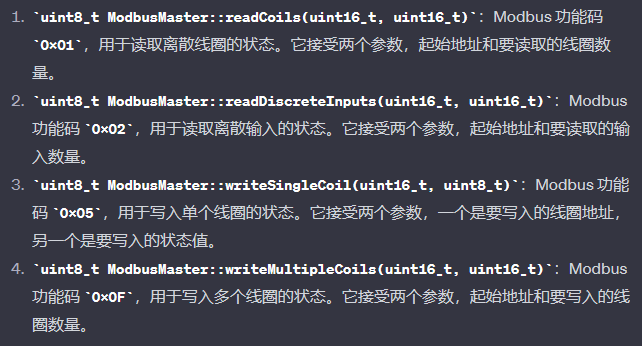
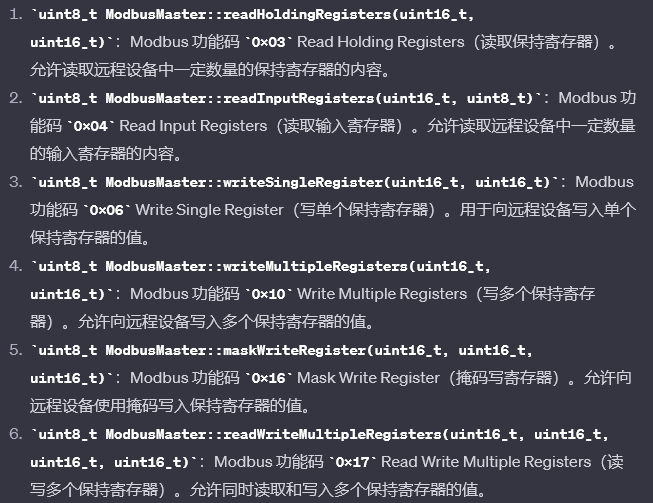
Modbus Master库个别函数含义：



**CRC检验码学习**

[CRC](https://so.csdn.net/so/search?q=CRC&spm=1001.2101.3001.7020)即循环冗余校验码，是数据通信领域中最常用的一种查错校验码，其特征是信息字段和校验字段的长度可以任意选定。循环冗余检查（CRC）是一种数据传输检错功能，对数据进行多项式计算，并将得到的结果附在帧的后面，接收设备也执行类似的算法，以保证数据传输的正确性和完整性。

**CRC校验原理：**  
 其根本思想就是先在要发送的帧后面附加一个数（这个就是用来校验的校验码，但要注意，这里的数也是二进制序列的，下同），生成一个新帧发送给接收端。当然，这个附加的数不是随意的，它要使所生成的新帧能与发送端和接收端共同选定的某个特定数整除（注意，这里不是直接采用二进制除法，而是采用一种称之为“模2除法”）。到达接收端后，再把接收到的新帧除以（同样采用“模2除法”）这个选定的除数。因为在发送端发送数据帧之前就已通过附加一个数，做了“去余”处理（也就已经能整除了），所以结果应该是没有余数。如果有余数，则表明该帧在传输过程中出现了差错

**一种校验方式：**// 计算 Modbus RTU 数据帧的 CRC 校验码

uint16\_t calculateCRC(uint8\_t data[], int length) {

uint16\_t crc = 0xFFFF; // 初始值

for (int i = 0; i < length; ++i) {

crc ^= data[i]; // 将每个字节与 CRC 寄存器进行异或操作

for (int j = 0; j < 8; ++j) {

if (crc & 0x0001) {

crc = (crc >> 1) ^ 0xA001; // 根据多项式进行右移和异或操作

} else {

crc = (crc >> 1);

}

}

}

return crc;

}

void setup() {

// 数据帧内容为 01H 06H 02H 00H 00H 01H

uint8\_t data[] = {0x01, 0x06, 0x02, 0x00, 0x00, 0x01}; // 数据帧

int dataLength = sizeof(data);

uint16\_t crc = calculateCRC(data, dataLength);

// 输出计算出的 CRC 校验码

Serial.begin(9600);

Serial.print("Calculated CRC: ");

Serial.println(crc, HEX);

}

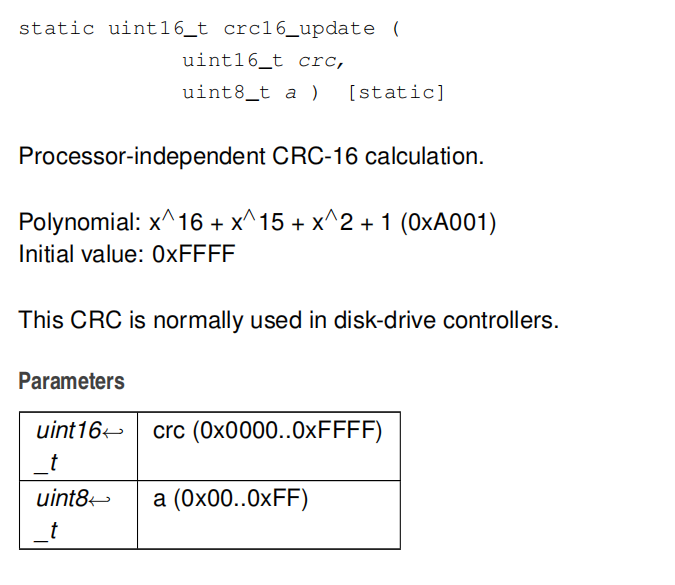
void loop() {

// 你的代码在 loop 中

}

* 从站地址：01H（设备地址）
* 功能码：03H（读取多个寄存器的功能码）
* 寄存器起始地址高位：0BH（参数的起始地址高位）
* 寄存器起始地址低位：02H（参数的起始地址低位）
* 读取寄存器数量高位：00H（要读取的寄存器数量的高位）
* 读取寄存器数量低位：04H（要读取的寄存器数量的低位）
* CRC 校验高位：E7H
* CRC 校验低位：EDH

**crc16\_update()**



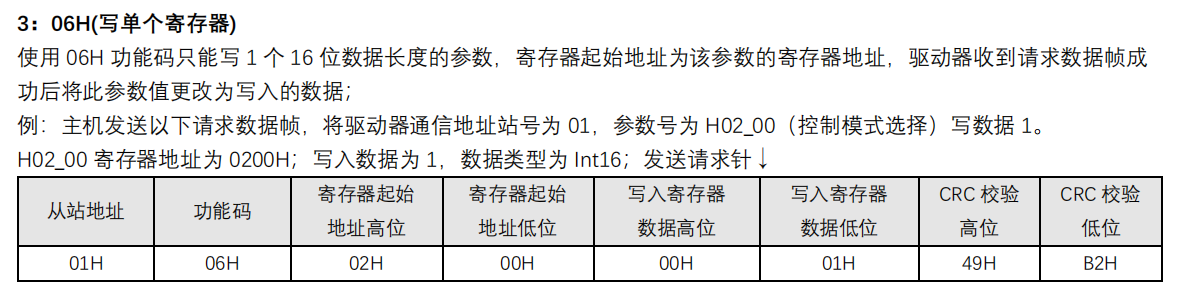
返回值为计算出的crc校验码，需要进行比对确认。注意高低位前后



电机到了之后试上位机通讯速度控制：



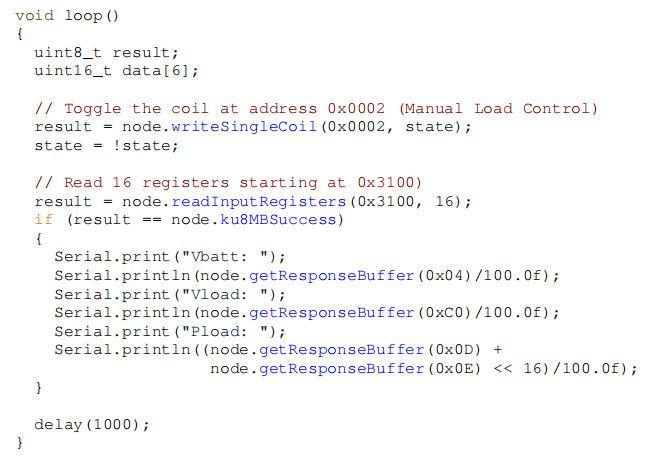
上位机通信没有问题，采用速度控制利用arduino对电机进行速度控制

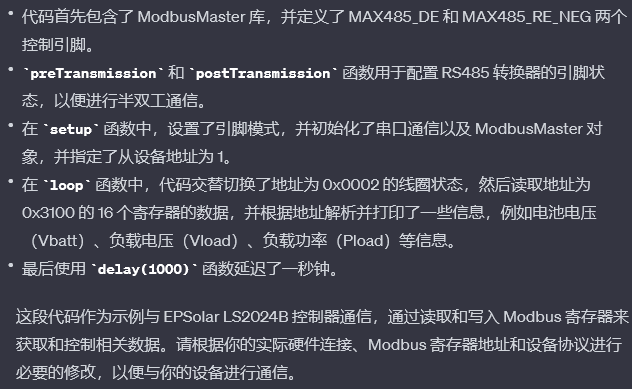


根据参数地址的编号写入以下指令桢0106 0603 0200

通信端口输出乱码问题：

重新解读modbus master例程



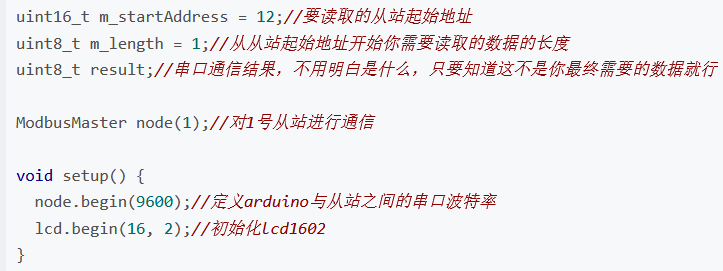


半双工通信是一种通信方式，允许数据在两个设备之间进行双向传输，但不同时间只能进行单向传输。换句话说，设备可以发送或接收数据，但不能同时进行这两个操作。在半双工通信中，设备必须在通信开始前约定好谁先发送谁先接收，以避免冲突。一般情况下，在发送数据后，发送端需要等待确认或特定的时间，然后再允许接收数据。对于 RS485 通信而言，它是一种常用于工业控制领域的串行通信标准。RS485 支持半双工通信，允许多个设备通过相同的通信线路进行通信。在 RS485 中，一个设备发送数据时，其他设备需要处于接收模式。在示例代码中使用的 MAX485 转换器是一种常用于 RS485 通信的硬件，它通过控制 DE（Driver Enable）和 RE（Receiver Enable）引脚，使设备可以在发送或接收模式之间切换。在半双工通信中，设备需要切换这些引脚以确定发送或接收数据的状态。示例代码中，preTransmission postTransmission 函数控制了 MAX485 转换器的 DE 和 RE 引脚状态，在数据传输之前和之后分别将其设置为发送和接收模式。这样做可以确保设备在发送和接收数据时引脚状态正确，避免冲突。

总的来说，半双工通信允许设备在同一通信线路上交替地发送和接收数据，但是不能同时进行发送和接收操作。通过合适地控制硬件和约定通信协议，可以实现有效的半双工通信。

例程中使用modbus master对电机进行线圈地址定位后读或写程序，crc校验使用仍有疑问。上位机对电机参数设置正常且能驱动，arduino收发信号时，转换器模块闪烁并tx闪烁，Serialmonitor显示“xx” 并print“输入数据有误”，判断应该是串口未成功接受发送的数据格式。

这里阅读了一个别人的modbusmaster例程：



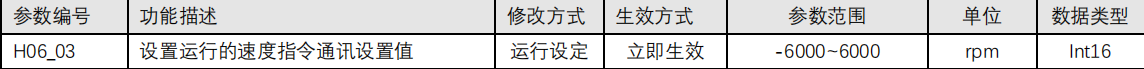


其中说明书寄存器信息：



从中可以看出地址为000CH也就是**12号地址的寄存器**放的是电压的情况。所以代码中uint16\_t m\_startAddress = 12;m\_startAddress写成地址12 （16位）   
uint8\_t m\_length = 1;读取1个地址的信息（8位）这样就可以愉快地读电压了。

（uint16\_t 与uint8\_t是类型名，类似与int,float….就比如说0100 1001就是一个uint8\_t变量，0010 0011 1100 1111 就是一个uint16\_t变量）

所以下一步我们对电机的说明书寄存器说明重新翻看：假设我们要对

H0603寄存器进行写入换算为16位无符号整数值为1539号寄存器所以代码中uint16\_t speedregister = 1539;写入 uint16\_t speedvalue = 200;

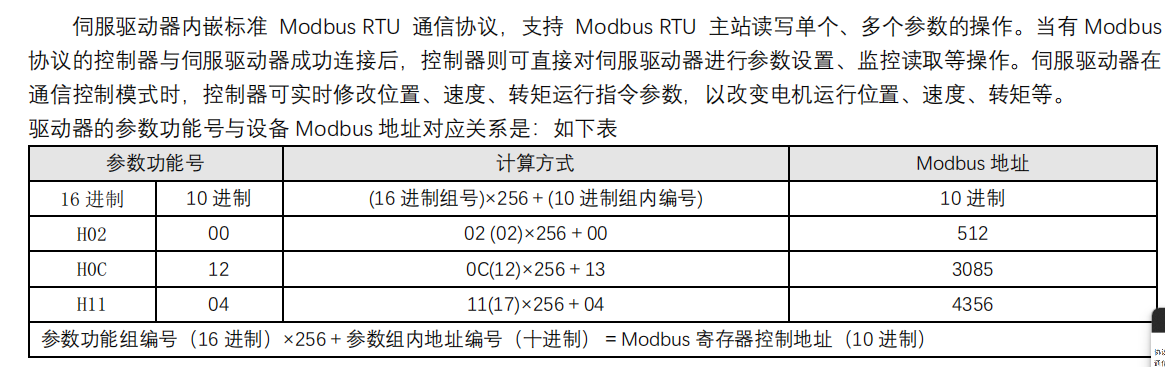
但是发现程序仍然报错：



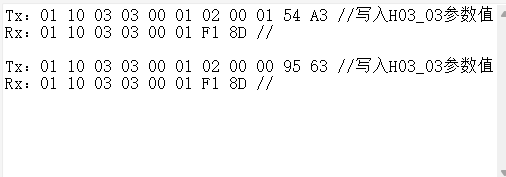
If判断中的node返回值存在问题，写入操作未成功

在 Arduino 的 ModbusMaster 库中，node.ku8MBSuccess 是一个常量，用于表示 Modbus 通信操作成功的状态码。这个常量的使用方式是通过 ModbusMaster 对象来访问

考虑是否可能是参数设置还存在问题，考虑参数范围对输入的速度值进行放缩发现还是不能写入参数，咨询客服后发现问题，



地址编号形式和上面例程的说明书不同，需要分段重新计算，但对于编号值小的不影响。

查阅arduino说明书发现，默认tx18，rx19引脚为串口默认输出。连接ttl转485模块后，根据上位机给出的指令，发现用的是10多寄存器功能码，找到问题，换用multiple函数进行写入，查阅资料得知，该函数需要提前根据数据选择缓存器的值调用。

烧录后，电机成功使能

读Tx：01 03 32 02 00 01 2B 72 //读取H32\_02参数值

Tx：01 03 00 00 00 64 44 21 //读取H00\_00参数值

Tx：01 03 06 05 00 64 54 A8 //读取H06\_05参数值

Tx：01 03 12 1E 00 03 60 B5 //读取H12\_30参数值

Rx：01 03 06 00 32 00 00 00 02 99 70 //

Tx：01 03 32 02 00 01 2B 72 //读取H32\_02参数值

Rx：01 03 02 00 02 39 85 //

Tx：01 03 00 00 00 64 44 21 //读取H00\_00参数值

Tx：01 03 06 05 00 64 54 A8 //读取H06\_05参数值

Tx：01 03 12 1E 00 03 60 B5 //读取H12\_30参数值

Rx：01 03 06 00 32 00 00 00 02 99 70 //

Tx：01 03 32 02 00 01 2B 72 //读取H32\_02参数值

Rx：01 03 02 00 02 39 85 //

Tx：01 03 00 00 00 64 44 21 //读取H00\_00参数值

Tx：01 03 06 05 00 64 54 A8 //读取H06\_05参数值

Tx：01 03 12 1E 00 03 60 B5 //读取H12\_30参数值

Rx：01 03 06 00 32 00 00 00 02 99 70 //

考虑到该驱动器读单个寄存器有的也用multiple函数10功能码，对读取报文进行打印，几个主要控制参数进行打印，方便之后的编程。

H303:01 10 03 03 00 01 02 00 01 54A3

H605:01 10 06 05 00 01 02 22 B8 D8D7

H606:01 10 06 06 00 01 02 22 B8 D8E4

H603:01 10 06 03 00 01 02 07 D0 C3CF

计划对一套程序单个电机展开编程，根据实验发现该驱动器不需要进行使能就能传入参数，所以首先参考速度控制所能控制的几个参数，定义函数块包括开关使能，速度设定，考虑设计要求读取该电机当前速度。

if(Serial.available()>0)

  {

   node.setTransmitBuffer(0,0);

      delay(100);

      node.writeMultipleRegisters(0x0603, 1);

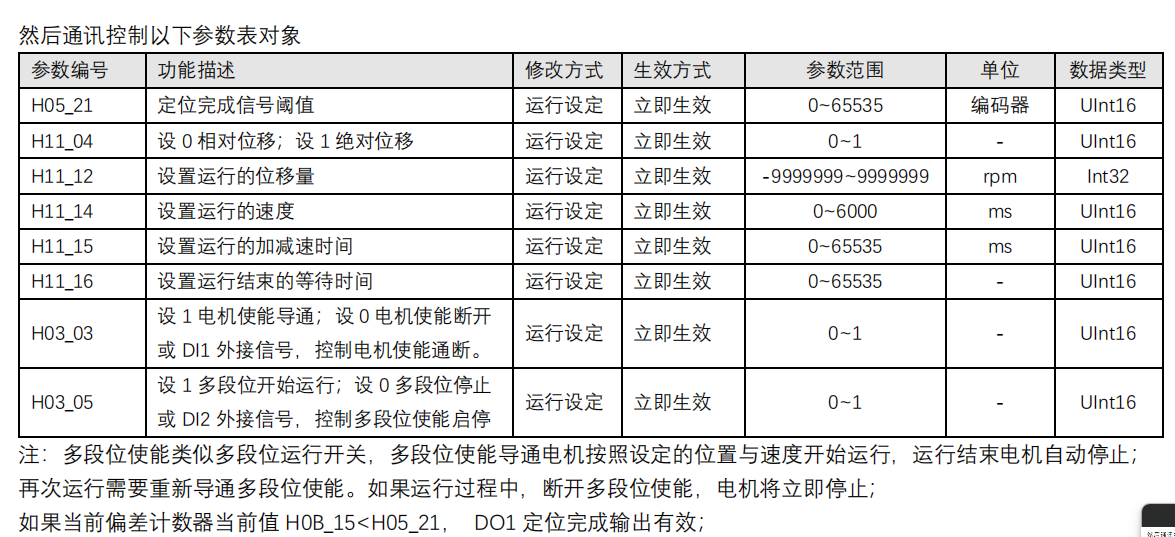
      delay(100);

  }//初始化电机当前速度为零

速度控制函数编写完成后，尝试进行位置控制



上位机设置如上



这里选择使用的参数，首先将不改变的参数设定，一段

Tx：01 10 03 05 00 01 02 00 01 54 C5 //写入H03\_05参数值1

Rx：01 10 03 05 00 01 11 8C //

然后调用一段

Tx：01 10 03 05 00 01 02 00 00 95 05 //写入H03\_05参数值0

Rx：01 10 03 05 00 01 11 8C //

之后进入下一次指令

位移设定

Tx：01 10 11 0C 00 02 04 17 70 00 00 37 C5 //写入H11\_12参数值6000

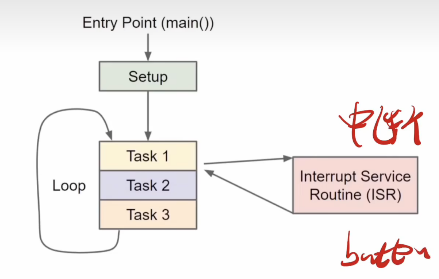
Rx：01 10 11 0C 00 02 84 F7 //

位移的速度设定

Tx：01 10 11 0E 00 01 02 0B B8 A1 3D //写入H11\_14参数值3000

Rx：01 10 11 0E 00 01 65 36 //

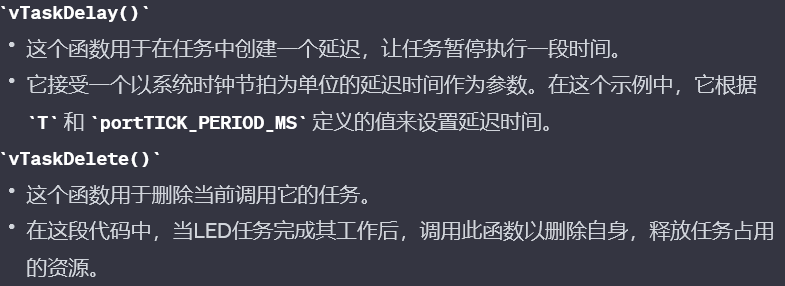
使用Freertos库

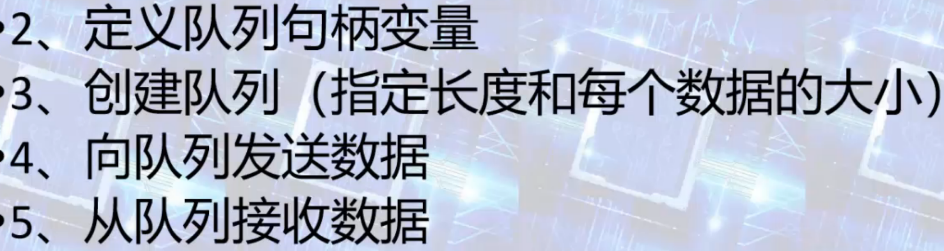


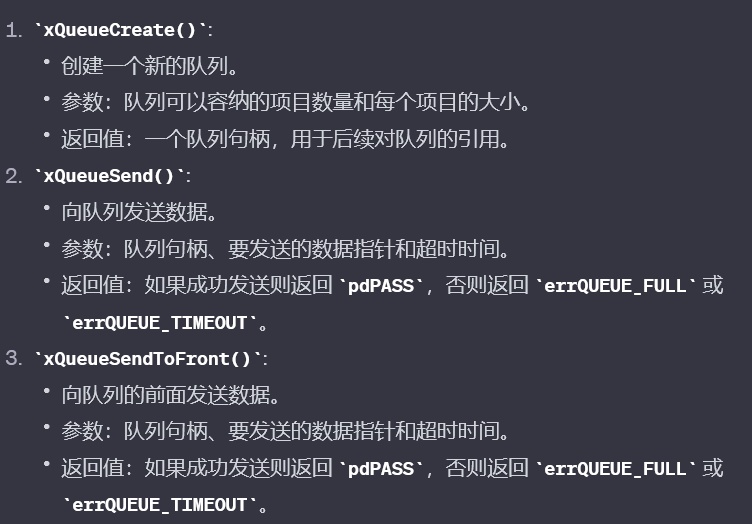


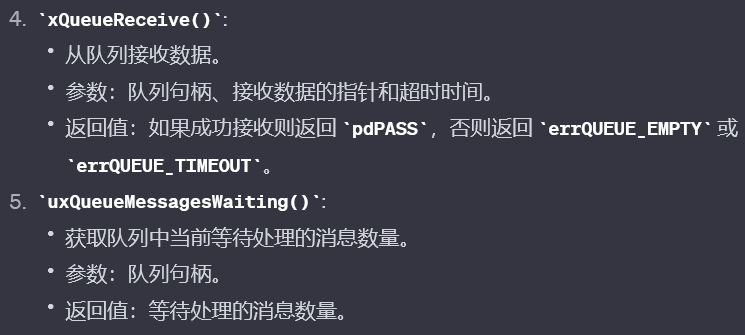
xTaskCreate(TaskPrint1, "Print1", 128, NULL, 1, NULL);

xTaskCreate(TaskPrint2, "Print2", 128, NULL, 2, NULL);



队列使用：





xTaskCreate(TaskPrint2, "Print2", 128, NULL, 2, NULL);