第 1 章

PLC 编程基础

PLC 的英文全称为 Programmable Logic Controller，即可编程控制器。PLC 是一种数字运算的电子操作系统，它采用一种可编程的存储器，存储用户程序，执行定时、计数、逻辑运算、顺序控制以及算术运算等面向用户的指令，并通过数字或模拟式输入 / 输出控制各种类型的机械或生产过程。

本章主要介绍 PLC 的工作原理、西门子 S7-1200 PLC 硬件、S7-1200 PLC 中如何访问数据以及 PLC 的数据类型等内容。

1.1 PLC 概述

1.1.1 PLC 工作原理

PLC 的工作过程可分为三个阶段，分别是输入采样阶段、用户程序执行阶段和输出刷新阶段，完成这三个阶段称为一个扫描周期。在 PLC 运行期间，PLC 的 CPU 以一定的速度重复执行上述三个阶段。

1）输入采样阶段

PLC 以扫描方式按顺序依次读入所有输入端的信号，并将它们存入 IO 映像区中相应的单元内寄存起来。输入采样结束后，转入用户程序执行阶段，在这个阶段中，即使输入状态和数据发生变化，IO 映像区中相应单元的状态和数据也不会改变，输入状态的变化只能在下一个工作

周期的输入采样阶段才会被重新读入。注意，如果输入是脉冲信号，则该脉冲信号的宽度必须大于一个扫描周期，才能保证在任何情况下所有的输入均能被读入。

2）用户程序执行阶段

在这一阶段PLC按顺序依次扫描用户程序。如果程序使用的是梯形图，则扫描是按从上到下、从左到右的顺序进行。在这一阶段中对由触点构成的控制线路进行逻辑运算，然后根据逻辑运算的结果刷新该逻辑线圈在系统 RAM 存储区中对应位的状态，或者刷新该输出线圈在 IO 映像区中对应位的状态，或者确定是否要执行该梯形图所规定的特殊功能指令。

3）输出刷新阶段

当扫描用户程序运行结束后，PLC 就进入输出刷新阶段。在此期间，CPU 按照 IO 映像区内对应的状态和数据刷新所有的输出锁存电路，再经过输出电路驱动相应的外部设备，这才是 PLC 的真正输出。因此，同样的若干条梯形图，其排列次序不同，执行的结果也不同。

1.1.2 西门子 S7-1200 PLC 介绍

SIMATIC S7-1200（简称 S7-1200）控制器结构紧凑、组态灵活、功能强大，可完成高级逻辑控制、HMI（Human Machine Interface，人机界面）和各种网络通信等任务，是中小型自动化系统的完美解决方案。对于需要进行网络通信和多屏显示的自动化系统，S7-1200 PLC 无疑是最佳选择。

S7-1200 CPU 将微处理器、集成电源、输入和输出电路、内置 PROFINET、高速运动控制 IO 以及板载模拟量输入组合到一个设计紧凑的外壳中来形成功能强大的控制器。S7-1200 PLC 支持小型运动控制系统、过程控制系统的高级应用功能，具有功能强大的指令集，这些特点的组合使它成为控制各种应用的完美解决方案。

CPU 根据用户程序逻辑监视输入并更改输出，用户程序可以包含布尔逻辑、计数、定时、复杂数学运算以及与其他智能设备的通信。CPU 提供一个 PROFINET 端口用于网络通信以及用户程序逻辑监视，用户可以监视布尔逻辑、定时器、计数器、数学运算、复杂逻辑控制等。还可以通过添加通信模块来实现 PROFIBUS、GPRS、RS485 或 RS232 等通信。

1.2 硬件介绍

1.2.1 模块概述

S7-1200 PLC 控制系统包括 CPU 模块、通信模块、信号板和信号模块，如图 1-1 所示。

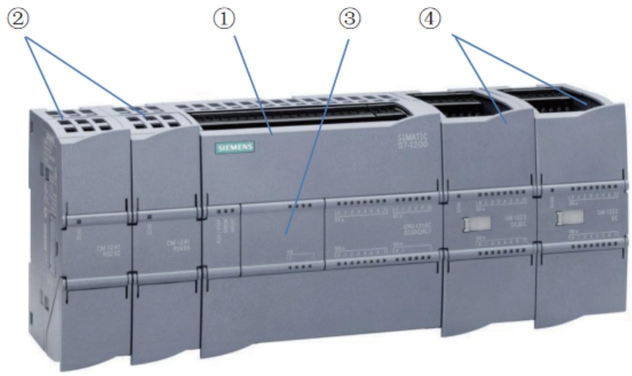


图 1-1　S7-1200 模块外观

说明：

① CPU（CPU 1211C、CPU 1212C、CPU 1214C、CPU 1215C、CPU 1217C）。

② 通信模块（CM）或通信处理器（CP）。

③ 信号板（SB）（数字 SB、模拟 SB）、通信板（CB）或电池板（BB）。

④ 信号模块（SM）（数字 SM、模拟 SM、热电偶 SM、RTD SM、工艺 SM）。

西门子 CPU 模块是一种超大规模集成电路，是计算和控制的核心，由控制单元、运算单元、存储单元等主要部分组成，它的基本任务是存储用户指令并执行输出。

通信模块是对 CPU 通信能力的扩展，用户可以配置不同的通信模块来完成多种通信任务，比如高速串行通信等。通信模块就像翻译器一样，把语言翻译成不同版本，在不同国家间建立交流。

信号板有两种，一种是扩展输入 / 输出模块，还有一种是扩展通信模块。信号板与通信模块和信号模块的功能一样，只是体积更小，不占空间，直接安装到 CPU 上，不影响控制器的实际大小。

信号模块是西门子为 CPU 设计的扩展模块，主要功能是增加输入和输出信号的数量，用户可以根据现场实际情况来控制设备数量，选配对应的信号模块。信号模块有不同的通道数量和功能，可以采集数字量、模拟量等。

1.2.2 CPU 模块功能

S7-1200 CPU 具有以下功能特点：

（1）有多种安全功能可用于保护 CPU 和对控制程序的访问。

（2）每个 CPU 都提供密码保护功能，用户可以通过该功能组态对 CPU 功能的访问权限。

（3）可以使用“专有技术保护”隐藏特定块中的代码。

（4）可以使用复制保护将程序绑定到特定存储卡或 CPU 中。

S7-1200 CPU 模块结构如图 1-2 所示。

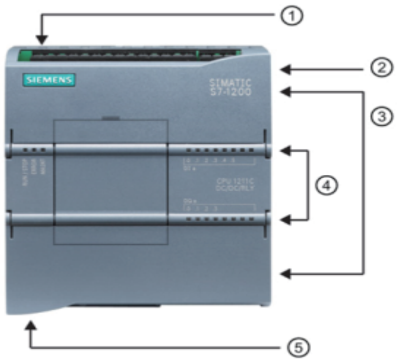


图 1-2 CPU 模块

说明：

① 可拆卸用户接线连接器。
② 存储卡插槽（上部保护盖下面）。
③ IO 状态的 LED。
④ PROFINET 连接器。
⑤ 可拆卸电缆连接。

1.2.3 CPU 选型

S7-1200 PLC 提供 5 种型号的 CPU，其技术规范如表 1-1 所示。

表1-1 S7-2.0 PLC各型号CPU的技术规范

<table>
<tr><th colspan="2">CPU型号</th><th>CPU 1211C</th><th>CPU 1212C</th><th>CPU 1214C</th><th>CPU 1215C</th><th>CPU 1217C</th></tr>
<tr><td>物理尺寸（mm）</td><td></td><td>90×100×75</td><td>90×100×75</td><td>110×100×75</td><td>130×100×75</td><td>150×100×75</td></tr>
<tr><td rowspan="3">用户存储器
负载能力
数据保持性内存</td><td>工作内存</td><td>50KB</td><td>75KB</td><td>100KB</td><td>125KB</td><td>150KB</td></tr>
<tr><td colspan="2">1MB
1MB</td><td colspan="3">4MB
4MB</td><td></td></tr>
<tr><td colspan="5">10KB</td><td></td></tr>
<tr><td rowspan="2">本地板载IO
模拟量</td><td>数字量</td><td>6路输入/
4路输出</td><td>8路输入/
6路输出</td><td colspan="3">14路输入/10路输出</td></tr>
<tr><td colspan="3">2路输入</td><td colspan="2">2路输入/2路输出</td><td></td></tr>
<tr><td rowspan="2">过程映像大小
输出（Q）</td><td>输入（I）</td><td colspan="5">1024字节</td></tr>
<tr><td colspan="5">1024字节</td><td></td></tr>
<tr><td colspan="2">位存储器（M）</td><td colspan="2">4096字节</td><td colspan="3">8192字节</td></tr>
<tr><td colspan="2">信号模块（SM）扩展</td><td>无</td><td>2</td><td colspan="3">8</td></tr>
<tr><td colspan="2">信号板（SB）、电池板（BB）
或通信板（CB）</td><td colspan="5">1</td></tr>
<tr><td colspan="2">通信模块（CM）（左侧扩展）</td><td colspan="5">3</td></tr>
<tr><td rowspan="5">高速计数器</td><td>总计</td><td colspan="5">最多可组态6个内置或SB输入的高速计数器</td></tr>
<tr><td>1MHZ</td><td colspan="4">--</td><td>Ib.2--Ib.5</td></tr>
<tr><td>100/80kHZ</td><td colspan="5">Ia.0--Ia.5</td></tr>
<tr><td>30/20kHZ</td><td>--</td><td>Ia.6 Ia.7</td><td colspan="2">Ia.6--Ib.5</td><td>Ia.6--Ib.1</td></tr>
<tr><td>200kHZ</td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td rowspan="4">脉冲输出</td><td>总计</td><td colspan="5">最多可组态4个内置或SB输出的脉冲输出</td></tr>
<tr><td>1MHZ</td><td colspan="4">--</td><td>Qa.0--Qa.3</td></tr>
<tr><td>100kHZ</td><td colspan="4">Qa.0--Qa.3</td><td>Qa.4--Qb.1</td></tr>
<tr><td>20kHZ</td><td>--</td><td>Qa.4 Qa.5</td><td></td><td>Qa.4--Qb.1</td><td>--</td></tr>
</table>

（续表）

CPU型号	CPU 1211C	CPU 1212C	CPU 1214C	CPU 1215C	CPU 1217C
存储卡	SIMATIC存储卡（可选）				
实时时钟保持时间	通常为20天，在高温（40℃）环境下最少可维持12天				
PROFINET端口	1			2	
实数数学运算执行速度	2.3us /指令				
布尔运算执行速度	0.08us /指令				

注：表中Ia.0--Ia.5表示高速计数器输入变量I的地址，例如I5.0--I5.5，其中，a表示数字0~9，最多6个。表中Qa.4--Qb.1表示脉冲输出变量Q的地址，例如Q1.4--Q2.1，其中a,b表示数字0~9，最多6个。

说明　对于具有继电器输出的 CPU 型号，必须安装数字量信号板（SB）才能使用脉冲输出。

每个 CPU 提供专用的 HMI 连接，支持最多 3 个 HMI 设备。支持的 HMI 总数受组态中 HMI 面板类型的影响。例如，可以将最多 3 个 SIMATIC 基本面板连接到 CPU，或者最多可以连接两个 SIMATIC 精智面板与一个附加基本面板。

不同的 CPU 型号提供了各种各样的特征和功能，这些特征和功能可帮助用户针对不同的应用创建有效的解决方案。

1.2.4 CPU 支持的块

不同型号的 CPU 支持的块、定时器和计数器不同。

S7-1200 PLC 各型号 CPU 支持的块如表 1-2 所示。

表1-2　S7-1200 PLC不同型号的CPU支持的块

元　素	说　明	
块	类型	OB、FB、FC、DB
	大小	30KB（CPU 1211C）50KB（CPU 1212C） 64KB（CPU 1214C和CPU 1215C）
	数量	一共多达1024个块（OB＋FB＋FC＋DB）
	FB、FC和DB的地址范围	1～65535（例如FB 1～FB 65535）
	嵌套深度	16（从程序循环或启动OB开始）；4（从延时中断、日时钟中断、循环中断、硬件中断、时间错误中断或诊断错误中断OB开始）
	监视	可以同时监视2个代码块的状态
OB	程序循环	多个：OB 1、OB 200～OB 65535
	启动	多个：OB 100、OB 200～OB 65535
	延时中断和循环中断	41（每个事件1个）：OB 200～OB 65535
	硬件中断（沿和HSC）	50（每个事件1个）：OB 200～OB 65535
	时间错误中断	1个 OB 80
	诊断错误中断	1个 OB 82

什么是块呢？ PLC 中存在两种程序，一种是 PLC 厂家固化在 PLC 内部的供 CPU 运行的系

统程序，另一种是用户自己编写的程序。而为了使所编写的程序的结构条理更加清晰和便于管理，就会用到各种程序块，我们将不同功能的程序写在不同的程序块中，当 PLC 开始运行时，CPU 就会按照程序需要运行的条件去调用相应的块来完成特定的任务，这就是块的功能。

S7-1200 CPU 中有组织块（OB）、函数块（又称功能块，FB）、函数（又称功能，FC）、数据块（DB）四种块，这四种块的功能和用法在第 5 章会有详细的介绍。

S7-1200 CPU 支持的定时器和计数器如表 1-3 所示。

表1-3 S7-1200 CPU支持的定时器和计数器

元　素	说　明	
定时器	类型	IEC
	数量	仅受存储器大小限制
	存储	DB结构，每个定时器16字节
计数器	类型	IEC
	数量	仅受存储器大小限制
	存储	DB结构，大小取决于计数类型 • SINT和USINT：3字节。 • INT和UINT：6字节。 • DINT和UDINT：12字节

延时中断和循环中断在 CPU 中使用相同的资源。延时中断和循环中断的总和只能为 4 个，不能有 4 个延时中断和 4 个循环中断。

1.3 PLC 数据访问

1.3.1 数据存储

西门子 STEP 7 系列编程软件简化了符号编程，用户可以为数据地址创建符号名称，作为与存储器地址和 IO 点相关的 PLC 变量或在代码块中使用的局部变量。要在用户程序中使用这些变量，只需输入用户创建的符号名称即可实现调用。例如用户定义 I0.0 的符号名称为“启动”，那么在程序中只需输入“启动”就可以调用 I0.0。

CPU 提供了以下几种方式，用于在执行用户程序期间存储数据：

- 全局储存器：CPU 提供了各种专用存储区，其中包括输入（I）、输出（Q）和位存储器（M）。所有代码块可以无限制地访问该储存器。
- PLC 变量表：在 STEP7 PLC 变量表中，可以输入特定存储单元的符号名称。这些变量在 STEP 7 程序中为全局变量，并允许用户在应用程序中使用自定义名称进行命名。
- 数据块（DB）：可在用户程序中加入 DB 以存储代码块的数据。从相关代码块执行开始直到结束，存储的数据始终存在。全局 DB 存储所有代码块均可使用的数据，而背景 DB 存储特定 FB 的数据并且由 FB 的参数进行构造。

- 临时存储器：只要调用代码块，CPU 的操作系统就会分配要在执行块期间使用的临时或本地存储器（L）。代码块执行完成后，CPU 将重新分配本地存储器，以用于执行其他代码块。每个存储单元都有唯一的地址。用户程序利用这些地址访问存储单元中的信息。对输入或输出存储区（例如 I0.3 或 Q1.7）的引用会访问过程映像。要立即访问物理输入或输出，需在引用后面添加“:P”（例如，I0.3:P、Q1.7:P 或 Stop:P）。

1.3.2 访问 PLC 中的数据

通常可在 PLC 变量（数据块）中创建变量，也可在 OB、FC 或 FB 顶部的接口中创建变量。这些变量包括名称、数据类型、偏移量和注释。另外，可以在数据块中指定起始值。在编程时，可以通过在指令参数中输入变量名称来使用这些变量，也可以选择在指令参数中输入绝对操作数（存储区、大小和偏移量）。程序编辑器会自动在绝对操作数前面插入 % 字符。以下各部分的实例介绍了如何输入绝对操作数。可以在程序编辑器中将视图切换到以下几种视图之一，即符号、符号和绝对，或绝对。

1 I（过程映像输入）

CPU 仅在每个扫描周期的循环 OB 执行之前对外围（物理）输入点进行采样，并将这些值写入输入过程映像。可以按位、字节、字或双字来访问输入过程映像。允许对过程映像输入进行读写访问，但过程映像输入通常为只读。I 存储器的绝对地址如表 1-4 所示。

表1-4　I存储器的绝对地址

位	I[字节地址].[位地址]	I0.1
字节、字或双字	I[大小][起始字节地址]	IB4、IW5或ID12

通过在地址后面添加“:P”，可以立即读取 CPU、SB 或 SM 的数字和模拟输入。使用 I_:P 访问与使用 I 访问的区别是，前者直接从被访问点而非输入过程映像获得数据。这种 I_:P 访问称为“立即读”访问，因为数据是直接从源而非副本获取的，这里的副本是指在上次更新输入过程映像时建立的副本。

因为物理输入点直接从与它连接的现场设备中接收值，所以不允许对这些点进行写访问，即与可读或可写的 I 访问不同的是，I_:P 访问是只读访问。

I_:P 访问也仅限于单个 CPU、SB 或 SM 所支持的输入大小（向上取整到最接近的字节）。例如，如果 2 DI/2 DQ SB 的输入被组态为从 I4.0 开始，则可按 I4.0:P 和 I4.1:P 形式或者按 IB4:P 形式访问输入点；不会拒绝 I4.2:P ～ I4.7:P 的访问形式，但没有任何意义，因为这些点未使用；但不允许 IW4:P 和 ID4:P 的访问形式，因为它们超出了与该 SB 相关的字节偏移量。

使用 I_:P 访问不会影响存储在输入过程映像中的相应值。I:P 存储器的绝对地址如表 1-5 所示。

表1-5 I:P存储器的绝对地址（立即）

位	I[字节地址].[位地址]:P	I0.1:P
字节、字或双字	I[大小][起始字节地址]:P	IB4:P、IW5:P或ID12:P

2 Q（过程映像输出）

CPU 将存储在输出过程映像中的值复制到物理输出点。可以按位、字节、字或双字访问输出过程映像。过程映像输出允许读访问和写访问。Q 存储器的绝对地址如表 1-6 所示。

表1-6 Q存储器的绝对地址

位	Q[字节地址].[位地址]	Q1.1
字节、字或双字	Q[大小][起始字节地址]	QB5、QW10、QD40

通过在地址后面添加“:P”，可以立即写入 CPU、SB 或 SM 的物理数字和模拟输出。使用 Q_:P 访问与使用 Q 访问的区别是，前者除了将数据写入输出过程映像外还直接将数据写入被访问点（写入两个位置）。这种 Q_:P 访问有时称为“立即写”访问，因为数据是被直接发送到目标点的，而目标点不必等待输出过程映像的下一次更新。

因为物理输出点直接控制与它连接的现场设备，所以不允许对这些点进行读访问。即与可读或可写的 Q 访问不同的是，Q_:P 访问为只写访问。

Q_:P 访问也仅限于单个 CPU、SB 或 SM 所支持的输出大小（向上取整到最接近的字节）。例如，如果 2 DI/2 DQ SB 的输出被组态为从 Q4.0 开始，则可按 Q4.0:P 和 Q4.1:P 形式或者按 QB4:P 形式访问输出点；不会拒绝 Q4.2:P ～ Q4.7:P 的访问形式，但没有任何意义，因为这些点未使用；但不允许 QW4:P 和 QD4:P 的访问形式，因为它们超出了与该 SB 相关的字节偏移量。

说明 指定绝对地址时，STEP 7 会为此地址加上“%”字符前缀，以指示它为绝对地址。编程时，可以输入带或不带“%”字符的绝对地址（例如 %I0.0 或 I.0）。如果忽略，则 STEP 7 将自动加上“%”字符。

使用 Q_:P 访问既影响物理输出，也影响存储在输出过程映像中的相应值。Q_:P 存储器的绝对地址如表 1-7 所示。

表1-7 Q_:P存储器的绝对地址（立即）

位	Q[字节地址].[位地址]:P	Q1.1:P
字节、字或双字	Q[大小][起始字节地址]:P	QB5:P、QW10:P或QD40:P

3 M（位存储区）

针对控制继电器及数据的位存储区（M 存储器）用于存储操作的中间状态或其他控制信息。可以按位、字节、字或双字访问 M 存储区。M 存储器允许读访问和写访问。M 寄存器的绝对地址如表 1-8 所示。

表1-8　M存储器的绝对地址

位	M[字节地址].[位地址]	M26.7
字节、字或双字	M[大小][起始字节地址]	MB20、MW30、MD50

CPU 在代码块启动（对于 OB）或被调用（对于 FC 或 FB）时为它分配临时存储器。为代码块分配临时存储器时，可能会重复使用其他 OB、FC 或 FB 先前使用的相同的临时存储单元。CPU 在分配临时存储器时不会对它进行初始化，因而临时存储器可能包含任何值。

临时存储器与 M 存储器类似，但有一个主要的区别：M 存储器在“全局”范围内有效，而临时存储器在“局部”范围内有效。

- M 存储器：任何 OB、FC 或 FB 都可以访问 M 存储器中的数据，也就是说这些数据可以全局性地用于用户程序中的所有元素。
- 临时存储器：只有创建或声明了临时存储单元的 OB、FC 或 FB 才可以访问临时存储器中的数据。临时存储单元是局部有效的，并且不会被其他代码块共享，即使在代码块调用其他代码块时也是如此。例如，当 OB 调用 FC 时，FC 无法访问对它进行调用的 OB 的临时存储器。

CPU 为三个优先级组 OB 中的每一个都提供了临时（本地）存储器，分别是：

- 高优先级组的临时存储器大小为 16 KB，用于启动和循环程序（包括相关的 FB 和 FC）。
- 中优先级组的临时存储器大小为 4 KB，用于标准中断事件（包括 FB 和 FC）。
- 低优先级组的临时存储器大小为 4 KB，用于错误中断事件（包括 FB 和 FC），只能通过符号寻址的方式访问该临时存储器。

4 DB（数据块）

DB 存储器用于存储各种类型的数据，其中包括操作的中间状态或 FB 的其他控制信息参数，以及许多指令（如定时器和计数器）所需的数据结构。可以按位、字节、字或双字访问数据块存储器。读 / 写数据块允许读访问和写访问，只读数据块只允许读访问。DB 存储器的绝对地址如表 1-9 所示。

表1-9　DB存储器的绝对地址

位	DB[数据块编号].DBX[字节地址].[位地址]	DB1.DBX2.3
字节、字或双字	DB[数据块编号].DB [大小][起始字节 地址]	DB1.DBB4、DB10.DBW2、DB20.DBD8

1.3.3 组态 IO

在组态画面中添加 CPU 和 IO 模块时，系统会自动分配 I 地址和 Q 地址。通过在组态画面中选择地址域并键入新编号，可以更改默认寻址设置，如图 1-3 所示。

无论模块是否使用所有的点，都按 8 点 / 组（1 字节）的方式分配数字量的输入和输出。模拟输入和输出按每组 2 点（4 字节）的方式进行分配。

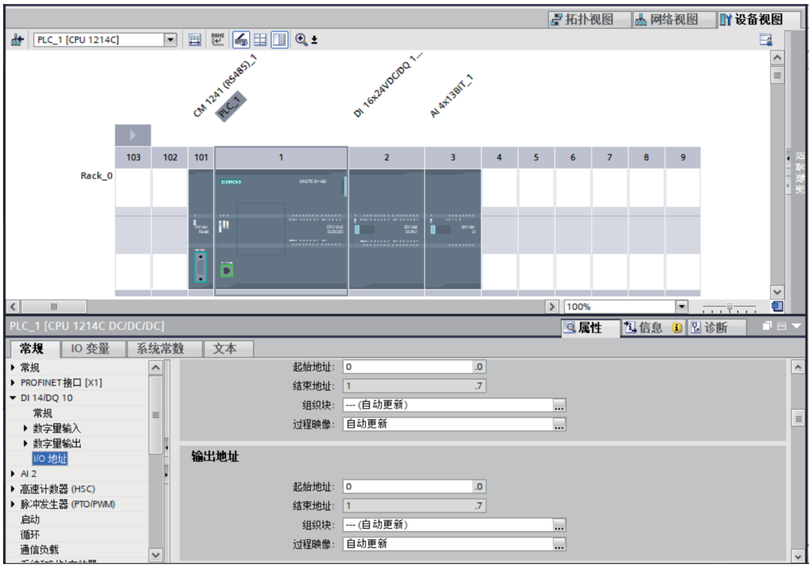


图 1-3 IQ 地址分配

模拟量信号模块可以提供输入信号，或等待表示电压范围或电流范围的输出值。这些范围是 ±10V、±5V、±2.5V 或 0 ～ 20mA。模块返回的值是整数值，其中，0 ～ 27648 表示电流的额定范围，-27648 ～ 27648 表示电压的额定范围。任何该范围之外的值即表示上溢或下溢。

在控制程序中，很可能需要以工程单位使用这些值，例如体积、温度、重量或其他数量值。要以工程单位使用模拟量输入，首先必须将模拟值标准化为 0.0 ～ 1.0 的实数（浮点）值，然后，将它标定为表示的工程单位的最小值和最大值。对于要转换为以工程单位表示的模拟量输出值，应首先将以工程单位表示的值标准化为 0.0 ～ 1.0 的值，然后将它标定为 0 ～ 27648 之间或 -27648 ～ 27648（取决于模拟模块的范围）的值。STEP 7 为此提供了 NORM_X 和 SCALE_X 指令，还可以使用 CALCULATE 指令来标定模拟值。

1.4 数据类型介绍

数据类型用于指定数据元素的大小以及解释数据。每个指令参数至少支持一种数据类型，而有些参数支持多种数据类型。将光标停留在指令的参数域上方，便可看到给定参数所支持的数据类型。

指令参数分形参和实参两种。形参指的是指令上标记该指令要使用的数据位置的标识符（例如 MOVE 指令 IN 输入）。实参指的是包含指令要使用的数据的存储单元（含“%”字符前缀）或常量（例如，%MW10）。用户指定的实参的数据类型必须与指令指定的形参所支持的数据类型匹配，否则会报错。

指定实参时，必须指定变量或者存储器地址。变量将符号名（变量名）与数据类型、存储区、存储器偏移量和注释关联在一起，并且可以在 PLC 变量编辑器或块（OB、FC、FB 和 DB）的

接口编辑器中进行创建。如果输入一个没有关联变量的绝对地址，那么使用的地址大小必须与所支持的数据类型相匹配，而默认变量将在输入时创建。

除 String（字符串类型）外，所有数据类型都可在 PLC 变量编辑器和块接口编辑器中使用，String 只能在块接口编辑器中使用。此外，还可以为许多输入参数输入常数值。

1.4.1 基本数据类型

基本数据类型包括位、位序列、整数、浮点数、字符串、日期和时间。

1 位和位序列

位和位序列如表 1-10 所示。

表1-10 位和位序列

位、位序列数据类型	位 大 小	数值类型	数值范围	常数实例	地址示例
Bool （布式型）	1	布尔运算	FALSE或TRUE	TRUE	I0.6 Q1.0 M50.5 DB1.DBX2.3 Tag_name
		二进制	2#0或2#1	2#0	
		无符号整数	0或1	1	
		八进制	8#0或8#1	8#1	
		十六进制	16#0或16#1	16#1	
Byte （字节型）	8	二进制	2#0～2#1111_1111	2#1001 _1100	IB2 MB10 DB1.DBB4 Tag_name
		无符号整数	0～255	13	
		有符号整数	−128～127	−55	
		八进制	8#0～8#377	8#17	
		十六进制	B#16#0～B#16#FF或 16#0～16#FF	B#16#F 16#F	
Word （字型）	16	二进制	2#0～2#1111_1111_1111_1111	2#1001_1000_0001_1011	MW10 DB1.DBW2 Tag_name
		无符号整数	0～65535	62032	
		有符号整数	−32768～32767	−200	
		八进制	8#0～8#177_777	8#132_167	
		十六进制	W#16#0～W#16#FFFF或 16#0～16#FFFF	W#16#F0F0 16#1FF9	
Dword （双字型）	32	二进制	2#0～2#1111_1111_1111_ 1111_1111_1111_1111_1111	2#1001_1000 _1100_1111_0001 _1011_0000_1100	MD10 DB1.DBD3 Tag_name
		无符号整数	0～4294967295	42333333	
		有符号整数	−2147483648～2147483647	−199999	
		八进制	8#0～8#37_777_777_777	8#132_167_326	
		十六进制	DW#16#0000_0000～DW #16#FFFF_FFFF或 16#0000_0000～ 16#FFFF_FFFF	DW#16#2A_F0F0 16#B_01F6	

2 整数数据类型

整数数据类型有六种，分别是无符号短整数类型（USINT）、有符号短整数类型（SINT）、无符号整数类型（UINT）、整数类型（INT）、无符号双整数类型（UDINT）和双整数类型（DINT）。这些数据类型的位大小和数值范围都不同，如表 1-11 所示。

表1-11 整数数据类型

数据类型	位 大 小	数值范围	常数示例	地址示例
USINT（无符号短整数类型）	8	0～255	78或2#01001110	MB0 DB1.DBB1 Tag_name
SINT（有符号短整数类型）	8	−128～127	50或16#60	
UINT（无符号整数类型）	16	0～65535	53300	MW3 DB1.DBW3 Tag_name
INT（整数类型）	16	−32768～32767	−300	
UDINT（无符号双整数类型）	32	0～4 294 967 295	42990	MD6 DB1.DBD6 Tag_name
DINT（双整数类型）	32	−2 147 483 648～ 2 147 483 647	−21648	

3 浮点数数据类型

实数（或浮点数）可以用 32 位单精度数“REAL”或 64 位双精度数“LREAL”表示。单精度浮点数的精度最高为 6 位有效数字，而双精度浮点数的精度最高为 15 位有效数字。

浮点数数据类型如表 1-12 所示。

表1-12 浮点数数据类型

数据类型	位 大 小	数值范围	常数示例	地址示例
REAL	32	−3.402823e+38～−1.175495e−38、 ±0、+1.175 495e−38～+3.402823e+38	123.456、−3.4、1.0e−5	MD100、 DB1.DBD8、 Tag_name
LREAL	64	−1.7976931348623158e+308～ −2.2250738585072014e−308、 ±0、 +2.2250738585072014e−308～ +1.7976931348623158e+308	12345.123456789e40、 1.2E+40	DB_name.var_name 规则： • 不支持直接寻址。 • 可在OB、FB或FC块接口数组中进行分配

4 字符串数据类型

在西门子 S7-1200 PLC 中，字符串作为复杂数据类型中的一种，是以字节为单位进行存储的，字符串复杂数据类型就由多个字节组成。字符串的存储格式分为 3 部分：第一部分是字符串最大长度，第二部分是字符串的实际长度，第三部分是字符串中的字符。

存储的逻辑过程：第一个地址用于存放字符串的最大字符个数，第二个地址用于存储字

符串中实际的字符格式，第三个地址用于存储字符串的第一个字节，第四个地址用于存储字符串中的第二个字节，后面以此类推。在实际运用中，一般会定义字符串的长度，比如定义一个 STRING 的数据类型 STRINGData[16]，它就是表示字符串的最大字符长度是 16 个字符。如果没有定义字符串的长度，那么默认值就是 254 字节。字符串数据类型如表 1-13 所示。

表1-13 字符串数据类型

总字符数	当前字符数	字符1	字符2	字符3	...	字符10
10	3	‘C’ （16#43）	‘A’ （16#41）	‘T’ （16#54）	...	—
字节0	字节1	字节2	字节3	字节4	...	字节11

5 时间和日期数据类型

时间和日期数据类型有三种，分别是 TIME、DATE（日期）和 TOD。

- TIME 数据作为有符号双整数存储，基本单位为毫秒。存储的数值是多少，就代表有多少毫秒。编辑时可以选择性地使用日期（d）、小时（h）、分钟（m）、秒（s）和毫秒（ms）作为单位。不需要指定全部时间单位，例如，T#5h10s 和 500h 均有效。所有指定单位值的组合值不能超过以毫秒表示的时间日期类型的上限或下限（−2 147 483 648ms ～ +2 147 483 647ms）。
- DATE 数据作为无符号整数值存储，被解释为添加到基础日期 1990 年 1 月 1 日的天数，用以获取指定日期。编辑器格式必须指定年、月和日。
- TOD（Time_of_day）数据作为无符号双整数值存储，被解释为自指定日期的凌晨算起的毫秒数（凌晨 =0ms）。必须指定小时（24 小时 / 天）、分钟和秒。可以选择性地指定小数秒格式。

时间和日期数据类型如表 1-14 所示。

表1-14 时间和日期数据类型

数据类型	大 小	范 围	常量输入示例
TIME	32位	T#−24d_20h_31m_23s_648ms～ T#24d_20h_31m_23s_647ms 存储形式：−2 147 438 648ms～ +2 147 183 647ms	T#5m_30s T#1d_2h_15m30s_45ms TIME#10d20h30m20s630ms 500h10000ms 10d20h30m20s630ms
DATE	16位	D#1990-1-1～D#2168-12-31	D#2009-12-31 DATE#2009-12-31 2009-12-31
TOD	32位	TOD#0:0:0.0～TOD#23:59:59.999	TOD#10:20:30.400 TIME_OF_DAY#10:20:30.400 23:10:1

1.4.2 结构数据类型（Struct）

Struct 类型是一种由多个不同数据类型元素组成的数据结构，其元素可以是基本数据类型，

也可以是 Struct、数组等复杂数据类型以及 PLC 数据类型（UDT）等。Struct 类型嵌套 Struct 类型的深度限制为 8 级。Struct 类型的变量在程序中可作为一个变量整体，也可单独使用组成该 Struct 的元素。Struct 类型可以在 DB 块、OB/FC/FB 接口区、PLC 数据类型（UDT）处定义。

Struct 数据类型的使用非常灵活，随时可以使用，但是相对于 PLC 数据类型（UDT）有以下缺点，因此建议需要使用 Struct 类型时，可以使用 PLC 数据类型（UDT）代替。

（1）维护成本增加，如果对一个 Struct 进行了多次复制，那么在更改过程中该 Struct 也必须进行相应的多次更改。

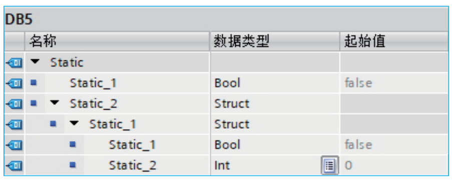
（2）Struct 与 PLC 数据类型（UDT）的相同结构不兼容。

（3）由于系统会检查所有结构元素的类型是否匹配，因而会导致性能下降。

（4）存储空间要求增加，每个 Struct 都是一个单独的对象，其描述信息将加载到 PLC 中。

应用示例

单独使用组成 Struct 的元素时，它和普通的变量没有区别，只是每出现一个 Struct 的嵌套层级，变量名就增加一个前缀，如图 1-4 所示。



DB5

名称	数据类型	起始值
Static		
Static_1	Bool	false
Static_2	Struct	
Static_1	Struct	
Static_1	Bool	false
Static_2	Int	0

图 1-4 Struct 的定义

Struct 在程序中的使用如图 1-5 所示。

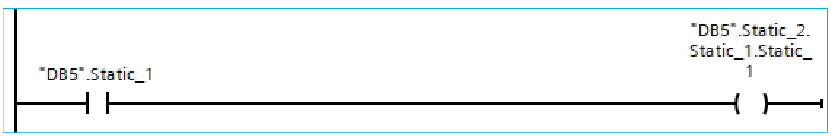


图 1-5 Struct 的使用

Struct 可以将同一种类型的变量放在一起，利于区分，如图 1-6 所示。

DB6

名称	数据类型	起始值
Static		
面粉	Real	0.0
黄油	Real	0.0
盐	Real	0.0
水	Real	0.0
蛋糕	Real	0.0
面包	Real	0.0
甜甜圈	Real	0.0

DB6

名称	数据类型	起始值
Static		
原料	Struct	
面粉	Real	0.0
黄油	Real	0.0
盐	Real	0.0
水	Real	0.0
成品	Struct	
蛋糕	Real	0.0
面包	Real	0.0
甜甜圈	Real	0.0

图 1-6 增加 Struct 类型的结构

1.4.3 PLC 数据类型（UDT）

UDT 类型是一种由多个不同数据类型元素组成的数据结构，元素可以是基本数据类型，也可以是Struct、数组等复杂数据类型以及其他UDT等。UDT类型嵌套UDT类型的深度限制为8级。

UDT 类型可以在 DB 块、OB/FC/FB 接口区处使用。从 TIA 博途 V13SP1、S7-1200 V4.0 开始，PLC 变量表中的 I 和 Q 也可以使用 UDT 类型。

UDT 类型可在程序中统一更改和重复使用，一旦某 UDT 类型发生修改，则执行软件全部编译后，并自动更新所有使用该数据类型的变量。

定义为 UDT 类型的变量在程序中可作为一个变量整体使用，也可单独使用组成该变量的元素。此外还可以在新建 DB 块时直接创建 UDT 类型的 DB，该 DB 只包含一个 UDT 类型的变量。

UDT 类型作为整体使用时，可以与 Variant、DB_ANY 类型及相关指令默契配合。理论上来说，UDT 是 Struct 类型的升级替代，功能基本完全兼容 Struct 类型。

下面我们来看看在博途编程软件中创建和使用 UDT 的方法。

（1）新建 UDT。在 CPU 菜单中依次单击“PLC 数据类型”→“添加新数据类型”选项，添加“用户数据类型 _1”，如图 1-7 所示。

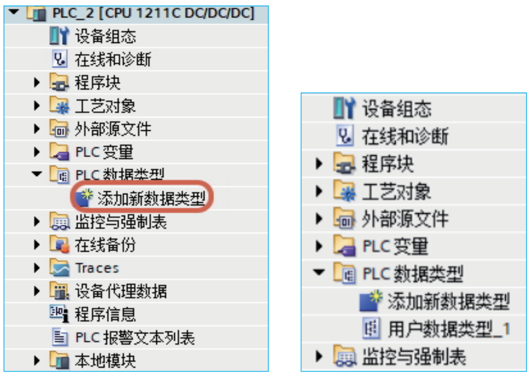


图 1-7　新建 UDT

（2）在弹出的“用户数据类型 _1”页面中可以添加需要的变量、类型、起始值、注释等，如图 1-8 所示。

图 1-8　定义 UDT 内的变量

（3）右击“用户数据类型”，在弹出的快捷菜单中选择“属性”→“常规”，在“常规”界面中可以修改该数据类型的名称，如图 1-9 所示。

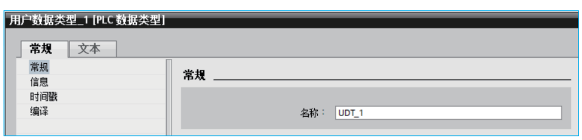


图 1-9　修改 UDT 名称

（4）创建好的 UDT 可以直接在 DB 块中调用，如图 1-10 所示。

数据块_2

	名称	数据类型	起始值	保持	可从 HMI/...	从 H...	在 HMI ...	设定值	注
1	▼ Static			☐	☐	☐	☐	☐	
2	▼ RCV	"UDT_1"		☐	☑	☑	☑	☐	
3	start	Bool	false	☐	☑	☑	☑	☐	
4	stop	Bool	false	☐	☑	☑	☑	☐	
5	Temputer	Int	0	☐	☑	☑	☑	☐	
6	pressure	Real	0.0	☐	☑	☑	☑	☐	

图 1-10 DB 块中调用 UDT

选择数据类型 UDT_1，则上面创建好的变量就直接调用到数据块中了。

（5）UDT 在变量表中的应用。

应用示例

将 PROFINET IO 通信中的 I 区数据存入 DB 块中，将 DB 块中的数据再传给 Q 区。

步骤 01 新建 I 区和 Q 区的两个 UDT，如图 1-11 和图 1-12 所示。

I区

	名称	数据类型	默认值
1	aa	Bool	false
2	bb	Bool	false
3	cc	Real	0.0
4	dd	DInt	0
5	ff	DInt	0
6	gg	DInt	0

图 1-11 I 区定义的 UDT

Q区

	名称	数据类型	默认值
1	A	Bool	false
2	B	Bool	false
3	C	Real	0.0
4	D	Int	0

图 1-12 Q 区定义的 UDT

步骤 02 PLC 变量表定义变量类型，如图 1-13 所示。

步骤 03 DB 块中定义变量类型，如图 1-14 所示。

变量表_1

	名称	数据类型	地址
1	▼ 通讯区_I区	"I区"	%I0.0
2	aa	Bool	%I0.0
3	bb	Bool	%I0.1
4	cc	Real	%ID2
5	dd	DInt	%ID6
6	ff	DInt	%ID10
7	gg	DInt	%ID14
8	▼ 通讯区_Q区	"Q区"	%Q0.0
9	A	Bool	%Q0.0
10	B	Bool	%Q0.1
11	C	Real	%QD2
12	D	Int	%QW6

图 1-13 PLC 变量表中调用已定义好的 UDT

DB块

	名称	数据类型	起始值
1	▼ Static		
2	▼ receive	"I区"	
3	aa	Bool	false
4	bb	Bool	false
5	cc	Real	0.0
6	dd	DInt	0
7	ff	DInt	0
8	gg	DInt	0
9	▼ send	"Q区"	
10	A	Bool	false
11	B	Bool	false
12	C	Real	0.0
13	D	Int	0

图 1-14 DB 块中调用已定义好的 UDT

步骤 04 通过 MOVE 指令完成数据的传输，如图 1-15 所示。

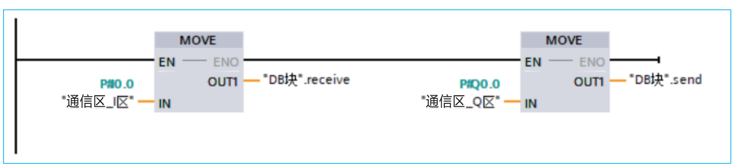


图 1-15 数据传送程序

1.4.4 数组数据类型（ARRAY）

1 ARRAY 类型的定义及注意事项

ARRAY 类型是由数目固定且数据类型相同的元素组成的数据结构。ARRAY 类型的定义和使用需要注意以下几点：

（1）ARRAY 类型可以在 DB 块、OB/FC/FB 接口区、PLC 数据类型处定义。无法在 PLC 变量表中定义。

（2）无法在 PLC 变量表中定义。

数组定义的格式如下：

```
Array[ 维度 1 下限 .. 维度 1 上限 , 维度 2 下限 ... 维度 2 上限 ,...]of < 数据类型 >
```

最多可包含 6 个维度。

（3）数组元素的数据类型包括除数组类型、Variant 类型以外的所有类型；数组下标的数据类型为整数，下限值必须小于或等于上限值，上下限的限值在 S7-1200 V3.0 及其以前为 INT 范围（−32768 ～ +32767），在 S7-1200 V4.0 及其以后为 DINT 范围（−2147483648 ～ +21474836487），可以使用局部常量或全局常量定义上下限值，数组的元素个数受 DB 块剩余空间大小以及单个元素大小的限制。

从 S7-1200 V2.0 开始，下标可以是常数、常量，也可以变量，还可以混合使用（多维数组），如果编程语言是 SCL 的话，下标还可以是表达式。使用数组的变量下标，可以在程序中很容易地实现间接寻址。注意，下标变量必须是符号名，不能是 DB1.DBW0 这种没有对应符号名的绝对地址。

（4）从 S7-1200 V4.2 开始，FC 的 Input/Output/InOut 以及 FB 的 InOut 可以定义形如 Array[*] 的变长数组，要求必须是优化 FC/FB 块，在调用 FC/FB 的实参中可以填写任意数据类型相同的数组变量；当然，也可以是多维变长的数组，例如 Array[*,*]of Int。

（5）数组可以使用单个数组元素（例如 "DB1".Static_1[1]），也可以使用整个数组（例如 "DB1".Static_1）。多维数组可以降维使用，例如三维数组 3D[0..2,0..3,0..4]of Int 是一个 3×4×5 大小的 INT 数组，3D[0] 是一个 4×5 大小的二维 INT 数组，3D[0,1] 是一个 5 个元素的一维 INT 数组，当然多维数组下标也可以换成变量，例如 3D[Tag_1,1]。

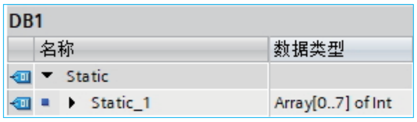
（6）从 S7-1200 V4.2 开始，多重背景支持数组形式，即 Array of FB，这样可以在 FB 中使用循环指令更方便地编写程序。不支持 Array[*] of FB。

> **注意** 在 TIA 博途 V10.5 SP2、S7-1200 V1.0 的时候，曾经引入 FieldRead 和 FieldWrite 指令用于数组下标的变址寻址，这种方法在 TIA 博途 V11 之后可以由下标变量完全取代，并且更为简化，所以该指令也只是位于“移动操作”→“原有”中，用于早期版本向上移植时使用，它的使用方法参见编程软件帮助。

S7-1200 数组的基本使用如下：

（1）在 DB 中创建数组，如图 1-16 所示。

（2）在 FC 的 InOut 中创建数组变量，如图 1-17 所示。



DB1

名称	数据类型
▼ Static	
▶ Static_1	Array[0..7] of Int

图 1-16 在 DB 中创建数组

FC1

名称	数据类型
▶ Input	
▶ Output	
▼ InOut	
▶ InOut_1	Array[0..7] of Int

图 1-17 在 FC 中创建数组变量

（3）在 FC 中使用数组元素（使用形参），如图 1-18 所示。

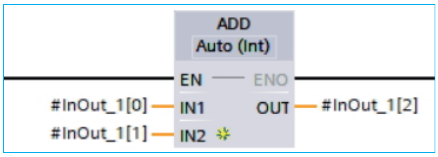
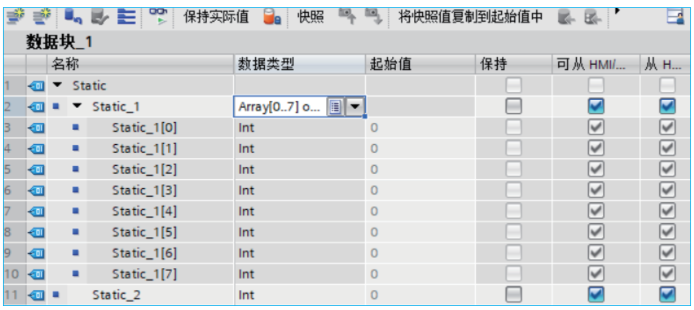


图 1-18 FC 中调用数组

2 实例编程

创建数组，实现以下功能：将 0 ～ 7 这 8 个数字传入创建的数组中。

步骤 01 创建数据块，添加数组变量，如图 1-19 所示。



保持实际值 快照 将快照值复制到起始值中

数据块_1

	名称	数据类型	起始值	保持	可从 HMI/...	从 H...
1	▼ Static			☐	☐	☐
2	▼ Static_1	Array[0..7] o...		☐	☑	☑
3	Static_1[0]	Int	0	☐	☑	☑
4	Static_1[1]	Int	0	☐	☑	☑
5	Static_1[2]	Int	0	☐	☑	☑
6	Static_1[3]	Int	0	☐	☑	☑
7	Static_1[4]	Int	0	☐	☑	☑
8	Static_1[5]	Int	0	☐	☑	☑
9	Static_1[6]	Int	0	☐	☑	☑
10	Static_1[7]	Int	0	☐	☑	☑
11	Static_2	Int	0	☐	☑	☑

图 1-19 添加变量

步骤 02 编程，程序段如图 1-20 ～图 1-22 所示。

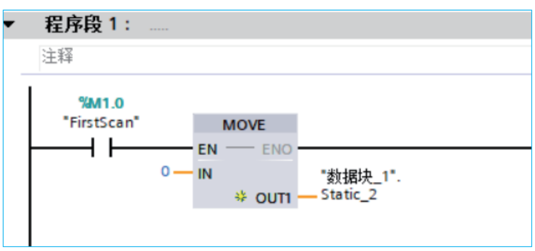


图 1-20 程序段 1

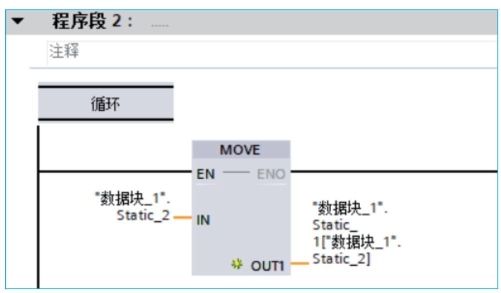


图 1-21 程序段 2

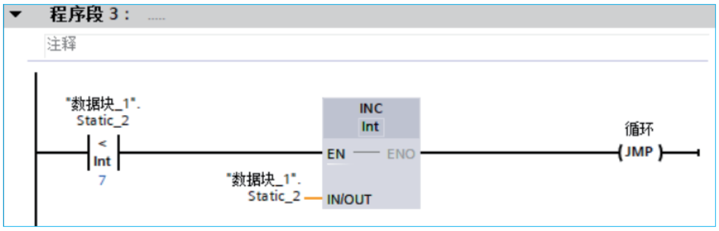


图 1-22　程序段 3

步骤 03　开启仿真，查看程序运行结果，如图 1-23 所示。

	i	名称	地址	显示格式	监视值
1		"数据块_1".Static_2		带符号十进制	7
2		"数据块_1".Static_1[0]		带符号十进制	0
3		"数据块_1".Static_1[1]		带符号十进制	1
4		"数据块_1".Static_1[2]		带符号十进制	2
5		"数据块_1".Static_1[3]		带符号十进制	3
6		"数据块_1".Static_1[4]		带符号十进制	4
7		"数据块_1".Static_1[5]		带符号十进制	5
8		"数据块_1".Static_1[6]		带符号十进制	6
9		"数据块_1".Static_1[7]		带符号十进制	7

图 1-23　数据监控

1.4.5 系统数据类型（SDT）

系统数据类型是由系统提供的具有预定义的结构，结构由固定数目的具有各种数据类型的元素构成，不能更改该结构。系统数据类型只能用于特定指令。可以在 DB 块、OB/FC/FB 接口区使用。

常见的系统数据类型如表 1-15 所示。

表1-15　常见的系统数据类型

系统数据类型	字节长度	说　明
IEC_TIMER	16	定时器结构。此数据类型用于TP、TOF、TON、TONR、RT和PT指令
IEC_SCOUNTER	3	计数值为SINT数据类型的计数器结构。 此数据类型用于CTU（加计数器）、CTD（减计数器）和CTUD（加减计数器）指令
IEC_USCOUNTER	3	计数值为USINT数据类型的计数器结构。 此数据类型用于CTU、CTD和CTUD指令
IEC_COUNTER	6	计数值为INT数据类型的计数器结构。 此数据类型用于CTU、CTD和CTUD指令
IEC_UCOUNTER	6	计数值为UINT数据类型的计数器结构。 此数据类型用于CTU、CTD和CTUD指令
IEC_DCOUNTER	12	计数值为DINT数据类型的计数器结构。 此数据类型用于CTU、CTD和CTUD指令

（续表）

系统数据类型	字节长度	说　明
IEC_UDCOUNTER	12	计数值为UDINT数据类型的计数器结构。 此数据类型用于CTU、CTD和CTUD指令
ERROR_STRUCT	28	编程错误信息或IO访问错误信息的结构 此数据类型用于GET_ERROR指令
CREF	8	数据类型ERROR_STRUCT的组成，在其中保存有关块地址的信息
NREF	8	数据类型ERROR_STRUCT的组成，在其中保存有关操作数的信息
VREF	12	用于存储VARIANT指针 此数据类型用在运动控制工艺对象块中
CONDITIONS	52	用户自定义的数据结构，定义数据接收的开始和结束条件。 此数据类型用于RCV_CFG指令
NREF	8	指定用来存储通过UDP实现开放用户通信的连接说明的数据块结构。 此数据类型用于TUSEND和TURCV指令
TCON_Param	64	指定用来存储那些通过工业以太网实现开放用户通信的连接说明的数据块结构。 此数据类型用于TSEND和TRCV指令
HSC_Period	12	使用扩展的高速计数器，指定时间段测量的数据块结构。 此数据类型用于CTRL_HSC_EXT指令

表中的部分数据类型还可以在新建 DB 块时直接创建系统数据类型的 DB，通过这种方法建立出的 DB 块可以配合 DB_ANY 类型使用。

此外通过以下方式也可以实现建立系统数据类型 DB：

（1）定时器计数器自动生成的背景 DB。

（2）计数器自动生成的背景 DB。

（3）开放式用户通信程序建立的基于 TCON_IP_V4（无法通过添加新块的方式创建）、TADDR_Param、TCON_Param 的 DB 块。

根据 SDT 定义 DB 块如图 1-24 所示。

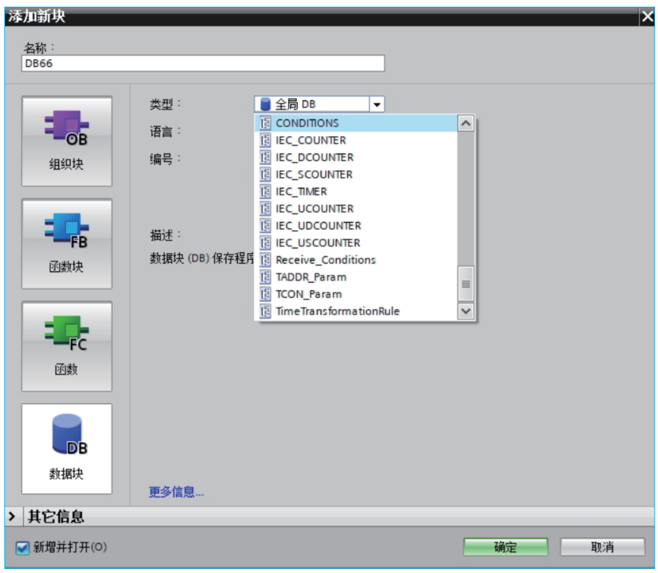


图 1-24　建立 SDT 类型的 DB

1.4.6　参数数据类型（Variant）

Variant 类型是一个参数数据类型，只能出现在除 FB 的静态变量以外的 OB/FC/FB 接口区。

Variant 类型的实参是一个可以指向不同数据类型变量的指针，它既可以指向基本数据类型，也可以指向复杂数据类型、UDT 等。Variant 数据类型的操作数不占用背景数据块或工作存储器中的空间，但是将占用 CPU 上的装载存储器的存储空间。调用某个块时，可以将该块的 Variant 参数连接任何数据类型的变量（除了传递变量的指针外，Variant 还会传递变量的类型信息）。在调用的块中可以利用 Variant 的相关指令，将它识别出并进行处理。

Variant 指向的实参可以是符号寻址，也可以是绝对地址寻址，还可以是形如 P#DB1.DBX0.0 BYTE 10 这种指针形式的寻址。

P# 指针说明：当 Variant 类型的实参指向形如 P#DB1.DBX0.0 BYTE 10 的指针时，指令内部将判定该形参为一个 10 字节的数组。P#DB1.DBX0.0 BYTE 10 这种结构起源于 S7-300/S7-400 的 Any 指针，S7-1200 无法像 S7-300/S7-400 一样定义以及拆解 Any 指针，但是在参数类型为 Variant 时，可以输入这种指针，并且如前所述，S7-1200 将识别它为数组。

P#DB1.DBX0.0 BYTE 10 的解释：

（1）指向从 DB1.DBX0.0 开始的 10 字节，并且 DB1 必须是非优化的 DB 块，并包含有 10 字节长度的变量。

（2）P#DB1.DBX 位置可以替换成其他 DB 块号，例如 P#DB10.DBX，或者 I 区 P#I、Q 区 P#Q、M 区 P#M。

（3）0.0 的位置为这种指针的起始地址，例如 1.0，100.0，…，并且小数点后一定是 0。

（4）BYTE 位置可以是以下类型：BOOL、BYTE、WORD、DWORD、INT、DINT、REAL、CHAR、DATE、TOD、TIME 类型。

（5）10 的位置为指针执行前面数据类型的个数，BOOL 类型比较特殊，只能是 1 或者 8 的倍数。P# 指针举例：P#I0.0 Bool 8，P#Q0.0 Word 20，P#M100.0 Int 50。

在早期版本的 TIA 博途软件中，只有一些通信指令使用 Variant 变量。从 TIA V13SP1 和 S7-1200 V4.0 开始，可以在程序块实参中定义 Variant 类型变量，并且还可以通过以下指令处理 Variant 类型的变量。

（1）判断类指令（见表 1-16），该类指令的作用是检查 Variant 类型的实参的实际类型，并不直接参与处理。

表1-16　判断类指令

LAD（梯形图）	SCL（结构化语言）	位　置
EQ_Type	TypeOf	基本指令→比较操作→变量
NE_Type		
EQ_ElemType	TypeOfElements	
NE_ElemType		
IS_NULL		
NOT_NULL		
IS_ARRAY	IS_ARRAY	
CountOfElements	CountOfElements	基本指令→移动操作→变量

（2）处理类指令（见表 1-17），该类指令可以对 Variant 类型的实参进行转化。

表1-17 处理类指令

LAD/SCL	位 置
Deserialize	基本指令→移动操作
Serialize	
MOVE_BLK_VARIANT	
VariantGet	基本指令→移动操作→变量
VariantPut	

（3）其他指令：DB_ANY_TO_VARIANT 与 VARIANT_TO_DB_ANY。

DB_ANY_TO_VARIANT 指令可以将 DB_ANY 转换为 Variant，从符合要求的数据块中生成 Variant 变量。IN 参数的操作数可以使用数据类型 DB_ANY，在创建程序时不需要知道数据块，将在运行时读取数据块编号。

VARIANT_TO_DB_ANY 指令可以将 Variant 转换为 DB_ANY。输入 IN 端的参数可以是背景数据块或 ARRAY 数据块。IN 参数的操作数可以是数据类型 Variant，在创建程序时不需要知道将被查询编号的数据块的数据类型。在运行期间将会读取数据块编号，并将它写入 RET_VAL 参数指定的操作数中。

1.4.7 日期和时间数据类型（DTL）

1 DTL 概述

日期和时间数据类型 DTL 的操作数长度为 12 字节，用于存储日期和时间信息。

表 1-18 列出了 DTL 数据类型的属性。

表1-18 日期和时间数据类型

长 度	格 式	值 范 围	输入值示例
12	日期和时间（年-月-日-小时-分钟-秒-纳秒）	最小值：DTL#1970-01-01-00:00:00.0 最大值：DTL#2262-04-11-23:47:16.854775807	DTL: 2008-12-16-20:30:20.250

DTL 数据类型的结构由几个部分组成，每一部分都包含不同的数据类型和取值范围。指定值的数据类型必须与相应元素的数据类型相匹配。表 1-19 给出了 DTL 数据类型的结构组成及其属性。

表1-19 DTL的结构组成及其属性

字 节	组 件	数据类型	值 范 围
0	年	UINT	1970～2262
1			

（续表）

字　节	组　件	数据类型	值 范 围
2	月	USINT	1～12
3	日	USINT	1～31
4	星期	USINT	1（星期日）～7（星期六）
5	小时	USINT	0～23
6	分钟	USINT	0～59
7	秒	USINT	0～59
8	纳秒	UDINT	0～999999999
9			
10			
11			

2 系统 / 本地时间区别

系统时间（System Time）：格林威治标准时间。

本地时间（Local Time）：根据 S7-1200 CPU 所处时区设置的本地标准时间。

例如，北京时间与系统时间相差 8 小时。

在 CPU 属性中进行时间设置如图 1-25 所示。

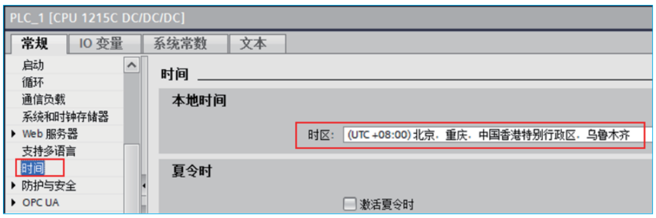


图 1-25 时间设定

3 读取 S7-1200 CPU 的系统 / 本地时钟

（1）调用读取 S7-1200 CPU 的系统 / 本地时钟的指令，如图 1-26 所示。

（2）要读取 S7-1200 CPU 的系统 / 本地时钟，需要在 DB 块中创建数据类型为 DTL 的变量，如图 1-27 所示。

图 1-26 调用时钟指令

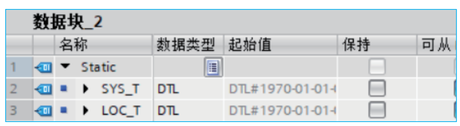


图 1-27 DB 中添加变量

（3）在 OB1 中编程，读出的系统 / 本地时间通过输出管脚“OUT”放入数据块相应的变量中，如图 1-28 所示。

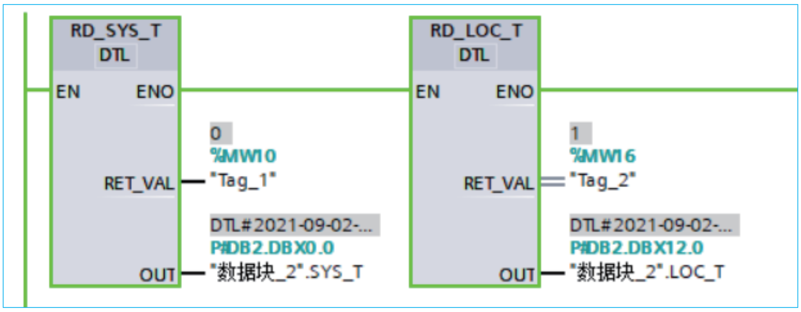


图 1-28 读取系统时间和本地时间

（4）通过变量监控可以查看实时数据，如图 1-29 所示。

	i	名称	地址	显示格式	监视值	强制值
1		"数据块_2".LOC_T	P#DB2.DBX12.0	DATE_AND_TIME	DTL#2021-09-0...	
2		"数据块_2".LOC_T.HOUR	%DB2.DBB17	无符号十进制	22	
3		"数据块_2".SYS_T	P#DB2.DBX0.0	DATE_AND_TIME	DTL#2021-09-0...	
4		"数据块_2".SYS_T.HOUR	%DB2.DBB5	无符号十进制	14	
5			<新增>			

图 1-29 变量监控

从图 1-29 中可以看出，读出的系统时间和本地时间相差 8 小时，这是因为 S7-1200 CPU 所设置的时间与格林威治时间相差 8 小时。

4 实例编程

使用本地时钟实现以下功能：

（1）设定每天晚上 7 点开灯，早上 7 点 30 分 10 秒关灯。

（2）设定 2021-10-01-09:00:00 执行 PLC 停机。

编程第一步：晚上 19:00 关灯，如图 1-30 所示。

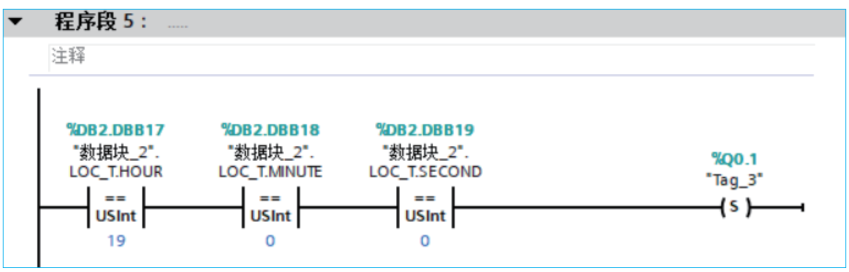


图 1-30 晚上 19:00 开灯

编程第二步：早晨 7:30:10 关灯，如图 1-31 所示。

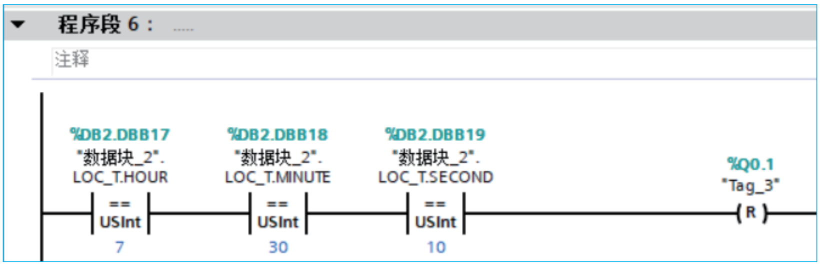


图 1-31 早晨 7:30:10 关灯

编程第三步：PLC 定时停机，如图 1-32 所示。

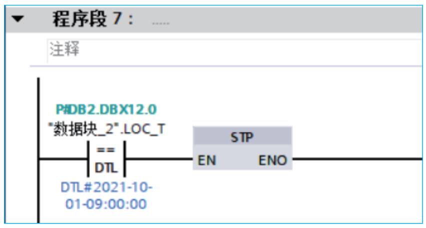


图 1-32 定时将 PLC 停机

1.4.8 指针数据类型

指针数据类型（Pointer、Any 和 Variant）可用于 FB 和 FC 代码块的块接口表中。可以从块接口数据类型下拉列表中选择指针数据类型，也可以将 Variant 数据类型用作指令参数。

1 Pointer 指针数据类型

Pointer 指针数据类型指向特殊变量，它会在存储器中占用 6 字节（48 位），可能包含以下信息：

DB 编号或 0（如果该数据未存储在 DB 中）

CPU 存储区中的变量地址如图 1-33 所示。

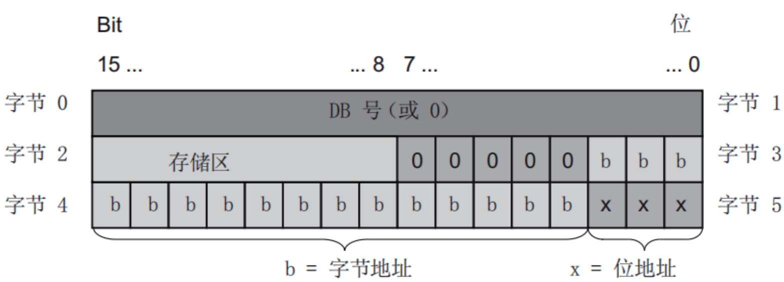


图 1-33 变量地址

可以使用指令声明以下三种类型的指针：

- 区域内部的指针：包含变量的地址数据。
- 跨区域指针：包含存储区中数据以及变量地址数据。

- DB 指针：包含数据块编号以及变量地址。

三种指针类型如表 1-20 所示。

表1-20 指针类型

类　型	格　式	示例输入
区域内部的指针	P#Byte.Bit	P#20.0
跨区域指针	P#Memory_area_Byte.Bit	P#M20.0
DB指针	P#Data_block.Data_element	P#DB10.DBX20.0

2 Any 指针数据类型

Any 指针数据类型指向数据区的起始位置，并指定其长度。Any 指针使用存储器中的 10 字节，可能包含以下信息：

- 数据类型：数据元素的数据类型。
- 重复因子：数据元素数目。
- DB 号：存储数据元素的数据块。
- 存储区：CPU 中存储数据元素的存储区。
- 起始地址：数据的“Byte.Bit”起始地址。

Any 指针的结构如图 1-34 所示。

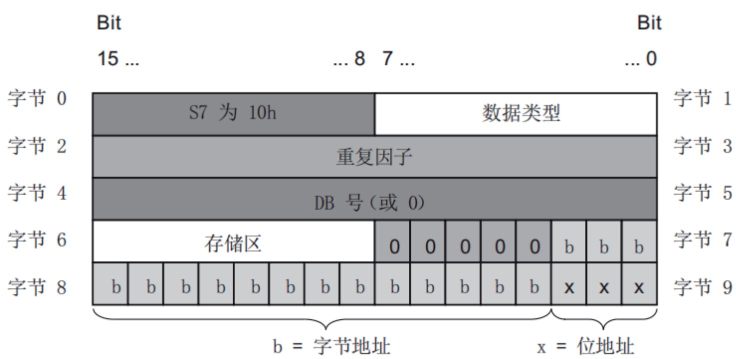


图 1-34 Any 指针结构

指针无法检测 Any 结构，只能将它分配给局部变量。

Any 指针的格式如表 1-21 所示。

表1-21 Any指针格式

格　式	条目示例	说　明
P#Data_block.Memory_area Data_address类型号	P#DB 11.DBX20.0 INT10	全局DB11中从DBX20.0开始的10字节
P#Memory_area Data_address 类型号	P#M 20.0 BYTE 10	从M20.0开始的10字节
	P#I 1.0 BOOL 1	输入I1.0

3 Variant 指针数据类型

Variant 指针可以指向结构和单独的结构元素，不会占用存储器的任何空间。Variant 指针的属性如表 1-22 所示。

表1-22 Variant指针属性

长　度	表示方式	格　式	示例输入
0	符号	操作数	MyTag
		DB_name.Struct_name.element_name	MyDB.Struct1.pressure1
	绝对	操作数	%MW10
		DB_number.Operand Type Length	P#DB10.DBX10.0 INT12