

JMaster3/L 脱机量产编程器

二次开发手册

V2.03e

南京伟福实业有限公司



目录

名词解释.....	1
修订记录表.....	2
概述.....	3
一、通过扩展口信号控制 JM3L.....	3
1.1 机台信号控制模式.....	3
1.2 串口（UART）控制模式.....	4
二、通过命令行方式控制 JM3L.....	10
2.1 命令行方式控制文件列表.....	11
2.2 命令行方式控制简单范例与解释.....	11
2.3 命令行方式控制参数列表.....	13
2.4 命令行方式控制注意事项.....	17
三、通过 DLL 动态链接库控制 JM3L（上位机二次开发）.....	19
3.1 编程器状态.....	19
3.2 用户程序基本流程图.....	19
3.2 DLL 接口函数列表.....	20
附录：命令行、DLL 函数调用返回信息定义.....	30



名词解释

JMaster3/L 或编程器	JMaster3/L(简称 JM3L)编程器，通常也称作“烧录器”。
编程	对目标板进行编程，“编程”通常也称作“烧录”
编程文件	也称“JMO 文件”，是可以被编程器识别并对目标板编程的文件，它包含用户程序、芯片配置、编程速度/电压/限定次数、自动编号规则、文件密码等完整信息，该文件后缀名“.JMO”，为加密文件，不可更改，且一旦经编程软件下载到编程器(或 TF 卡)，将进行二次加密，编程器的文件无法回读，TF 卡虽然可通过外置读卡器获取文件，但是无论采用何种方法获得文件，都无法在其它编程器上使用； 因此请妥善保管好 PC 上的编程文件 ；如生成编程文件时限制了特定序列号编程器，则仅可在授权范围内的编程器上使用；如生成文件时设置了密码，则仅可在已设置与文件相同密码的编程器上使用，否则可在任何编程器上打开。编程文件主要有配置信息、用户程序组成，配置信息占据固定大小，因此编程文件大小主要由用户程序大小决定。
文件描述	是指在生成编程文件时为该文件设置的文件描述。文件描述由用户指定，可用于快速识别文件作用、特性、日期，因此建议将描述设置为程序的关键信息（例如项目名、版本号、日期等）
监控态	联机模式下，打开编程文件前或未打开编程文件且点击“编程”前，此状态液晶屏显示“已连接 PC”，此时为监控态；脱机模式下未打开任何编程文件前，此时为监控态
编程态	联机模式下，打开编程文件后或未打开编程文件且点击“编程”后，此状态液晶屏显示代表 2 个通道的进度条，分别代表 2 个通道进度，此时为编程态；脱机模式下，选定编程文件后 OK 键确认正确(秘钥不一致或当前编程器序列号未在被授权范围内会打开失败而无法进入到编程态)后，此状态液晶屏显示代表 2 个通道的进度条以及当前被打开的文件信息，此时为编程态。
目标板	也称“用户板”，是指被编程的电路板或芯片
JMCMD	编程器命令行控制软件，支持批处理调用实现复杂的编程场景



修订记录表

修订日期 Date	版本.	修订记录
2025/01/27	V2.03e	解决 RUN 偶尔死机;CMD 增加 PAUSE/RUNTEST/SLEEP
2023/12/08	V2.03d	UART 指令增加了适用态(监控态/编程态)
2023/01/28	V2.03c	修正 DLL 部分函数功能部分错误描述; 调整部分内容排版
2022/06/20	V2.03b	串口 (UART) 控制模式增加 UART 注入编程信息; DLL JM_JM 函数功能新增
2022/04/29	V2.03	修改部分内容
2022/01/07	V2.02	增加部分内容
2021/07/12	V2.01	增加部分内容
2020/12/21	V2.0	dll 接口更新
2020/04/01	V1.0	初始版本



概述

通过对 JMaster3/L(以下简称“JM3L”)编程器的二次开发,可获得更多对编程器的控制方法,以适应更多的应用场景,可将编程器无缝集成到产线以及 ERP/MES 管理系统中。

二次开发所依赖的软件以及相关例程,最新版本均集成在编程器配套的编程软件中,用户可下载编程软件获得二次开发相关软件、例程。

编程软件下载链接 http://www.wave-cn.cn/dl_jm3p.html

***注:本手册仅包含“二次开发”相关内容,编程器的基本功能请参见“编程器手册”。

一、通过扩展口信号控制 JM3L

通过扩展口(侧面 14 脚插座,信号定义见“表 1”),可实现 2 种模式控制烧录器:

1. 机台信号控制模式,可与具备标准机台信号的机台进行接驳实现大批量烧录
2. 串口(UART)控制模式,可通过 USB-TTL 模块与工控机进行联机控制;也可自制 MCU 板进行控制烧录器。

扩展口信号定义如下:

13	11	9	7	5	3	1
14	12	10	8	6	4	2

表 1

电平特性	Output (3.3V)	Output (3.3V)	Output (3.3V)	Output (3.3V)	Output (3.3V)	Input (3.3V)	Power (5V)
信号名称	TXD	NG2	NG1	OK1	BUSY1	START1	VCC
引脚序号	13	11	9	7	5	3	1
	14	12	10	8	6	4	2
信号名称	RXD	GND	GND	OK2	BUSY2	START2	VCC
电平特性	Input (3.3V)	Power (0V)	Power (0V)	Output (3.3V)	Output (3.3V)	Input (3.3V)	Power (5V)

其中:

GND/VCC: 电源 ($5V \leq 500mA$), 建议作为输出; 作为输入赢避免 $>5V$ 电源输入

TXD/RXD: UART 控制模式通信信号接口 (3.3V TTL)

STARTx/BUSYx/OKx/NGx(x=1/2): 标准测试台信号, 3.3V TTL 电平标准, x 对应编程口 P1、P2; 可通过编程器软件设置有效电平; START 为启动信号, 最高耐压 5V (内部弱下拉, 当 START 设置低电平有效时, 外部需要放置 4.7K-10K 上拉电阻或使用具有强上拉结构的 IO 推挽模式驱动); BUSY/OK/NG 为状态信号。

1.1 机台信号控制模式

机台信号控制模式(以下简称“机台模式”)下, 可以通过 Start 信号启动编程, 通过



Busy/OK/NG 信号判断编程结果。

标准机台信号可以实现简单的编程操作及结果判断，但是无法实现诸如“选择/切换文件”、“切换编程态/监控态”等操作。

机台信号控制的前提是编程器当前处于编程态。由于 JM3L 支持存储多个编程文件，如 JM3L 在未设置自启动文件时，开机默认为监控态（文件列表/选择界面，等待用户选择需要编程的文件），监控态仅不可进行编程操作。当 JM3L 已设置自启动文件时，开机将自动打开自启动文件并进入编程态，此时可以进行编程操作，编程态仅用于编程，不可切换编程文件。

因此在进行机台信号控制 JM3L 前，需要预先打开一个编程文件（对于有液晶屏的机型）进入编程态，或者将需要烧录的文件设置为自启动文件，编程器在下次开机时将会自动打开该文件并进入到编程态（设置自启动文件的方法请参见“编程器使用手册”）。

在编程态，STARTx 输入有效电平，JM3L 立即启动烧录，START1 的有效电平输入等同于按下烧录器的 P1 按键，START2 的有效电平输入等同于按下烧录器的 P2 按键。

启动烧录后，可立即通过 BUSY/OK/NG 信号判断烧录结果，BUSY 信号有效时，OK(成功)/NG(失败)为无效，表示烧录正在进行(未完成)。BUSY 信号无效时，OK 或 NG 其中之一有效，另一个无效（也即 OK/NG 信号相反），表示烧录已完成(成功或失败)。

注：机台控制模式要求 JMO 文件启动方式（由生成 JMO 文件时设置）为自动编程机方式。

1.2 串口（UART）控制模式

1. UART 的工作波特率为 115200，8 位数据，1 个停止位；
2. 上位机向 JM3L 编程器发送命令是四个字节，格式为： 0x55 CMD DAT CHK
0x55: 命令头 固定为 0x55
CMD：命令（1 字节,详见第 4 条）
DAT：数据（1 字节,详见第 4 条）
CHK：校验和（1 字节，四个命令字节加起来，最低字节应该为 0x00）

注：串口控制模式要求 JMO 文件启动方式（由生成 JMO 文件时设置）为按键方式。

1.2.1 串口(UART)控制模式命令大全

命令	命令功能	说明
0x01 [监控态命令]	连接 JM3L 编程器 [监控态命令]	第一条必须是这条命令，执行这条命令后才能执行其它的命令
0x02 [监控态&编程态命令]	断开 JM3L 编程器连接 [监控态命令]	这是最后一条 UART 命令，断开 UART 与 JM3L 编程器的连接
0x03 [编程态命令]	读取 JM3L 编程器信息指令	JM3L 在编程时会将状态回送(编程、擦除、进度条等) 有 32 条信息缓冲器
0x04 [编程态命令]	编程控制指令	随后 DAT 数据值有以下选择： 1..72: 选中 JM3L 编程器中的第 n 号 JMO 文件，(1..72 为 JMO 文件序号，可以通过软件读出 JM3L 编程器上文件列表，包含 FLASH 和 TF 卡上的文件，最多 72 个) 0xFC: 终止对当前 JMO 文件的操作



		0xFD: 按 P1 键编程选中的 JMO 文件 0xFE: 按 P2 键编程选中的 JMO 文件 0xFF: 按 OK 键编程选中的 JMO 文件
0x05 [监控态命令]	回传 JMO 文件信息	随后 DAT 为 JMO 文件序号 (JM3L 编程器文件列表可以用软件读出, 1..72)
0x06 [监控态&编程态命令]	回传 JM3L 编程器当前状态	
0x07 [监控态&编程态命令]	复位 JM3L 编程器	软件复位
0x08 [编程态命令]	JM3L 编程器电源脚控制	随后的 DAT 格式如下 bit7: 选择编程端口 0: 编程口 1 1: 编程口 2 bit6..bit3: 未定义 bit2..bit0: 选择编程电压 0: OFF 3: 1.8V 4: 2.5V 5: 3.3V 6: 3.6V 7: 5.0V P1,P2 的编程电压是一致的, 改变电压时两个口同时改变, 但电源的开/关是可以独立控制的
0x09 [监控态命令]	设置自启动 (缺省) 文件	随后 DAT 定义如下 0: 取消自启动文件 1~72: 将第 n 号文件为自启动 (缺省) 文件 自启动 (缺省) 文件就是开机后自动选中此 JMO 文件, 按下 P1 或 P2 键就可以进行编程操作
0x11 0x12 0x13 [编程态命令]	复位 P1 外部数据输入缓冲区 复位 P2 外部数据输入缓冲区 复位 P1,P2 外部数据输入缓冲区	将已有的外部输入缓冲区清空 外部输入缓冲区的起始地址及长度在 JMO 中设定
0x19 0x1A 0x1B [编程态命令]	将数据注入 P1 外部数据输入缓冲区 将数据注入 P2 外部数据输入缓冲区 将数据注入 P1,P2 外部数据输入缓冲区	将一个字节的数据注入到外部输入缓冲区, 下次再次注入时将注入到下一个地址单元



0xC0 - 0xDF [编程态命令]	写 IO 口(16 个 IO)指令 其中: Bit7..5 固定'110' Bit 4 选择编程口 0: 编程口 1 1: 编程口 2 bit3..bit0: 口地址 0: LED 指示灯状态 1: IOCTRL 编程口信号方向 2: IODATA 编程口信号值 3..0xf: 与编程相关的其它地址(不向用户开放)	随后的 DAT 为输出数据值 LED : bit3..bit0: 口地址 bit0: LED bit2: RED bit3: GREEN bit4: FLASH IOCTRL bit6..bit0:7 个编程 IO 输入/输出方向控制 0:输入 1:输出 bit6: PIN9, bit5: PIN7, bit4: PIN6, bit3: PIN5, bit2: PIN4, bit1: PIN3, bit0: PIN2 IODATA bit6..bit0: 7 个编程 IO 脚的输出值/输入值
0xE0 - 0xFF [编程态命令]	读 IO 口指令	

通过 CMD_POWER 来控制 PIN1 的电源开/关及电压选择

通过 CMD_WRITEIO 来写地址 0 来控制编程 LED 灯的显示

通过 CMD_WRITE,CMD_READIO 来控制编程口的状态设置及读取。地址 1 控制 IO 端口的方向, 地址 2 控制 IO 端口的读写。

这些功能主要是给用户在编程前或后进行用户板的自动测试。

例:

55 01 00 AA //连接命令

55 04 02 A5 //选择 2 号文件

55 04 FD AA //按 P1 键进行编程

3. JM3L 编程器应答格式为:

0xAA LEN RES DAT1....DATn CHK

0xAA: 应答头固定为 0xAA

LEN: 后面的数据长度 应答数据最多为 64 字节

RES: 应答值

DAT: 数据

CHK: 校验和 所有字节加起来应该为 0

应答值	命令功能	说明
0x01	RES_MESSAGE	DAT1: ERR 0: 没有错误; 1: 编程发生错误 DAT2: ATE 状态 bit7: 0: 状态没有发生变化 1: 状态发生变化 (如从擦除改为编程, 进度度加 1 等) bit3..bit0: ATE 口状态



		<p>DAT3 : BAR 进度条 (0..20)</p> <p>DAT4 : NO 0: 编程口 1;1: 编程口 2</p> <p>DAT5 : 编程状态 0(低 8 位)</p> <p>DAT6 : 编程状态 1</p> <p>DAT7 : 编程状态 2</p> <p>DAT8 : 编程状态 3(高 8 位)</p> <p>编程状态(4 字节)定义:</p> <pre> #define bPGNull 0x00000000 #define bPGWaitKey 0x00000001 #define bPGAUTO 0x00000002 #define bPGPIN 0x00000004 #define bPGConnect 0x00000008 #define bPGRestore 0x00000010 #define bPGErase 0x00000020 #define bPGWrite 0x00000040 #define bPGRead 0x00000080 #define bPGVerify 0x00000100 #define bPGProtect 0x00000200 #define bPGUnProtect 0x00000400 #define bPGBlankCheck 0x00000800 #define bPGCTRL 0x00001000 #define bPGEPPROM 0x00002000 #define bPGBootLoad 0x00004000 #define bPGATE 0x00008000 #define bPGSVF 0x00010000 #define pPGIDCODE 0x00020000 #define bPGCheckOPT 0x00040000 #define bPGPIO 0x00080000 #define bPGWriteOPT 0x00100000 #define bPGReadOPT 0x00200000 #define bPGTEST 0x00400000 #define bPGUserID0 0x00800000 #define bPGUserID1 0x01000000 #define bPGUserID2 0x02000000 #define bPGExecute 0x04000000 #define bPGJMONo 0x08000000 #define bPGDownload 0x10000000 #define bPGPowerOn 0x20000000 #define bPGPowerOff 0x40000000 #define bPGFinish 0x80000000 </pre>
0x02	RES_READDATA	预留暂时不用
0x03	RES_WRITELOG	预留暂时不用
0x04	RES_WRITELOGH EAD	预留暂时不用



0x05	RES_READFUNC	预留暂时不用
0x06	RES_STATUS	DAT1 :0: JM3L 不在编程态 1: JM3L 在编程态 DAT2 : 编程口 1 ATE 管脚状态 DAT3 : 编程口 2 ATE 管脚状态 DAT4 : PCNT 0 DAT5 : PCNT 1 DAT6 : PCNT 2 DAT7 : PCNT 3 ATE 状态字: BIT3: START; BIT2: NG; BIT1: OK; BIT0: BUSY; PCNT:如果文件有编程次数, PCNT 为剩余次数
0x07	RES_INTERROR	DAT1 : 错误号 // Internal Error 1 监控错 2 编程算法错 3 RTC 错 4 FLASH 没有格式化 5 TF 卡错 6 FPGA ID 错 7 FPGA 编程数据错 8 FPGA IP 校验错 9 FLASH ID 错 10 FLASH 状态信息错 // 运行 JMO 时检查出错 20 编程不支持这种芯片 21 编程器的编号不对 22 编程记数已完 23 编程器的密码不对 24 编程文件的检验不对
0x08	RES_VERSION	DAT1 : 3: JM3L DAT2 : 主版本号 DAT3 : 副版本号 DAT4 : 0: 没有 LCD 1: 有 LCD DAT5 : 0: TF 卡错 1: TF 卡正常 DAT6 : JM3L 上 JMO 文件个数 DAT7 : 0:JM3L 在非编程态; 1:JM3L 在编程态 DAT8 : SNO 0 编程器编号 DAT9 : SNO 1 DAT10: SNO 2 DAT11: SNO 3 DAT12: 20 个字节的编程器名 DAT33: 1: 在 FLASH 2: 在 TF 卡 自启动文件所在盘 DAT34: 13 个字节的自启动文件名



0x09	RES_FILE	DAT1 : SNO: 0: 不写串号 1: 写串号 DAT2 : AES: 0: 没有用户加密 1: 有用户加密 DAT3 : PID: 0: 没有编程器序列号限制 1: 有编程器序列号限制 DAT4 : ATE: bit3..bit0 ATE 控制的极性(高/低有效) DAT5 : 编程启动方式: 1: 按键 2: 自动 3: ATE DAT6 : PROG_CNT 4 个字节的编程记数 DAT10: CHKSUM 4 个字节的 HEX 校验和 DAT14: ICType 4 个字节的芯片类型 DAT18: 20 个字节的文件描述字串 DAT38: 1: 在 FLASH 2: 在 TF 卡 文件所在盘 DAT39: 13 个字节的文件名
0x0A	RES_IO	DAT1: 读取的 IO 端口值
0xFB	RES_OK	命令执行正确
0xFC	RES_ERROR	命令执行错误
0xFD	RES_MODE	命令发送时编程器状态不正确——有些命令只能在编程态执行，有些只能在非编程态执行 在 JM3L 上选文件时为非编程态，选中文件开始编程时为编程态
0xFE	RES_CHKSUM	命令校检和不对
0xFF	RES_UNKNOWN	命令没定义



二、通过命令行方式控制 JM3L

JMCMD 是一款控制台界面（CMD 窗口）运行的 JM3L 编程器控制软件，用户可能通过这个软件来了解编程器控制方法，用户如果要编写自己的 JM3L 控制软件，也可以通过它了解第三方开发用到的 DLL 接口。JMCMD 使用的 JMO 必须在 JM3L 上，如果不在，先用 DOWNLOAD 命令下载到编程器中。JMO 文件要以“按键启动”方式保存。我们先用几条常用指令说明怎么进行编程操作，后面再详细介绍每条指令。

```
E:\JM3_jmcmd\JMCMD\JMCMD.EXE
J Master Command line interface Version 1.00
port usb
00 OK
jm 0
1 SNO: 0x80002005 NAME: JM3 测试2
2 SNO: 0x80002276 NAME:
3 SNO: 0x80002277 NAME:
```

```
E:\JM3_jmcmd\JMCMD\JMCMD.EXE
00 OK
jm 0
1 SNO: 0x80002005 NAME: JM3 测试2
2 SNO: 0x80002276 NAME:
3 SNO: 0x80002277 NAME:
file 1.0
FNO DISK FILENAME DESCRIBE CHKSUM COUNTER
1 FLASH TEST052.JMO test052 0x00007F80 0
2 FLASH TEST13_1.JMO BLINKY_XMC1300.HEX 0x00FB9575 0
3 FLASH TST1302.JMO test 1302 0x031CC2F0 0
4 TF R24FINE2.JMO fn250K 0x020E0FC0 0
5 TF GD32.JMO gd32test 0x0102E7E7 0
file 2.0
FNO DISK FILENAME DESCRIBE CHKSUM COUNTER
1 FLASH JM3_13_1.JMO BLINKY_XMC1300.HEX 0x03107513 0
file 1.3
DISK: FLASH
FILENAME: TST1302.JMO
DESCRIBE: test 1302
CHKSUM : 0x031CC2F0
DATETIME: 2020/5/15 16:31:00
CHIPTYPE: 0x00000002
SERIAL : N
AES : N
PTD : N
COUNTER : 0
START : KEY
ATEM : 0x00
AUTOFILE: N
```



```
DISK : FLASH
FILENAME: TST1302.JMO
DESCRIBE: test 1302
CHKSUM : 0x031CC2F0
DATETIME: 2020/5/15 16:31:00
CHIPTYPE: 0x00000002
SERIAL : N
AES : N
PID : N
COUNTER : 0
START : KEY
ATEM : 0x00
AUTOFILE: N
-openjmo 1,3
00 OK
-run 1,1
00 OK
-run 1,2
35 PROG CONNECT
-run 2,1
70 ERROR CONNECT
-closejmo 1,3
ERROR: 71 ERROR_PARAMETER
-closejmo
ERROR: 71 ERROR_PARAMETER
-closejmo
ERROR: 71 ERROR_PARAMETER
-closejmo 1
00 OK

-help
JM 0/JM_NO
PORT USB/COM
BAUDRATE 115200/230400/460800/500000/1000000
CMD "FILENAME"
LOGON CREATE/APPEND, "FILENAME"
LOGOFF
WRITELOG "STRING"
JM 0/JM_NO
MODE JM_NO
RESET 0/JM_NO
LASTERROR JM_NO
FID 0/JM_NO, TF/FLASH, "FILENAME"
OPENJMO 0/JM_NO, 0/F_NO
CLOSEJMO 0/JM_NO
FILE JM_NO, 0/F_NO
DELETE 0/JM_NO, 0/F_NO
DOWNLOAD 0/JM_NO, TF/FLASH, "FILENAME"
FORMAT 0/JM_NO
NAME JM_NO
SNO JM_NO
SETNAME 0/JM_NO, "NAME"
SETAUTOFILE 0/JM_NO, 0/F_NO
SETPASSWORD 0/JM_NO, "PASSWORD"
VERSION JM_NO
RUN 0/JM_NO, 0/1/2
START 0/JM_NO, 0/1/2
MESSAGE JM_NO, 0/1/2
COUNTER JM_NO
SERIAL 0/JM_NO, "STRING"HEX
POWER 0/JM_NO, VCC
READIO JM_NO, 1/2, A
WRITEIO 0/JM_NO, 0/1/2, A, D
EXIT
更详细说明参见DLL使用说明
```

2.1 命令行方式控制文件列表

[主要文件说明]

JM_CMD.EXE	调用 dll 控制编程器的动作、监测编程器状态
JM_CMD_OP.bat	调用 JM_CMD.EXE 的批处理文件
JM_CMD_OP.TXT	由 JM_CMD_OP.bat 批处理生成，文件名可由批处理更改

2.2 命令行方式控制简单范例与解释

命令范例：start /wait jmcmd.exe -openjmo 1,4 -run 1,1 -exit

Start 启动单独的“命令提示符”窗口来运行 jmcmd.exe 程序或命令。



/wait 启动应用程序，并等待其结束。

-OPENJMO 1, 4 打开 1#烧录器的 4#文件并进入编程状态，等待编程命令。

-run 1,1 对 1#烧录器的 1 号烧录通道执行 1 次烧录动作，完成后返回

-exit 退出 CMD.EXE 程序，关闭命令行窗口

用户可以通过建立 BAT 批处理文件，调用 JMCMD.EXE 并设置符合项目要求的参数实现对 JM3L 实现符合量产设备要求的复杂控制。

一个以批处理方式运行 JMCMD 进行编程的例子

```
@echo off
```

```
echo.
```

```
echo Start to execute MCU on board programming
```

```
echo It will take a few seconds, please wait..
```

```
echo.
```

```
start /wait jmcmd.exe -openjmo 1,4 -run 1,1 -exit
```

```
REM -openjmo 1,4 打开第一台编程器中的第四个 JMO。
```

```
REM 如果想要用 JMO 的文件名操作需要用两条命令： -fid 1,FLASH,"abc.jmo" 以及  
-openjmo 1, 0
```

```
REM -fid 命令找出 FLASH 盘上的"abc.jmo"的 JMO 文件序号； -openjmo 1,0 打开这个序  
号的 JMO
```

```
REM -run 1,1 启动第一台编程器 P 1 编程口编程，并等待编程结束，编程如果正确返回 OK  
(0)，不正确返回 31-63 的值，表示是在那个状态出错了
```

```
REM -exit 退出 jmcmd 程序，最后一条必须是这条命令，否则 jmcmd 会等待用户在界面输  
入新命令
```

```
if errorlevel 1 goto error
```

```
if errorlevel 0 goto success
```

```
:error
```

```
echo FAIL>JM_CMD_OP.TXT
```

```
echo.
```

```
echo programming FAILED!!!
```

```
goto end
```

```
:success
```

```
echo PASS>JM_CMD_OP.TXT
```

```
echo.
```

```
echo programming SUCCESSFUL
```

```
:end
```

```
echo.
```



2.3 命令行方式控制参数列表

***注 1：所有入参中的“JMNO”为编程器序号（编程器首个序号为 1）；

当命令行程序所在路径下不存在序号定义文件“JM_DEVICE.LST”时使用 JM 命令前，编程器序号则按系统完成枚举的先后顺序排列，因此当连接多台编程器的计算机每次重启系统后每个编程器的序号可能会改变；

当命令行程序所在路径下存在序号定义文件“JM_DEVICE.LST(由用户创建或修改已提供的模板)”时，使用 JM 命令后，编程器序号将重新按 JM_DEVICE.LST 内的序列号先后顺序排列；

[参数说明]

1.BAUDRATE

功能：设置波特率 只有用串口连接方式连接编程器时有效，缺省波特率为：115200

参数：115200/230400/460800/500000/1000000

返回：状态信息

2.CLOSEJMO JM_NO

功能：关闭打开的 JMO 文件，退出编程状态，返回监控态

参数：编程器序号，如果只连接一台编程器，就是 1

返回：状态信息

3.CMD "FILENAME"

功能：运行命令文件

参数：“FILENAME” FILENAME 文件中是要执行的命令行

返回：状态信息

***** 注 1：命令文件最大支持 1024 行，命令超过 1024 行时，可拆分到多个文件中使用 CMD 命令分次调用

4.COUNTER JM_NO

功能：读取编程限定次数

参数：JM_NO: 编程器序号

返回：当前所剩余的编程次数

5.DELETE JM_NO, F_NO/0

功能：删除编程器中的 JMO 文件

参数：JM_NO: 编程器序号 F_NO: JMO 文件序号 或 0: 用 FID 命令查找出的 JMO 文件序号

返回：状态信息

6.DOWNLOAD JM_NO, TF/FLASH,"FILENAME"

功能：下载 JMO 文件到编程器

参数：JM_NO: 编程器序号 TF/FLASH 指定要下载到的盘，TF 表示下载到 TF 卡，FLASH



表示下载到内部 FLASH

返回：状态信息

7.EXIT

功能：退出命令行窗口 在退出前会自动 DISCONNECT 已连接的编程器并 LOGOFF 关闭 LOG 文件

参数：无

返回：EXIT 前一条命令的状态信息

8.FID JM_NO, TF/FLASH,"FILENAME"

功能：查找指定文件的文件序号

参数：JM_NO: 编程器序号 TF/FLASH 指定要查找的盘，TF 表示 TF 卡，FLASH 表示内部 FLASH。"FILENAME"指令 JMO 文件名

返回：查找文件对的文件序号。在 OPENJMO,SETAUTOFILE,DELETE 中可以使用 FID 来指定相应文件

9.FILE JM_NO, 0/F_NO

功能：读取编程器中的 JMO 文件信息

参数：JM_NO: 编程器序号 0: 读取编程器中 JMO 的文件个数 1..N: 显示文件序号指定 JMO 的信息

返回：参数为 0 时返回编程器中 JMO 的文件个数，参数为 1..N 时返回文件序号指定 JMO 的信息

DISK : TF/FLASH JMO 文件是在 TF 卡还是内部 FLASH 盘
FILENAME : JMO 文件名
DESCRIBE : JMO 文件描述
CHKSUM : JMO 文件校验和
DATETIME : 创建 JMO 文件的日期、时间
CHIPTYPE : JMO 文件中的芯片型号
SERIAL: Y/N JMO 文件中是否设置了编程串号
AES: Y/N JMO 文件中是否设置了加密密码
PID : Y/N JMO 文件中是否限定了编程器序列号
COUNTER : 剩余次数/总次数 0/0 表示没有限定编程次数
START : KEY/AUTO/ATE 编程启动方式
ATEM : JMO 文件中设定的 ATE 接口有效电平
AUTOFILE : Y/N JMO 文件是不是自启动文件

10. FORMAT JM_NO

功能：格式化编程器中的 FLASH 盘

参数：JM_NO: 编程器序号

返回：状态信息

11. JM JM_NO

功能：计算电脑连接的编程器台数/连接方式

参数：JM_NO: 编程器序号



返回：当 JMO=0 时返回计算机连接的编程器台数

当 JMO=1..N 时返回这台编程器的连接方式，0：表示未连接 1:表示编程器直接连接计算机

HB.LB: 高位表示通过编程器扩展板连接, 高字节(HB)表示搞展板号,低字节(LB)表示扩展上编程器的序号

12. LASTERROR JM_NO

功能：读编程器上一次操作的返回值

参数：JMNO: 编程器序号

返回：上条命令的返回状态信息

13. HELP

功能：显示常用命令的命令列表

参数：无

返回：显示常用命令的命令列表

14. LOGOFF

功能：关闭 LOG 文件

参数：无

返回：状态信息

15. LOGON CREATE/APPEND, "FILENAME"

功能：记录编程信息

参数：CREATE: 创建一个新的名为 "FILENAME"LOG 文件 APPEND: 在原 "FILENAME"LOG 文件最后填加

返回：状态信息

16. MESSAGE JM_NO, 1/2/0

功能：读取编程器进度信息

参数：JM_NO: 编程器序号 1/2/0 指定编程口 1: P1 2: P2 0: P1 及 P2

返回：当前进度信息，是一个 32 位的字。如果当前没有新的信息，就返回最近的一次信息

// B3	B2	B1	B0
// PNO	SPPEDBAR	ATE	STATUS
// 1: P1 口信息	进度条位置	NG,OK,BUSY	编程序所处状态
// 2: P2 口信息			PROG_NULL 到 PROG_FINISH

17. MODE JM_NO

功能：显示编程器当前所处状态

参数：JM_NO: 编程器序号

返回：状态信息

18. NAME JM_NO

功能：读取编程器名称

参数：JM_NO: 编程器序号



返回：编程器名称字符串

19. OPENJMO JM_NO, F_NO/0

功能：打开 JMO 文件，进入编程状态

参数：JM_NO: 编程器序号 F_NO: JMO 文件序号 0: 由 FID 命令指定的 JMO 文件序号

返回：状态信息

20. PAUSE

功能：暂停；显示 Press “Enter” to continue...

参数：无

返回：0

21. POWER JM_NO, 1/2/0, VCC

功能：控制编程口的电源脚

参数：JM_NO: 编程器序号 1/2/0: 编程器端口号 VCC: 电压值 0: OFF 3: 1.8V
4: 2.5V 5: 3.3V 6: 3.6V 7: 5.0V

返回：状态信息

22. READIO JM_NO, 1/2, ADR

功能：读取编程寄存器

参数：JM_NO: 编程器序号 1/2: 编程器端口号 ADR: 编程寄存器地址 0..15

返回：指定编程端口号的指定寄存器值

23. RESET JM_NO

功能：复位编程器，复位后编程器断开连接，需要使用时必须用 CONNECT 命令再次连接

参数：JM_NO: 编程器序号

返回：状态信息

24. RUN JM_NO, 1/2/0

功能：执行一次编程，编程完成后返回

参数：JM_NO: 编程器序号

1/2/0 1: 编程端口 1 2: 编程口 2 0: 编程口 1 及编程口 1 同时

编程返回：状态信息

25. RUNTEST JM_NO, 1/2/0, COUNT, DELAYms

功能：编程测试，可指定编程次数、编程结束时间间隔

参数：JM_NO: 编程器序号

1/2/0 1: 编程端口 1 2: 编程口 2 0: 编程口 1 及编程口 1 同时编程

COUNT: 编程次数

DELAYms:编程结束后延迟若干毫秒后启动下一次编程

返回：状态信息



26. SERIAL JM_NO, 1/2/0, "STRING"HEX

功能：设定外部串号的值

参数：JM_NO: 编程器序号 1/2/0 1: 编程端口 1 2: 编程端口 2 0: 编程端口 1 及编程端口 2

返回：状态信息

27. SETAUTOFILE JM_NO, F_NO/0

功能：设定自启动文件

参数：JM_NO: 编程器序号 F_NO: JMO 文件序号, 当 F_NO 为 0 时表示取消自启动文件
0: 由 FID 命令指定的 JMO 文件序号

返回：状态信息

28. SETNAME JM_NO,"NAME"

功能：设定编程器名称

参数：JM_NO: 编程器序号 "NAME": 编程器名称

返回：状态信息

29. SETPASSWORD JM_NO, "PASSWORD"

功能：设定编程器密码

参数：JM_NO: 编程器序号 "PASSWORD": 编程器密码

返回：状态信息

30. SLEEP ms

功能：暂停执行

参数：ms: 暂停执行指定毫秒

返回：0(SLEEP 函数不与编程器进行任何通信)

31. SNO JM_NO

功能：读取编程器编号

参数：JM_NO: 编程器序号

返回：编程器编号

32. START JM_NO, 1/2/0

功能：启动编程，启动后不等待编程结束立即返回。需要用 MESSAGE 命令判断编程是否完成

参数：JM_NO: 编程器序号 1/2/0 1: 编程端口 1 编程 2: 编程端口 2 编程 0: 编程口 P1 及 P2 同时编程

返回：状态信息

33. VERSION JM_NO

功能：读取编程器软件版本号

参数：JM_NO: 编程器序号

返回：编程器软件版本号



34. WRITEIO JM_NO, 1/2/0, ADR, DAT

功能：设置编程寄存器

参数：JM_NO: 编程器序号 1/2/0:编程器端口 ADR:编程寄存器地址 0..15 DAT: 写入数据

返回：状态信息

35. PORT USB/COM

功能：选择软件与编程器的连接方式 缺省方式是 USB

参数：USB: USB 口接连 COM: 串口连接

返回：状态信息

2.4 命令行方式控制注意事项

2.4.1 JMCMD 控制的编程文件(JMO)必须在编程器上。如果不在需预先用 DOWNLOAD 命令下载到编程器中。

2.4.2 编程器上电后处于监控态，如果编程器有自启动文件，编程器上电后自动打开 JMO 文件，进入编程态。

2.4.3 出错不影响编程器的状态。可以用 MODE 命令查看当前状态。只有 2 条命令可以改变编程器状态：OPENJMO(将编程器从监控态转为编程态)、 CLOSEJMO(将编程器从编程态转为监控态)；

2.4.4 除下表命令外，其它命令均可在任意状态下执行

命令行控制参数一览表

监控态可用参数	编程态可用参数
MODE	MODE
JM	MESSAGE
SETAUTOFILE	RUN (编程结束返回)
DOWNLOAD	START (立即返回)
DELETE	SERIAL
FID	POWER
FILE	WRITEIO
FORMAT	READIO
SETPASSWORD	CLOSEJMO
SNO	COUNTER
NAME	
VERSION	
RESET	
LOGON	
LOGOFF	
WRITELOG	
OPENJMO	
EXIT	



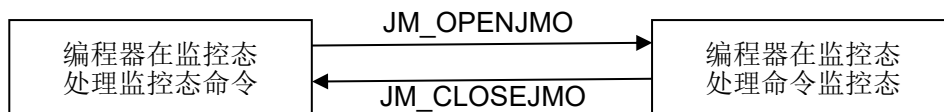
三、通过 DLL 动态链接库控制 JM3L（上位机二次开发）

JM3L 提供 Window 平台下的 DLL 动态链接库（32 位），并提供简单演示例程。

3.1 编程器状态

编程器有 2 个重要状态：监控态、编程态（具体见《名词解释》章节）

编程器在监控态下执行监控态相关命令，在编程态执行编程态相关命令。在监控态下用 JM_OPENJMO 打开一个 JMO 文件进入编程态、在编程态下用 JM_CLOSEJMO 可退出编程态进入监控态。



3.2 用户程序基本流程图

1	调用 JM_LOGON	[非必要]如需记录运行 LOG, 先打开 LOG 功能
2	调用 JM_JM(0)	连接所有联机编程器并生成序号, 序号在每次启动计算机后有可能发生改变, 如需锁定编程器序号, 可通过创建并修改 JM_DEVICE.LST 内编程器序列号顺序锁定, 具体参见 JM_JM 函数的“范例”
3	调用 JM_DOWNLOAD	如待编程文件不在编程器内
4	调用 JM_OPENJMO	打开待编程的编程文件
5	调用 JM_RUN 或 JM_START	启动编程, 二者区别参考函数列表
6	调用 JM_MESSAGE	读状态, 如 RUN 启动编程且返回 0 可忽略此步骤
7	调用 JM_CLOSEJMO	关闭文件, 切换到监控态
8	调用 JM_OPENJMO	[非必要]如需更换其他文件, 打开后重复步骤 5
9	调用 JM_LOGOFF	[非必要]关闭 LOG 功能
10	调用 JM_EXIT	关闭主程序前调用一次



3.2 DLL 接口函数列表

***注 1：所有入参中的“JMNO”为编程器序号（编程器首个序号为 1）；

当 DLL 所在路径下不存在序号定义文件“JM_DEVICE.LST”时或调用 JM_JM 函数前，编程器序号则按系统完成枚举的先后顺序排列，因此当连接多台编程器的计算机每次重启系统后每个编程器的序号可能会改变；

当 DLL 所在路径下存在序号定义文件“JM_DEVICE.LST(由用户创建或修改已提供的模板)”时，调用 JM_JM 函数后，编程器序号将重新按 JM_DEVICE.LST 内的序列号先后顺序排列；

二次开发接口功能/函数一览表

监控态可用功能	编程态可用功能
1. 读当前编程器模式 调用函数：JM_MODE	1. 读当前编程器模式 调用函数：JM_MODE
2. 连接编程器 调用函数：JM_JM	2. 读取编程器状态 调用函数：JM_MESSAGE
3. 设定/取消自启动文件 调用函数：JM_SETAUTOFILE	3. 编程 调用函数：JM_RUN（编程结束返回） 调用函数：JM_START(立即返回)
4. 下载 JMO 文件 调用函数：JM_DOWNLOAD	4. 外部数据注入 调用函数：JM_SERIAL
5. 删除 JMO 文件 调用函数：JM_DELETE	5. 编程口电源控制 调用函数：JM_POWER
6. 查找文件名对应的序号 调用函数：JM_FID	6. 编程寄存器控制 调用函数：JM_WRITEIO 调用函数：JM_READIO
7. 查看 JMO 信息 调用函数：JM_FILL	7. 关闭 JMO 调用函数：JM_CLOSEJMO
8. 格式化 FLASH 盘 调用函数：JM_FORMAT	8. 读编程记数 调用函数：JM_COUNTER
9. 设置编程器密码 调用函数：JM_SETPASSWORD	
10. 读编程器序号 调用函数：JM_SNO	
11. 设置编程器名称 调用函数：JM_NAME	
12. 读编程器版本号 调用函数：JM_VERSION	
13. 复位编程器 调用函数：JM_RESET	
14. 打开、关闭、写 LOG 文件 调用函数：JM_LOGON 调用函数：JM_LOGOFF 调用函数：JM_WRITELOG	
15. 打开 JMO 调用函数：JM_OPENJMO	
16. 退出 调用函数：JM_EXIT	



JM_BAUDRATE 函数

原 型:	C : u32 JM_BAUDRATE(u32 BAUDRATE); PAS : function JM_BAUDRATE(BAUDRATE: u32): u32;
功 能:	设置串口通信模式时的通信波特率
入 参:	波特率, 缺省值 115200, 可用值: 115200/230400/460800/500000
返回值:	状态值
注意项:	1.此函数仅适用于 JM_COM.DLL

JM_CLOSEJMO 函数

原 型:	C : u32 JM_CLOSEJMO(u32 JMNO); PAS : function JM_CLOSEJMO(JMNO: u32): u32;
功 能:	关闭已打开的 JMO(返回监控态)
入 参:	JMNO: 编程器序号, JMNO=0..N, JMNO=0 表示作用于所有编程器
返回值:	状态值
注意项:	---

JM_COUNTER 函数

原 型:	C : u32 JM_COUNTER (u32 JMNO); PAS : function JM_COUNTER (JMNO: u32): u32;
功 能:	读剩余编程次数
入 参:	JMNO: 编程器序号, JMNO=1..N
返回值:	剩余的编程次数; 如果 JMO 没有限定编程次数, 返回的值无意义
注意项:	---

JM_DELETE 函数

原 型:	C : u32 JM_DELETE(u32 JMNO, u32 F); PAS : function JM_DELETE(JMNO: u32; F: u32): u32;
功 能:	删除编程器中文件序号为 N 的 JMO 文件
入 参:	F: 文件序号, F=1~72
返回值:	状态值
注意项:	---

JM_DOWNLOAD 函数

原 型:	C : u32 JM_DOWNLOAD(u32 JMNO, u32 F, char * p); PAS : function JM_DOWNLOAD(JMNO: u32; F: u32; p: pointer): u32;
功 能:	下载 JMO 文件到编程器
入 参:	F: 下载目标盘 0: 内部 FLASH 1: TF p: 文件名字符串指针(0 结束)
返回值:	状态值
注意项:	---



JM_EXIT 函数

原 型:	C : u32 JM_EXIT(void); PAS : function JM_EXIT: u32;
功 能:	通知 DLL 即将结束对其的调用, DLL 将做结束调用前的处理。
入 参:	无
返回值:	状态值
注意项:	在主程序结束调用 DLL 前必须调用一次

JM_FID 函数

原 型:	C : u32 JM_FID(u32 JMNO, u32 F, char * p); PAS : function JM_FID(JMNO: u32; F: u32; p: pointer): u32;
功 能:	用文件名查找对应的文件序号
入 参:	F: 下载到的盘 0:内部 FLASH 1: TF p: 文件名字符串指针(0 结束)
返回值:	文件所对应的文件序号, 返回值=0 表示未找到
注意项:	---

JM_FILE 函数

原 型:	C : char * JM_FILE(u32 JMNO, u32 F); PAS : function JM_FILE(JMNO: u32; F: u32): pointer;
功 能:	读编程器 JMO 文件属性
入 参:	JMNO: 编程器序号, JMNO=0..N, JMNO=0 表示作用于所有编程器 F: 文件序号, N=1~72
返回值:	JM_FILE(JMNO, 0) 返回编程器中 JMO 文件个数 (字符串, 0 结束) JM_FILE(JMNO, N) 返回编程器中 N 号文件的属性 (字符串, 0 结束), DISK, FILENAME, DESCRIBE, CHKSUM, DATETIM, CHIPTYPE, SERIAL, AES, PID, COUNTER, S TART, ATEM, A UTOFILE, 每个属性用 “,” (逗号) 隔开
注意项:	---

JM_FORMAT 函数

原 型:	C : u32 JM_FORMAT(u32 JMNO); PAS : function JM_FORMAT(JMNO: u32): u32;
功 能:	格式化编程器中的内置 FLASH 盘(不支持格式化外置 TF 卡)
入 参:	JMNO: 编程器序号, JMNO=0..N, JMNO=0 表示作用于所有编程器
返回值:	状态值
注意项:	---

JM_JM 函数

原 型:	C : u32 JM_JM(u32 JMNO); PAS : function JM_JM(JMNO: u32): u32;
功 能:	返回计算机连接编程器数量、连接方式、创建联机编程器列表文件 JM_ONLINE.LST、按 JM_DEVICE.LST 自定义编程器序号



入 参:	JMNO: 编程器序号, JMNO=0..N, JMNO=0 表示作用于所有编程器															
返回值:	<p>当 JMNO=1..N 时返回指定编程器序号。</p> <p>当 JMNO=0 时,</p> <p>1、返回联机编程器的数量(bit7..0)、是否存在自定义编号文件(bit30)、自定义编号文件内序列号与所有在线编程器序列号是否一致(bit31);</p> <p>2、在可执行程序路径下创建联机编程器列表文件:JM_ONLINE.LST</p> <p>3、对联机编程器进行编号, 编号规则:</p> <p>a) 当可执行文件所在路径下不存在 JM_DEVICE.LST(返回值 bit30=0)文件时, 按编程器枚举顺序进行编号, 首个枚举成功的编程器序号为 1, 以此类推, 当连接多台编程器的计算机重启后编程器序号可能会改变;</p> <p>b) 当可执行文件所在路径下存在 JM_DEVICE.LST(返回值 bit30=1)时, 分 2 种情况:</p> <p>如果 JM_DEVICE.LST 内的序列号与在线编程器序列号完全相符 (次序不同会被忽略), 则按 JM_DEVICE.LST 的顺序编号, 也即 JM_DEVICE.LST 第 1 行序列号编号为 1, 以此类推, 返回值中的 bit31(最高位)清零, bit15..0 位为在线编程器数量;</p> <p>如果 JM_DEVICE.LST 内的序列号与在线编程器序列号不完全相符 (JM_DEVICE.LST 内有未在线序列号或在线编程器序列号不在 JM_DEVICE.LST 内), 则按现将 JM_DEVICE.LST 中在线编程器按 JM_DEVICE.LST 内的顺序编号, 再将在线但未在 JM_DEVICE.LST 内列出的按 USB 枚举顺序编号, 而未在线的编程器不予编号, 返回值中的 bit31(最高位)置 1, bit7..0 位为在线编程器数量</p>															
注意项:	<p>1. JM_DEVICE.LST 格式要求:</p> <p>1.1. 文件内均为数字, 有非数字的行将被忽略;</p> <p>1.2. 每行 1 个序列号, 回车符作为行结束符)</p> <p>JM_DEVICE.LST 的内容可以在所有编程器都联机时调用 JM_JM(0)后从 JM_ONLINE.LST 复制并修改 (包括增减序列号以及顺序调整)。</p>															
范 例:	<p>调用 JM_JM(0)后自动生成 JM_ONLINE.LST (假设内容如下表), 在 JM_ONLINE.LST 基础上编辑并另存 JM_DEVICE.LST (内容如下), 再次调用 JM_JM(0)后最终序号如下:</p> <table><tr><th>JM_ONLINE.LST 内容</th><th>JM_DEVICE.LST 内容</th><th>最终序号</th></tr><tr><td>80000001</td><td>80000001</td><td>1</td></tr><tr><td>80000002</td><td>80000002</td><td>2</td></tr><tr><td>80000003</td><td>80000003</td><td>3</td></tr><tr><td>80000006</td><td>80000005</td><td>4</td></tr></table>	JM_ONLINE.LST 内容	JM_DEVICE.LST 内容	最终序号	80000001	80000001	1	80000002	80000002	2	80000003	80000003	3	80000006	80000005	4
JM_ONLINE.LST 内容	JM_DEVICE.LST 内容	最终序号														
80000001	80000001	1														
80000002	80000002	2														
80000003	80000003	3														
80000006	80000005	4														

JM_LASTERROR 函数

原 型:	C : u32 JM_LASTERROR(u32 JMNO); PAS : function JM_LASTERROR(JMNO: u32): u32;
功 能:	读编程器上一次操作的返回值
入 参:	JMNO: 编程器序号, JMNO=1..N
返回值:	状态值
注意项:	---



JM_LOGOFF 函数

原 型:	C : u32 JM_LOGOFF(); PAS : function JM_LOGOFF: u32;
功 能:	关闭 LOG 功能; 关闭 LOG 功能后不再记录 LOG 信息
入 参:	无
返回值:	状态值
注意项:	---

JM_LOGON 函数

原 型:	C : u32 JM_LOGON(u32 A, char * s); PAS : function JM_LOGON(A: u32, s: pointer): u32;
功 能:	打开 LOG 功能, 开始记录 LOG
入 参:	A: 打开方式 0: 新建 LOG 文件 1: 在原 LOG 文件最后一行后添加 s: 指向文件名字串 (以 0 结束)
返回值:	状态值
注意项:	---

JM_MESSAGE 函数

原 型:	C : u32 JM_MESSAGE(u32 JMNO, u32 P); PAS : function JM_MESSAGE(JMNO: u32; P: u32): u32;
功 能:	读取编程器信息, 编程器在编程的过程中会将当前的进度, 编程操作等信息放到一个信息队列中(最多 32 条信息), 由上位机读取 JM_MESSAGE 用于 JM_START 启动后获取编程进度及编程状态, 过程: 1. 用 JM_START 启动编程 2. 用 JM_MESSAGE 等待 ATE.BUSY 有效, 这时表示编程开始 3. 用 JM_MESSAGE 等待 ATE.BUSY 无效, 这时表示编程结束 4. 用 JM_MESSAGE 查看 ATE.OK 及 ATE.NG, 如 ATE.OK 有效, 表示编程正确; 如果 ATE.NG 有效表示编程出错; 出错的具体信息在 BYTE0 中。
入 参:	JMNO: 编程器序号, JMNO=1..N P: 编程口; P=0/1/2, 1: 仅读取编程口 P1 信息, P2 信息会被丢弃; 2: 仅读取编程口 P2 信息, P1 信息会被丢弃; 0: 读取编程口 P1 及 P2 信息
返回值:	一个 32 位 (4 个字节), 每个字节对应不同的状态, 其中 BYTE3(最高字节): 1: 信息来自编程口 P1 2: 信息来自编程口 P2 BYTE2: 进度条信息 100 表示进度条满 BYTE1: ATE 接口信息(bit0 为 BUSY, bit1 为 OK, bit2 为 NG) BYTE0: 编程口所处的状态 如果当前没有新的状态信息, 返回最后一次的信息
注意项:	1.JM_START 返回时编程才刚开始, 调用后需要周期性调用 JM_MESSAGE 读取编程口状态直至启动编程口的 BUSY 位无效(表示编程已结束), 读取的状态可以用以刷新进度条、显示编程结果。



	2.BYTE1 中 BUSY、OK、NG 相应位为 1 表示有效，0 表示无效。 BUSY、OK、NG 在 ATE 接口的输出有效电平可以在生成 JMO 时设置为高有效或低有效；使用 JM_FILE 获取的文件属性中的“ATEM”即为文件设置的 BUSY、OK、NG(以及启动信号 START)有效电平。
--	---

JM_MODE 函数

原 型：	C : u32 JM_MODE(u32 JMNO); PAS : function JM_MODE(JMNO: u32): u32;
功 能：	当前编程器所处的工作状态，编程有两个工作状态 MODE_MONITOR: 监控态 MODE_PROGRAM: 编程态
入 参：	JMNO: 编程器序号，JMNO=1..N
返回值：	状态值
注意项：	---

JM_NAME 函数

原 型：	C : char * JM_NAME(u32 JMNO); PAS : function JM_NAME(JMNO: u32): pointer;
功 能：	读取编程器名称
入 参：	JMNO: 编程器序号，JMNO=1..N
返回值：	编程器名字字符串指针(0 结束)
注意项：	---

JM_OPENJMO 函数

原 型：	C : JM_OPENJMO(u32 JMNO, u32 F); PAS : function JM_OPENJMO(JMNO: u32; F: u32): u32;
功 能：	打开一个 JMO 文件 如果当前为 MODE_PROGRAM 编程态，会报错并保持打开已打开的 JMO 文件。编程态时，必须先使用 JM_CLOSEJMO 关闭已打开 JMO 文件后，才可打开另一个 JMO
入 参：	JMNO: 编程器序号，JMNO=0..N, JMNO=0 表示作用于所有编程器 F: JMO 文件的序号 F 为 0 表示用上一个 FID 函数的返回值为序号
返回值：	状态值
注意项：	F 是 JMO 文件的序号，如需打开指定文件名的 JMO，先用 FID 函数获取指定文件名 JMO 文件的序号，再用 JM_OPENJMO(JMNO, F)打开

JM_POWER 函数

原 型：	C : u32 JM_POWER(u32 JMNO, u32 P, u32 V); PAS : function JM_POWER(JMNO: u32; P: u32; V: u32): u32;
功 能：	控制编程口 PIN1 的电源输出
入 参：	JMNO: 编程器序号，JMNO=0..N, JMNO=0 表示作用于所有编程器 P: 编程口 1: 编程口 P1; 2: 编程口 P2; 0: 编程口 P1 和 P2; V: 电 压 0: 关闭; 3: 1.8V; 4: 2.5V; 5: 3.3V; 6: 3.6V; 7: 5V
返回值：	状态值
注意项：	---

**JM_POWER 函数**

原 型:	C : u32 JM_POWER(u32 JMNO, u32 P, u32 V); PAS : function JM_POWER(JMNO: u32; P: u32; V: u32): u32;
功 能:	控制编程口 PIN1 的电源输出
入 参:	JMNO: 编程器序号, JMNO=0..N, JMNO=0 表示作用于所有编程器 P: 编程口 1: 编程口 P1; 2: 编程口 P2; 0: 编程口 P1 和 P2; V: 电 压 0: 关闭; 3: 1.8V; 4: 2.5V; 5: 3.3V; 6: 3.6V; 7: 5V
返回值:	状态值
注意项:	---

JM_READIO 函数

原 型:	C : u32 JM_READIO(u32 JMNO, u32 P, u32 A); PAS : function JM_READIO(JMNO: u32; P: u32; A: u32): u32;
功 能:	读取编程口控制寄存器值(寄存器说明参见 JM_WRITEIO 函数)
入 参:	JMNO: 编程器序号, JMNO=1..N P: 编程口 1: 编程口 P1; 2: 编程口 P2 A: 编程口控制寄存器地址
返回值:	状态值
注意项:	---
范 例:	参见 JM_WRITEIO 函数范例

JM_RESET 函数

原 型:	C : u32 JM_RESET(u32 JMNO); PAS : function JM_RESET(JMNO: u32): u32;
功 能:	复位编程器, 复位后编程器将处于重新上电状态。
入 参:	JMNO: 编程器序号, JMNO=0..N, JMNO=0 表示作用于所有编程器
返回值:	状态值
注意项:	---

JM_RUN 函数

原 型:	C : u32 JM_RUN(u32 JMNO, u32 P); PAS : function JM_RUN(JMNO: u32; P: u32): u32;
功 能:	启动编程, 编程结束后才返回; 如需显示编程的进度“擦除”、“编程”、“校验”等过程信息, 请使用 JM_START 及 JM_MESSAGE
入 参:	JMNO: 编程器序号, JMNO=0..N, JMNO=0 表示作用于所有编程器 P: 编程口 1: 编程口 P1; 2: 编程口 P2; 0: 编程口 P1 和 P2;
返回值:	状态值
注意项:	JM_RUN 为阻塞型函数, 编程结束后才会返回编程结果

JM_SERIAL 函数

原 型:	C : u32 JM_SERIAL(u32 JMNO, u32 P, u32 L, char *s); PAS : function JM_SERIAL(JMNO: u32; P: u32; L: u32; s: pointer):u32;
------	---



功 能:	向数据区填充外部编号数据, 填充数据的具体地址长度在生成 JMO 时设定, 可根据编程次数、时间等生成不同的编程数据, 具体可参考编程器手册“自动编号”部分
入 参:	JMNO: 编程器序号, JMNO=0..N, JMNO=0 表示作用于所有编程器 P: 编程口 1: 编程口 P1; 2: 编程口 P2; 0: 编程口 P1 和 P2; L: 数据长度 s: 数据串指针
返回值:	状态值
注意项:	1.注入数据必须在启动编程(JM_RUN、 JM_START)前

JM_SETAUTOFILE 函数

原 型:	C : u32 JM_SETAUTOFILE(u32 JMNO, u32 F); PAS : function JM_SETAUTOFILE(JMNO: u32; F: u32): u32;
功 能:	设置自启动文件
入 参:	JMNO: 编程器序号, JMNO=0..N, JMNO=0 表示作用于所有编程器 F: 将文件序号为 F 的 JMO 文件设为自启动文件, F=1~72, F=255 表示取消自启动文件, F=0 时, 将 JM_FID 函数查到的序号文件设为自启动文件。(参见 JM_FID 函数)
返回值:	状态值
注意项:	---

JM_SETPASSWORD 函数

原 型:	C : u32 JM_SETPASSWORD(u32 JMNO, char * p); PAS : function JM_SETPASSWORD(JMNO: u32; p: pointer): u32;
功 能:	设置编程器的密码
入 参:	JMNO: 编程器序号, JMNO=0..N, JMNO=0 表示作用于所有编程器 p: 密码字符串指针(0 结束)
返回值:	状态值
注意项:	---

JM_SNO 函数

原 型:	C : u32 JM_SNO(u32 JMNO); PAS : function JM_SNO(JMNO: u32): u32;
功 能:	读取编程器序列号
入 参:	JMNO: 编程器序号, JMNO=1..N
返回值:	编程器序列号, 如果 JMNO 指定的序号编程器没有连接则返回值为 0
注意项:	---

JM_START 函数

原 型:	C : u32 JM_START(u32 JMNO, u32 P); PAS : function JM_START(JMNO: u32; P: u32): u32;
功 能:	启动编程, 启动后立即返回, 需用 JM_MESSAGE 获取编程进度和状态
入 参:	JMNO: 编程器序号, JMNO=0..N, JMNO=0 表示作用于所有编程器



	P: 编程口 1: 编程口 P1 2: 编程口 P2 0: 编程口 P1 和 P2
返回值:	状态值
注意项:	JM_START 返回时编程才刚开始, 调用后需要周期性调用 JM_MESSAGE 读取编程口状态直至启动编程口的 BUSY 位无效(表示编程已结束), 读取的状态可以用以刷新进度条、显示编程结果。

JM_VERSION 函数

原 型:	C : u32 JM_VERSION(u32 JMNO) ; PAS : function JM_VERSION(JMNO: u32): u32;
功 能:	读取编程器版本号
入 参:	JMNO: 编程器序号, JMNO=1..N
返回值:	编程器版本号
注意项:	---

JM_WRITEIO 函数

原 型:	C : u32 JM_WRITEIO(u32 JMNO, u32 P, u32 A, u32 D) ; PAS : function JM_WRITEIO(JMNO: u32; P: u32; A: u32; D: u32) :u32 ;
功 能:	写编程口控制寄存器值 JM_READIO, JM_WRITEIO 操作的编程口控制寄存器定义如下: 0: LED 寄存器 只写寄存器, 用于控制编程口的 LED bit7..bit5, bit1..bit0: 保留 bit2: 编程口灯红色 bit3: 编程口灯绿色 bit4: 编程口灯闪烁 1: IOCTRL 寄存器 只写寄存器, 用于控制编程口的方向 bit6..bit0: 7 个编程口 IO 输入/输出方向控制 0: 输入 1: 输出 Bit6..bit0 分别对应 PIN9,PIN7 ..PIN2 2: IODATA 寄存器 读写寄存器, 用于读入/输出编程口的信号 bit6..bit0: 7 个编程口 IO 输入/输出值 0: 低电平 1: 高电平 bit6..bit0 分别对应 PIN9,PIN 7 ..PIN2
入 参:	JMNO: 编程器序号, JMNO=0..N, JMNO=0 表示作用于所有编程器 P: 编程口 1: 编程口 P1; 2: 编程口 P2; 0: 编程口 P1 和 P2 A: 编程口控制寄存器地址 D: 写入数据
返回值:	状态值
注意项:	---
范 例:	// 写 LED JM_WRITEIO(1, 1, 0, 0x00)-----编程口 1 的 P1 灯灭 JM_WRITEIO(1, 1, 0, 0x04)-----编程口 1 的 P1 灯红色 JM_WRITEIO(1, 1, 0, 0x08)-----编程口 1 的 P1 灯绿色 JM_WRITEIO(1, 1, 0, 0x0C)-----编程口 1 的 P1 灯黄色



	JM_WRITEIO(1, 1, 0, 0x14)-----编程口 1 的 P1 灯红色且闪烁 // 写 IOCTRL JM_WRITEIO(1, 2, 1, 0x20)-----编程器 1 的 P2 口 PIN7 输出/其它输入 // 写 IODATA JM_WRITEIO(1, 2, 2, 0x20)-----编程器 1 的 P2 口 PIN7 输出 1/其它输出 0 // 读 IODATA JM_READIO(1, 2, 2) -----读取编程器 1 的 P2 口 IODATA 寄存器
--	---

JM_WRITELOG 函数

原 型:	C : u32 JM_WRITELOG(char *p); PAS : function JM_WRITELOG(p: pointer): u32;
功 能:	在 LOG 文件中写入一个字符串
入 参:	JMNO: 编程器序号, JMNO=0..N, JMNO=0 表示作用于所有编程器
返回值:	状态值
注意项:	---



附录：命令行、DLL 函数调用返回信息定义

// 状态信息定义

OK = 0; // 运行正常，没有错误

// 下面 1-10 为编程器上电自检错误信息

ERROR_MONI = 1; // 编程器监控程序错误

ERROR_ALGO = 2; // 编程器编程算法错误

ERROR_RTC = 3; // 编程器中的实时时钟错误，通常是电池没电了

ERROR_FLASH_FORMAT = 4; // 内部 FLASH 格式错误，需要重新格式化

ERROR_FLASH_TF = 5; // TF 卡错误

ERROR_FPGA_ID = 6; // 编程器中的 FPGA ID 错误

ERROR_FPGA_CFG = 7; // 编程器中的 FPGA 配置错误

ERROR_FPGA_IP = 8; // 编程器中的 FPGA IP 码错误

ERROR_FLASH_ID = 9; // 编程器中的 FLASH ID 错误

ERROR_FLASH_CFG = 10; // 编程器中的 FLASH 配置错误

// 20-24 为打开 JMO 的错误信息

ERROR_CHIPTYPE = 20; // 芯片类型错误

ERROR_PROGID = 21; // JMO 限定的编程器编号列表不包含当前编程器

ERROR_COUNTER = 22; // JMO 限定了编程记数，现在剩余编程记数为 0

ERROR_AES = 23; // JMO 设定了加密，现在编程器中的密码不对

ERROR_CHKSUM = 24; // JMO 的 CHKSUM 不对

// 31-63 为编程器当前的工作状态

PROG_NULL = 31; // 编程器空闲

PROG_WAIT_KEY = 32; // 编程器在等待用户按键启动编程

PROG_AUTO = 33; // 编程器在等待用户连接芯片

PROG_TEST_PIN = 34; // 编程器在编程完毕，可用 plugin 程序测试芯片

PROG_CONNECT = 35; // 编程器在连接芯片

PROG_RESTORE = 36; // 编程器在恢复芯片

PROG_ERASE = 37; // 编程器在擦除芯片

PROG_WRITE = 38; // 编程器在编程芯片

PROG_READ = 39; // 编程器在读芯片

PROG_VERIFY = 40; // 编程器在校验芯片

PROG_PROTECT = 41; // 编程器在编程加密信息

PROG_UNPROTECT = 42; // 编程器在解除芯片加密

PROG_BLANKCHECK = 43; // 编程器在做空片检查

PROG_CTRL = 44; // 编程器在 CTRL

PROG_EEPROM = 45; // 编程器在编程 EEPROM 或 OTP

PROG_BOOTLOAD = 46; // 编程器在编程 BOOTLOAD

PROG_ATE = 47; // 编程器在等待 ATE 接口的 START 信号

PROG_SVF = 48; // 下面 48-57 用于 JTAG 口编程

PROG_IDCODE = 49; // 编程器在执行 IDCODE

PROG_CHECK_OPT = 50; // 编程器在执行 CHECK_OPT

PROG_PIO = 51; // 编程器在执行 PIO



```
PROG_WRITE_OPT      = 52;    // 编程器在执行 WRITE_OPT
PROG_READ_OPT       = 53;    // 编程器在执行 READ_OPT
PROG_TEST           = 54;    // 编程器在执行 TEST
PROG_USER_ID0       = 55;    // 编程器在执行 USER_ID0
PROG_USER_ID1       = 56;    // 编程器在执行 USER_ID1
PROG_USER_ID2       = 57;    // 编程器在执行 USER_ID2
PROG_EXECUTE        = 58;    // 编程器在运行芯片中的程序
PROG_JMO_NO         = 59;    // 编程器在执行多 JMO 合并中的一个 JMO
PROG_DOWNLOAD       = 60;    // 编程器在下载编程程序
PROG_POWERON        = 61;    // 编程器在上电
PROG_POWEROFF       = 62;    // 编程器在下电
PROG_FINISH         = 63;    // 编程器完成工作
```

// 70-79 是命令执行过程中的错误信息

```
ERROR_CONNECT       = 70;    // 连接编程器错误
ERROR_PARAMETER     = 71;    // 命令的参数不对
ERROR_FILE_NOT_FOUND = 72;    // 文件未找到
ERROR_READ_FILE     = 73;    // 文件读错误
ERROR_WRITE_FILE    = 74;    // 文件写错误
ERROR_TOO_MANY_COMMAND = 75; // CMD 命令行超过 1024 行
ERROR_UNKNOWN_COMMAND = 76;  // 命令错
ERROR_JMO_FILE      = 77;    // JMO 文件错
ERROR_JMO_DOWNLOAD = 78;    // JMO 文件下载错
ERROR_UNKNOWN       = 79;    // 未知错误
```

// 80-81 为编程器状态

```
MODE_MONITOR      = 80;    // 编程器在监控态
MODE_PROGRAM      = 81;    // 编程器在编程态
```