第 3 章

ST 语言基本语法

俗话说,无规矩不成方圆。无论做什么事都要有规则,ST 语言同样如此。ST 语言即结构化文本语言,就是可以在文本文档中编辑的语言。与计算机高级语言一样,ST 语言也可以在任何可以编辑文字的软件中编辑,例如 Word、文本文档、记事本等,这给代码的编写带来了极大的自由。但是,自由意味着责任,ST 语言编写的程序是为了满足工业控制的需要,必须稳定可靠,稳定可靠的前提就是有完善的规则。

任何语言都有自己的语法规则,例如,在梯形图中,触点必须接在左母线上;线圈必须接在右母线上;触点不能在垂直线上等。在 ST 语言中,一旦不遵守语法规则,编译就无法通过。因此学习 ST 语言时,必须先了解并掌握它的语法规则,ST 语言经过多年的发展,已经有了一套完善的语法规则。

3.1 ST 语言的基本规则

3.1.1 不区分大小写

ST 语言不区分大小写,这是 ST 语言的特色。之所以反复强调,因为这是 ST 语言非常重要的规则,也是笔者认为 ST 语言更接近 Pascal 语言而不是 C 语言的原因。在 ST 语言中,if、IF、iF 是一样的,没有任何区别。这里所说的“不区分大小写”是指它们在编译的时候意义是一样的,可以充分利用 ST 语言的这个特点为编程服务。

3.1.2 变量必须先定义再使用

与所有计算机语言一样,ST 语言的变量必须先定义才能使用。现在的 PLC 编程软件都很智能,如果程序中使用没有预先定义的变量,会有提示,有的 PLC 还会自动为用户建立变量。一般情况下,如无特殊需要,定义的变量可以不用赋初值,但必须指定数据类型和属性。

3.1.3 使用英文输入法

编辑 ST 语言,一定要使用英文输入法,否则编译会报错。现在很多 PLC 支持中文变量,所以在使用中文变量时要注意切换输入法,严格说来,应当切换为半角状态和英文标点。

在中文输入法下，使用半角状态，英文标点输入也是没问题的。但在计算机中，英文输入法默认是半角状态和英文标点，而中文输入法默认是半角状态和中文标点。所以，使用英文输入法是没有任何问题的。以 Windows 10 自带的微软拼音输入法为例，英文输入法和中文输入法分别如图 3-1 和图 3-2 所示。

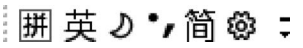

图 3-1　英文输入法

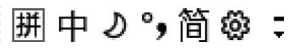

图 3-2　中文输入法

从图 3-1 和图 3-2 可以看出，虽然中文输入法也是半角状态，却是中文标点，是不符合 ST 语言规范的。所以，为了方便，建议读者在英文输入法状态下输入。错误的输入法状态如图 3-3 所示。

图 3-3　错误的输入法状态

采用其他输入法的读者，在输入前务必查看输入法的状态，一定要“半角状态”和“英文标点”。ST 语言中除了使用中文变量外，所有的输入都要在英文标点、半角状态下进行，否则编译时会报错。

3.2　ST 语言的基本组成

了解了 ST 语言的基本规则后，下面正式开始学习 ST 语言。先看一段基本的 ST 语言程序，如图 3-4 所示。

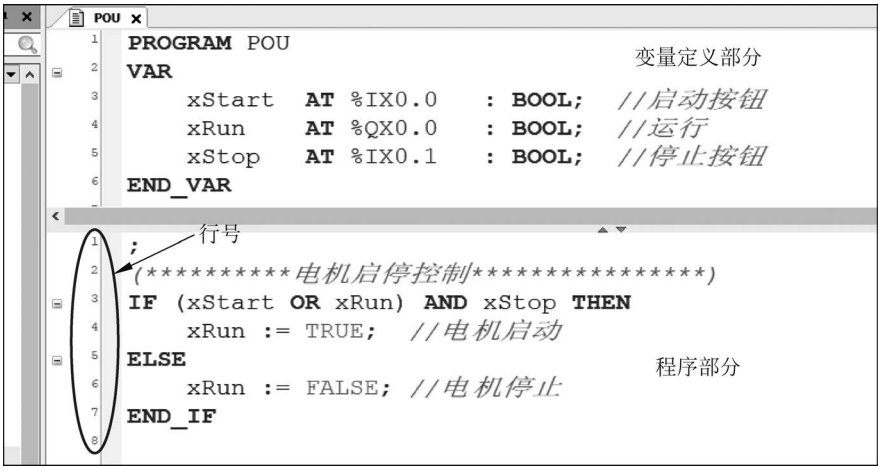

图 3-4　ST 语言的基本组成

从图 3-4 中可以看出，这段 ST 语言程序由“变量定义部分”和“程序部分”组成，与 C++、C＃等计算机语言不同，ST 语言的变量定义部分和程序部分是分开的。不同的 PLC 定义变量的格式是不同的，因此本书主要讨论 ST 语言的程序部分。本书的有些例程会省略变量定义部分，希望读者注意。

关键字 PROGRAM 表示这是一个名称为 POU 的程序。SoMachine 中程序和 POU 的界限是模糊的，这段程序，也可以认为是一个 POU（注意，这里的 POU 是指程序组织单元，而不是这段程序的名字 POU），所以本书在后续章节中不会刻意强调二者的区别。SoMachine 中，程序默认的名字就是 POU、POU_1、POU_2，可以根据需要更改，也支持中文命名。回顾 2.3.2 节讲述的变量属性，在这段 POU 中，关键字 VAR 和关键字 END_VAR 之间定义的变量 xStart、xStop、xRun 是局部变量，只能在本 POU 中使用；如果在其他 POU 中没有定义而直接使用，编译会报错。如果在 POU_1 中定义变量 xStart，那么这个变量跟 POU 中的变量 xStart 是两个完全不同的变量，没有任何关系。

下面介绍基本的 ST 语言程序的组成。

3.2.1 行号

图 3-4 中的程序部分，左侧的数字是行号，清晰地标示了代码的行数，可以快速定位语句。行号 1 的语句，就是“;”；行号 3 和行号 5 的前面有个“－”，表示这部分代码可以折叠。ST 语言编写的程序也是循环扫描，在每个扫描周期，程序按照行号，顺序执行。代码可以从任意行号开始编写，执行时，从最小的行号开始。

3.2.2 注释

注释是对程序的解释，注释部分不参与编译，只要 PLC 支持，可以添加任何文字、符号。注释分单行注释和多行注释两种，多行注释使用“(＊　　＊)”和程序分离，这两个符号之间的部分就是注释，多行注释如图 3-5 所示。

```
(**********电机启停控制**********

    本部分程序用于主轴电机的启停控制

**************************************)
```

图 3-5　ST 语言的多行注释

从图 3-5 中可以看出，采用“(＊　＊)”包围的注释部分可以跨越多行，因此多用于一段程序的注释。注释部分不受编辑语言的限制，可以自由输入。需要注意的是，注释符号必须成对出现。只有“(＊”和“＊)”之间的部分才是注释，单独出现一个，被认为是错误的符号，导致编译报错，下面的书写方式会导致编译报错。

```
(*
IF  (xStart OR xRun) AND xStop THEN
      xRun:= TRUE;
ELSE
      xRun:= FALSE;
END_IF
(**************** 缺少"*)" ********************** )
(*
IF  (xStart OR xRun) AND xStop THEN
      xRun:= TRUE;
ELSE
      xRun:= FALSE;
END_IF
*)*)
(*************** 有两个"*)" ********************** )
```

"//"后面的也是注释,不过是单行注释,多用于某一条语句的注释。

注释,对于程序是非常必要的,特别是对于初学者,不要认为程序简单,就省略注释,要养成随时写注释的习惯。

注意:注释符号"(""(*"")""*""//"必须在英文标点、半角状态下输入。"//"是连续输入两次"/",中间不能有空格。

3.2.3　空语句

图 3-4 中,只有分号(;)的这一行,即行号 1 的语句,是空语句,它表示没有执行任何操作。在 ST 语言的语句中,除了一些特殊语句,所有的语句都以";"结尾。在三菱 GX Works3 中,新建的 ST 语言程序都以";"开头,可以不用理会,另起一行写程序,也可以利用这个分号当作语句的结尾,如图 3-6 所示。

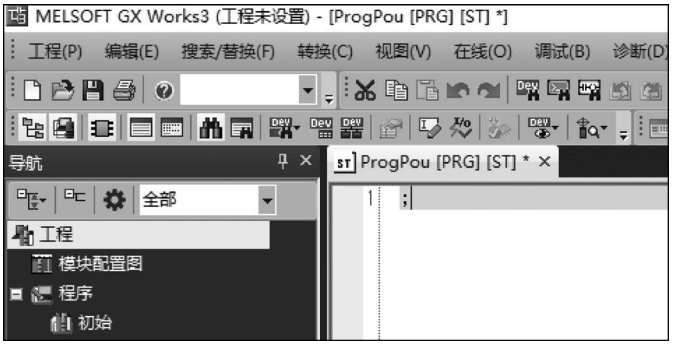

图 3-6　三菱 GX Works3 中的空语句

3.2.4　语句部分

语句是 ST 语言最主要的组成部分。图 3-4 中,关键字 IF 和关键字 END_IF 之间,即行

3～行 7，是 IF…ELSE…END_IF 语句，也称为选择语句，这是 ST 语言中的重要语句，程序的大部分功能，包括程序跳转等都是靠选择语句实现的。“xRun:= TRUE;”和“xRun:= FALSE;”这两句是赋值语句。

以上就是 ST 语言的基本组成。图 3-4 中的这段 ST 语言程序实现的就是梯形图中最经典的“启动-保持-停止”功能，即图 3-7 中的梯形图。

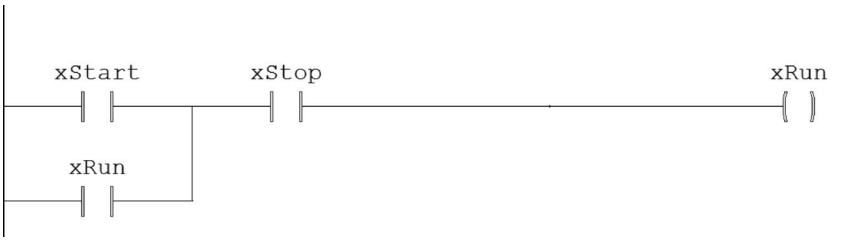

图 3-7　经典的“启动-保持-停止”梯形图

下面以赋值语句为切入点，介绍 ST 语言最基本的语句。

3.3　赋值语句

学习之前，先来看数学中最简单的计算式：“1＋1＝2”，这个计算式虽然简单，但数学中所有的运算都是以此为基础的。ST 语言也一样，复杂的语句都是由简单的语句组成的。“1＋1＝2”的实质是计算式子“1＋1”的值，并获取式子的计算结果 2。计算式包含了基本的数字“1”，对数字的操作“＋”，以及取得的结果等一系列要素。下面以此思路介绍 ST 语言中最基本的赋值语句。

3.3.1　语句组成

1. 操作数

操作数即“1＋1＝2”中的“1”，在 ST 语言中，操作数就是定义的变量。PLC 编程的实质是对变量的各种操作。

2. 操作符

操作符即“1＋1＝2”中的“＋”，也可称为运算符。操作符表示对变量的各种操作，也就是变量之间的各种运算。在 ST 语言中，有逻辑运算、数学运算和比较运算（也称为关系运算），共三种运算。

3. 表达式

把操作数用运算符连接起来就是表达式，即“1＋1”。根据参与运算的运算符，ST 语言有逻辑表达式、算术表达式和比较表达式。

4. 赋值

使用表达式的目的不是为了运算，而是为了取得运算结果，获取最终的结果才是程序的

目的。所以，表达式运算完成后，要把结果保存到某个变量中，这个保存的过程就是赋值。利用赋值运算符“:=”可以实现赋值运算，如下面的代码。

```
Data  := Data1 + Data2;
```

这句代码的含义是把变量 Data1 和变量 Data2 相加，并把计算的结果赋值给变量 Data。这样，由变量和运算符就组成了 ST 语言中最基本的语句——赋值语句。“;”用在赋值语句的结尾，表示语句的结束。“:=”是赋值运算符，中间不能有空格，否则编译会报错，如图 3-8 所示。

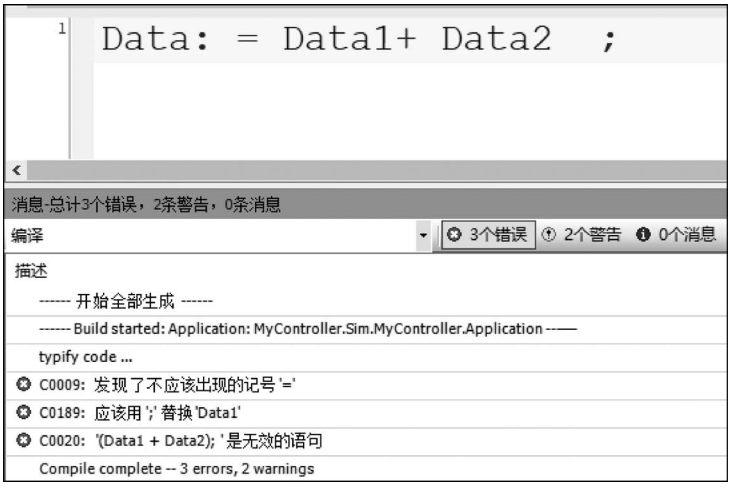

图 3-8　错误使用“:=”导致的编译报错

图 3-8 的程序中，在“:”和“=”之间加了空格，导致 PLC 认为这是两个运算符，不符合 ST 语言的规范，因此编译报错。

3.3.2　注意事项

赋值类似梯形图中的 MOVE 指令，它把“:=”右边表达式的运算结果或变量的值传递给“:=”左边的变量。因此，使用赋值语句，“:=”左边必须是变量，比如下面的代码。

```
Data + Data1 := Data2;
```

这行代码中，“:=”左侧是表达式，这是不符合规范的，编译会报错。“:=”右边可以是变量，也可以是表达式。赋值语句的实质是数据的传递，原则上要求“:=”两边变量的数据类型相同，并且“:=”左边变量的取值范围要大于“:=”右边变量的取值范围。这就好比小容器可以盛放的东西可以放入大容器，而大容器盛放的东西无法用小容器盛放。

原则上要求“:=”两边的数据类型要相等，因为不同的 PLC 对参与赋值运算的数据类型的要求是不同的，而记忆这些区别耗时耗力，却没有多少意义。例如，一个 INT 型的变量，赋值给 DWORD 型的变量，在 CODESYS 中是允许的，只是在编译时会警告“包含隐含转换，可能会改变符号”。而在三菱 GX Works3 中，是无法通过编译的。如果读者同时使用

多种 PLC 编程，记忆这些细微的规则会浪费不必要的精力。因此，笔者建议，参与赋值运算的变量，原则上要用相同的数据类型，在定义变量时，就要规划好变量的数据类型，特别是参与数学运算的变量。

3.4 赋值与相等

在数学中，“=”其实是有两种含义的：一种是计算式子的值；另一种是判断两个数是否相等。例如“1+1=2”，既可以理解为式子“1+1”的计算结果为 2，也可以理解为式子“1+1”的运算结果和数字 2 相等。但在工业控制中，计算结果和判断相等却是截然不同的运算。在 ST 语言中，其实就是表达式和语句的区别，判断相等是一种逻辑运算，用“=”表示；而赋值是一条语句。可以通过具体的例子理解，看下面的代码。

```
VAR
    xResults : BOOL;
    wData1   : WORD;
    wData2   : WORD;
END_VAR
xResults := ( wData1 = wData2);
```

这段代码是判断变量 wData1 和变量 wData2 的值是否相等，如果相等，变量 xResults 的值为 TRUE；否则变量 xResults 的值为 FALSE。可见，在 ST 语言中，用“=”判断两个变量是否相等，它是比较运算符；而赋值“:=”才是取运算结果。关于比较运算符，后续会有专门章节介绍。初学者非常容易混淆“=”和“:=”，希望读者注意，一个是判断是否相等，一个是赋值，用法完全不一样。这也是 ST 语言中所有运算符意义唯一性的体现。

在最后一行代码中，表达式“wData1 = wData2”加了括号，括号也是 ST 语言中的运算符，它和数学中四则混合运算中的括号一样，有括号的先算括号。在这行代码中，是否加括号对表达式的运算结果没有实质影响，但让代码的可读性大大提高。

在 ST 语言中，括号代表的是最高优先级。ST 语言中只有圆括号，括号可以嵌套，而且不限层数。关于运算优先级，后续章节会有详细介绍。对于初学者来说，勤加括号是非常有必要的，这是一个非常好的习惯，也是提高 ST 语言的可读性，让它更加直观易懂的重要手段和方法。

下面谈谈 ST 语言的编写技巧和方法。这对今后的学习和提高是非常有必要的。正所谓，磨刀不误砍柴工。

3.5 编写技巧和方法

从学习 ST 语言的第一条语句开始，就应当养成良好的编程习惯，以提高编程效率。编程效率的提高，对加快项目的进度至关重要，特别是随着自动化水平的提高，机械设备的控制越来越复杂，对程序的要求也越来越高，再加上项目进度的迫切要求，提高编程效率必须

放在第一位。在严格遵守 ST 语言的语法规则的前提下，必须采用科学的方法来提高效率，而科学的技巧和方法，也不是短时间内就能掌握的，需要不断地在实践中摸索和总结经验。下面笔者就分享一下个人经验，读者也可以借鉴各大互联网公司的代码规范，对 ST 语言的编写有着很强的参考意义。

3.5.1　缩进与对齐

缩进与对齐对于文本来说是很重要的规范。以 Word 为例，排版整齐的文章看上去让人赏心悦目，也有阅读下去的动力。一篇文章，就算内容写得再好，再吸引人，但排版非常糟糕，相信也没有多少人有继续阅读下去的兴趣。不妨来看一段 ST 语言编写的简单的代码，如图 3-9 所示。

图 3-9 中的这段程序错落有致，不同的层次结构相互对齐，先不关注这段代码的含义，至少排版能让人感到舒服。再看图 3-10 所示的样例。

```
IF xAuto THEN
  IF  xStart OR xRun AND xStop THEN
      xRun:=TRUE;
  ELSE
      xRun:=FALSE;
  END_IF
ELSE
    xRun:=FALSE;
END_IF
```

图 3-9　样例程序 1

```
   IF xAuto THEN
IF  xStart OR xRun AND xStop THEN
xRun:=TRUE;
     ELSE
      xRun:=FALSE;
END_IF
    ELSE
  xRun:=FALSE;
   END_IF
```

图 3-10　样例程序 2

图 3-10 中的这段程序，就让人失去了继续阅读的兴趣，只不过修改了它的排版，就变得杂乱。互联网行业的很多公司，对于代码中空格的数量甚至都有严格的规定，就是为了增强程序的可读性，方便日后维护。

缩进主要表示代码的层次结构，虽然是否使用缩进对编译效果没有影响，但使用缩进可以清晰地表达程序的层次结构，特别是存在嵌套的情况。如图 3-9 中的 IF 语句，同一层级的关键字 IF 跟关键字 IF_END 对齐，能方便看出它们的嵌套关系。ST 语言是一种文本语言，可以像使用 Word、文本文档一样使用 Tab 键，让语句对齐。

空格的最大作用是使语句不过于拥挤，而且排列整齐，有层次，提高可读性。例如下面这段代码。

```
diPosition   :=   REAL_TO_DINT(rlPosition/360 * rlRatio * 1000);
diVel        :=   REAL_TO_DINT(rlvel/360 * rlRatio * 1000);
```

由于赋值语句的变量 diPosition 和 diVel 的长度不同，适当地留点空格，对编译不会有任何影响。“:=”和变量以及表达式之间，也可以适当地留点空格，这样，程序看起来就非常清晰。注意，空格并不是随意增加的，关键字和变量之间必须以空格隔开，例如图 3-10 中的代码，第一行的关键字 IF 和变量 xAuto 之间必须留空格，否则 PLC 会把 IFxAuto 错认为

变量，编译会报错。

缩进和空格最大的作用，就是在符合语法规则的前提下，提高程序的可读性。再次提醒一下，由两个符号组成的运算符是不能加空格的，例如“:=”以及后面要学到的“>=”“<>”“<=”等。

3.5.2 快捷键

PLC 编程软件都是基于 Windows 操作系统的，因此，Windows 操作系统通用的快捷键，在 PLC 的编程软件中同样适用。常用的快捷键如下：

（1）Ctrl＋Z：撤销。

（2）Ctrl＋Y：恢复。

（3）Ctrl＋C：复制。

（4）Ctrl＋X：剪切。

（5）Ctrl＋V：粘贴。

以上快捷键都是 Windows 系统通用的，也是最常用的快捷键，这些快捷键对提高程序代码的编写速度有着很大的帮助。在开始学习 ST 语言编程时就应该尝试使用，并养成习惯。例如使用快捷键，可以方便地实现代码的复制粘贴，也可以使用“↑”（上）“↓”（下）“←”（左）“→”（右）键，将光标移动到需要复制的地方，如图 3-11 所示。

```
1  IF xAuto THEN
2    IF  xStart OR xRun AND xStop THEN
3        xRun:=TRUE;
4    ELSE
5        xRun:=FALSE;
6    END_IF
7  ELSE
8    xRun:=FALSE;
9  END_IF
10
```

图 3-11 移动光标

要选中图 3-11 中第 5 行代码中的“FALSE”，只需要按住 Shift 键，然后按“←”键，即可选中“FALSE”。选中的代码会高亮显示，如图 3-12 所示。

图 3-12 中，轻松使用快捷键就选中了需要的代码。此时，光标移动到了“FALSE”的最前面。利用 Shift 键和“↑”“↓”“←”“→”键，可以根据需要选中所要选的任何代码。如果需要选择整行代码，只需要把光标移动到这行最右边，然后按下 Shift 键和 Home 键即可。如果把光标移到这行最左边，按下 Shift 键和 End 键，也可以选中整行。常用的选择快捷键如下：

（1）Shift＋↑：选择本行前半部分的代码和上一行后半部分的代码。

（2）Shift＋↓：选择本行后半部分的代码和下一行前半部分的代码。

（3）Shift＋←：选择光标左边或上一行的一个字符。

```
IF xAuto THEN
  IF  xStart OR xRun AND xStop THEN
      xRun:=TRUE;
  ELSE
      xRun:=FALSE;
  END_IF
ELSE
  xRun:=FALSE;
END_IF
```

图 3-12　利用快捷键选中代码

(4) Shift+→：选择光标右边或下一行的一个字符。

(5) Shift+Home：选择光标前面的整行代码。

(6) Shift+End：选择光标后面的整行代码。

(7) Shift+PgUp：选择光标前面的整页代码。

(8) Shift+PgDn：选择光标后面的整页代码。

(9) Home：光标移动到本行最前面。

(10) End：光标移动到本行最后面。

(11) PgUp：光标移动到页首。

(12) PgDn：光标移动到页尾。

在选择大段代码时，使用快捷键比使用鼠标要方便得多。需要注意的是，14 寸的便携式计算机基本都没有小键盘，部分 15 寸的便携式计算机也没有。小键盘的很多功能都移到了其他按键上，特别是上、下、左、右键，要注意配合计算机键盘上的 Fn 键使用。具体可参阅操作说明。

3.5.3　注释

注释是对程序的合理解释，不但可以增强程序的可读性，还可以利用空白注释分割程序结构。合理的注释可以提高程序的可读性，复杂的注释容易成为累赘，因此注释应当简洁明了，让人一目了然，特别是一些关键部分的注释，便于日后查看。

注释是程序的一部分，不应该排斥它，更不要因为程序简单而忽略注释。当然，也不能为了注释而注释，喧宾夺主。

3.5.4　空语句和注释符号

在 3.2 节介绍 ST 语言的组成时，讲过空语句和注释符号。在编写代码时，可以充分利用它们的特性，分割大的程序段，提高程序代码的可读性。

在很多 PLC 中，新建的程序段都会自动加上“;”，可以利用这个符号分割程序层次，或者使用“;”把语句组合为块，实现类似 C# 语言中花括号“{}”的功能，当然也可以使用注释符号。看下面这段代码。

```
(****************** 电机控制 ******************** )
;
IF (A1_xStart OR A1_xRun) AND A1_xStop THEN
    A1_xRun := TRUE;          //A1 电机启动
ELSE
    A1_xRun := FALSE;         //A1 电机停止
END_IF;
;

//
IF (A2_xStart OR A2_xRun) AND A2_xStop THEN
    A2_xRun := TRUE;          //A2 电机启动
ELSE
    A2_xRun := FALSE;         //A2 电机停止
END_IF;
//
```

上面的代码实现的是两个电机的“启动-停止”控制，分别使用“;”和“//”把程序分隔成两个不同的块。这两个符号对程序的编译和执行没有影响，却大大提高了程序的可读性，无论是调试程序，还是后期维护修改，都能一目了然。也可以利用“(**********)”书写多行注释，把调试过程中遇到的问题、掌握的心得等记录下来，便于日后查看。总之，注释和空语句对于 PLC 编译器来说是没有意义的，但对于程序员来说，却是意义重大的利器。

3.5.5 变量命名

PLC 编程的目的是实现对变量的各种操作，因此，无论使用梯形图编程还是 ST 语言编程，都应当尽量使用变量而不是物理地址。

建议：不要使用物理地址！更不要使用变量和物理地址的混搭！

习惯使用传统 PLC 编程的读者，应当慢慢习惯使用变量，例如下面这段代码。

```
Data1 := Data3 + Data4;       //全部使用变量
Data2 := Data5 + %MD0;        //使用变量和物理地址混搭
Data3 := %MD0 + %MD1;         //全部使用物理地址
```

第 2 行代码使用了变量和物理地址的混搭，仅仅从视觉上就给人不舒服的感觉。更重要的是，使用物理地址在进行跨平台复制、粘贴时，会出现不必要的问题。例如，三菱 GX Works3 中，直接使用物理地址如 D0 参与各种运算，PLC 会把 D0 解析为 INT 型，这会给计算带来很大不便。使用变量，可以强制规定它的数据类型，能减少很多不必要的数据类型转换操作。

ST 语言最大的特点是不区分大小写，可以充分利用这个特点，合理科学地命名变量，提高程序的可读性。变量的命名，应当科学、合理、简洁，从名字上就能知道它的意义，这样才便于程序后期的维护及团队合作。

可以借鉴计算机高级语言中对变量的命名方法来命名 PLC 中的变量。一般有下面三种命名方法。

1. 骆驼命名法

骆驼(CAMEL)命名法,是指首个单词的首字母小写,其余单词的首字母大写,看上去像驼峰,高低起伏。例如,可以命名变量为 motorStart、motorStop、servoAlarmCode,分别表示启动电机、停止电机、伺服报警代码。

2. 帕斯卡命名法

与骆驼命名法不同,帕斯卡(PASCAL)命名法是将所有单词的首字母大写,例如 MotorStart、MotorStop、ServoAlarmCode 等。

3. 匈牙利命名法

匈牙利(HUNGARIAN)命名法采用“属性+描述”的方式来命名变量,也就是增加变量的属性。例如,可以命名变量 G_BOOL_MotorStart 和 L_WORD_ServoAlarmCode 分别表示“全局 BOOL 型变量电机启动”和“局部 WORD 型变量伺服报警代码”。在定义变量时,有些 PLC 也会自动增加一些属性,例如在西门子博途中,会在变量名中强制增加符号,“#MotorStart”和“MotorStart”虽然变量名相同,但增加了不同的符号,就表示这是不同类型的变量。这些符号是自动添加的,不需要用户干涉。笔者认为,这种处理方法非常好,读者可能觉得这是多此一举,当参与大型项目的编程和调试时,就会发现它的好处。

可以结合 PLC 编程及工业控制的特点,以及现场生产工艺,综合采用以上三种方法来命名变量。笔者推荐采用“变量属性+数据类型+描述”的方式,必要时可用下画线(_)分隔。笔者一般在变量属性和数据类型之间加下画线,而数据类型和描述之间不加。

1) 变量属性

变量属性前缀用来区分局部变量和全局变量,以及自定义功能块的输入/输出,使用小写字母,如表 3-1 所示。

表 3-1　变量属性前缀

变量属性	前　缀	变量属性	前　缀
VAR_GLOBAL	g	VAR_IN	i
VAR	l	VAR_OUT	q

局部变量和全局变量也可以不加前缀,或者为数量少的一类变量增加前缀,毕竟加前缀的目的是为了更快地区分两者,不是为了添加而添加。输入变量和输出变量建议增加前缀,在编写或者查看功能块程序时,能方便地知道某变量是输入变量还是输出变量,便于编程调试。

2) 数据类型

数据类型是变量的一个重要属性,也是在编程中必须要区分的,增加前缀是最好的办法。常见数据类型的前缀如表 3-2 所示。

表 3-2 常见数据类型的前缀

数据类型	前缀	数据类型	前缀
BOOL	x	TIME	t
INT	i 或 n	STRING	s
DINT	di 或 dn	ARRAY	a
WORD	w	STRUCT	st
DWORD	dw	POINTER	p
REAL	rl		

鉴于 STRING、ARRAY、STRUCT 及 POINTER 这几种数据类型的特殊性，可以不用加前缀；另外，INT 型变量使用前缀 i，容易与输入变量混淆，所以可以使用 n 来表示。当然，以上前缀并不是强制性规定，读者可以根据自己的习惯自行安排。很多 PLC 的编程手册中，也会有推荐的前缀符号，读者也可以参考。在定义数组时，也可以增加数组元素的数据类型，例如定义保存电机温度的数组，可以命名为 arlMotorTemp[0,9]。另外，时间型变量有 4 种，每一种也可以再增加它们的前缀区分。2.6.3 节介绍的 SoMachine 中常用的功能块和函数，就使用了表 3-2 中的前缀，读者可作为对照参考。

3）描述

描述是对变量表达含义的概括，也是最重要的部分，原则上采用动宾结构来描述控制对象的属性、动作、状态等。每个单词首字母大写，单词之间可以适当增加下画线，必要时，可以增加物理属性。总之，描述部分是变量命名的重点。当然，也不必拘泥于特定的形式，要根据实际情况灵活选用。

例如，定义 PLC 的输入/输出变量，这些变量总是和外部元件有着重要的联系。常见的按钮开关（SB）、限位开关（LI）、接近开关（SP）等，就可以使用标准的电气符号来表示，并和电气图纸一一对应。电机启动、电机停止等命令信号，电机运行、变频器故障等状态信号，以及碰左限位、急停按下等安全信号，都可以采用这种方式命名，例如 g_xMotorStart_SB1、l_diServoAlarmCode、q_rlServoPosition、q_xServoHomeSP2。当有多根轴时，可以增加轴号区分。部分单词长度较长，可以选用首字母或者缩写。如果这种方式非常冗长，可以直接使用元件名来命名变量，例如 SB1、SP2 等。这样，变量和输入/输出点及图纸的关系就一目了然，对调试和维修的帮助是非常大的。

此处重点介绍急停信号，它是非常重要的安全信号，可以命名为 E_STOP 或 E_STOP_SB0，全部使用大写字母，强调其重要性。限位及安全信号也可以使用这种命名方式。

有些物理量也可以直接使用它们的符号来命名，例如圆周率可以采用标准符号“PI”来命名；类似地，还有长度、体积、质量、速度、pH 值等；还有一些特殊的变量，从名称就可以知道它的数据类型的，也可以不用添加数据类型描述部分，例如定义的各种系统变量、上电第一次扫描、各种秒脉冲、常开、常闭等。另外，一些很难归类的或者约定俗成的及很难描述的变量，不必拘泥于既定的原则，可以多方面借鉴，例如，伺服电机的每转脉冲数，在三菱伺服驱动器中，该参数使用“FBP”表示，如果控制三菱伺服，就可以将变量命名为 l_wFBP。同

理，在施耐德伺服驱动器中，该参数使用“GEARratio”表示，如果控制施耐德伺服，就可以命名为 l_wGearRatio。在运动控制中，很多变量都可以使用此方法命名，可以方便地与被控的伺服建立联系，调试时就非常方便。

在自定义功能块时，输入/输出引脚可以借鉴并使用标准的 PLCopen 功能块的引脚，如图 3-13 所示。

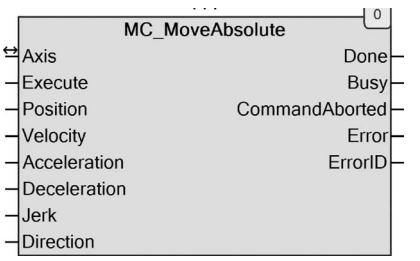

图 3-13　PLCopen 绝对定位功能块

图 3-13 是用于绝对定位的功能块，观察输入引脚，Execute 表示该功能块必须接收到上升沿信号时才会触发。如果自定义功能块需要上升沿信号触发，就可以使用 Execute。另外，还有接收高电平触发的引脚 Enable，也可以在自定义功能块时使用，表示该功能块必须接收到高电平信号才能触发。

再观察输出引脚，Done 表示功能块执行结束，无错误；Busy 表示功能块正在执行；CommandAborted 表示功能块的执行被中止；Error 表示功能块执行错误。这些引脚的含义都浅显易懂，在自定义功能块时可以直接使用，自定义的功能块和系统功能块就保持了统一。有些 PLC 可能把这些引脚名作为了系统的关键字，不允许直接使用，可以通过适当地增加前缀来解决，例如可以把 Busy 命名为 q_xBusy。当然，以上这些标准的引脚信号也可以用来命名变量，这样就使变量名和 PLC 中的功能块保持统一，例如伺服的定位位置可以命名为 AX1Position。不仅是这些标准的 PLCopen 功能块，不同的 PLC 也有大量的官方功能块，在自定义变量时都可以借鉴。

建议：原则上不要使用拼音命名变量，现在很多 PLC 支持中文变量名，对于英语不是很熟悉的读者，可以使用中文变量名。使用中文变量名的时候，一定要注意输入法的切换。变量的命名规则，是为方便编程服务的，而不是为了命名而命名，如果出现各种冗长的变量名，则本末倒置，失去了意义。

总之，科学合理地对变量命名，是一个不断完善的过程，需要在实践中不断总结。以上是笔者对变量命名的建议。在本书实例程序中，也将采用这样的原则命名。读者还可以根据自身情况，结合生产工艺和个人经验，制定命名原则。以上命名原则同样适用于自定义功能块、自定义函数、自定义 POU，以及任何需要命名的场合。对于自定义数据类型，例如结构体变量，由于它和 BOOL 型、INT 型等系统数据类型等价，可以全部使用大写，以达到统一。

对于 ST 语言中的关键字、函数和功能块、各种运算符等，例如 IF、AND、XOR 、CASE、FOR、TON、DINT_TO_REAL 等，一般采用大写。

总之，从一开始就养成良好的习惯，对于后续的学习非常重要。下面就开始 ST 语言学习之旅了。