# MODBUS TCP/IP 消息实现指南

## MODBUS Messaging on TCP/IP Implementation Guide

版本：V1.0b

日期：2006年10月24日

中文翻译版 | www.modbus.cn

# 1 引言

# 目录

# 1 引言

## 1.1 目标

本文档的目标是介绍基于 TCP/IP 的 MODBUS 消息服务，以提供参考信息帮助软件开发人员实现该服务。本文档不描述 MODBUS 功能码的编码方式，如需了解该信息，请阅读 MODBUS 应用协议规范 [1]。

本文档对 MODBUS 消息服务实现进行了准确而全面的描述。其目的是促进使用 MODBUS 消息服务的设备之间的互操作性。

本文档主要包括三个部分：

- MODBUS over TCP/IP 协议概述
- MODBUS 客户端、服务器和网关实现的功能描述
- 提出 MODBUS 实现示例对象模型的实现指南

## 1.2 客户端/服务器模型

MODBUS 消息服务在连接于 Ethernet TCP/IP 网络的设备之间提供客户端/服务器通信。

该客户端/服务器模型基于四种类型的消息：

- MODBUS 请求（Request）
- MODBUS 确认（Confirmation）
- MODBUS 指示（Indication）
- MODBUS 响应（Response）

| | | |
|---|---|---|
| MODBUS 客户端 | → 请求 → 指示 → | MODBUS 服务器 |
| MODBUS 客户端 | ← 响应 ← 确认 ← | MODBUS 服务器 |

MODBUS 请求是客户端在网络上发送的消息，用于发起一个事务。

MODBUS 指示是服务器端接收到的请求消息。

MODBUS 响应是服务器发送的响应消息。

MODBUS 确认是客户端端接收到的响应消息。

MODBUS 消息服务（客户端/服务器模型）用于实时信息交换：

- 两个设备应用程序之间
- 设备应用程序与其他设备之间

- HMI/SCADA 应用程序与设备之间
- PC 与提供在线服务的设备程序之间

## 1.3 参考文档

本节列出了在阅读本文档之前建议先阅读的文档：

[1] MODBUS Application Protocol Specification V1.1a.

[2] RFC 1122 Requirements for Internet Hosts -- Communication Layers

# 2 缩略语

| 缩略语 | 全称 | 说明 |
| --- | --- | --- |
| ADU | Application Data Unit | 应用数据单元 |
| IETF | Internet Engineering Task Force | 互联网工程任务组 |
| IP | Internet Protocol | 互联网协议 |
| MAC | Medium Access Control | 介质访问控制 |
| MB | MODBUS | MODBUS |
| MBAP | MODBUS Application Protocol | MODBUS 应用协议 |
| PDU | Protocol Data Unit | 协议数据单元 |
| PLC | Programmable Logic Controller | 可编程逻辑控制器 |
| TCP | Transport Control Protocol | 传输控制协议 |
| BSD | Berkeley Software Distribution | 伯克利软件发行版 |
| MSL | Maximum Segment Lifetime | 最大报文段生存时间 |

# 3 上下文

## 3.1 协议描述

### 3.1.1 通用通信架构

基于 MODBUS TCP/IP 的通信系统可以包括不同类型的设备：

- 连接到 TCP/IP 网络的 MODBUS TCP/IP 客户端和服务器设备
- 互连设备，如网桥、路由器或网关，用于 TCP/IP 网络与串行线子网络之间的互连，允许连接 MODBUS 串行线客户端和服务器终端设备。

图 1：MODBUS TCP/IP 通信架构

MODBUS 协议定义了一个独立于底层通信层的简单协议数据单元（PDU）。MODBUS 协议到特定总线或网络的映射可以在应用数据单元（ADU）上引入一些附加字段。

| 附加地址 | 功能码 | 数据 | 差错校验 |
|---|---|---|---|
| | PDU | | |
| | | | |

图 2：通用 MODBUS 帧

发起 MODBUS 事务的客户端构建 MODBUS 应用数据单元。功能码指示服务器要执行的操作类型。

### 3.1.2 MODBUS TCP/IP 应用数据单元

本节描述 MODBUS 请求或响应在 MODBUS TCP/IP 网络上传输时的封装方式。

| MBAP 头部 | 功能码 | 数据 |
|---|---|---|
| | PDU | |
| | | |

图 3：基于 TCP/IP 的 MODBUS 请求/响应

在 TCP/IP 上使用专用头部来标识 MODBUS 应用数据单元。它被称为 MBAP 头部（MODBUS Application Protocol header）。

该头部与串行线上使用的 MODBUS RTU 应用数据单元相比存在一些差异：

- MODBUS 串行线上通常使用的 MODBUS “从站地址” 字段被 MBAP 头部中的单字节“单元标识符” 所取代。“单元标识符”用于通过网桥、路由器和网关等设备进行通信，这些设备使用单个 IP 地址来支持多个独立的 MODBUS 终端单元。
- 所有 MODBUS 请求和响应的设计使得接收方可以验证消息是否已结束。对于 MODBUS PDU 具有固定长度的功能码，仅功能码即可满足要求。对于在请求或响应中携带可变数量数据的功能码，数据字段包含字节计数。
- 当 MODBUS 通过 TCP 传输时，MBAP 头部中携带额外的长度信息，允许接收方即使在消息被拆分为多个数据包传输时也能识别消息边界。显式和隐式长度规则的存在，以及在 Ethernet 上使用 CRC-32 差错校验码，使得请求或响应消息出现未检测到损坏的概率微乎其微。

### 3.1.3 MBAP 头部描述

MBAP 头部包含以下字段：

| 字段 | 长度 | 描述 | 客户端 | 服务器 |
|---|---|---|---|---|
| 事务标识符 (Transaction Identifier) | 2 字节 | 标识一个 MODBUS 请求/响应事务。由客户端初始化 | 由客户端初始化 | 服务器从接收到的请求中复制 |
| 协议标识符 (Protocol Identifier) | 2 字节 | 0 = MODBUS 协议 | 由客户端初始化 | 服务器从接收到的请求中复制 |
| 长度 (Length) | 2 字节 | 后续字节数 | 由客户端初始化（请求） | 由服务器初始化（响应） |
| 单元标识符 (Unit Identifier) | 1 字节 | 标识连接在串行线或其他总线上的远程从站 | 由客户端初始化 | 服务器从接收到的请求中复制 |

该头部共 7 字节长：

- 事务标识符 — 用于事务配对，MODBUS 服务器在响应中复制请求的事务标识符。
- 协议标识符 — 用于系统内多路复用。MODBUS 协议由值 0 标识。
- 长度 — 长度字段是后续字段的字节计数，包括单元标识符和数据字段。
- 单元标识符 — 该字段用于系统内路由目的。它通常用于通过 Ethernet TCP/IP 网络与 MODBUS 串行线之间的网关，与 MODBUS+ 或 MODBUS 串行线从站进行通信。该字段由 MODBUS 客户端在请求中设置，服务器必须在响应中以相同的值返回。

所有 MODBUS/TCP ADU 都通过 TCP 发送到注册端口 502。

注意：各字段以 Big-endian（大端序）方式编码。

## 3.2 MODBUS 功能码描述

MODBUS 应用层协议中使用的标准功能码在 MODBUS 应用协议规范 [1] 中有详细描述。

# 4 功能描述

此处介绍的 MODBUS 组件架构是一个通用模型，包含 MODBUS 客户端和服务器组件，可用于任何设备。

某些设备可能仅提供服务器或客户端组件。在本节的第一部分中，简要概述了 MODBUS 消息服务组件架构，随后描述了架构模型中呈现的每个组件。

## 4.1 MODBUS 组件架构模型

图 4：MODBUS 消息服务概念架构

- 通信应用层

MODBUS 设备可以提供客户端和/或服务器 MODBUS 接口。可以提供 MODBUS 后端接口，允许间接访问用户应用程序对象。

该接口可以由四个区域组成：输入离散量、输出离散量（线圈）、输入寄存器和输出寄存器。该接口与用户应用程序数据之间必须进行预映射（本地问题）。

| 主要表 | 对象类型 | 类型 | 说明 |
| --- | --- | --- | --- |
| 离散量输入 (Discretes Input) | 单比特 | 只读 | 此类数据可由 I/O 系统提供 |
| 线圈 (Coils) | 单比特 | 读写 | 此类数据可由应用程序修改 |
| 输入寄存器 (Input Registers) | 16 位字 | 只读 | 此类数据可由 I/O 系统提供 |
| 保持寄存器 (Holding Registers) | 16 位字 | 读写 | 此类数据可由应用程序修改 |

图 5：具有独立块的 MODBUS 数据模型

图 6：仅有 1 个块的 MODBUS 数据模型

• MODBUS 客户端

MODBUS 客户端允许用户应用程序显式控制与远程设备的信息交换。MODBUS 客户端根据用户应用程序发送到 MODBUS 客户端接口的请求中包含的参数构建 MODBUS 请求。

MODBUS 客户端使用 MODBUS 事务，其管理包括等待和处理 MODBUS 确认。

• MODBUS 客户端接口

MODBUS 客户端接口提供了一个接口，使应用程序能够为各种 MODBUS 服务构建请求，包括访问 MODBUS 应用对象。MODBUS 客户端接口（API）不属于本规范，尽管在实现模型中描述了一个示例。

• MODBUS 服务器

在接收到 MODBUS 请求时，该模块激活本地操作以读取、写入或执行其他操作。这些操作的处理对应用程序员完全透明。MODBUS 服务器的主要功能是在 502 TCP 端口上等待 MODBUS 请求，处理该请求，然后根据设备上下文构建 MODBUS 响应。

• MODBUS 后端接口

MODBUS 后端接口是从 MODBUS 服务器到用户应用程序的接口，在用户应用程序中定义了应用对象。

> 提示：后端接口不在本规范中定义。

• TCP 管理层

> 提示：本规范中的 TCP/IP 讨论部分基于参考文献 [2] RFC 1122，以帮助用户在 TCP/IP 上实现 MODBUS 应用协议规范 [1]。

消息服务的主要功能之一是管理通信的建立和终止，以及管理已建立 TCP 连接上的数据流。

• 连接管理

客户端和服务器 MODBUS 模块之间的通信需要使用 TCP 连接管理模块。它负责全局管理消息 TCP 连接。

连接管理提出了两种方案。要么由用户应用程序自身管理 TCP 连接，要么连接管理完全由此模块完成，因此对用户应用程序是透明的。后一种方案意味着灵活性较低。

TCP 侦听端口 502 保留用于 MODBUS 通信。默认必须在该端口上侦听。然而，某些市场或应用可能需要将另一个端口专用于 MODBUS over TCP。因此，强烈建议客户端和服务器为用户提供参数化 MODBUS over TCP 端口号的可能性。重要的是要注意，即使在某些应用中为 MODBUS 服务配置了另一个 TCP 服务器端口，除了任何应用特定端口外，TCP 服务器端口 502 仍必须可用。

- 访问控制模块

在某些关键环境中，必须禁止不受欢迎的主机访问设备内部数据。这就是需要安全模式的原因，如有需要可以实施安全过程。

- TCP/IP 协议栈层

TCP/IP 协议栈可以进行参数化，以适应不同产品或系统特有的数据流控制、地址管理和连接管理约束。通常使用 BSD socket 接口来管理 TCP 连接。

- 资源管理和数据流控制

为了平衡 MODBUS 客户端和服务器之间的入站和出站消息数据流，MODBUS 消息栈的所有层都提供了数据流控制机制。

资源管理和流控制模块首先基于 TCP 内部流控制，加上数据链路层和用户应用层的一些数据流控制。

# 4.2 TCP 连接管理

## 4.2.1 连接管理模块

### 4.2.1.1 一般描述

MODBUS 通信需要在客户端和服务器之间建立 TCP 连接。

连接的建立可以由用户应用程序模块显式激活，也可以由 TCP 连接管理模块自动激活。

在第一种情况下，必须在用户应用程序模块中提供应用编程接口来完全管理连接。该方案为应用程序员提供了灵活性，但需要对 TCP/IP 机制有良好的专业知识。

在第二种情况下，TCP 连接管理对用户应用程序完全隐藏，用户应用程序仅发送和接收 MODBUS 消息。TCP 连接管理模块负责在需要时建立新的 TCP 连接。

TCP 客户端和服务器连接数量的定义不在本文档的范围内（本文档中的值 n）。根据设备容量的不同，TCP 连接的数量可以不同。

实现规则：

1) 在没有明确用户要求的情况下，建议实现自动 TCP 连接管理。

2) 建议保持与远程设备的 TCP 连接打开，而不是为每个 MODBUS/TCP 事务打开和关闭连接。

注意：然而，MODBUS
客户端必须能够接受来自服务器的关闭请求并关闭连接。可以在需要时重新打开连接。

3) 建议 MODBUS 客户端与远程 MODBUS 服务器（相同 IP 地址）打开最少数量的 TCP 连接。每个应用一个连接可能是一个好的选择。

4) 多个 MODBUS 事务可以在同一 TCP 连接上同时激活。

注意：如果这样做，则必须使用 MODBUS 事务标识符来唯一标识匹配的请求和响应。

5) 在两个远程 MODBUS 实体之间的双向通信中（每个实体既是客户端又是服务器），需要为客户端数据流和服务器数据流分别打开连接。

6) 一个 TCP 帧必须只传输一个 MODBUS ADU。不建议在同一 TCP PDU 上发送多个 MODBUS 请求或响应。

图 7：TCP 连接管理活动图

### 1. 显式 TCP 连接管理

用户应用程序模块负责管理所有 TCP 连接：主动和被动建立、连接终止等。此管理适用于客户端和服务器之间的所有 MODBUS 通信。用户应用程序模块中使用 BSD Socket 接口来管理 TCP 连接。该方案提供了完全的灵活性，但意味着应用程序员具有足够的 TCP 知识。

必须根据设备能力和需求配置客户端和服务器连接数量的限制。

### 2. 自动 TCP 连接管理

TCP 连接管理对用户应用程序模块完全透明。连接管理模块可以接受足够数量的客户端和服务器连接。

尽管如此，在超过授权连接数量的情况下必须实现一种机制。在这种情况下，我们建议关闭最旧的未使用连接。

与远程伙伴的连接在从远程客户端或本地用户应用程序接收到第一个数据包时建立。如果从网络到达终止信号或在设备上本地决定终止，该连接将被关闭。在接收到连接请求时，可以使用访问控制选项来禁止未授权客户端的设备访问。

TCP 连接管理模块使用协议栈接口（通常是 BSD Socket 接口）与 TCP/IP 协议栈通信。

为了在系统需求和服务器资源之间保持兼容性，TCP 管理将维护 2 个连接池。

- 第一个池（优先连接池）由不会在本地主动关闭的连接组成。必须提供配置来设置此池。要实现的原则是将特定 IP 地址与此池的每个可能连接关联。具有此类 IP 地址的设备被称为“标记的”。来自标记设备的任何新连接请求必须被接受，并将从优先连接池中获取。还需要配置每个远程设备允许的最大连接数，以避免同一设备使用优先连接池的所有连接。
- 第二个池（非优先连接池）包含非标记设备的连接。此处的规则是：当来自非标记设备的新连接请求到达且池中没有可用连接时，关闭最旧的连接。

可以选择性地提供配置来分配每个池中可用的连接数。然而（非强制性的）设计者可以在设计时根据需要设置连接数。

#### 4.2.1.2 连接管理描述

- 连接建立：

MODBUS 消息服务必须在端口 502 上提供侦听套接字，以便接受新连接并与其他设备交换数据。

当消息服务需要与远程服务器交换数据时，它必须打开一个与远程端口 502 的新客户端连接，以便与该远程设备交换数据。本地端口必须大于 1024，且每个客户端连接的端口不同。

| 设备 | | 设备 |
|---|---|---|
| 客户端端口 | 连接 (@IP1 n, @IP2 502) | 服务器端口 |
| 502 | | 502 |
| n (n>1024) | | n (n>1024) |

图 8：MODBUS TCP 连接建立

如果客户端和服务器连接数大于授权连接数，则关闭最旧的未使用连接。可以激活访问控制机制来检查远程客户端的 IP 地址是否被授权。如果未被授权，则拒绝新连接。

- MODBUS 数据传输

MODBUS 请求必须在已打开的正确 TCP 连接上发送。使用远程的 IP 地址来查找 TCP 连接。如果与同一远程设备打开了多个 TCP 连接，必须选择一个连接来发送 MODBUS 消息，可以使用不同的选择标准，如最旧的、第一个。在所有 MODBUS 通信期间必须保持连接打开。如以下各节所述，客户端可以在不等待前一个事务结束的情况下与服务器发起多个 MODBUS 事务。

- 连接关闭

当客户端和服务器之间的 MODBUS 通信结束时，客户端必须发起关闭用于这些通信的连接。

### 4.2.2 操作模式对 TCP 连接的影响

某些操作模式（两个操作端点之间的通信中断、某一端点的崩溃和重启等）可能对 TCP 连接产生影响。连接可能在一侧被视为关闭或中止，而另一侧不知情。这种连接被称为“半开”（half-open）。

本节描述了每种主要操作模式的行为。假设两个端点都使用了 Keep Alive TCP 机制（见 4.3.2 节）。

#### 4.2.2.1 两个操作端点之间的通信中断：

通信中断的原因可能是服务器端的 Ethernet 电缆断开。预期行为如下：

- 如果当前没有在连接上发送数据包：如果通信中断持续时间小于 Keep Alive 定时器值，则不会检测到通信中断。如果通信中断持续时间超过 Keep Alive 定时器值，则向 TCP 管理层返回错误，可以重置连接。
- 如果在断开前后发送了一些数据包：TCP 重传算法（Jacobson 算法、Karn 算法和指数退避算法，见 4.3.2 节）被激活。这可能在 Keep Alive 定时器到期之前导致 TCP 协议栈层重置连接。

#### 4.2.2.2 服务器端点的崩溃和重启

服务器崩溃和重启后，连接在客户端是“半开”的。预期行为如下：

- 如果没有在半开连接上发送数据包：只要 Keep Alive 定时器未到期，TCP 半开连接从客户端被视为打开。此后，向 TCP 管理层返回错误，可以重置连接。
- 如果在半开连接上发送了一些数据包：服务器在不存在的连接上接收到数据。TCP 协议栈层发送 Reset 以关闭客户端的半开连接。

#### 4.2.2.3 客户端的崩溃和重启

客户端崩溃和重启后，连接在服务器端是“半开”的。预期行为如下：

- 没有在半开连接上发送数据包：只要 Keep Alive 定时器未到期，TCP 半开连接从服务器端被视为打开。此后，向 TCP 管理层返回错误，可以重置连接。
- 如果客户端在 Keep Alive 定时器到期之前打开新连接：需要研究两种情况：

- 连接打开与服务器端的半开连接具有相同的特征（相同的源端口和目标端口、相同的源 IP 地址和目标 IP 地址），因此连接打开将在 TCP 协议栈层在连接建立超时后失败（在大多数 Berkeley 实现中为 75 秒）。为避免在此长时间超时期间无法通信，建议确保在客户端重启后打开连接时使用与之前不同的源端口号。

- 连接打开与服务器端的半开连接不具有相同的特征（不同的源端口、相同的目标端口、相同的源 IP 地址和目标 IP 地址），因此连接在 TCP 协议栈层被打开并向服务器 TCP 管理层发出信号。

如果服务器 TCP 管理层仅支持来自一个远程客户端 IP 地址的一个连接，它可以关闭旧的半开连接并使用新连接。

如果服务器 TCP 管理层支持来自一个远程客户端 IP 地址的多个连接，新连接保持打开，旧连接也保持半开状态，直到 Keep Alive 定时器到期后向 TCP 管理层返回错误。此后，TCP 管理层将能够重置旧连接。

### 4.2.3 访问控制模块

该模块的目标是检查每个新连接，使用授权远程 IP 地址列表，该模块可以授权或禁止远程客户端 TCP 连接。

在关键环境中，应用程序员需要选择访问控制模式以保护其网络访问。在这种情况下，需要对每个远程 IP 地址进行授权/禁止访问。用户需要提供 IP 地址列表，并为每个 IP 地址指定是否被授权。默认情况下，在安全模式下，用户未配置的 IP 地址将被禁止。因此，在访问控制模式下，来自未知 IP 地址的连接将被关闭。

## 4.3 TCP/IP 协议栈的使用

TCP/IP 协议栈提供接口来管理连接、发送和接收数据，以及进行一些参数化以适应设备或系统约束的协议栈行为。

本节的目标是概述协议栈接口以及有关协议栈参数化的信息。本概述重点关注 MODBUS 消息使用的功能。

如需更多信息，建议阅读 RFC 1122，它为互联网通信软件的供应商和设计者提供指导。它列举了连接到互联网的主机必须使用的标准协议，以及一组明确的需求和选项。

协议栈接口通常基于 BSD（Berkeley Software Distribution）接口，本文档描述的就是该接口。

### 4.3.1 BSD Socket 接口的使用

> 注意：某些 TCP/IP 协议栈出于性能原因提出了其他类型的接口。MODBUS 客户端或服务器可以使用这些特定接口，但本规范不描述这种使用方式。

套接字（socket）是通信的端点。它是通信的基本构建块。MODBUS 通信通过套接字发送和接收数据来执行。TCP/IP 库仅提供使用 TCP 的流式套接字，提供基于连接的通信服务。

套接字通过 socket() 函数创建。返回一个套接字号，创建者随后使用它来访问套接字。套接字创建时没有地址（IP 地址和端口号）。在端口绑定到套接字之前，它不能用于接收数据。

bind() 函数用于将端口号绑定到套接字。bind() 在套接字和指定的端口号之间创建关联。

为了发起连接，客户端必须发出 connect() 函数，指定套接字号、远程 IP 地址和远程侦听端口号（主动连接建立）。为了完成连接，服务器必须发出 accept() 函数，指定在先前 listen() 调用中指定的套接字号（被动连接建立）。将创建一个与初始套接字具有相同属性的新套接字。这个新套接字连接到客户端的套接字，其号返回给服务器。初始套接字因此可以用于其他可能想要连接到服务器的客户端。

TCP 连接建立后，可以传输数据。send() 和 recv() 函数专门设计用于已连接的套接字。

setsockopt()
函数允许套接字创建者将选项与套接字关联。这些选项修改套接字的行为。这些选项的描述在 4.3.2 节中给出。

select() 函数允许程序员测试所有套接字上的事件。

shutdown() 函数允许套接字用户在套接字上禁用 send() 和/或 recv()。

一旦套接字不再需要，可以使用 close() 函数丢弃其套接字描述符。

图 9：MODBUS 交换 描述了客户端和服务器之间的完整 MODBUS 通信。客户端建立连接，向服务器发送 3 个 MODBUS 请求而不等待第一个请求的响应。在收到所有响应后，客户端正确地关闭连接。

图 9：MODBUS 交换

## 4.3.2 TCP 层参数化

TCP/IP 协议栈的一些参数可以调整，以使其行为适应产品或系统约束。以下参数可以在 TCP 层中调整：

- 每个连接的参数：

  SO_RCVBUF, SO_SNDBUF：

  这些参数允许设置发送和接收套接字的高水位标记。可以调整它们用于流控制管理。接收缓冲区的大小是该连接的最大通告窗口。必须增加套接字缓冲区大小以提高性能。然而，这些值必须小于内部驱动程序资源，以便在耗尽内部驱动程序资源之前关闭 TCP 窗口。

  接收缓冲区大小取决于 TCP 窗口大小、TCP 最大报文段大小和吸收传入帧所需的时间。如果最大报文段大小为 300 字节（MODBUS 请求最多需要 256 字节 + MBAP 头部大小），如果需要 3 帧缓冲，套接字缓冲区大小值可以调整为 900 字节。对于更大的需求和更好的调度时间，可以增大 TCP 窗口大小。

  TCP_NODELAY：

  小数据包（称为 tinygrams）在 LAN 上通常不是问题，因为大多数 LAN 不拥塞，但这些 tinygrams 可能导致广域网上的拥塞。一个简单的解决方案，称为“NAGLE 算法”，是收集少量数据并在前一个数据包的 TCP 确认到达时将它们在单个报文段中发送。

  为了获得更好的实时行为，建议直接发送少量数据而不尝试将它们聚集在单个报文段中。这就是为什么建议强制 TCP_NODELAY 选项以在客户端和服务器连接上禁用“NAGLE 算法”。

SO_REUSEADDR：

当 MODBUS 服务器关闭由远程客户端发起的 TCP 连接时，用于该连接的本地端口号在该连接处于“Time-wait”状态期间（持续两个 MSL：Maximum Segment Lifetime）不能用于新的打开。

建议为每个客户端和服务器连接指定 SO_REUSEADDR 选项以绕过此限制。该选项允许进程为处于 2MSL 等待状态的客户端和侦听套接字分配端口号。

SO_KEEPALIVE：

默认情况下，TCP/IP 协议不会在空闲 TCP 连接上发送数据。因此，如果 TCP 连接两端没有任何进程向对方发送数据，两个 TCP 模块之间不会有任何交换。这意味着客户端应用程序或服务器应用程序使用定时器来检测非活动状态以关闭连接。

建议在客户端和服务器连接上都启用 KEEPALIVE 选项，以便轮询另一端以了解远程设备是已崩溃并停机还是已崩溃并重启。

尽管如此，我们必须记住，启用 KEEPALIVE 可能在瞬时故障期间导致完全正常的连接被断开，如果 Keep Alive 定时器太短，还会在网络上消耗不必要的带宽。

- 整个 TCP 层的参数：

建立 TCP 连接的超时：

大多数 Berkeley 派生系统将新连接建立的时间限制设为 75 秒，此默认值应根据应用的实时约束进行调整。

Keep Alive 参数：

连接的默认空闲时间为 2 小时。超过此值的空闲时间会触发 Keep Alive 探测。在第一次 Keep Alive 探测之后，除非收到探测响应，否则每 75 秒发送一次探测，最多发送一定次数。

空闲连接上发送的 Keep Alive 探测最大次数为 8。如果发送最大次数的 Keep Alive 探测后未收到探测响应，TCP 向应用程序发出错误信号，应用程序可以决定关闭连接。

超时和重传参数：

如果检测到 TCP 数据包丢失，则会重传该数据包。检测丢失的一种方法是管理重传超时（RTO），如果未收到来自远程端的确认，RTO 将到期。

TCP 管理 RTO 的动态估计。为此，在发送每个非重传数据包后测量往返时间（RTT）。往返时间（RTT）是数据包到达远程设备并返回确认到发送设备所需的时间。连接的 RTT 是动态计算的，但如果 TCP 无法在 3 秒内获得估计值，则 RTT 的默认值设为 3 秒。

如果 RTO 已被估计，则它适用于下一个数据包的发送。如果下一个数据包的确认未在估计的 RTO 到期之前收到，则激活指数退避（Exponential BackOff）。在一定时间内允许同一数据包的最大重传次数。此后，如果未收到确认，则连接被中止。

最大重传次数和连接中止前的最长时间（tcp_ip_abort_interval）可以在某些协议栈上设置。

TCP 标准中定义了一些重传算法：

- Jacobson 的 RTO 估计算法用于估计重传超时（RTO）。
- Karn 算法表示不应在重传报文段上进行 RTO 估计。
- 指数退避（Exponential BackOff）定义每次重传的重传超时加倍，上限为 64 秒。
- 快速重传算法允许在收到三个重复确认后进行重传。建议使用此算法，因为在 LAN 上它可能比等待 RTO 到期更快地检测到数据包丢失。

建议在 MODBUS 实现中使用这些算法。

### 4.3.3 IP 层参数化

#### 4.3.3.1 IP 参数

在 MODBUS 实现的 IP 层中必须配置以下参数：

- 本地 IP 地址：IP 地址可以是 A 类、B 类或 C 类的一部分。
- 子网掩码：对 IP 网络进行子网划分可能有多种原因：使用不同的物理介质（如 Ethernet、WAN 等）、更有效地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 IP 地址类别一致。
- 默认网关：默认网关的 IP 地址必须与本地 IP 地址在同一子网中。值 0.0.0.0 被禁止。如果不需要定义网关，则此值应设置为 127.0.0.1 或本地 IP 地址。

> 注意：MODBUS 消息服务不需要 IP 层的分片功能。

本地 IP 端点应配置本地 IP 地址、子网掩码和默认网关（不同于 0.0.0.0）。

## 4.4 通信应用层

### 4.4.1 MODBUS 客户端

> 图 10：MODBUS 客户端单元

### 4.4.1.1 MODBUS 客户端设计

MODBUS/TCP 协议的定义允许客户端的简单设计。以下活动图描述了客户端为发送 MODBUS 请求和处理 MODBUS 响应而处理的主要操作。

图 11：MODBUS 客户端活动图

MODBUS 客户端可以接收三种事件：

- 来自用户应用程序的发送请求新需求，在这种情况下，必须编码 MODBUS 请求并使用 TCP 管理组件服务在网络上发送。底层（TCP 管理模块）可能由于 TCP 连接错误或其他错误而返回错误。
- 来自 TCP 管理的响应，在这种情况下，客户端必须分析响应内容并向用户应用程序发送确认。
- 由于无响应导致的超时到期。可以在网络上发送新的重试，或向用户应用程序发送否定确认。

注意：这些重试由 MODBUS 客户端发起，在 TCP 确认缺失的情况下，TCP 层也可以进行其他重试。

### 4.4.1.2 构建 MODBUS 请求

在收到来自用户应用程序的需求后，客户端必须构建 MODBUS 请求并将其发送到 TCP 管理。

构建 MODBUS 请求可以分为几个子任务：

- MODBUS 事务的实例化，使客户端能够记忆所有必要信息，以便稍后将响应对应到请求并向用户应用程序发送确认。
- MODBUS 请求的编码（PDU + MBAP 头部）。发起需求的用户应用程序必须提供所有必要信息，使客户端能够编码请求。MODBUS PDU 根据 MODBUS 应用协议规范 [1] 进行编码（MB 功能码、相关参数和应用数据）。MBAP 头部的所有字段都被填充。然后，通过在 PDU 前添加 MBAP 头部来构建 MODBUS 请求 ADU。
- 将 MODBUS 请求 ADU 发送到 TCP 管理模块，该模块负责找到通向远程服务器的正确 TCP 套接字。除了 MODBUS ADU 外，还必须传递目标 IP 地址。

图 12：请求构建活动图

以下示例描述了读取远程服务器中寄存器 #5 的 MODBUS 请求 ADU 编码：

| 描述 | 大小 | 示例 | |
|---|---|---|---|
| 事务标识符 Hi | 1 | 0x15 | MBAP 头部 |
| 事务标识符 Lo | 1 | 0x01 | |

| 描述 | 大小 | 示例 | |
|---|---|---|---|
| 协议标识符 | 2 | 0x0000 | |
| 长度 | 2 | 0x0006 | |
| 单元标识符 | 1 | 0xFF | |
| 功能码 (*) | 1 | 0x03 | MODBUS 请求 |
| 起始地址 | 2 | 0x0004 | |
| 寄存器数量 | 2 | 0x0001 | |

(*) 请参见 MODBUS 应用协议规范 [1]。

- 事务标识符

事务标识符用于将未来的响应与请求关联。因此，在某一时刻，在一条 TCP 连接上，此标识符必须是唯一的。使用事务标识符有多种方式：

- 例如，它可以作为简单的“TCP 序列号”使用，使用一个每次请求递增的计数器。
- 它也可以巧妙地用作智能索引或指针来标识事务上下文，以便记忆当前远程服务器和待处理的 MODBUS 请求。

通常，在 MODBUS 串行线上，客户端必须一次发送一个请求。这意味着客户端必须等待第一个请求的应答后再发送第二个请求。在 TCP/MODBUS 上，可以向同一服务器发送多个请求而无需等待确认。MODBUS/TCP 到 MODBUS 串行线网关负责确保这两种行为之间的兼容性。

服务器接受的请求数量取决于其在资源数量和 TCP 窗口大小方面的容量。同样，客户端同时初始化的事务数量也取决于其资源容量。此实现参数称为“NumberMaxOfClientTransaction”，必须作为 MODBUS 客户端特性之一进行描述。根据设备类型，此参数的值可以从 1 到 16。

- 单元标识符

该字段用于在 MODBUS+ 或 MODBUS 串行线子网络上寻址设备时的路由目的。在这种情况下，“单元标识符”携带远程设备的 MODBUS 从站地址：

如果 MODBUS 服务器连接到 MODBUS+ 或 MODBUS 串行线子网络，并通过网桥或网关进行寻址，则需要 MODBUS 单元标识符来标识网桥或网关后面的子网络上连接的从站设备。目标 IP 地址标识网桥本身，网桥使用 MODBUS 单元标识符将请求转发到正确的从站设备。

MODBUS 从站设备在串行线上的地址从 1 到 247（十进制）分配。地址 0 用作广播地址。

在 TCP/IP 上，使用 IP 地址寻址 MODBUS 服务器；因此，MODBUS 单元标识符无用。必须使用值 0xFF。

当寻址直接连接到 TCP/IP 网络的 MODBUS 服务器时，建议不要在“单元标识符”字段中使用有意义的 MODBUS 从站地址。如果在自动化系统中重新分配 IP 地址，且先前分配给 MODBUS 服务器的 IP 地址随后被分配给网关，使用有意义的从站地址可能因网关的错误路由而导致问题。使用无意义的从站地址，网关将简单地丢弃 MODBUS PDU 而不会产生问题。建议使用 0xFF 作为“单元标识符”的无意义值。

> 注意：值 0 也可以被接受用于直接与 MODBUS/TCP 设备通信。

#### 4.4.1.3 处理 MODBUS 确认

当在 TCP 连接上接收到响应帧时，MBAP 头部中携带的事务标识符用于将响应与先前在该 TCP 连接上发送的原始请求关联：

- 如果事务标识符不引用任何 MODBUS 待处理事务，则必须丢弃响应。
- 如果事务标识符引用了 MODBUS 待处理事务，则必须解析响应以便向用户应用程序发送 MODBUS 确认（肯定或否定确认）。

解析响应包括验证 MBAP 头部和 MODBUS PDU 响应：

- MBAP 头部

在验证协议标识符必须为 0x0000 之后，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 连接标识足以明确标识远程服务器。因此，MBAP 头部中携带的单元标识符无意义（值 0xFF），必须丢弃。如果远程服务器连接在串行线子网络上，且响应来自网桥、路由器或网关，则单元标识符（值 != 0xFF）标识最初发送响应的远程 MODBUS 服务器。

- MODBUS 响应 PDU

必须验证功能码并根据 MODBUS 应用协议分析 MODBUS 响应格式：

- 如果功能码与请求中使用的相同，且响应格式正确，则将 MODBUS 响应作为肯定确认（Positive Confirmation）提供给用户应用程序。
- 如果功能码是 MODBUS 异常码（功能码 + 0x80），则将 MODBUS 异常响应作为肯定确认提供给用户应用程序。
- 如果功能码与请求中使用的不同（= 非预期功能码），或响应格式不正确，则使用否定确认（Negative Confirmation）向用户应用程序发出错误信号。

> 注意：肯定确认是确认命令已被服务器接收并响应。它并不意味着服务器能够成功执行命令（未能成功执行命令由 MODBUS 异常响应指示）。

图 13：处理 MODBUS 确认活动图

#### 4.4.1.4 超时管理

对于 MODBUS/TCP 上的事务，有意没有规定所需的响应时间规范。

这是因为 MODBUS/TCP 预期将在尽可能广泛的通信情况下使用，从期望亚毫秒级定时的 I/O 扫描器到延迟数秒的长距离无线电链路。

从客户端的角度来看，超时必须考虑网络上的预期传输延迟，以确定“合理的”响应时间。

这种传输延迟对于交换式 Ethernet 可能是毫秒级，对于广域网连接可能是数百毫秒级。

反过来，客户端用于发起应用重试的任何“超时”时间应大于预期的最大“合理的”响应时间。如果不遵循这一点，可能会导致目标设备或网络过度拥塞，进而可能导致更多错误。这是一种应始终避免的特征。

因此在实践中，高性能应用中使用的客户端超时总是可能在某种程度上依赖于网络拓扑和预期的客户端性能。非时间关键的应用通常可以将超时值留给正常的 TCP 默认值，在大多数平台上，这将在几秒钟后报告通信失败。

### 4.4.2 MODBUS 服务器

图 14：MODBUS 服务器单元

MODBUS 服务器的角色是提供对远程 MODBUS 客户端的应用对象和服务的访问。

根据用户应用程序，可以提供不同类型的访问：

- 简单访问，如获取和设置应用对象属性
- 高级访问，以触发特定的应用服务

MODBUS 服务器必须：

- 将应用对象映射到可读和可写的 MODBUS 对象上，以便获取或设置应用对象属性。
- 提供一种在应用对象上触发服务的方式。

在运行时，MODBUS 服务器必须分析接收到的 MODBUS 请求，处理所需操作，并发送回 MODBUS 响应。

提示：后端接口的应用对象和服务根据功能码获取请求数据，用户负责。

#### 4.4.2.1 MODBUS 服务器设计

MODBUS 服务器设计取决于：

- 对应用对象的访问类型（对属性的简单访问或对服务的高级访问）
- MODBUS 服务器与用户应用程序之间的交互类型（同步或异步）

以下活动图描述了服务器为从 TCP 管理获取 MODBUS 请求，然后分析请求、处理所需操作并发送回 MODBUS 响应而处理的主要操作。

图 15：处理 MODBUS 指示活动图

如前面的活动图所示：

- 某些服务可以由 MODBUS 服务器本身立即处理，无需与用户应用程序直接交互；
- 某些服务可能还需要显式地与用户应用程序交互才能处理；
- 某些其他高级服务需要调用称为 MODBUS 后端服务的特定接口。例如，可以根据用户应用级协议使用一系列多个 MODBUS 请求/响应事务来触发用户应用程序服务。后端服务负责正确处理所有单个 MODBUS 事务，以执行全局用户应用程序服务。

以下各节给出了更完整的描述。

MODBUS 服务器可以接受同时服务多个 MODBUS 请求。服务器可以接受的同时 MODBUS 请求最大数量是 MODBUS 服务器的主要特性之一。此数量取决于服务器设计及其处理和内存能力。此实现参数称为“NumberMaxOfSeverTransaction”，必须作为 MODBUS 服务器特性之一进行描述。根据设备能力，其值可以从 1 到 16。

“NumberMaxOfTransaction”参数显著影响 MODBUS 服务器的行为和性能。特别是，需要注意的是，并发 MODBUS 事务的管理数量可能影响服务器对 MODBUS 请求的响应时间。

### 4.4.2.2 MODBUS PDU 检查

以下图描述了 MODBUS PDU 检查活动。

图 16：MODBUS PDU 检查活动图

MODBUS PDU 检查功能首先解析 MBAP 头部。必须检查协议标识符字段：

- 如果它与 MODBUS 协议类型不同，则简单丢弃该指示。
- 如果正确（= MODBUS 协议类型；值 0x00），则实例化 MODBUS 事务。

服务器可以实例化的 MODBUS 事务最大数量由“NumberMaxOfTransaction”参数（系统或配置参数）定义。

在没有更多可用事务的情况下，服务器构建 MODBUS 异常响应（异常码 6：服务器忙）。

如果有可用的 MODBUS 事务，则对其进行初始化以记忆以下信息：

- 用于发送指示的 TCP 连接标识符（由 TCP 管理提供）
- MODBUS 事务 ID（在 MBAP 头部中给出）
- 单元标识符（在 MBAP 头部中给出）

然后解析 MODBUS PDU。首先检查功能码：

- 如果无效，则构建 MODBUS 异常响应（异常码 1：非法功能）。
- 如果功能码被接受，服务器启动 MODBUS 服务处理活动。

### 4.4.2.3 MODBUS 服务处理

图 17：MODBUS 服务处理活动图

所需 MODBUS 服务的处理可以根据设备软件和硬件架构以不同方式完成，如以下示例所述：

- 在紧凑设备或单线程架构中，MODBUS 服务器可以直接访问用户应用程序数据，所需服务可以由服务器本身“本地”处理，而无需调用后端服务。处理根据 MODBUS 应用协议规范 [1] 完成。如果出现错误，则构建 MODBUS 异常响应。
- 在模块化多处理器设备或多线程架构中，“通信层”和“用户应用层”是 2 个独立实体，某些简单服务可以完全由通信实体处理，而其他一些可能需要使用后端服务与用户应用程序实体协作。

为了与用户应用程序交互，MODBUS 后端服务必须实现所有适当的机制，以处理用户应用程序事务并正确管理用户应用程序调用及相关响应。

### 4.4.2.4 用户应用程序接口（后端接口）

在 MODBUS 后端服务中可以实现多种策略来完成其工作，尽管它们在用户网络吞吐量、接口带宽使用、响应时间甚至设计工作量方面并不等价。

MODBUS 后端服务将使用适当的接口与用户应用程序通信：

- 可以是基于串行链路的物理接口，或双端口 RAM 方案，或简单的 I/O 线路，或基于操作系统提供的消息服务的逻辑接口。
- 与用户应用程序的接口可以是同步的或异步的。

MODBUS 后端服务还将使用适当的设计模式来获取/设置对象属性或触发服务。在某些情况下，简单的“网关模式”就足够了。在其他情况下，设计者必须实现“代理模式”及相应的缓存策略，从简单的交换表历史到更复杂的复制机制。

MODBUS 后端服务有责任实现协议转写以便与用户应用程序交互。因此，它可能必须实现数据包分片/重组、数据一致性保证和同步机制（按需实现）。

### 4.4.2.5 MODBUS 响应构建

请求处理完成后，MODBUS 服务器必须使用适当的 MODBUS 服务器事务构建响应，并将其发送到 TCP 管理组件。

根据处理结果，可以构建两种类型的响应：

- 肯定 MODBUS 响应：

响应功能码 = 请求功能码

• MODBUS 异常响应：

目标是向客户端提供有关处理过程中检测到的错误的相关信息；

响应功能码 = 请求功能码 + 0x80；

提供异常码以指示错误原因。

| 异常码 | MODBUS 名称 | 说明 |
| --- | --- | --- |
| 01 | Illegal Function Code | 服务器不知道该功能码 |
| 02 | Illegal Data Address | 取决于请求 |
| 03 | Illegal Data Value | 取决于请求 |
| 04 | Server Failure | 服务器在执行过程中失败 |
| 05 | Acknowledge | 服务器接受了服务调用，但服务需要相对较长的时间来执行。因此，服务器仅返回服务调用接收的确认。 |
| 06 | Server Busy | 服务器无法接受 MB 请求 PDU。客户端应用程序负责决定是否以及何时重新发送请求。 |
| 0A | Gateway problem | 网关路径不可用 |
| 0B | Gateway problem | 目标设备未能响应。网关生成此异常 |

MODBUS 响应 PDU 必须以 MBAP 头部为前缀，该头部使用事务上下文中记忆的数据构建。

• 单元标识符

单元标识符按接收到的 MODBUS 请求中给出的值复制，并记忆在事务上下文中。

• 长度

服务器计算 MODBUS PDU 的大小加上单元标识符字节。此值设置在“长度”字段中。

• 协议标识符

协议标识符字段设置为 0x0000（MODBUS 协议），与接收到的 MODBUS 请求中给出的值相同。

• 事务标识符

此字段设置为与原始请求关联并记忆在事务上下文中的“事务标识符”值。

然后，必须使用事务上下文中记忆的 TCP 连接将 MODBUS 响应返回给正确的 MODBUS 客户端。发送响应后，必须释放事务上下文。

# 5 实现指南

本节的目标是提出消息服务实现的示例。下面描述的模型可以在 MODBUS 消息服务的客户端或服务器实现期间用作指南。

> 提示：消息服务实现由用户负责。

## 5.1 对象模型图

> 图 18：MODBUS 消息服务对象模型图

对象模型图由四个主要包组成：

- 配置层（Configuration layer）— 配置和管理其他包组件的操作模式
- TCP 管理（TCP Management）— 接口 TCP/IP 协议栈和通信应用层，管理 TCP 连接。它涉及套接字接口的管理。
- 通信应用层（Communication application layer）— 由一侧的 MODBUS 客户端和另一侧的 MODBUS 服务器组成。此包与用户应用程序链接。
- 用户应用程序 — 对应于设备应用程序，完全依赖于设备，因此不属于本规范。

此模型独立于实现选择，如操作系统类型、内存管理等。为了保证这种独立性，在 TCP 管理层和通信层之间以及通信层和用户应用层之间使用通用接口层。

用户可以实现此接口的不同实现：两个任务之间的管道、共享内存、串行链路接口、过程调用等。

定义以下实现模型需要做一些假设：

- 静态内存管理
- 服务器同步处理
- 一个任务处理所有套接字上的接收

### 5.1.1 TCP 管理包

> 图 19：MODBUS TCP 管理包

TCP 管理包包含以下类：

- CInterfaceConnexion：该类的角色是管理连接的内存池。
- CItemConnexion：该类包含描述连接所需的所有信息。
- CTCPConnexion：该类提供自动管理 TCP 连接的方法（接口套接字由 CStackTCP_IP 提供）。

- CConnexionMngt：该类管理所有连接，并通过 CInterfaceIndicationMsg 和 CInterfaceResponseMsg 向 MODBUS 服务器/MODBUS 客户端发送查询/响应。此类还处理连接打开的访问控制。
- CMBAP：该类提供读取/写入/分析 MODBUS MBAP 的方法。
- CStackTCP_IP：该类实现套接字服务并提供协议栈的参数化。

## 5.1.2 配置层包

图 20：MODBUS 配置层包

配置层包包含以下类：

- TConfigureObject：该类汇集了配置其他每个组件所需的所有数据。此结构由 COperatingMode 类的 m_Configure 方法填充。每个需要配置的类从此对象获取其自己的配置数据。配置数据依赖于实现，因此该类的属性列表仅作为示例提供。
- COperatingMode：该类的角色是填充 TConfigureObject（根据用户配置）并管理以下所述类的操作模式：
  - CMODBUSServer
  - CMODBUSClient
  - CConnexionMngt

## 5.1.3 通信层包

图 21：MODBUS 通信应用层包

通信应用层包包含以下类：

- CMODBUSServer：通过 CInterfaceIndicationMsg 类（通过 m_ServerReceivingMessage 方法）接收 MODBUS 查询。该类的角色是根据查询（来自网络）构建 MODBUS 响应或 MODBUS 异常。此类实现 MODBUS 服务器的状态图。只有在 COperatingMode 类发送了用户配置和正确的操作模式后，才能构建响应。
- CMODBUSClient：从 CInterfaceUserApplication 类读取 MODBUS 查询，客户端任务通过 m_ClientReceivingMessage 方法接收查询。此类实现 MODBUS 客户端的状态图，并管理事务以将查询与响应（来自网络）关联。只有在 COperatingMode 类发送了用户配置和正确的操作模式后，才能通过网络发送查询。
- CTransaction：该类实现用于管理事务的方法和结构。

### 5.1.4 接口类

- CInterfaceUserApplication：该类代表与用户应用程序的接口，它提供两种方法来访问用户数据。在实际实现中，此方法可以根据硬件和软件设备能力以不同方式实现（等同于终端驱动程序，例如访问 PCMCIA、共享内存等）。
- CInterfaceIndicationMsg：此接口类用于从网络向 MODBUS 服务器发送查询，以及为客户端从网络发送响应。此类接口 TCP 管理和“通信应用层”包（来自网络）。此类的实现依赖于设备。
- CInterfaceResponseMsg：此接口类用于从服务器接收响应以及从客户端向网络发送查询。此类接口“通信应用层”包和“TCP 管理”包（到网络）。此类的实现依赖于设备。

## 5.2 实现类图

以下类图描述了提案实现的完整图。

图 22：类图

## 5.3 序列图

此处描述的两个序列图是一个示例，用于说明客户端 MODBUS 事务和服务器 MODBUS 事务。

图 23：MODBUS 客户端序列图

为了更好地理解客户端序列图的通用注释：

第一步：来自用户应用程序的读取查询（m_Read 方法）。

第二步：“客户端”任务接收 MODBUS 查询（m_ClientReceivingMessage 方法）。这是客户端的入口点。为了在响应到达时将查询与相应响应关联，客户端使用事务资源（CTransaction 类）。MODBUS 查询通过调用类接口 CInterfaceResponseMsg（m_MODBUSRequest 方法）发送到 TCP 管理。

第三步：如果连接已建立，则在连接方面无需做任何事情，消息可以通过网络发送。否则，在消息可以通过网络发送之前，必须先打开连接。此时客户端正在等待响应（来自远程服务器）。

第四步：一旦从网络收到响应，TCP/IP 协议栈接收数据（隐式调用 m_EventOnSocket 方法）。如果连接已建立，则读取 MBAP 以检索连接对象（连接对象提供内存资源和其他信息）。来自网络的数据被读取，并通过 CInterfaceIndicationMsg 类接口（m_MODBUSConfirmation 方法）向客户端任务发送确认。客户端任务接收 MODBUS 确认（m_ClientReceivingResponse 方法）。最后，响应被写入用户应用程序（m_WriteData 方法），并释放事务资源。

以下是 MODBUS 服务器交换的示例。

图 24：MODBUS 服务器图

为了更好地理解服务器序列图的通用注释：

第一步：客户端已通过网络发送查询（MODBUS 查询）。TCP/IP 协议栈接收数据（隐式调用 m_EventOnSocket 方法）。

第二步：查询可能是也可能是不是连接请求（m_IsConnexionRequest 方法）。如果查询是连接请求，则分配连接对象和用于接收和发送 MODBUS 帧的缓冲区（m_GetObjectConnexion 方法）。紧接着，必须检查并接受连接访问控制（m_AcceptConnexion 方法）。

第三步：如果查询是 MODBUS 请求，则可以读取完整的 MODBUS 查询（m_ReceiveData 方法）。此时必须分析 MBAP（m_IsMdbHeaderCorrect 方法）。完整的帧通过 CInterfaceIndicationMsg 类（m_MODBUSIndication 方法）发送到服务器任务。服务器任务接收 MODBUS 查询（m_ServerReceivingMessage 方法）并对其进行分析。如果发生错误（不支持的功能码等），则构建 MODBUS 异常帧（m_BuildMODBUSException），否则构建响应。

第四步：响应通过 CInterfaceResponseMsg（m_MODBUSResponse 方法）通过网络发送。连接对象上的处理由 m_SendData 方法完成（检索连接描述符等），数据通过网络发送。

## 5.4 类和方法描述

### 5.4.1 MODBUS 服务器类

#### 类 CMODBUSServer

```
class CMODBUSServer
    Stereotype: implementationClass
    □□□□□□□□□□□ MODBUS □□□□□
```

字段摘要：

| 字段 | 类型 | 说明 |
| --- | --- | --- |
| GlobalState | protected char | MODBUS 服务器的状态 |

构造函数摘要：

| 构造函数 | 说明 |
| --- | --- |
| CMODBUSServer(TConfigureObject * lnkConfigureObject) | 构造函数：创建内部对象 |

方法摘要：

| 方法 | 返回类型 | 说明 |
| --- | --- | --- |
| m_InitServerFunctions(void) | protected void | 构造函数调用的函数，用于填充函数数组 m_ServerFunction |
| m_Reset(void) | bool | 重置服务器的方法，重置成功返回 true |
| m_ServerReceivingMessage(TItem Connexion * lnkMODBUS) | int | 与 CIndicationMsg::m_MODBUSIndication 的接口，用于从网络接收查询。如有问题返回负值 |
| m_Start(void) | bool | 启动服务器的方法，启动成功返回 true |
| m_Stop(void) | bool | 停止服务器的方法，停止成功返回 true |
| m_tServerMODBUS(void) | protected void | 服务器 MODBUS 任务 |

## 5.4.2 MODBUS 客户端类

### 类 CMODBUSClient

```
class CMODBUSClient
    Stereotype: implementationClass
    □□□□□□□□□□ MODBUS □□□□□
```

字段摘要：

| 字段 | 类型 | 说明 |
| --- | --- | --- |
| GlobalState | protected char | MODBUS 客户端的状态 |

构造函数摘要：

| 构造函数 | 说明 |
| --- | --- |
| CMODBUSClient(TConfigureObject * lnkConfigureObject) | 构造函数：创建内部对象，将变量初始化为 0 |

方法摘要：

| 方法 | 返回类型 | 说明 |
| --- | --- | --- |
| m_ClientReceivingMessage(TItemConnexion * lnkMODBUS) | int | 为从应用层接收消息提供的接口。典型：调用 CInterfaceUserApplication::m_Read 读取数据，调用 CInterfaceConnexion::m_GetObjectConnexion 获取事务内存。如有问题返回负值 |
| m_ClientReceivingResponse(TItemConnexion * lnkTItemConnexion) | int | 与 CIndicationMsg::m_Confirmation 的接口，用于从网络接收响应。如有问题返回负值 |
| m_Reset(void) | bool | 重置组件的方法，重置成功返回 true |
| m_Start(void) | bool | 启动组件的方法，启动成功返回 true |
| m_Stop(void) | bool | 停止组件的方法，停止成功返回 true |
| m_tClientMODBUS(void) | protected void | 客户端 MODBUS 任务 |

## 5.4.3 接口类

### 5.4.3.1 接口指示类

### 类 CInterfaceIndicationMsg

□□□□□□□: CConnexionMngt
class CInterfaceIndicationMsg
    Stereotype: interface
    □□□ TCP_Management □ MODBUS □□□□□□□□□□□□□□□

方法摘要：

| 方法 | 返回类型 | 说明 |
| --- | --- | --- |
| m_MODBUSConfirmation(TItemConnexion * lnkObject) | int | 接收传入响应的方法，调用客户端：可以通过引用、消息队列、远程过程调用等 |
| m_MODBUSIndication(TItemConnexion * lnkObject) | int | 读取传入 MODBUS 查询并调用服务器的方法：可以通过引用、消息队列、远程过程调用等 |

### 5.4.3.2 接口响应类

## 类 CInterfaceResponseMsg

```
□□□□□□□: CMODBUSClient, CMODBUSServer
class CInterfaceResponseMsg
    Stereotype: interface
    □□□□□□□□□□□□ TCP_Management □□□□□□□□□□
```

方法摘要：

| 方法 | 返回类型 | 说明 |
| --- | --- | --- |
| m_GetMemoryConnexion(unsigned long IPDest) | TItemConnexion * | 从内存池获取 ITemConnexion 对象。如果没有足够的内存则返回 -1 |
| m_MODBUSRequest(TItemConnexion * lnkCMODBUS) | int | 将传入 MODBUS 查询客户端写入 ConnexionMngt 的方法：可以通过引用、消息队列、远程过程调用等 |
| m_MODBUSResponse(TItemConnexion * lnkObject) | int | 将 MODBUS 服务器的响应写入 ConnexionMngt 的方法：可以通过引用、消息队列、远程过程调用等 |

### 5.4.4 连接管理类

#### 类 CConnexionMngt

```
class CConnexionMngt
    Stereotype: implementationClass
    □□□□ TCP □□□□
```

字段摘要：

| 字段 | 类型 | 说明 |
| --- | --- | --- |
| GlobalState | protected char | 组件 ConnexionMngt 的全局状态 |
| NbConnectionSupported | int | 连接的全局数量 |
| NbLocalConnection | int | 本地客户端向远程服务器打开的连接数 |
| NbRemoteConnection | int | 远程客户端向本地服务器打开的连接数 |

构造函数摘要：

| 构造函数 | 说明 |
| --- | --- |
| CConnexionMngt(TConfigureObject * lnkConfigureObject) | 构造函数：创建内部对象，将变量初始化为 0 |

方法摘要：

| 方法 | 返回类型 | 说明 |
| --- | --- | --- |
| m_EventOnSocket(void) | int | 唤醒 |
| m_IsConnectionAuthorized(unsigned long IPAddress) | bool | 如果新连接被授权则返回 true |
| m_ReceiveData(TItemConnexion * lnkConnexion) | int | 与 CTCPConnexion::write 方法的接口，用于从网络读取数据。如有问题返回负值 |
| m_Reset(void) | bool | 重置 ConnectionMngt 组件的方法，重置成功返回 true |
| m_SendData(TItemConnexion * lnkConnexion) | int | 与 CTCPConnexion::read 方法的接口，用于向网络发送数据。如有问题返回负值 |
| m_Start(void) | bool | 启动 ConnectionMngt 组件的方法，启动成功返回 true |
| m_Stop(void) | bool | 停止组件的方法，停止成功返回 true |