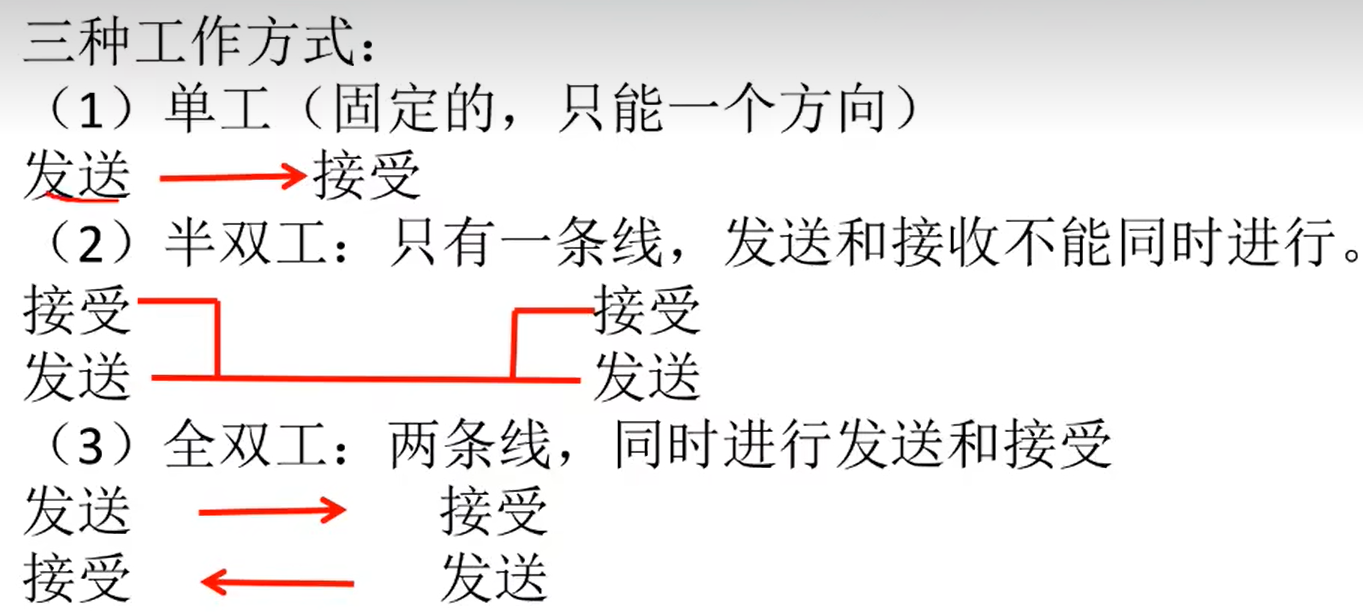
# 通信协议

## 通信学习1

### 1、串口通信参数设置

#### Rs-485通信：

9个接口，1，2，3，4，5为A+；6，7，8，9为B-；一般就是用两个接口：特点传输，半双工：交换数据不能同时发送，不能同时接受



#### 串行通信数据流和格式：

原理：串行通信方式是将字节（byte）拆分成一个接一个的位（bit）后，再传输出去，收到数据的一方，再将这些位组合成原来的字节。

要求：传输格式必须统一，传输方式（节拍）必须一致（速度一致）

波特率：表示每秒传输的位的个数，也就是每秒传输0和1的个数，常见是9600b/s,4800b/s,19200b/s,38400b/s

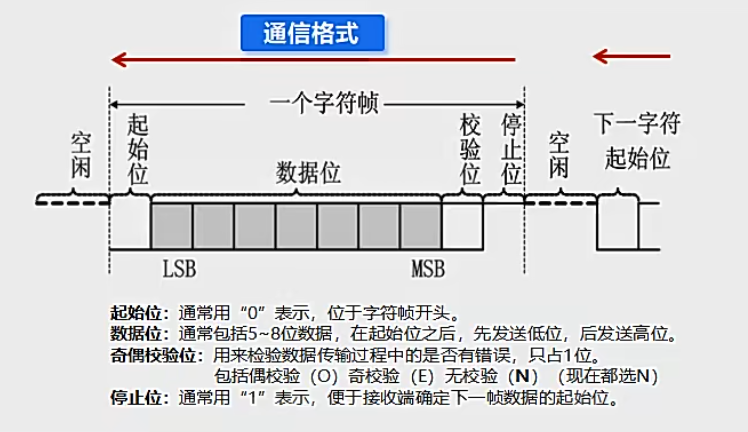
串行通信格式（单字节协议）

常见：（9600，N,8,1）波特率9600b/s，无奇偶校验位，8位数据位，1位停止位。

#### 通信格式：

起始位：判断是否要发送数据，一般0开头

必须存在间隔



### 2、modbus协议基础

#### 关于通信协议：

不同设备之间交换数据要遵循的规范，就好比人与人之间交流用的语言，必须要有语法。通常情况下，串行通信遵守Modbus协议，串口232，422，485都是按照Modbus协议，但是Modbus也可也用于以太网的TCP/IP通信。

#### Modbus与串口关系：

串口属于物理接口，属于硬件。

Modbus是国际标准的串行通信协议，是软件。

#### Modbus与串行通信关系：

串行通信有规定的格式，所以如何识别传送的多个字节和解析标准怎么规定？

Modbus：就是如何用串口一次连续传输多个有序字节的协议。它规定一次能传输多少个字节，以及字节的顺序如何排列。

#### Modbus网络传输的三种模式：

1. ASCII模式：MG标准信息交换码（0-9，a-z，A-Z），数据中每8个字节都用ASCII码发送。
2. RTU模式：远程终端单元模式通信，针对通信距离较长和工业环境恶劣而设计的痛惜结构。特点：消息中每8个bit的字节都包含4bit的十六进制符。
3. TCP模式：通过以太网和互联网连接传输数据使用的TCP/IP协议，简称TCP模式，硬件接口就是以太网交接口。

Modbus协议基础

3.5字符：时间间隔4ms

地址域：一个字节用来定位设备

功能码：一个字节，指定存储区域的操作类型（指定存储区域，表明是读取数据，还是写入数据）

数据域：传输数据啊



### 3、数据存储区和功能码

#### 线圈和寄存器

线圈：在电气控制中，接触器和继电器都是靠得电和失电来控制断开和闭合，用线圈来表示，寄存器在计算机中就是用来存储数据，所以非布尔量的数据用寄存器存储。

读写：以西门子plc为例，I表示输入，Q表示输出，输入意味着该存储区里的值是外部设备接入，是只读的，Q是输出表示输出结果给外部设备，所以可以读写。







我们一般人为沟通交流和在说明书中使用绝对地址；

在Modbus协议中，我们使用相对地址来表明某个区域，通过功能码来区分，是那个区域实现那种功能。

#### 寄存器设备地址（硬件地址）：

就是PLC硬件芯片中对应的地址，一般是十进制，一般是五位，第一位表明寄存器类型。

寄存器Modbus协议地址：指通信使用的寄存器寻址地址

编写代码时，我们使用的是Modbus的协议地址，但是我们最后通信到硬件上时，还是寄存器设备地址。

是通过Modbus协议通信时规定的功能区，来区分使用的Modbus地址对应设别地址中的哪一个寄存器。

#### 功能码：



保持—输出

### modbus通信报文分析

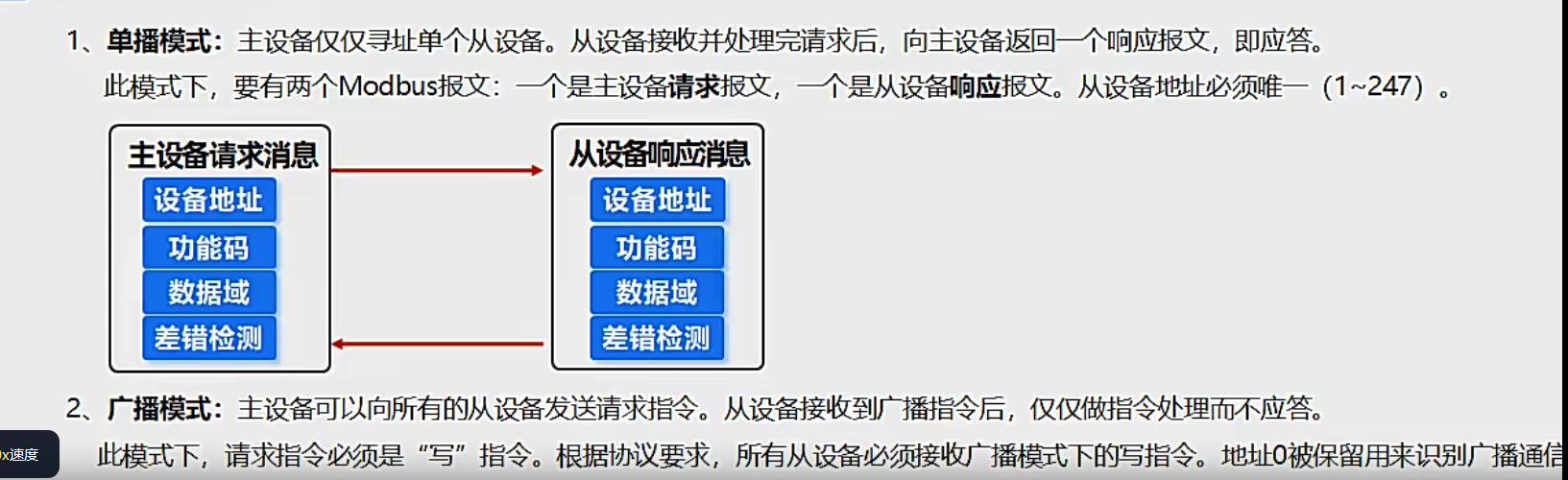
Modbus协议：是一种单主/多从的通信协议

特点：同一时间总线上只能有一个主设备，但是可以有一个或者多个(247)从设备

规定：通信只能由主设备发起，从设备没有收到主设备请求时，不会主动发送数据。

注意：设备之间不能相互通信。(从站之间不能相互传输数据)

寄存器就是一个字，一个字=两个字节





起始寄存器==》Modbus协议地址

### 主-从通信仿真测试

### c#通信库编写和测试

1. 建立ModbusRTU通信类库
2. 建立类库的构造函数
3. 创建串口对象，SerialPort
4. 创建接受报文的延迟方法
5. 创建连接方法Connect，其中要让串口对象绑定串口名PortName，波特率BaudRate，奇偶校验Parity，数据位Int DataBits（是几位），停止位StopBits；最后打开通信，建立关闭连接的方法DisConnect

### 7、ModbusRTU的通信

#### 1、读取输入，输出线圈0X01,0X02

1）、拼接报文，创建List<byte> list对象，依次添加从站地址，功能码，初始线圈地址（十进制/256，十进制%256）,线圈数量+CRC校验

2）、发送，接受报文，写成一个通用的方法。首先serialPort.Write（发送报文字节数组，0，发送报文字节数组长度）；延时Thread.Sleep（1000）接受报文；创建接受报文时间，建立MemoryStream内存对象，创建一个Byte字节容量；循环接受返回报文，判断BetysToRead是否存在返回报文的字节的数量，将其一次次添加到MemoryStream内存中，最后没有BytesToRead之后，全部写入给放回字节数组

3）验证发送，接受是否成功，验证报文是否正确，解析报文

ModbusRTU，读取输出0X01，读取输入0X02线圈方法

//第一步，拼接报文

//第二步：发送报文

//第三步：接收报文

//第四步：验证报文

//第五步：解析报文

/// <summary>

/// 读取输出线圈

/// </summary>

/// <param name="slave">从站地址</param>

/// <param name="startCoil">初始线圈地址十进制</param>

/// <param name="length">线圈的数量十进制</param>

/// <returns></returns>

public byte[] ReadOutPutCoils(byte slave,ushort startCoil,ushort length)

{

//第一步拼接报文

List<byte> sendReadOutPutCoils=new List<byte>();

//从站地址

sendReadOutPutCoils.Add(slave);

//功能码

sendReadOutPutCoils.Add(0X01);

//初始线圈地址

sendReadOutPutCoils.Add(Convert.ToByte(startCoil/256));

sendReadOutPutCoils.Add(Convert.ToByte(startCoil%256));

//线圈数量

sendReadOutPutCoils.Add(Convert.ToByte(length/256));

sendReadOutPutCoils.Add(Convert.ToByte(length%256));

//此时需要获得CRC校验

byte[] sendCRC = CRC16(sendReadOutPutCoils.ToArray(), sendReadOutPutCoils.Count);

sendReadOutPutCoils.Add(sendCRC[0]);

sendReadOutPutCoils.Add(sendCRC[1]);

//第二步,发送报文

//第三步，接受报文

byte[] receive = null;

//验证发送接受报文是否成功

if( SendAndReceive(sendReadOutPutCoils.ToArray(),ref receive))

{

//线圈长度

int coil = length%8==0 ? 1 : length/8+1;

//第四步验证报文

if (CheckCRC(receive)&&receive.Length==5+coil)

{

//第五步，解析报文

byte[] buffer = new byte[coil];

Array.Copy(receive, 3, buffer, 0, buffer.Length);

return buffer;

}

}

return null;

}

**发送，接受报文的通用方法**

/// <summary>

/// 发送，接受报文

/// </summary>

/// <param name="send">发送报文字节数组</param>

/// <param name="receive">接受报文字节数组</param>

/// <returns>判断是成功</returns>

public bool SendAndReceive(byte[] send,ref byte[] receive)

{

try

{

//发送报文

serialPort.Write(send, 0, send.Length);

//定义一个buffer，用来缓存每次接受的报文

byte[] buffer = new byte[1024];

//定义一个内存，MemoryStream

MemoryStream memoryStream = new MemoryStream();

//定义一个开始时间时间

DateTime start = DateTime.Now;

//这么处理的原因是防止一次读取不完整

//循环读取缓冲区的数据，如果大于0，就读取出来，放到内存里，如果等于0，说明读完了

//如果每次都没有读到，就要设置一个超时时间

while (true)

{

Thread.Sleep(ReceiveDelay);

if (serialPort.BytesToRead>0)//判断返回的报文的中间缓存区的字节数不为0，才可以读取接受报文

{

int count = serialPort.Read(buffer, 0, buffer.Length);

memoryStream.Write(buffer, 0, count);

}

else

{

if (memoryStream.Length>0)

{

break;

}

else if ((DateTime.Now-start).TotalMilliseconds>this.ReceiveRead)

{

return false;

}

}

}

receive=memoryStream.ToArray();

return true;

}

catch(Exception )

{

return false;

}

}