

平台脚本使用说明

版本说明

1. V1.0.01 2018/12/12 logos create
2. V2.0.01 2020/5/6 logos update

平台自 1.7.01 版本以后增加简单脚本功能，脚本以节点运行方式加入到平台中，主要实现以下功能：

1. 条件判断：使用 if 语句判断条件是否成立，如：if(a+b<10) a++配合平台跳转和循环功能，能适应大部分逻辑控制。
2. 循环控制 while(x < 10) { x++ y++ }
3. 算术运算：可实现大部分算术指令，+、-、*、/等等，支持括号优先级运算。如：a = (b+c)*(b-c)
4. 脚本直接使用平台全局变量进行运算及操作，不支持自定义局部变量。
5. 支持语法检查，新建或修改脚本时，会先检查语法是否通过，但因为变量是泛型，所以脚本在运行时可能会报错，因为不同的运算符支持的类型限制有所不同。

控制语法

脚本执行器目前支持条件控制语句，分别用于条件控制和循环控制。

1. if() do... 判断执行语句，当()内条件成立时执行后面的语句，若后面有多条执行语句，则使用{}包含起来。
2. elseif() do... 判断执行语句，与 if 语句配合使用，当()内条件成立时执行后面的语句，若后面有多条执行语句，则使用{}包含起来。
3. else do... 判断执行语句，与 if 语句配合使用，当前面的 if/elseif 语句不成立时，执行后面的语句，若后面有多条执行语句，则使用{}包含起来。
4. while() do... 条件循环语句，当()内条件成立时，重复执行后面的语句，若后面有多条执行语句，则使用{}包含起来。
5. break/return 条件中断语句，break 只适用于 while 循环内部，return 则会直接结束该脚本的执行。

常量

脚本执行器目前有定义几个关键字为常量，如下：

1. PI 圆周率 3.14159265358979323846
2. MIN 整型最小值 -2147483648
3. MAX 整型最大值 2147483647

变量操作

变量不支持在脚本中定义，只能在平台变量设置界面配置，变量名只能是英文字母+下划线+数字组成，且变量首字母不能是数字。变量没有类型，在设置变量时，变量管理模块会自动判定类型。目前变量支持的类型只有 3 种，字符串、整型和浮点型(小数)。

ID	名称	描述	初始值	当前值
101	debug_msg	OPPS调试专用	0	0
102	warn_r0	流程报警专用	0	0
103	warn_p11	流程报警专用	0	0
104	a	测试	0	0
105	b	测试	0	0
106	_t	test	0	0
*				

脚本支持的运算符

1. ‘++’ ‘--’ 运算符完成变量自增或自减 1 的操作，目标变量必须是整型，否则运行时 would 报错。该运算符如果在一段表达式中，那么会先计算该表达式值，然后再完成自增或自减的运算。如：a = b-- 可以看成是两条语句 a = b b = b - 1
2. ‘=’ 赋值运算符，把‘=’号之后的表达式值赋给‘=’号之前的变量，赋值运算符前面只能是变量，‘=’运算符支持串行输入。如：a = b = c++
3. ‘+=’ ‘-=’ ‘*=’ ‘/=’ 运算符，该类运算符其实是‘=’运算符的加强版，同‘=’运算符一样，前面必须是变量，在赋值给变量之前，先计算表达式的值，然后再用变量本身与结果做+、*、-、/ 运算，再赋值给变量。
4. ‘+’ 算术运算符，对两个表达式或变量进行加法处理，前提是两个表达式或变量的值必须是整数或小数。其中要注意的是字符串可以用于加法运算，但必须保证两个操作数都是字符串。
5. ‘-’ 算术运算符，两个操作数都必须是数字，进行减法运算，也可作为前置运算符。
6. ‘*’ 算术运算符，两个操作数都必须是数字，进行乘法运算。
7. ‘/’ 算术运算符，两个操作数都必须是数字，进行除法运算。
8. ‘%’ 算术运算符，两个操作数都必须是整数，进行取余运算。
9. ‘&&’ 逻辑运算符，一般用于 if/while 语句内部，用于逻辑与运算。如：if (a < 1 && a > -1) b=0，表达的意思是如果 a 小于 1 并且 a > -1 那么条件成立，执行 b=0 操作。

10. '||' 逻辑运算符，一般用于 if 语句内部，用于逻辑或运算。如：if (a < -1 || a > 1) b=0，表达的意思是如果 a 小于 -1 或者 a > 1 那么条件成立，执行 b=0 操作。
11. '==' 比较运算符，一般用于 if 语句内部，用于判断两个表达式或变量是否相等。如：if (a==b) c=0，表达的意思是如果 a 和 b 相等 那么条件成立，执行 c=0 操作。
12. '!=' 比较运算符，一般用于 if 语句内部，用于判断两个表达式或变量是否不相等。如：if (a!=b) c=0，表达的意思是如果 a 和 b 不相等 那么条件成立，执行 c=0 操作。
13. '<' 比较运算符，一般用于 if 语句内部，用于判断表达式 1 或变量 1 是否小于表达式 2 或变量 2 的值。如：if (a<b) c=0，表达的意思是如果 a 小于 b 那么条件成立，执行 c=0 操作。
14. '<=' 比较运算符，一般用于 if 语句内部，用于判断表达式 1 或变量 1 是否小于或等于表达式 2 或变量 2 的值。如：if (a<=b) c=0，表达的意思是如果 a 小于或等于 b 那么条件成立，执行 c=0 操作。
15. '>' 比较运算符，一般用于 if 语句内部，用于判断表达式 1 或变量 1 是否大于表达式 2 或变量 2 的值。如：if (a>b) c=0，表达的意思是如果 a 大于 b 那么条件成立，执行 c=0 操作。
16. '>=' 比较运算符，一般用于 if 语句内部，用于判断表达式 1 或变量 1 是否大于或等于表达式 2 或变量 2 的值。如：if (a>=b) c=0，表达的意思是如果 a 大于或等于 b 那么条件成立，执行 c=0 操作。
17. '&' '|' '^' 算术运算符 两个操作数都必须是整数，进行按位与运算、或运算、异或运算。
18. '<<' '>>' 算术运算符，两个操作数都必须是整数，按位左移或右移操作。
19. '!' 算术运算符，前置运算符，操作数必须是整数，进行取反操作，计算结果为 1 或 0。
20. '~' 算术运算符，前置运算符，操作数必须是整数，进行按位取反操作。
21. ',' 函数内部间隔参数运算符，用于间隔多个参数，仅限函数内部使用。
22. '"' 双引号，用于字符串表达式，字符串必须用双引号包括起来。

运算符优先级

脚本解析器支持的运算符优先级数字越小，优先级越高。

1. () {} 任何一个表达式用括号括起来，优先级是最高的。
2. ! - ++ -- ~ 这些作为前置运算符，优先级仅次于()。
3. * / % 算术运算符
4. + - 算术运算符
5. << >> 按位左移/右移运算符
6. < <= > >= 逻辑判断运算符
7. & 按位与计算
8. ^ 按位异或
9. | 按位或计算
10. && 逻辑与判断
11. || 逻辑或判断
12. = += -= *= /= 赋值运算符 从右到左依次运算

字符串操作

在脚本执行器中，对字符串有简单的操作

1. 表达方式，常量字符串必须在首尾用两个“”括起来。

2. 字符串支持赋值操作，支持加法操作，直接用运算符即可，如：

```
a = "hello" b = " world" c = a+b
```

运算后 c 结果为"hello world"。

3. 双引号内支持转义符操作 ‘\’，目前仅支持‘\r’ ‘\n’ ‘\t’ ‘\’ 如：

```
a = "hello\n" b = "hello\r"
```

4. 支持字符串首尾去掉特定字符，trim 函数可完成此功能。如：

```
a = " hello world \n" trim(a," \n")
```

同时去掉空格和换行符‘\n’
运行后 a 结果为"hello world"

5. 支持字符串分割操作, split 函数可完成此功能。如：

```
a = "hello world" b = split(a, " ")
```

把 a 按空格分割，第一个空格之前的内容装入 b 变量，
后面的内容留下保存在 a 变量

运行后结果为： a = "world" b = "hello"

脚本支持的函数

脚本引擎支持内部函数，函数的表达方式为 fun(p1,p2)，其中 fun 表示函数名称， p1,p2... 表示参数列表，参数之间用‘,’隔开，需要注意的是有些函数参数在语法检查时无法发现错误，仅在运行时才可能报错。函数调用需要注意的是，若入参为 double，则可以传入 int，若入参为 int，则只能传入 int，若入参为 string，则只能传入 string，否则会报错。

1. ‘if/elseif/else’ 用于判断一个条件表达式是否成立。表达式返回的值必须是整数，为 0 表示不成立，非 0 表示成立。若成立，则继续执行后续的表达式，若不成立，则不执行。

如：if(a<1) a++ 若 a<1 成立，则执行 a++操作，相反的，若条件不成立，则不执行 a++。
若后续语句为多段语句，则可使用{}包含起来，如：if(a<1) { a++ b++}。

2. ‘while’ 用于循环执行一个条件表达式，直到条件不成立。条件表达式返回的值必须是整数，为 0 表示不成立，非 0 表示成立。执行方式同 if 语句一致。

3. ‘str’ 算术函数，用于把操作数强转为字符串类型

函数原型 string str(void s[, int len])

输入参数：s 整型/浮点型/字符串

len 整型，可选参数，s 为整型标识返回的最小长度，s 为浮点型标识精度。

返回值： string 返回转换后的字符串。

4. ‘sin’ 三角函数，用于求正弦值。

函数原型 double sin(double s)

输入参数：s 弧度值角度

返回值： double 正弦值

5. 'cos' 三角函数，用于求余弦值。
函数原型 `double cos(double s)`
输入参数: `s` 弧度值角度
返回值: `double` 余弦值
6. 'tan' 三角函数，用于求正切值。
函数原型 `double tan(double s)`
输入参数: `s` 弧度值角度
返回值: `double` 正切值
7. 'asin' 三角函数，用于求反正弦值。
函数原型 `double asin(double s)`
输入参数: `s` 正弦值
返回值: `double` 弧度值角度
8. 'acos' 三角函数，用于求反余弦值。
函数原型 `double acos(double s)`
输入参数: `s` 余弦值
返回值: `double` 弧度值角度。
9. 'atan' 三角函数，用于求反正切值。
函数原型 `double atan(double s)`
输入参数: `s` 正切值
返回值: `double` 弧度值角度
10. 'sqrt' 算术函数，用于求一个数的平方根。
函数原型 `double sqrt(double s)`
输入参数: `s` 浮点型数字
返回值: `double` 平方根
11. 'abs' 算术函数，用于求一个数的绝对值。
函数原型 `double abs(double s)`
输入参数: `s` 整型或浮点型数字
返回值: `double` 整型或浮点型数字的绝对值
12. 'trim' 字符串处理函数，去除字符串首尾的特定字符。
函数原型 `string trim(string v, string dot)`
输入参数: `v` 变量名，待操作字符串，去除特定字符后会保存在这个变量
`dot` 要去除的特定字符串合集，可以是单个字符，也可以是多个
返回值: `string` 返回去除后的字符串
13. 'split' 字符串处理函数，用于分割字符串。
函数原型 `string split(string v, string dot)`
输入参数: `v` 变量名，待分割字符串，分割后，右边字符串也会保存在这个变量
`dot` 字符串，在 `v` 中从左往右查找这个字符串，进行分割。
返回值: `string` 变量 `v` 中，以 `dot` 为分割点，返回左边字符串
14. 'config' 平台内部函数，重载函数，用于获取配置项。
函数原型 `var config(string key, string name)`
输入参数: `key` 字符串 <32 字节
`name` 字符串 <32 字节
返回值: 根据配置项内容返回 `int/double/string`，配置项不存在返回空。

15. 'config' 平台内部函数，重载函数，用于写入一个配置项。
函数原型 `int config(string key, string name,int/double/string val)`
输入参数: `key` 字符串 <32 字节
 `name` 字符串 <32 字节
 `val` `int/double/string` 任意一种类型 <32 字节
返回值: `int` 成功返回 0，失败返回错误代码
16. 'sconfig' 平台内部函数，重载函数，用于获取系统配置项。
函数原型 `var sconfig(string key, string name)`
输入参数: `key` 字符串 <32 字节
 `name` 字符串 <32 字节
返回值: 根据配置项内容返回 `int/double/string`，配置项不存在返回空。
17. 'sconfig' 平台内部函数，重载函数，用于写入一个系统配置项。
函数原型 `int sconfig(string key, string name,int/double/string val)`
输入参数: `key` 字符串 <32 字节
 `name` 字符串 <32 字节
 `val` `int/double/string` 任意一种类型 <32 字节
返回值: `int` 成功返回 0，失败返回错误代码
18. 'year' 获取本地时间-年份。
函数原型 `int year()`
输入参数: `NULL`
返回值: `int` 返回当前年份。
19. 'month' 获取本地时间-月份。
函数原型 `int month()`
输入参数: `NULL`
返回值: `int` 返回当前月份。
20. 'day' 获取本地时间-每个月的几号。
函数原型 `int day()`
输入参数: `NULL`
返回值: `int` 返回当前日期。
21. 'week' 获取本地时间-星期几?。
函数原型 `int week()`
输入参数: `NULL`
返回值: `int` 返回当前周几。
22. 'hour' 获取本地时间-小时。
函数原型 `int hour()`
输入参数: `NULL`
返回值: `int` 返回当前时间。
23. 'minute' 获取本地时间-分钟。
函数原型 `int minute()`
输入参数: `NULL`
返回值: `int` 返回当前时间。

24. 'second' 获取本地时间-秒。
函数原型 `int second()`
输入参数: `NULL`
返回值: `int` 返回当前时间。
25. 'msecond' 获取本地时间-毫秒。
函数原型 `int msecond()`
输入参数: `NULL`
返回值: `int` 返回当前时间。
26. 'tick' 获取当前系统 CPU 时钟。
函数原型 `int tick()`
输入参数: `NULL`
返回值: `int` 返回 CPU 时钟 (毫秒)。
27. 'rpos' 平台内部函数, 用于获取单轴当前默认位置。
函数原型 `double rpos(int axis)`
输入参数: `axis` 整型 轴 ID
返回值: `double` 返回轴当前位置
28. 'rposf' 平台内部函数, 用于获取单轴当前规划位置。
函数原型 `double rposf(int axis)`
输入参数: `axis` 整型 轴 ID
返回值: `double` 返回轴当前规划位置
29. 'rio' 平台内部函数, 用于读取 IO 值。若 IO 不存在, 则返回 0
函数原型 `int rio(string name)`
输入参数: `name` 字符串 IO 名称或 ID
返回值: `IO` 当前值, 1 或 0
30. 'wio' 平台内部函数, 用于写入 IO 值。
函数原型 `int wio(string name, int val)`
输入参数: `name` 字符串 IO 名称或 ID
`val` 整型 1 或 0
返回值: `int` 成功返回 0, 失败返回错误代码
31. 'rad' 平台内部函数, 用于读取 AD 值。若 AD 不存在, 则返回 0
函数原型 `int ad(string name)`
输入参数: `name` 字符串 AD 名称或 ID
返回值: `double` AD 当前值
返回值: `int` 成功返回 0, 失败返回错误代码
32. 'wad' 平台内部函数, 用于写入 AD 值。
函数原型 `int wad(string name, double val)`
输入参数: `name` 字符串 IO 名称或 ID
`val` 浮点型电压值
返回值: `int` 成功返回 0, 失败返回错误代码
33. 'rstatus' 平台内部函数, 用于获取某个对象状态, 对象包含卡/工站/视觉/轴。
函数原型 `int rstatus(int pid)`
输入参数: `pid` 整型 卡/工站/视觉/轴 ID
返回值: 0 未初始化 1 报警 2 就绪 3 未回原 4 停止 5 运动中 6 回原中 7 暂停中 8 正在暂停 9 正在停止

34. 'init' 平台内部函数，用于初始化一个对象，对象包含卡/工站/视觉。
函数原型 `int init(int pid)`
输入参数: `pid` 整型 卡/工站/视觉
返回值: `int` 成功返回 0，失败返回错误代码。
35. 'deinit' 平台内部函数，用于反初始化一个对象，对象包含卡/工站/视觉。
函数原型 `int deinit(int pid)`
输入参数: `pid` 整型 卡/工站/视觉
返回值: `int` 成功返回 0，失败返回错误代码。
36. 'stop' 平台内部函数，用于停止一个工站运动。
函数原型 `int stop(int tid)`
输入参数: `tid` 整型 工站 ID
返回值: `int` 成功返回 0，失败返回错误代码。
37. 'reset' 平台内部函数，用于重置一个工站状态。
函数原型 `int reset(int tid)`
输入参数: `tid` 整型 工站 ID
返回值: `int` 成功返回 0，失败返回错误代码。
38. 'pause' 平台内部函数，用于暂停一个工站运动。
函数原型 `int pause(int tid)`
输入参数: `tid` 整型 工站 ID
返回值: `int` 成功返回 0，失败返回错误代码。
39. 'continue' 平台内部函数，用于恢复一个已暂停工站的运动。
函数原型 `int continue(int tid)`
输入参数: `tid` 整型 工站 ID
返回值: `int` 成功返回 0，失败返回错误代码。
40. 'send' 平台内部函数，用于发送自定义消息给一个对象，对象包含卡/工站/视觉。
函数原型 `int send(int pid,string msg)`
输入参数: `pid` 整型 卡/工站/视觉
 `msg` 字符串 消息内容
返回值: `int` 成功返回 0，失败返回错误代码。
41. 'recv' 平台内部函数，用于接收一个对象的自定义消息，对象包含卡/工站/视觉。
函数原型 `int recv(int pid,string msg,int timeout)`
输入参数: `pid` 整型 卡/工站/视觉
 `msg` 字符串 过滤字段或参数
 `timeout` 整型 超时时间
返回值: 成功返回消息内容，失败返回错误代码。
如: `a = recv(101,"",50)`
42. OPS 系统控制，直接在“系统流程”中控制变量可以达到控制效果。
系统启动: `ops_start`
系统复位: `ops_reset`
系统急停: `ops_estop`
系统暂停: `ops_pause`
系统停止: `ops_stop`
系统状态反馈: `ops_status`

43. 'jump' 平台内部函数，用于控制 ops 流程跳转。函数返回失败，流程会报警。

函数原型 `int jump(int id)`

输入参数: id 整型，步骤 ID/节点 ID

返回值: int 成功返回 0，失败返回错误代码。

44. 'print' 平台内部函数，用于上报日志给 UI。

函数原型 `void print(string msg)`

输入参数: msg 字符串 消息内容

返回值: NULL

45. 'pstatus' 平台内部函数，用于获取某个流程的状态。

函数原型 `int pstatus(int proclD)`

输入参数: proclD 整型 流程 ID

返回值: 0 就绪 1 运行中 2 暂停 3 停止 4 异常。

46. 'pstart' 平台内部函数，用于启动某个流程。

函数原型 `int pstart(int proclD)`

输入参数: proclD 整型 流程 ID

返回值: int 成功返回 0，失败返回错误代码。

47. 'pstop' 平台内部函数，用于手动停止某个流程。

函数原型 `int pstop(int proclD)`

输入参数: proclD 整型 流程 ID

返回值: int 成功返回 0，失败返回错误代码。

*PS: 调用此接口后流程状态会变成“停止”。

48. 'preset' 平台内部函数，用于重置某个流程。

函数原型 `int preset(int proclD)`

输入参数: proclD 整型 流程 ID

返回值: int 成功返回 0，失败返回错误代码。

*PS: 调用此接口后流程会停止运行，且状态会变成“就绪”。

49. 'ppause' 平台内部函数，用于暂停某个流程运行。

函数原型 `int ppause(int proclD)`

输入参数: proclD 整型 流程 ID

返回值: int 成功返回 0，失败返回错误代码。

50. 'pcontinue' 平台内部函数，用于恢复已暂停的流程运行。

函数原型 `int pcontinue(int proclD)`

输入参数: proclD 整型 流程 ID

返回值: int 成功返回 0，失败返回错误代码。

51. 'work_set' 平台内部函数，设置工作盘当前运行节点序号。

函数原型 `int work_set(int mid,int index)`

输入参数: mid 整型 工作盘 ID

index 整型 节点序号

返回值: int 成功返回 0，失败返回错误代码。

52. 'work_get' 平台内部函数，获取指定工作盘当前运行节点序号。

函数原型 `int work_get (int mid)`

输入参数: mid 整型 工作盘 ID

返回值: int 返回工作盘当前运行节点序号，0 标识不在运行中。

53. 'work_get_posx' 平台内部函数，获取指定工作盘当前运行节点位置 X。

函数原型 `int work_get_posx(int mid)`

输入参数: mid 整型 工作盘 ID

返回值: double 返回当前工作盘运行节点的 X 轴位置。

54. 'work_get_posy' 平台内部函数，获取指定工作盘当前运行节点位置 Y。

函数原型 `int work_get_posy(int mid)`

输入参数: mid 整型 工作盘 ID

返回值: double 返回当前工作盘运行节点的 Y 轴位置。

55. 'work_map_pos' 平台内部函数，三点矩阵映射出指定工作盘所有位置信息。

函数原型 `int work_map_pos(int mid,double x1,double y1,double x2,double y2,double x3,double y3)`

输入参数: mid 整型 工作盘 ID

x1 y1 起始点位置

x2 y2 列终点位置

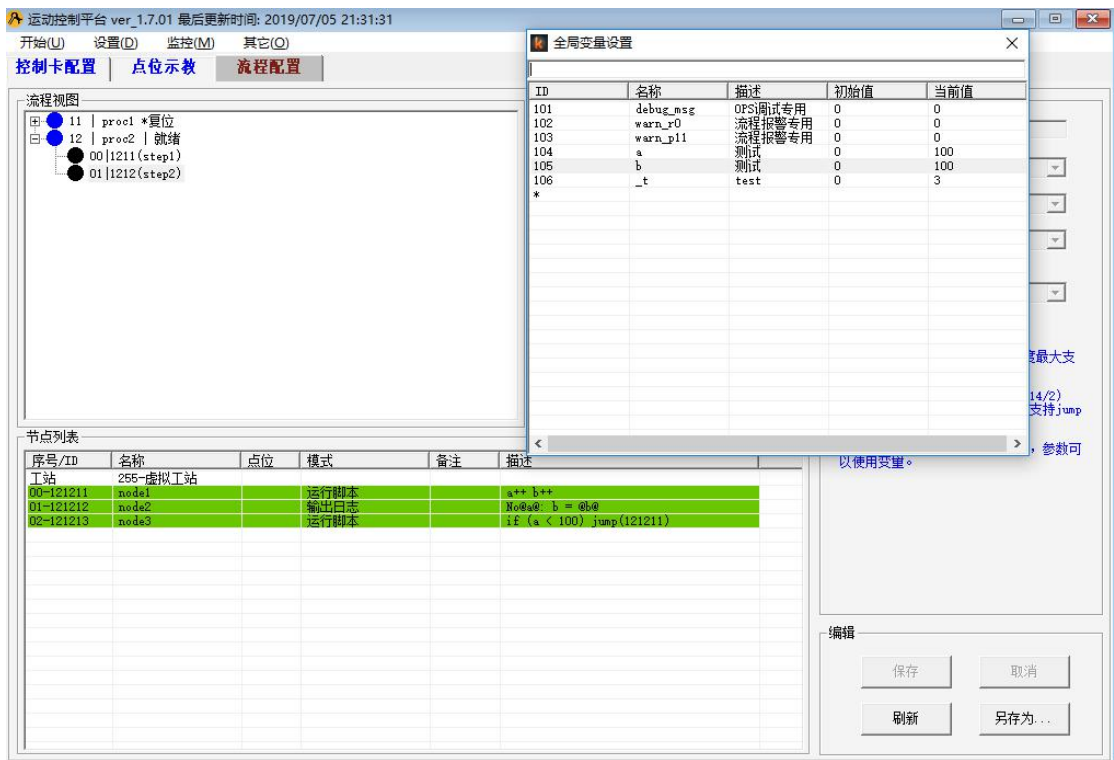
x3 y3 行终点位置

返回值: int 成功返回 0，失败返回错误代码。

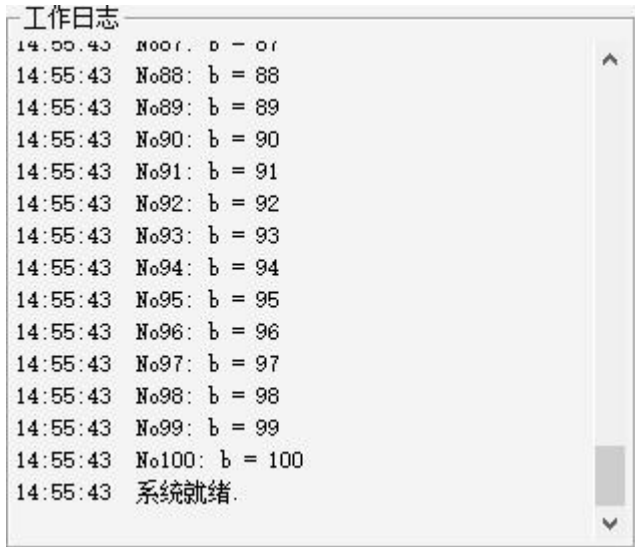
PS: 设置仅当前有效，在重启程序或修改工作盘后，位置信息会被覆盖

应用示例

循环打印 1-100 的数字



运行结果:



使用小技巧

1. 一条简单脚本中，只能有一个 if 语句，但是在 if 语句前可以有执行语句。
如：`a++ if(a<10) jump(123)`
首先 `a++` 是一定会执行的，然后判断 `a<10` 是否成立，若成立，则执行 `jump`
2. `if/while` 语句后面可以跟多条执行语句
如：`a++ if(a<10) {b = 0 jump(123)} b++`
`a++ b++` 一定会执行，判断 `a<10` 是否成立，若成立，则执行 `b=0` 和 `jump`，若条件不成立，则不执行 `b=0` 和 `jump`。
3. 串行表达式的计算方式
如：`a = b = c++` 等价于 `b = c c++ a = b`
先把 `c` 的值赋给 `b`，然后 `c` 自加 1，再把 `b` 的值赋给 `a`。
如：`a+=b+=2` 等价于 `b+=2 a+=b`
先把 `b` 的值自加 2，再把 `a` 的值自加 `b` 的值。
如：`a++ b++ c++` 会依次从左往右执行
如：`a = b = c = 10` 等价于 `c=10 b=c a=b` 依次从右往左执行赋值操作
如：`if(a<b<10)` 等价于 `if (a<b && b<10)`
4. 函数的返回值赋值给变量，用来判断运动报警
如：`a = pos(100101)`
获取轴 ID 为 100101 的当前位置，保存在 `a` 变量中
5. 使用一个流程来判断报警，平台会针对每个流程和任务自动创建一个运行变量，命名方式为 `txx` 其中 `xx` 为任务 ID，一旦流程报警，那报警变量的值为错误代码。可以在流程中判断各个变量的值来判定是否报警，并使用跳转函数 `jump` 来处理异常。
6. `str` 函数可以把 `int` 和 `double` 类型的转为字符串进行运算，字符串可以支持加法运算，此时使用 `str` 函数可以做到格式化字符串的效果。如 `a = "hello world " + str(b) + " bye ."` 此时不管 `b` 是什么类型，该脚本都可以执行，并且 `a` 的内容可以包含空格，假如 `b` 的值为 123，则脚本运行后 `a` 的值为 "hello world 123 bye."