Modbus

通讯协议

（RTU 传输模式）

本说明仅做内部参考，详细请参阅英文版本。

第一章　Modbus 协议简介

Modbus 协议是应用于电子控制器上的一种**通用语言**。通过此协议，控制器相互之间、控制器经由网络（例如以太网）和其它设备之间可以通信。它已经成为一通用工业标准。有了它，不同厂商生产的控制设备可以连成工业网络，进行集中监控。

此协议定义了一个控制器能认识使用的消息结构，而不管它们是经过何种网络进行通信的。它描述了一控制器请求访问其它设备的过程，如果回应来自其它设备的请求，以及怎样侦测错误并记录。它制定了消息域格局和内容的公共格式。

当在一 Modbus 网络上通信时，此协议决定了每个控制器须要知道它们的设备地址，识别按地址发来的消息，决定要产生何种行动。如果需要回应，控制器将生成反馈信息并用 Modbus 协议发出。在其它网络上，包含了 Modbus 协议的消息转换为在此网络上使用的帧或包结构。这种转换也扩展了根据具体的网络解决节地址、路由路径及错误检测的方法。

协议在一根通讯线上使用应答式连接（半双工），这意味着在一根单独的通讯线上信号沿着相反的两个方向传输。首先，主计算机的信号寻址到一台唯一的终端设备（从机），然后，在相反的方向上终端设备发出的应答信号传输给主机。 协议只允许在主计算机和终端设备之间，而不允许独立的设备之间的数据交换，这就不会在使它们初始化时占据通讯线路，而仅限于响应到达本机的查询信号。

1．1　传输方式

传输方式是一个信息帧内一系列独立的数据结构以及用于传输数据的有限规则，以 RTU 模式在 Modbus 总线上进行通讯时，信息中的每 8 位字节分成 2 个 4 位 16 进制的字符，每个信息必须连续传输下面定义了与 Modebus 协议– RTU 方式相兼容的传输方式。

代码系统

- 8 位二进制，十六进制数 0...9，A...F
- 消息中的每个 8 位域都是一个两个十六进制字符组成

每个字节的位

- 1 个起始位
- 8 个数据位，最小的有效位先发送
- 1 个奇偶校验位，无校验则无
- 1 个停止位（有校验时），2 个 Bit（无校验时）

错误检测域

- CRC(循环冗长检测)

1．2　协议

当信息帧到达终端设备时，它通过一个简单的“口”进入寻址到的设备，该设备去掉数据帧的“信封”（数据头），读取数据，如果没有错误，就执行数据所请求的任务，然后，它将自己生成的数据加入到取得的“信封”中，把数据帧返回给发送者。返回的响应数据中包含了以下内容：终端从机地址(Address)、被执行了的命令(Function)、执行命令生成的被请求数据(Data)和一个校验码(Check)。发生任何错误都不会有成功的响应。

1．2．1　信息帧

Address	Function	Data	Check
8-Bits	8-Bits	N x 8-Bits	16-Bits

图 1－1．信息帧格式

特注：Modbus 信息帧所允许的最大长度为 256 个字节，即 N 的范围是大于等于零且小于等于 252（N{0，252}）。

即，所有的数据一共 256 个，数据剩下 253 个。

1．2．2　地址（Address）域

信息帧地址域(信息地址)在帧的开始部分，由 8 位组成，有效的从机设备地址范围 0-247(十进制)，各从机设备的寻址范围为 1-247。主机把从机地址放入信息帧的地址区，并向从机寻址。从机响应时，把自己的地址放入响应信息的地址区，让主机识别已作出响应的从机地址。

地址 0 为广播地址，所有从机均能识别。当 Modbus 协议用于高级网络时，则不允许广播或其它方式替代。

1．2．3　功能（Function）域

信息帧功能域代码告诉了被寻址到的终端执行何种功能。有效码范围 1-225(十进制)，有些代码是适用于所有控制器，有些适应于某种控制器，还有些保留以备后用。有关功能代替码的全部内容见**附录 A**。

当主机向从句发送信息时，功能代码向从机说明应执行的动作。如读一组离散式线圈或输入信号的 ON/OFF 状态，读一组寄存器的数据，读从机的诊断状态，写线圈（或寄存器），允许下截、记录、确认从机内的程序等。当从机响应主机时，功能代码可说明从机正常响应或出现错误(即不正常响应)，正常响应时，从句简单返回原始功能代码；不正常响应时，从机返回与原始代码相等效的一个码，并把最高有效位设定为“1”。

如，主机要求从机读一组保持寄存器时，则发送信息的功能码为：

0000 0011 (十六进制 03)

若从机正确接收请求的动作信息后，则返回相同的代码值作为正常响应。发现错时，则返回一个不正常响信息：

1000 0011(十六进制 83)

从机对功能代码作为了修改，此外，还把一个特殊码放入响应信息的数据区中，告诉主机出现的错误类型和不正常响应的原因，不正常响应见**附录 B**。主机设备的应用程序负责处理不正常响应，典型处理过程是主机把对信息的测试和诊断送给从机，并通知操作者。表 1－1 列出了所有设备常用的功能码、它们的意义及它们的初始功能。

表 1－1 常用功能码

代码	名称	作用
01	读取线圈状态	取得一组逻辑线圈的当前状态（ON/OFF)
02	读取输入状态	取得一组开关输入的当前状态(ON/OFF)

03	读取保持寄存器	在一个或多个保持寄存器中取得当前的二进制值
04	读取输入寄存器	在一个或多个输入寄存器中取得当前的二进制值
05	强置单线圈	强置一个逻辑线圈的通断状态
06	预置单寄存器	放置一个特定的二进制值到一个单寄存器中
07	读取异常状态	取得 8 个内部线圈的通断状态
15	强置多线圈	强置一串连续逻辑线圈的通断
16	预置多寄存器	放置一系列特定的二进制值到一系列多寄存器中
17	报告从机标识	可使主机判断编址从机的类型及该从机运行指示灯的状态

1．2．4　数据域

数据域包含了终端执行特定功能所需要的数据或者终端响应查询时采集到的数据。这些数据的内容可能是数值、参考地址或者极限值。他由数据区有 2 个 16 进制的数据位（2 的 8 次方 256），数据范围为 00-FF(16 进制)。例如：功能域码告诉终端读取一个寄存器，数据域则需要指明从哪个寄存器开始及读取多少个数据，内嵌的地址和数据依照类型和从机之间的不同能力而有所不同。若无错误出现，从机向主机的响应信息中包含了请求数据，若有错误出现，则数据中有一个不正常代码，使主机能判断并作出下一步的动作。**数据区的长度可为“零”以表示某类信息。**

1．2．5　错误校验域

该域允许主机和终端检查传输过程中的错误。有时，由于电噪声和其它干扰，一组数据在从一个设备传输到另一个设备时在线路上可能会发生一些改变，出错校验能够保证主机或者终端不去响应那些传输过程中发生了改变的数据，这就提高了系统的安全性和效率，出错校验使用了 16 位循环冗余的方法，即 CRC 校验。

错误检测域包含一 16Bits 值(用两个 8 位的字符来实现)。错误检测域的内容是通过对消息内容进行循环冗长检测方法得出的。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的最后一个字节。

1．2．6　字符的连续传输

当消息在标准的 Modbus 系列网络传输时，每个字符或字节按由左到右的次序方式发送：

最低有效位（LSB）...最高有效位(MSB)。

位的序列是：

有奇偶校验

启始位	1	2	3	4	5	6	7	8	奇偶位	停止位

无奇偶校验

启始位	1	2	3	4	5	6	7	8	停止位	停止位

图 1 －2． 位顺序（RTU）

1．3 错误检测

1、**奇偶校验**

用户可以配置控制器是奇或偶校验，或无校验。这将决定了每个字符中的奇偶校验位是如何设置的。

如果指定了奇或偶校验，“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。如果便用了偶校验，帧的奇偶校验位将是 0，便得整个“1”的个数仍是 4 个。如果便用了奇校验，帧的奇偶校验位将是 1，便得整个“1”的个数是 5 个。

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。代替一附加的停止位填充至要传输的字符帧中。

2、CRC 检测

RTU 方式时，采用 CRC 方法计算错误校验码，CRC 校验传送的全部数据。它忽略信息中单个字符数据的奇偶校验方法。

循环冗余校验（CRC）域占用两个字节，包含了一个 16 位的二进制值。CRC 值由传送设备计算出来，然后附加到数据帧上，接收设备在接收数据时重新计算 CRC 值，然后与接收到的 CRC 域中的值进行比较，如果这两个值不相等，就发生了错误。

CRC 开始时先把寄存器的 16 位全部置成“1”，然后把相邻 2 个 8 位字节的数据放入当前寄存器中，只有每个字符的 8 位数据用作产生 CRC，起始位，停止位和奇偶校验位不加到 CRC 中。

在生成 CRC 时，每个 8 位字节与寄存器中的内容进行异或，然后将结果向低位移位，高位则用“0”补充，最低位（LSB）移出并检测，如果是 1，该寄存器就与一个预设的固定值进行一次异或运算，如果最低位为 0，不作任何处理。

上述处理重复进行，知道执行完了 8 次移位操作，当最后一位（第 8 位）移完以后，下一个 8 位字节与寄存器材的当前值进行异或运算，同样进行上述的另一个 8 次移位异或操作，当数据帧中的所有字节都作了处理，生成的最终值就是 CRC 值。

生成一个 CRC 的流程为：

1、 预置一个 16 位寄存器为 0FFFFH（全 1），称之为 CRC 寄存器。

2、 把数据帧中的第一个 8 位字节与 CRC 寄存器中的低字节进行异或运算，结果存回 CRC 寄存器。

3、 将 CRC 寄存器向右移一位，最高位填以 0，最低位移出并检测。

4、 如果最低位为 0：重复第 3 步（下一次移位）。

如果最低位为 1：将 CRC 寄存器与一个预设的固定值（0A001H）进行异或运算。

5、 重复第 3 步和第 4 步直到 8 次移位。这样处理完了一个完整的八位。

6、 重复第 2 步到第 5 步来处理下一个八位，直到所有的字节处理结束。

7、 最终 CRC 寄存器得值就是 CRC 的值。

CRC 值附加到信息时，低位在先，高位在后。查阅**附录 C** 中的一个实例，它详细说明了 CRC 的校验。

第二章　Modbus 数据和控制功能详解

Modbus 信息中的所有数据地址以零作为基准，各项数据的第一个数据地址的编号为 0。若无特殊说明在此节文中用＋进制值表示，图中的数据区则用十六进制表示。

图 2--1 为一个例子，说明了 Modbus 的查询信息，图 2--2 为正常响应的例子，这两例子中的数据均是 16 进制的，也表示了以 RTU 方式构成数据帧的方法。

主机查询是读保持寄存器，被请求的从机地址是 06，读取的数据来自地址 40108 保持

寄有器。注意，该信息规定了寄存器的起始地址为 0107 (006BH)。

从机响应返回该功能代码，说明是正常响应，字节数“Byle count”中说明有多少个 8 位字节被返回。它表明了附在数据区中 8 位字节的数量,当在缓冲区组织响应信息时，“字节数”区域中的值应与该信息中数据区的字节数相等。如 RTU 方式时，63H 用一个字节(01100011)发送。8 个位为一个单位计算“字节数”，它忽略了信息帧用组成的方法。

Addr	Fun	Data start reg hi	Data start reg lo	Data #of regs hi	Data #of regs lo	CRC16 hi	CRC16 lo
06H	03H	00H	6BH	00H	01H	XXH	XXH

图 2 – 1 Modbus 的查询信息

Addr	Fun	Byte count	Data1 hi	Data1 Lo	Data 2 hi	Data2 lo	Data3 hi	Data3 lo	CRC16 hi	CRC16 lo
06H	03H	06H	02H	2BH	00H	00H	00H	63H	XXH	XXH

图 2 – 2 Modbus 的响应信息

2．1 读取线圈状态（功能码 01）

读取从机离散量输出口（DO,0X 类型）的 ON/OFF 状态，不支持广播。

查询

查询信息规定了要读的起始线圈和线圈量，线圈的起始地址为 0000H，1-16 个线圈的寻址地址分为 0000H –0015H（DO1=0000H，DO2=0001H，依此类推）。

图 2 – 3 的例子是从地址为 17 的从机读取 DO1 至 DO6 的状态。

Addr	Fun	DO start reg hi	DO start reg lo	DO #of regs hi	DO #of regs lo	CRC16 hi	CRC16 lo
11H	01H	00H	00H	00H	06H	XXH	XXH

图 2 – 3 读取线圈状态----查询

响应

响应信息中的各线圈的状态与数据区的每一位的值相对应，即每个 DO 占用一位（1 = ON, 0 = OFF），第一个数据字节的 LSB 为查询中的寻址地址，其他的线圈按顺序在该字节中由低位向高位排列，直至 8 个为止，下一个字节也是从低位向高位排例。若返回的线圈数不是 8 的倍数，则在最后的数据字节中的剩余位至字节的最高位全部填 0，字节数区说明全部数据的字节数。

图 2 – 4 所示为线圈的输出状态响应的实例。

Addr	Fun	Byte count	Data	CRC16 hi	CRC16 lo
11H	01H	01H	2AH	XXH	XXH

数据

0	0	0	0	0	0	DO2	DO1
MSB	7	6	5	4	3	2	LSB

图 2 – 4 读取线圈状态----响应

2．2 读取输入状态（功能码 02）

读取从机离散量输入信号（DI,0X 类型）的 ON/OFF 状态，不支持广播。

查询

查询信息规定了要读的输入起始地址，以及输入信号的数量。输入的起始地址为 0000H，1-16 个输入口的地址分别为 0-15（DO1=0000H，DO2=0001H，依此类推）。

图 2－5 的例子是从地址为 17 的从机读取 DI1 到 DI16 的状态。

Addr	Fun	DI start addr hi	DI start addr lo	DI num hi	DI num lo	CRC16 hi	CRC16 lo
11H	02H	00H	00H	00H	10H	XXH	XXH

图 2－5 读取输入状态----查询

响应

响应信息中的各输入口的状态，分别对应于数据区中的每一位值，1 = ON; 0 = OFF，第一个数据字节的 LSB 为查询中的寻址地址，其他输入口按顺序在该字节中由低位向高位排列，直至 8 个位为止。下一个字节中的 8 个输入位也是从低位到高位排列。若返回的输入位数不是 8 的倍数，则在最后的数据字节中的剩余位直至字节的最高位全部填零。字节数区说明了全部数据的字节数。

图 2－6 所示为读数字输出状态响应的实例。

Addr	Fun	Byte coun	Data1	Data2	CRC16 hi	CRC16 lo
11H	02H	02H	33H	CCH	XXH	XXH

数据 1

DI8	DI7	DI6	DI5	DI4	DI3	DI2	DI1

MSB LSB

数据 2

DI16	DI15	DI14	DI13	DI12	DI11	DI10	DI9

MSB LSB

图 2－6 读取输入状态----响应

2．3 读取保持寄存器（功能码 03）

读取从机保持寄存器（4X 类型）的二进制数据，不支持广播。

查询

查询信息规定了要读的保持寄存器起始地址及保持寄存器的数量，保持寄存器寻址起始地址为 0000H，寄存器 1-16 所对应的地址分别为 0000H –0015H。

图 2－7 的例子是从 17 号从机读 3 个采集到的基本数据 U1、U2、U3，U1 的地址为 0000H，U2 的地址为 0001H，U3 的地址为 0002H。

Addr	Fun	Data start addr hi	Data start addr lo	Data #of regs hi	Data #of regs lo	CRC16 hi	CRC16 lo
11H	03H	00H	00H	00H	03H	XXH	XXH

图 2－7 读取保持寄存器----查询

响应

响应信息中的寄存器数据为二进制数据，每个寄存器分别对应 2 个字节，第一个字节为高位值数据，第二个字节为低位数据。

图 2 – 8 的例子是读取 U1,U2,U3(U1=03E8H,U2=03E7H,U3=03E9H)的响应。

Addr	Fun	Byte count	Data1 hi	Data1 Lo	Data 2 hi	Data2 lo	Data3 hi	Data3 lo	CRC16 hi	CRC16 lo
11H	03H	06H	03H	E8H	03H	E7H	03H	E9H	XXH	XXH

图 2 – 8 读取保持寄存器----响应

2.4 读取输入寄存器（功能码 04）

读取从机输入寄存器(3X 类型)中的二进制数据，不支持广播。

查询

查询信息规定了要读的寄存器的起始地址及寄存器的数量，寻止起始地址为 0，寄存器 1-16 所对应的地址分别为 0000H –0015H。

图 2 – 9 的例子是请求 17 号从机的 0009 寄存器。

Addr	Fun	DO addr hi	DO addr lo	Data #of regs hi	Data #of regs lo	CRC16 hi	CRC16 lo
11H	04H	00H	08H	00H	01H	XXH	XXH

图 2 – 9 读取输入寄存器----查询

响应

响应信息中的寄存器数据为每个寄存器分别对应 2 个字节，第一个字节为高位数据，第二个字节为低位数据。

图 2 – 10 的例子寄存器 30009 中的数据用 000AH 2 个字节表示。

Addr	Fun	Byte count	Data hi	Data Lo	CRC16 hi	CRC16 lo
11H	04H	02H	00H	0AH	XXH	XXH

图 2 – 10 读取输入寄存器----响应

2.5 强置单线圈（功能码 05）

强制单个线圈(DO，0X 类型)为 ON 或 OFF 状态，广播时，该功能可强制所有从机中同一类型的线圈均为 ON 或 OFF 状态。

该功能可越过控制器内存的保护状态和线圈的禁止状态。线圈强制状态一直保持有效直至下一个控制逻辑作用于线圈为止。控制逻辑中无线圈程序时，则线圈处于强制状态。

查询

查询信息规定了需要强制一个单独线圈的类型，线圈的起始地址为 0000H，1-16 个线圈的寻址地址分为 0000H –0015H（DO1=0000H，DO2=0001H，依此类推）。

由查询数据区中的一个常量，规定被请求线圈的 ON/OFF 状态， FF00H 值请求线圈处于 ON 状态，0000H 值请求线圈处于 OFF 状态，其它值对线圈无效，不起作用。

图示 2-11 的例子是请求 17 号从机开 DO1 的 On 状态。

Addr	Fun	DO addr hi	DO addr lo	Value hi	Value lo	CRC16 hi	CRC16 lo
11H	05H	00H	00H	FFH	00H	XXH	XXH

图示 2-11 强制单线圈----查询

响应

图 2 – 12 所示为对这个命令请求的正常响应是在 DO 状态改变以后传送接收到的数据。

Addr	Fun	DO addr hi	DO addr lo	Value hi	Value lo	CRC16 hi	CRC16 lo
11H	05H	00H	00H	FFH	00H	XXH	XXH

图示 2-12 强制单线圈----响应

2．6 预置单寄存器（功能码 06）

把一个值预置到一个保持寄存器（4X 类型）中，广播时，该功能把值预置到所有从机的相同类型的寄存器中。

该功能可越过控制器的内存保护。使寄存器中的预置值保持有效。只能由控制器的下一个逻辑信号来处理该预置值。若控制逻辑中无寄存器程序时，则寄存器中的值保持不变。

查询

查询信息规定了要预置寄存器的类型，寄存器寻址起始地址为 0000H，寄存器 1 所对应的地址为 0000H。

图示 2-13 的例子是请求 17 号从机 0040H.的值为 2717。

Addr	Fun	Data start reg hi	Data start reg lo	Value hi	Value lo	CRC hi	CRC lo
11H	06H	00H	40H	0AH	9DH	XXH	XXH

图示 2-13 预设单寄存器----查询

响应

图 2 – 14 所示对于预置单寄存器请求的正常响应是在寄存器值改变以后将接收到的数据传送回去。

Addr	Fun	Data start reg hi	Data start reg lo	Value hi	Value lo	CRC hi	CRC lo
11H	06H	00H	40H	0AH	9DH	XXH	XXH

图示 2-14 预设单寄存器----响应

2．7 读取异常状态（功能码 7）

读从中机中 8 个不正常状态线圈的数据，某些线圈号已在不同型号的控制器中预定义，而其它的线圈由用户编程，作为有关控制器的状态信息，如“machine ON/OFF”，“heads retraced”，(缩回标题)，“safeties satisfied”(安全性满意)，“error conditions”(存在错误条件)或其它用户定义的标志等。该功能码不支持广播。

该功能代码为存取该类信息提供了一种简单的方法，不正常线圈的类型是已知的(在功能代码中不需要线圈类型) 预定义的不正常线圈号如下：

控制器型号	线圈	设定
M84,184/384,584,984	1-8	用户定义
484	257	电池状态
	258-264	用户定义
884	761	电池状态
	762	内存保护状态
	763	R10 工况状态
	764-768	用户预定义

查询

图示 2-15 的例子是请求读从机设备 17 中的不正常状态。

Addr	Fun	CRC16 hi	CRC16 lo
11H	07H	XXH	XXH

图示 2-15 读取异常状态----查询

响应

正常响应包含 8 个不正常的线圈状态，为一个数据字节，每个线圈一位。LSB 对应为最低线圈类型的状态。

图 2 – 16 所示按查询要求返回响应：

Addr	Fun	DO Data	CRC16 hi	CRC16 lo
11H	07H	6DH	XXH	XXH

图示 2-16 读取异常状态----响应

该例子中，线圈数据为 6DH (二进制 0110 ,1101)，从左到右 (最高位至最低位) 的线圈状态分别为: OFF – ON – ON – OFF – ON – ON – OFF – ON。若控制器型号为 984，这些位表示线圈 8 至 1 的状态；若控制器型号为 484 则表示线圈 264 至 257 的状态。

2．8 强置多线圈（功能码 15）

按线圈的顺序把各线圈 (DO，0X 类型) 强制成 ON 或 OFF。广播时，该功能代码可对各从机中相同类型的线圈起强制作用。

该功能代码可越过内存保护和线圈的禁止状态线圈。保持强制状态有效，并只能由控制器的下一个逻辑来处理。若无线圈控制逻辑程序时，线圈将保持强制状态。

查询

查询信息规定了被强制线圈的类型，线圈的起始地址为 0000H，1-16 个线圈的寻址地址分为 0000H –0015H（DO1=0000H，DO2=0001H，依此类推）。

查询数据区规定了被请求线圈的 ON/OFF 状态，如数据区的某位值为“1”表示请求的相应线圈状态为 ON，位值为“0”，则为 OFF 状态。

图示 2-17 例子为请求从机设备 17 中一组 10 个线圈为强制状态，起始线圈为 20 (则寻址地址为 19 或 13H)，查询的数据为 2 个字节，CD01H (二进制 11001101 0000 0001) 相应线圈的二进制位排列如下：

Bit:	1	1	0	0	1	1	0	1	0	0	0	0	0	0	0	1
Coll:	27	26	25	24	23	22	21	20	-	-	-	-	-	-	29	28

传送的第一个字节 CDH 对应线圈为 27-20, LSB 对应线圈 20，传送的第二个字节为 01H，对应的线圈为 29-28， LSB 为继圈 28，其余未使用的位均填“0”。

Addr	Fun	DO addr hi	DO addr lo	Data #of reg hi	Data #of reg lo	Byte count	Value hi	Value lo	CRC hi	CRC lo
11H	0FH	00H	13H	00H	0AH	02H	CDH	01H	XXH	XXH

图示 2-17 强置多线圈----查询

响应

正常响应返回从机地址，功能代码，起始地址以及强制线圈数。

图 2 – 18 对上述查询返回的响应。

Addr	Fun	DO addr hi	DO addr lo	Data #of reg hi	Data #of reg lo	CRC16 hi	CRC16 lo
11H	0FH	00H	13H	00H	0AH	XXH	XXH

图示 2-18 强置多线圈----响应

2．9 预置多寄存器（功能码 16）

把数据按顺序预置到各 (4X 类型) 寄存器中，广播时该功能代码可把数据预置到全部从机中的相同类型的寄存器中。

该功能代码可越过控制器的内存保护，在寄存器中的预置值一直保持有效，只能由控制器的下一个逻辑来处理寄存器的内容，控制逻辑中无该寄存器程序时，则寄存器中的值保持不变。

查询

查询信息规定了要预置寄存器的类型，寄存器寻址起始地址为 0000H，寄存器 1 所对应的地址为 0000H。

图示 2-19 的例子是请求 17 号从机 0040H.的值为 178077833。

Addr	Fun	Data start reg hi	Data start reg lo	Data #of reg hi	Data #of reg lo	Byte count	Value hi	Value lo	Value hi	Value lo	CRC hi	CRC lo
11H	10H	00H	40H	00H	02H	04H	40H	89H	0AH	9DH	XXH	XXH

图示 2-19 预设多寄存器----查询

响应

图 2 – 20 所示对于预置单寄存器请求的正常响应是在寄存器值改变以后将接收到的数据传送回去。

Addr	Fun	Data start reg hi	Data start reg lo	Data #of reg hi	Data #of reg lo	CRC16 hi	CRC16 lo
11H	10H	00H	40H	00H	02H	XXH	XXH

图示 2-20 预设多寄存器----响应

2．10 报告从机标识（功能码 17）

返回一个从机地址控制器的类型，从机的当前状态，以及有关从机的其他说明，不支持广播。

查询

图示 2-21 的例子是请求报告从机设备 17 的 标识 ID 和状态。

Addr	Fun	CRC16 hi	CRC16 lo
11H	11H	XXH	XXH

图示 2-21 报告从机标识----查询

响应

图 2 – 22 所示正常响应格式，数据内容对应每台控制器的类型。

Addr	Fun	Byte Count	Slave ID	Run Indicator Status	Additfional Data	CRC16 hi	CRC16 lo
11H	11H	XXH	XXH	XXH	XXH	XXH	XXH

图示 2-22 报告从机标识----响应

从机 ID 总结

数据区第一个字节为 Modicon 控制器返回的从机 ID

Slave ID	Controller
0	Micro 84
1	484
2	184/384
3	584
8	884
9	984

特注：详细信息见 Modbus 协议英文版或中文版。

第三章 附录

附录 A：MODBUS 全部功能码

ModBus 网络是一个工业通信系统，由带智能终端的可编程序控制器和计算机通过公用线路或局部专用线路连接而成。其系统结构既包括硬件、亦包括软件。它可应用于各种数据采集和过程监控。下表 3--1 是 ModBus 的功能码定义。

表 3--1 ModBus 功能码

功能码	名称	作用
01	读取线圈状态	取得一组逻辑线圈的当前状态（ON/OFF)
02	读取输入状态	取得一组开关输入的当前状态（ON/OFF)
03	读取保持寄存器	在一个或多个保持寄存器中取得当前的二进制值
04	读取输入寄存器	在一个或多个输入寄存器中取得当前的二进制值
05	强置单线圈	强置一个逻辑线圈的通断状态
06	预置单寄存器	把具体二进值装入一个保持寄存器

07	读取异常状态	取得 8 个内部线圈的通断状态，这 8 个线圈的地址由控制器决定，用户逻辑可以将这些线圈定义，以说明从机状态，短报文适宜于迅速读取状态
08	回送诊断校验	把诊断校验报文送从机，以对通信处理进行评鉴
09	编程（只用于 484）	使主机模拟编程器作用，修改 PC 从机逻辑
10	控询（只用于 484）	可使主机与一台正在执行长程序任务从机通信，探询该从机是否已完成其操作任务，仅在含有功能码 9 的报文发送后，本功能码才发送
11	读取事件计数	可使主机发出单询问，并随即判定操作是否成功，尤其是该命令或其他应答产生通信错误时
12	读取通信事件记录	可是主机检索每台从机的 ModBus 事务处理通信事件记录。如果某项事务处理完成，记录会给出有关错误
13	编程（184/384 484 584）	可使主机模拟编程器功能修改 PC 从机逻辑
14	探询（184/384 484 584）	可使主机与正在执行任务的从机通信，定期控询该从机是否已完成其程序操作，仅在含有功能 13 的报文发送后，本功能码才得发送
15	强置多线圈	强置一串连续逻辑线圈的通断
16	预置多寄存器	把具体的二进制值装入一串连续的保持寄存器
17	报告从机标识	可使主机判断编址从机的类型及该从机运行指示灯的状态
18	（884 和 MICRO 84）	可使主机模拟编程功能，修改 PC 状态逻辑
19	重置通信链路	发生非可修改错误后，是从机复位于已知状态，可重置顺序字节
20	读取通用参数（584L）	显示扩展存储器文件中的数据信息
21	写入通用参数（584L）	把通用参数写入扩展存储文件，或修改之
22～64	保留作扩展功能备用	
65～72	保留以备用户功能所用	留作用户功能的扩展编码
73～119	非法功能	
120～127	保留	留作内部作用
128～255	保留	用于异常应答

ModBus 网络只是一个主机，所有通信都由他发出。网络可支持 247 个之多的远程从属控制器，但实际所支持的从机数要由所用通信设备决定。采用这个系统，各 PC 可以和中心主机交换信息而不影响各 PC 执行本身的控制任务。表 3--2 是 ModBus 各功能码对应的数据类型。

表 3--2 ModBus 功能码与数据类型对应表

代码	功能	数据类型
01	读	位
02	读	位
03	读	整型、字符型、状态字、浮点型
04	读	整型、状态字、浮点型
05	写	位
06	写	整型、字符型、状态字、浮点型
08	N/A	重复“回路反馈”信息
15	写	位
16	写	整型、字符型、状态字、浮点型
17	读	字符型

附录 B：不正常响应

不正常响应：

除广播外，主机向从机设备发送查询并希望有一个正常响应，主机查询中有可能产生 4 种事件：

- 从机接收查询，通讯错误正常处理信息，则返回一个正常响应事件。
- 由于通讯出错，从机不能接收查询数据，因而不返回响应。此时，主机依靠处理程序给出查询超时事件。
- 若从机接收查询，发现有（LRC 或 CRC）通讯错误，并返回响应，此时，依靠主机处理程序给出查询超时事件。
- 从机接收查询，无通讯错误，但无法处理(如读不存在的线圈和寄存器)时，向主机报告错误的性质。

不正常响应信息有 2 个与正常响应不相同的区域：

功能代码区：正常响应时，从机的响应功能代码区，带原查询的功能代码。所有功能代码的 MSB 为 0(其值低于 80H)。不正常响应时，从机把功能代码的 MSB 置为 1，使功能代码值大于 80H，高于正常响应的值。这样，主机应用程序能识别不正常响应事件，能检查不正常代码的数据区。

数据区：正常响应中，数据区含有(按查询要求给出的）数据或统计值，在不正常响应中，数据区为一个不正常代码，它说明从机产生不正常响应的条件和原因。

例：主机发出查询，从机不正常响应。（为十六进制数据）。

查询：

Addr	Fun	DO start reg hi	DO start reg lo	DO #of regs hi	DO #of regs lo	CRC16 Hi	CRC16 Lo
0AH	01H	04H	A1H	00H	01H	XXH	XXH

响应（不正常或例外）：

Addr	Fun	Exception Code	CRC16 Hi	CRC16 Lo
0AH	**81H**	02H	XXH	XXH

图 3－1．不正常信息帧格式

上例中，从机设备地址 10(0AH)，读线圈状态的功能代码(01)，主机请求线圈状态的地址为 1245(04A1H)。注意：只读一个指定线圈，地址为(0001).

若从机中不存在此线圈地址时，即以不正常代码(02)，向主机返回一个不正常响应。说明为不合法地址。

表 3--3 ModBus 的不正常代码：

代码	名称	含义
01	不合法功能代码	从机接收的是一种不能执行功能代码。发出查询命令后，该代码指示无程序功能。
02	不合法数据地址	接收的数据地址，是从机不允许的地址。
03	不合法数据	查询数据区的值是从机不允许的值。
04	从机设备故障	从机执行主机请求的动作时出现不可恢复的错误。
05	确认	从机已接收请求处理数据，但需要较长的处理时间，为避免主机出现超时错误而发送该确认响应。主机以此再发送一个“查询程序完成”未决定从机是否已完成处理。
06	从机设备忙碌	从机正忙于处理一个长时程序命令，请求主机在从机空闲时发送信息。
07	否定	从机不能执行查询要求的程序功能时，该代码使用十进制 13 或 14 代码，向主机返回一个“不成功的编程请求”信息。主机应请求诊断从机的错误信息。
08	内存奇偶校验错误	从机读扩展内存中的数据时，发现有奇偶校验错误，主机按从机的要求重新发送数据请求。

附录 C：CRC 校验生成程序

CRC 简单函数如下：

```
unsigned short CRC16(puchMsg, usDataLen)
unsigned char *puchMsg ; /* 要进行 CRC 校验的消息 */
unsigned short usDataLen ; /* 消息中字节数 */
{
unsigned char uchCRCHi = 0xFF ; /* 高 CRC 字节初始化 */
unsigned char uchCRCLo = 0xFF ; /* 低 CRC 字节初始化 */
unsigned uIndex ; /* CRC 循环中的索引 */
while (usDataLen--) /* 传输消息缓冲区 */
{
uIndex = uchCRCHi ^ *puchMsgg++ ; /* 计算 CRC */
uchCRCHi = uchCRCLo ^ auchCRCHi[uIndex} ;
uchCRCLo = auchCRCLo[uIndex] ;
```

```
    }
    return (uchCRCHi << 8 | uchCRCLo) ;
    }
/* CRC 高位字节值表 */
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
    } ;
/* CRC 低位字节值表*/
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
} ;
```

//

功能码	描述	是否支持广播	起始地址	备注
01	读线圈状态 DO	不支持	0000H	读**离散量输出口(0X 类型)**状态
02	读输入位状态 DI	不支持	0000H	读**离散量输入信号(0X 类型)**状态
03	读保持寄存器	不支持	0000H	读**保持寄存器**数据
04	读输入寄存器	不支持	0000H	读**输入寄存器(3X 类型)**数据
05	强制单个线圈 DO	支持	0000H	强制**单个线圈(0X 类型)**状态
06	预置单个保持寄存器	支持	0000H	置**保持寄存器（4X 类型）**中
07	读不正常状态	不支持		
08	诊断(见第 3 章)	不支持	0000H	
09	程序 484	不	0000H	没查到
10	查询 484	不	0000H	没查到
11	通讯事件控制	不支持		
12	通讯事件记录	不支持		
13	程序控制器	不	0000H	没查到
14	查询控制器	不	0000H	没查到
15	强制多个线圈 DO	支持	0000H	**强制各线圈 (0X 类型)**状态

16	预置多个保持寄存器	支持	0000H	置**保持寄存器（4X 类型）**中
17	报告从机 ID	不支持		
18	程序 884/M84	不	0000H	没查到
19	通讯链路复位	不	0000H	没查到
20	读通用参考值	不支持	0000H	**扩展寄存器（6X 类型）**
21	写通用参考值	不支持	0000H	**扩展寄存器（6X 类型）**
22	掩码写入 4X 类型寄存器	不支持	0000H	**保持寄存器（4X 类型）**
23	读/写 4X 类型寄存器	不支持	0000H	**保持寄存器（4X 类型）**
24	读 FIFO 查询数据	不支持	0000H	**保持寄存器（4X 类型）**