Modbus 通信协议

摘　要：工业控制已从单机控制走向集中监控、集散控制，如今已进入网络时代，工业控制器连网也为网络管理提供了方便。Modbus 就是工业控制器的网络协议中的一种。

关键词：Modbus 协议；串行通信；LRC 校验；CRC 校验；RS-232C

一、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。

1、在 Modbus 网络上转输

标准的 Modbus 口是使用一 RS-232C 兼容串行接口，它定义了连接口的针脚、电缆、信号位、传输波特率、奇偶校验。控制器能直接或经由 Modem 组网。

控制器通信使用主—从技术，即仅一设备（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。典型的主设备：主机和可编程仪表。典型的从设备：可编程控制器。

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。如果单独通信，从设备返回一消息作为回应，如果是以广播方式查询的，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、一错误检测域。

从设备回应消息也由 Modbus 协议构成，包括确认要行动的域、任何要返回的数据、和一错误检测域。如果在消息接收过程中发生一错误，或从设备不能执行其命令，从设备将建立一错误消息并把它作为回应发送出去。

2、在其它类型网络上转输

在其它网络上，控制器使用对等技术通信，故任何控制都能初始和其它控制器的通信。这样在单独的通信过程中，控制器既可作为主设备也可作为从设备。提供的多个内部通道可允许同时发生的传输进程。

在消息位，Modbus 协议仍提供了主—从原则，尽管网络通信方法是“ 对等 ”。如果一控制器发送一消息，它只是作为主设备，并期望从从设备得到回应。同样，当控制器接收到一消息，它将建立一从设备回应格式并返回给发送的控制器。

3、查询—回应周期

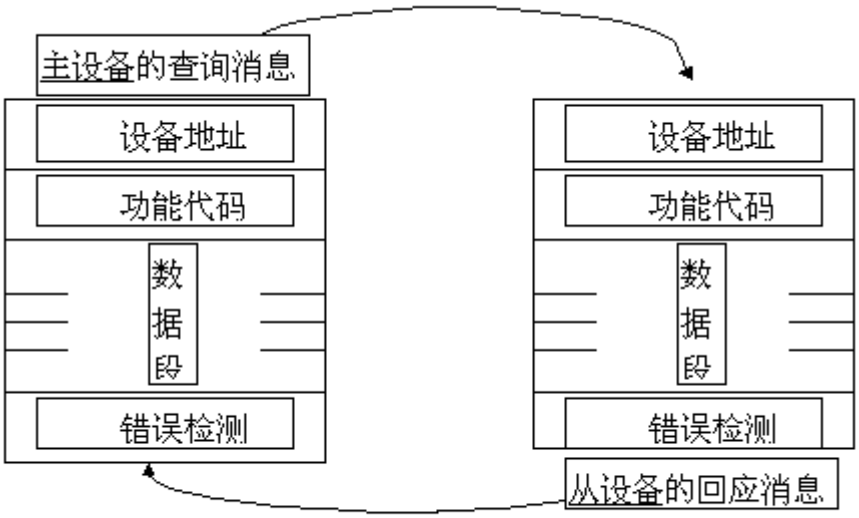

图 1 主一从 查询一回应周期表

（1）查询

查询消息中的功能代码告之被选中的从设备要执行何种功能。数据段包含了从设备要执行功能的任何附加信息。例如功能代码 03 是要求从设备读保持寄存器并返回它们的内容。数据段必须包含要告之从设备的信息：从何寄存器开始读及要读的寄存器数量。错误检测域为从设备提供了一种验证消息内容是否正确的方法。

（2）回应

如果从设备产生一正常的回应，在回应消息中的功能代码是在查询消息中的功能代码的回应。数据段包括了从设备收集的数据：象寄存器值或状态。如果有错误发生，功能代码将被修改以用于指出回应消息是错误的，同时数据段包含了描述此错误信息的代码。错误检测域允许主设备确认消息内容是否可用。

二、两种传输方式

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 网络通信。用户选择想要的模式，包括串口通信参数（波特率、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。

ASCII 模式

:	地址	功能代码	数据数量	数据 1	...	数据 n	LRC 高字节	LRC 低字节	回车	换行

RTU 模式

地址	功能代	数据数	数据 1	...	数据 n	CRC 高字	CRC 低字

	码	量				节	节

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。

在其它网络上（象 MAP 和 Modbus Plus）Modbus 消息被转成与串行传输无关的帧。

1、ASCII 模式

当控制器设为在 Modbus 网络上以 ASCII（美国标准信息交换代码）模式通信，在消息中的每个 8Bit 字节都作为两个 ASCII 字符发送。这种方式的主要优点是字符发送的时间间隔可达到 1 秒而不产生错误。

代码系统

- 十六进制，ASCII 字符 0...9，A...F
- 消息中的每个 ASCII 字符都是一个十六进制字符组成

每个字节的位

- 1 个起始位
- 7 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

- LRC(纵向冗长检测)

2、RTU 模式

当控制器设为在 Modbus 网络上以 RTU（远程终端单元）模式通信，在消息中的每个 8Bit 字节包含两个 4Bit 的十六进制字符。这种方式的主要优点是：在同样的波特率下，可比 ASCII 方式传送更多的数据。

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)

三、Modbus 消息帧

两种传输模式中（ASCII 或 RTU），传输设备以将 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），判知何时信息已完成。部分的消息也能侦测到并且错误能设置为返回结果。

1、ASCII 帧

使用 ASCII 模式，消息以冒号（:）字符（ASCII 码 3AH）开始，以回车换行符结束（ASCII 码 0DH,0AH）。

其它域可以使用的传输字符是十六进制的 0...9,A...F。网络上的设备不断侦测“：”字符，当有一个冒号接收到时，每个设备都解码下个域（地址域）来判断是否发给自己的。

消息中字符间发送的时间间隔最长不能超过 1 秒，否则接收的设备将认为传输错误。一个典型消息帧如下所示：

起始位	设备地址	功能代码	数据	LRC 校验	结束符
1 个字符	2 个字符	2 个字符	n 个字符	2 个字符	2 个字符

图 2 ASCII 消息帧

2、RTU 帧

使用 RTU 模式，消息发送至少要以 3.5 个字符时间的停顿间隔开始。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)。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，包括停顿间隔时间内。当第一个域（地址域）接收到，每个设备都进行解码以判断是否发往自己的。在最后一个传输字符之后，一个至少 3.5 个字符时间的停顿标定了消息的结束。一个新的消息可在此停顿后开始。

整个消息帧必须作为一连续的流转输。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。同样地，如果一个新消息在小于 3.5 个字符时间内接着前个消息开始，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	16Bit	T1-T2-T3-T4

图 3 RTU 消息帧

3、地址域

消息帧的地址域包含两个字符（ASCII）或 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。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。

地址 0 是用作广播地址，以使所有的从设备都能认识。当 Modbus 协议用于更高水准的网络，广播可能不允许或以其它方式代替。

4、如何处理功能域

消息帧中的功能代码域包含了两个字符（ASCII）或 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，有此是应用于某种控制器，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，但最重要的位置为逻辑 1。

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 03H）

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 83H）

除功能代码因异议错误作了修改外，从设备将一独特的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。

5、数据域

数据域是由两个十六进制数集合构成的，范围 00...FF。根据网络传输模式，这可以是由一对 ASCII 字符组成或由一 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：从设备必须用于进行执行由功能代码所定义的所为。这包括了象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。

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 03），数据域指定了起始寄存器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 10 十六进制），数据域则指明了要写的起始寄存器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 0B 十六进制），从设备不需任何附加的信息。

6、错误检测域

标准的 Modbus 网络有两种错误检测方法。错误检测域的内容视所选的检测方法而定。

ASCII

当选用 ASCII 模式作字符帧，错误检测域包含两个 ASCII 字符。这是使用 LRC（纵向冗长检测）方法对消息内容计算得出的，不包括开始的冒号符及回车换行符。LRC 字符附加在回车换行符前面。

RTU

当选用 RTU 模式作字符帧，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。

7、字符的连续传输

当消息在标准的 Modbus 系列网络传输时，每个字符或字节以如下方式发送（从左到右）：

最低有效位...最高有效位

使用 ASCII 字符帧时，位的序列是：

有奇偶校验

启始位	1	2	3	4	5	6	7	奇偶位	停止位

无奇偶校验

启始位	1	2	3	4	5	6	7	停止位	停止位

图 4．位顺序（ASCII）

使用 RTU 字符帧时，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 4．位顺序（RTU）

四、错误检测方法

标准的 Modbus 串行网络采用两种错误检测方法。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为正常反应。如果从设备测到一传输错误，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。

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、LRC 检测

使用 ASCII 模式，消息包括了一基于 LRC 方法的错误检测域。LRC 域检测了消息域中除开始的冒号及结束的回车换行号外的内容。

LRC 域是一个包含一个 8 位二进制值的字节。LRC 值由传输设备来计算并放到消息帧中，接收设备在接收消息的过程中计算 LRC，并将它和接收到消息中 LRC 域中的值比较，如果两值不等，说明有错误。

LRC 方法是将消息中的 8Bit 的字节连续累加，丢弃了进位。

LRC 简单函数如下：

```
static unsigned char LRC(auchMsg,usDataLen)
unsigned char *auchMsg ; /* 要进行计算的消息 */
unsigned short usDataLen ; /* LRC 要处理的字节的数量*/
{ unsigned char uchLRC = 0 ; /* LRC 字节初始化 */
while (usDataLen--) /* 传送消息 */
uchLRC += *auchMsg++ ; /* 累加*/
return ((unsigned char)(-((char_uchLRC))) ;
}
```

3、CRC 检测

使用 RTU 模式，消息包括了一基于 CRC 方法的错误检测域。CRC 域检测了整个消息的内容。

CRC 域是两个字节，包含一 16 位的二进制值。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 是先调入一值是全“ 1 ”的 16 位寄存器，然后调用一过程将消息中连续的 8 位字节各当前寄存器中的值进行处理。仅每个字符中的 8Bit 数据对 CRC 有效，起始位和停止位以及奇偶校验位均无效。

CRC 产生过程中，每个 8 位字符都单独和寄存器内容相或（OR），结果向最低有效位方向移动，最高有效位以 0 填充。LSB 被提取出来检测，如果 LSB 为 1，寄存器单独和预置的值或一下，如果 LSB 为 0，则不进行。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都执行之后的 CRC 值。

CRC 添加到消息中时，低字节先加入，然后高字节。

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
```

};

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 1 是 ModBus 的功能码定义。

表 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 2 是 ModBus 各功能码对应的数据类型。

表 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

（1）ModBus 的传输方式

在 ModBus 系统中有 2 种传输模式可选择。这 2 种传输模式与从机 PC 通信的能力是同等的。选择时应视所用 ModBus 主机而定，每个 ModBus 系统只能使用一种模式，不允许 2 种模式混用。一种模式是 ASCII（美国信息交换码），另一种模式是 RTU（远程终端设备）这两种模式的定义见表 3

表 3 ASCII 和 RTU 传输模式的特性

特性		ASCII(7 位)	RTU(8 位)
编码系统		十六进制（使用 ASCII 可打印字符：0～9，A～F)	二进制
每一个字符的位数	开始位	1 位	1 位
	数据位（最低有效位第一位）	7 位	8 位
	奇偶校验（任选）	1 位（此位用于奇偶校验，无校应则无该位）	1 位（此位用于奇偶校验，无校应则无该位）
	停止位	1 或 2 位	1 或 2 位
	错误校验	LRC（即纵向冗余校验）	CRC(即循环冗余校验）

ASCII 可打印字符便于故障检测，而且对于用高级语言（如 Fortan）编程的主计算机及主 PC 很适宜。RTU 则适用于机器语言编程的计算机和 PC 主机。

用 RTU 模式传输的数据是 8 位二进制字符。如欲转换为 ASCII 模式，则每个 RTU 字符首先应分为高位和低位两部分，这两部分各含 4 位，然后转换成十六进制等量值。用以构成报文的 ASCII 字符都是十六进制字符。ASCII 模式使用的字符虽是 RTU 模式的两倍，但 ASCII 数据的译玛和处理更为容易一些，此外，用 RTU 模式时报文字符必须以连续数据流的形式传送，用 ASCII 模式，字符之间可产生长达 1s 的间隔，以适应速度较快的机器。

表 4 给出了以 RTU 方式读取整数据的例子

以 RTU 方式读取整数据的例子

主机请求						
地址	功能码	第一个寄存器的高位地址	第一个寄存器的低位地址	寄存器的数量的高位	寄存器的数量的底位	错误校验
01	03	00	38	00	01	XX
从机应答						
地址	功能码	字节数	数据高字节	数据低字节	错误校验	
01	03	2	41	24	XX	
十六进制数 4124 表示的十进制整数为 16676，错误校验值要根据传输方式而定。						

（2）ModBus 的数据校验方式

CRC-16（循环冗余错误校验）

CRC-16 错误校验程序如下：报文（此处只涉及数据位，不指起始位、停止位和任选的奇偶校验位）被看作是一个连续的二进制，其最高有效位（MSB）首选发送。报文先与 X↑16 相乘（左移 16 位），然后看 X↑16+X↑15+X↑2+1 除，X↑16+X↑15+X↑2+1 可以表示为二进制数 11000000000000101。整数商位忽略不

记，16 位余数加入该报文（MSB 先发送），成为 2 个 CRC 校验字节。余数中的 1 全部初始化，以免所有的零成为一条报文被接收。经上述处理而含有 CRC 字节的报文，若无错误，到接收设备后再被同一多项式（X↑16+X↑15+X↑2+1）除，会得到一个零余数（接收设备核验这个 CRC 字节，并将其与被传送的 CRC 比较）。全部运算以 2 为模（无进位）。

习惯于成串发送数据的设备会首选送出字符的最右位（LSB-最低有效位）。而在生成 CRC 情况下，发送首位应是被除数的最高有效位 MSB。由于在运算中不用进位，为便于操作起见，计算 CRC 时设 MSB 在最右位。生成多项式的位序也必须反过来，以保持一致。多项式的 MSB 略去不记，因其只对商有影响而不影响余数。

生成 CRC-16 校验字节的步骤如下：

①装如一个 16 位寄存器，所有数位均为 1。
②该 16 位寄存器的高位字节与开始 8 位字节进行“异或”运算。运算结果放入这个 16 位寄存器。
③把这个 16 寄存器向右移一位。
④若向右（标记位）移出的数位是 1，则生成多项式 1010000000000001 和这个寄存器进行“异或”运算；若向右移出的数位是 0，则返回③。
⑤重复③和④，直至移出 8 位。
⑥另外 8 位与该十六位寄存器进行“异或”运算。
⑦重复③~⑥，直至该报文所有字节均与 16 位寄存器进行“异或”运算，并移位 8 次。
⑧这个 16 位寄存器的内容即 2 字节 CRC 错误校验，被加到报文的最高有效位。

另外，在某些非 ModBus 通信协议中也经常使用 CRC16 作为校验手段，而且产生了一些 CRC16 的变种，他们是使用 CRC16 多项式 X↑16+X↑15+X↑2+1，单首次装入的 16 位寄存器为 0000；使用 CRC16 的反序 X↑16+X↑14+X↑1+1，首次装入寄存器值为 0000 或 FFFFH。

LRC（纵向冗余错误校验）

LRC 错误校验用于 ASCII 模式。这个错误校验是一个 8 位二进制数，可作为 2 个 ASCII 十六进制字节传送。把十六进制字符转换成二进制，加上无循环进位的二进制字符和二进制补码结果生成 LRC 错误校验（参见图）。这个 LRC 在接收设备进行核验，并与被传送的 LRC 进行比较，冒号（：）、回车符号（CR）、换行字符（LF）和置入的其他任何非 ASCII 十六进制字符在运算时忽略不计。

表 5 LRC 生成范例——读取 02 号从机的前 8 个线圈

十六进制	二进制

地址	0		2		0000		0010
功能码	0		1		0000		0001
起始地址高位	0		0		0000		0000
起始地址低位	0		0		0000		0000
单元数量	0		0		0000		0000
	0	8		+	0000	1000	
					0000	1011	
			变成补码		1111	0101	
错误校验	F		5		F		5
接受 PC 把所有收到的数据字节（包括最后的 LRC）加在一起，8 位应全部为 0（注意：和可能超过 8 位，应略去最低位）					0000		0010
					0000		0001
					0000		0000
					0000		0000
					0000		0000
					0000		1000
			错误校验		1111	0101	
			和		0000	0000	