字 1	Bit7～0 运行状态	00H	参数设定
			01H	从机运行
			02H	JOG 运行
			03H	自学习运行
			04H	从机停车
			05H	JOG 停车
			06H	故障状态
			07H	工厂自检
			08H～0FFH	保留
		Bit15～8 故障信息	00H	变频器正常运行
			xxH	变频器故障状态，“xx”为故障代码
	7201H 状态字 2	Bit0 给定方向	1	-给定有效
			0	+给定有效
		Bit1 运行方向	1	频率输出反转
			0	频率输出正转
		Bit3～2 运行方式	00	速度控制方式
			01	转矩矩控制方式
			10	保留
			11	保留
		Bit4 参数保护	1	参数保护有效
			0	参数保护无效
		Bit6～5	保留	
		Bit8～7 给定方式	00	键盘控制
			01	端子控制
			10	通讯控制
			11	保留
		Bit9	保留	
		Bit10 警告	0	无警告
			1	警告状态（详情参看 7230H）
		Bit15～10	保留	
	7202H 监视频率+/-状态字 1（1：-；0：+）	Bit0	输出频率	
		Bit1	输入频率	
		Bit2	同步频率	
		Bit3	保留	
		Bit4	估算反馈频率	

		Bit5	估算滑差频率							
		Bit6	负载速度							
		Bit15～7	保留							
	7203H	输出频率								
	7204H	输出电压								
	7205H	输出功率								
	7206H	运行转速								
	7207H	母线电压								
	7208H	输出转矩								
	7209H	开关量输入 1	15	14	13	12	11	10	9	8
			*	*	*	*	*	X11	X10	X9
			7	6	5	4	3	2	1	0
			X8	X7	X6	X5	X4	X3	X2	X1
	720AH	开关量输入 2	15	14	13	12	11	10	9	8
			VX8	VX7	VX6	VX5	VX4	VX3	VX2	VX1
			7	6	5	4	3	2	1	0
			*	*	*	*	AI4	AI3	AI2	AI1
	720BH	开关量输出 1	15	14	13	12	11	10	9	8
			*	*	*	*	*	*	*	*
			7	6	5	4	3	2	1	0
			*	*	*	Y3	Y2	Y1	R2	R1
	720CH	开关量输出 2	15	14	13	12	11	10	9	8
			VY8	VY7	VY6	VY5	VY4	VY3	VY2	VY1
			7	6	5	4	3	2	1	0
			*	*	*	*	*	*	*	*
	720DH	前二次故障								
	720EH	前三次故障								
	720FH	最近一次故障								
	7210H	最近一次故障输出频率								
	7211H	最近一次故障输出电流								
	7212H	最近一次故障母线电压								
	7213H	最近一次故障运行状态								
	7214H	最近一次故障工作时间								
	7215H	设定加速时间								
	7216H	设定减速时间								
	7217H	累计长度								
	7218H	保留								
	7219H	UP/DOWN 偏移频率符号（0/1：+/-）								
	7224H	输出电流								
	7225H	给定频率								
	7228H	累计上电时间								

	7230H	警告号	0：无警告；其它：当前警告标识
	其它～73FFH	保留	
产品信息 7500H ～ 75FFH	7500H	性能软件序列号 1	与功能码 F12.22 对应
	7501H	性能软件序列号 2	与功能码 F12.23 对应
	7502H	功能软件序列号 1	与功能码 F12.24 对应
	7503H	功能软件序列号 2	与功能码 F12.25 对应
	7504H	键盘软件序列号 1	与功能码 F12.26 对应
	7505H	键盘软件序列号 2	与功能码 F12.27 对应
	7506H	产品序列号 1	与功能码 F12.28 对应
	7507H	产品序列号 2	与功能码 F12.29 对应
	7508H	产品序列号 3	与功能码 F12.30 对应
	7509H～75FFH	保留	
其它	保留		

1.3.5 帧数据长度定义

MODBUS 报文 RTU 帧 PDU 部分读/写寄存器数量在 1～16 范围内。针对不同功能码，其 RTU 帧实际长度会有不同，详细见表 3 所示。

表 3 RTU 帧长度与功能码对照表

功能码（0x）	RTU 帧长度（字节）			最大长度（字节）
	请求	正常响应	异常响应	
03	8	$5+2N_r$ [3]	5	37
41（06）	8	8	5	8
08	8	8	5	8
42（10）	$9+2N_w$ [4]	8	5	41

[3]：$N_r \leq 16$，表示请求读寄存器的数量；

[4]：$N_w \leq 16$，表示请求写寄存器的数量；

[5]：$N_w+N_r \leq 16$；

1.3.6 CRC 校验

CRC 校验低字节在前，高字节在后。

发送设备首先计算 CRC 值，并附在发送信息中。接收设备接收后将重新计算 CRC 值，并且把计算值与接收的 CRC 值做比较。如果两个值不相等，则说明发送过程中有错误发生。

CRC 校验的计算过程：

（1）定义一个 CRC 寄存器，并赋一个初值，FFFFH。

（2）将发送信息的第一个字节与 CRC 寄存器的值进行异或计算，并将结果放到 CRC

寄存器中。从地址码开始，起始位和停止位不参加计算。

（3）提取和检查 LSB（CRC 寄存器的最低位）。

（4）如果 LSB 是 1，CRC 寄存器的各位向右移动一位，最高位用 0 补充，把 CRC 寄存器的值与 A001H 进行异或计算，并将结果放到 CRC 寄存器中。

（5）如果 LSB 是 0，CRC 寄存器的各位向右移动一位，最高位用 0 补充。

（6）重复步骤 3、4、5，直到完成 8 次移位。

（7）重复步骤 2、3、4、5、6，处理发送信息的下一个字节。直到处理完发送信息的所有字节。

（8）计算完毕，CRC 寄存器的内容即为 CRC 校验的值。

（9）在时间资源有限的系统中，建议采用查表法来实现 CRC 校验。

CRC 简单函数如下(用 C 语言编程)：

```
unsigned int CRC_Cal_Value(unsigned char *Data, unsigned char Length)
{
      unsigned int crc_value = 0xFFFF;
      int i = 0;
      while(Length--)
      {
     crc_value ^= *Data++;
     for(i=0;i<8;i++)
     {
       if(crc_value & 0x0001)
       {
          crc_value = (crc_value>>1)^ 0xa001;
                  }
       else
       {
          crc_value = crc_value>>1;
                  }
     }
  }
  return(crc_value);
```

}

以上只为CRC校验理论阐述，运用此方法执行时间较长，特别是校验数据较长时，计算时间过长，故引用以下两种查表方法，分别针对16位和8位控制器。

- 8位处理器CRC16查表：（此程序最终返回结果为高字节在前，发送时请颠倒）

```
const Uint8 crc_l_tab[256] = {
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
};
constUint8 crc_h_tab[256] = {
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
};
Uint16CRC(Uint8 * buffer, Uint8 crc_len)
{
   Uint8    crc_i,crc_lsb,crc_msb;
   Uint16 crc;
   crc_msb = 0xFF;
   crc_lsb = 0xFF;
   while(crc_len--)
   {
      crc_i = crc_lsb ^ *buffer;
      buffer ++;
      crc_lsb = crc_msb ^ crc_l_tab[crc_i];
      crc_msb = crc_h_tab[crc_i];
   }
   crc = crc_msb;
   crc = (crc << 8) + crc_lsb;
   return crc;
}
```

- 16 位处理器 CRC16 查表：（此程序最终返回结果为高字节在前，发送时请颠倒）

```
const Uint16 crc_table[256] = {
0x0000,0xC1C0,0x81C1,0x4001,0x01C3,0xC003,0x8002,0x41C2,0x01C6,0xC006
,0x8007,0x41C7,0x0005,0xC1C5,0x81C4,0x4004,0x01CC,0xC00C,0x800D,0x41CD
,0x000F,0xC1CF,0x81CE,0x400E,0x000A,0xC1CA,0x81CB,0x400B,0x01C9,0xC009
```

```
,0x8008,0x41C8,0x01D8,0xC018,0x8019,0x41D9,0x001B,0xC1DB,0x81DA,0x401A
,0x001E,0xC1DE,0x81DF,0x401F,0x01DD,0xC01D,0x801C,0x41DC,0x0014,0xC1D4
,0x81D5,0x4015,0x01D7,0xC017,0x8016,0x41D6,0x01D2,0xC012,0x8013,0x41D3
,0x0011,0xC1D1,0x81D0,0x4010,0x01F0,0xC030,0x8031,0x41F1,0x0033,0xC1F3
,0x81F2,0x4032,0x0036,0xC1F6,0x81F7,0x4037,0x01F5,0xC035,0x8034,0x41F4
,0x003C,0xC1FC,0x81FD,0x403D,0x01FF,0xC03F,0x803E,0x41FE,0x01FA,0xC03A
,0x803B,0x41FB,0x0039,0xC1F9,0x81F8,0x4038,0x0028,0xC1E8,0x81E9,0x4029
,0x01EB,0xC02B,0x802A,0x41EA,0x01EE,0xC02E,0x802F,0x41EF,0x002D,0xC1ED
,0x81EC,0x402C,0x01E4,0xC024,0x8025,0x41E5,0x0027,0xC1E7,0x81E6,0x4026
,0x0022,0xC1E2,0x81E3,0x4023,0x01E1,0xC021,0x8020,0x41E0,0x01A0,0xC060
,0x8061,0x41A1,0x0063,0xC1A3,0x81A2,0x4062,0x0066,0xC1A6,0x81A7,0x4067
,0x01A5,0xC065,0x8064,0x41A4,0x006C,0xC1AC,0x81AD,0x406D,0x01AF,0xC06F
,0x806E,0x41AE,0x01AA,0xC06A,0x806B,0x41AB,0x0069,0xC1A9,0x81A8,0x4068
,0x0078,0xC1B8,0x81B9,0x4079,0x01BB,0xC07B,0x807A,0x41BA,0x01BE,0xC07E
,0x807F,0x41BF,0x007D,0xC1BD,0x81BC,0x407C,0x01B4,0xC074,0x8075,0x41B5
,0x0077,0xC1B7,0x81B6,0x4076,0x0072,0xC1B2,0x81B3,0x4073,0x01B1,0xC071
,0x8070,0x41B0,0x0050,0xC190,0x8191,0x4051,0x0193,0xC053,0x8052,0x4192
,0x0196,0xC056,0x8057,0x4197,0x0055,0xC195,0x8194,0x4054,0x019C,0xC05C
,0x805D,0x419D,0x005F,0xC19F,0x819E,0x405E,0x005A,0xC19A,0x819B,0x405B
,0x0199,0xC059,0x8058,0x4198,0x0188,0xC048,0x8049,0x4189,0x004B,0xC18B
,0x818A,0x404A,0x004E,0xC18E,0x818F,0x404F,0x018D,0xC04D,0x804C,0x418C
,0x0044,0xC184,0x8185,0x4045,0x0187,0xC047,0x8046,0x4186,0x0182,0xC042
,0x8043,0x4183,0x0041,0xC181,0x8180,0x4040};
Uint16 CRC16(Uint16 *msg , Uint16 len){
   Uint16 crcL = 0xFF , crcH = 0xFF;
   Uint16 index;
   while(len--){
      index = crcL ^ *msg++;
      crcL = ((crc_table[index] & 0xFF00) >> 8) ^ (crcH);
      crcH = crc_table[index] & 0xFF;
   }
```

```
    return (crcH<<8) | (crcL);
}
```

1.3.7 异常信息响应

当主站设备向从站设备发送请求时，主站希望得到一个正常的响应。主站的查询可能导致下列四种事件之一：

- 如果从站设备接收到无通信错误的请求，并且可以正常的处理询问，那么从站设备将返回一个正常的响应；
- 如果由于通信错误，从站设备没有接收到请求，那么不能返回信息。从站设备将视之为超时；
- 如果从站设备收到请求，但是检测到一个通信错误（奇偶校验、地址、帧错误等），那么不会返回响应。从站设备将视之为超时；
- 如果从站设备接收到无通信错误的请求，但是不能处理这个请求（如请求读一个不存在的寄存器等），从站将返回一个异常响应，通知主站错误的实际情况。

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，从站在相应的功能码域复制原始请求的功能码。所有功能码的 MSB 都为 0。在异常响应中，从站设置功能码的 MSB 为 1。即**异常响应功能码=正常响应功能码+0x80**
- 数据域：在正常响应中，从站可以在数据域中返回数据，在异常响应中从站在数据域中返回异常码。具体已定义异常码如表 4 异常码定义所示。

表 4 异常码定义

异常码	名称	含义
01H	非法功能	从站（变频器）接收到的功能码超出已配置范围（详见 1.3.3 功能码）
02H	非法数据地址	从站（变频器）接收到的数据地址是不允许的地址；特别是，寄存器起始地址和传输长度的组合是无效的（详见 1.3.4 寄存器地址分布）
03H	非法数据帧	从站（变频器）检测到询问数据帧长度或者 CRC 校验不对
04H	从设备故障	从站（变频器）试图执行请求操作时发生不可恢复差错，可能原因有逻辑错误或写 EEPROM 失败等
05H	数据超范围	从站（变频器）接收到的数据超出对应寄存器最小值～最大值范围
06H	参数只读	当前寄存器为只读，不能进行写操作
07H	参数运行中不可改	变频器处于运行状态，当前寄存器不能进行写操作，若需

		操作，请停机
08H	参数受密码保护	当前寄存器受密码保护

1.4 协议说明

1.4.1 帧间和帧内时间间隔定义

一个完整的 MODBUS 报文不仅包含必须的数据单元，也要有起始和结束标识。因此，如图 1 或图 3 所示，特定义大于等于 3.5 个字符传输时间的空闲电平作为起止标志，且在报文传输过程中若出现大于 1.5 个字符传输时间的空闲电平则认为传输异常。

具体起止和异常间隔时间与波特率相关，具体如表 5 所示。如波特率为 9600bps，采样周期为 1ms 时，则起止时间间隔为大于等于 4ms（3.5×10/9600=3.64≈4）的空闲电平，异常数据间隔时间为一帧数据各位之间间隔大于等于 2ms（1.5×10/9600＝1.56≈2）且小于 4ms 的空闲电平（则正常数据位之间的空闲电平小于等于 1ms）。

表 5 时间间隔与波特率对照表（$t_{调}$=1ms 时）

波特率（bps）	起止间隔时间 $T_{间}$（$t_{调}$）	异常间隔时间 $T_{异}$（$t_{调}$）	备注
4800	8	4	正常帧允许≤3ms 的空闲点电平，当出现≥8ms 的空闲电平则表明一帧数据结束
9600	4	2	正常帧允许≤1ms 的空闲点电平，当出现≥4ms 的空闲电平则表明一帧数据结束
19200	2	1	正常帧允许＜1ms 的空闲点电平，当出现≥2ms 的空闲电平则表明一帧数据结束
更高	**1**	**1**	**当出现 1ms 的空闲电平这表明一帧结束**

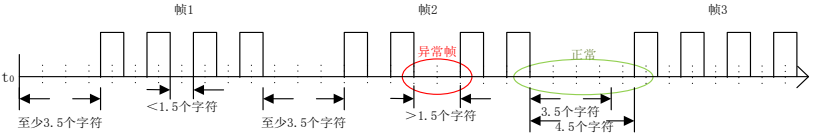


图 3 数据帧正误示意图

1.4.2 数据帧处理

接收完一帧数据后，系统要先进行预处理，判断是否为发给本机的合法帧，然后再判断数据是否正确，最后进行相应处理。若接收帧不是合法帧，则不会回发数据；若接收帧为合法帧，但是不正确，会回发相应异常信息帧。

合法帧：满足地址（本机或者广播）和长度（不小于 3）条件。

正确帧：为合法帧，且涉及内存地址正确、内存内容在定义范围内且当前可被处

理。

1.4.3 应答延时

定义变频器从接收到有效数据帧[6]（RS-485 网络上的数据，不同于键盘发送的指令），到解析数据，然后开始返回数据的时间间隔，为应答延时（由功能码 F10.04 设定）。因标准协议定义了起止符，故不可能没有应答延时，至少为“3.5 字符时间间隔+1ms（485 协议芯片稳定时间，$t_{等2}$）”，具体最短时间间隔与波特率相关。如波特率为 9600bps，最短应答延时为 5ms（3.5×10/9600+1=4.64≈5）。

若通讯数据涉及 EEPROM 操作，时间间隔会加长。

[6]:有效数据帧：由外部主站（不是键盘）发给本机，且功能码、数据长度和 CRC 都正确的数据。

图 4 中，数据发送段（$t_{发}$）、发送结束符段（$t_{等1}$）、75176 转发送等待段（$t_{等2}$）、数据返回段（$t_{返}$）和 75176 转接收等待段（$t_{等3}$）

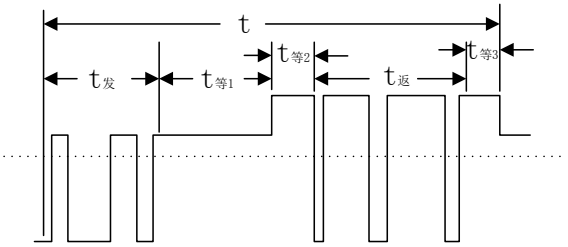


图 4 完整数据帧时序解析图

1.4.4 通讯超时

定义从站（变频器）从前一次接收到有效数据帧开始到下一次接收到有效数据帧结束时间间隔为通讯时间间隔△t，若△t 大于既定时间（功能码 F10.03 设定；若设为 0，则此功能无效），则认为通讯超时。

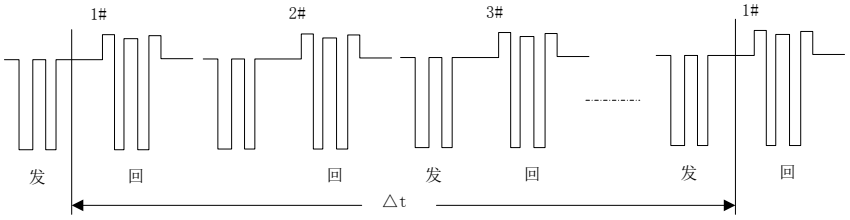


图 5 485 网络链路数据示意图

1.5 举例说明

1) 变频器正转运行

发：01 41 70 0000 01 E6 C5

回：01 41 70 0000 01 E6 C5（正常时）

回：01 C1 04 70 53（异常时，假设为从设备故障）

	发送		正常返回		异常返回	
*	帧头	≥3.5字符空闲				
1	地址	01	地址	01	地址	01
2	功能码	41	功能码	41	功能码	C1
3	寄存器地址Hi	70	寄存器地址Hi	70	异常码	04（假设）
4	寄存器地址Lo	00	寄存器地址Lo	00	CRC校验Lo	70
5	寄存器值Hi	00	寄存器值Hi	00	CRC校验Hi	53
6	寄存器值Lo	01	寄存器值Lo	01		
7	CRC校验Lo	E6	CRC校验Lo	E6		
8	CRC校验Hi	C5	CRC校验Hi	C5		
*	帧尾	≥3.5字符空闲				

2) 变频器自由停车

发：01 41 70 0000 07 66 C7

回：01 41 70 0000 07 66 C7（正常时）

回：01 C1 04 70 53（异常时，假设为从设备故障）

	发送		正常返回		异常返回	
*	帧头	≥3.5字符空闲				
1	地址	01	地址	01	地址	01
2	功能码	41	功能码	41	功能码	C1
3	寄存器地址Hi	70	寄存器地址Hi	70	异常码	04（假设）
4	寄存器地址Lo	00	寄存器地址Lo	00	CRC校验Lo	70
5	寄存器值Hi	00	寄存器值Hi	00	CRC校验Hi	53
6	寄存器值Lo	07	寄存器值Lo	07		

7	CRC校验Lo	66	CRC校验Lo	66	
8	CRC校验Hi	C7	CRC校验Hi	C7	
*	帧尾	≥3.5字符空闲			

3) 改变设定频率（如 50.00Hz/1388H）命令字（F00.04=7 时）

发：01 41 70 15 13 88 3B 97

回：01 41 70 15 13 88 3B 97（正常时）

回：01 C1 04 70 53（异常时，假设为从设备故障）

	发送		正常返回		异常返回	
*	帧头	≥3.5字符空闲				
1	地址	01	地址	01	地址	01
2	功能码	41	功能码	41	功能码	C1
3	寄存器地址Hi	70	寄存器地址Hi	70	异常码	04（假设）
4	寄存器地址Lo	15	寄存器地址Lo	15	CRC校验Lo	70
5	寄存器值Hi	13	寄存器值Hi	13	CRC校验Hi	53
6	寄存器值Lo	88	寄存器值Lo	88		
7	CRC校验Lo	3B	CRC校验Lo	3B		
8	CRC校验Hi	97	CRC校验Hi	97		
*	帧尾	≥3.5字符空闲				

1) 读取最近一次故障信息（读取 F19.00～F19.05 功能码）

发：01 03 13 00 00 06 C1 4C

回：01 03 0C 00 11 00 00 00 00 01 2C 00 00 00 0053 5B（正常时）

回：01 83 04 40 F3（异常时，假设为从设备故障）

	发送		正常返回		异常返回	
*	帧头	≥3.5字符空闲				
1	地址	01	地址	01	地址	01
2	功能码	03	功能码	03	功能码	83
3	起始地址 Hi	13	字节数	0C	异常码	04（假设）
4	起始地址 Lo	00	寄存器值 Hi（F19.00）	00	CRC校验Lo	40
5	寄存器数量 Hi	00	寄存器值 Lo（F19.00）	11	CRC校验Hi	F3
6	寄存器数量 Lo	06	寄存器值 Hi（F19.01）	00		
7	CRC校验Lo	C1	寄存器值 Lo（F19.01）	00		
8	CRC校验Hi	4C	寄存器值 Hi（F19.02）	00		
9			寄存器值 Lo（F19.02）	00		
10			寄存器值 Hi（F19.03）	01		
11			寄存器值 Lo（F19.03）	2C		
12			寄存器值 Hi（F19.04）	00		
13			寄存器值 Lo（F19.04）	00		

14		寄存器值 Hi（F19.05）	00	
15		寄存器值 Lo（F19.05）	00	
16		CRC校验Lo	53	
17		CRC校验Hi	5B	
*	帧尾	≥3.5字符空闲		

2） 检查线路是否连通

发：01 08 00 00 AA 55 5E 94

回：01 08 00 00 AA 55 5E 94（正常时）

回：01 88 04 47 C3（异常时，假设为从设备故障）

	发送		正常返回		异常返回	
*	帧头	≥3.5字符空闲				
1	地址	01	地址	01	地址	01
2	功能	08	功能	08	功能码	88
3	子功能码 Hi	00	子功能码 Hi	00	异常码	04（假设）
4	子功能码 Lo	00	子功能码 Lo	00	CRC校验Lo	47
5	数据 Hi	AA	数据 Hi	AA	CRC校验Hi	C3
6	数据 Lo	55	数据 Lo	55		
7	CRC校验Lo	5E	CRC校验Lo	5E		
8	CRC校验Hi	94	CRC校验Hi	94		
*	帧尾	≥3.5字符空闲				

3） 将载波频率（F00.23）改为 4.0kHz。(因为此类功能码一般改后希望存 EEPROM，故用 0x06 功能码)。

发：01 06 00 17 00 28 39 D0

回：01 06 00 17 00 28 39 D0（正常时）

回：01 86 04 43 A3（异常时，假设为从设备故障）

	发送		正常返回		异常返回	
*	帧头	≥3.5字符空闲				
1	地址	01	地址	01	地址	01
2	功能码	06	功能码	06	功能码	86
3	寄存器地址Hi	00	寄存器地址Hi	00	异常码	04（假设）
4	寄存器地址Lo	17	寄存器地址Lo	17	CRC校验Lo	43
5	寄存器值Hi	00	寄存器值Hi	00	CRC校验Hi	A3
6	寄存器值Lo	28	寄存器值Lo	28		
7	CRC校验Lo	39	CRC校验Lo	39		
8	CRC校验Hi	D0	CRC校验Hi	D0		
*	帧尾	≥3.5字符空闲				