Modbus 通讯协议

Modbus 协议

Modbus 协议最初由 Modicon 公司开发出来，在 1979 年末该公司成为施耐德自动化(Schneider Automation)部门的一部分，现在 Modbus 已经是工业领域全球最流行的协议。此协议支持传统的 RS-232、RS-422、RS-485 和以太网设备。许多工业设备，包括 PLC，DCS，智能仪表等都在使用 Modbus 协议作为他们之间的通讯标准。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。

当在网络上通信时，Modbus 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，控制器将生成应答并使用 Modbus 协议发送给询问方。

Modbus 协议包括 ASCII、RTU、TCP 等，并没有规定物理层。此协议定义了控制器能够认识和使用的消息结构，而不管它们是经过何种网络进行通信的。标准的 Modicon 控制器使用 RS232C 实现串行的 Modbus。Modbus 的 ASCII、RTU 协议规定了消息、数据的结构、命令和就答的方式，数据通讯采用 Maser/Slave 方式，Master 端发出数据请求消息，Slave 端接收到正确消息后就可以发送数据到 Master 端以响应请求；Master 端也可以直接发消息修改 Slave 端的数据，实现双向读写。

Modbus 协议需要对数据进行校验，串行协议中除有奇偶校验外，ASCII 模式采用 LRC 校验，RTU 模式采用 16 位 CRC 校验，但 TCP 模式没有额外规定校验，因为 TCP 协议是一个面向连接的可靠协议。另外，Modbus 采用主从方式定时收发数据，在实际使用中如果某 Slave 站点断开后（如故障或关机），Master 端可以诊断出来，而当故障修复后，网络又可自动接通。因此，Modbus 协议的可靠性较好。

下面我来简单的给大家介绍一下，对于 Modbus 的 ASCII、RTU 和 TCP 协议来说，其中 TCP 和 RTU 协议非常类似，我们只要把 RTU 协议的两个字节的校验码去掉，然后在 RTU 协议的开始加上 5 个 0 和一个 6 并通过 TCP/IP 网络协议发送出去即可。所以在这里我仅介绍一下 Modbus 的 ASCII 和 RTU 协议。

下表是 ASCII 协议和 RTU 协议进行的比较：

协议	开始标记	结束标记	校验	传输效率	程序处理
ASCII	:（冒号）	CR,LF	LRC	低	直观，简单，易调试
RTU	无	无	CRC	高	不直观，稍复杂

通过比较可以看到，ASCII 协议和 RTU 协议相比拥有开始和结束标记，因此在进行程序处理时能更加方便，而且由于传输的都是可见的 ASCII 字符，所以进行调试时就更加的直观，另外它的 LRC 校验也比较容易。但是因为它传输的都是可见的 ASCII 字符，RTU 传输的数据每一个字节 ASCII 都要用两个字节来传输，比如 RTU 传输一个十六进制数 0xF9,ASCII 就需要传输’F’’9’的 ASCII 码 0x39 和 0x46 两个字节，这样它的传输的效率就比较低。所以一般来说，如果所需要传输的数据量较小可以考虑使用 ASCII 协议，如果所需传输的数据量比较大，最好能使用 RTU 协议。

下面对两种协议的校验进行一下介绍。

1、LRC 校验

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 校验比较简单，它在 ASCII 协议中使用，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。它仅仅是把每一个需要传输的数据按字节叠加后取反加 1 即可。下面是它的 VC 代码：

```
BYTE GetCheckCode(const char * pSendBuf, int nEnd)//获得校验码

{

    BYTE byLrc = 0;

    char pBuf[4];

    int nData = 0;

    for(i=1; i<end; i+=2)  //i 初始为 1，避开“开始标记”冒号

    {

        //每两个需要发送的 ASCII 码转化为一个十六进制数

    pBuf [0] = pSendBuf [i];

    pBuf [1] = pSendBuf [i+1];

    pBuf [2] = '\0';

    sscanf(pBuf,"%x",& nData);

    byLrc += nData;

    }

    byLrc = ~ byLrc;

    byLrc ++;

    return byLrc;

}
```

2、CRC 校验

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。下面是它的 VC 代码：

```
WORD GetCheckCode(const char * pSendBuf, int nEnd)//获得校验码
{
    WORD wCrc = WORD(0xFFFF);
    for(int i=0; i<nEnd; i++)
    {
    wCrc ^= WORD(BYTE(pSendBuf[i]));
    for(int j=0; j<8; j++)
        {
            if(wCrc & 1)
            {
        wCrc >>= 1;
        wCrc ^= 0xA001;
            }
            else
            {
        wCrc >>= 1;
            }
        }
    }
    return wCrc;
}
```

对于一条 RTU 协议的命令可以简单的通过以下的步骤转化为 ASCII 协议的命令：

1、 把命令的 CRC 校验去掉，并且计算出 LRC 校验取代。

2、 把生成的命令串的每一个字节转化成对应的两个字节的 ASCII 码，比如 0x03 转化成 0x30,0x33（0 的 ASCII 码和 3 的 ASCII 码）。

3、 在命令的开头加上起始标记":"，它的 ASCII 码为 0x3A。

4、 在命令的尾部加上结束标记 CR,LF（0xD,0xA），此处的 CR,LF 表示回车和换行的 ASCII 码。

所以以下我们仅介绍 RTU 协议即可，对应的 ASCII 协议可以使用以上的步骤来生成。

下表是 Modbus 支持的功能码：

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
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

在这些功能码中较长使用的是 1、2、3、4、5、6 号功能码，使用它们即可实现对下位机的数字量和模拟量的读写操作。

1、读可读写数字量寄存器（线圈状态）：

计算机发送命令：[设备地址] [命令号 01] [起始寄存器地址高 8 位] [低 8 位] [读取的寄存器数高 8 位] [低 8 位] [CRC 校验的低 8 位] [CRC 校验的高 8 位]

例：[11][01][00][13][00][25][CRC 低][CRC 高]

意义如下：

<1>设备地址：在一个 485 总线上可以挂接多个设备，此处的设备地址表示想和哪一个设备通讯。例子中为想和 17 号(十进制的 17 是十六进制的 11)通讯。

<2>命令号 01：读取数字量的命令号固定为 01。

<3>起始地址高 8 位、低 8 位：表示想读取的开关量的起始地址(起始地址为 0)。比如例子中的起始地址为 19。

<4>寄存器数高 8 位、低 8 位：表示从起始地址开始读多少个开关量。例子中为 37 个开关量。

<5>CRC 校验：是从开头一直校验到此之前。在此协议的最后再作介绍。此处需要注意，CRC 校验在命令中的高低字节的顺序和其他的相反。

设备响应：[设备地址] [命令号 01] [返回的字节个数][数据 1][数据 2]...[数据 n][CRC 校验的低 8 位] [CRC 校验的高 8 位]

例：[11][01][05][CD][6B][B2][0E][1B][CRC 低][CRC 高]

意义如下：

<1>设备地址和命令号和上面的相同。

<2>返回的字节个数：表示数据的字节个数，也就是数据 1，2...n 中的 n 的值。

<3>数据 1...n：由于每一个数据是一个 8 位的数，所以每一个数据表示 8 个开关量的值，每一位为 0 表示对应的开关断开，为 1 表示闭合。比如例子中，表示 20 号(索引号为 19)开关闭合，21 号断开，22 闭合，23 闭合，24 断开，25 断开，26 闭合，27 闭合...如果询问的开关量不是 8 的整倍数，那么最后一个字节的高位部分无意义，置为 0。

<4>CRC 校验同上。

2、读只可读数字量寄存器（输入状态）：

和读取线圈状态类似，只是第二个字节的命令号不再是 1 而是 2。

3、写数字量（线圈状态）：

计算机发送命令：[设备地址] [命令号 05] [需下置的寄存器地址高 8 位] [低 8 位] [下置的数据高 8 位] [低 8 位] [CRC 校验的低 8 位] [CRC 校验的高 8 位]

例：[11][05][00][AC][FF][00][CRC 低][CRC 高]

意义如下：

<1>设备地址和上面的相同。

<2>命令号:写数字量的命令号固定为 05。

<3>需下置的寄存器地址高 8 位，低 8 位：表明了需要下置的开关的地址。

<4>下置的数据高8位,低8位:表明需要下置的开关量的状态。例子中为把该开关闭合。注意,此处只可以是[FF][00]表示闭合[00][00]表示断开，其他数值非法。

<5>注意此命令一条只能下置一个开关量的状态。

设备响应：如果成功把计算机发送的命令原样返回，否则不响应。

4、读可读写模拟量寄存器（保持寄存器）：

计算机发送命令：[设备地址] [命令号 03] [起始寄存器地址高 8 位] [低 8 位] [读取的寄存器数高 8 位] [低 8 位] [CRC 校验的低 8 位] [CRC 校验的高 8 位]

例：[11][03][00][6B][00][03][CRC 低][CRC 高]

意义如下：

<1>设备地址和上面的相同。

<2>命令号:读模拟量的命令号固定为 03。

<3>起始地址高 8 位、低 8 位：表示想读取的模拟量的起始地址(起始地址为 0)。比如例子中的起始地址为 107。

<4>寄存器数高 8 位、低 8 位：表示从起始地址开始读多少个模拟量。例子中为 3 个模拟量。注意，在返回的信息中一个模拟量需要返回两个字节。

设备响应：[设备地址] [命令号 03] [返回的字节个数][数据 1][数据 2]...[数据 n][CRC 校验的低 8 位] [CRC 校验的高 8 位]

例：[11][03][06][02][2B][00][00][00][64][CRC 低][CRC 高]

意义如下：

<1>设备地址和命令号和上面的相同。

<2>返回的字节个数：表示数据的字节个数，也就是数据 1，2...n 中的 n 的值。例子中返回了 3 个模拟量的数据，因为一个模拟量需要 2 个字节所以共 6 个字节。

<3>数据 1...n：其中[数据 1][数据 2]分别是第 1 个模拟量的高 8 位和低 8 位，[数据 3][数据 4]是第 2 个模拟量的高 8 位和低 8 位，以此类推。例子中返回的值分别是 555，0，100。

<4>CRC 校验同上。

5、读只可读模拟量寄存器（输入寄存器）：

和读取保存寄存器类似，只是第二个字节的命令号不再是 2 而是 4。

6、写单个模拟量寄存器（保持寄存器）：

计算机发送命令：[设备地址] [命令号 06] [需下置的寄存器地址高 8 位] [低 8 位] [下置的数据高 8 位] [低 8 位] [CRC 校验的低 8 位] [CRC 校验的高 8 位]

例：[11][06][00][01][00][03][CRC 低][CRC 高]

意义如下：

<1>设备地址和上面的相同。

<2>命令号:写模拟量的命令号固定为 06。

<3>需下置的寄存器地址高 8 位，低 8 位：表明了需要下置的模拟量寄存器的地址。

<4>下置的数据高 8 位，低 8 位：表明需要下置的模拟量数据。比如例子中就把 1 号寄存器的值设为 3。

<5>注意此命令一条只能下置一个模拟量的状态。

设备响应：如果成功把计算机发送的命令原样返回，否则不响应