# FreeModbus V1.5 主机使用说明

----------

## 一、简述

FreeModbus是一款开源的Modbus协议栈，但是只有从机开源，主机源码是需要\*\*收费\*\*的。同时网上也没有发现比较好的开源的Modbus主机协议栈，所以才开发这款支持主机模式的FreeModbus协议栈。本版FreeModbus版本号更改为V1.6，特性如下：

- 新增加的主机源码与原有从机的风格及接口保持一致；

- 支持主机与从机在同一协议栈运行；

- 支持实时操作系统及裸机移植；

- 为应用提供多种请求模式，用户可以选择阻塞还是非阻塞模式，自定义超时时间等，方便应用层灵活调用；

- 支持所有常用的Modbus方法。

### 1.1、文件结构

|源文件 |描述|

|:------------------------------ |:----- |

|FreeModbus\modbus\mb.c |给应用层提供Modbus从机设置及轮询相关接口|

|FreeModbus\modbus\mb\_m.c ||

|FreeModbus\modbus\ascii\mbascii.c |ASCII模式设置及其状态机|

|FreeModbus\modbus\functions\mbfunccoils.c |从机线圈相关功能|

|FreeModbus\modbus\functions\mbfunccoils\_m.c |主机线圈相关功能|

|FreeModbus\modbus\functions\mbfuncdisc.c |从机离散输入相关功能|

|FreeModbus\modbus\functions\mbfuncdisc\_m.c |主机离散输入相关功能|

|FreeModbus\modbus\functions\mbfuncholding.c |从机保持寄存器相关功能|

|FreeModbus\modbus\functions\mbfuncholding\_m.c |主机保持寄存器相关功能|

|FreeModbus\modbus\functions\mbfuncinput.c |从机输入寄存器相关功能|

|FreeModbus\modbus\functions\mbfuncinput\_m.c |主机输入寄存器相关功能|

|FreeModbus\modbus\functions\mbfuncother.c |其余Modbus功能|

|FreeModbus\modbus\functions\mbutils.c |一些协议栈中给应用层提供Modbus主机设置及轮询相关接口需要用到的小工具|

|FreeModbus\modbus\rtu\mbcrc.c |CRC校验功能|

|FreeModbus\modbus\rtu\mbrtu.c |从机RTU模式设置及其状态机|

|FreeModbus\modbus\rtu\mbrtu\_m.c |主机RTU模式设置及其状态机|

|FreeModbus\modbus\tcp\mbtcp.c |TCP模式设置及其状态机|

|FreeModbus\port\port.c |实现硬件移植部分接口|

|FreeModbus\port\portevent.c |实现从机事件移植接口|

|FreeModbus\port\portevent\_m.c |实现主机事件及错误处理移植接口|

|FreeModbus\port\portserial.c |从机串口移植|

|FreeModbus\port\portserial\_m.c |主机串口移植|

|FreeModbus\port\porttimer.c |从机定时器移植|

|FreeModbus\port\porttimer\_m.c |主机定时器移植|

|FreeModbus\port\user\_mb\_app.c |定义从机数据缓冲区，实现从机Modbus功能的回调接口|

|FreeModbus\port\user\_mb\_app\_m.c |定义主机数据缓冲区，实现主机Modbus功能的回调接口|

> 注：所有带\_m后缀的文件为主机模式下必须使用的文件，如使用从机模式则无需这些文件。

----------

## 二、移植

对于协议栈的移植主要包括两个方面，硬件及软件。用户需要根据自己的需求进行自行选择。

> 注：以下所有说明都主要针对Modbus主机模式进行介绍。

### 2.1、软件

软件方面支持基于裸机及实时操作系统的移植；支持单个主机与单个从机同时独立运行。另外用户也可以修改协议栈的事件回调接口，使主机请求的接口采用阻塞及非阻塞模式；主机资源等待方面，用户也可以设置等待超时时间等等，诸多功能将会一一介绍。

#### 2.1.1、操作系统与裸机

对于操作系统与裸机目前协议栈都是支持的，但个人更加推荐采用实时操作系统，因为这样会使得接口调用及接口移植变得更加简单。目前移植完成的操作系统包括国人的 [RT-Thread][1] （详见项目源码） 、 UCOS 及 FreeRTOS。

操作系统与裸机移植的过程中涉及的文件为`FreeModbus\port\portevent\_m.c`

该文件主要有以下需要用户移植的接口

|接口 |功能描述|

|:----- |:----|

|xMBMasterPortEventInit |主机事件初始化|

|xMBMasterPortEventPost |主机发送事件|

|xMBMasterPortEventGet |主机获取事件|

|vMBMasterOsResInit |主机操作系统资源初始化|

|xMBMasterRunResTake |主机资源获取|

|vMBMasterRunResRelease |主机资源释放|

|vMBMasterErrorCBRespondTimeout |主机响应超时回调接口|

|vMBMasterErrorCBReceiveData |主机接收数据出错回调接口|

|vMBMasterErrorCBExecuteFunction |主机执行Modbus方法出错回调接口|

|vMBMasterCBRequestScuuess |主机请求执行成功回调接口|

|eMBMasterWaitRequestFinish |主机等待请求完成处理回调接口|

在 \*\*基于操作系统移植\*\* 时，主要用到操作系统线程同步方面的技术，Modbus 协议栈自身需要使用操作系统自带的事件机制来实现事件的发送通知与等待获取，同时用户请求 Modbus 功能的线程与 Modbus 协议栈自身线程（Modbus Poll 线程）需要通过事件机制实现两个线程的同步；主机协议栈还需要一个主机资源占用的信号量，初始化默认为1，采用信号量保证了多线程同时发送主机请求时，只有一个线程可以使用主机。

在 \*\*基于裸机移植\*\* 时，需要通过软件模拟方式实现事件通知机制，事件等待及资源等待都得采用用户自定义延时及标志变量来实现，实现起来比操作系统模式下的线程同步机制要复杂很多。

#### 2.1.2、数据缓冲区

数据缓冲区定义的位置位于 `FreeModbus\port\user\_mb\_app\_m.c` 文件顶部，共计 \*\*4种\*\* 数据类型。

FreeModbus从机默认使用 \*\*一维数组\*\* 作为缓存区数据结构，主机可以存储所有网内从机的数据，所以主机采用 \*\*二维数组\*\* 对所有从机节点数据进行存储。二维数组的列号代表寄存器、线圈及离散量地址，行号代表从机节点ID，但需要做减一处理，例如`usMRegHoldBuf[2][1]`代表从机ID为 3，保持寄存器地址为 1 的从机数据。

#### 2.1.3、Modbus数据处理回调接口

Modbus 一共有4种不同的数据类型，所有的 Modbus 功能都围绕这些数据类型进行操作。由于不同的用户数据缓冲区结构可能有所不同，那么对应的 Modbus 数据处理方式也就存在差异，所以用户需要把每种数据类型对应的操作，按照自己的数据缓冲区结构进行定制实现。

所有的 Modbus 数据处理回调接口如下：

|接口 |功能描述|

|:----- |:----|

|eMBMasterRegInputCB |输入寄存器回调接口|

|eMBMasterRegHoldingCB |保持寄存器回调接口|

|eMBMasterRegCoilsCB |线圈回调接口|

|eMBMasterRegDiscreteCB |离散输入回调接口|

> 对于数组形式的数据缓冲区结构，源码中已经做好了移植，直接使用即可。也可以使用 [EasyDataManager](https://github.com/armink/EasyDataManager) 库，采用链表作为缓冲区，该库还支持事件驱动，做到数据变化自动通知应用层。

### 2.2、硬件

移植 FreeModbus 协议栈主机部分时，在硬件方面需要修改串口及定时器配置，文件位于port文件下，用户需要根据自己的CPU进行移植修改。

> 注：协议栈默认自带STM32F103X移植文件，用户可以参考移植

这里提一下基于操作系统设备驱动框架的移植，后期协议栈会增加对 [RT-Thread][1] 自带设备驱动框架的移植，只要是 RT-Thread 的 BSP 支持的 MCU，用户都无需考虑底层的移植过程，减低移植成本。

#### 2.2.1、串口

涉及到串口的移植文件位于`FreeModbus\port\portserial\_m.c`，在这个文件中用户需要对以下接口方法进行修改

|接口 |功能描述|

|:----- |:----|

|vMBMasterPortSerialEnable |使能和失能串口的发送及接收功能，如使用485总线，需要注意收发模式切换|

|vMBMasterPortClose |关闭串口|

|xMBMasterPortSerialInit |串口初始化，如果使用485，收发模式切换引脚也要在此初始化|

|xMBMasterPortSerialPutByte |串口发送单字节数据|

|xMBMasterPortSerialGetByte |串口接收单字节数据|

|prvvUARTTxReadyISR |串口发送完成中断服务程序接口，按照默认方式，直接引用`pxMBMasterFrameCBTransmitterEmpty`方法即可|

|prvvUARTRxISR |串口接收中断服务程序接口，按照默认方式，直接引用`pxMBMasterFrameCBByteReceived`方法即可|

> 还需要在文件末尾增加 CPU 的自带的串口服务程序，将上表中的发送及接收中断程序接口，放到对应的中断服务程序中去即可。

#### 2.2.2、定时器

涉及到定时器的移植文件位于`FreeModbus\port\porttimer\_m.c`，在这个文件中用户需要对以下接口方法进行修改

|接口 |功能描述|

|:----- |:----|

|xMBMasterPortTimersInit |定时器初始化，将定时器预分频数及T3.5时间计数值分别备份到`usPrescalerValue`及`usT35TimeOut50us`|

|vMBMasterPortTimersT35Enable |设置定时器按照T3.5时间开始计数|

|vMBMasterPortTimersConvertDelayEnable |设置定时器按照广播帧的转换延时时间开始计数|

|vMBMasterPortTimersRespondTimeoutEnable |设置定时器按照响应超时时间开始计数|

|vMBMasterPortTimersDisable |失能定时器，定时器将停止计数|

|prvvTIMERExpiredISR |定时器中断服务程序接口，按照默认方式，直接引用`pxMBMasterPortCBTimerExpired`方法即可|

> 注：

> 1、`usPrescalerValue`及`usT35TimeOut50us`在文件顶部有定义

> 2、转换延时时间及响应超时时间在`FreeModbus\modbus\include\mbconfig.h`，用户可以根据自己系统的特点自行设置。

除上面接口方法外，用户需要在文件末尾增加 CPU 的自带的定时器中断服务程序，将上表中的定时器中断服务程序接口放进去。

## 三、API

Modbus 主机使用过程中与从机有很大不同，从机是需要被动等待主机请求，而主机则是主动发出请求，并接收处理从机响应。在主机发送广播请求的时候，从机不需要返回响应，所以广播请求适合主机的写从机数据命令，不适合读从机数据命令。

主机请求API中的所有方法的返回值格式都相同，返回值意义如下。

|返回值 |描述|

|:----- |:----|

|MB\_MRE\_NO\_ERR |正常，没错误|

|MB\_MRE\_NO\_REG |寄存器、线圈或离散输入地址出错|

|MB\_MRE\_ILL\_ARG |入参格式有误|

|MB\_MRE\_REV\_DATA |接收数据出错|

|MB\_MRE\_TIMEDOUT |响应超时。主机在设定的时间内未收到从机响应。|

|MB\_MRE\_MASTER\_BUSY |主机忙。在设定的时间内，没有请求没有被发送。|

|MB\_MRE\_EXE\_FUN |主机收到响应后，执行Modbus方法（功能）出错。|

> 所有的主机请求方法都是 \*\*线程安全\*\* 的也是 \*\*阻塞模式\*\* 的。在使用过程中，只要在设定的超时时间内没有得到主机资源，就会返回主机忙；如果在设定的超时时间内得到主机资源，那么必须等待得到请求结果后才会返回。

### 3.1、写单个保持寄存器

往从机某个保持寄存器中写入数据

```C

eMBMasterReqErrCode eMBMasterReqWriteHoldingRegister( UCHAR ucSndAddr,

USHORT usRegAddr,

USHORT usRegData,

LONG lTimeOut );

```

|参数 |描述|

|:----- |:----|

|ucSndAddr |请求的从机地址，0代表广播。|

|usRegAddr |写寄存器的地址|

|usRegData |写寄存器的数据|

|lTimeOut |请求超时时间。支持永久等待，使用操作系统的永久等待参数即可。|

### 3.2、写多个保持寄存器

往从机多个保持寄存器中写入数据。

```C

eMBMasterReqErrCode eMBMasterReqWriteMultipleHoldingRegister( UCHAR ucSndAddr,

USHORT usRegAddr,

USHORT usNRegs,

USHORT \* pusDataBuffer,

LONG lTimeOut )

```

|参数 |描述|

|:----- |:----|

|ucSndAddr |请求的从机地址，0代表广播。|

|usRegAddr |写寄存器的起始地址|

|usNRegs |写寄存器的总数|

|pusDataBuffer |写寄存器的数据|

|lTimeOut |请求超时时间。支持永久等待，使用操作系统的永久等待参数即可。|

### 3.3、读多个保持寄存器

读取多个保持寄存器中的数据

```C

eMBMasterReqErrCode eMBMasterReqReadHoldingRegister( UCHAR ucSndAddr,

USHORT usRegAddr,

USHORT usNRegs,

LONG lTimeOut );

```

|参数 |描述|

|:----- |:----|

|ucSndAddr |请求的从机地址，0代表广播。|

|usRegAddr |读寄存器的地址|

|usRegData |读寄存器的数量|

|lTimeOut |请求超时时间。支持永久等待，使用操作系统的永久等待参数即可。|

### 3.4、读写多个保持寄存器

先读多个寄存器，然后再写多个寄存器。

```C

eMBMasterReqErrCode eMBMasterReqReadWriteMultipleHoldingRegister( UCHAR ucSndAddr,

USHORT usReadRegAddr,

USHORT usNReadRegs,

USHORT \* pusDataBuffer,

USHORT usWriteRegAddr,

USHORT usNWriteRegs,

LONG lTimeOut )

```

|参数 |描述|

|:----- |:----|

|ucSndAddr |请求的从机地址，0代表广播。|

|usReadRegAddr |读寄存器的地址|

|usNReadRegs |读寄存器的数量|

|pusDataBuffer |写寄存器的数据|

|usWriteRegAddr |写寄存器的地址|

|usNWriteRegs |写寄存器的数量|

|lTimeOut |请求超时时间。支持永久等待，使用操作系统的永久等待参数即可。|

### 3.5、读多个输入寄存器

读取多个输入寄存器中的数据

```C

eMBMasterReqErrCode eMBMasterReqReadInputRegister( UCHAR ucSndAddr,

USHORT usRegAddr,

USHORT usNRegs,

LONG lTimeOut );

```

|参数 |描述|

|:----- |:----|

|ucSndAddr |请求的从机地址，0代表广播。|

|usRegAddr |读寄存器的地址|

|usRegData |读寄存器的数量|

|lTimeOut |请求超时时间。支持永久等待，使用操作系统的永久等待参数即可。|

### 3.6、写单个线圈

往从机某个线圈中写入数据

```C

eMBMasterReqErrCode eMBMasterReqWriteCoil( UCHAR ucSndAddr,

USHORT usCoilAddr,

USHORT usCoilData,

LONG lTimeOut )

```

|参数 |描述|

|:----- |:----|

|ucSndAddr |请求的从机地址，0代表广播。|

|usCoilAddr |写线圈的地址|

|usCoilData |写线圈的数量|

|lTimeOut |请求超时时间。支持永久等待，使用操作系统的永久等待参数即可。|

### 3.7、写多个线圈

往从机多个线圈中写入数据。

```C

eMBMasterReqErrCode eMBMasterReqWriteMultipleCoils( UCHAR ucSndAddr,

USHORT usCoilAddr,

USHORT usNCoils,

UCHAR \* pucDataBuffer,

LONG lTimeOut)

```

|参数 |描述|

|:----- |:----|

|ucSndAddr |请求的从机地址，0代表广播。|

|usCoilAddr |写线圈的起始地址|

|usNCoils |写线圈的总数|

|pucDataBuffer |写线圈的数据|

|lTimeOut |请求超时时间。支持永久等待，使用操作系统的永久等待参数即可。|

### 3.8、读多个线圈

读取多个线圈中的数据

```C

eMBMasterReqErrCode eMBMasterReqReadCoils( UCHAR ucSndAddr,

USHORT usCoilAddr,

USHORT usNCoils ,

LONG lTimeOut )

```

|参数 |描述|

|:----- |:----|

|ucSndAddr |请求的从机地址，0代表广播。|

|usCoilAddr |读线圈的地址|

|usNCoils |读线圈的数量|

|lTimeOut |请求超时时间。支持永久等待，使用操作系统的永久等待参数即可。|

### 3.9、读多个离散输入

读取多个离散输入中的数据

```C

eMBMasterReqErrCode eMBMasterReqReadDiscreteInputs( UCHAR ucSndAddr,

USHORT usDiscreteAddr,

USHORT usNDiscreteIn,

LONG lTimeOut )

```

|参数 |描述|

|:----- |:----|

|ucSndAddr |请求的从机地址，0代表广播。|

|usDiscreteAddr |读离散输入的地址|

|usNDiscreteIn |读离散输入的数量|

|lTimeOut |请求超时时间。支持永久等待，使用操作系统的永久等待参数即可。|

## 四、流程

### 4.1、初始化配置流程

本协议栈所有配置参数都位于`FreeModbus\modbus\include\mbconfig.h`，目前协议栈支持主机及从机两种模式，并且支持\*\*两种模式同时开启\*\*。从机支持Modbus RTU 、Modbus ASCII 及Modbus TCP 3种模式，主机现在只支持常用的\*\*Modbus RTU\*\*模式。在使用主机的过程中，用户需要对广播的转换延时时间、命令响应超时时间及从机数量做以配置。需要注意的是，目前协议栈只支持\*\*从机地址连续\*\*，并且起始地址\*\*从1开始\*\*。

### 4.2、正常使用流程

这里只介绍主机的正常使用流程，在使用主机前，需要先把协议栈移植到自己的项目中去，包括上述的软件及硬件部分，移植完成后的使用流程如下

- 1、调用`eMBMasterInit`方法初始化Modbus主机协议栈，主机涉及到的一些硬件就在这个时候做了初始化

- 2、调用`eMBMasterEnable`方法启动Modbus主机

- 3、通过在线程或者定时器轮询调用`eMBMasterPoll`方法，轮询周期决定了命令的响应时间。

- 4、调用主机请求API方法，设定一定的请求超时时间，直到方法有结果后才会返回。如果方法执行成功并且命令是读命令，可以通过查看Modbus主机的数据缓冲区，获取最新从机数据。

### 4.3、异常处理流程

异常处理主要出现在主机正常使用过程中，所有的主机请求API的错误码都在第三章开头已经做以描述，针对的这些错误码，用户需要根据自己的产品特征去完成不同的动作。建议用户自己封装实现主机请求方法的重发机制，这样实现方式比较灵活，一般是在接收到帧数据出错及命令响应超时的错误码时需要重发，重发次数自动加一，如果重发次数超过设定值则认为从机掉线，以后所有只要是发给这个从机命令都被提前拦截掉；如果第二次重发命令响应成功，则\*\*自动清零\*\*该从机重发次数。

上述所有功能可以利用主机请求方法或者使用`FreeModbus\port\portevent\_m.c`中的回调接口来实现，用户可以根据自己的需求灵活选择。

[1]: http://www.rt-thread.org/