# 润伟领域特性语言（RWDSL）

# 使用说明文档

## 前言

传统软件开发是基于特定的语言，比如C#、JAVA、C++等等，其中会包含一些领域特性的语言，比如制作网页需要用到HTML、CSS、Javascript等，同时基于数据库开发时，还需要用到SQL语言。此时，我们将C#、JAVA、C++等基本的语言称之为“通用编程语言”。将HTML、CSS、Javascript、SQL称之为“领域特性语言”或“领域语言”，以下称之为“DSL”。

根据以上概念，领域特性语言（DSL）是指在特定的领域或工作环境下所使用的编程语言，且语言只适用于解决特定的问题，比如HTML只用于显示在浏览器中；INI文件一般用于做配置文件；SQL用于对数据库的操作等等；

此时，我们可以想象到，那么所谓的通用编程语言，在一定条件下，也是DSL的一种，比如汇编语言与JAVA，JAVA语言则是小儿科了，此时JAVA语言也是一种DSL。而HTML等此类语言是更加细化的DSL，以上就是对于DSL的介绍和理解。

那么，DSL需要如何应用呢？对程序员而言，每一门语言都有他自己特有的的语法、词法、关键字等等特性，需要先掌握其特性才能对其进行编程的开发和应用。每一门语言都是五花八门的编程案例、“入门到精通”，用于让大家更快的熟悉使用这门语言。

通过对以上概念和情况的了解，编程语言是人发明出来的，主要的目的是解决其他语言和类似语言解决不了的问题，或是提高效率的语言。那么，我们是否可以创建一个属于自己的编程语言呢？

是的，我们创建了一款RWDSL，主要目的是将试验过程的动作变为可编辑的，以减少试验过程对软件设计的依赖，而软件设计也无需去深入了解被试品的动作工艺，可以专注于软件本身的开发上。

软件利用了领域特性语言的概念，使用了Antlr4作为编译器，开发出了RWDSL，下面将对RWDSL进行详细的介绍。

目录

[润伟领域特性语言（RWDSL） 1](#_Toc3960)

[使用说明文档 1](#_Toc20764)

[前言 2](#_Toc13530)

[第一章 总体概述 5](#_Toc6923)

[1.1 适用人员 5](#_Toc20965)

[1.2 适用环境 5](#_Toc22529)

[第二章 领域语言语法说明 6](#_Toc20706)

[2.1 结构 6](#_Toc25362)

[2.2 特性 6](#_Toc6844)

[2.2.1 行命令 7](#_Toc15099)

[2.2.2 语法块 7](#_Toc3769)

[2.3 系统组成 7](#_Toc6016)

[2.3.1 过程语言 7](#_Toc19667)

[2.3.2 模块对象 8](#_Toc27350)

[2.3.3 内置模块 9](#_Toc4105)

[2.3.4 函数 13](#_Toc14129)

[2.3.5 系统变量 17](#_Toc38)

[2.4 动作指令集 17](#_Toc8686)

[2.4.1 定义 18](#_Toc24270)

[2.4.2 等待 18](#_Toc14583)

[2.4.3 打开/关闭/脉冲 19](#_Toc20940)

[2.4.4 设置 19](#_Toc17487)

[2.4.5 检测 20](#_Toc23797)

[2.4.6 条件语法 20](#_Toc3299)

[2.4.7 循环 22](#_Toc28713)

[2.4.8 调用 23](#_Toc2843)

[2.4.9 事件 24](#_Toc21033)

[2.4.10 退出控制流程 24](#_Toc17088)

[2.4.11 错误 25](#_Toc14180)

[2.4.12 打印 25](#_Toc32556)

[2.4.13 异步 26](#_Toc3050)

[2.4.14 同步 26](#_Toc31201)

[2.4.15 函数引用 27](#_Toc29211)

[2.4.16 模块引用 28](#_Toc1787)

[2.4.17 采集 29](#_Toc17375)

[2.5 数组 31](#_Toc32285)

[2.6 值表达式类型 31](#_Toc4694)

[2.6.1 数值 31](#_Toc3008)

[2.6.2 数组 31](#_Toc16493)

[2.6.3 进制表示 32](#_Toc13847)

[2.6.4 真假 32](#_Toc30394)

[2.6.5 字符串 33](#_Toc31195)

[2.6.6 变量 33](#_Toc25480)

[2.6.7 模块调用 34](#_Toc18115)

[2.6.8 时间 34](#_Toc4096)

[2.6.9 数学运算&逻辑运算 34](#_Toc8307)

[2.7 表达式 34](#_Toc7113)

[2.7.1 且 35](#_Toc4133)

[2.7.2 或 35](#_Toc31397)

[2.7.3 非 36](#_Toc13405)

[2.7.4 逻辑运算优先级 36](#_Toc27585)

[2.8 语法友好性 36](#_Toc26460)

[2.8.1 赋值简写 36](#_Toc24802)

[2.8.2 累加/逐减 37](#_Toc31295)

[2.8.3 中英命令支持 37](#_Toc31630)

[2.9 异常处理 38](#_Toc12768)

[2.10 执行日志 38](#_Toc14558)

[2.11 执行建议 39](#_Toc23541)

[第三章 参考示例 40](#_Toc26795)

[3.1 在项目中使用的案例 40](#_Toc17928)

[第四章 领域模块介绍 44](#_Toc13091)

[4.1 前言 44](#_Toc27955)

[4.2 定义 44](#_Toc18445)

[4.3 C#代码应用领域模块 46](#_Toc13519)

[4.4 Winform设计关联 48](#_Toc27982)

[第五章 软件安装 50](#_Toc9068)

[第六章 专业术语 51](#_Toc3691)

# 总体概述

在非标测控设备开发领域中，正在呈交付周期短、利润低的方向前进。进入行业的厂家也越来越多，以“利润=售价-成本”的公式得出，要想提高利润，售价上不占优势的大环境下，只能通过降低成本来提高利润。而成本有多方面的原因，这篇文章主要针对效率和人工成本的降低。

另外，由于非标测控设备的非标准的性质，在进行定制化开发后，往往需要大量的时间在客户现场进行使用指导、功能调优、性能优化等工作。并且，交付后的一两年内，均需要对设备进行维护升级工作。

企业经过10余年的发展，已经积累了大量的非标设备的设计经验，这些经验在一定程度上是可以被标准化的。企业的软件专家们，基于这些经验进行总结分析和抽象建模，形成了“润伟领域特性语言（RWDSL）”。该语言主要解决了不同设备之间的点位差异和动作差异。将这种差异进行抽象，形成了“领域模块”和“领域过程”的概念，不再依赖“硬编码”来实现上述差异，采用了“软编码”的方式灵活动态的编码，在无需软件设计人员介入的情况下即可完成编码的工作。通过上述方式，提高了非标设备的标准性，降低了设备的维护成本、升级成本。同时，使用此方法，在同类设备的软件设计中具有一定的适应性，可快速的进行同类设备的开发。

综上所述，开发设计了一款可供工艺人员（领域专家）编制领域模块、领域过程编程的语言，该语言简单易学，只需掌握少量的关键字和语法结构，即可完成测控设备的动作逻辑编制，大大减少了软件设计必须参与测试的问题，降低了测控设备开发成本，提高了软件可复用性，创建了一种新的在测控设备领域中快速研发的方法和途径。

## 适用人员

软件设计、调试人员、工艺员、操作员。

## 适用环境

软件环境为.NET Framework3.5以上，同时可适用于Labview等支持.NET平台的工控软件设计工具。

# 领域语言语法说明

## 结构



（功能组成图）

## 特性

该语法使用中文语法编程，语法友好。使用每行一段代码，不跨行的语法结构。使用空格作为缩进和断句。语法结构与Python有相似之处，如果有相关经验，可以更快的理解此语法，本文主要面对领域专家，将会描述每个细节。

RWDSL使用“动作 模块 值”此结构进行编程开发比如：

“**打开 试验台.工频电源**，”

“**设置 变频器.转速=1500**”

“**定义 @变量=100**”“**等待 1 秒**”

### 行命令

语法使用行命令模式，即一行一命令，不跨行。在进行代码编写时，需要主要一行内容不宜过多，过长的代码不易读，不容易排错。建议优化代码。

由于字符串的特殊性，在编写时，除了声明和赋值字符串时，使用’’’进行跨行外，其他条件均不允许跨行。

### 语法块

语法块用于拆分不同功能代码和分组语句，语法支持缩进的语法，我们使用首字符为“ ”（空格）来表示一个缩进，两个空格表示两个缩进，通常应用在条件语法和循环语法中。多个缩进称之为嵌套语法。

在v1.2.0+的版本中，我们新增了Tab键缩进语法，可以使用Tab键作为缩进条件，但需要注意，如果第一行使用空格，则后续都只能使用空格，Tab也是同理。

语法块，主要用于条件语句和循环控制语句中。

## 系统组成

### 过程语言

过程语言即是我们的“润伟领域特性语言”，包括但不仅限于“润伟领域特性语言”，我们规划了python、javascript等常用脚本语言作为过程语言。我们将“自动试验过程”使用过程语言进行编写，同时与上层进行模块对接，从而实现了动态编写过程语言的功能。由于python和javascript都是规划，暂时将不对其进行介绍。以下都以“润伟领域特性语言”为基础进行介绍和演示。

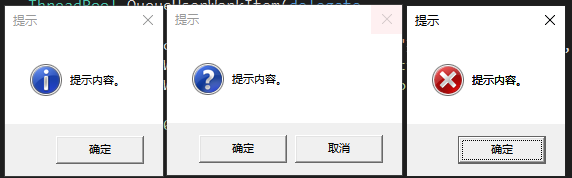
### 内置模块

本系统中，为了丰富语言的特性，增加了一些内置模块，方便使用者调用。下面将一一进行介绍。

#### 系统模块

主要用于系统弹窗的功能，功能等同于“MessageBox.Show”弹出对话框。

系统的标准格式为：“**系统 命令 内容**”，内容通常为一个字符串。



(提示样式)

|  |  |  |
| --- | --- | --- |
| **序号** | **命令** | **说明** |
| 1 | 消息/提示 | 弹出默认对话框。 |
| 2 | 错误 | 弹出一个有错误图标的对话框。 |
| 3 | 警告 | 弹出一个有感叹号图标的对话框。 |
| 4 | 确认 | 弹出一个有问号图标的对话框，并提供了确定、取消按钮。点击确定时，返回true。 |
| 5 | 警告确认 | 弹出一个有黄色感叹号图标的对话框，并提供了确定、取消按钮。点击确定时，返回true。 |
| 6 | 错误确认 | 弹出一个有错误图标的对话框，并提供了确定、取消按钮。点击确定时，返回true。 |
| 7 | 消息确认 | 弹出一个有蓝色i号图标的对话框，并提供了确定、取消按钮。点击确定时，返回true。 |

（系统模块：命令表）

示例：

|  |
| --- |
| #系统模块调用示例  系统 消息 ‘这是一个普通消息’  系统 警告 ‘请确认电路已正确连接，点击确定后继续试验。’  系统 错误 ‘程序有一些错误’  @result=系统 警告确认 ‘确认要进行试验吗？’  如果 @result  错误 ‘试验异常结束’  @result2=系统 错误确认 ‘气压检测异常，点击确定继续试验。’ |

（系统模块：示例）

#### **日志模块**

日志模块主要提供给领域专家在编写领域代码时，由于异常信号或异常逻辑时，方便通过软件的方式进行记录。日志模块采用文件流的方式，支持高并发。默认保存路径在“程序目录\logs\dsl\_user.log”文件中。领域专家通过编写指定日志代码后，当程序运行时，会按照指定的规则将日志写入到文件中。日志格式默认为”时间：内容”为一行一次的记录。

日志记录标准命令为：“**日志 命令 值**”，值应该为一个字符串类型。

日志命令表：

|  |  |  |
| --- | --- | --- |
| **序号** | **命令** | **说明** |
| 1 | 写入 | 记录一次普通的日志信息，不做任何标记。 |
| 2 | 信息 | 记录一次信息的日志信息，将日志的内容标记为[INFO]，格式为”时间:[INFO]内容”。 |
| 3 | 警告 | 记录一次警告的日志信息，将日志的内容标记为[WARNING]，格式为”时间:[WARNING]内容”。 |
| 4 | 错误 | 记录一次错误的日志信息，将日志的内容标记为[ERROR]，格式为”时间:[ERROR]内容”。 |
| 5 | 调试 | 记录一次调试的日志信息，将日志的内容标记为[DEBUG]，格式为”时间:[DEBUG]内容”。 |

（日志模块：命令表）

示例：

|  |  |
| --- | --- |
| #日志模块调用示例  日志 写入('==写入的日志内容==')  日志 警告('这里有一条警告内容')  日志 错误('这是一个错误的消息')  日志 信息('这里是一个信息内容')  日志 调试('这里是一个调试内容') | （日志文件） |

（日志模块：示例）

#### **计时器模块**

计时器模块主要用于需要对时间进行记录时进行调用，系统内置了一个默认计时器，名称为‘Default’，可通过“切换”命令切换不同的计时器。对于单计时器应用，1，2，3命令均无需使用。对于多计时器应用，可通过命名创建、切换不同的计时器。后续调用则直接使用切换后的计时器，原计时器如果未停止，仍然会运行。

计时器模块是线程不安全的，未对多线程进行优化。多线程同时操作计时器时，请勿使用。否则可能导致异常的情况。

计时器模块标准格式为：“**计时器 命令 [名称]**”，名称为可选项，特定命令才支持。

|  |  |  |  |
| --- | --- | --- | --- |
| **序号** | **命令** | **类型** | **说明** |
| 1 | 创建 | 方法 | 创建一个新的计时器，并命名，创建后自动切换成新的计时器。当未调用时，其计时器默认名称为“Default”。 |
| 2 | 切换 | 方法 | 切换到指定计时器中。 |
| 3 | 名称 | 方法 | 显示当前的计时器名称 |
| 3 | 开始 | 方法 | 启动当前的计时器。并开始计时。 |
| 4 | 停止 | 方法 | 停止当前的计时器，并结束计时。 |
| 5 | 暂停 | 方法 | 暂停当前的计时器，并暂停计时。 |
| 6 | 重置 | 方法 | 重置当前的计时器，将时间清空为空，如果未停止，则会置为停止状态。 |
| 7 | 毫秒数 | 属性 | 返回当前的毫秒数 |
| 8 | 秒数 | 属性 | 返回当前的秒数 |
| 9 | 分钟数 | 属性 | 返回当前的分钟数 |
| 10 | 小时数 | 属性 | 返回当前的小时数 |
| 11 | 天数 | 属性 | 返回当前的天数 |
| 12 | 时间 | 属性 | 返回当前的时间数，格式为TimeSpan，可直接转换成double类型，单位为毫秒数。 |

（计时器模块：命令表）

#### **线程模块**

此模块仅在v1.2.0+以上可用，请注意版本。

由于新增了异步过程的代码，特新增了线程模块，用于独立控制线程的执行。这非常便于在多线程的环境下使用。介于线程模块的特殊性，线程模块的操作对象会根据线程的不同而不同。即为每个线程都会单独创建一个对象，而实际使用中并不需要关心是哪个对象，只需清楚知道当前在操作哪个线程。

线程定义请参考 **错误！未定义书签。** 的使用。

当需要操作当前线程或指定线程，可以使用内置提供的线程模块，使用方法与模块的使用完全相同。

线程模块的标准语法为：“**线程 命令 [参数]**”，目前支持了少量的对象，后续会继续增加。

|  |  |  |  |
| --- | --- | --- | --- |
| **序号** | **命令** | **类型** | **说明** |
| 1 | 名称 | 属性 | 获取当前指定的线程标识名称 |
| 2 | 日志 | 属性 | 打开或关闭线程级别的日志开关。 |
| 3 | 跳出 | 属性 | 获取或设置跳出开关=true时，会跳出当前循环 |
| 4 | 运行 | 方法 | 可手动运行指定的线程 |
| 5 | 停止 | 方法 | 可手动停止指定的线程，并等待线程结束 |
| 6 | 运行状态 | 属性 | 查看当前线程的运行状态，返回true、false |
| 7 | 采集状态 | 属性 | 设置当前的参数状态 |
| 8 | 关闭采集 | 方法 | 可关闭指定变量名的采集线程 |

（线程模块函数属性表）

使用示例：

|  |
| --- |
| 线程块[自动=false]:  关闭 线程 日志  @i=0  循环 true  打印 @i  @i++  等待 1 秒  ;  等待 1 秒  线程 运行(‘线程块’)  等待 5 秒  线程 停止(‘线程块’)  打印 @i |

（线程使用示例）

### 函数

函数使系统拥有更大的适应性，系统可用性更高。函数分为系统函数和自定义函数。系统函数主要是常用的数学函数。自定义函数由用户自己定义名称、参数等。

函数是一种特殊的领域过程，可以与领域过程编写完全相同。同时领域过程中，又可引用另外一个领域过程，我们称另外的领域过程称之为“子过程”，而这个子过程，我们将以“函数”的方式来进行调用。以下针对函数进行系统的说明。

#### 系统函数

系统函数主要作用于数据处理，原则上，采用基本的运算，同样可以实现函数的功能，而函数的意义在于可以快速算出结果。提高了设计效率以及降低开发周期的作用。

我们可以通过下面的常用函数，实现一些需要算法设计的功能。

系统函数的通用语法为“$name [arg1] [arg2]”参数数量根据函数类型不同而确定。

由于函数的参数也可以是任意的变量、计算公式等。所以，每个变量考虑使用“（）”进行包裹，以确定函数的范围。最终形式为：“$name (arg1) (arg2) (arg3)...”

或者也可以使用”$name(arg1,arg2,arg3..)”的方式进行调用。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 序号 | 函数名称  (不分大小写) | 中文命名 | 参数 | 说明 |
| 1 | Sin | 正弦 | 角度 | 正弦 |
| 2 | Cos | 余弦 | 角度 | 余弦 |
| 3 | Tan | 正切 | 角度 | 正切 |
| 4 | Cot | 余切 | 角度 | 余切 |
| 5 | Abs | 绝对值 | 实数 | 绝对值，如“$abs -123”=123 |
| 6 | sqrt/square | 平方根 | 正实数 | 平方根，如“$sqrt(4)”=2 |
| 7 | Pow/Power | 开方 | 实数 | 平方，如“$pow 4”=16 |
| 8 | Log | 对数 | 1. 底数 2. 对数 | 对数；如“$log 10 100”=2 |
| 9 | Round | 四舍五入 | 实数 | 四舍五入；如“$round 1.234 2”=1.23 |
| 10 | ceil/Ceiling | 最小整数 | 实数 | 大于的最小整数；如“$ceiling 1.23”=2 |
| 11 | Floor | 最大整数 | 实数 | 小于的最大整数；如“$floor 1.23”=1 |
| 12 | Max | 最大值 | 数组/多个值 | 求数组的最大值，或多个参数中最大值 |
| 13 | Min | 最小值 | 数组/多个值 | 求数组的最小值 |
| 14 | Avg | 平均值 | 数组/多个值 | 求平均值 |
| 15 | Pi | π | 常数 | 数学公式π，约为3.1415926535897931 |
| 16 | E |  | 常数 | 自然常数，约为2.7182818284590451 |
| 17 | Random | 随机数 | 1. 最小值 | 随机数 |
| 18 | Number | 数字 | 字符串 | 将字符串转换成数字； |
| 19 | Sort/Sortasc | 正序 | 数组 | 从小到大排序，返回新的集合 |
| 20 | Sortdesc | 倒序 | 数组 | 从大到小排序，返回新的集合 |
| 21 | Tostr/format | 格式化 | 任意 | 将数字转换成字符串 |
| 22 | toint |  | 任意 | 将字符串转换成数字 |
| 23 | Count  Length  Size | 数量 | 数组 | 计算数组的长度，或字符串的字数 |
| 24 | lt | 小于 | 1. 数值或数组 2. 数值 | 返回参数1是否小于参数2的bool类型。 |
| 25 | lte | 小于等于 | 返回参数1是否小于等于参数2的bool类型。 |
| 26 | gt | 大于 | 返回参数1是否大于参数2的bool类型。 |
| 27 | gte | 大于等于 | 返回参数1是否大于等于参数2的bool类型。 |
| 28 | eq | 等于 | 返回参数1是否等于参数2的bool类型。 |
| 29 | neq | 不等于 | 返回参数1是否不等于参数2的bool类型。 |

（系统函数：对照表）

#### 自定义函数

当系统内置的函数无法满足业务的需求，系统提供了一种允许用户手动注入新的函数，此函数可由程序员通过代码的方式注入进系统中。

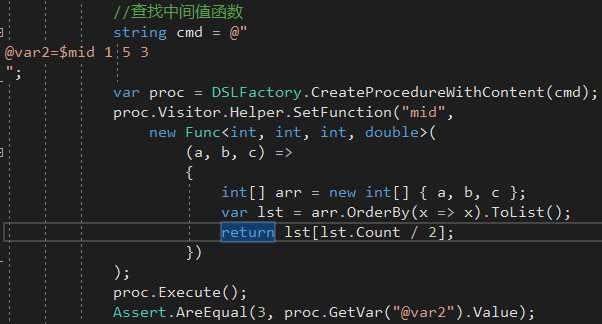
自定义函数的优先级高于系统函数，当出现与系统函数同名的自定义函数时，系统将自动使用自定义函数。此功能的目的是为了允许用户扩展和优化。

函数注入提供的相关方法和说明：

|  |  |  |
| --- | --- | --- |
| 序号 | 方法名 | 说明 |
| 1 | SetFunction(string name,Delegate del) | 设置一个函数以及执行过程 |
| 2 | RemoveFunction(string name) | 移除指定名称的函数 |
| 3 | ClearFunction() | 清空所有的函数 |
| 4 | ExecuteFunction(string name, params object[] args) | 执行函数，指定函数名称以及参数 |
| 5 | ContainsFunction(string name) | 是否拥有指定名称的函数 |

（函数注入操作方法）

**示例**：新增一个查找中间值函数。



（示例：查找中间值）

### 系统变量

由于设计需要，系统提供一些系统变量方便大家使用。

#### @当前时间

用于显示和计算当前时间，此变量每次使用时，值都不同。

示例：

|  |
| --- |
| @开始时间=@当前时间  等待 1 秒  @提示信息=’开始时间是：{@开始时间}，当前时间是：{@当前时间}’ |

## 动作指令集

动作是本语言的核心，该语言目前设置的动作仅有不到20个，后续虽有增加，但对于一款编程语言来说，已经足够简单了。

### 定义

语法：“**定义 @变量=值**”

说明：使用“定义”关键字作为动作名称，使用“@”关键字作为变量，使用“=”作为赋值动作。

应用：变量是一门语言中最常用的概念，使用变量可以存储信息，如需要循环5次，中间需要用到变量进行累加；使用变量，可以使语法的适应性更强，只需要修改某个变量，使系统完成不同被试品的要求。还能使用“定义”定义时间变量，语法是“定义 @变量=值 单位”。“定义”关键字可以被省略。

示例：

|  |
| --- |
| 定义 @时间=1 秒  定义 @i=0  @j=100 |

### 等待

语法：“**等待 值 单位**”

说明：使用“等待”关键字作为动作名称，意为延迟。

使用“值”关键字表示需要延迟的时间值，该值可以是参数（请参考2.3.1 定义）

使用“单位”可以有效的对时间进行缩放，支持的单位有“秒、分、时、毫秒”，可不填，默认为毫秒。

应用：等待也是一个常用操作，比如动作后需要延迟等待几秒后再进行操作，可以适用该语法。比如，要将10个动作中间增加一定的间隔时间，也可以适用此命令，否则10个指令的动作时间将会非常之快；比如需要保压10分钟，也可以适用此命令。“等待”命令也支持变量语法，可以引用变量进行等待。

示例：

|  |
| --- |
| 等待 1 秒  等待 5 分  等待 @稳定时间  等待 @稳定时间 分 |

### 打开/关闭/脉冲

语法：“**打开 模块 属性”、“关闭 模块 属性**”、**“脉冲 模块 属性”**

说明：使用“打开”/“关闭”/“脉冲”关键字用于控制DO模块的输出。特别说明，脉冲的含义是：控制“模块 属性”为010的跳变脉冲，如果是bool类型，则是false\true\false的动作。间隔时间为100ms，系统默认，暂不支持更改。

应用：此功能，仅针对DO点，即模块的属性为Bool类型的变量进行赋值操作。

示例：

|  |
| --- |
| 打开 试验台 总电源  打开 电磁阀 R1  关闭 电磁阀 R4  脉冲 电磁阀 R2 |

在v2.0+版本领域特性语言中，增加了超时和条件执行。可以使用语法“打开 模块 属性[超时=1 秒,检测=模块 属性]”

例：

|  |
| --- |
| 打开 试验台 总电源[超时=3 秒,检测=试验台 总电源检测] |

### 设置

语法：“**设置 模块 属性=值**”、“**设置 @变量=值”**

说明：使用“设置”关键字表示此段语法为复制语句。其中“设置”指令可选，系统自动默认使用该语法。如果“模块 属性=值”、“@变量=值”

应用：赋值一般用于控制，通常是对AO或DO进行输出，或者是将临时变量进行赋值，当变量未定义时，会自动初始化变量，初始化的值为0或false，根据后续引用时的类型。

示例：

|  |
| --- |
| 设置 压力传感器 EP01=100  设置 电磁阀 R1=true  设置 @变量=100  设置 @数组=[1,2,3,4]  设置 @输入压力=压力传感器 EP01 |

### 检测

语法：“**检测 条件值**”

说明：使用“检测”关键字作为动作名称，意为检测条件值为True时，会跳过此代码。

使用“条件值”作为条件，可以是复合表达式，支持“或”、“且”逻辑组合。请参照表达式的章节查看使用说明。

应用：检测是一个自定义条件等待，与“等待”命令类似，但是此命令支持自定义的条件，同时支持检测时间判断。比如：“检测 压力传感器.EP01>8 或 时间>2 分”，可以看到有2个条件，只要一个条件为True则会跳过此命令，否则一直检测。注意：“**时间**”指令是“条件值”命令的专用判断，避免当检测条件不成立时，会一直检测导致无限等待的问题；

检测可应用于需要循环判断的地方，该语法比“循环”语法更加的精简。

示例：

|  |
| --- |
| 检测 压力传感器.EP01>8  检测 压力传感器.EP01>8 或 时间>2 分 |

### 条件语法

条件语法是语言中最重要的一节，对于试验过程，有很多条件需要进行判断，比如如果试验气压大于100，则关闭电磁阀R2等等，有前提条件才能进行的动作。再比如，如果试验气压大于100，关闭电磁阀R2，否则打开电磁阀R2。不仅有“如果”还有对应的“否则”语句，这样的描述方法，非常符合自然语言。条件语法包含“如果 xxx 否则 xxx”的语法，同时还包含“如果 xxx 否则 如果 xxx”、“如果 （如果 xxx 否则 xxx） 否则 xxx”这种高级的嵌套用法。

#### 如果

语法：“**如果 表达式**

**[空格]任意命令**

**[空格]任意命令**”

说明：使用“如果”作为关键字，表示判断，如果“表达式”结果为True，则进入语句块。使用首字符“ ”空格表示语句块，只有如果表达式成功才会进入语句块，如果不成功，则会自动执行到没有语句块为止。语法本身支持多层嵌套，如果“如果”语法中嵌套了“如果”命令，则使用2个空格，依次类推。保证嵌套规则不乱。

应用：当变频器状态已启动时，要先关闭掉变频器。如果电磁阀R2是打开的，先关闭电磁阀R2。这样的需求，就可以使用此命令。对于条件命令，请查看后面的示例程序。

示例：

|  |
| --- |
| 如果 变频器 就绪  变频器 运行  变频器 频率=10  检测 变频器 频率>=10 或 时间> 1 分  如果 变频器 频率<10  错误 ‘频率未上升至指定值’ |

条件语法支持单行模式，例如，“如果 true @a=100”，注意，单行条件语法只支持一条命令。

#### **否则**

语法：“否则

[空格]任意命令”

说明：配合“如果”指令使用，“如果”指令的表示如果不成立，则会自动跳到否则指令中，同时还可以“否则 如果 表达式”这样的指令进行编辑。如果、否则是我们日常中很常见的逻辑判断，将生活中的语法移植到此语法过程中。

应用：如果5点钟了，那么下班，否则继续上班；

嵌套应用：如果 5点钟了，那么如果工作完成了就下班，否则继续上班。

否则如果应用：如果5点钟了，那么下班，否则如果周报没写完，那么写周报，否则去休息一下。

示例：

|  |
| --- |
| 如果 变频器 就绪  设置 变频器 运行  否则 如果 !试验台 主电源  设置 试验台 主电源=true  否则  错误 ‘变频器未到达就绪状态’ |

#### **否则 如果**

语法：“否则 如果 1==1

[空格] 任意命令”

说明：通常，“否则 如果”指令，只适用于“如果”条件不成立时使用。可以连续接多个“否则 如果”，用于连续判断。

### 循环

语法：“循环 表达式

[空格]任意命令”

说明：当需要多次使用一种类似的动作时，可使用循环语句，循环语句不会自动计数，需要创建相关的变量或者循环跳出条件，多次循环判断循环的条件，如果为true则继续，否则跳出循环。“循环”语句也包含语句块，语句块通过换行且多一个空格来表示属于这个语句块。

**警告：版本1未实现多重嵌套循环的功能，请勿使用嵌套循环。**

**在库版本v1.2.0+已经实现了嵌套循环，可以任意进行嵌套，请查看库版本后决定。**

应用：当某个试验过程需要多次执行相同的操作（部分参数不同，但动作的点位基本相同）时，可以使用循环语句。比如，跑合试验台中，需要分别跑300，600，1000的磨合试验，然后循环3次（或者使用数组也可以），每次执行不同的速度。

示例：

|  |
| --- |
| 打印 @当前时间  定义 @次数=0  循环 @次数<3  @次数++  等待 1 秒  打印 @次数  打印 @当前时间 |

（循环简单示例）

|  |
| --- |
| 打印 '冒泡排序'  @数组=[5,4,1,2,3]  @len=5  @i=0  @j=0  循环 @i<@len  @j=@i+1  循环 @j<@len  如果 @数组[@j]>@数组[@i]  @temp=@数组[@i]  @数组[@i]=@数组[@j]  @数组[@j]=@temp  @j++  @i++  打印 @数组 |

（冒泡排序算法）

#### 2.3.7.1 循环退出

语法：“[BREAK]”

说明：当希望使用循环命令，而无需等待循环执行完毕时，可以在循环内部使用[BREAK]语法，用于执行循环中断指令。

示例：

|  |
| --- |
| 定义 @次数=0  循环 @次数<5  @次数++  如果 @次数==3  [BREAK] |

### 调用

语法：“调用 模块 属性”或

“调用 模块 方法 参数1,参数2...参数N”

说明：只要是开发，就不可避免的会有外部的调用，该领域语言支持了外部库文件的调用（暂时支持.NET Framework4.5及以下的调用），可以调用库文件的静态属性或者静态方法。其中，方法的调用需要传入参数，多个参数以“逗号,”分隔。

示例：

|  |
| --- |
| 调用 excel write N55,234 #往N55的excel中写入234的值  调用 filter avg 1,2,3,4,5,6,7,8,9 #对一个数组进行滤波  调用 变频器 线性启动(100,200,50) |

### 事件

语法：“事件 名称 事件参数1...N”

说明：由于领域特性语言是一种纯脚本语言，未关联UI层，一旦需要在试验过程中进行UI交互，就只能使用“调用”语法，调用语法需要预先写好方法名，而实际上编写领域过程代码时还没来得及写，此时我们推荐使用“事件”的方式进行触发调用，事件的好处是无需定义具体的方法，源代码和领域代码处于松耦合状态，符合设计原则。

当完成一个自动试验时，通常要返回一个值更新到UI上、写入到报表、写入到数据库等操作。使用事件的方式，触发值的返回，而不关心如何处理值。

示例：

|  |
| --- |
| 事件 泄漏量 @leak  事件 保压试验 @start,@end,@leak,@time |

### 退出控制流程

#### 正常退出

语法：“[END]”

代码调用：Visitor.Stopped(); 或者 Procedure.StopExecute(); 都会触发退出指令。

我们通常在设计自动试验时，会使用两个功能按钮，“开始”和“停止”，开始的意义就是将此代码正确的执行，直到结束。“停止”的意义是，当用户不希望试验继续执行，可提前结束自动试验。或者，用户希望开启某个开关，而此开关一直无法开启，需要去查原因，那么在程序中就需要自动退出试验，因为将影响到后面的动作。

应用：该语法只做声明使用，当用户点击“停止”按钮时，通过代码调用指定的方法“DSLProcedure.StopExecute()”，或者“Visitor.Stopped()”方法。调用方法后，代码将会暂时中断运行，直到遇到“[END]”指令，将[END]指令后的代码执行完毕。如未声明[END]，系统将直接忽略所有指令。

#### **急停控制**

代码执行：Visitor.Scrum(bool scrum);

说明：当执行的领域特性语言出现了未知异常，通常是遇见了安全问题或试验过程数据偏离预期值，此时需要完全中断程序的执行，可通过代码调用此方法。

应用：通常的设计是与试验台的急停控制灯进行关联，同时传入scrum状态，true表示急停，false表示恢复急停。

#### **忽略等待(调试用)**

代码：Visitor.SkipWait();

说明：当进行开发调试时，为了快速进行功能验证，可能需要跳过一些“等待”、“检测”的指令，该方法可以快速的忽略掉检测时间以便更快的完成验证。

应用：开发时，可以增加一个“跳过”的按钮，此按钮调用了该方法。那么程序运行任何时候都可以进行跳过的行为。当功能验证完成后，删除按钮或隐藏按钮。

### 错误

语法：“错误 ‘错误内容’”

说明：当我们在执行自动试验过程中，难免会有异常的错误，比如：某个阀动作不了，那么我们需要中断应用程序，并且需要提示指定的内容；比如，自动试验过程中，传入了很多参数，其中某个参数值不在指定的范围。一般来说，“错误”指定会结合“如果”指令一起使用，如果条件成立，则提示错误。

示例：

|  |
| --- |
| 如果 电磁阀 R1=false  错误 ‘充气阀无法动作’ |

### 打印

语法：“**打印 值**”

说明：此功能只用于调试，后续增加调试助手工具，此命令可以被调试助手识别。或使用开发工具调试时，也能被识别。一般用作打印动作执行过程中关键值或者关键动作。

示例：

|  |
| --- |
| @var=123  打印 @var  @var2=’字符串’  打印 @var2  打印 ‘测试打印内容’ |

### 异步

语法：“异步名1[自动=true]:

动作代码

动作代码

;

”

说明：异步语法是在普通语法外，包了一层“modulename:;”的描述，实际的语法与标准语法完全相同。重点在于执行，当使用异步语法时，该段的语句块在其结束后，采用异步的方式执行，此时如果后面还有代码，则会同时执行。（线程的先后顺序是随机，无法保证后一行代码和线程的执行顺序）

默认情况下，“[自动=true]”可省略，而异步代码也是在”；”号结束时自动开始，当需要手动开始、停止时，可以使用线程模块进行设置。“线程 运行(‘test’)”或“线程 停止(‘test’)”进行线程调用。

示例：

|  |
| --- |
| @a=100  test:  等待 100  如果 @a=100  @a=101  否则 如果 @a=200  @a=201  ;  @a=200  等待 200 |

思考：示例代码执行的结果@a变量是多少？

结果是201，如果答错了，请仔细看上面的说明。

### 同步

语法：“**同步 同步名1,同步名2,同步名3... 超时=5 秒**”

说明：同步的出现是由于异步代码的出现，类似异步的语法，如果我希望在下一行代码中，先执行完异步代码（可以是多个异步代码）。然后，再往下执行，如果没有同步的话，那么只能依靠估算异步代码的执行时间，如异步代码中“等待 200”的功能。而同步语法，可以直接精确的等待指定的异步代码执行完毕。

由于同步可能存在无限等待的问题，增加了超时命令，可使用“超时”命令限定等待时间。该命令是可选的，默认为空表示是无限等待。

示例：

|  |
| --- |
| @a=100  test:  等待 100  如果 @a=100  @a=101  否则 如果 @a=200  @a=201  ;  同步 test  @a=200 |

本示例与异步示例代码唯一的区别在于增加了同步语法。那么，根据说明，最终的@a变量值为多少呢？

结果是@a=200，是不是觉得不如不加异步代码，如果没有异步代码，@a的值仍然是200，那异步的意义在哪呢？我再举一个例子。

|  |
| --- |
| @a=100  test:  等待 5 秒  @a=101  ;  test2:  等待 6 秒  @a=102  ;  同步 test,test2 |

我想，这个代码，@a的值应该是显而易见@a=102吧（不知道的话，就重新看一遍异步代码说明）。我的问题是，请问这段代码会执行多久？答案是6秒，请仔细思考。

### 函数引用

引用语法：“$name=’url..’”

调用语法：“$name(arg1,arg2...)”、“$name arg1 arg2”

说明：自定义函数可以在DSL中引入另外一个DSL代码，使用方法与函数调用完全相同，当调用引用函数时，系统会将主函数中的所有模块、子函数同步添加到子函数中，两边可以同步使用模块。但是，变量是函数私有不会添加到子函数中，子函数中的变量也不能引用主函数的变量。

由于是以函数的方式定义，所以函数的特性则需要输入、输出。输入即函数的参数，输出及函数的返回值。

**函数参数**：主函数调用子函数的参数是以“@参数”变量的形式添加到子函数中，子函数中可以使用“@参数”变量获取到传递过来的参数，该参数是一个数组，可以使用“@参数[索引]”获取到指定位置的参数。

**函数返回值**：子函数如需将结果值返回给主函数，那么可以在子函数中给“@返回值”变量赋值，主函数会自动接收到返回值。

示例代码：

|  |
| --- |
| #求最大值  $findmax=’max.rw1’  @arr=[1,2,5,3,4]  @最大值1=$findmax @arr  @最大值2=$max @arr |
| #max.rw1  @数组=@参数[0]  @最大值  循环 @i  如果 @数组[@i]>@返回值  @最大值=@数组[@i]  @i++  @返回值=@最大值 |

（自定义函数示例）

示例代码中，我们使用了自定义函数及系统函数分别求数组的最大值，其结果相同，引用方式不同。由此，我们可以增加自己专属的函数库，用来扩展我们的DSL。

### 模块引用

语法：“模块 自定义名称=’模块名称’”(V2.2.8+可用)

说明：当程序中无法直接指定使用具体模块时，支持自定义引用模块的功能。这个功能在多工位时、相同的模块时特别好用，如，定义的模块中有“DOGroup1”、“DOGroup2”、“DOGroup3”、“DOGroup4”，而程序需要针对DOGroup1-DOGroup4做相同的操作时，我们需要将程序写成四份，如下所示。

|  |
| --- |
| 如果 @工位1  DOGroup1 DO01=true  否则 如果 @工位2  DOGroup2 DO01=true  否则 如果 @工位3  DOGroup3 DO01=true  否则 如果 @工位4  DOGroup4 DO01=true |

通常我们按照上面的这种写法，而模块引用的功能，可以无需明确模块的名称，使用字符串拼接的方式获得模块。然后针对写一次模块设置即可。模块引用如下：

|  |
| --- |
| 模块 DOGroup=’DOGroup’+@工位号  DOGroup DO01=true |

是不是以上方式节省了大量的代码。当然这么使用也是有前提条件的：

1. 目标模块必须要含有相同的属性或方法，否则会调用失败，提示“找不到方法或属性”的错误。
2. 模块名称字符串必须能找到特定的模块，否则此处将提示“找不到模块”错误。

使用注意：模块暂时无法进行模块的声明，只是对已有的模块通过字符串的方式进行引用而已。需要提前声明号模块后即可使用。

### 采集

**语法：**“采集 采集方法 @变量=值表达式[间隔=1000,忽略=10,数量=100]”

（v2.2.10+可用）

**示例：**“采集 最大值 @振动最大值=振动模块 值[间隔=1000,数量=30,忽略=3]”

**示例说明：**每秒钟采集一次并填充队列到30个，将数据排序忽略掉3个最大值，取剩余队列中的最大值，返回“振动模块 值”相同数量的最大结果值到“@振动最大值”变量中。

**说明：**采集模块将以指定的间隔时间，按照特定的方法将“值表达式”定期存储到指定变量中。采集是一种内置的线程，如有多个采集，请使用不同的变量接收（必须，否则将引起数据错乱）。

**值表达式：**值表达式可以是变量、数组、模块值、模块的数组，通常取“模块 值”来取所有通道的值进行采集。

**采集参数：**采集参数支持“间隔”、“忽略”、“数量”

1. 间隔：表示每次采集值表达式的间隔时间，默认单位为毫秒，可以使用其他单位。
2. 数量：表示本次采集的最大队列，采用先进先出原则，保证队列中只保留了最近的数量\*间隔的记录。如(100\*100=10000即保留10秒的数据)
3. 忽略：指定了忽略会将队列中的数据进行排序，然后将最大和最小的忽略数量剔除后再进行计算结果。指定此值可有效的在周期内进行滤波。

**方法：**采集需使用一种采集方法，系统内置的几种采集方法，可适应绝大多数场景。如平均值、最大值、均方根值，可以很好的对数据进行滤波。

内置采集方法有以下几种：

|  |  |  |
| --- | --- | --- |
| 序号 | 采集方法 | 示例 |
| 1 | 最大值 | 取指定数量池中的最大值 |
| 2 | 最小值 | 取指定数量池中的最小值 |
| 3 | 平均值 |  |
| 4 | 均方根值 |  |
| 5 | 滑动平均值 |  |
| 6 | 极差值 |  |
| 7 | 平均偏差值 |  |
| 8 | 标准差值 |  |

* + - 1. 采集控制

采集指令在调用时将会自动开启，无需手动开启采集线程。但可以在DSL中控制其停止，允许单独停止和全部停止。

全部停止：“采集 关闭所有()”，使用此指令和停止DSL中全部的采集线程。

单独停止：“采集 关闭(‘@变量名’)”，使用此指令可停止采集某个变量的采集线程。

## 数组

语法：“声明 @变量数组[值1，值2，值3]”、“设置 @变量数组[0]=100”

说明：数组使用“[]”进行定义和使用，定义时，每个对象都使用“,”(英文逗号、半角)进行分隔，自动设置数据长度。而在获取变量或设置某个变量时，使用数组下标进行读取和写入，下标是从0开始的整数，0表示第一个，1表示第二个，以此类推。

应用：当我们的程序，需要执行多次相同的动作时，我们可以使用数组，然后结合“循环”指令，和定义一个临时变量，用来标记需要执行的次数。当然我们的自动程序不一定要使用数组，但使用数据可以节省大量的代码，而且便于修改。

## 值表达式类型

在上一节中我们看到很多命令都用到了“值”关键字，值具体是什么呢？在目前的定义中，值可以是数值、TRUE、FALSE、单引号的字符串、变量、模块调用、时间、四则运算表达式。数据的交换和传递需要使用值表达式。下面逐一进行介绍：

### 数值

数值可以是整数、小数、正数、负数，这几种表现形式，数值型最大支持64位，即double类型或long类型，更大的系统暂未支持。

**示例：**

|  |
| --- |
| @变量=123  @变量=123.456  @变量=-123 |

### 数组

数组是由一组值组成的集合，数组使用了逗号(,)连接，用于表示多个值，数组的从左至右排序，索引号从0开始。可以引用数组中的索引值，使用[下标]的方式进行引用，系统自动根据下标查找指定数组中的值。需要注意的是，数组的索引值被引用时，赋给其他变量时，如果原始数组值发生改变，不会引发变量值的改变。

**示例：**

|  |
| --- |
| @变量=1,2,3,4  @引用=@变量[0]  @计算=@变量[0]+@变量[1]+@变量[2]+@变量[3]  打印 @计算 #计算值=10  @变量[0]=5  打印 @变量[0] #这里输出5  打印 @计算 #仍旧是10 |

### 进制表示

系统支持16进制的表示方式，也支持二进制的表示方式。系统默认的表示方式为10进制，如果将不同进制混合进行运算，最终的结果也将是10进制数。如果希望将数字转换成不同进制的显示方式，可以利用系统函数支持。

**示例：**

|  |
| --- |
| @变量1=0x1234 #4660  @变量2=0b0010 #2  @变量3=@变量1+@变量2  打印 @变量3 #变量3=4662  @字符1=$tostr @变量1 16 #将会变成字符串’1234’  @字符2=$tostr @变量1 2 #将会变成字符串’1001000110100’  @变量11=$toint @字符1 16  @变量12=$toint @字符2 2 |

### 真假

真假为系统常量，可以使用true，false来表示（不区分大小写），真假值可用于条件判断，循环判断等。另外，0和1也能表示真假，0表示假，1表示真。在条件判断和循环判断时，0和1也能作为判断条件。且所有大于0的值都能表示真。

**示例：**

|  |
| --- |
| @变量=true  @变量=False  如果 @变量  打印 true  否则  打印 false |

### 字符串

字符串是系统中最常用的一种交互方式，由于试验过程中，常常因为意外情况导致中止，中止时的信息则使用字符串进行表述。

为了区分字符串和命令模块，本系统字符串采用一组’（单引号）或”（双引号）进行标识，一对符号内的内容即是字符串信息。如’这是一个文本信息’，声明了一个字符串信息，该段内容不会被识别为命令或模块。

但是，实际应用中，往往不只是普通字符串，可能还会在字符串中包含一些变量、模块值等其他信息。系统中支持字符串拼接模式。允许使用’+’（加号）连接两个不同的字符串。同时，我们还支持在字符串中添加标识符，自动注入替换标识的方式进行声明，采用格式化字符串的方式进行替换。

|  |
| --- |
| **字符串是系统唯一支持跨行使用的对象，请注意结束字符串声明。** |

**示例：**

|  |
| --- |
| @字符串1=’这是一个普通字符串’  @字符串2=’连接另外一个’+’字符串内容’  @字符串3=’连接变量和字符串,@变量=’+@变量  @字符串4=’连接变量和字符串,@变量=%s’(@变量)  @字符串5=’@变量1=%s，@变量2=%s’(@变量1,@变量2)  @字符串6=’@变量1=%s’($round @变量1 2)  @跨行字符串=’’’  使用三个单/双引号，可以使字符串跨行  ‘’’ |

### 变量

变量是值的一种形态，是一种临时数据存储区，当变量标识在左边时，表示变量的声明，声明后的变量即为全局变量，可以在代码中任意位置引用。当变量标识在右边时，表示变量的引用，此处被当做“值表达式”处理，如果变量未声明而直接引用，系统将自动赋默认值，根据实际情况，可能为null，也可能为0。

变量是一种值类型，引用时，只会引用当时的值，如果变量A引用了变量B后，变量B发生改变时，并不会更新到变量A。

示例：

|  |
| --- |
| @变量B=100  @变量A=@变量B  @变量B=200 #变量A仍然为100 |

### 模块调用

模块是系统与硬件进行通讯的桥梁，变量是系统与系统之间交流的工具。模块的调用也能作为“值表达式”进行使用。

示例：

|  |
| --- |
| @变量=模块A.属性B  @温度=温湿度传感器 温度  @湿度=温湿度传感器 湿度  @频率=变频器 输出频率  @气压=EP阀 EP01 + 100 |

### 时间

时间值 单位（时间单位暂只支持，毫秒，秒，分，时，天）

示例：

|  |
| --- |
| @变量=10 分  @变量=100时  @变量=1 天  @变量= |

### 数学运算&逻辑运算

先乘除、再加减，有括号先算括号内的。如(1+2)\*3;同时四则运算的运算中间量又可以是值、变量、模块等。实现复合逻辑运算。支持的运算符有“+-\*/%”。

当前所支持的运算符有：

## 表达式

用于**计算求值、真假判断的复合表达式**，例如“1+2\*3”的解析，使用了四则运算中，先乘后加，其结果为7；当然我们的语法支持了高级的语法模式，例如“(1+2)\*3-4/5”，该表达式也能正确的被解析；当然，此语法还支持更加强大的表达式，变量引用和模块调用的模式，例如“变频器.频率\*60/电机参数.极对数”，领域专家知道，这是频率转速的换算关系。

**逻辑表达式**，表达式还有一个应用就是真假判断，比如“循环 表达式”、“如果 表达式”都用到了真假表达式，真假表达式可以用于判断条件是否为真，循环条件为真则会继续循环，“如果”条件为真，则会进入语句块，表达式同时也支持复合型表达式，所有值的语法计算都支持，同时还支持联合条件表达式，比如“表达式1 且 表达式2 或 表达式3”可以看到，这段语法中有3个表达式，使用“且”、“或”进行相连，这种是数学中常见的语法，下面对语法进行介绍，如果有一定数学知识和编程知识的可跳过此节。

### 且

用于连接两个表达式，形成一个新的表达式结果。比如“表达式1 且 表达式2”那么，形成了表达式12，该表达式是此表达式是运算后的结果。通常连接的表达式都是真假判断，所以新表达式也就是真假，其结果满足下表规则。

|  |  |  |
| --- | --- | --- |
| 表达式1 | 表达式2 | 结果 |
| 真 | 真 | 真 |
| 真 | 假 | 假 |
| 假 | 真 | 假 |
| 假 | 假 | 假 |

（表格3：且）

### 或

与“且”的用法相同，意义不同。用于表示其中一个结果成立即可。真假规则如下：

|  |  |  |
| --- | --- | --- |
| 表达式1 | 表达式2 | 结果 |
| 真 | 真 | 真 |
| 真 | 假 | 真 |
| 假 | 真 | 真 |
| 假 | 假 | 假 |

（表格4：或）

### 非

非表达式与“且”、“或”表达式不同，非只作用于一个表达式，用于取反，如果结果为真，取反则为假，反之为真。我们习惯的使用“！”（半角符号）用来表示取反。我们修改上面的表达式运算，“表达式1 且 !表达式2 或 !表达式3”，其返回的结果则不相同了。

|  |  |  |
| --- | --- | --- |
| 取非 | 表达式 | 结果 |
| 非 | 真 | 假 |
| 非 | 假 | 真 |

（表格5：非）

### 逻辑运算优先级

逻辑运算同样也支持括号行为，比如上述表达式可以这样使用“表达式1 且 (表达式2 或 表达式3)”，括号的永远是最先执行的。比较这3个表达式，其优先级顺序为“非”、“且”、“或”。

## 语法友好性

由于该项目前期使用纯语输入编辑，后期才加入工具化的功能，对一些常用用法进行了缩减，以减少代码编辑量，同时减少语法错误的频率。

当前已缩减语法的命令有：定义、设置。由于设置命令语法可以自动创建变量，所以当输入“@变量=100”时，系统自动完成创建变量和赋值的操作。

### 赋值简写

赋值的语法使用“设置 @变量=1”、“设置 模块 属性=100”当使用赋值语句时，我们对其进行了优化，允许忽略掉“设置”命令，直接使用“@变量=1”、“模块 属性=100”这样的方式，减少了代码的编辑量。

### 累加/逐减

当我们使用循环时，需要使用到“@变量=@变量+1”此种语法，虽然已经经过优化了，但还是不够简洁。参考其他编程语法，我们使用了“@变量++”、“@变量--”来代替需要+1或者-1的动作。

当累加幅度不是1时，如“@变量=@变量+10”我们也可以使用“@变量+=10”这样的写法，最大程度减少代码量。

### 中英命令支持

由于项目可能会有出口项目，为了便于推广应用，编译器支持中英命令。比如，“定义 @变量=100”可以使用英语的语法“DEF @变量=100”，这两段代码，所表示意义完全一样。当然变量的命名，编译器无法翻译，所以建议变量命名时，使用英文会容易被翻译。下面列出了系统所有命令的中英对照表，可供参考使用：

|  |  |  |  |
| --- | --- | --- | --- |
| 序号 | 中文命令 | 英文命令 | 说明 |
| 1 | 定义 | DEF | 定义变量 |
| 2 | 设置 | SET | 给模块或变量写入值 |
| 3 | 等待 | WAIT | 延迟等待 |
| 4 | 打开 | OPEN | 将DO设置为打开状态 |
| 5 | 关闭 | CLOSE | 将DO设置为关闭状态 |
| 6 | 脉冲 | PULSE | 设置DO的脉冲信号 |
| 7 | 检测 | CHECK | 检测一个条件是否为true |
| 8 | 打印 | PRINT | 调试输出用 |
| 9 | 返回 | RETURN | 返回一个值 |
| 10 | 如果 | IF | 条件判断，如果 |
| 11 | 否则 | ELSE | 与如果命令相对 |
| 12 | 循环 | LOOP | 循环指令 |
| 13 | 调用 | CALL |  |
| 14 | 事件 | EVENT |  |
| 15 | 错误 | ERROR |  |
| 16 | 同步 | SYNC |  |
| 17 | 跳出 | BREAK |  |

（表格6：中英命令对比表）

|  |  |  |  |
| --- | --- | --- | --- |
| 序号 | 中英符号表 | 英文符号 | 说明 |
| 1 | 天 | Day/d |  |
| 2 | 时 | Hour/h |  |
| 3 | 分 | Min/m |  |
| 4 | 秒 | Sec/s |  |
| 5 | 毫秒 | Ms/ms |  |
| 6 | 或 | OR/|| |  |
| 7 | 且 | AND/&& |  |
| 8 | 时间 | TIME | 在检测语法中，此符号表示当前执行的时间 |

（表格7：中英符号对比表）

对于英文语法，截止目前未设置成忽略大小写模式，目的是为了提高语法命令的识别率，根据后续需求可能会考虑忽略大小写。同时，对于多语种系统，语法命令是否需要增加其他语言，也将在项目后期进行讨论。

## 异常处理

程序中自带了友好的异常处理，如“模块[XXX]未找到”、“模块[XXX]的属性[xxx]未找到”,类似的错误有很多。所以对于执行领域特性语言，请务必使用try{}catch{}语法，捕捉运行时异常。

## 执行日志

为了便于调试，以及后期跟踪处理，本系统新增了日志系统，默认情况下，使用DSL执行的所有命令将会被记录到“应用程序目录/logs/dsl.log”文件中，此文件包含了DSL执行的所有过程，包括执行结果以及循环体执行的过程。考虑到文件写入的性能，文件会自动备份，当日志文件超过5MB(默认情况下，允许修改)，会自动备份为新文件。

## **执行建议**

由于领域过程的特殊性，允许中间进行等待延时，当调用执行时，请使用多线程的方式调用，如点击“开始”按钮，将“开始”设置按钮不可用，然后调用领域过程，执行完毕后，将“开始”按钮设置为可用状态。

如果项目是Web项目，那么可以无需考虑多线程执行，因为Web请求本身就是多线程。并不会导致主线程卡住，但需要注意，这个请求此时将是挂起状态，请注意请求超时时间。当然，Web项目如果需要使用领域特性语言，最好采用队列方式，由专用服务来执行过程。如有Web项目需要引用领域过程，请在具体设计时与我联系。

# 参考示例

## **在项目中使用的案例**

|  |
| --- |
| #用于HNRWS18107驱动试验台  #定义 @测试时间=[3 秒,5 秒,1 秒,2 秒]  #定义 @运行速度=[100,200,100,200]  #定义 @方向=[true,true,false,false]  #定义 @温度最大=[100,100,100,100]  #定义 @振动最大=[10,10,10,10]  #定义 @速度范围最小=[100,100,100,100,100]  #定义 @速度范围最大=[200,200,200,200,200]  打开 试验台.变频电源  等待 1 秒  检测 试验台.变频电源 或 时间>1 秒  检测 变频器.就绪 或 时间>1 分  如果 !变频器.就绪  错误 '变频器未就绪'  检测 试验台.风机电源检测 或 时间>1 秒  定义 @上一次方向=0  定义 @次数=0  循环 @次数<@数量  事件 ItemStarted @次数,@方向[@次数],@运行速度[@次数],@测试时间[@次数],@振动最大[@次数],@温度最大[@次数]  如果 @上一次方向!=@方向[@次数] 且 @次数!=0  关闭 变频器.运行  检测 变频器.速度<10 或 时间>1 分  等待 2 秒    设置 变频器.反向=!@方向[@次数]  设置 变频器.速度=@运行速度[@次数]  打开 变频器.运行  设置 @上一次方向=@方向[@次数]  等待 @测试时间[@次数]  事件 ItemExecuted @次数,@方向[@次数],@运行速度[@次数],速度传感器.值,@测试时间[@次数],@速度范围最小[@次数],@速度范围最大[@次数],@振动最大[@次数],@温度最大[@次数],温度巡检仪.值,振动模块.值,噪音传感器.值  @次数++  [END]  关闭 变频器.运行  检测 变频器.速度<10 或 时间>1 分  关闭 试验台.变频电源 |

综上所述，一种新的试验台，需要使用RWDSL，完全只需进行少量的代码改造，甚至说是删除掉很多代码，增加几行代码，即可支持领域特性语言了。

第四步，示例是在已有相关代码的情况下，需要增加描述说明。如果说没有上述代码就可以直接跳过，直接应用领域模块即可。