

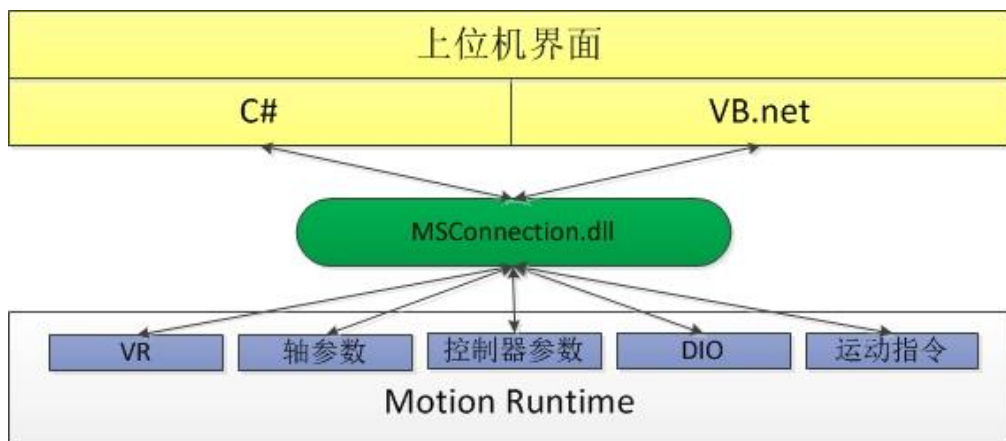
# 研华MAS控制器

## MSConnection 函数库使用说明

## 第 1 章 研华 MSConnection 介绍

### 1.1 MSConnection 概述

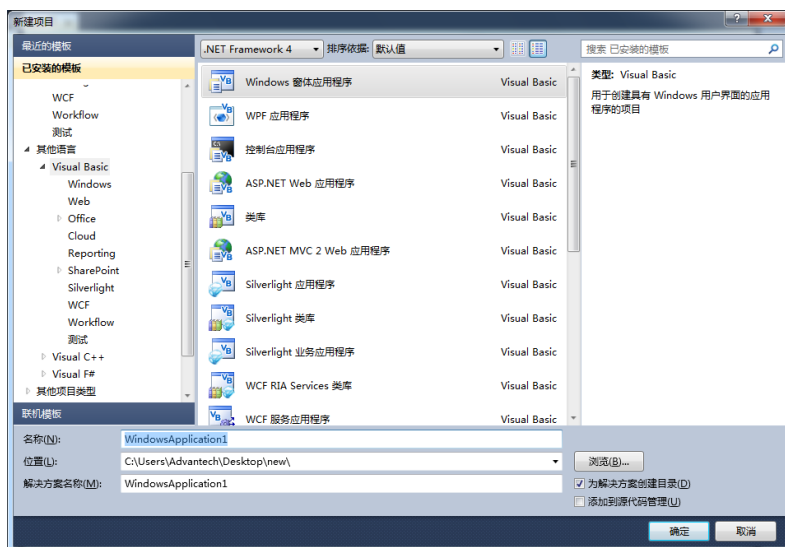
MSConnection.dll是一个动态链接库,主要是为C#或者VB.net程序与研华 Motion Runtime之间的数据交互提供接口。通过提供相应的API,上位机程序可以直接对Motion Runtime中的VR、轴参数、控制器参数和DIO进行读/写操作,如下图所示。



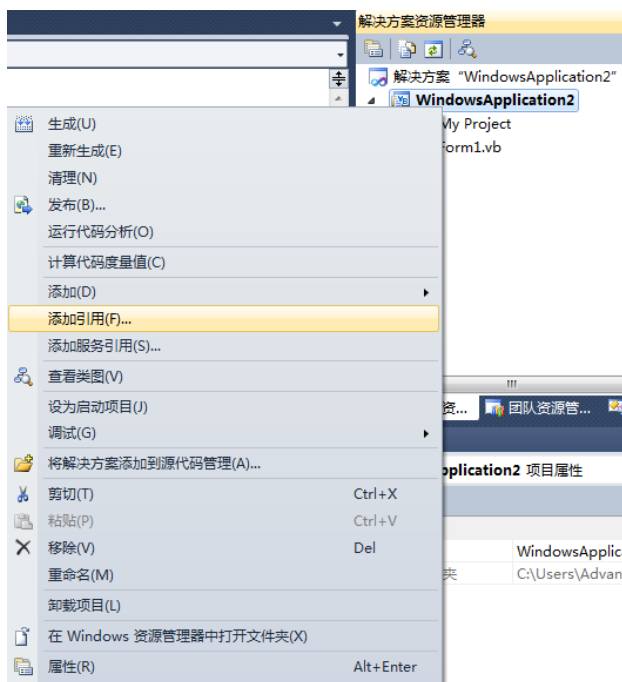
### 1.2 MSConnection 函数库的使用

#### 1.2.1 Visual Basic 2010 中的使用

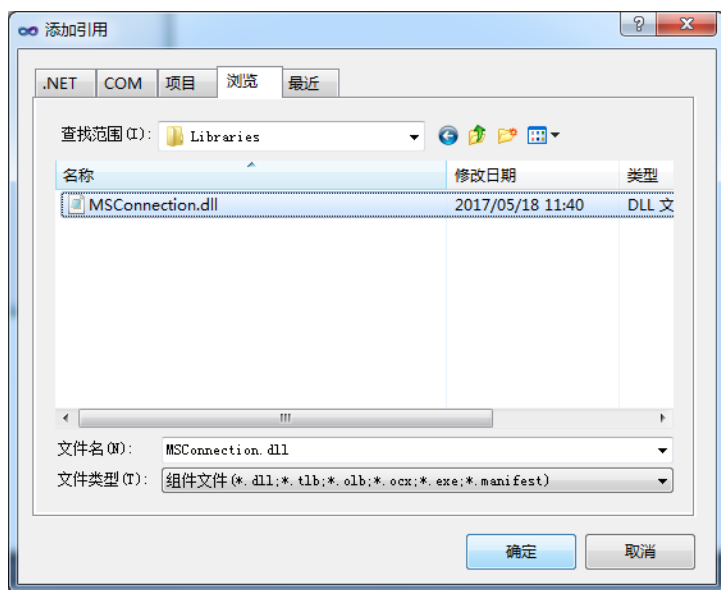
(1) 启动 Visual Studio 2010, 按照“文件”->“新建”, 选择建立 VB 工程;



(2) 单击开发环境右上角处的工程名, 右键选择[添加引用], 如下图所示;



(3) 单击[添加引用] 对话框的[浏览]，从搜索路径选择 “C:\Advantech\Libraries” 文件夹中的 “MSConnection.dll”，然后单击[OK]，如下图所示；



(4) 在编辑界面上右击，选择[查看代码]进入程序源代码编辑界面，在原始参考命名空间下添加 “Imports MSConnection”，如下图所示：

```
Imports MSConnection

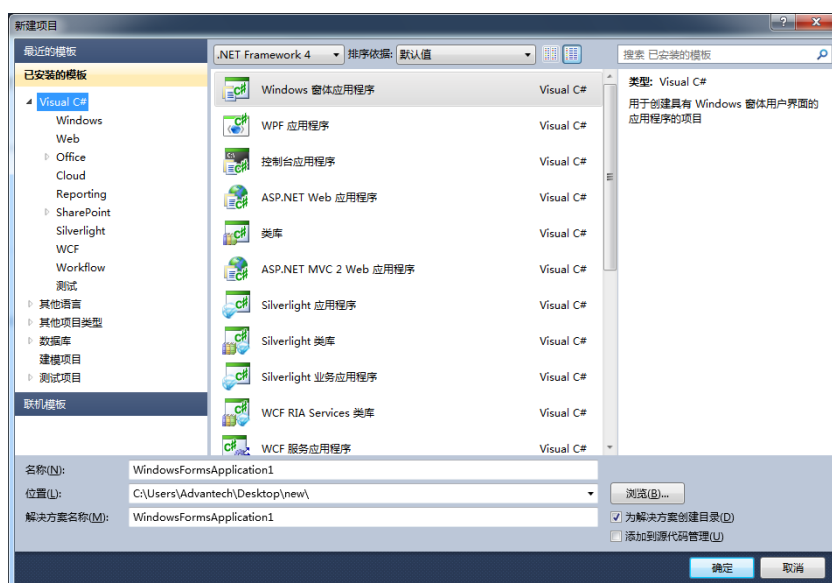
Public Class Form1

End Class
```

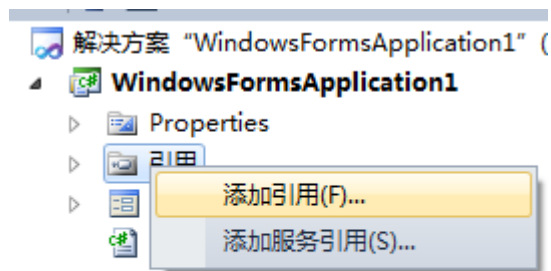
至此，用户就可以在 Visual Studio 2010 中使用 VB2010 模块调用函数库中的任何函数，开始编写应用程序。

## 1.2.2 Visual C# 2010 中的使用

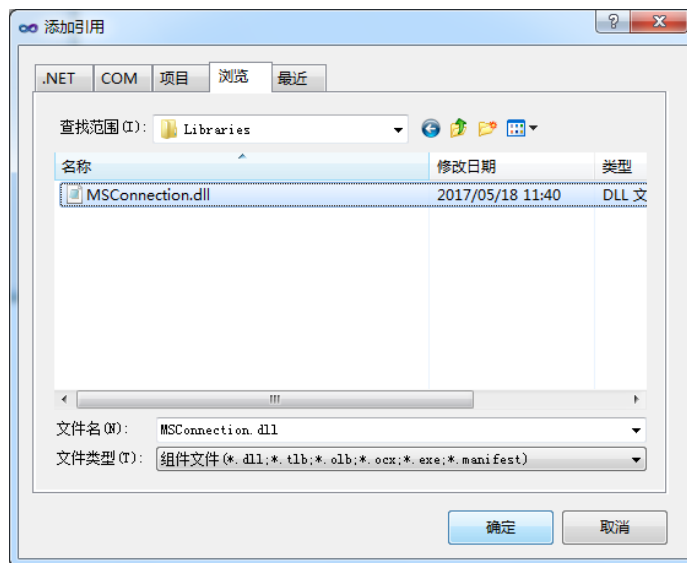
(1) 启动 Visual Studio 2010，按照“文件”->“新建”，选择建立 C# 工程；



(2) 单击开发环境右上角处的[引用]，右键选择[添加引用]，如下图所示；



(3) 单击[添加引用]对话框的[浏览]，从搜索路径选择“C:\Advantech\Libraries”文件夹中的“MSConnection.dll”，然后单击[OK]，如下图所示；



(4) 在编辑界面上右击，选择[查看代码]进入程序源代码编辑界面，在原始参考命名空间下添加“using MSConnection”，如下图所示：

```

using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using MSConnection;

namespace WindowsFormsApplication1
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            ..
        }
    }
}

```

至此，用户就可以在 Visual Studio 2010 中使用 C#模块调用函数库中的任何函数，开始编写应用程序。

## 第 2 章 MSConnection 指令详解

### MSConnection 相关注意事项

在引入 MSConnection.dll 文件后，为使用相关 API，需要首先在新建的窗体程序内创建 MSClient 客户端对象，其构造方法为：

```
public MSClient (string IP){...} ,
```

参数(IP)为 Motion Runtime 所在电脑的 IP 地址，如果是本机，地址则为“127.0.0.1”，如下所示：

```
public MSClient Wrapper = new MSClient (“127.0.0.1”);
```

如果 Motion Runtime 和窗体程序不在同一台电脑上，须确保两边的 IP 地址在同一网段。比如，Motion Runtime 端的 IP 地址为 “192.168.0.1 ”，则窗体程序端的电脑 IP 地址应在“192.168.0.x”(x 为正整数)。创建 MSClient 对象如下所示：

```
public MSClient Wrapper = new MSClient (“192.168.0.1”);
```

如果上位机程序有多个窗体程序需要和 Motion Runtime 进行数据交互，建议在主窗体程序中创建一个静态的客户端对象，然后其他子程序来调用这个静态对象，如下所示：

```
public static MSClient Wrapper = new MSClient (“127.0.0.1”);
```

**注意：**以下 API 的使用默认程序中已经创建了 MSClient 客户端对象 Wrapper。

### 2.1 系统

#### 本节指令概览

章节	API	说明
2.1.1	CheckStatus	获取与 Runtime 的连接状态
2.1.2	Dispose	释放资源，一般在退出程序时调用

#### 2.1.1 CheckStatus

**格 式：** public bool CheckStatus()  
**目 的：** 获取与 Runtime 的连接状态  
**参 数：** 无  
**返回值：** true，连接正常；false，连接断开

### 2.1.2 Dispose

**格 式：** public void Dispose()  
**目 的：** 释放资源，一般在退出程序时调用  
**参 数：** 无  
**返回值：** 无

## 2.2 VR/参数读取

### 本节指令概览

章节	API	说明
2.2.1	ReadList_I16	获取 short 类型的 VR 或参数数组
2.2.2	ReadList_U16	获取 ushort 类型的 VR 或参数数组
2.2.3	ReadList_I32	获取 int 类型的 VR 或参数数组
2.2.4	ReadList_U32	获取 uint 类型的 VR 或参数数组
2.2.5	ReadList_F32	获取 float 类型的 VR 或参数数组

#### 2.2.1 ReadList\_I16

**格式 1：** public short[] ReadList\_I16(int length, int startAddress)  
**目 的：** 获取 short 类型的 VR 或参数数组  
**参 数：**

名称	类型	说明
length	Int	VR 或参数数组的个数
startAddress	Int	VR 或参数数组的 ModBUS 首地址

**返回值：** 读取成功，返回 short 类型的数组，长度等于参数 1；  
           读取失败，返回空数组

**例 程：** 请参照 2.2.5 的例程

**格式 2：** public bool ReadList\_I16(ref short[] arr,int length, int startAddress)

**目的：** 获取 short 类型的 VR 或参数数组

**参 数：**

名称	类型	说明
arr	Short[ ]	输出参数，short 类型数组
length	Int	VR 或参数数组的个数
startAddress	Int	VR 或参数数组的 ModBUS 首地址

**返回值：** 读取成功，返回 true；读取失败，返回 false

**例 程：** 请参照 2.2.5 的例程

### 2.2.2 ReadList\_U16

**格式 1：** public ushort[] ReadList\_U16(int length, int startAddress)

**目的：** 获取 ushort 类型的 VR 或参数数组

**参 数：**

名称	类型	说明
length	Int	VR 或参数数组的个数
startAddress	Int	VR 或参数数组的 ModBUS 首地址

**返回值：** 读取成功，返回 ushort 类型的数组，长度等于参数 1；

读取失败，返回空数组

**例 程：** 请参照 2.2.5 的例程

**格式 2：** public bool ReadList\_U16(ref ushort[] arr,int length, int startAddress)

**目的：** 获取 ushort 类型的 VR 或参数数组

**参 数：**

名称	类型	说明
arr	Ushort[ ]	输出参数，ushort 类型数组
length	Int	VR 或参数数组的个数
startAddress	Int	VR 或参数数组的 ModBUS 首地址

**返回值：** 读取成功，返回 true；读取失败，返回 false



**例 程：** 请参照 2.2.5 的例程

### 2.2.3 ReadList\_I32

**格 式：** public int[] ReadList\_I32(int length, int startAddress)

**目 的：** 获取 int 类型的 VR 或参数数组

**参 数：**

名称	类型	说明
length	Int	VR 或参数数组的个数
startAddress	Int	VR 或参数数组的 ModBUS 首地址

**返回值：** 读取成功，返回 int 类型的数组，长度等于参数 1；

读取失败，返回空数组

**例 程：** 请参照 2.2.5 的例程

**格式 2：** public bool ReadList\_I32(ref int[] arr,int length, int startAddress)

**目 的：** 获取 int 类型的 VR 或参数数组

**参 数：**

名称	类型	说明
arr	Int[ ]	输出参数，int 类型数组
length	Int	VR 或参数数组的个数
startAddress	Int	VR 或参数数组的 ModBUS 首地址

**返回值：** 读取成功，返回 true；读取失败，返回 false

**例 程：** 请参照 2.2.5 的例程

### 2.2.4 ReadList\_U32

**格式 1：** public uint[] ReadList\_U32(int length, int startAddress)

**目 的：** 获取 uint 类型的 VR 或参数数组

**参 数：**

名称	类型	说明
----	----	----

length	Int	VR 或参数数组的个数
startAddress	Int	VR 或参数数组的 ModBUS 首地址

**返回值：** 读取成功，返回 uint 类型的数组，长度等于参数 1；  
读取失败，返回空数组

**例 程：** 请参照 2.2.5 的例程

**格式 2：** public bool ReadList\_U32(ref uint[] arr,int length, int startAddress)

**目 的：** 获取 uint 类型的 VR 或参数数组

**参 数：**

名称	类型	说明
arr	UInt[ ]	输出参数，uint 类型数组
length	Int	VR 或参数数组的个数
startAddress	Int	VR 或参数数组的 ModBUS 首地址

**返回值：** 读取成功，返回 true；读取失败，返回 false

**例 程：** 请参照 2.2.5 的例程

## 2.2.5 ReadList\_F32

**格式 1：** public float[] ReadList\_F32(int length, int startAddress)

**目 的：** 获取 float 类型的 VR 或参数数组

**参 数：**

名称	类型	说明
length	Int	VR 或参数数组的个数
startAddress	Int	VR 或参数数组的 ModBUS 首地址

**返回值：** 读取成功，返回 float 类型的数组，长度等于参数 1；  
读取失败，返回空数组

**格式 2：** public bool ReadList\_F32(ref float[] arr,int length, int startAddress)

**目 的：** 获取 float 类型的 VR 或参数数组

**参 数：**

名称	类型	说明
arr	Float[ ]	输出参数，float 类型数组
length	Int	VR 或参数数组的个数
startAddress	Int	VR 或参数数组的 ModBUS 首地址

**返回值：** 读取成功，返回 true；读取失败，返回 false

**例 程：**

1. 获取从 VR(2)到 VR(4)的数据，如下图所示，VR(2)、VR(3)和 VR(4)的数值分别为-100、1 和 200，VR 数组长度即为 3，开始地址为 40005：

VR表					
名称	当前值	描述	初始值	Modbus	数据类型
范围[0-1...					
VR(0)	0		0	40001	BIT_32_F...
VR(1)	0		0	40003	BIT_32_F...
VR(2)	-100		0	40005	BIT_32_F...
VR(3)	1		0	40007	BIT_32_F...
VR(4)	200		0	40009	BIT_32_F...
VR(5)	0		0	40011	BIT_32_F...

float [] a = null; //声明一个 F32 类型的数组，并赋值 null

格式 1 的用法：

a = Wrapper.ReadList\_F32(3, 40005); //将 API 返回结果赋值给 a

格式 2 的用法：

```
If ( Wrapper.ReadList_F32(ref a, 3, 40005) ) {
    //读取正确后对应的操作...
}; //将 a 作为输出参数，返回值为 true 或 false
```

则 a[0]、a[1]和 a[2]的值分别等于-100、1 和 200。

2. 获取轴 0 的初速度、运行速度、加速度和减速度(参照“3.2.4 轴参数、配置对应的 ModBUS 地址”，这四个参数的 ModBUS 地址是连续的，从 45001 开始，数值为 float 类型)，如下图所示：

轴参数		
参数	轴(0)	
UNIT_NUM	1	
UNIT_DENO...	1	
Speed		
MAXVEL	5,000,000	
MAXACC	50,000,000	
MAXDEC	50,000,000	
VH	8,000	
VL	2,000	
ACC	10,000	
DEC	10,000	
JK	0	

```
float [] a = null; //声明一个 F32 类型的数组，并赋值 null
```

格式 1 的用法：

```
a = Wrapper.ReadList_F32(4, 45001); //将 API 返回结果赋值给 a
```

格式 2 的用法：

```
If ( Wrapper.ReadList_F32(ref a, 4, 45001) ) {  
    //读取正确后对应的操作...  
}; //将 a 作为输出参数，返回值为 true 或 false
```

则 a[0]、a[1]、a[2]和 a[3]的值分别等于 2000、8000、10000 和 10000。

## 2.3 VR/参数设置

### 本节指令概览

章节	API	说明
2.3.1	WriteList_I16	设置 short 类型的 VR 或参数数组
2.3.2	WriteList_U16	设置 ushort 类型的 VR 或参数数组
2.3.3	WriteList_I32	设置 int 类型的 VR 或参数数组
2.3.4	WriteList_U32	设置 uint 类型的 VR 或参数数组
2.3.5	WriteList_F32	设置 float 类型的 VR 或参数数组

#### 2.3.1 WriteList\_I16

**格 式：** public bool WriteList\_I16(short[] List, int index, int startAddress, int length)

**目 的：** 设置 short 类型的 VR 或参数数组

**参 数：**

名称	类型	说明
List	Short[ ]	输入 short 类型数组
index	Int	数组的起始索引
startAddress	Int	VR 或参数数组的 ModBUS 首地址
length	Int	VR 或参数数组的个数

**返回值：** 设置成功，返回 true；设置失败，返回 false

**例 程：** 请参照 2.3.5 的例程

#### 2.3.2 WriteList\_U16

**格 式：** public bool WriteList\_U16(ushort[] List, int index, int startAddress, int length)

**目 的：** 设置 ushort 类型的 VR 或参数数组

**参 数：**

名称	类型	说明
----	----	----

List	Ushort[ ]	输入 ushort 类型数组
index	Int	数组的起始索引
startAddress	Int	VR 或参数数组的 ModBUS 首地址
length	Int	VR 或参数数组的个数

**返回值：** 设置成功，返回 true；设置失败，返回 false

**例 程：** 请参照 2.3.5 的例程

### 2.3.3 WriteList\_I32

**格 式：** public bool WriteList\_I32(int[] List, int index, int startAddress, int length)

**目 的：** 设置 int 类型的 VR 或参数数组

**参 数：**

名称	类型	说明
List	Int[ ]	输入 int 类型数组
index	Int	数组的起始索引
startAddress	Int	VR 或参数数组的 ModBUS 首地址
length	Int	VR 或参数数组的个数

**返回值：** 设置成功，返回 true；设置失败，返回 false

**例 程：** 请参照 2.3.5 的例程

### 2.3.4 WriteList\_U32

**格 式：** public bool WriteList\_U32(uint[] List, int index, int startAddress, int length)

**目 的：** 设置 uint 类型的 VR 或参数数组

**参 数：**

名称	类型	说明
List	UInt[ ]	输入 uint 类型数组
index	Int	数组的起始索引
startAddress	Int	VR 或参数数组的 ModBUS 首地址

length	Int	VR 或参数数组的个数
--------	-----	-------------

**返回值：** 设置成功，返回 true；设置失败，返回 false

**例 程：** 请参照 2.3.5 的例程

### 2.3.5 WriteList\_F32

**格 式：** public bool WriteList\_F32(float[] List, int index, int startAddress, int length)

**目 的：** 设置 float 类型的 VR 或参数数组

**参 数：**

名称	类型	说明
List	Float[ ]	输入 float 类型数组
index	Int	数组的起始索引
startAddress	Int	VR 或参数数组的 ModBUS 首地址
length	Int	VR 或参数数组的个数

**返回值：** 设置成功，返回 true；设置失败，返回 false

**例 程：**

```
short [] a= new short[5]{1,2,3,4,5}; //声明一个长度为 5 的 F32 类型数组
```

1. 将 a 数组的数据赋值给 VR(1)到 VR(5)：

参数 1 对应 a，参数 2 对应 a 的起始索引 0，参数 3 对应 VR(1)的 ModeBUS 地址 40003，参数 4 对应 VR 的个数 5

```
Wrapper.WriteList_F32( a, 0, 40003, 5);
```

运行结果如下图所示：

VR表					
名称	当前值	描述	初始值	Modbus	数据类型
范围[0-1...					
VR(0)	0		0	40001	BIT_32_F...
VR(1)	1		0	40003	BIT_32_F...
VR(2)	2		0	40005	BIT_32_F...
VR(3)	3		0	40007	BIT_32_F...
VR(4)	4		0	40009	BIT_32_F...
VR(5)	5		0	40011	BIT_32_F...
VR(6)	0		0	40013	BIT_32_F...
VR(7)	0		0	40015	BIT_32_F...

2. 将 a[2]的数据赋值给 VR(6)：

参数 1 对应 a,参数 2 对应 a 的起始索引 2 ,参数 3 对应 VR(6)的 ModBUS 地址 40013 , 参数 4 对应 VR 的个数 1

```
Wrapper.WriteList_F32( a, 2, 40013, 1);
```

运行结果如下图所示：

VR表						
名称	当前值	描述	初始值	Modbu	数据类型	
范围[0-1...						
VR(0)	0		0	40001	BIT_32_F...	▼
VR(1)	1		0	40003	BIT_32_F...	▼
VR(2)	2		0	40005	BIT_32_F...	▼
VR(3)	3		0	40007	BIT_32_F...	▼
VR(4)	4		0	40009	BIT_32_F...	▼
VR(5)	5		0	40011	BIT_32_F...	▼
VR(6)	3		0	40013	BIT_32_F...	▼
VR(7)	0		0	40015	BIT_32_F...	▼

short [] b= new short[4]{1999,2999,29999,29999}; //声明一个长度为 4 的 F32 类型数组

3. 将 b 数组赋值给轴 0 的初速度、运行速度、加速度和减速度：

参照 “3.2.4 轴参数、配置对应的 ModBUS 地址” , 轴 0 的初速度、运行速度、加速度和减速度的 ModBUS 地址是连续的 从 45001 开始 数值为 float 类型 , 则参数 1 对应 b ,参数 2 对应 b 的起始索引 0 ,参数 3 对应轴 0 初速度的 ModBUS 地址 45001 , 参数 4 对应参数的个数 4

```
Wrapper.WriteList_F32( b, 0, 45001, 4);
```

运行结果如下图所示：

轴参数		
参数	轴(0)	
UNIT_NUM	1	
UNIT_DENO...	1	
Speed		
MAXVEL	5,000,000	
MAXACC	50,000,000	
MAXDEC	50,000,000	
VH	2,999	
VL	1,999	
ACC	29,999	
DEC	29,999	



## 2.4 DIO 读取

### 本节指令概览

章节	API	说明
2.4.1	ReadList_DIN	获取 DIN 数据
2.4.2	ReadList_DOUT	获取 DOUT 数据

#### 2.4.1 ReadList\_DIN

**格式 1：** public byte[] ReadList\_DIN(int length)

**目 的：** 获取 DIN 数据

**参 数：**

名称	类型	说明
length	Int	DIN 数组的长度，从索引 0 开始

**返回值：** 读取成功，返回 byte 类型的数组，长度等于参数 1；

读取失败，返回空数组

**例 程：** 请参照 2.4.2 的例程

**格式 2：** public bool ReadList\_DIN(ref byte[] din,int startAddress,int length)

**目 的：** 获取 DIN 数据

**参 数：**

名称	类型	说明
din	Byte[ ]	输出参数，byte 类型数组
startAddress	Int	DIN 数组的 ModBUS 首地址
length	Int	DIN 数组的个数

**返回值：** 读取成功，返回 true；读取失败，返回 false

**例 程：** 请参照 2.4.2 的例程

#### 2.4.2 ReadList\_DOUT

**格 式：** public byte[] ReadList\_DOUT(int length)

**目 的：** 获取 DOUT 数据

**参 数：**

名称	类型	说明
length	Int	DOUT 数组的长度，从索引 0 开始

**返回值：** 读取成功，返回 byte 类型的数组，长度等于参数 1；  
读取失败，返回空数组

**格式 2：** public bool ReadList\_DOUT(ref byte[] dout,int startAddress,int length)

**目 的：** 获取 DOUT 数据

**参 数：**

名称	类型	说明
dout	Byte[ ]	输出参数，byte 类型数组
startAddress	Int	DOUT 数组的 ModBUS 首地址
length	Int	DOUT 数组的个数

**返回值：** 读取成功，返回 true；读取失败，返回 false

**例 程：**

获取 DOUT 数组从索引 0 到 4 的数据，如下图所示，DOUT(0)到 DOUT(4)的值分别为 0、1、1、0 和 1，DOUT 数组的长度即为 5，ModBUS 首地址为 1。

I/O表					
名称	当前值	描述	初始值	Modbus	数据类型
<b>Motion_PCI-1245-MA5</b>					
DOUT(0)	0	Axis-0 OUT...	0	1	BOOL
DOUT(1)	1	Axis-0 OUT...	0	2	BOOL
DOUT(2)	1	Axis-0 OUT...	0	3	BOOL
DOUT(3)	0	Axis-0 OUT...	0	4	BOOL
DOUT(4)	1	Axis-1 OUT...	0	5	BOOL
DOUT(5)	0	Axis-1 OUT...	0	6	BOOL

byte [] a = null; //声明一个 byte 类型数组，并赋值 null

格式 1 的用法：

a = Wrapper.ReadList\_DOUT(5); //将 API 返回值赋值给 a

格式 2 的用法：

If ( Wrapper. ReadList\_DOUT (ref a, 1, 5) ) {

//读取正确后对应的操作...

}; //将 a 作为输出参数，返回值为 true 或 false

则 a[0]到 a[4]的值分别等于 0、1、1、0 和 1。

## 2.5 D0 设置

### 本节指令概览

章节	API	说明
2.5.1	Write_DOUT	设置 DOUT 数据

#### 2.5.1 Write\_DOUT

**格式 1：** public bool Write\_DOUT(int address, int value)

**目的：** 设置单个 DOUT 数据

**参 数：**

名称	类型	说明
address	Int	DOUT 的 ModBUS 地址
value	Int	0 对应关闭 DOUT，1 对应打开 DOUT

**返回值：** 设置成功，返回 true；设置失败，返回 false

**例 程：**

1. 设置 DOUT(5)打开，其 ModBUS 地址为 6，则参数 1 为 6，参数 2 为 1，  
Wrapper.Write\_DOUT(6, 1);

运行结果如下图所示：

I/O表					
名称	当前值	描述	初始值	Modbus	数据类型
Motion_PCI-1245-MA					
DOUT(0)	0	Axis-0 OUT...	0	1	BOOL
DOUT(1)	0	Axis-0 OUT...	0	2	BOOL
DOUT(2)	0	Axis-0 OUT...	0	3	BOOL
DOUT(3)	0	Axis-0 OUT...	0	4	BOOL
DOUT(4)	0	Axis-1 OUT...	0	5	BOOL
DOUT(5)	1	Axis-1 OUT...	0	6	BOOL

2. 设置 DOUT(5)关闭，其 ModBUS 地址为 6，则参数 1 为 6，参数 2 为 0，  
Wrapper.Write\_DOUT(6, 0);

运行结果如下图所示：

I/O表					
名称	当前值	描述	初始值	Modbus	数据类型
<b>Motion_PCI-1245-MA:</b>					
DOUT(0)	0	Axis-0 OUT...	0	1	BOOL
DOUT(1)	0	Axis-0 OUT...	0	2	BOOL
DOUT(2)	0	Axis-0 OUT...	0	3	BOOL
DOUT(3)	0	Axis-0 OUT...	0	4	BOOL
DOUT(4)	0	Axis-1 OUT...	0	5	BOOL
DOUT(5)	0	Axis-1 OUT...	0	6	BOOL

**格式 2：** public bool Write\_DOUT(bool[] List, int index, int startAddress, int length)

**目 的：** 设置多个 DOUT 数据

**参 数：**

名称	类型	说明
List	Bool[ ]	输入 bool 类型数组
index	Int	数组的起始索引
startAddress	Int	DOUT 数组的 ModBUS 首地址
length	Int	DOUT 数组的个数

**返回值：** 设置成功，返回 true；设置失败，返回 false

**例 程：**

设置 DOUT(1)- DOUT(5)状态分别为打开，关闭，打开，关闭，打开：

1. bool[] a = new bool[] {true, false, true, false, true }; //声明并赋值一个 bool 类型数组
2. 调用 Write\_DOUT 函数 ,参数 1 为 bool 类型数组 a ,参数 2 为 a 的起始索引 0 , 参数 3 为 DOUT(1)的 ModBUS 地址 2 , 参数 4 为 DOUT 个数 5  

```
If ( Wrapper. Write_DOUT (a, 0, 2, 5) ) {
    //设置正确后对应的操作...
}; //返回值为 true 或 false
```

运行结果如下图所示：

I/O表					
名称	当前值	描述	初始值	Modbus	
Motion_MAS-328X Sin					
DOUT(0)	0	Axis-0 OUT...	0	1	BOOL
DOUT(1)	1	Axis-0 OUT...	0	2	BOOL
DOUT(2)	0	Axis-0 OUT...	0	3	BOOL
DOUT(3)	1	Axis-0 OUT...	0	4	BOOL
DOUT(4)	0	Axis-1 OUT...	0	5	BOOL
DOUT(5)	1	Axis-1 OUT...	0	6	BOOL
DOUT(6)	0	Axis-1 OUT...	0	7	BOOL

## 2.6 轴状态读取

### 本节指令概览

章节	API	说明
2.6.1	GetMIOState	获取轴 MIO 状态
2.6.2	GetAxDPOS	获取轴理论位置
2.6.3	GetAxMPOS	获取轴反馈位置
2.6.4	GetAxState	获取轴当前状态
2.6.5	GetAxCmdVel	获取轴理论速度
2.6.6	GetAxError	获取轴 LastError

#### 2.6.1 GetMIOState

格 式： public int GetMIOState (int AxisIndex, int Index)

目 的： 获取轴 MIO 状态

参 数：





















名称	类型	说明
AxisIndex	Int	所选轴号
Index	Int	MIO 相应 bit 的索引 0 ：RDY 信号 1 ：ALM 信号 2 ：LMTP 信号

- 3 : LMTN 信号
- 4 : ORG 信号
- 5 : DIR 信号
- 6 : EMG 信号
- 7 : PCS 信号
- 8 : ERC 信号
- 9 : EZ 信号
- 10 : CLR 信号
- 11 : LTC 信号
- 12 : SD 信号
- 13 : INP 信号
- 14 : SVON 信号
- 15 : ALRM 信号
- 16 : SLMTP 信号
- 17 : SLMTN 信号
- 18 : CMP 信号
- 19 : CAMDO 信号

返回值： 0 为信号 disable ; 1 为信号 enable

例 程：

获取轴 1 的 SVON 信号，如下图所示，则参数 1 为 1，参数 2 为 14，

轴状态							
轴	描述	STATE	SVON	EL+	EL-	ORG	ALM
轴(0)	轴(0)	READY					
轴(1)	轴(1)	READY					
轴(2)	轴(2)	READY					
轴(3)	轴(3)	READY					

int a = 0; //声明 int 类型的变量

a = Wrapper. GetMIOState (1, 14); //将 API 返回值赋值给 a

则 a 的值等于 1。

2.6.2 GetAxDPOS

**格 式：** public float GetAxDPOS(int AxisIndex)

**目 的：** 获取轴理论位置

**参 数：**

名称	类型	说明
AxisIndex	Int	所选轴号

**返回值：** 返回 float 类型的数据，值等于对应轴的理论位置

**例 程：**

获取轴 1 的理论位置，如下图所示，则参数为 1，

轴状态												
轴	描述	STATE	SVON	EL+	EL-	ORG	ALM	SEL+	SEL-	INP	EMG	DPOS
轴(0)	轴(0)	READY										0
轴(1)	轴(1)	READY										1,999
轴(2)	轴(2)	READY										0
轴(3)	轴(3)	READY										0

float a = Wrapper.GetAxDPOS(1); //将 API 返回值赋值给 float 类型的变量 a  
则 a 的值等于 1999。

2.6.3 GetAxMPOS

**格 式：** public float GetAxMPOS (int AxisIndex)

**目 的：** 获取轴实际位置

**参 数：**

名称	类型	说明
AxisIndex	Int	所选轴号

**返回值：** 返回 float 类型的数据，值等于对应轴的实际位置

**例 程：** 请参照 2.6.2 的例程

2.6.4 GetAxState

**格 式：** public ushort GetAxState(int AxisIndex)

**目 的：** 获取轴当前状态

**参 数：**

名称	类型	说明
AxisIndex	Int	所选轴号

**返回值：** 返回 ushort 类型的数据，值等于对应轴的当前状态，如下

- 0：轴不可用状态
- 1：Ready 状态
- 2：轴停止状态，但未 Ready
- 3：轴处于错误状态，轴被停止运动
- 4：轴正在执行回原点运动中
- 5：轴正在执行单轴点位运动中
- 6：轴正在执行单轴连续运动中
- 7：轴正在参与插补运动中或同步运动中
- 8：轴处于外部 JOG 模式中
- 9：轴处于外部 MPG 模式中

**例 程：** 请参照 2.6.2 的例程

### 2.6.5 GetAxCmdVel

**格 式：** public float GetAxCmdVel(int AxisIndex)

**目 的：** 获取轴理论速度

**参 数：**

名称	类型	说明
AxisIndex	Int	所选轴号

**返回值：** 返回 float 类型的数据，值等于对应轴的理论速度

**例 程：** 请参照 2.6.2 的例程

### 2.6.6 GetAxError

**格 式：** public uint GetAxError (int AxisIndex)

**目 的：** 获取轴 LastError

**参 数：**



名称	类型	说明
AxisIndex	Int	所选轴号

**返回值：** 返回 uint 类型的数据，值等于对应轴的错误信息

**例 程：** 请参照 2.6.2 的例程

## 2.7 控制器状态读取

### 本节指令概览

章节	API	说明
2.7.1	GetCtrlErr	获取控制器 LastError
2.7.2	GetAxCnt	获取控制器轴数量
2.7.3	GetDOUTCnt	获取控制器 DO 数量
2.7.4	GetDINCnt	获取控制器 DI 数量

#### 2.7.1 GetCtrlErr

**格 式：** public uint GetCtrlErr()

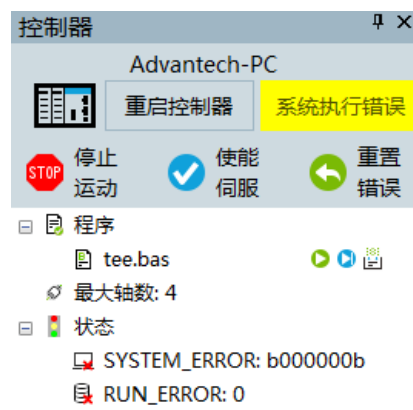
**目 的：** 获取控制器 LastError

**参 数：** 无

**返回值：** 返回 uint 类型的数据，值等于控制器最近一次错误代码

**例 程：**

获取控制器最近一次错误代码，如下图所示，



uint a = Wrapper.GetCtrlErr(); //将 API 返回值赋值给 unit 类型的 a  
则 a 的值等于 2952790027(b000000b 的十进制转换结果)。

### 2.7.2 GetAxCnt

**格 式：** public uint GetAxCnt()

**目 的：** 获取控制器轴数量

**参 数：** 无

**返回值：** 返回 uint 类型的数据，值等于控制器轴的数量

**例 程：** 请参照 2.7.1 的例程

### 2.7.3 GetDOUTCnt

**格 式：** public uint GetDOUTCnt()

**目 的：** 获取控制器 DOUT 数量

**参 数：** 无

**返回值：** 返回 uint 类型的数据，值等于控制器 DOUT 的数量

**例 程：** 请参照 2.7.1 的例程

### 2.7.4 GetDINCnt

**格 式：** public uint GetDINCnt()

**目 的：** 获取控制器 DIN 数量

**参 数：** 无

**返回值：** 返回 uint 类型的数据，值等于控制器 DIN 的数量

**例 程：** 请参照 2.7.1 的例程

## 2.8 运动指令

### 本节指令概览

章节	API	说明
2.8.1	MOVE 运动指令	操作 ModBUS 地址执行单轴相对运动
2.8.2	JOG 运动指令	操作 ModBUS 地址执行 JOG 运动

2.8.3	Home 运动指令	操作 ModBUS 地址执行 Home 运动
2.7.4	停止运动指令	操作 ModBUS 地址停止运动

### 2.8.1 MOVE 运动指令

#### 说 明：

当上位机写 1 到“指令对应的 ModBUS 地址”时,对应的轴会执行单轴相对运动,运动的距离为该轴对应的“运动距离对应的 ModBUS 地址”里的值;上位机写 0 到“指令对应的 ModBUS 地址”时,控制器不做任何处理。

说明	指令对应的 ModBUS 地址 (数据类型: Bit)	运动距离对应的 ModBUS 地址 (数据类型: F32)
轴 0 相对运动	9001	44501
轴 1 相对运动	9002	44503
.....		
轴 15 相对运动	9016	44531

#### 例 程：

目的：操作 ModBUS 地址，使轴 1 正向运动 20000 个脉冲当量

步骤：

```
float[] a = new float[1]{20000}; //声明 float 类型数组，存放运动距离
```

调用 WriteList\_F32，设置运动距离，如下所示：

```
Wrapper. WriteList_F32(a,0,44503,1);
```

调用 Write\_DOUT，指令对应的 ModBUS 地址写 1，如下所示：

```
Wrapper. Write_DOUT(9002,1);
```

结果：

轴 1 执行单轴相对运动一次，运动距离为 20000 个脉冲当量。

### 2.8.2 JOG 运动指令

#### 说 明：

当上位机写 1 到“指令对应的 ModBUS 地址”时,对应的轴会执行 JOG 运动。当上位机写 0 到“指令对应的 ModBUS 地址”时,对应的轴会停止当前的 JOG 运动。

说明	指令对应的 ModBUS 地址(数据类型 : Bit)
轴 0 正向 JOG 运动	9033
轴 1 正向 JOG 运动	9034
.....	
轴 15 正向 JOG 运动	9048
轴 0 负向 JOG 运动	9065
轴 1 负向 JOG 运动	9066
.....	
轴 15 负向 JOG 运动	9080

#### 例 程：

目的：操作 ModBUS 地址，上位机按下 JOG 按钮使轴 1 负向 JOG 运动，松开按钮停止轴 1 的 JOG 运动

步骤：

按下按钮触发 mousedown 事件，事件中调用 Write\_DOUT，指令对应的 ModBUS 地址写 1，如下所示：

```
Wrapper. Write_DOUT(9066,1); //写 1 使轴 1 开始 JOG 负向运动
```

松开按钮触发 mouseup 事件，事件中调用 Write\_DOUT，指令对应的 ModBUS 地址写 0，如下所示：

```
Wrapper. Write_DOUT(9066,0); //写 0 使轴 1 停止 JOG 运动
```

结果：按下 JOG 按钮，轴 1 开始负向 JOG 运动，松开按钮则运动停止。

### 2.8.3 Home 运动指令

#### 说 明：

当上位机写 1 到“指令对应的 ModBUS 地址”时，对应的轴会执行回原点运动。

当上位机写 0 到“指令对应的 ModBUS 地址”时，控制器不做任何处理。

说明	指令对应的 ModBUS 地址(数据类型 : Bit)
轴 0 正向回原点运动	9097
轴 1 正向回原点运动	9098

.....	
轴 15 正向回原点运动	9112
轴 0 负向回原点运动	9129
轴 1 负向回原点运动	9130
.....	
轴 15 负向回原点运动	9144

#### 例 程：

目的：操作 ModBUS 地址，上位机按下 Home 按钮使轴 1 负向 Home 运动

步骤：

按下 Home 按钮触发 click 事件，事件中调用 Write\_DOUT，指令对应的 ModBUS 地址写 1，如下所示：

```
Wrapper. Write_DOUT(9130,1); //写 1 使轴 1 开始 Home 负向运动
```

结果：按下 Home 按钮，轴 1 开始负向 Home 运动。

### 2.8.4 停止运动指令

#### 说 明：

当上位机写 1 到“指令对应的 ModBUS 地址”时，控制器会控制对应的轴停止运动。当上位机写 0 到“指令对应的 ModBUS 地址”时，控制器不做任何处理。

说明	指令对应的 ModBUS 地址(数据类型：Bit)
停止轴 0 运动	9161
停止轴 1 运动	9162
.....	
停止轴 15 运动	9176

#### 例 程：

目的：操作 ModBUS 地址，上位机按下 Stop 按钮使轴 1 停止运动

步骤：

按下 Stop 按钮触发 click 事件，事件中调用 Write\_DOUT，指令对应的 ModBUS 地址写 1，如下所示：

Wrapper. Write\_DOUT(9162,1); //写 1 使轴 1 停止运动

结果：按下 Stop 按钮，轴 1 停止运动。

## 第 3 章 附录

### 3.1 错误代码说明

#### 3.1.1 RUN\_ERROR 错误代码表

错误代码	说明
0x00000000	SUCCESS
0x80000000	InvalidDevNumber
0x80000001	DevRegDataLost
0x80000002	LoadDllFailed
0x80000003	GetProcAddressFailed
0x80000004	MemAllocateFailed
0x80000005	InvalidHandle
0x80000006	CreateFileFailed
0x80000007	OpenEventFailed
0x80000008	EventTimeOut
0x80000009	InvalidInputParam
0x8000000a	PropertyIDNotSupport
0x8000000b	PropertyIDReadOnly
0x8000000c	ConnectWinIrqFailed
0x8000000d	InvalidAxCfgVel
0x8000000e	InvalidAxCfgAcc
0x8000000f	InvalidAxCfgDec
0x80000010	InvalidAxCfgJerk
0x80000011	InvalidAxParVelLow
0x80000012	InvalidAxParVelHigh
0x80000013	InvalidAxParAcc
0x80000014	InvalidAxParDec
0x80000015	InvalidAxParJerk
0x80000016	InvalidAxPulseInMode

0x80000017	InvalidAxPulseOutMode
0x80000018	InvalidAxAlarmEn
0x80000019	InvalidAxAlarmLogic
0x8000001a	InvalidAxInPEn
0x8000001b	InvalidAxInPLogic
0x8000001c	InvalidAxHLmtEn
0x8000001d	InvalidAxHLmtLogic
0x8000001e	InvalidAxHLmtReact
0x8000001f	InvalidAxSLmtPEn
0x80000020	InvalidAxSLmtPReact
0x80000021	InvalidAxSLmtPValue
0x80000022	InvalidAxSLmtMEn
0x80000023	InvalidAxSLmtMReact
0x80000024	InvalidAxSLmtMValue
0x80000025	InvalidAxOrgLogic
0x80000026	InvalidAxOrgEnable
0x80000027	InvalidAxEzLogic
0x80000028	InvalidAxEzEnable
0x80000029	InvalidAxEzCount
0x8000002a	InvalidAxState
0x8000002b	InvalidAxInEnable
0x8000002c	InvalidAxSvOnOff
0x8000002d	InvalidAxDistance
0x8000002e	InvalidAxPosition
0x8000002f	InvalidAxHomeModeKw
0x80000030	InvalidAxCntInGp
0x80000031	AxInGpNotFound
0x80000032	AxIsInOtherGp
0x80000033	AxCannotIntoGp



0x80000034	GpInDevNotFound
0x80000035	InvalidGpCfgVel
0x80000036	InvalidGpCfgAcc
0x80000037	InvalidGpCfgDec
0x80000038	InvalidGpCfgJerk
0x80000039	InvalidGpParVelLow
0x8000003a	InvalidGpParVelHigh
0x8000003b	InvalidGpParAcc
0x8000003c	InvalidGpParDec
0x8000003d	InvalidGpParJerk
0x8000003e	JerkNotSupport
0x8000003f	ThreeAxNotSupport
0x80000040	DevIpoNotFinished
0x80000041	InvalidGpState
0x80000042	OpenFileFailed
0x80000043	InvalidPathCnt
0x80000044	InvalidPathHandle
0x80000045	InvalidPath
0x80000046	IoctlError
0x80000047	AmnetRingUsed
0x80000048	DeviceNotOpened
0x80000049	InvalidRing
0x8000004a	InvalidSlaveIP
0x8000004b	InvalidParameter
0x8000004c	InvalidGpCenterPosition
0x8000004d	InvalidGpEndPosition
0x8000004e	InvalidAddress
0x8000004f	DeviceDisconnect
0x80000050	DataOutBufExceeded

0x80000051	SlaveDeviceNotMatch
0x80000052	SlaveDeviceError
0x80000053	SlaveDeviceUnknow
0x80000054	FunctionNotSupport
0x80000055	InvalidPhysicalAxis
0x80000056	InvalidVelocity
0x80000057	InvalidAxPulseInLogic
0x80000058	InvalidAxPulseInSource
0x80000059	InvalidAxErcLogic
0x8000005a	InvalidAxErcOnTime
0x8000005b	InvalidAxErcOffTime
0x8000005c	InvalidAxErcEnableMode
0x8000005d	InvalidAxSdEnable
0x8000005e	InvalidAxSdLogic
0x8000005f	InvalidAxSdReact
0x80000060	InvalidAxSdLatch
0x80000061	InvalidAxHomeResetEnable
0x80000062	InvalidAxBacklashEnable
0x80000063	InvalidAxBacklashPulses
0x80000064	InvalidAxVibrationEnable
0x80000065	InvalidAxVibrationRevTime
0x80000066	InvalidAxVibrationFwdTime
0x80000067	InvalidAxAlarmReact
0x80000068	InvalidAxLatchLogic
0x80000069	InvalidFwMemoryMode
0x8000006a	InvalidConfigFile
0x8000006b	InvalidAxEnEvtArraySize
0x8000006c	InvalidAxEnEvtArray
0x8000006d	InvalidGpEnEvtArraySize

0x8000006e	InvalidGpEnEvtArray
0x8000006f	InvalidIntervalData
0x80000070	InvalidEndPosition
0x80000071	InvalidAxisSelect
0x80000072	InvalidTableSize
0x80000073	InvalidGpHandle
0x80000074	InvalidCmpSource
0x80000075	InvalidCmpMethod
0x80000076	InvalidCmpPulseMode
0x80000077	InvalidCmpPulseLogic
0x80000078	InvalidCmpPulseWidth
0x80000079	InvalidPathFunctionID
0x8000007a	SysBufAllocateFailed
0x8000007b	SpeedFordFunNotSpported
0x8000007c	InvalidNormVector
0x8000007d	InvalidCmpTimeTableCount
0x8000007e	InvalidCmpTime
0x8000007f	FWDownLoading
0x80000080	FWVersionNotMatch
0x80000081	InvalidAxParHomeVelLow
0x80000082	InvalidAxParHomeVelHigh
0x80000083	InvalidAxParHomeAcc
0x80000084	InvalidAxParHomeDec
0x80000085	InvalidAxParHomeJerk
0x80000086	InvalidAxCfgJogVelLow
0x80000087	InvalidAxCfgJogVelHigh
0x80000088	InvalidAxCfgJogAcc
0x80000089	InvalidAxCfgJogDec
0x8000008a	InvalidAxCfgJogJerk

0x8000008b	InvalidAxCfgKillDec
0x8000008c	NotOpenAllAxes
0x8000008d	NotSetServoComPort
0x8000008e	OpenComPortFailed
0x8000008f	ReadComPortTimeOut
0x80000090	SetComPortStateFailed
0x80000091	SevroTypeNotSupport
0x80000092	ReadComBufFailed
0x80000096	SlaveIOUpdateError
0x80000097	NoSlaveDevFound
0x80000098	MasterDevNotOpen
0x80000099	MasterRingNotOpen
0x800000c8	InvalidDIPort
0x800000c9	InvalidDOPort
0x800000ca	InvalidDOValue
0x800000cb	CreateEventFailed
0x800000cc	CreateThreadFailed
0x800000cd	InvalidHomeModeEx
0x800000ce	InvalidDirMode
0x800000cf	AxHomeMotionFailed
0x800000d0	ReadFileFailed
0x800000d1	PathBufIsFull
0x800000d2	PathBufIsEmpty
0x800000d3	GetAuthorityFailed
0x800000d4	GpIDAllocatedFailed
0x800000d5	FirmWareDown
0x800000d6	InvalidGpRadius
0x800000d7	InvalidAxCmd
0x800000d8	InvalidaxExtDrv

0x800000d9	InvalidGpMovCmd
0x800000da	SpeedCurveNotSupported
0x800000db	InvalidCounterNo
0x800000dc	InvalidPathMoveMode
0x800000dd	PathSelStartCantRunInSpeedForewareMode
0x800000de	InvalidCamTableID
0x800000df	InvalidCamPointRange
0x800000e0	CamTableIsEmpty
0x800000e1	InvalidPlaneVector
0x800000e2	MasAxIDSameSlvAxID
0x800000e3	InvalidGpRefPlane
0x800000e4	InvalidAxModuleRange
0x800000e5	DownloadFileFailed
0x800000e6	InvalidFileLength
0x800000e7	InvalidCmpCnt
0x800000e8	JerkExceededMaxValue
0x800000e9	AbsMotionNotSupport
0x800000ea	invalidAiRange
0x800000eb	AI ScaleFailed
0x800000ec	AxInRobot
0x800000ed	Invalid3DarcFlat
0x800000ee	InvalidIpoMap
0x800000ef	DataSizeNotCorrect
0x800000f0	AxisNotFound
0x800000f1	InvalidPathVelHigh
0x80002000	HlmtPExceeded
0x80002001	HlmtNExceeded
0x80002002	SlmtPExceeded

0x80002003	SlmtNExceeded
0x80002004	AlarmHappened
0x80002005	EmgHappened
0x80002006	TimeLmtExceeded
0x80002007	DistLmtExceeded
0x80002008	InvalidPositionOverride
0x80002009	OperationErrorHappened
0x8000200a	SimultaneousStopHappened
0x8000200b	OverflowInPAPB
0x8000200c	OverflowInIPO
0x8000200d	STPHappened
0x8000200e	SDHappened
0x8000200f	AxisNoCmpDataLeft
0x10000001	Warning_AxWasInGp
0x10000002	Warning_GpInconsistRate
0x10000003	Warning_GpInconsistPPU
0x10000004	Warning_GpMoveDistanceCanntBeZero
0x80004001	DevEvtTimeOut
0x80004002	DevNoEvt
0x80005001	ERR_SYS_TIME_OUT
0x80005002	Dsp_PropertyIDNotSupport
0x80005003	Dsp_PropertyIDReadOnly
0x80005004	Dsp_InvalidParameter
0x80005005	Dsp_DataOutBufExceeded
0x80005006	Dsp_FunctionNotSupport
0x80005007	Dsp_InvalidConfigFile

0x80005008	Dsp_InvalidIntervalData
0x80005009	Dsp_InvalidTableSize
0x8000500a	Dsp_InvalidTableID
0x8000500b	Dsp_DataIndexExceedBufSize
0x8000500c	Dsp_InvalidCompareInterval
0x8000500d	Dsp_InvalidCompareRange
0x8000500e	Dsp_PropertyIDWriteOnly
0x8000500f	Dsp_NcError
0x80005010	Dsp_CamTableIsInUse
0x80005011	Dsp_EraseBlockFailed
0x80005012	Dsp_ProgramFlashFailed
0x80005013	Dsp_WatchdogError
0x80005014	Dsp_ReadPrivateOverMaxTimes
0x80005015	Dsp_InvalidPrivateID
0x80005016	Dsp_DataNotReady
0x80005017	Dsp_LastOperationNotOver
0x80005018	Dsp_WritePrivateTimeout
0x80005019	Dsp_FwIsDownloading
0x80005020	Dsp_FwDownloadStepError
0x80005101	Dsp_InvalidAxCfgVel
0x80005102	Dsp_InvalidAxCfgAcc
0x80005103	Dsp_InvalidAxCfgDec
0x80005104	Dsp_InvalidAxCfgJerk
0x80005105	Dsp_InvalidAxParVelLow
0x80005106	Dsp_InvalidAxParVelHigh
0x80005107	Dsp_InvalidAxParAcc
0x80005108	Dsp_InvalidAxParDec
0x80005109	Dsp_InvalidAxParJerk

0x8000510a	Dsp_InvalidAxPptValue
0x8000510b	Dsp_InvalidAxState
0x8000510c	Dsp_InvalidAxSvOnOff
0x8000510d	Dsp_InvalidAxDistance
0x8000510e	Dsp_InvalidAxPosition
0x8000510f	Dsp_InvalidAxHomeMode
0x80005110	Dsp_InvalidPhysicalAxis
0x80005111	Dsp_HlmtPExceeded
0x80005112	Dsp_HlmtNExceeded
0x80005113	Dsp_SlmtPExceeded
0x80005114	Dsp_SlmtNExceeded
0x80005115	Dsp_AlarmHappened
0x80005116	Dsp_EmgHappened
0x80005117	Dsp_CmdValidOnlyInConstSec
0x80005118	Dsp_InvalidAxCmd
0x80005119	Dsp_InvalidAxHomeDirMode
0x8000511a	Dsp_AxisMustBeModuloAxis
0x8000511b	Dsp_AxIdCantSameAsMasId
0x8000511c	Dsp_CantResetPosiOfMasAxis
0x8000511d	Dsp_InvalidAxExtDrvOperation
0x8000511e	Dsp_AxAccExceededMaxAcc
0x8000511f	Dsp_AxVelExceededMaxVel
0x80005120	Dsp_NotEnoughPulseForChgV
0x80005121	Dsp_NewVelMustGreaterThanVelLow
0x80005122	Dsp_InvalidAxGearMode
0x80005123	Dsp_InvalidGearRatio
0x80005124	Dsp_InvalidPWMDDataCount
0x80005125	Dsp_InvalidAxPWMFreq
0x80005126	Dsp_InvalidAxPWMDuty



0x80005127	Dsp_AxGantryExceedMaxDiffValue
0x80005128	Dsp_ChanelIsDisable
0x80005129	Dsp_ChanelBufferIsFull
0x80005130	Dsp_ChanelBufferIsEmpty
0x80005131	Dsp_InvalidDoChanelID
0x80005132	Dsp_LatchHappened
0x80005201	Dsp_InvalidAxCntInGp
0x80005202	Dsp_AxInGpNotFound
0x80005203	Dsp_AxIsInOtherGp
0x80005204	Dsp_AxCannotIntoGp
0x80005205	Dsp_GpInDevNotFound
0x80005206	Dsp_InvalidGpCfgVel
0x80005207	Dsp_InvalidGpCfgAcc
0x80005208	Dsp_InvalidGpCfgDec
0x80005209	Dsp_InvalidGpCfgJerk
0x8000520a	Dsp_InvalidGpParVelLow
0x8000520b	Dsp_InvalidGpParVelHigh
0x8000520c	Dsp_InvalidGpParAcc
0x8000520d	Dsp_InvalidGpParDec
0x8000520e	Dsp_InvalidGpParJerk
0x8000520f	Dsp_JerkNotSupport
0x80005210	Dsp_ThreeAxNotSupport
0x80005211	Dsp_DevIpoNotFinished
0x80005212	Dsp_InvalidGpState
0x80005213	Dsp_OpenFileFailed
0x80005214	Dsp_InvalidPathCnt
0x80005215	Dsp_InvalidPathHandle
0x80005216	Dsp_InvalidPath

0x80005217	Dsp_GpSlavePositionOverMaster
0x80005218	Dsp_GpPathBufferOverflow
0x80005219	Dsp_InvalidPathFunctionID
0x8000521a	Dsp_SysBufAllocateFailed
0x8000521b	Dsp_InvalidGpCenterPosition
0x8000521c	Dsp_InvalidGpEndPosition
0x8000521d	Dsp_InvalidGpCmd
0x8000521e	Dsp_AxHasBeenInInGp
0x8000521f	Dsp_ThreeAxNotSupport
0x80005220	Dsp_InvalidPathRange
0x80005221	Dsp_InvalidNormVector

### 3.1.2 SYSTEM\_ERROR 错误代码表

错误代码	说明
0	SUCCESS
0x90000000	AMI_NULL_PROJECT_EXIST
0x90000001	AMI_INVALID_INPUT_PARAMS
0x90000002	AMI_INVALID_RETURN
0x90000003	AMI_INVALID_CTRL_MODE
0x90000004	AMI_CONTROLLER_LOCKED
0x90000005	AMI_GET_MAC_FAILED
0x90000006	AMI_INVALID_COMMAND
0x90000007	AMI_SET_MEM_FAILED
0x90000008	AMI_GET_VERSION_FAILED
0x9000000a	AMI_CTRL_ENCODED_ALREADY
0x9000000b	AMI_CTRL_INVALID_PASSWORD
0x9000000c	AMI_GET_VARIABLE_FAILED
0x9000000d	AMI_NUM_CONVERT_FAILED
0x90000032	AMI_ACTION_NOT_ALLOWED

0x90000064	AMI SOCK_TIME_OUT
0x90000065	AMI_LOAD_FILE_FAILED
0x90000066	AMI_DOWN_FILE_FAILED
0x90000067	AMI_LOAD_PROJECT_FAILED
0x90000068	AMI_DOWN_PROJECT_FAILED
0x90000069	AMI SOCK_ALREADY_CONNECTED
0x9000006A	AMI SOCK_COMMU_FAILED
0x90000096	AMI_CONNECTION_FAILED
0x90000097	AMI_DISCONNECTION_FAILED
0x90000098	AMI_SEND_COMMAND_TIMEOUT
0x900000C8	AMI_OPEN_FILE_FAILED
0x900000C9	AMI_CREATE_FILE_FAILED
0x900000CA	AMI_REMOVE_FILE_FAILED
0x900000CB	AMI_PATH_NOT_EXIST
0x900000CC	AMI_SET_NON_BLOCK_FAILED
0x900000CD	AMI_SET_BLOCK_FAILED
0x900000CE	AMI_CFG_FILE_NOT_EXISTS
0x900000CF	AMI_REF_FILE_NOT_EXISTS
0x900000D0	AMI_HEAD_FILE_NOT_EXISTS
0x900000D1	AMI_FILE_NOT_EXISTS
0x900000D2	AMI_FILE_INVALID_FORMAT
0x900000DC	AMI_PRJ_FILE_LOAD_FAILED
0x900000DD	AMI_SOURCE_FILE_NOT_EXISTS
0x900000DE	AMI_DST_FILE_EXISTS_ALREADY
0x900000FA	AMI_XML_LOAD_FAILED
0x900000FB	AMI_XML_CHECK_FAILED
0x900000FC	AMI_XML_SAVE_FAILED

0x900000FD	AMI_XML_ADD_FAILED
0x900000FE	AMI_XML_DELETE_FAILED
0x900000FF	AMI_XML_CREATE_FAILED
0x90000100	AMI_XML_INVALID_ELEMENT
0x9000012C	AMI_TASK_NOT_EXIST
0x9000012D	AMI_FORK_PROCESS_FAILED
0x9000012E	AMI_POPEN_FILE_FAILED
0x9000012F	AMI_STILL_RUNNING
0x90000130	AMI_NOT_IN_IDLE
0x90000131	AMI_GET_NO_ERROR
0x90000132	AMI_GET_NO_INFO
0x90000133	AMI_GET_MSG_NOTFINISHED
0x90000134	AMI_CREAT_PIPE_FAILED
0x90000136	AMI_RUN_FAILED
0x90000137	AMI_STOP_FAILED
0x90000138	AMI_NOT_RUNNING
0x90000140	AMI_DB_INIT_FAILED
0x90000141	AMI_DB_COMPILE_FAILED
0x90000142	AMI_DB_BREAKPOINT_FAILED
0x90000143	AMI_DB_CLEARPOINT_FAILED
0x90000144	AMI_DB_DELETEPOINTS_FAILED
0x90000145	AMI_DB_RUN_FAILED
0x90000146	AMI_DB_CONTINUE_FAILED
0x90000147	AMI_DB_NEXT_FAILED
0x90000148	AMI_DB_PROGRAM_NOT_RUN
0x90000149	AMI_DB_STOP_FAILED
0x9000014A	AMI_DB_OUT_OF_RANGE
0x9000014B	AMI_DB_EXIT_NOMARLLY

0x9000014C	AMI_DB_GET_LOCAL_VAR_FAILED
0x9000014D	AMI_DB_NOT_READY
0x9000014E	AMI_DB_ALREADY_PAUSED
0x9000014F	AMI_GET_RUNNING_TASKLIST_FAILED
0x90000190	AMI_MB_ILLEGAL_FUNCTION
0x90000191	AMI_MB_CRC_FAILED
0x90000192	AMI_MB_ILLEGAL_LENGTH
0x900001F4	AMI_MEM_UPDATE_VR_MBADDR_ERROR
0x900001F5	AMI_MEM_UPDATE_TABLE_MBADDR_ERROR
0x900001F6	AMI_MEM_UPDATE_DO_INIT_VALUE_ERROR
0x90000258	AMI_BASIC_RESET_ERROR
0x90000259	AMI_BASIC_INITIAL_ERROR
0x9000025A	AMI_BASIC_REFRESH_ERROR
0x9000025B	AMI_BASIC_GET_OFFSET_VALUE_FAILED
0xb0000000	AMI_GetSharedMemFailed
0xb0000001	AMI_GetTaskNameFailed
0xb0000002	AMI_IsNotInitialized
0xb0000003	AMI_IsAlreadyInitialized
0xb0000004	AMI_LoadXMLFailed
0xb0000005	AMI_ParseXMLFailed
0xb0000006	AMI_CreateDevListFailed
0xb0000007	AMI_InitializeDeviceFailed
0xb0000008	AMI_InitializeSharedMemFailed
0xb0000009	AMI_RefreshSharedMemFailed
0xb000000a	AMI_SetDeviceCfgFailed

0xb000000b	AMI_IncorrectCommand
0xb000000c	AMI_DeviceLargerList
0xb000000d	AMI_SerialPortError
0xb000000e	AMI_EthernetError
0xb000000f	AMI_LogOpenFailed
0xb0000010	AMI_StartTaskFailed
0xb0000011	AMI_StopTaskFailed
0xb0000012	AMI_CreatEventFailed
0xb0000013	AMI_CreatEThreadsFailed
0xb0000014	AMI_AllocatePMotionInfoFiled
0xb0000015	AMI_FAILEDTOCHECKEVENT
0xb0000016	AMI_FailedCloseCheckingEventThread
0xb0001000	AMI_CardsNotFound
0xb0001001	AMI_MotionBoardIDNotFound
0xb0001002	AMI_AxesOrGroupCountNotFound
0xb0001003	AMI_AxisIDorPhyIDNotFound
0xb0001004	AMI_GroupNotFound
0xb0001005	AMI_AxisInfoError
0xb0001006	AMI_MotionDeviceCountError
0xb0001007	AMI_InputBoardIDNotFound
0xb0001008	AMI_InputDeviceCountError
0xb0001009	AMI_InDAQdeviceCountError
0xb000100a	AMI_OutputBoardIDNotFound
0xb000100b	AMI_OutputDeviceCountError
0xb000100c	AMI_OutDAQdeviceCountError
0xb000100d	AMI_ActualDeviceCountError
0xb0002000	AMI_WrongAxisIndex

0xb0002001	AMI_WrongDoIndex
0xb0002002	AMI_WrongDiIndex
0xb0002003	AMI_NoAxis
0xb0002004	AMI_AxisInDifferentDevice
0xb0002005	AMI_WaitModeNotMatch
0xb0002006	AMI_MasterAxisIndexError
0xb0002007	AMI_GANTRYAxisNotInSameDev
0xb0002008	AMI_GEARAxisNotInSameDev
0xb0002009	AMI_AddPathAxisCntError
0xb000200a	AMI_AddPathHELIX3PnotSupport
0xb0003000	AMI_EthernetModeError
0xb0003001	AMI_EthernetOpened
0xb0003002	AMI_EthernetOpenFailed
0xb0003003	AMI_EthernetCloseFailed
0xb0003004	AMI_EthernetWrongNum
0xb0003005	AMI_EthernetNotOpen
0xb0003006	AMI_EthernetReadFailed
0xb0003007	AMI_EthernetResetFailed
0xb0003008	AMI_EthernetWriteFailed
0xb0003009	AMI_EthernetReadVRFailed
0xb000300a	AMI_EthernetWriteVRFailed
0xb0003100	AMI_SerialPortWrongID
0xb0003101	AMI_SerialPortOpenFailed
0xb0003102	AMI_SerialPortCloseFailed
0xb0003103	AMI_SerialPortNotOpen
0xb0003104	AMI_SerialPortWrongCfg
0xb0003105	AMI_SerialPortSetCfgFailed
0xb0003106	AMI_SerialPortWriteFailed

0xb0003107	AMI_SerialPortReadFailed
0xb0003108	AMI_SerialPortResetFailed
0xb0003109	AMI_SerialPortWriteVRFailed
0xb000310a	AMI_SerialPortReadVRFailed



## 3.2 ModBUS 地址

Motion Studio 中的变量内建有 ModBUS 地址，用于与外部的通信。变量与 ModBUS 地址的对应关系分五大区块，如下表：

区块	ModBUS 组别	ModBUS 地址范围	读/写
标准变量—DOUT	COIL STATUS	1~1024	可读/可写
标准变量—DI	INPUT STATUS	10001~11024	只读
标准变量—状态 (运动控制相关)	INPUT REGISTER	30001~31600	只读
标准变量—参数、配置 (运动控制相关)	HOLDING REGISTER	45001~49800	可读/可写
内建自定义变量—VR	HOLDING REGISTER	40001~44000	可读/可写

注：以下各区块变量与 ModBUS 对应关系中数据类型说明如下：

- **Bit**：位
- **UINT16**: 16 位无符号整型
- **INT16**: 16 位整型
- **UINT32**：32 位无符号整型
- **INT 32**: 32 位整型
- **F32**：32 位浮点型

3.2.1 DOUT 对应的 ModBUS 地址

变量	说明	ModBUS 地址	数据类型
DOUT(0)	索引为 0 的数字量输出	1	Bit
DOUT(1)	索引为 1 的数字量输出	2	Bit
.....			Bit
DOUT(1023)	索引为 1023 的数字量输出	1024	Bit

3.2.2 DI 对应的 ModBUS 地址

变量	说明	ModBUS 地址	数据类型
DI(0)	索引为 0 的数字量输入	10001	Bit
DI(1)	索引为 1 的数字量输入	10002	Bit
.....			Bit
DI(1023)	索引为 1023 的数字量输入	11024	Bit

### 3.2.3 轴状态对应的 ModBUS 地址

轴状态	ModBUS 地址范围
轴 0	30001~30100
轴 1	30101~30200
.....	
轴 15	31501~31600

#### ● 轴 0 状态的 ModBUS 地址表

变量	说明	ModBUS 地址	数据类型
DPOS	累计指令位置（理论位置）	30001	F32
MPOS	累计反馈位置（实际位置）	30003	F32
STATE	当前轴运动状态	30005	UINT16
M_STATUS	暂无作用，预留	30006	UINT32
DSPEED	当前轴运动指令速度（理论速度）	30008	F32
MIO	运动相关 I/O 的状态	30010	UINT32
RUN_ERROR	轴错误信息	30012	UINT32

#### ● 轴 1 状态的 ModBUS 地址表

变量	说明	ModBUS 地址	数据类型
DPOS	累计指令位置（理论位置）	30101	F32
MPOS	累计反馈位置（实际位置）	30103	F32
STATE	当前轴运动状态	30105	UINT16
M_STATUS	暂无作用，预留	30106	UINT32
DSPEED	当前轴运动指令速度（理论速度）	30108	F32
MIO	运动相关 I/O 的状态	30110	UINT32
RUN_ERROR	轴错误信息	30112	UINT32

#### ● .....

● 轴 15 状态的 ModBUS 地址表

变量	说明	ModBUS 地址	数据类型
DPOS	累计指令位置（理论位置）	31501	F32
MPOS	累计反馈位置（实际位置）	31503	F32
STATE	当前轴运动状态	31505	UINT16
M_STATUS	暂无作用，预留	31506	UINT32
DSPEED	当前轴运动指令速度（理论速度）	31508	F32
MIO	运动相关 I/O 的状态	31510	UINT32
RUN_ERROR	轴错误信息	31512	UINT32

### 3.2.4 轴参数、配置对应的 ModBUS 地址

轴参数、配置	ModBUS 地址范围
轴 0	45001~45300
轴 1	45301~45600
.....	
轴 15	49501~49800

● 轴 0 参数、配置的 ModBUS 地址表

变量	说明	ModBUS 地址	数据类型
VL	轴初速度	45001	F32
VH	轴运行速度	45003	F32
ACC	轴加速度	45005	F32
DEC	轴减速度	45007	F32
JK	速度曲线类型：S 型或 T 型	45009	F32
MAXVEL	轴运行速度限值	45011	F32
MAXACC	轴加速度限值	45013	F32
MAXDEC	轴减速度限值	45015	F32
MAXJK	暂无作用，预留	45017	F32
INSTOP_DEC	INSTOP 功能中的减速度	45019	F32
UNIT_NUM	脉冲当量分子	45031	UINT32
UNIT_DENOM	脉冲当量分母	45033	UINT32
HOME_CROSS	回原点运行中的跨越距离	45051	F32
HOME_EX	暂无作用，预留	45053	UINT16
ORG_LOGIC	ORG 信号的有效逻辑电平	45054	UINT16
ORG_MODE	回原点运动结束时的停止模式	45055	UINT16
EZ_LOGIC	Z 相输入信号的有效逻辑电平	45056	UINT16
HOME_RESET	回原点后清零位置值功能	45057	UINT16
HOME_OFFSETDIST	回原点成后的偏移距离	45058	F32
HOME_OFFSETVEL	回原点成后偏移距离的移动速	45060	F32

	度		
HOME_MODE	回原点模式	45062	UINT16
HOME_VL	回原点运动的初速度	45063	F32
HOME_VH	回原点运动的运行速度	45065	F32
HOME_ACC	回原点运动的加速度	45067	F32
HOME_DEC	回原点运动的减速度	45069	F32
HOME_JK	回原点运动的速度曲线类型	45071	F32
JOG_VLTIME	JOG 运动低段速度运行的时间	45081	UINT32
JOG_VL	JOG 运动低段速度	45083	F32
JOG_VH	JOG 运动高段速度	45085	F32
JOG_ACC	JOG 运动的加速度	45087	F32
JOG_DEC	JOG 运动的减速度	45089	F32
EL_EN	使能硬件限位功能	45101	UINT16
EL_LOGIC	硬件限位信号的有效逻辑电平	45102	UINT16
EL_MODE	硬限位触发时的停止模式	45103	UINT16
PEL_TOL_EN	正方向硬极限容差功能使能	45104	UINT16
PEL_TOL	正方向硬极限容差值	45105	UINT32
NEL_TOL_EN	负方向硬极限容差功能使能	45107	UINT16
NEL_TOL	负方向硬极限容差值	45108	UINT32
PIN_MODE	编码器脉冲输入类型	45121	UINT16
PIN_LOGIC	编码器脉冲输入逻辑反相	45122	UINT16
PIN_MAXFREQ	编码器脉冲输入的最高频率限值	45123	UINT16
POUT_MODE	指令脉冲输出类型	45131	UINT16
POUT_REVERSE	指令脉冲输出逻辑反相	45132	UINT16
ALM_FILTER	报警(ALM) 端口的滤波时间	45141	UINT16
NEL_FILTER	负方向硬限位端口滤波时间	45142	UINT16
PEL_FILTER	正方向硬限位端口滤波时间	45143	UINT16
ORG_FILTER	原点信号端口滤波时间	45144	UINT16

IN1_FILTER	IN1 端口的滤波时间	45145	UINT16
IN2_FILTER	IN2 端口的滤波时间	45146	UINT16
IN4_FILTER	IN4 端口的滤波时间	45147	UINT16
IN5_FILTER	IN5 端口的滤波时间	45148	UINT16
EXT_SRC	哪个轴的外部驱动输入端口作为外部驱动信号源	45161	UINT16
EXT_EN	暂无作用，预留	45162	UINT16
EXT_MODE	手轮模式外部驱动的脉冲输入模式	45163	UINT16
EXT_PULSE	手轮模式外部驱动时，每个手轮脉冲输入对应多少个指令脉冲输出值	45164	UINT32

● 轴 1 参数、配置的 ModBUS 地址表

变量	说明	ModBUS 地址	数据类型
VL	轴初速度	45301	F32
VH	轴运行速度	45303	F32
ACC	轴加速度	45305	F32
DEC	轴减速度	45307	F32
JK	速度曲线类型：S 型或 T 型	45309	F32
MAXVEL	轴运行速度限值	45311	F32
MAXACC	轴加速度限值	45313	F32
MAXDEC	轴减速度限值	45315	F32
MAXJK	暂无作用，预留	45317	F32
INSTOP_DEC	INSTOP 功能中的减速度	45319	F32
UNIT_NUM	脉冲当量分子	45331	UINT32
UNIT_DENOM	脉冲当量分母	45333	UINT32
HOME_CROSS	回原点运行中的跨越距离	45351	F32
HOME_EX	暂无作用，预留	45353	UINT16
ORG_LOGIC	ORG 信号的有效逻辑电平	45354	UINT16



ORG_MODE	回原点运动结束时的停止模式	45355	UINT16
EZ_LOGIC	Z 相输入信号的有效逻辑电平	45356	UINT16
HOME_RESET	回原点后清零位置值功能	45357	UINT16
HOME_OFFSETDIST	回原点成后的偏移距离	45358	F32
HOME_OFFSETVEL	回原点成后偏移距离的移动速度	45360	F32
HOME_MODE	回原点模式	45362	UINT16
HOME_VL	回原点运动的初速度	45363	F32
HOME_VH	回原点运动的运行速度	45365	F32
HOME_ACC	回原点运动的加速度	45367	F32
HOME_DEC	回原点运动的减速度	45369	F32
HOME_JK	回原点运动的速度曲线类型	45371	F32
JOG_VLTIME	JOG 运动低段速度运行的时间	45381	UINT32
JOG_VL	JOG 运动低段速度	45383	F32
JOG_VH	JOG 运动高段速度	45385	F32
JOG_ACC	JOG 运动的加速度	45387	F32
JOG_DEC	JOG 运动的减速度	45389	F32
EL_EN	使能硬件限位功能	45401	UINT16
EL_LOGIC	硬件限位信号的有效逻辑电平	45402	UINT16
EL_MODE	硬限位触发时的停止模式	45403	UINT16
PEL_TOL_EN	正方向硬极限容差功能使能	45404	UINT16
PEL_TOL	正方向硬极限容差值	45405	UINT32
NEL_TOL_EN	负方向硬极限容差功能使能	45407	UINT16
NEL_TOL	负方向硬极限容差值	45408	UINT32
PIN_MODE	编码器脉冲输入类型	45421	UINT16
PIN_LOGIC	编码器脉冲输入逻辑反相	45422	UINT16
PIN_MAXFREQ	编码器脉冲输入的最高频率限值	45423	UINT16
POUT_MODE	指令脉冲输出类型	45431	UINT16

POUT_REVERSE	指令脉冲输出逻辑反相	45432	UINT16
ALM_FILTER	报警(ALM) 端口的滤波时间	45441	UINT16
NEL_FILTER	负方向硬限位端口滤波时间	45442	UINT16
PEL_FILTER	正方向硬限位端口滤波时间	45443	UINT16
ORG_FILTER	原点信号端口滤波时间	45444	UINT16
IN1_FILTER	IN1 端口的滤波时间	45445	UINT16
IN2_FILTER	IN2 端口的滤波时间	45446	UINT16
IN4_FILTER	IN4 端口的滤波时间	45447	UINT16
IN5_FILTER	IN5 端口的滤波时间	45448	UINT16
EXT_SRC	哪个轴的外部驱动输入端口作为外部驱动信号源	45461	UINT16
EXT_EN	暂无作用，预留	45462	UINT16
EXT_MODE	手轮模式外部驱动的脉冲输入模式	45463	UINT16
EXT_PULSE	手轮模式外部驱动时，每个手轮脉冲输入对应多少个指令脉冲输出值	45464	UINT32

● .....

● 轴 15 参数、配置的 ModBUS 地址表

变量	说明	ModBUS 地址	数据类型
VL	轴初速度	49501	F32
VH	轴运行速度	49503	F32
ACC	轴加速度	49505	F32
DEC	轴减速度	49507	F32
JK	速度曲线类型：S 型或 T 型	49509	F32
MAXVEL	轴运行速度限值	49511	F32
MAXACC	轴加速度限值	49513	F32
MAXDEC	轴减速度限值	49515	F32

MAXJK	暂无作用，预留	49517	F32
INSTOP_DEC	INSTOP 功能中的减速度	49519	F32
UNIT_NUM	脉冲当量分子	49531	UINT32
UNIT_DENOM	脉冲当量分母	49533	UINT32
HOME_CROSS	回原点运行中的跨越距离	49551	F32
HOME_EX	暂无作用，预留	49553	UINT16
ORG_LOGIC	ORG 信号的有效逻辑电平	49554	UINT16
ORG_MODE	回原点运动结束时的停止模式	49555	UINT16
EZ_LOGIC	Z 相输入信号的有效逻辑电平	49556	UINT16
HOME_RESET	回原点后清零位置值功能	49557	UINT16
HOME_OFFSETDIST	回原点成后的偏移距离	49558	F32
HOME_OFFSETVEL	回原点成后偏移距离的移动速度	49560	F32
HOME_MODE	回原点模式	49562	UINT16
HOME_VL	回原点运动的初速度	49563	F32
HOME_VH	回原点运动的运行速度	49565	F32
HOME_ACC	回原点运动的加速度	49567	F32
HOME_DEC	回原点运动的减速度	49569	F32
HOME_JK	回原点运动的速度曲线类型	49571	F32
JOG_VLTIME	JOG 运动低段速度运行的时间	49581	UINT32
JOG_VL	JOG 运动低段速度	49583	F32
JOG_VH	JOG 运动高段速度	49585	F32
JOG_ACC	JOG 运动的加速度	49587	F32
JOG_DEC	JOG 运动的减速度	49589	F32
EL_EN	使能硬件限位功能	49601	UINT16
EL_LOGIC	硬件限位信号的有效逻辑电平	49602	UINT16
EL_MODE	硬限位触发时的停止模式	49603	UINT16
PEL_TOL_EN	正方向硬极限容差功能使能	49604	UINT16
PEL_TOL	正方向硬极限容差值	49605	UINT32

NEL_TOL_EN	负方向硬极限容差功能使能	49607	UINT16
NEL_TOL	负方向硬极限容差值	49608	UINT32
PIN_MODE	编码器脉冲输入类型	49621	UINT16
PIN_LOGIC	编码器脉冲输入逻辑反相	49622	UINT16
PIN_MAXFREQ	编码器脉冲输入的最高频率限值	49623	UINT16
POUT_MODE	指令脉冲输出类型	49631	UINT16
POUT_REVERSE	指令脉冲输出逻辑反相	49632	UINT16
ALM_FILTER	报警(ALM) 端口的滤波时间	49641	UINT16
NEL_FILTER	负方向硬限位端口滤波时间	49642	UINT16
PEL_FILTER	正方向硬限位端口滤波时间	49643	UINT16
ORG_FILTER	原点信号端口滤波时间	49644	UINT16
IN1_FILTER	IN1 端口的滤波时间	49645	UINT16
IN2_FILTER	IN2 端口的滤波时间	49646	UINT16
IN4_FILTER	IN4 端口的滤波时间	49647	UINT16
IN5_FILTER	IN5 端口的滤波时间	49648	UINT16
EXT_SRC	哪个轴的外部驱动输入端口作为外部驱动信号源	49661	UINT16
EXT_EN	暂无作用，预留	49662	UINT16
EXT_MODE	手轮模式外部驱动的脉冲输入模式	49663	UINT16
EXT_PULSE	手轮模式外部驱动时，每个手轮脉冲输入对应多少个指令脉冲输出值	49664	UINT32

### 3.2.5 运动指令对应的 Modbus 地址

#### 3.2.5.1 MOVE 运动指令

当上位机写 1 到“指令对应的 Modbus 地址”时，对应的轴会执行单轴相对运动，运动的距离为该轴对应的“运动距离对应的 Modbus 地址”里的值；上位机写 0 到“指令对应的 Modbus 地址”时，控制器不做任何处理。

说明	指令对应的 Modbus 地址 (数据类型 : Bit)	运动距离对应的 Modbus 地址 (数据类型 : F32)
轴 0 相对运动	9001	44501
轴 1 相对运动	9002	44503
.....		
轴 15 相对运动	9016	44531

### 3.2.5.2 JOG 运动指令

当上位机写 1 到“指令对应的 Modbus 地址”时，对应的轴会执行 JOG 运动。当上位机写 0 到“指令对应的 Modbus 地址”时，对应的轴会停止当前的 JOG 运动。

说明	指令对应的 Modbus 地址(数据类型 : Bit)
轴 0 正向 JOG 运动	9033
轴 1 正向 JOG 运动	9034
.....	
轴 15 正向 JOG 运动	9048
轴 0 负向 JOG 运动	9065
轴 1 负向 JOG 运动	9066
.....	
轴 15 负向 JOG 运动	9080

### 3.2.5.3 回原点运动指令

当上位机写 1 到“指令对应的 Modbus 地址”时，对应的轴会执行回原点运动。当上位机写 0 到“指令对应的 Modbus 地址”时，控制器不做任何处理。

说明	指令对应的 Modbus 地址(数据类型 : Bit)
轴 0 正向回原点运动	9097
轴 1 正向回原点运动	9098
.....	
轴 15 正向回原点运动	9112

轴 0 负向回原点运动	9129
轴 1 负向回原点运动	9130
.....	
轴 15 负向回原点运动	9144

#### 3.2.5.4 停止运动指令

当上位机写 1 到“指令对应的 Modbus 地址”时，控制器会控制对应的轴停止运动。当上位机写 0 到“指令对应的 Modbus 地址”时，控制器不做任何处理。

说明	指令对应的 Modbus 地址(数据类型：Bit)
停止轴 0 运动	9161
停止轴 1 运动	9162
.....	
停止轴 15 运动	9176