

## 前言

### -----串行链路和 TCP/IP 上的 MODBUS 标准介绍

该标准包括两个通信规程中使用的 MODBUS 应用层协议和服务规范：

- 串行链路上的 MODBUS

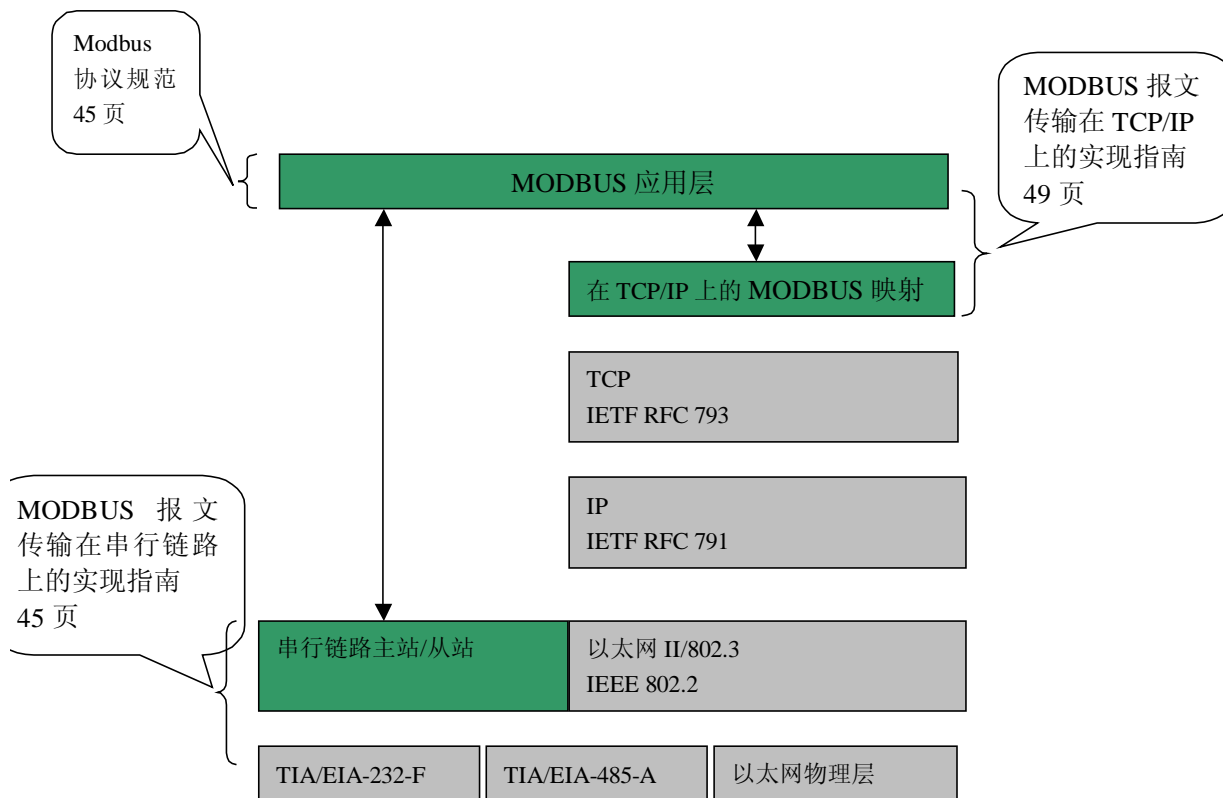
MODBUS 串行链路取决于 TIA/EIA 标准：232-F 和 485-A。

- TCP/IP 上的 MODBUS

MODBUS TCP/IP 取决于 IETF 标准：RFC793 和 RFC791 有关。

串行链路和 TCP/IP 上的 MODBUS 是根据相应 ISO 层模型说明的两个通信规程。

下图强调指出了该标准的主要部分。绿色方框表示规范。灰色方框表示已有的国际标准（TIA/EIA 和 IETF 标准）。



MODBUS 标准分为三部分。第一部分（“Modbus 协议规范”）描述了 MODBUS 事物处理。第二部分（“MODBUS 报文传输在 TCP/IP 上的实现指南”）提供了一个有助于开发者实现 TCP/IP 上的 MODBUS 应用层的参考信息。第三部分（“MODBUS 报文传输在串行链路上的实现指南”）提供了一个有助于开发者实现串行链路上的 MODBUS 应用层的参考信息。

# 第一部分：Modbus 协议

欢迎来到控制中文网

<http://www.cechinamag.com>

# 第一部分：Modbus 协议

## 1 引言

### 1.1 范围

MODBUS 是 OSI 模型第 7 层上的应用层报文传输协议，它在连接至不同类型总线或网络的设备之间提供客户机/服务器通信。

自从 1979 年出现工业串行链路的事实标准以来，MODBUS 使成千上万的自动化设备能够通信。目前，继续增加对简单而雅观的 MODBUS 结构支持。互联网组织能够使 TCP/IP 栈上的保留系统端口 502 访问 MODBUS。

MODBUS 是一个请求/应答协议，并且提供功能码规定的服务。MODBUS 功能码是 MODBUS 请求/应答 PDU 的元素。本文件的作用是描述 MODBUS 事务处理框架内使用的功能码。

### 1.2 规范性引用文件

1. RFC791, 互联网协议, Sep81 DARPA
2. MODBUS 协议参考指南 Rev J,MODICON, 1996 年 6 月, doc#PI\_MBUS\_300

MODBUS 是一项应用层报文传输协议，用于在通过不同类型的总线或网络连接的设备之间的客户机/服务器通信。

目前，使用下列情况实现 MODBUS：

以太网上的 TCP/IP。

各种媒体（有线：EIA/TIA-232-E、EIA-422、EIA/TIA-485-A；光纤、无线等等）上的异步串行传输。

MODBUS PLUS，一种高速令牌传递网络。

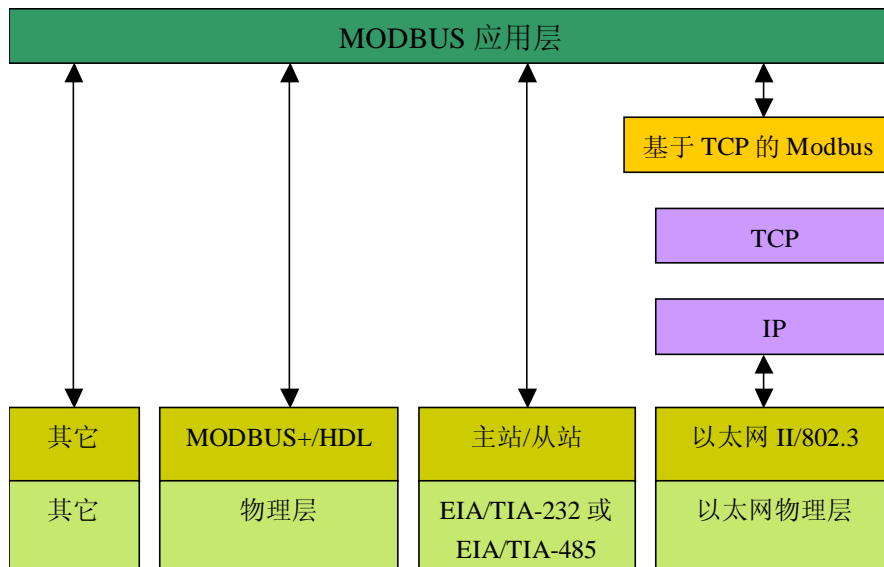


图 1：MODBUS 通信栈

## 2 缩略语

ADU 应用数据单元

- HDLC 高级数据链路控制
- HMI 人机界面
- IETF 因特网工程工作组
- I/O 输入/输出设备
- IP 互连网协议
- MAC 介质访问控制
- MB MODBUS 协议
- MBAP MODBUS 协议
- PDU 协议数据单元
- PLC 可编程逻辑控制器
- TCP 传输控制协议

### 3 背景概要

MODBUS 协议允许在各种网络体系结构内进行简单通信。

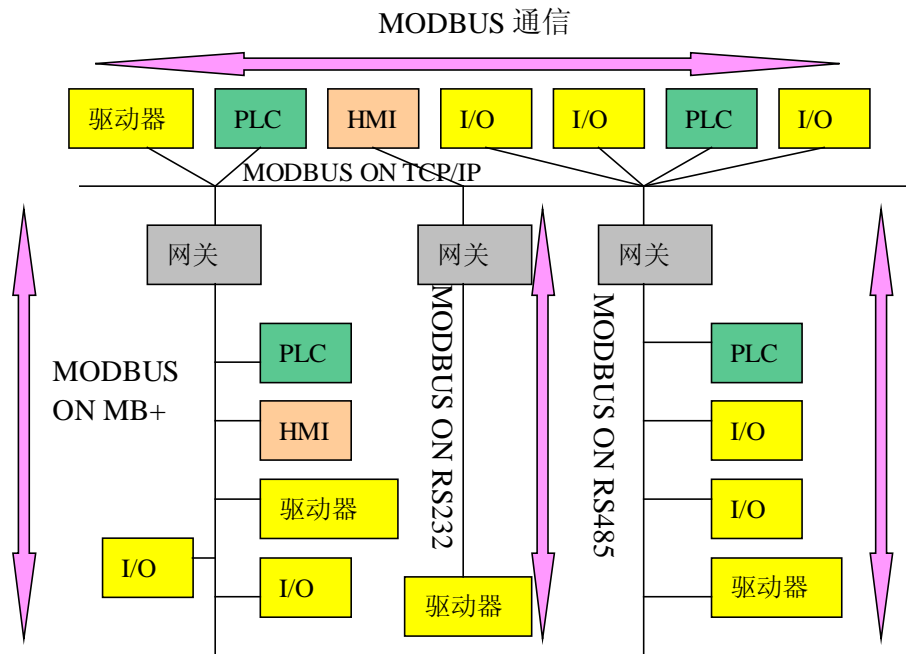


图 2: MODBUS 网络体系结构的实例

每种设备（PLC、HMI、控制面板、驱动程序、动作控制、输入/输出设备）都能使用 MODBUS 协议来启动远程操作。

在基于串行链路和以太 TCP/IP 网络的 MODBUS 上可以进行相同通信。

一些网关允许在几种使用 MODBUS 协议的总线或网络之间进行通信。

### 4 总体描述

#### 4.1 协议描述

MODBUS 协议定义了一个与基础通信层无关的简单协议数据单元（PDU）。特定总线或网络上的 MODBUS 协议映射能够在应用数据单元（ADU）上引入一些附加域。

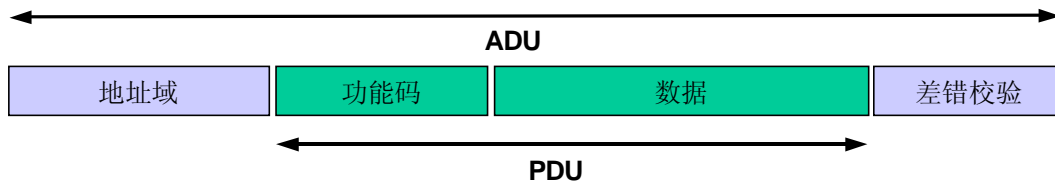


图 3: 通用 MODBUS 帧

启动 MODBUS 事务处理的客户机创建 MODBUS 应用数据单元。功能码向服务器指示将执行哪种操作。

MODBUS 协议建立了客户机启动的请求格式。

用一个字节编码 MODBUS 数据单元的功能码域。有效的码字范围是十进制 1-255 (128-255 为异常响应保留)。当从客户机向服务器设备发送报文时, 功能码域通知服务器执行哪种操作。

向一些功能码加入子功能码来定义多项操作。

从客户机向服务器设备发送的报文数据域包括附加信息, 服务器使用这个信息执行功能码定义的操作。这个域还包括离散项目和寄存器地址、处理的项目数量以及域中的实际数据字节数。

在某种请求中, 数据域可以是不存在的 (0 长度), 在此情况下服务器不需要任何附加信息。功能码仅说明操作。

如果在一个正确接收的 MODBUS ADU 中, 不出现与请求 MODBUS 功能有关的差错, 那么服务器至客户机的响应数据域包括请求数据。如果出现与请求 MODBUS 功能有关的差错, 那么域包括一个异常码, 服务器应用能够使用这个域确定下一个执行的操作。

例如, 客户机能够读一组离散量输出或输入的开/关状态, 或者客户机能够读/写一组寄存器的数据内容。

当服务器对客户机响应时, 它使用功能码域来指示正常 (无差错) 响应或者出现某种差错 (称为异常响应)。对于一个正常响应来说, 服务器仅对原始功能码响应。

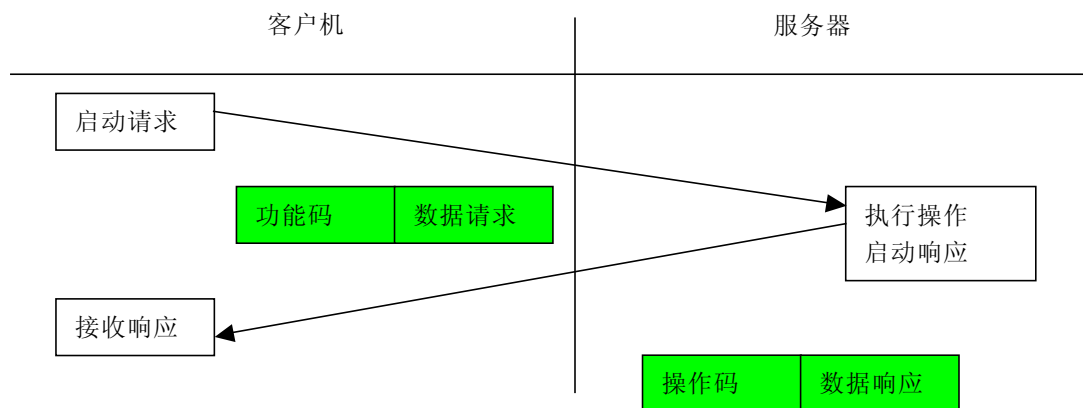


图 4: MODBUS 事务处理 (无差错)

对于异常响应, 服务器返回一个与原始功能码等同的码, 设置该原始功能码的最高有效位为逻辑 1。

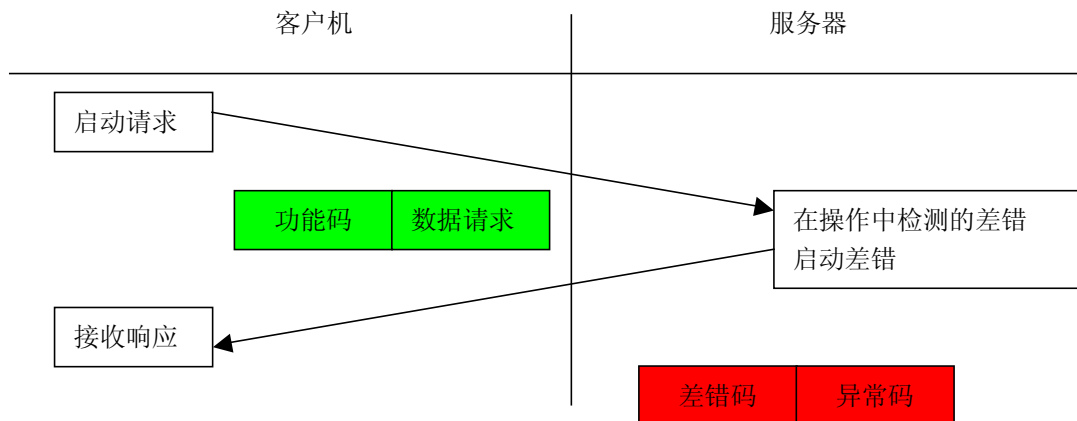


图 5 MODBUS 事务处理（异常响应）

**F 注释：**需要管理超时，以便明确地等待可能不会出现的应答。

串行链路上第一个 MODBUS 执行的长度约束限制了 MODBUS PDU 大小(最大 RS485 ADU=256 字节)。

因此，对串行链路通信来说，MODBUS PDU=256-服务器地址（1 字节）-CRC（2 字节）=253 字节。

从而：

RS232 / RS485 ADU = 253 字节+服务器地址(1 byte) + CRC (2 字节) = 256 字节。

TCP MODBUS ADU = 249 字节+ MBAP (7 字节) = 256 字节。

MODBUS 协议定义了三种 PDU。它们是：

- I MODBUS 请求 PDU, mb\_req\_pdu
- I MODBUS 响应 PDU, mb\_rsp\_pdu
- I MODBUS 异常响应 PDU, mb\_excep\_rsp\_pdu

定义 mb\_req\_pdu 为：

mb\_req\_pdu = { function\_code, request\_data }，其中

function\_code - [1 个字节] MODBUS 功能码

request\_data - [n 个字节]，这个域与功能码有关，并且通常包括诸如可变参考、变量、数据偏移量、子功能码等信息。

定义 mb\_rsp\_pdu 为：

mb\_rsp\_pdu = { function\_code, response\_data }，其中

function\_code - [1 个字节] MODBUS 功能码

response\_data - [n 个字节]，这个域与功能码有关，并且通常包括诸如可变参考、变量、数据偏移量、子功能码等信息。

定义 mb\_excep\_rsp\_pdu 为：

mb\_excep\_rsp\_pdu = { function\_code, request\_data }，其中

function\_code - [1 个字节] MODBUS 功能码 + 0x80

exception\_code - [1 个字节]，在下表中定义了 MODBUS 异常码。

#### 4.2 数据编码

I MODBUS 使用一个‘big-Endian’表示地址和数据项。这意味着当发射多个字节时，首先发送最高有效位。例如：

寄存器大小            值

16- 比特            0x1234            发送的第一字节为    0x12            然后 0x34

**F 注释：**更详细的信息参见[1]。

### 4.3 MODBUS 数据模型

MODBUS 以一系列具有不同特征表格上的数据模型为基础。四个基本表格为：

| 基本表格  | 对象类型   | 访问类型 | 内容             |
|-------|--------|------|----------------|
| 离散量输入 | 单个比特   | 只读   | I/O 系统提供这种类型数据 |
| 线圈    | 单个比特   | 读写   | 通过应用程序改变这种类型数据 |
| 输入寄存器 | 16-比特字 | 只读   | I/O 系统提供这种类型数据 |
| 保持寄存器 | 16-比特字 | 读写   | 通过应用程序改变这种类型数据 |

输入与输出之间以及比特寻址的和字寻址的数据项之间的区别并没有暗示任何应用操作。如果这是对可疑对象核心部分最自然的解释，那么这种区别是可完全接受的，而且很普通，以便认为四个表格全部覆盖了另外一个表格。

对于基本表格中任何一项，协议都允许单个地选择 65536 个数据项，而且设计那些项的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。

很显然，必须将通过 MODBUS 处理的所有数据放置在设备应用存储器中。但是，存储器的物理地址不应该与数据参考混淆。要求仅仅是数据参考与物理地址的链接。

MODBUS 功能码中使用的 MODBUS 逻辑参考数字是以 0 开始的无符号整数索引。

#### I MODBUS 模型实现的实例

下例实例示出了两种在设备中构造数据的方法。可能有不同的结构，这个文件中没有全部描述出来。每个设备根据其应用都有它自己的数据结构。

##### 实例 1：有 4 个独立块的设备

下例实例示出了设备中的数据结构，这个设备含有数字量和模拟量、输入量和输出量。由于不同块中的数据不相关，每个块是相互独立。按不同 MODBUS 功能码访问每个块。

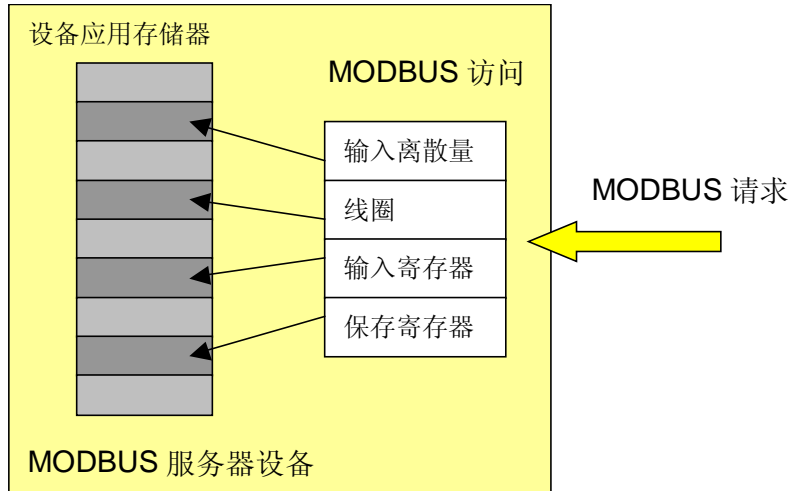


图 6: 带有独立块的 MODBUS 数据模型

**实例 2: 仅有 1 个块的设备**

在这个实例中, 设备仅有 1 个数据块。通过几个 MODBUS 功能码可能得到一个相同数据, 或者通过 16 比特访问或 1 个访问比特。

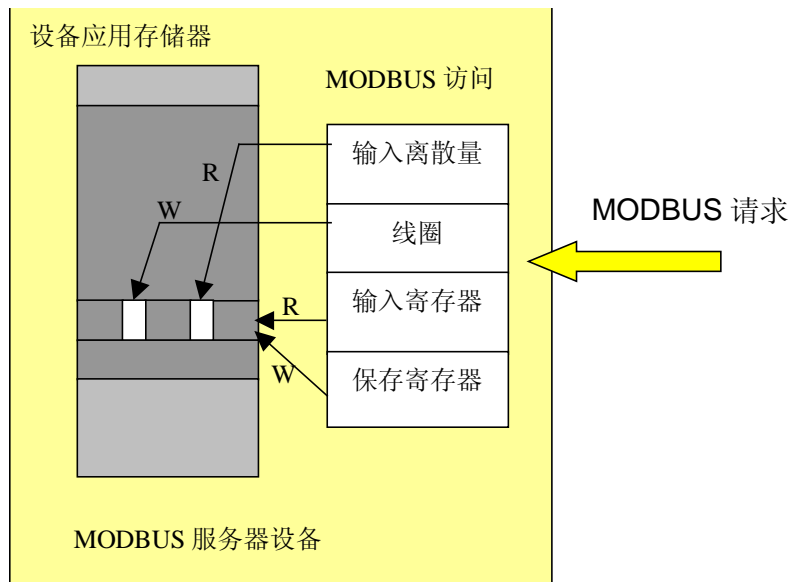


图 7: 仅带有 1 个块的 MODBUS 数据模型

**4.4 MODBUS 事务处理的定义**

下列状态图描述了在服务器侧 MODBUS 事务处理的一般处理过程。



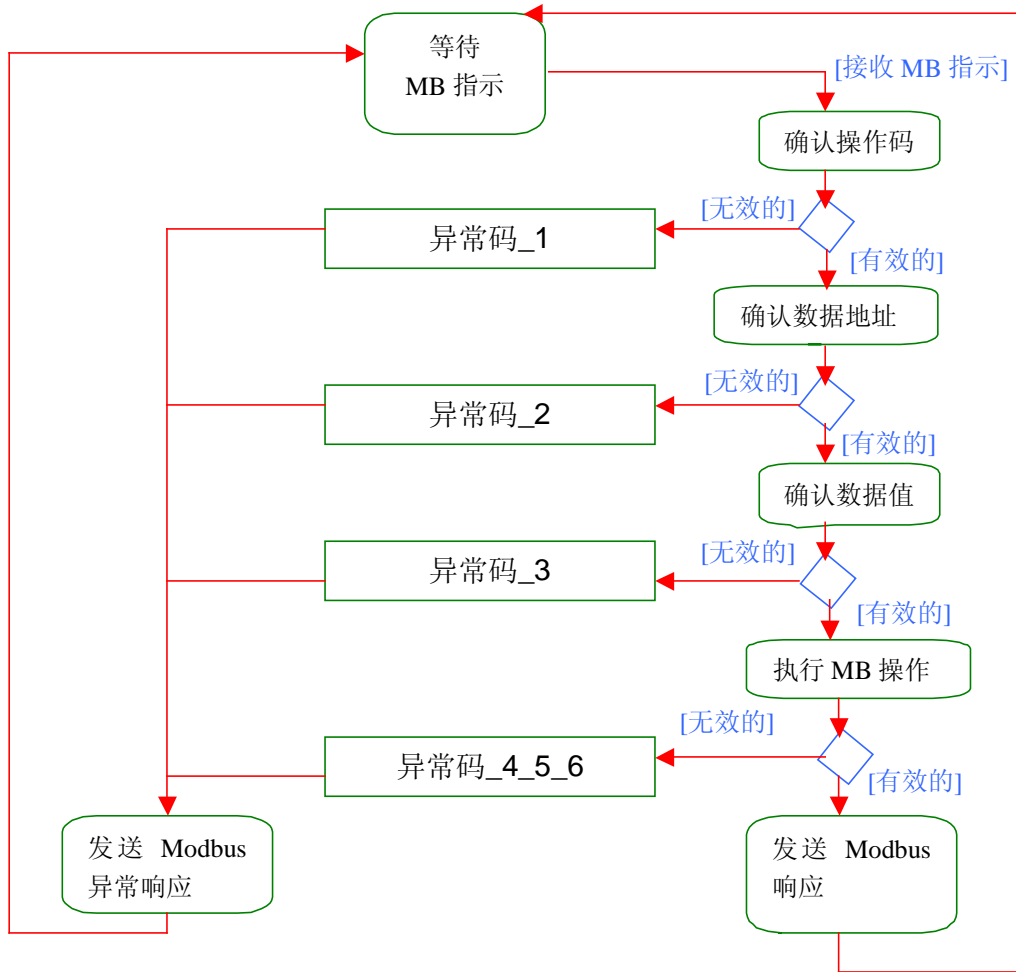


图 8: MODBUS 事务处理的状态图

一旦服务器处理请求，使用合适的 MODBUS 服务器事务建立 MODBUS 响应。  
根据处理结果，可以建立两种类型响应：

- I 一个正 MODBUS 响应：
  - 响应功能码 = 请求功能码
- I 一个 MODBUS 异常响应(参见第 6.14 节)：
  - I 用来为客户机提供处理过程中与被发现的差错相关的信息；
  - I 响应功能码 = 请求功能码 + 0x80；
  - I 提供一个异常码来指示差错原因。

## 5 功能码分类

有三类 MODBUS 功能码。它们是：

### 公共功能码

- I 是较好地定义的功能码，
- I 保证是唯一的，
- I MODBUS 组织可改变的，
- I 公开证明的，
- I 具有可用的一致性测试，
- I MB IETF RFC 中证明的，

- 包含已被定义的公共指配功能码和未来使用的未指配保留供功能码。

**用户定义功能码**

- 有两个用户定义功能码的定义范围，即 65 至 72 和十进制 100 至 110。
- 用户没有 MODBUS 组织的任何批准就可以选择和实现一个功能码
- 不能保证被选功能码的使用是唯一的。
- 如果用户要重新设置功能作为一个公共功能码，那么用户必须启动 RFC，以便将改变引入公共分类中，并且指配一个新的公共功能码。

**保留功能码**

- 一些公司对传统产品通常使用的功能码，并且对公共使用是无效的功能码。

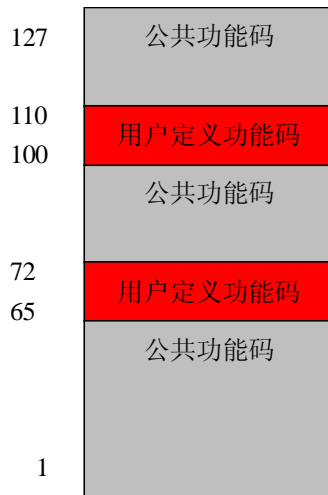


图 9: MODBUS 功能码分类

### 5.1 公共功能码定义

|      |         |               |          | 功能码    |    | (十六进制)             | 页                  |  |
|------|---------|---------------|----------|--------|----|--------------------|--------------------|--|
|      |         |               |          | 码      | 子码 |                    |                    |  |
| 数据访问 | 比特访问    | 物理离散量输入       | 读输入离散量   | 02     |    | 02                 | <a href="#">11</a> |  |
|      |         | 内部比特或物理线圈     | 读线圈      | 01     |    | 01                 | <a href="#">10</a> |  |
|      |         |               | 写单个线圈    | 05     |    | 05                 | <a href="#">16</a> |  |
|      |         |               | 写多个线圈    | 15     |    | 0F                 | <a href="#">37</a> |  |
|      | 16 比特访问 | 输入存储器         | 读输入寄存器   | 04     |    | 04                 | <a href="#">14</a> |  |
|      |         | 内部存储器或物理输出存储器 | 读多个寄存器   | 03     |    | 03                 | <a href="#">13</a> |  |
|      |         |               | 写单个寄存器   | 06     |    | 06                 | <a href="#">17</a> |  |
|      |         |               | 写多个寄存器   | 16     |    | 10                 | <a href="#">39</a> |  |
|      |         |               | 读/写多个寄存器 | 23     |    | 17                 | <a href="#">47</a> |  |
|      |         |               | 屏蔽写寄存器   | 22     |    | 16                 | <a href="#">46</a> |  |
|      |         | 文件记录访问        | 读文件记录    | 20     | 6  | 14                 | <a href="#">42</a> |  |
|      | 写文件记录   |               | 21       | 6      | 15 | <a href="#">44</a> |                    |  |
|      | 封装接口    |               |          | 读设备识别码 | 43 | 14                 | 2B                 |  |

## 6 功能码描述

### 6.1 01 (0x01) 读线圈

在一个远程设备中，使用该功能码读取线圈的 1 至 2000 连续状态。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编号。从零开始寻址线圈。因此寻址线圈 1-16 为 0-15。

根据数据域的每个比特将响应报文中的线圈分成为一个线圈。指示状态为 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顺序。

如果返回的输出数量不是八的倍数，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完整字节数。

#### 请求 PDU

|      |       |                  |
|------|-------|------------------|
| 功能码  | 1 个字节 | 0x01             |
| 起始地址 | 2 个字节 | 0x0000 至 0xFFFF  |
| 线圈数量 | 2 个字节 | 1 至 2000 (0x7D0) |

#### 响应 PDU

|      |       |           |
|------|-------|-----------|
| 功能码  | 1 个字节 | 0x01      |
| 字节数  | 1 个字节 | N*        |
| 线圈状态 | N 个字节 | n=N 或 N+1 |

\*N=输出数量/8，如果余数不等于0，那么N = N+1

**错误**

|     |       |                   |
|-----|-------|-------------------|
| 功能码 | 1 个字节 | 功能码+0x80          |
| 异常码 | 1 个字节 | 01 或 02 或 03 或 04 |


这是一个请求读离散量输出 20-38 的实例：

| 请求      |        | 响应         |        |
|---------|--------|------------|--------|
| 域名      | (十六进制) | 域名         | (十六进制) |
| 功能      | 01     | 功能         | 01     |
| 起始地址 Hi | 00     | 字节数        | 03     |
| 起始地址 Lo | 13     | 输出状态 27-20 | CD     |
| 输出数量 Hi | 00     | 输出状态 35-28 | 6B     |
| 输出数量 Lo | 13     | 输出状态 38-36 | 05     |

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 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 27 至 20。下一个字节的输出从左到右为 35 至 28。当串行发射比特时，从 LSB 向 MSB 传输：20... 27、28... 35 等等。

在最后的的数据字节中，将输出状态 38-36 表示为十六进制字节值 05，或二进制 0000 0101。输出 38 是左侧第六个比特位置，输出 36 是这个字节的 LSB。用零填充五个剩余高位比特。

 **注：**用零填充五个剩余比特（一直到高位端）。

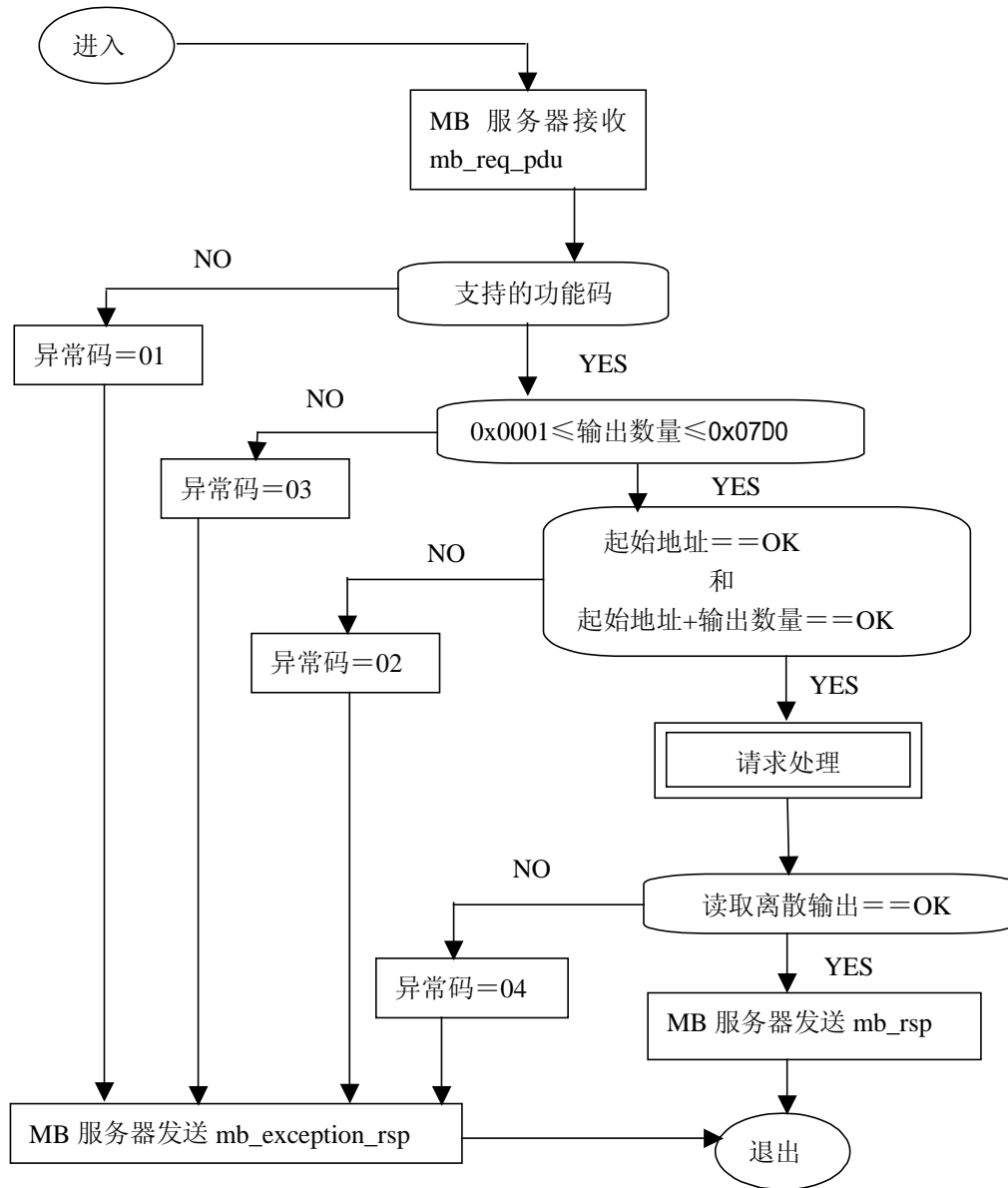


图 10: 读取线圈状态图

## 6.2 02 (0x02) 读离散量输入

在一个远程设备中，使用该功能码读取离散量输入的 1 至 2000 连续状态。请求 PDU 详细说明了起始地址，即指定的第一个输入地址和输入编号。从零开始寻址输入。因此寻址输入 1-16 为 0-15。

根据数据域的每个比特将响应报文中的离散量输入分成一个输入。指示状态为 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入。其它输入依次类推，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顺序。

如果返回的输入数量不是八的倍数，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完整字节数。

**请求 PDU**

|      |       |                  |
|------|-------|------------------|
| 功能码  | 1 个字节 | 0x02             |
| 起始地址 | 2 个字节 | 0x0000 至 0xFFFF  |
| 输入数量 | 2 个字节 | 1 至 2000 (0x7D0) |

**响应 PDU**

|      |          |      |
|------|----------|------|
| 功能码  | 1 个字节    | 0x82 |
| 字节数  | 1 个字节    | N*   |
| 输入状态 | N*×1 个字节 |      |

\*N=输出数量/8，如果余数不等于 0，那么N = N+1

**错误**

|     |      |                   |
|-----|------|-------------------|
| 差错码 | 1 字节 | 0x82              |
| 异常码 | 1 字节 | 01 或 02 或 03 或 04 |

这是一个请求读取离散量输入 197-218 的实例：

| 请求      |        | 响应           |        |
|---------|--------|--------------|--------|
| 域名      | (十六进制) | 域名           | (十六进制) |
| 功能      | 02     | 功能           | 02     |
| 起始地址 Hi | 00     | 字节数          | 03     |
| 起始地址 Lo | C4     | 输入状态 204-197 | AC     |
| 输出数量 Hi | 00     | 输入状态 212-205 | DB     |
| 输出数量 Lo | 16     | 输入状态 218-213 | 35     |

将离散量输入状态 204-197 表示为十六进制字节值 AC，或二进制 1010 1100。输入 204 是这个字节的 MSB，输入 197 是这个字节的 LSB。

将离散量输入状态 218-213 表示为十六进制字节值 35，或二进制 0011 0101。输入 218 位于左侧第 3 比特，输入 213 是 LSB。



**注：**用零填充 2 个剩余比特（一直到高位端）。

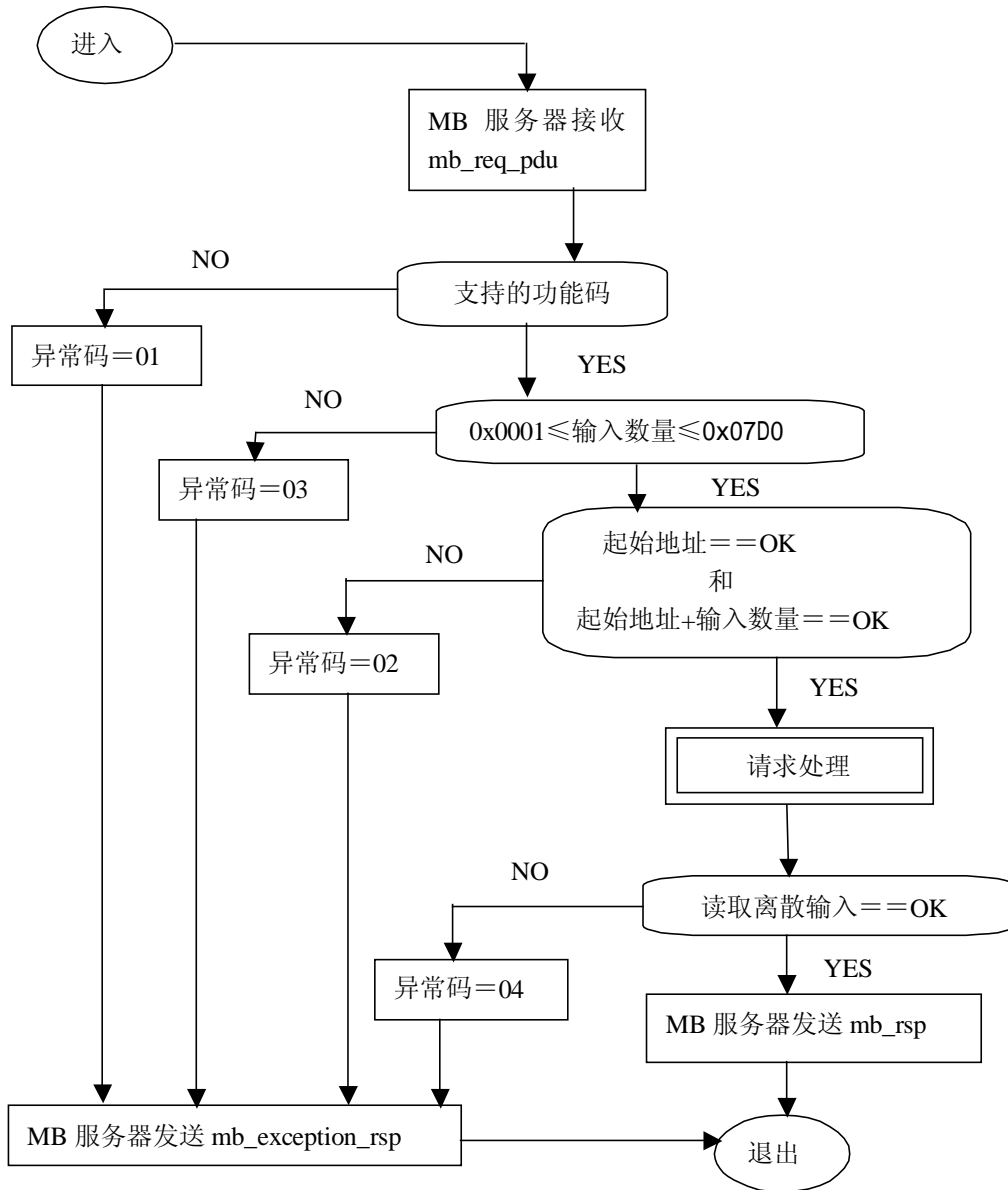


图 11: 读离散量输入的状态图

### 6.3 03 (0x03) 读保持寄存器

在一个远程设备中，使用该功能码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。从零开始寻址寄存器。因此，寻址寄存器 1-16 为 0-15。

将响应报文中的寄存器数据分成每个寄存器有两字节，在每个字节中直接地调整二进制内容。

对于每个寄存器，第一个字节包括高位比特，并且第二个字节包括低位比特。

#### 请求

|       |       |                 |
|-------|-------|-----------------|
| 功能码   | 1 个字节 | 0x03            |
| 起始地址  | 2 个字节 | 0x0000 至 0xFFFF |
| 寄存器数量 | 2 个字节 | 1 至 125 (0x7D)  |

**响应**

|      |                 |             |
|------|-----------------|-------------|
| 功能码  | 1 个字节           | <b>0x03</b> |
| 字节数  | 1 个字节           | <b>2×N*</b> |
| 寄存器值 | <b>N*×2 个字节</b> |             |

\*N=寄存器的数量

**错误**

|     |       |                   |
|-----|-------|-------------------|
| 差错码 | 1 个字节 | <b>0x83</b>       |
| 异常码 | 1 个字节 | 01 或 02 或 03 或 04 |

这是一个请求读寄存器 108-110 的实例：

| 请求     |        | 响应            |        |
|--------|--------|---------------|--------|
| 域名     | (十六进制) | 域名            | (十六进制) |
| 功能     | 03     | 功能            | 03     |
| 高起始地址  | 00     | 字节数           | 06     |
| 低起始地址  | 6B     | 寄存器值 Hi (108) | 02     |
| 高寄存器编号 | 00     | 寄存器值 Lo (108) | 2B     |
| 低寄存器编号 | 03     | 寄存器值 Hi (109) | 00     |
|        |        | 寄存器值 Lo (109) | 00     |
|        |        | 寄存器值 Hi (110) | 00     |
|        |        | 寄存器值 Lo (110) | 64     |

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。



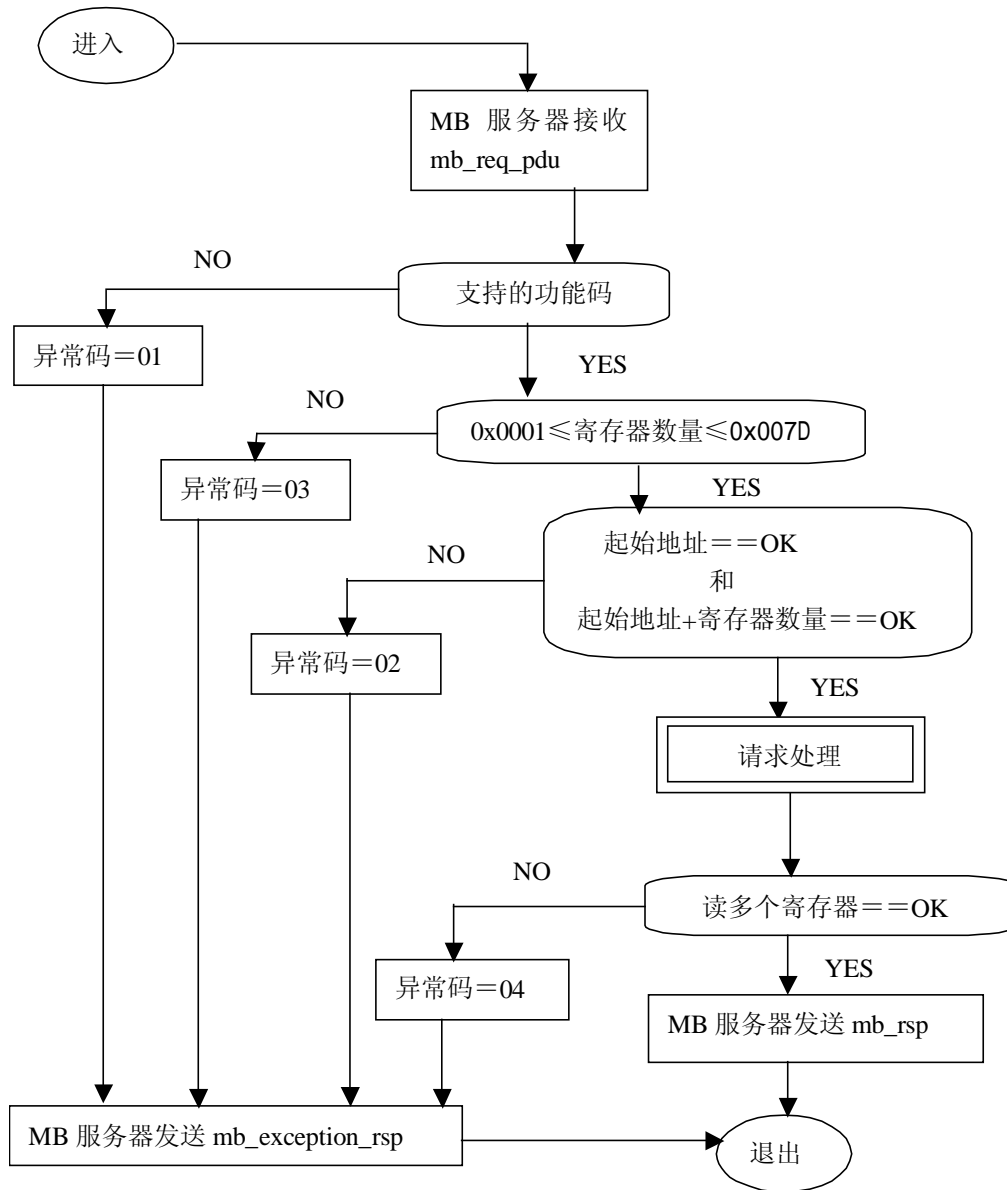


图 12: 读保持寄存器的状态图

#### 6.4 04(0x04) 读输入寄存器

在一个远程设备中,使用该功能码读取 1 至大约 125 的连续输入寄存器。请求 PDU 说明了起始地址和寄存器数量。从零开始寻址寄存器。因此,寻址输入寄存器 1-16 为 0-15。

将响应报文中的寄存器数据分成每个寄存器为两字节,在每个字节中直接地调整二进制内容。对于每个寄存器,第一个字节包括高位比特,并且第二个字节包括低位比特。

##### 请求

|         |       |                 |
|---------|-------|-----------------|
| 功能码     | 1 个字节 | <b>0x04</b>     |
| 起始地址    | 2 个字节 | 0x0000 至 0xFFFF |
| 输入寄存器数量 | 2 个字节 | 0x0001 至 0x007D |

**响应**

|       |          |             |
|-------|----------|-------------|
| 功能码   | 1 个字节    | <b>0x04</b> |
| 字节数   | 1 个字节    | 2×N*        |
| 输入寄存器 | N*×2 个字节 |             |

\*N=输入寄存器的数量

**错误**

|     |       |                   |
|-----|-------|-------------------|
| 差错码 | 1 个字节 | <b>0x84</b>       |
| 异常码 | 1 个字节 | 01 或 02 或 03 或 04 |

这是一个请求读输入寄存器 9 的实例：

| 请求         |        | 响应         |        |
|------------|--------|------------|--------|
| 域名         | (十六进制) | 域名         | (十六进制) |
| 功能         | 04     | 功能         | 04     |
| 起始地址 Hi    | 00     | 字节数        | 02     |
| 起始地址 Lo    | 08     | 输入寄存器 9 Hi | 00     |
| 输入寄存器数量 Hi | 00     | 输入寄存器 9 Lo | 0A     |
| 输入寄存器数量 Lo | 01     |            |        |

将输入寄存器 9 的内容表示为两个十六进制字节值 00 0A，或十进制 10。

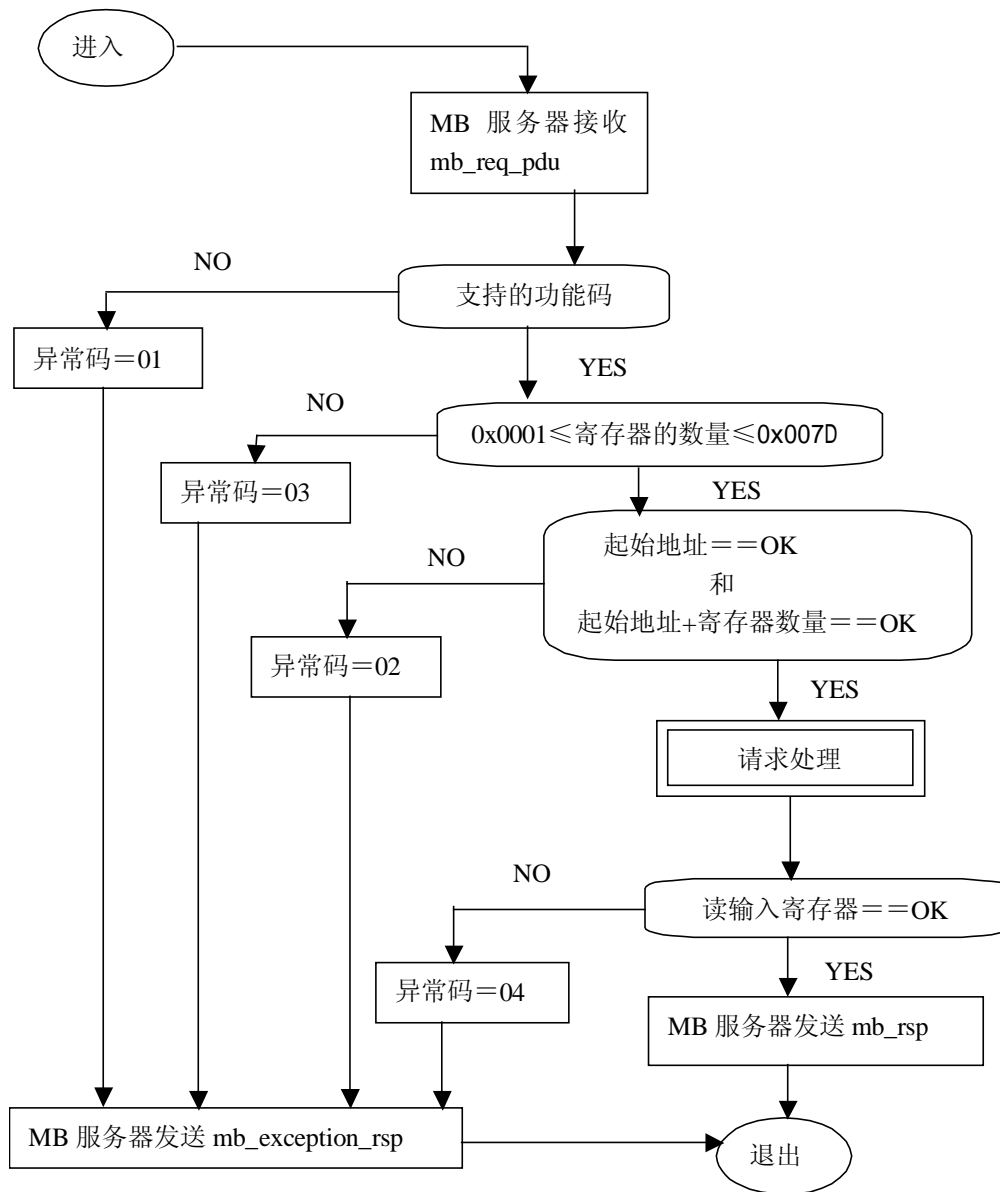


图 13: 读输入寄存器的状态图

### 6.5 05 (0x05) 写单个线圈

在一个远程设备上，使用该功能码写单个输出为 ON 或 OFF。

请求数据域中的常量说明请求的 ON/OFF 状态。十六进制值 FF 00 请求输出为 ON。十六进制值 00 00 请求输出为 OFF。其它所有值均是非法的，并且对输出不起作用。

请求 PDU 说明了强制的线圈地址。从零开始寻址线圈。因此，寻址线圈 1 为 0。线圈值域的常量说明请求的 ON/OFF 状态。十六进制值 0XFF00 请求线圈为 ON。十六进制值 0X0000 请求线圈为 OFF。其它所有值均为非法的，并且对线圈不起作用。

正常响应是请求的应答，在写入线圈状态之后返回这个正常响应。

**请求**

|      |       |                 |
|------|-------|-----------------|
| 功能码  | 1 个字节 | <b>0x05</b>     |
| 输出地址 | 2 个字节 | 0x0000 至 0xFFFF |
| 输出值  | 2 个字节 | 0x0000 至 0x00   |

**响应**

|      |       |                 |
|------|-------|-----------------|
| 功能码  | 1 个字节 | <b>0x05</b>     |
| 输出地址 | 2 个字节 | 0x0000 至 0xFFFF |
| 输出值  | 2 个字节 | 0x0000 至 0xFF00 |

**错误**

|     |       |                   |
|-----|-------|-------------------|
| 差错码 | 1 个字节 | <b>0x85</b>       |
| 异常码 | 1 个字节 | 01 或 02 或 03 或 04 |

这是一个请求写线圈 173 为 ON 的实例：

| 请求      |        | 响应      |        |
|---------|--------|---------|--------|
| 域名      | (十六进制) | 域名      | (十六进制) |
| 功能      | 05     | 功能      | 05     |
| 输出地址 Hi | 00     | 输出地址 Hi | 00     |
| 输出地址 Lo | AC     | 输出地址 Lo | AC     |
| 输出值 Hi  | FF     | 输出值 Hi  | FF     |
| 输出值 Lo  | 00     | 输出值 Lo  | 00     |

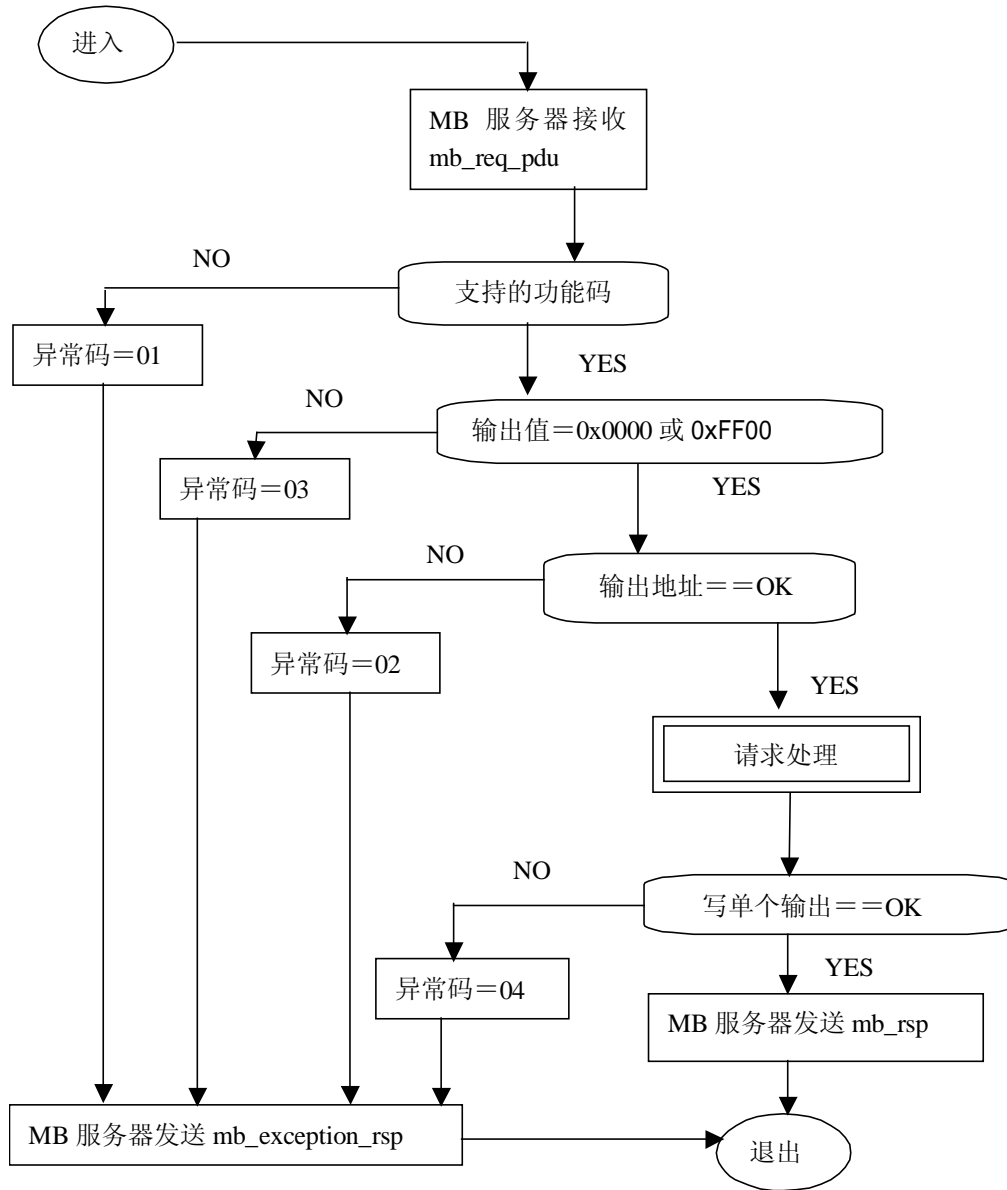


图 14: 写单个输出状态图

### 6.6 06 (0x06) 写单个寄存器

在一个远程设备中，使用该功能码写单个保持寄存器。

请求 PDU 说明了被写入寄存器的地址。从零开始寻址寄存器。因此，寻址寄存器 1 为 0。

正常响应是请求的应答，在写入寄存器内容之后返回这个正常响应。

#### 请求

|       |       |                 |
|-------|-------|-----------------|
| 功能码   | 1 个字节 | 0x06            |
| 寄存器地址 | 2 个字节 | 0x0000 至 0xFFFF |
| 寄存器值  | 2 个字节 | 0x0000 至 0xFFFF |

**响应**

|       |       |                 |
|-------|-------|-----------------|
| 功能码   | 1 个字节 | <b>0x06</b>     |
| 寄存器地址 | 2 个字节 | 0x0000 至 0xFFFF |
| 寄存器值  | 2 个字节 | 0x0000 至 0xFFFF |

**错误**

|     |       |                   |
|-----|-------|-------------------|
| 差错码 | 1 个字节 | <b>0x86</b>       |
| 异常码 | 1 个字节 | 01 或 02 或 03 或 04 |

这是一个请求将十六进制 00 03 写入寄存器 2 的实例：

| 请求       |        | 响应      |        |
|----------|--------|---------|--------|
| 域名       | (十六进制) | 域名      | (十六进制) |
| 功能       | 06     | 功能      | 06     |
| 寄存器地址 Hi | 00     | 输出地址 Hi | 00     |
| 寄存器地址 Lo | 01     | 输出地址 Lo | 01     |
| 寄存器值 Hi  | 00     | 输出值 Hi  | 00     |
| 寄存器值 Lo  | 03     | 输出值 Lo  | 03     |

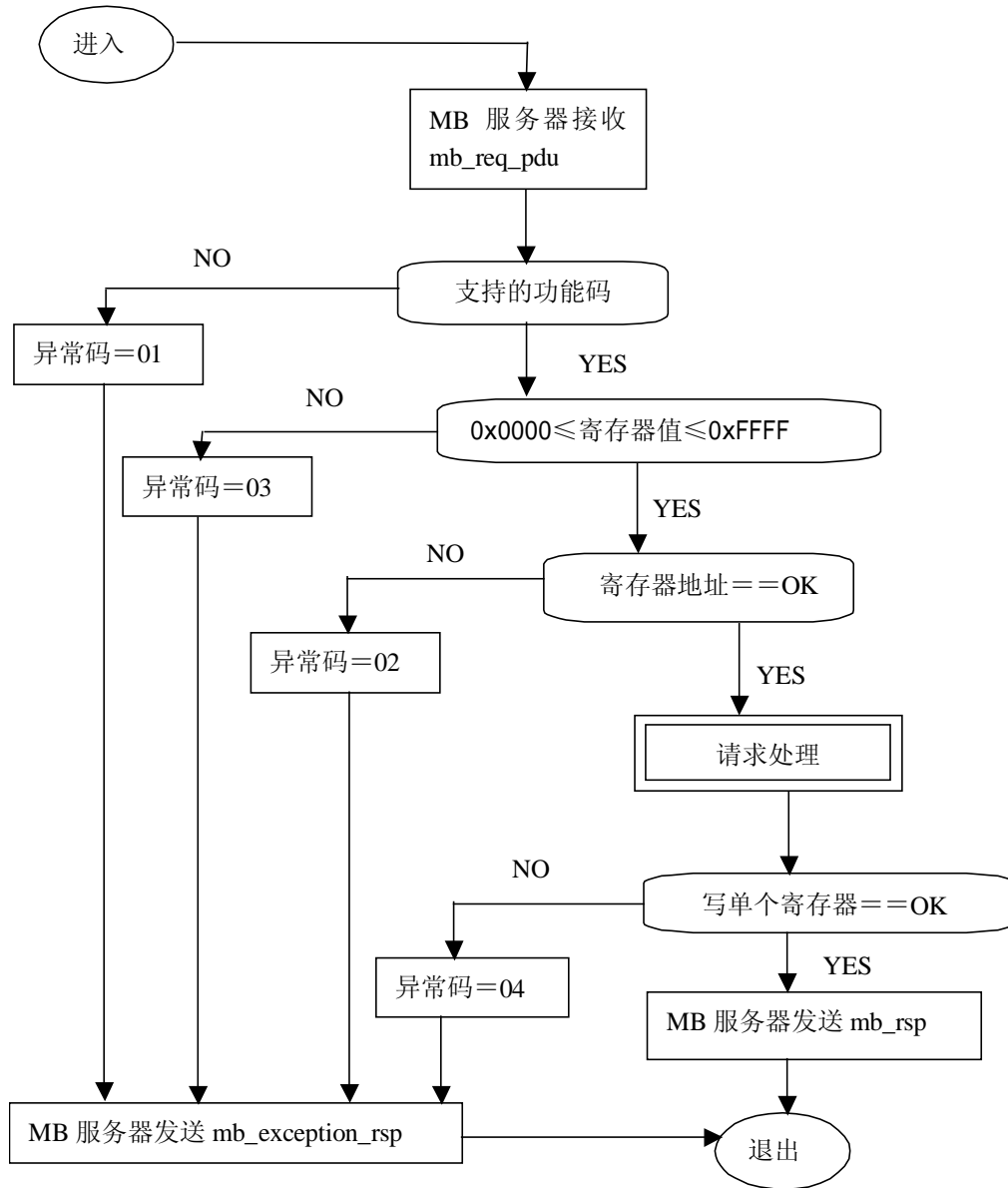


图 15: 写单个寄存器状态图

### 6.7 15 (0x0F) 写多个线圈

在一个远程设备中,使用该功能码强制线圈序列中的每个线圈为 ON 或 OFF。请求 PDU 说明了强制的线圈参考。从零开始寻址线圈。因此,寻址线圈 1 为 0。

请求数据域的内容说明了被请求的 ON/OFF 状态。域比特位置中的逻辑“1”请求相应输出为 ON。域比特位置中的逻辑“0”请求相应输出为 OFF。

正常响应返回功能码、起始地址和强制的线圈数量。

**请求 PDU**

|      |                 |                 |
|------|-----------------|-----------------|
| 功能码  | 1 个字节           | <b>0x0F</b>     |
| 起始地址 | 2 个字节           | 0x0000 至 0xFFFF |
| 输出数量 | 2 个字节           | 0x0001 至 0x07B0 |
| 字节数  | 1 个字节           | <b>N*</b>       |
| 输出值  | <b>N*×1 个字节</b> |                 |

\*N=输出数量/8, 如果余数不等于 0, 那么N = N+1

**响应 PDU**

|      |       |                 |
|------|-------|-----------------|
| 功能码  | 1 个字节 | <b>0x0F</b>     |
| 起始地址 | 2 个字节 | 0x0000 至 0xFFFF |
| 输出数量 | 2 个字节 | 0x0001 至 0x07B0 |

**错误**

|     |       |                   |
|-----|-------|-------------------|
| 差错码 | 1 个字节 | <b>0x8F</b>       |
| 异常码 | 1 个字节 | 01 或 02 或 03 或 04 |

这是一个请求从线圈 20 开始写入 10 个线圈的实例:

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 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)。

应该用零填充最后数据字节中的未使用比特。

| 请求      |        | 响应      |        |
|---------|--------|---------|--------|
| 域名      | (十六进制) | 域名      | (十六进制) |
| 功能      | 0F     | 功能      | 0F     |
| 起始地址 Hi | 00     | 起始地址 Hi | 00     |
| 起始地址 Lo | 13     | 起始地址 Lo | 13     |
| 输出数量 Hi | 00     | 输出数量 Hi | 00     |
| 输出数量 Lo | 0A     | 输出数量 Lo | 0A     |
| 字节数     | 02     |         |        |
| 输出值 Hi  | CD     |         |        |
| 输出值 Lo  | 01     |         |        |



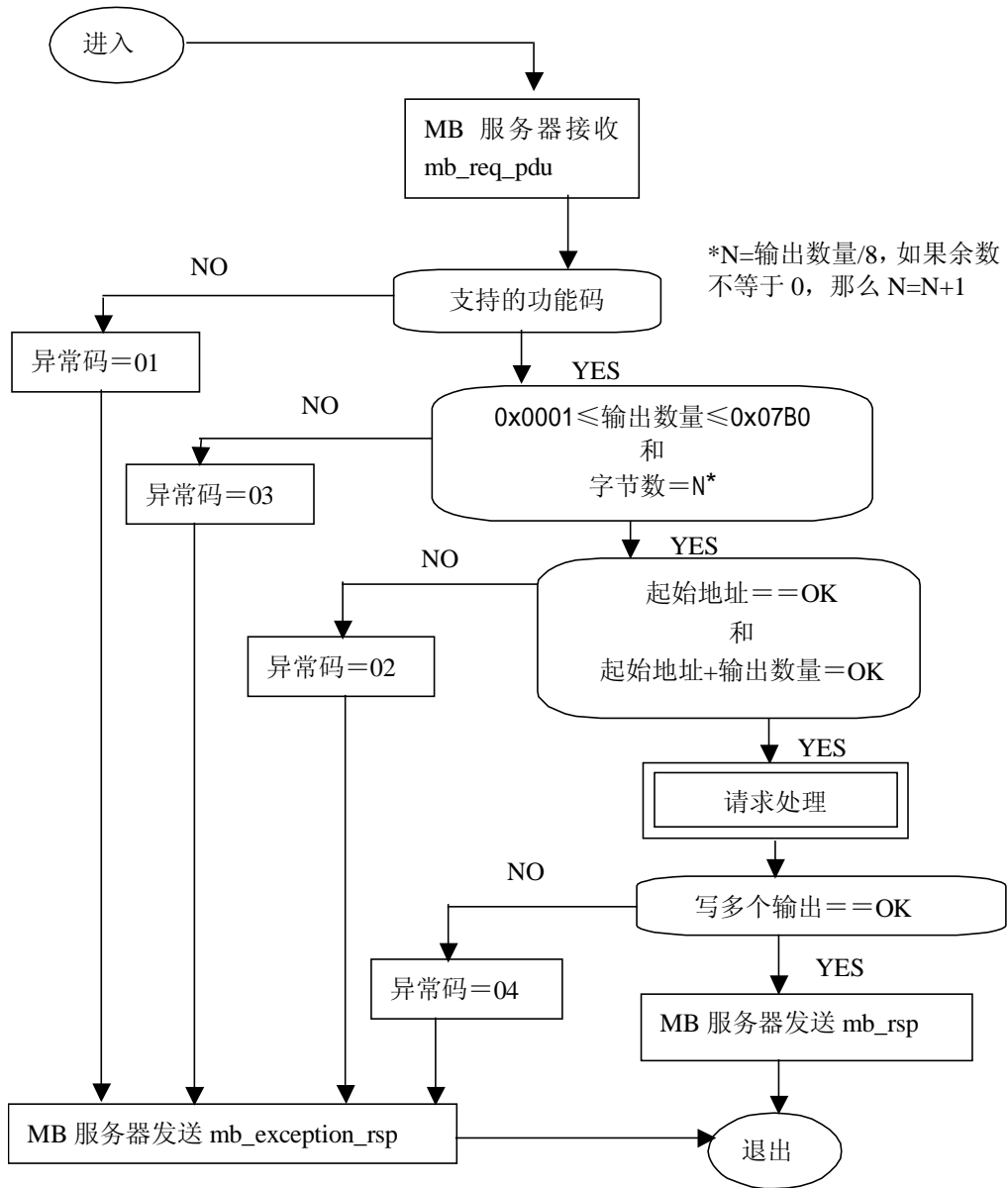


图 16: 写多个输出的状态图

### 6.8 16 (0x10) 写多个寄存器

在一个远程设备中, 使用该功能码写连续寄存器块(1 至约 120 个寄存器)。在请求数据域中说明了请求写入的值。每个寄存器将数据分成两字节。正常响应返回功能码、起始地址和被写入寄存器的数量。

**请求 PDU**

|       |          |                 |
|-------|----------|-----------------|
| 功能码   | 1 个字节    | <b>0x10</b>     |
| 起始地址  | 2 个字节    | 0x0000 至 0xFFFF |
| 寄存器数量 | 2 个字节    | 0x0001 至 0x0078 |
| 字节数   | 1 个字节    | 2×N*            |
| 寄存器值  | N*×2 个字节 | 值               |

\*N=寄存器数量

**响应 PDU**

|       |       |                 |
|-------|-------|-----------------|
| 功能码   | 1 个字节 | <b>0x10</b>     |
| 起始地址  | 2 个字节 | 0x0000 至 0xFFFF |
| 寄存器数量 | 2 个字节 | 1 至 123 (0x7B)  |

**错误**

|     |       |                   |
|-----|-------|-------------------|
| 差错码 | 1 个字节 | <b>0x90</b>       |
| 异常码 | 1 个字节 | 01 或 02 或 03 或 04 |

这是一个请求将十六进制 00 0A 和 01 02 写入以 2 开始的两个寄存器的实例：

| 请求       |        | 响应       |        |
|----------|--------|----------|--------|
| 域名       | (十六进制) | 域名       | (十六进制) |
| 功能       | 10     | 功能       | 10     |
| 起始地址 Hi  | 00     | 起始地址 Hi  | 00     |
| 起始地址 Lo  | 01     | 起始地址 Lo  | 01     |
| 寄存器数量 Hi | 00     | 寄存器数量 Hi | 00     |
| 寄存器数量 Lo | 02     | 寄存器数量 Lo | 02     |
| 字节数      | 04     |          |        |
| 寄存器值 Hi  | 00     |          |        |
| 寄存器值 Lo  | 0A     |          |        |
| 寄存器值 Hi  | 01     |          |        |
| 寄存器值 Lo  | 02     |          |        |

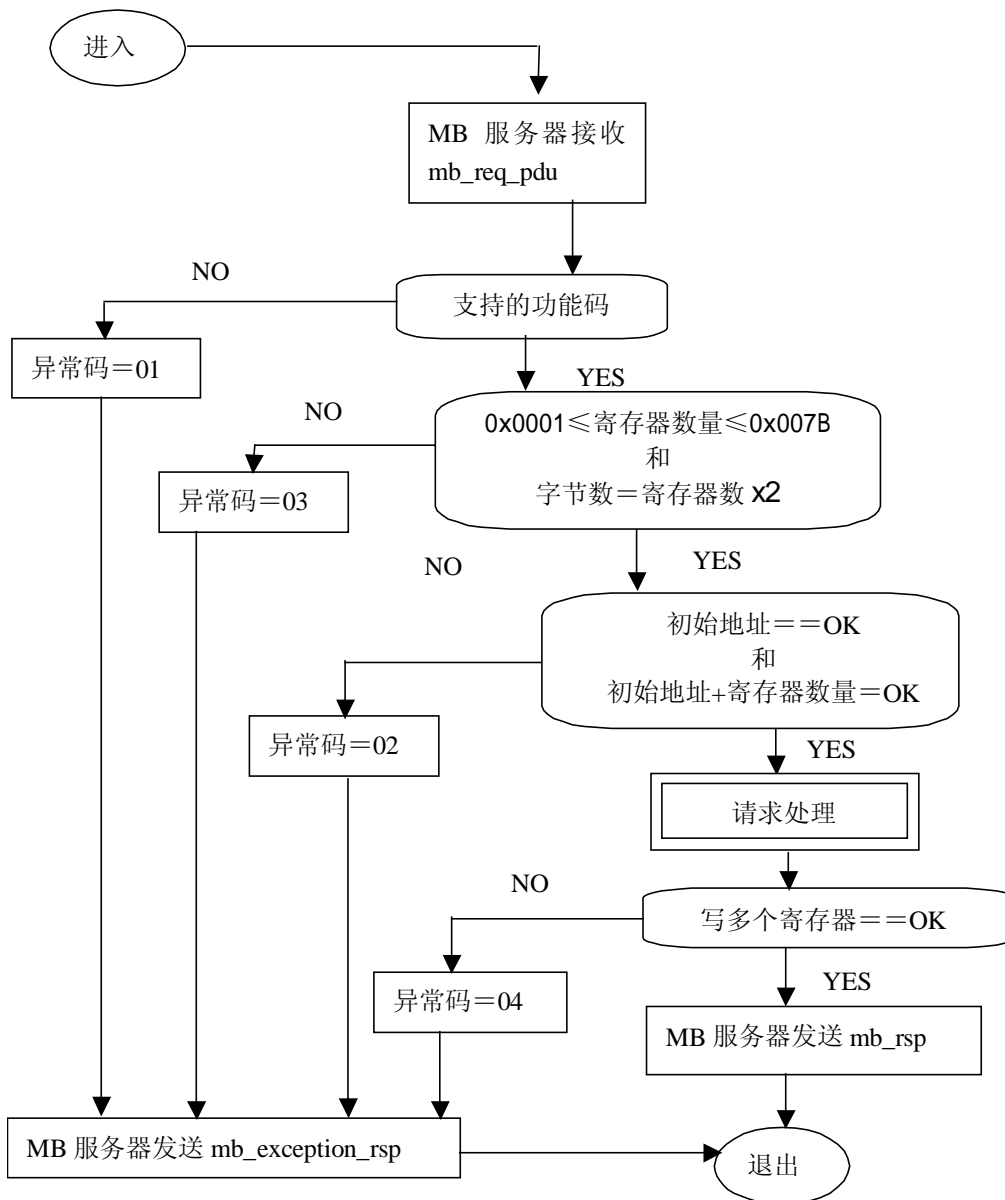


图 17: 写多个寄存器状态图

### 6.9 20 (0x14) 读文件记录

使用该功能码进行文件记录读取。根据字节数量提供所有请求数据长度，并且根据寄存器提供所有记录长度。

文件是记录的结构。每个文件包括 10000 个记录，寻址这些记录为十进制 0000 至 9999 或十六进制 0X0000 至 0X270F，例如寻址记录 12 为 12。

该功能可以读取多个参考组。这些组可以是分散的(不连续的)，但每组中的参考必须是连续的。用含有 7 个字节的独立“子请求”域定义每个组：

参考类型：1 个字节(必须规定为 6)

文件号：2 个字节

文件中的起始记录号：2 个字节

被读出的记录长度：2 个字节

被读取的寄存器数量不能超过 MODBUS 报文允许的长度：256 个字节，这个寄存器数量与预期响应中的所有其它域组合。

正常响应是一系列“子响应”，与“子请求”一一对应。字节数域是所有“子响应”中的全部组合字节数。另外，每个“子响应”都包括一个表示自身字节数的域。

**请求 PDU**

|             |       |                 |
|-------------|-------|-----------------|
| 功能码         | 1 个字节 | <b>0x14</b>     |
| 字节数         | 1 个字节 | 0x07 至 0xF5 字节  |
| 子请求 x, 参考类型 | 1 个字节 | 06              |
| 子请求 x, 文件号  | 2 个字节 | 0x0000 至 0xFFFF |
| 子请求 x, 记录号  | 2 个字节 | 0x0000 至 0x270F |
| 子请求 x, 记录长度 | 2 个字节 | N               |
| 子请求 x+1,... |       |                 |

**响应 PDU**

|               |         |             |
|---------------|---------|-------------|
| 功能码           | 1 个字节   | <b>0x14</b> |
| 响应数据长度        | 1 个字节   | 0x07 至 0xF5 |
| 子请求 x, 文件响应长度 | 1 个字节   | 0x07 至 0xF5 |
| 子请求 x, 参考类型   | 1 个字节   | 6           |
| 子请求 x, 记录数据   | N×2 个字节 |             |
| 子请求 x+1,...   |         |             |

**错误**

|     |       |                        |
|-----|-------|------------------------|
| 差错码 | 1 个字节 | <b>0x94</b>            |
| 异常码 | 1 个字节 | 01 或 02 或 03 或 04 或 08 |

这是一个请求从远程设备读取两个参考组的实例：

组 1 包括文件 4 中的 2 个寄存器，以寄存器 1 开始（地址 0001）。

组 2 包括文件 3 中的 2 个寄存器，以寄存器 9 开始（地址 0009）。

| 请求             |        | 响应             |        |
|----------------|--------|----------------|--------|
| 域名             | (十六进制) | 域名             | (十六进制) |
| 功能             | 14     | 功能             | 14     |
| 字节数            | 0C     | 响应数据长度         | 0E     |
| 子请求 1, 参考类型    | 06     | 子请求 1, 文件响应长度  | 05     |
| 子请求 1, 文件号 Hi  | 00     | 子请求 1, 参考类型    | 06     |
| 子请求 1, 文件号 Lo  | 04     | 子请求 1, 纪录数据 Hi | 0D     |
| 子请求 1, 记录号 Hi  | 00     | 子请求 1, 纪录数据 Lo | FE     |
| 子请求 1, 纪录号 Lo  | 01     | 子请求 1, 纪录数据 Hi | 00     |
| 子请求 1, 记录长度 Hi | 00     | 子请求 1, 纪录数据 Lo | 20     |
| 子请求 1, 纪录长度 Lo | 02     | 子请求 2, 文件响应长度  | 05     |
| 子请求 2, 参考类型    | 06     | 子请求 2, 参考类型    | 06     |

|                |    |                |    |
|----------------|----|----------------|----|
| 子请求 2, 文件号 Hi  | 00 | 子请求 2, 纪录数据 Hi | 33 |
| 子请求 2, 文件号 Lo  | 03 | 子请求 2, 纪录数据 Lo | CD |
| 子请求 2, 记录号 Hi  | 00 | 子请求 2, 纪录数据 Hi | 00 |
| 子请求 2, 记录号 Lo  | 09 | 子请求 2, 纪录数据 Lo | 40 |
| 子请求 2, 记录长度 Hi | 00 |                |    |
| 子请求 2, 记录长度 Lo | 02 |                |    |

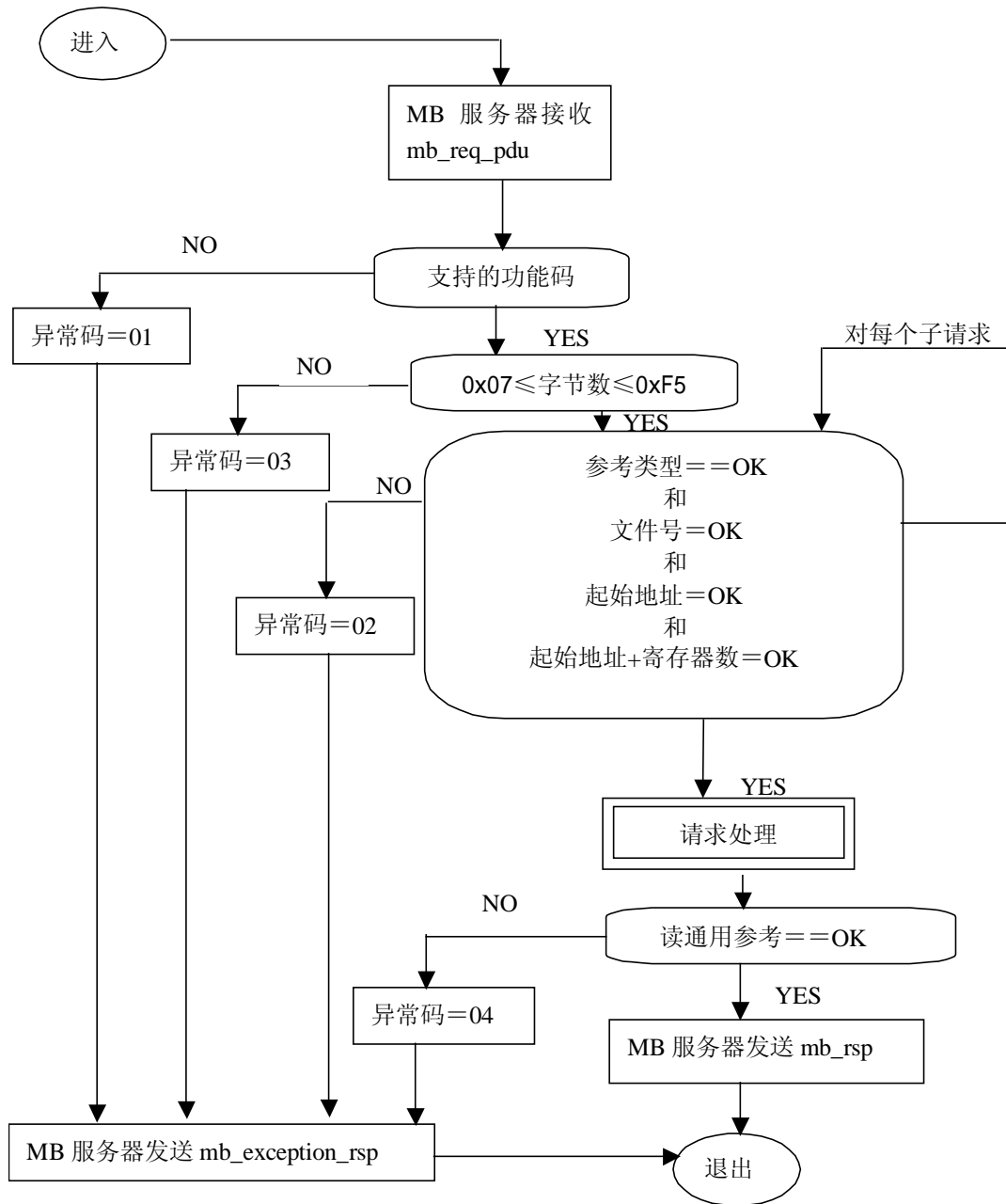


图 18: 读文件记录状态图

### 6.9.1 21 (0x15) 写文件记录

使用该功能码进行文件记录写入。根据字节数量提供所有请求数据长度，并且根据 16 比特字的

数量提供所有记录长度。

文件是记录的结构。每个文件包括 10000 个记录，寻址这些记录为十进制 0000 至 9999 或十六进制 0X0000 至 0X270F，例如寻址记录 12 为 12。

该功能可以写多个参考组。这些组可以是分散的，即不连续的，但每组内的参考必须是连续的。用含有 7 个字节和数据的独立“子请求”域定义每个组：

参考类型：1 个字节(必须规定为 6)

文件号：2 个字节

文件中的起始记录号：2 个字节

被写入的记录长度：2 个字节

被写入的数据：每个寄存器为 2 字节。

被写入的寄存器数量不能超过 MODBUS 报文允许的长度：256 个字节，这个寄存器数量与询问中的所有其它域组合。

正常响应是请求的应答。

### 请求 PDU

|             |         |                 |
|-------------|---------|-----------------|
| 功能码         | 1 个字节   | 0x14            |
| 请求数据长度      | 1 个字节   | 0x07 至 0xF5     |
| 子请求 x, 参考类型 | 1 个字节   | 06              |
| 子请求 x, 文件号  | 2 个字节   | 0x0000 至 0xFFFF |
| 子请求 x, 记录号  | 2 个字节   | 0x0000 至 0x270F |
| 子请求 x, 记录长度 | 2 字节    | N               |
| 子请求 x, 记录数据 | N×2 个字节 |                 |
| 子请求 x+1,... |         |                 |

### 响应 PDU

|             |         |                   |
|-------------|---------|-------------------|
| 功能码         | 1 个字节   | 0x15              |
| 响应数据长度      | 1 个字节   |                   |
| 子请求 x, 参考类型 | 1 个字节   | 06                |
| 子请求 x, 文件号  | 2 个字节   | 0x0000 至 0xFFFF   |
| 子请求 x, 记录号  | 2 个字节   | 0x0000 至 0xFFFF   |
| 子请求 x, 记录长度 | 2 个字节   | 0x0000 至 0xFFFF N |
| 子请求 x, 记录数据 | N×2 个字节 |                   |
| 子请求 x+1,... |         |                   |

### 错误

|     |       |                        |
|-----|-------|------------------------|
| 差错码 | 1 个字节 | 0x95                   |
| 异常码 | 1 个字节 | 01 或 02 或 03 或 04 或 08 |

这是一个请求将一个参考组写入远程设备的实例：

组包括文件 4 中的 3 个寄存器，以寄存器 7 开始（地址 0007）。

| 请求              |        | 响应              |        |
|-----------------|--------|-----------------|--------|
| 域名              | (十六进制) | 域名              | (十六进制) |
| 功能              | 15     | 功能              | 15     |
| 请求数据长度          | 0D     | 请求数据长度          | 0D     |
| 子请求 1, 参考类型     | 06     | 子请求 1, 参考类型     | 06     |
| 子请求 1, 文件号 Hi   | 00     | 子请求 1, 文件号 Hi   | 00     |
| 子请求 1, 文件号 Lo   | 04     | 子请求 1, 文件号 Lo   | 04     |
| 子请求 1, 记录号 Hi   | 00     | 子请求 1, 记录号 Hi   | 00     |
| 子请求 1, 纪录号 Lo   | 07     | 子请求 1, 纪录号 Lo   | 07     |
| 子请求 1, 记录长度 Hi  | 00     | 子请求 1, 记录长度 Hi  | 00     |
| 子请求 1, 纪录长度 Lo  | 03     | 子请求 1, 纪录长度 Lo  | 03     |
| 子请求 1, 记录数据 Hi  | 06     | 子请求 1, 记录数据 Hi  | 06     |
| 子请求 1, 记录数据 Lo  | AF     | 子请求 1, 记录数据 Lo  | AF     |
| 子请求 1, 记录数据 Hi  | 04     | 子请求 1, 记录数据 Hi  | 04     |
| 子请求 1, 记录数据 Lo  | BE     | 子请求 1, 记录数据 Lo  | BE     |
| 子请求 1, 记录数据 Hi  | 10     | 子请求 1, 记录数据 Hi  | 10     |
| 子请求 1, 寄存器数据 Lo | 0D     | 子请求 1, 寄存器数据 Lo | 0D     |

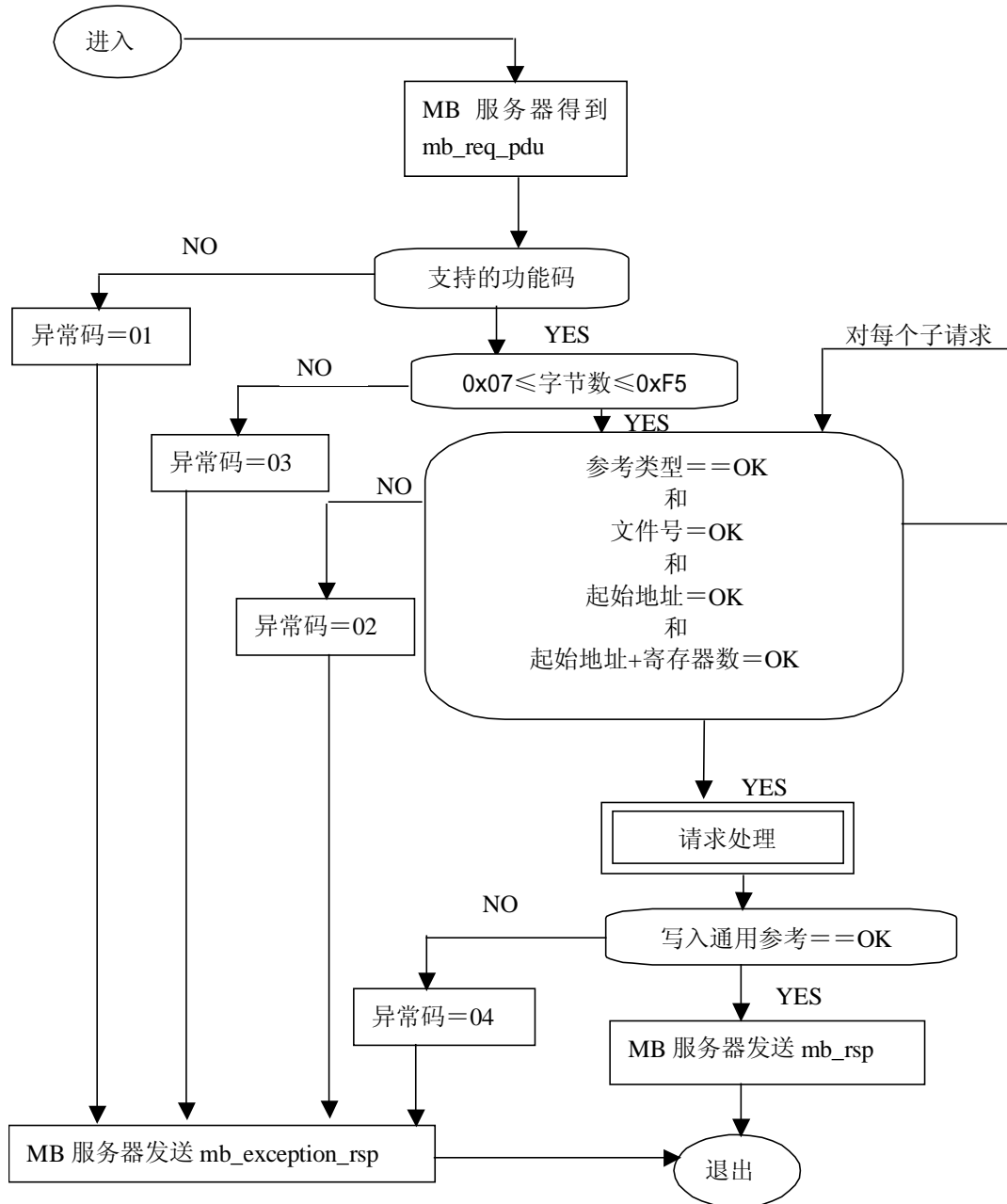


图 19: 写文件记录状态图

### 6.10 22 (0x16) 屏蔽写寄存器

该功能码用于通过利用 AND 屏蔽、OR 屏蔽以及寄存器内容的组合来修改特定保持寄存器的内容。使用这个功能设置或清除寄存器中的单个比特。

请求说明了被写入的保持寄存器、AND 屏蔽使用的数据以及 OR 屏蔽使用的数据。

从 0 开始寻址寄存器。因此，寻址寄存器 1-16 为 0-15。


功能的算法为：

结果= (当前内容 AND And\_Mask) OR (Or\_Mask AND And\_Mask)

例如：



|            | 十六进制 | 二进制       |
|------------|------|-----------|
| 当前内容 =     | 12   | 0001 0010 |
| And_Mask = | F2   | 1111 0010 |
| Or_Mask =  | 25   | 0010 0101 |
| And_Mask = | 0D   | 0000 1101 |
| 结果=        | 17   | 0001 0111 |

 注:

如果 Or\_Mask 值为零，那么结果是当前内容和 And\_Mask 的简单逻辑 AND（与）。如果 And\_Mask 值为零，结果等于 Or\_Mask 值。

可以使用读保持寄存器功能（功能码 03）读出寄存器的内容。于是，当控制器扫描它的用户逻辑程序时，随后可以改变寄存器的内容。

正常的响应是请求的应答。在已经写入寄存器之后，返回响应。

### 请求 PDU

|          |       |                 |
|----------|-------|-----------------|
| 功能码      | 1 个字节 | <b>0x16</b>     |
| 参考地址     | 2 个字节 | 0x0000 至 0xFFFF |
| And_Mask | 2 个字节 | 0x0000 至 0xFFFF |
| Or_Mask  | 2 个字节 | 0x0000 至 0xFFFF |

### 响应 PDU

|          |       |                 |
|----------|-------|-----------------|
| 功能码      | 1 个字节 | <b>0x16</b>     |
| 参考地址     | 2 个字节 | 0x0000 至 0xFFFF |
| And_Mask | 2 个字节 | 0x0000 至 0xFFFF |
| Or_Mask  | 2 个字节 | 0x0000 至 0xFFFF |

### 错误

|     |       |                   |
|-----|-------|-------------------|
| 差错码 | 1 个字节 | <b>0x96</b>       |
| 异常码 | 1 个字节 | 01 或 02 或 03 或 04 |

这是一个利用上述屏蔽值在远程设备中对寄存器 5 的屏蔽写入实例。

| 请求          |        | 响应          |        |
|-------------|--------|-------------|--------|
| 域名          | (十六进制) | 域名          | (十六进制) |
| 功能          | 16     | 功能          | 16     |
| 参考地址 Hi     | 00     | 参考地址 Hi     | 00     |
| 参考地址 Lo     | 04     | 参考地址        | 04     |
| And_Mask Hi | 00     | And_Mask Hi | 00     |
| And_Mask    | F2     | And_Mask    | F2     |
| Or_Mask Hi  | 00     | Or_Mask Hi  | 00     |
| Or_Mask     | 25     | Or_Mask     | 25     |

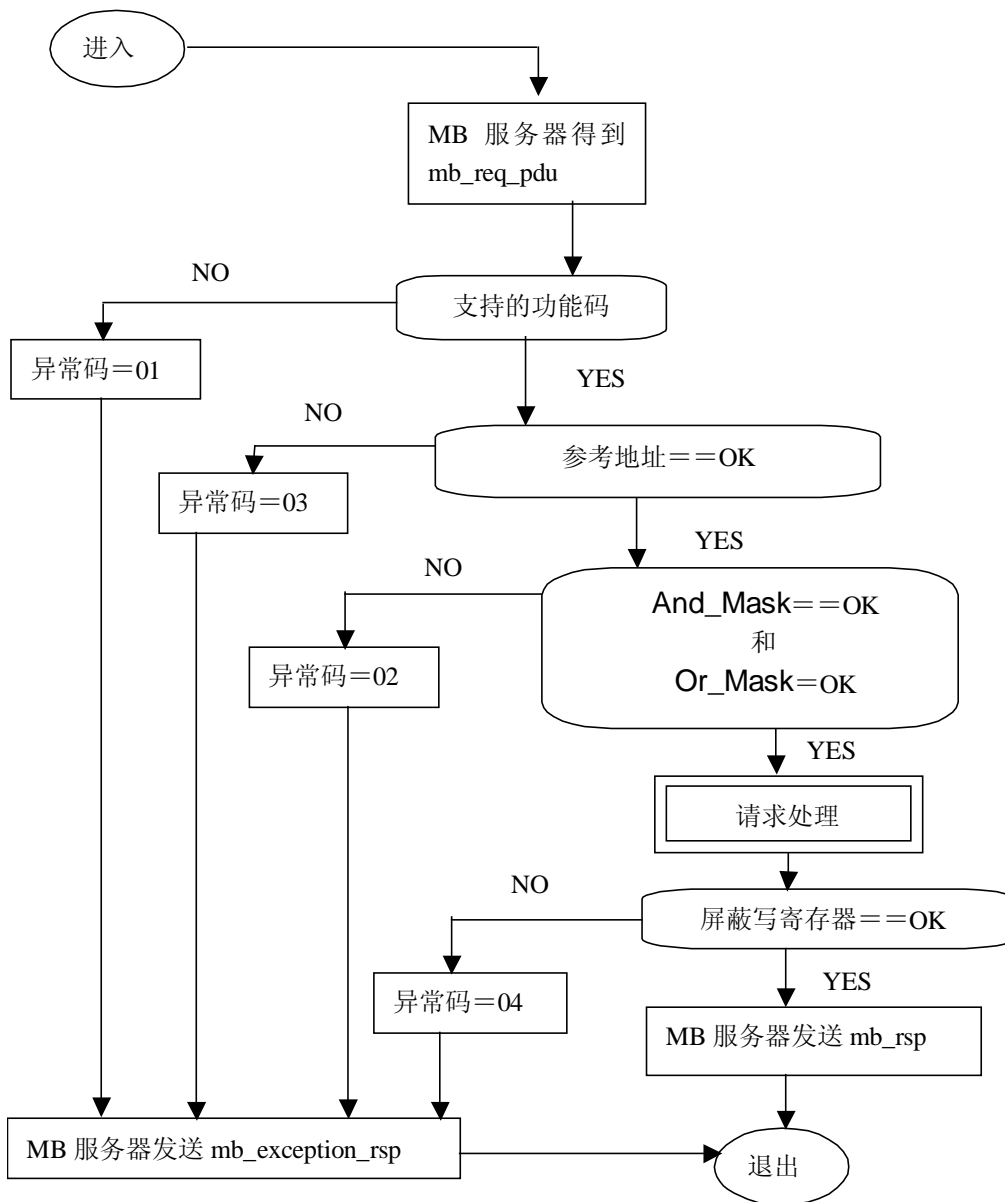


图 20: 屏蔽写保持寄存器状态图

6.11 23 (0x17) 读/写多个寄存器

在一个单独 MODBUS 事务中，这个功能码实现了一个读操作和一个写操作的组合。从零开始寻址保持寄存器。因此，寻址保持寄存器 1-16 为 0-15。

请求说明了起始地址、被读取的保持寄存器号和起始地址、保持寄存器号以及被写入的数据。在写数据域中，字节数说明随后的字节号。

正常响应包括被读出的寄存器组的数据。在读数据域中，字节数域说明随后的字节数量。

请求 PDU

|       |       |                 |
|-------|-------|-----------------|
| 功能码   | 1 个字节 | 0x17            |
| 读起始地址 | 2 个字节 | 0x0000 至 0xFFFF |

|       |          |                   |
|-------|----------|-------------------|
| 读的数量  | 2 个字节    | 0x0001 至近似 0x0076 |
| 写起始地址 | 2 个字节    | 0x0000 至 0xFFFF   |
| 写的数量  | 2 个字节    | 0x0001 至近似 0x0076 |
| 写字节数  | 1 个字节    | 2×N*              |
| 写寄存器值 | N*×2 个字节 |                   |

\*N=写的数量

**响应 PDU**

|       |          |             |
|-------|----------|-------------|
| 功能码   | 1 个字节    | <b>0x17</b> |
| 字节数   | 1 个字节    | 2×N*        |
| 读寄存器值 | N*×2 个字节 |             |

\*N=读的数量

**错误**

|     |       |                   |
|-----|-------|-------------------|
| 差错码 | 1 个字节 | <b>0x97</b>       |
| 异常码 | 1 个字节 | 01 或 02 或 03 或 04 |

这是一个请求从寄存器 4 开始读六个寄存器并且从寄存器 15 开始读三个寄存器的实例：

| 请求       |        | 响应       |        |
|----------|--------|----------|--------|
| 域名       | (十六进制) | 域名       | (十六进制) |
| 功能       | 17     | 功能       | 17     |
| 读起始地址 Hi | 00     | 字节数      | 0C     |
| 读起始地址 Lo | 03     | 读寄存器值 Hi | 00     |
| 读的数量 Hi  | 00     | 读寄存器值 Lo | FE     |
| 读的数量 Lo  | 06     | 读寄存器值 Hi | 0A     |
| 写起始地址 Hi | 00     | 读寄存器值 Lo | CD     |
| 写起始地址 Lo | 0E     | 读寄存器值 Hi | 00     |
| 写的数量 Hi  | 00     | 读寄存器值 Lo | 01     |
| 写的数量 Lo  | 03     | 读寄存器值 Hi | 00     |
| 写字节数     | 06     | 读寄存器值 Lo | 03     |
| 写寄存器值 Hi | 00     | 读寄存器值 Hi | 00     |
| 写寄存器值 Lo | FF     | 读寄存器值 Lo | 0D     |
| 写寄存器值 Hi | 00     | 读寄存器值 Hi | 00     |
| 写寄存器值 Lo | FF     | 读寄存器值 Lo | FF     |
| 写寄存器值 Hi | 00     |          |        |
| 写寄存器值 Lo | FF     |          |        |

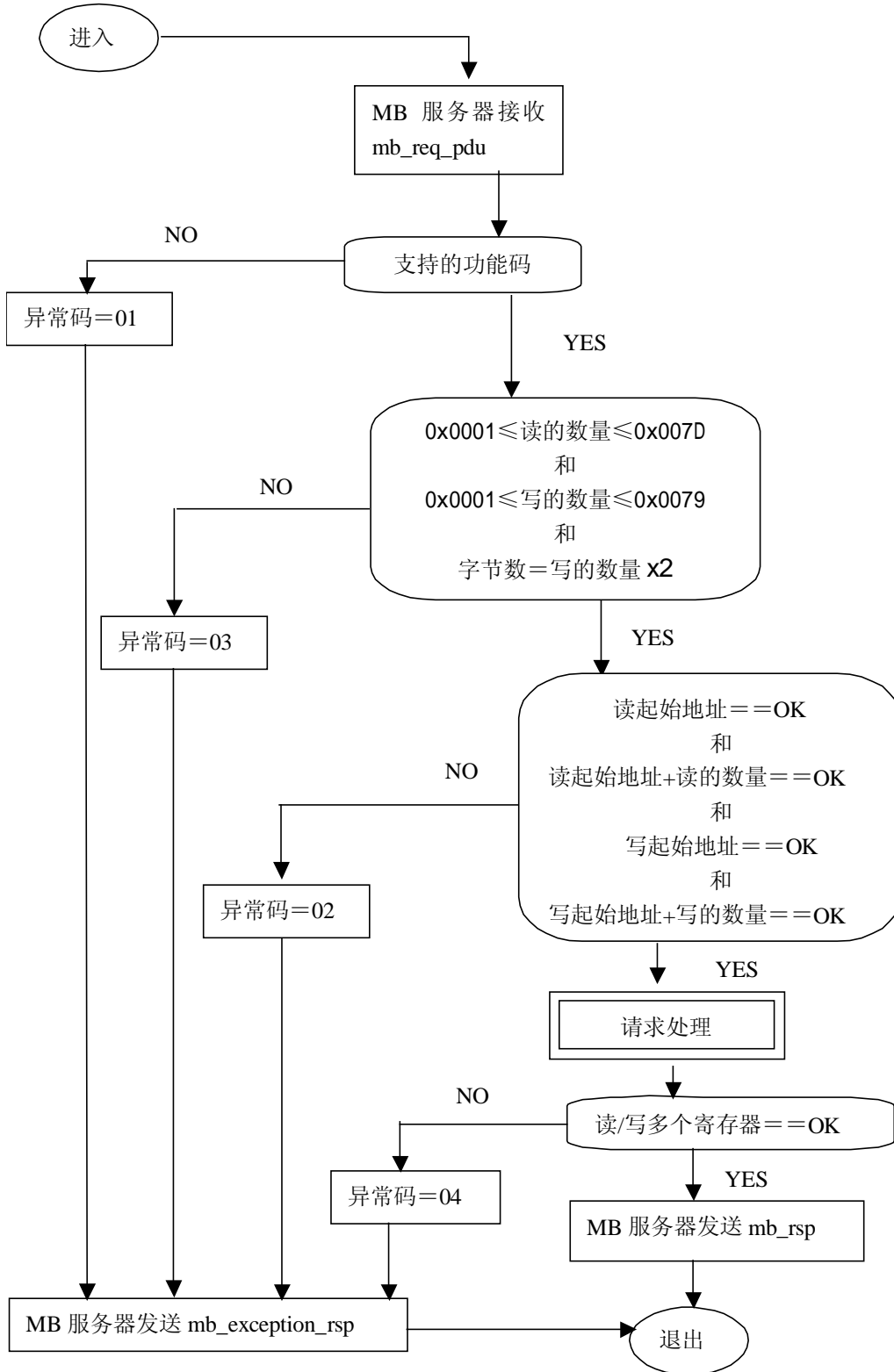


图 21: 读/写多个寄存器状态图

### 6.12 43 (0x2B)读设备识别码

这个功能码允许读取与远程设备的物理描述和功能描述相关的识别码和附加报文。

将读设备识别码接口模拟为一个地址空间，这个地址空间由一组可寻址数据元素组成。数据元素是被叫对象，并且对象 Id 确定这个数据元素。

接口由 3 种对象组成：

- 1 基本设备识别码。所有此种对象都是必备的：厂商名称、产品代码和修订本号。
- 1 正常设备识别码。除基本数据对象以外，设备提供了附加的和可选择的识别码以及数据对象描述。按标准定义所有种类的对象，但是这种对象的执行是可选的。
- 1 扩展设备识别码。除正常数据对象以外，设备提供了附加的和可选择的识别码以及专用数据描述。所有这些数据都是与设备有关的。

| 对象 Id               | 对象名称/描述                          | 类型        | M/O | 种类 |
|---------------------|----------------------------------|-----------|-----|----|
| 0x00                | 厂商名称                             | ASCII 字符串 | 强制的 | 基本 |
| 0x01                | 产品代码                             | ASCII 字符串 | 强制的 |    |
| 0x02                | 主要修订本                            | ASCII 字符串 | 强制的 |    |
| 0x03                | VendorUrl                        | ASCII 字符串 | 可选的 | 规则 |
| 0x04                | 产品名称                             | ASCII 字符串 | 可选的 |    |
| 0x05                | 模式名称                             | ASCII 字符串 | 可选的 |    |
| 0x06                | 用户应用名称                           | ASCII 字符串 | 可选的 |    |
| 0x07<br>...<br>0x7F | 保留                               |           | 可选的 |    |
| 0x80<br>...<br>0xFF | 可选择地定义专用对象<br>范围[0x80—0xFF]与产品有关 | 相关设备      | 可选的 | 扩展 |

#### 请求 PDU

|             |       |             |
|-------------|-------|-------------|
| 功能码         | 1 个字节 | 0x2B        |
| MEI 类型      | 1 个字节 | 0x0E        |
| ReadDevId 码 | 1 个字节 | 01/02/03/04 |
| 对象 id       | 1 个字节 | 0x00 至 0xFF |

#### 响应 PDU

|              |       |             |
|--------------|-------|-------------|
| 功能码          | 1 个字节 | 0x2B        |
| MEI 类型       | 1 个字节 | 0x0E        |
| ReadDevId 代码 | 1 个字节 | 01/02/03/04 |
| 一致性等级        | 1 个字节 |             |
| 随后更多         | 1 个字节 | 00/FF       |
| 下一个对象 Id     | 1 个字节 | 对象 ID 号     |
| 对象号          | 1 个字节 |             |
| 对象 ID 的列表    | 1 个字节 |             |
| 对象长度         | 1 个字节 |             |
| 对象值          | 1 个字节 |             |

**错误**

|        |      |                       |
|--------|------|-----------------------|
| 功能码    | 1 字节 | 0xAB:<br>Fc 0x2B+0x80 |
| MEI 类型 | 1 字节 | 14                    |
| 异常码    | 1 字节 | 01、02、03、04           |

**请求参数描述:**

指配号为 14 的 MODBUS 封装接口识别读识别码请求。定义四种访问类型:

- 01: 请求获得基本设备识别码 (流访问)
- 02: 请求获得正常设备识别码 (流访问)
- 03: 请求获得扩展设备识别码 (流访问)
- 04: 请求获得特定识别码对象 (专用访问)

在识别码数据不适合单独响应的情况下, 可以需要几个请求/响应事务处理。对象 id 字节给出了获得的第一个对象识别码。对于第一个事物处理来说, 客户机必须设置对象 id 为 0, 以便获得设备识别码数据的开始。对于下列事务来说, 客户机必须设置对象 id 为前面响应中服务器的返回值。

如果对象 id 不符合任何已知对象, 那么服务器象指向对象 0 那样响应 (从头开始)。

在单个访问的情况下: ReadDevId 代码 04, 请求中的对象 id 给出了获得的对象识别码。

如果对象 id 不符合任何已知对象, 那么服务器返回一个异常码=02 (非法数据地址) 的异常响应。

**响应参数描述:**

- 功能码: 功能码 43 (十进制) 0x2B (十六进制)
- MEI 类型: 为设备识别码接口指配号的 14 (0x0E) MEI 类型
- ReadDevId 码: 与请求 ReadDevId 码相同: 01、02、03 或 04
- 一致性等级: 设备的识别码一致性等级和支持访问的类型
  - 01: 基本识别码(仅流访问)
  - 02: 正常识别码(仅流访问)
  - 03: 扩展识别码(仅流访问)
  - 81: 基本识别码(流访问和单个访问)
  - 82: 正常识别码(流访问和单个访问)
  - 83: 扩展识别码(流访问和单个访问)

**随后更多:** 在 ReadDevId 码 01、02 或 03 (流访问)的情况下, 如果识别码数据不符合单个响应, 那么需要几个请求/响应事务处理。

- 00: 对象不再是可利用的
  - FF: 其它识别码对象是可利用的, 并且需要更多 MODBUS 事务处理
- 在 ReadDevId 码 04(单个访问)的情况下,** 必须设置这个域为 00。

下一个对象 Id: 如果 “随后更多=FF”, 那么请求下一个对象的识别码  
如果 “随后更多=00”, 那么必须设置为 00 (无用的)

对象号 在响应中返回的对象识别码号  
(对于单个访问, 对象号码= 1)

对象 0.id PDU 中返回的第一个对象识别码(流访问)或请求对象的识别码 (单个访问)

Object0.长度 第一个对象的字节长度

Object0.值            第一个对象的值(对象 0.长度字节)  
 ...  
 ObjectN.id            最后对象的识别码(在响应中)  
 ObjectN.长度        最后对象的字节长度  
 ObjectN.值            最后对象的值(对象 N.长度字节)

“基本设备识别码”的读设备识别码请求的实例：在这个实例中，一个响应 PDU 中发送所有的报文。

| 请求          |    | 响应          |         |
|-------------|----|-------------|---------|
| 域名          | 值  | 域名          | 值       |
| 功能          | 2B | 功能          | 2B      |
| MEI 类型      | 0E | MEI 类型      | 0E      |
| ReadDevId 码 | 01 | ReadDevId 码 | 01      |
| 对象 id       | 00 | 一致性等级       | 01      |
|             |    | 更多继续        | 00      |
|             |    | 下一个对象 id    | 00      |
|             |    | 对象号         | 03      |
|             |    | 对象 id       | 00      |
|             |    | 对象长度        | 16      |
|             |    | 对象值         | “公司识别码” |
|             |    | 对象 id       | 01      |
|             |    | 对象长度        | 0A      |
|             |    | 对象值         | “产品代码”  |
|             |    | 对象 id       | 02      |
|             |    | 对象长度        | 05      |
|             |    | 对象值         | “V2.11” |

如果一个设备需要几个事务处理发送响应，那么启动下列事务处理。  
 第一个事务处理：

| 请求          |    | 响应          |         |
|-------------|----|-------------|---------|
| 域名          | 值  | 域名          | 值       |
| 功能          | 2B | 功能          | 2B      |
| MEI 类型      | 0E | MEI 类型      | 0E      |
| ReadDevId 码 | 01 | ReadDevId 码 | 01      |
| 对象 id       | 00 | 一致性等级       | 01      |
|             |    | 更多继续        | FF      |
|             |    | 下一个对象 id    | 02      |
|             |    | 对象号         | 03      |
|             |    | 对象 id       | 00      |
|             |    | 对象长度        | 16      |
|             |    | 对象值         | “公司识别码” |
|             |    | 对象 id       | 01      |

|  |  |      |                       |
|--|--|------|-----------------------|
|  |  | 对象长度 | 1A                    |
|  |  | 对象值  | “产品代码”<br>XXXXXXXXXX” |

第二个事务处理：

| 请求          |    | 响应          |         |
|-------------|----|-------------|---------|
| 域名          | 值  | 域名          | 值       |
| 功能          | 2B | 功能          | 2B      |
| MEI 类型      | 0E | MEI 类型      | 0E      |
| ReadDevId 码 | 01 | ReadDevId 码 | 01      |
| 对象 id       | 02 | 一致性等级       | 01      |
|             |    | 更多继续        | 00      |
|             |    | 下一个对象 id    | 00      |
|             |    | 对象号         | 03      |
|             |    | 对象 id       | 02      |
|             |    | 对象长度        | 05      |
|             |    | 对象值         | “V2.11” |



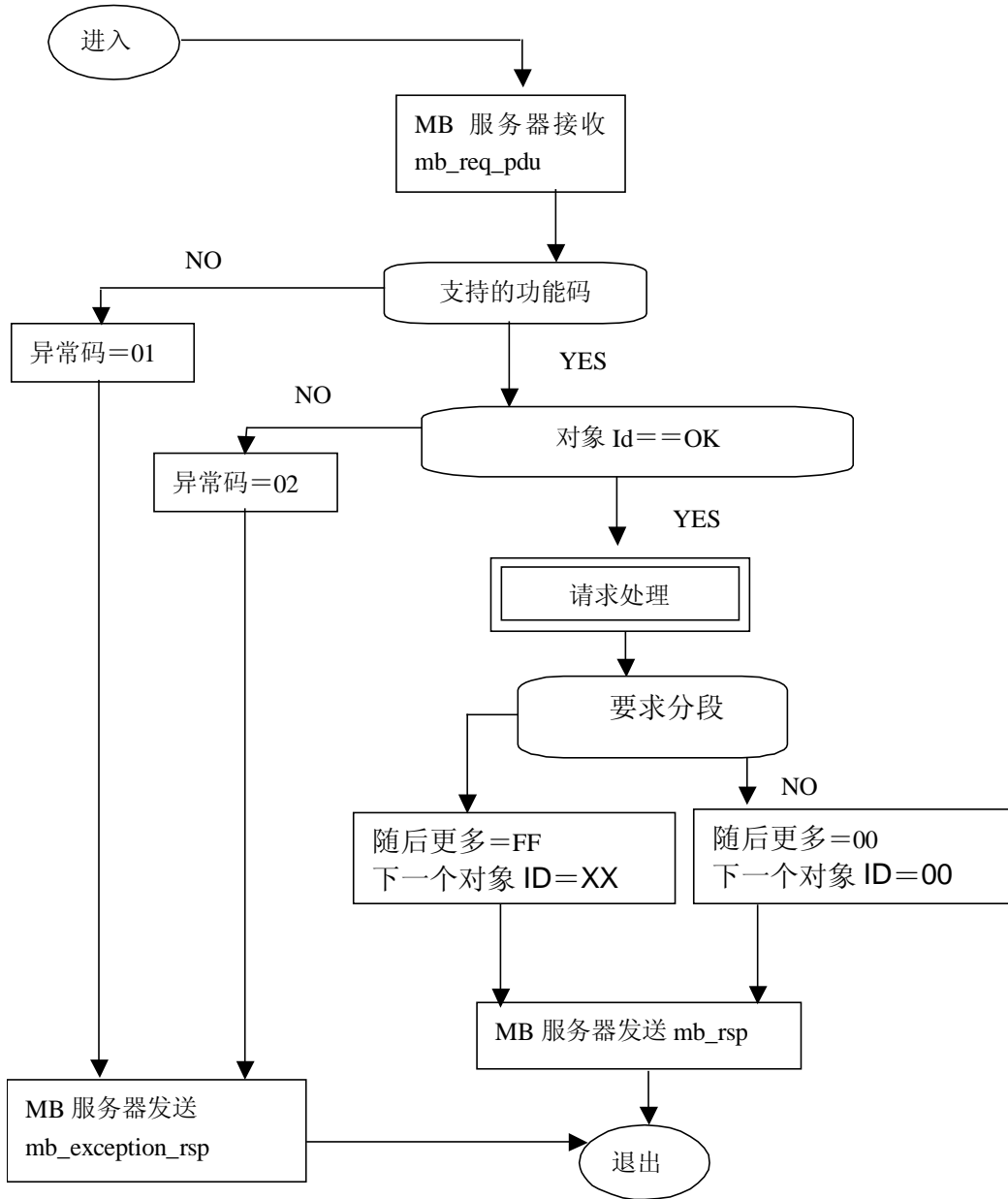


图 22: 读设备识别码状态图

## 7 MODBUS 异常响应

当客户机设备向服务器设备发送请求时，客户机希望一个正常响应。从主站询问中出现下列四

种可能事件之一：

- l 如果服务器设备接收到无通信错误的请求，并且可以正常地处理询问，那么服务器设备将返回一个正常响应。
- l 如果由于通信错误，服务器没有接收到请求，那么不能返回响应。客户机程序将最终处理请求的超时状态。
- l 如果服务器接收到请求，但是检测到一个通信错误（奇偶校验、LRC、CRC、...），那么不能返回响应。客户机程序将最终处理请求的超时状态。
- l 如果服务器接收到无通信错误的请求，但不能处理这个请求（例如，如果请求读一个不存在的输出或寄存器），服务器将返回一个异常响应，通知用户错误的本质特性。

异常响应报文有两个与正常响应不同的域：

**功能码域：**在正常响应中，服务器利用响应功能码域来应答最初请求的功能码。所有功能码的最高有效位（MSB）都为 0（它们的值都低于十六进制 80）。在异常响应中，服务器设置功能码的 MSB 为 1。这使得异常响应中的功能码值比正常响应中的功能码值高十六进制 80。

通过设置功能码的 MSB，客户机的应用程序能够识别异常响应，并且能够检测异常码的数据域。

**数据域：**在正常响应中，服务器可以返回数据域中数据或统计表（请求中要求的任何报文）。在异常响应中，服务器返回数据域中的异常码。这就定义了产生异常的服务器状态。

客户机请求和服务器异常响应的实例：

| 请求      |        | 响应  |        |
|---------|--------|-----|--------|
| 域名      | (十六进制) | 域名  | (十六进制) |
| 功能      | 01     | 功能  | 81     |
| 起始地址 Hi | 04     | 异常码 | 02     |
| 起始地址 Lo | A1     |     |        |
| 输出数量 Hi | 00     |     |        |
| 输出数量 Lo | 01     |     |        |

在这个实例中，客户机对服务器设备寻址请求。功能码(01)用于读输出状态操作。它将请求地址 1245(十六进制 04A1)的输出状态。值得注意的是，象输出域(0001)号码说明的那样，只读出一个输出。

如果在服务器设备中不存在输出地址，那么服务器将返回异常码(02)的异常响应。这就说明从站的非法数据地址。

从下页开始异常码的列表。

| MODBUS 异常码 |        |   |
|------------|--------|---|
| 代码         | 名称     | 含义  |
| 01         | 非法功能   | 对于服务器(或从站)来说，询问中接收到的功能码是不可允许的操作。这也许是因为功能码仅仅适用于新设备而在被选单元中是不可实现的。同时，还指出服务器(或从站)在错误状态中处理这种请求，例如：因为它是未配置的，并且要求返回寄存器值。           |
| 02         | 非法数据地址 | 对于服务器(或从站)来说，询问中接收到的数据地址是不可允许的地址。特别是，参考号和传输长度的组合是无效的。对于带有 100 个寄存器的控制器来说，带有偏移量 96 和长度 4 的请求会成功，带有偏移量 96 和长度 5 的请求将产生异常码 02。 |

|    |            |  |
|----|------------|--|
| 03 | 非法数据值      | 对于服务器(或从站)来说, 询问中包括的值是不可允许的值。这个值指示了组合请求剩余结构中的故障, 例如: 隐含长度是不正确的。并不意味着, 因为 MODBUS 协议不知道任何特殊寄存器的任何特殊值的重要意义, 寄存器中被提交存储的数据项有一个应用程序期望之外的值。 |
| 04 | 从站设备故障     | 当服务器(或从站)正在设法执行请求的操作时, 产生不可重新获得的差错。  |
| 05 | 确认         | 与编程命令一起使用。服务器(或从站)已经接受请求, 并切正在处理这个请求, 但是需要长的持续时间进行这些操作。返回这个响应防止在客户机(或主站)中发生超时错误。客户机(或主站)可以继续发送轮询程序完成报文来确定是否完成处理。                     |
| 06 | 从属设备忙      | 与编程命令一起使用。服务器(或从站)正在处理长持续时间的程序命令。服务器(或从站)空闲时, 用户(或主站)应该稍后重新传输报文。   |
| 08 | 存储奇偶性差错    | 与功能码 20 和 21 以及参考类型 6 一起使用, 指示扩展文件区不能通过一致性校验。<br>服务器(或从站)设法读取记录文件, 但是在存储器中发现一个奇偶校验错误。客户机(或主方)可以重新发送请求, 但可以在服务器(或从站)设备上要求服务。          |
| 0A | 不可用网关路径    | 与网关一起使用, 指示网关不能为处理请求分配输入端口至输出端口的内部通信路径。通常意味着网关是错误配置的或过载的。  |
| 0B | 网关目标设备响应失败 | 与网关一起使用, 指示没有从目标设备中获得响应。通常意味着设备未在网络中。  |

## 第二部分：Modbus 协议在串行链路上的实现指南

## 1 引言

### 1.1 范围

Modbus 标准定义了 OSI 模型第 7 层上的应用层报文传输协议，它在连接至不同类型总线或网络的设备之间提供客户机/服务器通信。它还将串行链路上的协议标准化，以便在一个主站和一个或多个从站之间交换 Modbus 请求。

本文件的目的是表述串行链路上的 Modbus 协议。使用对象为在他们的产品实现串行链路 Modbus 协议的系统设计者。

本文件将增进使用 Modbus 协议的设备之间的互通性。

本文件还是对“Modbus 协议规范”的补充。

第 5 章定义了“Modbus 串行链路”的实现等级。级别的规范是对一个设备能够属于某个级别而必须遵守的要求的总和。

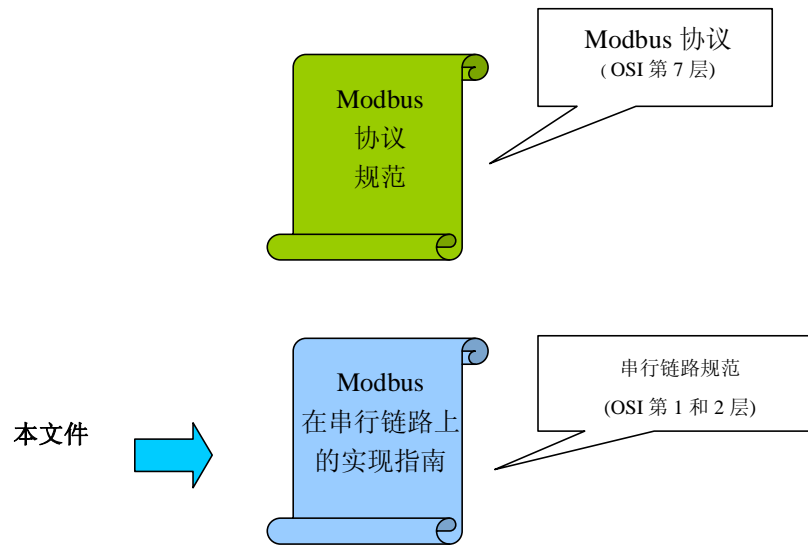


图 1: Modbus 文件概要

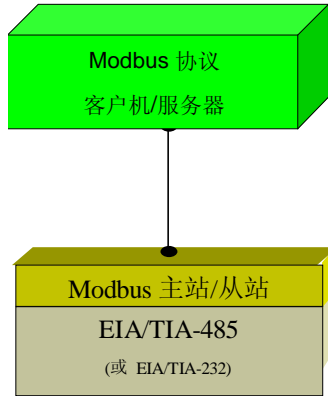
### 1.2 协议概述

本文件描述 Modbus 串行链路协议。**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。作为附加的选项，也可以实现 RS485 四线制接口。当只需要短距离的点到点通信时，TIA/EIA-232-E (RS232) 串行接口也可以使用。(参见有关“物理层”的章节)

下图给出了 Modbus 串行通信栈对应于 7 层 OSI 模型的一般关系。



| 层 | ISO/OSI 模型 |                             |
|---|------------|-----------------------------|
| 7 | 应用层        | Modbus 协议                   |
| 6 | 表示层        | 空                           |
| 5 | 会话层        | 空                           |
| 4 | 传输层        | 空                           |
| 3 | 网络层        | 空                           |
| 2 | 数据链路层      | Modbus 串行链路协议               |
| 1 | 物理层        | EIA/TIA-485 (或 EIA/TIA-232) |

图 1 Modbus 协议和 ISO/OSI 模型

位于 OSI 模型第 7 层的 Modbus 应用层报文传输协议，提供了连接于总线或网络的设备之间的客户机/服务器通信。在 Modbus 串行链路上客户机的功能由主节点提供而服务器功能由子节点实现。

### 1.3 约定

在本文件中，下列词汇用于定义每一种**要求**的重要程度。

#### § "必须" / "要求的"

含有词语 "必须" 的要求是**强制的**。词语**必须**，或形容词**要求的**，表示该项为实现的绝对要求。这些词语带有下划线。

#### § "应该" / "建议的"

所有包含 "应该"，或形容词**建议的**建议，为期望的功能。这些建议**应该**作为选择不同的实现选项时的指南。在有合理的理由的特定条件下，可以忽略这些项。但是，对其全部含义应该理解并且基于情况做出选择时应仔细斟酌。这些词语带有下划线。

#### § "可以" / "可选的"

词语 "可以"，或形容词**可选的**，表示该项为真正意义的可选的。某个设计者可以选择包含该项(基于特定的市场需求或产品功能增强)；而另一个可以选择忽略该项。

#### 1.4 一致性

如果某个实现不满足实现级别中一个或多个**必须**的要求，则是**不符合的**。

如果某个实现满足实现级别中所有的**必须**要求和所有的**应该**的建议，则称为**"无条件符合的"**。

如果某个实现满足实现级别中所有的**必须**要求和不是所有的**应该**的建议，则称为**"有条件符合的"**。

#### 1.5 缩略语

定义本文件中用到的特定词汇、符号和缩略语。

|                   |   |
|-------------------|---|
| 2W                | 在“电气接口”一章中定义的两线制配置，或其中的一个接口。  |
| 4W                | 在“电气接口”一章中定义的四线制配置，或其中的一个接口。  |
| 2W+2W             | 在两线制系统中使用四线制接口的特殊配置。<br>(需要时可参见 Schneider Electric Momentum 文件 870 USE 101 10)。                               |
| AUI               | 连接单元接口 (Attachment Unit Interface)  |
| 公共端               | EIA/TIA 标准中的信号公共端 (Common)。在两线制或四线制 RS485 Modbus 网络中，信号和可选的电源的公共端。 Power Supply 公共端                           |
| DCE               | 一个 Modbus 设备。例如，实现了 RS232 数据电路设备 (Data-Circuit Equipment) 的可编程控制器适配器。也称作数据通信设备(Data Communication Equipment)。 |
| 设备                | 或“Modbus 设备”：参见其定义。   |
| 驱动器               | 发生器，或发送器  |
| DTE               | 一个 Modbus 设备。例如，实现了 RS232 数据终端设备 (Data Terminal Equipment) 编程终端或 PC   |
| ITr               | 干缆侧的物理总线接口 (Interface on Trunk Side)。   |
| IDv               | 设备侧的物理总线接口 (Interface on Derivation Side)。  |
| LT                | 线路终端(Line Termination)。   |
| Modbus 设备         | 实现了 Modbus 串行链路并遵循技术规范的设备。  |
| RS232             | EIA/ TIA -232 标准。   |
| RS485             | EIA/ TIA -485 标准。   |
| RS485 Modbus      | 与该技术标准一致的两线制或四线制网络  |
| 收发器 (Transceiver) | 发送器和接收器。(a Transmitter and a Receiver 或驱动器和接收器)。  |

## 2 Modbus 数据链路层

### 2.1 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:

è 在**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。

è 在**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。

单播和广播模式的区别在一个多点的结构下(如 RS485)更加易于理解。

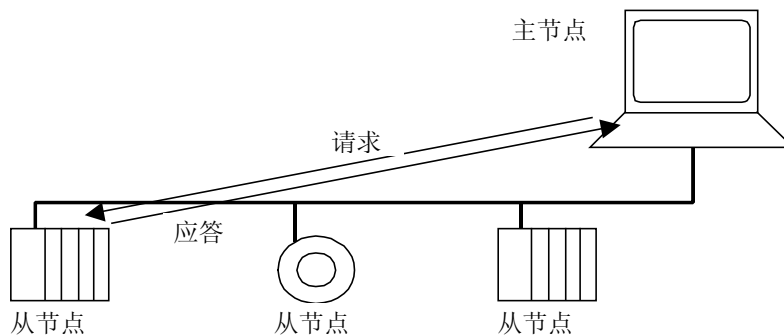


图 2: 单播模式

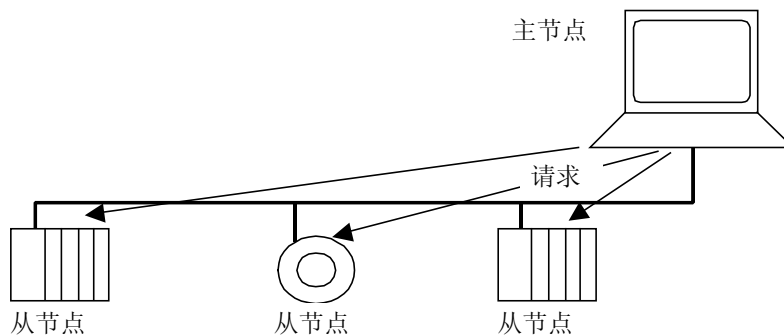


图 3: 广播模式



## 2.2 Modbus 地址规则

Modbus 寻址空间有 256 个不同地址。

|      |         |          |
|------|---------|----------|
| 0    | 1 ~ 47  | 248 ~ 55 |
| 广播地址 | 子节点单独地址 | 保留       |

地址 0 保留为广播地址。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。

## 2.3 Modbus 帧描述

Modbus 应用协议 [1] 定义了简单的独立于其下面通信层的**协议数据单元(PDU - Protocol Data Unit)**:

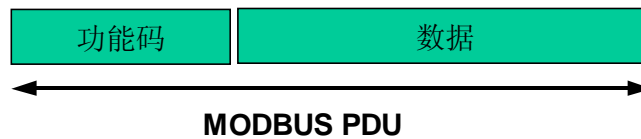


图 5: Modbus 协议数据单元

在不同总线或网络的 Modbus 协议映射在**协议数据单元**之外引入了一些附加的域。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适当的通信 PDU。

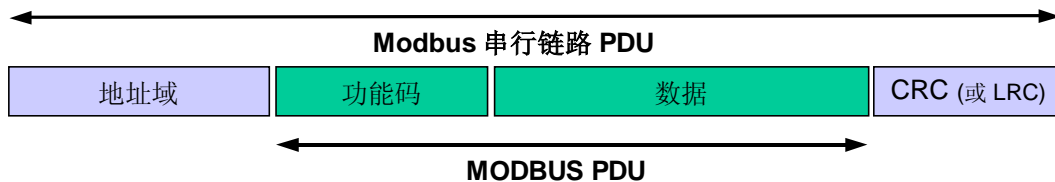


图 6: 串行链路上的 Modbus 帧

§ 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。(参见 2.5 节，“两种串行传输模式”)

## 2.4 主站/从站状态图

Modbus 由两个不同的子层组成：

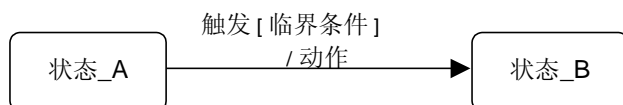
- 主/从协议
- 传输模式 (RTU 和 ASCII 模式)

下面的章节描述了主节点和子节点与传输模式无关的状态图。

RTU 和 ASCII 传输模式在下一章用两个状态图具体说明。描述了一个帧的接收和发送。

状态图词法：

下面的状态图使用与 UML 标准标记法绘制。标记法要点如下：



当一个系统处于 "状态\_A" 时发生 "触发" 事件，只有当 "临界条件" 为真时系统会转换到 "状态\_B"，然后，一个 "动作" 被执行。

### 2.4.1 主站状态图

下图描述了主节点的状态特征：

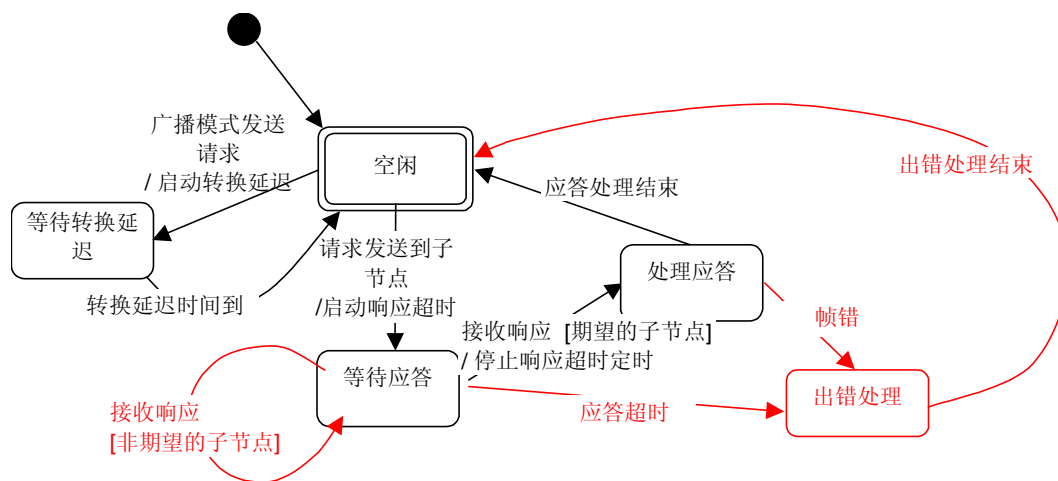


图 4： 主节点状态图

对上面的状态图的一些解释：

- § 状态 "空闲" = 无等待的请求。这是电源上电后的初始状态。只有在 "空闲" 状态请求才能被发送。发送一个请求后，主节点离开 "空闲" 状态，而且不能同时发送第二个请求。
- § 当单播请求发送到一个子节点，主节点将进入 "等待应答" 状态，同时一个临界超时定时启动。这个超时称为 "响应超时"。它避免主节点永远处于 "等待应答" 状态。响应超时的时间依赖于具体应用。
- § 当收到一个应答时，主节点在处理数据之前检验应答。在某些情况下，检验的结果可能为错误。如收到来自非期望的子节点的应答，或接收的帧错误。在收到来自非期望子节点的应答时，响应超时继续计时；当检测到帧错时，可以执行一个重试。

- § 响应超时但没有收到应答时，则产生一个错误。那么主节点进入“空闲”状态，并发出一个重试请求。重试的最大次数取决于主节点 的设置。
- § 当广播请求发送到串行总线上，没有响应从子节点返回。然而主节点需要进行延迟以便使子节点在发送新的请求处理完当前请求。该延迟被称作 “转换延迟”。因此，主节点会在返回能够发送另一个请求的“空闲”状态之前，到“等待转换延迟”状态。
- § 在单播方式，响应超时必须设置到足够的长度以使任何子节点都能处理完请求并返回响应。而广播转换延迟必须有足够的长度以使任何子节点都能只处理完请求而可以接收新的请求。因此，转换延迟应该比响应超时要短。典型的响应超时在 9600 bps 时从 1 秒到几秒，而转换延迟从 100 ms 到 200ms。
- § 帧错误包括：1) 对每个字符的奇偶校验; 2) 对整个帧的冗余校验。详细解释参见 §2.6 “差错检验方法”。

状态图以简洁的方式绘出。它没有包含对线路的访问、报文帧及传输错误重试等等。有关帧传输的细节，请参见 2.5 中的图，“两种串行式”。

### 2.4.2 从站状态图

下图描述了子节点的状态特征:

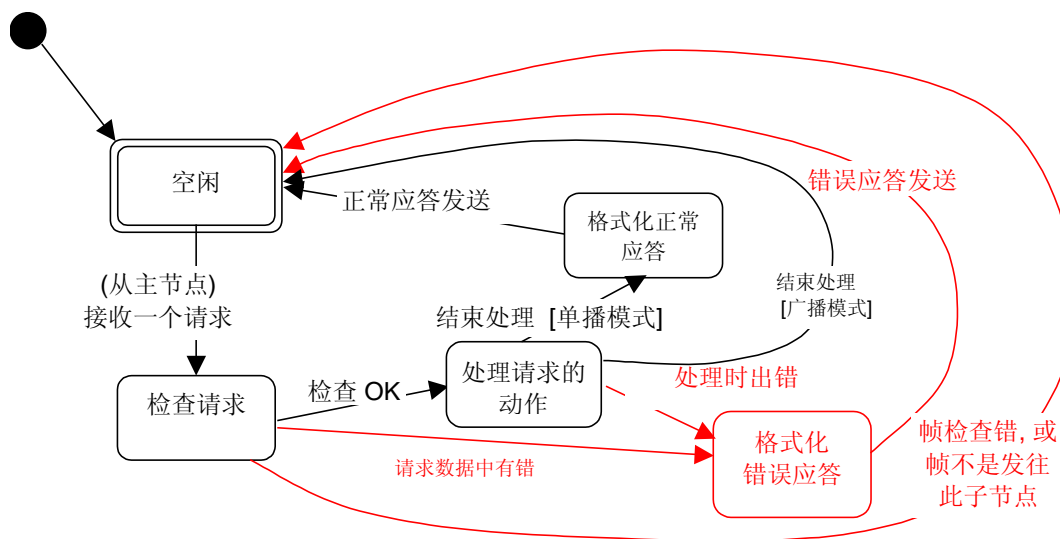


图 5: 子节点状态图

对上面的状态图的一些解释：

- § 状态“空闲”= 没有等待的请求。这是电源上电后的初始状态。
- § 当收到一个请求时，子节点在处理请求中要求的动作前检验报文包。不同的错误可以发生于：请求的格式错，非法动作，…… 当检测到错误时，必须向主节点发送应答。
- § 当要求的动作完成后，单播报文要求必须格式化一个应答并发往主节点。
- § 如果子节点在接收到的帧中检测到错误，则没有响应返回到主节点。
- § 任何子节点均应该定义并管理 Modbus 诊断计数器以提供诊断信息。通过使用 Modbus 诊断功能码，可以得到这些计数值。（参见 附录 A， 和 Modbus 应用协议规范 [1]）。

### 2.4.3 主站/从站通信时序图

下面的图显示了主/从通信的 3 种典型情况。

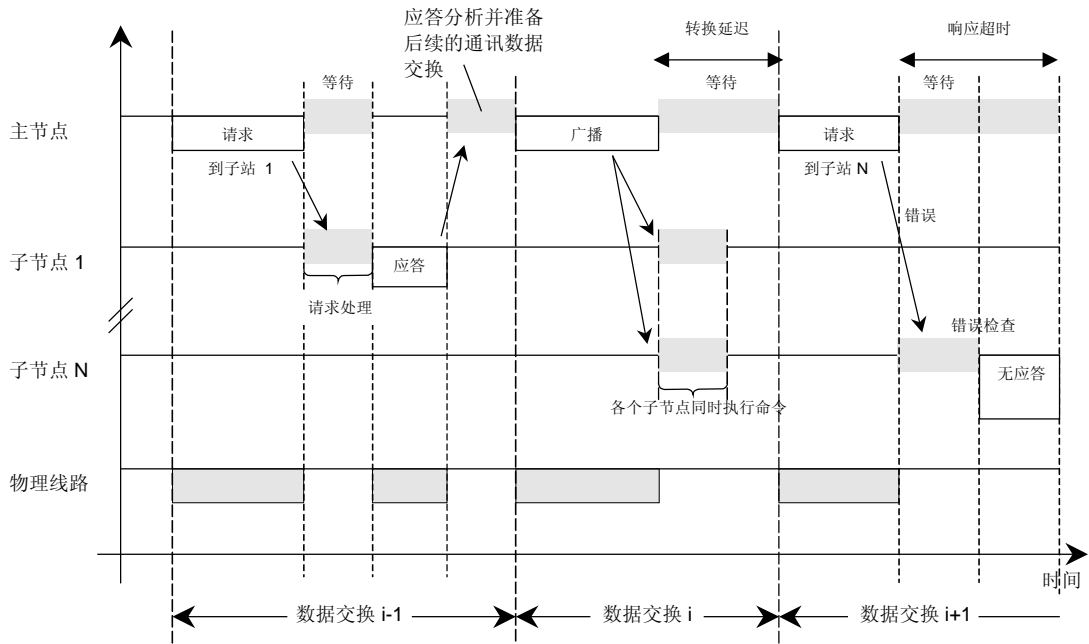


图 6: 各种情形的主/从通信时序图

注：

- § 请求， 应答， 广播阶段的持续时间依赖于通信特征 (帧长度和吞吐量)。
- § 等待和处理阶段的持续时间取决于子节点应用的请求处理时间。

## 2.5 两种串行传输模式

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。

设备应该由用户设成期望的模式，RTU 或 ASCII。默认设置必须为 RTU 模式。

### 2.5.1 RTU 传输模式

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)
- Bits per Byte:** 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)



图 7: 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:

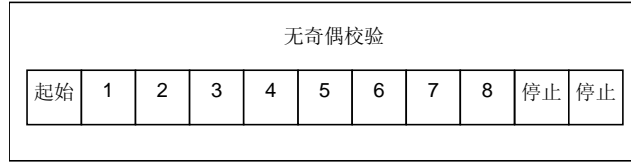


图 8: RTU 模式位序列 (无校验的特殊情况)

**帧检验域:** 循环冗余校验 (CRC)

**帧描述 :**

|       |      |            |                                      |
|-------|------|------------|--------------------------------------|
| 子节点地址 | 功能代码 | 数据         | CRC                                  |
| 1 字节  | 1 字节 | 0 到 252 字节 | 2 字节<br><small>CRC 低   CRC 高</small> |

图 9: RTU 报文帧

è Modbus RTU 帧最大为 256 字节。

2.5.1.1 Modbus 报文 RTU 帧

由发送设备将 Modbus 报文构造为带有已知起始和结束标记的帧。这使设备可以在报文的开始接收新帧，并且知道何时报文结束。不完整的报文必须能够被检测到而错误标志必须作为结果被设置。在 RTU 模式，报文帧由时长至少为 3.5 个字符时间的空闲间隔区分。在后续的部分，这个时间区间被称作 t3.5。

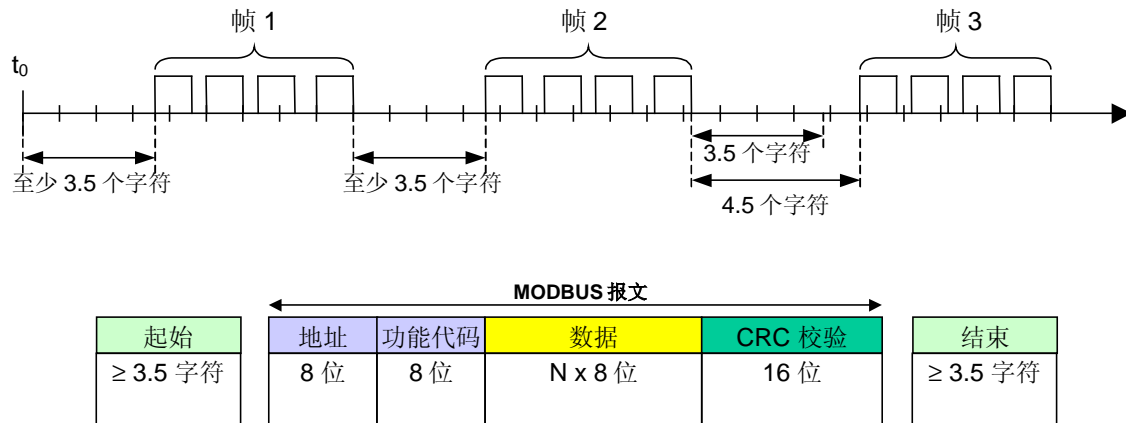
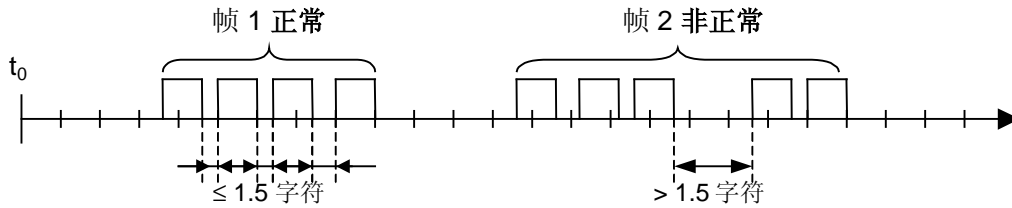


图 10: RTU 报文帧

整个报文帧必须以连续的字符流发送。  
如果两个字符之间的空闲间隔大于 1.5 个字符时间，则报文帧被认为不完整应该被接收节点丢弃。



注：

RTU 接收驱动程序的实现，由于  $t_{1.5}$  和  $t_{3.5}$  的定时，隐含着大量的对中断的管理。在高通信速率下，这导致 CPU 负担加重。因此，在通信速率等于或低于 19200 Bps 时，这两个定时必须严格遵守；对于波特率大于 19200 Bps 的情形，应该使用 2 个定时的固定值：建议的字符间超时时间( $t_{1.5}$ )为  $750\mu s$ ，帧间的超时时间 ( $t_{1.5}$ ) 为 1.750ms。

下图表示了对 RTU 传输模式状态图的描述。“主节点”和“子节点”的不同角度均在相同的图中表示：

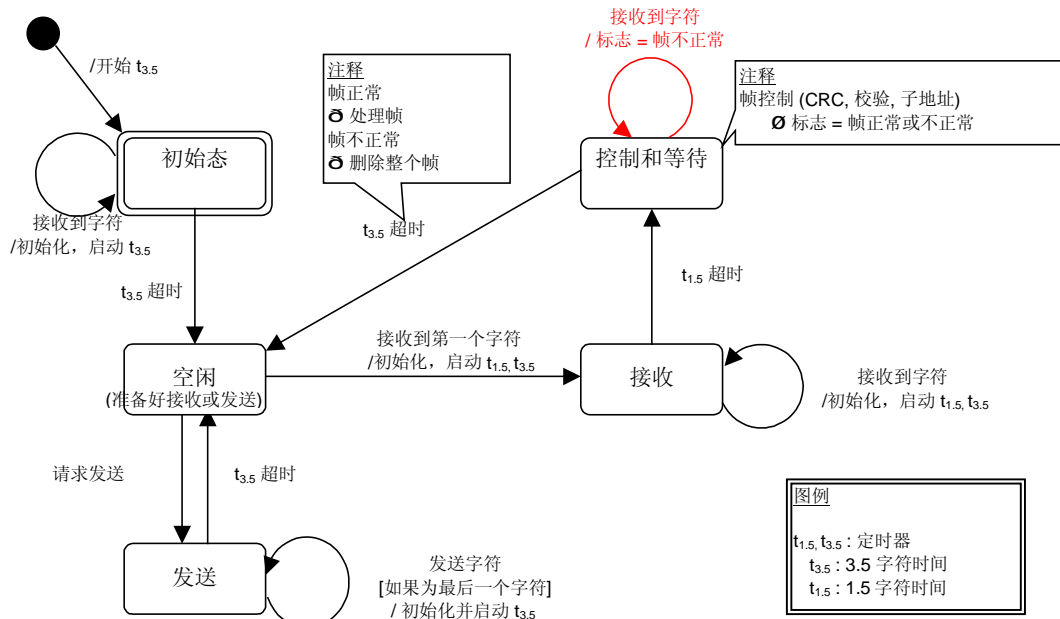


图 11: RTU 传输模式状态图

上面状态图的一些解释：

- § 从“初始”态到“空闲”态转换需要  $t_{3.5}$  定时超时：这保证帧间延迟
- § “空闲”态是没有发送和接收报文要处理的正常状态。
- § 在 RTU 模式，当没有活动的传输的时间间隔达 3.5 个字符长时，通信链路被认为在“空闲”态。
- § 当链路空闲时，在链路上检测到的任何传输的字符被识别为**帧起始**。链路变为“活动”状态。然后，当链路上没有字符传输的时间间隔达到  $t_{3.5}$  后，被识别为**帧结束**。
- § 检测到帧结束后，完成 CRC 计算和检验。然后，分析地址域以确定帧是否发往此设备，如果不是，则丢弃此帧。为了减少接收处理时间，地址域可以在一接到就分析，而不需要等到整个帧结束。这样，CRC 计算只需要在帧寻址到该节点（包括广播帧）时进行。

### 2.5.1.2 CRC 校验

在 RTU 模式包含一个对全部报文内容执行的，基于循环冗余校验 (CRC - Cyclical Redundancy Checking) 算法的错误检验域。CRC 域检验整个报文的内容。不管报文有无奇偶校验，均执行此检验。

CRC 包含由两个 8 位字节组成的一个 16 位值。

CRC 域作为报文的最后的域附加在报文之后。计算后，首先附加低字节，然后是高字节。CRC 高字节为报文发送的最后一个字节。

附加在报文后面的 CRC 的值由发送设备计算。接收设备在接收报文时重新计算 CRC 的值，并将计算结果于实际接收到的 CRC 值相比较。如果两个值不相等，则为错误。

CRC 的计算，开始对一个 16 位寄存器预装全 1。然后将报文中的连续的 8 位子节对其进行后续的计算。只有字符中的 8 个数据位参与生成 CRC 的运算，起始位，停止位和校验位不参与 CRC 计算。

CRC 的生成过程中，每个 8-位字符与寄存器中的值异或。然后结果向最低有效位(LSB)方向移动(Shift) 1 位，而最高有效位(MSB)位置充零。然后提取并检查 LSB：如果 LSB 为 1，则寄存器中的值与一个固定的预置值异或；如果 LSB 为 0，则不进行异或操作。

这个过程将重复直到执行完 8 次移位。完成最后一次（第 8 次）移位及相关操作后，下一个 8 位字节与寄存器的当前值异或，然后又同上面描述过的一样重复 8 次。当所有报文中子节都运算之后得到的寄存器中的最终值，就是 CRC。

当 CRC 附加在报文之后时，首先附加低字节，然后是高字节。在附录 B 含有 CRC 生成的详细示例。



### 2.5.2 ASCII 传输模式

当 Modbus 串行链路的设备被配置为使用 ASCII (American Standard Code for Information Interchange) 模式通信时，报文中的每个 8 位子节以两个 ASCII 字符发送。当通信链路或者设备无法符合 RTU 模式的定时管理时使用该模式。

注：由于一个子节需要两个字符，此模式比 RTU 效率低。

例：子节 0X5B 会被编码为两个字符：0x35 和 0x42 (ASCII 编码 0x35 ="5", 0x42 ="B" )。

#### ASCII 模式每个字节 (10 位) 的格式为：

- 编码系统:** 十六进制, ASCII 字符 0-9, A-F。  
报文中每个 ASCII 字符含有 1 个十六进制字符
- Bits per Byte:** 1 起始位  
7 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)



图 12: ASCII 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:



图 13: ASCII 模式模式位序列(无校验的特殊情况)

**帧检验域:**纵向冗余校验 (LRC - Longitudinal Redundancy Checking)

### 2.5.2.1 Modbus ASCII 报文帧

由发送设备将 Modbus 报文构造为带有已知起始和结束标记的帧。这使设备可以在报文的开始接收新帧，并且知道何时报文结束。不完整的报文必须能够被检测到而错误标志必须作为结果被设置。

报文帧的地址域含有两个字符。

在 ASCII 模式，报文用特殊的字符区分帧起始和帧结束。一个报文必须以一个‘冒号’（:）(ASCII 十六进制 3A)起始，以‘回车-换行’(CR LF)对 (ASCII 十六进制 0D 和 0A)结束。

注：LF 字符可以通过特定的 Modbus 应用命令 (参见 Modbus 应用协议规范) 改变。

对于所有的域，允许传送的字符为十六进制 0-9，A-F (ASCII 编码)。设备连续的监视总线上的‘冒号’字符。当收到这个字符后，每个设备解码后续的字符一直到帧结束。

报文中字符间的时间间隔可以达一秒。如果有更大的间隔，则接受设备认为发生了错误。

下图显示了一个典型的报文帧。

| 起始        | 地址   | 功能   | 数据              | LRC  | 结束            |
|-----------|------|------|-----------------|------|---------------|
| 1 字符<br>: | 2 字符 | 2 字符 | 0 到 to 2x252 字符 | 2 字符 | 2 字符<br>CR,LF |

图 14: ASCII 报文帧

注：每个字符子节需要用两个字符编码。因此，为了确保 ASCII 模式和 RTU 模式在 Modbus 应用级兼容，ASCII 数据域最大数据长度为 (2x252) 是 RTU 数据域 (252) 的两倍。

必然的，Modbus ASCII 帧的最大尺寸为 513 个字符。

ASCII 报文帧的要求在下面的状态图中综合。“主节点”和“子节点”的不同角度均在相同的图中表示：

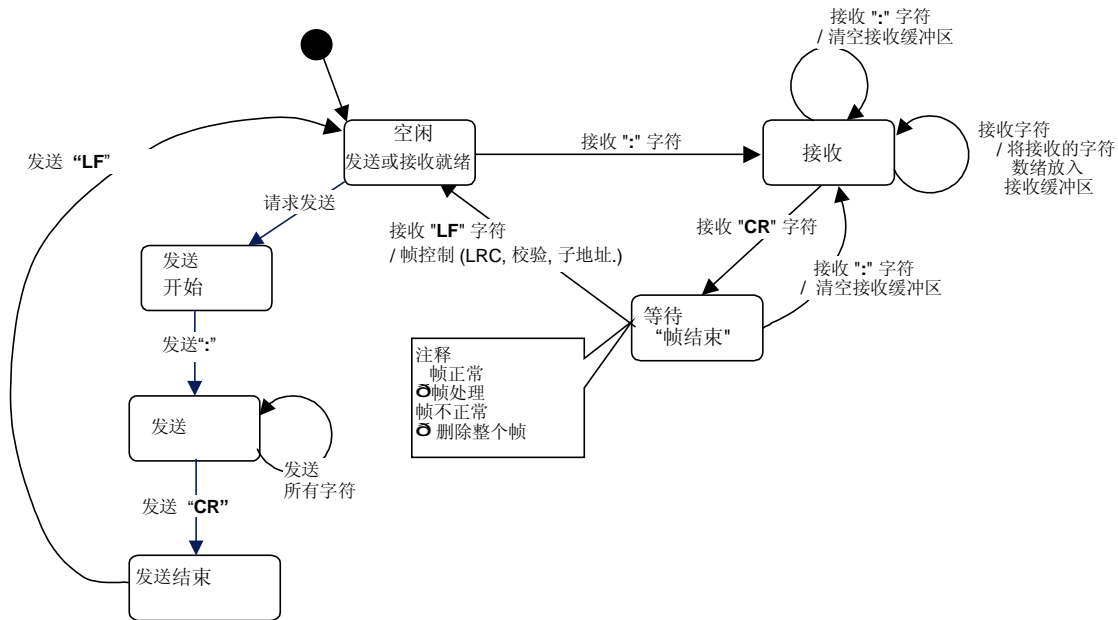


图 15: ASCII 传输模式状态图

上面状态图的一些解释:

- § “空闲”态是没有发送和接收报文要处理的正常状态。
- § 每次接收到 ":" 字符表示新的报文的开始。如果在一个报文的接收过程中收到该字符，则当前地报文被认为不完整并被丢弃。而一个新的接收缓冲区被重新分配。
- § 检测到帧结束后，完成 LRC 计算和检验。然后，分析地址域以确定帧是否发往此设备，如果不是，则丢弃此帧。为了减少接收处理时间，地址域可以在一接到就分析，而不需要等到整个帧结束。

### 2.5.2.2 LRC 校验

在 ASCII 模式，包含一个对全部报文内容执行的，基于纵向冗余校验 (LRC - Longitudinal Redundancy Checking) 算法的错误检验域。LRC 域检验不包括起始“冒号”和结尾 CRLF 对的整个报文的内容。不管报文有无奇偶校验，均执行此检验。

LRC 域为一个子节，包含一个 8 位二进制值。LRC 值由发送设备计算，然后将 LRC 附在报文后面。接收设备在接收报文时重新计算 LRC 的值，并将计算结果于实际接收到的 LRC 值相比较。如果两个值不相等，则为错误。

LRC 的计算，对报文中的所有的连续 8 位字节相加，忽略任何进位，然后求出其二进制补码。执行检验针对不包括起始“冒号”和结尾 CRLF 对的整个 ASCII 报文域的内容。在 ASCII 模式，LRC 的结果被 ASCII 编码为两个字节并放置于 ASCII 模式报文帧的结尾，CRLF 之前。

附录 B 含有 LRC 生成的详细示例。

## 2.6 差错检验方法

标准 Modbus 串行链路的可靠性基于两种错误检验:

§ 奇偶校验 (偶或奇) 应该被每个字符采用。

§ 帧检验 (LRC or CRC) 必须运用于整个报文。

由设备 (主节点或子节点) 生成的字符检验和帧检验发送前附加于报文体。设备 (子节点或主节点) 在接收时检验每个字符和整个报文。

主节点被用户配置为在放弃事务处理前等待一个预定的超时间隔 (响应超时)。这个间隔被设置成任何子节点有足够的时间正常响应 (单播请求)。如果子节点检测到错误, 则报文不起作用。子节点将不会构造对主节点的响应。因此, 将达到超时时间能使主节点的程序处理错误。注意, 当寻址到不存在的子设备的报文也会导致超时错误。

### 2.6.1 奇偶检验

用户可以配置设备使用偶 (要求的) 或奇校验, 或无校验 (建议的)。这将确定每个字符的奇偶位如何设置。

无论指定了偶还是奇校验, 则数据部分的为 1 的位的总数被计数 (ASCII 模式 7 数据位, RTU 8 数据位)。而奇偶位会被设置为 0 或 1 以使为 1 的位的总数为偶数或奇数。

例如, RTU 字符帧的数据为:

1100 0101

为 1 的位的总数为 4。如果使用偶校验, 帧的奇偶位为 0, 使为 1 的位的总数仍然为偶数(4); 如果使用奇校验, 帧的奇偶位为 1, 使为 1 的位的总数为奇数(5)。

当报文发送时, 奇偶位被计算并作用于每个字符帧。接收的设备计算为 1 的位的总数, 如果与设备配置不符, 则设置错误标记。(Modbus 串行链路的所有设备必须被配置成使用相同的奇偶检验方法)。注意, 奇偶检验只能检测到一个字符帧在传输过程中奇数个的增加或丢失的位。例如, 假如使用奇校验, 字符帧中含有的 3 个为 1 的位丢失了两个, 而为 1 的位的计数的结果仍然为奇数。

如果没有指定奇偶检验, 奇偶位不会被传送, 也不可以进行奇偶检验: 一个附加的位被传送以填充字符帧。

### 2.6.2 帧检验

依赖于传输模式, 两种检验方法被使用: RTU 或 ASCII。

§ 在 RTU 模式, 包含一个对全部报文内容执行的, 基于循环冗余校验 (CRC - Cyclical Redundancy Checking) 算法的错误检验域。CRC 域检验整个报文的内容。不管报文有无奇偶校验, 均执行此检验。

§ 在 ASCII 模式, 包含一个对全部报文内容执行的, 基于纵向冗余校验 (LRC - Longitudinal Redundancy Checking) 算法的错误检验域。LRC 域检验不包括起始“冒号”和结尾 CRLF 对整个报文的内容。不管报文有无奇偶校验, 均执行此检验。

有关差错检验方法的详细内容, 参见前面的章节。

### 3 物理层

#### 3.1 引言

新的串行链路上的 MODBUS 解决方案应该按照 EIA/TIA-485(即已知的 RS485 标准)实现电气接口。该标准允许“两线结构”的点对点 and 多点系统。此外,某些设备可能实现“四线”RS485 接口。

设备也可能实现 RS232 接口。

在这种 MODBUS 系统中,一个主站和一个或几个从站在一个无源串行链路上通信。

在标准的 MODBUS 系统中,所有设备(并行)连结在一条由 3 条导线组成的干线电缆上。其中两条导线(“两线”结构)形成一对平衡双绞线,双向数据在其上传送,典型比特率为每秒 9600 比特。每台设备可能连结(见图 19):

- 或是双向连到主干电缆上,形成菊花链,
- 或是经分支电缆连到一个无源接头上,
- 或是经特种电缆连到一个有源接头上。

在设备上可用螺钉端子, RJ45, 或 9 芯 D-型连接器与电缆相接(见“机械接口”章节)。

#### 3.2 数据信号发送速率

要求 9600bps 波特率。

推荐 19200bps 波特率。

该值(19200)必须被作为约定值来实现。

其它波特率可选择来实现: 1200, 2400, 4800, … 38400bps, 56Kbps, 115Kbps, …

每种波特率,对发送方,要求其精度必须高于 1%,而对接收方,必须允许 2% 误差。

### 3.3 电气接口

#### 3.3.1 多点串行总线结构

图 19 展现的是 MODBUS 多点串行链路系统中串行总线结构的总貌。

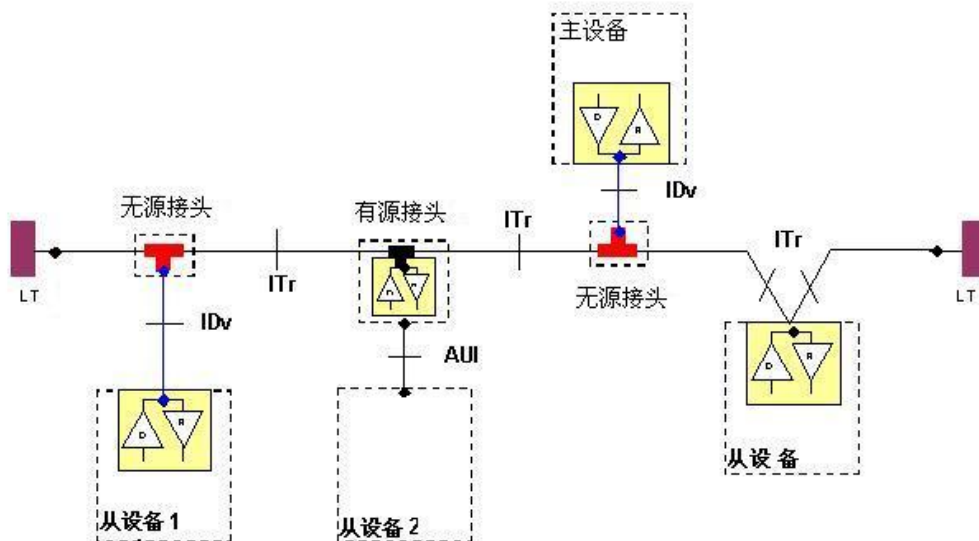


图 19: 串行总线基本结构

一个 MODBUS 多点串行链路系统是由主电缆(主干),和一些可能的分支电缆组成。

在主干电缆的两端需要有线路终端以使阻抗匹配(详见 § “2 线-MODBUS 定义” 和 “可选 4 线-MODBUS 定义” )。

如图 19 所示,不同的设备可以在同一个 MODBUS 串行链路系统中运行:

- § 集成有通信收发器的设备通过**无源接头**和分支电缆连接到主干上(例如从站 1 和主站);
- § 没有集成通信收发器的设备通过**有源接头**和分支电缆连接到主干上(有源接头集成有收发器)(例如从站 2);
- § 设备以**菊花链形式**直接连接到主干电缆上(例如从站 n)

我们采用下列规定:

- § 主干间的接口称为 **ITr**(主干接口)
- § 设备和**无源接头**间的接口称为 **IDv** (分支接口)
- § 设备和有源接头间的接口称为 **AUI** (附加单元接口)

**注:**

1. 某些情况下,接头可能直接连接到设备的 IDv-插槽或 AUI-插槽上,而不使用分支电缆。
2. 一个接头可能有几个 IDv 插槽以连接几台设备。当它是无源接头时,称为**分配器**。

3. 当使用有源接头时，可以通过接头的 AUI 或 ITr 接口向其提供电源。

在接下来的章节中，我们将介绍 ITr 和 IDv 接口(见 §“2 线-MODBUS 定义” 和 “4 线-MODBUS 定义” )

### 3.3.2 2 线-MODBUS 定义

串行链路上的 MODBUS 解决方案应当依照 EIA/TIA-485 标准实现 “2-线” 电气接口。

在这个 2 线-总线上，在任何时候只有一个驱动器有权发送信号。

实际上，还有第三条导线把总线上所有设备相互连接：公共地。

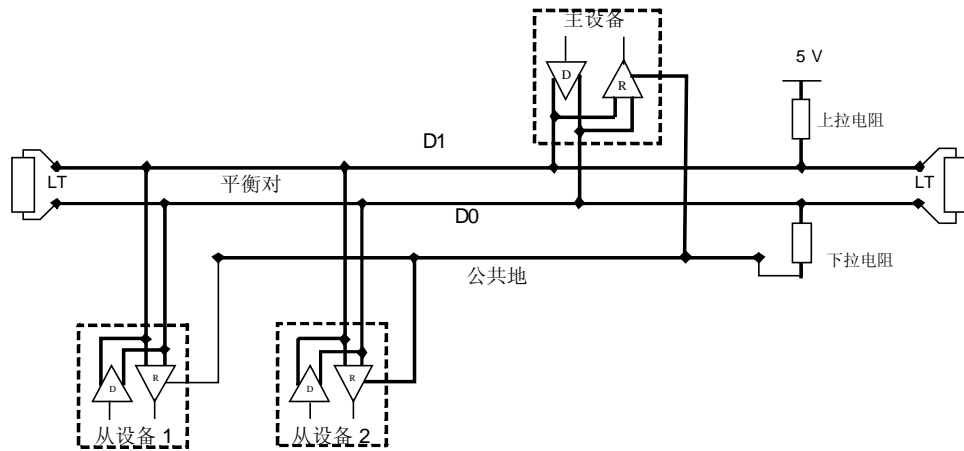


图 20: 2-线制的一般拓扑结构

#### 2 线-MODBUS 电路定义

| 所需电路    |         | 设备  | 设备需求 | EIA/TIA-485 的命名 | 说明  |
|---------|---------|-----|------|-----------------|---|
| 在 ITr 上 | 在 IDv 上 |     |      |                 |   |
| D1      | D1      | I/O | X    | B/B'            | 收发器端子 1, V1 电压<br>(V1 > V0 表示二进制 1[OFF] 状态) |
| D0      | D0      | I/O | X    | A/A'            | 收发器端子 0, V0 电压<br>(V0 > V1 表示二进制 0[ON] 状态)  |
| 公共地     | 公共地     | --  | X    | C/C'            | 信号和可选的电源公共地                                 |

注:

- 对于线路终端 (TL)，即上拉和下拉电阻，请参考“多点系统要求”部分。
- 在与设备和接头有关的文件(用户指南，连线指南，…)中，必须使用 D0, D1 和公共地的电路名字，以提高互操作能力。
- 可以增加可选的电气接口，例如:

§ 电源: 5·24 V 直流电。

§ 端口模式控制: PMC 电路(TTL 兼容)。当需要的时候，可由这个外电路和/或另一种方式(如该

设备上的一个开关)来控制端口模式。在第一种情况下, 尽管一个开路 PMC 将要求 2-线 MODBUS 模式, 但实际上, 在 PMC 上的低电平将把端口置于 4-线 MODBUS 模式或 RS232-MODBUS 模式。

### 3.3.3 可选的 4 线-MODBUS 定义

这种 MODBUS 设备同样允许实现 2 对总线(4 线)单向数据传输。在**主对总线**(RXD1-RXD2)上的数据只能由从站接收, 而在**从对总线**(TXD0-TXD1)上的数据只能由主站接收。

实际上, 公共地作为第五条导线必须把 4-线总线上的所有设备相互连接。

和 2 线-MODBUS 一样, 在任何时刻只有一个驱动器有权力发送数据。

这种设备必须**依照 EIA/TIA-485** 对每一对平衡线实现一个驱动器和一个收发器。(有时候这种方式被称为“RS422”, 这是错误的: RS422 标准不支持几台设备在一对平衡线上。)

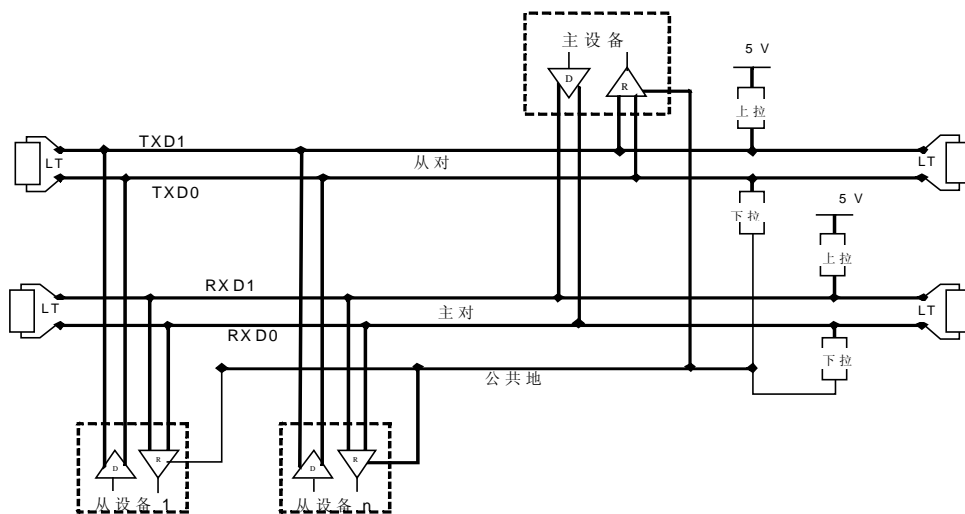


图 21: 4--线制的一般拓扑结构



可选 4 线-MODBUS 电路定义

| 所需电路        |         | 设备  | 设备需求       | EIA/TIA-485<br>的命名 | 对 IDv 的说明                                    |
|-------------|---------|-----|------------|--------------------|--|
| 在 Itr 上     | 在 IDv 上 |     |            |                    |  |
| <b>TXD1</b> | TXD1    | Out | <b>X</b>   | <b>B</b>           | 发生器端子 1, Vb 电压<br>(Vb>Va 表示二进制 1[OFF]状态)     |
| <b>TXD0</b> | TXD0    | Out | <b>X</b>   | <b>A</b>           | 发生器端子 0, Va 电压<br>(Va > Vb 表示二进制 0[ON]状态)    |
| <b>RXD1</b> | RXD1    | In  | <b>(1)</b> | <b>B'</b>          | 接收器端子 1, Vb'电压<br>(Vb' > Va'表示二进制 1[OFF]状态 ) |
| <b>RXD0</b> | RXD0    | In  | <b>(1)</b> | <b>A'</b>          | 接收器端子 0, Va'电压<br>(Va' > Vb'表示二进制 0[ON]状态 )  |
| <b>公共地</b>  | 公共地     | --  | <b>X</b>   | <b>C/C'</b>        | 信号和可选的电源公共地                                  |

注:

- 对于线路终端 (LT)，即上拉和下拉电阻，请参考“多点系统要求”部分。
- 仅当选择实现 4 线-MODBUS 时，才需表中**(1)**所示那些电路。
- 在与设备和接头有关的文件(用户指南，连线指南，…)中，必须使用这 5 类所需电路的名字，以提高互操作能力。
- 可以增加可选的电气接口，例如：

§ 电源： 5··24V 直流电。

§ PMC 电路： 见前述(2 线-MODBUS 电路定义)中关于该可选电路的注。

### 3.3.3.1 4 线-电缆系统中的重点

在这种 4 线-MODBUS 中，主、从站均有带相同 5 类所需电路的 IDv 接口。

作为主站应该：

- 自从对总线 (TXD1-TXD0) 上接收来自从站的数据，
- 在主对总线 (RXD1-RXD0) 上发送数据，由从站接收，

**4 线-电缆系统必须在 ITr 与主站的 IDv 之间，使两对总线交叉：**

|    | 主站 IDv 上的信号 |     | EIA/TIA-485 的命名 | ITr 上的电路 |
|----|-------------|-----|-----------------|----------|
|    | 名字          | 类型  |                 |          |
| 从对 | RXD1        | In  | B'              | TXD1     |
|    | RXD0        | In  | A'              | TXD0     |
| 主对 | TXD1        | Out | B               | RXD1     |
|    | TXD0        | Out | A               | RXD0     |
|    | 公共地         | --  | C/C'            | 公共地      |

这种交叉由交叉电缆实现，但是在 2 线系统中这种交叉电缆的连接可能会引起损害。解决连接 4 线主站的好方法是使用含有交叉功能的接头。

### 3.3.3.2 4 线与 2 线电缆的兼容性

为了将执行 2 线物理接口的设备接入一个已存在的 4 线系统，4 线电缆系统可以按下述修改：

- § TXD0 信号应与 RXD0 信号连接，使之成为 D0 信号。
- § TXD1 信号应与 TXD0 信号连接，使之成为 D1 信号。
- § 上拉，下拉电阻和线路终端电阻应重新安排以正确地适应 D0，D1 信号。

下图给出一个使用 2 线接口的从站 2 和 3 能与使用 4 线接口的主站和从站 1 一起工作的例子。

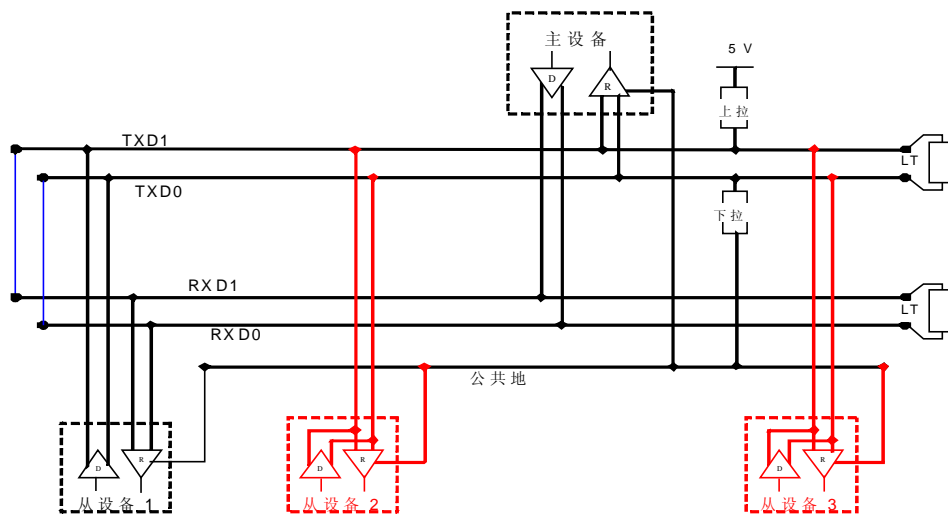


图 22： 将 4-线电缆系统变为 2-线电缆系统

为了将执行 4 线物理接口的设备接入一个已存在的 2 线系统，该新接入设备的 4 线接口可以按下述安排：

在每一个 4 线设备接口上：

- § TXD0 信号应与 RXD0 信号连接，之后连接到主干的 D0 信号线上；
- § TXD1 信号应与 RXD0 信号连接，之后连接到主干的 D1 信号线上。

下图给出一个使用 4 线制的从站 2 和 3 能与使用 2 线制的主站和从站 1 一起工作的例子。

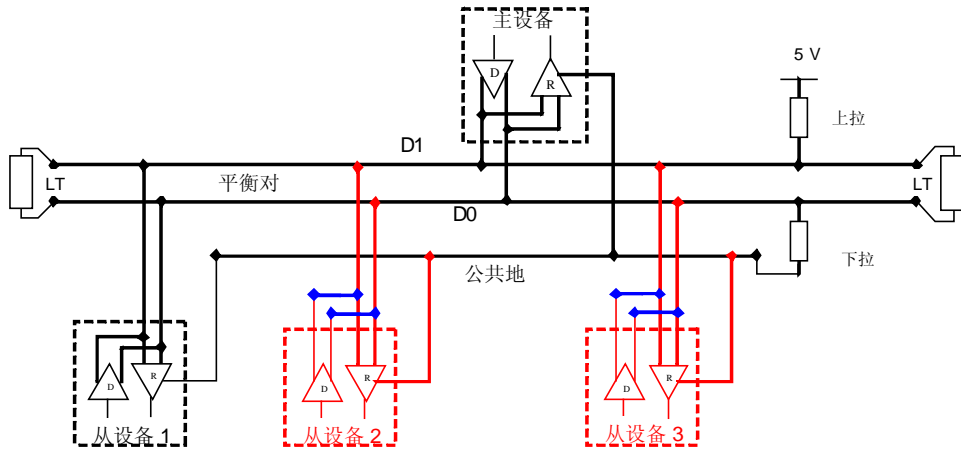


图 23： 将带 4-线接口的设备连接到 2-线电缆系统中

### 3.3.4 RS232-MODBUS 定义

某些设备是应用 RS232 接口以实现 DCE 和 DTE 通信。

#### RS232-MODBUS 的电路定义

| 信号  | DCE | <u>DCE(1)要求</u> | <u>DTE(1)要求</u> | 备注                    |
|-----|-----|-----------------|-----------------|-----------------------|
| 公共地 | --  | <b>X</b>        | <b>X</b>        | 信号地                   |
| CTS | In  |                 |                 | 为发送而清除                |
| DCD | --  |                 |                 | 被侦测数据载波 (从 DCE 到 DTE) |
| DSR | In  |                 |                 | 数据设置就绪                |
| DTR | Out |                 |                 | 数据终端就绪                |
| RTS | Out |                 |                 | 请求发送                  |
| RXD | In  | <b>X</b>        | <b>X</b>        | 接收的数据                 |
| TXD | Out | <b>X</b>        | <b>X</b>        | 发送的数据                 |

注：

- 标有“X”的信号只在选择执行 RS232-MODBUS 时才需要。
- 信号都要符合 EIA/TIA-232 标准。
- 每个 TXD 都与另一设备的 RXD 连接。

- RTS 可以与另一设备的 CTS 连接。
- DTR 可以与另一设备的 DSR 连接。
- 可以增加可选的电气接口，例如：
  - **电源：** 5··24V 直流电源。
  - **PMC 电路：** 见上述（2 线-MODBUS 电路定义）关于该可选电路注。

### 3.3.5 RS232-MODBUS 要求

这种可选串行链路系统上的 MODBUS 只应用于短距离（一般小于 20m）的点到点的互连。

其次，必须遵守 EIA/TIA-232 标准：

⇒ 电路定义。

⇒ 最大线路对地电容量（2500pF，对 100pF/m 的电缆，则长为 25m）。

关于屏蔽和使用第 5 类电缆的可能性，请参阅“电缆”章节。

设备提供的文件必须指出：

⇒ 该设备是否必须作为 DCE 或是 DTE，

⇒ 若是上述情况，可选的电路该怎样工作。

### 3.4 多点系统需求

对于任何 EIA/TIA-485 多点系统，无论是 2 线配置还是 4 线配置，以下要求均适用。

#### 3.4.1 无中继器情况下，最大设备数量

在没有中继器的任何 RS485-MODBUS 系统中，总是允许最多有 **32 台设备**。

其依据是：

- 所有允许的地址，
- 设备使用的 RS485 单元负载总量，
- 以及需要的线偏置，

一个 RS485 系统可以容纳大量的设备。在没有中继器情况下，某些设备可在设备数大于 32 台的 RS485-MODBUS 串行链路中运行。

该种情况下，必须为这些 MODBUS 设备提供文件，说明没有中继器时能允许有多少此类设备。

在两个重载的 RS485-MODBUS 之间使用中继器也是可以的。

#### 3.4.2 拓扑结构

没有配置中继器的 RS-485 MODBUS 有一个主干电缆，所有的设备沿着它直接（菊花链）或通过短的分支电缆连接起来。

主干电缆，又称总线，可以很长（见下面），它的两端必须连接在线路终端上。

在多个 RS-485 MODBUS 之间使用中继器也是可以的。

#### 3.4.3 长度

**主干电缆端到端**的长度必须有限制。其长度由波特率，电缆（规格，电容或特征阻抗），菊花链上的负载数，以及网络配置（*2 线或 4 线制*）所决定。

对于最高波特率为 9600，AWG26（或更粗）规格的电缆，其最大长度为 1000m。在图 22 所示特殊情况下（4 线制用作 2 线制的系统中）最大长度必须除以 2。

分支必须短，不能超过 20m。如果使用 n 分支的多口接头，每个分支最大长度必须限制为 40m 除以 n。

#### 3.4.4 接地形式

《公共地》电路（信号与可选电源公共地）必须直接连到保护地上，最好是整条总线只接在一点。通常该点可选在主站上或其接头上。

#### 3.4.5 线路终端

沿线路传播的移动信号波遇到不连续的阻抗，造成在传输线路中的反射。为了使在 RS-485 电缆终端的反射最小，需要在接近总线两端点处放置线路终端。

由于传播是双向的，故在线路两端都加置终端是非常重要的，但在一个无源 D0-D1 平衡对线上，加的 LT 不能超过 2 个。也不能在分支电缆上放置任何 LT。

每个线路终端必须连接在平衡线 D0 和 D1 的两条导线之间。

线路终端可以是 150 欧姆（0.5W）的电阻。

当对线极性偏置时（见下述），最好选择电容（1nF，最低 10V）与 120 欧姆（0.25W）电阻串联。

在 4 线系统中，在总线的两端，每一对线都必须有终端。

在 RS232 互联中，可以不用连接终端。

### 3.4.6 线路极性偏置

当没有数据在 RS-485 平衡对线上传递时，该线路不被驱动，因此易受外部噪声与干扰的影响。为确保它的接收器处于一个稳定状态，在没有数据信号出现时，一些设备需要使网络偏置。

每个 MODBUS 设备都必须用文件说明：

- 该设备是否需要线路极性偏置，
- 该设备是否执行或可执行如此线路极性偏置。

如果一个或多个设备需要线路极性偏置，则必须在该 RS-485 平衡对线上接一对电阻：

- D1 线上的上拉电阻至 5V 电压，
- D0 线上的下拉电阻至公共地线。

这些电阻的值必须介于 450 和 650 欧姆之间，650 欧姆的电阻值可以允许在串行链路总线上有较多设备。

在这种情况下，对在一个局部区域的整个串行总线，必须实现对线的极性偏置。通常该点选在主站或其接头上。其他设备不可实现任何极性偏置。

在此类 MODBUS 串行链路上允许的最多设备数，比无极性偏置的 MODBUS 系统少 4 个。

### 3.5 机械接口

螺钉端子可用在 IDv 与 ITr 两种连接中。有关每个信号确切位置的所有信息都必须提供给用户，这些信号名称要符合前面“电气接口”章节所述。

如果一台设备上使用一个 RJ45（或小型 DIN 或 D 型）连接器作为 MODBUS 机械接口，则必须选用屏蔽母连接器。而电缆终端必须使用屏蔽的公连接器。

#### 3.5.1 2 线-MODBUS 连接器的输出引脚

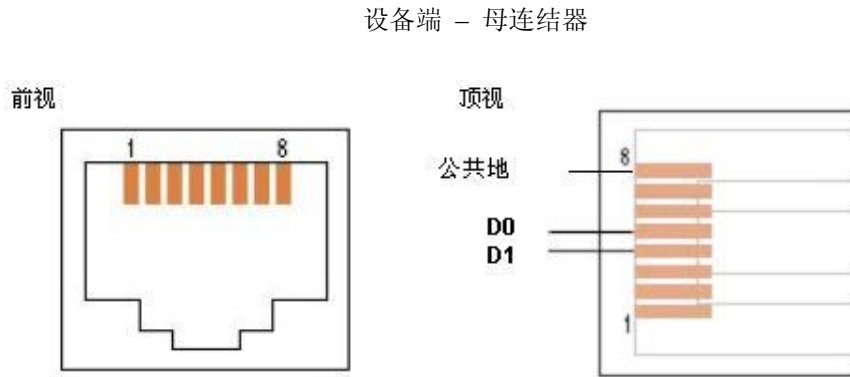


图 24：2 线-MODBUS 中使用的 RJ45 连接器( 必需的输出引脚 )

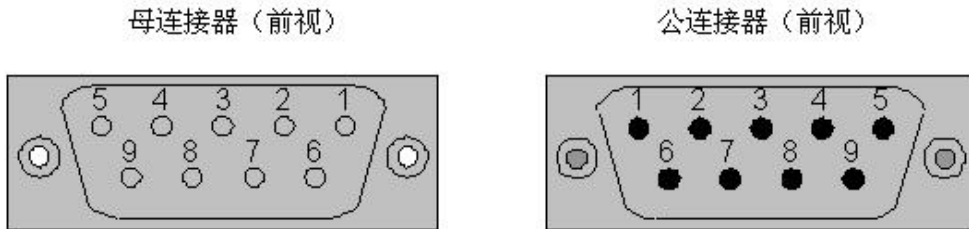


图 25： 9 引脚 D 型连接器

也可使用螺钉型的连接器。

若一台标准的 MODBUS 设备使用 RJ45 或 9 引脚 D 型连接器，对每种实际电路必须注意下述输出引脚。

2 线-MODBUS 的 RJ45 和 9 引脚 D 型连接器输出引脚

| RJ45 引脚 | D9 型连接器引脚 | 级别要求 | IDv 电路 | ITr 电路 | EIA/TIA-485 命名 | IDv 中的描述                                   |
|---------|-----------|------|--------|--------|----------------|--|
| 3       | 3         | 可选   | PMC    | --     | --             | 端口模式控制                                     |
| 4       | 5         | 必需   | D1     | D1     | B/B'           | 收发器端子 1, V1 电压 (V1 > V0 表示二进制的 1 [OFF] 状态) |
| 5       | 9         | 必需   | D0     | D0     | A/A'           | 收发器端子 0, V0 电压 (V0 > V1 表示二进制的 0 [ON] 状态)  |
| 7       | 2         | 推荐   | VP     | --     | --             | 正 5...24 V 直流电源                            |
| 8       | 1         | 必需   | 公共地    | 公共地    | C/C'           | 信号和电源公共地                                   |

3.5.2 可选 4 线-MODBUS 连接器输出引脚

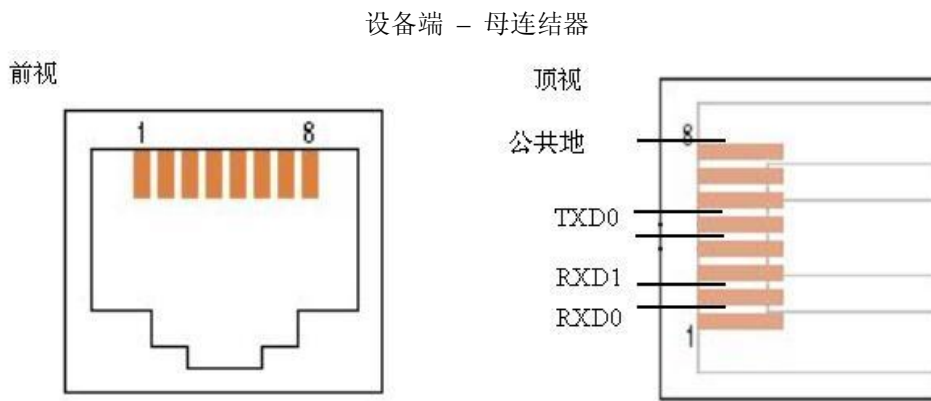


图 26: 4 线 MODBUS 上的 RJ45 连接器( 必需的输出引脚 )

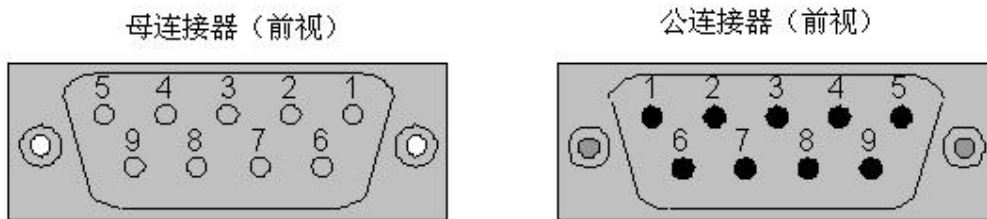


图 27: 9 引脚 D 型连接器

也可使用螺钉型连接器。

如果一个 4 线 MODBUS 设备使用 RJ45 或 9 引脚 D 型连接器, 对每种实际电路必须注意下述输出引脚。



可选 4 线-MODBUS 上的 RJ45 和 9 引脚 D 型连接器输出引脚

| RJ45 引脚 | D9 型连接器引脚 | 级别要求 | IDv 信号 | ITr 信号 | EIA/TIA-485 命名 | IDv 中的描述  |
|---------|-----------|------|--------|--------|----------------|---|
| 1       | 8         | 必需   | RXD0   | RXD0   | A'             | 接收器端子 0, Va' 电压 ( Va' > Vb' 表示二进制的 0 [ON] 状态 )  |
| 2       | 4         | 必需   | RXD1   | RXD1   | B'             | 接收器端子 1, Vb' 电压 ( Vb' > Va' 表示二进制的 1 [OFF] 状态 ) |
| 3       | 3         | 可选   | PMC    | --     | --             | 端口模式控制  |
| 4       | 5         | 必需   | TXD1   | TXD1   | B              | 发送器端子 1, Vb 电压 ( Vb > Va 表示二进制的 1 [OFF] 状态 )    |
| 5       | 9         | 必需   | TXD0   | TXD0   | A              | 发送器端子 0, Va 电压 ( Va > Vb 表示二进制的 0 [ON] 状态 )     |
| 7       | 2         | 推荐   | VP     | --     | --             | 正 5...24 V 直流电源                                 |
| 8       | 1         | 必需   | 公共地    | 公共地    | C/C'           | 信号和电源公共地  |

注：当在同一个接口上既有 2 线又有 4 线制的配置时，必须遵守 4 线配置要求。

3.5.3 可选 RS232-MODBUS 中的 RJ45 与 9 引脚 D 型连接器输出引脚

如果一个 RS232-MODBUS 设备使用 RJ45 或 9 引脚 D 型连接器，对每种实际电路必须注意下述输出引脚。

| DCE<br>下滑线引脚可以输出 |           |      | 电路         |       |         | DTE<br>下滑线引脚可以输出 |          |           |
|------------------|-----------|------|------------|-------|---------|------------------|----------|-----------|
| RJ45 引脚          | D9 型连接器引脚 | 级别要求 | 名字         | 描述    | RS232 源 | 级别要求             | RJ45 引脚  | D9 型连接器引脚 |
| <u>1</u>         | <u>2</u>  | 必需   | <u>TXD</u> | 发送数据  | DTE     | 必需               | <u>2</u> | <u>3</u>  |
| 2                | 3         | 必需   | <u>RXD</u> | 接收数据  | DCE     | 必需               | 1        | 2         |
| 3                | 7         | 可选   | <u>CTS</u> | 发送准备  | DCE     | 可选               | 6        | 8         |
| <u>6</u>         | <u>8</u>  | 可选   | <u>RTS</u> | 请求发送  | DTE     | 可选               | <u>3</u> | <u>7</u>  |
| 8                | 5         | 必需   | <u>公共地</u> | 信号公共地 | --      | 必需               | 8        | 5         |

重要注释：某些 DCE 的输出引脚会与 DTE 的同名输出引脚交叉：

在一台 DTE（如一台 PC）和一台 DCE（如一台 PLC）之间必须使用

引脚到引脚的直接连接电缆（无交叉）。

### 3.6 电缆

MODBUS 串行链路电缆必须屏蔽。在电缆两端，其屏蔽必须接到保护地上。若在这个端部使用了连接器，该连接器外壳要连在电缆屏蔽上。

AS485-MODBUS 必须使用一对平衡对线(用于 D0-D1)和第三根导线(用于公共地)。此外，在 4 线-MODBUS 系统中还必须使用第二对平衡对线（用于 RXD0-RXD1）。

若使用连接 4 对线的第 5 类电缆，请记住在用户指南中对用户的提醒：

“在 2 线-MODBUS 系统中连接一根交叉电缆可能造成损害”。

为使电缆连接中错误最少，推荐在 RS485-MODBUS 电缆连线中采用色彩标记

|          | 信号名称    | 推荐色彩 |
|----------|---------|------|
|          | D1-TXD1 | 黄    |
|          | D0-TXD0 | 褐    |
|          | 公共地     | 灰    |
| 4 线(可选项) | RXD0    | 白    |
| 4 线(可选项) | RXD1    | 兰    |

图 28: RS485-MODBUS 连线的色彩标记

注：第 5 类电缆使用其它色彩。

对 RS485-MODBUS，必须选择足够大的连线直径以达到最大长度（1000m）。AGW24 对 MODBUS 数据总是满足的。

在 RS485-MODBUS 中使用第 5 类电缆，最大长度可达 600m.

对在 RS485-系统中使用的平衡对线，建议特征阻抗高于 100 欧姆，特别是对 19200 和更高波特率。

### 3.7 可视诊断

为可视诊断，必须用 LED（发光两极管）指示通信状态和设备状态：

| 发光两极管 | 级别要求 | 说 明  | 推荐色彩 |
|-------|------|--|------|
| 通信    | 必须   | 在帧接收或发送期间置于 ON.<br>(两个 LED 表示帧接收和帧发送，或一个 LED 表示这两个意思。) | 黄    |
| 故障    | 推荐   | 置于 ON: 内部故障<br>闪烁: 其它故障（通信故障或配置故障）                     | 红    |
| 设备状态  | 可选   | 置于 ON: 设备通电  | 绿    |

## 4 安装和文件

### 4.1 安装

**产品提供者应该注意**提供给用户有关 Modbus 系统或 Modbus 设备充分有用的信息，使他们避免**错误接线**或错误使用接线附件：

- 一些其它的现场总线，例如 CANOpen，使用相同的连接器类型（D-型，RJ45...）。
- 正在进行的基于以太网的研究，相同的平衡双绞线传输电源。
- 用于 I/O 电路的其它产品使用相同的连接器类型（D-型，RJ45...）

对于这些连接器的绝大部分，**没有任何验证**（偏置程度或其他实现）。

### 4.2 用户指南

任何 Modbus 设备或接线系统组件**必须**含有但不限于下面一类或两类信息：

#### 4.2.1 任何 Modbus 产品：

下列信息应该出现在文件中：

- § 所有的实现要求。
- § 操作模式。
- § 可视诊断。
- § 可访问的寄存器和支持的功能码。
- § 安装规则。

§ 下列要求的信息也应该出现在文件中：

- ⇒ "两线制 Modbus 定义" (叙述要求的电路)；
- ⇒ "可选的四线制 Modbus 定义" (叙述要求的电路)；
- ⇒ "线路偏置" (叙述可能的需求或实现)；
- ⇒ "电缆" (交叉的电缆的特别注意)。

§ 有关设备地址的**专门指示**，以重要警告的方式书写：

*"在设定设备地址的过程中，保证不存在有相同地址的两个设备非常重要。如果发生重复，整个串行总线工作将不正常，而主节点将无法于总线上所有存在的节点通信。"*

§ 强烈建议编写 "**简易入门**" 一章，作为简易入门，同时给出一个典型的应用实例。

#### 4.2.2 实现了可选项的 Modbus 产品：

可选的参数的区别**必须**清晰详尽的描述：

- ⇒ 可选的串行传输模式；
- ⇒ 可选的奇偶校验；
- ⇒ 可选的波特率；
- ⇒ 可选的电路：电源，端口配置；
- ⇒ 可选的接口；
- ⇒ 如果最大允许的设备数目大于 32 (无中继器)。

## 5 实现等级

Modbus 串行链路上的每个设备必须遵守相同实现等级的所有的强制的要求。

下列参数用于 Modbus 串行链路设备的分级:

- 寻址
- 广播
- 传输模式
- 波特率
- 字符格式
- 电气接口参数

有两个建议的实现等级，基本和常规等级。

常规等级必须提供可配置能力。

|       | 基本                          |                             | 常规  | 默认值                                     |
|-------|-----------------------------|-----------------------------|---|---|
| 寻址    | 子节点：<br>可配置地址，<br>从 1 到 247 | 主节点：<br>可以寻址地址 1 到 247 的子节点 | 与基本相同                                     | -                                       |
| 广播    | 是                           |                             | 是   | -                                       |
| 波特率   | 9600 (19200 也同样是建议的)        |                             | 9600, 19200 + 附加的可配置的波特率                  | 19200<br>(if implemented,<br>else 9600) |
| 奇偶校验  | 偶                           |                             | 偶 + 可以配置为无和奇校验                            | 偶                                       |
| 模式    | RTU                         |                             | RTU + ASCII                               | RTU                                     |
| 电气接口  | RS485 两线制接线<br>或 RS232      |                             | RS485 两线制接线 (四线制接线<br>作为附加的选项)<br>或 RS232 | RS485 两线制接线                             |
| 连接器类型 | RJ 45 ( 建议的 )               |                             |   | -                                       |

6 附录

6.1 附录 A – 串行链路诊断计数器的管理

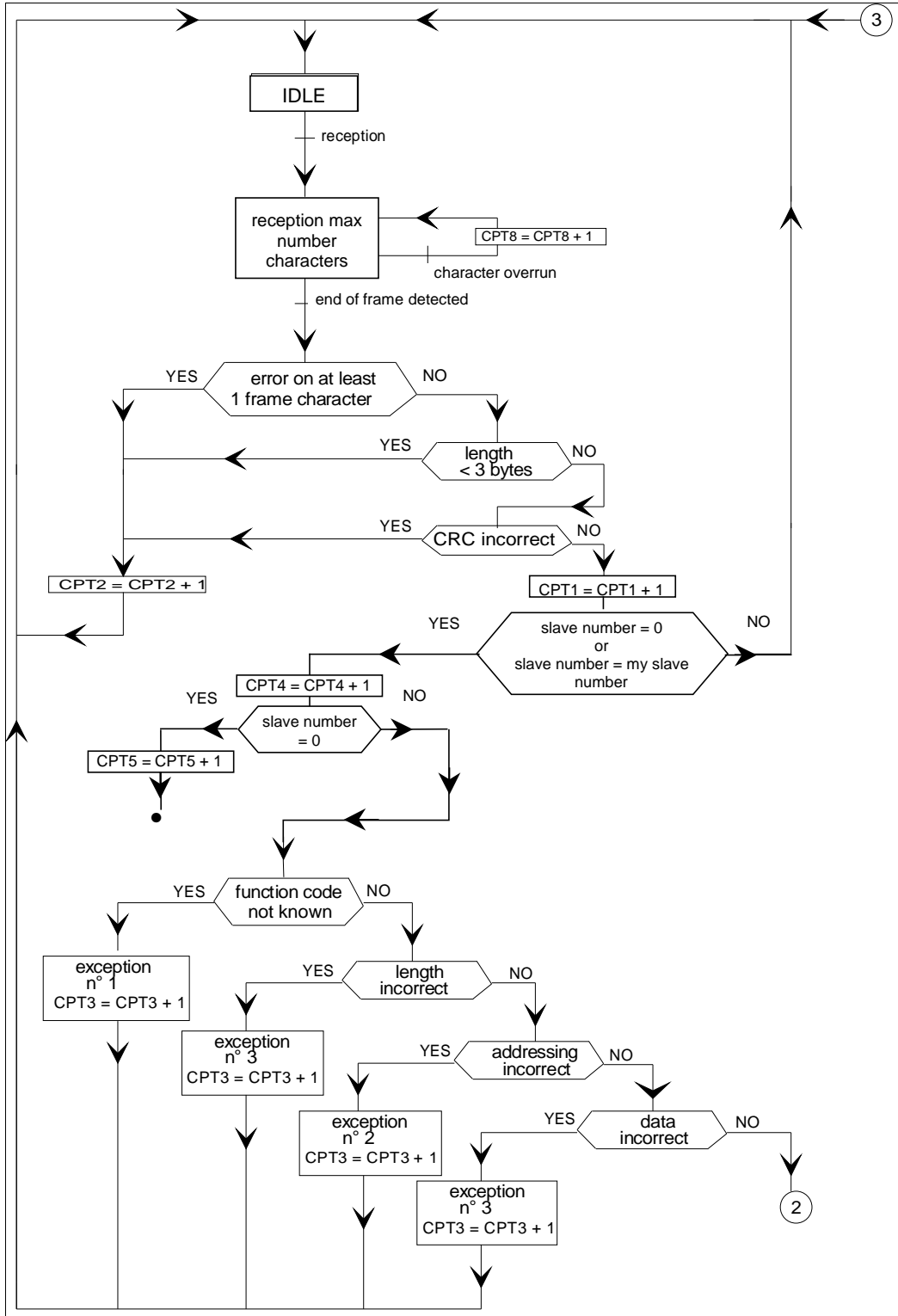
6.1.1 总体描述

Modbus 串行链路定义了一个诊断计数器列表，用于性能和错误管理。这些计数值可以通过 Modbus 应用协议和其诊断功能访问 (功能码 08)。每个计数值都可以通过一个与计数器编号绑在一起的子功能码得到。所有的计数值都可以用子功能码 0x0A 清除。诊断功能的格式在 Modbus 应用协议规范中描述。下表为串行链路设备支持的诊断和相应子功能码的列表。

| 子功能码 | 计数器编号 | 计数器名称        | 注释<br>(配合后续图表)   |
|------|-------|--------------|--|
| 十六进制 | 十进制   |              |  |
| 0x0B | 1     | 返回总线报文计数     | 自上一次重新启动、清除计数器操作或上电之后，远程节点在通信系统检测到的报文的总和。CRC 不正确地报文，不在计数范围。  |
| 0x0C | 2     | 返回总线通信错误计数   | 自上一次重新启动、清除计数器操作或上电之后，远程节点在通信系统检测到的 CRC 错误的报文的总和。检测到的字符级别的错误 (overrun, 奇偶错), 或报文长度 < 3 字节, 接收设备不能计算 CRC. 在这种情形下, 计数值也增加。 |
| 0x0D | 3     | 返回子节点异常错误计数  | 自上一次重新启动、清除计数器操作或上电之后，远程节点检测到的 Modbus 异常错。它也包含广播报文中检测到的错误，即使此种情况下没有异常报文返回。异常应答在"Modbus 应用协议规范"文件中描述并列出。                  |
| 0x0E | 4     | 返回子节点报文计数    | 自上一次重新启动、清除计数器操作或上电之后，远程节点处理的包括广播报文在内的报文数的总和。  |
| 0x0F | 5     | 返回子节点无响应计数   | 自上一次重新启动、清除计数器操作或上电之后，远程节点没有相应(既无正常响应也没有异常响应)的报文的总和。那么，这个计数器统计接收到的广播报文的数目。   |
| 0x10 | 6     | 返回子节点 NAK 计数 | 自上一次重新启动、清除计数器操作或上电之后，寻址到的远程节点返回否定确认(NAK)的异常应答报文的总和。异常应答在"Modbus 应用协议规范"文件中描述并列出。  |
| 0x11 | 7     | 返回子节点忙计数     | 自上一次重新启动、清除计数器操作或上电之后，寻址到的远程节点返回子设备忙的异常应答报文的总和。异常应答在"Modbus 应用协议规范"文件中描述并列出。   |
| 0x12 | 8     | 返回总线字符溢出计数   | 自上一次重新启动、清除计数器操作或上电之后，寻址到的远程节点由于字符溢出的情况而无法处理的报文的总和。如果字符抵达端口的速度高于它们可以被存储的速度，或由于硬件故障而丢失字符，均会导致字符溢出。                        |

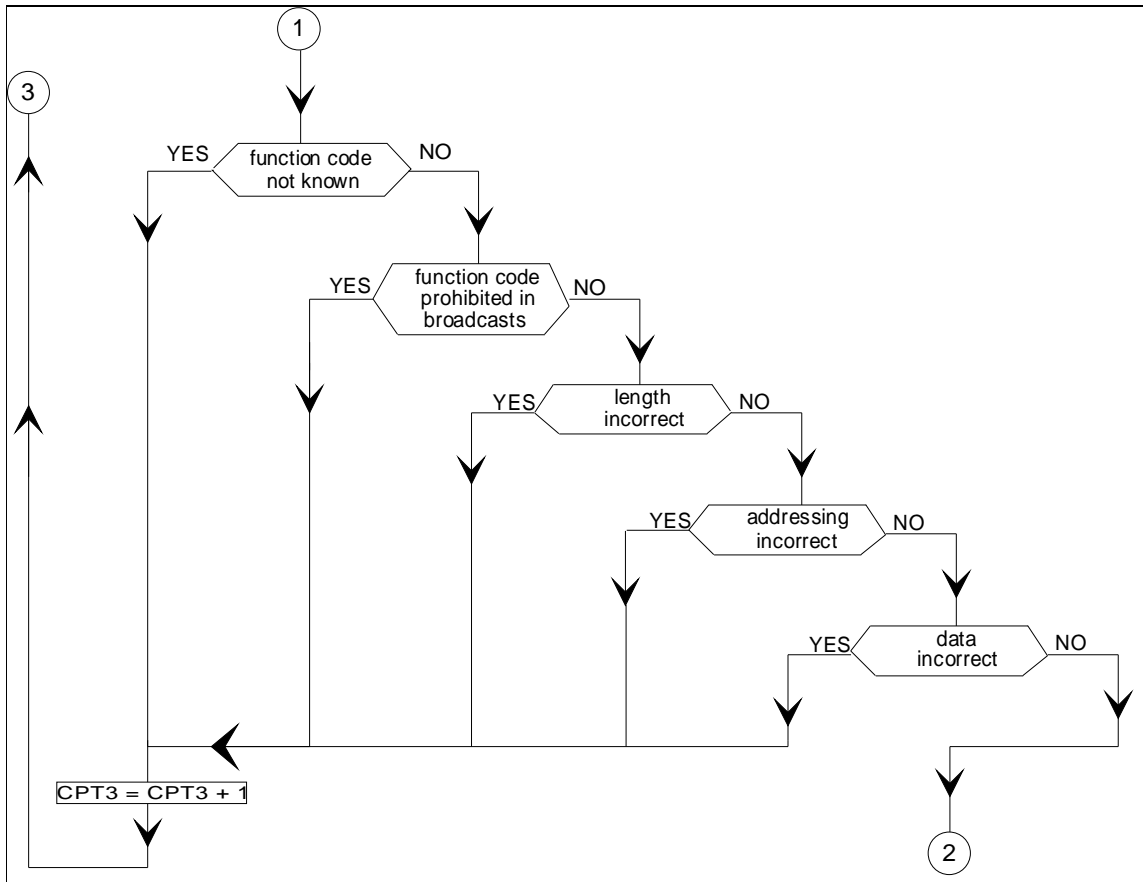
6.1.2 计数器管理流程图

下面的图表描述了每个计数器的计数值在什么时候必须增加(加1)。

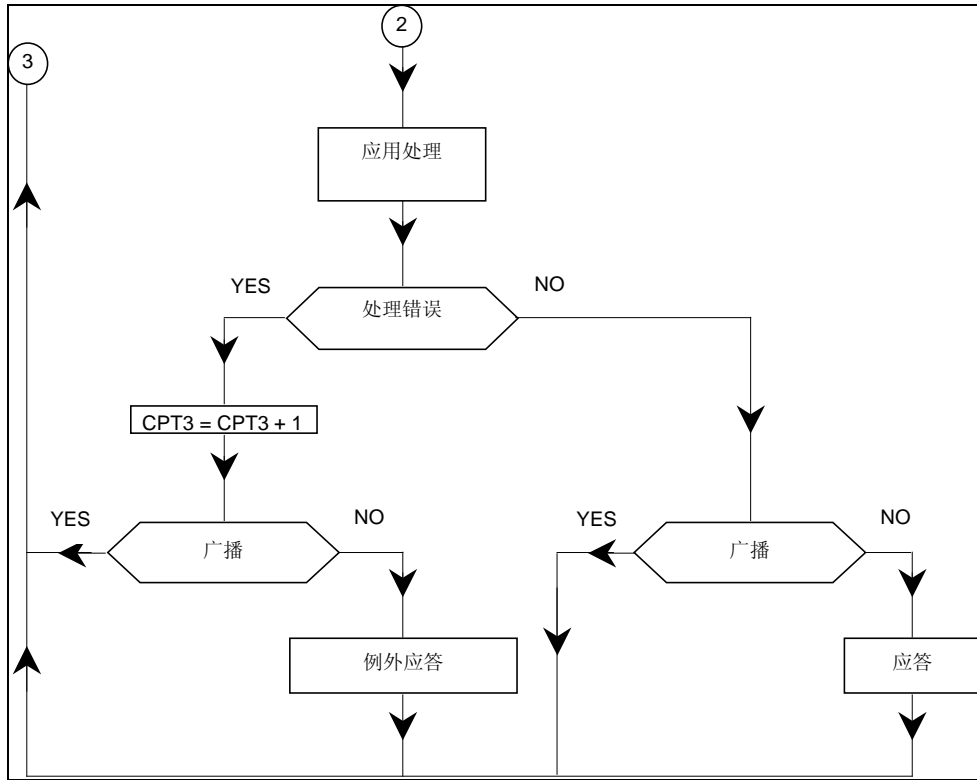


IDLE: 空闲, reception:接收, character overrun: 字符溢出, end of frame detected:检测到帧结束, error on at least 1 frame character: 至少有一个帧字符错误, slave number: 子节点地址, my slave

number:本机子节点地址 exception:异常 length incorrect: 长度不正确 addressing incorrect: 地址不正确, data incorrect: 数据不正确



function code not kown: 未知功能码, function code prohibited in broadcasts: 功能码广播方式禁用, length incorrect: 长度不正确  
addressing incorrect: 地址不正确, data incorrect: 数据不正确





## 6.2 附录 B - LRC/CRC 的生成

### 6.2.1 LRC 的生成

纵向冗余校验(LRC)为一个字节，含有 8 位二进制值。LRC 由发送设备计算，并附加 LRC 到报文。接收设备在接收文时计算 LRC，并将计算的结果与在 LRC 接收到的实际值相比较，如果两个值不相等，则结果为错。

LRC 的计算，对报文中的所有的连续 8 位字节相加，忽略任何进位，然后求出其二进制补码。LRC 为一个 8 位域，那么每个会导致值大于 255 新的相加只是简单的将域的值在零“回绕”。因为没有第 9 位，进位被自动放弃。

生成一个 LRC 的过程为:

- 1.不包括起始“冒号”和结束 CRLF 的报文中的所有字节相加到一个 8 位域，故此进位被丢弃。
- 2.从 FF (全 1)十六进制中减去域的最终值，产生 1 的补码(二进制反码)。
- 3.加 1 产生二进制补码。

#### 将 LRC 置于报文

当 8 位 LRC (2 个 ASCII 字符) 在报文中传送时，高位字符首先发送，然后是低位字符。例如，如果 LRC 值为十六进制 61 (0110 0001):



图 16: LRC 字符序列

例: 下面给出了执行生成 LRC 的 C 语言函数。

函数带有两个参数:

- unsigned char \*auchMsg; 指向含有用于生成 LRC 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.

#### LRC 生成函数

```
static unsigned char LRC(auchMsg, usDataLen) /* 函数返回 unsigned char 类型的 LRC 结果 */
/*
unsigned char *auchMsg; /* 要计算 LRC 的报文*/
unsigned short usDataLen; /* 报文的字节数 */
{
    unsigned char uchLRC = 0; /* LRC 初始化 */
    while (usDataLen--) /* 完成整个报文缓冲区 */
        uchLRC += *auchMsg++; /* 缓冲区字节相加，无进位 */
    return ((unsigned char)-((char)uchLRC)); /* 返回二进制补码 */
}
```

### 6.2.2 CRC 的生成

循环冗余校验 (CRC) 域为两个字节, 包含一个二进制 16 位值。附加在报文后面的 CRC 的值由发送设备计算。接收设备在接收报文时重新计算 CRC 的值, 并将计算结果于实际接收到的 CRC 值相比较。如果两个值不相等, 则为错误。

CRC 的计算, 开始对一个 16 位寄存器预装全 1。然后将报文中的连续的 8 位子节对其进行后续的计算。只有字符中的 8 个数据位参与生成 CRC 的运算, 起始位, 停止位和校验位不参与 CRC 计算。

CRC 的生成过程中, 每个 8-位字符与寄存器中的值异或。然后结果向最低有效位 (LSB) 方向移动(Shift) 1 位, 而最高有效位 (MSB) 位置充零。然后提取并检查 LSB: 如果 LSB 为 1, 则寄存器中的值与一个固定的预置值异或; 如果 LSB 为 0, 则不进行异或操作。

这个过程将重复直到执行完 8 次移位。完成最后一次 (第 8 次) 移位及相关操作后, 下一个 8 位字节与寄存器的当前值异或, 然后又同上面描述过的一样重复 8 次。当所有报文中子节都运算之后得到的寄存器中的最终值, 就是 CRC。

生成 CRC 的过程为:

1. 将一个 16 位寄存器装入十六进制 FFFF (全 1)。将之称作 CRC 寄存器。
2. 将报文的第一个 8 位字节与 16 位 CRC 寄存器的低字节异或, 结果置于 CRC 寄存器。
3. 将 CRC 寄存器右移 1 位 (向 LSB 方向), MSB 充零。提取并检测 LSB。
4. (如果 LSB 为 0): 重复步骤 3 (另一次移位)。  
(如果 LSB 为 1): 对 CRC 寄存器异或多项式值 0xA001 (1010 0000 0000 0001)。
5. 重复步骤 3 和 4, 直到完成 8 次移位。当做完此操作后, 将完成对 8 位字节的完整操作。
6. 对报文中的下一个字节重复步骤 2 到 5, 继续此操作直至所有报文被处理完毕。
7. CRC 寄存器中的最终内容为 CRC 值。
8. 当放置 CRC 值于报文时, 如下面描述的那样, 高低字节必须交换。

#### 将 CRC 放置于报文

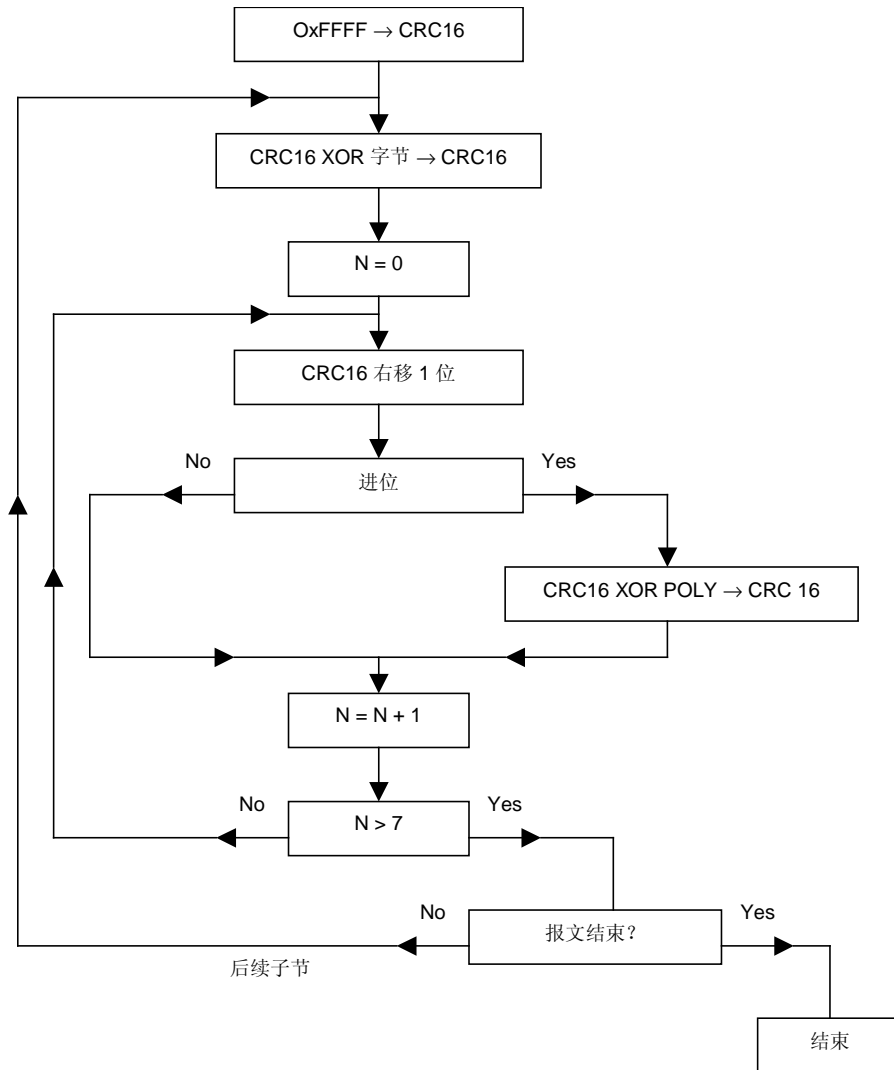
当 16 位 CRC (2 个 8 位字节) 在报文中传送时, 低位字节首先发送, 然后是高位字节。

例如, 如果 CRC 值为十六进制 1241 (0001 0010 0100 0001):



图 17: CRC 字节序列

### CRC 16 计算算法



XOR = 异或

N = 字节的信息位

POLY = CRC 16 多项式计算 = 1010 0000 0000 0001

(生成多项式 =  $1 + x_2 + x_{15} + x_{16}$ )

在 CRC 16 中， 发送的第一个字节为低字节。

CRC 计算示例 (帧 02 07)

CRC 寄存器初始化  
XOR 第一个字符

标志 1, XOR 多项式

标志 1, XOR 多项式

XOR 第二个字符

|  |      |       |      |      |        |
|--|------|-------|------|------|--------|
|  |      | 1111  | 1111 | 1111 | 1111   |
|  |      | 0000  | 0000 | 0000 | 0000   |
|  |      | <hr/> |      |      |        |
|  | 移位 1 | 1111  | 1111 | 1111 | 1101   |
|  |      | 0111  | 1111 | 1111 | 1110 1 |
|  |      | 1010  | 0000 | 0000 | 0001   |
|  |      | <hr/> |      |      |        |
|  | 移位 2 | 1101  | 1111 | 1111 | 1111   |
|  |      | 0110  | 1111 | 1111 | 1111 1 |
|  |      | 1010  | 0000 | 0000 | 0001   |
|  |      | <hr/> |      |      |        |
|  | 移位 3 | 1100  | 1111 | 1111 | 1110   |
|  |      | 0110  | 0111 | 1111 | 1110 0 |
|  | 移位 4 | 0011  | 0011 | 1111 | 1111 1 |
|  |      | 1010  | 0000 | 0000 | 0001   |
|  |      | <hr/> |      |      |        |
|  | 移位 5 | 1001  | 0011 | 1111 | 1110   |
|  |      | 0100  | 1001 | 1111 | 1111 0 |
|  | 移位 6 | 0010  | 0100 | 1111 | 1111 1 |
|  |      | 1010  | 0000 | 0000 | 0001   |
|  |      | <hr/> |      |      |        |
|  | 移位 7 | 1000  | 0100 | 1111 | 1110   |
|  |      | 0100  | 0010 | 0111 | 1111 0 |
|  | 移位 8 | 0010  | 0001 | 0011 | 1111 0 |
|  |      | 1010  | 0000 | 0000 | 0001   |
|  |      | <hr/> |      |      |        |
|  |      | 1000  | 0001 | 0011 | 1110   |
|  |      | 0000  | 0000 | 0000 | 0111   |
|  |      | <hr/> |      |      |        |
|  | 移位 1 | 1000  | 0001 | 0011 | 1001   |
|  |      | 0100  | 0000 | 1001 | 1100 1 |
|  |      | 1010  | 0000 | 0000 | 0001   |
|  |      | <hr/> |      |      |        |
|  | 移位 2 | 1110  | 0000 | 1001 | 1101   |
|  |      | 0111  | 0000 | 0100 | 1110 1 |
|  |      | 1010  | 0000 | 0000 | 0001   |
|  |      | <hr/> |      |      |        |
|  | 移位 3 | 1101  | 0000 | 0100 | 1111   |
|  |      | 0110  | 1000 | 0010 | 0111 1 |
|  |      | 1010  | 0000 | 0000 | 0001   |
|  |      | <hr/> |      |      |        |
|  | 移位 4 | 1100  | 1000 | 0010 | 0110   |
|  |      | 0110  | 0100 | 0001 | 0011 0 |
|  | 移位 5 | 0011  | 0010 | 0000 | 1001 1 |
|  |      | 1010  | 0000 | 0000 | 0001   |
|  |      | <hr/> |      |      |        |
|  | 移位 6 | 1001  | 0010 | 0000 | 1000   |
|  |      | 0100  | 1001 | 0000 | 0100 0 |
|  | 移位 7 | 0010  | 0100 | 1000 | 0010 0 |
|  | 移位 8 | 0001  | 0010 | 0100 | 0001 0 |



帧的 CRC 16 则为: 4112

例

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, 另一个数组含有低位字节的值。

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。

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) ;
}

```

## 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
};

```

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
};
```

6.3 附录 E – 规范性引用文件

- |                                  |   |
|----------------------------------|---|
| <b>ANSI/ TIA/ EIA-232-F-1997</b> | <b>Interface Between Data Terminal Equipment and Data Circuit-Terminating Equipment Employing Serial Binary Data Interchange.</b> |
| <b>ANSI/ TIA/ EIA-485-A-1998</b> | <b>Electrical Characteristics of Generators and Receivers for Use in Balanced Digital Multipoint Systems.</b>                     |
| <b>Modbus .org Modbus</b>        | <b>Application Protocol Specification</b>   |



## 第三部分：Modbus 协议在 TCP/IP 上的实现指南

## 1 引言

### 1.1 范围

这个文件的范围是介绍 TCP/IP 上的 MODBUS 报文传输服务，提供参考信息以帮助软件开发者使用这种服务。这个文中不包括 MODBUS 功能码的编码内容，这些信息请参阅 MODBUS 协议规范[2]。

这个文件准确而全面地描述了 MODBUS 报文传输服务的实现。其目的是便于在那些使用 MODBUS 报文传输服务的设备之间进行可互操作。

这个文件主要由三部分组成：

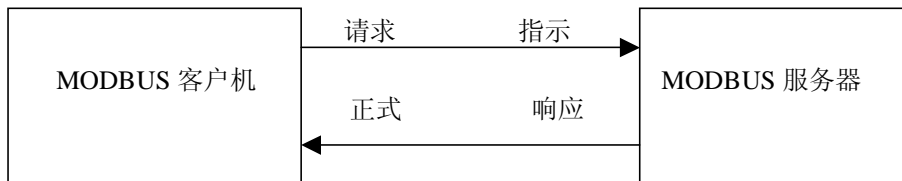
- l 在 TCP/IP 上的 MODBUS 协议概述
- l MODBUS 客户机、服务器和网关工具的功能描述
- l 针对一个 MODBUS 实现实例的目标模型建议的实现准则。

### 1.2 客户机/服务器模型

MODBUS 报文传输服务提供设备之间的客户机/服务器通信，这些设备联接在一个 Ethernet（以太网） TCP/IP 网络上。

这个客户机/服务器模式是基于 4 种类型报文：

- l MODBUS 请求
- l MODBUS 证实
- l MODBUS 指示
- l MODBUS 响应



MODBUS 请求是客户机在网络上发送用来启动事务处理的报文

MODBUS 指示是服务端接收的请求报文

MODBUS 响应是服务器发送的响应信息

MODBUS 证实是在客户端接收的响应信息

MODBUS 报文传输服务（客户机/服务器模型）用于实时信息交换：

- l 在两个设备应用程序之间
- l 在设备应用和其它设备之间
- l 在 HMI/SCADA 应用程序和设备之间
- l 在一个 PC 和一个提供在线服务的设备程序之间

### 1.3 规范性引用文件

本章给出了在这个文件之前喜欢阅读的文件列表：

- [2] MODBUS 协议规范
- [4] RFC1122

## 2 缩略语

|      |           |
|------|-----------|
| ADU  | 应用数据单元    |
| IETF | 因特网工程工作组  |
| IP   | 互连网协议     |
| MAC  | 介质访问控制    |
| MB   | MODBUS    |
| MBAP | MODBUS 协议 |
| PDU  | 协议数据单元    |
| PLC  | 可编程序逻辑控制器 |
| TCP  | 传输控制协议    |
| BSD  | 伯克利软件分配   |
| MSL  | 最大段寿命     |

### 3 背景概要

#### 3.1 协议描述

##### 3.1.1 总体通信结构

MODBUS TCP/IP 的通信系统可以包括不同类型的设备：

- I 连接至 TCP/IP 网络的 MODBUS TCP/IP 客户机和服务器设备
- I 互连设备，例如：在 TCP/IP 网络和串行链路子网之间互连的网桥、路由器或网关，联接，该子网允许将 MODBUS 串行链路客户机和服务器终端设备连接起来。

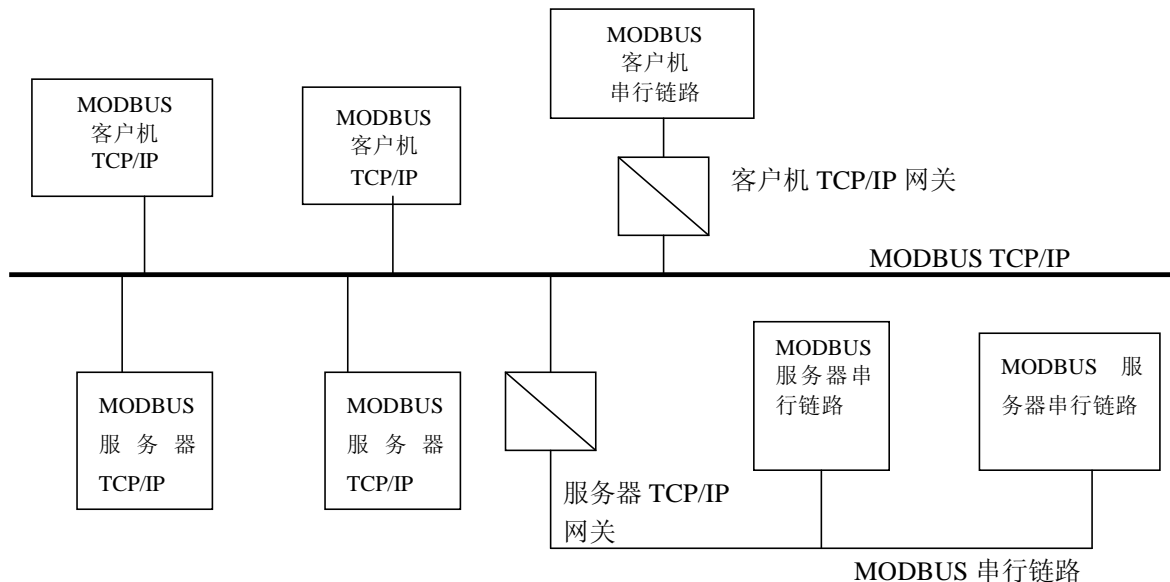


图 1：MODBUS TCP/IP 通信结构

MODBUS 协议定义了一个与基础通信层无关的**简单协议数据单元 (PDU)**。特定总线或网络上的 MODBUS 协议映射能够在**应用数据单元 (ADU)** 上引入一些附加域。

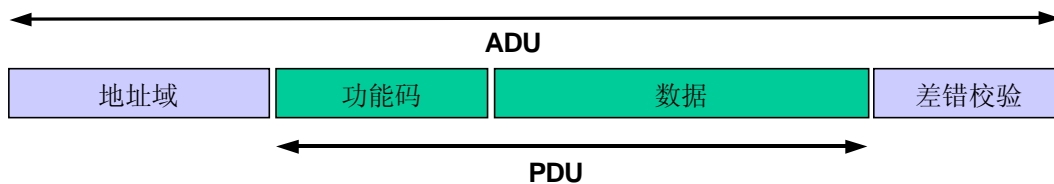


图 2：通用 MODBUS 帧

启动 MODBUS 事务处理的客户机建立 MODBUS 应用数据单元。这个功能码向服务器指示执行哪种操作。

### 3.1.2 TCP/IP 上的 MODBUS 应用数据单元

这节描述了 MODBUS TCP/IP 网络中进行的 MODBUS 请求或响应的封装。

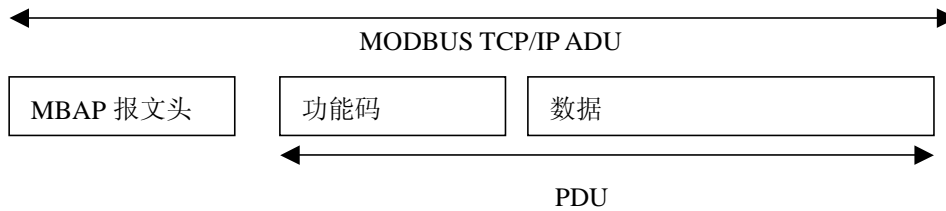


图 3: TCP/IP 上的 MODBUS 的请求/响应

在 TCP/IP 上使用一种专用报文头识别 MODBUS 应用数据单元。将这种报文头称为 MBAP 报文头 (MODBUS 协议报文头)。

这种报文头提供一些与串行链路上使用的 MODBUS RTU 应用数据单元比较的差别:

- 1 用 MBAP 报文头中的单个字节单元标识符取代 MODBUS 串行链路上通常使用的 MODBUS 从地址域。这个单元标识符用于设备的通信, 这些设备使用单个 IP 地址支持多个独立 MODBUS 终端单元, 例如: 网桥、路由器和网关。
- 1 用接收者可以验证完成报文的方式设计所有 MODBUS 请求和响应。对于 MODBUS PDU 有固定长度的功能码来说, 仅功能码就足够了。对于在请求或响应中携带一个可变数据的功能码来说, 数据域包括字节数。
- 1 当在 TCP 上携带 MODBUS 时, 即使将报文分成多个信息包来传输, 办事在 MBAP 报文头上携带附加长度信息, 以便接收者能识别报文边界。显式和隐式长度规则的存在以及 CRC-32 差错校验码的使用 (在以太网上) 将对请求或响应报文产生极小的未检出干扰。

### 3.1.3 MBAP 报文头描述

MBAP 报文头包括下列域:

| 域      | 长度    | 描述                    | 客户机        | 服务器            |
|--------|-------|-----------------------|------------|----------------|
| 事务元标识符 | 2 个字节 | MODBUS 请求/响应事务处理的识别码  | 客户机启动      | 服务器从接收的请求中重新复制 |
| 协议标识符  | 2 个字节 | 0=MODBUS 协议           | 客户机启动      | 服务器从接收的请求中重新复制 |
| 长度     | 2 个字节 | 以下字节的数量               | 客户机启动 (请求) | 服务器 (响应) 启动    |
| 单元标识符  | 1 个字节 | 串行链路或其它总线上连接的远程从站的识别码 | 客户机启动      | 服务器从接收的请求中重新复制 |

报文头为 7 个字节长:

事务处理标识符: 用于事务处理配对。在响应中, MODBUS 服务器复制请求的事务处理标识符。

协议标识符: 用于系统内的多路复用。通过值 0 识别 MODBUS 协议。

长度: 长度域是下一个域的字节数, 包括单元标识符和数据域。

单元标识符: 为了系统内路由, 使用这个域。专门用于通过以太网 TCP-IP 网络和 MODBUS 串行链路之间的网关对 MODBUS 或 MODBUS+ 串行链路从站的通信。MODBUS 客户机在请求中设置这个域, 在响应中服务器必须利用相同的值返回这个域。

在注册的 502 端口上利用 TCP 发送所有 MODBUS/TCP ADU。

注：用 Big-endian 编码不同域。

### 3.2 MODBUS 功能码描述

在 MODBUS 协议规范[2]中详细说明了 MODBUS 应用层协议上使用的标准功能码。

#### 4 功能描述

这里提供的 MODBUS 组件结构是一个既包含 MODBUS 客户机又包含 MODBUS 服务器组件的通用模型，适用于任何设备。

有些设备可能仅提供服务器或客户机组件。

本章的第一部分，给出一个有关 MODBUS 报文传输服务组件结构的简要概述，然后，给出结构模型内部每一个组件的描述。

##### 4.1 MODBUS 组件结构模型

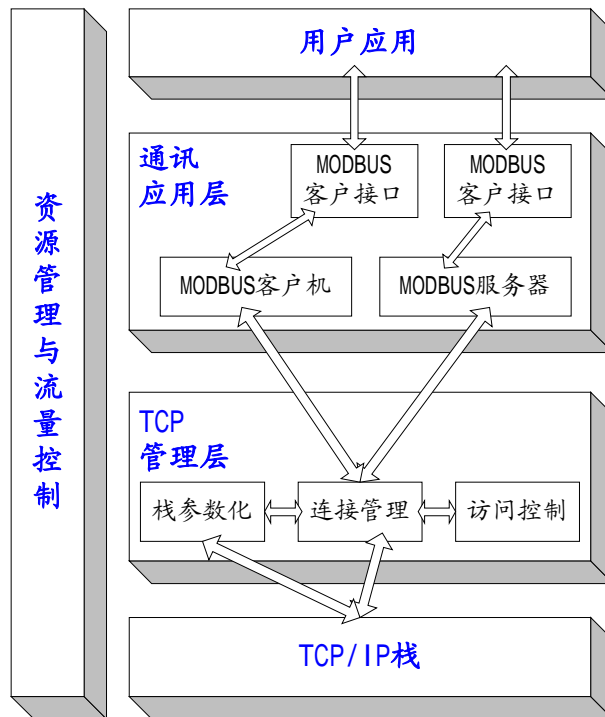


图 4 MODBUS 报文传输服务概念结构

##### I 通信应用层

一个 MODBUS 设备可以提供一个客户机和/或服务器 MODBUS 接口。

可提供一个 MODBUS 后台接口，允许间接的访问用户应用对象。

此接口由四部分组成：离散量输入、离散量输出（线圈）、寄存器输入和寄存器输出。此接口与用户应用数据之间的映射必须加以定义（本地问题）。

| 基本数据表 | 对象类型  | 属性  | 说明             |
|-------|-------|-----|----------------|
| 离散量输入 | 1 位   | 只读  | 此类数据可来自 I/O 系统 |
| 线圈    | 1 位   | 读-写 | 此类数据可被应用程序修改   |
| 寄存器输入 | 16 位字 | 只读  | 此类数据可来自 I/O 系统 |

|       |       |    |              |
|-------|-------|----|--------------|
| 寄存器输出 | 16 位字 | 只写 | 此类数据可被应用程序修改 |
|-------|-------|----|--------------|

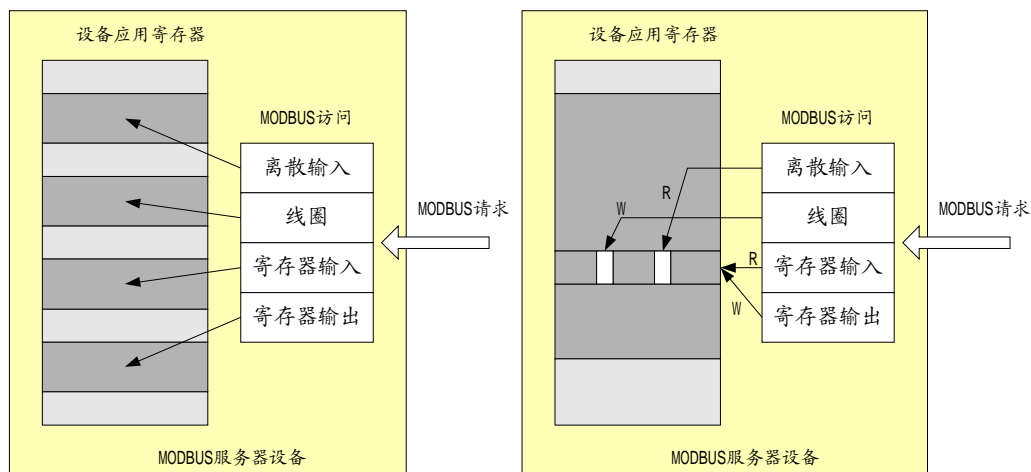


图 5、分离数据块的 MODBUS 数据模型

6、单一数据块的 MODBUS 数据模型

### Ø MODBUS 客户机

MODBUS 客户机允许用户应用清晰地控制与远端设备的信息交换。MODBUS 客户机根据用户应用向 MODBUS 客户接口发送的需求中所包含的参数生成一个 MODBUS 请求。

MODBUS 客户机调用一个 MODBUS 的事务处理，事务处理管理包括 MODBUS 证实的等待和处理。

### Ø MODBUS 客户机接口

MODBUS 客户机接口提供一个接口，使得用户应用能够生成对包括访问 MODBUS 应用对象在内的各类 MODBUS 服务的请求。尽管在实现模型中以实例说明，但是 MODBUS 客户机接口（API）在这里不进行描述。

### Ø MODBUS 服务器

在收到一个 MODBUS 请求以后，模块激活一个本地操作进行读、写、或完成其他操作。这些操作的处理对应用程序开发人员来说都是透明的。MODBUS 服务器的主要功能是等待来自 TCP502 口的 MODBUS 请求，处理这一请求，然后生成一个 MODBUS 应答，应答取决于设备状况（**场境**）。

### Ø MODBUS 后台接口

MODBUS 后台接口是一个从 MODBUS 服务器到定义应用对象的用户应用之间的接口。

## I TCP 管理层

报文传输服务的主要功能之一是管理通信的建立和结束，管理建立在 TCP 连接上的数据流。

### Ø 连接管理

在客户机和服务器的 MODBUS 模块之间的通信需要调用 TCP 连接管理模块。它负责全面管理报文传输 TCP 连接。

连接管理中存在两种可能：用户应用自身管理 TCP 连接，或全部由这个模块进行连接管理，而对用户应用透明。后一种方案灵活性较差。



TCP 502 口的侦听是为 MODBUS 通信保留的。在缺省状态下，强制侦听这个口。然而，有些市场或应用可能需要其他口作为 TCP 上 MODBUS 的通信之用。当需要与非施奈德（Schneider）产品进行互操作时，就属于这种情况，例如：在建筑控制中。为此，强烈建议：客户机和服务器均应向用户提供对 TCP 口号上的 MODBUS 参数进行配置的可能性。重要的是：即使在某一个特定的应用中为 MODBUS 服务配置了其他 TCP 服务器口，除一些特定应用口外，TCP 服务器 502 口必须仍然是可用的。

#### Ø 访问控制模块

在某些至关重要的场合，必须禁止不必要的主机对设备内部数据的访问。这既是为什么需要安全模式，也是在需要时实现安全处理的原因。

#### I TCP/IP 栈层

TCP/IP 的栈可以进行参数配置，以便于使得数据流控制、地址管理和连接管理适应于特定的产品或系统的不同的约束。一般说来，BSD 套接字接口就用来管理 TCP 连接。

#### I 资源管理和数据流控制

为了平衡 MODBUS 客户机与服务器之间进出报文传输的数据流，在 MODBUS 报文传输栈的所有各层均设置了数据流控制机制。资源管理和数据流控制模块首先是基于 TCP 内部数据流控制，附加数据链路层的某些数据流控制，以及用户应用层的数据流控制。

## 4.2 TCP 连接管理

### 4.2.1 连接管理模块

#### 4.2.1.1 总体描述

MODBUS 通信需要建立客户机与服务器之间的 TCP 连接。

连接的建立可以由用户应用模块直接实现，也可以由 TCP 连接管理模块自动完成。

在第一种情况下，用户应用模块必须提供应用程序接口，以便完全管理连接。这种方式为应用开发人员提供了灵活性，但需要 TCP/IP 机制方面的专长。

在第二种方案中，TCP 连接管理完全不出现，用户应用仅需要发送和接受 MODBUS 报文。TCP 连接管理模块负责在需要时建立新的 TCP 连接。

TCP 客户机和服务器连接数量的定义不属于本文件的范畴（在本文中采用 n）。根据设备能力，TCP 连接的数量会不同。

实现规则：

1) 如果没有明确的用户需求，建议采用自动的 TCP 连接管理

2) 建议：打开并保持与远端设备的连接，而不要在每次 MODBUS/TCP 事务处理时打开和关闭连接。

*注：然而，MODBUS 客户必须能够接收来自服务器的关闭请求，并关闭连接。当需要时，连接可以被重新打开。*

3) 建议：每一个 MODBUS 客户至少要打开与远端 MODBUS 服务器的 TCP 连接（同一 IP 地址）。一个

应用建立一个连接是好的选择。

4)几个 MODBUS 事务处理可以在同一个 TCP 连接上被同时激活

注：如果以此方式，MODBUS 事务处理标识必须被用来唯一地识别请求与响应的匹配。

5)在两个远端 MODBUS 设备（一个客户机和一个服务器）之间双向通信的情况下，有必要为客户机数据流和服务器数据流分别建立连接。

6)一个 TCP 帧只能传送一个 MODBUS ADU。建议：不要在同一个 TCP PDU 中发送多个请求或应答。

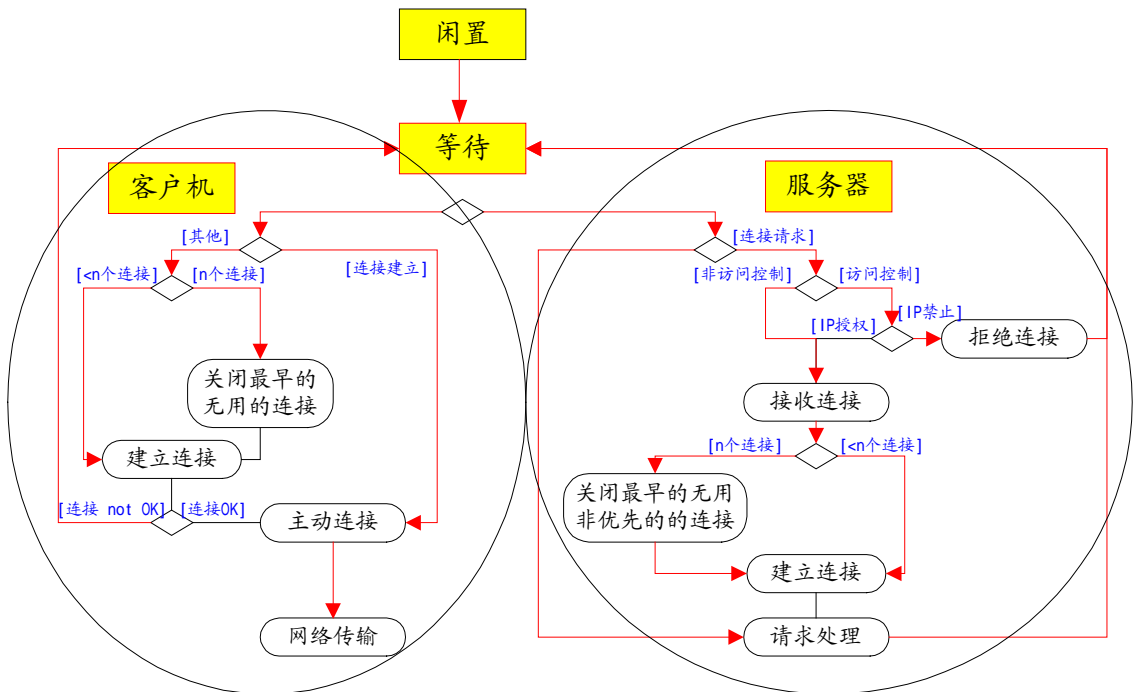


图 7：TCP 连接管理操作图

1. 显式 TCP 连接管理

用户应用模块负责管理所有的 TCP 连接：主动的和被动的连接建立、连接结束……。对客户机与服务器间所有的连接进行这种管理。BSD 套接字接口用在用户应用模块中来管理 TCP 连接。这种方案提供了完全的灵活性，但也意味着应用开发人员要具备充分的有关 TCP 的知识。

考虑到设备的能力和 demand，必须进行配置客户机与服务器间连接数的限制。

2. 自动 TCP 连接管理

TCP 连接管理对用户应用模块是完全透明的。连接管理模块可以接受足够数量的客户机/服务器连接。否则，在超过所授权数量的连接时必须有一种实现机制。在这种情况下，我们建议：关闭最早建立的不使用的连接。

在收到第一个来自远端客户机或本地用户应用的数据包后，就建立了与远端对象的连接。如果一个网络进行终止或本地设备决定终止，此连接将被关闭。在接收连接请求时，访问控制选项可用来禁止未授权客户访问设备的可能性。

TCP 连接管理模块采用栈接口（通常 BSD 套接字接口）来与 TCP/IP 栈进行通信。

为了保持系统需求与服务器资源之间的兼容，TCP 管理将保持两个连接库。

- § 第一个库（优先连接库）由那些从不被本地主动关闭的连接组成。必须提供一个配置来建立这个库。实现的原理是将这个库的每一个可能的连接与一个特定的 IP 地址联系起来。具有这个 IP 地址的设备被称为“标记的”。任何一个被“标记的”设备的新的连接请求必须被接收，并从优先连接库中取出。还有必要设置允许每个远端设备最多建立连接的数量，以避免同一设备使用优先连接库中所有的连接。
- § 第二个库（非优先连接库）包括了非标记设备的连接。这里采用的规则是：当有来自非标记设备的新的连接请求，以及库中没有连接可用时，关闭早些时候建立的连接。

一个配置可作为选项提供来分配每个库中可用连接的数量。然而（非强制性的），如果需要，设计人员可在设计期间设定连接的数量。

#### 4.2.1.2 连接管理描述

##### § 连接建立

MODBUS 报文传输服务必须在 502 口上提供一个侦听套接字，允许接收新的连接和与其他设备交换数据。

当报文传输服务需要与远端服务器交换数据时，它必须与远端 502 口建立一个新的客户连接，以便与远距离交换数据。本地口必须高于 1024，并且每个客户连接各不相同。

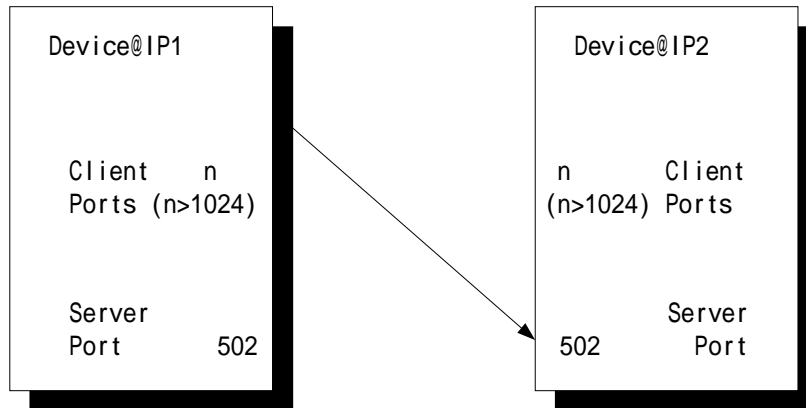


图 8: MODBUS TCP/IP 连接建立

如果客户机与服务器的连接数量大于授权的数量，则最早建立的无用的连接被关闭。激活访问控制机制检查远端客户机的 IP 地址是否是经过授权的。如果未经授权，将拒绝新的连接。

##### § MODBUS 数据变换

基于已经打开的正确的 TCP 连接发送 MODBUS 请求。远端设备的 IP 地址用于寻找所建的 TCP 连接。在与同一个远端设备建立多个连接时，必须选择其中一个连接用于发送 MODBUS 报文，可以采取不同的选择策略，例如：最早的连接、第一个连接。在 MODBUS 通信的全过程中，连接必须始终保持打开。如同下列各章所描述的一样，一个客户机可以向一个服务器启动多个事务处理，而不必等待前序事物处理结束。

## § 连接关闭

当客户机与服务器间的 MODBUS 通信结束时，客户机必须关闭用于通信的连接。

### 4.2.2 操作模式对 TCP 连接的影响

某些操作模式（两操作端点之间通信断开、一个端点的故障和重新启动、……）会对 TCP 连接产生影响。一个连接可被视为在这一侧关闭或异常终止而没有另一侧的确认，称这种连接为“半打开”的连接。

本章描述每种主要操作模式的特性。假设在连接的两端采用了“保持连接”TCP 机制（参见第 4.3.2 节）。

#### 4.2.2.1 两操作端之间通信断开

通信断开的原因可以是服务器侧以太网连接电缆断开。预期的特性是：

- 如果在连接上没有正在发送数据包：

如果通信断开持续的时间短于“保持连接”计时器的值，将察觉不到通信断开。如果通信断开时间超过“保持连接”计时器的值，将一个错误返回到 TCP 连接层，由其复位连接。

- 如果在连接断开的前后发送一些数据包：

TCP 重新传输算法（Jacobson 算法、Karn 算法以及指数补偿算法，参见第 4.3.2 节）被激活。这可能导致在“保持连接”计时器终止之前 TCP 栈连接层复位。

#### 4.2.2.2 服务器端的故障和重新启动

在服务器故障和重新启动以后，客户端处于“半打开”连接状态。预期的特性是：

- 如果在半打开的连接上没有发送数据包：

只要“保持连接”计时器还在计时中，从客户端看，连接是半打开的。之后，将返回一个错误到 TCP 管理层，由其复位连接。

- 如果在半打开的连接上发送一些数据包：

服务器在不存在的连接上接收数据。TCP 层的栈发送一个复位指令来关闭客户端的半打开的连接。

#### 4.2.2.3 客户机端的故障和重新启动

在客户机故障和重新启动以后，服务器侧处于“半打开”连接状态。预期的状态是：

- 如果在半打开的连接上没有发送数据包：

只要“保持连接”计时器还在计时中，从服务器端看，这种连接是半打开的。之后，将返回一个错误到 TCP 管理层，由其复位连接。

- 如果在“保持连接”计时器完成计时前，客户机打开一个新的连接：

必须分两种情况研究：

- § 所打开的连接与服务器侧半打开的连接具有相同的特性（相同的源和目的口、相同的源和目的 IP 地址），所以，在连接建立超时后（伯克利实现的多数情况下为 75ms），TCP 栈层将不能打开连接。为了避免较长超时时间内不能进行通信，建议：在客户机端重新启动后，确保使用与原有连接不同的源口号建立连接。
- § 所打开的连接与服务器侧半打开的连接具有不同的特性（不同的源口和相同的目的口、相同的源和目的 IP 地址），所以，在 TCP 栈层上打开连接，并向服务器侧的 TCP 管理层发送信号。

如果服务器侧 TCP 管理层仅支持一个远端客户机 IP 地址的连接，那么可以关闭原来的半打开的连接，使用新的连接。

如果服务器侧 TCP 管理层支持多个远端客户机 IP 地址的连接，那么新的连接保持打开状态，原来的连接也保持半打开状态，直到“保持连接”计时器计时结束，此时，将返回一个错误到 TCP 管理层。之后，TCP 管理层将能够复位原有的连接。

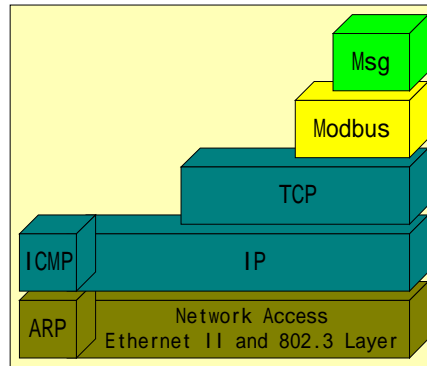
#### 4.2.3 访问控制模块

这个模块的目的是检查每一个新的连接，对照一个合法授权的远程 IP 地址列表，它可以授权或禁止一个远端客户机的 TCP 连接。

在至关重要的场合，应用开发人员需要选择访问控制模块来保证网络的访问。在这种情况下，需要对每个远端 IP 授权或禁止访问。用户需提供提供一个 IP 地址的列表，并特别注明每个 IP 地址是否合法授权。在缺省情况下，在安全模式中，用户未配置的 IP 地址均被禁止。所以，借助于访问控制模式，关闭来自未知的 IP 地址的访问连接。

#### 4.3 TCP/IP 栈的使用

TCP/IP 栈提供了一个接口，用来管理连接、发送和接收数据，还可以进行参数配置，以使得栈的特性适应于设备或系统的限制。



本章的目的是给出有关栈接口的综述，以及一些与栈的参数配置有关的信息。总揽综述主要是 MODBUS 报文传输所使用的一些特性。

对于更多的信息，建议阅读 RFC 1122，这个 RFC 1122 为厂商和开发商提供了互联网通信软件的指南。RFC 1122 详述了一个连接到互联网的主机必须采用的标准协议，以及一组明确的需求和选项。

栈接口一般是基于本文件中描述的 BSD（伯克利软件分配代码）接口。

#### 4.3.1 BSD 套接字接口中的应用

*注：有些 TCP/IP 栈从性能考虑提出其他类型的接口。MODBUS 客户机或服务器可以使用这些特定的接口，但是在本文件中对这种使用不做描述。*

一个套接字是一个通信端点，它是通信中的基本构成块。通过套接字发送和接收数据可以执行一个 MODBUS 通信。TCP/IP 库仅提供了使用 TCP 和提供基于连接的通信服务的流套接字。

socket()函数用来创建套接字。返回的一个套接字号被创建者用来访问套接字。套接字创建时没有地址（IP 地址和口号）。直到一个口被绑定到该套接字时，方可接收数据。

bind()函数用来绑定一个口号到套接字。bind()函数在套接字与所指定的口号之间建立一个连接。

为了初始化一个连接，客户端必须发送 connect()函数来指定套接字号、远端 IP 地址和远端侦听口号（主动连接建立）。

为了完成连接，服务器端必须发送 accept()函数指定以前在 listen()调用中所指定的套接字号（被动连接建立）。一个新的套接字被创建，并具有与最初相同的特性。这个新的套接字连接到客户机的套接字，而将套接字号返回到服务器端。于是，释放初始套接字，以便为其他欲与服务器连接的客户机使用。

在 TCP 连接建立以后，数据即可被传递。将 send()和 recv()函数专门地设计成与已经连接的套接字一道使用。

setsockopt()函数允许套接字的创建者用套接字建立若干选项。这些选项描述了套接字的操作特征。在第 4.3.2 节中给出这些选项的描述。

select()函数允许编程人员测试所有套接字上的事件。

shutdown()函数允许套接字的使用者来终止 send()和/或 recv()。

一旦不再需要套接字，可以使用 close()函数来放弃套接字的描述信息。

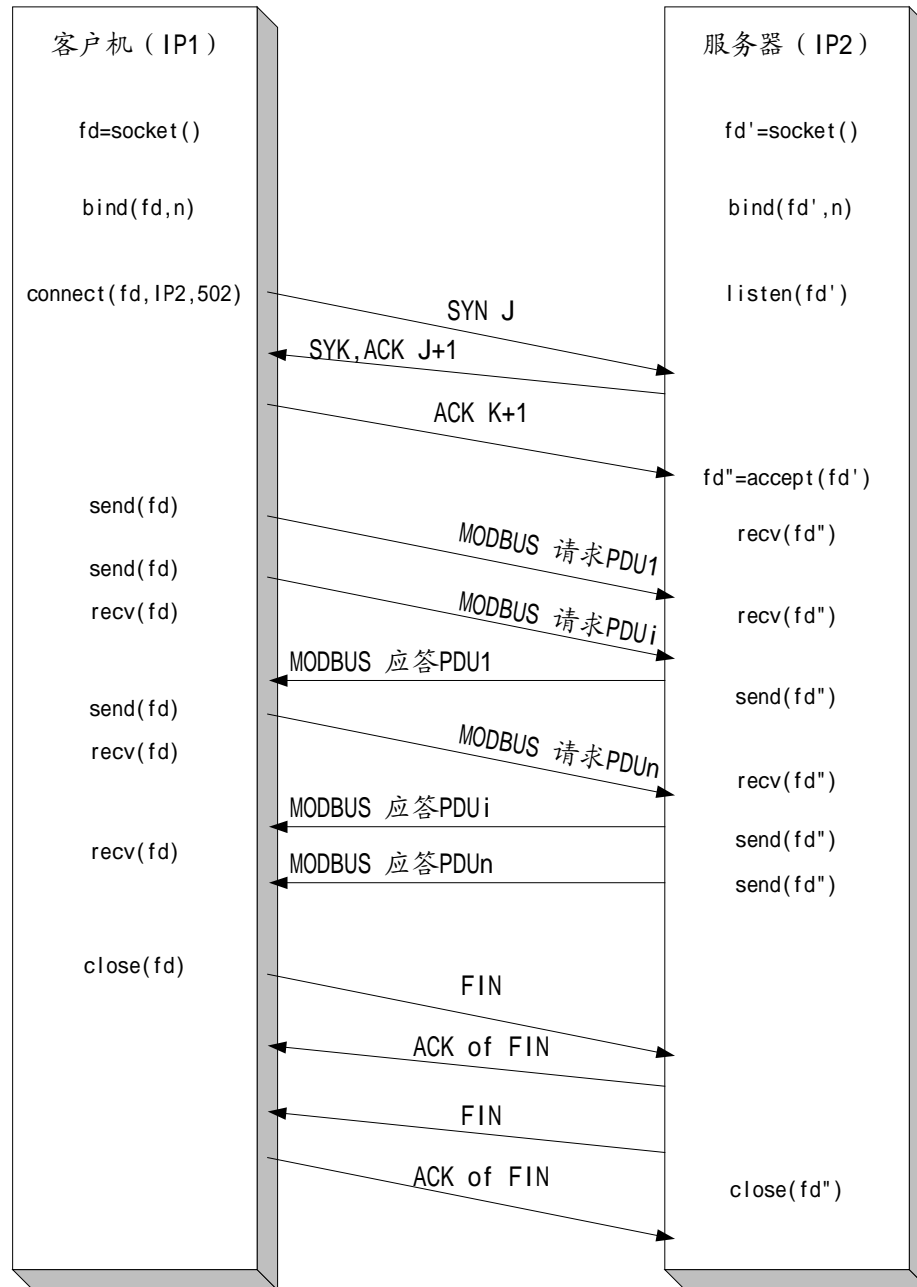


图9: MODBUS信息交换

上图给出了客户机与服务器间的完整的 MODBUS 通信过程。客户机建立一个连接，向服务器发送 3 个 MODBUS 请求，而不等待第一个请求的应答到来。在收到所有的应答后，客户机正常地关闭连接。

#### 4.3.2 TCP 层参数配置

可以调整 TCP/IP 栈的一些参数以使得其特性满足产品或系统的限制。TCP 层的下列参数可以进行调整:

##### I 每个连接的参数

SO-RCVBUF, SO-SNDBUF:

这些参数允许为发送和接收用套接字接口设定高限位。可以通过调整这些参数来实现流量控制管理。接收缓存区的大小即为每个连接 advertised window 的最大值。为了提高性能，必须增加套接字缓存区的大小。否则，这些值必须小于内部驱动器的资源，以便在内部驱动器的资源耗尽之前关闭 TCP 窗口。

接收缓存区大小取决于 TCP 窗口大小、TCP 最大段的大小和接收输入帧所需的时间。由于最大段的尺寸为 300 个字(一个 MODBUS 请求需要最大 256 字+MBAP 报文头),如果需要 3 帧进行缓存,可将套接字缓存区大小调整为 900 字。为了满足最大的缓存需求和预定的时间,可以增加 TCP 窗口的大小。

#### TCP-NODELAY:

通常，小报文（称为：tinygrams）在局域网（LAN）上的传输不会产生问题，因为多数局域网是不拥堵的，但是，这些 tinygrams 在广域网上将会造成拥堵。一个称为“NAGLE 算法”的简单方案是：收集小量的数据，当前面报文的 TCP 确认到达时再用单个进行发送。

为了获得更好的实时特性，建议：将小量的数据直接发送，而不要试图将其收集到一个段内再发送。这就是为什么建议强制 TCP-NODELAY 选项，这个选项禁用客户机和服务器连接的“NAGLE 算法”。

#### SO-REUSEADDR:

当 MODBUS 服务器关闭一个由远端客户启动的 TCP 连接时，在这个连接处于“时间等待”状态（两个 MSL：最大段寿命）的过程中，该连接所用的本地口号不能被再次用来打开一个新的连接。

建议：为每个客户机和服务器连接，指明 SO-REUSEADDR 选项，以迂回这个限制。此选项允许为自身分配一个口号，它作为连接的一部分在 2MSL 期间内等待客户机并侦听套接字接口。

#### SO-KEEPALIVE:

TCP/IP 协议缺省状态下，不通过空闲的 TCP 连接发送数据。因此，如果在 TCP 连接端这个过程没有发送数据，在两个 TCP 模块间就没有交换任何数据。这就假设客户机端应用和服务器端应用均采用计数器来探测连接的存活性，以便关闭连接。

建议：在客户机与服务器连接两端均采用 KEEPALIVE 选项，以便查询另一端得知对方是否故障并死机，或故障并重新启动。

然而，我们必须牢记，采用 KEEPALIVE 可能引起一个非常良好的连接，在瞬间故障时通信中断，如果保持连接计时器计时周期太短，将占用不必要的网络带宽。

### I 整个 TCP 层的参数

#### TCP 连接建立超时:

多数伯克利推出的系统将新连接建立的时限设定为 75 秒，这个缺省值应该适应于实时的应用限制。

#### 保持连接参数:

连接的缺省空闲时间是 2 小时。超过此空闲时间将触发一个保持连接试探过程。第一个保持连接试探后，在最大次数内每隔 75 秒发送一个试探，直到收到对试探的应答为止。

在一个空闲连接上发出保持连接试探的最大数是 8 次。如果发出最大试探次数之后而没有收到应答，TCP 向应用发出一个错误信号，由应用来决定关闭连接。

#### 超时与重发参数:



如果检测到一个 TCP 报文丢失，将重发此报文。检测丢失的方法之一是管理重发超时（RTO），如果没有收到来自远端的确认，超时终止。

TCP 进行 RTO 的动态评估。为此，在发送每个非重发的报文后测量往返时间(RTT)。往返时间（RTT）是指报文到达远端设备并从远端设备获得一个确认所用的时间。一个连接的往返时间是动态计算的，然而，如果 TCP 不能在 3 秒钟内获得 RTT 的估计，那么，就设定 RTT 的缺省值为 3 秒。

如果已经估算出 RTO，它将被用于下一个报文的发送。如果在估算的 RTO 终止之前没有收到下一个报文的确认，启用**指数补偿**算法。在一个特定的时间段内，允许相同报文最大次数的重发。之后，如果收不到确认，连接终止。

可以对某些栈设置连接终止之前重发的最大次数和重发的最长时间。

在 TCP 标准中定义了一些重发算法：

- § **Jacobson RTO 估计算法**用来估计重发超时(RTO)；
- § **Karn 算法**指出，在重发段，不应进行 RTO 估计；
- § **指数补偿算法**定义：对于 64 秒时间上限内每一次重发，加倍重发超时；
- § **快速重发算法**允许在收到 3 个重复确认之后进行重发。考虑这个算法是因为：在 LAN 上，可能会导致报文丢失的检测快于等待 RTO 终止的检测。

在 MODBUS 实现中，推荐使用这些算法。

### 4.3.3 IP 层的参数配置

#### 4.3.3.1 IP 参数

下列参数必须在 MODBUS 实现的 IP 层进行配置：

- **本地 IP 地址**：IP 地址可以是 A、B 或 C 类的一种。
- **子网掩码**：可基于各种原因，将 IP 网络划分成子网：使用不同的物理介质（例如：以太网、广域网等）、更有效的使用网络地址、以及控制网络流量的能力。子网掩码必须与本地 IP 地址的类型相一致。
- **缺省网关**：缺省网关的 IP 地址必须与本地 IP 地址在同一子网内。禁止使用 0.0.0.0 的值。如果没有定义网关，那么此值可设为 127.0.0.1 或本地 IP 地址。

*注：MODBUS 报文传输服务在 IP 层上不要求段功能。*

应该利用本地 IP 地址、子网掩码和省缺网关（不同于 0.0.0.0）配置本地 IP 端。

## 4.4 通信应用层

### 4.4.1 MODBUS 客户端

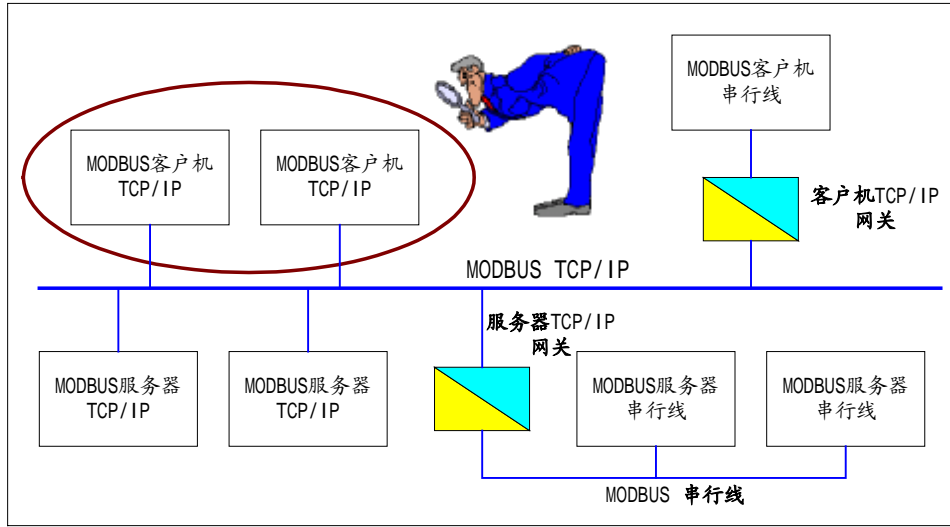


图 10: MODBUS 客户端

#### 4.4.1.1 MODBUS 客户端设计

MODBUS/TCP 协议使得能够对一个客户端进行简单的设计。下图描述了客户端发送 MODBUS 请求并处理 MODBUS 应答的主要处理过程。

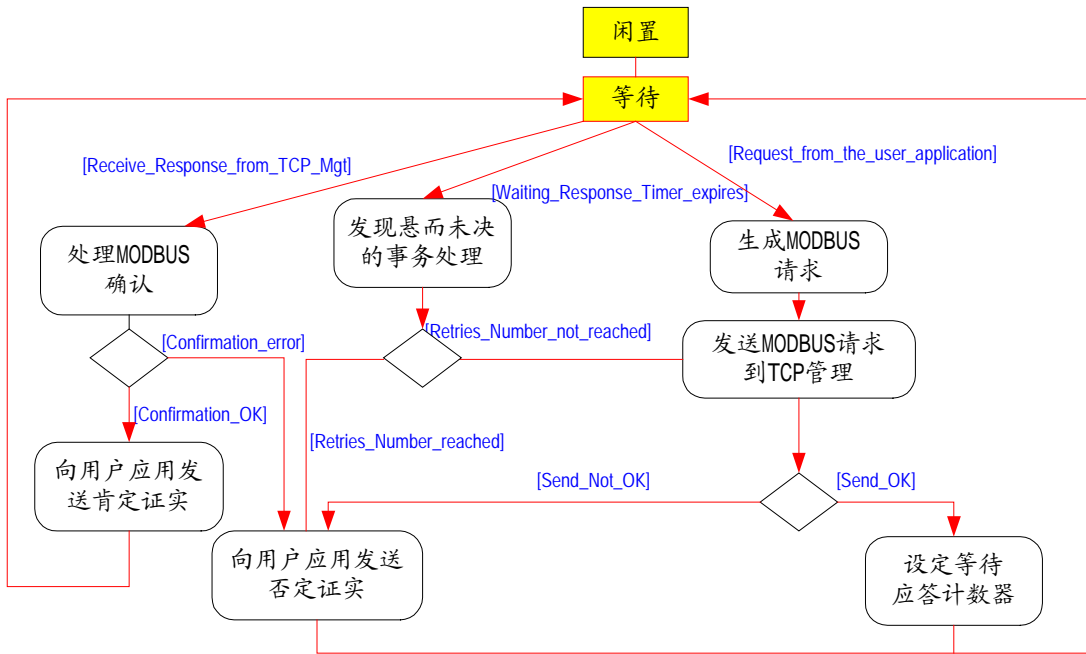


图 11: MODBUS 客户端操作示意图

一个 MODBUS 客户机可以接收三类事件：

§ 一个来自用户应用的发送请求的新需求，在这种情况下，必须对 MODBUS 请求进行编码，并使用 TCP 管理组件服务通过网络进行发送 MODBUS 请求。下层（TCP 管理模块）会返回一个错误信息，这些错误信息是由于 TCP 连接错误或其他错误信息所导致的。

§

§ 来自 TCP 管理的一个响应，在这种情况下，客户端必须分析响应的内容，并向用户应用发送一个证实。

§ 由于无响应而超时结束。可以通过网络发送一个重试电文，或向用户应用发送一个否定证实。

注：这些重试是由 MODBUS 客户机启动的，可以在无 TCP 确认的情况下由 TCP 层来进行其它类型的重试。

#### 4.4.1.2 MODBUS 请求的生成

在收到来自用户应用的需求后，客户端必须生成一个 MODBUS 请求，并发送到 TCP 管理。

可以将生成 MODBUS 请求分解成为几个子任务：

§ MODBUS 事务处理的实例化，使客户机能够存储所有需要的信息，以便将响应与相应的请求匹配，并向用户应用发送证实。

§ MODBUS 请求（PDU+MPAB 报文头）的编码。启动需求的用户应用必须提供所有需要的信息，使得客户机能够将请求编码。根据 MODBUS 协议进行 MODBUS PDU 的编码（MODBUS 功能码、相关参数和应用数据[2]）。填充 MBAP 报文头的所有域。然后，将 MBAP 报文头作为 PDU 前缀，生成 MODBUS 请求 ADU。

§ 发送 MODBUS 请求 ADU 到 TCP 管理模块，TCP 管理模块负责对远端服务器寻找正确 TCP 的套接字。除了 MODBUS ADU 以外，还必须传递目的 IP 地址。

下图比图 17 更深入地描述了请求生成的过程。

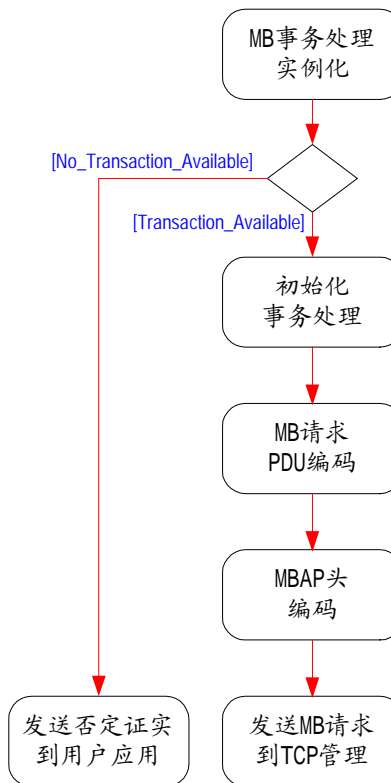


图 12：请求生成操作示意图

下面给出了实例：从地址为 05 的远端服务器读 1 个字的 MODBUS 请求 ADU 编码

§ MODBUS 请求 ADU 编码：

|           | 说明         | 大小 | 实例     |
|-----------|------------|----|--------|
| MBAP 报文头  | 事务处理标识符 Hi | 1  | 0x15   |
|           | 事务处理标识符 Lo | 1  | 0x01   |
|           | 协议标识符      | 2  | 0x0000 |
|           | 长度         | 2  | 0x0006 |
|           | 单元标识符      | 1  | 0xFF   |
| MODBUS 请求 | 功能码 (*)    | 1  | 0x03   |
|           | 起始地址       | 2  | 0x0005 |
|           | 寄存器数量      | 2  | 0x0001 |

(\*) 参见 MODBUS 协议规范 [2]

§ 事务处理标识符

事务处理标识符用于将请求与未来响应之间建立联系。因此，对 TCP 连接来说，在同一时刻，这个标识符必须是唯一的。有几种使用此标识符的方式：

- 例如：可以作为一个带有计数器的简单“TCP 序号”，在每一个请求时增加计数器；
- 也可以用作智能索引或指针，来识别事务处理的内容，以便记忆当前的远端服务器和未处理的请求。

通常，在 MODBUS 串行链路上，客户机必须一次发送一个请求。这意味着这个客户机在发送第二个请求之前必须等待对第一个请求的回答。在 MODBUS TCP 上，可以向同一个服务器发送多个请求而不需等待服务器的证实。MODBUS/TCP 到 MODBUS 串行链路之间的网关负责保证这两种操作之间的兼容性。

服务器收接受的请求数量取决于其容量，即：服务器资源量和 TCP 窗口尺寸。同样，客户机同时启动事务处理的数量也取决于客户机的资源容量。这个实现参数称为“NnnumberMaxofClientTransaction”，必须作为 MODBUS 客户机的一个特性进行描述。根据设备的类型，此参数取值为 1~16。

§ 单元标识符

在 MODBUS 或 MODBUS+ 串行链路子网中对设备进行寻址时，这个域是用于路由的目的。在这种情况下，“Unit Identifier”携带一个远端设备的 MODBUS 从站地址：

- 如果 MODBUS 服务器连接到 MODBUS+ 或 MODBUS 串行链路子网，并通过一个桥或网关配置地址这个服务器，MODBUS 单元标识符对识别连接到网桥或网关后的子网的从站设备是必需的。目的 IP 地址识别了网桥本身的地址，而网桥则使用 MODBUS 单元标识符将请求转交给正确的从站设备。
- 分配串行链路上 MODBUS 从站设备地址为 1~247（10 进制），地址 0 作为广播地址。

对 TCP/IP 来说，利用 IP 地址寻址 MODBUS 服务器；因此，MODBUS 单元标识符是无用的。必需使用值 0xFF。

- 当对直接连接到 TCP/IP 网络上的 MODBUS 服务器寻址时，建议不要在“单元标识符”域使用有效的 MODBUS 从站地址。在一个自动系统中重新分配 IP 地址的情况下，并且如果以前分配

给MODBUS服务器的IP地址又被指配给网关，使用一个有效的从站地址可能会由于网关的路由不畅而引起麻烦。使用无效的从站地址，网关仅是简单地废弃MODBUS PDU，而不会有任何问题。建议：在采用0xFF作为“单元标识符”的无效值。

注：0 也可以用作与MODBUS/TCP 设备直接通信。

#### 4.4.1.3 处理 MODBUS 证实

在 TCP 连接中，当收到一个响应帧时，位于 MBAP 报文头中的事务处理标识符用来将响应与先前发往 TCP 连接的原始请求联系起来：

- § 如果事务处理标识符没有提及任何未解决的事务处理，那么必须废弃响应；
- § 如果事务处理标识符提及了未解决的事务处理，那么必须分解响应，以便向用户应用发送 MODBUS 证实（肯定的或否定的证实）；

分解响应就是检验 MBAP 报文头和 MODBUS PDU 的响应：

- MBAP 报文头

在检验协议标识符必为 0x0000 以后，长度给出了 MODBUS 响应的大小。

如果响应来自直接连接到 TCP/IP 网络的 MODBUS 服务器设备，TCP 连接识别码足以清晰地识别出远端服务器。因此，MBAP 头中携带的单元标识符是无效的，必须废弃这个单元标识符。

如果将远端服务器连接在一个串行链路子网上，并且响应来自一个网桥、路由或网关，那么单元标识符（值≠0xFF）识别发送初始响应的远端 MODBUS 服务器。

- MODBUS 响应 PDU

必须检验功能码，根据 MODBUS 协议，分析 MODBUS 的响应格式：

- § 如果功能码与请求中所用的功能码相同，并且如果响应的格式是正确的，那么，向用户应用发出MODBUS响应作为肯定的证实。
- § 如果功能码是一个MODBUS异常码（功能码+80H），向用户应用发出一个异常响应作为肯定的证实。
- § 如果功能码与请求中所用的功能码不同（= 非预期的功能码），或如果响应的格式是错误的，那么，向用户应用发出一个错误信号作为否定的证实。

注：肯定证实是指服务器收到请求命令并做出响应的证实。并不意味着服务器能够成功地完成请求命令中要求的操作（MODBUS 异常响应指明执行操作失败）。

下图比图 17 更深入地描述了证实处理的过程。

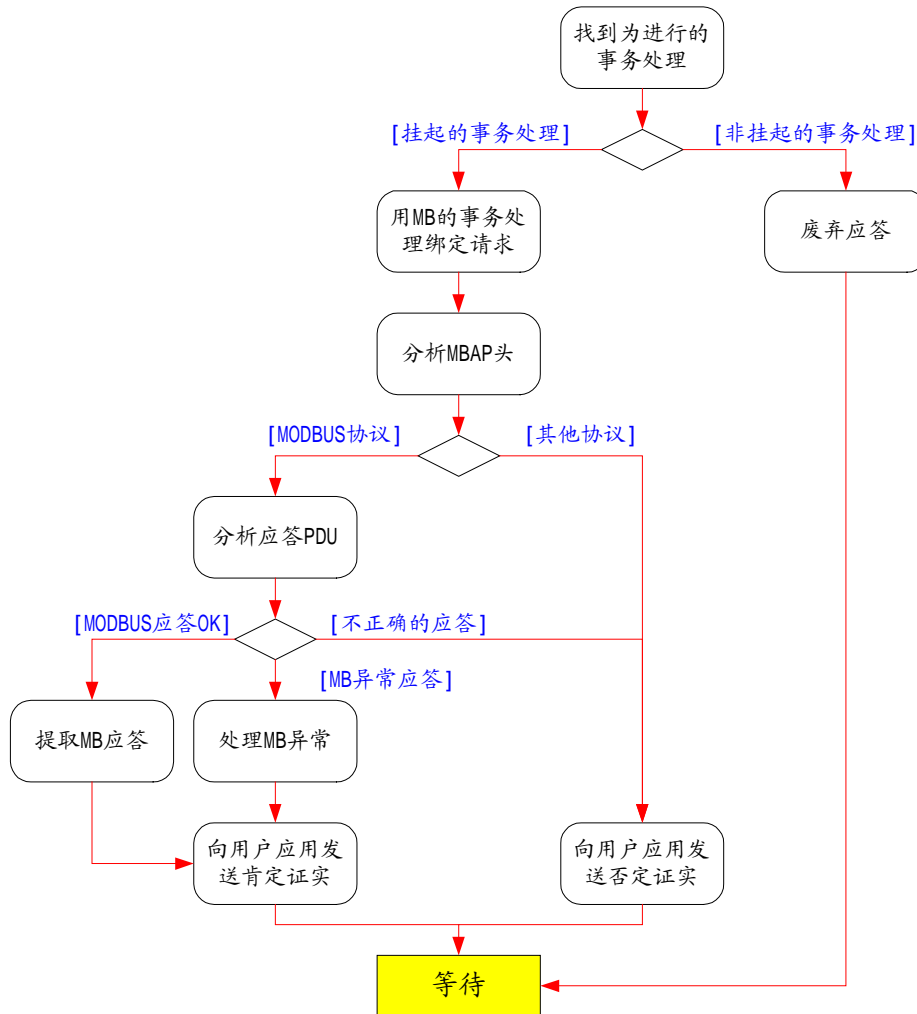


图 13: MODBUS 证实处理操作示意图

#### 4.4.1.4 超时管理

对 MODBUS/TCP 上事务处理所需响应时间有意不作规定。

这是因为：从毫秒级的 I/O 扫描到延时几秒钟的远距离无线链路，预期 MODBUS/TCP 会用于可能最宽泛的通信场合。

从客户机的角度，超时必须考虑网络上预期的传输延迟，以便确定一个合理的响应时间。这种传输延迟可能是交换式以太网中的几个毫秒，或广域网连接中的几百毫秒。

反过来讲，任何客户机启动应用重试使用的超时时间应该大于预期的最大的合理响应时间。如果不遵循这一点，目标设备或网络就存在过度拥挤的潜在危险，而反过来会导致更多的错误。这是一个应该始终避免的特性。

因此，在实际中，在高性能应用中所使用的客户机超时似乎总是与网络拓扑和期望的客户机性能有关。

时间因素不很重要的系统经常采用 TCP 缺省值作为超时值，在多数平台上，几秒钟之后将报告通信故障。

#### 4.4.2 MODBUS 服务器端

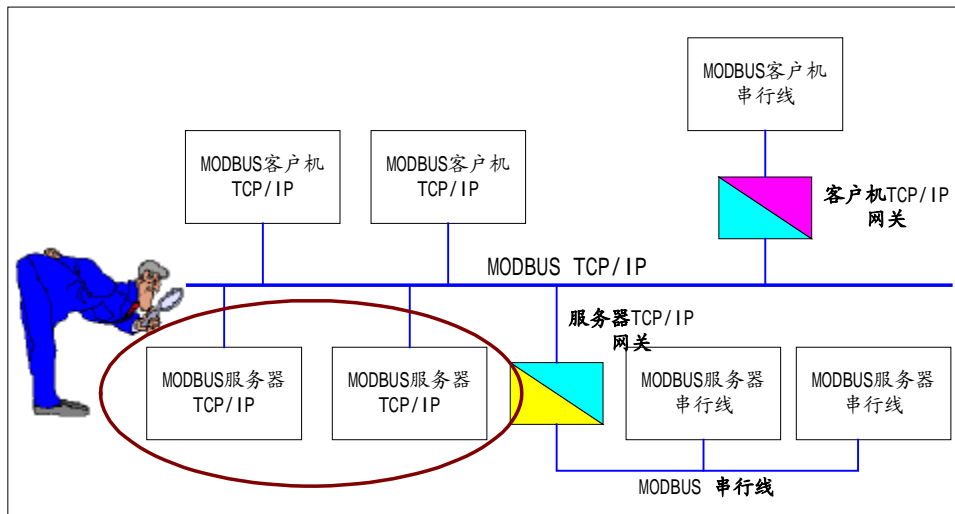


图 14: MODBUS 服务器端

MODBUS 服务器的作用是为应用对象提供访问以及为远端客户机提供服务。

根据用户应用，提供不同类型的访问：

- § 简单访问：获得或设定应用对象的属性；
- § 高级访问：启动一个特定的应用服务

MODBUS 服务器必须：

- § 将一个应用对象映射成可读或可写的MODBUS对象，以便获得或设定应用对象的属性；
- § 提供一种对应用对象启动服务的方法；

在运行过程中，MODBUS 服务器必须分析接收到的 MODBUS 请求，处理所需的操作，返回 MODBUS 响应。

##### 4.4.2.1 MODBUS 服务器设计

MODBUS 服务器设计取决于如下两个方面：

- § 对应用对象访问的类型（对属性的简单访问或对服务的高级访问）；
- § MODBUS服务器与用户应用之间交互作用的类型（同步或异步）。

下图描述了服务器进行的主要处理过程，以便获得来自 TCP 管理的 MODBUS 请求，然后，分析请求，处理所需的操作，返回 MODBUS 响应。

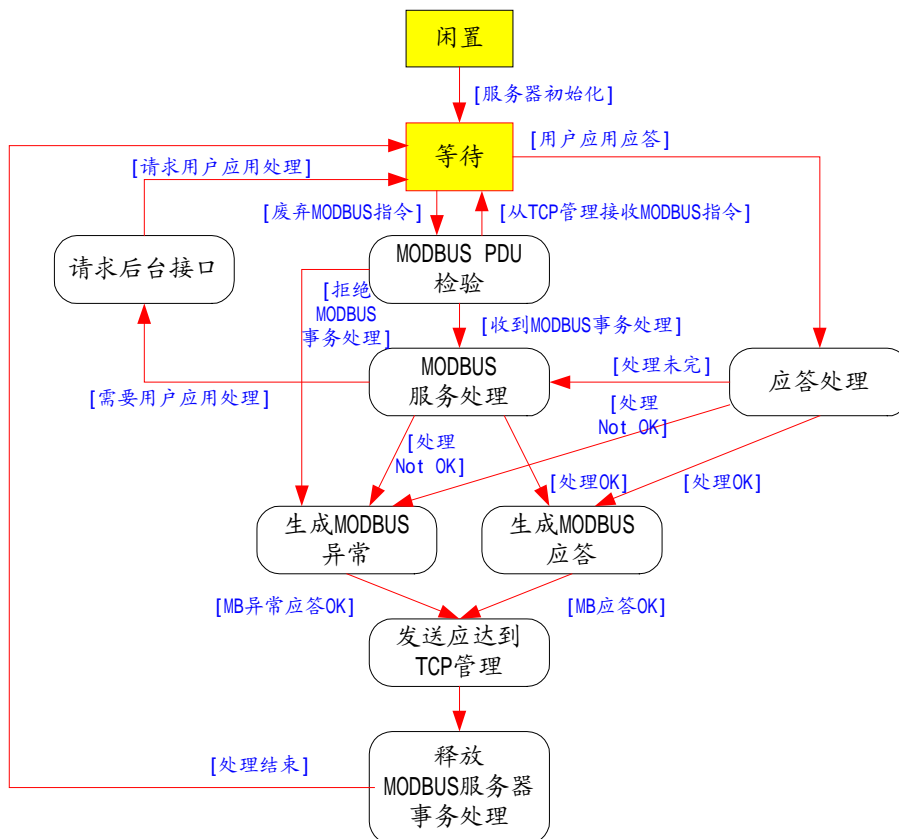


图 15: 处理 MODBUS 指令操作示意图

象前面的操作示意图示出的那样：

- § MODBUS服务器本身可以立即处理一些服务，没有与用户应用之间的交互作用；
- § 一些服务还可能需要与被处理的用户应用进行明显的交互作用；
- § 有些高级服务需要调用特定的接口，即：MODBUS后台服务。例如：可能根据用户应用层协议，使用若干个MODBUS请求/响应事务处理的时序来启动用户应用服务。后台服务负责所有单个MODBUS事务处理的正确进行，以便于执行全局用户应用服务。

在下列各章中给出更完整的描述。

MODBUS 服务器可以接收并同时为多个 MODBUS 请求提供服务。服务器可以同时接收 MODBUS 请求的最大数量是 MODBUS 服务器的主要特性之一。这个数量取决于服务器的设计以及它的处理和存储能力。将这个实现参数称为“NumberMaxOfServerTransaction”，必须作为 MODBUS 服务器的一个特性描述这个实现参数。根据设备的能力，它的取值范围为：1~16，。

“NumberMaxOfServerTransaction”参数对 MODBUS 服务器的操作和性能有非常显著的影响。尤其重要的是，所管理的并发 MODBUS 事务处理的数量可能影响服务器对 MODBUS 请求的响应时间。

#### 4.4.2.2 MODBUS PDU 检验

下图描述了 MODBUS PDU 检验操作。



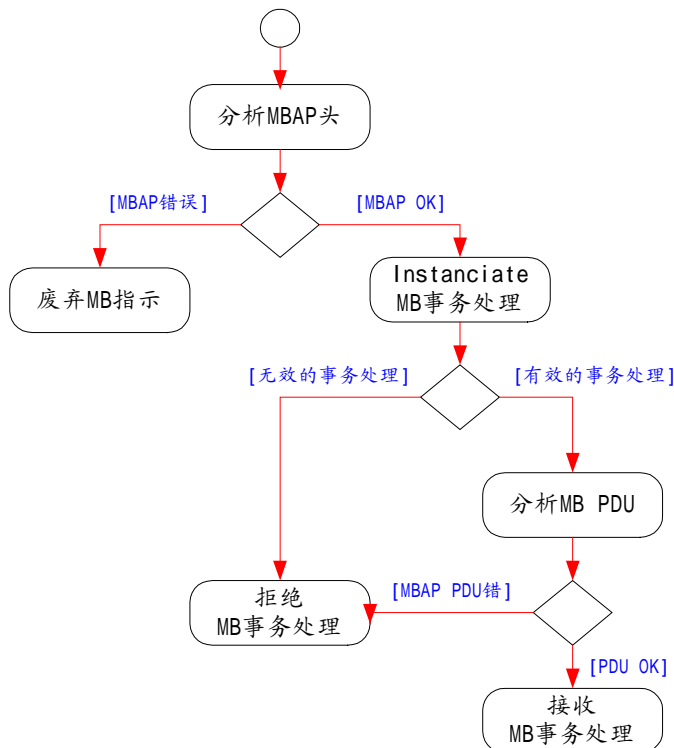


图 16: MODBUS PDU 检验操作流程

MODBUS PDU 检验功能首先是分解 MBAP 报文头。必须检验协议标识符域:

- § 如果与 MODBUS 协议类型不同，那么废除这个指示。
- § 如果是正确的 (= MODBUS 协议类型; 值为 0x00)，立即举例说明一个 MODBUS 事务处理。

一个服务器可以距离说明的 MODBUS 事务处理的最大数量由参数“NumberMaxOfTransaction”(系统或配置参数)来定义。

在无效的事务处理的情况下，服务器生成一个 MODBUS 异常响应 (异常码 6: 服务器繁忙)。如果事务处理是有效的，它将被启动，以便存储下列信息:

- § 用于发送指示的TCP连接标识符 (由TCP管理给出)
- § MODBUS事务处理ID (MBAP报文头中给出)
- § 单元标识符 (MBAP报文头中给出)

然后，分解 MODBUS PDU。首先分析功能码:

- § 当无效时，生成MODBUS异常响应 (异常码1: 无效功能)
- § 如果接收功能码，服务器启动一个“MODBUS服务处理”操作。

#### 4.4.2.3 MODBUS 服务处理

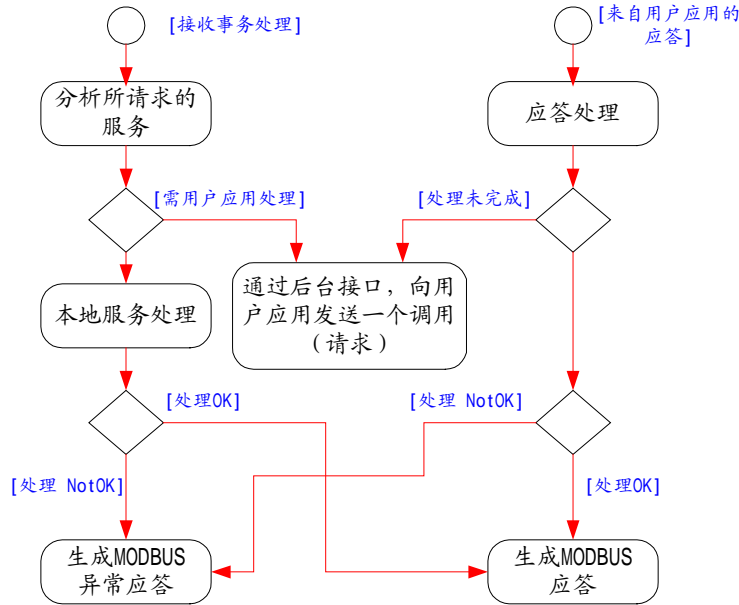


图 17: MODBUS 服务处理操作流程

根据后面实例中的设备软件和硬件结构，可以用不同的方式进行要求的 MODBUS 服务处理：

- 在一个小型设备或单线程体系结构内，MODBUS服务器可以直接访问用户应用数据，服务器自身可以本地处理要求的服务，而无需调用后台服务。

根据“MODBUS 协议规范”，进行这种处理。在出现错误的情况下，生成 MODBUS 异常响应。

- 在一个模块化的多处理器的设备或多线程体系结构中，“通信层”和“用户应用层”是两个独立的实体，通信实体可以完全地处理一些不重要的服务，而其他的服务需要应用后台服务与用户应用实体协调完成。

为了实现与用户应用的交互作用，MODBUS 后台服务必须执行所有适当的机制，以便处理用户应用的事务处理，并且正确管理用户应用调用和相应的响应。

#### 4.4.2.4 用户应用接口（后台接口）

在 MODBUS 后台服务中，可以执行几种策略来完成工作，虽然从用户网络吞吐量、接口带宽使用、响应时间、甚至设计工作量的角度，这几种策略是不均衡的。

MODBUS 后台服务将对用户应用采用适当接口：

- 或基于串行链路的物理接口，或双口RAM方案，或一条简单的 I/O 电缆，或由操作系统提供的基于报文传输服务的逻辑接口。
- 到用户应用的接口可以是同步的或异步的。

MODBUS 后台服务还将使用适当的设计模式来得到/设定目标属性或触发服务。在某些情况下，一个简单的“网关模式”将是足够的。在其他情况下，从简单的交换表历史到更复杂的重复机制中，设计者将必须执行带有高速缓存策略的“代理服务器模式”。

MODBUS 后台服务有责任实现协议的转换，以便与用户应用进行交互作用。因此，它必须具有机

制来实现报文的分拆和重组、数据一致性保证以及所有需要的同步等功能。

#### 4.4.2.5 MODBUS 响应的生成

一旦处理请求，MODBUS 服务器必须使用适当的 MODBUS 服务器事务处理生成一个响应，并且必须将响应发送到 TCP 管理组件。

根据处理结果，可以生成两类响应：

§ 肯定的MODBUS响应：

§ 响应功能码 = 请求功能码

§ MODBUS异常响应：

§ 目的是为客户机提供与处理过程检测到的错误相关的信息

§ 响应功能码 = 请求功能码+0x80

§ 提供异常码来表明出错的原因。

| 异常码 | MODBUS 名称 | 备注   |
|-----|-----------|--|
| 01  | 非法的功能码    | 服务器不了解功能码                                      |
| 02  | 非法的数据地址   | 与请求有关  |
| 03  | 非法的数据值    | 与请求有关  |
| 04  | 服务器故障     | 在执行过程中，服务器故障                                   |
| 05  | 确认        | 服务器接受服务调用，但是需要相对长的时间完成服务。因此，服务器仅返回一个服务调用接收的确认。 |
| 06  | 服务器繁忙     | 服务器不能接受 MODBUS 请求 PDU。客户应用由责任决定是否和何时重发请求。      |
| 0A  | 网关故障      | 网关路径是无效的。                                      |
| 0B  | 网关故障      | 目标设备没有响应。网关生成这个异常信息。                           |

MODBUS 响应 PDU 必须以 MBAP 报文头做前缀，使用事务处理正文中的数据生成 MBAP 报文头。

- 单元标识符

当在所收到的 MODBUS 请求中给出单元标识符时，拷贝这个单元标识符，并将其存储在事务处理的正文中。

- 长度

服务器计算 MODBUS PDU 和单元标识符字的大小。在“长度”域中设置这个值。

- 协议标识符

设置协议标识符域为 0x0000 (MODBUS 协议)，在所收到的 MODBUS 请求中给出协议标识符。

- 事务处理标识符

设置这个域为“事务处理标识符”值，它与初始请求有关，并将其存储在。

利用事务处理正文中存储的 TCP 连接对正确的 MODBUS 客户机返回 MODBUS 响应。当发送响应时，事务处理正文必须释空闲的。

## 5 实现指南

本章的目的是提出一个实现报文传输服务的实例。下面所描述的模型可用作客户机或服务器实现 MODBUS 报文传输服务过程的指南。

### 5.1 对象模型示意图

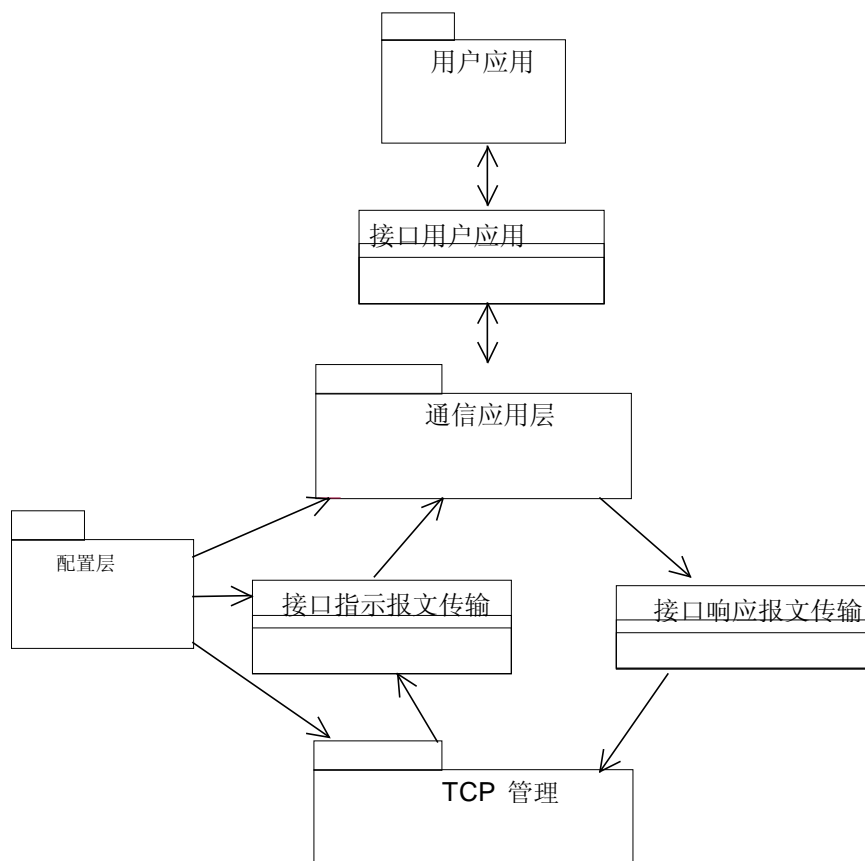


图 18: MODBUS 报文传输服务对象模型示意图

四种主要程序包构成对象模型示意图:

- l 配置层，它配置和管理其它程序包组件的操作模式
- l TCP 管理，它使 TCP/IP 栈和管理 TCP 连接的通信应用层连接。这指的是套接字接口的管理。
- l 通信应用层，它由在一侧的 MODBUS 客户机和在另一侧的 MODBUS 服务器组成。该程序包和用户应用链接。
- l 用户应用，它和设备应用相对应，它完全与设备有关，因此在本文件中不予讨论。

本模型与实现的选择无关，例如：OS 类型、存储管理等。为保证这种无相关性，在 TCP 管理层和通信层之间以及在通信层和用户应用层之间使用普通界面层（generic Interface layers）。

有不同的实现方法实现该界面：两项任务之间的传输、共享存储器、串行链接界面、过程呼叫

等。

为定义下面的实现模型，作了一些假定：

- I 静态存储器管理
- I 服务器的同步处理
- I 处理有关所有套接字接收的任务。

### 5.1.1 TCP 管理程序包

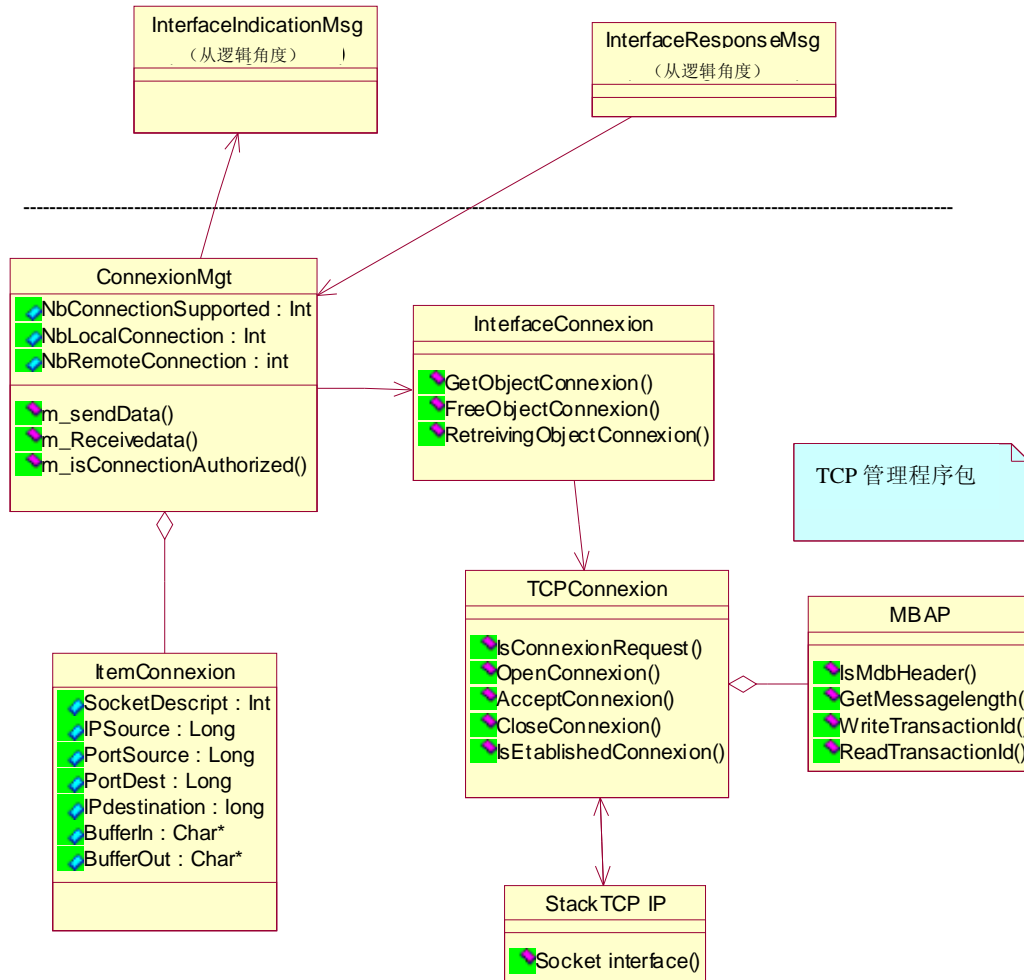


图 19: MODBUS TCP 管理程序包

TCP 管理程序包包括下列类：

**CInterfaceConnexion**：该类的作用是管理用于连接的存储库。

**CItemConnexion**：该类含有描述连接所需要的所有信息。

**CTCPConnexion**：该类提供自动管理 TCP 连接的方法（CStackTCP\_IP 提供接口套接字）。

**CconnexionMngt**：该类管理所有连接，并通过 CinterfaceindicationMsg 和 CinterfaceRoseponseMsg 向 MODBUS 服务器/MODBUS 客户机发送请求/响应。该类还处理连接建立的访问控制。

**CMBAP**：该类提供读/写/分析 MODBUS MBAP 的方法。

**CStackTCP\_IP**：该类执行套接字服务并提供栈的参数配置。

### 5.1.2 配置层程序包

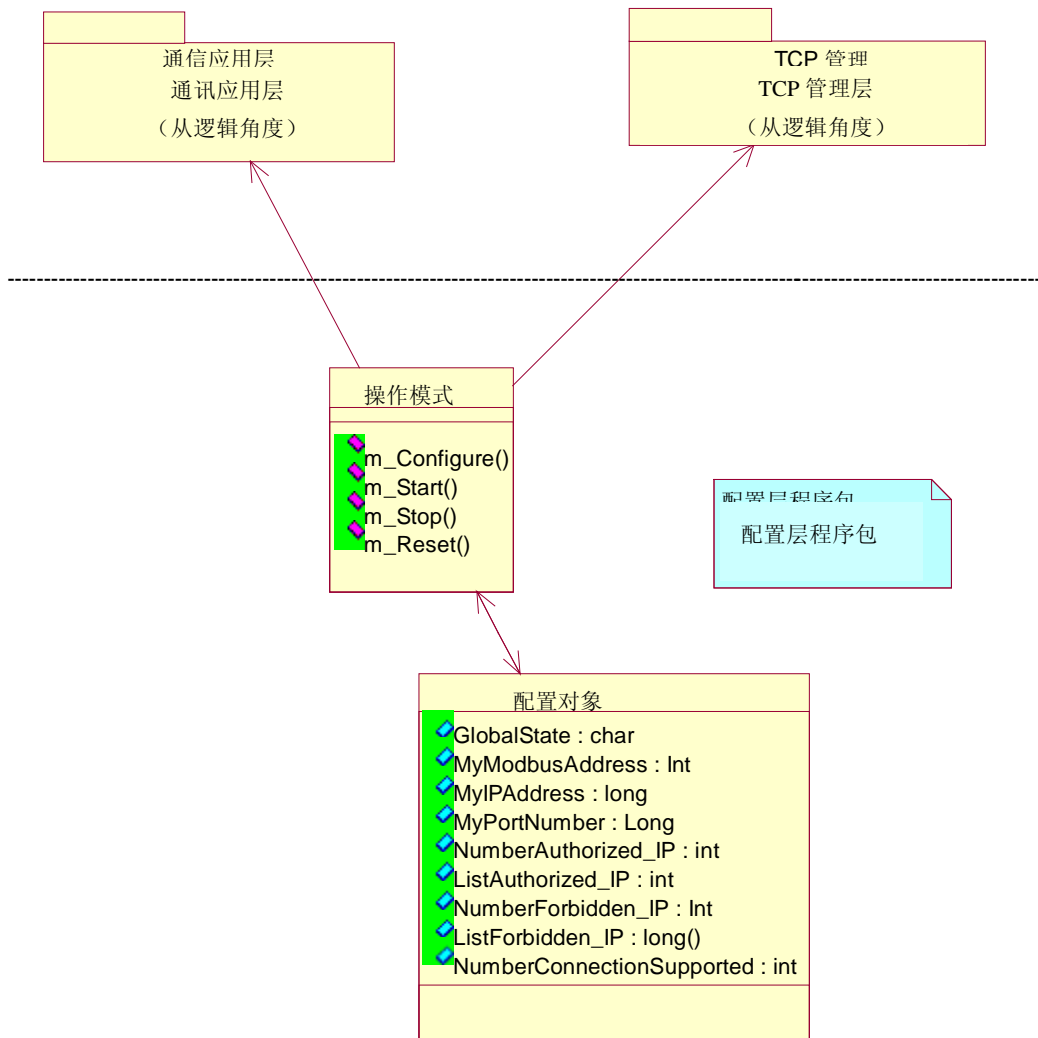


图 20: MODBUS 配置层程序包

配置层程序包包括下列类:

**TConfigureObject:** 该类将各组件相互配置所需的数据分组。用 CoperatingMode 类中的 m\_Confire 方法填充这个结构。每个需要配置的类从这个对象中取得自己的数据。配置数据与实现本身有关。因此，提供该类的属性表作为一个实例。

**CoperatingMode:** 该类的作用是填充 TConfigureObject（根据用户的配置）和管理下述类的操作模式。

- | CMODBUSServer
- | CMODBUSClient
- | CconnexionMngt

### 5.1.3 通信层程序包

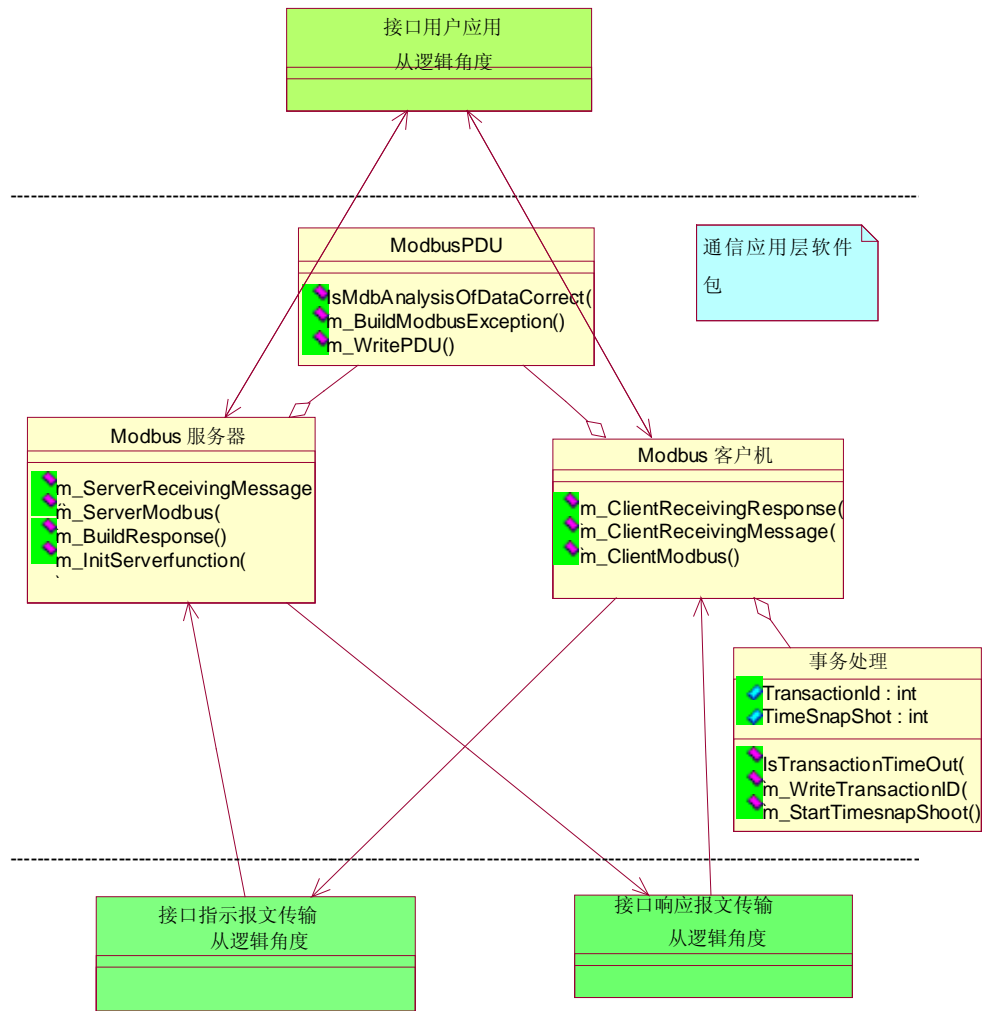


图 21: MODBUS 通信应用层程序包

通信应用层程序包包括以下各类:

**CMODBUSServer**: 从 CinterfaceIndicationMsg 中接收 MODBUS 询问 (通过 m\_ServerReceivingMessage 方法)。该类的作用是根据询问建立 MODBUS 响应或 MODBUS 异常 (从网络进入)。该类实现 MODBUS 服务器的 Graph State。只有类 CoperatingMode 发送了用户配置和正确的操作模式, 才可能生成响应。

**CMODBUSClient**: 从类 CinterfaceUserApplication 中读取 MODBUS 询问, 客户机的任务是用 m\_ClientReceivingMessage 方法接收询问。该类实现 MODBUS 客户机的 State Graph, 并且管理链接带有响应的询问的事务处理 (来自网络的)。只有类 CoperatingMode 发送了用户配置和正确的操作模式, 才能通过网络发送询问。

**CTransaction**: 该类实现管理事务的方法和结构。

#### 5.1.4 接口类

**CInterfaceUserApplication**: 该类表示与用户应用的接口, 它提供两种访问用户数据的方法。在实际的实现中, 根据硬件和软件设备的能力, 用不同的方式实现这种方法 (相当于一个终端驱动器、

访问 PCMCIA 的实例、共享存储器等)。

**CInterfaceIndicationMsg:** 该接口类用来从网络向 MODBUS 服务器发送询问, 以及从网络向客户机发送响应。该类使 TCPManagement 和通信应用层程序包连接 (从网络中)。该类的实现与网络有关。

**CInterfaceResponseMsg:** 该接口类用于从服务器接受响应, 以及从客户机向网络发送询问。该类使通信应用层程序包和 TCPManagement 连接 (向网络)。该类的实现和设备有关。

## 5.2 通信实现的类的示意图

下列类的示意图是一个完整的实现方案的示意图。



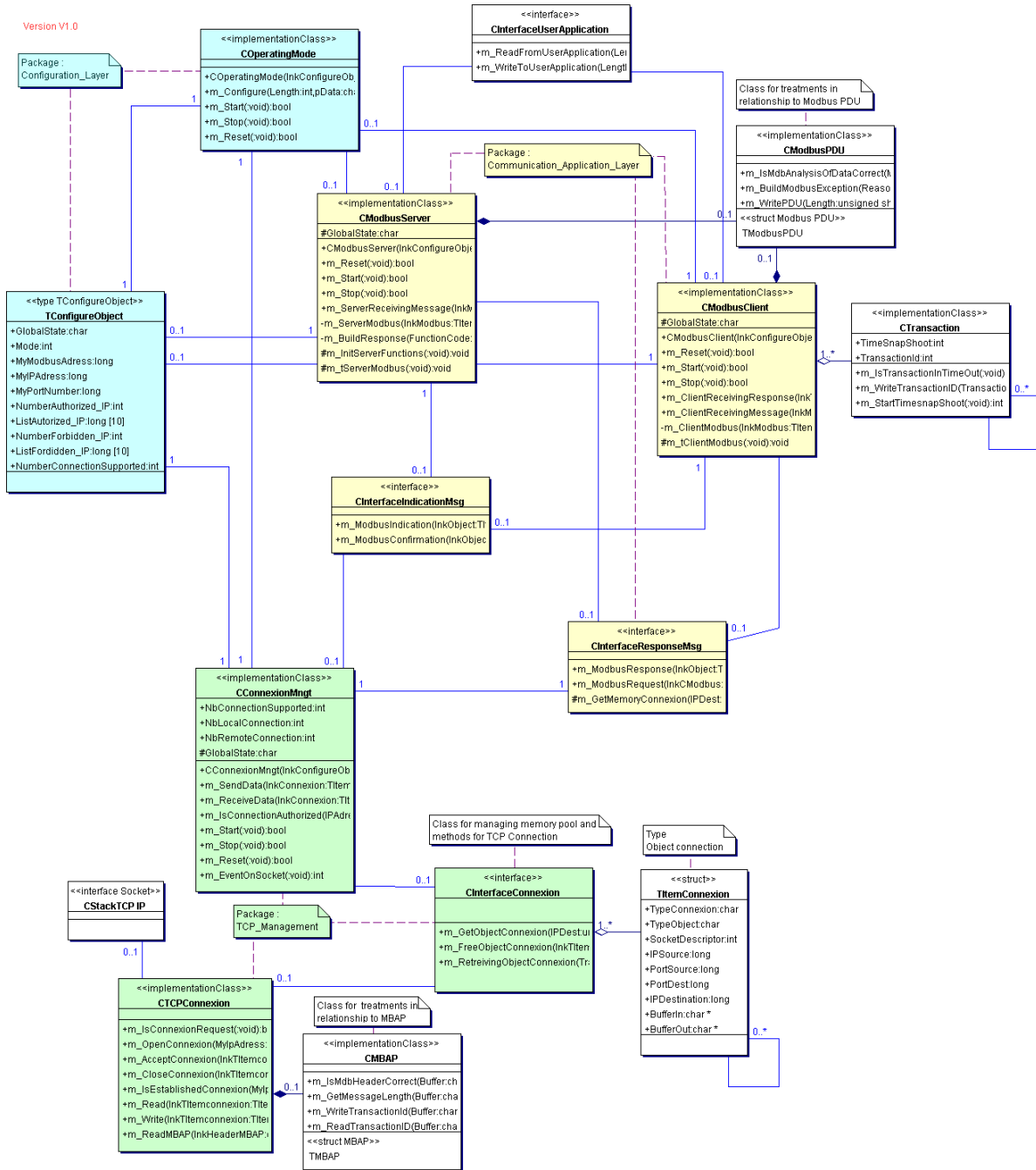


图 22: 类的示意图

### 5.3 序列图

下面描述了两个序列图，以便说明客户机 MODBUS 事务处理和服务器 MODBUS 事务处理。

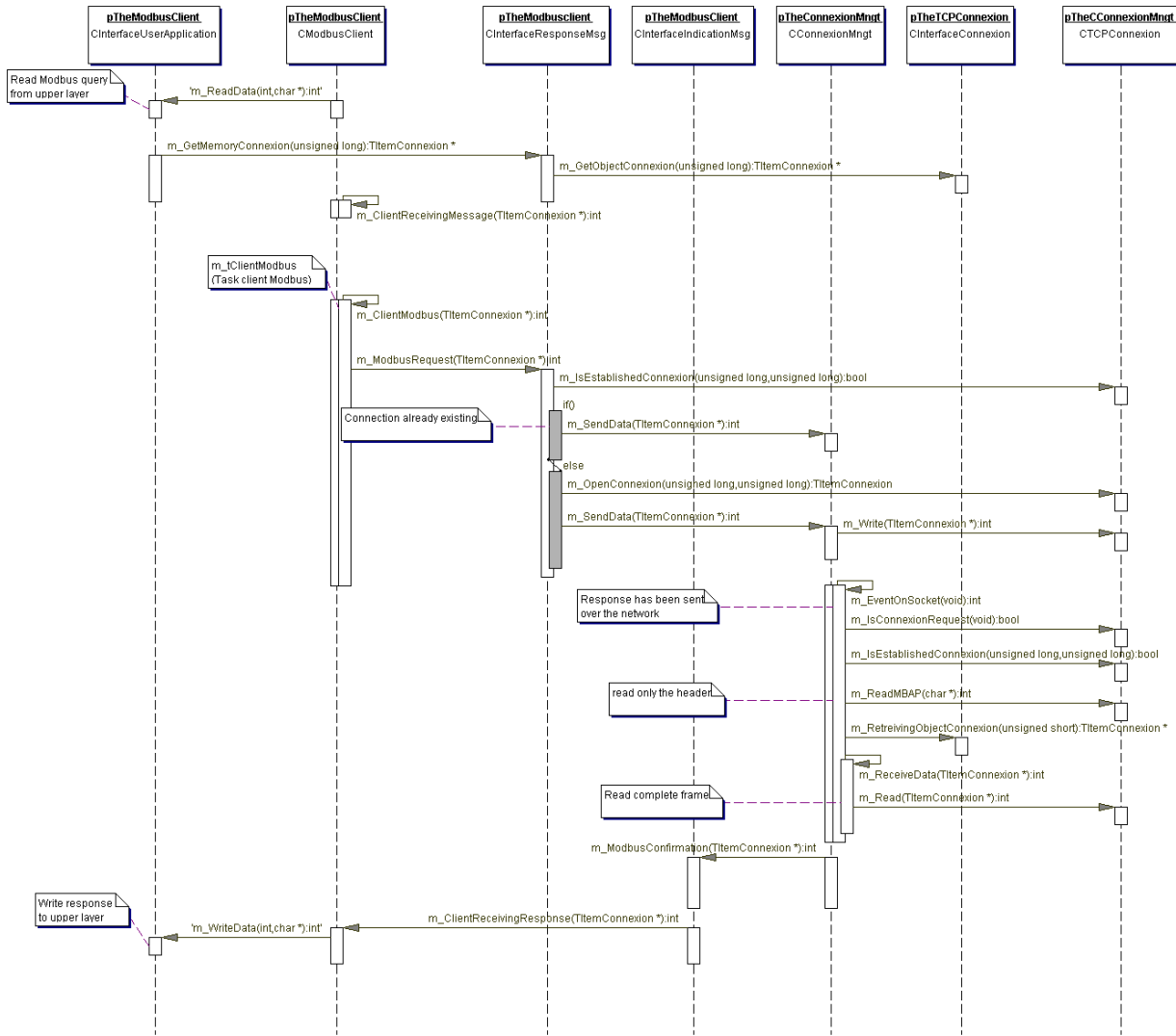


图 23: MODBUS 客户机序列图

为更好地理解客户机序列图，简单的注释为：

第一步：读来自用户应用的询问(通过 `m_Read` 方法)。

第二步：客户机的任务是接收 MODBUS 询问（通过 `m_ClientReceivingMessage` 方法）。这是客户机的进入点。为了使询问和相应的响应相互协调，当得到询问时，客户机使用事务处理资源（类名：`CTransaction`）。通过呼叫类接口 `CInterfaceResponseMsg` 向 `TCP_Management` 发送 MODBUS 询问（通过 `m_MODBUSRequest` 方法）。

第三步：如过已建立连接，并且不需要对连接做什么，那么通过网络发送报文。否则，在网络上可以发送报文之前，必须有开放的连接。

此时，客户机等待最后讨论的响应（来自远程服务器）。

第四步：若从已经从网络得到响应，那么 `TCP/IP` 栈接收数据（隐蔽地呼叫 `m_EventOnStack` 方法）。

如果已建立连接，那么读 `MBAP` 以恢复连接对象（连接对象提供存储资源和其它信息）。

读取来自网络的数据，通过类接口 `CInterfaceIndicationMsg` 向客户机发送证实（通过 `m_MODBUSConfirmation` 方法）。客户机的任务是接收 MODBUS 证实（通过

m\_ClientReceivingResponse 方法)。

最后，向用户应用写入响应（通过 m\_WriteData 方法），并且事务处理资源是空闲的。

下面是 MODBUS 服务器交换的实例：

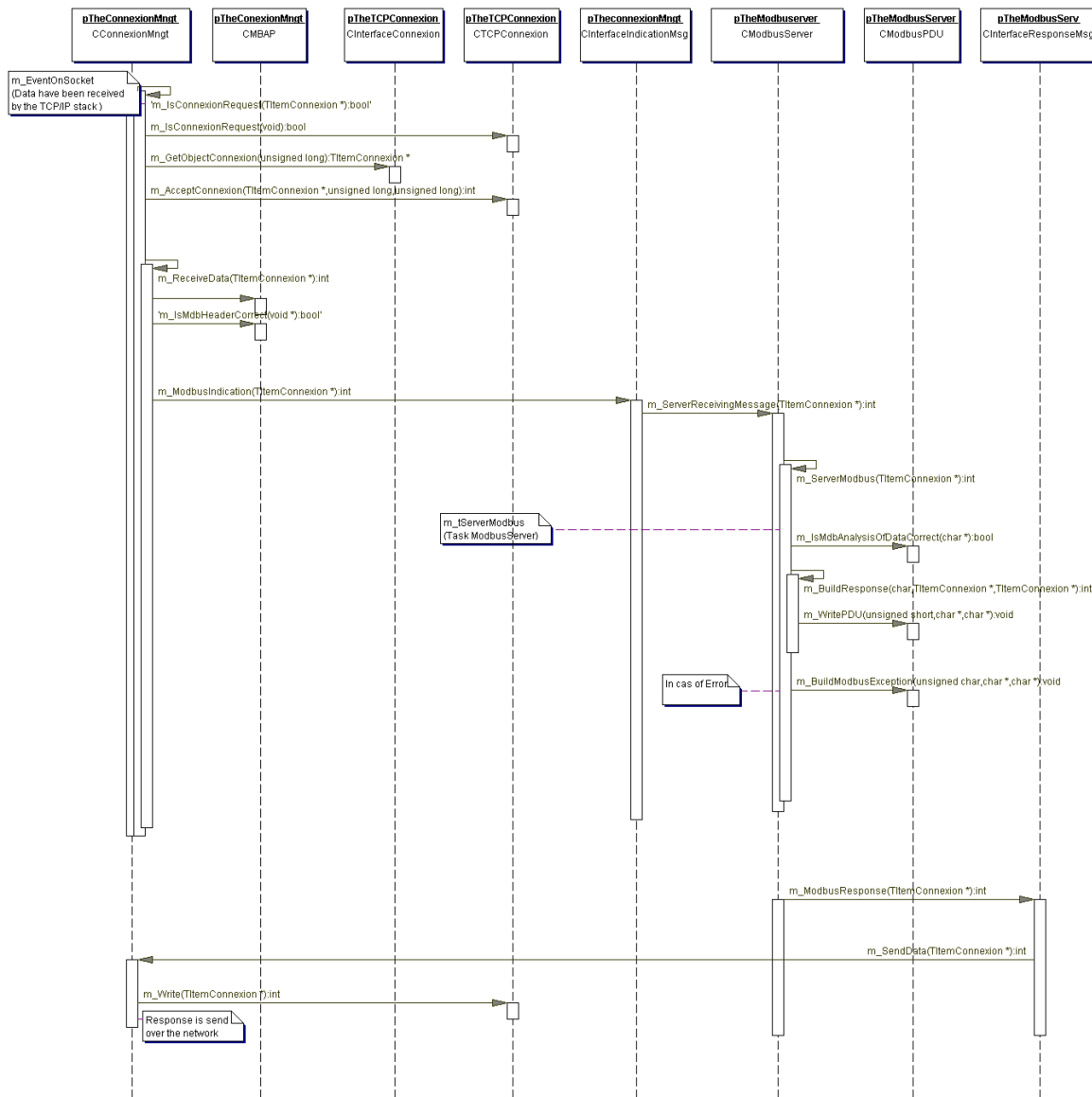


图 24: MODBUS 服务器序列图

为更好地理解客户机序列图，简单的注释为：

第一步：某客户机已通过网络发送了询问（MODBUS 询问）。TCP/IP 栈接收数据（隐蔽地呼叫 m\_EventOnSocket 方法）。

第二步：请求可能是连接请求，也可能不是连接请求（通过 m\_IsConnexionRequest 方法）。如果请求是连接请求，那么分配连接对象和收发 MODBUS 帧的缓冲器（m\_GetObjectConnexion）。之后，必须检查和接受连接访问控制。

第三步：如果询问是 MODBUS 请求，那么可以读出完整的 MODBUS Query（通过 m\_ReceiveData 方法）。此时，必须分析 MBAP（通过 m\_IsMdbHeadercorrect 方法）。通过 CInterfaceIndicationMessaging 类向服务器任务（task）发送完整的帧（通过 m\_MODBUSIndication 方法）。服务器的任务是接收 MODBUS QUERY（通过 m\_ServerReceivingMessage 方法）并加以分析。

如果有差错出现（未支持的功能码等），生成 MODBUS 异常帧形（m\_BuildMODBUSException），否则生成响应。

第四步：通过 CInterfaceResponseMessaging（通过 m\_MODBUSResponse 方法）在网络上发送响应。通过 m\_SendData 方法进行对连接对象的处理（恢复连接描述符等），在网络上发送数据。

## 5.4 类和方法的描述

### 5.4.1 MODBUS 服务器端的类

类名：CMODBUSServer

---

类名：CMODBUSServer

#### Stereotype 实现类

提供用服务器模式管理 MODBUS 报文传输的方法

---

|                |  |
|----------------|--|
| <b>域综述</b>     |  |
| protected char | <a href="#">GlobalState</a><br>MODBUS 服务器的状态 |

|   |  |
|---|--|
| <b>构造器综述</b>  |  |
| <a href="#">CMODBUSServer</a> (TConfigureObject * InkConfigureObject) |  |
| 构造器：生成内部对象  |  |

|                |  |
|----------------|--|
| <b>方法综述</b>    |  |
| protected void | <a href="#">m_InitServerFunctions</a> (void )<br>为填充功能阵列 “m_ServerFunction” 数列构造器呼叫的功能   |
| bool           | <a href="#">m_Reset</a> (void )<br>重新设置服务器的方法，如果重新设置，那么返回肯定值   |
| int            | <a href="#">m_ServerReceivingMessage</a> (TItemConnexion * InkMODBUS)<br>与 CindicationMsg:: m_MODBUSIndication 的接口，以从网络接收询问，如有问题，返回否定值 |
| bool           | <a href="#">m_Start</a> (void )<br>启动服务器的方法，如果启动，返回肯定值   |
| bool           | <a href="#">m_Stop</a> (void )<br>停止服务器的方法，如果停止，返回肯定值  |
| protected void | <a href="#">m_tServerMODBUS</a> (void )<br>服务器的 MODBUS 任务..  |

### 5.4.2 MODBUS 客户机类

类名: CMODBUSClient

类名: CMODBUSClient

提供用客户机模式管理 MODBUS 报文传输的方法

**Stereotype** 实现类

|                |  |
|----------------|--|
| 域综述            |  |
| protected char | <a href="#">GlobalState</a><br>MODBUS 客户机的状态 |

|   |  |
|---|--|
| 构造器综述   |  |
| <a href="#">CMODBUSClient</a> (TConfigureObject * InkConfigureObject) |  |
| 构造器: 生成内部对象, 启动 0 变量  |  |

|              |   |
|--------------|---|
| 方法综述         |   |
| Int          | <a href="#">m_ClientReceivingMessage</a> (TItemConnexion * InkMODBUS)<br>从应用层接收报文传输的接口: 呼叫<br>读数据呼叫的 CinterfaceUserApplication::m_Read<br>为事务处理取得存储器的 CInterfaceConnexion::m_GetObjectConnexion<br>如果有问题, 那么返回否定值 |
| Bool         | <a href="#">m_Reset</a> (空的)<br>重新设置组件的方法, 如果重新设置, 返回肯定值  |
| bool         | <a href="#">m_Start</a> (空的)<br>启动组件的方法, 如果启动, 返回肯定值  |
| bool         | <a href="#">m_Stop</a> (空的)<br>停止组件的方法, 如果停止, 返回肯定值   |
| protected 空的 | <a href="#">m_tClientMODBUS</a> (空的)<br>客户机的 MODBUS 任务....  |

### 5.4.3 接口的类

#### 5.4.3.1 接口指示类

类名: CInterfaceIndicationMsg

直接已知的子类

[CConnexionMngt](#)

类名: **CInterfaceIndicationMsg**

从 TCP\_Management 向 MODBUS 服务器或客户机发送报文的类

Stereotype 接口

| 方法综述 |   |
|------|---|
| int  | <a href="#">m MODBUSConfirmation(TItemConnexion * InkObject)</a><br>接收进入响应和呼叫客户机的方法: 通过可用参考值、报文排队、远程过程呼叫等, ...      |
| int  | <a href="#">m MODBUSIndication(TItemConnexion * InkObject)</a><br>读进入 MODBUS 询问和呼叫服务器的方法: 通过可用参考值、报文排队、远程过程呼叫等, ... |

#### 5.4.3.2 接口响应类

类名: **CInterfaceResponseMsg**

直接已知的子类:

[CMODBUSClient](#)、[CMODBUSServer](#)

class **CInterfaceResponseMsg**

从客户机或服务器向 TCP\_Management 发送响应或询问的类

Stereotype 接口

| 方法综述                             |   |
|----------------------------------|---|
| <a href="#">TItemConnexion *</a> | <a href="#">m GetMemoryConnexion(unsigned long IPDest)</a><br>从存储库得到对象 ItemConnexion, 如果存储不充足, 返回值-1                            |
| int                              | <a href="#">m MODBUSRequest(TItemConnexion * InkCMODBUS)</a><br>将进入 MODBUS 询问客户机写入 ConnexionMngt 的方法: 通过可用参考值、报文排队、远程过程呼叫等, ... |
| int                              | <a href="#">m MODBUSResponse(TItemConnexion * InkObject)</a><br>将 MODBUS 服务器的响应写入 ConnexionMngt 的方法: 通过可用参考值、报文排队、远程过程呼叫等, ...  |

#### 5.4.4 连接管理类

类名: **CConnexionMngt**

类名: **CConnexionMngt**

管理所有 TCP 连接的类

**Stereotype 实现类**

| 域综述            |  |
|----------------|--|
| protected char | <a href="#">GlobalState</a><br>组件 ConnexionMngt 的综合状态    |
| Int            | <a href="#">NbConnectionSupported</a><br>连接的全部数量         |
| Int            | <a href="#">NbLocalConnection</a><br>本地客户机向远程服务器开放的连接数量  |
| Int            | <a href="#">NbRemoteConnection</a><br>远程客户机向本地服务器开放的连接数量 |

| 构造器综述   |  |
|---|--|
| <a href="#">CconnexionMngt(TConfigureObject * InkConfigureObject)</a> |  |
| 构造器：生成内部对象、启动 0 变量  |  |

| 方法综述 |  |
|------|--|
| int  | <a href="#">m_EventOnSocket</a> (空的)<br>唤醒   |
| bool | <a href="#">m_IsConnectionAuthorized</a> (unsigned long IPAdress)<br>如果授权新连接，返回肯定值                             |
| int  | <a href="#">m_ReceiveData(TItemConnexion * InkConnexion)</a><br>与 CTCPCnexion::write 接口，从网络中读数据的方法，如果有问题，返回否定值 |
| bool | <a href="#">m_Reset</a> (空的)<br>重新设置 ConnectionMngt 组件的方法，如果重新设置，返回肯定值   |
| int  | <a href="#">m_SendData(TItemConnexion * InkConnexion)</a><br>为 CTCPCnexion::read 接口，向网络发送数据的方法，如果有问题，返回否定值     |
| bool | <a href="#">m_Start</a> (空的)<br>启动 ConnectionMngt 组件的方法，如果启动，返回肯定值   |
| bool | <a href="#">m_Stop</a> (空的)<br>停止组件的方法，如果停止，返回肯定值  |