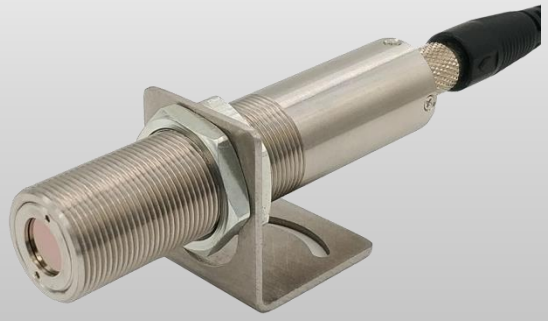
IRTS 红外温度传感器通信协议

DagaSensor

在线、实时、非接触、精确测量物体温度！



IRTSM 系列

- ★ 工业级非接触式红外测温传感器
- ★ ±1℃/1%测量精度
- ★ 从-40℃到 400℃多种测温范围可选
- ★ 响应时间仅 20ms(95%)
- ★ 发射率（0.1~1.5）可调
- ★ IP65 防护等级
- ★ RS485 和模拟量同时输出
- ★ 支持 Modbus 通信协议
- ★ 不锈钢材质，更坚固可靠

1 协议简介

该协议定义了 DagaSensor 公司 IRTS 红外温度传感器与上位计算机和 PLC 之间进行通信规则和内容。

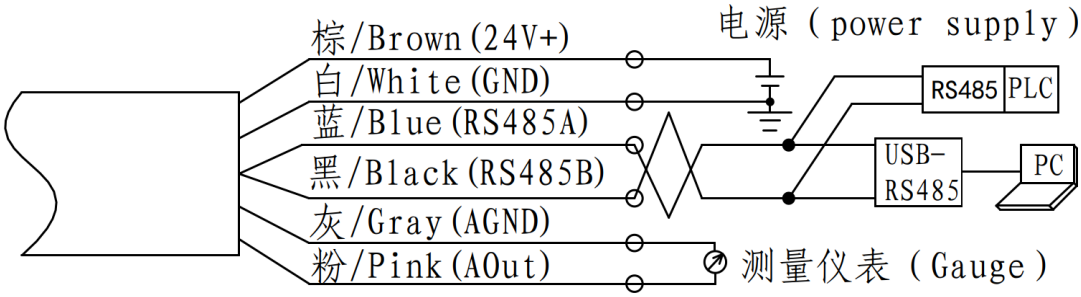
协议建立在 RS485 通信基础上，实现了 Modbus-RTU 协议，很好的兼容了 PC 和 PLC 两种主流的上位控制器。通过该协议，用户可以非常方便地对传感器进行参数设定、数据读取和存储，便于后期分析。

适用型号系列：(列表处于不断更新中)

IRTSS、ITRSM、IRTSH、IRTSU

2 接线图

典型的数字型红外温度传感器通信接线图如下：(该图仅说明 RS485 通信线缆的接法示例，由于传感器系列不同，线缆接法和颜色定义可能不同，详细线缆接法请参考对应型号的技术规格书)



连接方法：其他型号的接线图参考该图，将 RS485A（+）与 RS485B（-）线缆与目标 PC 或 PLC 适用屏蔽双绞线连接起来，然后将线缆的屏蔽层与传感器线缆屏蔽层连接起来、然后一并接入大地。线缆建议采用屏蔽双绞线，线芯粗于 AWG26.

东莞大镓传感技术有限公司
Dongguan DagaSensor Technology Co., Ltd.
http://www.dagasensor.com sales@dagasensor.com
地址：广东省东莞市松山湖园区礼智路3号生产力大厦421室
电话：+86 15013699186

3 Modbus-RTU 协议内容

该协议的通信方式是问答模式，上位机为主动控制器（Master），传感器为被动应答器（Slave），问答模式由主动控制器根据协议内容发送相应的请求，然后由指定 ID 的被动应答器响应请求，执行命令，反馈回数据或执行结果。

协议内容定义如下：

参数类别	寄存器地址	功能号	寄存器说明	单位	默认值	读写权限	值说明
实测变量区	0x0002	3/4	标定后温度值	0.01℃		只读	32 位有符号整数，温度=整数/100. 高地址为高 16 位，低地址为低 16 位。
	0x0004	3/4	感应器温度	0.01℃		只读	16 位有符号整数，温度=整数/100. 温度值为：(高字节*256+低字节)/100
	0x0005~9	3/4	未定义				
	0x000A	3/4	报警			只读	16 位无符号整数 bit0: 测量温度超过设定上限； bit1：测量温度超过设定下限； bit2：测量温度超过系统绝对上限； bit3：测量温度超过系统绝对下限； bit4：测温探头未连接； Bit5：测温探头高温报警； bit6：测温探头低温报警； bit7：电子盒高温报警； bit8：电子盒低温报警；
测量设置区	0x0100	3/4/6	发射率	0.001	950	读写	16 位无符号整数，100~1500
	0x0101	3/4/6	透射率	0.001	1000	读写	16 位无符号整数，100~1500
	0x0102	3/4/6	滤波时间	ms	100	读写	响应时间设置：20~5000 该时间决定内部计算温度数据的影响时间长度
	0x0103	3/4/6	信号处理方法		0	读写	在响应时间内采集的数据做如下处理后输出。 0：平均；1：中值滤波；2：大镓分位数滤波；3：峰值保持；4：谷值保持
	0x0104	3/4/6	指数权重λ	1/100	0	读写	控制信号处理完毕的数值是否再经过 EWMA 处理后输出。 0: 不启用 EWMA 10-100： 启用 EWMA，且λ等于设定值/100。如 25 表示λ=0.25
	0x0105	3/4/6	传感器使能控制		1	读写	Bit0:0：不输出模拟量；1：输出模拟量

参数类别	寄存器地址	功能号	寄存器说明	单位	默认值	读写权限	值说明
	0x0106	3/4/6	模拟输出量程下限	0.1℃	由型号指定	读写	上下限对应模拟输出值的最大值和最小值，此上下限不能超过系统上下限，且范围不能小于系统上下限的 0.1 倍，其默认值等于系统上下限。
	0x0107	3/4/6	模拟输出量程上限	0.1℃	由型号指定	读写	
	0x0108	3/4/6	报警下限	0.1℃	由型号指定	读写	设置报警上下限。
	0x0109	3/4/6	报警上限	0.1℃	由型号指定	读写	
	0x010A	3/4/16	红外温度标定系数 K0		0	读写	32 位浮点数，由用户指定的二次标定曲线系数。Y = K2 * X^2 + K1 * X + K0
	0x010C	3/4/16	红外温度标定系数 K1		1	读写	
	0x010E	3/4/16	红外温度标定系数 K2		0	读写	
通信设置区	0x0200	3/4/6	Modbus 通信地址		1	读写	1-100 注意:系统恢复出厂设置时 ID 会被置成 1，而网络中若同时存在多个相同 ID 的传感器时，通信会失效。因此，若多只传感器组成网络时，慎用重启恢复出厂设置功能。恢复出厂设置后，需要重启方能生效。
控制命令区	0x0205	6	用户命令			只写	写入相应数值表示相应控制功能。 100：重启系统 101：恢复出厂设置【慎用！】

注意事项：

1）协议命令不能跨区域。比如读命令，地址只能允许在相同的区地址内。比如批量读操作，实测变量区地址只能从 0x0002~0x000A，不能延伸到测量设置区。同样读取测量设置区数据时，不能延伸到通信设置区。

2）写命令的功能号是 6 的（单字写），如表格中标识支持功能号 6 的变量，仅支持 16 位写。也就是必须单独对这个变量进行写，不能垮到其他变量范围。

3）写命令的功能号是 16 的（多字写），如表格中标识支持功能号 16 的变量，仅支持 32 位写。也就是必须单独对这个 32 位变量进行写操作，不能垮其他变量范围。

4）恢复出厂设置后，必须重启才能生效，当前的通信 ID 在本次设置中依然有效，但重启后，才会启动新设定的 ID.

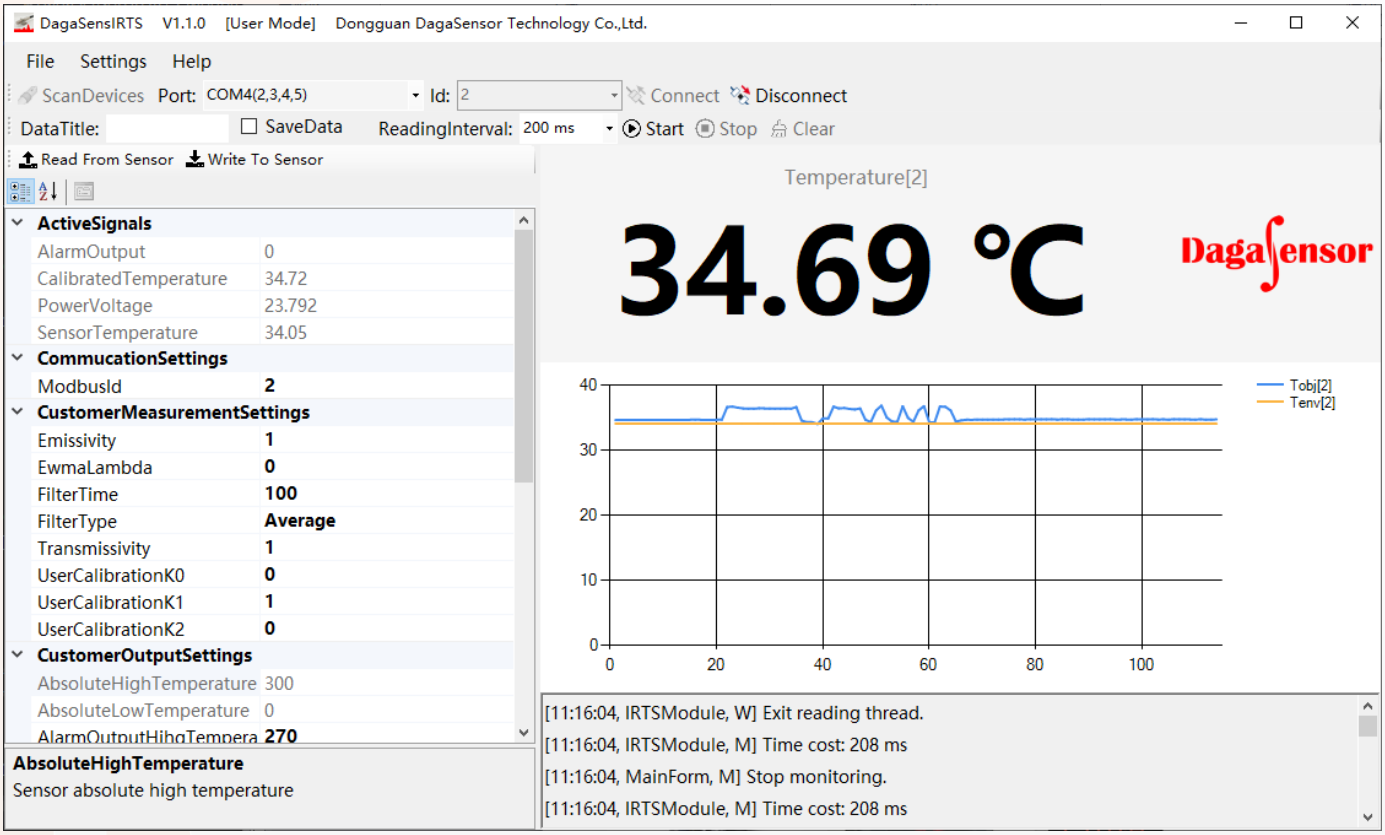
5）对响应时间的解释：传感器的实际硬件响应时间是 20ms. 当响应时间被设置为较大数时，传感器会将过采样的数据按照设定的信号处理方法进行处理，然后再输出。

6）严禁对以上表格指定通信地址以外的区域进行写操作，可能会引起严重故障。

4 传感器设置软件 DagaSensIRTS

DagaSensor 公司为用户提供了免费高效的软件 DagaSensIRTS. 通过 DagaSensIRTS 软件可以轻松管理和设置满足该协议的传感器。大多数用户需求可通过该软件完成设置和数据的收集。该软件支持多只传感器组成的网络轮流访问模式获取数据。对于需要把传感器集成为其他系统的用法，可以参照协议内容

表对传感器进行命令和数据交互，实现相应的功能。



软件主要功能

- 自动发现传感器
- 读取并记录传感器温度数据
- 设置通信 ID
- 设置目标被测物体的发射率
- 设置模拟输出范围
- 关闭模拟输出
- 再次根据现场标定传感器
- 设置信号处理方式

5 编程参考

该协议使用大端模式传输数据，使用 CRC16 校验，因此用户获取数据时，可参考如下代码解析数据。

（C#实现）

```
#region crc data define
byte[] auchCRCHi =
{
    0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81,
    0x40, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0,
    0x80, 0x41, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x00, 0xC1, 0x81, 0x40, 0x01,
    0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0, 0x80, 0x41,
    0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x00, 0xC1, 0x81,
    0x40, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0,
    0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01,
    0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40,
```

```
            0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81,
            0x40, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0,
            0x80, 0x41, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x00, 0xC1, 0x81, 0x40, 0x01,
            0xC0, 0x80, 0x41, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41,
            0x00, 0xC1, 0x81, 0x40, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81,
            0x40, 0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0,
            0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01,
            0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41,
            0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81,
            0x40
        };

        byte[] auchCRCLo =
        {
            0x00, 0xC0, 0xC1, 0x01, 0xC3, 0x03, 0x02, 0xC2, 0xC6, 0x06, 0x07, 0xC7, 0x05, 0xC5, 0xC4,
            0x04, 0xCC, 0x0C, 0x0D, 0xCD, 0x0F, 0xCF, 0xCE, 0x0E, 0x0A, 0xCA, 0xCB, 0x0B, 0xC9, 0x09,
            0x08, 0xC8, 0xD8, 0x18, 0x19, 0xD9, 0x1B, 0xDB, 0xDA, 0x1A, 0x1E, 0xDE, 0xDF, 0x1F, 0xDD,
            0x1D, 0x1C, 0xDC, 0x14, 0xD4, 0xD5, 0x15, 0xD7, 0x17, 0x16, 0xD6, 0xD2, 0x12, 0x13, 0xD3,
            0x11, 0xD1, 0xD0, 0x10, 0xF0, 0x30, 0x31, 0xF1, 0x33, 0xF3, 0xF2, 0x32, 0x36, 0xF6, 0xF7,
            0x37, 0xF5, 0x35, 0x34, 0xF4, 0x3C, 0xFC, 0xFD, 0x3D, 0xFF, 0x3F, 0x3E, 0xFE, 0xFA, 0x3A,
            0x3B, 0xFB, 0x39, 0xF9, 0xF8, 0x38, 0x28, 0xE8, 0xE9, 0x29, 0xEB, 0x2B, 0x2A, 0xEA, 0xEE,
            0x2E, 0x2F, 0xEF, 0x2D, 0xED, 0xEC, 0x2C, 0xE4, 0x24, 0x25, 0xE5, 0x27, 0xE7, 0xE6, 0x26,
            0x22, 0xE2, 0xE3, 0x23, 0xE1, 0x21, 0x20, 0xE0, 0xA0, 0x60, 0x61, 0xA1, 0x63, 0xA3, 0xA2,
            0x62, 0x66, 0xA6, 0xA7, 0x67, 0xA5, 0x65, 0x64, 0xA4, 0x6C, 0xAC, 0xAD, 0x6D, 0xAF, 0x6F,
            0x6E, 0xAE, 0xAA, 0x6A, 0x6B, 0xAB, 0x69, 0xA9, 0xA8, 0x68, 0x78, 0xB8, 0xB9, 0x79, 0xBB,
            0x7B, 0x7A, 0xBA, 0xBE, 0x7E, 0x7F, 0xBF, 0x7D, 0xBD, 0xBC, 0x7C, 0xB4, 0x74, 0x75, 0xB5,
            0x77, 0xB7, 0xB6, 0x76, 0x72, 0xB2, 0xB3, 0x73, 0xB1, 0x71, 0x70, 0xB0, 0x50, 0x90, 0x91,
            0x51, 0x93, 0x53, 0x52, 0x92, 0x96, 0x56, 0x57, 0x97, 0x55, 0x95, 0x94, 0x54, 0x9C, 0x5C,
            0x5D, 0x9D, 0x5F, 0x9F, 0x9E, 0x5E, 0x5A, 0x9A, 0x9B, 0x5B, 0x99, 0x59, 0x58, 0x98, 0x88,
            0x48, 0x49, 0x89, 0x4B, 0x8B, 0x8A, 0x4A, 0x4E, 0x8E, 0x8F, 0x4F, 0x8D, 0x4D, 0x4C, 0x8C,
            0x44, 0x84, 0x85, 0x45, 0x87, 0x47, 0x46, 0x86, 0x82, 0x42, 0x43, 0x83, 0x41, 0x81, 0x80,
            0x40
        };
#endregion

        /// <summary>
        /// CRC16校验
        /// </summary>
        /// <param name="data"></param>
        /// <param name="ifrom"></param>
        /// <param name="ito"></param>
        /// <returns></returns>
        public ushort CRC16(byte[] data, int ifrom, int ito)
        {
            byte uchCRCHi = 0xff;
            byte uchCRCLo = 0xff;
            byte uindex;
            for(int i = ifrom; i <= ito; i++)
            {
                uindex = (byte)(uchCRCHi ^ data[i]);
                uchCRCHi = (byte)(uchCRCLo ^ auchCRCHi[uindex]);
                uchCRCLo = auchCRCLo[uindex];
            }
            return (ushort)((uchCRCHi << 8) | uchCRCLo);
        }

        /// <summary>
        /// 从大端数组中获取short型数据
        /// </summary>
```

```
        /// <param name="data"></param>
        /// <param name="sindex"></param>
        /// <returns></returns>
        public static short GetShortFromBigEndianArray(byte[] data, int sindex)
        {
            byte[] bytes = new byte[2];
            if (data.Length >= sindex + 2)
            {
                Array.Copy(data, sindex, bytes, 0, 2);
                Array.Reverse(bytes);
                return BitConverter.ToInt16(bytes, 0);
            }
            return 0;
        }

        /// <summary>
        /// 从大端数组中获取int32型数据
        /// </summary>
        /// <param name="data"></param>
        /// <param name="sindex"></param>
        /// <returns></returns>
        public static int GetIntFromBigEndianArray(byte[] data, int sindex)
        {
            byte[] bytes = new byte[4];
            if (data.Length >= sindex + 4)
            {
                Array.Copy(data, sindex, bytes, 0, 4);
                Array.Reverse(bytes);
                return BitConverter.ToInt32(bytes, 0);
            }
            return 0;
        }

        /// <summary>
        /// 从大端数组中获取float数据
        /// </summary>
        /// <param name="data"></param>
        /// <param name="sindex"></param>
        /// <returns></returns>
        public static float GetFloatFromBigEndianArray(byte[] data, int sindex)
        {
            byte[] bytes = new byte[4];
            if (data.Length >= sindex + 4)
            {
                Array.Copy(data, sindex, bytes, 0, 4);
                Array.Reverse(bytes);
                return BitConverter.ToSingle(bytes, 0);
            }
            return 0;
        }
```

文件名：DGS_IRTS_CommProtocol

文件版本：V1.0

发布日期：2020 年 8 月 3 日

作者：SK