SAVCh®

SDV3 系列伺服系统

通用型

脚本功能详细说明书

创无限 赢久远
Innovate for more Win forever
SAVCH

用户手册

目录

一、 脚本概述 1
二、 脚本编辑器 1
2.1 脚本编辑区 3
2.2 函数列表区 3
2.3 输出区 4
2.4 脚本状态区 4
2.5 脚本操作区 4
2.5.1 在线 4
2.5.2 运行 5
2.5.3 停止 5
2.5.4 编译 5
2.5.5 上载 5
2.5.6 下载 6
2.5.7 变量监控 6
2.6 快捷程序块 6
2.7 脚本工具菜单 6
三、 脚本变量 7
四、 定时器 9
五、 脚本语法 9
5.1 基本语法 9
5.1.1 标示符 10
5.1.2 关键词 10
5.1.3 注释 10
5.2 数据类型 10
5.2.1 nil（空） 11
5.2.2 boolean（布尔） 11
5.2.3 number（数字） 11
5.2.4 string（字符串） 11
5.2.5 table（表） 12
5.2.6 function（函数） 12
5.3 变量 13

5.4 运算符 14
5.4.1 算术运算符 14
5.4.2 关系运算符 14
5.4.3 逻辑运算符 15
5.4.4 其他运算符 15
5.4.5 运算符优先级 15
5.5 循环 15
5.5.1 while 循环 16
5.5.2 for 循环 16
5.5.3 repeat...until 循环 16
5.5.4 循环嵌套 17
5.5.5 循环控制语句 17
5.5.6 无限循环 18
5.6 流程控制 18
5.6.1 if 语句 18
5.6.2 if...else 语句 18
5.6.3 if...elseif...else 语句 19
5.6.4 if 嵌套语句 20
5.7 函数 20
5.7.1 函数定义 20
5.7.2 多返回值 21
5.8 数组 21
5.8.1 一维数组 21
5.8.2 多维数组 21
5.9 table(表) 22
5.9.1 table(表)的构造 22
5.9.2 Table 操作 22
六、 加密与解密 23

一、 脚本概述

SDV3 系列伺服新增了脚本功能，可通过脚本功能灵活的控制伺服，增加一些自定义的控制功能，或者在一些简单的场合可以省去控制器，从而降低成本。

脚本规格	
编程语言	Lua
定时器	30 个
变量	M 变量(位变量)共 128 个
	D 变量(32 位变量)共 128 个
程序容量	32768 bytes
支持的伺服控制指令	端子控制
	参数控制
	状态读取
	定位控制
	速度控制
	扭矩控制

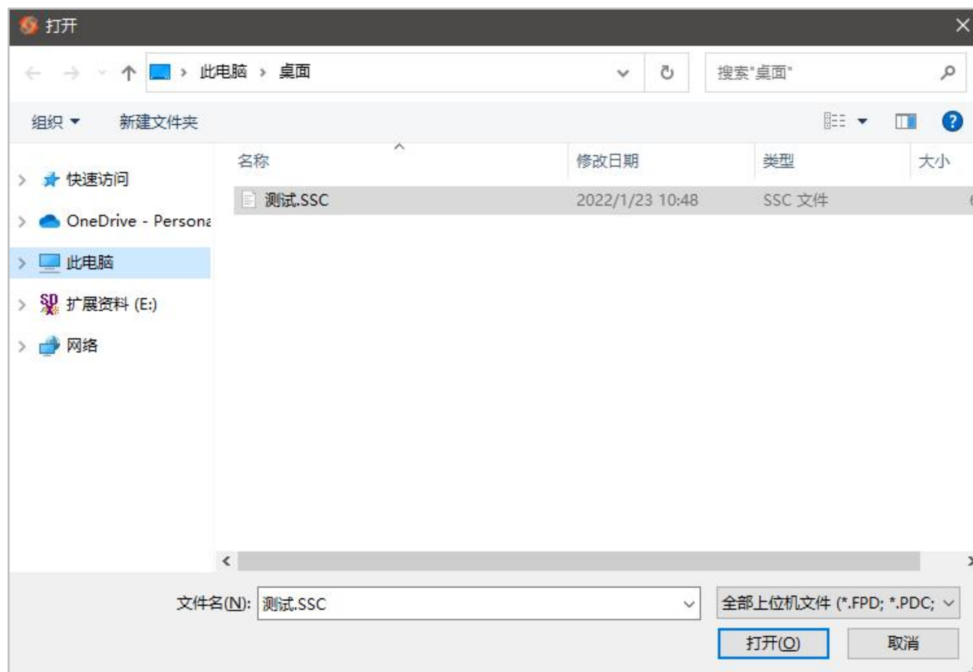
二、 脚本编辑器

Savchsoft_SRV 上位机集成了脚本编辑器功能，可通过以下三种方式打开脚本编辑器界面：

菜单栏->“菜单(M)“->”脚本编辑器“

新建文件对话框->”SDV3”->”脚本”

打开文件，选择脚本文件(.SSC)



脚本编辑的主界面如下图所示，包括以下 6 个功能区：

1.脚本编辑区、2.函数列表、3.输出区、4.脚本状态、5.脚本操作区、6.快捷程序块。

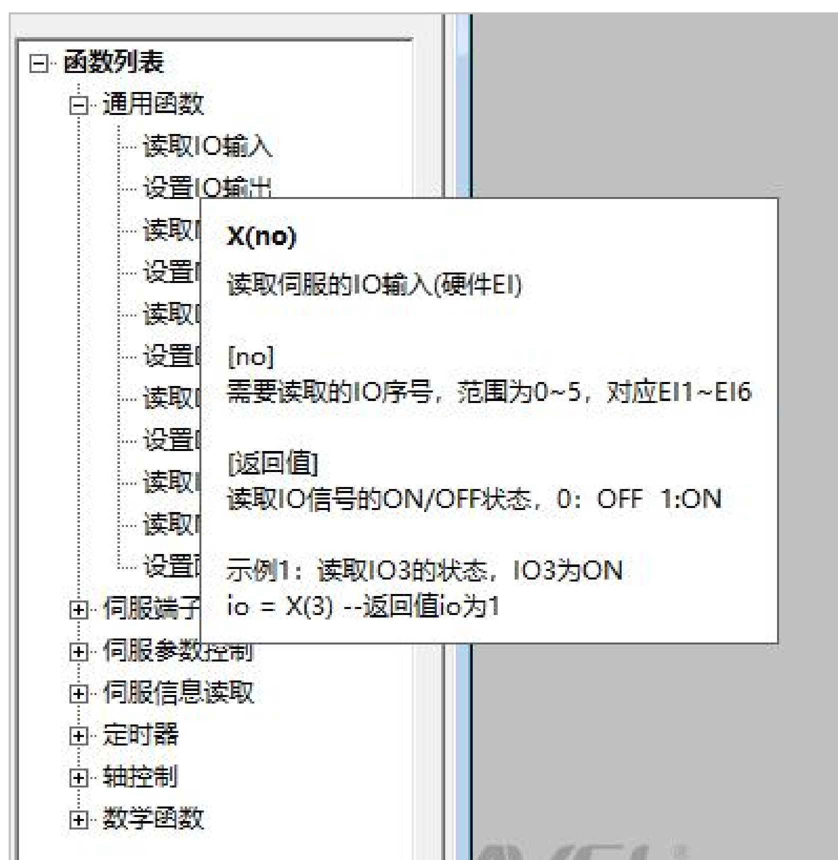
2.1 脚本编辑区

脚本编辑区内可进行脚本文件的编辑处理，支持常用的复制、粘贴和撤销等快捷键操作，在此区域单击鼠标右键，会显示功能菜单，通过功能菜单，还能便捷的插入常用程序块和注释等。

新建的脚本文件会在脚本编辑区插入默认代码，提示了变量定义的位置、初始化处理的位置及主循环处理的位置，该默认代码仅作为提示用，用户可以按照各自的习惯进行修改。

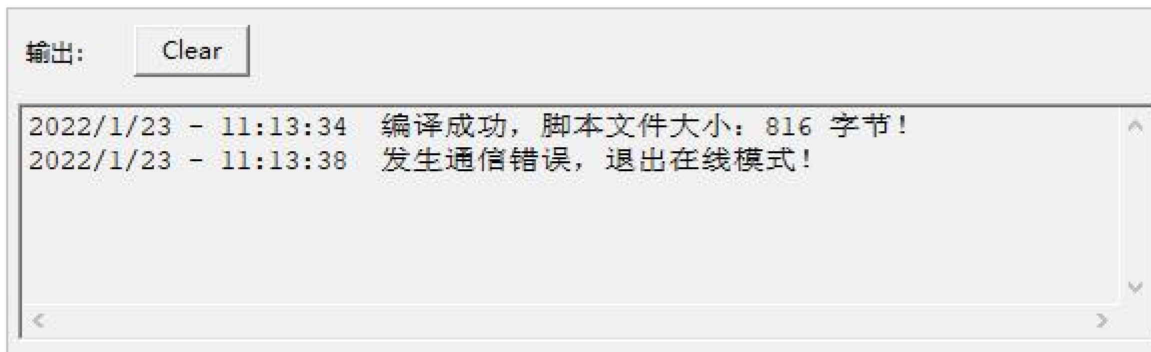
2.2 函数列表区

函数列表列出了 SDV3 伺服支持的各项函数，鼠标左键单击列表的加号可以展开函数列表，双击展开后的某项函数，该函数会自动添加到脚本编辑区光标所在的位置。当鼠标光标移动到展开的函数位置时，会自动显示该函数的相关说明，双击列表中的函数，即可在编辑栏的光标处插入选中的函数。



2.3 输出区

输出区会显示各项操作的结果及脚本执行过程中的一些错误信息以及该信息的时间戳。



2.4 脚本状态区

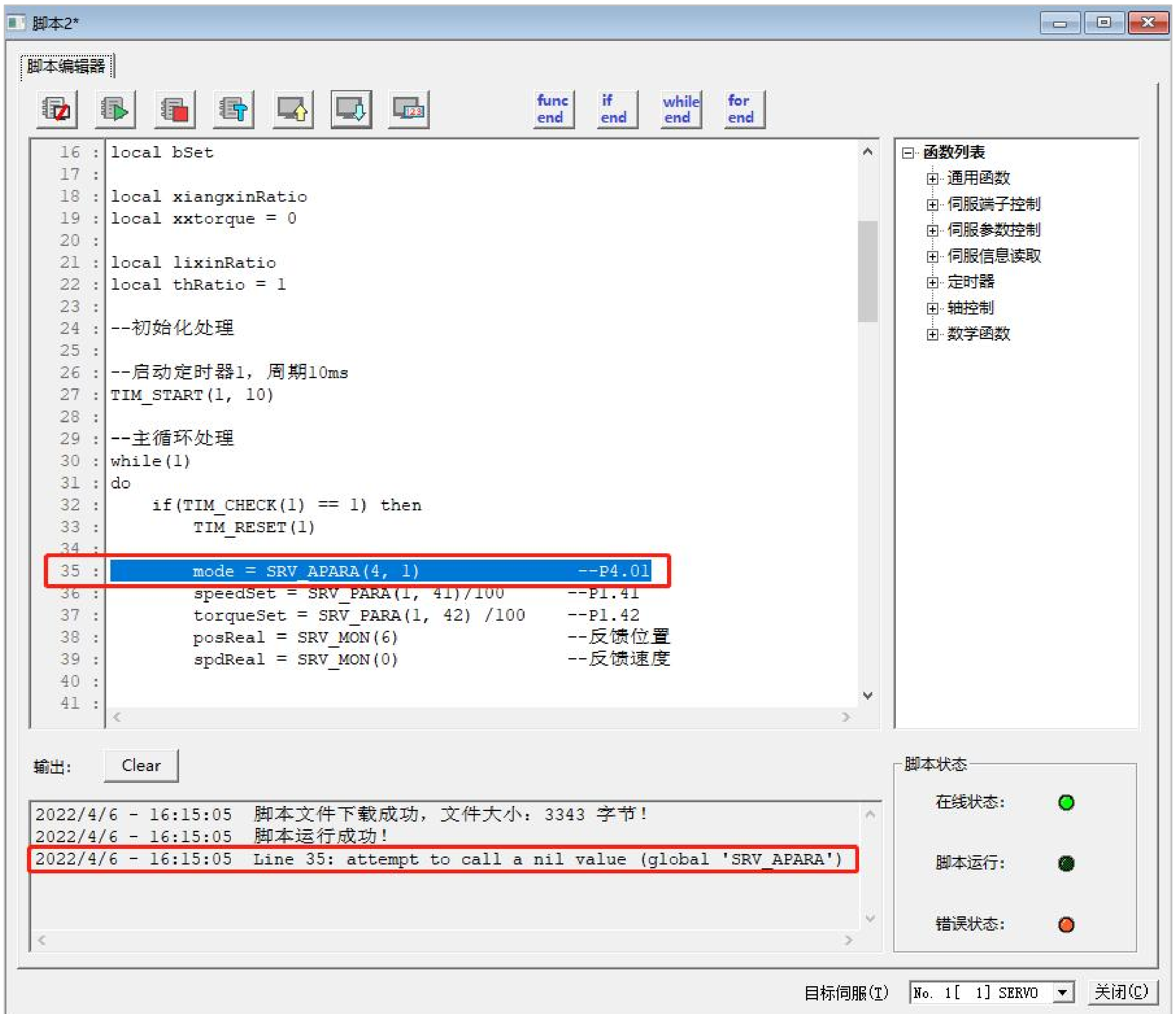
包括在线状态、脚本运行状态及错误状态，当处于在线状态时，在线状态指示灯会闪烁，此时如果脚本开始运行了，则脚本运行状态绿灯会常亮，发生错误时，测错误红灯会常量。

2.5 脚本操作区

脚本操作区包含 7 个功能按钮，从左到右分别是在线、运行、停止、编译、上传、下载和变量监控。鼠标光标移动到按钮上时，会显示出该按钮的名称，各按钮的功能如其名称所示。

2.5.1 在线

单击在线按钮，可以实时检测伺服内部脚本的执行状态，并在“脚本状态”区显示，且此时如果脚本执行发生了异常，会在“输出区”提示异常的信息，同时会将错误的那一行脚本在“脚本编辑区”中选中。



2.5.2 运行

当伺服内存在脚本，且脚本未运行时，可以通过“运行”按钮来启动脚本的执行。

2.5.3 停止

当脚本正在运行时，可以通过“停止”按钮来停止脚本的运行。

2.5.4 编译

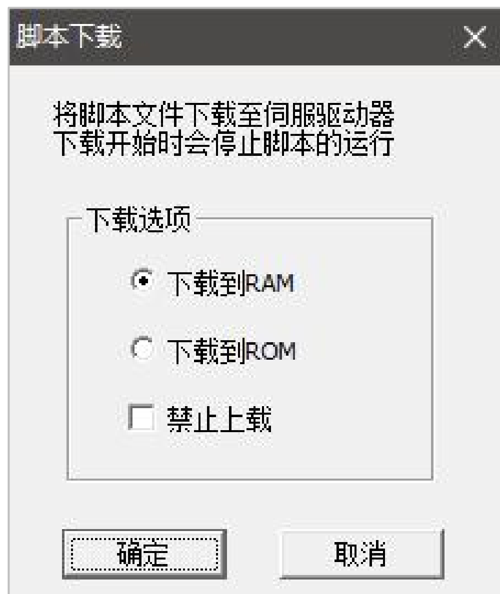
对“脚本编辑区”编写的脚本进行编译，检查是否有语法错误，当检查到语法错误时，会在“输出区”提示错误的信息，并在“脚本编辑区”内选中出错的行，当没有语法错误时，会在“输出区”显示编译成功以及脚本的大小。

2.5.5 上载

将伺服内的脚本文件读取到脚本编辑器，如果在下载时选中了“禁止上载”，则脚本将无法上载至编辑器，若是脚本加了密，则需要先解密后，才能完成上载操作。（加密解密的操作见第六章）

2.5.6 下载

将当前"脚本编辑区"的脚本下载至伺服，在下载前会先执行编译操作，以确保下载的脚本没有语法错误，下载前可以选择"下载至 RAM"和"下载至 ROM"，"下载至 RAM"的脚本在伺服断电后会丢失，但是下载速度快，不会对伺服内部存储器造成影响，适合用于调试阶段，"下载至 ROM"的脚本会储存在伺服内的存储器中，伺服重新上电后会自动执行。在下载的选项页面中，还有"禁止上载"的选项，选中后，下载至伺服的脚本将无法上载至编辑器。

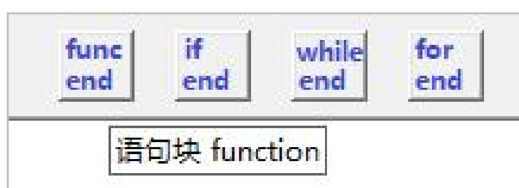


2.5.7 变量监控

打开变量监控对话框，在"变量监控"对话框中，可监控脚本的 M 变量和 D 变量，详细信息见第三章。

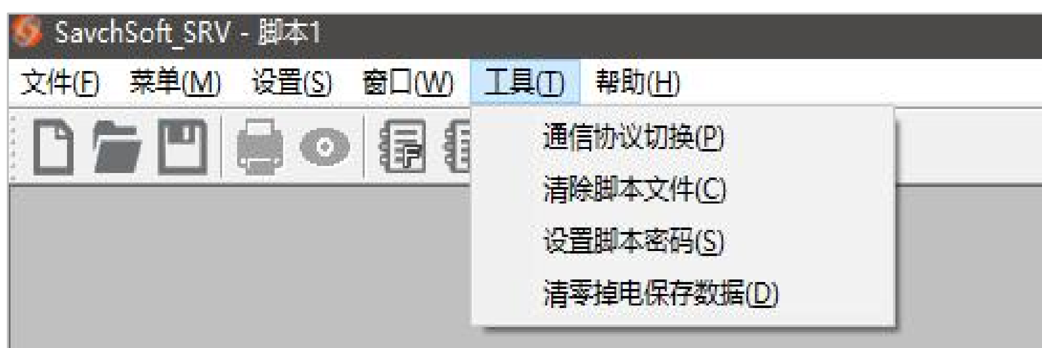
2.6 快捷程序块

该部分包含 4 个按钮，按下按钮可在编辑区当前光标处快速插入函数、if 分支、while 循环和 for 循环四种程序块。



2.7 脚本工具菜单

打开脚本编辑器后，菜单栏的"工具(T)"会有脚本专用的 3 个功能菜单，如下图所示。



三、 脚本变量

SDV3 系列伺服脚本内置了 128 个 M 变量(位变量)和 128 个 D 变量(32 位),其中 M 变量的 M96~M127 为掉电保存的变量，D 变量的 D100~D127 为掉电保存的变量，D 变量可保存 32 位带符号的整数，也可以保存单精度浮点数。

注：脚本内自定义的变量并未做个数限制。

上位机可通过 modbus 协议访问脚本变量，地址如下表所示：

变量类型	功能码	地址	备注
M 变量	0x01、0x05、0x0F	0x700~0x77F	M0~M127 分别对应 0x700~0x77F
M 变量	0x03、0x10	0xC000~0xC006	以 32 位方式访问 M 变量
D 变量	0x03、0x10	0xC100~0xC1FE	D 变量为 32 位数据，每个 D 变量占用 2 个地址

在脚本中，需通过通过函数对 M 变量和 D 变量进行访问，M 变量通过函数 M()进行访问，D 变量可通过 DD()和 DF()函数分别对 D 变量进行整数和浮点数格式的访问，示例如下：

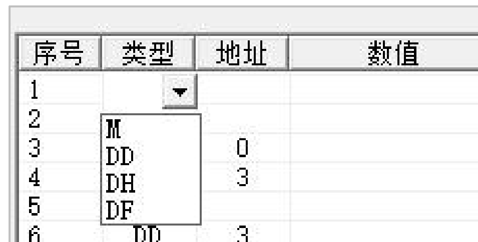
```
Mval = M(0)      --读取 M0 的值，并存入变量 Mval 中
M(0, 1)          --设置 M0 的值为 ON

Dval = DD(0)     --以整数的格式读取 D0 的值，并存入变量 Dval 中
Fval = DF(1)     --以浮点数的格式读取 D1 中的数据，并存入变量 Fval 中

DD(0, 100)       --设置 D0 的值为整数 100
DF(1, 3.14)      --设置 D1 的值为浮点数 3.14
```

注：整数与浮点数在 D 变量内储存的格式不同，对于同一个 D 变量不可混用 DD()函数和 DF()函数，否则可能出现数据错误。

在脚本编辑器软件中，还能打开变量监控表，实时监控变量的值。在脚本编辑区的脚本操作区单击“变量监控”按钮，即可打开变量监控表。

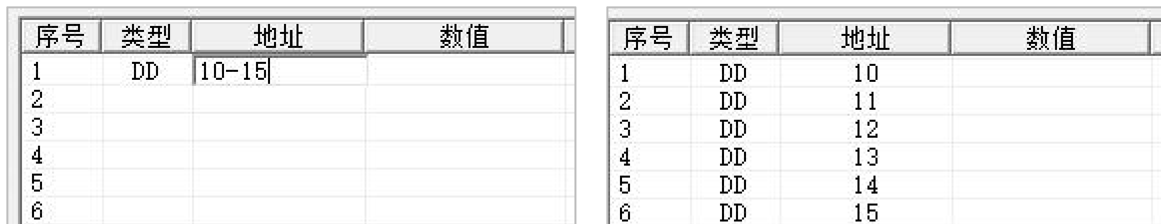
变量监控表如下图所示，监控状态灯亮起表示处于在线监控状态，通过“监控开”按钮打开在线变量监控，“监控关”关闭在线变量监控，当打开变量监控时脚本编辑器不处于在线状态，此时会自动先连接至在线状态，再打开变量监控。

单击类型框，会显示下拉框，选择要监控的变量类型



其中 DD 是 10 进制显示的 D 变量数值，DH 是 16 进制显示的 D 变量数值，DF 是以浮点数格式显示的 D 变量数值。

在地址栏输入变量的序号，在开启监控后，会在数值栏显示对于变量的数值，在地址栏输入两个数字中间以-连接，则会自动添加这两个数字包含的地址，例如在地址栏输入 10-15，则会自动添加 10~15 的地址.



在变量类型和地址已经设定的情况下，双击“数值”栏，会弹出数值设置对话框，可以强制修改变量的值。

数据下方的 ON OFF 单选框仅对 M 变量生效。

在“备注”栏可添加变量的注释。

序号	类型	地址	数值	备注
1	DD	10		测试变量

要删除已添加的变量和地址可通过选中该行然后单击键盘 del 键来删除。

四、 定时器

SDV3 系列伺服脚本内置了 30 个定时器，所有定时器的时基均为 1ms，最大可设置定时时间为 65535ms。定时器相关的操作函数共 6 个，分别为启动、停止、重置、暂停、恢复和检查。

名称	函数接口	函数说明
启动	TIM_START(t_no, t_val)	配置定时时间并启动对应序号的定时器
停止	TIM_STOP(t_no)	停止对应序号的定时器，停止后的定时器需通过启动函数才可再次启动
重置	TIM_RESET(t_no)	将启动后的定时器的计数值重置为 0，重置之后定时器将继续执行计数处理
暂停	TIIM_PAUSE(t_no)	停止对应序号正在计数的定时器，对于未启动的定时器无效
恢复	TIM_RESUME(t_no)	恢复对应序号暂停的定时器，对于未启动和暂停的定时器无效
检查	TIM_CHECK(t_no)	检查对应序号的定时器是否到达定时值，到达则返回 1，未到达或者未启动返回 0

以下示例简单描述了每 200ms 执行一次的定时器使用方式：

```
--变量定义
test = 1                                    --测试变量，初始值 1
--初始化处理
TIM_START(1, 200)                           -- 启动定时器 1，定时时间 200 毫秒
--主循环处理
while(1)
do
    if(TIM_CHECK(1) == 1) then              --检查定时器 1 是否到达定时值
        TIM_RESET(1)                        --定时器 1 到达 200ms 定时，重置定时器 1 的计数值为 0
        test= test+ 1                       --测试变量值每 200ms 执行加 1 操作
    end
end
```

五、 脚本语法

5.1 基本语法

SDV3 系列伺服使用的脚本语言为 Lua，遵循 Lua 脚本语言的语法。

SDV3 系列伺服仅支持单个文件，且该文件仅执行一次，所以在需要重复执行任务时，需要手动编写无限循环的处理，在新建脚本文件时，会生成默认脚本语句，如下所示：

```
--变量定义


--初始化处理


--主循环处理

```

```
while(1)
do

end
```

默认文件按照定义变量、初始化处理和无限循环的顺序，给出了默认的代码结构，用户在使用时可完全自己编写，页可以在默认文件的基础上修改。

注：后续的示例为了方便使用了 print()函数，在 SDV3 伺服实际使用时 print()函数并不会打印字符串，该功能在后续版本会添加，且需要上位机脚本编辑器在线时才可在输出栏中看到打印数据。

5.1.1 标示符

标示符用于定义一个变量，函数获取其他用户定义的项。标示符以一个字母 A 到 Z 或 a 到 z 或下划线 _ 开头后加上 0 个或多个字母，下划线，数字（0 到 9）。

最好不要使用下划线加大写字母的标示符，因为 Lua 的保留字也是这样的。

不允许使用特殊字符如 @, $, 和 % 来定义标示符。Lua 是一个区分大小写的编程语言，因此在 Hello 与 hello 是两个不同的标示符。以下列出了一些正确的标示符：

aaa、haha、Abc、a_ccc、a_123、srv5、_tmp、i、A23b9

5.1.2 关键词

保留关键字不能作为常量或变量或其他用户自定义标示符，以下列出了保留关键词：

and	break	do	else	nil	not	or	repeat
elseif	end	false	for	return	then	true	until
function	if	in	local	while	goto		

5.1.3 注释

注释包括单行注释和多行注释，在 SDV3 脚本中，注释也会占用程序的储存容量。

```
-- 单行注释
--[[
   多行注释
 --]]
```

5.2 数据类型

SDV3 脚本使用的数据类型共 6 种，如下表所示：

数据类型	描述
nil	只有值 nil 属于该类，表示一个无效值（在条件表达式中相当于 false）
boolean	包含两个值：false 和 true
number	表示单精度类型的实浮点数
string	字符串由一对双引号或单引号来表示

数据类型	描述
function	编写的函数
table	表（table）其实是一个"关联数组"（associative arrays），数组的索引可以是数字、字符串或表类型。

5.2.1 nil（空）

nil 类型表示一种没有任何有效值，它只有一个值 nil，例如一个没有赋值的变量，他的值就是 nil。

对于全局变量和 table，nil 还有一个"删除"作用，给全局变量或者 table 表里的变量赋一个 nil 值，等同于把它们删掉。

5.2.2 boolean（布尔）

boolean 类型只有两个可选值：true（真） 和 false（假），碰到比较判断时，false 和 nil 都看作是 false，其他的都为 true，数字 0 也是 true。

```
if  false  or  nil  then
    ans = 0
else
    ans = 1      -- 这条语句会执行
end

if  0  then
    ans = 0      -- 这条语句会执行
else
    ans = 1
end
```

5.2.3 number（数字）

number 类型只有一种 float 类型，以下几种写法都被看作是 number 类型：

```
fv = 5
fv = 3.2
fv = 2e+1
fv = 0.6e-1
```

5.2.4 string（字符串）

字符串由一对双引号或单引号来表示。

```
str1 = "savch1"
str2 = 'savch2'
```

也可以用 2 个方括号 "[[]]" 来表示"一块"字符串。

```
str3 = [[
Hello savch!
Hello world!
]]
```

在对一个数字字符串上进行算术操作时，会尝试将这个数字字符串转成一个数字:

```
len1 = "2" + 6       --len1 的值为 8.0
len2 = "2" + "6"     --len2 的值为 8.0
```

字符串连接使用的是" .. " ，如：

```
str = "a" .. 'b'     --str 为"ab"
```

使用 # 来计算字符串的长度，放在字符串前面，如下实例：

```
str = "savch"
len = #str           -- len 的值为 5
```

5.2.5 table（表）

table 的创建是通过"构造表达式"来完成，最简单构造表达式是{}，用来创建一个空表。也可以在表里添加一些数据，直接初始化表:

```
local tbl1 = {}                -- 创建一个空的 table
local tbl2 = {"a", "b", "c"}   -- 直接初始表
```

table（表）其实是一个"关联数组"（associative arrays），数组的索引可以是数字或者是字符串。

```
a = {}
a["key"] = "value"         --给表 a 添加了元素"value"索引是"key"
a[6] = 55                  --给表 a 添加了元素 55 索引是 6
```

不同于其他语言的数组把 0 作为数组的初始索引，在 Lua 里表的默认初始索引一般以 1 开始。

```
tbl = {"a", "b", "c"}
print(tbl[0], tbl[1], tbl[3])   -- > nil a c，索引 0 对应的元素为 nil，索引 1 和 3 对应"a"和"c"
```

table 不会固定长度大小，有新数据添加时 table 长度会自动增长，没初始的 table 都是 nil。

5.2.6 function（函数）

函数被看作是"第一类值（First-Class Value）"，函数可以存在变量里:

```
function add(a, b)
    return a + b
end


add2 = add
sum = add2(2, 3)       -- sum 的值为 5
```

function 可以以匿名函数（anonymous function）的方式通过参数传递:

```
function testFun(a, fun)
    return a + fun(a)
end


res = testFun(6,       --res 的值为 6 + (6 + 2) = 14
function(v)            --匿名函数
```

```
    return v + 2;
end
);
```

5.3 变量

变量在使用前，需要在代码中进行声明，即创建该变量。

编译程序执行代码之前编译器需要知道如何给语句变量开辟存储区，用于存储变量的值。

Lua 变量有三种类型：全局变量、局部变量、表中的域。

Lua 中的变量全是全局变量，哪怕是语句块或是函数里，除非用 local 显式声明为局部变量。

局部变量的作用域为从声明位置开始到所在语句块结束。

变量的默认值均为 nil。

```
a = 5               -- 全局变量
local b = 5         -- 局部变量

function func()
    c = 5           -- 全局变量
    local d = 6     -- 局部变量，d 的作用域仅在 func 函数中
end

func()              -- 执行 func 函数
print(c,d)          --> 5  nil，此时 d 已经被释放

do
    local a = 6     -- 局部变量，作用域仅在此程序块
    b = 6           -- 对局部变量重新赋值
    print(a,b);     --> 6 6
end

print(a,b)          --> 5 6 ，此时的 a 为全局变量的 a
```

赋值是改变一个变量的值和改变表域的最基本的方法。

Lua 可以对多个变量同时赋值，变量列表和值列表的各个元素用逗号分开，赋值语句右边的值会依次赋给左边的变量。

```
a, b = 10, 2*x       -- a=10; b=2*x
```

遇到赋值语句 Lua 会先计算右边所有的值然后再执行赋值操作，所以我们可以这样进行交换变量的值：

```
x, y = y, x                     -- 交换 x 和 y 的值
a[i], a[j] = a[j], a[i]         -- 交换 a[i]和 a[j]的值
```

当变量个数和值的个数不一致时，若变量个数大于值的个数会按变量个数补足 nil，若变量个数小于值的个数则多余的值会被忽略。

```
a, b, c = 0, 1
print(a,b,c)             --> 0   1   nil
```

```
a, b = a+1, b+1, b+2     -- b+2 的值被忽略
print(a,b)               --> 1   2

a, b, c = 0
print(a,b,c)             --> 0   nil   nil
```

上面最后一个例子是一个常见的错误情况，注意：如果要对多个变量赋值必须依次对每个变量赋值。

多值赋值经常用来交换变量，或将函数调用返回给变量：

```
a, b = f()
```

f( )返回两个值，第一个赋给 a，第二个赋给 b。

应该尽可能的使用局部变量，有两个好处：1. 避免命名冲突。 2. 访问局部变量的速度比全局变量更快。

5.4 运算符

运算符是一个特殊的符号，用于告诉解释器执行特定的数学或逻辑运算。Lua 提供了以下几种运算符类型：算术运算符、关系运算符、逻辑运算符和其他运算符。

5.4.1 算术运算符

下表列出了 Lua 语言中的常用算术运算符，设定 A 的值为 10，B 的值为 20：

操作符	描述	实例
+	加法	A + B 输出结果 30
-	减法	A - B 输出结果 -10
*	乘法	A * B 输出结果 200
/	除法	B / A 输出结果 2
%	取余	B % A 输出结果 0
^	乘幂	A^2 输出结果 100
-	负号	-A 输出结果 -10

5.4.2 关系运算符

下表列出了常用关系运算符，设定 A 的值为 10，B 的值为 20：

操作符	描述	实例
==	等于，检测两个值是否相等，相等返回 true，否则返回 false	(A == B) 为 false
~=	不等于，检测两个值是否相等，不相等返回 true，否则返回 false	(A ~= B) 为 true
>	大于，如果左边的值大于右边的值，返回 true，否则返回 false	(A > B) 为 false
<	小于，如果左边的值大于右边的值，返回 false，否则返回 true	(A < B) 为 true
>=	大于等于，如果左边的值大于等于右边的值，返回 true，否则返回 false	(A >= B) 返回 false
<=	小于等于， 如果左边的值小于等于右边的值，返回 true，否则返回 false	(A <= B) 返回 true

5.4.3 逻辑运算符

下表列出了 Lua 语言中的常用逻辑运算符，设定 A 的值为 true，B 的值为 false：

操作符	描述	实例
and	逻辑与操作符。 若 A 为 false，则返回 A，否则返回 B。	(A and B) 为 false
or	逻辑或操作符。 若 A 为 true，则返回 A，否则返回 B。	(A or B) 为 true
not	逻辑非操作符。与逻辑运算结果相反，如果条件为 true，逻辑非为 false。	not(A and B) 为 true

5.4.4 其他运算符

下表列出了 连接运算符与计算表或字符串长度的运算符：

操作符	描述	实例
..	连接两个字符串	a..b，其中 a 为 "Hello "，b 为 "World",输出结果为 "Hello World"。
#	一元运算符，返回字符串或表的长度。	#"Hello" 返回 5

5.4.5 运算符优先级

运算符的优先级如下表所示，优先级数字越低，优先级越高：

优先级	运算符
1	**^**
2	**not - (unary)**
3	*** / %**
4	**+ -**
5	**..**
6	**< > <= >= ~= ==**
7	**and**
8	**or**

注：括号内的运算会独立执行，为了防止优先级出错，可以在编程时，通过增加括号来明确优先级。

5.5 循环

很多情况下我们需要做一些有规律性的重复操作，因此在程序中就需要重复执行某些语句。

一组被重复执行的语句称之为循环体，能否继续重复，决定循环的终止条件。

循环结构是在一定条件下反复执行某段程序的流程结构，被反复执行的程序被称为循环体。

循环语句是由循环体及循环的终止条件两部分组成的。

Lua 语言提供了以下几种循环处理方式：

循环类型	描述
while 循环	在条件为 true 时，让程序重复地执行某些语句。执行语句前会先检查条件是否为 true。
for 循环	重复执行指定语句，重复次数可在 for 语句中控制。
repeat...until	重复执行循环，直到指定的条件为真时为止
循环嵌套	可以在循环内嵌套一个或多个循环语句

5.5.1 while 循环

while 循环语句在判断条件为 true 时会重复执行循环体语句，语法格式如下：

```
while(条件) do
   <循环体>
end
```

循环体语句可以是一条或多条语句，条件 可以是任意表达式，在 条件 为 true 时执行循环体语句。

```
a=0
while( a < 3 ) do
   a = a+1           -- 循环结束时，a 的值为 3
end
```

5.5.2 for 循环

for 循环语句可以重复执行指定语句，重复次数可在 for 语句中控制，语法格式如下:

```
for var=exp1, exp2, exp3 do
     <循环体>
end
```

var 从 exp1 变化到 exp2，每次变化以 exp3 为步长递增，并执行一次 <循环体>。exp3 是可选的，如果不指定，默认为 1。exp1、epx2 和 exp3 都可以是表达式或者函数，且都是在循环开始前一次性求值，将计算得到的值用于循环判断，在之后的循环中不再进行求值运算。

```
a = 0
for i=1, 3, 1 do
    a = a + i        -- 循环结束时，a 的值为 0 +  1 + 2 + 3 = 6
end
```

5.5.3 repeat...until 循环

repeat...until 和 while 循环类似，但是 repeat...until 循环的条件语句在当前循环结束后判断。

repeat...until 循环语法格式：

```
repeat
    <循环体>
until( 条件 )
```

循环条件判断语句在循环体末尾部分，所以在条件进行判断前循环体都会执行一次。

如果条件判断语句为 false，循环会重新开始执行，直到条件判断语句为 true 才会停止执行。

```
a = 0
repeat
    a = a + 1       -- 循环结束时，a 的值为 4
until( a > 3 )
```

5.5.4 循环嵌套

Lua 编程语言中允许循环中嵌入循环。

除了可以使用同类型循环嵌套外，我们还可以使用不同的循环类型来嵌套，如 for 循环体中嵌套 while 循环。

5.5.5 循环控制语句

循环控制语句用于控制程序的流程，以实现程序的各种结构方式。

Lua 支持以下循环控制语句：

控制语句	描述
break 语句	退出当前循环或语句，并开始脚本执行紧接着的语句。
goto 语句	将程序的控制点转移到一个标签处。

break 语句

Lua 编程语言 break 语句插入在循环体中，用于退出当前循环或语句，并开始脚本执行紧接着的语句。

如果你使用循环嵌套，break 语句将停止最内层循环的执行，并开始执行的外层的循环语句。

以下实例执行 while 循环，在变量 a 小于 20 时输出 a 的值，并在 a 大于 15 时终止执行循环：

```
a = 10
while( a < 20 ) do
   a = a + 1
   if( a > 15) then
      break                --使用 break 语句终止循环，while 循环执行完成后，a 的值为 16
   end
end
```

goto 语句

goto 语句允许将控制流程无条件地转到被标记的语句处，语法格式如下所示：

```
goto Label
```

Label 的格式为：

```
:: Label ::
```

以下实例在判断语句中使用 goto：

```
local a = 1
local b = 0
::label:: b = b + 1
```

```
a = a+1
if a < 10 then
    goto label              -- a 小于 10 的时候跳转到标签 label,当整段语句执行完成时，b 的值等与 9
end
```

注：label 标签处的语句只要运行到该位置就会执行，而不是在 goto 语句跳转后才执行。

5.5.6 无限循环

在循环体中如果条件永远为 true 循环语句就会永远执行下去，以下以 while 循环为例：

```
while( true )
do
   print("循环将永远执行下去")
end
```

5.6 流程控制

Lua 编程语言流程控制语句通过程序设定一个或多个条件语句来设定。在条件为 true 时执行指定程序代码，在条件为 false 时执行其他指定代码。控制结构的条件表达式结果可以是任何值，其中 false 和 nil 为假，true 和非 nil 为真，要注意的是 0 也为 true。

Lua 提供了以下控制结构语句：

语句	描述
if 语句	**if 语句** 由一个布尔表达式作为条件判断，其后紧跟其他语句组成。
if...else 语句	**if 语句** 可以与 **else 语句**搭配使用,在 if 条件表达式为 false 时执行 else 语句代码。
if...elseif...else 语句	你可以在 **if** 或 **else if** 中使用一个或多个 **if** 或 **else if** 语句 。
if 嵌套语句	你可以在 **if** 或 **else if** 中使用一个或多个 **if** 或 **else if** 语句 。

5.6.1 if 语句

if 语句 由一个布尔表达式作为条件判断，其后紧跟其他语句组成，语法格式如下：

```
if(布尔表达式)then
   --[ 在布尔表达式为 true 时执行的语句 --]
end
```

例，以下实例用于判断变量 a 的值是否小于 20：

```
a = 10
b = 0
if( a < 20 ) then
      b = 10          -- a 小于 20, 所以 b 等于 10
end
```

5.6.2 if...else 语句

if 语句可以与 else 语句搭配使用，在 if 条件表达式为 false 时执行 else 语句代码块，语法格式如下：

```
if(布尔表达式) then
   --[ 布尔表达式为 true 时执行该语句块 --]
else
   --[ 布尔表达式为 false 时执行该语句块 --]
end
```

在布尔表达式为 true 时会 if 中的代码块会被执行，在布尔表达式为 false 时，else 的代码块会被执行。

以下实例用于判断变量 a 的值：

```
a = 100;
b = 0
if( a < 20 ) then
   b = 10
else
   b = 20              -- 由于 a 大于 100，所以执行 b = 20，最后 b 的值为 20
end
```

5.6.3 if...elseif...else 语句

if 语句可以与 elseif...else 语句搭配使用，在 if 条件表达式为 false 时执行 elseif...else 语句代码块，用于检测多个条件语句，语法格式如下：

```
if( 布尔表达式 1)then
   --[ 在布尔表达式 1 为 true 时执行该语句块 --]
elseif( 布尔表达式 2)then
   --[ 在布尔表达式 2 为 true 时执行该语句块 --]
elseif( 布尔表达式 3)then
   --[ 在布尔表达式 3 为 true 时执行该语句块 --]
else
   --[ 如果以上布尔表达式都不为 true 则执行该语句块 --]
end
```

以下实例对变量 a 的值进行判断：

```
a = 100
b = 0
if( a == 10 ) then
   b = 1
elseif( a == 50 ) then
   b = 2
elseif( a == 100 ) then
   b = 3                 --由于 a = 100,因此该条语句被执行，b 的值为 3
else
   b = 4
end
```

5.6.4 if 嵌套语句

if 语句允许嵌套，这就意味着你可以在一个 if 或 else if 语句中插入其他的 if 或 else if 语句，语法格式如下：

```
if( 布尔表达式 1)then
   --[ 布尔表达式 1 为 true 时执行该语句块 --]
   if(布尔表达式 2) then
      --[ 布尔表达式 2 为 true 时执行该语句块 --]
   end
end
```

你可以用同样的方式嵌套 else if...else 语句。

```
a = 100
b = 200
c = 0
if( a == 100 ) then
   c = 10
   if( b == 200 ) then
      c = 20                --由于 a 和 b 同时满足，固 c=20 语句被执行，c 的值为 20
   end
end
```

5.7 函数

函数是对语句和表达式进行抽象的主要方法。既可以用来处理一些特殊的工作，也可以用来计算一些值。

函数主要有两种用途：

1.完成指定的任务，这种情况下函数作为调用语句使用；

2.计算并返回值，这种情况下函数作为赋值语句的表达式使用。

5.7.1 函数定义

编程语言函数定义格式如下：

```
选项  function  函数名称(参数 1, 参数 2, 参数 3, ..., 参数 n)

    函数体

    return 返回值
end
```

选项：用于指定函数是全局函数还是局部函数，未设置该参数默认为全局函数，指定为局部函数需要使用关键字 local。

函数名称 ：指定函数名称。

参数 1~n ：函数参数，多个参数以逗号隔开，函数也可以不带参数，参数可以是常数、表达式或者函数。

函数体：函数中需要执行的代码语句块。

返回值：执行完函数返回的值，可以返回多个值，每个值以逗号隔开，也可以不返回值。

以下实例定义了函数 max()，参数为 num1, num2，用于比较两值的大小，并返回最大值：

```
function max(num1, num2)
   if (num1 > num2) then
      result = num1;
   else
      result = num2;
   end
   return result;
end
print("最大值为 ",max(10,4))   -->打印 “最大值为 10”
print("最大值为 ",max(5,6))    -->打印 “最大值为 6”
```

5.7.2 多返回值

函数可以返回多个结果值，在 return 后列出要返回的值的列表即可返回多值，如：

```
function ret2(v)
    return v+2, v+3
end

a,b = ret2(3)        -- a = 5, b = 6
```

5.8 数组

数组，就是相同数据类型的元素按一定顺序排列的集合，可以是一维数组和多维数组。

数组的索引键值可以使用整数表示，数组的大小不是固定的。

5.8.1 一维数组

一维数组是最简单的数组，其逻辑结构是线性表。一维数组可以用 for 循环出数组中的元素，我们可以使用整数索引来访问数组元素，如果知道的索引没有值则返回 nil。

数组默认索引值是以 1 为起始，但你也可以指定 0 开始。

除此外我们还可以以负数为数组索引值：

```
array = {}

for i= -2, 2 do
   array[i] = i *2
end

for i = -2,2 do
   print(array[i])        --打印输出 -4 -2 0 2 4
end
```

5.8.2 多维数组

多维数组即数组中包含数组或一维数组的索引键对应一个数组。

以下是一个三行三列的阵列多维数组：

```
array = {}               -- 初始化数组
for i=1,3 do
   array[i] = {}
      for j=1,3 do
         array[i][j] = i*j
      end
end

for i=1,3 do
   for j=1,3 do
      print(array[i][j])   --打印输出结果 > 1 2 3 2 4 6 3 6 9
   end
end
```

以上的实例中，数组设定了指定的索引值，这样可以避免出现 nil 值，有利于节省内存空间。

5.9 table(表)

table 是一种数据结构用来帮助我们创建不同的数据类型，如：数组、字典等。

table 使用关联型数组，你可以用任意类型的值来作数组的索引，但这个值不能是 nil。

table 是不固定大小的，你可以根据自己需要进行扩容。

5.9.1 table(表)的构造

构造器是创建和初始化表的表达式。最简单的构造函数是{ }，用来创建一个空表。

```
mytable = {}          -- 初始化表
mytable[1]= 123       -- 指定值
mytable = nil         -- 移除引用，垃圾回收会释放内存
```

当我们为 table a 设置元素，然后将 a 赋值给 b，则 a 与 b 都指向同一个内存。如果 a 设置为 nil ，则 b 同样能访问 table 的元素。如果没有指定的变量指向 a，垃圾回收机制会清理相对应的内存。

5.9.2 Table 操作

Table 连接

函数原型：table.concat (table [, sep [, start [, end]]]):

table.concat()函数列出参数中指定 table 的数组部分从 start 位置到 end 位置的所有元素，元素间以指定的分隔符(sep)隔开。

```
tbl= {1,2,3,4,5}
a = table.concat(tbl)           -- 表 tbl 内所有数据连接后为 123，所以 a = 12345
a = table.concat(tbl,8)         --表 tbl 以 8 为分隔符连接后为 18283，所以 a = 182838485
a = table.concat(tbl,’’, 3)     --表 tbl 以空字符连接表中第 3 到最后一个数据后为 345，所以 a = 345
a = table.concat(tbl,’’, 2,4)   --表 tbl 以空字符连接表中第 2 到第 4 个数据后为 234，所以 a = 234
```

插入和移除

插入函数原型：table.insert (table, [pos,] value):

在 table 的数组部分指定位置(pos)插入值为 value 的一个元素. pos 参数可选, 默认为数组部分末尾.

移除函数原型：table.remove (table [, pos])

返回 table 数组部分位于 pos 位置的元素. 其后的元素会被前移. pos 参数可选, 默认为 table 长度, 即从最后一个元素删起。

```
tbl = {1,2,3}
table.insert(tbl,4)        --在末尾插入 4，tbl={1,2,3,4}
table.insert(tbl,2,20)     -- 在索引为 2 的键处插入 20,tbl={1,20,2,3,4}
table.remove(tbl)          --移除最后一个元素，tbl={1,20,2,3}
table.remove(tbl,2)        --移除索引为 2 的元素，tbl={1,2,3}
```

Table 排序

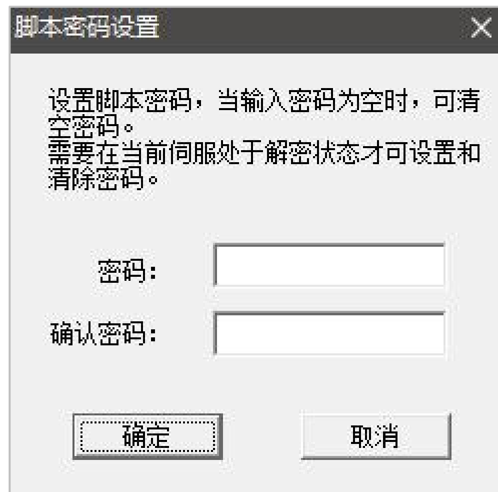
函数原型：table.sort (table [, comp])

对给定的 table 进行升序排序。

```
tbl = {2,10,5,20,7,62,1}
table.sort(tbl)            -- sort()函数执行后，tbl 中的数据变为{1,2,5,7,10,20,62}
```

六、 加密与解密

在“工具”菜单栏中，选择“设置脚本密码”可以打开脚本密码设置对话框。



输入密码后，会将伺服内脚本加密，对脚本进行上载和下载操作前，均需要执行解密操作。

解密操作执行后，伺服会一直处于解密状态，直到伺服重启或者重新设置密码。

- 创无限 | 赢久远
- 工业智能 | 节能 | 绿色电能

三基微信服务号

生产总部

泉州市鲤城区江南高新园区紫新路 3 号

电话：0595-24678267　　传真：0595-24678203

服务网络

客服电话：400-6161-619　　网址：www.savch.net

已获资质

ISO9001 体系认证及 CE 产品认证

版权所有，侵权必究！如有改动，恕不另行通知！

销售服务联络地址