# MODBUS通信协议

## 第1章 MODBUS协议介绍

### 1.1 MODBUS标准简介

Modbus是由 Modicon（现为施耐德电气公司的一个品牌）在1979年发明的，是全球第一个真正用于工业现场的总线协议。

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。

为更好地普及和推动Modbus在基于以太网上的分布式应用，目前施耐德公司已将Modbus协议的所有权移交给 IDA（Interface for Distributed Automation，分布式自动化接口）组织，并成立了Modbus-IDA组织，为Modbus今后的发展奠定了基础。

在中国，Modbus已经成为国家标准。

标准编号：GB/T19582-2008

标准名称：《基于 Modbus 协议的工业自动化网络规范》

分3个部分：

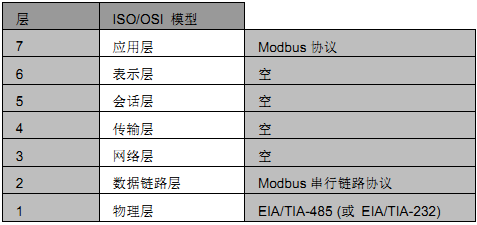
《GB/T 19582.1-2008第1部分：Modbus应用协议》

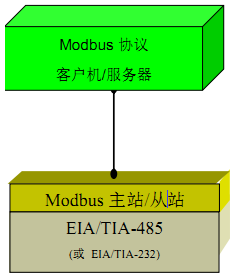
《GB/T 19582.2-2008第2部分：Modbus协议在串行链路上的实现指南》

《GB/T 19582.3-2008第3部分：Modbus协议在TCP/IP上的实现指南》

### 1.2 MODBUS协议概述

按照7层OSI通信模型，Modbus标准包括应用层、数据链路层、物理层。





Modbus串行链路协议是一个主/从协议。该协议位于OSI模型的第二层。

一个主从类型的系统有一个向某个“子”节点发出显式命令并处理响应的节点(主节点)。典型的子节点在没有收到主节点的请求时并不主动发送数据，也不与其它子节点通信。

在物理层，Modbus 串行链路系统可以使用不同的物理接口(RS485、RS232)。最常用的是TIA/EIA-485(RS485)两线制接口。

### 1.3 Modbus主站/从站协议原理

Modbus串行链路协议是一个主-从协议。在同一时刻，只有一个主节点连接于总线，一个或多个子节点(最大编号为247)连接于同一个串行总线。Modbus通信总是由主节点发起。子节点在没有收到来自主节点的请求时，从不会发送数据。子节点之间从不会互相通信。主节点在同一时刻只会发起一个Modbus事务处理。

主节点以两种模式对子节点发出Modbus请求:

1．单播模式

主节点以特定地址访问某个子节点，子节点接到并处理完请求后，子节点向主节点返回一个报文(一个'应答')。在这种模式，一个Modbus事务处理包含2个报文：一个来自主节点的请求，一个来自子节点的应答。

每个子节点必须有唯一的地址(1到247)，这样才能区别于其它节点被独立的寻址。

2．广播模式

主节点向所有的子节点发送请求。对于主节点广播的请求没有应答返回。广播请求一般用于写命令。所有设备必须接受广播模式的写功能。地址0是专门用于表示广播数据的。

地址规则：

Modbus寻址空间有256个不同地址。

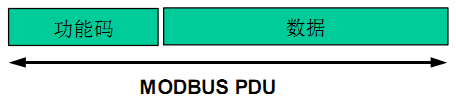
|  |  |  |
| --- | --- | --- |
| 0 | 1 ~ 247 | 248 ~ 255 |
| 广播地址 | 子节点单独地址 | 保留 |

地址0为广播地址。所有的子节点必须识别广播地址。

Modbus主节点没有地址，只有子节点必须有一个地址。该地址必须在Modbus串行总线上唯一。

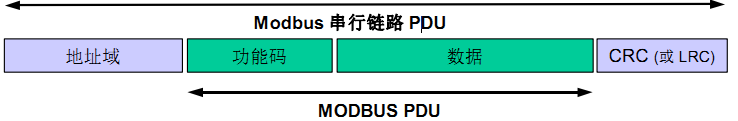
### 1.4 Modbus 帧描述

《Modbus应用协议》定义了简单的协议数据单元(PDU - Protocol Data Unit)：



Modbus 协议数据单元

发起Modbus事务处理的客户端构造Modbus PDU，然后添加附加的域以构造通信PDU。



串行链路上的 Modbus 帧

在Modbus串行链路，地址域只含有子节点地址。

如前文所述，合法的子节点地址为十进制0–247。每个子设备被赋予1–247范围中的地址。主节点通过将子节点的地址放到报文的地址域对子节点寻址。当子节点返回应答时，它将自己的地址放到应答报文的地址域以让主节点知道哪个子节点在回答。

功能码指明服务器要执行的动作。功能码后面可跟有表示含有请求和响应参数的数据域。

错误检验域是对报文内容执行"冗余校验"的计算结果。根据不同的传输模式(RTU or ASCII)使用两种不同的计算方法。

### 1.5 RTU传输模式

有两种串行传输模式被定义: RTU模式和ASCII模式。

它定义了报文域的位内容在线路上串行的传送。它确定了信息如何打包为报文和解码。

Modbus串行链路上所有设备的传输模式(和串行口参数)必须相同。

尽管在特定的领域ASCII模式是要求的，但达到Modbus设备之间的互操作性只有每个设备都有相同的模式：所有设备必须实现RTU模式。ASCII传输模式是选项。

当设备使用RTU(Remote Terminal Unit)模式在Modbus串行链路通信，报文中每个8位字节含有两个4位十六进制字符。这种模式的主要优点是较高的数据密度，在相同的波特率下比ASCII模式有更高的吞吐率。每个报文必须以连续的字符流传送。

RTU模式每个字节(11位)的格式为:

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)

每字节的bit流:

1起始位

8数据位，首先发送最低有效位

1位作为奇偶校验

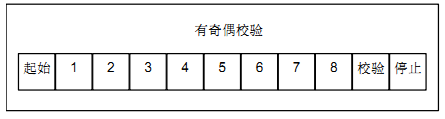
1停止位

偶校验是要求的，其它模式 (奇校验，无校验)也可以使用。为了保证与其它产品的最大兼容性，同时支持无校验模式是建议的。默认校验模式必须为偶校验。

注 : 使用无校验要求2个停止位。

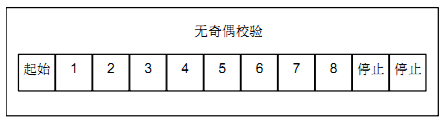
字符是如何串行传送的：

每个字符或字节均由此顺序发送(从左到右)：最低有效位(LSB) . . .最高有效位(MSB)



RTU模式位序列

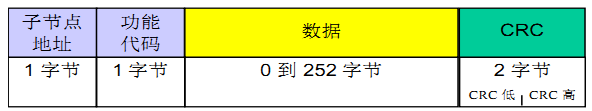
设备配置为奇校验、偶校验或无校验都可以接受。如果无奇偶校验，将传送一个附加的停止位以填充字符帧：



RTU模式位序列 (无校验的特殊情况)

帧检验域:循环冗余校验(CRC),2字节。

帧描述:



RTU报文帧

Modbus RTU帧总长度最大为256字节。

#### 1.5.1 Modbus报文RTU帧

## 第2章 MODBUS POLL协议分析

### 2.1读保持寄存器 03H

1.说明

读保持寄存器。可读取单个或多个保持寄存器。

2.查询

从机地址为 11H。保持寄存器的起始地址为 00H，结束地址为 3C H。该次查询总共访问多

个保持寄存器。

000308-Tx: 11 03 00 6B 00 03 76 87

读保持寄存器-查询

|  |  |  |
| --- | --- | --- |
| 数组序号 | Hex | 说明 |
| 0 | 11 | 从机地址 |
| 1 | 03 | 功能码 |
| 2 | 00 | 寄存器地址高字节 |
| 3 | 6B | 寄存器地址低字节 |
| 4 | 00 | 寄存器数量高字节 |
| 5 | 03 | 寄存器数量低字节 |
| 6 | 76 | CRC低字节 |
| 7 | 87 | CRC高字节 |

3.响应

保持寄存器的长度为 2 个字节。对于单个保持寄存器而言，寄存器高字节数据先被传输，低字节数据后被传输。保持寄存器之间，低地址寄存器先被传输，高地址寄存器后被传输。

000309-Rx: 11 03 06 00 6B 00 13 00 00 38 B9

读保持寄存器-响应

|  |  |  |
| --- | --- | --- |
| 数组序号 | Hex | 说明 |
| 0 | 11 | 从机地址 |
| 1 | 03 | 功能码 |
| 2 | 06 | 字节数 |
| 3 | 00 | 数据 1 高字节(006BH) |
| 4 | 6B | 数据 1 低字节(006BH) |
| 5 | 00 | 数据 2 高字节(006CH) |
| 6 | 13 | 数据 2 低字节(006CH) |
| 7 | 00 | 数据 3 高字节(006DH) |
| 8 | 00 | 数据 3 低字节(006DH) |
| 9 | 38 | CRC低字节 |
| 10 | B9 | CRC高字节 |

保持寄存器 006BH 到 006DH 结果

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 006BH | 006BH | 006CH | 006CH | 006DH | 006DH |
| 高字节 | 低字节 | 高字节 | 低字节 | 高字节 | 低字节 |
| 0 | 6B | 0 | 13 | 0 | 0 |

### 2.2读输入寄存器04H

1.说明：

读输入寄存器命令。该命令支持单个寄存器访问也支持多个寄存器访问。

2.查询：

从机地址为 11H。输入寄存器的起始地址为 0008H，寄存器的结束地址为 0009H。本次访问访 2个输入寄存器。

读输入寄存器-查询

|  |  |  |
| --- | --- | --- |
| 数组序号 | Hex | 说明 |
| 0 | 11 | 从机地址 |
| 1 | 04 | 功能码 |
| 2 | 00 | 寄存器起始地址高字节 |
| 3 | 08 | 寄存器起始地址低字节 |
| 4 | 00 | 寄存器个数高字节 |
| 5 | 02 | 寄存器个数低字节 |
| 6 | F2 | CRC 低字节 |
| 7 | 99 | CRC 高字节 |

3响应：

输入寄存器长度为 2 个字节。对于单个输入寄存器而言，寄存器高字节数据先被传输，低字节数据被传输。输入寄存器之间，低地址寄存器先被传输，高地址寄存器后被传输。

读寄存器-响应

|  |  |  |
| --- | --- | --- |
| 数组序号 | Hex | 说明 |
| 0 | 11 | 从机地址 |
| 1 | 04 | 功能码 |
| 2 | 04 | 字节数 |
| 3 | 00 | 数据 1 高字节(0008H) |
| 4 | 0A | 数据 1 低字节(0008H) |
| 5 | 00 | 数据 2 高字节(0009H) |
| 6 | 0B | 数据 2 低字节(0009H) |
| 7 | 8B | CRC低字节 |
| 8 | 80 | CRC高字节 |

|  |  |  |  |
| --- | --- | --- | --- |
| 0008H | 0008H | 0009H | 0009H |
| 高字节 | 低字节 | 高字节 | 低字节 |
| 00 | 0A | 00 | 0B |

### 2.3写单个保持寄存器06H

1.说明

写保持寄存器。注意 06 指令只能操作单个保持寄存器，16 指令可以设置单个或多个保持寄存器。

2.查询

从机地址为 11H。保持寄存器地址为 0001H。寄存器内容为 0003H。

写单个保持寄存器——呼叫

|  |  |  |
| --- | --- | --- |
| 数组序号 | Hex | 说明 |
| 0 | 11 | 从机地址 |
| 1 | 06 | 功能码 |
| 2 | 00 | 寄存器地址高字节 |
| 3 | 01 | 寄存器地址低字节 |
| 4 | 00 | 数据高字节 |
| 5 | 01 | 数据低字节 |
| 6 | 1B | CRC低字节 |
| 7 | 5A | CRC高字节 |

写单个保持寄存器——响应

|  |  |  |
| --- | --- | --- |
| 数组序号 | Hex | 说明 |
| 0 | 11 | 从机地址 |
| 1 | 06 | 功能码 |
| 2 | 00 | 寄存器地址高字节 |
| 3 | 01 | 寄存器地址低字节 |
| 4 | 00 | 寄存器数据高字节 |
| 5 | 01 | 寄存器数据低字节 |
| 6 | 1B | CRC低字节 |
| 7 | 5A | CRC高字节 |

### 2.4写多个保持寄存器10H

1.说明

写多个保持寄存器。

2.查询

从机地址为 11H。保持寄存器的其实地址为 0001H，寄存器的结束地址为 0002H。总共访问2个寄存器。保持寄存器 0001H 的内容为 000AH，保持寄存器 0002H 的内容为 0102H。

写多个保持寄存器——请求

|  |  |  |
| --- | --- | --- |
| 数组序号 | Hex | 说明 |
| 0 | 11 | 从机地址 |
| 1 | 10 | 功能码 |
| 2 | 00 | 寄存器地址高字节 |
| 3 | 01 | 寄存器地址低字节 |
| 4 | 00 | 寄存器数量高字节 |
| 5 | 02 | 寄存器数量低字节 |
| 6 | 04 | 字节数 |
| 7 | 00 | 数据 1 高字节 |
| 8 | 0A | 数据 1 低字节 |
| 9 | 01 | 数据 2 高字节 |
| 10 | 02 | 数据 2 低字节 |
| 11 | C6 | CRC低字节 |
| 12 | F0 | CRC高字节 |

保持寄存器 0001H 到 0002H 内容

|  |  |  |  |
| --- | --- | --- | --- |
| 0001H | 0001H | 0002H | 0002H |
| 高字节 | 低字节 | 高字节 | 低字节 |
| 00 | 0A | 01 | 02 |

写多个保持寄存器——响应

|  |  |  |
| --- | --- | --- |
| 数组序号 | Hex | 说明 |
| 0 | 11 | 从机地址 |
| 1 | 10 | 功能码 |
| 2 | 00 | 寄存器地址高字节 |
| 3 | 01 | 寄存器地址低字节 |
| 4 | 00 | 寄存器数量高字节 |
| 5 | 02 | 寄存器数量低字节 |
| 6 | 12 | CRC低字节 |
| 7 | 98 | CRC高字节 |

## 第3章 文档更新记录

|  |  |  |  |
| --- | --- | --- | --- |
| 版本 | 更改说明 | 作者 | 发布日期 |
| V0.1 | 刘博 | 刘博 | 2015-12-02 |
| V0.5 |  |  | 2016-01-11 |