Modbus 协议栈

应用手册

作者：木南

版本：V2.0.0

目 录

第 1 章 前言

Modbus 是全球第一个真正用于工业现场的总线协议。Modbus 通讯在工业网络通讯中应用十分广泛，而且方便，受到大家的欢迎。

1.1、编写原因

一直以来，在我们自己的产品和项目中都多次使用 Modbus 通讯协议。每次都是使用者自行开发或者网上搜索符合要求的源码。但每次的应用都有不同，每次都需要很多的重复劳动。而且协议站如应用软件的紧密结合也使得代码有些混乱。所以一直以来都想要开发一个比较通用的协议栈能在后续的项目中复用，而不必每次都写一遍。现在利用项目研发的机会，开发一个自己的 Modbus 协议栈。

1.2、版权说明

本套软件最初虽然是为了我们自用而开发，但现已在 GitHub 上开源。所以任何人都可以复制、传播和使用，无论是个人学习还是商业应用都没有限制。对应用方法及修改欢迎探讨，但对于在使用过程中出现了任何问题我们均不负责。

软件源码及其说明可以上 GitHub 自由下载，下载地址：https://github.com/foxclever/Modbus

1.3、更新记录

前一个版本我们已经实现了基于串行链路的 Modbus RTU 和基于以太网的 Modbus TCP 两种。在这一个版本中，我们添加了基于串行链路的 Modbus ASCII

方式。并对上一个版本中已知的问题做了修正。

软件设计及本文档更新记录如下：

版本	更新说明	作者	日期
V1.0.0	初版，共 5 章，介绍了主要 API 函数	木南	2015 年 5 月 8 日
V1.1.0	添加了第 6、7 两章	木南	2016 年 12 月 23 日
V1.1.5	修改了 3、4、5 三章的大部分内容	木南	2018 年 2 月 20 日
V2.0.0	增加了 ASCII 相关内容，并修改已知问题	木南	2018 年 9 月 18 日
V2.1.0	增加了 RTU 和 ASCII 多主站操作的类容;增加了 TCP 客户端多网段操作的内容。	木南	2019 年 4 月 29 日

第 2 章、PDU 函数

在应用数据单元我们处理 Modbus 数据通讯的基本报文格式，这些基本是通用的，目前版本只支持 01、02、03、04、05、06、15、16 功能码。

4.1.1、功能码枚举

/*定义 Modbus 的操作功能码，支持 01、02、03、04、05、06、15、16 功能码*/

typedef enum {

```
    ReadCoilStatus=0x01,          /*读线圈状态（读多个输出位的状态）*/
    ReadInputStatus=0x02,         /*读输入位状态（读多个输入位的状态）*/
    ReadHoldingRegister=0x03,     /*读保持寄存器（读多个保持寄存器的数值）*/
    ReadInputRegister=0x04,       /*读输入寄存器（读多个输入寄存器的数值）*/
    WriteSingleCoil=0x05,         /*强制单个线圈（强制单个输出位的状态）*/
    WriteSingleRegister=0x06,     /*预制单个寄存器（设定一个寄存器的数值）*/
    WriteMultipleCoil=0x0F,       /*强制多个线圈（强制多个输出位的状态）*/
    WriteMultipleRegister=0x10    /*预制多个寄存器（设定多个寄存器的数值）*/
}FunctionCode;
```

4.1.2、Modbus 状态枚举

```
/*定义接收到指令检测错误时的错误码*/
typedef enum{
    MB_OK=0x00,
    InvalidFunctionCode=0x01,//不合法功能代码
    IllegalDataAddress=0x02,//非法的数据地址
    IllegalDataValues=0x03,//非法的数据值或者范围
    OperationFail=0x04
```

```
}ModbusStatus;
```

4.1.3、站信息结构

```
/*定义用于传递要访问从站（服务器）的信息*/

typedef struct{
   uint8_t unitID;
   FunctionCode functionCode;
   uint16_t startingAddress;
   uint16_t quantity;
}ObjAccessInfo;
```

定义用于传递从站（服务器）访问的各种信息，包括站号、功能码、起始地址和数量。

4.1.4、生成访问从站 PDU 单元函数

```
/*作为 RTU 主站（TCP 客户端）时，生成读写 RTU 从站（TCP 服务器）对象的命令*/

uint16_t GenerateReadWriteCommand(ObjAccessInfo objInfo,bool *statusList,uint16_t *registerList,uint8_t commandBytes[]);
```

参数：ObjAccessInfo objInfo，要访问的从站对象

bool *statusList，需要往下写的状态量对象数据列表

uint16_t *registerList，需要往下写的寄存器量对象的数据列表

uint8_t commandBytes[]，生成的访问命令

返回值：生成的访问命令的长度，以字节为单位。

4.1.5、解析读取的返回数据函数

/*解析主站（客户端）从服务器读取的数据*/

void TransformClientReceivedData(uint8_t * receivedMessage,uint16_t quantity,bool *statusList,uint16_t *registerLister);

参数：uint8_t * receivedMessage，接收到的信息报文

uint16_t quantity，读取的对象数量

bool *statusList，解析出来的读取到的状态对象的数据列表

uint16_t *registerLister，解析出来的读取到的寄存器对象的数据列表

返回值：无

4.1.6、从站生成读数据命令的响应报文函数

/*生成主站读访问的响应，包括 0x01、0x02、0x03、0x04 功能码,返回相应信息的长度*/

uint16_t GenerateMasterAccessRespond(uint8_t *receivedMesasage,bool *statusList,uint16_t *registerList,uint8_t *respondBytes);

参数：uint8_t *receivedMesasage，接收到的主战读操作报文

bool *statusList，读取的从站状态对象数据列表

uint16_t *registerList，读取到的从站寄存器对象数据列表

uint8_t *respondBytes，生成的从站响应报文信息

返回值：响应报文的长度，以字节为单位

4.1.7、功能码检验函数

/*检查功能码是否正确*/

ModbusStatus CheckFunctionCode(FunctionCode fc);

参数：FunctionCode fc，功能码，由枚举定义

返回值：Modbus 操作状态，由枚举定义

4.1.8、生成 TCP 客户端命令函数

//生成读写服务器对象的命令

uint16_t SyntheticReadWriteTCPServerCommand(ObjAccessInfo objInfo,bool *statusList,uint16_t *registerList,uint8_t *commandBytes);

参数：ObjAccessInfo objInfo，TCP 服务器访问信息对象

bool *statusList，写服务器状态量对象数据列表

uint16_t *registerList，写服务器寄存器量对象数据列表

uint8_t *commandBytes，生成的命令信息报文

返回值：命令的长度，以字节为单位

4.1.9、生成 TCP 服务器响应函数

/*合成对服务器访问的响应*/

uint16_t SyntheticServerAccessRespond(uint8_t *receivedMesasage,bool *statusList,uint16_t *registerList,uint8_t *respondBytes);

参数：uint8_t *receivedMesasage，接收到的客户端命令信息报文

bool *statusList，读 TCP 服务器的状态量对象数据列表

uint16_t *registerList，读 TCP 服务器的寄存器量对象数据列表

uint8_t *respondBytes，生成的响应客户端命令的响应信息报文

返回值：响应信息报文长度，以字节为单位

4.1.10、CRC 校验检查

/*通过 CRC 校验接收的信息是否正确*/

bool CheckRTUMessageIntegrity (uint8_t *message,uint8_t length);

参数：uint8_t *message，待校验的数据报文

uint8_t length，带教言的数据报文长度，以字节为单位

返回值：返回校验是否通过的状态信号

4.1.11、生成读写 RTU 从站的主站命令函数

/*生成读写从站数据对象的命令,命令长度包括 2 个校验字节*/

uint16_t SyntheticReadWriteSlaveCommand(ObjAccessInfo slaveInfo,bool *statusList,uint16_t *registerList,uint8_t *commandBytes);

参数：ObjAccessInfo slaveInfo，从站访问信息对象，由结构体定义

bool *statusList，写从站状态量数据列表

uint16_t *registerList，写从站寄存器量数据列表

uint8_t *commandBytes，生成的访问命令报文

返回值：访问命令的长度，以字节为单位

4.1.12、生成从站应答信息报文函数

/*生成从站应答主站的响应*/

uint16_t SyntheticSlaveAccessRespond(uint8_t *receivedMesasage,bool *statusList,uint16_t *registerList,uint8_t *respondBytes);

参数：uint8_t *receivedMesasage，

bool *statusList，

uint16_t *registerList，

uint8_t *respondBytes，

返回值：生成的响应信息报文的长度，以字节为单位

4.1.13、LRC 校验检查函数

/*判断 ASCII 数据信息是否正确*/

bool CheckASCIIMessageIntegrity(uint8_t *usMsg, uint16_t usLength)

参数：uint8_t *usMsg，待校验的消息列表，16 进制数据

uint16_t usLength，待校验的消息列表的长度

返回值：校验结果，校验正确返回“true”，否则返回“false”

4.1.14、将 ASCII 消息转换为 16 进制

/*接收到的 ASCII 消息转换为 16 进制*/

bool CovertAsciiMessageToHex(uint8_t *aMsg, uint8_t *hMsg, uint16_t aLen);

参数：uint8_t *aMsg，ASCII 码信息

uint8_t *hMsg，转换后的 16 进制信息

uint16_t aLen，ASCII 码信息的长度

返回值：转换是否正确，转换正确返回“true”，否则返回“false”

4.1.15、生成读写 ASCII 从站的命令函数

/*生成读写从站的命令，应用于主站，含校验及起始结束符*/

uint16_t SyntheticReadWriteAsciiSlaveCommand(ObjAccessInfo slaveInfo, bool *statusList, uint16_t *registerList, uint8_t *commandBytes)

参数：ObjAccessInfo slaveInfo，从站访问信息对象，由结构体定义

bool *statusList，写从站状态量数据列表

uint16_t *registerList，写从站寄存器量数据列表

uint8_t *commandBytes，生成的访问命令报文

返回值：访问命令的长度，以字节为单位

4.1.16、生成 ASCII 从站应答主站命令的函数

/*生成应答主站的响应，应用于从站*/

uint16_t SyntheticAsciiSlaveAccessRespond(uint8_t *receivedMessage, bool *statusList, uint16_t *registerList, uint8_t *respondBytes)

参数：uint8_t *receivedMesasage，接受到主站报文信息

bool *statusList，主站命令要读取的状态量值列表

uint16_t *registerList，主站命令要读取的寄存器量值列表

uint8_t *respondBytes，生成的响应信息报文

返回值：生成的响应信息报文的长度，以字节为单位

第 3 章、应用封装函数

对应用级封装，我们只封装 RTU 主站应用，RTU 从站应用、TCP 客户端应用和 TCP 服务器端应用四类。

4.2.1、RTU 主站对象与从站对象

```
/*  定义被访问 RTU 从站对象类型  */
typedef struct AccessedRTUSlaveType{
  uint8_t stationAddress;          //站地址
  uint8_t cmdOrder;                //当前命令在命令列表中的位置
```

```
    uint16_t commandNumber;          //命令列表中命令的总数
    uint8_t (*pReadCommand)[8];     //读命令列表
    uint8_t *pLastCommand;           //上一次发送的命令
    uint32_t flagPresetCoil;         //预置线圈控制标志位
    uint32_t flagPresetReg;          //预置寄存器控制标志位
}RTUAccessedSlaveType;

/*  定义本地 RTU 主站对象类型  */
typedef struct LocalRTUMasterType{
    uint32_t  flagWriteSlave[8];     //写一个站控制标志位，最多 256 个站，与站地址对应。
    uint16_t slaveNumber;            //从站列表中从站的数量
    uint16_t readOrder;              //当前从站在从站列表中的位置
    RTUAccessedSlaveType *pSlave;            //从站列表
    UpdateCoilStatusType pUpdateCoilStatus;          //更新线圈量函数
    UpdateInputStatusType pUpdateInputStatus;        //更新输入状态量函数
    UpdateHoldingRegisterType  pUpdateHoldingRegister;       //更新保持寄存
```

器量函数

UpdateInputResgisterType pUpdateInputResgister; //更新输入寄存器量函数

}RTULocalMasterType;

4.2.2、RTU 主站对象初始化函数

/*初始化 RTU 主站对象*/

void InitializeRTUMasterObject(RTULocalMasterType *master,uint16_t slaveNumber,RTUAccessedSlaveType *pSlave,UpdateCoilStatusType pUpdateCoilStatus,UpdateInputStatusType pUpdateInputStatus,UpdateHoldingRegisterType pUpdateHoldingRegister,UpdateInputResgisterType pUpdateInputResgister);

参数：RTULocalMasterType *master，本地主站对象

uint16_t slaveNumber，从站的数量

RTUAccessedSlaveType *pSlave，从站列表

UpdateCoilStatusType pUpdateCoilStatus，更新线圈量函数指针

UpdateInputStatusType pUpdateInputStatus，更新状态量函数指针

UpdateHoldingRegisterType pUpdateHoldingRegister，更新保持寄存器量函数指针

UpdateInputResgisterType pUpdateInputResgister，更新输入寄存器量函数指针

返回值：无返回值

4.2.3、生成 RTU 主站访问从站命令函数

/*生成访问服务器的命令*/

uint16_t CreateAccessSlaveCommand(ObjAccessInfo objInfo,void *dataList,uint8_t *commandBytes);

参数：ObjAccessInfo objInfo，从站访问信息对象，由结构体定义

void *dataList，写从站对象的数据列表

uint8_t *commandBytes，生成的命令报文

返回值：生成的访问命令的长度，以字节为单位

4.2.4、RTU 主站解析从站的响应函数

/*解析收到的服务器响应信息*/

void ParsingSlaveRespondMessage(uint8_t *recievedMessage,uint8_t *command);

参数：uint8_t *recievedMessage，接收到的返回信息

uint8_t *command，已经下发到从站的命令列表

返回值：无

4.2.5、RTU 主站解析从站返回消息命令匹配函数

/*接收到返回信息后，判断是否是发送命令列表中命令的返回信息*/

int FindCommandForRecievedMessage(uint8_t *recievedMessage,uint8_t (*commandList)[8],uint16_t commandNumber);

参数：uint8_t *recievedMessage，接收的从站返回信息

uint8_t (*commandList)[8]，主站下发过的命令列表

uint16_t commandNumber，存储的命令的条数

返回值：返回匹配命令所在的位置，无匹配时，返回“-1”

本函数只在 RTU 主站中，访问的数据量大且频繁，或者访问多个从站时使用，便于作正确的数据解析。

4.2.6、RTU 从站解析主站访问命令函数

/*解析接收到的信息，并返回合成的回复信息和信息的字节长度，通过回调函数*/

uint16_t ParsingMasterAccessCommand(uint8_t *receivedMesasage,uint8_t *respondBytes,uint16_t rxLength);

参数：uint8_t *receivedMesasage，接收到的主站报文信息

uint8_t *respondBytes，生成响应信息报文

uint16_t rxLength，接收到的报文的长度

返回值：生成的响应信息的长度，以字节为单位

4.2.7、TCP 客户端对象与 TCP 服务器对象

/* 定义被访问 TCP 服务器对象类型 */

typedef struct AccessedTCPServerType{

```
    union {
      uint32_t ipNumber;
      uint8_t ipSegment[4];
    }ipAddress;                                          //服务器的 IP 地
址
    uint32_t flagPresetServer;                           //写服务器请求标志
    WritedCoilListHeadNode pWritedCoilHeadNode;          //可写的线圈量列表
    WritedRegisterListHeadNode  pWritedRegisterHeadNode;  //可写的保持寄存器
列表
    struct AccessedTCPServerType *pNextNode;             //下一个 TCP 服务
器节点
}TCPAccessedServerType;

/* 定义服务器链表头类型 */
typedef struct ServerListHeadType {
    TCPAccessedServerType *pServerNode;                  //服务器节点指针
    uint32_t serverNumber;                               //服务器的数量
}ServerListHeadNode;

/* 定义本地 TCP 客户端对象类型 */
typedef struct LocalTCPClientType{
    uint32_t transaction;                                //事务标识符
    uint16_t cmdNumber;                                  //读服务器命令的
数量
    uint16_t cmdOrder;                                   //当前从站在从站
列表中的位置
    uint8_t (*pReadCommand)[12];                         //读命令列表
```

ServerListHeadNode ServerHeadNode; //Server 对象链表的头节点

UpdateCoilStatusType pUpdateCoilStatus; //更新线圈量函数

UpdateInputStatusType pUpdateInputStatus; //更新输入状态量函数

UpdateHoldingRegisterType pUpdateHoldingRegister; //更新保持寄存器量函数

UpdateInputResgisterType pUpdateInputResgister; //更新输入寄存器量函数

}TCPLocalClientType;

4.2.8、TCP 客户端对象初始化函数

/*初始化 TCP 客户端对象*/

void InitializeTCPClientObject(TCPLocalClientType *client,uint16_t cmdNumber,uint8_t (*pReadCommand)[12],UpdateCoilStatusType pUpdateCoilStatus,UpdateInputStatusType pUpdateInputStatus,UpdateHoldingRegisterType pUpdateHoldingRegister,UpdateInputResgisterType pUpdateInputResgister);

参数：TCPLocalClientType *client，客户端对象

uint16_t cmdNumber，读命令的数量

uint8_t (*pReadCommand)[12]，读命令列表

UpdateCoilStatusType pUpdateCoilStatus，更新线圈量函数指针

UpdateInputStatusType pUpdateInputStatus，更新状态量函数指针

UpdateHoldingRegisterType pUpdateHoldingRegister，更新保持寄存器函数指针

UpdateInputResgisterType pUpdateInputResgister，更新输入寄存器量函数指针

返回值：无返回值

4.2.9、生成 TCP 客户端访问服务器命令函数

/*生成访问服务器的命令*/

uint16_t CreateAccessServerCommand(ObjAccessInfo objInfo,void *dataList,uint8_t *commandBytes);

参数：ObjAccessInfo objInfo，访问服务器信息对象

void *dataList，些服务器对象数据列表

uint8_t *commandBytes，生成的访问服务器命令报文

返回值：访问命令的长度，以字节为单位

4.2.10、TCP 客户端解析服务器返回信息函数

/*解析收到的服务器相应信息*/

void ParsingServerRespondMessage(uint8_t *recievedMessage);

参数：uint8_t *recievedMessage，接受到的服务器响应信息

返回值：无

4.2.11、TCP 服务器解析客户端命令函数

/*解析接收到的信息，返回响应命令的长度*/

uint16_t ParsingClientAccessCommand(uint8_t *receivedMessage,uint8_t *respondBytes);

参数：uint8_t *receivedMessage，接受到的客户端访问命令

uint8_t *respondBytes，生成的服务器响应信息

返回值：生成的响应信息报文的长度，以字节为单位

4.2.12、ASCII 主站对象与从站对象

```
/*  定义被访问 ASCII 从站对象类型  */
typedef    struct AccessedASCIISlaveType{
   uint8_t stationAddress;            //站地址
   uint8_t cmdOrder;                  //当前命令在命令列表中的位置
   uint16_t commandNumber;            //命令列表中命令的总数
   uint8_t (*pReadCommand)[17];       //读命令列表
   uint8_t *pLastCommand;             //上一次发送的命令
```

```
    uint32_t flagPresetCoil;          //预置线圈控制标志位

    uint32_t flagPresetReg;           //预置寄存器控制标志位

}AsciiAccessedSlaveType;


/*  定义本地 ASCII 主站对象类型  */

typedef struct LocalASCIIMasterType{

    uint32_t  flagWriteSlave[8];     //写一个站控制标志位，最多 256 个站，与站地址对应。

    uint16_t slaveNumber;             //从站列表中从站的数量

    uint16_t readOrder;               //当前从站在从站列表中的位置

    AsciiAccessedSlaveType *pSlave;             //从站列表

    UpdateCoilStatusType pUpdateCoilStatus;          //更新线圈量函数

    UpdateInputStatusType pUpdateInputStatus;        //更新输入状态量函数

    UpdateHoldingRegisterType  pUpdateHoldingRegister;        //更新保持寄存器量函数

    UpdateInputResgisterType  pUpdateInputResgister;          //更新输入寄存器量函数

}AsciiLocalMasterType;
```

4.2.13、ASCII 主站对象初始化函数

/*初始化 RTU 主站对象*/

void InitializeASCIIMasterObject(AsciiLocalMasterType *master,uint16_t slaveNumber,AsciiAccessedSlaveType *pSlave,UpdateCoilStatusType pUpdateCoilStatus,UpdateInputStatusType pUpdateInputStatus,UpdateHoldingRegisterType pUpdateHoldingRegister,UpdateInputResgisterType pUpdateInputResgister);

参数：AsciiLocalMasterType *master，ASCII 主站对象

uint16_t slaveNumber，从站的数量

AsciiAccessedSlaveType *pSlave，从站列表

UpdateCoilStatusType pUpdateCoilStatus，更新线圈量回调函数

UpdateInputStatusType pUpdateInputStatus，更新状态量回调函数

UpdateHoldingRegisterType pUpdateHoldingRegister，更新保持寄存器量回调函数

UpdateInputResgisterType pUpdateInputResgister，更新输入寄存器量回调函数

返回值：无返回值

4.2.14、生成访问 ASCII 从站的命令

/*生成访问 ASCII 从站的命令*/

uint16_t CreateAccessAsciiSlaveCommand(ObjAccessInfo objInfo,void

*dataList,uint8_t *commandBytes);

参数：ObjAccessInfo objInfo，从站访问信息对象，由结构体定义

void *dataList，写从站对象的数据列表

uint8_t *commandBytes，生成的命令报文

返回值：生成的访问命令的长度，以字节为单位

4.2.15、ASCII 主站解析从站的响应函数

/*解析收到的服务器相应信息*/

void ParsingAsciiSlaveRespondMessage(uint8_t *recievedMessage, uint8_t *command,uint16_t rxLength);

参数：uint8_t *recievedMessage，接收到的返回信息

uint8_t *command，已经下发到从站的命令列表

uint16_t rxLength，接收到的消息的长度

返回值：无

4.2.16、ASCII 主站解析从站返回消息命令匹配函数

/*接收到返回信息后，判断是否是发送命令列表中命令的返回信息*/

int FindAsciiCommandForRecievedMessage(uint8_t *recievedMessage,uint8_t (*commandList)[17],uint16_t commandNumber);

参数：uint8_t *recievedMessage，接收的从站返回信息

uint8_t (*commandList)[17]，主站下发过的命令列表

uint16_t commandNumber，存储的命令的条数

返回值：返回匹配命令所在的位置，无匹配时，返回“-1”

本函数只在 RTU 主站中，访问的数据量大且频繁，或者访问多个从站时使用，便于作正确的数据解析。

4.2.17、ASCII 从站解析主站访问命令函数

/*解析接收到的信息，并返回合成的回复信息和信息的字节长度，通过回调函数*/

uint16_t ParsingAsciiMasterAccessCommand(uint8_t *receivedMessage, uint8_t *respondBytes, uint16_t rxLength, uint8_t StationAddress);

参数：uint8_t *receivedMesasage，接收到的主站报文信息

uint8_t *respondBytes，生成响应信息报文

uint16_t rxLength，接收到的报文的长度

返回值：生成的响应信息的长度，以字节为单位

第 4 章、数据处理函数

具体应用中还涉及到对数据对想的处理，不同的应用数据结构与处理方式千差万别，所以定义一些回调函数（实际使用弱化的函数__weak 关键字）用于数据的处理。

4.3.1、获取线圈量值的函数

/*获取想要读取的 Coil 量的值*/

```
__weak void GetCoilStatus(uint16_t startAddress,uint16_t quantity,bool *statusList)
{
  //如果需要 Modbus TCP Server/RTU Slave 应用中实现具体内容
}
```

参数：uint16_t startAddress，要读取对象的起始地址

uint16_t quantity，要读取对象的数量

bool *statusList，获取的线圈量的数据值列表

返回值：无

4.3.2、获取输入状态值的函数

/*获取想要读取的 InputStatus 量的值*/

__weak void GetInputStatus(uint16_t startAddress,uint16_t quantity,bool *statusValue)

{

//如果需要 Modbus TCP Server/RTU Slave 应用中实现具体内容

}

参数：uint16_t startAddress，要读取对象的起始地址

uint16_t quantity，要读取对象的数量

bool *statusValue，获取的输入状态量的数据值列表

返回值：无

4.3.3、获取保持寄存器值的函数

/*获取想要读取的保持寄存器的值*/

__weak void GetHoldingRegister(uint16_t startAddress,uint16_t quantity,uint16_t *registerValue)

{

```
  //如果需要 Modbus TCP Server/RTU Slave 应用中实现具体内容
}
```

参数：uint16_t startAddress，要读取的保持寄存器对象的起始地址

uint16_t quantity，要读取的保持寄存器的数量

uint16_t *registerValue，获取的保持寄存器的值

返回值：无

4.3.4、获取输入寄存器值的函数

```
/*获取想要读取的输入寄存器的值*/
__weak void GetInputRegister(uint16_t startAddress,uint16_t quantity,uint16_t *registerValue)
{
  //如果需要 Modbus TCP Server/RTU Slave 应用中实现具体内容
}
```

参数：uint16_t startAddress，要读取的输入寄存器的起始地址

uint16_t quantity，要读取的输入寄存器的数量

uint16_t *registerValue，获取的输入寄存器的值

返回值：无

4.3.5、设置单个线圈的值函数

/*设置单个线圈的值*/

```
__weak void SetSingleCoil(uint16_t coilAddress,bool coilValue)
{
  //如果需要 Modbus TCP Server/RTU Slave 应用中实现具体内容
}
```

参数：uint16_t coilAddress，所需要预置值的线圈地址

bool coilValue，所预置的值

返回值：无

4.3.6、设置单个寄存器值的函数

/*设置单个寄存器的值*/

```
__weak void SetSingleRegister(uint16_t registerAddress,uint16_t registerValue)
{
  //如果需要 Modbus TCP Server/RTU Slave 应用中实现具体内容
}
```

参数：uint16_t registerAddress，所预置的保持寄存器的地址

uint16_t registerValue，所预置的保持寄存器的值

返回值：无

4.3.7、设置多个线圈值的函数

/*设置多个线圈的值*/

__weak void SetMultipleCoil(uint16_t startAddress,uint16_t quantity,bool *statusValue)

{

//如果需要 Modbus TCP Server/RTU Slave 应用中实现具体内容

}

参数：uint16_t startAddress，要预置的多个线圈的起始地址

uint16_t quantity，所要预知的线圈的数量

bool *statusValue，要预知的线圈值的列表

返回值：无

4.3.8、设置多个寄存器值的函数

/*设置多个寄存器的值*/

__weak void SetMultipleRegister(uint16_t startAddress,uint16_t quantity,uint16_t *registerValue)

{

//如果需要 Modbus TCP Server/RTU Slave 应用中实现具体内容

}

参数：uint16_t startAddress，所预置的多个保持寄存器的起始地址

uint16_t quantity，所要预置的保持寄存器的数量

uint16_t *registerValue，预知的保持寄存器值的列表

返回值：无

4.3.9、从站更新获取到的线圈量值的函数

/*更新读回来的线圈状态*/

__weak void UpdateCoilStatus(uint16_t startAddress,uint16_t quantity,bool *stateValue)

{

//在客户端（主站）应用中实现

}

参数：uint16_t startAddress，读取值的线圈对象的起始地址

uint16_t quantity，所读取的线圈的数量

bool *stateValue，读取的线圈地址的列表

返回值：无

本程序为默认处理函数，各个主站对象可以单独定义更新函数，若没有定义则调用本函数。

4.3.10、从站更新获取到的输入状态值的函数

/*更新读回来的输入状态值*/

```
__weak void UpdateInputStatus(uint16_t startAddress,uint16_t quantity,bool *stateValue)
{
  //在客户端（主站）应用中实现
}
```

参数：uint16_t startAddress，所读取的输入状态对象的起始地址

uint16_t quantity，所读取的输入状态对象的数量

bool *stateValue，读取到的输入状态对象的值

返回值：无

本程序为默认处理函数，各个主站对象可以单独定义更新函数，若没有定义则调用本函数。

4.3.11、从站更新保持寄存器值的函数

/*更新读回来的保持寄存器*/

```
__weak void UpdateHoldingRegister(uint16_t startAddress,uint16_t quantity,uint16_t *registerValue)
{
  //在客户端（主站）应用中实现
```

}

参数：uint16_t startAddress，所读取的保持寄存器对象的起始地址

uint16_t quantity，所读取的保持寄存器对象的数据

uint16_t *registerValue，读取到的保持寄存器对象值列表

返回值：无

本程序为默认处理函数，各个主站对象可以单独定义更新函数，若没有定义则调用本函数。

4.3.12、从站更新输入寄存器值得函数

```
/*更新读回来的输入寄存器*/

__weak void UpdateInputResgister(uint16_t startAddress,uint16_t quantity,uint16_t *registerValue)

{

  //在客户端（主站）应用中实现

}
```

参数：uint16_t startAddress，所读取的输入寄存器对象的起始地址

uint16_t quantity，所读取的输入寄存器对象的数量

uint16_t *registerValue，读取到的输入寄存器值得列表

返回值：无

本程序为默认处理函数，各个主站对象可以单独定义更新函数，若没有定义则调用本函数。

第 5 章 一些相关说明

对于 Modbus 协议栈的整个开发内容，前面已经说得很清楚了，接下来我们说明一下与开发没有直接关系的内容。

首先，关于我为什么开发这个协议栈的问题。我们的初衷只是想能够在开发产品时不用每次都重写这一部分，而是可以不断的改进和使用达到复用的目的。当然在后来，我们觉得不只是我们自己可以使用，也可以将其公开，让任何愿意使用的人使用。源码网址是：https://github.com/foxclever/Modbus

其次，Modbus 协议有国标，包括三个文件。我们这个协议栈就是按照国标开发的标准协议，包括有读写各种类型对象数据的功能，在一般的工业应用场合是完全够的。三个标准文件：

GB/T 19582.1-2008 《基于 Modbus 协议的工业自动化网络规范 第 1 部分：Modbus 应用协议》

GB/T 19582.2-2008 《基于 Modbus 协议的工业自动化网络规范 第 1 部分：Modbus 协议在串行链路上的实现指南》

GB/T 19582.3-2008 《基于 Modbus 协议的工业自动化网络规范 第 1 部分：Modbus 协议在 TCP/IP 上的实现指南》

最后，欢迎大家使用这个协议栈，但我们不就使用的最终结果负责。当然如果发现任何的不足，我们非常并欢迎大家将发现的问题告知我们，以便我们持续

的改进之。

感谢您关注本软件，有需要的同仁可以上 GitHub 下载，同时欢迎给我们提出改进意见，有兴趣的同仁可以关注我们的公众号：