[MODBUS]

[杨庆伟][2013-7-29][V1.0]

1．版本信息

编写人	Xxx	审核人	（整理，检查，测试，注释）
应用归属	通讯		
软件信息	（版本，库）	硬件信息	（硬件型号）
其他			
版本信息	修改内容		修改人
V1.00	创建		

2 通讯原理介绍

2．1 基本信息

Modbus 通讯协议，是 Modicon PLC 所指定的资料交换通讯接口标准，于 1979 年首先制定串行通信标准（含 Modbus 异步及 Modbus Plus 同步，在本文档中只讨论 Modbus 异步通讯），于 1997 年制定网络通信标准（Modbus/TCP）。Modbus 通讯协议属于 OSI 所定义的通信层次的第七层应用层（Application Layer）。其通讯形式为 Client/Server 或者称为 Master/Slave。

由于 Modbus 协议只在 OSI 层次结构的应用层中做出定义，因此，Modbus 既可运行在 RS232, RS485, RS422 上，也可运行在以太网路上。本文档将分别从串行通讯和以太网通讯这两方面来说明 Modbus 的通讯原理，并且介绍在贝加莱系统平台上如何现 Modbus 通讯。

2．2 通讯原理

Modbus 的通讯方式是 Master/Slave（主/从）方式，在串行总线上 Modbus 只能以单主多从的方式进行通讯，而在以太网路上 Modbus 的通讯方式可以支持多主多从。

在 Modbus 网路中，一定要有一方扮演 Master，并主动发送 Query Message 给对应的 Slave 方。Slave 一旦收到消息后，马上根据 Query Message 中的内容准备 Response Message 并回送给 Master。Master 是以地址号来区分不同的 Slave，因此在同一个 Modbus 网路中不允许存在拥有相同地址号的 Slave。

一般情况下，当 PLC 和仪表、电表、执行机构等设备通讯时，PLC 可作为 Master；而当 PLC 与 HMI 设备（触摸屏）或者上位计算机进行通讯时，PLC 通常作为 Slave。

2．3 通讯协议

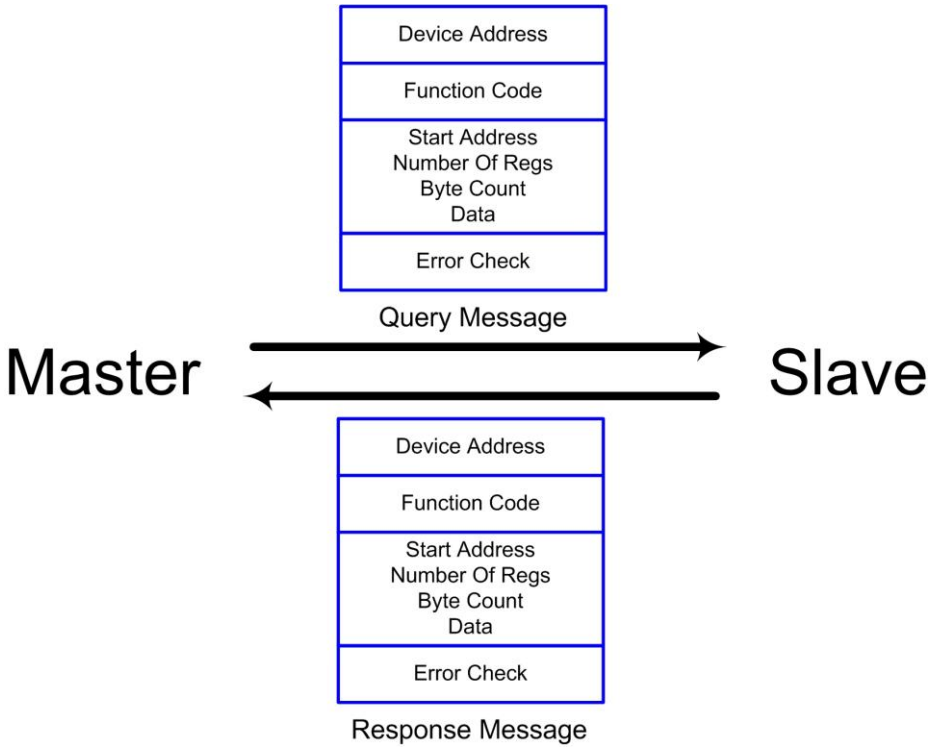

图 2-1 Modbus 消息结构

如图 2-1 所示，为每一次 Master 和 Slave 之间的询问/响应消息的基本内容。

- Device Address：Slave 的地址号，1～255
- Function Code：功能码

Modbus 规定，在 Slave 中必须拥有 4 种最基本的寄存器，即：

Coils：可读可写开关量寄存器，长度为 1 BIT；地址编号以 0 开始，如 00003 代表第 3 个寄存器地址。

Input Status：只读开关量寄存器，长度为 1 BIT；地址编号以 1 开始，如 10005 代表第 5 个寄存器地址。

Input Registers：只读模拟量寄存器，长度为 1 WORD；地址编号以 3 开始，如 30105 代表第 105 个寄存器地址。

Holding Registers：可读可写模拟量寄存器，长度为 1 WORD；地址编号以 4 开始，如 41105 代表第 1105 个寄存器地址。

每一条功能码包含两层意思，即，要访问的寄存器类型（如 Input Registers）和操作的类型（读或者写）。以下表格罗列出了一些最常用的 Modbus 访问指令：

Modbus Function Code	说明
01	Read Coils Status
02	Read Input Status
03	Read Holding Registers
04	Read Input Registers
05	Force Single Coil
06	Preset Single Holding Register
15	Force Multiple Coils
16	Preset Multiple Holding Registers

表 2-1 Modbus 常用指令表

- Start Address 和 Number Of Points/Registers：这两个内容结合在一起，代表 Master 需要操作 Slave 寄存器的起始地址以及寄存器数量。
- Byte Count：写入或读出寄存器数据的总长度。
- Data：写入或读出寄存器的数据。
- Error Check：校验数据，通常有 CRC 校验（Modbus RTU）或者 LRC 校验（Modbus ASCII）。由于在 TCP/IP 层已经存在校验机制，因此，Modbus TCP 无需校验字段。

从通讯方式说，Modbus 通讯属于轮询方式。因此，Slave 站点越多，使用的寄存器类型越多或者需要操作的目标寄存器越不连续，Master 所需要的访问指令也就越多，通讯效率会随之降低。因此，在建立 Slave 寄存器时要尽可能地使用一片连续的地址。

2．4 拓扑方式

2．4．1 Modbus 串行通讯

Modbus 串行通讯可以选择 RS232、RS485 或者 RS422，根据选择的不同，其拓扑方式略有不同。

- RS232：只能进行点对点通讯，即，只能有一个 Master 和一个 Slave。

- RS485：能进行点对多通讯，即一个 Master 可以带多个 Slave。其拓扑方式为总线式，最大的通信距离约为 1219m，最大传输速率为 10Mbps，传输速率与传输距离成反比，在 100Kb/S 的传输速率下，才可以达到最大的通信距离，如需传输更长的距离，需要加 485 中继器。RS-485 总线一般最大支持 32 个节点，如果使用特制的 485 芯片，可以达到 128 个或者 256 个节点，最大的可以支持到 400 个节点

- RS422：能进行点对多通讯，即一个 Master 可以带多个 Slave。其拓扑方式为总线式，最大的通信距离约为 1219m，最大传输速率为 10Mbps，传输速率与传输距离成反比，在 100Kb/S 的传输速率下，才可以达到最大的通信距离，只有在很短的距离下才能获得最高速率传输。一般 100 米长的双绞线上所能获得的最大传输速率仅为 1Mb/s。RS-422 需要一终端电阻，要求其阻值约等于传输电缆的特性阻抗。在短距离传输时可不需终端电阻，即一般在 300 米以下不需终端电阻。终端电阻接在传输电缆的最远端。

2．4．2 Modbus 以太网通讯

Modbus 以太网通讯可以选用一般的以太网线缆或者光纤进行通讯。通讯速度可达 100Mbps，拓扑方式通常采用星形或者树形结构。通常使用 Modbus TCP 或者 Modbus UDP（本文档不做讨论）来实现 Modbus 以太网通讯。可以实现多主，多从的通讯方式。

2．5 通讯帧结构

最基本的 Modbus 通讯帧可以分为 Modbus RTU，Modbus ASCII 和 Modbus TCP，这三

种通讯帧在格式上略有不同。

2．5．1 Modbus RTU

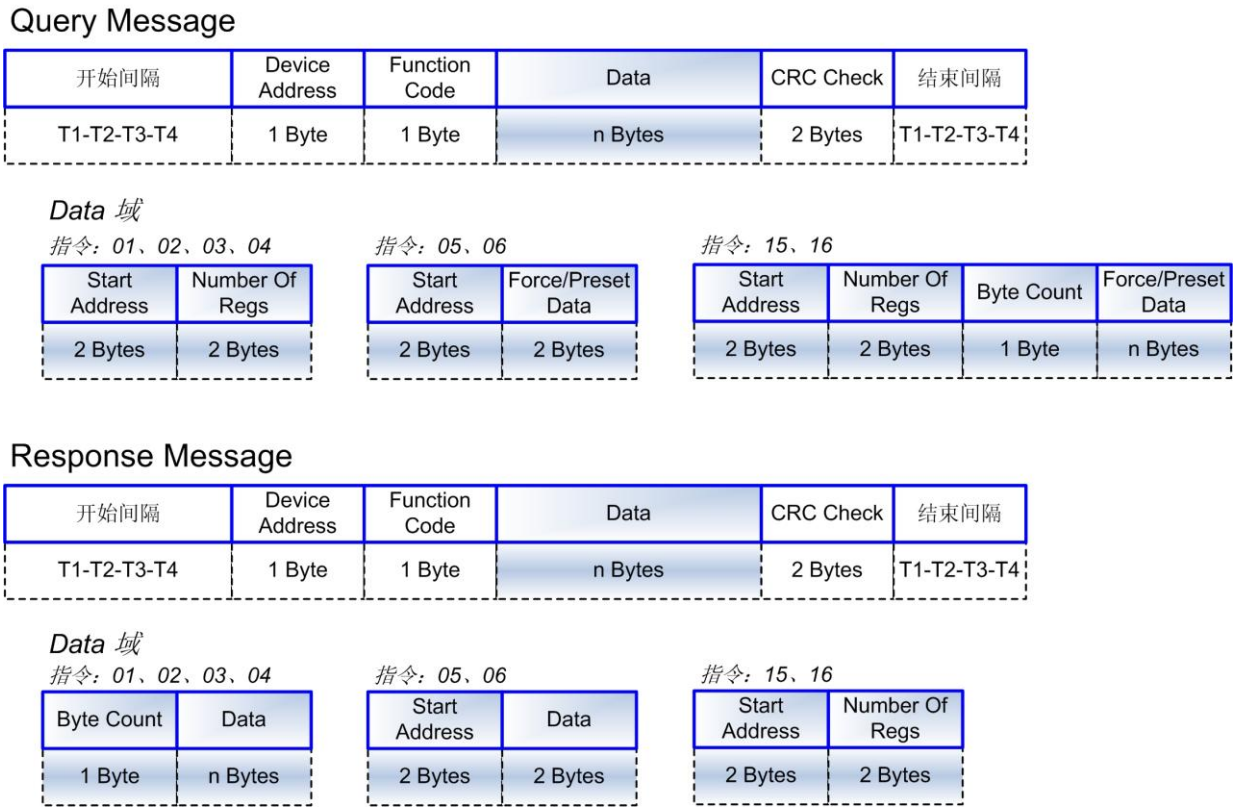

图 2-2 Modbus RTU 帧结构

- 开始/结束间隔：RTU 规定每次 Query 或 Response Message 的结束，是以未再接到下一个字符间隔时间来判断。规定为 3.5 字符的通信时间。一般在报文中不会被截获。
- Device Address：Slave 的站号，占用 1 个字节。
- Function Code：功能码，占用 1 个字节。
- Start Address：访问寄存器的起始地址，用 2 个字节来表示，如第 1372 号地址，表示成 0x 055C。
- Number Of Regs：操作寄存器的个数，占用 2 个字节，如需要操作总共 1000 个寄存器，表示成 0x03E8。
- Byte Count：占用 1 个字节，表示数据区长度。
- Data：读取或者写入寄存器的数值，每个开关量的信息只占用 1 Bit。
- CRC Check：Modbus RTU 采用 CRC 16 进行校验。

2．5．2 Modbus ASCII

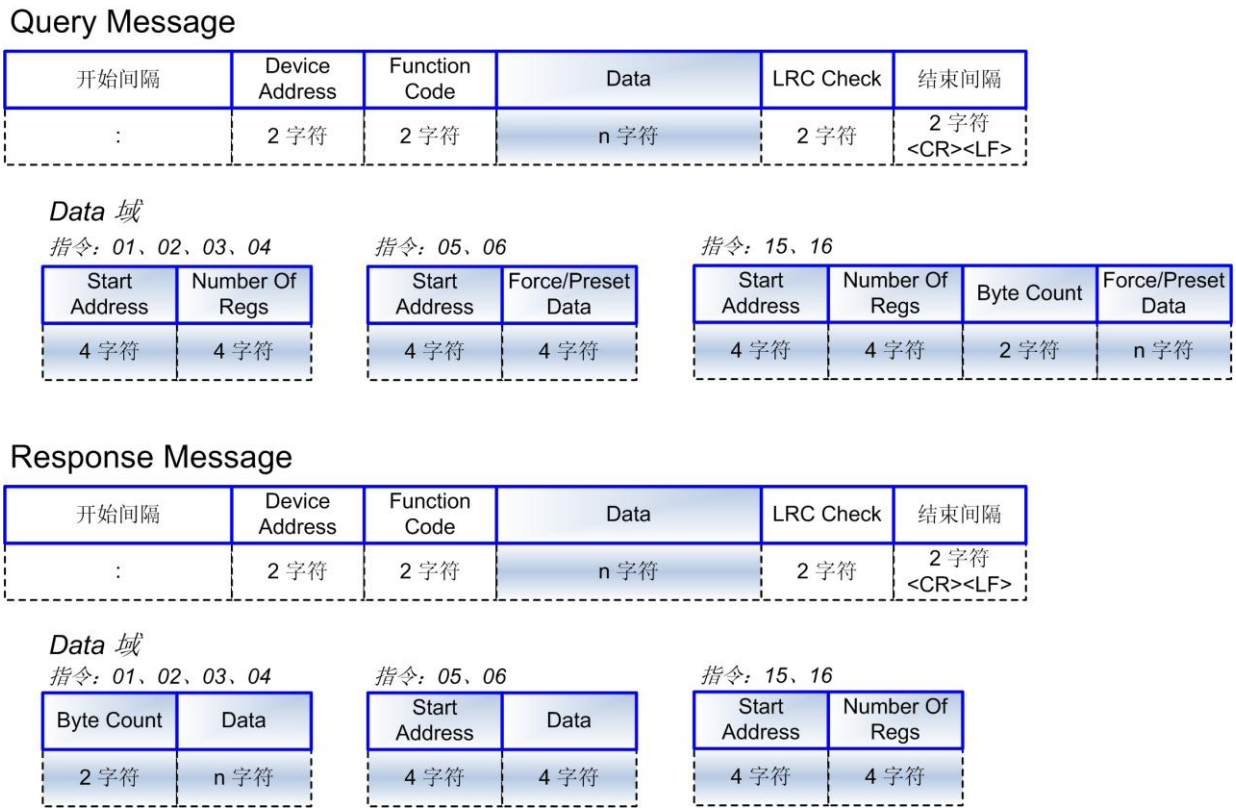

图 2-3 Modbus ASCII 帧结构

Modbus ASCII 与 Modbus RTU 比较，不同之处在于，Modbus ASCII 中所有内容均表示成 ASCII 字符。例如 Function Code 为 0F（Force Multiple Coils）时，在 Modbus RTU 中表示成 16 进制数 0x0F，只占 1 个字节；而在 Modbus ASCII 中，表示成字符‘0’和字符‘F’，用 16 进制数表示成 0x00 和 0x46，占用 2 个字节。

- 开始/结束间隔：Modbus ASCII 中的开始和结束字符分别以 ASCII 码中的‘:’以及<CR><LF>（回车换行）来表示。
- Device Address：Slave 的站号，2 个 ASCII 字符，占用 2 个字节。
- Function Code：功能码，2 个 ASCII 字符，占用 2 个字节。
- Start Address：访问寄存器的起始地址，用 4 个 ASCII 字符来表示，如第 1372 号地址，换算成 16 进制数为 0x 055C，表示成‘0’‘5’‘5’‘C’。
- Number Of Regs：操作寄存器的个数，用 4 个 ASCII 字符来表示，如需要操作总共 1000 个寄存器，换算成 16 进制数为 0x03E8，表示成‘0’‘3’‘E’‘8’。
- Byte Count：用 1 个 ASCII 字符表示，表示数据区长度。
- CRC Check：Modbus ASCII 采用 LRC 进行校验。

2．5．3 Modbus TCP

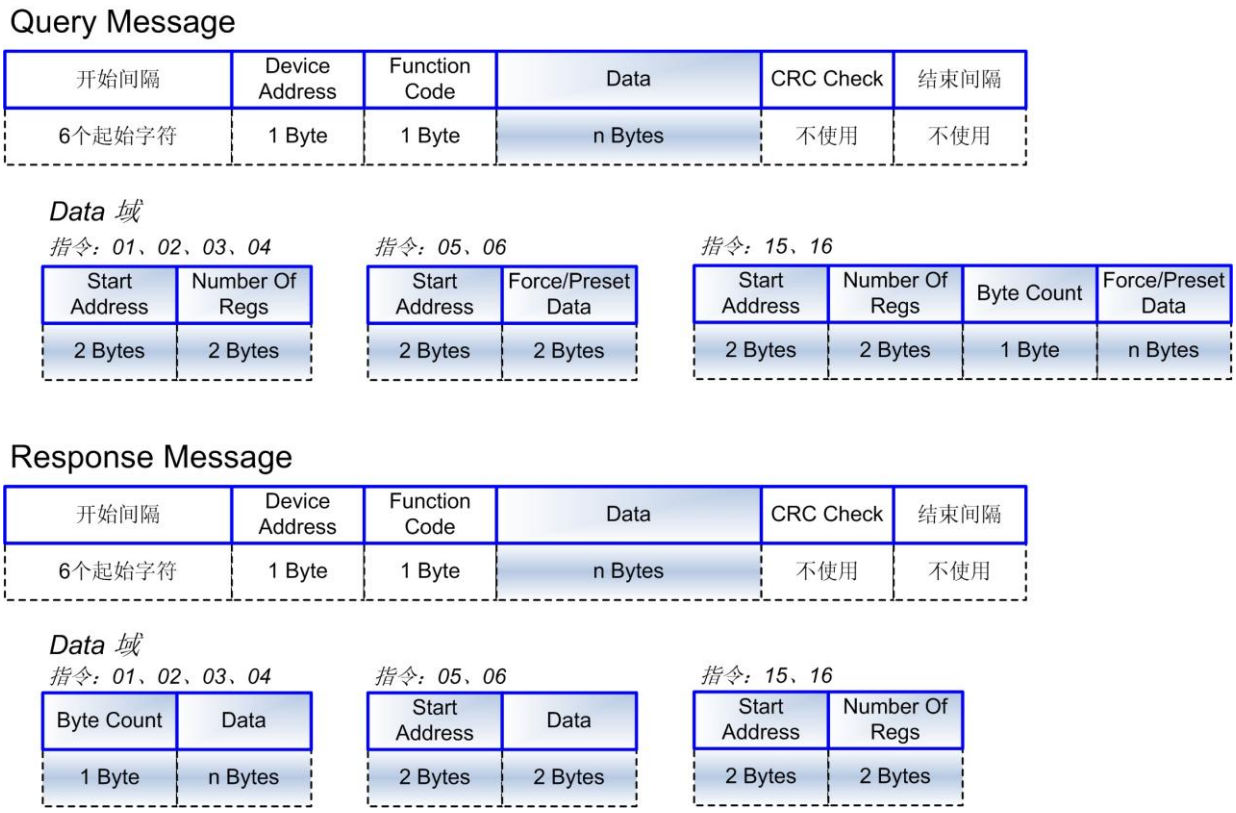

图 2-4 Modbus TCP 帧结构

Modbus TCP 与 Modbus RTU 比较，不同之处只体现在开始字符与校验，其余部分都一样。

- 开始/结束间隔：Modbus TCP 的起始字符用 6 个 Byte 的数字表示，以定义一些 TCP/IP 的需要系数。说明如下：

  Byte0 和 Byte1：用两个字节的内容组成本次通信 Message 的编号，以区分每次 Message。一般由 Master 进行编号，Slave 则将传来的 Message 照样通过 Response Message 回传至 Master。

  Byte2 和 Byte3：通信协议识别号码。

  Byte4 和 Byte5：Message 长度（由 Device Address 至 Data 为止），长度不能超过 256。

- Device Address：Slave 的站号，占用 1 个字节。
- Function Code：功能码，占用 1 个字节。
- Start Address：访问寄存器的起始地址，用 2 个字节来表示，如第 1372 号地址，表示成 0x 055C。
- Number Of Regs：操作寄存器的个数，占用 2 个字节，如需要操作总共 1000 个寄存器，表示成 0x03E8。
- Byte Count：占用 1 个字节，表示数据区长度。
- Data：读取或者写入寄存器的数值，每个开关量的信息只占用 1 Bit。

3 B&R 软硬件支持

3.1 Modbus RTU/ASCII

基于串口的 Modbus 通讯可以运行在 RS232、RS422 或者 RS485 总线上。因此对于所有含有这些通讯接口的通讯卡都能够支持 Modbus 串口通讯。

如果使用 SG4/SG3 CPU，可以选用标准库中的 DRV_mbus 功能库来实现通讯。

如果使用 SGC CPU 需要注意以下几点：

- RS485/RS422

  若要使用 DRV_mbus 实现基于 RS485/RS422 总线的 Modbus 通讯，**只能**选用总线型 CPU+IF 通讯的方案。在所有 SGC 的 CPU 上，若使用 CS 通讯模块来进行 Modbus 通讯，DRV_mbus 将无法得到支持。

- RS232

  SGC CPU 本身自带的 RS232 口也可作为 Modbus 通讯口，并且可以使用 DRV_mbus 功能库。在必要时，可以选购第三方的 RS232 转 RS485 模块，来实现基于 RS485 的 Modbus 通讯。

3.2 Modbus TCP

Modbus TCP 可以运行在任何带有以太网口的 CPU 或通讯卡上。在同一个以太网口上，Modbus TCP 可以与其他以太网通讯协议共存。例如，在同一个通讯口上可以实现 Pvi、IMA 和 Modbus TCP 的通讯，但是会影响通讯效率。

4 AS 中使用

在 AS 中实现 Modbus 通讯主要通过编写程序（Modbus TCP Master 除外）。

4．1 使用 DRV_mbus 实现 Modbus RTU/ASCII

注意：DRV_mbus 只能支持 Stream 通讯模式。

4．1．1 Modbus RTU/ASCII Master

在作为 Master 时，标准库中提供了两种方式实现，即 MBMaster()或者 MBMCmd()。以下分别介绍这两种功能块的实现区别：

a. MBMaster()

➢ 建立一个 Data Object 文件，在文件中配置需要执行的 Master 指令，格式如下：

```
"LibDRVmbus.EventPV4", 16, $01, "LibDRVmbus.LocalPV4", 0000, 0003
```

"LibDRVmbus.EventPV4"：事件变量，用户可以自行定义。如果是 Global 变量，直接在双引号中填写变量名称，如果是 Local 变量，需要在变量名称前

加上任务名称并以“:”分隔，如"Task1:testVar"。关于事件变量说明如下：

i. 只有当事件变量的值为 TRUE 时，才执行当前指令。指令执行完后，此事件变量的值自动清 0（FALSE），无需在程序中设定。
ii. 如果需要一直执行当前指令，可以在程序中一直对事件变量赋值为 TRUE，或者在双引号中不填写任何变量，即""。

16：Function Code（指令码），使用 MBMaster()功能函数时，只支持 01、02、03、04、05、06、15 和 16 指令。

$01: Slave 站号，”$”代表是十六进制数，也可以直接填写十进制数。

"LibDRVmbus.LocalPV4"：PLC 中的变量。即，从 Slave 中读取的数值会存储到用户所指定的 PLC 变量中，或者将 PLC 变量的值传送个 Slave 中对应寄存器，用户可自行定义此变量。如果是 Global 变量，直接在双引号中填写变量名称，如果是 Local 变量，需要在变量名称前加上任务名称并以“:”分隔，如"Task1:testVar"。进一步说明如下：

i. 变量的类型需要与访问的 Slave 寄存器保持一致。例如，使用 01、02、05 或 15 指令时，对应寄存器是 BOOL 类型，则 PLC 变量也应当是 BOOL 类型
ii. 操作多个寄存器时，需要建立数组予以对应。例如，操作 5 个 Holding Register，则在 PLC 中需要建立一个 INT 或者 UINT 类型的数组，数组长度至少是 5。
iii. 如果操作的 PLC 变量从第一个数组元素开始，可以按两种方式进行填写："LibDRVmbus.LocalPV4"或"LibDRVmbus.LocalPV4[0]"。如果操作的 PLC 变量不是从第一个数组元素开始，则必须填写数组下标，如："LibDRVmbus.LocalPV4[3]"。

0000：操作寄存器的起始地址，至少要填写 4 位数字，最大值为 65535。要注意的是，在贝加莱 PLC 中操作寄存器的起始地址是以 0 开始的，而很多 Slave 设备寄存器起始地址是以 1 开始的，需要换算。

0003：操作寄存器的个数，至少要填写 4 位数字，对于不同指令，操作寄存器的最大个数也不同（RTU 模式）：

i. 03、04 指令访问的最大寄存器数是 125
ii. 01、02 指令访问的最大寄存器数是 2000
iii. 15 指令操作的最大寄存器数是 1968
iv. 16 指令操作的最大寄存器数是 123

- 编写程序
  执行 MBMOpen 进行通讯端口初始化、指定 Modbus 指令文件的名称、指定 Modbus 通讯的方式（RTU 或者 ASCII）

b. MBMCmd()
- MBMCmd()函数也可完成与 MBMaster()函数一样的功能。所不同的是，MBMCmd()无需指定一个 DataObject 作为指令存储表，而且功能块本身能支持

几乎所有的 Modbus 指令。

例程 ： 此例程中包含了 MBMaster()和 MBMCmd()这两种功能块的使用方式。

DataObject 中的内容（文件名为 datamod）

```
;--------------------

; Modbus command file

;--------------------

"EventPV1 ", 01, $01, "LocalPV1 ", 0000, 0005

"EventPV2 ", 02, $01, "LocalPV2 ", 0000, 0003
```

初始化程序：

```
S2:=1;

EventPV1:=1;

EventPV2:=1;

LocalPV1:=1;

LocalPV2:=1;

MBMOpen_xx.enable:=1;

MBMOpen_xx.pDevice:=ADR('SL0.SS1.IF1');

MBMOpen_xx.pMode:=ADR('RS232,9600,E,8,1');

MBMOpen_xx.pConfig:=ADR('datamod');

MBMOpen_xx.timeout:=2000;

MBMOpen_xx.ascii:=0;

MBMOpen_xx();
```

循环程序：

```
EventPV1:=1;

EventPV2:=1;
```

```
IF(S1=0)THEN

    MBMaster_xx.enable:=1;

    MBMaster_xx.ident:=MBMOpen_xx.ident;

    MBMaster_xx();

ELSE

    MBMCmd_xx.enable:=1;

    MBMCmd_xx.ident:=MBMOpen_xx.ident;

    MBMCmd_xx.mfc:=1;

    MBMCmd_xx.node:=1;

    MBMCmd_xx.data:=ADR(LocalPV1);

    MBMCmd_xx.offset:=0;

    MBMCmd_xx.len:=5;

    MBMCmd_xx();

END_IF

IF(S2=0)THEN

    MBMClose_xx.enable:=1;

    MBMClose_xx.ident:=MBMOpen_xx.ident;

    MBMClose_xx();

END_IF
```

变量声明：

Name	Data Type	Scope
EventPV1	BOOL	Global
EventPV2	BOOL	Global
LocalPV1	BOOL	Global
LocalPV2	BOOL	Global

MBMOpen_xx	MBMOpen	Local
MBMaster_xx	MBMaster	Local
MBMClose_xx	MBMClose	Local
MBMCmd_xx	MBMCmd	Local
S1	BOOL	Local
S2	BOOL	Local

4．1．2　Modbus RTU/ASCII Slave

使用标准库来实现 Modbus RTU/ASCII Slave 功能非常方便，可按照如下方式实现：

- 执行 MBSOpen()函数进行初始化
  当贝加莱 PLC 作为 Modbus Slave 时可以由两种方式指定 Modbus Slave 的寄存器地址：
  i. 在 Global 域中按照以下方式声明称固定格式的变量名

变量名称（数组）	数据类型	对应 Modbus 寄存器类型	支持的 Modbus 访问指令
MB0[x]	BOOL, USINT, SINT	Coil Status	1, 5, 15
MB1[x]	BOOL, USINT, SINT	Status	2
MB3[x]	UINT, INT	Input register	4
MB4[x]	UINT, INT	Holding register	3, 6, 16

ii. 在 MBSOpen()函数的如下接口中指定 PLC 变量

I/O	Parameter	Data Type	Description
IN	pCoilStat	UDINT	Coil Status (Modbus simulation variable)
IN	pInputStat	UDINT	Input Status (Modbus simulation variable)
IN	pInputReg	UDINT	Input Register (Modbus simulation variable)
IN	pHoldingReg	UDINT	Holding Register (Modbus simulation variable)

在指定的接口处，将对应变量的名称以字符串的方式进行连接。

例如：

```
_LOCAL    Coils[5]          /* instead of _GLOBAL MB0[5] */

MBSOpen_ptr.pCoilStat = "Coils";
```

- 执行 MBSlave()函数

例程：

初始化程序：

```
S1:=0;

MB0[0]:=0;

MB0[1]:=1;

MB0[2]:=1;

MB0[3]:=0;

MB0[4]:=1;

MB1[0]:=0;

MB1[1]:=1;

MB1[2]:=0;

MBSOpen_xx.enable:=1;

MBSOpen_xx.pDevice:=ADR('SL0.SS1.IF1');

MBSOpen_xx.pMode:=ADR('RS232,9600,E,8,1');

MBSOpen_xx.own_ID:=1;

MBSOpen_xx.timeout:=2000;

MBSOpen_xx();
```

循环程序

```
MBSlave_xx.enable:=1;

MBSlave_xx.ident:=MBSOpen_xx.ident;

MBSlave_xx();

IF(S1=1)THEN
```

```
    MBSClose_xx.enable:=1;

    MBSClose_xx.ident:=MBSOpen_xx.ident;

    MBSClose_xx();

END_IF
```

变量声明：

Name	Data Type	Scope
MB0	BOOL[5]	Global
MB1	BOOL[3]	Global
MBSOpen_xx	MBSOpen	Local
MBSlave_xx	MBSlave	Local
MBSClose_xx	MBSClose	Local
S1	BOOL	Local

4．2　使用 MbusTCP 库实现 Modbus TCP

4．2 .1 Modbus TCP Master

Modbus TCP Master 对应的是 TCP Client，必须主动连接作为 TCP Server 的 Modbus TCP Slave。在 MbusTCP 库中，用 MBClient()功能块实现 Modbus TCP Master 通讯。

MBClient()功能块参数：

I/O	名称	数据类型	描述
IN	enable	BOOL	功能块使能。
IN	port	UINT	Master(TCP Client)的端口，用户可以自己指定。
IN	server_ip_addr	STRING	Modbus TCP Slave (TCP Server)的 IP 地址。
IN	p_cfg	modbus_client_cfg_typ	指向 modbus_client_cfg_typ 类型的结构体指针，此结构体将存放 Modbus 操作指令。
IN	receive_timeout	UDINT	指令发出后，主站等待从站回应的超时

			时间，默认值为 1000ms(1s)。
OUT	status	UINT	功能块返回的状态值。

modbus_client_cfg_typ 结构体参数：

结构体成员	数据类型	描述
action_enable	modbus_client_action_enable_typ[20]	其中有 single 和 cyclic 两个成员变量： single：当前指令将立即执行 cyclic：每间隔一段用户指定的时间执行一次
action_param	modbus_client_action_param_typ[20]	拥有如下成员： type： Modbus 功能代码，支持 01、02、03、04、05、06、15 和 16 start_addr：寄存器起始地址，以 0 为起始地址 quantity：操作寄存器的个数 p_pv：PLC 中变量的地址 timer：当设定 cyclic 时，指定执行指令的间隔时间，单位为 ms

注意：

- 若要实现在同一台 PLC 中与多个 Modbus TCP Slave 进行通讯，则要建立多个 MBClient 实例。如 MBClient_01、MBClient_02…，每个 MBClient 实例的 port 必须**不相同**。
- 对于一个 MBClient 实例来说，最多只能拥有 20 条 Modbus 指令。

例程：

如下例程将会用 MBclient 功能块实现 Modbus TCP 主站（Master）功能，在 Master 端将循环执行 Modbus 指令，并将访问的数据存放于用户指定的变量中。在 Master 端将循环执行 04 以及 03 指令（分别是读取 input registers 以及 output registers 中的值），并将值存放到变量数组中去。04 指令的间隔时间为 1000ms(1s)，03 指令的间隔时间为 1500ms(1.5s), 主站发出指令后等待从站应答的超时时间为 2000ms(2s)。

初始化程序：

```
memcpy(adr(ipStr), adr("10.86.13.80"), 18)

Para_MbTCP.action_enable[0].cyclic=1

Para_MbTCP.action_enable[0].single=0

Para_MbTCP.action_param[0].type=$04

Para_MbTCP.action_param[0].start_addr=$00
```

```
Para_MbTCP.action_param[0].quantity=$5

Para_MbTCP.action_param[0].p_pv= adr(AI_Data)

Para_MbTCP.action_param[0].timer=1000



Para_MbTCP.action_enable[1].cyclic=1

Para_MbTCP.action_enable[1].single=0

Para_MbTCP.action_param[1].type=$03

Para_MbTCP.action_param[1].start_addr=$00

Para_MbTCP.action_param[1].quantity=$8

Para_MbTCP.action_param[1].p_pv= adr(AO_Data)

Para_MbTCP.action_param[1].timer=1500
```

循环程序：

```
MBclient_0.enable= 1

MBclient_0.port= 502

MBclient_0.server_ip_addr=ipStr

MBclient_0.p_cfg= adr(Para_MbTCP)

MBclient_0.receive_timeout= 2000

MBclient_0 FUB MBclient()
```

变量声明：

Name	Data Type	Scope
Para_MbTCP	modbus_client_cfg_typ	Local
AI_Data	INT[10]	Local
AO_Data	INT[10]	Local
ipStr	STRING(18)	Local
MBclient_0	MBclient_typ	Local

4．2 .2 Modbus TCP Slave

Modbus TCP Slave 对应的是 TCP Server，当接收到来自 Modbus TCP Master (TCP Client) 发出的请求后，Modbus TCP Slave 会建立一条连接与当前的 Modbus TCP Master 进行通讯。在 MbusTCP 库中使用 MBServer()功能块来实现 Modbus TCP Slave。

MBServer()功能块参数：

I/O	名称	数据类型	描述
IN	enable	BOOL	功能块使能。
IN	p_cfg	modbus_server_cfg_typ	指向 modbus_server_cfg_typ 类型结构体的指针。贝加莱 PLC 的变量将通过此结构体转变成为 Modbus 寄存器
IN	TimeOut	UDINT	超时时间 当 Slave 检测到某条连接在一段时间内没有收到任何数据，其会自动关闭这条连接。单位是 10ms，默认超时时间为 30s。
IN	ipString	STRING	本机 IP 地址，若不填写该参数，则 Slave 将侦听所有通讯口，可以实现双网冗余。
OUT	status	UINT	功能块返回的状态值。

modbus_server_cfg_typ 结构体参数：

结构体成员	数据类型	描述
unit	modbus_server_unit_typ	将 PLC 变量转换成 Modbus 寄存器，其中含有以下成员： p_discrete_inputs：只读开关量寄存器 p_coils：读/写开关量寄存器 p_input_registers：只读模拟量寄存器 p_holding_registers：读/写模拟量寄存器

modbus_server_cfg_typ 结构体中，定义了每种寄存器的数量，由于不同 CPU 的内存有所限制，因此作为 Modbus TCP Slave 时，有两个版本的库：

- SG4
  每种寄存器的最大存储数量为 2000 个
- SGC/SG3

每种寄存器的最大存储数量为 500 个

实现 Modbus TCP Slave 功能时可以按照如下方式进行：

i. 声明 modbus_server_cfg_typ 类型的变量，例如声明变量 MBServerCfg

ii. 在程序的初始化部分填写 PLC 变量与 Modbus 寄存器的连接，例如：

- 将 PLC 变量 motorRun(BOOL 类型)和 motorStop(BOOL 类型)分别对应到 Modbus 地址为 10001 和 10002 的寄存器(只读开关量寄存器，Discrete Input Status)(ST 语言)：

```
MBServerCfg.unit.p_discrete_inputs[0] := ADR(motorRun);
MBServerCfg.unit.p_discrete_inputs[1] := ADR(motorStop);
```

- 将 PLC 变量 valveOpen (BOOL 类型)和 valuveClose(BOOL 类型)分别对应到 Modbus 地址为 00101 和 00102 的寄存器(读/写开关量寄存器，Coils)(ST 语言)：

```
MBServerCfg.unit.p_coils[100] := ADR(valveOpen);
MBServerCfg.unit.p_coils[101] := ADR(valveClose);
```

- 将 PLC 变量 step (INT 类型)、temperature(REAL 类型)和 status(DINT 类型)分别对应到 Modbus 地址为 30101、30102 和 30104 的寄存器(只读模拟量寄存器，Input Registers)(ST 语言)：

```
MBServerCfg.unit.input_registers[100] := ADR(step);

(*temperature 为 REAL 类型变量，长度为 2 words，因此需要两个寄存器来存储。分别取出 temperature 的首地址和首地址+2*)
MBServerCfg.unit.input_registers[101] := ADR(temperature);
MBServerCfg.unit.input_registers[102] := ADR(temperature) + 2;

(*status 为 DINT 类型变量，长度为 2 words，因此需要两个寄存器来存储。分别取出 DINT 的首地址和首地址+2*)
MBServerCfg.unit.input_registers[103] := ADR(status);
MBServerCfg.unit.input_registers[104] := ADR(status) + 2;
```

- 将 PLC 变量 mode (INT 类型)、setTemperature(REAL 类型)和 setCounter(DINT 类型)分别对应到 Modbus 地址为 40201、40202 和 40204 的寄存器(读/写模拟量寄存器，Holding Registers)(ST 语言)：

```
MBServerCfg.unit.holding_registers[200] := ADR(mode);

(*temperature 为 REAL 类型变量，长度为 2 words，因此需要两个寄存器来存储。分别取出 temperature 的首地址和首地址+2*)
MBServerCfg.unit. holding _registers[201] := ADR(setTemperature);
MBServerCfg.unit. holding _registers[202] := ADR(setTemperature) + 2;

(*status 为 DINT 类型变量，长度为 2 words，因此需要两个寄存器来存储。分别取出 DINT 的首地址和首地址+2*)
MBServerCfg.unit. holding _registers[203] := ADR(setCounter);
MBServerCfg.unit. holding _registers[204] := ADR(setCounter) + 2;
```

注意：

- 不同 CPU 需要选择不同版本的库（SG4 或者 SG3/SGC）
  SG4 库：每种类型的寄存器最多可容纳 2000 个对象。
  SG3/SGC 库：每种类型的寄存器最多可容纳 500 个对象。
- Modbus TCP Slave 同一时间最多可以连接 5 个对象。需要特别说明的是，任何一个 Modbus TCP Master 就是 1 个连接对象；如果同一个 Modbus TCP Master 在同一时间有两条网络（不同网段）连接在同一个 Modbus TCP Slave 上，则占用 2 个连接对象。
- 为了获得较快的 Slave 响应速度，应该将 Modbus TCP Slave 通讯程序放在等级较高的循环任务中执行（如 Cyclic#1 或 Cyclic#2），并将循环时间设得尽量短（如 SG4：2ms， SGC：10ms）。

例程：

以下程序会将 PLC 中的变量通过 MBTcpSlaveCfg 结构体转变成 Modbus 寄存器，并接受 Modbus TCP Master 的操作。例程中的 coils 以及 holding registers 可以双向操作，即，Modbus TCP Master 修改寄存器后，对应 PLC 变量的值会相应改变，反之，如果在 PLC 中更改了变量的值，Modbus TCP Master 中对应寄存器的值也会随之改变。

初始化程序：

```
(*Coils definiation*)

MBTcpSlaveCfg.unit.p_coils[0] := ADR(RIO1DO1Ch01);

MBTcpSlaveCfg.unit.p_coils[1] := ADR(RIO1DO1Ch02);

(*Discrete input status definiation*)

MBTcpSlaveCfg.unit.p_discrete_inputs[0] := ADR(RIO1DI1Ch01);

MBTcpSlaveCfg.unit.p_discrete_inputs[1] := ADR(RIO1DI1Ch02);

(*Input registers definiation*)

MBTcpSlaveCfg.unit.p_input_registers[0] := ADR(actValue);

MBTcpSlaveCfg.unit.p_input_registers[1] := ADR(actValue)+2;

MBTcpSlaveCfg.unit.p_input_registers[2] := ADR(node);

MBTcpSlaveCfg.unit.p_input_registers[3] := ADR(RMSHeart);

(*Holding registers definiation*)

MBTcpSlaveCfg.unit.p_holding_registers[0] := ADR(setValue);
```

```
MBTcpSlaveCfg.unit.p_holding_registers[1] := ADR(setValue) + 2;

MBTcpSlaveCfg.unit.p_holding_registers[2] := ADR(TestINT1);

MBTcpSlaveCfg.unit.p_holding_registers[3] := ADR(TestUINT1) + 2;
```

循环程序：

```
MBTcpSlave.enable := 1;
MBTcpSlave.p_cfg := ADR(MBTcpSlaveCfg);
MBTcpSlave.TimeOut := 500;  (*If there is no data received from client in 5 seconds, server will close the connection*)
MBTcpSlave();
```

变量声明：

Name	Data Type	Scope
MBTcpSlaveCfg	modbus_server_cfg_typ	Local
RIO1DO1Ch01	BOOL	Local
RIO1DO1Ch02	BOOL	Local
RIO1DI1Ch01	BOOL	Local
RIO1DI1Ch02	BOOL	Local
actValue	REAL	Local
node	UINT	Local
RMSHeart	INT	Local
setValue	REAL	Local
TestINT1	INT	Local
TestUINT1	UINT	Local
MBTcpSlave	MBServer	Local

Modbus TCP Slave 程序诊断

当 Modbus TCP Slave 程序在运行时，可以通过监控变量的方式对 Modbus TCP Slave 的运行状态进行检测和诊断。方法是，选择通讯程序，进入监控模式，将 MBServer 类型的变量添加进来，并查看此结构体中成员变量的值，如下图：

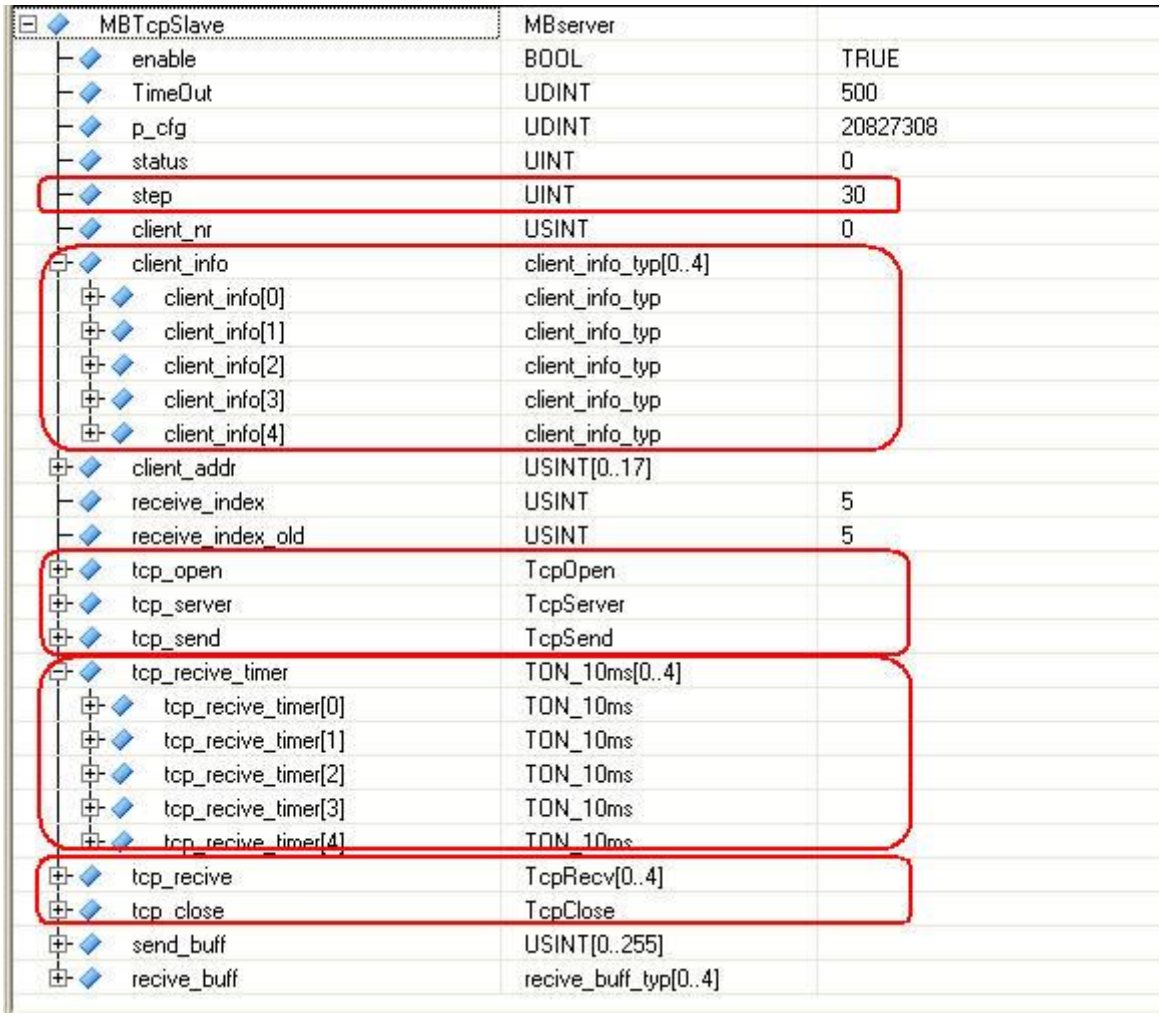

图 4-1 MBServer 结构体

其中“step”表示当前程序运行的步序号，运行正常时，步序号应为 20、30 或 40，如果出现错误，步序号会显示 250。此时，可通过进一步观察 tcp_open、tcp_server、tcp_send 或者 tcp_recive 中的 status 进一步分析错误产生的原因。引起步序 250 的原因大多是由于下载通讯程序后未对 CPU 重新启动，从而导致 tcp_server 出错。

client_info 中有 5 个成员，分别对应连接的 5 个 Client。假设有一个 Client 成功连接到当前 Slave，系统会自动分配一条空闲的链接与该 Client 进行通讯（假设是 2 号链接），并将 Client 的信息记录在对应 client_info 中（如 client_info[2]）（如 IP 地址，Client 的端口号等）。如果 client_info 中的信息为空，则表明该资源为空闲。若在监控时看到 5 个 client_info 中都有内容，则表明 5 个链接已全部占用，此时无法再接受新的 Modbus TCP Master。

针对每一个已连接的 client（Modbus TCP Master），结构体中都会有一个对应的 tcp_recive_timer。如果 Slave 没有收到来自某个已连接 Master 的任何消息，则对应 tcp_recive_timer 会开始计时（展开 tcp_recive_timer，会看到有变化的数字），一旦计时到设定的 Timeout 时间后，系统会自动释放当前链接（清空对应的 client_info）。

4．2 通过 AS 配置实现 Modbus TCP Master

若要实现 Modbus TCP Master 功能，除了编程，还可以通过在 AS3.0 中进行配置来实现。方法如下：

- 选择需要用来通讯的以太网口，如 IF2，并进入以太网口的配置界面。
- 在配置界面中（如下图），将“Active Modbus communication”功能激活，并点击“保存”

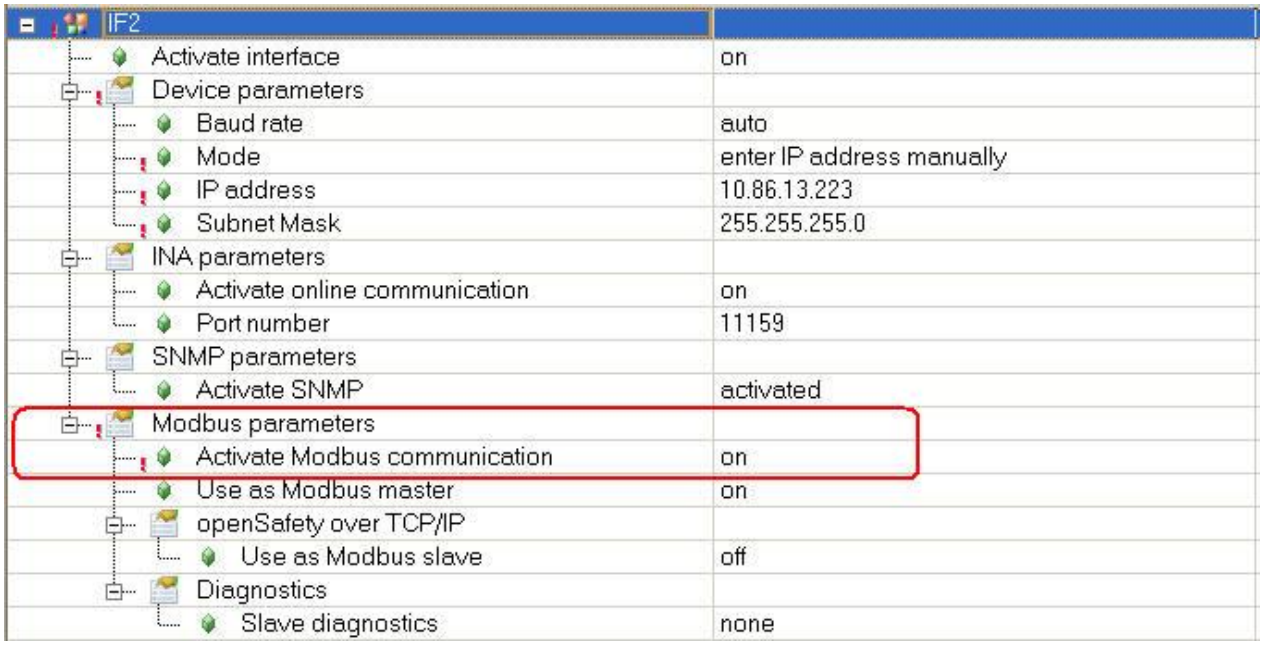

图 4-1 Modbus TCP Master 功能激活

- 在以太网接口下插入 Modbus TCP Slave，并输入 Slave 站号，默认为 1，如下图：

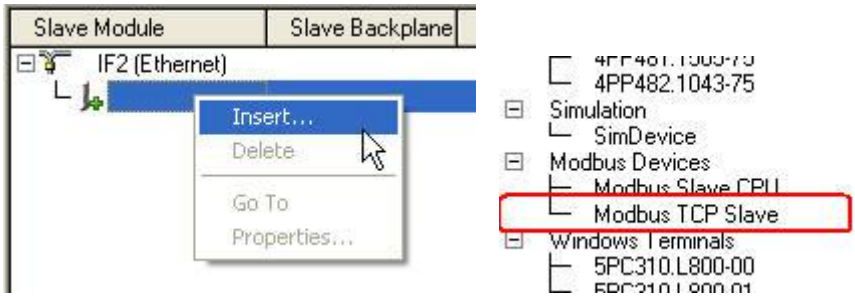

图 4-2 插入 Modbus TCP Slave

- 插入 Modbus TCP Slave 后，选择该设备，点击鼠标右键，选择“Open I/O Configuration”进入配置界面，如下图：

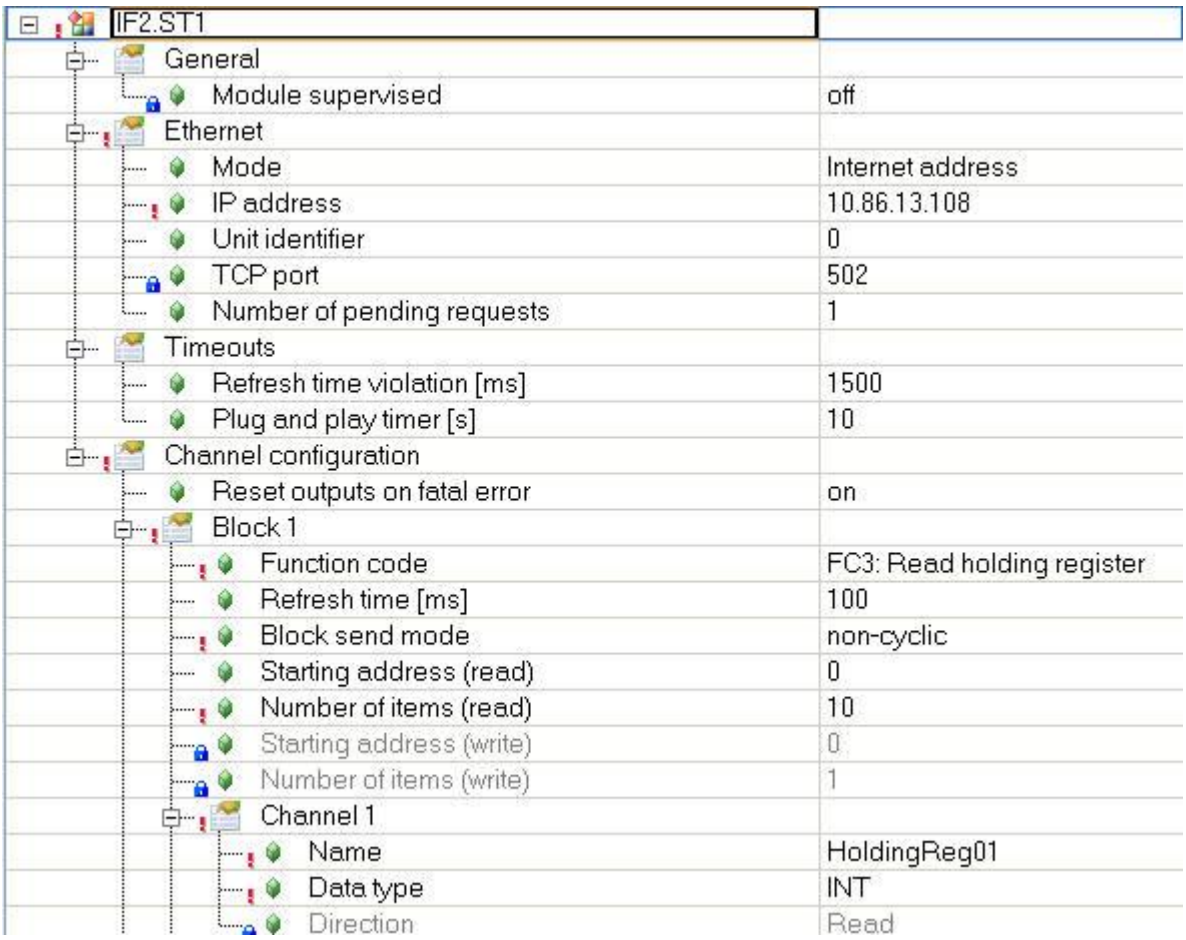

图 4-3 Modbus TCP Slave 配置

需要配置如下参数：

- IP address：Modbus TCP Slave （目标从站）的 IP 地址
- Function code：每一个 Block 中包含一条 Modbus 指令，用户可在 Function code 中配置需要的 Modbus 指令。
- Refresh time：本条指令的发送间隔时间。
- Block send mode：指令发送的方式，有如下 3 中方式：

i.Cyclic：指令循环重复发送

ii.Non cyclic：用户可以通过一个变量来控制是否重复发送当前指令，设成 non cyclic 模式后，可在 I/O Mapping 处见到如下 I/O 通道：

图 4-4 non cyclic 控制通道

当 DisableBlock01 通道值为 TRUE 时，该 Block 的 Modbus 指令将不会被发送，否则，循环发送。

iii.Once：指令只发送一次，配置成 once 模式后，在 I/O Mapping 处见到如下 I/O 通道：

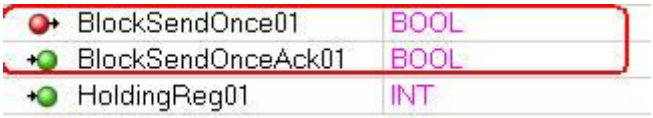

图 4-5 once 控制通道

若捕捉到 BlockSendOnce01 通道值的上升沿，本条指令将会被发送一次，成功发送后 BlockSendOnceAck01 通道将自动置为 TRUE。一旦检测到 BlockSendOnceAck01 为 TRUE 后，用户需要在程序中将 BlockSendOnce01 置为 False，以便为下一次指令的发送做准备。这一机制只对 05、06、15 以及 16 指令有效。对于其他基本指令（01、02、03、04），即使设置为 Once，该指令也会被不停循环发送。

- 配置 I/O 通道：配置通道名称及其数据类型，以便将 Modbus 寄存器与 PLC 变量相连接。

图 4-6 I/O 通道配置

- I/O 通道映射：配置完成后，再次选在 Modbus TCP Slave 设备后，在 I/O Mapping 中即可看到配置的通道，可以对其添加 PLC 变量，如下图：

图 4-7 I/O 通道映射

5 诊断工具

在对 Modbus 进行调试和诊断时，经常使用的调试软件有：

- Modbus Poll：Modbus RTU/ASCII/TCP Master 仿真软件。
- Modbus Slave：Modbus RTU/ASCII/TCP Slave 仿真软件。
- 串口调试助手：帮助截获 Modbus RTU/ASCII 的通讯帧。
- WireShark：帮助截获 Modbus TCP 的通讯帧。

当 Modbus 通讯不正常时，可以先使用仿真软件进行测试，从而缩小问题的排查范围。若不能找到问题，可使用截帧软件（串口调试助手或者 WireShark）截获通讯帧，进一步分析问题存在的原因。

以下对 Modbus Poll 和 Modbus Slave 两款仿真软件的使用做一个简单介绍

5.1 Modbus Poll

Modbus Poll 可用作 Modbus RTU/ASCII/TCP Master 的计算机仿真，可按如下步骤进行操作：

i.打开软件后，点击菜单栏中的“Connection”->选择“Connect”

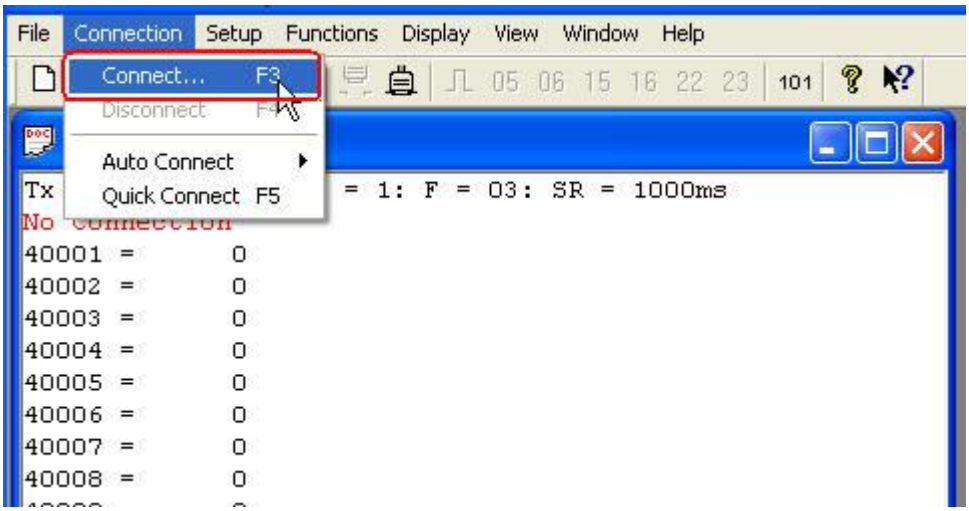

图 5-1 Modbus Poll Connect

ii.进入连接配置界面，如下图：

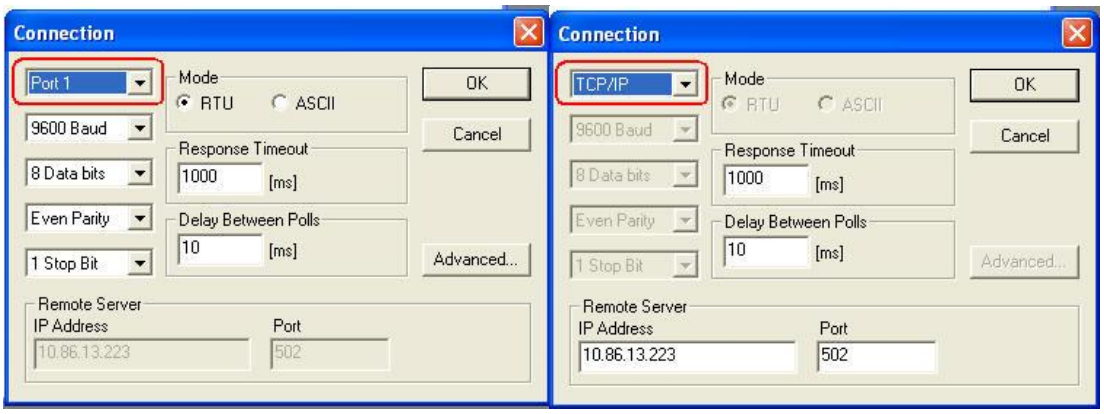

图 5-2 Modbus Poll 连接配置

在左上角可以选择通讯方式。如果是串口方式，选择 Port n，并对波特率，奇偶校验等通讯参数进行设置。如果是 TCP/IP 方式，选择 TCP/IP，并设定 Modbus TCP Slave（目标从站）的 IP 地址。

iii.配置完后出现通讯见面，选择菜单栏的“Setup”，配置需要操作的 Modbus 指令，如下图：

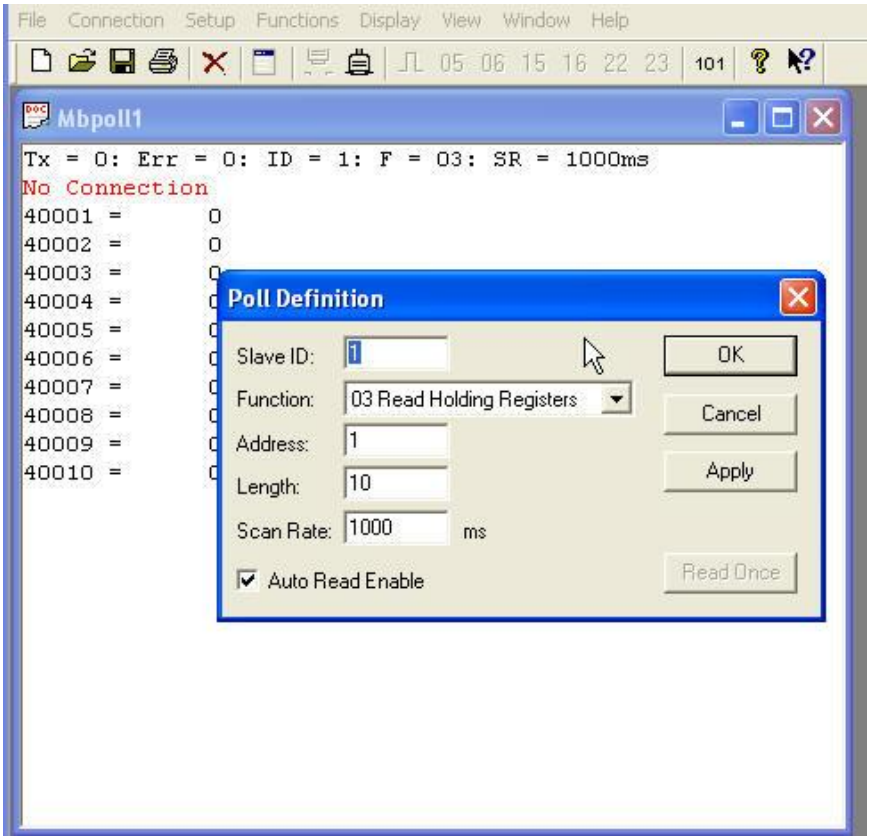

图 5-3 Modbus Poll 指令配置

在其中可以配置需要访问从站的节点号、操作的 Modbus 指令类型、操作寄存器的起始地址以及寄存器个数。

如果需要配置多条 Modbus 指令，可点击菜单栏中的“File”->“New”进行创建。

iv. 最后，点击菜单栏中的“Connection”->“Connect”进行连接。如果成功连接，在当前视图中不会出现任何红色字体，如“No Connection”或者“Response Timeout”等。

5.2 Modbus Slave

Modbus Slave 可用作 Modbus RTU/ASCII/TCP Slave 的计算机仿真，可按如下步骤进行操作：

i. 打开软件后，点击菜单栏中的“Connection”->选择“Connect”进入连接配置界面，如下图：

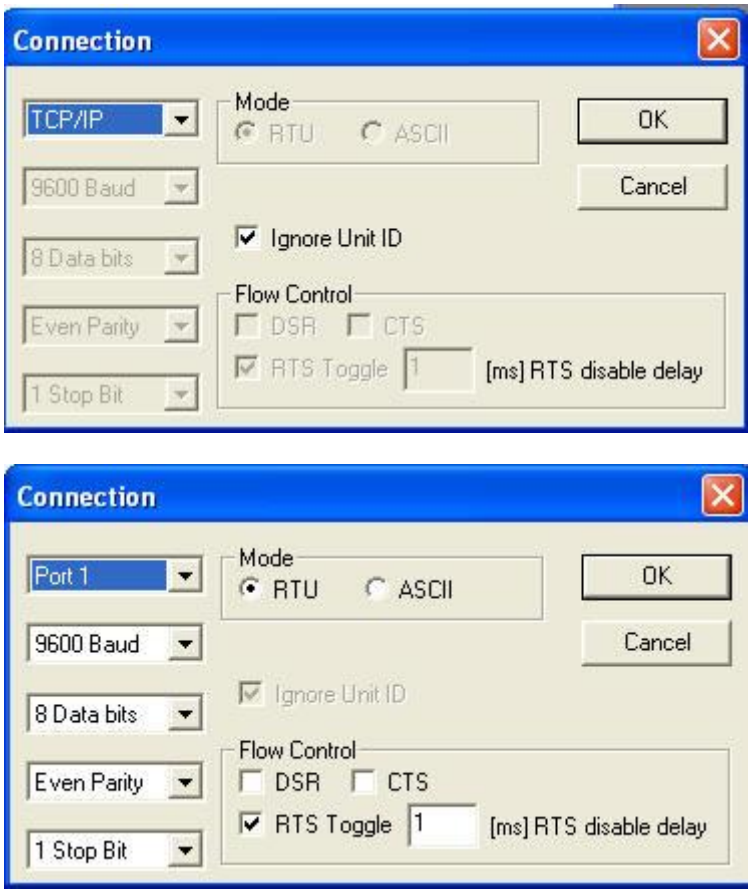

图 5-4 Modbus Slave 连接配置界面

Modbus Slave 与 Modbus Poll 的配置界面几乎一样，因此可以参考 Modbus Poll 的配置方式。所不同的是，在配置 TCP/IP 通讯方式时，无需配置 IP 地址。

ii. 点击菜单栏中“setup”->“Slave Definition”对 Slave 仿真其进行配置，如下图：

图 5-5 Modbus Slave 从站配置

在从站配置界面中可配置当前仿真 Slave 的节点号、寄存器类型、起始地址以及寄存器个数。如果需要配置不同类型的寄存器，可以反复点击菜单栏中“File”->“New”来进行创建并配置。