开放型 Modbus/TCP 规范

修订版 1.0，1999 年 3 月 29 日

Andy Swales

Schneider 电气公司

aswales@modicon.com

目录

该规范的发展概况
原始版本 1997 年 9 月 3 日

作为公共评论的草案。

再版 1999 年 3 月 29 日，即修订版 1.0。

没有大的技术改动，仅作了补充说明。
增加了附录 A 和 B 作为对一些常用执行问题的回应。

该 MODBUS/TCP 规范在万维网上公开发行。它表明开发者的意愿是把它作为工业自动化领域具有互用性的标准。

既然 MODBUS 和 MODBUS/TCP 作为事实上的“ 实际 ”标准，而且很多生产商已经实现了它的功能，此规范主要是阐述在互连网上具有普遍可用性的基于 TCP 通讯协议的 MODBUS 报文的特殊编码。

2.概述

MODBUS/TCP 是简单的、中立厂商的用于管理和控制自动化设备的 MODBUS 系列通讯协议的派生产品。显而易见，它覆盖了使用 TCP/IP 协议的“ Intranet ”和“ Internet ”环境中 MODBUS 报文的用途。协议的最通用用途是为诸如 PLC ’ s，I/O 模块，以及连接其它简单域总线或 I/O 模块的网关服务的。

MODBUS/TCP 协议是作为一种（实际的）自动化标准发行的。既然 MODBUS 已经广为人知，该规范只将别处没有收录的少量信息列入其中。然而，本规范力图阐明 MODBUS 中哪种功能对于普通自动化设备的互用性有价值，哪些部分是 MODBUS 作为可编程的协议交替用于 PLC ’ s 的“ 多余部分 ”。

它通过将配套报文类型“ 一致性等级 ”，区别那些普遍适用的和可选的，特别是那些适用于特殊设备如 PLC ’ s 的报文。

2.1 面向连接

在 MODBUS 中，数据处理传统上是无国界的，使它们对由噪音引起的中断有高的抵抗力，而且在任一端只需要最小的维护信息。

编程操作，另一方面，期望一种面向连接的方法。这种方法对于简单变量通过唯一的“ 登录 ”符号完成，对于 Modbus Plus 变量，通过明确的“ 程序路径 ”容量来完成，而“ 程序路径 ”容量维持了一种双向连接直到被彻底击穿。

MODBUS/TCP 处理两种情况。连接在网络协议层很容易被辨认，单一的连接可以支持多个独立的事务。此外，TCP 允许很大数量的并发连接，因而很多情况下，在请求时重新连接或复用一条长的连接是发起者的选择。

熟悉 MODBUS 的开发者会感到惊讶：为什么面向连接 TCP 协议比面向数据报的 UDP 要应用广泛。主要原因是通过封装独立的“ 事务 ”在一个连接中，此连接可被识别，管理和取消而无须请求客户和服务器采用特别的动作。这就使进程具有对网络性能变化的适应能力，而且容许安全特色如防火墙和代理可以方便的添加。

类似的推理被最初的万维网的开发者所采用，他们选用 TCP 及端口 80 去实现一个作为单一事务的最小的环球网询问。

2.2 数据编码

MODBUS 采用“ big-endian ”来表示地址和数据对象。 这就意味着当一个数字表示的数量大于所传输的单一字节，最大有效字节将首先被发送。例如：

16 - bits 0x1234 将为 0x12 0x34

32 - bits 0x12345678L 将为 0x12 0x34 0x56 0x78

2.3 参考编号的解释

MODBUS 将其数据模型建立在一系列具有不同特征的表的基础之上。这四个基本表如下

离散输入 单比特，由 I/O 系统提供，只读
离散输出 单比特，由应用程序更改，读写
输入寄存器 16 比特，数值，由 I/O 系统提供 ，只读
输出寄存器 16 比特，数值，由应用程序更改，读写

输入和输出之间以及可寻址位和可寻址代码的数据对象之间的差别并不意味着任何应用性能的不同。如果这是我们所讨论的目标机械的最自然的解释，那么认为所有的四个基本表是相互覆盖的看法也是非常普通而完全可以接受的。

对于每一个基本表，协议允许单独选择 65536 个数据对象中的任何一个，而且对那些对象的读写操作可以跨越多个连续的数据对象，直到达到基于处理事务功能代码的数据大小限制。

这儿没有假定数据对象代表一种真正邻接的数据阵列，而这是大多数简单 PLC ’ s 的解释。

“ 读写常用参考 ”功能代码被定义为携带 32 位的参考值并且能允许在“ 非常 ”大的空间里可以直接访问数据对象。现在没有可以利用这一特点的 PLC 设备。

一个易造成混乱的潜在来源是用于 MODBUS 功能的参考值和用于 Modicon PLC ’ s 的“ 寄存器值 ”之间的关系。由于历史原因，用户参考值使用从 1 开始的十进制数表示。而 MODBUS 采用更普通的从 0 开始的无符号整数进行软件数据整理分析。
于是，请求从 0 读取寄存器的 Modbus 消息将已知值返回建立在寄存器 4：00001（存储类型 4=输出寄存器，参考值 00001）中的应用程序。

2.4 隐含长度基本原则

所有的 MODBUS 请求和响应都被设计成在此种方法下工作，即接收者可确认消息的完整性。 对于请求和响应为固定长度的功能代码，仅发送功能代码就足够了。对于在请求和响应中携带不定长数据的功能代码，数据部分前将加上一个字节的数据统计。

当 Modbus 通过 TCP 运送，前缀中携带附加的长度信息以便接收者识别消息的边界，甚至消息被分成若干组进行传输。 外在的和隐含的长度准则的存在，以及 CRC-32 检错代码（以太网）的使用使请求和响应消息中发生未被识别的错误的机率减至无限小。

3. 一致性等级概述

当从草稿开始定义一种新的协议，有可能加强编码方式和阐述的一致性。MODBUS 由于其先进的特性，已经在很多地方得到了实施，必须避免破坏它已经存在的实施。

因此，已经存在的成套的处理类型被划分出一致性等级：等级 0 代表普遍使用且总体上一致的功能；等级 2 代表有用的功能，但带有某些特性。现存装置的不适应于互用性的功能也已确认。

必须注意到，将来对该标准的扩充将定义附加的功能代码来处理现存事实标准不适用的情形。然而，被提议扩充的详细资料出现在本手册中将会另人误解。通过将代码“ 随机的 ”发送或者即便是通过检查异常响应的类型来确定特别的目标装置是否支持特别的功能代码总是可能的，而且该方法将保证引入这些扩充的现使用的 MODBUS 设备的连续的互用性。事实上，这就是当前功能代码的分级原则。

3.1 等级 0

这是最小的有用功能，对主站和从站来说。
读乘法寄存器 (fc 3)
写乘法寄存器 (fc 16)

3.2 等级 1

这是附加的被普遍实现的和能共同使用的成套功能，正如前面介绍过的，许多从站把输入，输出，离散值和寄存器值作为同等的进行处理。

读线圈 (fc 1)
读离散输入 (fc 2)
读寄存器输入 (fc 4)
写线圈 (fc 5)
写单一寄存器 (fc 6)
读异常状态字 (fc 7)
此功能对于每一个从站系列显然具有不同的含义。

3.3 等级 2

这些是需要 HMI 和管理等例行操作的数据传送功能。
强制型多路线圈 (fc 15)
读一般参考值 (fc 20)
该功能可以处理并发的多个请求，而且能接收 32 位的参考数值。当前的 584 和 984PLC ’ s 仅使用此功能接收类型 6 的参考值（扩展的寄存器文件）。
该功能最适于扩充以处理大的寄存器空间和缺少诸如“ 未定位 ”变量的参考值的数据对象。

写一般参考值 (fc 21)
此功能可以处理并发的多个请求 ，也可接收 32 位的参考数值。当前的 584 和 984PLC ’ s 仅使用此功能接收类型 6 的参考值（扩展的寄存器文件）。
该功能最适于扩充以处理大的寄存器空间和缺少诸如“ 未定位 ”变量的参考值的数据对象。
掩膜写寄存器 (fc 22)
读/写寄存器 (fc 23)
此功能把一定范围的寄存器输入和输出当作单一的处理事务。使用 MODBUS 是执行规则的带有 I/O 模块的状态影象交换的最好办法。
如此，高性能的通用的数据采集装置可以执行功能 3，16 和 23，从而把快捷的数据规则交换（23）和执行特殊数据对象的需求询问或更新的能力结合起来（3 和 16）。

读 FIFO 队列 (fc 24)
一个有点专用的功能，打算将表结构的数据象 FIFO（用到 584/984 上的 FIN 和 FOUT

功能模块）一样传送到主机。对于某种事件录入软件很有用。

3.4 机器/厂家/网络的特殊功能

以下所有的功能，虽然在 MODBUS 协议手册中提到，但由于它们有很强的机器依赖性，因而不适于互用性的目的。

诊断 (fc 8)
编程 (484) (fc 9)
轮询 (484) (fc 10)
获取通讯事件计数器值(Modbus) (fc 11)
获取通讯事件记录(Modbus) (fc 12)
编程 (584/984) (fc 13)
轮询(584/984) (fc 14)
通告从站 ID (fc 17)
编程 (884/u84) (fc 18)
恢复通讯连接 (884/u84) (fc 19)
编程 (原理) (fc 40)
固件置换 (fc 125)
编程 (584/984) (fc 126)
通告本地地址 (Modbus) (fc 127)

4. 协议结构

本部分阐述了通过 MODBUS/TCP 网络携带的 MODBUS 请求和或响应封装的一般格式。必须注意到请求和响应本体（从功能代码到数据部分的末尾）的结构和其它 MODBUS 变量具有完全相同的版面格式和含义，如：

MODBUS 串行端口 - ASCII 编码
MODBUS 串行端口 - RTU (二进制) 编码
MODBUS PLUS 网络 – 数据通道

这些其它案例仅在组帧次序，检错模式和地址描述等格式有所不同。

所有的请求通过 TCP 从寄存器端口 502 发出。

请求通常是在给定的连接以半双工的方式发送。也就是说，当单一连接被响应所占用，就不能发送其它的请求。有些装置采用多条 TCP 连接来维持高的传输速率。 然而一些客户端设备尝试“ 流水线式 ”的请求。允许服务器以这种方式工作的技术在附录 A 中阐述。

MODBUS “ 从站地址 ”字段被单字节的“ 单元标识符 ”替换，从而用于通过网桥和网关等设备的通讯，这些设备用单一 IP 地址来支持多个独立的终接单元。

请求和响应带有六个字节的前缀，如下：

byte 0: 事务处理标识符 – 由服务器复制 – 通常为 0
byte 1: 事务处理标识符 – 由服务器复制 – 通常为 0
byte 2: 协议标识符= 0
byte 3: 协议标识符= 0
byte 4: 长度字段 （上半部分字节） = 0 (所有的消息长度小于 256)
byte 5: 长度字段 (下半部分字节) = 后面字节的数量

byte 6: 单元标识符 (原“ 从站地址 ”)
byte 7: MODBUS 功能代码
byte 8 on: 所需的数据

因而处理示例“ 以 4 的偏移从 UI 9 读 1 寄存器 ”返回 5 的值将是

请求：00 00 00 00 00 06 09 03 00 04 00 01
响应：00 00 00 00 00 05 09 03 02 00 05

一致性等级 0-2 的功能代码的应用的例子见后续部分

熟悉 MODBUS 的设计师将注意到 MODBUS/TCP 中不需要“ CRC-16 ”或“ LRC ”检查字段。而是采用 TCP/IP 和链路层（以太网）校验和机制来校验分组交换的准确性。

5. 一致性等级的协议参考值

注意到在例子中，请求和响应列在功能代码字节的前面。如前所述，在 MODBUS/TCP 案例中有一个依赖传输的包含 7 个字节的前缀。

ref ref 00 00 00 len unit

前面两个字节的“ ref ref ”在服务器中没有具体的值，只是为方便客户端而从请求和响应中逐字的复制过来。单客户机通常将该值置为 0。

在这个例子中，请求和响应的格式如下（例子是“ 读寄存器 ”请求，详述见后面部分）。

03 00 00 00 01 => 03 02 12 34

这表示给前缀加上一个十六进制的串联的字节，这样，TCP 连接上的整个消息将是（假设单元标识符还是 09）

请求： 00 00 00 00 00 06 09 03 00 00 00 01
响应： 00 00 00 00 00 05 09 03 02 12 34

(所有的这些请求和响应通过查询 Modicon Quantum PLC 规范采用自动工具来进行校验。

5.1 等级 0 指令详述

5.1.1 读乘法寄存器(FC 3)

请求

Byte 0: FC = 03
Byte 1-2: 参考数值
Byte 3-4: 指令数(1-125)

响应
Byte 0: FC = 03
Byte 1: 响应的字节数 (B=2 x 指令数)
Byte 2-(B+1): Register values

异常
Byte 0: FC = 83 (hex)
Byte 1: 异常代码 = 01 or 02

示例
读参考值为 0 (Modicon 984 中为 40001)时的 1 寄存器得到十六进制的值 1234
03 00 00 00 01 => 03 02 12 34

5.1.2 写乘法寄存器(FC 16)

请求
Byte 0: FC = 10 (hex)
Byte 1-2: 参考数值
Byte 3-4: 指令数 (1-100)
Byte 5: 字节数 (B=2 x word count)
Byte 6-(B+5): 寄存器值

响应
Byte 0: FC = 10 (hex)
Byte 1-2: 参考数值
Byte 3-4: 指令数

异常
Byte 0: FC = 90 (hex)
Byte 1: 异常代码 = 01 or 02

示例

读参考值为 0(Modicon 984 中为 40001)时的 1 寄存器得到十六进制的值 1234
10 00 00 00 01 02 12 34 => 10 00 00 00 01

5.2 等级 1 指令详述

5.2.1 读线圈 (FC 1)

请求
Byte 0: FC = 01
Byte 1-2: 参考数值
Byte 3-4: 比特数(1-2000)
响应
Byte 0: FC = 01
Byte 1: 响应的字节数 (B=(比特数+7)/8)
Byte 2-(B+1): 比特值(最小意义位首先绕线圈!)

异常
Byte 0: FC = 81 (hex)
Byte 1: exception code = 01 or 02

示例
读参考值为 0 (Modicon 984 中为 00001)时的 1 线圈得到的值 1
01 00 00 00 01 => 01 01 01

注意到返回的数据的格式和 big-endian 体系结构不同。而且此请求如果调用乘法指令字且这些指令不以 16 位为界排列，那么该请求将在从站得到计算强化。

5.2.2 读离散输入 (FC 2)

请求
Byte 0: FC = 02
Byte 1-2: 参考数值
Byte 3-4: 比特数 (1-2000)

响应
Byte 0: FC = 02
Byte 1: 响应的字节数 (B=(比特数+7)/8)
Byte 2-(B+1): 比特值 (最小意义位首先绕线圈!)

异常
Byte 0: FC = 82 (16 进制)
Byte 1: 异常代码 = 01 or 02

示例
读参考值为 0 (Modicon 984 中为 10001)时的 1 离散输入得到的值 1
02 00 00 00 01 => 02 01 01

注意到返回的数据的格式和 big-endian 体系结构不同。而且此请求如果调用乘法指令字且这些指令不以 16 位为界排列，那么该请求将在从站得到计算强化。

5.2.3 读输入寄存器 (FC 4)

请求
Byte 0: FC = 04
Byte 1-2: 参考数值
Byte 3-4: 指令数 (1-125)

响应
Byte 0: FC = 04
Byte 1: 响应的比特数 (B=2 x 指令数)
Byte 2-(B+1): 寄存器值

异常
Byte 0: FC = 84 (hex)
Byte 1: 异常代码 = 01 or 02

示例
读参考值为 0 (Modicon 984 中为 30001)时的 1 输入寄存器得到十六进制的值 1234
04 00 00 00 01 => 04 02 12 34

5.2.4 写线圈 (FC 5)

请求
Byte 0: FC = 05
Byte 1-2: 参考数值
Byte 3: = FF 打开线圈, =00 关闭线圈
Byte 4: = 00

响应
Byte 0: FC = 05
Byte 1-2: 参考数值
Byte 3: = FF 打开线圈, =00 关闭线圈(回波)

Byte 4: = 00

异常
Byte 0: FC = 85 (16 进制)
Byte 1: 异常代码 = 01 or 02

示例
将值 1 在参考值为 0（Modicon 984 中为 00001）时写入 1 线圈
05 00 00 FF 00 => 05 00 00 FF 00

5.2.5 写单一寄存器(FC 6)

请求
Byte 0: FC = 06
Byte 1-2: 参考数值
Byte 3-4: 寄存器值

响应
Byte 0: FC = 06
Byte 1-2: 参考数值
Byte 3-4: 寄存器值
异常
Byte 0: FC = 86 (16 进制)
Byte 1: 异常代码= 01 or 02

示例
将十六进制值 1234 在参考值为 0（Modicon 984 中为 40001）时写入 1 线圈
06 00 00 12 34 => 06 00 00 12 34

5.2.6 读异常状态字 (FC 7)

注意“异常状态字”和“异常响应”没有关系。“读异常状态字”消息欲在采用小波特率轮询多点网络的早期 MODBUS 中允许最大的响应速度。PLC’s 将特别规划一个 8 线圈（离散输出）的范围用此消息进行询问。

请求
Byte 0: FC = 07

响应
Byte 0: FC = 07
Byte 1: 异常状态字 (通常预先确定 8 线圈的范围)

异常
Byte 0: FC = 87 (16 进制)
Byte 1: 异常代码 = 01 or 02

示例
读异常状态字得到 16 进制值 34
07 => 07 34

5.3 等级 2 指令详述

5.3.1 强制多点线圈 (FC 15)

请求
Byte 0: FC = 0F (16 进制)
Byte 1-2: 参考数值
Byte 3-4: 比特数 (1-800)
Byte 5: 字节数 (B = (比特数 + 7)/8)
Byte 6-(B+5): 写入的数据 (最小意义位 = 第一个线圈)

响应
Byte 0: FC = 0F (16 进制)
Byte 1-2: 参考数值
Byte 3-4: 比特数

异常
Byte 0: FC = 8F (16 进制)
Byte 1: 异常代码 = 01 or 02

示例
当参考值为 0（在 Modicon 984 中为 00001）时给 3 线圈写入值 0，0，1
0F 00 00 00 03 01 04 => 0F 00 00 00 03

注意到返回的数据的格式和 big-endian 体系结构不同。而且此请求如果调用乘法指令字且这些指令不以 16 位为界排列，那么该请求将在从站得到计算强化。

5.3.2 读一般参考值 (FC 20)

请求
Byte 0: FC = 14 （16 进制）
Byte 1: 请求余项的字节数 (=7 x 组数)
Byte 2: 第一组的参考值类型 = 适合于 6xxxx 扩展寄存外存储器的 06
Byte 3-6: 第一组的参考数值

= 适于 6xxxx 外存储器的存储器偏移量
= 适于 4xxxx 寄存器的 32 位参考数值
Byte 7-8: 第一组的指令
Bytes 9-15: (至于 2-8 字节，适于第二组)
. . .

响应
Byte 0: FC = 14 (16 进制)
Byte 1: 响应的全部字节数
(=组数+ 组的总的字节数)
Byte 2: 第一组的字节数 (B1=1 + (2 x 指令数))
Byte 3: 第一组的参考类型
Byte 4-(B1+2): 第一组的寄存器值
Byte (B1+3): 第二组的字节数 (B2=1 + (2 x 指令数))
Byte (B1+4): 第二组的参考类型
Byte (B1+5)-(B1+B2+2): 第二组的寄存器值
. . .

异常
Byte 0: FC = 94 (16 进制)
Byte 1: 异常代码 = 01 或 02 或 03 或 04

示例
参考值为 1 时读 1 扩展寄存器： 2 （在 Modicon 984 中外存储器 1 偏移量 2）得到 16 进制值 1234
14 07 06 00 01 00 02 00 01 => 14 04 03 06 12 34
(将来)

参考值 0 时读 1 寄存器返回 16 进制值 1234，参考值 5 时读 2 寄存器返回 16 进制值 5678 和 9abc。

14 0E 04 00 00 00 00 00 01 04 00 00 00 05 00 02 => 14 0A 03 04 12 34 05 04 56 78 9A BC

注意传输尺寸限制很难用数学公式精确定义。概括说来，由于缓冲的大小的限制以及考虑到每个请求和响应数据帧的总长度请求和响应的消息尺寸均限于 256 个字节。如果从站由于响应太大而拒绝发送此消息将产生异常类型 04。
5.3.3 写一般参考值(FC 21)

请求
Byte 0: FC = 15 (16 进制)
Byte 1: 请求余额的字节数
Byte 2: 第一组的参考值类型= 6xxxx 扩展寄存器存储器的 06
Byte 3-6: 第一组的参考数值
= 适于 6xxxx 外存储器的存储器偏移量

= 用于 4xxxx 寄存器的 32 位的参考数值
Byte 7-8: 第一组的指令数 (W1)
Byte 9-(8 + 2 x W1): 第一组的寄存器数据

(从字节 2 开始为其它组复制组的数据帧)
. . .

响应
响应是对询问的直接回应
Byte 0: FC = 15 (16 进制)
Byte 1: 请求余额的字节数
Byte 2: 第一组的参考值类型 = 6xxxx 扩展寄存器存储器的 06
Byte 3-6: 第一组的参考数值
= 6xxxx 外存储器的存储器偏移量
=用于 4xxxx 寄存器的 32 位的参考数值
Byte 7-8: 第一组的指令数 (W1)
Byte 9-(8 + 2 x W1): 第一组的寄存器数据

(从字节 2 开始为其它组复制组的数据帧)
. . .

异常
Byte 0: FC = 95 (16 进制)
Byte 1: 异常代码= 01 或 02 或 03 或 04
示例
参考值为 1 时写 1 扩展寄存器： 2 （在 Modicon 984 中外存储器 1 偏移量 2）得到 16 进制值 1234

15 09 06 00 01 00 02 00 01 12 34 => 15 09 06 00 01 00 02 00 01 12 34
(将来)

参考值 0 时写 1 寄存器返回 16 进制值 1234，参考值 5 时写 2 寄存器返回 16 进制值 5678 和 9abc。

15 14 04 00 00 00 00 00 01 12 34 04 00 00 00 05 00 02 56 78 9A BC
?15 14 04 00 00 00 00 00 01 12 34 04 00 00 00 05 00 02 56 78 9A BC

注意传输尺寸限制很难用数学公式精确定义。概括说来，由于缓冲的大小的限制以及考虑到每个请求和响应数据帧的总长度请求和响应的消息尺寸均限于 256 个字节。如果从站由于响应太大而拒绝发送此消息将产生异常类型 04。

5.3.4 掩膜写寄存器 (FC 22)

请求

Byte 0: FC = 16 (16 进制)
Byte 1-2: 参考数值
Byte 3-4: AND 掩膜用于寄存器
Byte 5-6: OR 掩膜用于寄存器

响应
Byte 0: FC = 16 (16 进制)
Byte 1-2: 参考数值
Byte 3-4: AND 掩膜用于寄存器
Byte 5-6: OR 掩膜用于寄存器

异常
Byte 0: FC = 96 (16 进制)
Byte 1: 异常代码 = 01 或 02

示例
在参考值为 0（Modicon 984 中为 40001）时将寄存器的 0-3 位字段改为 16 进制值 4 (AND 用 000F, OR 用 0004)

16 00 00 00 0F 00 04 => 16 00 00 00 0F 00 04

5.3.5 读/写寄存器 (FC 23)

请求
Byte 0: FC = 17 (16 进制)
Byte 1-2: 用于读的参考数值
Byte 3-4: 用于读的指令数 (1-125)
Byte 5-6: 用于写的参考数值
Byte 7-8: 用于写的指令数 (1-100)
Byte 9: 字节数 (B = 2 x 用于写的指令数)
Byte 10-(B+9): 寄存器值

响应
Byte 0: FC = 17 (16 进制)
Byte 1: 字节数 Byte count(B = 2 x 用于读的指令数)
Byte 2-(B+1) 寄存器值

异常
Byte 0: FC = 97 (16 进制)
Byte 1: 异常代码 = 01 或 02

示例
参考值为 3（在 Modicon 984 中为 40004）时写入 1 寄存器 16 进制值 0123，参考值为 0 时读 2 寄存器返回值 0004 和 5678（16 进制）

17 00 00 00 02 00 03 00 01 02 01 23 => 17 04 00 04 56 78

注意如果寄存器交替的进行读写操作，结果是不明确的。一部分设备先写后读，另部分则先读后写。

5.3.6 读 FIFO 队列 (FC 24)

请求
Byte 0: FC = 18 (16 进制)
Byte 1-2: 参考数值

响应
Byte 0: FC = 18 (16 进制)
Byte 1-2: 字节数 (B = 2 + 指令数) (最大 64)
Byte 3-4: 指令数 (FIFO 中累积的指令数) (最大 31)
Byte 5-(B+2): 从 FIFO 前开始的寄存器数据

异常
Byte 0: FC = 98 (16 进制)
Byte 1: 异常代码 = 01 或 02 或 03

示例
读从参考值 0005 （Modicon 984 中为 40006）开始的 FIFO 区段内容，其中包括 2 指令的值 1234 和 5678（16 进制）
18 00 05 => 18 00 06 00 02 12 34 56 78

注意到执行在 984 上的该功能在通用性方面非常有限-假定寄存器的该区段包括含有从 0 到 31 值的计数器，后面还跟着最大到 31 指令字的数据。当该功能完成，该计数器指令字不会象经过 FIFO 操作所期望的回复为 0。

一般说来，这可被看作函数 16-读乘法寄存器的有限子集，既然后者可用来完成所必须的功能性。

6. 异常代码

在出问题的时候，有一系列定义过的异常代码被从站送回。注意到主站会“ 投机地 ”发送指令，利用接收到的成功或异常代码来确定支配设备的哪一个 MODBUS 愿意响应以及从站不同可用数据区的大小。

所有的异常通过添加 0x80 到请求的功能代码来标记，跟随此字节的是一个单一的原因字节如下例所示：

03 12 34 00 01 => 83 02

当索引 0x1234 响应异常类型 2-“ 非法的数据地址 ”时请求读 1 寄存器

异常情况列举如下：
01 非法的功能

对从站来说，在询问过程中收到的功能代码是不允许的行为。这可能是由于功能代码只适用于新近的控制器，而不能在所选的单元使用。也可推断出从站处于错误的状态而发出这样的一种请求，例如未经配置而被要求返回寄存器值。

02 非法的数据地址
对从站来说，在询问过程中收到的数据地址不是允许的地址。更明确一点，参考数值和传输长度的结合是无效的。对于一个有 100 个寄存器的控制器来说，具有偏移 96 和长度 4 的请求将能成功，而具有偏移 96 和长度 5 的请求将产生异常 02。

03 非法的数据值
对从站来说，在询问数据区段所包含的值是不允许的。这推断出在复杂请求余额的结构中的一个错误，例如隐含长度是不正确的。既然 MODBUS 协议不了解一些特殊寄存器的特殊值的意义，因此这并不意味着寄存器中被提交用于存储的数据对象有一个应用程序期望值之外的值，

04 非法的响应长度
指出加外框的请求将产生一个尺寸超出可用 MODBUS 数据尺寸的响应。仅用于由功能所产生的多部分响应，如功能 20 和 21。

05 确认
专用于关联程序设计指令。

06 从站设备忙
专用于关联程序设计指令。

07 否认
专用于关联程序设计指令。

08 存储器奇偶校验错误
专用于关联功能代码 20 和 21，指出扩展文件区没通过一致性检验。

0A 网关通路不可用
专用于关联 Modbus Plus 网关，指出网关未能分配 Modbus Plus 路径以处理请求。通常意味着网关配置错误。

0B 网关目标设备响应失败
专用于关联 Modbus Plus 网关，指出从目标设备未能获得响应。通常意味着设备没有连接到网络上。

附录

A. 客户机和服务器应用指南

本部分的注释不应当作与客户机和服务器的任何特殊的应用捆绑起来。但是，当应用多厂商系统和网关去安装 MODBUS 设备，遵从这些内容将大大减小综合的 “ 疑难 ”。
下面的软件结构假设在熟悉 BSD Sockets 服务器接口基础上，正如用于 UNIX and Windows NT。

A.1 客户机设计

MODBUS/TCP 的设计使客户机的设计尽可能简化。软件的例子在别处有，但是处理事务的基本过程如下所给：

使用 CONNECT（）建立到所需服务器端口 502 的连接。
准备一个 MODBUS 请求，用前面介绍的方法编码。
提交 MODBUS 请求，包括其 6 字节的 MODBUS/TCP 前缀，作为单一的缓冲用 SEND（）传输。

在同一 TCP 连接上等待响应出现。如果希望所考虑的通讯比 TCP 正常报告的快，用选择（）在这一步随意的运行超时指令。

用 RECV（）读取响应的前 6 个字节，它将指出响应消息的实际长度。
用 RECV（）读取响应的剩余字节。

如果接下来没有可能的连接到这个特别的目标，就关闭 TCP 连接以使服务器的资源可以在间歇期为其它客户机服务。允许向客户机开放连接的最大时间是 1 秒。

在超时等待响应的事件中，发布单方面关闭连接指令，打开新的一个连接，重新提交请求。此技术允许客户机控制适时重试，这优于 TCP 默认（设置）所能提供的功能。它也允许可靠备用策略，例如在基干网失效时，用一个总体上独立的通讯网络提交请求给一个备用的 IP 地址。

A.2 服务器设计

MODBUS/TCP 服务器应设计的可以支持多个并发的客户机，甚至在计划内用户只有单一客户时也要能响应（并发）。这就允许客户机高速的顺序关闭和重开连接以对无发送的响应作出快速反应。

如果使用了传统的 TCP 协议组，减小接收和发送缓冲区的尺寸可以节省内存资源。一个采用 UNIX 或 NT 的 TCP 服务器通常分配每个连接 8K 字节或更多的接收缓存以鼓励从如

文件服务器等设备“ 流畅的 ”传送数据。这样的缓冲器空间在 MODBUS/TCP 中没有价值，因为请求和发送的最大尺寸小于 300 字节。通常为附加的连接资源交换存储空间是可行的。

或者多线程或者单线程的模型能用于处理多个连接。在后续章节叙述。

A.2.1 多线程服务器

操作系统或鼓励多线程应用的语言，如 JAVA，能用多线程策略，叙述如下：
用 LISTEN（）等待引入到端口 502 的 TCP 连接。
当收到新的连接请求时，用 ACCEPT（）接受它并产生新的线程来进行连接操作。
在新的线程期间，无限循环的做以下工作：

为 6 字节的 MODBUS/TCP 头部发布一个 RECV（6）请求。不要在此设置超时，而要等待直到一个请求到达或是连接关闭。两种情况都能自动激起线程。

分析这头部。如果频繁出现，如协议字段非 0 或消息长度超过 256 字节，那么单方面关闭连接。这是服务器对隐含 TCP 编码出错的情况的正常反应。

为消息剩下的已知长度的字节发布一个 RECV（）。特别注意发布一个带有这样长度限制的 RECV（）将允许坚持“ 流水线操作 ”请求的客户机。任何这样的流水线操作请求将保留在服务器或是客户端的 TCP 缓冲区，并在当前的的请求得到完全服务后选取。

现在处理引入的 MODBUS 消息，如果必要，挂起当前的线程直到正确的响应计算出来。最终，你或者拥有了一个有效的 MODBUS 消息，或者作为响应的异常消息。

为响应产生 MODBUS/TCP 前缀，从请求的字节 0 和 1 复制“ 事件标识符 ”字段，并重新计算长度字段。

提交包括 MODBUS/TCP 前缀的响应，当作连接上的单一传送缓冲器，用 SEND（）。
Go back and wait for the next 6 byte prefix record.

最后，当客户端选择关闭连接，6 字节前缀的 RECV（）将失效。有序的关闭通常会使 RECV（）字节数回 0。强制关闭可能会产生从 RECV（）返回的错误。在任一情形下，关闭连接并取消当前的线程。

A.2.2 单线程服务器

一些嵌入式的系统和较老的操作系统如 UNIX 和 MS-DOS 鼓励多连接的处理，采用“ SELECT ”访套接字界面。在这样的系统中，不是在它们的线程处理各自并发的请求，而是在一个普通的处理器上作为多态计算机来处理请求。

其结构如下：

将其状态设为“ 空闲 ”来初始化多态计算机。

LISTEN（）用于引入的到端口 502 的 TCP 连接

现在，开始无限循环检查“ LISTEN ”端口和多态计算机如下：

在收取端口，如果收到一个新的连接请求，用 ACCEPT（）接收它并且促使其中一个多态计算机从“ 空闲 ”转入“ 新的请求 ”状态以处理引入的连接。

对每一多态计算机

如果状态是“ 新的请求 ”：

用 SELECT（）来看请求是否到达通常设超时为 0，既然由于该特殊连接处于休止状态，你不愿意挂起该事务。

如果 SELECT（）表明有组出现，象在多线程案例中一样用 RECV（6）读取头部。如果头部出错，关闭连接并设多态计算机为空闲。

如果读取成功且 SELECT（）表明更多的输入可用，读取余下的请求。

如果请求是完整的，改变话路状态为“ 等待响应 ”。

如果 RECV（）返回值表明连接不在使用中，关闭连接并重置多态计算机为“ 空闲 ”。

如果状态是“ 等待响应 ”

看如果申请响应信息可用，建立响应分组，并用 SEND（）发送，严格类似于多线程的情况。设状态为“ 新的请求 ”。

通过在每循环基础上把多个 SELECT（）调用结合为一个单一的调用来优化性能而不影响应用程序的函数结构是可能的。

A.3 必须的及期望的性能

这儿没有处理 MODBUS 或 MODBUS/TCP 事务所必须的响应时间的规范。

这是因为 MODBUS/TCP 被希望用于尽可能最宽的各种通讯情况，从亚毫秒的时延的 I/O 扫描设备到几秒时延的长距离无线连接。

此外，MODBUS 家族设计用于支持网络间的自动转换，通过“ 非智能性的 ”转换网关。这样的设备包括 Schneider 公司的“ Modbus + 网桥以太网 ”，以及从 MODBUS/TCP 到 MODBUS 串行连接的各类设备。这些设备的使用意味着当前 MODBUS 设备的性能和 MODBUS/TCP 的使用是一致的。

一般来说，象 PLC ’ s 这样的设备所展示的“ 扫描 ”行为将在一个扫描周期内对引入的请求作出反应， 20 该周期将在 20 毫秒和 200 毫秒之间变化。

从客户机的观点，时间必须按照通过网络的预期的传输时延来延长，以确定一个“ 合理的 ”响应时间。这个时延对于交换以太网可能是几毫秒，对广域网连接是几百毫秒。

依次，客户机所用的用来发起新的申请重试的“ 超时 ”的时间都应大于预期最大的“ 合理的 ”响应时间。否则，将可能导致网络及终端设备的过度拥塞，从而导致更大的错误。这是必须避免的情况。

因此实际上，用于高性能应用中的客户机超时设定总是有些依赖网络拓扑和客户机的性能。

通过局域以太网扫描 10 个 I/O 设备，将超时设定为 30 毫秒是合理的，每个设备再 1 毫秒后响应。另一方面，当通过网关及串行连接管理慢速的 PLC ’s 使用 1 秒的超时设定值可能更合适，这儿正常的扫描序列在 300 毫秒内完成。

非时间临近的应用系统经常将超时设定值置为正常的 TCP 默许值，在主平台几秒钟后报告通讯失败。

客户机被鼓励关闭和重建仅用于数据访问（非 PLC 程序设计）的 MODBUS/TCP 连接，而且此处的预期时间在下次使用之前是有意义的，比方说超过 1 秒。如果客户机遵从这一原则，具有有限连接资源的服务器将能为大批客户机提供服务，也有利于错误校正策略如可选目标 IP 地址的挑选。应该记得，关闭和重开连接所造成的的额外通讯和 CPU 负荷比得上单个 Modbus 事务所造成的。

B. 非指令数据的编码

在 MODBUS 上传送大批量信息的最有效办法是采用功能代码 3（读寄存器），16（写寄存器），或可能的 23（读/写寄存器）。

虽然这些功能根据它们在 16 位寄存器上的操作来定义，它们能够将任何类型的信息从一台设备转到另一台，只要这些信息能用接近 16 位指令的区段来描述。

早期的 MODBUS-capable PLC ’s 专用于使用 “ big-endian ” 体系结构的计算机。大多数现代的 PLC ’s 基于采用 “ little-endian ” 体系结构的商用微处理器。MODBUS 潜在的被用于在这两种体系之间交换数据这一事实引入一些会迷惑粗心者的微妙之处。

几乎所有不同于原始“ 离散比特 ”和“ 16 位寄存器值 ”的数据类型在引入 little-endian 微处理器后被提出来。因此 MODBUS 上数据类型的表示法遵从了 little-endian 模式，含义

第一个寄存器比特 15 - 0 = 数据对象比特 15 - 0
第二个寄存器比特 15 - 0 = 数据对象比特 31 - 16
第三个寄存器比特 15 - 0 = 数据对象比特 47 - 32
等等

B.1 指令字中的比特数

Modicon PLC ’s 在 984 Ladder Language 中有预先确定的功能，它将一系列临近的寄存器值转换为相等长度的 1-比特 “ 离散 ” 的块。最常用的功能是 BLKM (块移动)。
由于与最初的 big-endian 体系结构保持一致，该离散值从最大有效位开始编号会带来混乱，所有的编号序列从 1 开始，而非从 0。（本手册中的比特编号总是从作为最小意义位的 0 开始，以与现代的软件文件保持一致。）

这样一个指令字 (寄存器值)中

离散 1 将是 bit 15 (值 0x8000)
离散 2 将是 bit 14 (值 0x4000)
离散 3 将是 bit 13 (值 0x2000)
离散 4 将是 bit 12 值 0x1000)
离散 5 将是 bit 11 (值 0x0800)

离散 6 将是 bit 10 (值 0x0400)
离散 7 将是 bit 9 (值 0x0200)
离散 8 将是 bit 8 (值 0x0100)
离散 9 将是 bit 7 (值 0x0080)
离散 10 将是 bit 6 (值 0x0040)
离散 11 将是 bit 5 (值 0x0020)
离散 12 将是 bit 4 (值 0x0010)
离散 13 将是 bit 3 (值 0x0008)
离散 14 将是 bit 2 (值 0x0004)
离散 15 将是 bit 1 (值 0x0002)
离散 16 将是 bit 0 (值 0x0001)

当多于 16 比特时，例如一个 32 点的离散输入模数，离散 1 到 16 将在第一寄存器，离散 17 到 32 将在第二寄存器。

这个编码的约定对于理解在 MODBUS/TCP 上何时处理离散输入和输出设备，什么地方离散编码和 Modicon PLC ’ s 相一致特别重要。

特别的，注意到指令字中的比特的 IEC-1131 编码约定是从 0（最小意义位）到 15（最大意义位），这与离散编码是相反的。

B.2 多指令字变量

原则上，任何能被 “ 投 ” 到 16 位指令字序列的数据结构都可被传输，并且以同样的数据格式到达设备。

以下的 PLC 数据类型应该注意

B.2.1 984 数据类型

984 16-位 无符号整数

正常含义： 整数的位 15 - 0 = 寄存器的位 15 - 0

984 16-位 有符号整数

正常含义： 整数的位 15 - 0 = 寄存器的位 15 - 0

984 ASCII

尽管 PLC ’s 没有这样的电文处理能力，最初的 ladder language 的编者允许用 2 个 ASCII 字符来表示寄存器。第一个字符表示上半字节（比特 15-8），第二个字符表示下半字节（比特 7-0）。特别注意这与现代 PLC ’ s 所用的高级语言如 C 等的使用是相反的。

984 浮点数

Intel 单精度实数

第一寄存器包含 32-位数的 15-0 位（有效位的 15-0 位）
第二寄存器包含 32-位数的 31-16 位（说明部分和有效位的 23-16）

984 单精度十进制无符号数
尽管值的范围局限于 0-9999，数据表示和 16 位无符号整数相同。

984 双精度十进制无符号数
这个数据格式现在很少用，除了旧格式的 4 位十进制表示。

值的范围从 0 到 99999999。第一个寄存器包含了最有意义的 4 位，第二寄存器包含了最无意义的 4 位，每个用 0-9999 范围内的二进制值表示。

B.2.2 IEC-1131 数据类型

所有的 IEC-1131 数据类型在 Modicon PLC ’s 以 little-endian 的形式来表示。例子如下

BYTE
8-位数。
寄存器的 7-0 位 = BYTE 的 7 - 0 位

DINT
32-位数。
第一寄存器的 15-0 位 = DINT 的 15 - 0 位
第二寄存器的 15-0 位 = DINT 的 31-16 位

INT
寄存器的 15-0 位 = INT 的 15 - 0 位
REAL

32-位 Intel 单精度实数。
第一寄存器的 15-0 位 = REAL 的 15 - 0 位 （有效位的 15-0 位）
第二寄存器的 15-0 位 = REAL 的 31-16 位（说明及有效位的 23-16 位）

UDINT
32-位数。
第一寄存器的 15-0 位 = UDINT 的 15 - 0 位
第二寄存器的 15-0 位 = UDINT 的 31-16 位

UINT

寄存器的 15-0 位 = UINT 的 15 - 0 位

对于其它类型，参阅相关的 IEC-1131 程序手册。