MODBUS标准通讯协议 （简版）

（基于 Modbus应用协议 RTU通讯模式）

1. 前言

本协议适用于符合 MODBUS标准通讯协议的规定以及在 Modbus网络上以 RTU模式通信运行的设备和应用软件。本协议按照 Modbus 应用协议标准制定。

2. 波特率可选范围

代码	6	7	8	9	10	11	12	13	14
波特率	2400	4800	9600	14400	19200	28800	38400	57600	115200

3. RTU 通讯数据传输模式

3.1 RTU 模式每个字节（11 位）的格式为 ：

通讯传输为异步方式，并以字节（数据帧）为单位。在主站和从站之间传递的每一个数据帧都是 11位的串行数据流。

编码系统 : 8 -位二进制，报文中每个 8 位字节含有两个 4 位十六进制字符 (0 -9, A -F)

数 据 位: 1 个 起始位

8个 数据位， 首先发送最低有效位

1个 奇偶校验（注：偶校验是要求的，其它模式 （ 奇校验，无校验 ）也可以使用）

1个 停止位 （注 ： 使用无校验时要求 2个停止位）

帧校验域：循环冗余校验 (CRC)

3.2 字符的串行传送方式 ：

每个字符或字节按如下顺序发送 (从左到右): 最低有效位 (LSB) . . . 最高有效位 (MSB)

有奇偶校验

起始位	1	2	3	4	5	6	7	8	校验位	停止位

通过配置，设备可以接受奇校验、偶校验或无校验。如果无奇偶校验，那么传送一个附加的停止位来填充数据帧使其成为完整的 11 位异步字符：

无奇偶校验

起始位	1	2	3	4	5	6	7	8	停止位	停止位

3.3 数据编码：

Modbus 处理的所有数据按照存储数据的类型可以分为位寄存器（容量为 1 位）和 16 位寄存器（容量为 16 位）两种，它们的宽度都是 16 位(Data is packed as two bytes per register) ，协议允许单个选择 65536 个数据项，而且其读写操作可以越过多个连续数据项直到数据大小规格限制，这个数据大小规格限制与事务处理功能码有关。在 Modbus PDU中从 0～65535 寻址每个数据。

Modbus使用一个 ‘big-Endian ’表示地址和数据项，即最高有效字节在低地址存储，最低有效字节在高字节存储。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 位 0x1234 发送的第一字节为 0x12，然后发 0x34。

4. RTU 报文帧结构

Modbus RTU报文帧格式如下：

地址码	功能码	数据区	错误校验码	
1 字节	1 字节	0 到 252 字节	2 字节	
			CRC低	CRC高

4.1 地址码

地址码为通讯传输的第一字节， 这个字节表明， 由用户设定地址码的从站将接收由主站发送来的数据。每个从站都有唯一的地址码，只有符合主站发送的地址码的从站才能响应回送，且响应回送均以各自的地址码开始。主站发送的地址码表明将发送的从站地址，而从站发送的地址表明从站回送的地址。地址 0 用作广播地址，以使所有从站都能识别，从站的地址范围为（ 1～247） 。

4.2 数据区

数据区根据功能码的不同而不同。数据区包含需要从站执行什么动作，或由从站采集的返回信息。这些信息可以是实际数值、设置点、主站发给从站或从站发给主站的地址等。数据区的保持和输入寄存器值都是 16 位（2 字节），且高字节在前 ， 低字节在后。

4.3 错误校验码

主站或从站可用校验码判别报文在通讯过程中是否出错。错误检测域包含一个 16 位的值（用两个 8 位的字符来实现） ，错误检测域的内容是通过对报文内容进行循环冗长检测（ CRC） 方法得出的。 CRC 域附加在报文的最后，添加时先是低字节然后是高字节。故 CRC的高位字节是发送报文的最后一个字节。

4.4 功能码

功能码为通讯传输的第二字节。 Modbus 协议定义的功能码范围是 1～255，对于不同的控制器，功能码范围不同。主站发送请求，通过功能码告诉从站执行什么动作；从站响应请求，发送的功能码与主站发送来的功能码一样表明从站响应主站的操作。如果从站发送的功能码最高位为 1，表明从站没有响应或发送出错，主站可以根据得到的异常响应做进一步的处理，比如重发命令。

广播方式是主站向所有从站发送命令（从站地址为 0），不需要等待从站应答；从站接到广播命令后，执行命令，也不向主站应答。

M-2036 数字化就地处理箱能够处理的功能码如下表所示。

功能码	名 称	作 用
01（01H）	读取线圈状态	取得一组逻辑线圈的当前状态
03（03H）	读取保持寄存器	在一个或过个保持寄存器中取得当前的二进制值
15（0FH）	写多个线圈	设置一组逻辑线圈的状态
16（10H）	写多个寄存器	设置一个或多个寄存器的值

各功能码描述如下：

4.1.1 功能码 01（0x01）读取线圈状态

使用该功能从一个远程设备中读取 1～2000 个连续的线圈通断状态。请求 PDU详细说明了起始地址，即指定的第一个线圈地址和线圈数目。 在 PDU中从 0 开始寻址线圈， 因此因此编号 1～16 的线圈寻址为 0～15。

响应报文中的线圈按数据域的每位一个线圈进行打包。状态被表示为 1= ON 和 0= OFF。 第一个数据字节的 LSB（最低有效位）包括在询问中寻址的输出。其它线圈依次类推，一直到这个字节的高位端为止，并在后续字节中从低位到高位的顺序。

如果返回的输出数量不是 8 的倍数，将用零填充最后数据字节中的剩余比特（一直到字节的高位端） 。字节数量域说明了数据的全部字节数。

以下例子是从 17 号从站读取 20～37 的离散输出。

询问 RTU帧如下表：

从站地址	功能码	寄存器起始地址		线圈数量		CRC16校验码	
		高字节	低字节	高字节	低字节	低字节	高字节
11H	01H	00H	13H	00H	12H	4EH	52H

应答 RTU帧如下表：

从站地址	功能码	字节计数	数据	CRC16校验码
11H	01H	03H	CDH 6BH 03H	C0H 98H

将输出 27～20 的状态表示为十六进制字节值 CD，或二进制 1100 1101。输出 27 是这个字节的 3 MSB，输出 20 是 LSB。

通常，将一个字节内的比特表示为 MSB位于左侧， LSB 位于右侧。第一字节的输出从左至右为 26 至 19。下一个字节的输出从左到右为 34 至 27。当串行发射比特时， 从 LSB向 MSB传输： 19 . . . 26、27 . . . 34 等等。

在最后的数据字节中，将输出 37～36 表示为十六进制字节值 03，或二进制 0000 0011。输出 37 是左侧第 7 个比特位置，输出 36 是这个字节的 LSB。用零填充 6 个剩余高位比特。

4.1.2 功能码 03（0x03）读取保持寄存器

使用该功能从一个远程设备中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。在 PDU中从 0 开始寻址寄存器，因此编号 1～16 的寄存器寻址为 0～15。

响应报文中的寄存器数据被打包成每个寄存器有两个字节，对于每个寄存器第一字节为高位字节，第二字节为低位字节。

以下例子是从 17 号从站读取寄存器 108～ 10 。

询问 RTU帧如下表：

从站地址	功能码	寄存器起始地址		寄存器数量		CRC16校验码	
		高字节	低字节	高字节	低字节	低字节	高字节
11H	03H	00H	6BH	00H	03H	76H	87H

应答 RTU帧如下表：

从站地址	功能码	字节计数	数据	CRC16校验码
11H	03H	06H	02H 2BH 00H 00H 00H 64H	C8H BAH

将寄存器 108 的内容表示为两个十六进制字节值 02 2B，或十进制 555。将寄存器 109-110 的内容分别表示为十六进制 00 00 和 00 64，或十进制 0 和 100。

4.1.3 功能码 15（0x0F）写多个线圈

该功能将一个远程设备中的一组线圈的每个线圈强制为 ON或 OFF。请求 PDU指定了被强制的线圈编号。从 0 开始寻址线圈，因此，编号为 1 的线圈被寻址为 0.

请求数据域的内容指定了被请求的 ON/OFF状态。数据域中为逻辑 “1 ”的位请求相应输出为 ON，为逻辑 “0 ”的位请求相应输出位 OFF。

正常的响应返回从站地址、功能码、起始地址和被强制的线圈数量。

以下例子是从 17 号从站的线圈 20 开始写入 10 个线圈。

询问 RTU帧如下表：

从站地址	功能码	寄存器起始地址		线圈数量		字节数	输出数据	CRC16校验码	
		高字节	低字节	高字节	低字节			低字节	高字节
11H	0FH	00H	13H	00H	0AH	02H	CDH 01H	BFH	0BH

应答 RTU帧如下表：

从站地址	功能码	寄存器起始地址	线圈数量	CRC16校验码
11H	0FH	00H 13H	00H 0AH	26H 99H

请求的数据内容为两个字节：十六进制 CD 01 (二进制 1100 1101 0000 0001) 。使用下列方法，二进制比特对应输出。

比特： 1 1 0 0 1 1 0 1 0 0 0 0 0 0 0 1

输出： 27 26 25 24 23 22 21 20 －－－－－－29 28

传输的第一字节 (十六进制 CD)寻址为输出 27～20, 在这种设置中， 最低有效比特寻址为最低输出 （20）。传输的下一字节 (十六进制 01) 寻址为输出 29～28, 在这种设置中，最低有效比特寻址为最低输出（ 28） 。

用零填充最后数据字节中的未使用比特。

4.1.4 功能码 16（0x10）写多个寄存器

使用该功能码在一个远程设备中写连续寄存器块（ 1～123） 。

在请求数据域中指定了请求写入的值。将数据打包成每个寄存器两字节。

正常的响应返回从站地址功能码、起始地址和被写入寄存器的数量。

以下例子是从 17 号从站将十六进制 00 0A 和 01 02 写入以 2 开始的两个寄存器。

询问 RTU帧如下表：

从站地址	功能码	寄存器起始地址		寄存器数量		字节数	寄存器值	CRC16校验码	
		高字节	低字节	高字节	低字节			低字节	高字节
11H	10H	00H	01H	00H	02H	04H	00H 0AH 01H02H	C6H	F0H

应答 RTU帧如下表：

从站地址	功能码	寄存器起始地址	寄存器数量	CRC16校验码
11H	10H	00H 01H	00H 02H	12H 98H

4.1.5 异常应答 RTU帧

该功能是从站接收主站询问后有异常，如 CRC校验错误、无法解析等，进行异常应答。应答帧包括从站地址、功能码、错误编号、 CRC 校验码。

客户机请求和服务器异常响应的实例：

询问 RTU帧如下表：

从站地址	功能码	寄存器起始地址		线圈数量		CRC16校验码	
		高字节	低字节	高字节	低字节	低字节	高字节
11H	01H	04H	A1H	00H	01H	AFH	88H

异常应答 RTU帧如下表：

从站地址	功能码	错误代码	CRC16校验码
11H	81H	02H	C0H 54H

在这个实例中，主站对从站发出寻址请求。功能码 (01) 用于读输出状态操作。它请求地址为 1245(十六进制 04A1)的输出状态。根据输出域 (0001) 所指定的数量，只读出一个输出。

如果在从站中不存在该输出地址，那么从站将返回带有异常码 (02) 的异常响应。这就说明主站指定的是非法的从站数据地址。

异常码如下表：

代码	Modbus名称	含义
00H	其它未定义的错误	
01H	非法功能	从站无法解读功能码
02H	非法数据地址	主站读取或发送非法的寄存器地址
03H	非法数据值	主站发送的数据不完整或字节数错
04H	从设备故障（ CRC 校验码错）	从站正在执行请求的操作时， 产生不可恢复的差错。
05H	确认（从站未准备好或无法提供数据）	

5. 关于四字节变量的编址及访问

Modbus 应用协议对于寄存器的数值是用 16 位整形表示一个数据的，也就是 -32768～32768 的范围。由于浮点数 (float) 、长整形数 (long intger) 都是 32 位（ 4 字节）的数据长度，因此规定对于 4 字节变量通过两个连续的寄存器进行访问。举例说明如下：

地 址	变量名称	描 述	类型
4001H	FailureThd	失效报警阈值	Float
4003H	AlertThd	警告报警阈值	Float
4005H	HighThd	高值报警阈值	Float

询问 RTU帧如下表：

从站地址	功能码	寄存器起始地址		寄存器数量		CRC16校验码	
		高字节	低字节	高字节	低字节	低字节	高字节
11H	03H	40H	01H	00H	02H	82H	9BH

应答 RTU帧如下表：

从站地址	功能码	字节计数	数据	CRC16校验码
11H	03H	04H	3AH C4H 9BH A6H	6AH 46H

主站询问 17 号从站的变量 FailureThd ， FailureThd 的 4 字节按从高到低的顺序发送，即 4001H 为高位寄存器， 4002H 为低位寄存器。例如 FailureThd 的值为 0.0015，对应的 4 字节从高到低为 3AH C4H 9BH A6H,从站的发送顺序为： 3AH C4H 9BH A6H。

对于 4 字节的长整形数变量的访问格式同上。

附录： CRC的生成函数

执行 CRC生成的 C 语言的函数在下面示出。所有的可能的 CRC值都被预装在两个数组中，当计算报文内容时可以简单的索引即可。一个数组含有 16 位 CRC域的所有 256 个可能的高位字节，另一个数组含有地位字节的值。

这种索引访问 CRC的方式提供了比对报文缓冲区的每个新字符都计算新的 CRC更快的方法。

注意： 此函数内部执行高 / 低 CRC字节的交换。此函数返回的是已经经过交换的 CRC值。也就是说，从该函数返回的 CRC值可以直接放置于报文用于发送。

```
高字节表
/*  高位字节的  CRC值*/
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
}

低字节表
/*　低位字节的　CRC值*/
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
};

函数使用两个参数 :
unsigned char *puchMsg; 指向含有用于生成 CRC的二进制数据报文缓冲区的指针
unsigned short usDataLen; 报文缓冲区的字节数 .

CRC生成函数

```
unsigned short CRC16 ( puchMsg  ,  usDataLen )  /*  函数以  unsigned short  类型返回  CRC */
unsigned char *puchMsg ;  /*  用于计算  CRC的报文 */
unsigned short usDataLen ;  /*  报文中的字节数 */
{  unsigned char uchCRCHi = 0xFF ;  /* CRC  的高字节初始化 */
   unsigned char uchCRCLo = 0xFF ;  /* CRC  的低字节初始化 */
   unsigned uIndex ;  /* CRC  查询表索引 */
   while (usDataLen--)  /*  完成整个报文缓冲区 */
   {  uIndex = uchCRCLo ^ *puchMsgg++ ;  /*  计算  CRC */
      uchCRCLo = uchCRCHi ^ auchCRCHi[uIndex] ;
      uchCRCHi = auchCRCLo[uIndex] ;
   }
   return (uchCRCHi << 8 | uchCRCLo) ;
}
```