

通信协议

COMMUNICATION PROTOCOL

成就精密光电测量美好未来

Striving for the Bright Future of Precision Optoelectronic Measurement.

1. 出厂默认通讯参数	1
2. 通讯格式	1
2.1 ASCII格式	1
2.1.1 温控器通用参数指令.....	1
2.1.2 温控器通道控制指令.....	1
2.2 Modbus-RTU格式	1
3. 通讯指令	2
3.1 目标温度配置	2
3.1.1 查询/设置目标温度.....	2
3.2 温度传感器参数配置	3
3.2.1 查询/设置实际温度.....	3
3.2.2 查询传感器电阻值.....	3
3.2.3 查询/设置温度求解模型选择.....	3
3.2.4 查询/设置NTC电阻的B值.....	3
3.2.5 查询/设置NTC电阻的R0(25°C) 值.....	3
3.2.6 查询/设置温控器NTC通道内部参考电阻.....	4
3.2.7 查询/设置PT1000电阻的R0值.....	4
3.2.8 查询/设置PT1000电阻的Callendar-van-Dusen系数 A.....	4
3.2.9 查询/设置PT1000电阻的Callendar-van-Dusen系数 B.....	4
3.2.10 查询/设置PT1000电阻的Callendar-van-Dusen系数 C.....	4
3.2.11 查询/设置温控器PT1000通道内部参考电阻.....	4
3.2.12 查询/设置MF501电阻的A值.....	4
3.2.13 查询/设置MF501电阻的B值.....	5
3.2.14 查询/设置MF501电阻的C值.....	5
3.2.15 查询/设置A0~A7系数的浮点数部分.....	5
3.2.16 查询/设置A0~A7系数的指数部分.....	5
3.2.17 查询/设置过温保护高阈值.....	5

3.2.18 查询/设置过温保护低阈值.....	5
3.2.19 查询/设置传感器开路短路输出保护功能.....	6
3.2.20 查询/设置开机电源输出模式.....	6
3.2.21 查询输出电流.....	6
3.2.22 设置最大输出电流.....	6
3.3 温控器输出配置.....	6
3.3.1 查询/设置最大输出占空比.....	6
3.3.2 查询/设置输出使能.....	6
3.3.3 查询/设置开机延迟输出.....	7
3.3.4 查询/设置输出模式.....	7
3.3.5 查询/设置输出极性.....	7
3.3.6 查询/设置输出电压百分比.....	7
3.3.7 查询/设置温度变化斜率.....	7
3.3.8 查询/设置正向启动电压.....	7
3.3.9 查询/设置反向启动电压.....	7
3.3.10 查询/设置PWM引脚输出频率.....	8
3.3.11 查询/设置传感器过温保护模式.....	8
3.3.12 查询/设置温控器模式选择.....	8
3.4 PID配置.....	8
3.4.1 查询/设置PID中的P.....	8
3.4.2 查询/设置PID中的I.....	8
3.4.3 查询/设置PID中的D.....	8
3.4.4 查询/设置PID自整定.....	9
3.5 温控器通用参数配置.....	9
3.5.1 查询当前温控器型号.....	9
3.5.2 查询固件版本号.....	9
3.5.3 查询/设置温控器485地址.....	9

3.5.4 查询/设置UART TTL波特率.....	9
3.5.5 查询/设置RS485波特率.....	9
3.5.6 查询温控器自身温度.....	10
3.5.7 查询/设置温控器自身过温阈值.....	10
3.5.8 查询错误代码.....	10
3.5.9 恢复出厂设置.....	11
3.6 其它数据查询.....	11
3.6.1 查询：一次性查询全面数据.....	11
3.6.2 查询：一次性查询关键数据.....	12
4. 指令预览表.....	12
附录1 温度解算模型和多项式修正.....	16
01 NTC热敏电阻温度传感器.....	16
02 PT铂电阻温度传感器.....	16
03 多项式温度校正.....	17
案例说明：NTC温度传感器配合温控器的多项式温度校正.....	17
附录2 编程案例.....	18
01 C++编程案例.....	18
02 Python编程案例.....	24

1. 出厂默认通讯参数

参数	值
波特率 (默认)	38400 (TTL接口) / 9600 (RS485接口)
数据位	8
停止位	1
奇偶校验位	无
地址 (默认)	1

当前温控器配套上位机软件仅支持与波特率为38400的串口通讯，若需通过RS485实现通讯，需设置485波特率为38400。

2. 通讯格式

注意：必须严格依照指令格式操作进行通讯，否则设备无法响应。

2.1 ASCII格式

2.1.1 温控器通用参数指令

示例：查询/设置 PWM 引脚输出频率 FPWM

读：	发送：FPWM=?@
	返回：OKFPWM=2@\r\n
写：	发送：FPWM=2@
	返回：OKFPWM=2@\r\n

2.1.2 温控器通道控制指令

示例：查询/设置目标温度，通道一ASCII码通信（通道二将TC1改成TC2即可）

读：	发送：TC1:TG=?@
	返回：OKTC1:TG=2500000@\r\n
写：	发送：TC1:TG=2500000@
	返回：OKTC1:TG=2500000@\r\n

2.2 Modbus-RTU格式

1. 本产品支持 03H,10H 功能码格式
2. 通道控制指令：此协议中以通道一为例，其地址范围为0x1000H~0x1999H，通道二在通道一的基础上地址偏移0x1000H，以此类推。

功能码 (HEX)	说明
0x03H	读单个或多个寄存器

0x10H	写单个或多个寄存器
-------	-----------

●读：

上位机发送格式 (HEX)：

站号	功能码	寄存器地址		寄存器数量		CRC	
		高字节	低字节	高字节	低字节	高字节	低字节
01	03	10	00	00	02	C0	CB

下位机应答 (HEX)：

站号	功能码	字节数量	寄存器数据				CRC	
			数据1	数据2	数据3	数据4	高字节	低字节
01	03	04	00	26	25	A0	01	10

●写：

上位机发送格式 (HEX)：

站号	功能码	寄存器地址		寄存器数量		字节数量	寄存器数据				CRC	
		高字节	低字节	高字节	低字节		数据1	数据2	数据3	数据4	高字节	低字节
01	10	10	00	00	02	04	00	26	25	A0	C5	4C

下位机应答 (HEX)：

站号	功能码	寄存器地址		寄存器数量		CRC	
		高字节	低字节	高字节	低字节	高字节	低字节
01	10	10	00	00	02	45	08

●示例：查询/设置目标温度

写：	发送(HEX): 01 10 10 00 00 02 04 00 26 25 A0 C5 4C 返回(HEX): 01 10 10 00 00 02 45 08
注释：	发送：01 为设备地址/站号，10 为写命令，10 00为该功能寄存器地址，00 02(2个寄存器数量)，04表示4个字节，00 26 25 A0代表2500000(从高位到低位)，C5 4C为25°C时的CRC校验码 (低八位先发送)。 返回：01 为设备地址/站号，10为写命令确认，10 00 寄存器写入地址确认，00 02寄存器写入数量确认，45 08 为此行命令CRC校验码(低八位先发送)。

3. 通讯指令

3.1 目标温度配置

3.1.1 查询/设置目标温度

指令类型	ASCII码指令	寄存器地址	数据类型	寄存器数量	权限	数据范围
通道控制指令	TG	0x1000H	int32	2	读/写	-40000000~100000000

说明：2500000代表就是25.00000°C。

3.2 温度传感器参数配置

温度解算模型和多项式修正，详情见附录1。

3.2.1 查询/设置实际温度

指令类型	ASCII码指令	寄存器地址	数据类型	寄存器数量	权限	数据范围
通道控制指令	TCADJTEMP	0x1002H	int32	2	读/写	-40000000~100000000

说明：2500000代表就是25.00000°C。

3.2.2 查询传感器电阻值

指令类型	ASCII码指令	寄存器地址	数据类型	寄存器数量	权限	数据范围
通道控制指令	RESISTOR	0x1004H	uint64	4	只读	1~500000000000

说明：10000000000代表当前通道传感器的电阻值为10.000000kΩ。

3.2.3 查询/设置温度求解模型选择

指令类型	ASCII码指令	寄存器地址	数据类型	寄存器数量	权限	数据范围
通道控制指令	POLYOMIAL	0x1300H	uint16	1	读/写	0~3

说明：0：B值模型(参数：NTC R_0 和B值，多项式温度修正功能启用)。
 1：PT模型(参数：PT R_0 、A、B和C值，多项式温度修正功能启用)。
 2：Steinhart-Hart(S-H) 模型(参数： A_0 、 A_1 、 A_2 、 A_3 和 A_4 值，多项式温度修正功能停用)。
 3：MF501模型 (参数：MF501 A、B、C，多项式温度修正功能停用)。

注释：多项式温度修正功能： A_0 - A_7 ， $T(\text{修后}) = T(\text{测}) + A_0 + A_1 * T(\text{测}) + A_2 * T(\text{测})^2 + A_3 * T(\text{测})^3 + A_4 * T(\text{测})^4 + A_5 * T(\text{测})^5 + A_6 * T(\text{测})^6 + A_7 * T(\text{测})^7$ 。

3.2.4 查询/设置NTC电阻的B值

指令类型	ASCII码指令	寄存器地址	数据类型	寄存器数量	权限	数据范围
通道控制指令	BX	0x1301H	uint32	2	读/写	100000~5000000

说明：395000代表B值就是3950.00。B值取决于具体的热敏电阻材料和制造工艺，实际以传感器厂商标注为准。

3.2.5 查询/设置NTC电阻的 $R_0(25^\circ\text{C})$ 值

指令类型	ASCII码指令	寄存器地址	数据类型	寄存器数量	权限	数据范围
通道控制指令	RP	0x1303H	uint32	2	读/写	1~9000000

说明：10000代表NTC电阻在25°C时的标准值为10.000kΩ。

3.2.6 查询/设置温控器NTC通道内部参考电阻

指令类型	ASCII码指令	寄存器地址	数据类型	寄存器数量	权限	数据范围
通道控制指令	NTCRP	0x1305H	uint64	4	读/写	1~11000000000

说明：10000000000代表NTC电阻的内部参考电阻为10.000000000kΩ。

3.2.7 查询/设置PT1000电阻的R₀值

指令类型	ASCII码指令	寄存器地址	数据类型	寄存器数量	权限	数据范围
通道控制指令	PT1000RP	0x1309H	uint32	2	读/写	0~10000000

说明：1000000代表PT1000电阻在0°C时的标称阻值为1.000kΩ。

3.2.8 查询/设置PT1000电阻的Callendar-van-Dusen系数 A

指令类型	ASCII码指令	寄存器地址	数据类型	寄存器数量	权限	数据范围
通道控制指令	PTA	0x130BH	int32	2	读/写	-9000000~9000000

说明：3908300代表PT模型中的A值为3.9083E-3。系数A是 Callendar-Van Dusen 方程的线性项系数，描述铂电阻在温度变化时电阻的基础线性变化趋势，是决定电阻-温度关系的最主要系数。ABC系数默认值为标准PT传感器的核心系数值，由铂材料固有特性决定。各个标准规定的铂电阻略有不同。此为DIN EN 60751中的系数值。

3.2.9 查询/设置PT1000电阻的Callendar-van-Dusen系数 B

指令类型	ASCII码指令	寄存器地址	数据类型	寄存器数量	权限	数据范围
通道控制指令	PTB	0x130DH	int32	2	读/写	-9000000~9000000

说明：-577500代表PT模型中的B值为-5.775E-7。系数B是二次项修正系数，用于补偿铂电阻在0°C~850°C正温区的非线性偏差。

3.2.10 查询/设置PT1000电阻的Callendar-van-Dusen系数 C

指令类型	ASCII码指令	寄存器地址	数据类型	寄存器数量	权限	数据范围
通道控制指令	PTC	0x130FH	int32	2	读/写	-90000~90000

说明：-41830代表PT模型中的C值为-4.183E-12。系数C是三次项修正系数，用于负温区，补偿该区间铂电阻更显著的非线性偏差。

3.2.11 查询/设置温控器PT1000通道内部参考电阻

指令类型	ASCII码指令	寄存器地址	数据类型	寄存器数量	权限	数据范围
通道控制指令	PTRP	0x1311H	uint64	4	读/写	1~2100000000

说明：2000000000代表PT1000电阻的内部参考电阻为2.000000kΩ。

3.2.12 查询/设置MF501电阻的A值

指令类型	ASCII码指令	寄存器地址	数据类型	寄存器数量	权限	数据范围
通道控制指令						

通道控制指令	MF501A	0x1342H	int64	2	读/写	-10000000000000~1000000000000
--------	--------	---------	-------	---	-----	-------------------------------

说明：1000000代表A值是1.000000，方程中的常数项，是拟合曲线的基础参数，影响低温段电阻与温度的对应关系。需要注意的是，不同厂商生产的MF501热敏电阻，ABC 值可能因型号规格不同而有差异，具体数值需参考该产品的数据手册。

3.2.13 查询/设置MF501电阻的B值

指令类型	ASCII码指令	寄存器地址	数据类型	寄存器数量	权限	数据范围
通道控制指令	MF501B	0x1346H	int64	2	读/写	-10000000000000~1000000000000

说明：1000000代表B值是1.000000，主要影响中温段的电阻与温度的对应关系，也是常见的“B 值”的基础来源。

3.2.14 查询/设置MF501电阻的C值

指令类型	ASCII码指令	寄存器地址	数据类型	寄存器数量	权限	数据范围
通道控制指令	MF501C	0x134AH	int64	2	读/写	-10000000000000~1000000000000

说明：1000000代表C值是1.000000，高阶修正项系数，主要用于修正高温段的拟合精度，让高温下的电阻计算更准确（低精度场景可能忽略 C 值，仅用 A 和 B）。

3.2.15 查询/设置A₀~A₇系数的浮点数部分

指令类型	ASCII码指令	寄存器地址	数据类型	寄存器数量	权限	数据范围
通道控制指令	POLA0	0x1315H	int64	4	读/写	-999999999999~999999999999

说明：A₀系数科学记数法的浮点数部分：999999999999代表9.999999999999（后A₁~A₇同）；
A_n的 ASCII 指令：POLAn（n为0~7）；
Modbus A₀~A₇地址：0x1315H, 0x131AH, 0x131FH, 0x1324H, 0x1329H, 0x132EH, 0x1333H, 0x1338H。

3.2.16 查询/设置A₀~A₇系数的指数部分

指令类型	ASCII码指令	寄存器地址	数据类型	寄存器数量	权限	数据范围
通道控制指令	POLEA0	0x1319H	int16	1	读/写	-100~100

说明：A₀系数科学记数法的指数部分：100。此时A₀=9.999999999999E+100（后A₁~A₇同）；
A_n的 ASCII指令：POLEAn（n为0~7）；
Modbus A₀~A₇地址：0x1319H, 0x131EH, 0x1323H, 0x1328H, 0x132DH, 0x1332H, 0x1337H, 0x133CH。

3.2.17 查询/设置过温保护高阈值

指令类型	ASCII码指令	寄存器地址	数据类型	寄存器数量	权限	数据范围
通道控制指令	OVERTEMPUP	0x133DH	int32	2	读/写	-300000000~500000000

说明：当测温温度大于设置的温度最大值时，会按“3.3.12 查询/设置传感器过温保护模式”的设置停止/继续输出。500000000代表5000°C。

3.2.18 查询/设置过温保护低阈值

指令类型	ASCII码指令	寄存器地址	数据类型	寄存器数量	权限	数据范围
------	----------	-------	------	-------	----	------

通道控制指令	OVERTEMPLOWER	0x133FH	int32	2	读/写	-300000000~500000000
说明： 当测温温度小于设置的温度最小值时，会按“3.3.12 查询/设置传感器过温保护模式”的设置停止/继续输出。 300000000代表3000°C。						

3.2.19 查询/设置传感器开路短路输出保护功能

指令类型	ASCII码指令	寄存器地址	数据类型	寄存器数量	权限	数据范围
通道控制指令	ONSENSOR	0x110CH	int16	1	读/写	0~1
说明： 0：不操作（可通过外部通信指令“3.2.1 查询/设置实际温度”给温度）； 1：当未接传感器或传感器短路时，温控器不输出电压。						

3.2.20 查询/设置开机电源输出模式

指令类型	ASCII码指令	寄存器地址	数据类型	寄存器数量	权限	数据范围
通道控制指令	POWERMODE	0x1110H	uint16	1	读/写	0~2
说明： 0：开机跟随上一次输出状态； 1：上电输出； 2：上电关闭。						

3.2.21 查询输出电流

指令类型	ASCII码指令	寄存器地址	数据类型	寄存器数量	权限	数据范围
通道控制指令	CURRENT	0x1111H	uint16	1	只读	0~900000000
说明： 3256代表目前输出电流为3.256A。						

3.2.22 设置最大输出电流

指令类型	ASCII码指令	寄存器地址	数据类型	寄存器数量	权限	数据范围
通道控制指令	SETCURRENT	0x1112H	uint16	1	读写	5~150
说明： 10代表1.0A。						

3.3 温控器输出配置

3.3.1 查询/设置最大输出占空比

指令类型	ASCII码指令	寄存器地址	数据类型	寄存器数量	权限	数据范围
通道控制指令	LIMITED	0x110EH	int16	1	读/写	0~90
说明： 50代表当前通道最大输出电压为输入电压的50%。						

3.3.2 查询/设置输出使能

指令类型	ASCII码指令	寄存器地址	数据类型	寄存器数量	权限	数据范围
通道控制指令	ENABLE	0x1100H	uint16	1	读/写	0~1

说明：0：不使能输出电压，1：使能输出电压。

3.3.3 查询/设置开机延迟输出

指令类型	ASCII码指令	寄存器地址	数据类型	寄存器数量	权限	数据范围
通道控制指令	STARTUPDELAY	0x110FH	uint16	1	读/写	3~180

说明：x：开机延迟x秒后启动（仅在断电前为输出状态时，下一次开机才会生效延迟启动）。

3.3.4 查询/设置输出模式

指令类型	ASCII码指令	寄存器地址	数据类型	寄存器数量	权限	数据范围
通道控制指令	MODE	0x1101H	uint16	1	读/写	0~3

说明：0：双向模式，1：制冷模式，2：加热模式，3：由通信设置输出电压百分比。

3.3.5 查询/设置输出极性

指令类型	ASCII码指令	寄存器地址	数据类型	寄存器数量	权限	数据范围
通道控制指令	PIDPOL	0x1102H	uint16	1	读/写	0~1

说明：0：正向极性输出，1：反向极性输出。

3.3.6 查询/设置电压输出百分比

指令类型	ASCII码指令	寄存器地址	数据类型	寄存器数量	权限	数据范围
通道控制指令	PWMDUTY	0x1103H	int64	4	读/写	-2000000~2000000

说明：200000代表当前通道实际输出电压百分比为10%。

3.3.7 查询/设置温度变化斜率

指令类型	ASCII码指令	寄存器地址	数据类型	寄存器数量	权限	数据范围
通道控制指令	SPEED	0x1108H	uint16	1	读/写	0~10000

说明：温度变化斜率，单位 $^{\circ}\text{C}/\text{s}$ 。0 代表不限制温度变化斜率，1000 代表 $1^{\circ}\text{C}/\text{s}$ 。

设置范围：0~ $10^{\circ}\text{C}/\text{s}$ 。调节分辨率：0.001 $^{\circ}\text{C}/\text{s}$ 。

3.3.8 查询/设置正向启动电压

指令类型	ASCII码指令	寄存器地址	数据类型	寄存器数量	权限	数据范围
通道控制指令	FDEADV	0x110AH	uint16	1	读/写	0~400

说明：200代表1%的正向输出初始电压百分比。

3.3.9 查询/设置反向启动电压

指令类型	ASCII码指令	寄存器地址	数据类型	寄存器数量	权限	数据范围
通道控制指令	BDEADV	0x110BH	uint 16	1	读/写	0~400

说明：200代表-1%的反向输出初始电压百分比。

3.3.10 查询/设置PWM引脚输出频率

指令类型	ASCII码指令	寄存器地址	数据类型	寄存器数量	权限	数据范围
通用参数指令	FPWM	0x000DH	uint16	1	读/写	0~3

说明：0：PWM输出频率为0.5HZ； 1：PWM输出频率为1HZ；
2：PWM输出频率为10HZ； 3：PWM输出频率为100HZ。

3.3.11 查询/设置传感器过温保护模式

指令类型	ASCII码指令	寄存器地址	数据类型	寄存器数量	权限	数据范围
通用参数指令	OVERTTEMP	0x000BH	uint16	1	读/写	0~1

说明：0：当传感器温度大于高阈值或者低于低阈值时，TEC端的输出继续；
1：当传感器温度大于高阈值或者低于低阈值时，TEC端的输出关闭。

3.3.12 查询/设置温控器模式选择

指令类型	ASCII码指令	寄存器地址	数据类型	寄存器数量	权限	数据范围
通用参数指令	CONTMODE	0x0004H	int16	1	读/写	0~3

说明：0：各通道独立控制（默认值）；
1：通道1的实际目标温度=通道2的测量温度+通道1的设定目标温度；
2：通道2的电压/PWM输出跟随通道1的电压/PWM输出；
3：通道1的实际目标温度=通道2的测量温度+通道1的设定目标温度，通道2的电压/PWM输出跟随通道1的电压/PWM输出。
注意：实用CONTMODE=2和3模式时，应该在通道2传感器接口接入任意电阻温度传感器。

3.4 PID配置

3.4.1 查询/设置PID中的P

指令类型	ASCII码指令	寄存器地址	数据类型	寄存器数量	权限	数据范围
通道控制指令	KP	0x1200H	uint32	2	读/写	0~9000000

说明：PID公式中P的系数。

3.4.2 查询/设置PID中的I

指令类型	ASCII码指令	寄存器地址	数据类型	寄存器数量	权限	数据范围
通道控制指令	KI	0x1202H	uint32	2	读/写	0~9000000

说明：PID公式中I的系数。

3.4.3 查询/设置PID中的D

指令类型	ASCII码指令	寄存器地址	数据类型	寄存器数量	权限	数据范围
------	----------	-------	------	-------	----	------

通道控制指令	KD	0x1204H	uint32	2	读/写	0~9000000
说明：PID公式中D的系数。						

3.4.4 查询/设置PID自整定

指令类型	ASCII码指令	寄存器地址	数据类型	寄存器数量	权限	数据范围
通道控制指令	AUTOPID	0x1107H	uint16	1	读/写	0~2
说明：0：非PID自整定和非PID实时自动优化； 1：PID自整定； 2：PID实时自动优化（预留指令）。						

3.5 温控器通用参数配置

3.5.1 查询当前温控器型号

指令类型	ASCII码指令	寄存器地址	数据类型	寄存器数量	权限	数据范围
通用参数指令	TEC	0x0001H	uint16	1	只读	0~255
说明：1：103；2：207L；3：207；4：215L；5：215；6：215Pro；7：107L；8：107；9：115L；10：115；11：115Pro 12：100L；13：100；14：100Pro；15：403L；16：403；17：403Pro；18：415L；19：415；20：603L；21：603 22：615L；23：615；24：615Pro；25：803L；26：803；27：815L；28：815；29：815Pro；30：203L；31：203						

3.5.2 查询固件版本号

指令类型	ASCII码指令	寄存器地址	数据类型	寄存器数量	权限	数据范围
通用参数指令	FPV	0x000CH	uint16	1	只读	100~9999
说明：100代表版本号为1.0.0。						

3.5.3 查询/设置温控器485地址

指令类型	ASCII码指令	寄存器地址	数据类型	寄存器数量	权限	数据范围
通用参数指令	ADDRESS	0x0002H	uint16	1	读/写	0~255
说明：温控器设备485通讯地址。						

3.5.4 查询/设置UART TTL波特率

指令类型	ASCII码指令	寄存器地址	数据类型	寄存器数量	权限	数据范围
通用参数指令	BOUNDTABLEONE	0x0008H	uint16	1	读/写	0~7
说明：0：4800；1：9600；2：19200；3：38400；4：57600；5：115200；6：230400；7：460800。						

3.5.5 查询/设置RS485波特率

指令类型	ASCII码指令	寄存器地址	数据类型	寄存器数量	权限	数据范围
通用参数指令	BOUNDTABLETWO	0x0009H	uint16	1	读/写	0~7

说明: 0: 4800; 1: 9600; 2: 19200; 3: 38400; 4: 57600; 5: 115200; 6: 230400; 7: 460800。如需与温控器配套软件进行通讯, 修改波特率为38400。

3.5.6 查询温控器自身温度

指令类型	ASCII码指令	寄存器地址	数据类型	寄存器数量	权限	数据范围
通用参数指令	SINTERIORTEMP	0x0003H	int16	1	只读	-20~120

说明: 20代表温控器自身温度为20°C。

3.5.7 查询/设置温控器自身过温阈值

指令类型	ASCII码指令	寄存器地址	数据类型	寄存器数量	权限	数据范围
通用参数指令	OVERTVPT	0x000AH	uint16	1	读/写	40~100

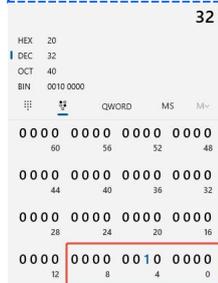
说明: 70代表温度控制器的自身温度到达70后开始逐步减低输出功率。

3.5.8 查询错误代码

指令类型	ASCII码指令	寄存器地址	数据类型	寄存器数量	权限	数据范围
通用参数指令	ERRORCODE	0x0007H	uint16	1	只读	按Bit位定义

说明: 温控器通过 16 位二进制状态字 (十进制数值) 反映系统运行状态, 各 bit 位定义如下:

状态	位编号	触发条件
温控器系统错误位	Bit0	高温报警 (温度超过温控器自身过温阈值, 限制输出)
	Bit1	过温报警 (温度超过温控器内部设定最大温度值, 停止输出)
	Bit2	电压过低报警(电压小于 7V)
	Bit3	电压过高报警(电压大于 30V)
通道一错误位	Bit4	/
	Bit5	通道 1 传感器温度大于高阈值或者低于低阈值
	Bit6	输出电流达到设定最大电流(限流中)
通道二错误位	Bit7	/
	Bit8	/
	Bit9	通道 2 传感器温度大于高阈值或者低于低阈值
	Bit10	通道 2 输出电流达到设定最大电流(限流中)
	Bit11	/



如值为 32 时, Bit5 为 1, 错误为通道 1 传感器温度大于高阈值或者低于低阈值。

3.5.9 恢复出厂设置

指令类型	ASCII码指令	寄存器地址	数据类型	寄存器数量	权限	数据范围
通用参数指令	RESET	0x0000H	uint16	1	只写	1

说明：恢复出厂设置。

3.6 其它数据查询

3.6.1 查询：一次性查询全面数据

指令类型	ASCII码指令	寄存器地址	数据类型	寄存器数量	权限	数据范围
通用参数指令	INQUIRE	/	/	/	/	1

说明：

发送：INQUIRE=1@

应答：

OKTC1:TG=2500000@TC2:TG=2500000@OKTC1:LIMITED=30@TC2:LIMITED=30@OKTC1:MODE=0@TC2:MODE=0@OKTC1:ENABLE=0@TC2:ENABLE=0@OKTC1:KP=3000@TC2:KP=3000@OKTC1:KI=150@TC2:KI=150@OKTC1:KD=0@TC2:KD=0@OKTC1:RP=10000@TC2:RP=10000@OKTC1:BX=395000@TC2:BX=395000@OKTEC=215OKTC1:PT1000RP=1000000@TC2:PT1000RP=1000000@OKTC1:CHRATIO=100@TC2:CHRATIO=100@OKTC1:SPEED=0@TC2:SPEED=0@OKTC1:STEADYIOB=0@TC2:STEADYIOB=0@OKTC1:OVERTEMPUP=500000000@TC2:OVERTEMPUP=500000000@OKTC1:OVERTEMPLOWER=-300000000@TC2:OVERTEMPLOWER=-300000000@OKTC1:FDEADV=0@TC2:FDEADV=0@OKTC1:BDEADV=0@TC2:BDEADV=0@OKTC1:NTCRP=1000000000@TC2:NTCRP=1000000000@OKTC1:PTRP=1000000000@TC2:PTRP=1000000000@OKTC1:PTA=3908300@TC2:PTA=3908300@OKTC1:PTB=-5775000@TC2:PTB=-5775000@OKTC1:PTC=-4183000@TC2:PTC=-4183000@OKTC1:PIDPOL=0@TC2:PIDPOL=0@

代表目前通道一的目标温度为25度，通道二的目标温度为25度。通道一最大的输出为百分之30，通道二最大的输出为百分之30。通道一的输出模式为双向模式，通道二的输出模式为双向模式。通道一的输出使能为关闭，通道二的输出使能为关闭。通道一的PID的P为3000，通道二的PID的P为3000。通道一的PID的I为150，通道二的PID的I为150。通道一的PID的D为0，通道二的PID的D为0。一通道NTC电阻的标准电阻10KΩ，二通道NTC电阻的标准电阻10KΩ。通道一NTC电阻的B值为3950，二通道NTC电阻的B值为3950。目前温控器的版本号为215。通道一PT1000的标准电阻1KΩ，通道二PT1000的标准电阻1KΩ。通道一制冷系数/制热系数为1.00，通道二制冷系数/制热系数为1.00。通道一的温度变化斜率变化不限制，通道二的温度变化斜率变化不限制。预留指令STEADYIOB（STATE 引脚触发条件）通道一控制温度到目标温度附近时STATE引脚输出高电平，通道二控制温度到目标温度附近时STATE引脚输出高电平。通道一当测温温度大于5000℃时，控制器会关闭输出；通道二当测温温度大于5000℃时，控制器会关闭输出。通道一当测温温度小于-3000℃时，控制器会关闭输出；通道二当测温温度小于-3000℃时，控制器会关闭输出。通道一正向死区电压为0，通道二正向死区电压为0，通道一负向死区电压为0，通道二负向死区电压为0。通道一NTC电阻的内部参考电阻为10KΩ，通道二NTC电阻的内部参考电阻为10KΩ。通道一PT1000电阻的内部参考电阻为1KΩ，通道二PT1000电阻的内部参考电阻为1KΩ。通道一Callendar-van-Dusen公式中的A值除以10E9，通道二Callendar-van-Dusen公式中的A值除以10E9。通道一Callendar-van-Dusen公式中的B值除以10E12，通道二Callendar-van-Dusen公式中的B值除以10E12。通道一Callendar-van-Dusen公式中的C值除以10E16，通道二Callendar-van-Dusen公式中的C值除以10E16。通道一极性输出为正向，通道二极性输出为正向。

注释：如果是双通道温控器，则发送完TC1的数据后会直接发送TC2的数据。

3.6.2 查询：一次性查询关键数据

指令类型	ASCII码指令	寄存器地址	数据类型	寄存器数量	权限	数据范围
通用参数指令	DATADEMAND	/	/	/	/	1~2
<p>说明： 发送：DATADEMAND=1@ 应答： TC1:TCADJTEMP=2259187@TC1:RESISTOR=11139104486@TC1:PWM=0@TC2:TCADJTEMP=999999999@TC2:RESISTOR=0@TC2:PWM=0@SINTERIORTEMP=23@ 代表目前通道一的温度为22.59187°C，通道一的电阻为11139.104486Ω，通道一的实际电压输出百分比为0。通道二未接温度传感器显示999999999，通道二的电阻为0，通道二的实际电压输出百分比为0。温控器自身温度为23°C。</p> <p>发送：DATADEMAND=2@ 应答： TC1: TCADJTEMP=2518788@TC1: RESISTOR=9916909257@TC1: OUTV=1000000000@TC2: TCADJTEMP=999999999@TC2: RESISTOR=0@TC2: OUTV=0@SINTERIORTEMP=34@ 代表目前通道一的温度为25.18788°C，通道一的电阻为9916.909257Ω，通道一的实际输出电压为10V。通道二未接温度传感器显示999999999，通道二的电阻为0，通道二的实际输出电压为0V。温控器自身温度为34°C。</p> <p>注释：如果是双通道温控器，则发送完TC1的数据后会直接发送TC2的数据。</p>						

4. 指令预览表

功能	寄存器名称	通信协议	寄存器地址 (0x)	权限	类型	数据范围	默认值 (未接传感器)
目标温度配置	查询/设置目标温度	TC1:TG	1000(通道一) 2000(通道二)	读写	int32	-40000000~10000000	2500000
温度传感器参数配置	查询/设置实际温度	TC1:TCADJTEMP	1002(通道一) 2002(通道二)	读写	int32	-40000000~10000000	999999999 9
	查询传感器电阻值	TC1:RESISTOR	1004(通道一) 2004(通道二)	只读	uint64	1~500000000000	0
	查询/设置温度求解模型选择	TC1:POLYOMIAL	1300(通道一) 2300(通道二)	读写	uint16	0~3	0
	查询/设置NTC电阻的B值	TC1:BX	1301(通道一) 2301(通道二)	读写	uint32	100000~5000000	395000
	查询/设置NTC电阻的R0(25°C)值	TC1:RP	1303(通道一) 2303(通道二)	读写	uint32	1~9000000	10000
	查询/设置温控器NTC通道内部参考电阻	TC1:NTCRP	1305(通道一) 2305(通道二)	读写	uint64	1~11000000000	1000000000
	查询/设置PT1000电阻的R0值	TC1:PT1000RP	1309(通道一) 2309(通道二)	读写	uint32	0~10000000	1000000
	查询/设置PT1000	TC1:PTA	130B(通道一)	读写	int32	-9000000~90000	3908300

电阻的 Callendar-van-Du sen系数 A		230B(通道二)			00	
查询/设置PT1000 电阻的 Callendar-van-Du sen系数 B	TC1:PTB	130D(通道一) 230D(通道二)	读写	int32	-9000000~90000 00	-577500
查询/设置PT1000 电阻的 Callendar-van-Du sen系数 C	TC1:PTC	130F(通道一) 230F(通道二)	读写	int32	-90000~90000	-41830
查询/设置温控器 PT1000通道 内部参考电阻	TC1:PTRP	1311(通道一) 2311(通道二)	读写	uint64	1~2100000000	10000000 00
查询/设置MF501 电阻的A值	TC1:MF501A	1342(通道一) 2342(通道二)	读写	int64	-10000000000000 0~100000000000 000	/
查询/设置MF501 电阻的B值	TC1:MF501B	1346(通道一) 2346(通道二)	读写	int64	-10000000000000 0~100000000000 000	/
查询/设置MF501 电阻的C值	TC1:MF501C	134A(通道一) 234A(通道二)	读写	int64	-10000000000000 0~100000000000 000	/
查询/设置A0系数 的浮点数部分	TC1:POLA0	1315(通道一) 2315(通道二)	读写	int64	-9999999999999 ~999999999999 9	0
查询/设置A0系数 的指数部分	TC1:POLEA0	1319(通道一) 2319(通道二)	读写	int16	-100~100	0
查询/设置A1系数 的浮点数部分	TC1:POLA1	131A(通道一) 231A(通道二)	读写	int64	-9999999999999 ~999999999999 9	0
查询/设置A1系数 的指数部分	TC1:POLEA1	131E(通道一) 131E(通道二)	读写	int16	-100~100	0
查询/设置A2系数 的浮点数部分	TC1:POLA2	131F(通道一) 231F(通道二)	读写	int64	-9999999999999 ~999999999999 9	0
查询/设置A2系数 的指数部分	TC1:POLEA2	1323(通道一) 2323(通道二)	读写	int16	-100~100	0
查询/设置A3系数 的浮点数部分	TC1:POLA3	1324(通道一) 2324(通道二)	读写	int64	-9999999999999 ~999999999999 9	0
查询/设置A3系数 的指数部分	TC1:POLEA3	1328(通道一) 2328(通道二)	读写	int16	-100~100	0
查询/设置A4系数	TC1:POLA4	1329(通道一)	读写	int64	-9999999999999	0

	的浮点数部分		2329(通道二)			~99999999999999999 9	
	查询/设置A4系数的指数部分	TC1:POLEA4	132D(通道一) 232D(通道二)	读写	int16	-100~100	0
温度传感器 参数配置	查询/设置A5系数的浮点数部分	TC1:POLA5	132E(通道一) 232E(通道二)	读写	int64	-99999999999999999 ~99999999999999999 9	0
	查询/设置A5系数的指数部分	TC1:POLEA5	1332(通道一) 2332(通道二)	读写	int16	-100~100	0
	查询/设置A6系数的浮点数部分	TC1:POLA6	1333(通道一) 2333(通道二)	读写	int64	-99999999999999999 ~99999999999999999 9	0
	查询/设置A6系数的指数部分	TC1:POLEA6	1337(通道一) 2337(通道二)	读写	int16	-100~100	0
	查询/设置A7系数的浮点数部分	TC1:POLA7	1338(通道一) 2338(通道二)	读写	int64	-99999999999999999 ~99999999999999999 9	0
	查询/设置A7系数的指数部分	TC1:POLEA7	133C(通道一) 233C(通道二)	读写	int16	-100~100	0
	查询/设置过温保护高阈值	TC1:OVERTEMPUP	133D(通道一) 233D(通道二)	读写	int32	-300000000~500000000	500000000 0
	查询/设置过温保护低阈值	TC1:OVERTEMPDOWN	133F(通道一) 233F(通道二)	读写	int32	-300000000~500000000	-300000000 0
	查询/设置传感器开路短路输出保护功能	TC1:ONSENSOR	110C(通道一) 210C(通道二)	读写	int16	0~1	1
	查询/设置开机电源输出模式	TC1:POWERMODE	1110(通道一) 2110(通道二)	读写	uint16	0~2	0
	查询输出电流	TC1:CURRENT	1111(通道一) 2111(通道二)	只读	uint16	0~900000000	0
	设置最大输出电流	TC1:SETCURRENT	1112(通道一) 2112(通道二)	读写	uint16	5~150	/
温控器输出 配置	查询/设置最大输出占空比	TC1:LIMITED	110E(通道一) 210E(通道二)	读写	int16	0~90	30
	查询/设置输出使能	TC1:ENABLE	1100(通道一) 2100(通道二)	读写	uint16	0~1	0
	查询/设置开机延迟输出	TC1:STARTUPDELAY	110F(通道一) 210F(通道二)	读写	uint16	3~180	3
	查询/设置输出模式	TC1:MODE	1101(通道一) 2101(通道二)	读写	uint16	0~3	0
	查询/设置输出极性	TC1:PIDPOL	1102(通道一) 2102(通道二)	读写	uint16	0~1	0
	查询/设置电压输出百分比	TC1:PWMOUTPUT	1103(通道一) 2103(通道二)	读写	int64	-2000000~2000000	0

	查询/设置温度变化斜率	TC1:SPEED	1108(通道一) 2108(通道二)	读写	uint16	0~10000	0
	查询/设置正向启动电压	TC1:FDEAD V	110A(通道一) 210A(通道二)	读写	uint16	0~400	0
	查询/设置反向启动电压	TC1:BDEAD V	110B(通道一) 210B(通道二)	读写	uint16	0~400	0
	查询/设置PWM引脚输出频率	FPWM	000D	读写	uint16	0~3	2
	查询/设置传感器过温保护模式	OVERTTEMP	000B	读写	uint16	0~1	1
	查询/设置温控器模式选择	CONTMODE	0004	读写	int16	0~3	0
PID配置	查询/设置PID中的P	TC1:KP	1200(通道一) 2200(通道二)	读写	uint32	0~9000000	3000
	查询/设置PID中的I	TC1:KI	1202(通道一) 2202(通道二)	读写	uint32	0~9000000	150
	查询/设置PID中的D	TC1:KD	1204(通道一) 2204(通道二)	读写	uint32	0~9000000	0
	查询/设置PID自整定	TC1:AUTOPI D	1107(通道一) 2107(通道二)	读写	uint16	0~2	0
温控器基础配置	查询当前温控器型号	TEC	0001	只读	uint16	0~255	依型号而定
	查询固件版本号	FPV	000C	只读	uint16	100~9999	依版本而定
	查询/设置温控器485地址	ADDRESS	0002	读写	uint16	0~255	1
	查询/设置UART TTL波特率	BOUNDTABLEONE	0008	读写	uint16	0~7	3
	查询/设置RS485波特率	BOUNDTABLETWO	0009	读写	uint16	0~7	1
	查询温控器自身温度	SINTERIORT EMP	0003	只读	int16	-20~120	/
	查询/设置温控器自身过温阈值	OVERTVPT	000A	读写	uint16	40~100	70
	查询错误代码	ERRORCODE	0007	只读	uint16	按 Bit 位定义	0
	恢复出厂设置	RESET	0000	只写	uint16	1	0
其它数据查询	查询:一次性查询全面数据	INQUIRE	/	/	/	1	/
	查询:一次性查询关键数据	DATADEMAND	/	/	/	1~2	/

注意: 由于产品更新, 旧版本产品不一定支持最新的通信协议, 敬请理解!

附录 1 温度解算模型和多项式修正

01 NTC 热敏电阻温度传感器

Basic 方程计算法 (B-Value 模型)	
$R = R_0 \times \exp[B \times (1 / (T + 273.15) - 1 / 298.15)]$	
T	温度, 单位摄氏度 (°C)
R	传感器实际电阻值, 单位欧姆 (Ω)
R ₀	NTC 在 25°C 时的电阻值 R (25°C), 单位欧姆 (Ω)
B	传感器β值参数
C 语言温度计算公式: $T = 1 / (1 / (298.15) + 1 / B \times \ln (R / R_0)) - 273.15$	

Steinhart-Hart 方程计算法 (S-H 模型)	
$1 / (T + 273.15) = A_0 + A_1 \times \ln (R) + A_2 \times [\ln (R)]^2 + A_3 \times [\ln (R)]^3 + A_4 \times [\ln (R)]^4$	
T	温度, 单位摄氏度 (°C)
R	传感器实际电阻值, 单位欧姆 (Ω)
A ₀ , ..., A ₄	传感器系数 (与多项式温度校正共用)

02 PT 铂电阻温度传感器

-200~0°C 温度范围 (PT 模型)	
$R = R_0 \times [1 + A \times T + B \times T^2 + C \times (T - 100) T^3]$	
T	温度, 单位摄氏度 (°C)
R	传感器实际电阻值, 单位欧姆 (Ω)
R ₀	PT 在 0°C 时的电阻值, 单位欧姆 (Ω)
A, B, C	传感器系数
0~800°C 温度范围 (PT 模型)	
$R = R_0 \times [1 + A \times T + B \times T^2]$	
T	温度, 单位摄氏度 (°C)
R	传感器实际电阻值, 单位欧姆 (Ω)

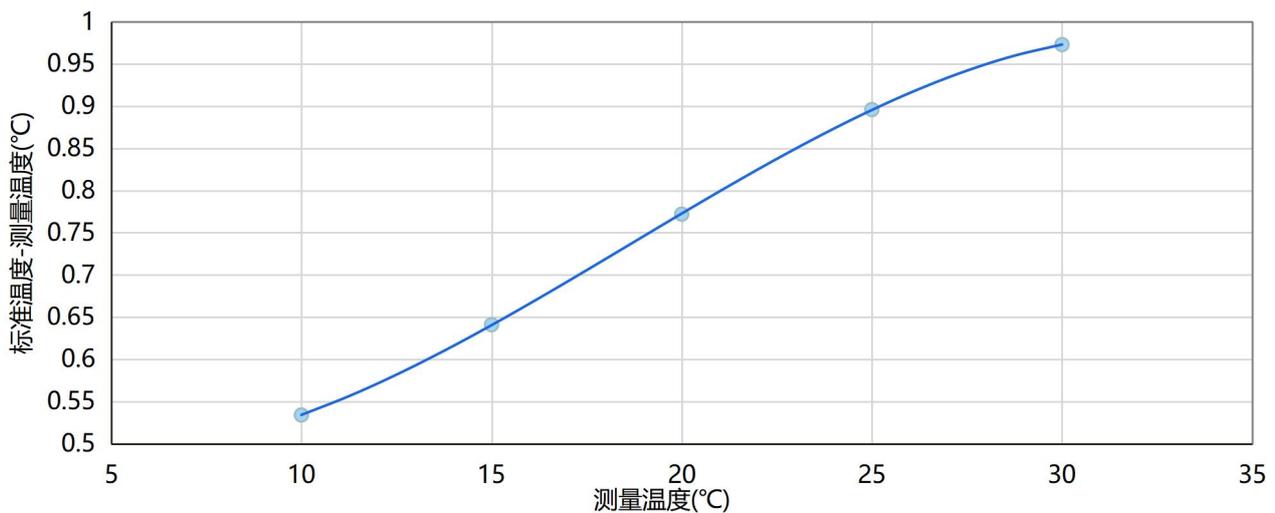
R_0	PT 在 0°C时的电阻值, 单位欧姆 (Ω)
A, B	传感器系数

03 多项式温度校正

$T_{\text{修正后}} = T_{\text{修正前}} + A_0 + A_1 \times T_{\text{修正前}} + A_2 \times T_{\text{修正前}}^2 + A_3 \times T_{\text{修正前}}^3 + A_4 \times T_{\text{修正前}}^4 + A_5 \times T_{\text{修正前}}^5 + A_6 \times T_{\text{修正前}}^6 + A_7 \times T_{\text{修正前}}^7$	
$T_{\text{修正后}}$	多项式温度校正后的温度, 单位摄氏度 ($^{\circ}\text{C}$)
$T_{\text{修正前}}$	多项式温度校正前的温度, 单位摄氏度 ($^{\circ}\text{C}$), 它可以是 B-Value 或者 PT 模型解算得的温度, 但不可以是 S-H 模型解算的温度。也就是说当选择 B-Value 和 PT 模型时, 多项式温度校正功能处于一直开启中。但当选择 S-H 模型时, 多项式温度校正功能则停用。(注意硬件版本 v4.2.2 及以上版本才有该功能, 敬请谅解)
A_0, \dots, A_7	温度修正系数

案例说明: NTC 温度传感器配合温控器的多项式温度校正

一台仪器采用 NTC 温度传感器进行温度测量, 设定的 R_0 和 B 值分别为 10000 Ω 和 3950。将多项式修正系数 $A_0 \sim A_7$ 设置为 0, 用标准温度传感器对整套系统进行温度校正。然后测量温度和标准温度表如下表所示:

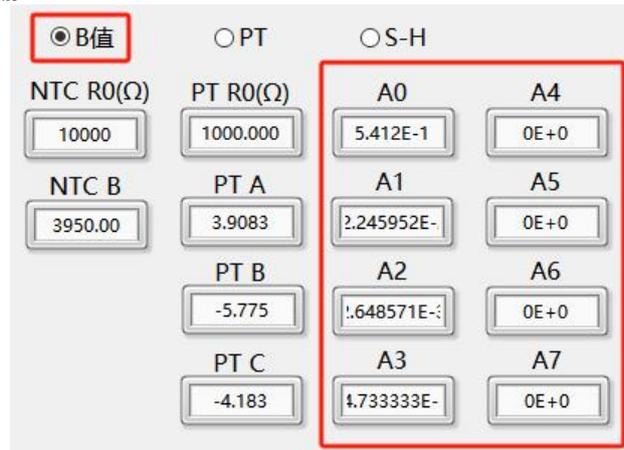


测量温度 ($^{\circ}\text{C}$) $T_{\text{修正前}}$	标准温度 ($^{\circ}\text{C}$) $T_{\text{标准}}$	标准温度-测量温度 ($^{\circ}\text{C}$) ΔT
10.000	10.534	0.534
15.000	15.641	0.641
20.000	20.772	0.772
25.000	25.896	0.896
30.000	30.973	0.973

利用 Excel 的多项式对温度差 (标准温度-测量温度) vs 测量温度进行 3 次多项式拟合 (依据情况具体选择几次多项式), 得到多项式修正公式为: $T_{\text{修正后}} = T_{\text{修正前}} + (5.412000e-1) + (-2.245952e-2) \times T_{\text{修正前}} +$

$(2.648571e-3) \times T_{修正前}^2 + (-4.733333e-5) \times T_{修正前}^3$ ，即：

温度修正系数	值
A_0	5.412000e-1
A_1	-2.245952e-2
A_2	2.648571e-3
A_3	-4.733333e-5
$A_4 \dots A_7$	0



通过电脑软件写入 $A_0 \sim A_7$ 的值，如上图所示，即多项式温度修正完成。

注意：我司默认配套的温度传感器是未经计量校准的，若需要计量校准，可以联系官方客服微信：18718688108。

附录 2 编程案例

01 C++ 编程案例

```
#include <iostream>
#include <windows.h>
// 本案例使用 C 语言编写与光测未来温度控制模块通信, 实现了通过 TTL 指令与 ModBus 指令分别设置目标温度数据与获取当前温度数据, 其他指令请参照程控手册自行开发
// 如通信失败请仔细核对串口号、波特率及温控器是否开启
// MODBUS CRC 校验函数
uint16_t crc_update(int data_length, uint8_t*data) {
    uint16_t CRC_Value = 0xFFFF; // CRC 初始值
    uint8_t i, j;
    // 计算 CRC 校验值
    for (i = 0; i < data_length; i++)
    {
        CRC_Value ^= static_cast<uint8_t>(data[i]);
        for (j = 0; j < 8; j++)
        {
            if (CRC_Value & 0x0001)
                CRC_Value = (CRC_Value >> 1) ^ 0xA001;
            else
                CRC_Value = (CRC_Value >> 1);
        }
    }
    return CRC_Value;
}
```

```
int ExampleTest() {
    // 打开串口句柄, 假设串口名为 COM9
    HANDLE hSerial = CreateFile(L"COM9",
        GENERIC_READ | GENERIC_WRITE,
        0,
        NULL,
        OPEN_EXISTING,
        FILE_ATTRIBUTE_NORMAL,
        NULL);
    if (hSerial == INVALID_HANDLE_VALUE) {
        std::cout << "打开串口失败" << std::endl;
        return 1;
    }
    // 配置串口参数
    DCB dcb = { 0 };
    dcb.DCBlength = sizeof(dcb);
    GetCommState(hSerial, &dcb);
    dcb.BaudRate = CBR_38400; // 波特率 38400
    dcb.ByteSize = 8;        // 数据位 8
    dcb.Parity = NOPARITY;  // 无校验
    dcb.StopBits = ONESTOPBIT; // 停止位 1
    SetCommState(hSerial, &dcb);

    // 设置超时
    COMMTIMEOUTS timeouts = { 0 };
    timeouts.ReadIntervalTimeout = 50;
    timeouts.ReadTotalTimeoutConstant = 50;
    timeouts.ReadTotalTimeoutMultiplier = 10;
    SetCommTimeouts(hSerial, &timeouts);

    // TTL 指令设置一通道目标温度为 25°C

    // 发送数据
    const char* sendData = "TC1:TG=2500000@\n";
    DWORD bytesWritten;
    if (!WriteFile(hSerial, sendData, strlen(sendData), &bytesWritten, NULL)) {
        std::cerr << "发送数据失败" << std::endl;
        CloseHandle(hSerial);
        return 1;
    }
}
```

```
// 接收数据
char receivedData[32] = { 0 };
DWORD bytesRead;
if (!ReadFile(hSerial, receivedData, sizeof(receivedData) - 1, &bytesRead, NULL)) {
    std::cerr << "接收数据失败" << std::endl;
    CloseHandle(hSerial);
    return 1;
}

// 回复判断
if (strcmp(receivedData, "OKTC1:TG=2500000@\r\n") != 0) {
    std::cout << "温度设置失败" << std::endl;
}
else {
    std::cout << "目标温度已设置为 25°C" << std::endl;
}

// TTL 指令询问一通道当前温度
float myCurrentTemperature = 0.0f;

// 发送数据
Sleep(5); // 等待 5 毫秒, 确保数据不会连发
const char* sendData1 = "TC1:TCADJTEMP=?@\n";
DWORD bytesWritten1;
if (!WriteFile(hSerial, sendData1, strlen(sendData1), &bytesWritten1, NULL)) {
    std::cerr << "发送数据失败" << std::endl;
    CloseHandle(hSerial);
    return 1;
}

// 接收数据
char receivedData1[32] = { 0 };
DWORD bytesRead1;
if (!ReadFile(hSerial, receivedData1, sizeof(receivedData1) - 1, &bytesRead1, NULL)) {
    std::cerr << "接收数据失败" << std::endl;
    CloseHandle(hSerial);
    return 1;
}

// 回复判断, 前几位是不是"OKTC1:TCADJTEMP=",最后面是不是"@\r\n"
if (strncmp(receivedData1, "OKTC1:TCADJTEMP=", 16) != 0 || strcmp(receivedData1 + bytesRead1 -
```

```
4, "@\r\n") != 0) {
    std::cout << "回复数据错误" << std::endl;
}
else {
    // 解析数据
    char tempBuffer[16] = { 0 };
    int tempLength = bytesRead1 - 19; // 去掉最后三个字符
    if (tempLength > 0 && tempLength < sizeof(tempBuffer)) {
        strncpy_s(tempBuffer, sizeof(tempBuffer), receivedData1 + 16, tempLength);
        myCurrentTemperature = (float)(atoi(tempBuffer) / 100000.0f);
        std::cout << "当前温度为: " << myCurrentTemperature << "°C" << std::endl;
    }
    else {
        std::cerr << "解析温度数据失败" << std::endl;
    }
}
// ModBus 指令设置一通道目标温度为 25°C

// 设置发送指令
int8_t sendData2[13];
int index = 0;
sendData2[index++] = 0x01; // 假设温控器 ModBus 地址为 0x01
sendData2[index++] = 0x10; // 寄存器写入指令功能码
sendData2[index++] = 0x10; // 与下一字节拼接
sendData2[index++] = 0x00; // 与上一字节拼接: 0x1000 表示设置目标温度
sendData2[index++] = 0x00; // 与下一字节拼接
sendData2[index++] = 0x02; // 与上一字节拼接: 0x0002 表示发送两个寄存器数量
sendData2[index++] = 0x04; // 0x04 表示数据需要四个字节发送
sendData2[index++] = 2500000 >> 24; // 2500000 的高字节
sendData2[index++] = 2500000 >> 16; // 2500000 的次高字节
sendData2[index++] = 2500000 >> 8; // 2500000 的次低字节
sendData2[index++] = 2500000; // 2500000 的低字节

// CRC 校验和
uint16_t crc = crc_update(index, reinterpret_cast<uint8_t*>(sendData2));
sendData2[index++] = crc & 0xFF;
sendData2[index++] = crc >> 8;

// 发送数据
Sleep(5); // 等待 5 毫秒, 确保数据不会连发
DWORD bytesWritten2;
```

```
WriteFile(hSerial, sendData2, index, &bytesWritten2, NULL);

// 接收数据
int8_t receivedData2[32] = { 0 };
DWORD bytesRead2;
ReadFile(hSerial, receivedData2, sizeof(receivedData2) - 1, &bytesRead2, NULL);

// 回复判断
// CRC 校验
crc =crc_update(bytesRead2 - 2, reinterpret_cast<uint8_t*>(receivedData2));
uint16_t crc_rcv = (receivedData2[bytesRead2 - 1] << 8) + receivedData2[bytesRead2 - 2];
if (crc != crc_rcv) {
    std::cout << "CRC 校验失败" << std::endl;
}
else {
    // CRC 校验成功, 解析数据
    if (receivedData2[0] != 0x01)
    {
        std::cout << "设备错误" << std::endl;
    }
    else if (receivedData2[1] != 0x10)
    {
        std::cout << "不是寄存器写入指令" << std::endl;
    }
    else if (receivedData2[2] != 0x10 || receivedData2[3] != 0x00)
    {
        std::cout << "寄存器地址错误" << std::endl;
    }
    else if ((receivedData2[4]<<8|receivedData2[5]) != 0x0002)
    {
        std::cout << "寄存器数量错误" << std::endl;
    }
    else
    {
        std::cout << "目标温度已设置为 25°C" << std::endl;
    }
}

// ModBus 指令读取一通道当前温度

// 设置发送指令
```

```
int8_t sendData3[8];
index = 0;
sendData3[index++] = 0x01; // 假设温控器 ModBus 地址为 0x01
sendData3[index++] = 0x03; // 寄存器读取指令功能码
sendData3[index++] = 0x10; // 与下一字节拼接
sendData3[index++] = 0x02; // 与上一字节拼接: 0x1002 表示一通道当前温度
sendData3[index++] = 0x00; // 与下一字节拼接
sendData3[index++] = 0x02; // 与上一字节拼接: 0x0002 表示发送两个寄存器数量

// CRC 校验和
crc = crc_update(index, reinterpret_cast<uint8_t*>(sendData3));
sendData3[index++] = crc & 0xFF;
sendData3[index++] = crc >> 8;
// 发送数据
Sleep(5); // 等待 5 毫秒, 确保数据不会连发
DWORD bytesWritten3;
WriteFile(hSerial, sendData3, index, &bytesWritten3, NULL);

// 接收数据
int8_t receivedData3[32] = { 0 };
DWORD bytesRead3;
ReadFile(hSerial, receivedData3, sizeof(receivedData3) - 1, &bytesRead3, NULL);

// 回复判断
// CRC 校验
crc = crc_update(bytesRead3 - 2, reinterpret_cast<uint8_t*>(receivedData3));
crc_rcv = (static_cast<uint8_t>(receivedData3[bytesRead3 - 2]) |
           (static_cast<uint8_t>(receivedData3[bytesRead3 - 1]) << 8));
if (crc != crc_rcv) {
    std::cout << "CRC 校验失败" << std::endl;
}
else {
    // CRC 校验成功, 解析数据
    if (receivedData3[0] != 0x01)
    {
        std::cout << "设备错误" << std::endl;
    }
    else if (receivedData3[1] != 0x03)
    {
        std::cout << "不是寄存器读取指令" << std::endl;
    }
}
```

```
else if (receivedData3[2] != 0x04)
{
    std::cout << "数据长度错误" << std::endl;
}
else
{
    // 解析数据
    myCurrentTemperature = ((static_cast<uint8_t>(receivedData3[3])<< 24)
        | (static_cast<uint8_t>(receivedData3[4]) << 16)
        | (static_cast<uint8_t>(receivedData3[5]) << 8)
        | static_cast<uint8_t>(receivedData3[6]))/100000.0f;
    std::cout << "当前温度为: " << myCurrentTemperature <<"°C" << std::endl;
}
}
if (hSerial != INVALID_HANDLE_VALUE) {
    CloseHandle(hSerial);
}
return 0;
}
int main()
{
    ExampleTest();
}
```

02 Python 编程案例

```
import serial
import time
from typing import List

def crc_update(data: bytes) -> int:
    """
    MODBUS CRC-16 校验计算
    :param data: 需要计算 CRC 的字节数据
    :return: 计算得到的 CRC 校验值
    """
    crc = 0xFFFF # CRC 初始值
    for byte in data:
        crc ^= byte # 异或操作
        for _ in range(8): # 对每个 bit 进行处理
```

```
        if crc & 0x0001: # 如果最低位为 1
            crc = (crc >> 1) ^ 0xA001 # 右移并异或多项式
        else:
            crc = crc >> 1 # 直接右移
    return crc

def example_test():
    """
    温度控制器通信示例测试函数
    包含 TTL 指令和 ModBus 协议两种通信方式
    """
    try:
        # 打开串口 (COM9, 波特率 38400)
        # 参数说明:
        # port: 串口号
        # baudrate: 波特率
        # bytesize: 数据位(8 位)
        # parity: 校验位(N 无校验)
        # stopbits: 停止位(1 位)
        # timeout: 读取超时时间(秒)
        with serial.Serial('COM9', baudrate=38400, bytesize=8,
                           parity='N', stopbits=1, timeout=0.05) as ser:

            # ===== TTL 指令设置目标温度 =====
            # 发送设置温度指令(通道 1 目标温度 25°C)
            # 指令格式: TC1:TG=2500000@\n
            # 2500000 对应 25.00000°C
            send_data = b"TC1:TG=2500000@\n"
            ser.write(send_data) # 发送指令
            received_data = ser.read(32).decode('ascii') # 读取返回数据

            # 检查返回数据是否正确
            if received_data == "OKTC1:TG=2500000@\r\n":
                print("目标温度已设置为 25°C")
            else:
                print("温度设置失败")
            return

            # ===== TTL 指令查询当前温度 =====
            time.sleep(0.005) # 等待 5ms, 避免数据连发
            # 发送查询温度指令(通道 1 当前温度)
```

```
# 指令格式: TC1:TCADJTEMP=?@\n
send_data = b"TC1:TCADJTEMP=?@\n"
ser.write(send_data) # 发送指令
received_data = ser.read(32).decode('ascii') # 读取返回数据

# 检查返回数据格式是否正确
# 正确格式: "OKTC1:TCADJTEMP=xxxx@\r\n"
if not (received_data.startswith("OKTC1:TCADJTEMP=") and
        received_data.endswith("@\r\n")):
    print("回复数据错误")
else:
    # 提取温度数值部分(去掉前后固定字符)
    temp_str = received_data[16:-3]
    try:
        # 将字符串转换为整数并计算实际温度值
        # 温度值需要除以 100000 得到实际温度(°C)
        current_temp = int(temp_str) / 100000.0
        print(f"当前温度为: {current_temp}°C")
    except ValueError:
        print("解析温度数据失败")

# ===== ModBus 协议设置目标温度 =====
time.sleep(0.005) # 等待 5ms

# 构造 ModBus 指令帧
# 功能码 0x10: 写多个寄存器
# 寄存器地址 0x1000: 目标温度设置
# 数据: 2500000 (对应 25.00000°C)
data = bytearray([
    0x01,      # 设备地址(假设为 1)
    0x10,      # 功能码(写多个寄存器)
    0x10, 0x00, # 寄存器地址(0x1000)
    0x00, 0x02, # 寄存器数量(2 个)
    0x04,      # 数据字节数(4 字节)
    (2500000 >> 24) & 0xFF, # 温度值高字节
    (2500000 >> 16) & 0xFF,
    (2500000 >> 8) & 0xFF,
    2500000 & 0xFF      # 温度值低字节
])

# 计算 CRC 校验并添加到指令末尾
```

```
crc = crc_update(data)
data.append(crc & 0xFF)    # CRC 低字节
data.append(crc >> 8)     # CRC 高字节

# 发送 ModBus 指令
ser.write(data)

# 读取响应数据
received_data = ser.read(32)

if len(received_data) < 8: # 检查最小响应长度
    print("ModBus 响应数据过短")
else:
    # 验证 CRC 校验
    crc_calc = crc_update(received_data[:-2])
    crc_rcv = received_data[-2] | (received_data[-1] << 8)

    if crc_calc != crc_rcv:
        print("ModBus CRC 校验失败")
    else:
        # 解析响应数据
        if received_data[0] != 0x01:
            print("设备地址错误")
        elif received_data[1] != 0x10:
            print("功能码错误")
        elif received_data[2] != 0x10 or received_data[3] != 0x00:
            print("寄存器地址错误")
        elif (received_data[4] << 8 | received_data[5]) != 0x0002:
            print("寄存器数量错误")
        else:
            print("ModBus 目标温度已设置为 25°C")

# ===== ModBus 协议查询当前温度 =====
time.sleep(0.005) # 等待 5ms

# 构造 ModBus 查询指令
# 功能码 0x03: 读保持寄存器
# 寄存器地址 0x1002: 当前温度读取
data = bytearray([
    0x01,    # 设备地址
    0x03,    # 功能码(读保持寄存器)
```

```
    0x10, 0x02, # 寄存器地址(0x1002)
    0x00, 0x02 # 读取寄存器数量(2 个)
])

# 计算 CRC 校验
crc = crc_update(data)
data.append(crc & 0xFF)
data.append(crc >> 8)

# 发送查询指令
ser.write(data)

# 读取响应数据
received_data = ser.read(32)

if len(received_data) < 9: # 检查最小响应长度
    print("ModBus 响应数据过短")
else:
    # 验证 CRC 校验
    crc_calc = crc_update(received_data[:-2])
    crc_rcv = received_data[-2] | (received_data[-1] << 8)

    if crc_calc != crc_rcv:
        print("ModBus CRC 校验失败")
    else:
        # 解析响应数据
        if received_data[0] != 0x01:
            print("设备地址错误")
        elif received_data[1] != 0x03:
            print("功能码错误")
        elif received_data[2] != 0x04:
            print("数据长度错误")
        else:
            # 提取温度值(4 字节)
            temp_value = ((received_data[3] << 24) |
                          (received_data[4] << 16) |
                          (received_data[5] << 8) |
                          received_data[6])

            current_temp = temp_value / 100000.0
            print(f"ModBus 当前温度为: {current_temp}°C")
```

```
except serial.SerialException as e:
    print(f"串口通信错误: {e}")
except Exception as e:
    print(f"程序错误: {e}")

if __name__ == "__main__":
    example_test()
```

版本变更日志

版本变更日志	变更内容	变更日期	审核人
1.0	最新初始版本	2025/4/10	WST、WYR
1.0-1.1	新增 4.指令预览表 新增温控器型号	2025/4/18	WST、WYR
1.1-1.2	内容修订 新增 Modbus 示例和编程案例	2025/8/8	WST、WYR
1.2-1.3.0	内容修订	2026/1/12	WST、YJH、ZZA

网站: www.sensefuture.com.cn

商城: store.sensefuture.com.cn

电话: 18718688108 (官方微信同号)

邮箱: sales@sensefuture.com

地址: 深圳市光明区凤凰街道三环科技大厦 A1 区 1308



初心定未来
创新造价值
分享聚人心

期待与您的合作共赢!

产品资料下载

