山里智能®
SENSE INTELLIGENCE

NeoMove Lib 2.0

高集成运动控制函数库

使用手册 V2.0

版权声明

上海山里智能科技有限公司（以下简称山里智能）保留在不事先通知的情况下，修改本手册中的产品和产品规格等文件的权力。

山里智能不承担由于使用本手册或产品不当，所造成的的直接的、间接的、特殊的、附带的或相应产生的损失或责任。

山里智能具有本产品及其软件的专利权、版权和其它知识产权。未经授权，不得直接或者间接地复制、制造、加工、使用本产品及其相关部分。

运动中的机器有危险！使用者有责任在机器中设计有效的出错处理和安全保护机制，山里智能没有义务或责任对由此造成的附带的或相应产生的损失负责。

联系我们

地址：上海市浦东新区建韵路 500 号 1 栋 115

电话：+86-21-61183291

电子邮件：sales@sense-shanghai.com

网址：http://www.sense-shanghai.com

文档版本

版本号	修订日期
V1.0	2021.09.27, 初版发行
V1.1	2022.06.24 修正
V1.2	2022.11.22 修正
V1.3	2023.02.24 修正
V1.4	2023.06.06 修正
V2.0	2024.07.08 修正

前言

感谢选用山里智能运动控制器

为回报客户，我们将以品质一流的运动控制器、完善的售后服务、高效的技术支持，帮助您建立自己的控制系统。

山里智能产品的更多信息

山里智能的网址是 http://www.sense-shanghai.com.cn 在我们的网页上可以得到更多关于公司和产品的信息，包括：公司简介、产品介绍、技术支持、产品最新发布等等。

编程手册的用途

用户通过阅读本手册，能够了解运动函数库的基本控制功能，掌握函数的用法，熟悉特定控制功能的编程实现。最终，用户可以根据自己特定的控制系统，编制用户应用程序，实现控制要求。

编程手册的使用对象

本编程手册适用于具有 C++/C# 语言编程基础或 Windows 环境下使用动态链接库的基础，同时具有一定运动控制工作经验，对伺服或步进控制的基本结构有一定了解的工程开发人员。

编程手册的主要内容

本手册详细介绍了运动函数库的基本控制功能及编程实现。

目录

第 1 章　软件简介

1.1　Windows 系统下动态链接库的使用

1.1.1　Visual Studio .Net C#中的使用

（1）　在 Visual Studio 环境中选择新建工程

（2）　在对话框中选择 Visual C#下的 Windows 中的 Windows 窗体应用程序。

（3）　选择菜单“项目”->“添加引用”，在弹出的对话框中选择“浏览”，添加 NeoMoveCSharp.dll

（4）　在项目中添加以下申明

```
using NeoMoveCSharp;
```

1.1.2　Visual Studio .Net C++中的使用

（1）　在 Visual Studio 环境中选择新建工程

（2）　在对话框中选择 VisualC++下的 Win32 中的 Win32 项目

（3）　选择菜单“项目”->“工程属性”，在弹出的属性页对话框中选择“配置属性”->“链接器”->“输入”->“附加依赖项”，在附加依赖项中加入“NeoMove.lib。”

（4）　在应用程序文件中加入函数库头文件的声明，例如：

```
#include "NeoMove.h"
```

第 2 章　E2-M300 函数指令列表

2.1　系统和初始化函数

序号	函数名称	描述
系统和初始化函数		
01	Open	打开控制器
02	Close	关闭控制器
03	GetVersion	获取版本号
04	GetBusInfo	获取总线状态信息
05	GetSlaveInfo	获取从站信息
06	ControllerBusReset	控制器总线重置
07	GetMasterStatus	获取主站状态
08	EtherCATReadPDO	EtherCAT 总线读取 PDO
09	EtherCATWritePDO	EtherCAT 总线写入 PDO
10	EtherCATReadSDO	从从站中获取 SDO 信息
11	EtherCATWriteSDO	从从站中写入 SDO 信息

2.2 运动 IO 和运动状态函数

运动 IO 和运动状态函数		
12	ServoOnOff	伺服使能/去使能
13	GetAxisStatus	获取轴状态
14	SetZeroPos	设置当前位置为零位
15	SetPosition	设置轴位置
16	ClearAlarm	清除轴报警
17	SetEncoderScale	设置脉冲比例
18	SetSoftLimit	设置软限位
19	GetSoftLimit	获取软限位信号
20	SetAxisMode	设置轴总线模式
21	GetAxisMode	获取轴总线模式
22	SetVelocityType	设置速度类型
23	GetVelocityType	获取速度类型

2.3 轴运动函数

轴运动函数		
24	AxisHome	轴回零
25	AxisQuickStop	轴急停
26	AxisStop	轴停止
27	Motion_PositionMode_Abs	绝对位置运动
28	Motion_PositionMode_Rel	相对位置运动
29	Motion_JogMode	点动（Jog 模式）
30	Motion_TorqueMode	力矩模式运动

2.4 龙门函数

龙门函数		
31	SetSyncMasterSlave	设置主从轴同步
32	SetGantryCoupling1	设置龙门解耦 1
33	SetGantryCoupling2	设置龙门解耦 2
34	ResolveSync	主从轴同步解散
35	SetCompensation	设置轴补偿值
36	GetCompensation	获取轴补偿值
37	EnableCompensation	启用轴补偿

2.5 前瞻插补函数

前瞻插补函数		
38	SetPathIntplLookaheadConfiguration	设置前瞻插补配置参数
39	DismissPathIntplLookahead	前瞻插补组解散
40	Motion_PathIntplLookahead	启动前瞻插补运动
41	AddPathIntplLookahead	添加前瞻插补数据
42	Stop_PathIntpl	停止前瞻插补运动
43	GetPathIntplLookaheadStatus	获取前瞻插补运动状态
44	PathIntplLookaheadClearCount	清除前瞻插补缓存
45	GetPosition_PathIntplLookahead	获取插补通道轴位置

2.6 PVT 功能函数

PVT 功能函数		
46	SetPVTConfiguration	设置 PVT 配置参数
47	DismissPVTGroup	PVT 组解散
48	Motion_PVT	启动 PVT 运动
49	Stop_PVT	停止 PVT 运动
50	GetPVTStatus	获取 PVT 运动状态
51	PVTClearCount	清除 PVT 缓存
52	AddPVTPoint	添加 PVT 点数据
53	GetPosition_PVT	获取 PVT 通道轴位置

2.7 PVT 运动功能

PVT 运动功能		
21	PVTConfiguration	PVT 运动配置参数
22	PVTPoint	PVT 运动指令
23	PVTStatus	PVT 运动状态

2.8 示波器函数

示波器函数		
54	OSC_SetConfig	示波器参数设置
55	OSC_GetConfig	示波器参数获取
56	OSC_Start	示波器启动
57	OSC_Stop	示波器停止
58	OSC_GetData	示波器数据获取

2.9 示波器功能

示波器功能		
24	OSCItem	示波器通道配置
25	OSCConfiguration	示波器参数配置
26	OSCItemType	示波器通道类型
27	OSCAxisInfo	示波器轴信息类型

第 3 章　NeoMove 函数返回代码

代码	定义	错误描述
0	OK	函数执行 OK
-901	ERROR_NOTOPEN	控制器未打开
-902	ERROR_ALREADYOPEN	控制器已经打开
-903	ERROR_LIBNOTOPEN	运动库未连接
-904	ERROR_HOMETYPEERROR	回零模式错误
-905	ERROR_MOTIONLIB_TIMEOUT	运动库连接超时
-906	ERROR_NOTSUPPORT	功能不支持
-999	ERROR_WRONGCOMMAND	指令错误
-1000	ERROR_TIMEOUT	指令应答超时
-1002	ERROR_DATACOUNT	指令交互数量有限
-1003	ERROR_NODATA	指令交互无数据
-1004	ERROR_PLANFAIL	速度规划错误
-1005	ERROR_GROUPENABLE	Group 已启用
-1006	ERROR_GROUPDISABLE	Group 未启用
-1007	ERROR_AXISINDEX	轴号错误
-1008	ERROR_AXISBUSY	轴正忙
-1009	ERROR_VELOCITYTYPE	速度模式错误
-1010	ERROR_CHANNELNUMBER	通道号错误

-1011	ERROR_GANTRYSLAVEOVER	龙门从轴数量满
-1012	ERROR_GROUPBUSY	Group 忙
-1013	ERROR_MOVEMODE	运动模式错误
-1014	ERROR_OUTOFRANGE	超出范围
-1015	ERROR_MOVEDIR	运动方向错误
-1	ERROR_ERT_FUN	不支持的函数
-2	ERROR_ERT_CREATE_SH	创建共享内存错误
-3	ERROR_ERT_OPEN_SH	打开共享内存错误
-4	ERROR_ERT_NULL	参数为空
-5	ERROR_ERT_PARA	参数越界
-6	ERROR_ERT_TIMER	定时器错误
1	ERROR_LOAD	实施服务进程启动失败
2	ERROR_SYS	未和控制台连接成功
3	ERROR_RUNTIME	实时服务共享内存打开失败
4	ERROR_CONSOLE	和控制台连接失败
5	ERROR_CRC	文件校验出错
6	ERROR_GW_TIMOUT	网关超时，已废弃
7	ERROR_TEXT	多语言文件加载错误
8	ERROR_LIC	授权权限不够
9	ERROR_CHN	字符串中存在中文字符
10	ERROR_RTOS	实时系统没有启动
11	ERROR_SYS_ALIGN	实时对齐错误

12	ERROR_EXIT	控制台已经退出或者重启，重新连接
13	ERROR_NODE_ENA	主站节点未启用
14	ERROR_NOTIME	加载实施程序失败
15	ERROR_NODE_BCD	指定节点未运行
16	ERROR_PARA	输入参数非法
17	ERROR_FUN	不支持的功能
18	ERROR_DYNC_MALLOC	内存分配失败
19	ERROR_COLD_ERR	系统冷复位后需重启应用程序
20	ERROR_INVALID_DOUBLE	非法的浮点数
21	ERROR_API_VER	使用了不同版本的 API 库
100	ERROR_EBUS	总线未初始化完成
101	ERROR_SDO_OV	SDO 列表溢出
102	ERROR_SDO_U	SDO 站号非法
103	ERROR_SDO_OD	SDO 对象字典不存在或者不可读写
200	ERROR_BUSY	轴正在使用中
201	ERROR_READY	驱动器未使能
202	ERROR_LIMIT_H	已经到硬限位
203	ERROR_LIMIT_S	已经到软限位
204	ERROR_AXIS_INDEX	轴序号不存在
205	ERROR_ADVANCE	已废弃
206	ERROR_MODE_MISMATCH	运动模式不匹配
207	ERROR_CIA_CSP	此功能需要轴切换成 CSP 模式

300	ERROR_CONTI_DATA	连续运动缓冲区内无数据
301	ERROR_CONTI_OV	单轴连续运动缓冲区溢出
302	ERROR_FOLLOW_MASTER	主轴模式已启用，禁用此功能
303	ERROR_CAM_MPOS	凸轮主轴位置要求单调增加
304	ERROR_NODE_UNI	联动轴不在同一主站内
400	ERROR_CRD_INIT	坐标系已初始化
401	ERROR_CRD_DEINIT	坐标系未初始化
402	ERROR_CRD_REPEAT	坐标系轴序号重复
403	ERROR_CRD_ACTIVE	坐标系运行中
404	ERROR_CRD_FULL	坐标系缓冲区已满，容量 4096
405	ERROR_CRD_AXIS	坐标系内的轴有报警
406	ERROR_CRD_WARN	坐标系有报警
407	ERROR_CRD_SINGLE	坐标系内的轴处于单轴运行模式
408	ERROR_CRD_CSP	坐标系绑定的轴需要在 CSP 模式下
409	ERROR_CRD_REPEAT2	坐标系外轴和内部轴重合
410	ERROR_CRD_PAUSE	坐标系暂停中
411	ERROR_CRD_INC	有相对指令不支持暂停模式
412	ERROR_CRD_STOPPING	坐标系正在减速停止
413	ERROR_CRD_GANTRY	龙门轴不能加在轴列表中
500	ERROR_PITCH_ENA	在螺距补偿禁用状态下更新配置
501	ERROR_PITCH_REPEAT	螺距补偿和 2D 补偿不能同时启用
502	ERROR_PITCH_DATA	补偿值大于间距的 20%

503	ERROR_PITCH_DRV	禁止在伺服 Ready 下启用/禁用机械补偿
504	ERROR_CONPENSATION	补偿功能已启用，禁用此功能
600	ERROR_QUENE_OV	位置比较器缓冲区队列已满，在空闲数量大于 0 时写入比较数据
601	ERROR_QUENE_FIX	固定模式的位置比较器，开始比较后不能再写入数据
700	ERROR_ELEM_TYPE	元件类型错误
701	ERROR_ELEM_INDEX	元件序号错误
702	ERROR_ELEM_MEN	元件内存错误
750	ERROR_UMOTION_OV	自定义算法缓冲区溢出

第 4 章　E2-M300 结构体和枚举

4.1　系统和初始化

4.1.1　Version

结构体，用于获取版本信息

string motionLibVersion	运动库版本号
string coreVersion	核心库版本号
string controllerVersion	控制器版本号
string dllVersion	dll 版本号

4.1.2　NM_BusInfo

结构体，用于获取总线状态信息

uint running	总线状态，0：初始化中，1：完全运行中
int min_payload	系统最小负载，单位 us
int max_payload	系统最大负载，单位 us
int cur_payload	系统当前负载，单位 us
int min_shift	最小 DC 偏移，单位 us
int max_shift	最大 DC 偏移，单位 us
int cur_shift	当前 DC 偏移，单位 us
uint config_num	配置的从站数量，包含未启用站点和移除站点
uint enable_num	配置中未移除的从站数量
uint active_num	实际检测到的从站数量
uint ecat_reset_phase	总线初始化流程状态
uint debug1	调试 1
uint dog_status	授权错误状态
uint debug2	调试 2
uint debug3	调式 3
uint lost_frames	帧丢失计数
uint lost_wkc	wkc 帧丢失计数

uint dc_cycle	同步周期，单位 us
uint offline_num	掉线的从站数量

4.1.3 NM_Slave

结构体，用于获取从站信息

int slaveID	从站 ID
SlaveType slaveType	从站类型

4.1.4 SlaveType

枚举，从站类型

UNKNOWN	未知类型
STEPMOTOR	步进电机
SERVOMOTOR	伺服电机
R2_S600	RTEX 总线步进驱动器
R2_S244	RTEX 总线 IO 模块

4.1.5 ControllerType

枚举，控制器类型

R2_M100	RTEX 总线控制器
R2_M300	RTEX 总线控制器
G2_M100	脉冲控制器
E2_M200A	EtherCAT 总线板卡
E2_M300	EtherCAT 总线控制器

4.1.6 NM_BusResetMode

NM_BusResetMode
枚举，总线重置模式

WARM	热重置
COLD	冷重置

4.1.7 NM_VelocityType

枚举，速度类型

S	S 曲线
T	梯形曲线

4.1.8 NM_MasterStatus

枚举，主站状态

STOP	停止状态
RUN	运行状态
TRANSITION	过渡状态

4.2 轴运动和运动 IO 状态

4.2.1 NM_AxisStatus

结构体，用于获取单轴轴状态

uint servoOn	伺服上电状态，1：上电，0：未上电
uint servoOffline	伺服在线状态，1：离线，0：在线
uint ampAlarm	驱动器报警，1：有报警，0：无报警
uint axisAlarm	轴报警，1：有报警，0：无报警
uint followingErrorAlarm	跟随误差报警，1：有报警，0：无报警

uint motionComplete	运动指令完成，1：完成，0：未完成
uint accFlag	加速标志，1：正在加速，0：未加速
uint inPos	轴到位，1：到位，0：运动中
uint positiveLS	正限位开关，1：触发，0：未触发
uint negativeLS	负限位开关，1：触发，0：未触发
uint positiveSoftLimit	正软限位开关，1：触发，0：未触发
uint negativeSoftLimit	负软限位开关，1：触发，0：未触发
uint homeSwitch	原点开关，1：触发，0：未触发
uint homeDone	回零完成，1：已完成，0：未完成
uint torqueLimit	力矩限制标志，1：触发，0：未触发
uint ampAlarmCode	驱动器报警代码
uint master	主轴标志，1：是主轴，0：不是主轴
uint slave	从轴标志，1：是从轴，0：不是从轴
uint moveMode	轴运动模式 0：位置模式 1：速度模式 2：龙门模式 3：插补模式 4：PVT 模式
uint gantryFollowingError	龙门跟随误差超差，1：超差，0：未超差
double posCmd	指令位置
double actualPos	实际位置
double velocityCmd	指令速度
double actualVelocity	实际速度
double torqueCmd	指令力矩
double actualTorque	实际力矩
double actualFollowingError	实际跟随误差
double syncOffset	实际同步误差

4.2.2 NM_AXISMODE

枚举，轴控制模式

TQ	EtherCAT TQ 模式
HM	EtherCAT HM 模式
CSP	EtherCAT CSP 模式

4.2.3 NM_HomeParam

结构体，回零函数参数

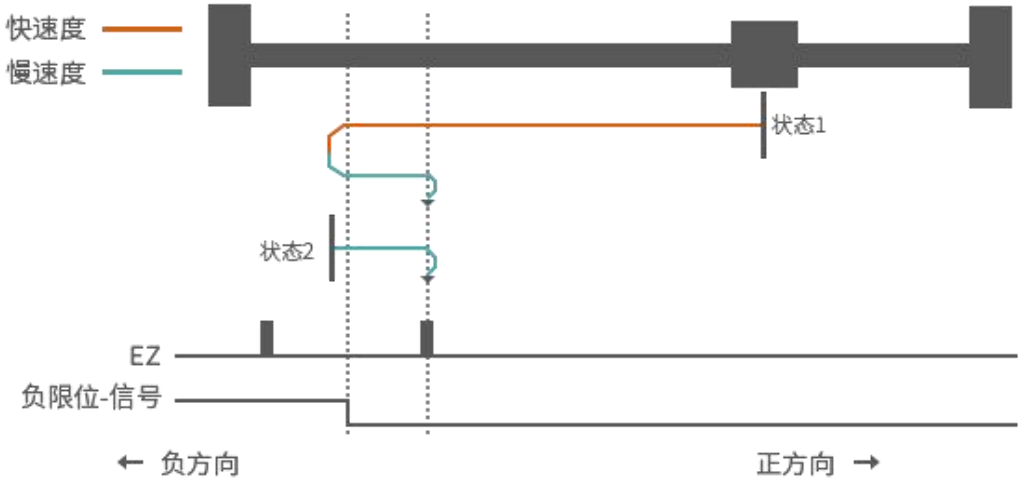
uint isGotoFinishPos	回零结束后是否运动到结束位置，1：运动，0：不运动

uint doublePass
uint homeDirection　　回零方向
uint slaveAxis　　是否是从轴，1：是从轴，0：不是从轴
NM_HOMEMODE homeType　　回零类型
uint keepPosition　　回零完成后当前位置是否清零，1：清零，0：不清零
double homingVelocitySlow　　回零慢速度
double homingVelocityFast　　回零快速度
double homingAcc　　回零加速度
double homingDec　　回零减速度
double homePosition　　原点位置
double finishPosition　　回零完成后的移动距离
double maxProbeMove　　找探针的最大移动距离
double maxSwitchMove　　找原点开关的最大移动距离

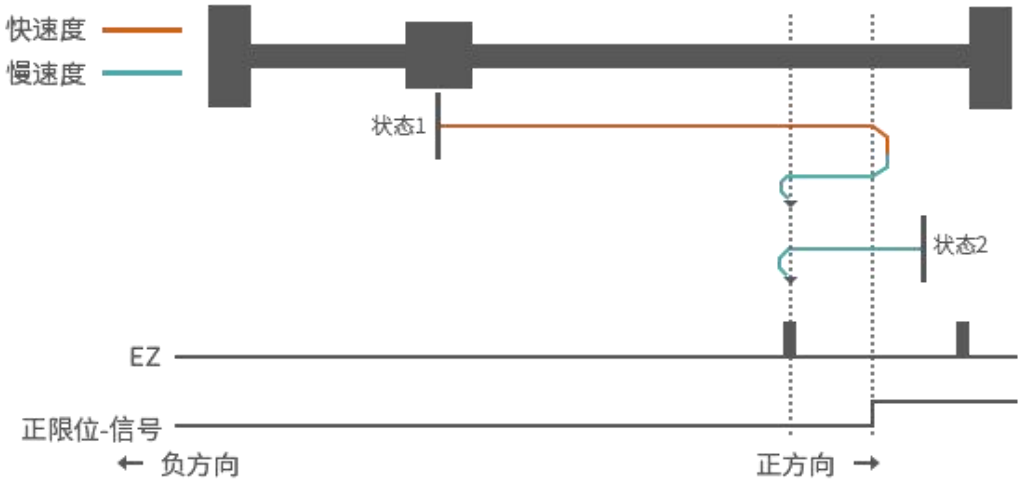
4.2.4 NM_HOMEMODE

枚举，回零模式

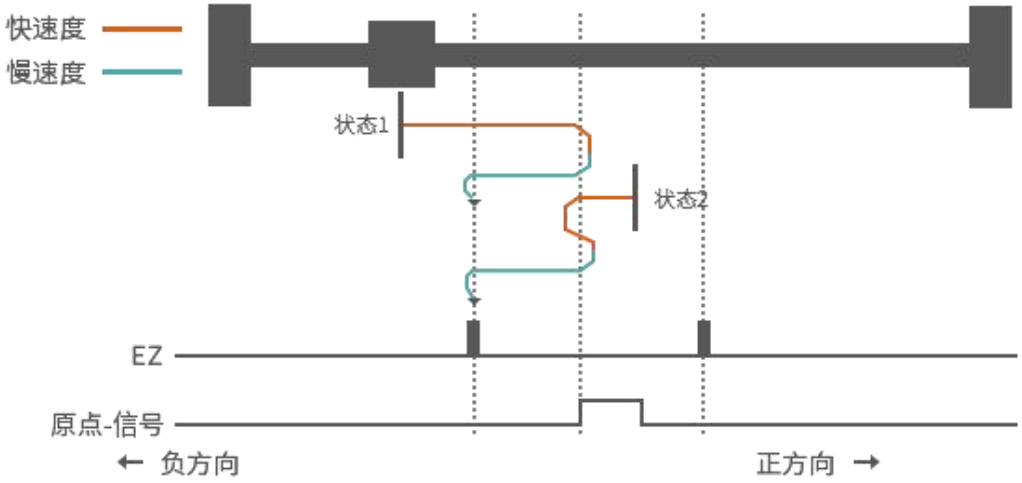
ETHERCAT_1　　EtherCAT 回零模式 1，负限位+探针



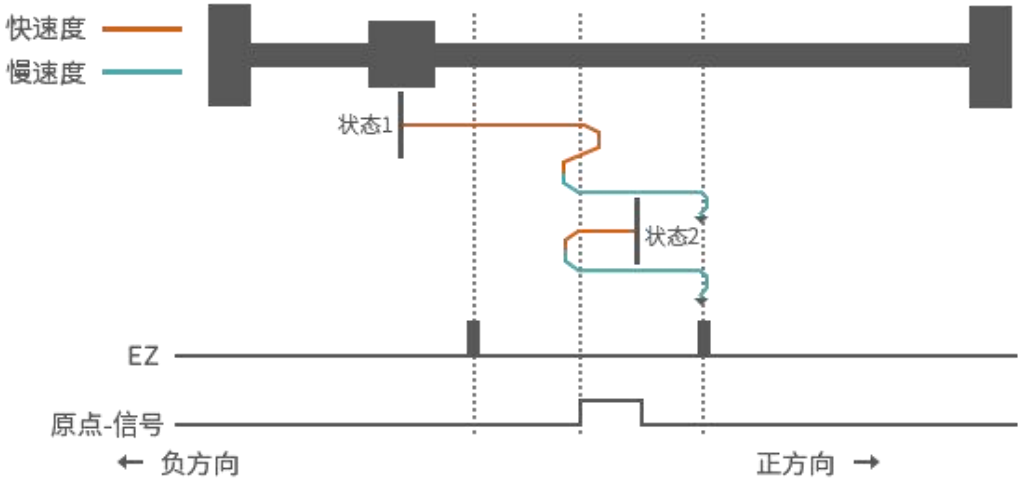
ETHERCAT_2　　EtherCAT 回零模式 2，正限位+探针



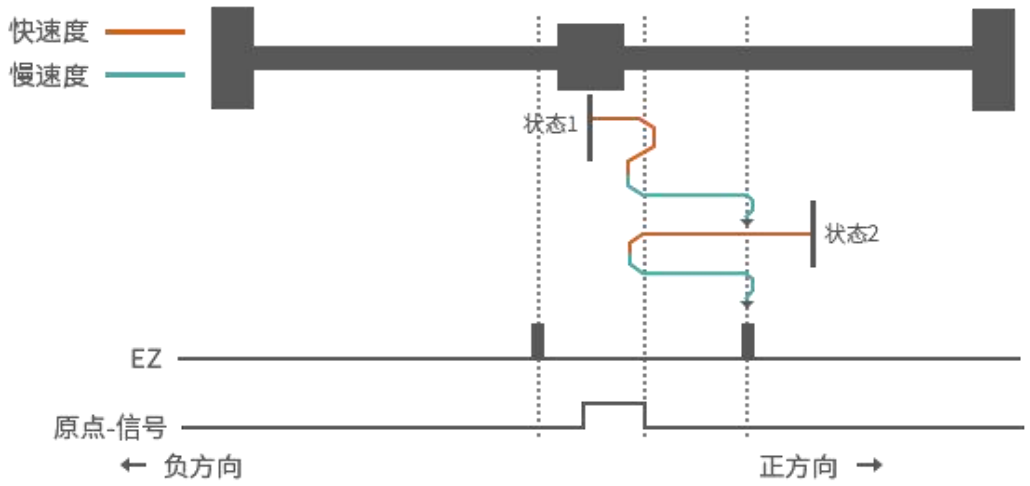
ETHERCAT_3　　EtherCAT 回零模式 3，原点+探针



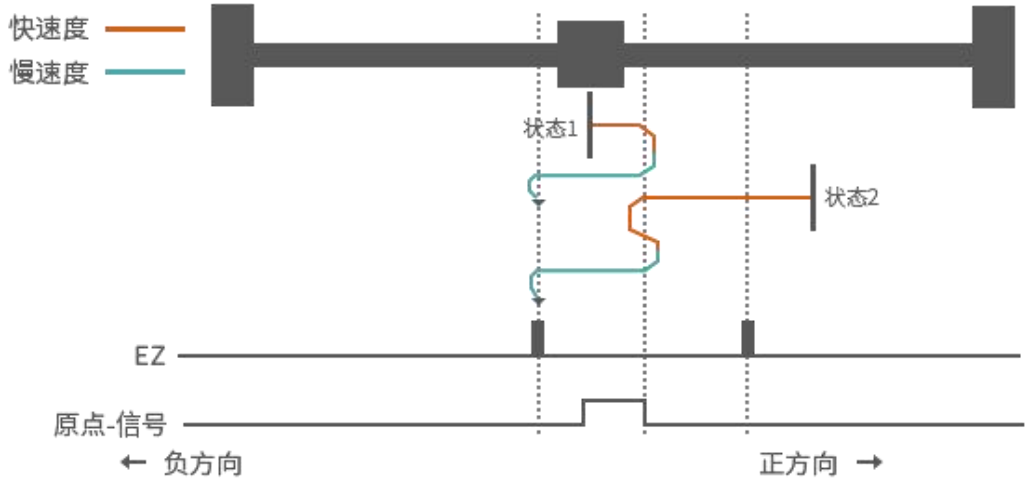
ETHERCAT_4　　EtherCAT 回零模式 4，原点+探针



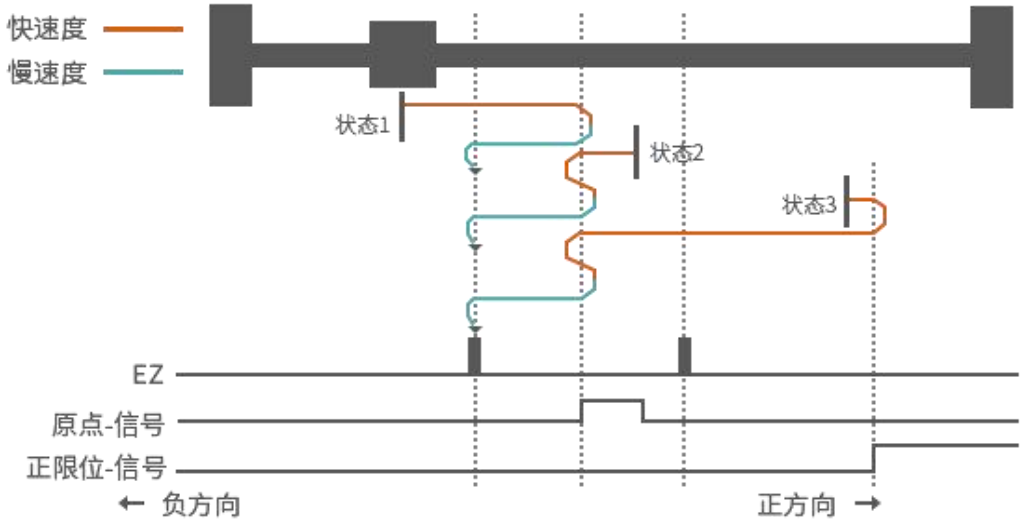
ETHERCAT_5　　EtherCAT 回零模式 5，原点+探针



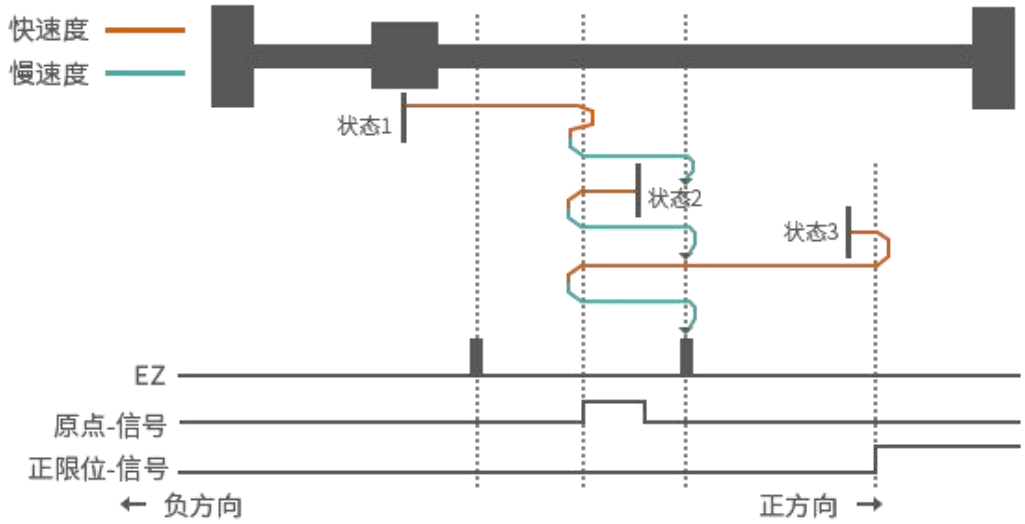
ETHERCAT_6　　EtherCAT 回零模式 6，原点+探针



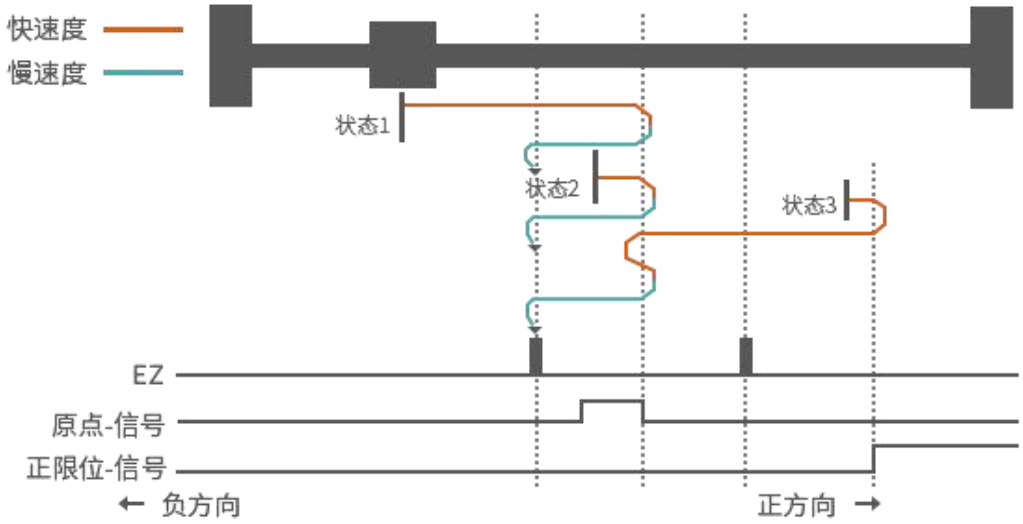
ETHERCAT_7　　EtherCAT 回零模式 7，正限位+原点+探针



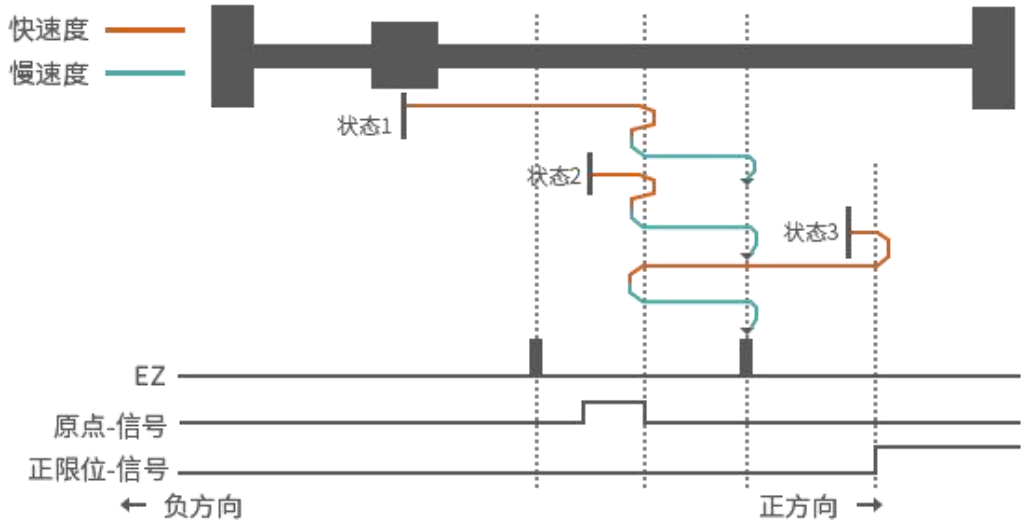
ETHERCAT_8　　EtherCAT 回零模式 8，正限位+原点+探针



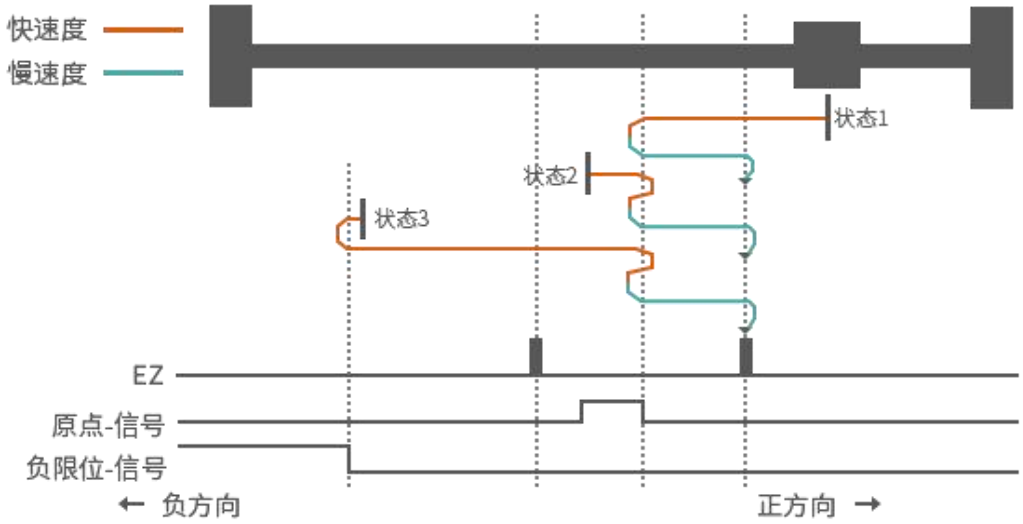
ETHERCAT_9　　EtherCAT 回零模式 9，正限位+原点+探针



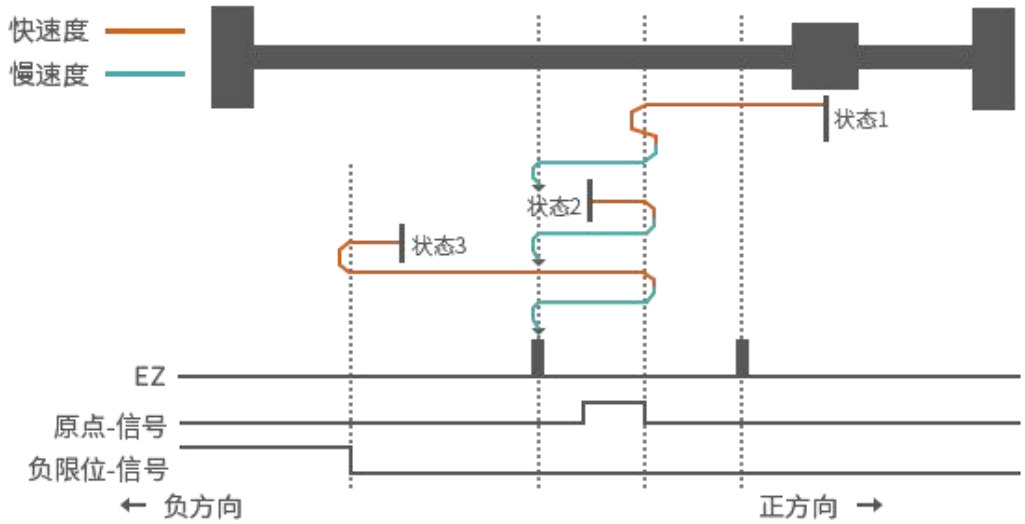
ETHERCAT_10　　EtherCAT 回零模式 10，正限位+原点+探针



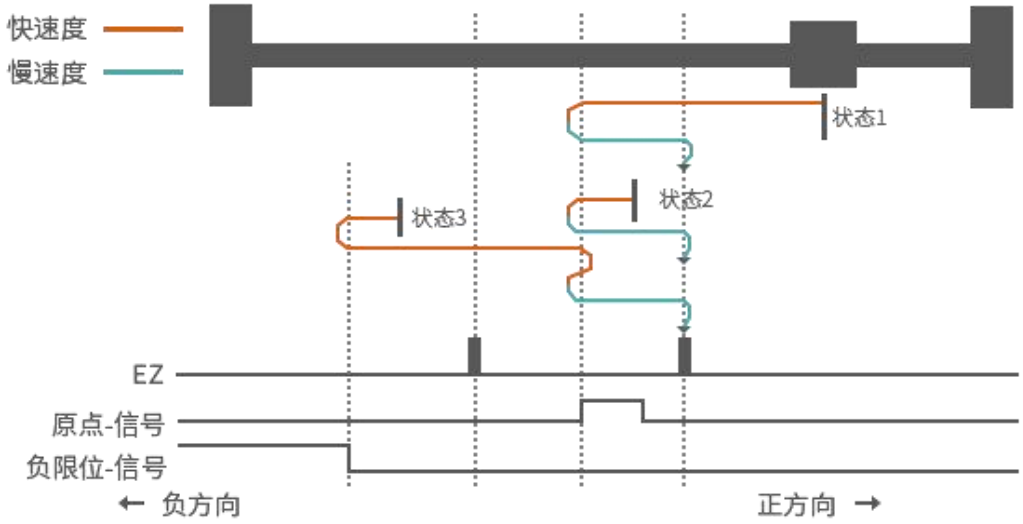
ETHERCAT_11　　EtherCAT 回零模式 11，负限位+原点+探针



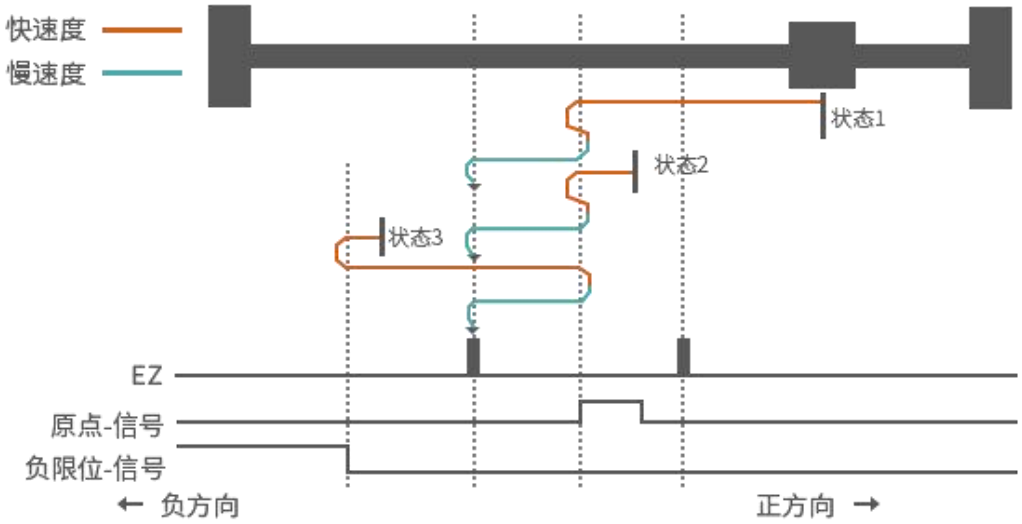
ETHERCAT_12　　EtherCAT 回零模式 12，负限位+原点+探针



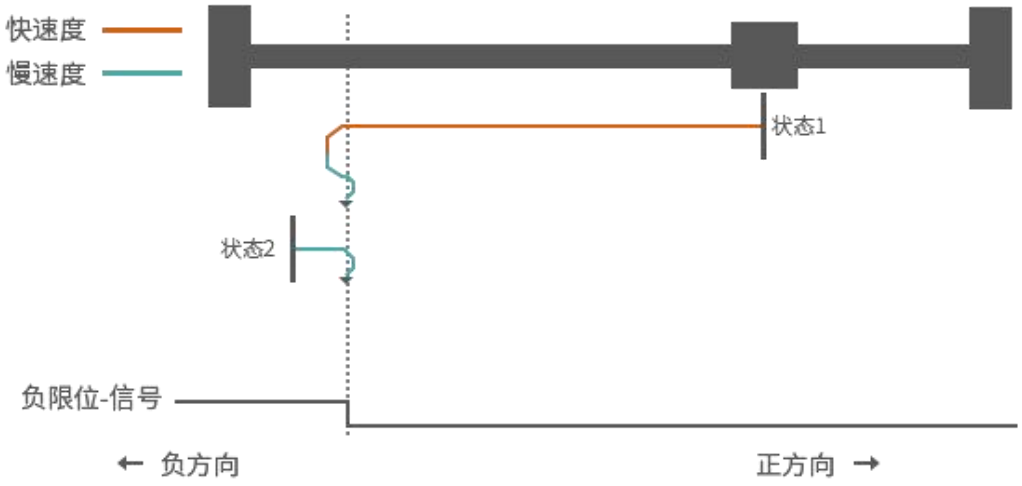
ETHERCAT_13　　EtherCAT 回零模式 13，负限位+原点+探针



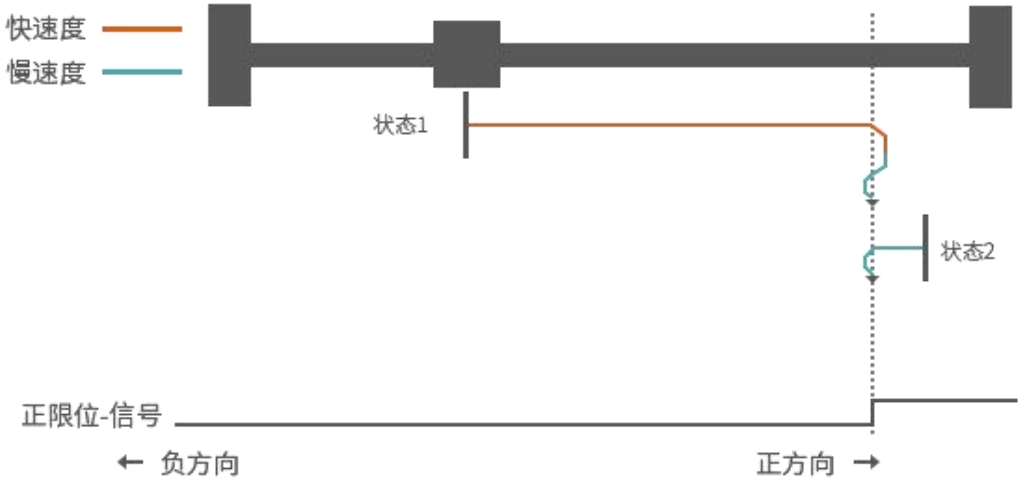
ETHERCAT_14　　EtherCAT 回零模式 14，负限位+原点+探针



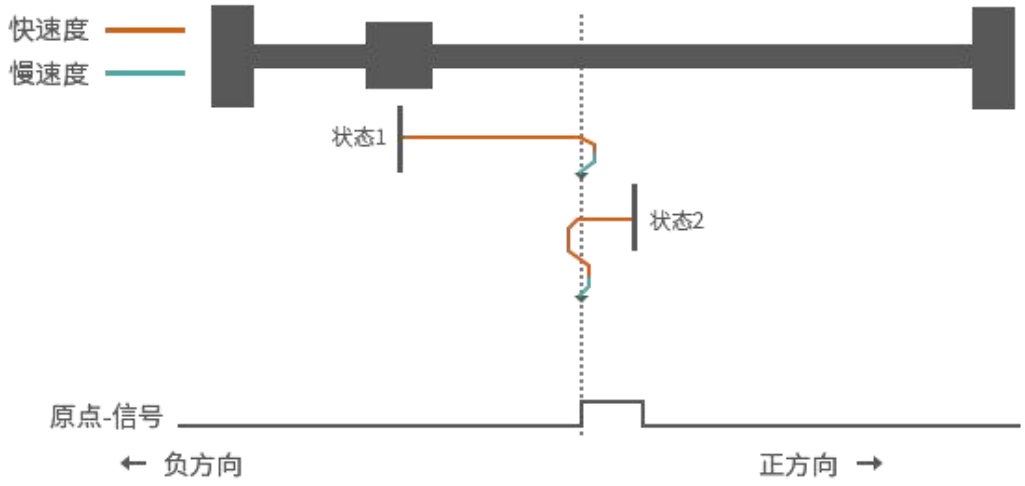
ETHERCAT_17　　EtherCAT 回零模式 17，找负限位



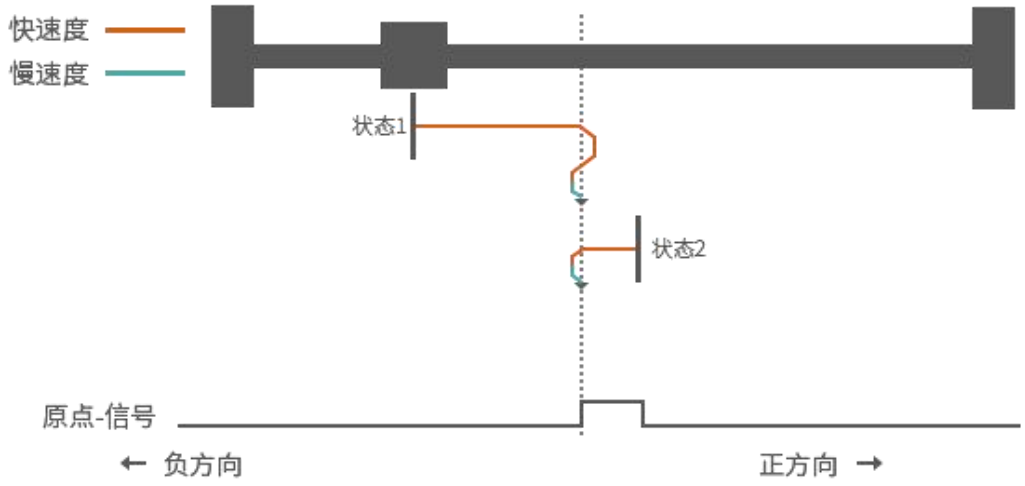
ETHERCAT_18　　EtherCAT 回零模式 18，找正限位



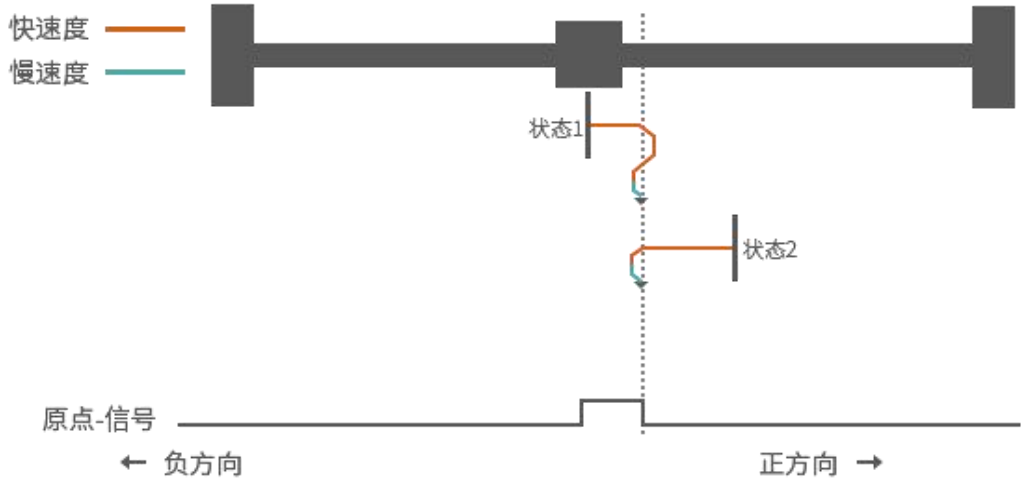
ETHERCAT_19　　EtherCAT 回零模式 19，找原点



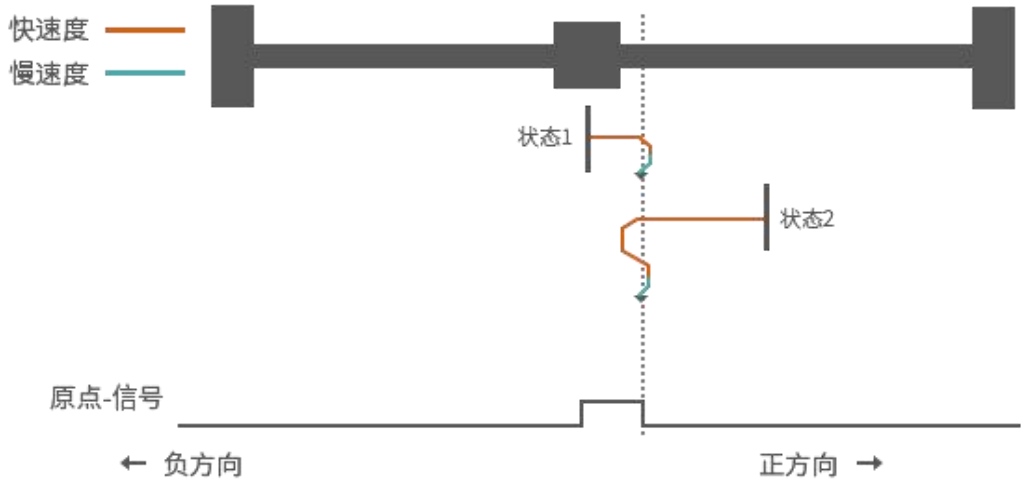
ETHERCAT_20　　EtherCAT 回零模式 20，找原点



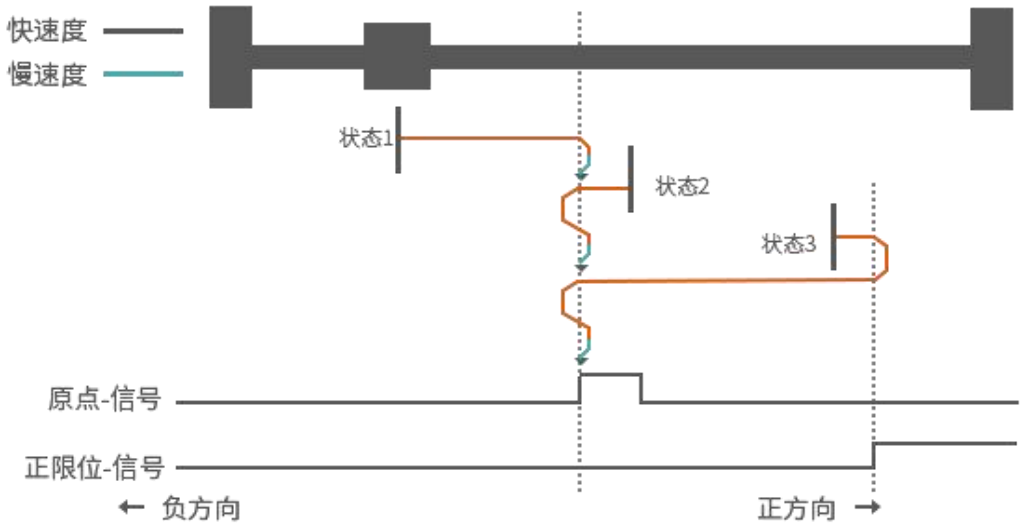
ETHERCAT_21　　EtherCAT 回零模式 21，找原点



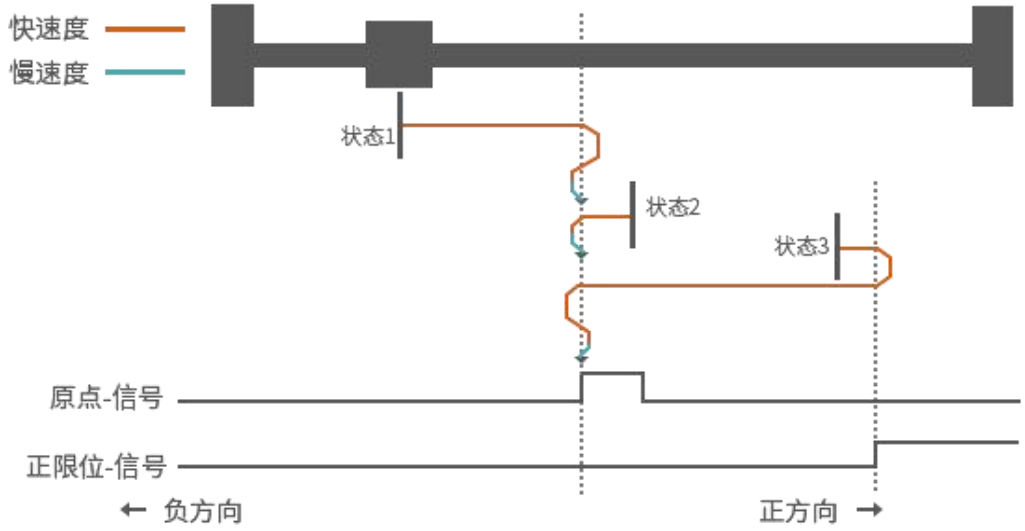
ETHERCAT_22　　EtherCAT 回零模式 22，找原点



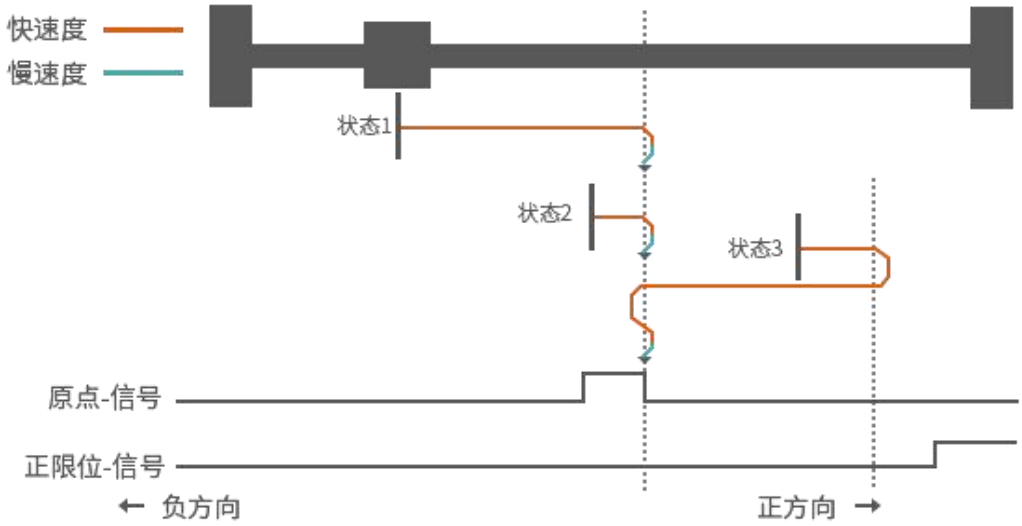
ETHERCAT_23　　EtherCAT 回零模式 23，正限位+原点



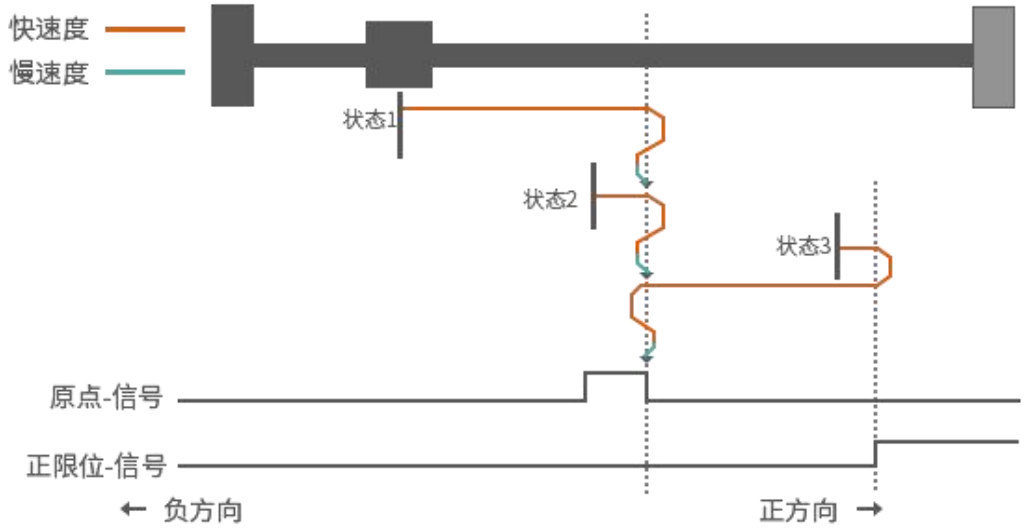
ETHERCAT_24　　EtherCAT 回零模式 24，正限位+原点



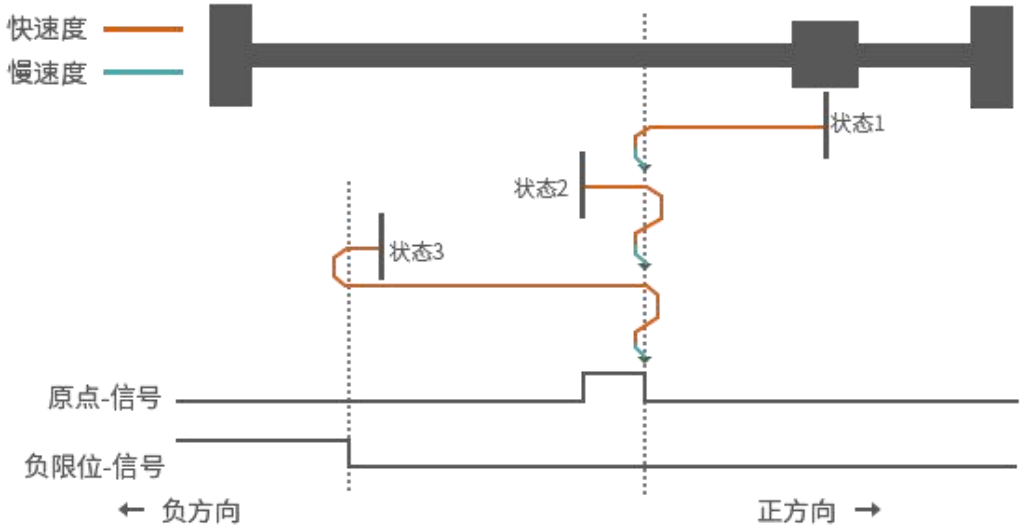
ETHERCAT_25　　EtherCAT 回零模式 25，正限位+原点



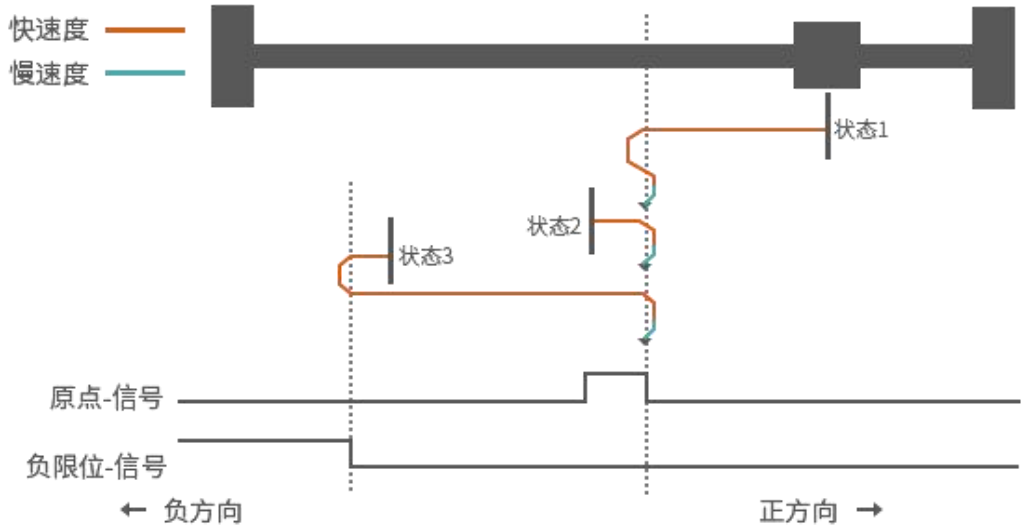
ETHERCAT_26　　EtherCAT 回零模式 26，正限位+原点



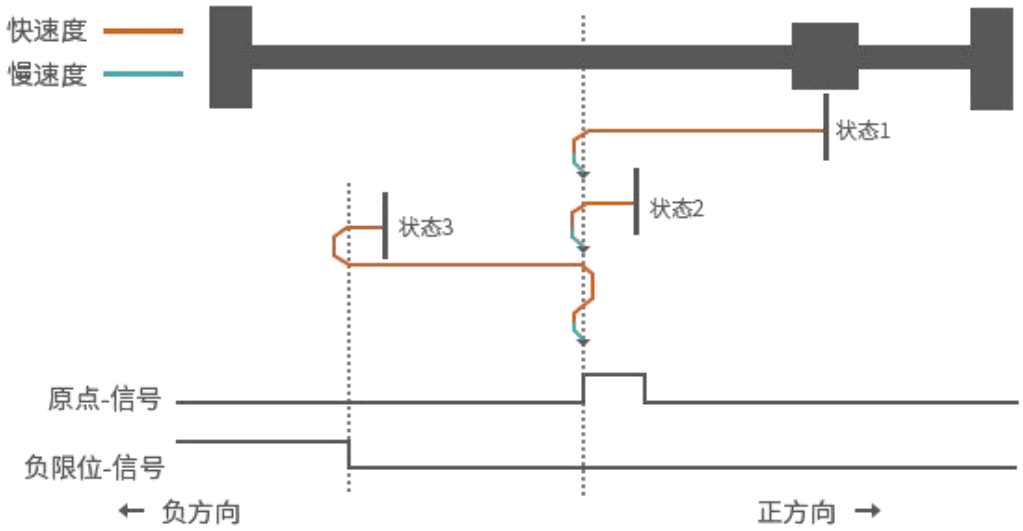
ETHERCAT_27　　EtherCAT 回零模式 27，负限位+原点



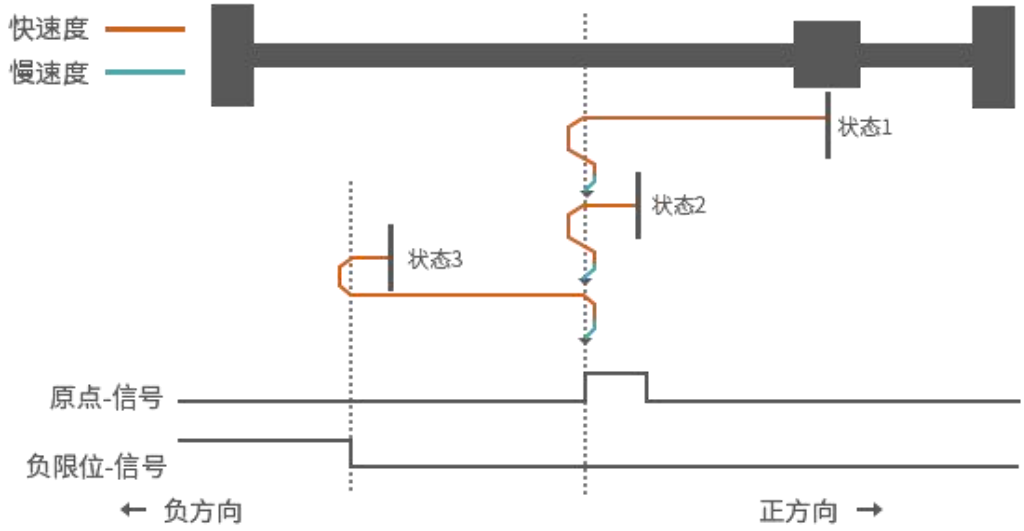
ETHERCAT_28　　EtherCAT 回零模式 28，负限位+原点



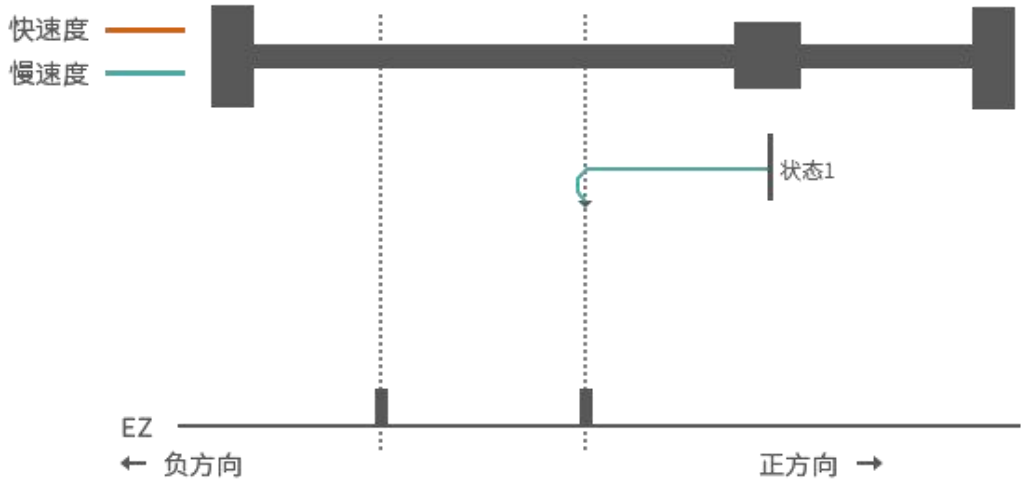
ETHERCAT_29　　EtherCAT 回零模式 29，负限位+原点



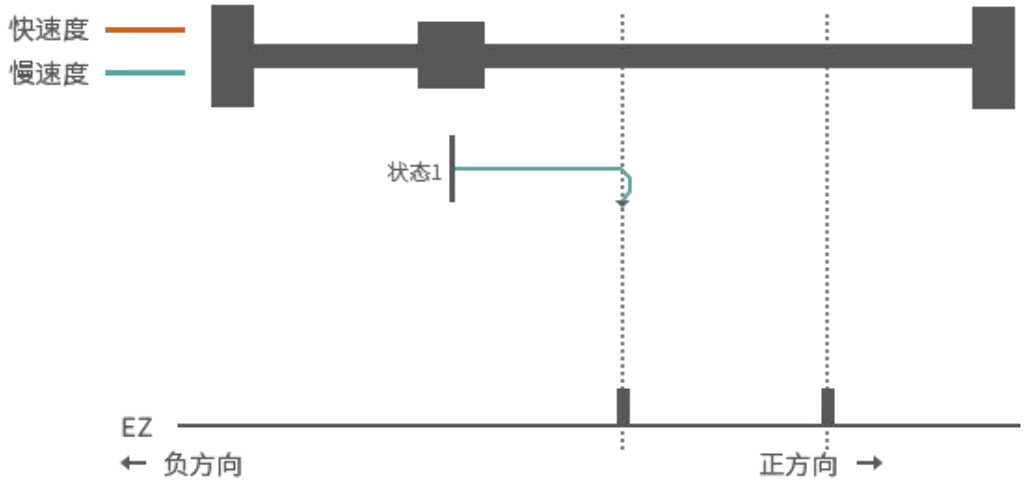
ETHERCAT_30　　EtherCAT 回零模式 30，负限位+原点



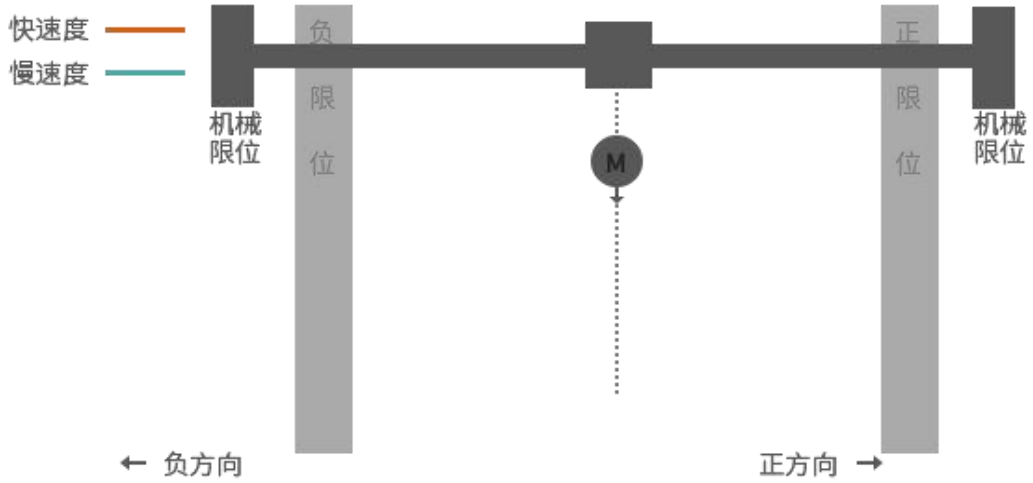
ETHERCAT_33　　EtherCAT 回零模式 33，负方向找探针



ETHERCAT_34　　EtherCAT 回零模式 34，正方向找探针



ETHERCAT_35　　EtherCAT 回零模式 35，当前位置



4.2.5 NM_PositionCommand

结构体，用于位置模式移动指令

uint axisIndex	单轴运动轴号，从 0 开始
double target	目标位置
double velocity	运动速度
double acc	运动加速度
double dec	运动减速度
double smoothTime	加/减速平滑时间，单位 s

4.2.6 NM_JogCommand

结构体，用于 Jog 模式移动指令

uint axisIndex	单轴运动轴号，从 0 开始
double target	目标位置
double velocity	运动速度
double acc	运动加速度
double dec	运动减速度
double smoothTime	加/减速平滑时间，单位 s

4.2.7 NM_TorqueCommand

结构体，用于力矩模式移动指令

uint axisIndex	单轴运动轴号，从 0 开始
int torque	力矩限制值，单位 0.1%
velocityLimit	速度限制值

4.2.8 NM_SegmentType

枚举，插补线段数据类型

SEG_STARTPOINT	起点
SEG_LINEAR	直线
SEG_FASTLINEAR	快速直线
SEG_ARCCW	顺时针圆弧
SEG_ARCCCW	逆时针圆弧

SEG_HELICAL	螺旋线
SEG_SETOUTPUT	数字 IO 输出
SEG_SLEEP	延时

4.2.9 NM_CompensationConfig

结构体，用于补偿功能参数

uint originIndex	补偿原点序号
uint count	补偿数量
double originPosition	原点坐标位置
double pitchInterval	补偿点间隔距离
double catchUpVel	调整速度
double catchUpAcc	调整加速度

4.2.10 NM_FollowingConfig

结构体，用于龙门同步功能参数

uint master	主轴轴号，从 0 开始
uint slave	从轴轴号，从 0 开始
uint beamAxis	横梁轴号，从 0 开始
uint master_GIndex_AccFeedforward	主轴加速度前馈 PDO 编号
uint master_GIndex_VelFeedforward	主轴速度前馈 PDO 编号
uint slave_GIndex_AccFeedforward	从轴加速度前馈 PDO 编号
uint slave_GIndex_VelFeedforward	从轴速度前馈 PDO 编号
double followingErrorLimit	主从轴跟随误差限制，设置 0 为无限制
double maxBeamAxisAcc	横梁轴最大加速度
double accFeedforwardRange	加速度前馈值范围
double velFeedforwardScale	速度前馈比例

4.3 前瞻插补功能

4.3.1 NM_PathIntplLookaheadCommandPoint

结构体，用于插补运动数据插入

uint setSegmentVel	是否使用段速度设定值 0：不使用，使用插补配置中的合成速度 1：使用段速度
int segmentType	数据段类型，类型参见枚举 NM_SegmentType
double setgmentType	段速度
double segmentEndVel	数据段终点速度
double[] position	轨迹位置，最多支持 8 轴
double[] auxPoint	辅助点位置，圆弧段使用，代表圆心

4.3.2 NM_PathIntplLookaheadConfiguration

结构体，用于插补运动配置参数设置

int axisCount	插补通道轴数
double compositeVel	合成速度
double compositeAcc	合成加速度
double fastVel	快速移动速度
double compositeDec	合成减速度
double smoothTime	插补加减速平滑时间
double thresholdAngle	插补路径拐角减速阈值
uint[] axisList	插补通道轴列表，最多 8 轴

4.3.3 NM_PathIntplLookaheadStatus

结构体，用于获取插补通道状态

uint state	运动状态，0：停止，1：运动
uint enable	插补通道是否启用，0：未启用，1：已启用
int bufferCount	插补缓存中的总段数
int segmentIndex	插补运动当前段号

4.4 PVT 运动功能

4.4.1 NM_PVTConfiguration

结构体，用于 PVT 配置参数设置

int axisCount	PVT 通道轴数
uint[] axisList	PVT 通道轴列表，最多 8 轴
double[] maxVel	轴最大速度
double[] maxAcc	轴最大加速度
double[] maxDec	轴最大减速度
double[] smoothTime	轴平滑时间

4.4.2 NM_PVTPoint

结构体，用于 PVT 运动数据插入

double time	段运动时间
uint[] enableVel	启用指定速度，0：不启用，1：启用
double[] position	目标位置
double[] velocity	指定速度

4.4.3 NM_PVTStatus

结构体，用于获取 PVT 通道状态

uint state	运动状态，0：停止，1：运动
uint enable	PVT 否启用，0：未启用，1：已启用
int bufferCount	PVT 缓存中的总段数
int segmentIndex	PVT 运动当前段号

4.5 示波器功能

4.5.1 NM_OSCItem

结构体，示波器通道配置

uint enable 通道启用，0：不启用，1：启用
uint axisIndex 轴号，模式为 NM_OSCItemType.AXISINFO 时有用
uint uIndex U 序号，模式为 NM_OSCItemType.PDO 时有用
uint gIndex G 序号，模式为 NM_OSCItemType.PDO 时有用
uint gDword PDO 值是否为双字节,模式为 NM_OSCItemType.PDO 时有用
int oscItemType 示波器通道类型，设定值参见 NM_OSCItemType
int axisInfoType 轴信息类型，模式为 NM_OSCItemType.AXISINFO 是有用，设定值参见 NM_OSCAxisInfo

4.5.2 NM_OSCConfiguration

结构体，用于示波器参数配置

uint depth 最大采集点数
uint interval 采集周期倍率，间隔时间=总线周期时间*采样周期倍率
NM_OSCItem item0 采样通道 0 配置
NM_OSCItem item1 采样通道 1 配置
NM_OSCItem item2 采样通道 2 配置
NM_OSCItem item3 采样通道 3 配置
NM_OSCItem item4 采样通道 4 配置
NM_OSCItem item5 采样通道 5 配置
NM_OSCItem item6 采样通道 6 配置
NM_OSCItem item7 采样通道 7 配置
NM_OSCItem item8 采样通道 8 配置
NM_OSCItem item9 采样通道 9 配置
NM_OSCItem item10 采样通道 10 配置
NM_OSCItem item11 采样通道 11 配置
NM_OSCItem item12 采样通道 12 配置
NM_OSCItem item13 采样通道 13 配置
NM_OSCItem item14 采样通道 14 配置
NM_OSCItem item15 采样通道 15 配置

4.5.3 OSCItemType

枚举，示波器通道类型

AXISINFO	轴信息
PDO	PDO 数据
UNKNOWNITEMTYPE	未知

4.5.4 OSCAxisInfo

枚举，示波器轴信息类型

CMDPOS	指令位置
ACTPOS	实际位置
CMDVEL	指令速度
ACTVEL	实际速度
UNKNOWNAXISINFO	未知

第 5 章　函数注解

5.1　系统和初始化

Open	打开控制器

描述：
此函数用来初始化并打开控制器操作。

句法：
NM_ReturnCode Open(int Controller);

参数：
int Controller：控制器号。

返回值
参见返回值列表

Close	关闭控制器

描述：
此函数用来关闭控制器操作。

句法：
NM_ReturnCode Close(int Controller);

参数：
int Controller：控制器号。

返回值
参见返回值列表

GetVersion	获取版本号

描述：
此函数用来获取控制器的版本号，通过结构体 Version，可以获取运动库版本、核心库版本、控制器版本以及 dll 版本。

句法：
```
void GetVersion(out Version version);
```

参数：
Version version：版本号结构体。

返回值
无

GetBusInfo	获取总线状态信息

描述：
此函数用来获取总线的状态信息,获取到的总线状态信息放在结构体 NM_BusInfo 当中

句法：
```
NM_ReturnCode GetBusInfo(int Controller, ref NM_Businfo busInfo);
```

参数：
int Controller：控制器号；
NM_Businfo：总线状态信息结构体。

返回值
参见返回值列表

GetSlaveInfo	获取从站信息

描述：
此函数用来获取总线从站的信息，从站信息通过 NM_Slave 类型的数组输出。

句法：
NM_ReturnCode GetSlaveInfo(int controller, out NM_Slave[] slave);

参数：
int controller：控制器号。
NM_Slave slave：从站信息结构体数组，包含从站 ID 和从站类型。

返回值
参见返回值列表

ControllerBusReset	控制器总线重置

描述：
此函数用来对控制器进行复位操作。

句法：
NM_ReturnCode ControllerBusReseat(Int Controller, NM_BusResetMode mode);

参数：
int Controller：控制器号。
NM_BusResetMode mode：重置模式，参见枚举 NM_BusResetMode

返回值
参见返回值列表

GetMasterStatus	获取主站状态

描述：
此函数用来获取主站状态的操作。

句法：
NM_ReturnCode GetMasterStatus(Int Controller, out NM_MasterStatus status);

参数：
int Controller：控制器号。

NM_MasterStatus status：总线状态结构体。

返回值
参见返回值列表

EtherCATReadPDO	EtherCAT 总线读取 PDO

描述：
此函数用来从总线中的 PDO 存储器进行读取值。

句法：
NM_ReturnCode EtherCATReadPDO(Int Controller, uint uIndex, uint gIndex, ushort[] ptr, uint len);

参数：
int Controller：控制器号。
uint uIndex：从站号，U 序号。
uint gIndex：G 序号。
ushort[] ptr：数据，数组长度为 2
uint len：数据长度。

返回值
参见返回值列表

EtherCATWritePDO	EtherCAT 总线写入 PDO

描述：
此函数用来给总线中的 PDO 存储器进行写入值。

句法：
NM_ReturnCode EtherCATWritePDO(Int Controller, uint uIndex, uint gIndex, ushort[] ptr, uint len);

参数：
int Controller：控制器号。
uint uIndex：从站号，U 序号。
uint gIndex：G 序号。
ushort[] ptr：数据，数组长度为 2。
uint len：数据长度。

返回值
参见返回值列表

EtherCATReadSDO	从从站中获取 SDO 信息

描述：
此函数可通过 SDO 方法从特定从站中获取数据。

句法：
```
NM_ReturnCode EtherCATReadSDO(Int controller, uint uIndex, uint mainIndex,
uint subIndex, uint byteNum, out uint val);
```

参数：
int controller：控制器号。
uint uIndex：从站号，U 序号。
uint mainIndex：主索引。
uint subIndex：子索引。
uint byteNum：数据长度。
out uint val：返回数值

返回值
参见返回值列表

EtherCATWriteSDO	从从站中写入 SDO 信息

描述：
此函数可通过 SDO 方法将数据设置到特定的从站。

句法：
```
NM_ReturnCode EtherCATReadSDO(Int controller, uint uIndex, uint mainIndex,
uint subIndex, uint byteNum, uint val);
```

参数：
int controller：控制器号。
uint uIndex：从站号，U 序号。
uint mainIndex：主索引。
uint subIndex：子索引。

uint byteNum：数据长度。
uint val：数值。

返回值
参见返回值列表

5.2 运动 IO 和运动状态

ServoOnOff	伺服使能/去使能

描述：
此函数用于给特定轴的伺服驱动器的使能/去使能命令。

句法：
```
NM_ReturnCode ServoOnOff(int controller, uint axisIndex, int onOff);
```

参数：
int controller：控制器号。
uint axisIndex：轴号。
int onOff：使能/去使能参数。onOff = 0：去使能；onOff = 1：上使能。

返回值
参见返回值列表

GetAxisStatus	获取轴状态

描述：
此函数可以通过结构体 axisStatus 获取轴的状态，状态值保存在结构体 NM_AxisStatus 中。

句法：
```
NM_ReturnCode GetAxisStatus(int controller, uint axisIndex, ref
NM_AxisStatus axisStatus);
```

参数：
int controller：控制器号。
uint axisIndex：轴号。
NM_AxisStatus axisStatus：返回轴状态结构体。

返回值
参见返回值列表

SetPosition	设置轴位置

描述：
此函数可用来在不动电机的情况下，将当前位置设置为任意位置。

句法：
```
NM_ReturnCode SetPosition(int controller, uint axisIndex, double position);
```

参数：
int controller：控制器号。
uint axisIndex：轴号。
double position：轴的位置。

返回值
参见返回值列表

ClearAlarm	清除轴报警

描述：
此函数可以用来清除一些不是关键性的报警信号。

句法：
```
NM_ReturnCode ClearAlarm(int controller, uint axisIndex);
```

参数：
int controller：控制器号。
uint axisIndex：轴号。

返回值
参见返回值列表

SetEncoderScale	设置脉冲比例

描述：

此函数可以用来设置主控和驱动器之间的脉冲比例。

句法：
NM_ReturnCode SetEncoderScale(int controller, uint axisIndex, int encoderScale);

参数：
int controller：控制器号。
uint axisIndex：轴号。
int encoderScale：脉冲比例。

返回值
参见返回值列表

SetSoftLimit	设置软限位

描述：
此函数可以用来在主控上设置软限位信号。

句法：
NM_ReturnCode SetSoftLimit(int controller, uint axisIndex, int enable, double positiveLimit, double negativeLimit);

参数：
int controller：控制器号。
uint axisIndex：轴号。
int enable：启用/禁用软限位；enable = 1：启用；enable = 0：禁用。
double positiveLimit：正软限位位置。
double negativeLimit：负软限位位置。

返回值
参见返回值列表

GetSoftLimit	获取软限位信号

描述：
此函数可以用来获取轴运动是否运动到软限位位置。

句法：
NM_ReturnCode GetSoftLimit(int controller, uint axisIndex, int enable, out double positiveLimit, out double negativeLimit);

参数：
int controller：控制器号。
uint axisIndex：轴号。
int enable：启用/禁用软限位；enable = 1：启用；enable = 0：禁用。
double positiveLimit：正软限位位置。
double negativeLimit：负软限位位置。

返回值
参见返回值列表

SetAxisMode	设置轴总线模式

描述：
此函数可以用来设置轴总线控制模式。

句法：
NM_ReturnCode SetAxisMode(int controller, uint axisIndex, NM_AxisMode axisMode);

参数：
int controller：控制器号。
uint axisIndex：轴号。
NM_AxisMode axisMode：轴总线模式，参见枚举 NM_AxisMode

返回值
参见返回值列表

GetAxisMode	获取轴总线模式

描述：
此函数可以用来获取当前轴总线模式，通过结构体 AxisMode，可以获取总线模式为力矩模式、回原点模式和同步周期位置模式。

句法：
NM_ReturnCode GetAxisMode(int controller, uint axisIndex, out NM_AxisMode axisMode);

参数：
int controller：控制器号。
uint axisIndex：轴号。
NM_AxisMode axisMode：轴总线模式，参见枚举 NM_AxisMode。

返回值
参见返回值列表

SetVelocityType	设置速度类型

描述：
此函数可以用来设置轴的速度类型。

句法：
```
NM_ReturnCode SetVelocityType(int controller, NM_VelocityType type);
```

参数：
int controller：控制器号。
NM_VelocityType type：速度类型，参见枚举 NM_VelocityType。

返回值
参见返回值列表

GetVelocityType	获取速度类型

描述：
此函数可以用来获取轴的速度类型。

句法：
```
NM_ReturnCode GetVelocityType(int controller, out NM_VelocityType type);
```

参数：
int controller：控制器号。
NM_VelocityType type：速度类型，参见枚举 NM_VelocityType

返回值
参见返回值列表

5.3 轴运动函数

AxisHome	轴回零

描述：
此函数用于开始轴的回零运动，回零参数通过结构体 HomeParam 写入。
在 E2-M300 控制器中，默认轴控制模式为 CSP，此时执行 AxisHome 都为上位控制器回零，若要使用驱动器回零，需要先用 SetAxisMode 函数将轴控制模式设置为 HM，然后再执行 AxisHome 函数。

句法：
```
NM_ReturnCode AxisHome(int controller, uint axisIndex, NM_HomeParam hParam);
```

参数：
int controller：控制器号。
uint axisIndex：轴号。
NM_HomeParam hParam：结构体，回零参数配置。

返回值
参见返回值列表

AxisQuickStop	轴急停

描述：
此函数可以用来立即停止设定轴的运动。

句法：
```
NM_ReturnCode AxisQuickStop(int controller, uint axisIndex);
```

参数：
int controller：控制器号。
uint axisIndex：轴号。

返回值
参见返回值列表

AxisStop	轴停止

描述：
此函数可以用来立即停止设定轴的运动，停止过程可以通过设定减速参数进行停止运动。

句法：
NM_ReturnCode AxisStop(int controller, uint axisIndex);

参数：
int controller：控制器号。
uint axisIndex：轴号。

返回值
参见返回值列表

Motion_PositionMode_Abs	绝对位置运动

描述：
此函数用来执行位置模式的单轴运动，进行一个绝对位置的运动。

句法：
NM_ReturnCode Motion_PositionMode_Abs(int controller, NM_PositionCommand pCommand);

参数：
int controller：控制器号。
NM_PositionCommand pCommand：运动指令，参见结构体 NM_PositionCommand。

返回值
参见返回值列表

Motion_PositionMode_Rel	相对位置运动

描述：
此函数用来执行位置模式的单轴运动，进行一个相对位置的运动，在结构体 PositionCommand 设置位置运动指令的参数。

句法：

NM_ReturnCode Motion_PositionMode_Rel(int controller, NM_PositionCommand pCommand);

参数：
int controller：控制器号。
NM_PositionCommand pCommand：运动指令，参见结构体 NM_PositionCommand。

返回值
参见返回值列表

Motion_JogMode	点动（Jog 模式）

描述：
此函数用于开始/停止点动，进行一段 JOG 运动，在结构体 JogCommand 设置 Jog 运动指令的参数。

句法：
NM_ReturnCode Motion_JogMode(int controller, NM_JogCommand jCommand);

参数：
int controller：控制器号。
NM_JogCommand jCommand：运动指令，参见结构体 NM_JogCommand。

返回值
参见返回值列表

Motion_TorqueMode	力矩模式运动

描述：
此函数用于开始力矩模式运动，在结构体 TorqueCommand 设置力矩模式运动指令的参数。

句法：
NM_ReturnCode Motion_TorqueMode(int controller, NM_TorqueCommand tCommand);

参数：
int controller：控制器号。
NM_TorqueCommand tCommand：运动指令，参见结构体 NM_TorqueCommand。

结构体力矩运动指令参数：
axisIndex：轴号。
torque：力矩限制值。
velocityLimit：速度限制值。

返回值
参见返回值列表

5.4 龙门功能函数

SetSyncMasterSlave	设置主从轴同步

描述：
此函数用于设定两个轴为主从轴同步的操作，通过结构体 NM_FollowingConfig 设置同步轴的相关参数。同一个主轴下，最多可以挂接 4 个从轴。

句法：
```
NM_ReturnCode SetSyncMasterSlave(int controller, NM_FollowingConfig fConfig);
```

参数：
int controller：控制器号。
NM_FollowingConfig fConfig：同步轴参数，参见结构体 NM_FollowingConfig。

返回值
参见返回值列表

SetGantryCoupling1	设置龙门解耦 1

描述：
此函数用来启用/禁用并进行龙门解耦功能，解耦 1 主要用于横梁轴运动解耦。

句法：
```
NM_ReturnCode SetGantryCoupling1(int controller, uint master, uint slave, int enable);
```

参数：
int controller：控制器号。
uint master：主轴轴号。
uint slave：从轴轴号。

int enable：启用/禁用该功能；enable = 1：启用；enable = 0：禁用。

返回值
参见返回值列表

SetGantryCoupling2	设置龙门解耦 2

描述：
此函数用来启用/禁用并进行龙门解耦功能，解耦 2 主要用于龙门轴运动解耦。

句法：
```
NM_ReturnCode SetGantryCoupling2(int controller, uint master, uint slave,
int enable);
```

参数：
int controller：控制器号。
uint master：主轴轴号。
uint slave：从轴轴号。
int enable：启用/禁用该功能；enable = 1：启用；enable = 0：禁用。

返回值
参见返回值列表

ResolveSync	主从轴同步解散

描述：
此函数用来将主从轴同步解除操作。

句法：
```
NM_ReturnCode ResolveSync(int controller, uint slave);
```

参数：
int controller：控制器号。
uint slave：从轴轴号。

返回值
参见返回值列表

SetCompensation	设置轴补偿值

描述：
此函数用于设定轴的补偿值，在一些特定应用中需要手动添加补偿值，可通过此函数操作。

句法：
```
NM_ReturnCode SetCompensation(int controller, uint axis,
NM_CompensationConfig config,double[] compensationData);
```

参数：
int controller：控制器号。
uint axis：轴号。
NM_CompensationConfig config：轴补偿配置参数，参见结构体NM_CompensationConfig
double[] compensationData：补偿数据。

返回值
参见返回值列表

GetCompensation	获取轴补偿值

描述：
此函数用于获取轴的补偿值。

句法：
```
NM_ReturnCode SetCompensation(int controller, uint axis, out
NM_CompensationConfig config,double[] compensationData);
```

参数：
int controller：控制器号。
uint axis：轴号。
NM_CompensationComfig config：轴补偿配置参数，参见结构体NM_CompensationConfig
double[] compensationData：补偿数据。

返回值
参见返回值列表

示例：

EnableCompensation	启用轴补偿

描述：
此函数用于启用/禁用轴补偿功能。

句法：
NM_ReturnCode EnableCompensation(int controller, uint axis, int enable);

参数：
int controller：控制器号。
uint axis：轴号。
int enable：启用/禁用该功能；enable = 1：启用；enable = 0：禁用。

返回值
参见返回值列表

5.5 前瞻插补功能函数

SetPathIntplLookaheadConfiguration	设置前瞻插补配置参数

描述：
此函数用于走插补运动时，设置前瞻插补配置参数。

句法：
NM_ReturnCode SetPathIntplLookaheadConfiguration(int controller, int channel, NM_PathIntplLookaheadConfiguration config);

参数：
int controller：控制器号。
int channel：通道号。
NM_PathIntplLookaheadConfiguration config：前瞻插补运动指令配置参数，参见结构体 NM_PathIntplLookaheadConfiguration

返回值
参见返回值列表

DismissPathIntplLookahead	前瞻插补组解散

描述：
此函数用于解散前瞻插补组。

句法：
NM_ReturnCode DismissPathIntplLookahead(int controller, int channel);

参数：
int controller：控制器号。
int channel：通道号。

返回值
参见返回值列表

Motion_PathIntplLookahead	启动前瞻插补运动

描述：
此函数用于启动前瞻插补运动功能。

句法：
NM_ReturnCode Motion_PathIntplLookahead(int controller, int channel);

参数：
int controller：控制器号。
int channel：通道号。

返回值
参见返回值列表

AddPathIntplLookahead	添加前瞻插补数据

描述：
此函数用于添加前瞻插补的数据。

句法：
NM_ReturnCode AddPathIntplLookahead(int controller, int channel, int pointCount, NM_PathIntplLookaheadCommandPoint[] points);

参数：
int controller：控制器号。
int channel：通道号。
int pointCount：点数。
NM_PathIntplLookaheadCommandPoint[] points：插补点数据。

返回值

参见返回值列表

Stop_PathIntpl	停止前瞻插补运动

描述：
此函数用于停止前瞻插补运动功能。

句法：
NM_ReturnCode Stop_PathIntpl(int controller, int channel);

参数：
int controller：控制器号。
int channel：通道号。

返回值
参见返回值列表

GetPathIntplLookaheadStatus	获取前瞻插补运动状态

描述：
此函数用于获取前瞻插补运动的状态。

句法：
NM_ReturnCode GetPathIntplLookaheadStatus(int controller, int channel, ref NM_PathIntplLookaheadStatus intplStatus);

参数：
int controller：控制器号。
int channel：通道号。
NM_PathIntplLookaheadStatus intplStatus：前瞻插补运动插补状态，参见结构体 NM_PathIntplLookaheadStatus

返回值
参见返回值列表

PathIntplLookaheadClearCount	清除前瞻插补缓存

描述：
此函数用于清除前瞻插补缓存。

句法：
NM_ReturnCode PathIntplLookaheadClearCount(int controller, int channel);

参数：
int controller：控制器号。
int channel：通道号。

返回值
参见返回值列表

GetPosition_PathIntplLookahead	获取插补通道轴位置

描述：
此函数用于在同一周期中，获取插补通道内的所有轴位置。

句法：
NM_ReturnCode GetPosition_PathIntplLookahead(int controller, int channel, out double[] position);

参数：
int controller：控制器号。
int channel：通道号。
double[] position：位置数据

返回值
参见返回值列表

5.6 PVT 功能函数

SetPVTConfiguration	设置 PVT 配置参数

描述：
此函数用于设置 PVT 配置参数。

句法：
NM_ReturnCode SetPVTConfiguration(int controller, int channel, NM_PVTConfiguration config);

参数：
int controller：控制器号。
int channel：通道号。
NM_PVTConfiguration config：PVT 运动配置参数，参见结构体 NM_PVTConfiguration。

返回值
参见返回值列表

DismissPVTGroup	PVT 组解散

描述：
此函数用于解散 PVT 组。

句法：
```
NM_ReturnCode DismissPVTGroup(int controller, int channel);
```

参数：
int controller：控制器号。
int channel：通道号。

返回值
参见返回值列表

示例

Motion_PVT	启动 PVT 运动

描述：
此函数用于启动 PVT 运动。

句法：
```
NM_ReturnCode Motion_PVT(int controller, int channel);
```

参数：
int controller：控制器号。
int channel：通道号。

返回值

参见返回值列表

Stop_PVT	停止 PVT 运动

描述：
此函数用于停止 PVT 运动。

句法：
NM_ReturnCode Stop_PVT(int controller, int channel);

参数：
int controller：控制器号。
int channel：通道号。

返回值
参见返回值列表

GetPVTStatus	获取 PVT 运动状态

描述：
此函数用于获取 PVT 运动状态，通过结构体 PVTStatus 来获取 PVT 运动的相关返回状态。

句法：
NM_ReturnCode GetPVTStatus(int controller, int channel, ref NM_PVTStatus pvtStatus);

参数：
int controller：控制器号。
int channel：通道号。
NM_PVTStatus pvtStatus：PVT 运动状态，参见结构体 NM_PVTStatus。

返回值
参见返回值列表

PVTClearCount	清除 PVT 缓存

描述：
此函数用于清除 PVT 缓存。

句法：
NM_ReturnCode PVTClearCount(int controller, int channel);

参数：
int controller：控制器号。
int channel：通道号。

返回值
参见返回值列表

AddPVTPoint	添加 PVT 点数据

描述：
此函数用于添加 PVT 点数据,通过结构体 PVTPoint 来添加 PVT 运动点的相关参数。

句法：
NM_ReturnCode AddPVTPoint(int controller, int channel, int pointCount, NM_PVTPiont[] points);

参数：
int controller：控制器号。
int channel：通道号。
pointCount：点数。
NM_PVTPoint[] points：PVT 点数据，参见结构体 NM_PVTPoint。

返回值
参见返回值列表

GetPosition_PVT	获取 PVT 通道轴位置

描述：
此函数用于在同一周期中，获取 PVT 通道内的所有轴位置。

句法：

```
NM_ReturnCode GetPosition_PVT(int controller, int channel, out double[] position);
```

参数：

int controller：控制器号。
int channel：通道号。
double[] position：位置数据。

返回值

参见返回值列表

5.7 示波器功能函数

OSC_SetConfig	示波器参数设置

描述：

此函数用于示波器功能的参数设置，通过结构体 OSCConfiguration 来配置示波器的相关参数。

句法：

```
NM_ReturnCode OSC_SetConfig(int controller, NM_OSCConfiguration oscConfig);
```

参数：

int controller：控制器号。
NM_OSCConfiguration oscConfig:示波器参数,参见结构体 NM_OSCConfiguration。

返回值

参见返回值列表

OSC_GetConfig	示波器参数获取

描述：

此函数用于获取示波器配置的参数。

句法：

```
NM_ReturnCode OSC_SetConfig(int controller, ref NM_OSCConfiguration
oscConfig);
```

参数：
int controller：控制器号。
NM_OSCConfiguration oscConfig:示波器参数,参见结构体 NM_OSCConfiguration。

返回值
参见返回值列表

OSC_Start	示波器启动

描述：
此函数用于启动示波器功能。

句法：

```
NM_ReturnCode OSC_Start(int controller);
```

参数：
int controller：控制器号。

返回值
参见返回值列表

OSC_Stop	示波器停止

描述：
此函数用于停止示波器功能。

句法：

```
NM_ReturnCode OSC_Stop(int controller);
```

参数：
int controller：控制器号。

返回值
参见返回值列表

OSC_GetData	示波器数据获取

描述：

此函数用于获取示波器上的数据。

句法：

```
NM_ReturnCode OSC_GetData(int controller, double[] buffer, uint
bufferSize, out uint retSize);
```

参数：

int controller：控制器号。
double[] buffer：示波器缓存数据。
uint bufferSize：缓存数据大小。
uint retSize：返回有效数据大小。

返回值

参见返回值列表

第 6 章　NeoMove 例程

6.1　系统和初始化

```
using NeoMove_Csharp;

//建立 NeoMove 类实体，构造函数参数选择 ControllerType.E2_M300
NeoMove neo = new NeoMove(ControllerType.E2_M300);
NM_ReturnCode rtn;
//打开控制器
rtn = neo.Open(0);
if ( rtn == NM_ReturnCode.OK )
{
//控制器成功连接
}
else
{
//控制器连接失败
}

//获取控制器版本
neo.GetVersion(out Version version);
//核心库版本
string coreVersion = version.coreVersion;
//运动库版本
string motionLibVersion = version.motionLibVersion;
//控制器版本
string controllerVersion = version.controllerVersion;
//dll 版本
string dllVersion = version.dllVersion;

NM_BusInfo busInfo = new NM_BusInfo();
neo.GetBusInfo(0, ref busInfo);
if ( busInfo.running == 1 )
{
//总线正常运行
}

//关闭控制器
rtn = neo.Close(0);
```

6.2 数字 IO

```
//数字 IO 写入

//主站号 0
int nodeIndex = 0;
//从站号 2
uint uIndex = 2;
//PDO 编号 100
uint gIndex = 100;
//将所有端口（8 个端口）都点亮
ushort[] ptr = new ushort[2];
ptr[0] = 0xff;
//写入 PDO 数据
neo.EtherCATWritePDO(nodeIndex, uIndex, gIndex, ptr, 1);

//数字 IO 读取

//主站号 0
int nodeIndex = 0;
//从站号 2
uint uIndex = 2;
//PDO 编号 0
uint gIndex = 0;
//PDO 数据，定义 ushort 数组，长度 2
ushort[] ptr = new ushort[2];
//读取 PDO 数据
neo.EtherCATReadPDO(nodeIndex, uIndex, gIndex, ptr, 1);
```

6.3 单轴运动

```
NM_PositionCommand pCommand = new NM_PositionCommand();
NM_ReturnCode rtn;

//轴号 0
pCommand.axisIndex = 0;
//目标位置 100
pCommand.target = 100;
//速度 50
pCommand.velocity = 50;
//加速度 2000
pCommand.acc = 2000;
//减速度 2000
pCommand.dec = 2000;
//平滑时间 0.01s
pCommand.smoothTime = 0.01;
//执行运动
rtn = neo.Motion_PositionMode_Abs(0, pCommand);
Thread.Sleep(10);
//查询运动状态，等待运动完成
while (true)
{
    NM_AxisStatus axisStatus = new NM_AxisStatus();
    rtn = neo.GetAxisStatus(0, 0, ref axisStatus);
    if(axisStatus.inPos == 1)
    {
        //运动到位
        break;
    }
}

//运动停止
rtn = neo.AxisStop(0, 0);
```

6.4 龙门设置

```
NM_FollowingConfig fConfig = new NM_FollowingConfig();
NM_ReturnCode rtn;
//主轴号 0
fConfig.master = 0;
//从轴号 1
fConfig.slave = 1;
//跟随误差 0.1mm
fConfig.followingErrorLimit = 0.1;
//横梁轴 2
fConfig.beamAxis = 2;
//反馈 PDO 编号
fConfig.master_GIndex_AccFeedforward = 112;
fConfig.master_GIndex_VelFeedforward = 113;
fConfig.slave_GIndex_AccFeedforward = 112;
fConfig.slave_GIndex_VelFeedforward = 113;
//龙门同步设置
rtn = neo.SetSyncMasterSlave(0, fConfig);
if( rtn != NM_ReturnCode.OK )
{
    //同步错误
}
//龙门解散，从轴号 1
rtn = neo.ResolveSync(0, 1);
```

6.5 插补运动

```
NM_ReturnCode rtn;
//通道内含 2 个轴，轴 0，轴 1
NM_PathIntplLookaheadConfiguration config = new
NM_PathIntplLookaheadConfiguration(2);
config.axisList[0] = 0;
config.axisList[1] = 1;
//合成速度 50，合成加速度 2000，合成减速度 2000
config.compositeAcc = 2000;
config.compositeDec = 2000;
config.compositeVel = 50;
//快速移动速度 150
config.fastVel = 150;
//平滑时间 0.02s
config.smoothTime = 0.02;
//拐角减速阈值为 135 度
config.thresholdAngle = 135;
//启用插补通道 0
rtn = neo.SetPathIntplLookaheadConfiguration(0, 0, config);

NM_PathIntplLookaheadCommandPoint[] points = new
NM_PathIntplLookaheadCommandPoint[5];
//起点（0，0）
points[0] = new
NM_PathIntplLookaheadCommandPoint(NM_SegmentType.STARTPOINT);
points[0].position[0]=0;
points[0].position[1]=0;
points[0].setSegmentVel = 0;
//直线（10，0）
points[1] = new
NM_PathIntplLookaheadCommandPoint(NM_SegmentType.LINEAR);
points[1].position[0]=10;
points[1].position[1]=0;
points[1].setSegmentVel = 0;
//直线（10，10）
points[2] = new
NM_PathIntplLookaheadCommandPoint(NM_SegmentType.LINEAR);
points[2].position[0]=10;
points[2].position[1]=10;
points[2].setSegmentVel = 0;
//直线（0，10）
points[3] = new
NM_PathIntplLookaheadCommandPoint(NM_SegmentType.LINEAR);
points[3].position[0]=0;
```

```
points[3].position[1]=10;
points[3].setSegmentVel = 0;
//直线（0，0）
points[4] = new
NM_PathIntplLookaheadCommandPoint(NM_SegmentType.LINEAR);
points[4].position[0]=0;
points[4].position[1]=0;
points[4].setSegmentVel = 0;

//添加轨迹
rtn = neo.AddPathIntplLookahead(0, 0, 5, points);

//启动运动
rtn = neo.Motion_PathIntplLookahead(0, 0);

//停止运动
rtn = neo.Stop_PathIntpl(0, 0);
```

6.6 PVT 运动

```
NM_PVTConfiguration config = new NM_PVTConfiguration(2);
//通道内含两个轴，轴 0，轴 1
config.axisList[0]=0;
config.axisList[1]=1;
//轴 0 最大速度，最大加速度，最大减速度，平滑时间设定
config.maxVel[0] = 100;
config.maxAcc[0] = 5000;
config.maxDec[0] = 5000;
config.smoothTime[0] = 0.01;
//轴 1 最大速度，最大加速度，最大减速度，平滑时间设定
config.maxVel[1] = 100;
config.maxAcc[1] = 5000;
config.maxDec[1] = 5000;
config.smoothTime[1] = 0.01;
//通道开启
rtn = neo.SetPVTConfiguration(0, 0, config);

//PVT 数据设定
NM_PVTPoint[] points = new NM_PVTPoint[4];
//第 1 点时间 0.5s，位置（10，20）
points[0] = new NM_PVTPoint(0.5);
points[0].enableVel[0] = 0;
points[0].velocity[0] = 0;
points[0].position[0] = 10;
points[0].enableVel[1] = 0;
points[0].velocity[1] = 0;
points[0].position[1] = 20;

//第 2 点时间 0.8s，位置（12，25）
points[1] = new NM_PVTPoint(0.8);
points[1].enableVel[0] = 0;
points[1].velocity[0] = 0;
points[1].position[0] = 12;
points[1].enableVel[1] = 0;
points[1].velocity[1] = 0;
points[1].position[1] = 25;

//第 3 点时间 1.2s，位置（15，29）
points[2] = new NM_PVTPoint(1.2);
points[2].enableVel[0] = 0;
points[2].velocity[0] = 0;
points[2].position[0] = 15;
points[2].enableVel[1] = 0;
```

```
points[2].velocity[1] = 0;
points[2].position[1] = 29;

//第 4 点时间 2s，位置（25，40）
points[3] = new NM_PVTPoint(2);
points[3].enableVel[0] = 0;
points[3].velocity[0] = 0;
points[3].position[0] = 25;
points[3].enableVel[1] = 0;
points[3].velocity[1] = 0;
points[3].position[1] = 40;

//载入 PVT 数据
rtn = neo.AddPVTPoint(0, 0, 4, points);

//启动 PVT 运动
rtn = neo.Motion_PVT(0, 0);

//停止 PVT 运动
rtn = neo.Stop_PVT(0, 0);
```

扫一扫 访问我们

上海山里智能科技有限公司
+86-21-61183291
上海市浦东新区建韵路 500 号 1 栋 115
www.sense-shanghai.com