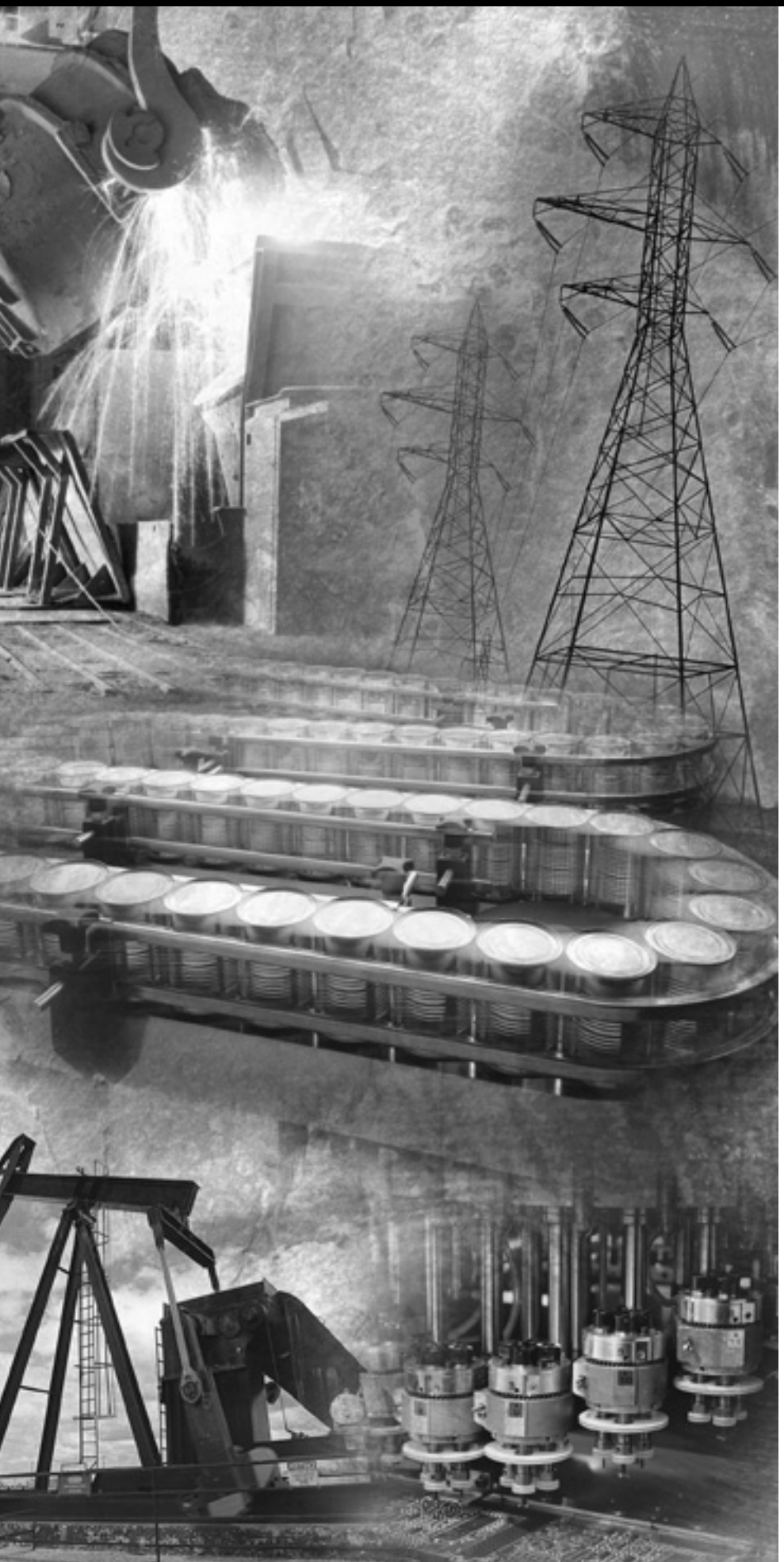


Allen-Bradley

Logix5000™ 控制器常用过程

1756 ControlLogix®,
1769 CompactLogix™,
1789 SoftLogix™,
1794 FlexLogix™,
PowerFlex 700S with DriveLogix

编程手册

Rockwell Automation

重要用户信息

固态设备具有与机电设备不同的运作特性。*固态控制的应用、安装和维护安全准则*（出版物 SGI-1.1，可从当地 Rockwell Automation 销售处或者从 http://www.ab.com/manuals/gi 联机获得）说明了固态设备与硬连接机电设备之间的重要差别。由于存在这些区别，同时由于固态设备的广泛应用，负责应用此设备的所有人员都必须确保仅以可接受的方式应用此设备。

对于由于使用或应用此设备而导致的任何直接或间接的损害，Rockwell Automation, Inc. 在任何情况下都不承担任何责任。

本手册中的示例和图表仅供说明之用。由于任何特定的安装都存在很多差异和要求，Rockwell Automation, Inc. 对于依据这些示例和图表所进行的实际应用不承担任何责任和义务。

对于本手册中所述信息、电路、设备或软件之使用，Rockwell Automation, Inc. 不负担专利责任。

未经 Rockwell Automation, Inc. 书面许可，任何单位或个人不得复制本手册之全部或部分内容。

在整本手册中，我们在必要的地方做出了说明，以告知您安全注意事项。

警告

指明在危险环境下可能导致爆炸进而造成人身伤害或死亡、财产损坏或经济损失的行为或情况的信息。

重要

指明成功应用和理解产品的关键信息。

注意

指明可能造成人身伤害或死亡、财产损坏或经济损失的行为或情况的信息。“注意”帮助您：

- 确定危险情况
- 避免发生危险
- 了解可能的后果

触电危险

标签可能会位于驱动器上或内部以警告可能存在危险电压。

燃烧危险

标签可能会位于驱动器上或内部以警告表面温度可能很危险。

介绍

该版本的文档包括了新增和更改的信息。为了找到这些内容，特将有变化的项目列于下段。

更新信息

该文档包括如下更新信息：

- 由于适用的信息在相关的控制器用户手册中介绍，本手册中的内容有所减少
  - 1756-UM001 ControlLogix 控制器用户手册
  - 1769-UM007 CompactLogix 控制器用户手册
  - 1794-UM001 FlexLogix 控制器用户手册
- 数组下标超出范围的信息 （请参阅第 2-27 页）。
- 支持事件性任务的附加模块 （请参阅第 3-24 页）。
- 关于选择 SFC 重启位置的信息 （请参阅第 5-21 页）。
- 在线编辑 SFC 时考虑的问题 （请参阅第 5-22 页）。
- 关于 S:FS 的信息 （请参阅第 14-2 页）。
- 附加的主故障代码 （请参阅第 15-15 页）。
- CompactFlash 卡的其他使用 （请参阅第 17-17 页）。
- RSI 安全服务器的全局操作 （请参阅第 18-15 页）。

注释：

本手册的目的

本手册指导为 Logix5000™ 控制器开发项目。它提供有关如何执行以下任务的逐步过程，这些过程对所有 Logix5000 控制器通用：

- 组织任务、程序和例程
- 组织标记
- 设计顺序流程图
- 使用梯形逻辑、功能块图、流程图或结构化文本编程语言编写例程
- 与其他控制器通信
- 通信和处理 ASCII 信息
- 处理故障

Logix5000 控制器 一词指任何基于 Logix5000 操作系统的控制器，例如：

- CompactLogix™ 控制器
- ControlLogix® 控制器
- DriveLogix™ 控制器
- FlexLogix™ 控制器
- SoftLogix5800™ 控制器

本手册的适用对象

本手册面向使用 Logix5000 控制器编写应用程序的人员，例如：

- 软件工程师
- 控制工程师
- 应用程序工程师
- 设备工具技术人员

何时使用本手册

执行以下操作时使用本手册：

- 为应用程序开发基础代码
- 修改现有应用程序
- 执行应用程序的隔离测试

将应用程序与系统中的 I/O 设备、控制器和网络集成时：

- 请参阅您的特定类型控制器的用户手册。
- 在需要时使用本手册作为参考。

如何使用本手册

使用本手册时，您将看到一些与其它正文格式不同的术语：

文本：	标识：	例如：	含义：
斜体	屏幕上或示例中项的实际名称	右击 *User-Defined* …	右击名为 User-Defined 的项。
粗体	“术语表”中的条目	键入**名称** …	如果需要额外信息，请参见“术语表”中的**名称**。 如果您正在浏览本手册的 PDF 文件，请单击**名称**跳转至术语表条目。
courier	必须根据应用提供的信息（变量）	右击 *name_of_program* …	必须标识应用程序中的特定程序。通常，这是您已定义的名称或变量。
括在方括号内	键盘按键	按 [Enter]。	按 Enter 键。

与 I/O 通讯

章 1

组织标记

章 2

管理任务

章 3

章 4

设计顺序流程图

章 5

编程流程图

章 8

功能块编程

章 9

与其他设备通信

章 10

生成大数组

章 11

与 ASCII 设备通信

章 12

处理 ASCII 字符

章 13

强制逻辑单元

章 14

访问状态信息

章 15

处理主故障

章 16

监视次故障

章 17

使用非易失性内存存储和加载项目

章 18

确保项目安全

章 19

确定控制器内存信息

附录 A

管理多个消息

附录 B

向多个控制器发送消息

附录 C

符合 IEC61131-3 标准

Notes:

与 I/O 通讯

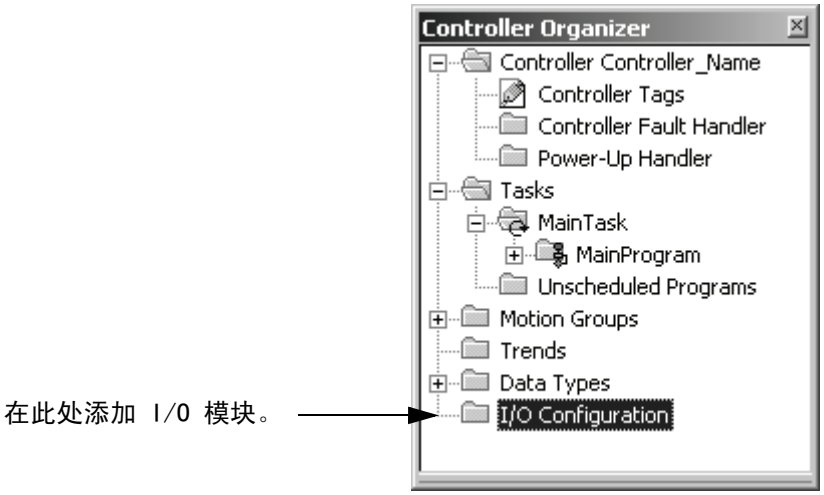
使用本章

本章提供 Logix5000 控制器如何与 I/O 模块通讯的基本信息。

获取该信息或步骤	请参阅此页：
配置 I/O 模块	1-1
I/O 数据寻址	1-7
缓存 I/O	1-8

配置 I/O 模块

要与用户系统中的 I/O 模块通讯，用户需向控制器的 I/O 配置文件夹中添加模块。



用户添加模块时，同时为模块定义特定的配置。虽然不同模块的配置选项不同，不过有一些是用户典型配置的共同选项：

- 请求信息包间隔
- 通讯格式
- 电子钥匙

请求信息包间隔

Logix5000 控制器使用连接传送 I/O 数据。

术语：	定义：
连接	两个设备间的通讯链接，如控制器和 I/O 模块、PanelView 终端或其它设备。 连接是资源分配，与非连接的消息相比，能够为设备提供更可靠的通讯。 单个控制器连接数有限。 用户通过配置控制器与其它系统设备之间的通讯，间接确定控制器连接。 下面的通讯类型使用连接： • I/O 模块 • 生成的和使用的标记 • 某些类型的消息 (MSG) 指令 （并非所有类型都使用连接）
请求数据包间隔 (RPI)	RPI 指定连接中数据更新时间。 例如，输入模块以用户分配给模块的 RPI 周期向控制器发送数据。 • 一般以毫秒 (ms) 为单位配置 RPI。 范围为 0.2 ms （200 微秒）至 750 ms。 • 如果通过 ControlNet 网络连接设备，RPI 在 ControlNet 网络数据流中保留时槽。 此时槽的计时与精确 RPI 值可能不相符，但是控制系统至少可以按指定的 RPI 速率传输数据。

在 Logix5000 控制器中，用户通过项目的 I/O 配置文件夹配置 I/O 值的更新周期。值更新与逻辑执行异步。在特定间隔中，控制器独立于逻辑控制的执行更新值。

注意

任务执行全过程中，注意确保数据内存包括合适值。用户可以在扫描一开始复制或缓存数据，为逻辑控制提供引用值。

- 任务中的程序直接从控制器域内存访问输入和输出数据。
- 任何任务中的逻辑可以修改控制器域数据。
- 数据和 I/O 值异步，可以在任务执行过程中更改。
- 任务开始执行时引用的输入值与后来引用的不同。
- 要在扫描过程中防止输入值的更改，将值复制到其它标记，并从该位置使用数据 （缓存值）。要缓存 I/O 值，请参阅 页 1-8。

通讯格式

用户选择的通讯格式确定模块所关联标记的数据结构。很多 I/O 模块支持不同的格式。每种格式使用不同的数据结构。用户选择的通讯格式也确定：

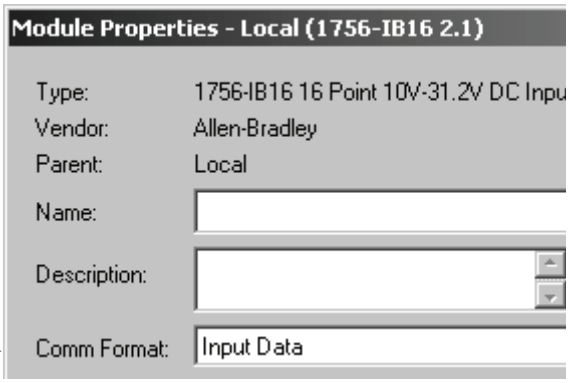
- 直接或机架优化连接
- 所有权

直接或机架优化连接

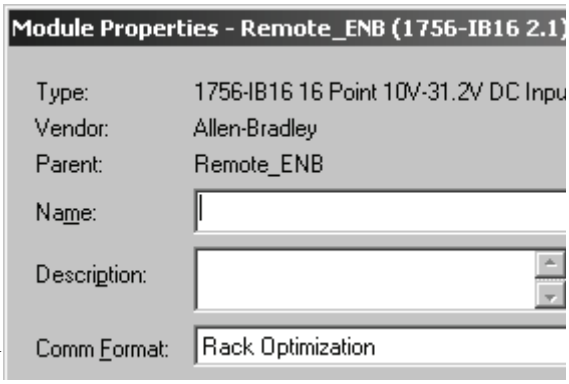
Logix5000 控制器使用连接传送 I/O 数据。 这些连接可以是直接连接或机架优化连接。

术语：	定义：
直接连接	直接连接是在控制器和 I/O 模块之间的实时数据传送链接。控制器保持并监视与 I/O 模块的连接。任何连接中断，如模块故障或在有电时卸下模块，都会在与模块相关的数据区中设置故障位。
机架优化连接	对于数字 I/O 模块，用户可以选择机架优化通讯。机架优化连接合并控制器和机架（或 DIN 导轨）中所有数字 I/O 模块之间的连接。每个 I/O 模块不是单独、直接连接，而是整个机架 （或 DIN 导轨）建立一个连接。

直接连接是一种不 使用机架优化通讯格式的连接）。



机架优化连接



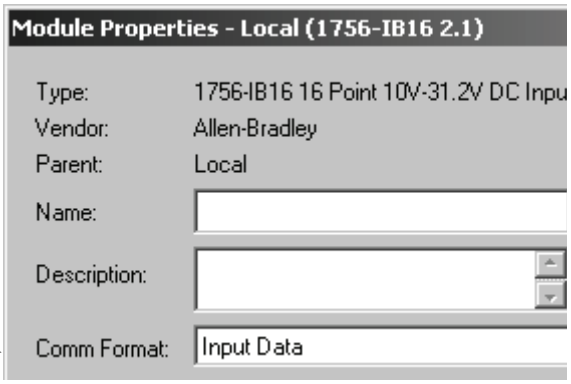
所有权

在 Logix5000 系统中，模块多点传送数据。意思是多个设备可以同时接收来自单个设备的相同数据。

用户选择通讯格式时，还要选择与模块建立所有关系还是仅侦听关系。

所有者控制器

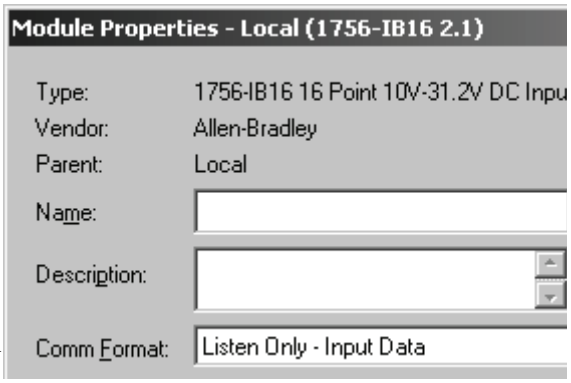
为模块创建主要配置和通讯连接的控制器。所有者控制器写入配置数据，并可以与模块建立连接。



所有者连接是一种不包括仅侦听通讯格式的连接方法。

仅侦听连接

在 I/O 连接中其它控制器拥有 / 提供 I/O 模块配置数据。使用仅侦听连接的控制器仅监视模块。它不写入配置数据，且仅能在所有者控制器积极控制 I/O 模块时与 I/O 模块保持连接。



仅侦听连接

使用下表为模块选择所有权类型：

如果此模块是：	并且另一控制器：	用户想：	则使用下面的连接类型：
输入模块	不拥有模块	→	所有者 （即，非仅侦听）
	拥有模块	如果与其它控制器断开通讯，则保持与模块的通讯	所有者 （即，非仅侦听） 在其它所有者控制器上使用相同的配置。
		如果与其它控制器断开通讯，则停止与模块的通讯	仅侦听
输出模块	不拥有模块	→	所有者 （即，非仅侦听）
	拥有模块	→	仅侦听

控制输入模块和控制输出模块存在显著的不同。

控制：	关系：	说明：
输入模块	所有者	输入模块由建立所有者连接的控制器配置。该配置控制器是第一个建立所有者连接的控制器。 一旦输入模块被配置 （被控制器拥有），则其它控制器可以与该模块建立所有者连接。这允许额外所有者在原所有者控制器与模块断开连接时，继续接收多点传送数据。所有其它额外所有者必须与原所有者控制器具有相同的配置数据和通讯格式，否则连接被拒绝。
	仅侦听	一旦输入模块被配置 （被控制器拥有），则其它控制器可以与该模块建立仅侦听连接。这些控制器可以在其它控制器拥有模块时接收多点传送数据。如果全部所有者控制器都与输入模块断开连接，则所有具有仅侦听连接的控制器不再接收多点传送数据。
输出模块	所有者	输出模块由建立所有者连接的控制器配置。输出模块仅允许一个所有者连接。如果其它控制器试图建立所有者连接，则连接被拒绝。
	仅侦听	一旦输出模块被配置 （被控制器拥有），则其它控制器可以与该模块建立仅侦听连接。这些控制器可以在其它控制器拥有模块时接收多点传送数据。 如果所有者控制器与输出模块断开连接，则所有具有仅侦听连接的控制器不再接收多点传送数据。

电子钥匙

注意

用户禁用电子钥匙时要特别注意。如果使用不当，该选项可以导致人员伤亡、财产损坏或经济损失。

用户配置模块时，指定模块槽号。但是，也可能会在该槽中置入不同模块，不管是故意还是意外。

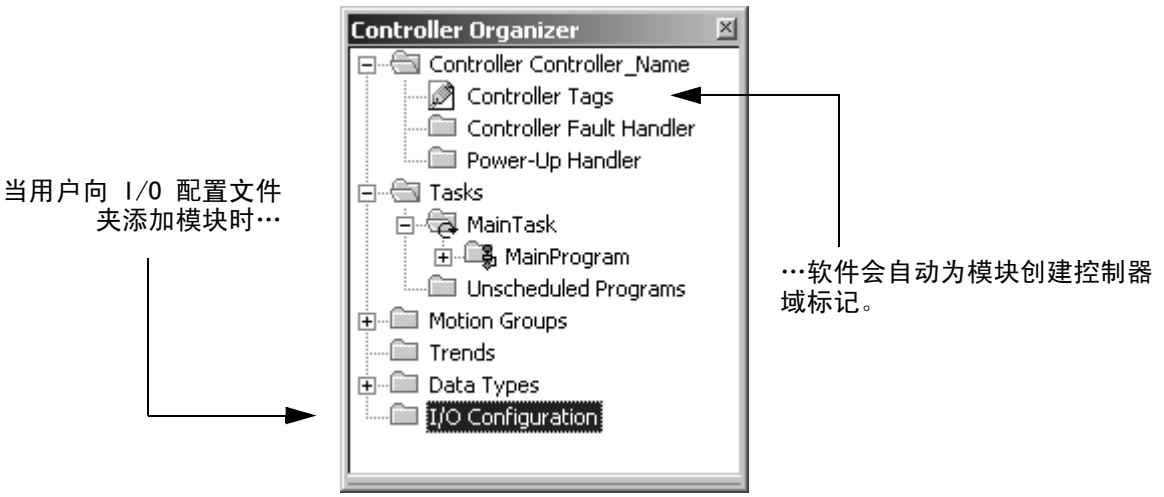
用户使用电子钥匙可以保护自己的系统，防止在槽中意外置入错误的模块。用户选择的钥匙选项决定了槽中的模块必须匹配该槽配置的程度。

如果：	则选择：
必须匹配所有信息： • 类型 • 产品编号 • 供应商 • 主要或次要修订号	完全匹配
匹配除了次要修订号的其它所有信息	兼容模块
没有匹配的信息	禁用钥匙

I/O 数据寻址

I/O 信息显示为一组标记。

- 每个标记使用一种数据结构。结构由 I/O 模块的特性决定。
- 标记名与系统中 I/O 模块的位置有关。



I/O 地址采用下列格式：

Location（位置）	*:Slot（槽）*	*:Type（类型）*	*.Member（成员）*	*.SubMember（子成员）*	*.Bit（位）*

= 可选

其中：	是：
Location（位置）	网络位置 LOCAL = 控制器机架或 DIN 导轨 *ADAPTER_NAME* = 标识远程通讯适配器或网桥模块
Slot（槽）	I/O 模块在它的机架或 DIN 导轨中的槽号
Type（类型）	数据类型 I = 输入 O = 输出 C = 配置 S = 状态
Member（成员）	从 I/O 模块中来的特定数据；由模块可以存储的数据类型决定。 • 对于数字模块，Data 成员常存储输入或输出位值。 • 对于模拟模块，Channel 成员（CH#）常存储通道数据。
SubMember（子成员）	指定与 Member（成员）相关的数据。
Bit（位）	数字 I/O 模块中的特定点；由 I/O 模块大小决定（对于 32 点的模块为 0-31）

缓存 I/O

何时缓存 I/O

缓存是一项技术，此时逻辑不直接引用或操作实时 I/O 设备的标记。相反，逻辑使用 I/O 数据副本。在下面的情况下缓存 I/O：

- 在程序执行过程中防止输入或输出值的更改。（I/O 更新与逻辑执行**异步**。
- 要向结构成员或数组元素复制输入或输出标记。

缓存 I/O

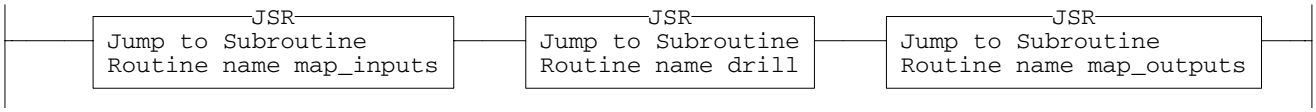
要缓存 I/O，执行下面的操作：

1. 在功能逻辑前面的梯级图中，将请求的输入标记的数据复制或移到相应的缓存标记。
2. 在功能逻辑中引用缓存标记。
3. 在功能逻辑后面的梯级图上，从缓存标记复制数据到相应的输出标记。

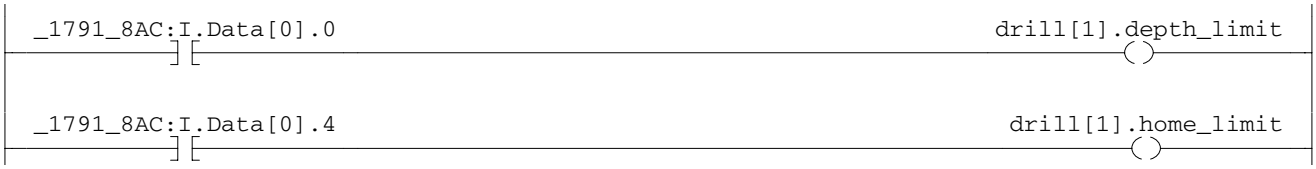
下面的实例向钻机结构标记复制输入和输出数据。

示例 缓存 I/0

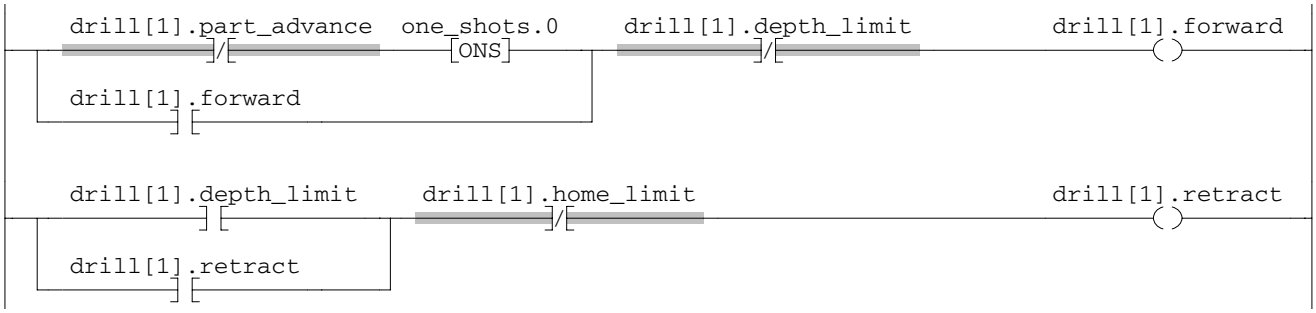
程序主例程执行下面流程中的子例程。



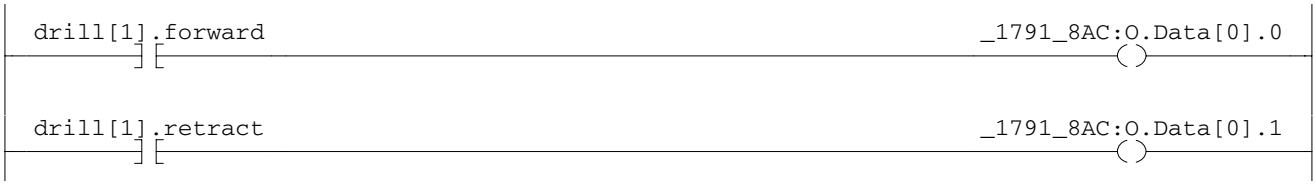
map_inputs 例程将输入设备值复制到钻探例程使用的相应标记。



钻探例程执行钻机的逻辑程序。



map_outputs 例程将钻探例程中的输出标记复制到相应的输出设备。



42369

下面的例子使用 CPS 指令复制数组数据，这些数据代表 DeviceNet 网络的输入设备。

示例 缓存 I/O

Local:0:I.Data 存储 DeviceNet 网络的输入数据，该网络连接到槽 0 中的 1756-DNB 模块。要将输入与应用程序同步，CPS 指令将输入数据复制到 input_buffer。

- 在 CPS 指令复制数据的同时，I/O 更新不能改变数据。
- 应用程序执行时，将 input_buffer 中的输入数据作为输入。

42578

组织标记

使用本章

使用本章组织 Logix5000 控制器的数据。

有关信息：	请参见页：
定义标记	2-1
标记准则	2-6
创建标记	2-9
创建数组	2-9
创建用户定义的数据类型	2-14
说明用户定义的数据类型	2-18
寻址标记数据	2-20
分配 Alias 标记	2-21
分配间接地址	2-24

定义标记

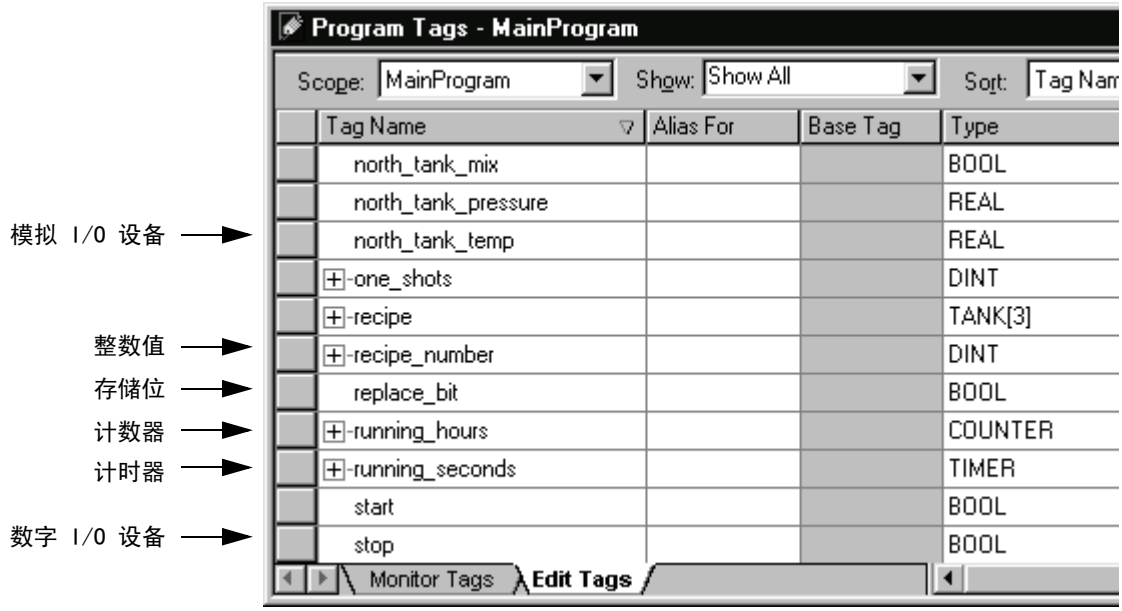
利用 Logix5000 控制器，可使用标记 （字母数字名称）寻址数据（变量）。

术语：	定义：
标记	控制器的存储数据区域的基于文本的名称。 • 标记是分配内存、从逻辑引用数据以及监视数据的基础机制。 • 标记的最小内存分配为 4 字节。 • 创建存储小于 4 字节的数据的标记时，控制器分配 4 字节，但数据只填充所需的部分。

控制器内部使用标记名称，无须交叉引用物理地址。

- 在传统可编程控制器中，物理地址标识每项数据。
  - 地址遵循依据数据类型的固定的数字格式，例如 N7:8、F8:3。
  - 需要符号来使逻辑更容易解释。
- 在 Logix5000 控制器中，没有固定的数字格式。标记名称本身标识数据。这使您可以：
  - 组织数据以镜像机器
  - 开发应用程序时进行记录 （通过标记名称）

示例 标记



标记类型

标记类型定义标记在项目中的工作方式。

如果希望标记：	则选择此类型：
存储值供项目中的逻辑使用	Base
表示其他标记。	Alias
将数据发送到其他控制器	Produced
从其他控制器接收数据	Consumed

如果计划使用生成标记或使用标记，组织标记时必须遵循以下额外的原则。参考第 9-1 页上的 " 与其他设备通信 "。

数据类型

术语：	定义：
数据类型	数据类型定义标记存储的数据的类型，例如位、整数、浮点值、字符串等。
结构	其他数据类型综合的数据类型。 • 设置结构格式以创建匹配特定要求的唯一数据类型。 • 在结构中，每个数据类型称为一个成员。 • 和标记类似，成员具有名称和数据类型。 • Logix5000 控制器包含一组预定义的结构 （数据类型）供特定指令使用，例如计时器、计数器、功能块等。 • 您可以创建自己的结构，称为用户定义的数据类型。

下表列出最常见的数据类型以及何时使用它们。

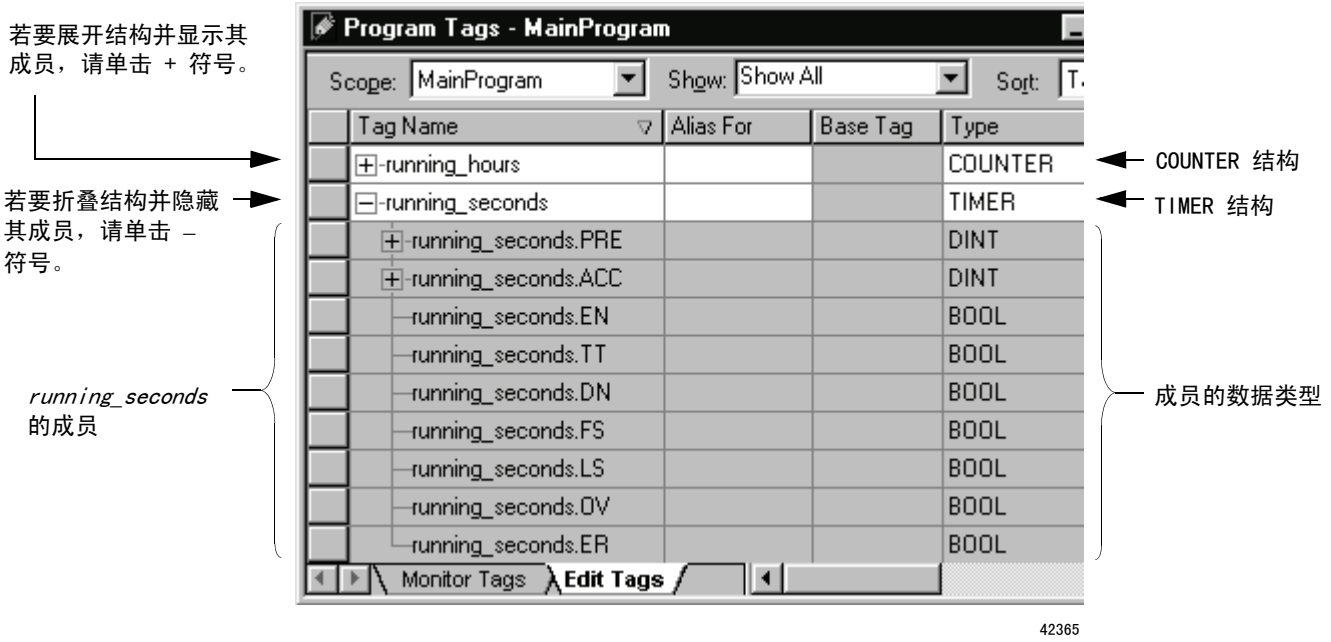
表 2.1 数据类型

对于：	选择：
浮点模式的模拟设备	实数
整数模式的模拟设备 （非常高的采样率）	INT
ASCII 字符	字符串
位	BOOL
计数器	计数器
数字 I/O 点	BOOL
浮点数	实数
整数 （整个数字）	DINT
定序器	CONTROL
计时器	计时器

标记的最小内存分配为 4 字节。创建存储小于 4 字节的数据的标记时，控制器分配 4 字节，但数据只填充所需的部分。

数据类型	位			
	31 16	15 8	7 1	0
BOOL	不使用			0 或 1
SINT	不使用		-128 至 +127	
INT	不使用	-32,768 至 +32767		
DINT	-2,147,483,648 至 +2,147,483,647			
实数	$-3.40282347E^{38}$ 至 $-1.17549435E^{-38}$ （负值） 0 $1.17549435E^{-38}$ 至 $3.40282347E^{38}$ （正值）			

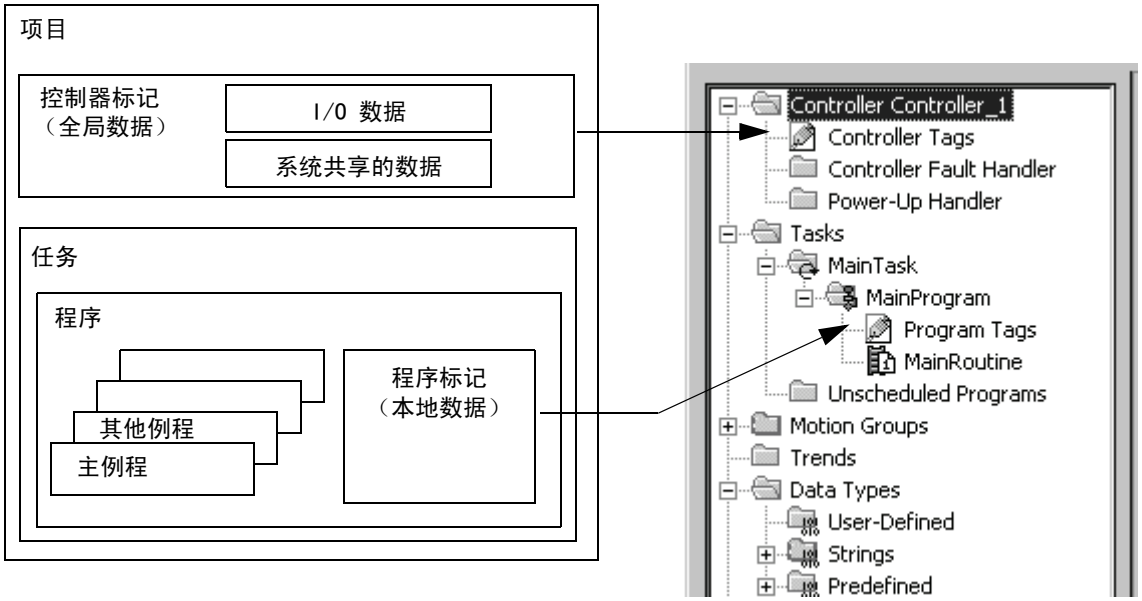
COUNTER 和 TIMER 数据类型是通常使用的结构的示例。



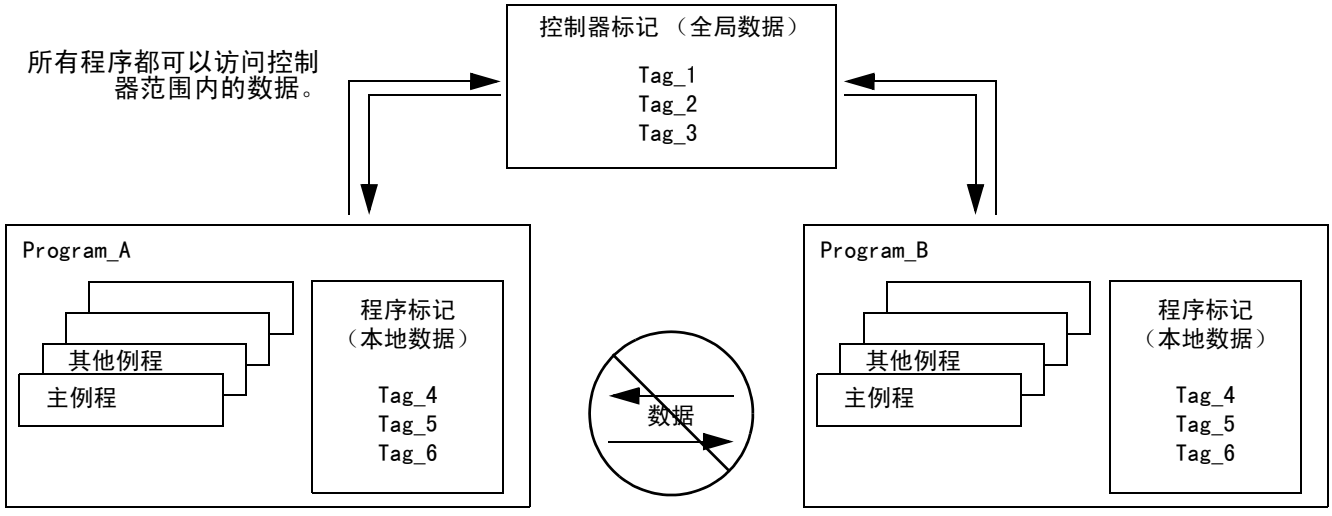
若要将数据复制到结构，请使用 COP 指令。 请参见 *Logix5000 控制器基本指令集参考手册*，出版物 1756-RM003。

范围

创建标记时，您将其定义为控制器标记 （全局数据）或特定程序的程序标记 （本地数据）。



Logix5000 控制器使您可以将应用程序分割为多个程序，每个程序有自己的数据。无需管理程序间冲突的标记名称。这使得更容易在多个程序间重新使用代码和标记名称。



程序范围内的数据与其他程序隔离。

- 例程不能访问其他程序的程序范围内的数据。
- 可以在多个程序内重新使用程序范围的标记的标记名称。

  例如，Program_A 和 Program_B 都可以拥有名为 Tag_4 的程序标记。

避免同时为控制器和程序标记使用相同的名称。在程序中，如果程序的程序标记与控制器标记名称相同，则不能引用控制器标记。

某些标记必须是控制器范围 （控制器标记）。

如果希望将标记用于：	则分配此范围：
项目中的多个程序中	控制器范围 （控制器标记）
消息 (MSG) 指令中	
产生或使用数据	
与 PanelView 终端通信	
以上都不是	程序范围 （程序标记）

标记准则

使用以下准则为 Logix5000 项目创建标记：

准则：		详细信息：
❑	1. 创建用户定义的数据类型。	用户定义的数据类型 （结构）使您可以组织数据以匹配机器或进程。用户定义的数据类型提供以下优势： • 一个标记包含程序特定方面的所有相关数据。这样将相关数据保存在一起便于查找，不考虑其数据类型。 • 每个数据 （成员）得到一个描述性名称。这样自动为逻辑创建初始水平的文档。 • 可以使用该数据类型创建具有相同数据布局的多个标记。 例如，使用用户定义的数据类型存储罐的所有参数，包括温度、压力、阀门位置和预设值。然后基于该数据类型为每个罐创建标记。
❑	2. 使用数组快速创建一组类似标记。	数组在一个公共标记名称下创建数据类型的多个实例。 • 数组使您可以组织一组使用相同数据类型并执行类似功能的标记。 • 以一维、二维或三维组织数据以匹配数据表示的内容。 例如，使用二维数组组织油库的数据。数组的每个元素表示一个油罐。元素在数组中的位置表示罐的地理位置。 重要：尽量减少使用 BOOL 数组。许多数组指令不能在 BOOL 数组上使用。这使得更难以初始化和清除 BOOL 数据数组。 • 通常对 PanelView 屏幕的位级别对象使用 BOOL 数组。 • 否则，使用 DINT 标记的各个位或 DINT 数组。

<table>
<tr><th colspan="2">准则：</th><th>详细信息：</th></tr>
<tr><td>❑</td><td>3. 利用程序范围内的标记。</td><td>如果希望多个标记具有相同名称，请在不同程序的程序范围定义每个标记（程序标记）。这使您可以在多个程序中重新使用逻辑和标记名称。

避免同时为控制器和程序标记使用相同的名称。 在程序中，如果程序的程序标记与控制器标记名称相同，则不能引用控制器标记。

某些标记必须是控制器范围 （控制器标记）。

<table><tr><th>如果希望将标记用于：</th><th>则分配此范围：</th></tr><tr><td>项目中的多个程序中</td><td rowspan="4">控制器范围 （控制器标记）</td></tr><tr><td>消息 (MSG) 指令中</td></tr><tr><td>产生或使用数据</td></tr><tr><td>与 PanelView 终端通信</td></tr><tr><td>以上都不是</td><td>程序范围 （程序标记）</td></tr></table></td></tr>
<tr><td>❑</td><td>4. 对于整数，使用 DINT 数据类型。</td><td>要提高逻辑效率，请尽量减少 SINT 或 INT 数据类型。条件允许时，对整数使用 DINT 数据类型。

• Logix5000 控制器通常将值作为 32 位值 （DINT 或 REAL）进行比较或操作。
• 控制器使用值前，通常自动将 SINT 或 INT 值转换为 DINT 或 REAL 值。
• 如果目标是 SINT 或 INT 标记，控制器通常将该值转换回 SINT 或 INT 值。
• 与 SINT 或 INT 的转换自动进行，无需额外编程。但占用额外执行时间和内存。</td></tr>
</table>

准则：		详细信息：
❑	5. 限制标记名称为 40 个字符。	下面是标记名称的规则： • 仅字母字符 （A-Z 或 a-z）、数字字符 (0-9) 和下划线 (_) • 必须以字母字符或下划线开始 • 不能多于 40 个字符。 • 不以下划线 (_) 开头或结尾 • 不区分大小写
❑	6. 使用混合大小写。	尽管标记不区分大小写 （大写 *A* 和小写 *a* 相同），但混合大小写更容易阅读。

这些标记更容易阅读：	相比这些标记：
Tank_1	TANK_1
Tank1	TANK1
	tank_1
	tank1

❑	7. 考虑标记的字母顺序。	RSLogix 5000 软件以字母顺序显示相同范围的标记。要更容易监视相关标记，请对要保存在一起的标记使用类似的开始字符。

对罐的每个标记以 *Tank* 开始将标记保存在一起。

标记名称
Tank_North
Tank_South
...

否则，标记可能彼此分隔。

标记名称
North_Tank
...
...
...
South_Tank

← 以 *o*、*p*、*q* 等开始的其他字符。

创建标记

重要

当您进行以下操作时 RSLogix 5000 软件自动创建标记：

- 将元素添加到流程图 (SFC)
- 将流程块指令添加到流程块图

Tags （标记）窗口使您使用标记的电子表格样式的视图创建和编辑标记。

1. 从 Logic （逻辑）菜单选择 Edit Tags （编辑标记）。

2. 为标记选择**范围**：

42350

如果将把标记用于：	则选择：
项目中的多个程序	*name_of_controller （控制器）*
作为产生者或使用者	
消息中	
项目中的仅一个程序中	将使用标记的程序

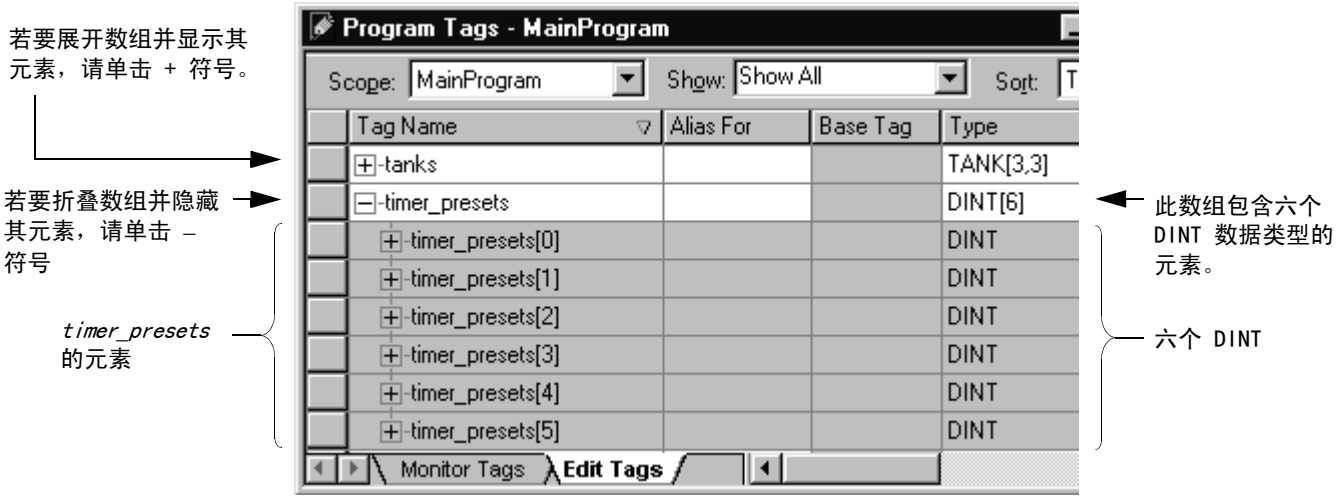
3. 键入标记的**名称**、**数据类型**和**说明** （可选）。

创建数组

Logix5000 控制器还允许您使用数组组织数据。

术语：	定义：
数组	包含一组多个数据的标记。 • 数组类似于文件。 • 在数组中，每个数据称为一个元素。 • 每个元素使用相同的数据类型。 • 数组标记占据控制器中的一个连续内存块，每个元素顺序排列。 • 您可以使用数组和定序器指令操作或索引数组中的元素。 • 可以将数据组织到一维、二维或三维中。

下标标识数组中的每个**元素**。下标从 0 开始，至元素数减一的位置（从零开始）。



下面的示例比较结构和数组：

这是一个使用 Timer 结构 （数据类型）的标记。

标记名称	数据类型
⊟ Timer_1	计时器
⊞ Timer_1.PRE	DINT
⊞ Timer_1.ACC	DINT
Timer_1.EN	BOOL
Timer_1.TT	BOOL
Timer_1.DN	BOOL

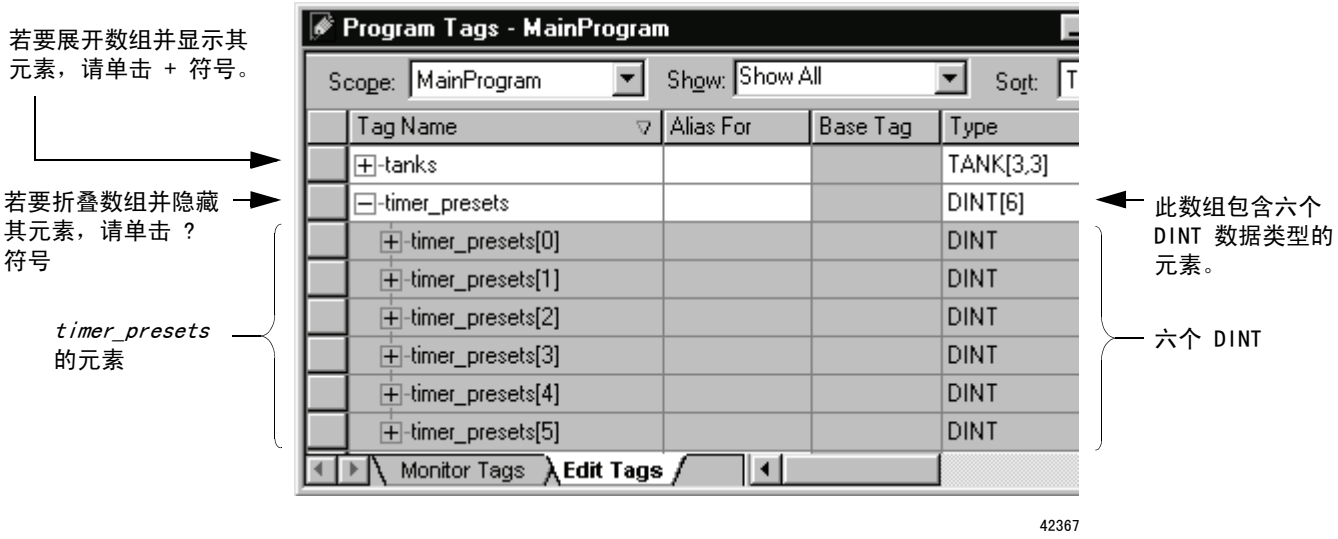
这是一个使用 Timer 数据类型数组的标记。

标记名称	数据类型
⊟ 计时器	TIMER[3]
⊞ Timer[0]	计时器
⊞ Timer[1]	计时器
⊞ Timer[2]	计时器

示例

一维数组

在本示例中，一个计时器指令对多个步骤进行计时。每个步骤需要一个不同的预设值。因为所有值类型相同 (DINT)，所以使用数组。



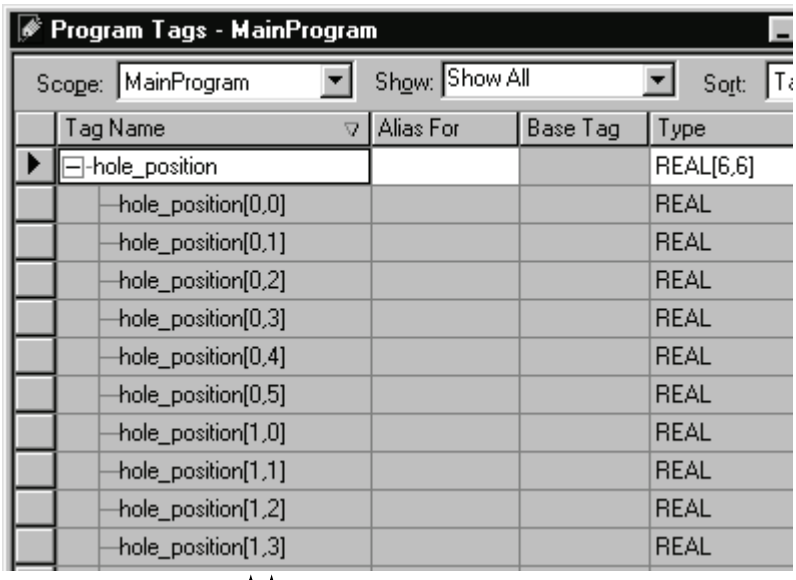
示例

二维数组

钻机可以在书中钻 1 至 5 个孔。机器需要每个孔距离书前沿距离的值。为将值组织到配置中，使用一个二维数组。第一个下标指示值对应的孔，第二个下标指示钻了多少孔（1 至 5）。

第一个维度的下标	第二个维度的下标 0	1	2	3	4	5	说明
0							
1		1.5	2.5	1.25	1.25	1.25	第一个孔距离书前沿的位置
2			8.0	5.5	3.5	3.5	第二个孔距离书前沿的位置
3				9.75	7.5	5.5	第三个孔距离书前沿的位置
4					9.75	7.5	第四个孔距离书前沿的位置
5						9.75	第五个孔距离书前沿的位置

在 Tags （标记）窗口中，元素顺序如下。



创建数组

要创建数组，创建一个标记并为数据类型分配维度：

1. 从 Logic （逻辑）菜单选择 Edit Tags （编辑标记）。

42350

2. 键入标记的**名称**并为标记选择一个**范围**：

如果将把标记用于：	**则选择：**
项目中的多个程序	*name_of_controller （控制器）*
作为产生者或使用者	
消息中	
项目中的仅一个程序中	将使用标记的程序

3. 分配数组维度：

如果标记是：	**则键入：**	**位置：**
一维度数组	*data_type[x]*	*data_type* 是标记存储的数据类型。 *x* 是第一个维度的**元素**数量。 *y* 是第二个维度的元素数量。 *z* 是第三个维度的元素数量。
二维数组	*data_type[x,y]*	
三维数组	*data_type[x,y,z]*	

创建用户定义的数据类型

用户定义的数据类型 （结构）使您可以组织数据以匹配机器或进程。

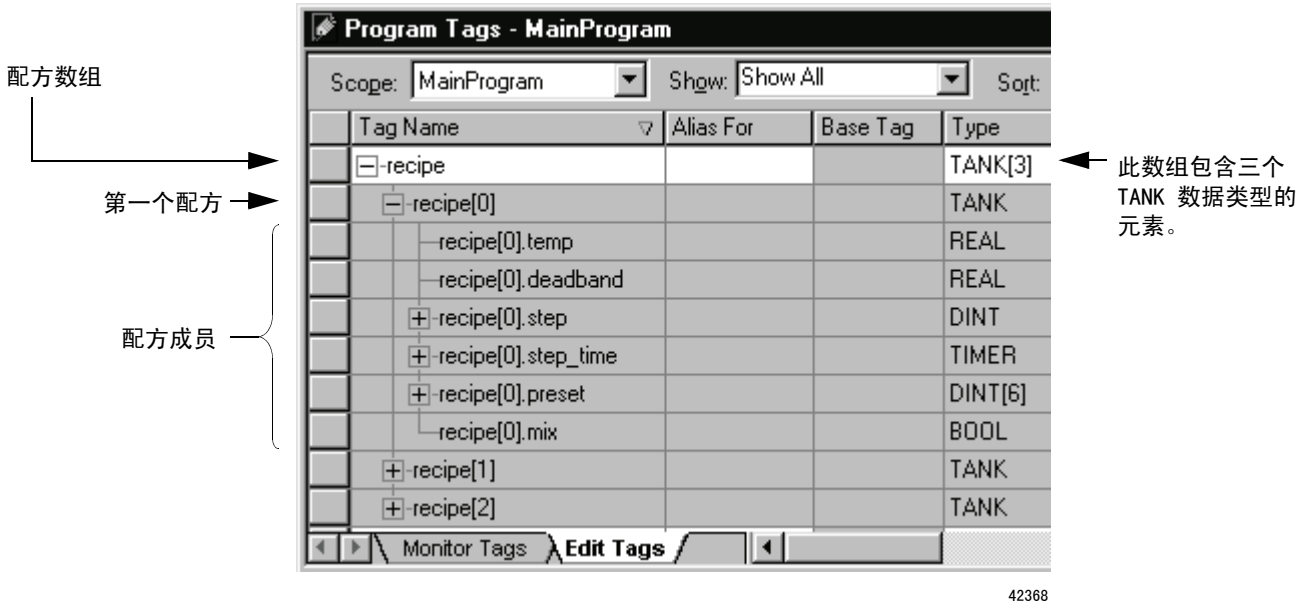
示例

存储配方的用户定义的数据类型

在多罐系统中，每个罐可以运行各种配方。因为配方需要混合数据类型 （REAL、DINT、BOOL 等），所以使用用户定义的数据类型。

名称 （数据类型）：TANK	
成员名称	**数据类型**
temp	实数
deadband	实数
step	DINT
step_time	计时器
preset	DINT[6]
mix	BOOL

基于此数据类型的数组如下所示：



示例

存储运行机器所需的数据的用户定义的数据类型。

因为一些钻站需要以下混合数据，所以使用用户定义的数据类型

名称 （数据类型）：DRILL_STATION	
成员名称	**数据类型**
part_advance	BOOL
hole_sequence	CONTROL
类型	DINT
hole_position	实数
depth	实数
total_depth	实数

基于该数据类型的数组如下所示：

用于定义的数据类型的准则

创建用户定义的数据类型时，记住以下准则：

- 如果包括表示 I/O 设备的成员，必须使用逻辑在结构中的成员和相应 I/O 标记间复制数据。 参考第 1-7 页上的"I/O 数据寻址".
- 如果包括数组作为成员，请限制数组为一维。多维数组***不*** 允许采用用户定义的数据类型。
- 使用 BOOL、SINT 或 INT 数据类型时，将使用相同数据类型的成员放置在序列中：

更加高效

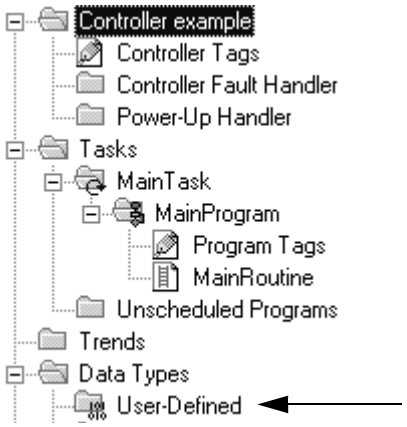
BOOL
BOOL
BOOL
DINT
DINT

更加低效

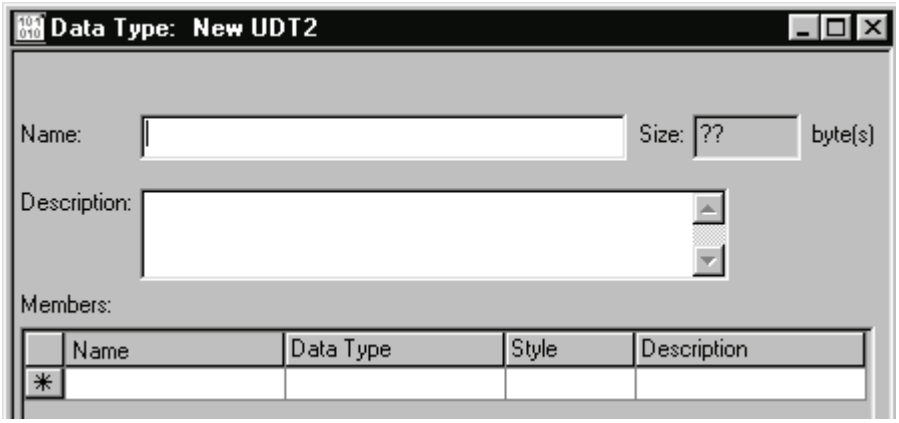
BOOL
DINT
BOOL
DINT
BOOL

创建用户定义的数据类型

1. 从 Data Types （数据类型）下的 User-Defined （用户定义）文件夹右击 User-Defined （用户定义）并选择 New Data Type （新建数据类型）。



2. 键入数组的**名称**和数组的 **说明**。对于数组的每个**成员**，键入名称、数据类型、样式和说明 （选项）。



42196

限制数组为一维。

要以不同**样式** （radix） 显示成员值，请选择该样式。

3. 单击 Apply （应用）。

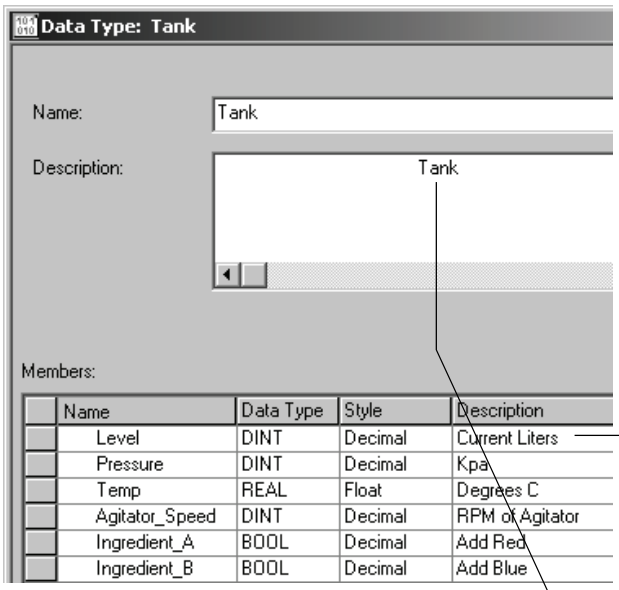
4. 添加所需成员。

说明用户定义的数据类型

RSLogix 5000 软件
13.0 或更高

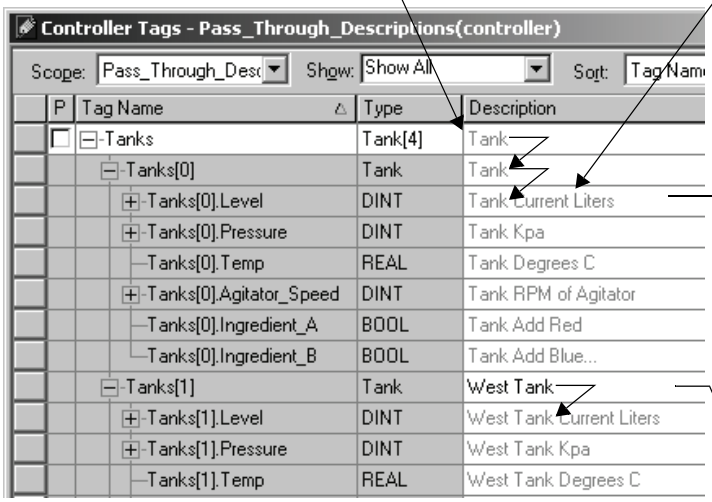
RSLogix 5000 软件使您可以在用户定义的数据类型的说明外自动生成说明。这样极大减少记录项目的时间。

组织用户定义的数据类型时，记住 RSLogix 5000 软件的以下功能：



传递说明 – 条件允许时，RSLogix 5000 软件查找标记、元素或成员的可用说明。

- 用户定义的数据类型的说明传递到使用该数据类型的标记。
- 数组标记的说明传递到数组的元素和成员。



追加说明到 base 标记 – RSLogix 5000 软件自动为使用用户定义的数据类型的标记的每个成员生成说明。它以标记的说明开始，然后添加该数据类型成员的说明。

粘贴传递说明 – 使用数据类型和数组说明作为更具体说明的基础。

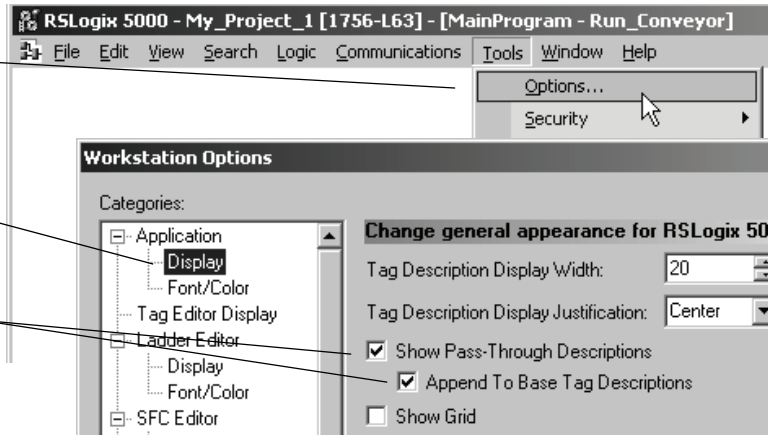
在本示例中，Tank 成为 West Tank。

RSLogix 5000 软件使用不同颜色说明：

此颜色说明：	是：
灰色	传递说明
黑色	手动输入说明

打开或关闭传递和追加说明

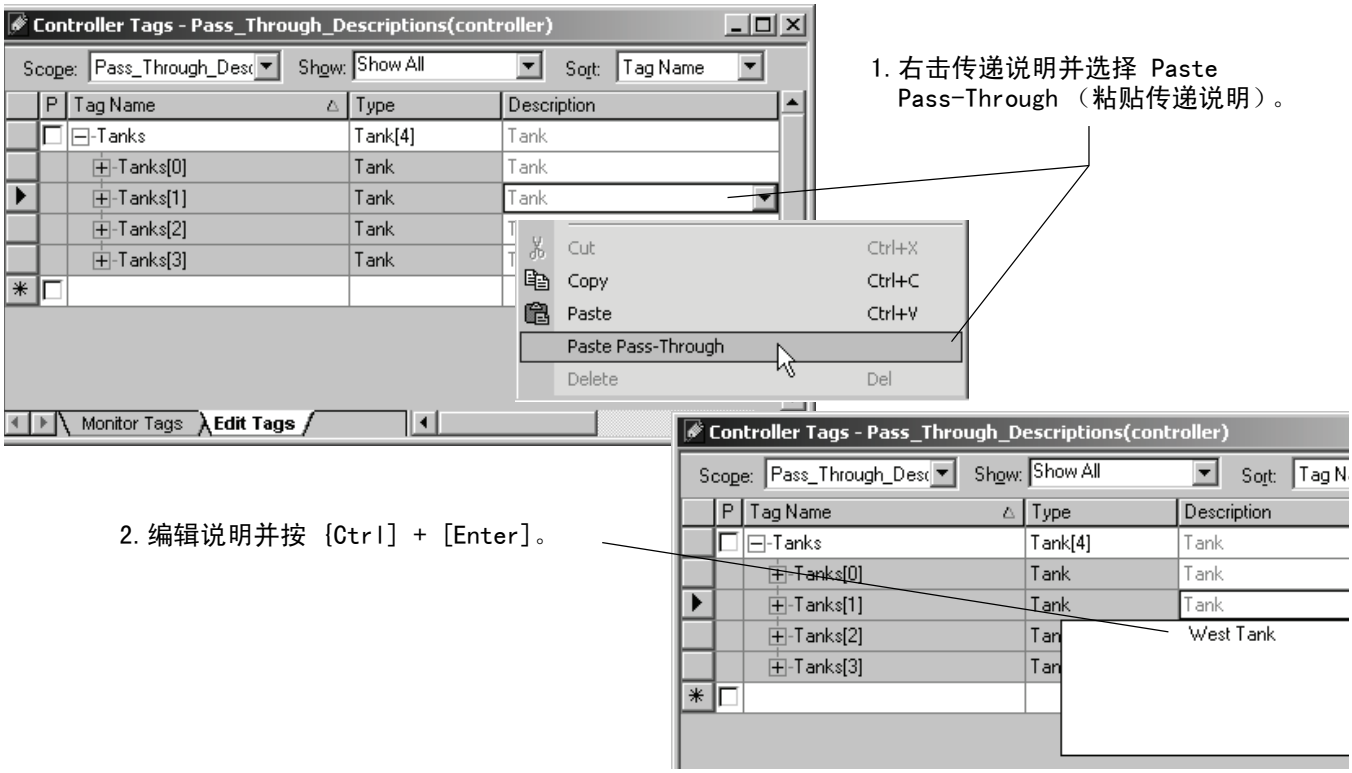
1. 在 RSLogix 5000 软件中，选择 Tools （工具）> Options （选项）。
2. 选择 Application （应用程序）> Display （显示）。
3. 打开 （选中）或关闭 （取消选中）所需选项。



粘贴传递说明

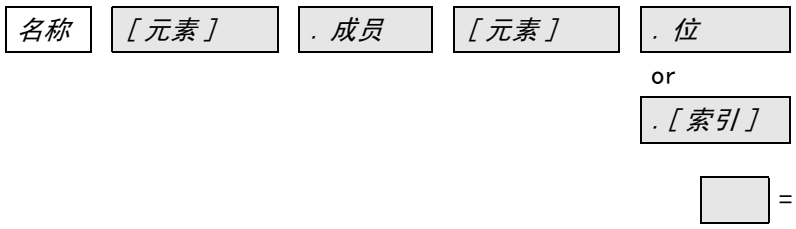
将传递说明用作更具体说明的开始点：

1. 右击传递说明并选择 Paste Pass-Through （粘贴传递说明）。
2. 编辑说明并按 [Ctrl] + [Enter]。



寻址标记数据

标记名称遵循此格式：



位置：	为：
名称	标识此特定标记的名称。
元素	指向数组中特定元素的下标。 • 仅当标记或成员是数组时使用元素标识符。 • 对数组的每个维度使用一个下标。例如： [5]、[2,8]、[3,2,7]。 若要间接 （动态）引用元素，请使用能提供元素号的标记或数值表达式。 • 数值表达式使用标记、常数、运算符和函数的组合计算值。例如，Tag_1-Tag_2、 Tag_3+4、 ABS(Tag_4)。 • 保存数组维度中标记或数值表达式的值。例如，如果数组一个维度包含 10 个元素，则标记或数值表达式的值必须在 0 至 9 之间 （10 个元素）。
成员	指定结构的成员。 • 仅当标记是结构时使用成员标识符。 • 如果结构包含其他结构作为其成员，请使用额外级别的 . 成员 格式指定所需成员。
位	指定整数数据类型 （SINT、INT 或 DINT）的位。
索引	若要间接 （动态）引用整数的位，请使用能提供位号的标记或数值表达式。 • 数值表达式使用标记、常数、运算符和函数的组合计算值。例如，Tag_1-Tag_2、 Tag_3+4、 ABS(Tag_4)。 • 保存整数标记位范围中标记或数值表达式的值。例如，如果整数标记是 DINT （32 位），则索引的值必须在 0 至 31 （32 位）之间。

分配 Alias 标记

alias 标记使您可以创建表示其他标记的标记。

- 两个标记具有相同的值。
- 当一个标记的值更改时，另一个标记也反映变化。

在下列情况下使用别名：

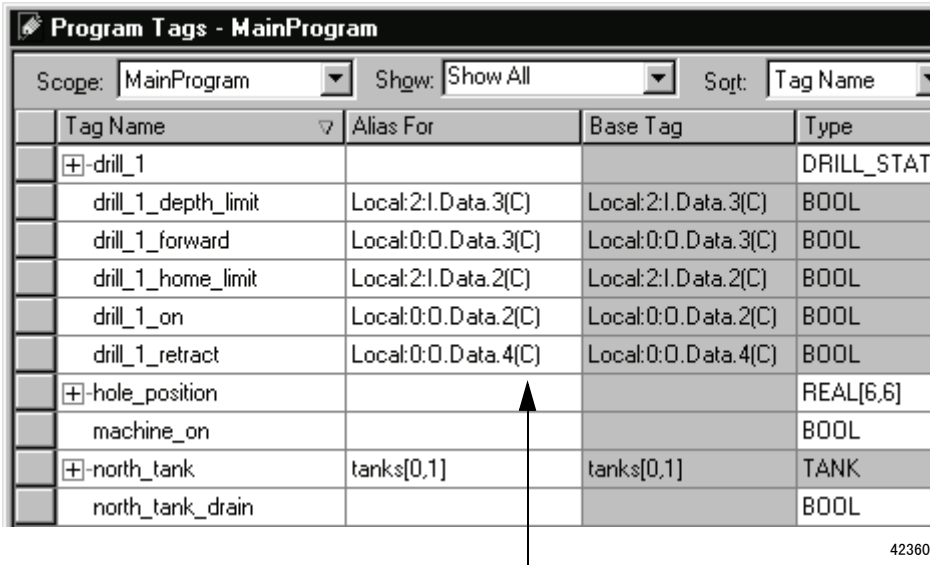
- 在接线图前编程逻辑
- 为 I/O 设备分配描述性名称
- 为复杂标记提供更简单的名称
- 为数组元素使用描述性名称

标记窗口显示别名信息。

drill_1_depth_limit 是 *Local:2:I.Data.3*（数字输入点）的别名。输入为 on 时，alias 标记也为 on。

drill_1_on 是 *Local:0:O.Data.2*（数字输出点）的别名。alias 标记为 on 时，输出标记也为 on。

north_tank 是 *tanks[0,1]* 的别名。

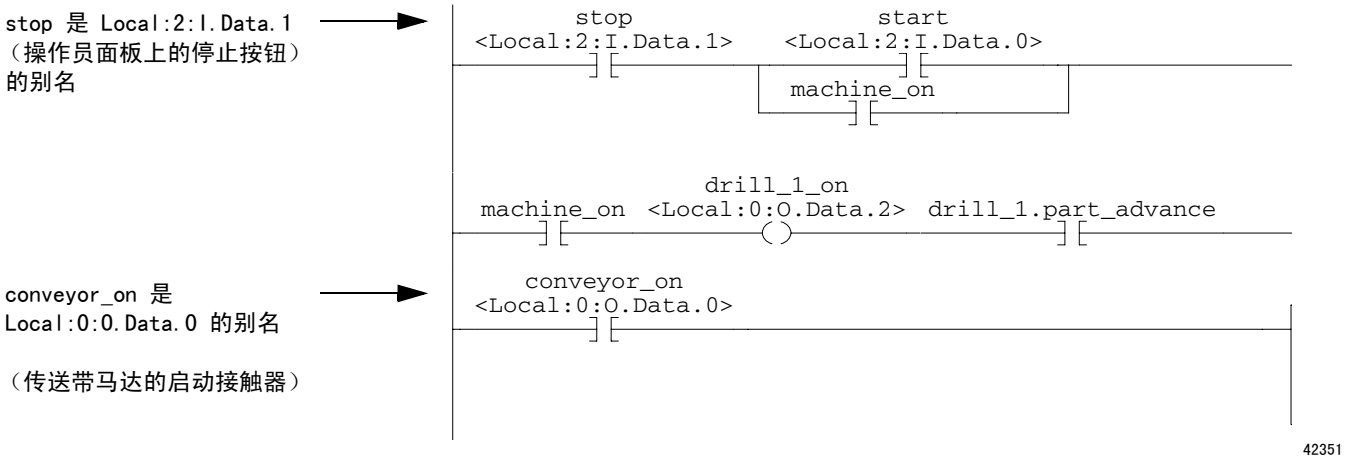


(C) 指示标记在控制器范围内。

alias 标记的常见用途是在接线图可用前编程逻辑：

1. 对于每个 I/O 设备，创建一个带有描述设备的名称的标记，例如用于传送带马达的 ***conveyor***。
2. 使用描述性标记名称编程逻辑。（甚至可以无需连接到 I/O 而测试逻辑。）
3. 之后当接线图可用时，将 I/O 模块添加到控制器的 I/O 配置。
4. 最后将描述性标记转换为对应 I/O 点或通道的别名。

下面的逻辑最初使用描述性标记名称编程，例如 stop 和 conveyor_on。之后，标记转换为对应 I/O 设备的别名。



显示别名信息

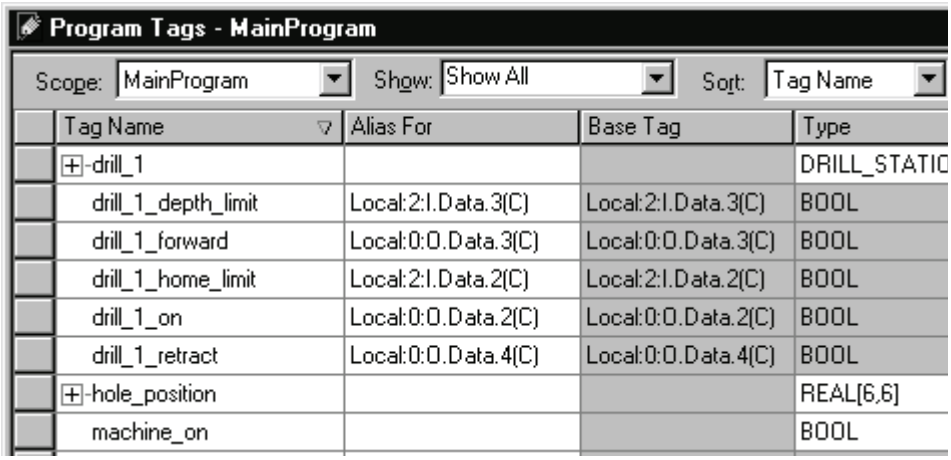
显示 （逻辑中）别名指向的标记：

1. 从 Tools （工具）菜单选择 Options （选项）。
2. 选择 Ladder Display （梯形显示）选项卡。
3. 选中 Show Tag Alias Information （显示标记别名信息）复选框。
4. 单击 OK （确定）。

分配别名

分配标记作为其他标记的 **alias 标记**：

1. 从 Logic （逻辑）菜单选择 Edit Tags （编辑标记）。



2. 选择标记的**范围**。
3. 在标记名称右侧单击 Alias For （别名）单元格。

   单元格显示 ▼
4. 单击 ▼
5. 选择别名将表示的标记：

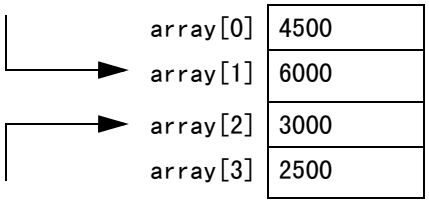
有关：	执行：
选择标记	双击标记名称。
选择位号	A. 单击标记名称。 B. 在标记名称右侧单击 ▼ C. 单击所需的位。

6. 按 [Enter] 或单击其他单元格。

分配间接地址

如果希望指令访问数组中的不同元素，请使用数组下标 （间接地址）中的标记。通过更改标记值，可以更改逻辑引用的数组元素。

索引等于 1 时 array[index] 指向此处。



索引等于 2 时 array[index] 指向此处。

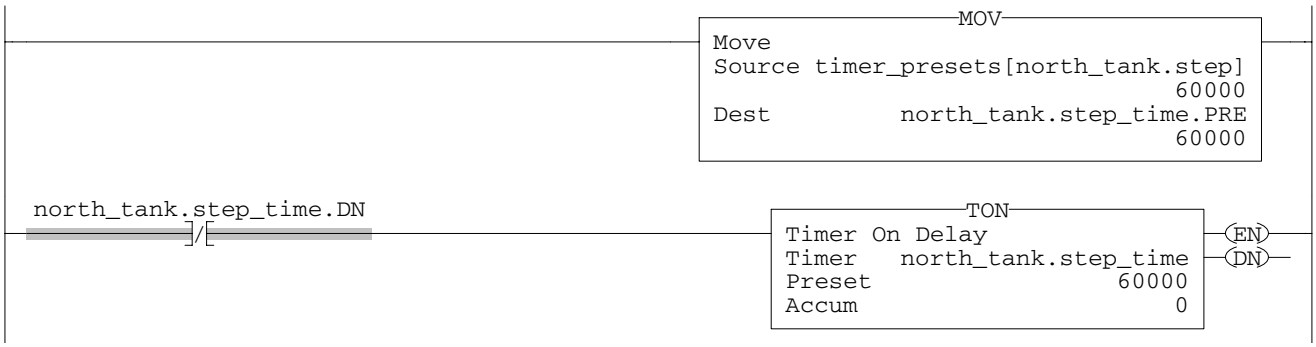
下表列出间接地址的一些常见用途：

有关：	在下标中使用标记并：
从配方数组中选择一个配方	在标记中输入配方的编号。
从可行设置数组中加载特定机器设置	在标记中输入所需设置。
从数组一次加载一个参数或状态	A. 在第一个元素上执行所需操作。
记录错误代码	
对数组元素执行一些操作然后索引到下一个元素	B. 使用 ADD 指令增加标记值并指向数组中的下一个元素。

下面的示例将一系列预设值加载到计时器中，一次一个值 （数组元素）。

示例 步进数组

timer_presets 数组存储下一个梯级中计数器的一系列预设值。north_tank.step 标记指向要使用的数组元素。例如，当 north_tank.step 等于 0 时，指令将 timer_presets[0] 加载到 timer (60,000 ms) 中。



当 north_tank.step_time 完成后，梯级递增 north_tank.step 至下一个数并且 timer_presets 数组的该元素加载到计时器中。

当 north_tank.step 超过数组大小时，梯级重置标记以从数组的第一个元素开始。（数组包含元素 0 至 3。）

42358

表达式

还可以使用表达式指定数组下标。

- 表达式使用运算符如 + 或 - 计算值。
- 控制器计算表达式的结果并用作数组下标。

可以使用这些运算符指定数组的下标：

运算符：	说明：
+	加
-	减 / 求负
*	乘
/	除
ABS	绝对值
AND	AND
FRD	BCD 至整数

运算符：	说明：
MOD	取模
NOT	取补
OR	或
SQR	平方根
TOD	整数至 BCD
TRN	截断
XOR	异或

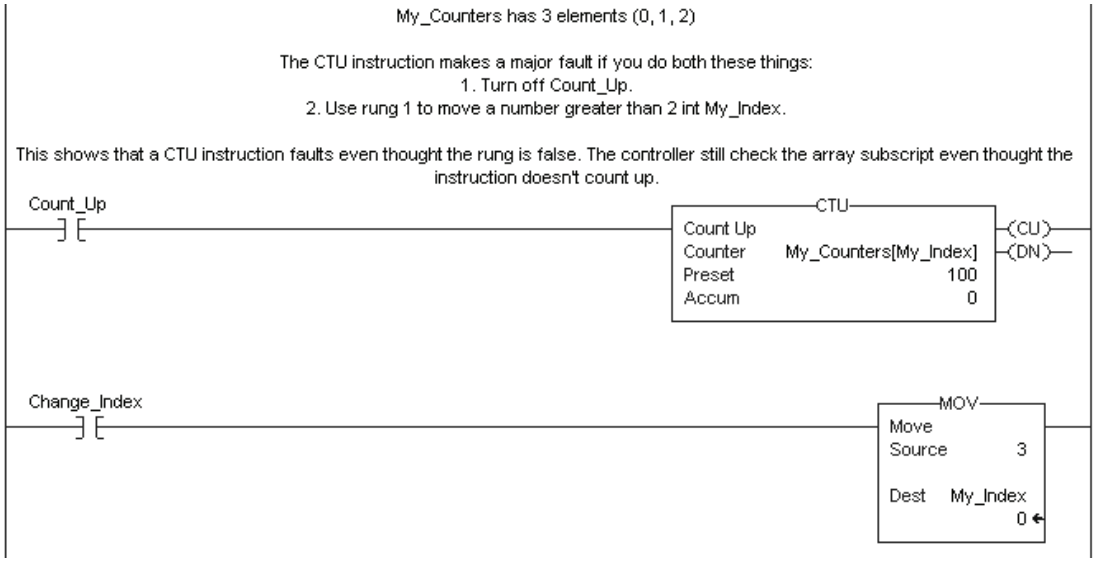
按如下方法设置表达式格式：

如果运算符要求：	使用此格式：	示例：
一个值 （标记或表达式）	*运算符 （值）*	ABS(tag_a)
两个值 （标记、常数或表达式）	*value_a 运算符 value_b*	• tag_b + 5 • tag_c AND tag_d • (tag_e ** 2) MOD (tag_f / tag_g)

数组下标超过范围

如果数组下标超过范围，每个指令将生成一个主故障。即使梯级为 false，传输指令也会生成主故障。即使梯级为 false，控制器也检查这些指令中的数组下标。

示例



有关处理主故障的更多信息，请参见第 15 章处理主故障。

注释：

管理任务

使用本章

默认的 RSLogix 5000 项目为用户所有逻辑程序提供一个单独的任务。这能满足很多应用的要求，但是一些情况需要多个任务。

本章帮助用户管理项目中的任务：

选择控制器任务

Logix5000 控制器支持使用多个任务来基于特定标准计划并确定用户程序执行的优先级。这可以平衡控制器过程处理时间。

- 控制器一次只能执行一个任务。
- 不同的任务可以中断正在执行的任务并取得控制权。
- 对于任何特定的任务，一次只能执行一个程序。

Logix5000 控制器支持三种类型的任务：

如果用户想按以下方式执行逻辑程序：	则使用下列类型的任务：	说明：
全部时间	连续性任务	连续性任务在后台执行。任何未分配给其它操作 （例如：运动、通信以及周期型任务或事件性任务）的 CPU 时间都将用于执行连续性任务中的程序。 • 连续性任务始终运行。当连续性任务完成一次全扫描之后，它会立刻重新开始。 • 项目不一定需要连续性任务。如果需要，只能有一个连续性任务。
• 以一个固定的周期（例如：每 100 ms） • 在扫描其它逻辑程序时多次运行某一程序	周期性任务	周期性任务按照指定的周期执行一项功能。只要到达周期性任务的周期时限，周期性任务就会： • 中断任何低优先级的任务 • 执行一次 • 将控制权交回到先前任务停止处 用户可以配置时间周期从 0.1 ms 到 2000 s。 • 默认值为 10 ms。 • 周期性任务的性能由 Logix5000 控制器的类型和任务的逻辑程序决定。
当事件发生时立刻执行	事件性任务	事件性任务是当特定的事件 （触发）发生时执行一项功能。只要事件性任务的触发发生，事件性任务会： • 中断任何低优先级的任务 • 执行一次 • 将控制权交回到先前任务停止处 触发可以是： • 一个数字量输入的改变 • 模拟量数据的新采样 • 特定的运动操作 • 使用标记 • EVENT 指令 **重要：** 一些 Logix5000 控制器并不支持所有种类的触发。请参阅第 3-20 页上的表 3.1。

这里是任务的一些实例情况：

对于下面的实例情况：	使用以下类型任务：
将池水升到最高位置，然后打开排水阀	连续性任务
收集并处理系统参数，然后发送以显示	连续性任务
以控制顺序完成步骤 3 阎囟ㄎ幌滠交幌蚱□	连续性任务
用户必须每 0.1 s 检查一次系统工作臂的位置，然后计算位置的平均改变率。通过它来确定制动压力。	周期性任务
每 20 ms 读取一次纸张卷的厚度。	周期性任务
一条包装线粘合纸盒。当纸盒传送到粘合位置时，控制器必须立即执行粘合例程。	事件性任务
在高速度的装配操作中，有一台光学传感器来检测某种类型的废品。当传感器检测到废品时，机器立即转移废品。	事件性任务
在引擎测试过程中，用户要在每次采样后，立即捕获并存档每个模拟数据。	事件性任务
收到新的生产数据后立即向工作站点加载数据	事件性任务
在包装棒糖的生产线，必须确保在正确的位置穿孔。每次套准传感器检测套准标记、检查轴的精确位置并执行所需调整。	事件性任务
粘合站点必须根据轴速变化来调整所用胶水的数量。如有需要，在运动计划器执行后，检查轴的命令速度并改变胶水的数量。	事件性任务
如果程序检测到在生产线上存在危险条件，必须关闭整个生产线。不管是什么样的危险条件，关闭过程都相同。	事件性任务

由控制器决定支持的任务数量：

下面的控制器：	支持以下任务数量：	注释：
ControlLogix SoftLogix5800	32	只能有一个连续性任务。
CompactLogix DriveLogix FlexLogix	8	

特别注意所用的任务数量

一般情况下，每个任务都会占用控制器执行其它任务的时间。如果用户有太多任务，则：

- 完成连续性任务可能占用太长的时间。
- 可能与其它任务的时间重叠。如果一项任务中断太频繁或中断时间太长，可能在其被再次触发前，不能成功执行。

更多信息，请参阅第 第 3-8 页 页上的 “避免重叠”。

区分周期性任务和事件性任务的优先级

尽管一个项目可以包括多个任务，但是控制器一次只能执行一个任务。如果在其它任务正在执行时，触发周期性或事件性任务，则由每个任务的优先级确定控制器的操作。

优先级的级别数依赖于控制器：

下面的 Logix5000 控制器：	具有的优先级别数：
CompactLogix	15
ControlLogix	15
DriveLogix	15
FlexLogix	15
SoftLogix5800	3

根据下面的原则为任务设置优先级：

如果用户希望：	则	注释：
该任务中断其它任务	为该任务设置的优先级号小于 （较高优先级）其它任务的优先级号。	• 高优先级的任务可以中断所有低优先级的任务。 • 一个高优先级的任务可以多次中断一个低优先级的任务。
其它任务中断该任务	为该任务设置的优先级号大于 （较低优先级）其它任务的优先级号。	
该任务与其它任务共享控制器时间	为两个任务设置相同的优先级。	控制器在两个任务间来回切换，每个任务执行 1 ms。

其它考虑的事项

估计任务执行中断时，请考虑下面的事项：

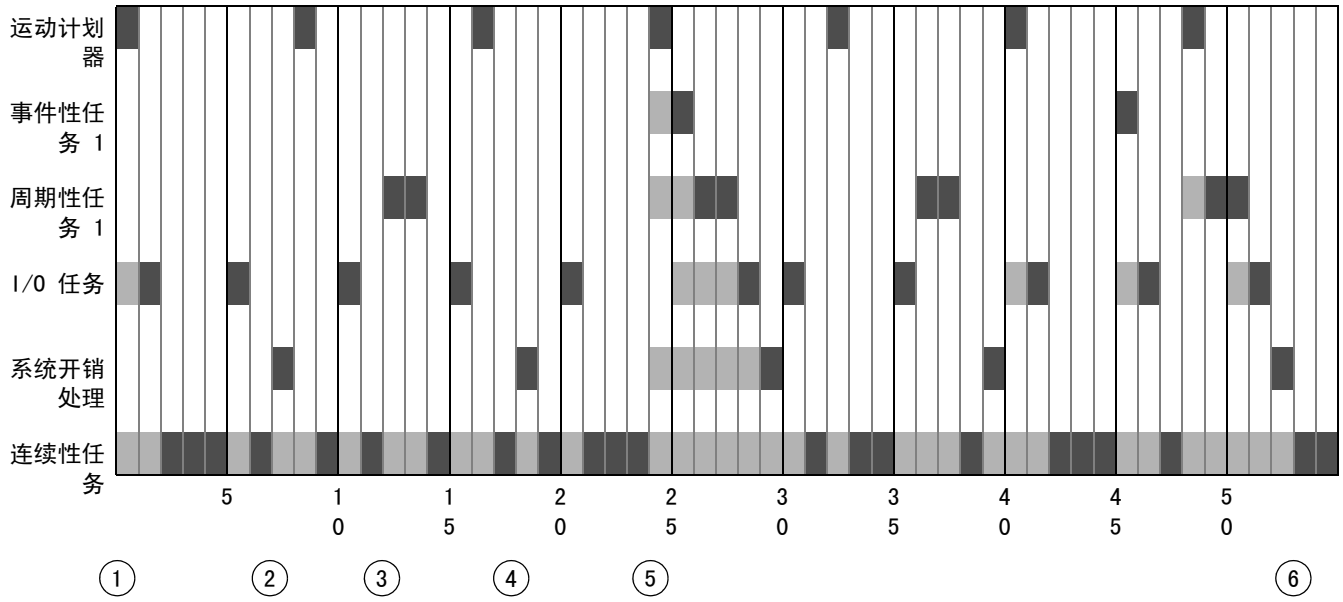
<table>
<tr><th>考虑：</th><th>说明：</th></tr>
<tr><td>运动计划器</td><td>运动计划器中断其它所有任务 （不管任务的优先级）。
轴的数量和运动组粗略更新间期影响运动计划器的执行时间和频率。
如果在触发任务时执行运动计划器，则等到运动计划器完成后才执行任务。
如果在任务执行的同时，出现粗略更新间期，则任务会暂停以执行运动计划器。</td></tr>
<tr><td>I/O 任务</td><td>CompactLogix、FlexLogix 和 DriveLogix 控制器使用专门的周期性任务处理 I/O 数据。这些 I/O 任务：
不显示在控制器的任务文件夹中。
不算在控制器的任务限制内。
操作优先级别为 7。
以用户确定的最快 RPI 速率执行。
执行的时间与扫描配置 I/O 模块的时间一样。
用户为任务设置优先级时，考虑 I/O 任务：<table><tr><th>如果希望任务：</th><th>则设置下面的一个优先级：</th></tr><tr><td>中断或延迟 I/O 数据处理</td><td>1 至 6</td></tr><tr><td>与 I/O 数据处理共享控制器时间</td><td>7</td></tr><tr><td>让 I/O 数据处理中断或延迟任务</td><td>8 至 15</td></tr></table></td></tr>
<tr><td>系统开销处理</td><td>系统开销处理是控制器花费在非确定性通信上的时间。
非确定性通信是一种不通过项目 I/O 配置文件夹配置的通信，比如信息指令 (MSG) 以及与 HMI 或工作站的通信。
系统开销处理只中断连续性任务。
系统开销处理时间片指定控制器专用于非确定性通信的时间百分比（除了周期性任务或事件性任务的时间）。
控制器每次执行非确定性通信花费的时间约为 1 ms，然后继续连续性任务。</td></tr>
<tr><td>连续性任务</td><td>用户不为连续性任务设置优先级。它始终运行在最低优先级。所有其它任务都能中断连续性任务。</td></tr>
</table>

示例

下面的例子描述了带有三个用户任务的项目的执行：

任务：	优先级：	周期：	执行时间：	持续时间：：
运动计划器	n/a	8 ms （粗略更新速率）	1 ms	1 ms
事件性任务 1	1	n/a	1 ms	1 至 2 ms
周期性任务 1	2	12 ms	2 ms	2 至 4 ms
I/O 任务 — ControlLogix 和 SoftLogix 控制器不适用。请参阅 第 3-5 页。	7	5 ms （最快的 RPI）	1 ms	1 至 5 ms
系统开销处理	n/a	时间片 = 20%	1 ms	1 至 6 ms
连续性任务	n/a	n/a	20 ms	48 ms

图例： 任务执行。 任务中断 （暂停）。



说明：	
1	首先，控制器执行运动计划器和 I/O 任务 （如果其中一个存在）。
2	执行连续性任务 4 ms 后，控制器触发系统开销处理。
3	到达周期性任务 1 的周期 (12 ms)，因此该任务中断连续性任务。
4	执行连续性任务 4 ms 后，控制器触发系统开销处理。
5	事件性任务 1 的触发发生。 运动计划器完成后执行事件性任务 1。 低优先级任务遇到较长时间的延迟。
6	连续性任务自动重新开始。

RSLogix 5000 编程软件在分发 CD 上包括任务监视工具。用户可以使用该工具分析任务是如何执行的。

为非确定性通信留出足够时间

只有在周期型任务或事件性任务不运行时，才出现非确定性通信。如果用户使用多任务，请确保为非确定性通信留出足够的任务扫描时间和执行间隔。

1. 最高优先级任务的执行时间比更新率少得多。
2. 用户所有任务的总执行时间比最低优先级任务的更新率少得多。

例如：在下面任务的配置中：

任务：	优先级：	执行时间：	更新率
1	较高	20 ms	80 ms
2	较低	30 ms	100 ms
	总执行时间	50 ms	

- 最高优先级任务 （任务 1）的执行时间比更新率少得多（20 ms 比 80 ms 少）。
- 所有任务的总执行时间比最低优先级任务的更新率少得多（50 ms 比 100 ms 少）。

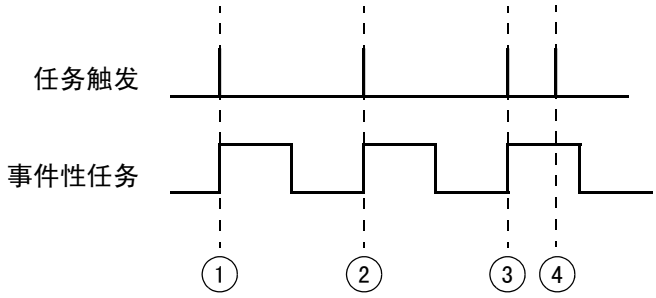
这样通常可以为非确定性通信留出足够时间。

- 如有需要，调整任务更新率使得执行逻辑程序和服务非确定性通信之间达成最好的平衡。
- 如果用户项目中有连续性任务，非确定性通信占据控制器时间的一定百分比 （除了周期性任务或事件性任务的时间）。请参阅第 第 3-5 页 页上的 “系统开销处理”。

避免重叠

重叠 是任务 （周期性或事件性）经过上次触发还在执行时又被触发的情况。

重要 如果出现重叠，控制器忽略引起重叠的触发。换句话来说，用户可能会错过重要任务的执行。



说明：	
1	发生任务触发。 任务执行。
2	发生任务触发。 任务执行。
3	发生任务触发。 任务执行。
4	出现重叠。任务在执行的同时被触发。 触发不会重新开始任务。触发被忽略。

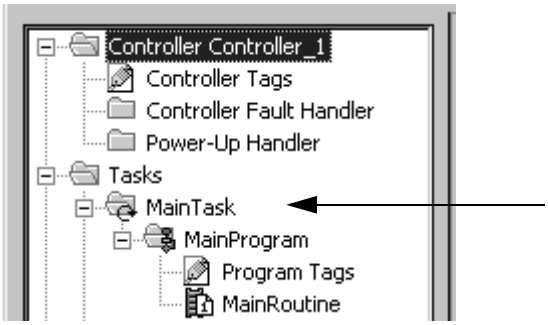
每个任务在再次触发前需要足够的完成时间。请确保任务的扫描时间比触发发生率小得多。如果出现重叠，减少触发任务的频率：

如果任务类型为：	则：
周期性	增加任务周期
事件性	调整系统配置以较低频率触发任务。

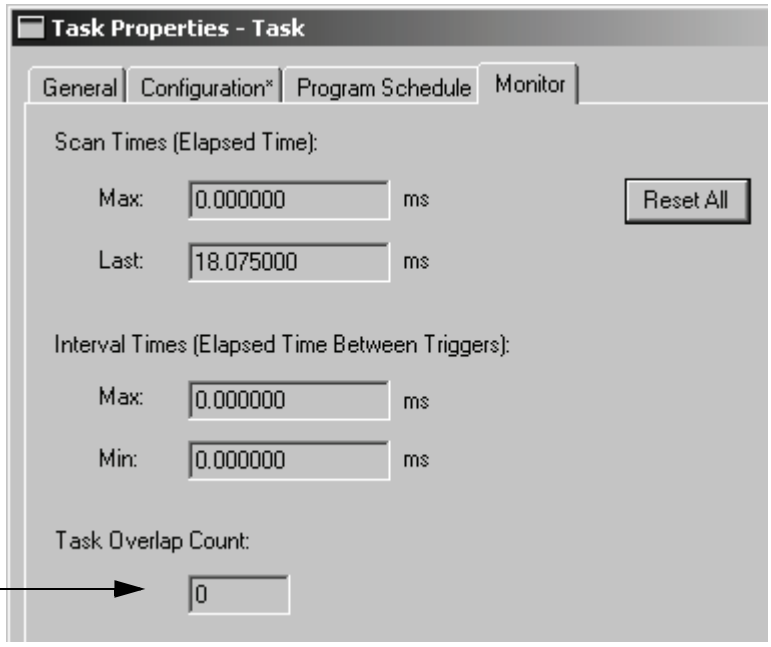
手动检查重叠

手动检查任务是否出现重叠：

1. 在控制器管理器中，右键单击任务并选择 Properties （属性）。



2. 选择 Monitor （监视）选项卡。



计数器最后一次重置以来的重叠次数

3. 单击 OK

编程检查重叠

重叠发生时，控制器执行下列操作：

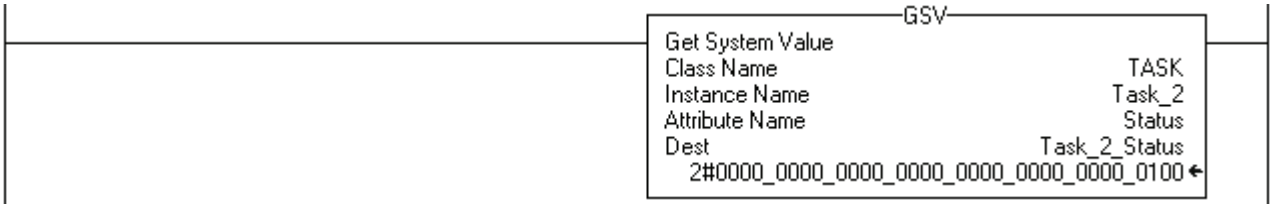
- 记录 FAULTLOG 对象的次故障
- 将任务的重叠信息存储在 TASK 对象中

要编写逻辑程序检查重叠，使用获取系统值指令 (GSV) 监视下面的对象：

如果用户希望：	则访问对象和属性：			
	对象：	**属性：**	**数据类型：**	**说明：**
确定是否所有任务都出现重叠	FAULTLOG	监视故障位	DINT	指示次故障的单独位：
				确定下面的状态： **检查下列位：**
				指令产生次故障 4
				任务出现重叠 6
				串口产生次故障 9
				未安装电池或需要替换 10
确定一个特定的任务是否出现重叠	TASK	状态	DINT	任务的状态信息。一旦控制器设置其中一位，必须手动清零该位。
				确定下面的状态： **检查下列位：**
				EVENT 指令触发任务 （仅事件性任务）。 0
				超时触发任务 （仅事件性任务）。 1
				该任务出现重叠。 2
确定重叠出现次数。	TASK	OverlapCount	DINT	对于事件或周期性任务有效 要清零计数，将属性设置为 0。

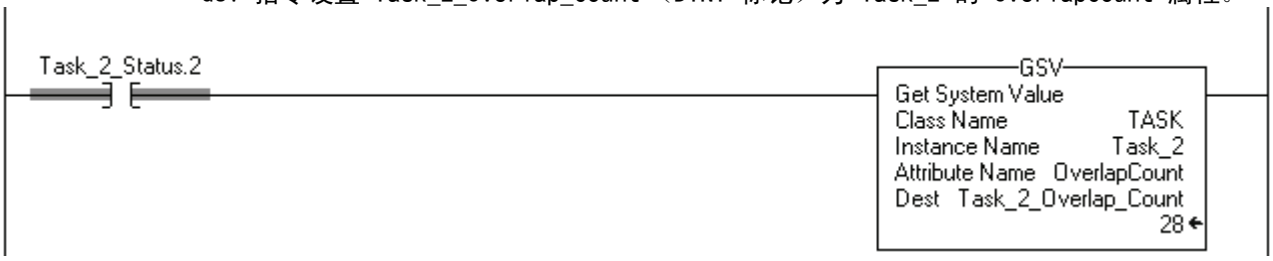
示例 编程检查重叠

1. GSV 指令设置 Task_2_Status 为 Task_2 的状态属性 （DINT 值）。



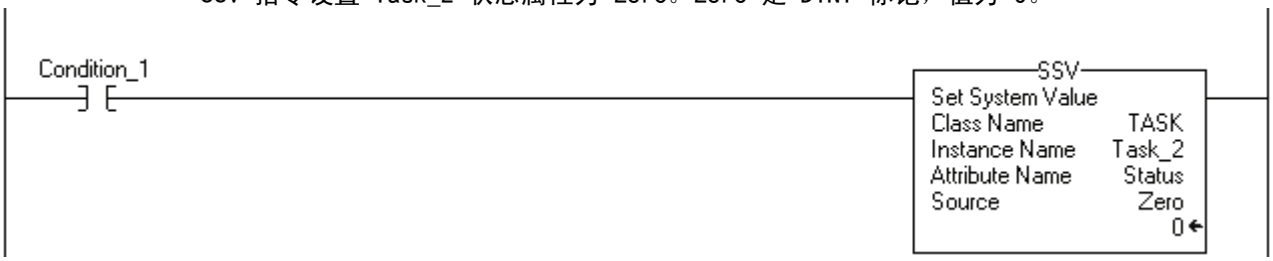
2. 如果 Task_2_Status.2 = 1，则出现重叠，因此获取重叠计数：

GSV 指令设置 Task_2_Overlap_Count （DINT 标记）为 Task_2 的 OverlapCount 属性。



3. 如果 Condition_1 = 1，则清零 Task_2 状态属性位：

SSV 指令设置 Task_2 状态属性为 Zero。Zero 是 DINT 标记，值为 0。

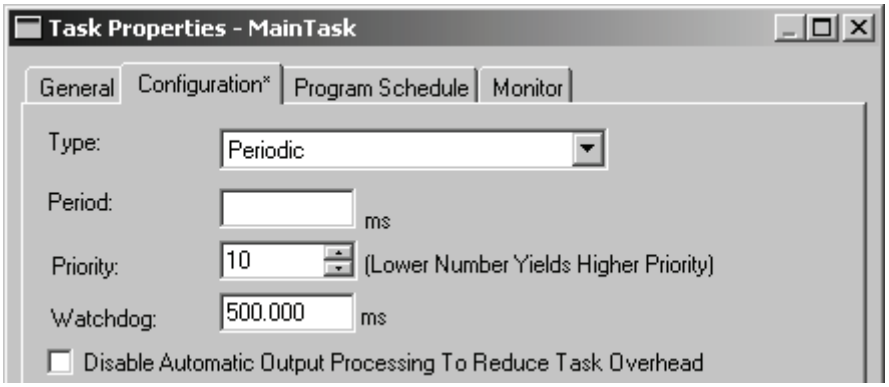


配置任务的输出过程

任务结束后，控制器执行系统中 I/O 模块的系统开销处理操作 （输出过程）。与更新模块不同，输出过程影响系统中 I/O 模块的更新。

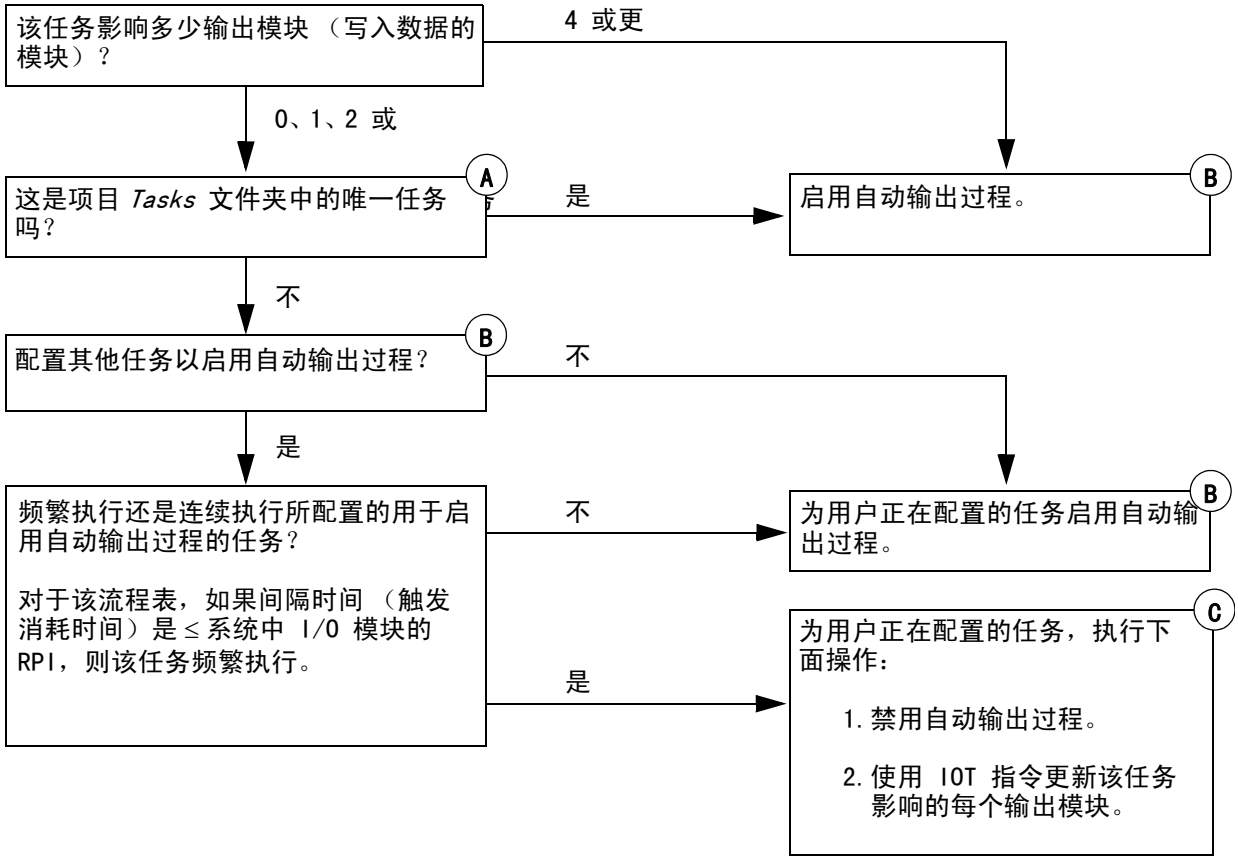
作为一项选择，用户可以关闭特定任务的输出过程，从而减少任务的消耗时间。

在任务结束后启用或禁用输出过程 →



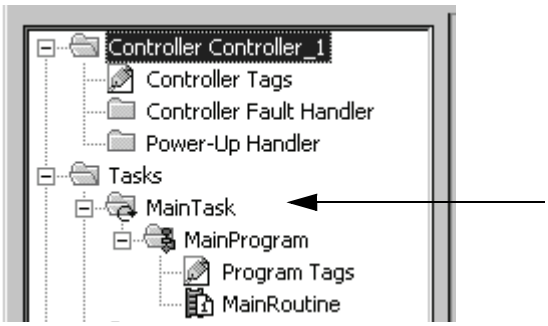
选择如何配置任务的输出过程：

图 3.1 选择如何配置任务的输出过程。

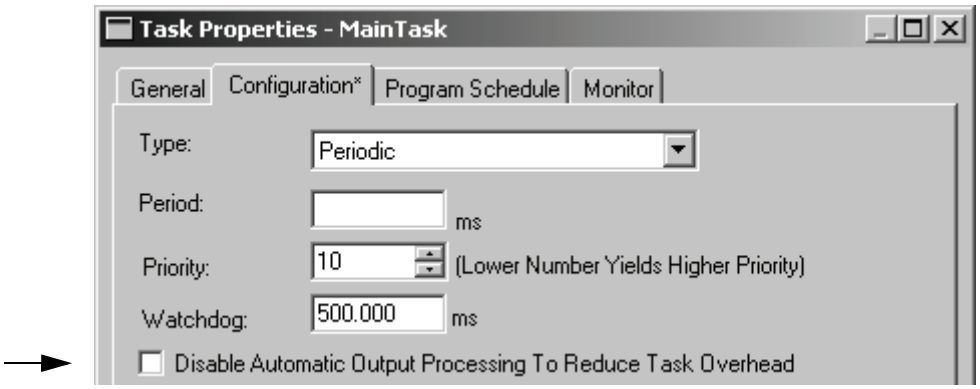


手动配置输出过程

1. 在控制器管理器中，右键单击任务并选择 Properties （属性）。



2. 选择 Configuration （配置）选项卡。



3. 配置任务的输出过程：

如果用户希望：	则：
在任务结束后启用输出过程	清除 （取消选中）禁用自动输出过程以减少任务开销处理复选框 （默认）。
在任务结束后禁用输出过程	选择 （选中）禁用自动输出过程以减少任务开销处理复选框。

4. 单击 OK

以编程方式配置输出过程

要编写逻辑程序配置任务的输出过程，使用设置系统值指令 (SSV)。访问任务的 TASK 对象属性：

如果用户希望：	则访问下面的属性：	数据类型：	指令：	说明：	
在任务结束后启用或禁用输出过程	DisableUpdateOutputs	DINT	GSV SSV	**目的：**	**设置属性值为：**
				在任务结束后启用输出过程	0
				在任务结束后禁用输出过程	1 （或任何非零值）

示例 以编程方式配置输出过程

如果 Condition_1 = 0，则在任务完成后，让 Task_2 处理输出。

1. ONS 指令限制 SSV 指令执行一次扫描。
2. SSV 指令设置 Task_2 的 DisableUpdateOutputs 属性为 0。这允许任务完成后自动处理输出。

如果 Condition_1 = 1，则在任务完成后，不让 Task_2 处理输出。

1. ONS 指令限制 SSV 指令执行一次扫描。
2. SSV 指令设置 Task_2 的 DisableUpdateOutputs 属性为 1。这阻止在任务执行后，自动处理输出。

禁止任务

默认情况下，每个任务按照它的触发 （事件、周期性或连续性）执行。作为一项选择，用户可以在触发发生时阻止任务的执行 （即禁止任务）。这对于项目测试、诊断或启动非常有用。

如果用户希望：	则：
在触发发生时执行任务	不禁止任务 （默认）。
防止发生触发时执行任务	禁止任务。

示例

禁止任务

在运转使用多个任务的系统时，用户可以首先单独测试每个任务。

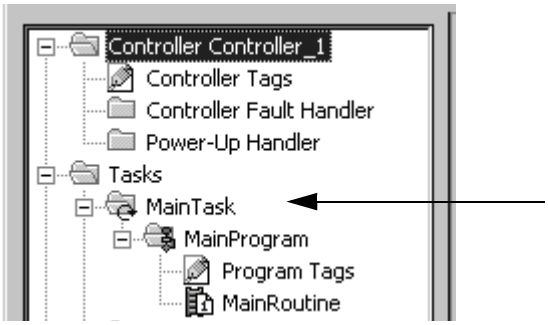
1. 禁止除了要测试的任务以外的所有任务，然后测试该任务。
2. 一旦该任务满足用户要求，禁止该任务并取消禁止另一个任务。
3. 继续该过程直到测试所有任务。

如果某个任务被禁止，当控制器从程序模式转变为运行或测试模式时，控制器仍预扫描该任务。

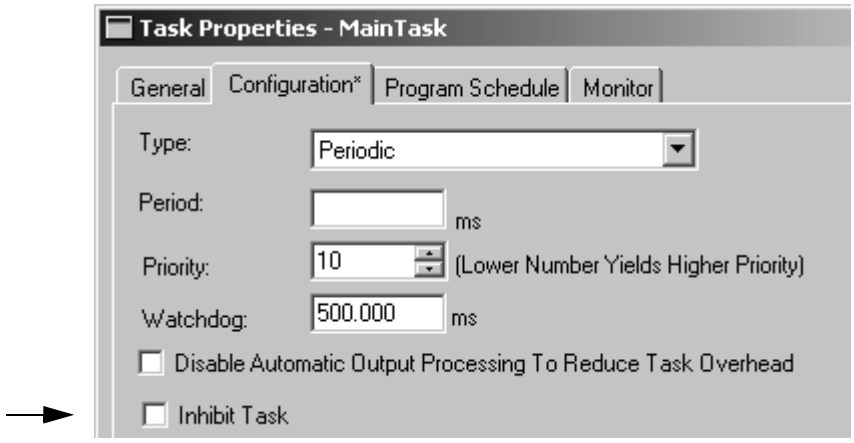
手动禁止或不禁止任务

要手动禁止或不禁止任务的执行，执行下面的操作：

1. 在控制器管理器中，右键单击任务并选择 Properties （属性）。



2. 选择 Configuration （配置）选项卡。



3. 禁止或不禁止任务：

如果用户希望：	则：
在触发发生时执行任务	清零 （取消选中）*Inhibit Task* （禁止任务）复选框 （默认）。
防止发生触发时执行任务	选择 （选中）*Inhibit Task* （禁止任务）复选框。

4. 单击 OK

编程禁止或不禁止任务

编写逻辑程序以禁止或不禁止任务，使用 SSV （设置系统值）指令访问任务 TASK 对象属性。

属性：	数据类型：	指令：	说明：	
InhibitTask	DINT	GSV	阻止任务的执行。	
		SSV	**目的：**	**设置属性值为：**
			启用任务	0 （默认）
			禁止任务	1 （或任何非零值）

示例 编程禁止或不禁止任务

如果 Condition_1 = 0，则执行 Task_2。

1. ONS 指令限制 SSV 指令执行一次扫描。
2. SSV 指令设置 Task_2 的 InhibitTask 属性值为 0。这样取消禁止了任务。

如果 Condition_1 = 1，则不能执行 Task_2。

1. ONS 指令限制 SSV 指令执行一次扫描。
2. SSV 指令设置 Task_2 的 InhibitTask 属性值为 1。这样禁止了任务。

为事件性任务选择触发

如果正确配置，事件性任务会中断所有其他任务，以将响应事件的时间将到最低。每个事件性任务需要一个特定的触发，确定何时执行任务。

在下面的情况下触发事件性任务：	使用下面的触发：	考虑下面的事项：
打开或关闭数字输入设备	模块输入数据状态改变	• 仅一个输入模块可以触发特定的事件性任务。 • 输入模块根据模块状态改变 (COS) 配置触发事件性任务。COS 配置确定在打开或关闭时哪些点提示模块生成数据。这种数据生成 （由于 COS）触发事件性任务。 • 一般情况下，仅为模块上的一个点启用 COS。如果用户为多个点启用 COS，可能出现事件性任务重叠。
模拟模块采样数据	模块输入数据状态改变	• 仅一个输入模块可以触发特定的事件性任务。 • 在通道的每个实时采样后 (RTS)，模拟模块触发事件性任务。 • 模块的所有通道都使用同一个 RTS。
控制器通过使用标记获取新数据	使用标记	• 只有一个使用标记可以触发特定的事件性任务。 • 一般情况下，在生成控制器中使用 IOT 指令显示新数据的生成。IOT 指令在生成标记中设置事件触发。该触发传递给使用标记，同时触发事件性任务。 • 当使用标记触发事件性任务时，事件性任务在所有数据到达后才开始执行。
轴的套准输入打开 （或关闭）	轴套准 1 或 2	• 为了套准输入触发事件性任务，首先执行运动提供套准指令 (MAR)。这允许轴检测套准输入，并接着触发事件性任务。 • 一旦套准输入触发事件性任务，再次执行 MAR 指令重新为下一次套准输入提供轴。 • 如果常规逻辑程序的扫描时间不够快以为下一次套准输入重新提供轴，考虑在事件性任务中执行 MAR 指令。
轴到达监视点位置	轴监视	• 为了套准输入触发事件性任务，首先执行运动提供监视指令 (MAW)。 这允许轴检测监视位置，并接着触发事件性任务。 • 一旦监视位置触发事件性任务，再次执行 MAW 指令重新为下一个监视位置提供轴。 • 如果常规逻辑程序的扫描时间不够快以为下一个监视位置重新提供轴，考虑在事件性任务中执行 MAW 指令
运动计划器完成其执行	运动组执行	• 运动组的粗略更新间期同时触发运动计划器和事件性任务的执行。 • 由于运动计划器中断所有其他任务，所以它首先被执行。如果用户设置事件性任务为最高优先级任务，则在运动计划器后执行该任务。
在程序逻辑中出现的特定条件	EVENT 指令	多个 EVENT 指令可以触发同一个任务。这允许用户从不同的程序执行任务。

这里是事件性任务以及相应触发的一些实例情况：

对于下面的实例情况：	在下面的触发中使用事件性任务：
一条包装线粘合纸盒。 当纸盒传送到粘合位置时，控制器必须立即执行粘合例程。	模块输入数据状态改变
一条生产线使用一个感应传感器检测部件的存在。由于感应传感器打开的时间很短 （脉冲），连续性任务可能在传感器变换时丢失检测数据。	模块输入数据状态改变
在引擎测试时，用户必须捕获并存档模拟数据的每个采样值。	模块输入数据状态改变
控制器 A 产生一组生产数据传送给控制器 B。用户希望控制器 B 在控制器 A 正在更新数组时不使用值：	使用标记
在包装棒糖的生产线，用户必须确保在正确的位置穿孔。 每次套准传感器检测套准标记、检查轴的精确位置并执行所需调整。	轴套准 1 或 2
在装瓶线的贴标站，用户希望检查瓶上的标签位置。在轴到达监视点位置时，检查标签。	轴监视
粘合站点必须根据轴速变化来调整所用胶水的数量。 如有需要，在运动计划器执行后，检查轴的命令速度并改变胶水的数量。	运动组执行
如果程序检测到在生产线上存在危险条件，必须关闭整个生产线。 不管是什么样的危险条件，关闭过程都相同。	EVENT 指令

用户事件性任务中可使用的触发随 Logix5000 控制器的类型改变。

重要 RSLogix 5000 软件可能允许用户为事件性任务配置用户控制器不支持的触发。项目将被校验并成功下载，但是不执行事件性任务。

表 3.1 确定支持所有类型事件触发的 Logix5000 控制器。

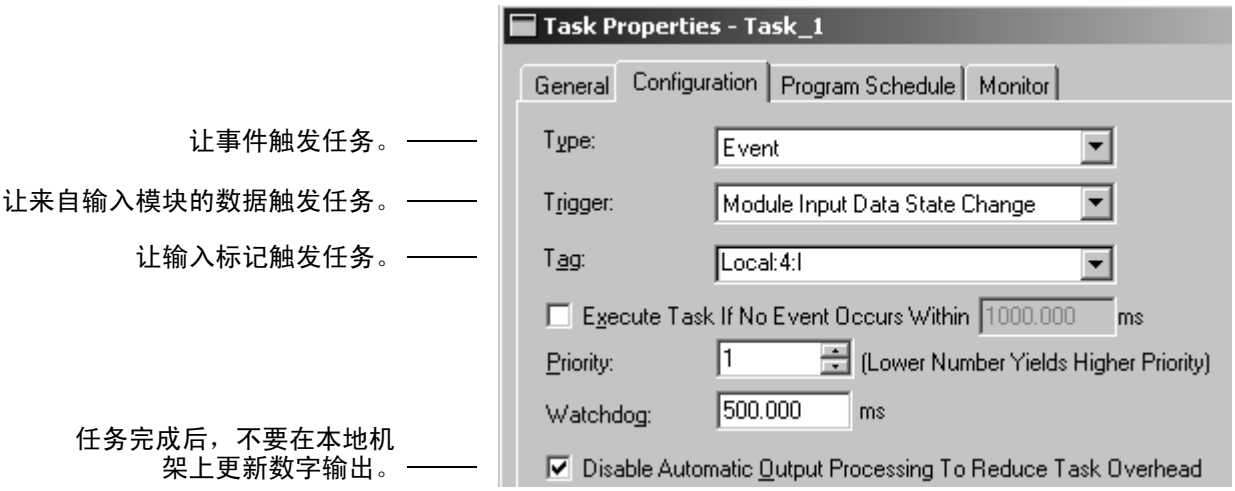
如果用户有下面的控制器：	则可以使用下面的事件性任务触发：					
	模块输入数据状态改变	使用标记	轴套准 1 或 2	轴监视	运动组执行	EVENT 指令
CompactLogix		✔				✔
FlexLogix		✔				✔
ControlLogix	✔	✔	✔	✔	✔	✔
DriveLogix		✔	✔	✔	✔	✔
SoftLogix5800	✔(1)	✔(2)	✔	✔	✔	✔

(1) 需要 1756 I/O 模块或虚拟背板。

(2) SoftLogix5800 控制器仅在 ControlNet 网络生成和使用标记。

使用模块输入数据状态改变触发

要基于输入模块数据触发事件性任务，使用模块输入数据状态改变触发。



I/O 模块如何触发事件性任务

下面的术语适用于输入模块操作：

术语：	定义：
多播	模块在网络上发送数据的同时被多个接收器 （设备）接收的机制。描述 Logix5000 I/O 线的功能，它支持多个控制器同时接收同一个 I/O 模块的输入数据。
请求数据包间隔 (RPI)	RPI 指定了模块多点广播其数据的速率。例如，输入模块以用户设置给模块的 RPI 周期向控制器发送数据。 • 时间值从 0.2 ms （200 微秒）到 750 毫秒。 • 当一个指定的 RPI 时间帧后，模块开始多点广播数据。也称循环更新。
实时采样 (RTS)	RTS 确定模拟模块何时扫描通道，并多点广播数据 （更新输入数据缓冲区并多点广播数据）。 • RPI 指定了模块何时不扫描 （更新）通道而多点广播输入数据缓冲区中的现有数据。 • 每一次产生 RTS 传递时，该模块都重置 RPI 定时器。
状态改变 (COS)	只要一个特定的输入点从 On 转换为 → Off 或从 Off 转换为 → On，COS 参数就指示数字输入模块多点广播数据。 • 用户启用每个点上的 COS。 • 任何一个启用 COS 的点在收到特定改变时，模块多点广播所有点上的数据。 • 在默认情况下，启用 COS 监测所有点的状态从 On 改变为 → Off 和从 Off 改变为 → On。 • 不管是否启用 COS，用户都必须指定一个 RPI。 如果 RPI 时间帧内没有改变状态发生，则模块就以 RPI 指定的速率发送数据。

下面的表格总结了输入模块何时多点广播数据，何时在自身机架内触发事件性任务。

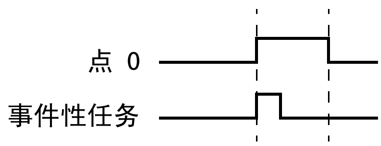
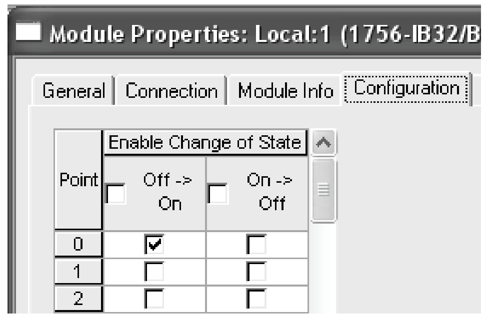
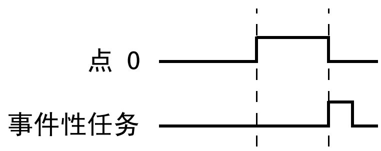
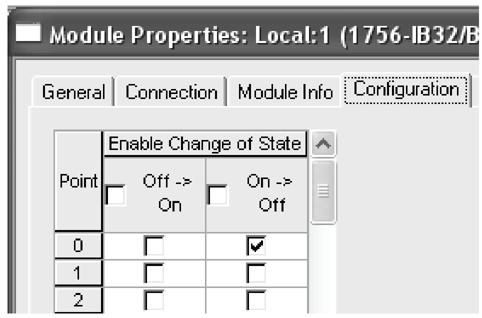
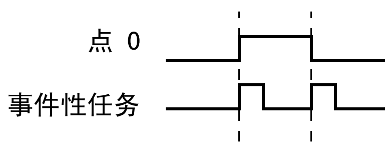
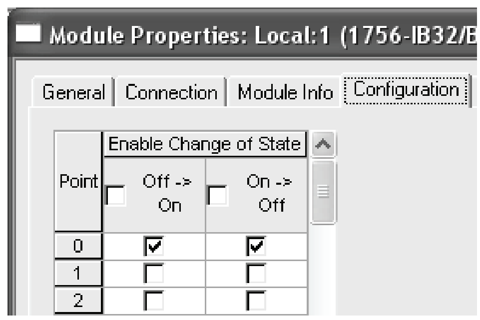
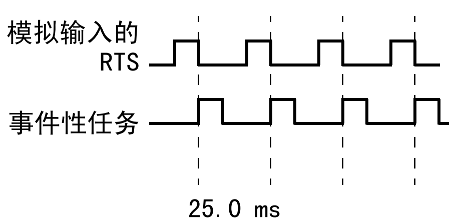
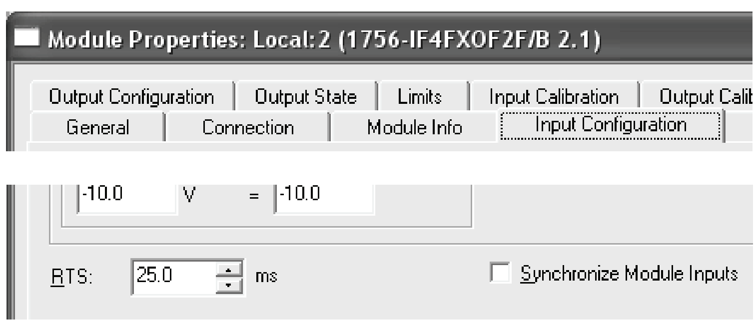
如果输入模块为：	并且：	则多点广播数据：	且触发事件性任务：
数字	模块上的任何点启用 COS	• 在任何点启用 COS 时，接收特定的改变 • 按照 RPI 速率	在任何点启用 COS 时，接收特定的改变
	模块上的任何点都未启用 COS	按照 RPI 速率	从未
模拟	RTS ≤ RPI	按照 RTS 速率 （最近更新过的通道数据）	按照模块的 RTS 速率
	RTS > RPI	• 按照 RTS 速率 （最近更新过的通道数据） • 按照 RPI 速率 （不包含通道更新过的数据）	按照模块的 RTS 速率

如果模块在远程机架上，只有 RPI 值决定控制器什么时候接收网络上的数据和事件触发。

通过下面的这些网络：	控制器接收的数据：
EtherNet/IP	平均值接近 RPI 值
ControlNet	实际包间隔 （≤ RPI）

下面是显示 COS 和 RTS 配置的一些实例：

重要	如果用户使用数字模块触发事件性任务，只要配置模块上的一点来改变状态 (COS)。 如果用户配置多点，则发生任务重叠。

如果用户希望：	则按照下面的方法配置输入模块 （仅以点 0 为例）
	状态改变 → 没有改变现有点的状态 → 
	状态改变 → 没有改变现有点的状态 → 
	状态改变 → 没有改变现有点的状态 → 
	 输入的实时采样

确保用户模块能够触发事件性任务

使用输入模块去触发一个事件性任务，该模块必须支持事件性任务触发。如果模块在远程位置，相关的通信模块必须也支持事件触发。

下表列出我们已经测试的支持事件触发的 Rockwell Automation 模块。第三方的模块可能也支持事件性任务触发。用户使用该模块前，应与供应商验证模块的操作。

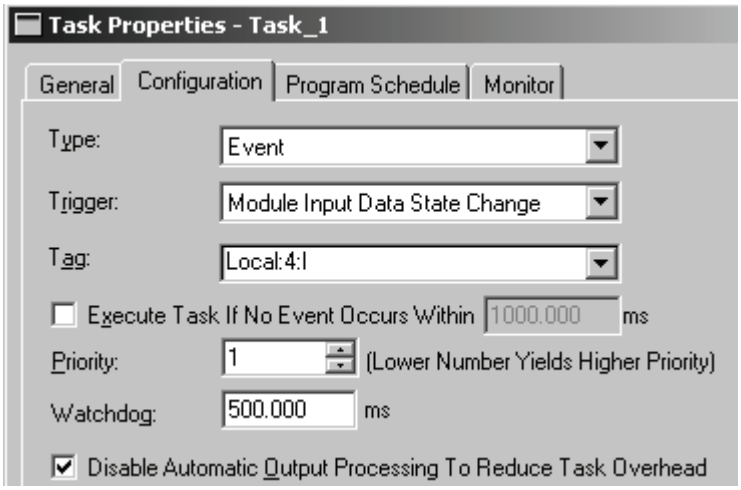
种类：	模块：
支持状态改变的数字 I/O 模块	1756-IA8D 1756-IA16 1756-IA16I 1756-IA32 1756-IB16 1756-IB16D 1756-IB16I 1756-IB16ISOE 1756-IB32 1756-IC16 1756-IG16 1756-IH16I 1756-IH16ISOE 1756-IM16I 1756-IN16 1756-IV16 1756-IV32
支持实时采样的模拟 I/O 模块	1756-IF16 1756-IF4FXOF2F/A 1756-IF6CIS 1756-IF6I 1756-IF8 1756-IR6I 1756-IT6I 1756-IT6I2
提供机架优化连接的通信模块	1756-CNB/A 1756-CNB/B 1756-CNB/D 1756-CNBR/A 1756-CNBR/B 1756-CNBR/D 1756-DNB 1756-ENBT/A 1756-SYNCH/A 1784-PCIDS/A
符合 CIP 事件通信的通用 I/O 模块	1756-MODULE 1789-MODULE

输入事件性任务列表

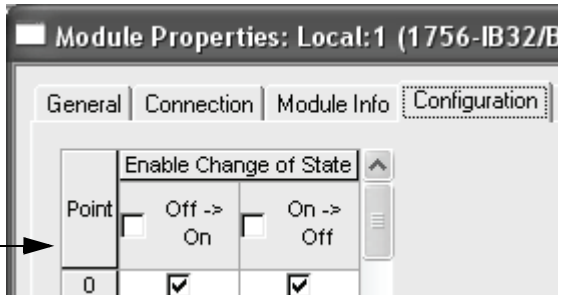
对于下面的：		确保：
❑	1. 输入模块类型	使用下面的模块作出最快响应： • 要作出最快的数字响应，使用 1756-IB32/B 模块。 • 要作出最快的模拟响应，使用 1756-IF4FXOF2F 模块。
❑	2. I/O 模块位置	将触发事件的模块和响应事件的模块 （输出）置于控制器的同一个机架中。 远程模块向响应时间中添加网络通信时间。
❑	3. 本地模块的数量	限制本地机架中模块的数量。 额外的模块增加了背板延迟的可能
❑	4. 改变状态 (COS)	如果数字设备触发事件，则仅对触发事件性任务的点启用 COS。 • 为触发任务的转变类型启用改变状态，从 Off → On、On → Off 或两者。 • 如果用户同时为 Off → On 和 On → Off 启用 COS，则只要点打开或关闭就触发一个事件性任务。确保输入的持续时间比任务的扫描时间长。否则，可能会发生重叠。 • 禁用 （清零）输入模块上现有点的 COS。如果用户配置模块上多点的 COS，则每个点都可以触发事件性任务。这可能会引起重叠。
❑	5. 任务优先级	配置事件性任务的优先级为最高。 如果周期型任务优先级较高，则事件性任务必须等到周期性任务完成后执行。
❑	6. 运动计划器	运动计划器中断其它所有任务 （不管任务的优先级）。 • 轴的数量和运动组粗略更新间期影响运动计划器的执行时间和频率。 • 如果在触发任务时执行运动计划器，则运动计划器完成后执行任务。 • 如果在任务执行的同时，出现粗略更新间期，则需要暂停任务以执行运动计划器。
❑	7. 事件性任务的数量	限制事件性任务的数量 每个额外的任务减少其它任务的处理时间。这可能会引起重叠。
❑	8. 自动输出过程	对于事件性任务，用户可以禁用自动输出过程 （默认）。这样减少任务执行时间。 要校验这些决定，请参阅第 第 3-13 页 页上的 图 3.1。
❑	9. IOT 指令	对用户在事件性任务中引用的每个输出模块使用 IOT 指令。 IOT 指令替代了模块的 RPI 值并立即发送数据。

示例

零件移过换向器位置时，控制器必须确定是否打开换向器。一旦打开换向器，则必须在下一个零件到达该位置前将其关闭。由于生产线的速度影响，事件性任务控制换向器。



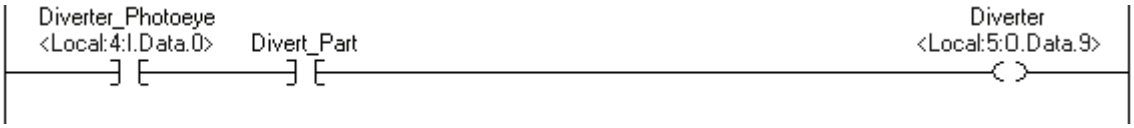
换向器位置的摄像头指示零件何时到达该位置。输入连接到本地机架槽 4 中的模块。



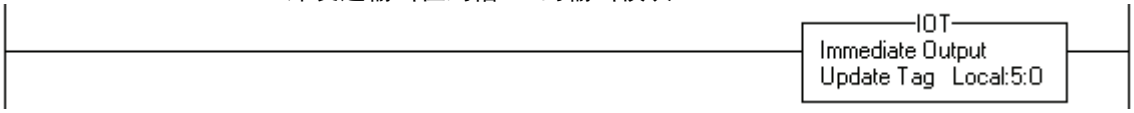
换向器摄像头（point 0）配置关闭和打开两种状态的改变。这允许摄像头在打开和关闭时触发事件性任务。

事件性任务使用下面的逻辑程序控制换向器。

如果 Diverter_Photoeye = 1 （零件在换向器位置）
且 Divert_Part = 1 （转移该零件）
　　然后 Diverter = 1 （打开换向器）
否则 Diverter = 0 （关闭换向器）



立即发送输出值到槽 5 的输出模块。



估计吞吐量

要估计从输入到输出的吞吐量时间，使用下面的工作表：

考虑：	值：
1. 触发事件性任务的模块的输入滤波时间是多少？ 一般以毫秒表示。 转换为微秒 (µs)。	µs
2. 触发事件性任务的模块的硬件响应时间是多少？ 确保使用的转变类型合适 （Off → On 或 On → Off）。请参阅第 3-28 页上的表 3.2。	µs
3. 背板通信时间是多少？ **如果机架大小为：** **使用下面的值 （最糟情况）：** 4 槽 13 µs 7 槽 22 µs 10 槽 32 µs 13 槽 42 µs 17 槽 54 µs	µs
4. 事件性任务程序的总执行时间是多少？	µs
5. 背板通信时间是多少？ （与 第 3 步值相同。）	µs
6. 输出模块的硬件响应时间是多少。	µs
7. 在 6 中添加步骤 1。这是吞吐量的最小估计值，其中执行运动计划器或其他任务不会延迟或中断事件性任务。	µs
8. 运动组的扫描时间是多少？	µs
9. 比此事件性任务的优先级高的任务 （如果存在）的总扫描时间是多少？	µs
10. 添加步骤 7 到 9。这是吞吐量的额定估计值，其中执行运动计划器或其他任务会延迟或中断事件性任务。	µs

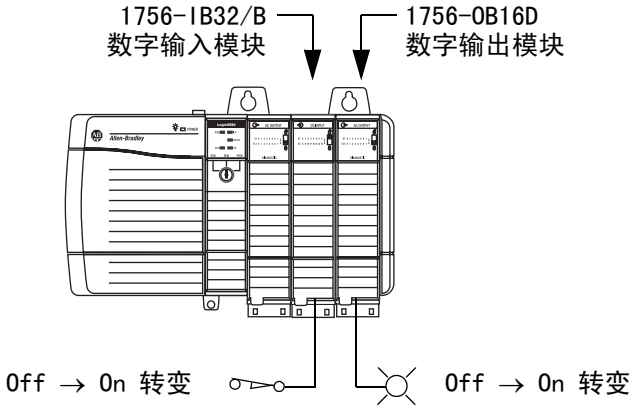
表 3.2 常用于事件性任务的 1756 I/O 模块的额定硬件响应时间。

模块：	额定响应时间 μ：			
	25ϒϑ C		60ϒϑ C	
	Off → On	On → Off	Off → On	On → Off
1756-IB16	265	582	265	638
1756-IB16D	303	613	305	673
1756-IB32/B	330	359	345	378
1756-IV16	257	435	254	489
1756-IV32	381	476	319	536
1756-OB16D	48	519	51	573
1756-OB16E	60	290	61	324
1756-OB32	38	160	49	179
1756-OV16E	67	260	65	326
1756-OV32E	65	174	66	210

示例

估计吞吐量

下面的例子显示系统吞吐量的考虑事项。本例中，吞吐量是打开输入到打开输出的时间。



考虑：	值：
1. 触发事件性任务的模块的输入滤波时间是多少？ 一般以毫秒表示。 转换为微秒 (µs)。	0 µs
2. 触发事件性任务的模块的硬件响应时间是多少？ 确保使用的转变类型合适（Off → On 或 On → Off）。 请参阅第 3-28 页上的表 3.2。	330 µs
3. 背板通信时间是多少？	13 µs

如果机架大小为：	使用下面的值 （最糟情况）：
4 槽	13 µs
7 槽	22 µs
10 槽	32 µs
13 槽	42 µs
17 槽	54 µs

4. 事件性任务程序的总执行时间是多少？	400 µs
5. 背板通信时间是多少？ （与第 3 步 值相同。）	13 µs
6. 输出模块的硬件响应时间是多少？	51 µs
7. 添加步骤 1 到 6。这是吞吐量的最小估计值，其中执行运动计划器或其他任务不会延迟或中断事件性任务。	807 µs
8. 运动组的扫描时间是多少？	1130 µs
9. 比此事件性任务的优先级高的任务 （如果存在）的总扫描时间是多少？	0 µs
10. 添加步骤 7 到 9。这是吞吐量的额定估计值，其中执行运动计划器或其他任务会延迟或中断事件性任务。	1937 µs

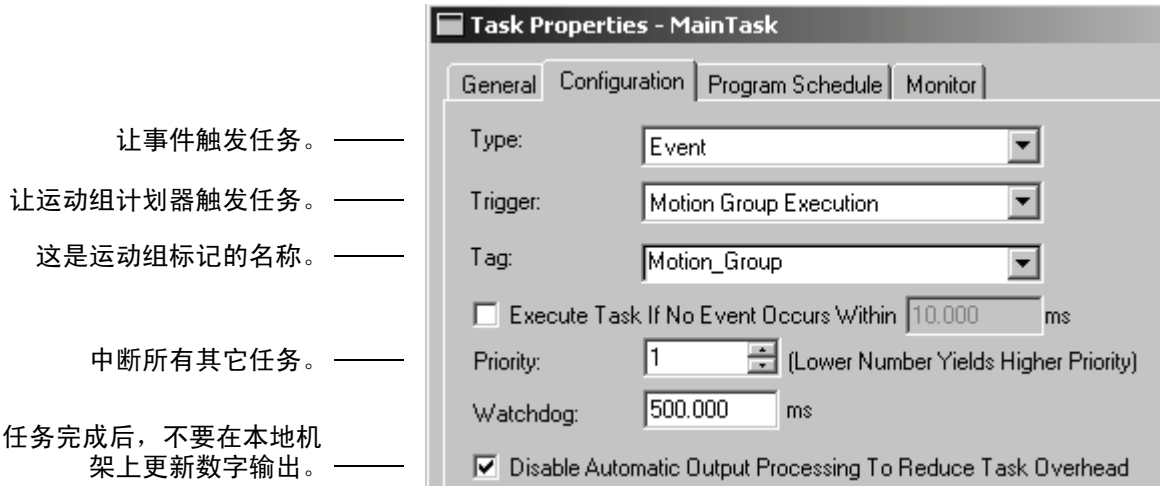
额外的考虑事项

下面的考虑事项影响事件性任务的扫描时间，它影响其响应输入信号的速度。

考虑：	说明：
事件性任务中的代码数	每个逻辑程序元素 （梯级图、指令、结构化文本等）都向任务添加扫描时间。
任务优先级	如果事件性任务优先级不是最高，则优先级较高的任务可能延迟或中断事件性任务的执行。
CPS 和 UID 指令	如果其中一个指令被激活，事件性任务不能中断当前正在执行的任务。（该任务带有 CPS 或 UID 指令。）
通信中断	不管任务的优先级如何，控制器的下列操作会中断该任务 • 与 I/O 模块通信 带有大数据包的模块对其产生较大影响，如 1756-DNB 模块。 • 串口通信

使用运动组触发

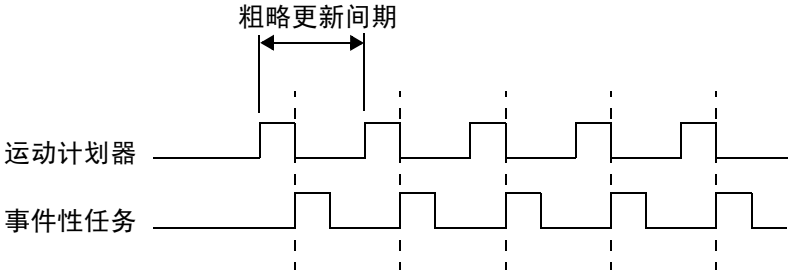
要将事件性任务的执行和运动计划器的执行组合，使用运动组执行触发。



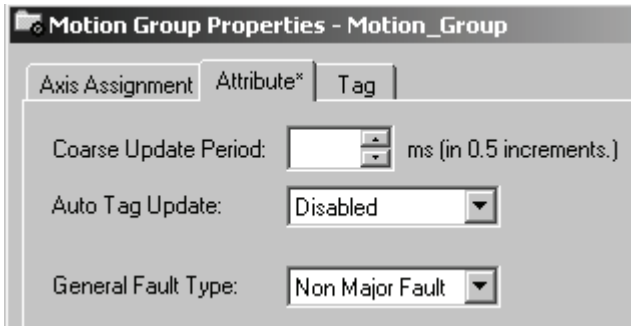
运动组执行触发按下面的步骤运行：

- 运动组的粗略更新间期同时触发运动计划器和事件性任务的执行。
- 由于运动计划器中断所有其他任务，所以它首先被执行。 如果用户将事件性任务的优先级设置为最高，则在运动计划器执行后立即执行该任务。

时间图表显示运动计划器和事件性任务两者之间的关系。



运动组的粗略更新间期同时触发运动计划器和事件性任务。

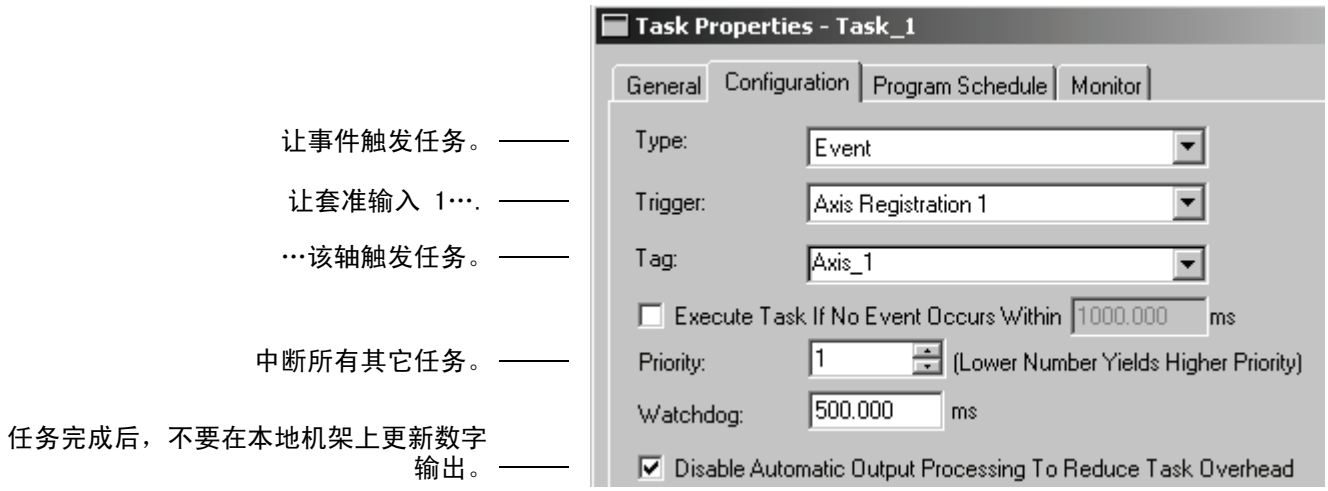


运动组任务清单

对于下面的：		确保：
❑	1. 扫描时间	确保事件性任务的扫描时间比运动组粗略更新间期小得多。否则，可能发生任务重叠。
❑	2. 任务优先级	配置事件性任务的优先级为最高。 如果周期性任务优先级较高，则事件性任务必须等到周期性任务完成后执行。
❑	3. 事件性任务的数量	限制事件性任务的数量 每个额外的任务减少其它任务的处理时间。 这可能会引起重叠。
❑	4. 自动输出过程	对于事件性任务，用户可以禁用自动输出过程 （默认）。 这样减少任务执行时间。 要校验上面的决定，请参阅第 第 3-13 页 页上的图 3.1。

使用轴套准触发

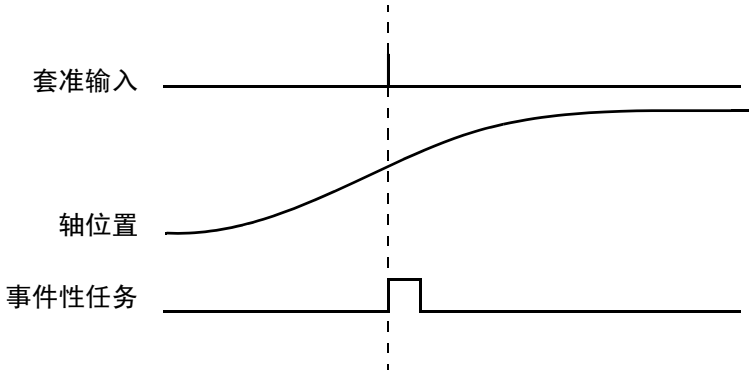
要让轴套准输入触发事件性任务，使用轴套准 （1 或 2）触发。



当特定的套准输入满足触发条件时，会触发事件性任务。

- 在配置事件性任务时，指定用户要触发任务的套准输入。选择轴套准 1 或轴套准 2。
- 用户必须首先使用运动提供套准 (MAR) 指令提供套准输入。
- 在 MAR 指令中，触发条件操作数确定哪种套准输入转变 （Off → On 或 On → Off）触发事件性任务。
- 一旦套准输入触发任务，用户必须重新提供套准输入。

下面的时间图表显示套准输入和事件性任务两者之间的关系。



轴套准任务清单

对于下面的：		确保：	
❑	1. 套准输入	提供套准输入 （MAR 指令）这允许轴检测套准输入并触发事件性任务。 • 首先，提供套准输入检测第一个触发条件。 • 每个事件性任务执行后重新提供套准输入。 • 以足够快的速度重新提供套准输入以检测每个触发条件。	
		如果用户常规的逻辑程序为：	**则：**
		以足够快的速度重新在触发条件间隔内提供套准输入 例如：用户常规逻辑程序在套准输入间至少完成 2 次扫描。	如有必要，在用户常规逻辑程序中提供套准输入。
		不能 以足够快的速度重新提供套准输入	在事件性任务中提供套准输入。
❑	2. 任务优先级	配置事件性任务的优先级为最高。 如果周期型任务优先级较高，则事件性任务必须等到周期性任务完成后执行。	
❑	3. 事件性任务的数量	限制事件性任务的数量 每个额外的任务减少其它任务的处理时间。 这可能会引起重叠。	
❑	4. 自动输出过程	对于事件性任务，用户可以禁用自动输出过程 （默认）。 这样减少任务执行时间。 要校验上面的决定，请参阅第 第 3-13 页 页上的 图 3.1。	

示例

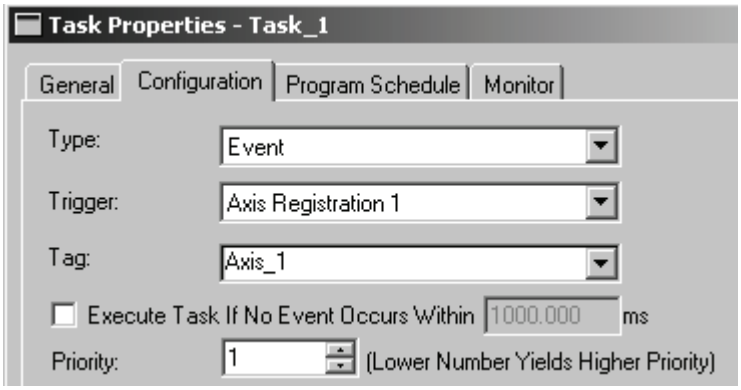
在包装棒糖的生产线，必须确保在正确的位置穿孔。

- 每次套准传感器检测套准标记、检查轴的精确位置并执行所需调整。
- 由于生产线的速度影响，用户必须在事件性任务内提供套准输入。

一个套准传感器作为套准输入 1 连接… →

…轴被命名为 *Axis_1*。 →

该事件性任务中断所有其它任务。 →



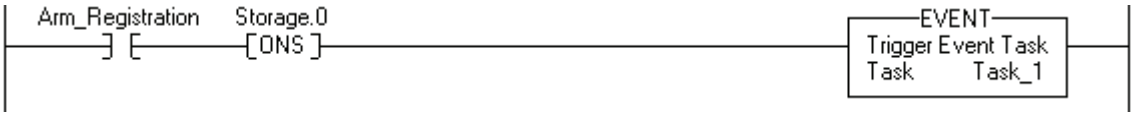
逻辑程序提供并重新提供套准输入。

连续性任务

如果 Arm_Registration = 1 （系统准备查找套准标记），则

ONS 指令限制 EVENT 指令执行一次扫描。

EVENT 指令触发 Task_1 的执行 （事件性任务）。



Task_1 （事件性任务）

GSV 指令设置 Task_Status （DINT 标记）为事件性任务的状态属性。在实例名属性中，THIS 表示指令所在任务的 TASK 对象 （即 Task_1）。

接下页

如果 Task_Status.0 为 1，则 EVENT 指令触发事件性任务。在连续性任务中，EVENT 第一次提供套准。

JMP 指令引起控制器跳转到 Arm LBL 指令。它跳过例程的所有逻辑程序，除了提供轴套准的梯级语句。

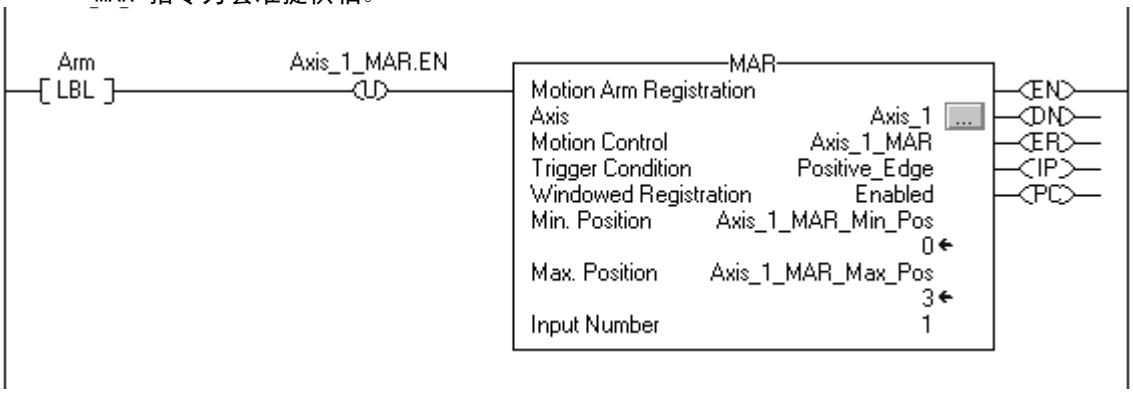
-
- 其他逻辑程序
-

MAR 指令在每次任务执行和为套准提供 Axis_1 时执行。

OTU 指令设置 MAR 指令的 EN 位为 0。

- MAR 是一条转变指令。
- 要执行 MAR 指令，梯形条件必须从假变为真。
- 通过第一次清零 EN 位，指令通过梯形条件从假变为真来响应。

MAR 指令为套准提供轴。

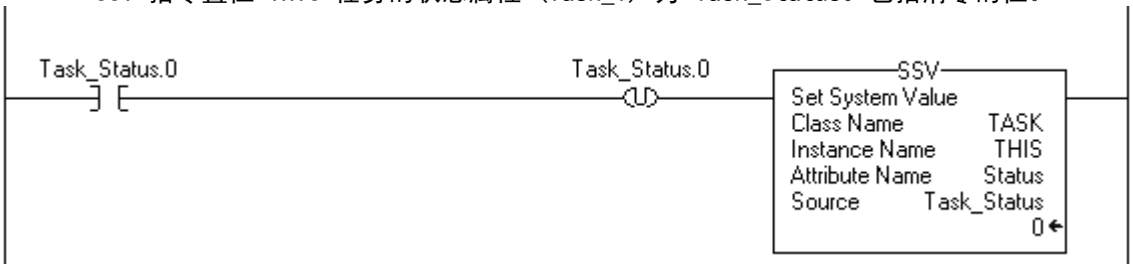


一旦置位状态属性后，控制器不再清零该位。要使用一位记录新的状态信息，用户必须手动清零该位。

如果 Task_Status.0 为 1，则清零该位。

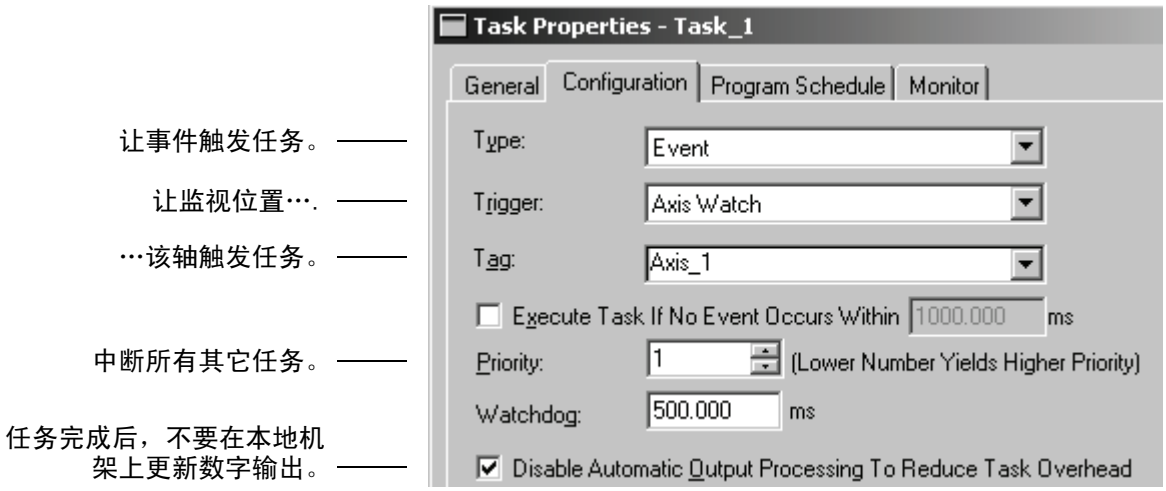
OTU 指令将 Task_Status.0 置 0。

SSV 指令置位 THIS 任务的状态属性 (Task_1) 为 Task_Status。包括清零的位。



使用轴监视触发

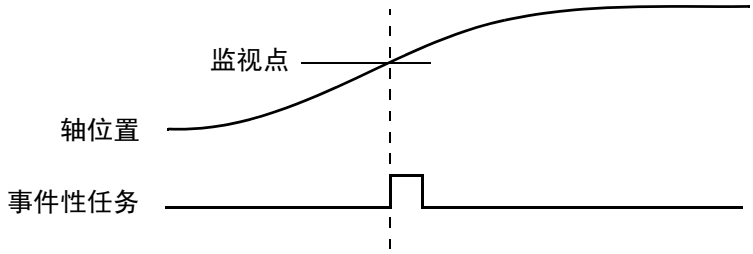
要让轴的监视位置触发事件性任务，使用轴监视触发。



在轴到达监视位置时，触发事件性任务。

- 用户必须首先使用运动提供监视指令 (MAW) 提供轴监视位置。
- 在 MAW 指令中，触发条件操作数定义触发事件性任务轴必须移动的方向。
- 一旦轴到达监视位置且触发事件性任务，用户需要再次为下一个监视位置提供轴。

下面的时间图表显示监视位置和事件性任务两者之间的关系。



轴监视任务清单

对于下面的：		确保：	
❑	1. 监视位置	使用 MAW 指令设置监视位置。这允许轴在到达监视位置时触发事件性任务。 • 首先，提供轴检测第一个监视位置。 • 一旦轴到达监视位置且触发事件性任务，再次为下一个监视位置提供轴。 • 以足够快的速度重新提供轴以检测每个监视位置。	
		如果用户常规的逻辑程序为：	**则：**
		以足够快的速度重新在监视位置之间提供轴。 例如：用户常规逻辑程序在监视位置间至少完成 2 次扫描。	如有必要，在用户常规逻辑程序中提供轴。
		不能 以足够快的速度重新提供轴	在事件性任务中提供轴。
❑	2. 任务优先级	配置事件性任务的优先级为最高。 如果周期型任务优先级较高，则事件性任务必须等到周期性任务完成后执行。	
❑	3. 事件性任务的数量	限制事件性任务的数量 每个额外的任务减少其它任务的处理时间。 这可能会引起重叠。	
❑	4. 自动输出过程	对于事件性任务，用户可以禁用自动输出过程 （默认）。 这样减少任务执行时间。 要校验上面决定，请参阅第 第 3-13 页 页上的图 3.1。	

示例

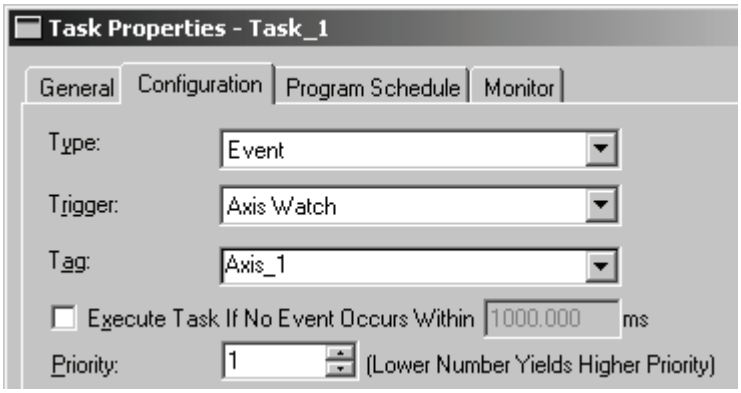
在装瓶线的贴标站，用户希望检查瓶上的标签位置。

- 轴到达监视点位置时，检查标签并执行必要的调整。
- 由于生产线速度的影响，用户必须在事件性任务内为监视位置提供轴。

让监视位置…

…名为 *Axis_1* 的轴触发事件性任务。

该事件性任务中断所有其它任务。



该逻辑程序为监视位置提供和重新提供轴。

连续性任务

如果 Arm_Watch = 1 （系统准备设置监视位），则

ONS 指令限制 EVENT 指令执行一次扫描。

EVENT 指令触发 Task_1 的执行 （事件性任务）。

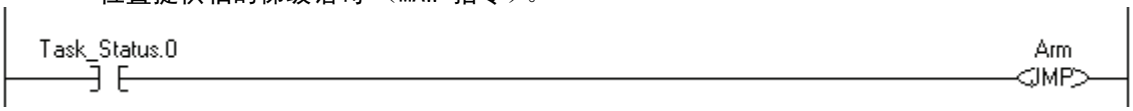
Task_1 （事件性任务）

GSV 指令设置 Task_Status （DINT 标记）为事件性任务的状态属性。 在实例名属性中，THIS 表示指令所在任务的 TASK 对象 （即 Task_1）。

接下页

如果 Task_Status.0 为 1，则 EVENT 指令触发事件性任务。 在连续性任务中，EVENT 第一次设置监视位置。

JMP 指令引起控制器跳转到 Arm LBL 指令。 它跳过例程中的所有逻辑程序，除了为监视位置提供轴的梯级语句 （MAW 指令）。



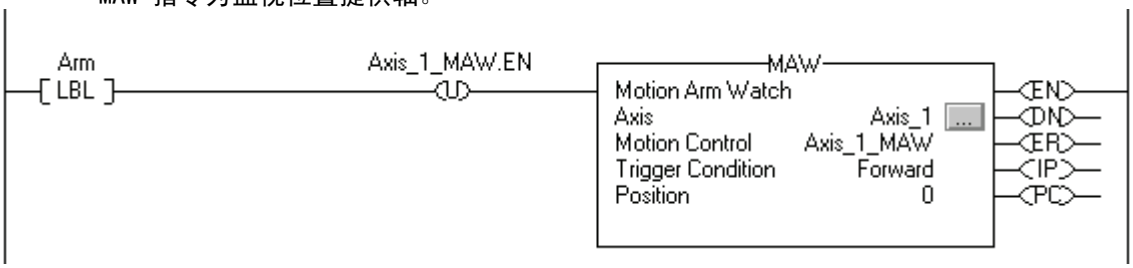
-
- 其他逻辑程序
-

MAW 指令在每次任务执行和为监视位置提供 Axis_1 时执行。

OTU 指令设置 MAW 指令的 EN 位为 0。

- MAW 是一条转变指令。
- 要执行 MAW 指令，梯形条件必须从假变为真。
- 通过第一次清零 EN 位，指令通过梯形条件从假变为真来响应。

MAW 指令为监视位置提供轴。

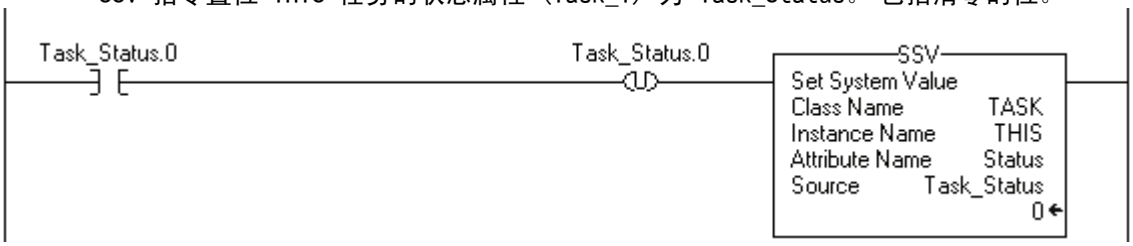


一旦置位状态属性后，控制器不再清零该位。 要使用一位记录新的状态信息，用户必须手动清零该位。

如果 Task_Status.0 为 1，则清零该位。

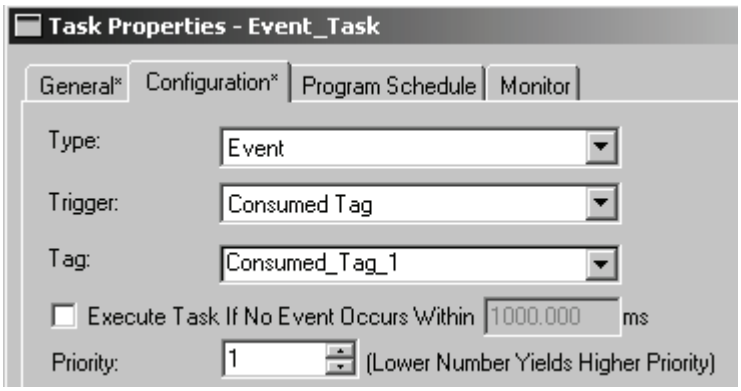
OTU 指令将 Task_Status.0 置 0。

SSV 指令置位 THIS 任务的状态属性 （Task_1） 为 Task_Status。 包括清零的位。

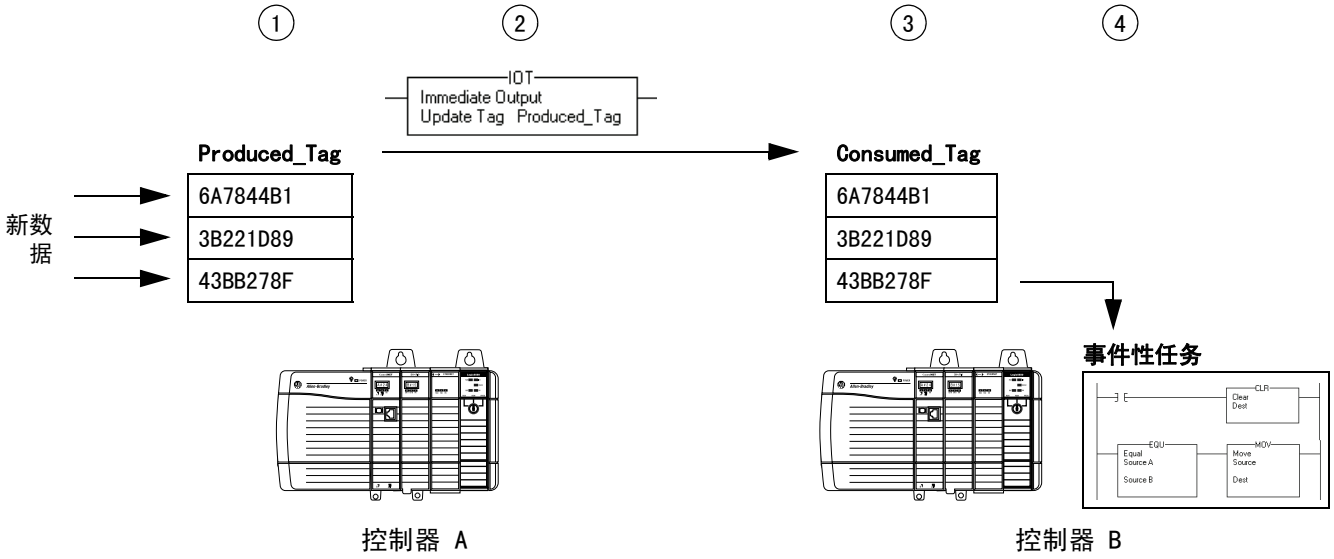


通过使用标记触发

要基于来自使用标记的数据触发事件性任务，通过使用标记触发。



生成标记和使用标记之间的关系可以传递事件触发数据到使用者控制器。一般情况下，用户可以使用一个立即输出（IOT）指令向使用者控制器发送事件触发。



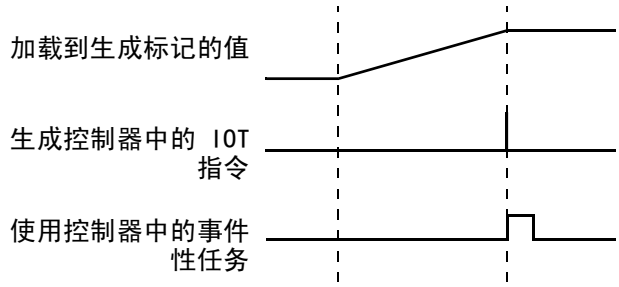
说明：	
1	在控制器 A 中，逻辑程序更新生成标记值。
2	一旦完成更新，控制器 A 执行 IOT 指令向控制器 B 发送数据和事件触发
3	控制器 B 使用新数据。
4	在控制器 B 更新使用标记后，执行事件性任务。

控制器间的网络类型决定了使用控制器何时通过 IOT 指令接收新数据和事件触发。

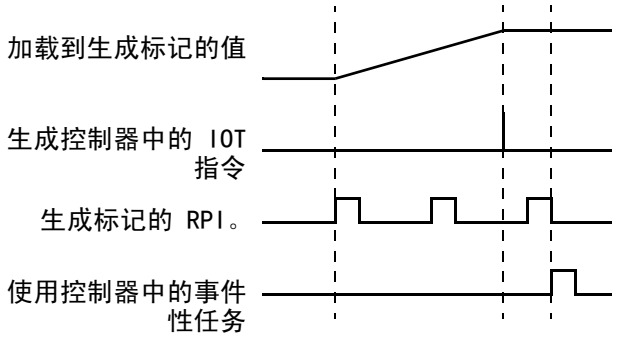
使用下面的控制器：	通过下面的这些网络：	使用设备接收数据和事件触发：
ControlLogix	背板	立即
	EtherNet/IP 网络	立即
	ControlNet 网络	在使用标记的实际包间隔 (API) 内（连接）
SoftLogix5800	用户仅可以在 ControlNet 网络生成和使用标记。	在使用标记的实际包间隔 (API) 内（连接）

下面的图表比较在 EtherNet/IP 和 ControlNet 网络上通过 IOT 指令接收数据。

EtherNet/IP 网络 （ControlLogix 控制器）

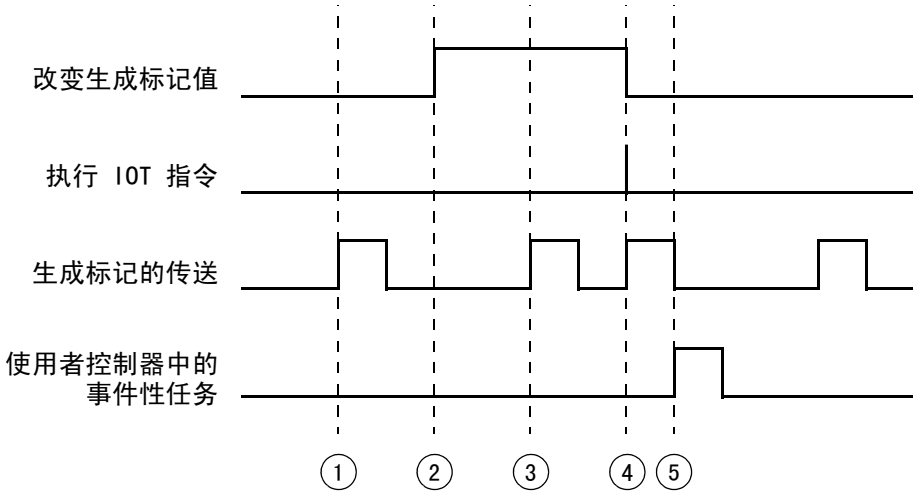


ControlNet 网络



保持数据的完整性

带有使用标记触发的事件性任务提供简单机制向控制器传递数据，同时保证控制器不在数据改变时使用数据。

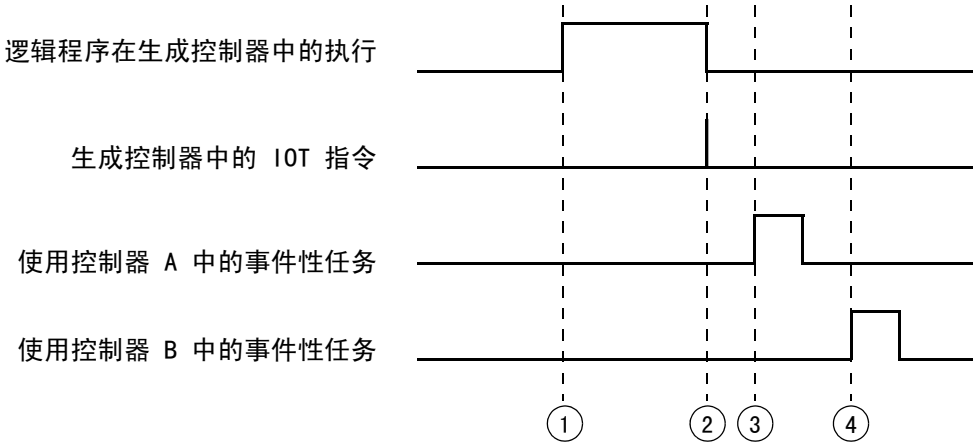


说明：	
1	生成标记出现 RPI。 生成标记向使用控制器传输旧数据。
2	生成者控制器开始更新生成标记值。
3	生成标记再次出现 RPI。 生成标记同时向使用控制器传输新旧数据。
4	生成者控制器完成生成标记值更新。 生成者控制器执行立即输出指令 （IOT）。 生成标记立即向使用控制器传输所有新数据。
5	在使用者控制器接收到所有数据后，执行事件性任务。

尽管生成控制器在加载新数据后立即执行 IOT 指令，但是直到使用标记接收到所有新数据后才触发事件性任务 （使用控制器中）。这保证控制器使用完整的新数据操作。

同步多个控制器

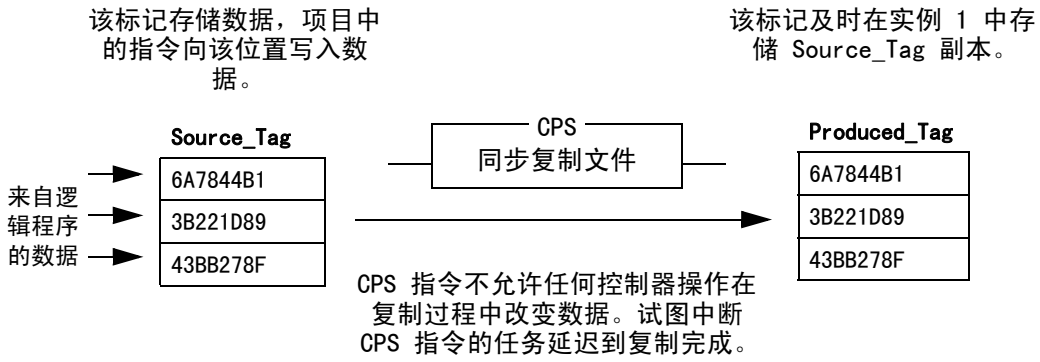
用户也可以使用生成标记和使用标记关系同步控制器。在这种情况下，生成标记和使用标记仅作为触发机制。



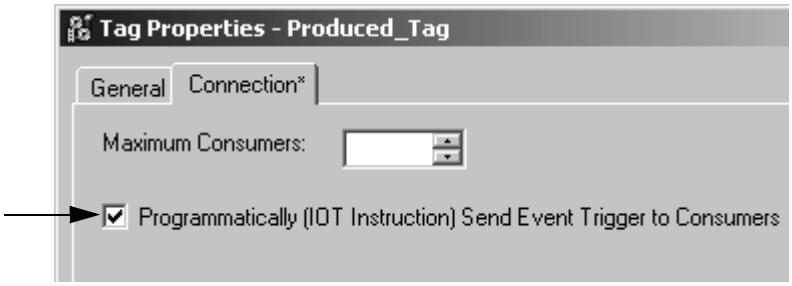
说明：	
1	第一个控制器执行操作时，其他控制器需要和它保持同步。
2	操作完成后，控制器执行 IOT 指令。IOT 指令将生成标记作为目标。
3	控制器 A 接收到生成标记后，执行事件性任务。
4	控制器 B 接收到生成标记后，执行事件性任务。

生成者控制器的清单

对于下面的：		确保：
❑	1. 数据缓冲	如果用户要及时发送数据的完整映像，则生成数据副本，如下面所示：



❑	2. 生成标记属性	在生成标记的连接属性中，选中框：

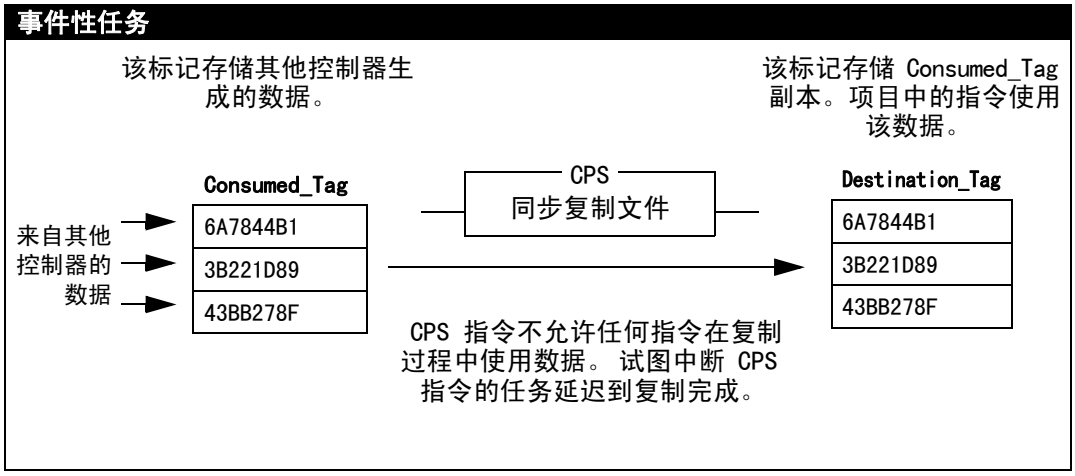


如果用户清零 （取消选中）该框，则生成控制器在任务结束时，触发事件性任务，这些任务自动更新本地输出。换句话说，除了 IOT 指令，任务扫描也将触发事件。

❑	3. IOT 指令	在用户逻辑程序的一点使用 IOT 指令，在这里用户要触发事件性任务。 IOT 指令代替标记的 RPI，立即发送事件触发和标记数据。

使用者控制器的清单

对于下面的：		确保：
❑	1. 数据缓冲	如果用户希望控制器在数据改变时，不使用来自使用标记的数据，请使用使用标记副本。使用事件性任务复制数据，如下面显示：



❑	2. 任务优先级	配置事件性任务的优先级为最高。 如果周期型任务优先级较高，则事件性任务必须等到周期性任务完成后执行。
❑	3. 事件性任务的数量	限制事件性任务的数量 每个额外的任务减少其它任务的处理时间。 这可能会引起重叠。
❑	4. 自动输出过程	对于事件性任务，用户可以禁用自动输出过程 （默认）。 这样减少任务执行时间。 要校验上面决定，请参阅第 第 3-13 页 页上的图 3.1。

示例

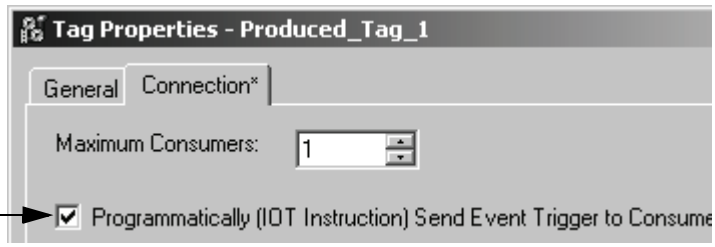
在零件移过生产线时，每个站点需要该站点零件的生产规范。要确保站点不执行以前的数据，事件性任务标记下一个零件新数据的到达。

生成者控制器

该控制器控制站点 24，并为下一个站点生成数据 （站点 25）。要标记新数据的传输，控制器使用下面的元素：

生成标记属性

通过 IOT 指令配置 Produced_Tag 以更新事件触发。

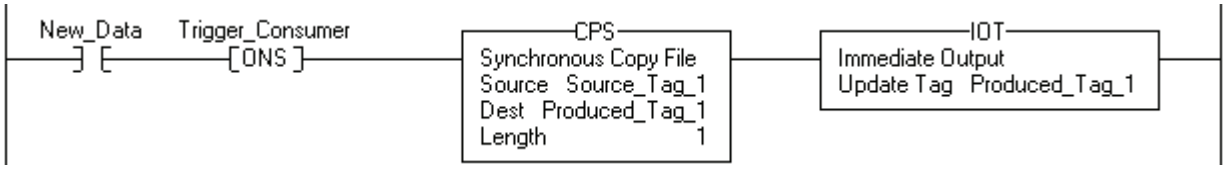


梯形逻辑程序

如果 New_Data 为 on，则进行一次扫描：

CPS 指令置位 Produced_Tag_1 为 Source_Tag_1。

IOT 指令更新 Produced_Tag_1，并发送此更新到使用控制器 （站点 25）。当使用控制器接收该更新时，触发该控制器中相关的事件性任务。



接下页

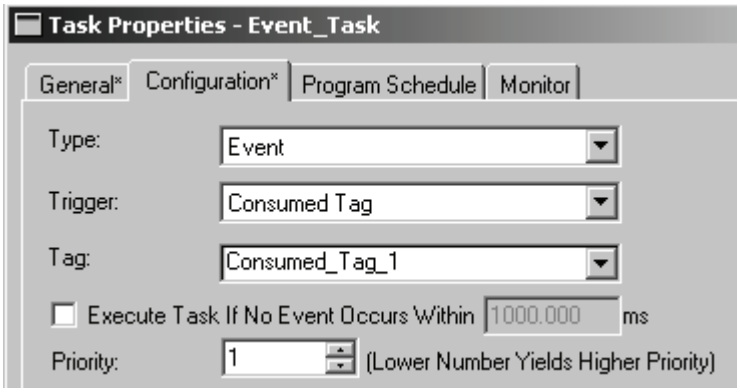
使用者控制器 站点 25 的控制器使用站点 24 生成的数据。要确定新数据何时到达，控制器使用事件性任务。

事件性任务属性

让事件触发任务。

让一个使用标记触发任务。

让该使用标记触发任务。

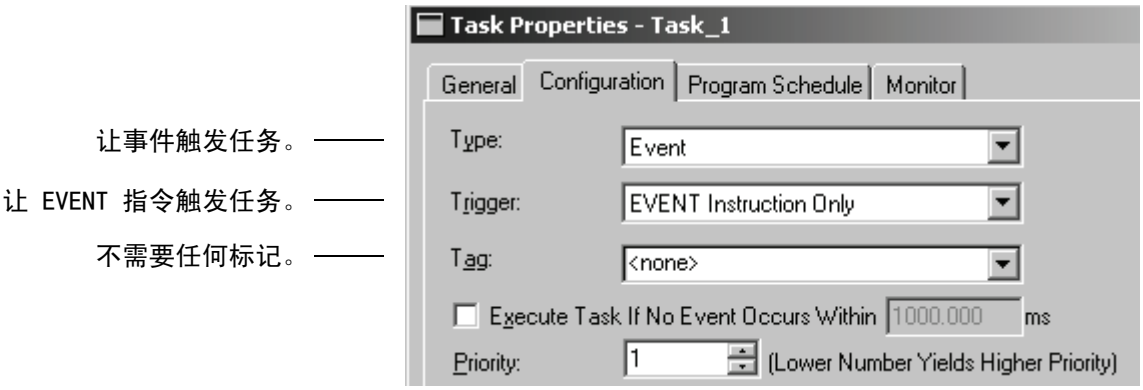


事件性任务中的梯形图

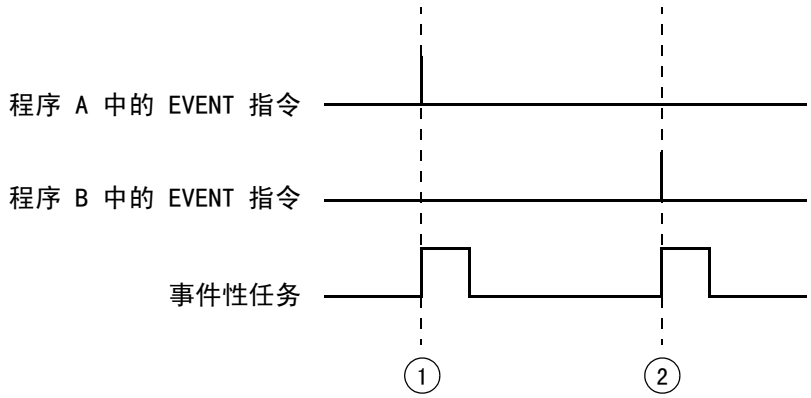
执行事件性任务时，CPS 指令置位 Destination_Tag_1 为 Consumed_Tag_1 （来自生成控制器的值）。该控制器中现有的逻辑程序使用来自 Destination_Tag_1 的值。

使用 EVENT 指令触发

要基于用户逻辑程序的条件触发事件性任务，使用仅由 EVENT 指令触发。



仅由 EVENT 指令触发需要用户使用触发事件性任务指令 (EVENT) 触发任务。用户可以在项目的多点使用 EVENT 指令。每次执行指令时，触发特定事件性任务。



说明：	
1	程序 A 执行 EVENT 指令。 EVENT 指令指定的事件性任务仅执行一次。
2	程序 B 执行 EVENT 指令。 EVENT 指令指定的事件性任务仅执行一次。

编程确定 EVENT 指令是否触发任务

要确定 EVENT 指令是否触发事件性任务，使用获取系统值指令 (GSV) 监视任务的状态属性。

表 3.3 TASK 对象的状态属性

属性：	数据类型：	指令：	说明：	
状态	DINT	GSV SSV	提供任务的状态信息。一旦控制器置位后，用户必须手动清零该位以确定是否发生其他类型故障。	
			确定下面的状态：	**检查下列位：**
			EVENT 指令触发任务 （仅事件性任务）。	0
			超时触发任务 （仅事件性任务）。	1
			该任务出现重叠。	2

一旦置位状态属性后，控制器不再清零该位。

- 要使用一位记录新的状态信息，用户必须手动清零该位。
- 使用设置系统值指令 (SSV) 设置不同属性值。

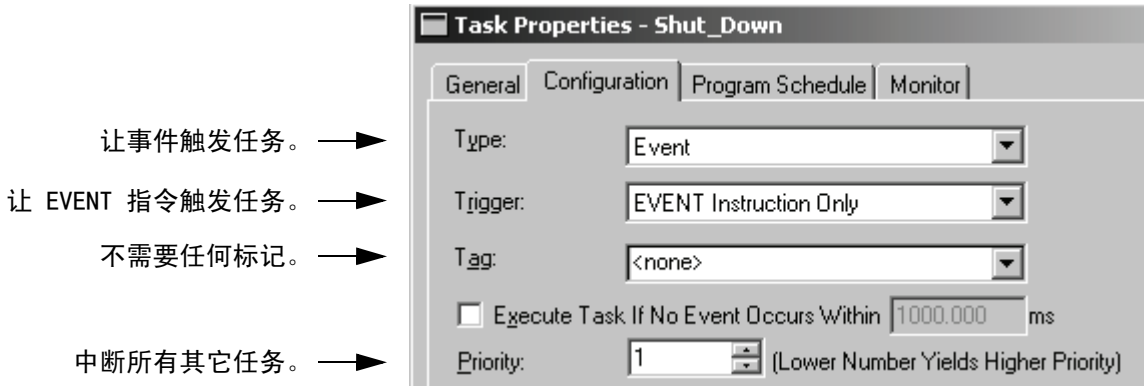
EVENT 指令任务清单

对于下面的：		确保：
❑	1. EVENT 指令	在用户逻辑程序中用户希望触发事件性任务的每点使用触发事件性任务指令 (EVNT)。
❑	2. 任务优先级	配置事件性任务的优先级为最高。 如果周期型任务优先级较高，则事件性任务必须等到周期性任务完成后执行。
❑	3. 事件性任务的数量	限制事件性任务的数量 每个额外的任务减少其它任务的处理时间。 这可能会引起重叠。
❑	4. 自动输出过程	对于事件性任务，用户可以禁用自动输出过程 （默认）。 这样减少任务执行时间。 要校验上面决定，请参阅第 第 3-13 页 页上的图 3.1。

示例

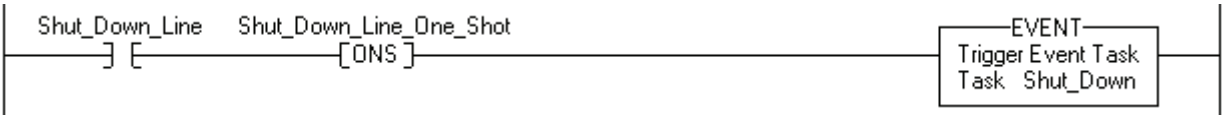
控制器使用多个程序，除了一个公用的关闭程序。每个程序使用名为 Shut_Down_Line 的程序范围标记，如果程序检测到需要关闭的条件该标记打开。

事件性任务属性



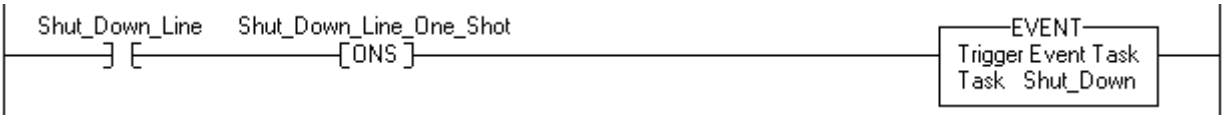
Program_A 中的梯形图

如果 Shut_Down_Line 为 on （需要关闭条件），则
执行一次 Shut_Down 任务



Program_B 中的梯形图

如果 Shut_Down_Line 为 on （需要关闭条件），则
执行一次 Shut_Down 任务

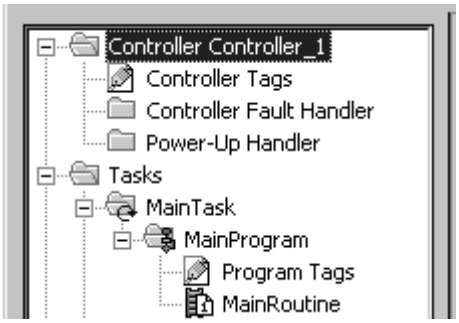


创建任务

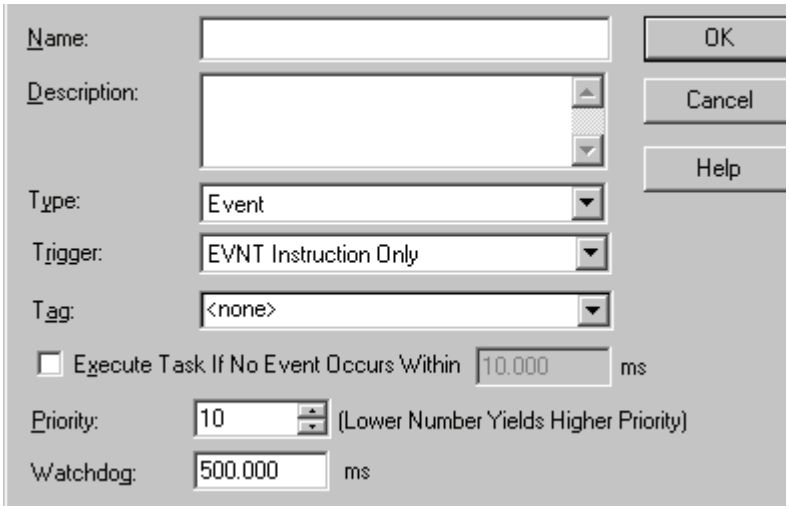
创建事件性任务

要创建事件性任务，执行下面操作：

1. 在控制器管理器中，右键单击任务文件夹并选择 Properties（属性）。



2. 为该任务输入**名称**。为该任务类型选择事件，为任务选择触发，为触发数据选择标记，输入任务的 **优先级**和**监视** 时间。



3. 单击 OK

创建周期性任务

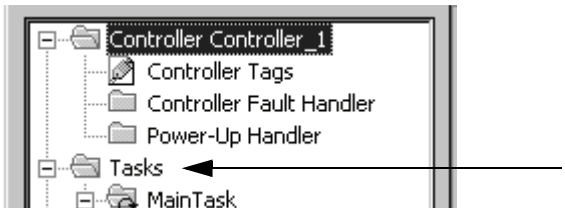
周期性任务以特定速率执行功能。

重要

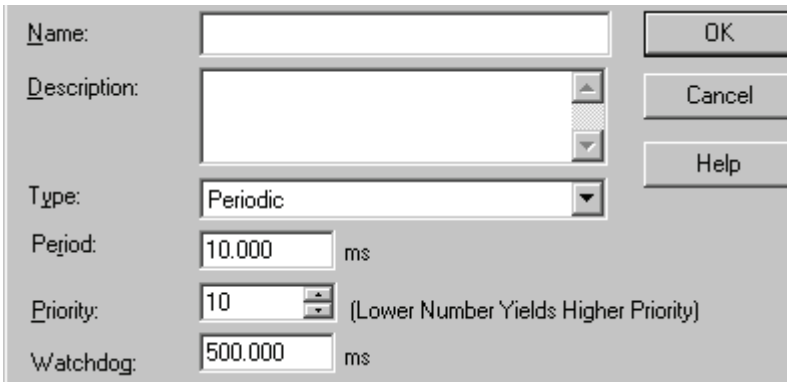
保证时间周期比执行任务的所有程序的时间长。

- 如果控制器检测到已经操作的任务发生周期性任务触发，出现次故障（重叠）。
- 其他任务的优先级和执行时间也会引起重叠。

1. 在控制器管理器中，右键单击任务文件夹并选择 New Task （新建任务）。



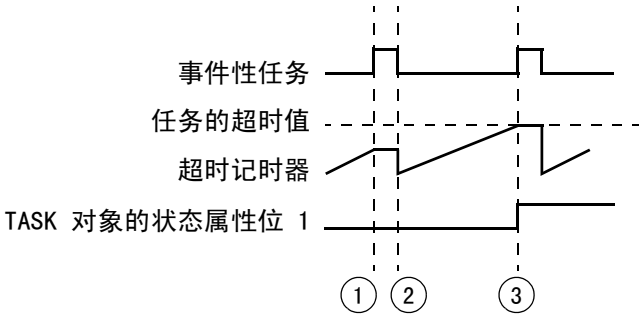
2. 输入任务**名称**，为任务类型选择周期性 （默认）输入用户希望执行任务的间期，输入任务**优先级**，并输入任务**监视**时间。



3. 单击 OK

定义事件性任务的超时值

如果用户希望在某个时间内的触发失败时自动执行事件性任务，设置任务的超时值。事件性任务完成后，超时记时器开始增加。如果记时器在事件性任务触发前达到预设值，则自动执行事件。

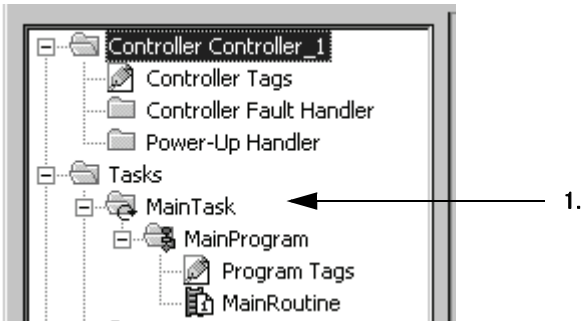


说明：	
1	执行事件性任务。 停止增加超时时间。
2	完成事件性任务。 重置并开始增加超时计数器。
3	超时计数器达到超时值。 自动执行事件性任务。 打开 TASK 对象状态属性中的位 1。

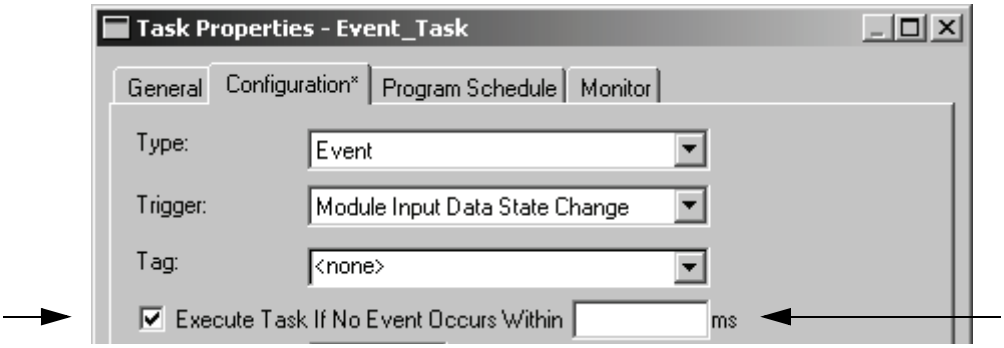
设置事件性任务的超时值

设置事件性任务的超时值，执行下列操作：

1. 在控制器管理器中，右键单击事件性任务并选择 Properties（属性）。



2. 选择 Configuration （配置）选项卡。



3. 选中 Execute Task If No Event Occurs Within （如果没有事件则执行任务）复选框，并以毫秒为单位输入超时值。

4. 单击 OK

编程配置超时值

要编程配置超时值，使用获取系统值指令 (GSV) 访问任务属性。

表 3.4 TASK 对象的状态属性

属性：	数据类型：	指令：	说明：	
Rate	DINT	GSV	**如果任务类型为：**	**则更新率属性指定下列值：**
		SSV	周期性	任务周期。以微秒计时。
			事件	任务的超时值。以微秒计时。
EnableTimeOut	DINT	GSV	启用或禁用事件性任务的超时功能。	
		SSV	**目的：**	**设置属性值为：**
			禁用超时功能	0 （默认）
			启用超时功能	1 （或任何非零值）

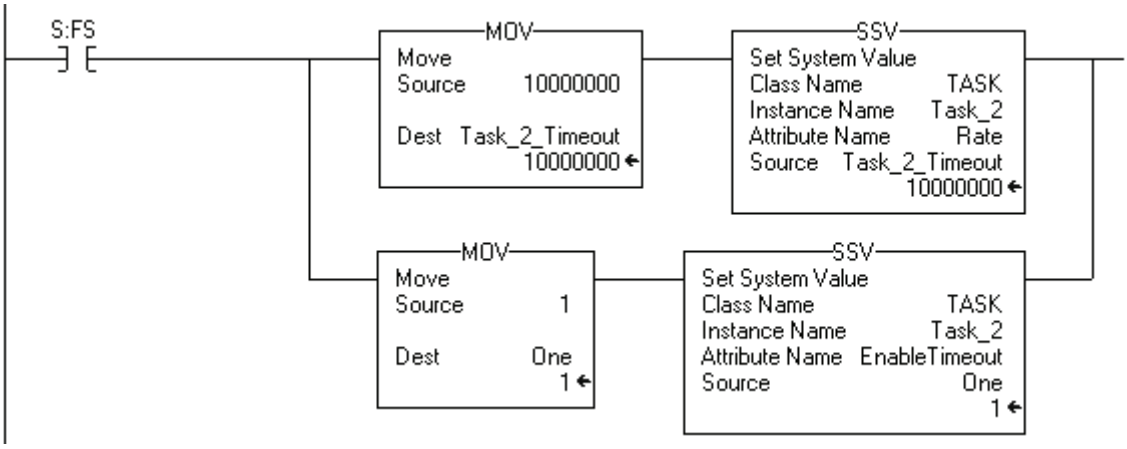
示例

编程配置超时值

始终保证定义并启用事件性任务的超时值，逻辑程序在控制器进入运行模式时配置超时。

如果 S:FS 为 1 （首次扫描），则设置 Task_2 的超时值，并启用超时功能：

1. 第一条 MOV 指令设置 Task_2_Timeout 为 10000000 µ（DINT 值）。接着 GSV 指令设置 Task_2 的 Rate 属性为 Task_2_Timeout。这配置任务超时值。
2. 第二个 MOV 指令置位 One 为 1 （DINT 值）。接着 GSV 指令置位 Task_2 的 EnableTimeout 属性为 One。这启用任务超时功能。



编程确定是否发生超时

要确定是否由于超时执行 EVENT 指令，使用获取系统值指令 (GSV) 监视任务的 Status 属性。

表 3.5 TASK 对象的状态属性

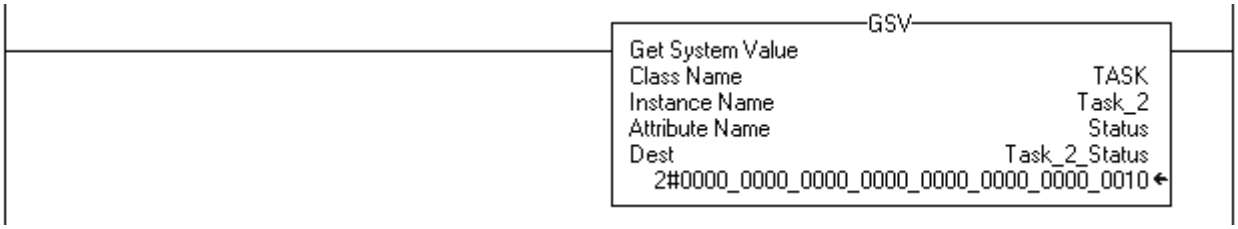
属性：	数据类型：	指令：	说明：	
Status	DINT	GSV SSV	提供任务的状态信息。 一旦控制器置位后，用户必须手动清零该位以确定是否发生其他类型故障。	
			确定下面的状态：	**检查下列位：**
			EVENT 指令触发任务 （仅事件性任务）。	0
			超时触发任务 （仅事件性任务）。	1
			该任务出现重叠。	2

示例

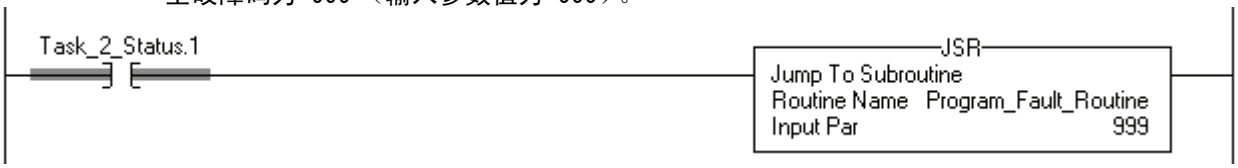
定义事件性任务的超时值

如果事件性任务发生超时，则与触发设备的通信可能会失败。这需要关闭过程。要关闭控制器，事件性任务调用程序的故障例程，并提供用户定义的故障码 （在本例中为 999）。

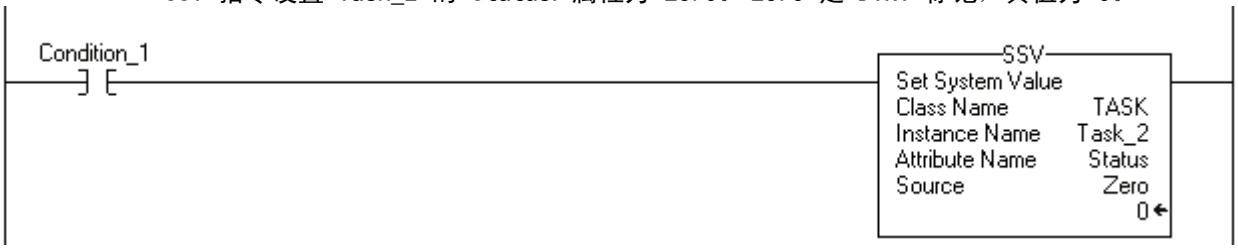
1. GSV 指令设置 Task_2_Status 为 Task_2 的 Status 属性 （DINT 值）。



2. 如果 Task_2_Status.1 为 1，则发生超时，因此关闭控制器并置位主故障码为 999。
   JSR 指令调用程序的故障例程。这产生主故障。
   主故障码为 999 （输入参数值为 999）。



3. 如果 Condition_1 为真，则清零 Task_2 状态属性位：
   SSV 指令设置 Task_2 的 Status. 属性为 Zero。 Zero 是 DINT 标记，其值为 0。



关于关闭控制器的更多信息，请参阅第 第 15-11 页 页上的 “创建用户定义的主故障”。

调整系统开销处理时间片

Logix5000 控制器以特定的速率 （确定性）或有可用处理时间来服务通信 （非确定性）时与其他设备通信 （I/O 模块、控制器、HMI 终端等）。

通讯类型：	是：
更新 I/O 数据 （不包括块传送）	确定性通信
生成或使用标记	
与编程设备通信 （如 RSLogix 5000 软件）	非确定性通信
与 HMI 设备通信	
执行通信指令 (MSG)，包括块传送	
从其他控制器响应消息	
同步冗余系统的从控制器	
重新建立并监视 I/O 连接 （如，带电卸除和插入；这不包括逻辑程序执行过程中的正常 I/O 更新。	
通过 ControlLogix 背板从控制器串口转发到其他 ControlLogix 设备的通信	

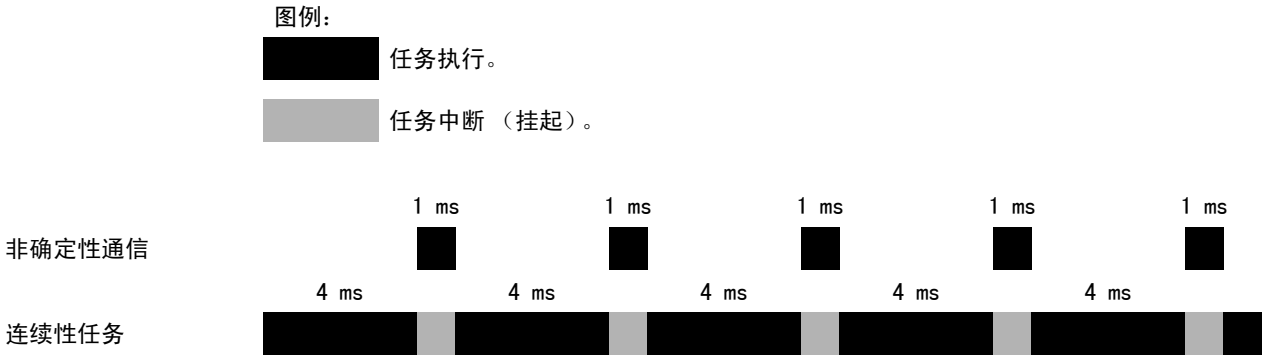
非确定性通信是用户不使用项目的 I/O 配置文件夹配置的通信。

- **系统开销时间片** 用来指定控制器专用于非确定性通信的时间百分率 （不包括周期性或事件性任务时间）。
- 控制器每次执行非确定性通信花费的时间约为 1 ms，然后继续连续性任务。

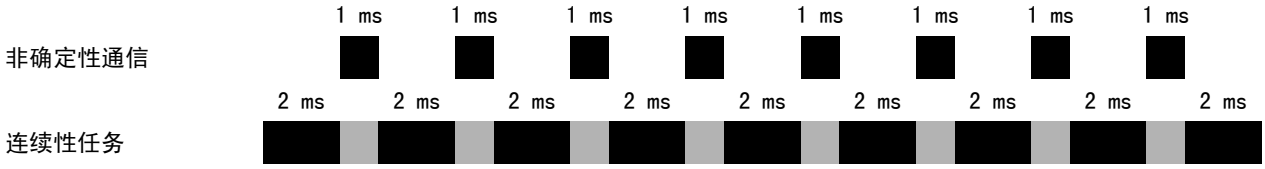
下表显示在不同系统开销处理时间片上连续性任务和非确定性通信之间的比率：

按照下列时间片：	连续性任务运行：	非确定性通信发生持续：
10%	9 ms	1 ms
20%	4 ms	1 ms
33%	2 ms	1 ms
50%	1 ms	1 ms

如果系统开销处理时间片为 20 %，非确定性通信每 4 ms 中断连续性任务并执行 1 ms。

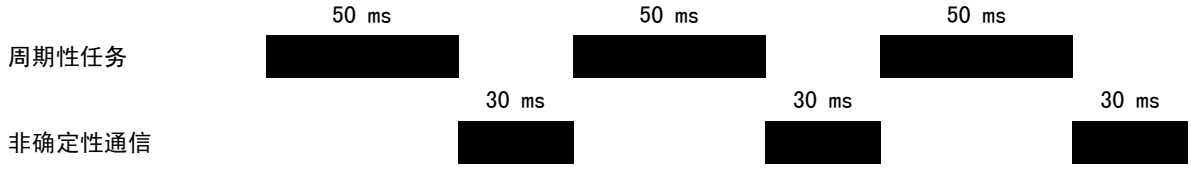


系统开销处理时间片增加到 33 %，非确定性通信每 2 ms 中断连续性任务并执行 1 ms。



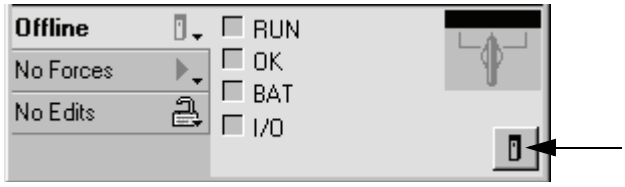
如果控制器只包含周期性任务，系统开销处理时间片将没有影响。非确定性通信会在周期性任务不运行的时间运行。

例如：如果用户花费 50 毫秒执行任务，配置其更新率为 80 毫秒，控制器每 80 毫秒非确定性通信占 30 毫秒。

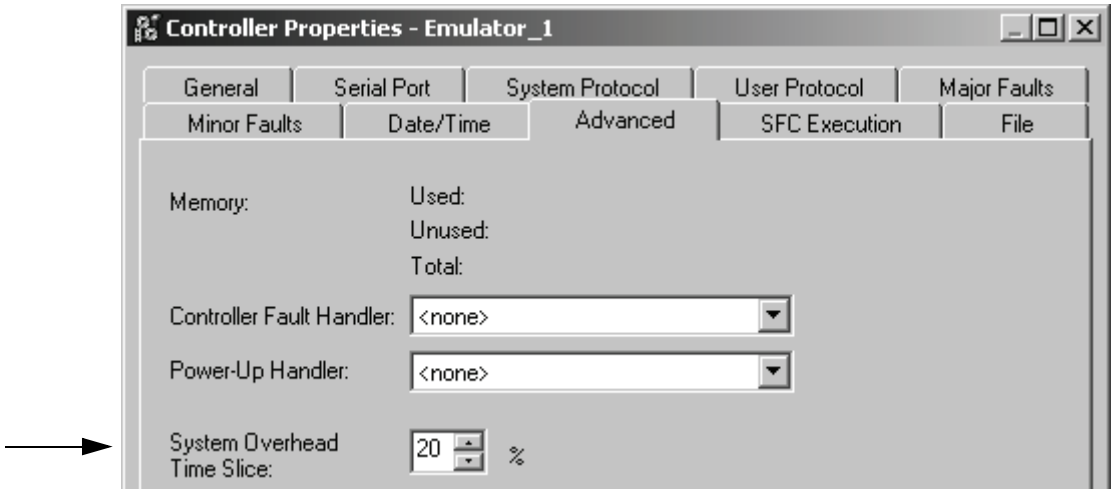


调整系统开销处理时间片

1. 在 Online （在线）工具栏，单击控制器属性按钮。



2. 选择 Advanced （高级）选项卡，输入系统开销处理时间片。



3. 单击 OK

调整监视时间

每个任务都包含一个监视计时器，指定触发一个主故障前任务运行的时间。

注意

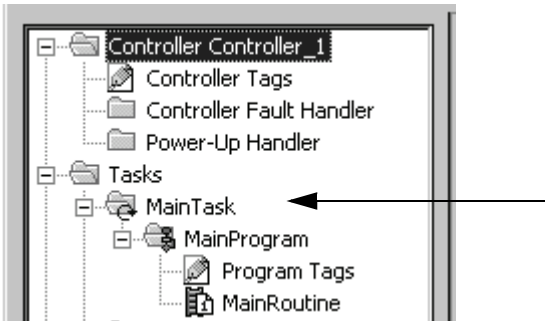
如果监视计时器达到配置预设值，则发生主故障。依据控制器故障处理程序，可能需要关闭控制器。

- 监视计时器范围从 1 毫秒到 2,000,000 毫秒 （2000 秒）。默认值为 500 毫秒。
- 监视计时器在任务启动时开始计时，任务所有程序执行后停止计时。
- 如果执行任务的时间比监视时间长，发生主故障。时间包括其它任务引起的中断时间。
- 如果任务执行时再次触发任务 （任务重叠），则也发生监视超时故障 （主故障）。如果高优先级任务中断低优先级任务，则会延迟低优先级任务的完成。
- 用户可以使用控制器故障处理程序清零监视故障。如果在同一个逻辑程序扫描过程中，再次发生相同的监视故障，则不管控制器故障处理程序是否清零监视故障，控制器都会进入故障模式。

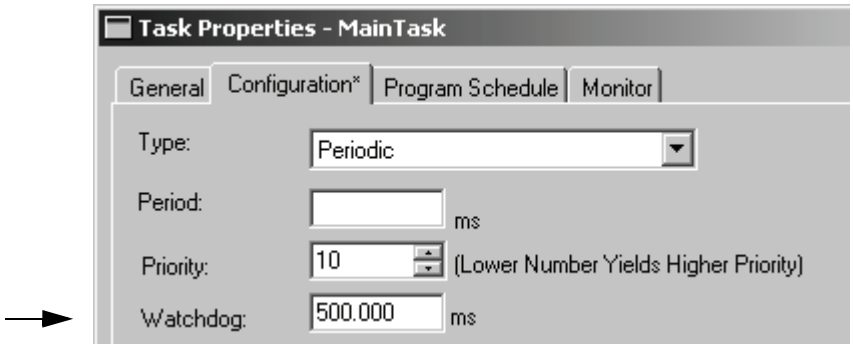
调整任务监视计时器

要改变任务监视计时器，使用任务属性对话框。

1. 在控制器管理器中，右键单击任务并选择 Properties （属性）。



2. 选择 Configuration（配置）选项卡，然后选择任务 watchdog timeout （监视超时），以毫秒为单位。



3. 单击 OK

注释：

设计顺序流程图

何时使用本章

使用本章为加工过程或控制系统设计**流程图 (SFC)**。SFC 类似于加工过程的流程。它定义系统处理的步骤或状态。使用 SFC：

- 针对您的系统，组织各种特定的功能
- 将系统作为一系列步骤和转变进行编程和控制

使用 SFC 指定进程可以获得以下优势：

- 因为 SFC 是加工过程的图形表示，所以比文本版本更容易组织和阅读。此外，RSLogix 5000 软件允许您：
  - 添加步骤的说明或捕获重要信息供以后使用
  - 打印 SFC 以与其他人共享信息
- 因为 Logix5000 控制器支持 SFC，所以无需再次输入指定操作步骤。指定操作步骤时，就在对系统进行编程。

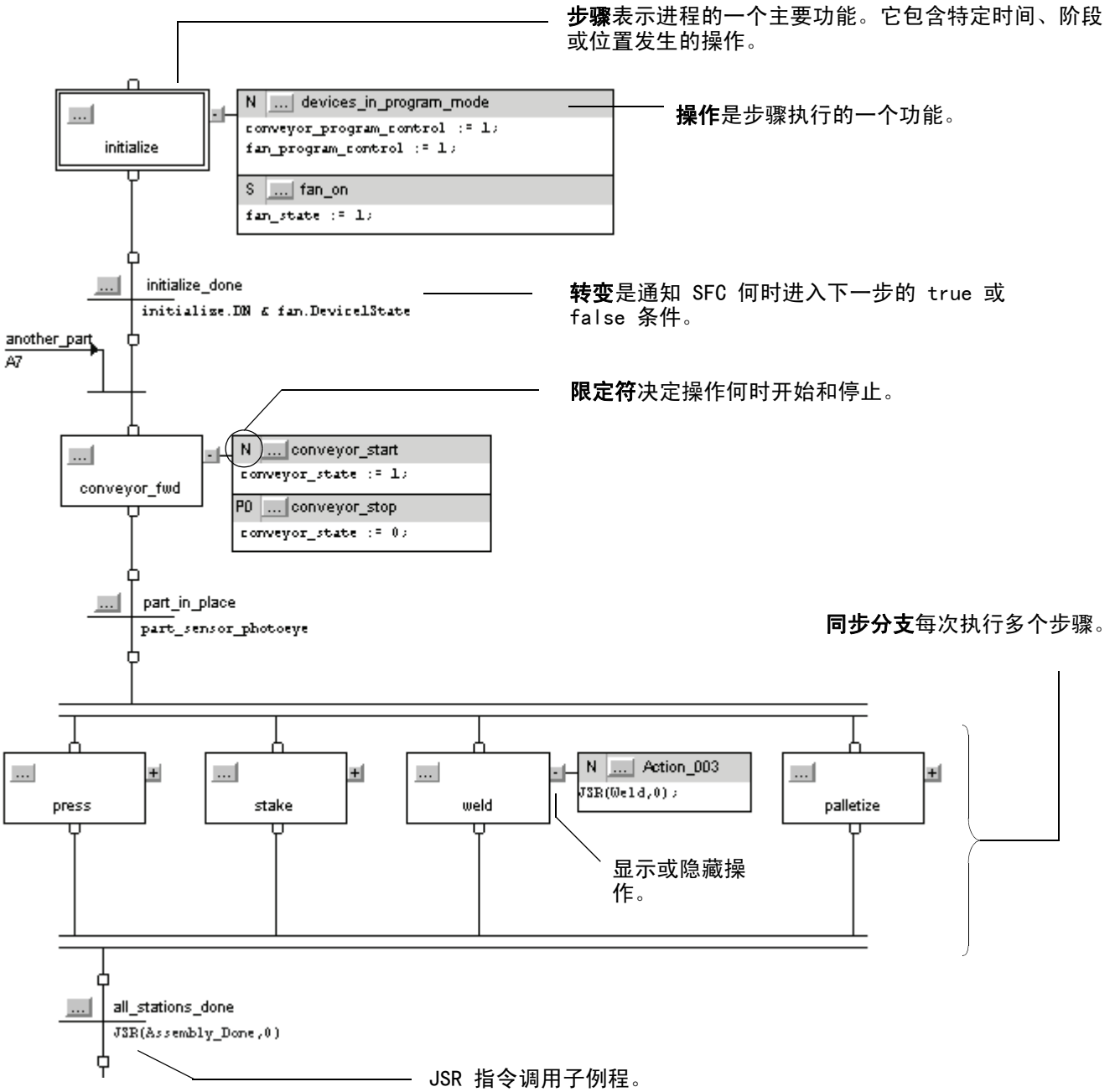
使用 SFC 编程进程可以获得以下优势：

- 通过图形化的方式将加工过程划分为主要的逻辑块 （步骤）
- 更快速重复执行各个逻辑片断
- 更简单的屏幕显示
- 缩短设计和调试程序的时间
- 更快速简单的排除故障
- 直接访问机器出现故障的逻辑点
- 方便升级和增强

什么是顺序流程图？

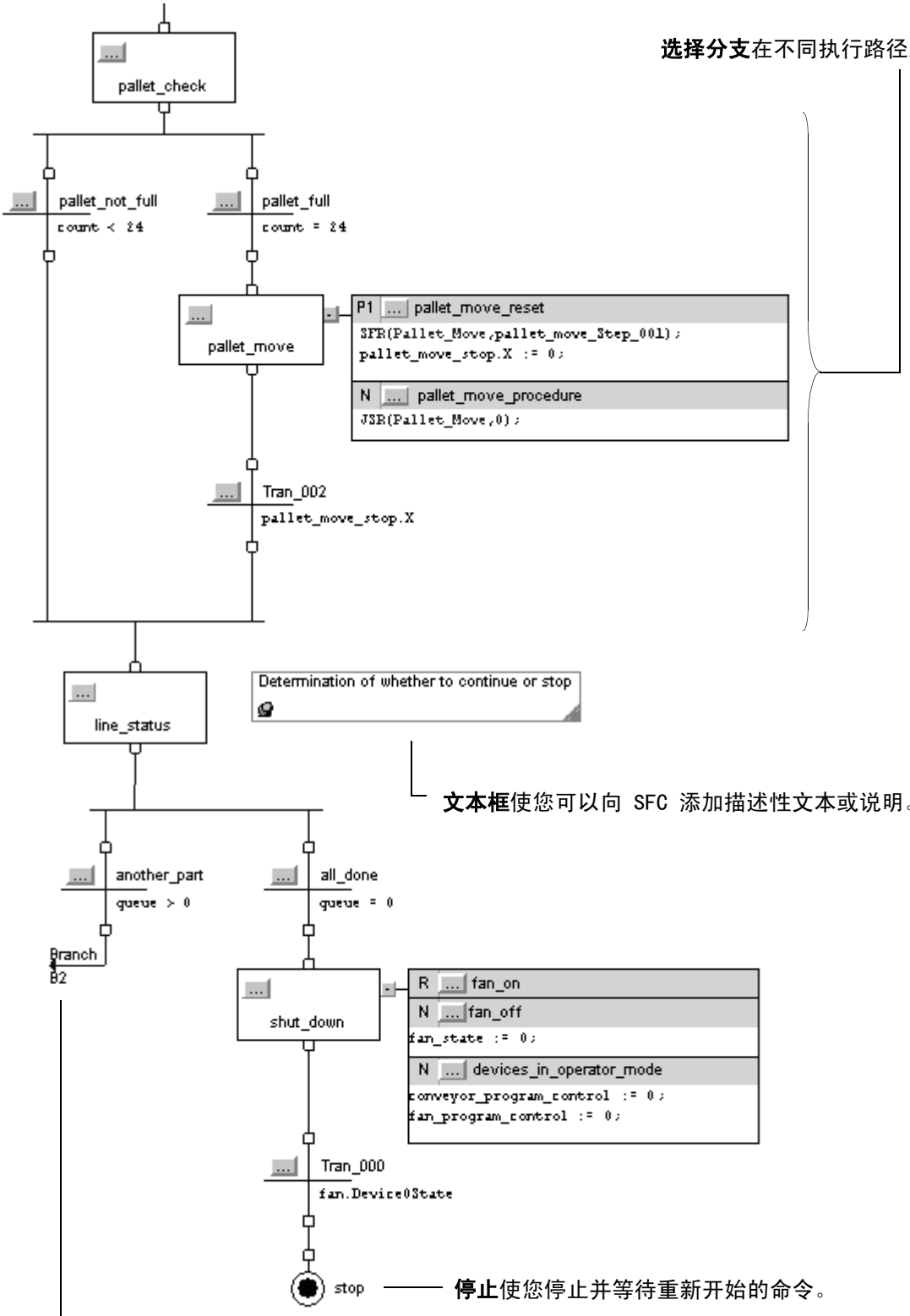
流程图（SFC）类似于过程图。它使用步骤和转变执行特定操作。此示例显示 SFC 的元素：

图 4.1 SFC 示例



（下页继续）

图 4.2 SFC 示例 （继续上页）



设计顺序流程图：

定义任务

设计 SFC 的第一步是将设备的配置和调节与设备的命令分开。Logix5000 控制器使您可以将项目划分到一个**持续任务**和多个**定期任务**以及**事件任务**中。

1. 组织项目：

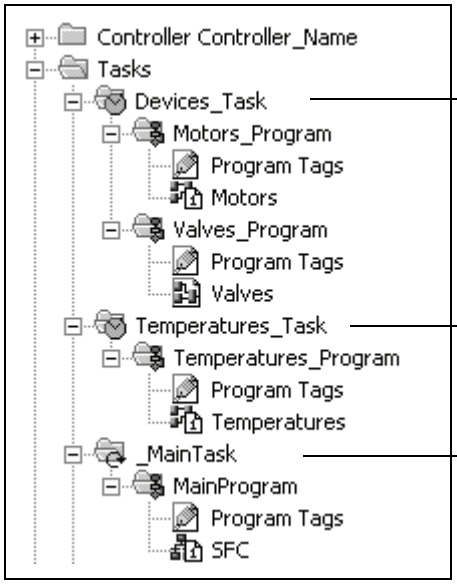
这些功能：	转至：
配置和调节设备	周期性任务
命令设备至特定状态	持续任务中的 SFC
进程的执行顺序	

2. 对于周期性任务中的功能，请根据类似的更新速率分组这些功能。为每个更新速率创建一个周期性任务。

   例如，2 状态设备可能需要比 PID 回路更快的更新。为每个使用不同的周期性任务。

在此示例中，项目使用两个周期性任务调节马达、阀门和温度回路。SFC 控制进程。

示例 定义任务



此任务 （定期）使用功能方框图关闭或打开马达以及关闭或打开阀门。MainTask 中的 SFC 命令每个设备的状态。功能方框图设置并维护该状态。

此任务 （定期）使用功能方框图配置和调节温度回路。MainTask 中的 SFC 命令温度。功能方框图设置并维护这些温度。

此任务 （持续）执行顺序流程图 (SFC)。SFC 命令每个设备或温度回路的特定状态或温度。

选择如何执行 SFC

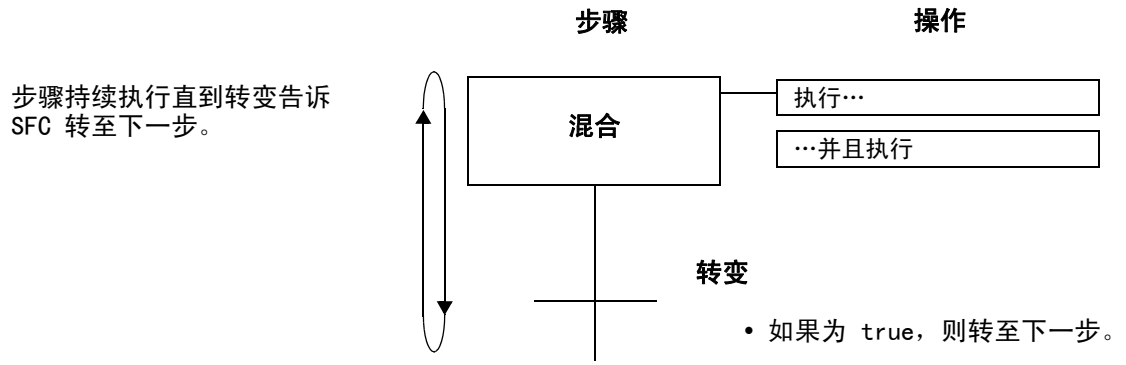
要执行 SFC，请将其配置为程序的主例程或作为子例程调用。

如果：	则：
SFC 是程序中的唯一例程。	配置 SFC 作为程序的主例程。
SFC 调用程序的所有 其他例程。	
程序要求其他例程独立于 SFC 执行。	1. 配置另一个例程作为程序的主例程。
SFC 使用布尔值操作。	2. 使用主例程将 SFC 作为子例程调用。

如果 SFC 使用布尔值操作，则其他逻辑必须独立于 SFC 运行并监视 SFC 的状态位。

定义进程的步骤

步骤表示进程的主要功能。它包含特定时间、阶段或位置发生的操作。

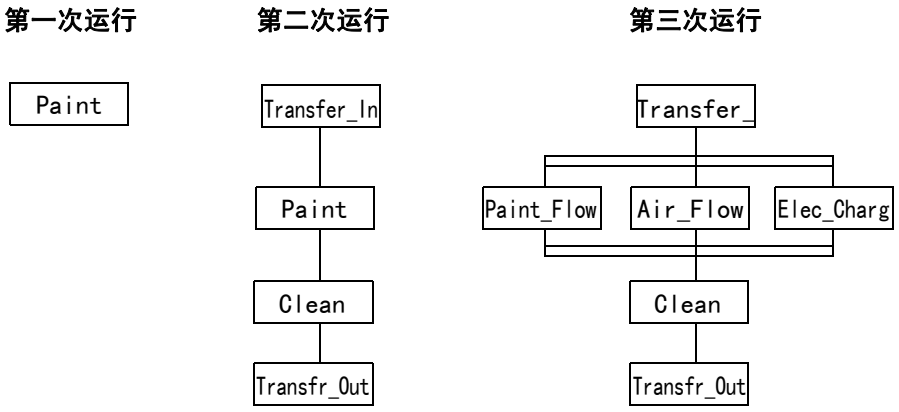


转变结束步骤。转变定义必须发生或更改以转至下一步的物理情况。

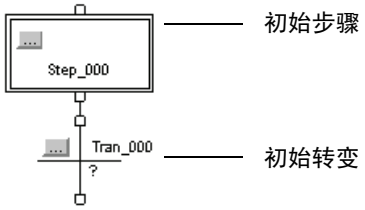
步骤准则

定义进程步骤时请遵循以下准则：

- 以大步骤开始，在多次运行中优化步骤。



- 首次打开 SFC 例程时，它包含一个初始步骤和转变。使用此步骤初始化进程。



控制器执行初始步骤：

– 项目下载后控制器进入运行模式时。

– 如果配置了 SFC，控制器转变为运行模式和启动时

– 包含图的例程联机修改和控制器转变为或转变自测试模式时

- 要识别步骤，请在系统中寻找物理改变，例如就位的新部件，达到的温度，达到的预设时间，或者发生的配方选择。步骤是该更改前发生的操作。

- 当步骤有重大增量时停止。例如：

此步骤组成：	为：
produce_solution	可能过大
set_mode, close_outlet, set_temperature, open_inlet_a, close_inlet_a, set_timer, reset_temperature, open_outlet, reset_mode	可能过小
preset_tank, add_ingredient_a, cook, drain	可能刚好

SFC_STEP 结构

每个步骤使用一个标记提供关于步骤的信息。通过 Step Properties （步骤属性）对话框或 Tags （标记）窗口的 Monitor Tags （监视标记）选项卡访问此信息：

如果希望：	则选中或设置此成员：	数据类型：	详细信息：
决定步骤激活的时间长度（毫秒）	T	DINT	当步骤激活时，Timer (T) 值重置，然后开始以毫秒计数。计时器继续计数直到步骤不激活，不考虑 Preset (PRE) 值。
当步骤在特定时长 （毫秒）激活时标记	PRE	DINT	将时间输入 Preset (PRE) 成员。当 Timer (T) 达到 Preset 值时，Done (DN) 位变为 on 并保持直到步骤再次激活。 作为一个选项，输入在运行时计算时间的数值表达式。
	DN	BOOL	当 Timer (T) 达到 Preset (PRE) 值时，Done (DN) 位变为 on 并保持直到步骤再次激活。
如果步骤执行时间不够长则标记	LimitLow	DINT	将时间输入 LimitLow 成员 （毫秒）。 • 如果 Timer (T) 达到 LimitLow 值前步骤转为不激活，则 AlarmLow 位将变为 on。 • AlarmLow 位保持 on 直到您重置。 • 要使用此警告功能，请打开 （选中） AlarmEnable (AlarmEn) 位。 作为一个选项，输入在运行时计算时间的数值表达式。
	AlarmEn	BOOL	要使用警告位，请打开 （选中）AlarmEnable (AlarmEn) 位。
	AlarmLow	BOOL	如果 Timer (T) 达到 LimitLow 值前步骤转为不激活，则 AlarmLow 位将变为 on。 • 该位保持 on 直到您重置。 • 要使用此警告功能，请打开 （选中） AlarmEnable (AlarmEn) 位。

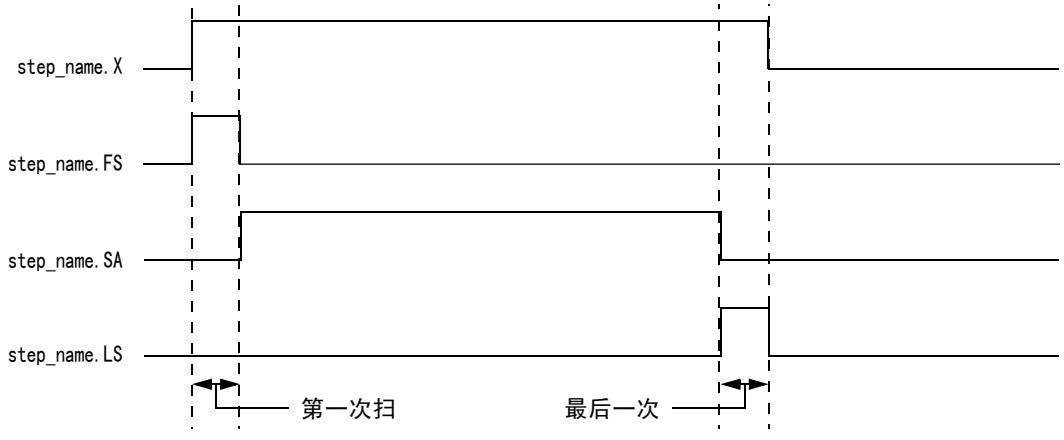
如果希望：	则选中或设置此成员：	数据类型：	详细信息：
如果步骤执行时间过长则标记	LimitHigh	DINT	将时间输入 LimitHigh 成员 （毫秒）。 • 如果 Timer (T) 达到 LimitHigh 值，则 AlarmHigh 位变为 on。 • AlarmHigh 位保持 on 直到您重置。 • 要使用此警告功能，请打开 （选中） AlarmEnable (AlarmEn) 位。 作为一个选项，输入在运行时计算时间的数值表达式。
	AlarmEn	BOOL	要使用警告位，请打开 （选中）AlarmEnable (AlarmEn) 位。
	AlarmHigh	BOOL	如果 Timer (T) 达到 LimitHigh 值，则 AlarmHigh 位变为 on。 • 该位保持 on 直到您重置。 • 要使用此警告功能，请打开 （选中） AlarmEnable (AlarmEn) 位。
在步骤激活期间进行操作（包括第一次和最后一次扫描）	X	BOOL	步骤激活 （执行）的全部时间内 X 位为 on。 通常建议使用带有 N Non-Stored （不存储） 限定符的操作来完成。
在步骤变为激活时执行一次	FS(1)	BOOL	步骤的第一次扫描期间 FS 位为 on。 通常建议使用带有 P1 Pulse (Rising Edge)（上升边缘）限定符的操作来完成。
在步骤激活期间除 第一次和最后一次扫描外进行操作	SA	BOOL	在步骤激活期间除步骤的第一次和最后一次扫描外，SA 位为 on。
在步骤的最后一次扫描时执行一次	LS(1)	BOOL	步骤的最后一次扫描期间 LS 位为 on。 仅在执行下面的操作时使用此位：在 Controller Properties （控制器属性）对话框 SFC Execution （SFC 执行）选项卡上将 Last Scan of Active Step （激活步骤的最后一次扫描）设置为 Don’t Scan （不扫描）或 Programmatic reset （以编程方式重置）。 通常建议使用带有 P0 Pulse (Falling Edge)（下降边缘）限定符的操作来完成。
决定 SFC Reset (SFR) 指令的目标	重置	BOOL	SFC Reset (SFR) 指令将 SFC 重置为指令指定的步骤或停止。 • Reset 位指示 SFC 将开始再次执行的步骤或停止。 • SFC 执行后，Reset 位清除。
决定步骤在其任何执行期间激活的最大时间	TMax	DINT	将其用于诊断目的。仅当您在初始步骤选择 Restart Position of Restart （重新启动的重新启动位置）并且控制器更改模式或关闭并重新启动时控制器清除此值。

如果希望：	则选中或设置此成员：	数据类型：	详细信息：
决定 Timer (T) 值是否翻转为负值	OV	BOOL	将其用于诊断目的。
决定步骤激活的次数	Count	DINT	这不是步骤的扫描计数。 • 每次步骤激活时计数递增。 • 仅在步骤变为不激活然后再次激活时它递增。 • 仅当您配置 SFC 在初始步骤重新启动时计数重置。该配置情况下，它将在控制器从程序模式更改为运行模式时重置。
对此步骤的各个状态位使用一个标记	状态	DINT	

对于此成员	使用此位：
重置	22
AlarmHigh	23
AlarmLow	24
AlarmEn	25
OV	26
DN	27
LS	28
SA	29
FS	30
X	31

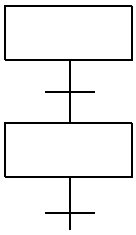
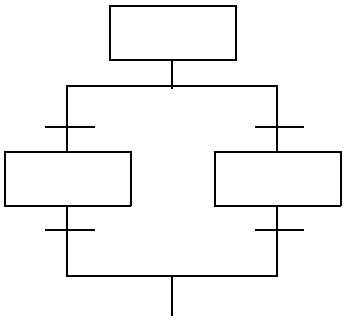
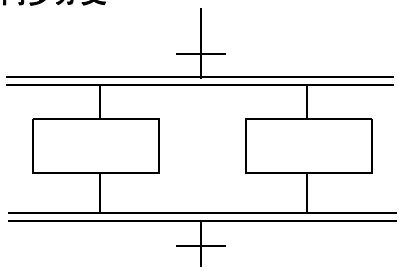
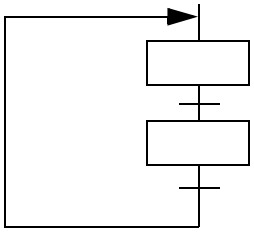
(1)FS 和 LS 位仅在步骤执行期间激活。当步骤完成执行其操作中的代码后，FS 和 / 或 LS 位重置。如果在项目其他步骤中 SFC 例程的代码外引用任一个位，则它们始终清零 (0)。

下图显示 X、FS、SA 和 LS 位的关系。

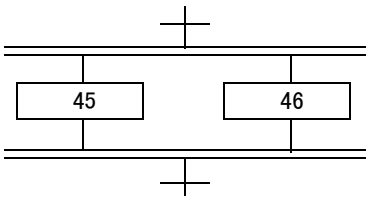
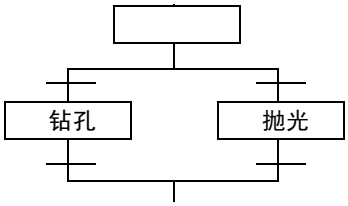
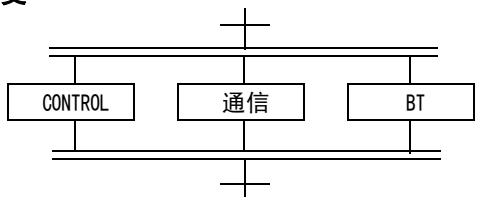
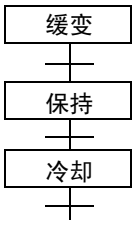
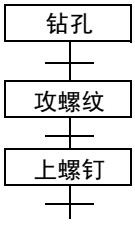
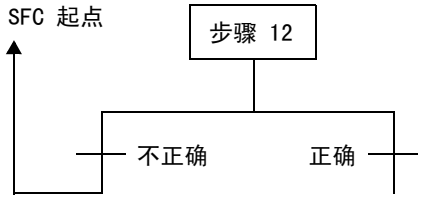


组织步骤

定义进程的步骤后，请将它们组织为顺序、同步分支、选择分支或回路。

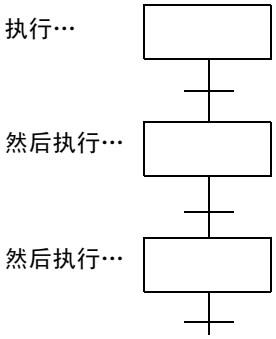
要：	使用此结构：	注意事项：
在顺序中执行 1 或更多步骤： • 一个步骤重复执行。 • 然后下一个步骤重复执行。	**顺序** 	SFC 在步骤末尾检查转变： • 如果为 true，则 SFC 跳至下一步。 • 如果为 false，则 SFC 重复该步骤。
• 根据逻辑条件选择替换步骤 • 根据逻辑条件执行步骤或跳过步骤	**选择分支** 	• 路径可以没有步骤而只有一个转变。这将使 SFC 跳过选择分支。 • 默认情况下，SFC 从左到右检查开始每个路径的转变。它采用第一个为 true 的路径。 • 如果没有转变为 true，则 SFC 重复上一步。 • RSLogix 5000 软件允许您更改 SFC 检查转变的顺序。
同时执行 2 个或更多步骤。继续 SFC 前所有路径必须完成	**同步分支** 	• 单个转变结束分支。 • SFC 在每个路径的最后一步至少执行一次后检查结束转变。如果转变为 false，则 SFC 重复上一步。
循环回上一步	**连线上一步** 	• 将线连接至要转到的步骤或同步分支。 • 不连线出、连线入或在同步分支间连线。

下面是一些不同情况下的 SFC 结构示例：

示例情况：	示例解决方案：
装配线的站点 45 和 46 同步处理部件。当两个站点都完成时，部件移至下一站。	**同步分支** 
根据生成代码，站点可能钻孔或抛光。	**选择分支** 
为简化编程，我希望分离通信并阻止其他控制逻辑的传输。所有操作同时发生。	**同步分支** 
在热处理区域，温度以特定速率升高，保持该温度特定时间，然后以特定速率冷却。	**顺序** 
在站点 12，机器钻孔、攻螺纹并对部件上螺钉。步骤依次进行。	**顺序** 
步骤 12 检查进程是否正确混合化学物。 • 如果正确，则继续剩下的步骤。 • 如果不正确，则转至 SFC 顶部并清除系统。	**连线** 

顺序

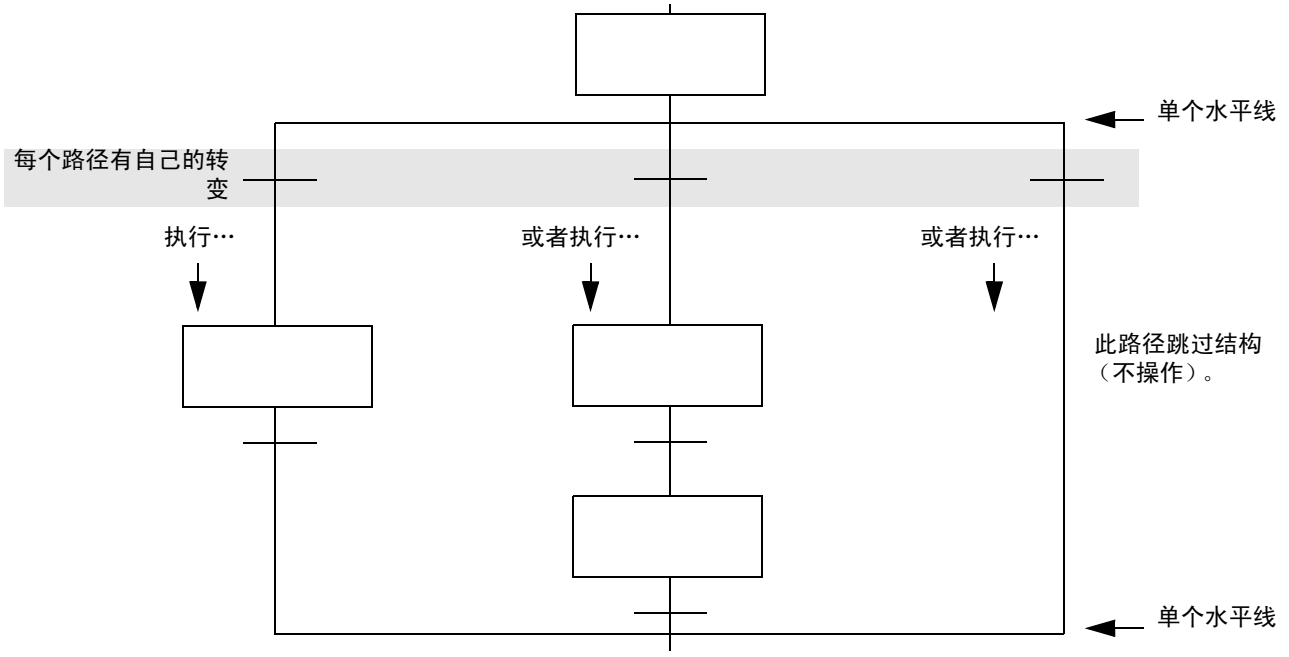
顺序是依次执行的一组步骤。



选择分支

选择分支表示一个路径 （步骤或一组步骤）与其他路径之间的选择（例如 OR 结构）。

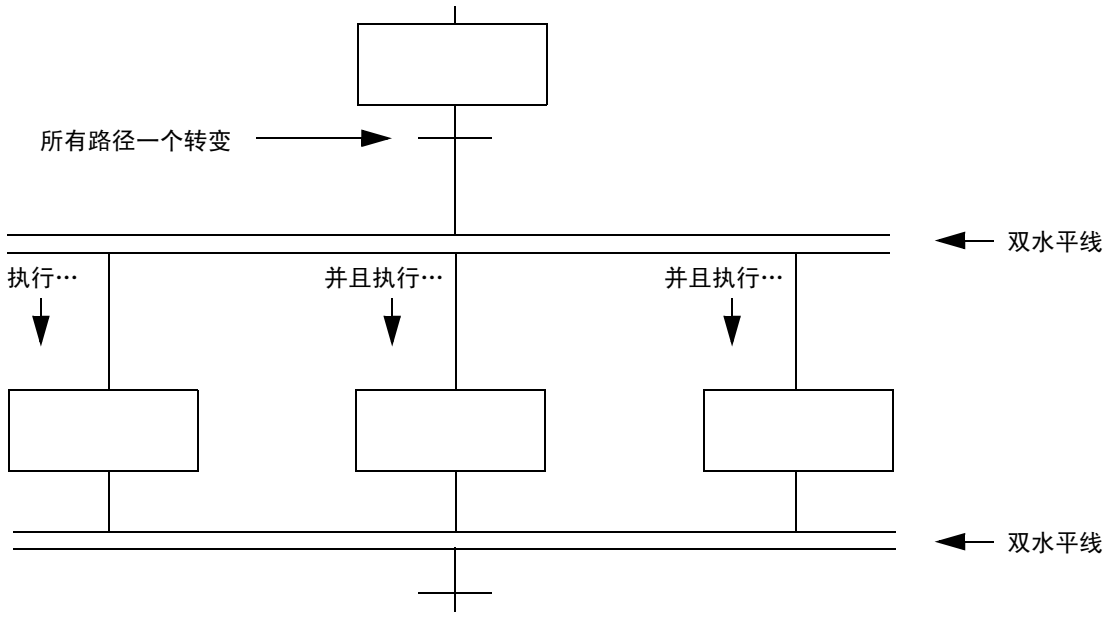
- 只有一个路径执行。
- 默认情况下 SFC 从左到右检查转变。
  - SFC 采用第一个 true 路径。
  - RSLogix 5000 软件允许您更改 SFC 检查转变的顺序 （请参阅第 5 章）。



同步分支

同步分支表示同时发生的路径 （步骤或一组步骤）（例如 AND 结构）。

- 所有路径都执行。
- 继续 SFC 前所有路径必须完成。
- SFC 在每个路径的最后一步至少执行一次后检查转变。

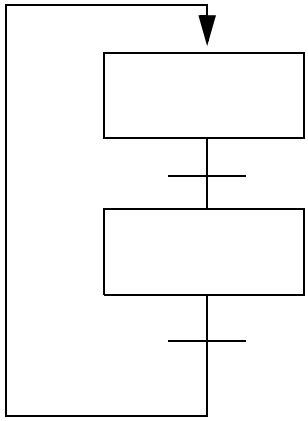


连线上一步

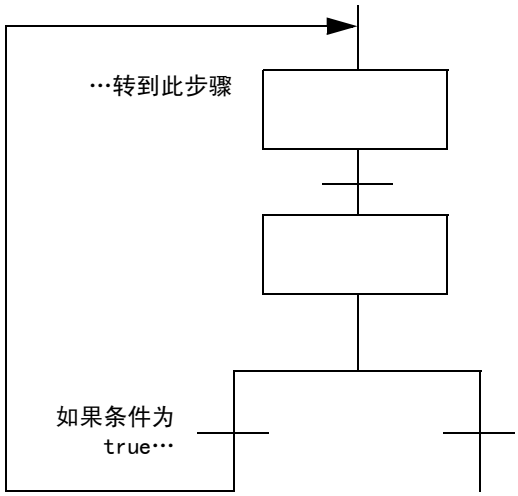
除了连接顺序的步骤、同步分支和选择分支，还可以选择将步骤与 SFC 前面的点连接。这使您可以：

- 向后循环和重复步骤
- 返回 SFC 的起点并重新开始

例如：



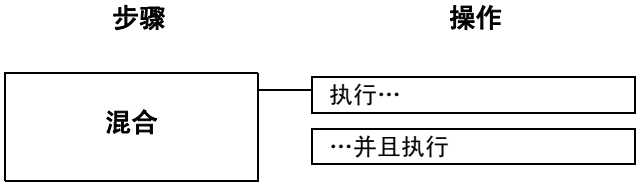
重复整个 SFC 的简单循环



返回以前步骤的选择分支的路径

添加每个步骤的操作

使用**操作**将步骤划分为步骤执行的多个功能，例如命令马达、设置阀门的值或者将一组设备置于特定模式下。



您希望如何使用操作？

有两种类型操作：

如果希望：	则：
直接执行 SFC 中的结构化文本	使用非布尔值操作
调用子例程	
使用自动重置选项在离开步骤时重置数据	
仅设置一个位并编程其他逻辑监视该位以决定何时执行。	使用布尔值操作

使用非布尔值操作

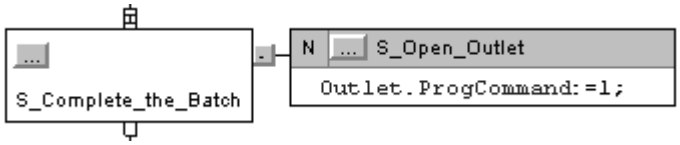
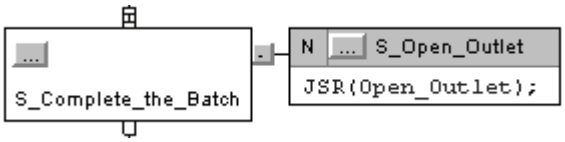
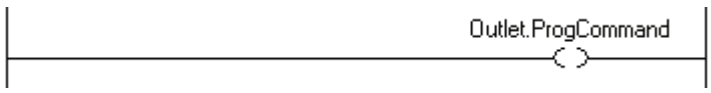
非布尔值操作包含操作的逻辑。它使用结构化文本执行分配和指令或调用子例程。

利用非布尔值操作，还可以选择在离开步骤前**后扫描** （自动重置）分配和指令：

- 后扫描期间，如果所有条件为 false，控制器执行分配和指令。
- 控制器后扫描嵌入结构化文本和操作调用的任何子例程。

要自动重置分配和指令，请参阅第 4-28 页的 “在步骤结尾关闭设备”。

要编程非布尔值操作，可以：

如果希望：	则：
• 在没有额外例程时执行逻辑 • 使用结构化文本分配、构造和指令	嵌入式结构化文本。 例如：  当 S_Complete_the_Batch 步骤激活时 S_Open_Outlet 操作执行。操作设置 Outlet.ProgCommand 标记等于 1，此设置打开出口阀门。
• 在多个步骤中重复使用逻辑 • 使用其他语言编程操作，例如梯形逻辑 • 嵌套 SFC	调用子例程。 例如：  当 S_Complete_the_Batch 步骤激活时 S_Open_Outlet 操作执行。 操作调用 Open_Outlet 例程。 **Open_Outlet 例程**  当 Open_Outlet 例程执行时，OTE 指令设置 Outlet.ProgCommand 标记等于 1，此设置打开出口阀门。

不能在同一 SFC 内重复使用非布尔值操作，重置存储的操作除外。每个 SFC 只允许一个特定非布尔值操作实例。

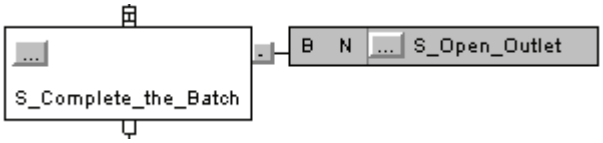
使用布尔值操作

布尔值操作不包含操作的逻辑。它简单设置其标记中的位（SFC_ACTION 结构）。要执行操作，其他逻辑必须监视该位并在该位为 on 时执行。

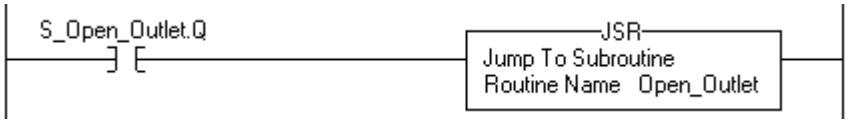
利用布尔值操作，可以手动重置操作关联的分配和指令。因为操作和执行操作间的逻辑没有链接，自动重置选项不影响布尔值操作。

例如：

示例



当 S_Complete_the_Batch 步骤激活时 S_Open_Outlet 操作执行。 当操作激活时，其 Q 位变为 on。



梯形逻辑例程监视 Q 位 (S_Open_Outlet.Q)。当 Q 位为 on 时，JSR 指令执行并打开阀门。

可以在同一 SFC 内重复使用多次布尔值操作。

SFC_ACTION 结构

每个操作 （非布尔值和布尔值）使用标记提供操作的信息。通过 Action Properties （操作属性）对话框或 Tags （标记）窗口的 Monitor Tags （监视标记）选项卡访问此信息。

如果希望：	则选中或设置此成员：	数据类型：	详细信息：	
决定操作何时激活	Q	BOOL	Q 位的状态根据操作是布尔值操作还是非布尔值操作：	
			如果操作是：	**则 Q 位为：**
			布尔值	在操作激活的整个时间内为 on (1)，包括操作的最后一次扫描
			非布尔值	在操作激活期间为 on (1) 但 在操作的最后一次扫描时为 off (0)
			要使用一个位决定操作何时激活，请使用 Q 位。	
	A	BOOL	操作激活的整个期间 A 位为 on。	
决定操作激活的时间长度（毫秒）	T	DINT	当操作激活时，Timer (T) 值重置，然后开始以毫秒计数。计时器继续计数直到操作不激活，不考虑 Preset (PRE) 值。	
使用以下基于时间的限定符之一：L、SL、D、DS、SD	PRE	DINT	输入 Preset (PRE) 成员的时间限制或延迟。Timer (T) 达到 Preset 值时操作开始或停止。 作为一个选项，输入在运行时计算时间的数值表达式。	
决定操作激活的次数	Count	DINT	这不是 操作的扫描计数。 • 每次操作激活时计数递增。 • 仅在操作变为不激活然后再次激活时它递增。 • 仅当您配置 SFC 在初始步骤重新启动时计数重置。 该配置情况下，它将在控制器从程序模式更改为运行模式时重置。	
对此操作的各个状态位使用一个标记	状态	DINT	**对于此成员**	**使用此位：**
			Q	30
			A	31

以伪代码说明每个操作

要组织操作的逻辑，首先以伪代码说明操作。如果您对伪代码不熟悉：

- 使用一系列描述应发生的情况的短语句。
- 使用以下类似术语或符号：如果、那么、否则、直到、和、或、=、>、<。
- 按照语句应执行的顺序排序语句。
- 如有必要，命名首先检查的条件和第二检查的操作。

将伪代码输入操作体中。输入伪代码后，可以：

- 优化伪代码使其和结构化文本一样执行。
- 使用伪代码设计逻辑并将伪代码作为注释。因为所有结构化文本注释下载到控制器，所以伪代码始终作为操作的文档可用。

要将伪代码转换为结构化文本注释，请添加以下注释符号：

对于注释：	**使用以下格式之一：**
在一行上	*// 注释*
分布在多行	（* *注释开始 . . . 注释结束* *） */* 注释开始 . . . 注释结束 ***

选择操作的限定符

每个操作 （非布尔值和布尔值）使用一个**限定符**决定何时开始和停止。

默认限定符为不存储。操作在步骤激活时开始，步骤取消激活时停止。

要更改操作开始或停止的时间，请指定不同限定符：

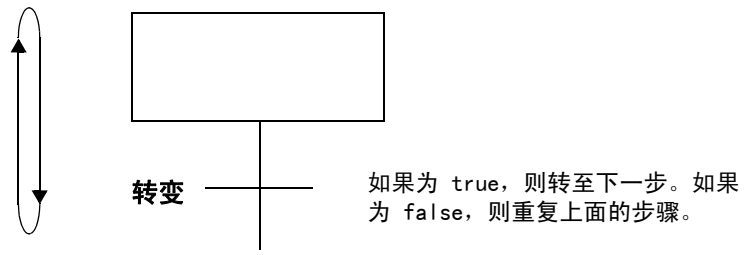
表 4.1 为操作选择限定符

如果希望操作：	并且：	则指定此限定符：	表示：
步骤激活时开始	步骤取消激活时停止	N	不存储
	仅执行一次	P1	脉冲 （上升边缘）
	步骤取消激活前或步骤取消激活时停止	L	时间限制
	保持激活直到 Reset 操作关闭此操作	S	存储
	保持激活直到 Reset 操作关闭此操作 或在特定时间后，即使步骤取消激活	SL	存储且时间限制
步骤激活后开始特定时间并且 步骤仍激活	步骤取消激活时停止	D	时间延迟
	保持激活直到 Reset 操作关闭此操作	DS	延迟且存储
在步骤激活后开始特定时间，即使此段时间前步骤取消激活	保持激活直到 Reset 操作关闭此操作	SD	存储且延迟
步骤激活时执行一次	步骤取消激活时执行一次	P	脉冲
步骤取消激活时开始	仅执行一次	P0	脉冲 （下降边缘）
关闭 （重置）存储的操作： • S 存储 • SL 存储且时间限制 • DS 延迟且存储 • SD 存储且时间延迟	→	R	重置

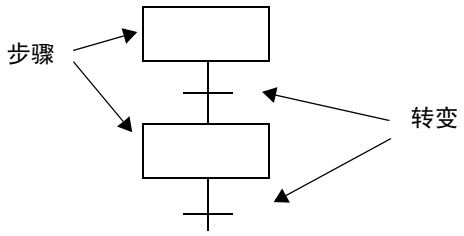
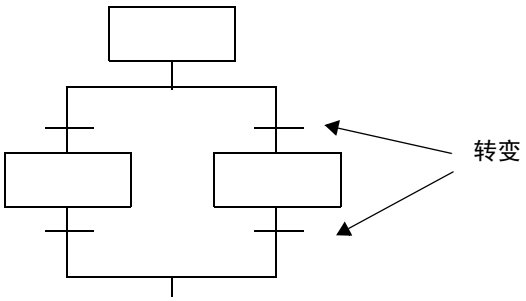
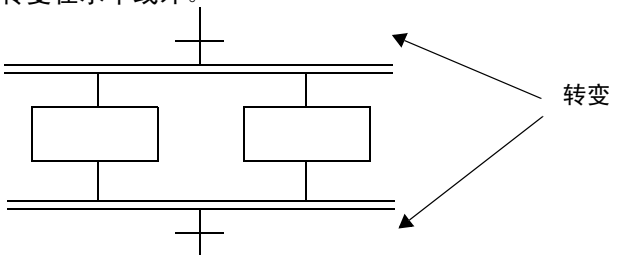
定义转变条件

转变是必须发生或更改以转至下一步的物理情况。

转变通知 SFC 何时进入下一步。



转变发生在：

对于此结构：	确保：
顺序	转变在每个步骤之间。 
选择分支	转变在水平线中。 
同步分支	转变在水平线外。 

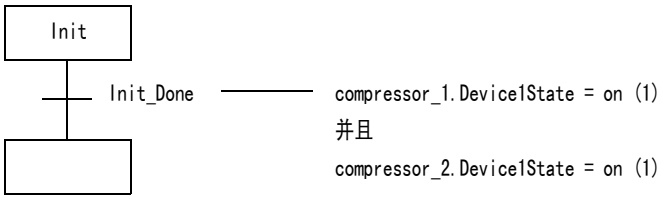
下面是两个转变示例：

示例

您希望：

a. 打开 2 个压缩机。当压缩机打开时 Device1State 位为 on。
b. 当两个压缩机都打开时，则转至下一步。

解决方案：

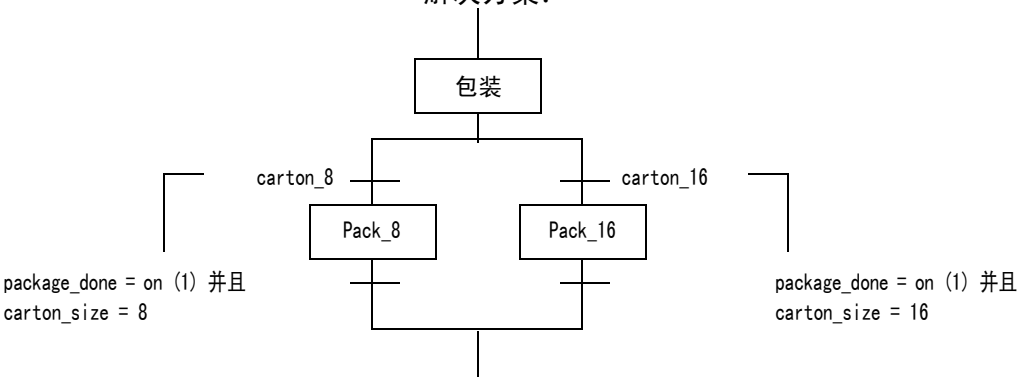


示例

您希望：

a. 包装产品。当产品包装后，package_done 位变为 on。
b. 每箱 8 或 16 个产品包装。

解决方案：



要重写转变的状态，请参阅第 13-1 页的 “强制逻辑单元”。

转变标记

每个转变使用一个 BOOL 标记表示转变的 true 或 false 状态。

如果转变为：	值为：	并且：
true	1	SFC 转至下一步。
false	0	SFC 继续执行当前步骤。

您希望如何编程转变？

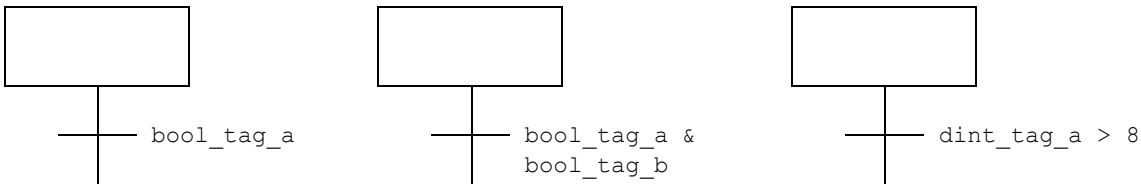
要编程转变，可以选择：

如果希望：	则：
将条件作为结构化文本中的表达式输入	使用 BOOL 表达式
将条件作为其他例程中的指令输入	调用子例程
对多个转变使用相同的逻辑	

使用 BOOL 表达式

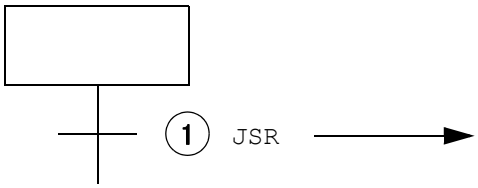
编程转变最简单的方法是将条件作为结构化文本中的 **BOOL 表达式**输入。BOOL 表达式使用 bool 标记、关系运算符和逻辑运算符比较值或检查条件为 true 还是 false。例如 tag1>65。

下面是一些 BOOL 表达式的示例。

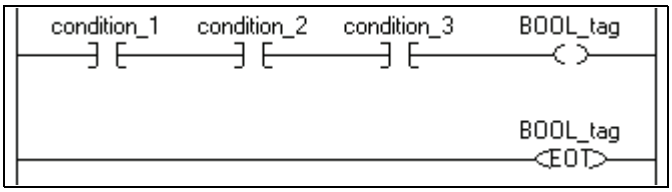


调用子例程

要使用子例程控制转变，请在子例程中包含转变结束 (EOT) 指令。EOT 指令将条件的状态返回转变，如下所示。



2



3

- 或 -

2

```
If condition_1 & condition_2 &
condition_3 then
     BOOL_tag := 1;
Else
     BOOL_tag := 0;
End_if;
```

3

```
EOT(BOOL_tag);
```

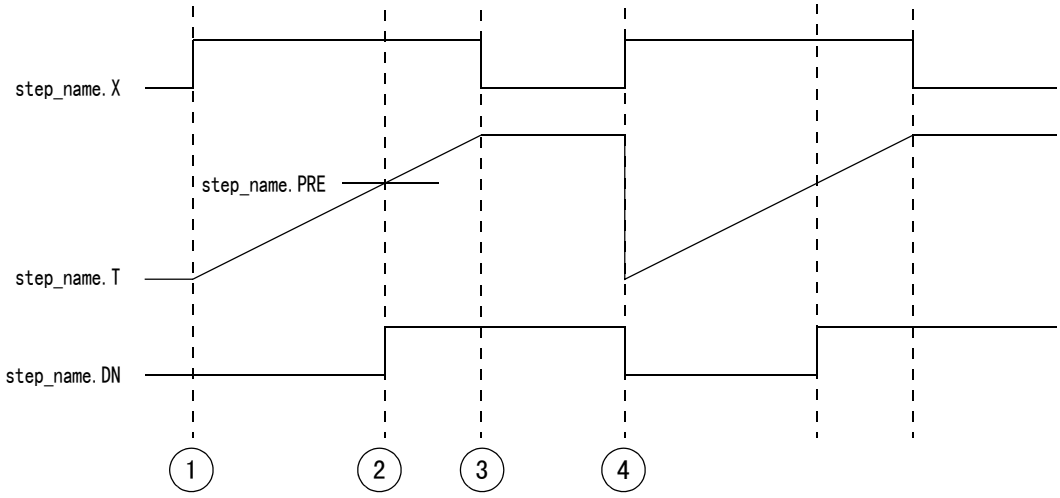
1. 调用子例程。
2. 检查必需的条件。当这些条件为 true 时，打开 BOOL 标记。
3. 使用 EOT 指令设置转变的状态等于 BOOL 标记的值。如果 BOOL 标记为 on (true)，则转变为 true。

指定时间后转变

SFC 的每个步骤包含一个当步骤激活时运行的毫秒计时器。使用计时器：

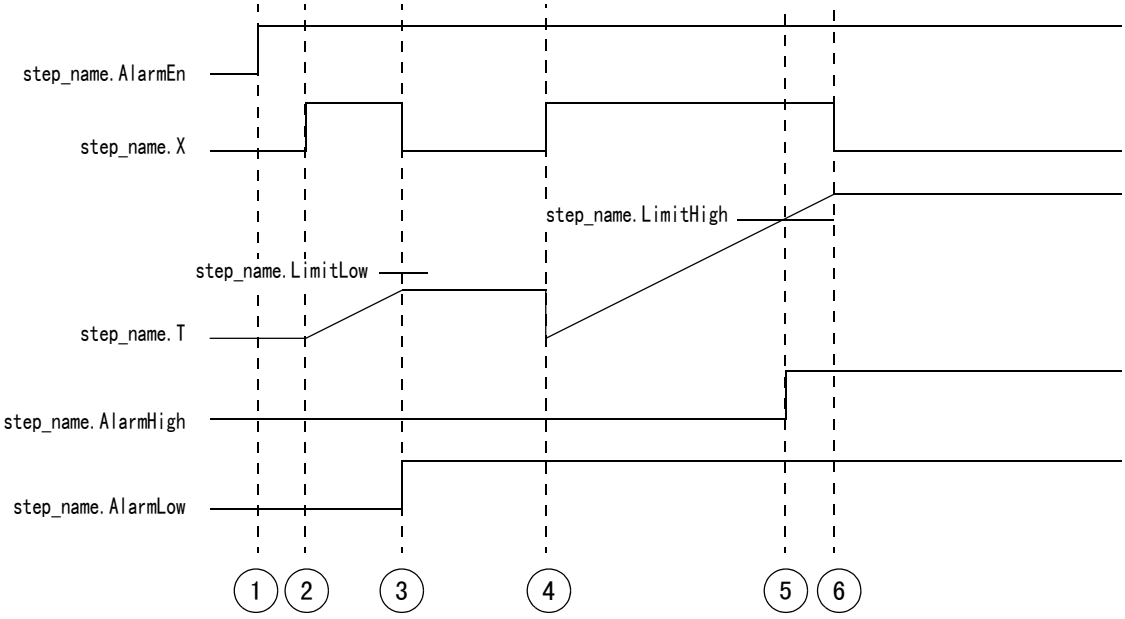
- 当步骤已运行所需时间且 SFC 应转至下一步时发出信号
- 当步骤已运行过久且 SFC 应转至错误步骤时发出信号

图 4.3 计时器操作和步骤的相关位：



说明：
1. 步骤激活。 X 位变为 on。 Timer (T) 开始递增。
2. Timer 达到步骤的 Preset (PRE) 值。 DN 位变为 on。 Timer 继续递增。
3. 步骤变为不激活。 X 位变为 off。 Timer 保持其值。 DN 保留为 on。
4. 步骤激活。 X 位变为 on。 Timer 清除然后开始递增。 DN 位变为 off。

图 4.4 步骤的高低警告的操作：



说明：
1. AlarmEn 为 on。要使用高低警告，请将此位设为 on。通过属性对话框或步骤的标记将此位设为 on。
2. 步骤激活。 X 位变为 on。 Timer (T) 开始递增。
3. 步骤变为不激活。 X 位变为 off。 Timer 保持其值。 因为 Timer 小于 LimitLow，AlarmLow 位变为 on。
4. 步骤变为激活。 X 位变为 on。 Timer 清除然后开始递增。 AlarmLow 保持 on。（必须手动关闭。）
5. Timer 达到步骤的 LimitHigh 值。 AlarmHigh 位变为 on。 Timer 继续递增。
6. 步骤变为不激活。 X 位变为 off。 Timer 保持其值。 AlarmHigh 保持 on。（必须手动关闭。）

下面是使用步骤的 Preset 时间的示例。

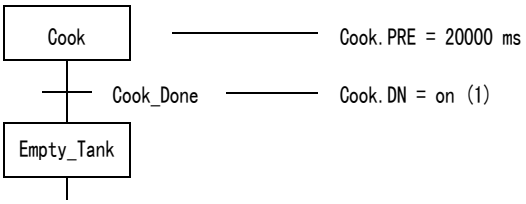
示例

功能规格指示：

a. 在罐内煮原料 20 秒。

b. 清空罐。

解决方案：



下面是一个使用步骤高警告的示例。

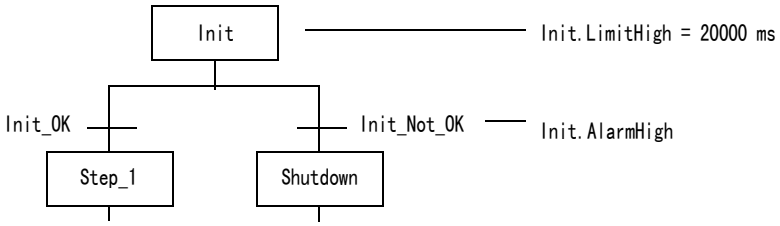
示例

功能规格指示：

a. 复位 8 个设备。

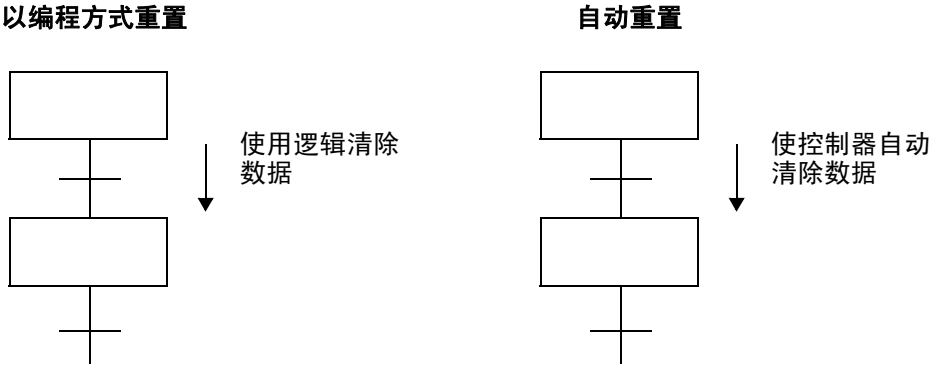
b. 如果 8 个设备没有在 20 秒内复位，则关闭系统。

解决方案：



在步骤结尾关闭设备

当 SFC 离开步骤时，有多种方法关闭步骤打开的设备。



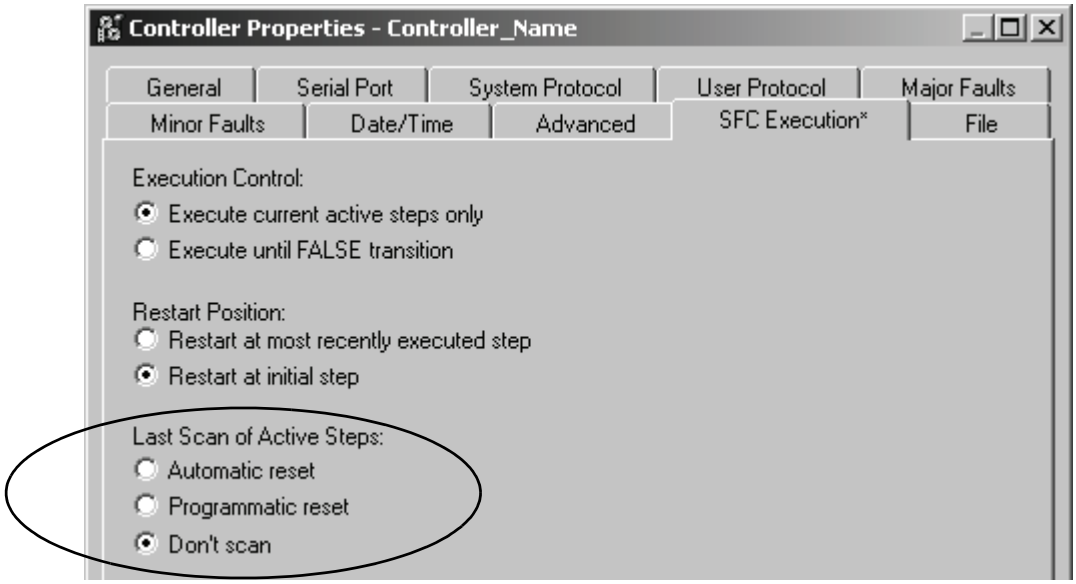
每个选项要求您进行以下选择：

1. 选择最后扫描选项。
2. 根据最后扫描选项设计逻辑使最后扫描将数据返回为所需值。

选择最后扫描选项

在每一步的最后扫描时，具有以下选项。您选择的选项适用于此控制器的所有 SFC 中的所有步骤。

如果希望：	并且在步骤的最后一次扫描：	则：	请参阅：
控制要清除的数据	仅执行 P 和 PO 操作并使用它们清除所需数据。	使用不扫描选项	第 4-30 页
	执行所有操作并使用任一操作清除所需数据： • 控制逻辑条件的步骤或操作的状态位 • P 和 PO 操作	使用以编程方式重置选项	第 4-31 页
让控制器清除数据	——►	使用自动重置选项	第 4-33 页



此表比较处理步骤最后一次扫描的不同选项：

特征：	在步骤的最后一次扫描期间，此选项：		
	不扫描	以编程方式重置	自动重置
执行操作	仅 P 和 PO 操作执行。它们根据逻辑执行。	所有操作根据它们的逻辑执行。	• P 和 PO 操作根据它们的逻辑执行。 • 所有其他操作以后扫描模式执行。 • 在例程的下一次扫描时，P 和 PO 操作以后扫描模式执行。
保留数据值	所有数据保留它们的当前值。	所有数据保留它们的当前值。	• 数据回复它们的值以进行后扫描。 • [:=] 左边的标记分配清零。
清除数据的方法	使用 P 和 PO 操作。	使用： • 控制逻辑条件的步骤或操作的状态位 • P 和 PO 操作	使用： • [:=] 分配（非保留分配） • 后扫描期间清除数据的指令
嵌套 SFC 的重置	嵌套 SFC 保留在当前步骤。	嵌套 SFC 保留在当前步骤。	对于 Restart Position（重新启动位置）属性，如果您选择 Restart at initial step（在初始步骤重新启动）选项，则： • 嵌套 SFC 重置为其初始步骤。 • 嵌套 SFC 中停止元素的 X 位清零。

使用不扫描选项

处理步骤最后一次扫描的默认选项是 ***Don't scan***（不扫描）。利用此选项，SFC 离开步骤时所有数据保留其当前值。这要求您使用额外分配或指令清除要在步骤末尾关闭的所有数据。

在步骤末尾关闭设备：

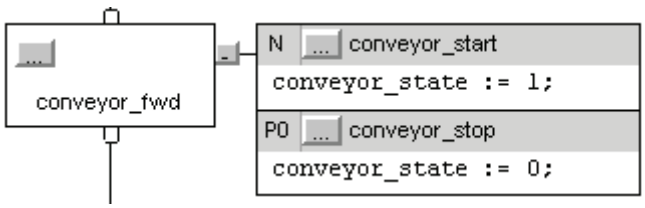
1. 确保 Last Scan of Active Steps（激活步骤的最后一次扫描）属性设置 Don't scan （不扫描）选项 （默认）。
2. 使用 P0 扫描 （下降边缘）操作清除所需数据。确保 P0 操作在步骤的操作顺序的最后。

步骤最后一次扫描期间，Don't scan （不扫描）选项仅执行 P 和 P0 操作。操作的分配和指令根据它们的逻辑条件执行。

- 控制器***不*** 执行分配或指令的**后扫描**。
- SFC 离开步骤时，所有数据保留它们的当前值。

此示例使用操作在步骤开始处打开传送带。另一个操作在步骤末尾关闭传送带。

示例 使用不扫描选项



此操作打开传送带。当 conveyor_state 为 on 时，传送带打开。

SFC 离开步骤前，P0 操作关闭传送带。在步骤的最后一次扫描时 conveyor_state 变为 off。这将关闭传送带。

使用以编程方式重置选项

以编程方式在步骤末尾关闭 （清除）设备的一个可选方法是在步骤的最后一次扫描时执行所有操作。这使您执行正常逻辑并在步骤末尾关闭 （清除）设备。

1. 在 Last Scan of Active Steps （激活步骤的最后一次扫描）属性中选择 Programmatic reset （以编程方式重置）选项：

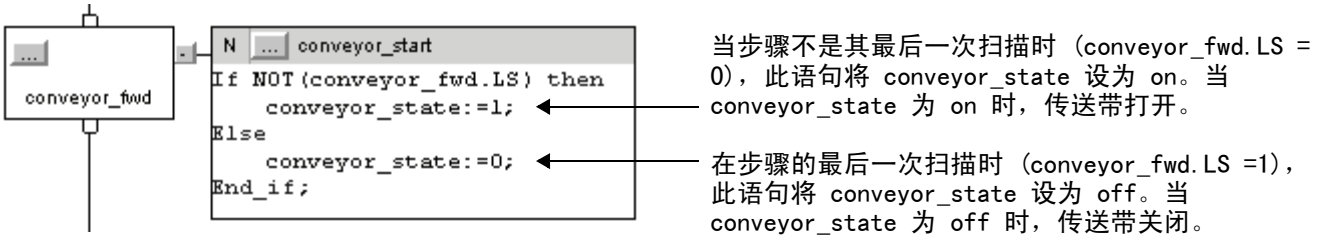
2. 使用以下任意方法清除所需数据：
   - 向正常逻辑添加清除所需数据的逻辑。使用步骤的 LS 位或操作的 Q 位判断逻辑执行的条件。
   - 使用 P0 脉冲 （下降边缘）操作清除所需数据。 确保 P0 操作在步骤的操作顺序的最后。

在步骤的最后一次扫描期间，Programmatic reset （以编程方式重置）选项根据逻辑条件执行所有分配和指令。

- 控制器不**后扫描**分配或指令。
- SFC 离开步骤时，所有数据保留它们的当前值。

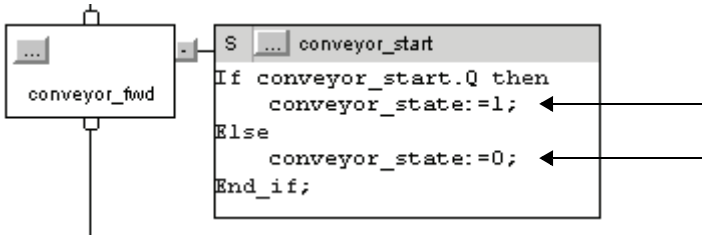
此示例使用一个操作打开和关闭传送带。步骤的 LS 位判断逻辑执行的条件。请参阅第 4-8 页上的 “SFC_STEP 结构”。

示例 使用以编程方式重置选项和 LS 位



对于使用一个存储限定符的操作，使用操作的 Q 位判断逻辑的条件。请参阅第 4-18 页上的 “SFC_ACTION 结构”。

示例 使用以编程方式重置选项和 Q 位

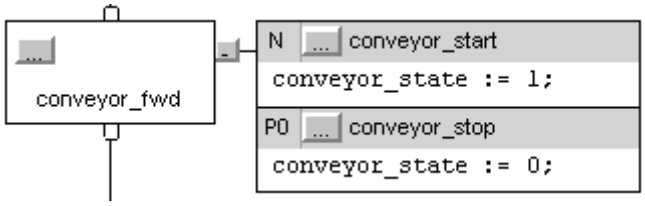


当操作不是其最后一次扫描时 (conveyor_start.Q =1)，此语句将 conveyor_state 设为 on。当 conveyor_state 为 on 时，传送带打开。

在操作的最后一次扫描时 (conveyor_start.Q =0)，此语句将 conveyor_state 设为 off。当 conveyor_state 为 off 时，传送带关闭。

还可以使用 P0 脉冲 （下降边缘）操作清除数据。此示例使用操作在步骤开始处打开传送带。 另一个操作在步骤末尾关闭传送带。

示例 使用以编程方式重置选项和 P0 操作



此操作打开传送带。当 conveyor_state 为 on 时，传送带打开。

SFC 离开步骤前，P0 操作关闭传送带。 在步骤的最后一次扫描时 conveyor_state 变为 off。这将关闭传送带。

使用自动重置选项

在步骤末尾自动关闭 （清除）设备：

1. 在 Last Scan of Active Steps （步骤最后一次扫描）属性中选择 Automatic reset （自动重置）选项。
2. 要在步骤末尾关闭设备，请使用以下类似分配或指令控制设备状态：
   - [:=] 分配 （非保留分配）
   - 子例程中的 Output Energize (OTE) 指令

在每个步骤的最后一次扫描期间，Automatic reset （自动重置）选项：

- 根据它们的逻辑条件执行 P 和 PO 操作
- 清除 [:=] 左边的标记分配
- 执行嵌入结构化文本的**后扫描**
- 执行操作通过 Jump to Subroutine (JSR) 指令调用的任何子例程的后扫描
- 重置任何嵌套的 SFC （操作作为子例程调用的 SFC）

重要 操作的后扫描实际发生在操作从激活变为不激活时。根据操作的限定符，后扫描可能发生在步骤的最后一次扫描前后。

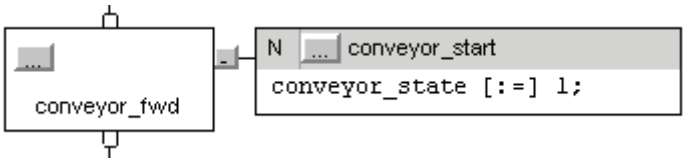
作为一个通用规则，后扫描执行时假定所有条件都为 false。例如 Output Energize (OTE) 指令在后扫描时清除它的数据。

一些指令在后扫描时不遵循此通用规则。有关特定指令在后扫描时如何执行的说明，请参阅：

- *Logix5000 控制器基本指令参考手册*，出版物 1756-RM003
- *Logix5000 控制器过程和驱动器指令参考手册*，出版物 1756-RM006
- *Logix5000 控制器运动指令集参考手册*，出版物 1756-RM007

下面是一个使用非保留分配控制传送带的示例。它在步骤开始时打开传送带，当步骤完成时自动关闭传送带。

示例 自动清除数据



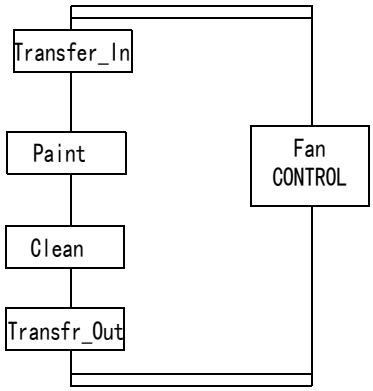
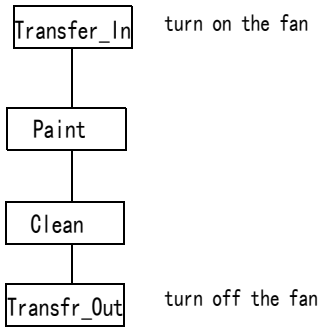
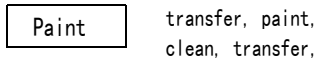
此操作打开传送带。 当 conveyor_state 为 on 时，传送带打开。

SFC 离开步骤时 conveyor_state 变为 off。这将关闭传送带。

逐步操作

您希望如何控制设备?

要在多个时段或阶段 （步骤）中平滑控制设备，请执行以下操作之一：

选项：	示例：
使用同步分支 指定一个单独步骤控制设备。	
存储和重置操作 注意打开设备的步骤和关闭设备的步骤。 之后定义一个存储和重置操作对来控制设备。	
使用大步骤 使用包含设备打开时发生的所有操作的大步骤。	

使用同步分支

在一个或多个步骤中控制设备的简单方法是为设备创建单独步骤。然后使用同步分支在进程的其余时间执行步骤。

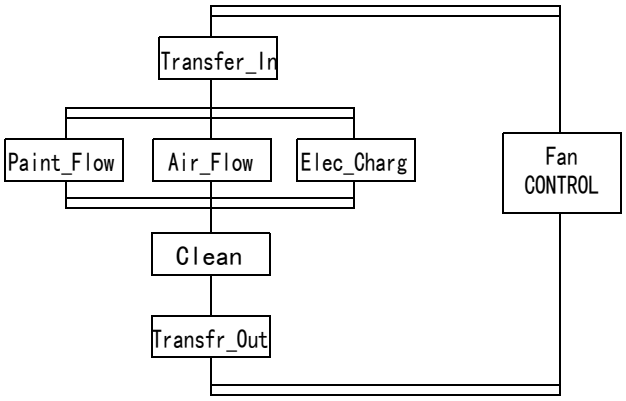
示例

喷涂操作：

1. 将产品送至喷涂车间。
2. 使用 3 个独立喷涂枪对产品喷涂。
3. 清洁枪。
4. 将产品送至喷涂炉。

整个过程中系统必须控制车间风扇。

解决方案：



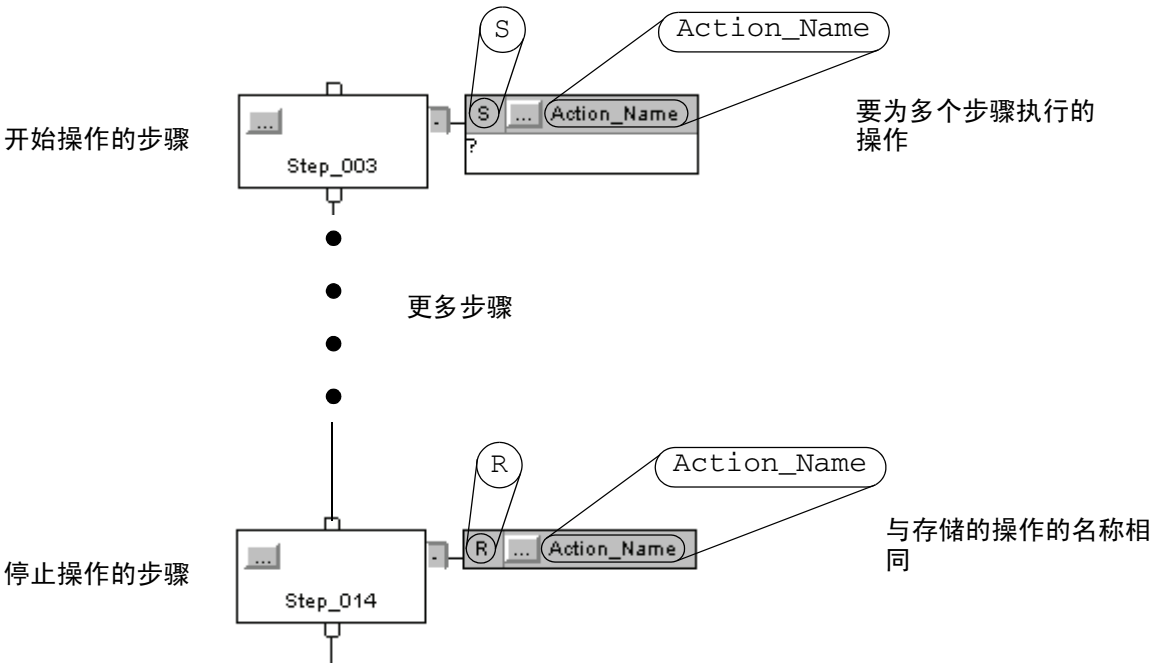
存储和重置操作

通常 SFC 进入下一步时操作关闭 （停止执行）。要保持设备在步骤之间打开而不转变，请存储控制设备的操作：

1. 在打开设备的步骤中，向控制设备的操作分配存储的限定符。有关存储的限定符的列表，请参阅第 4-20 页的表 4.1。

2. 在关闭设备的步骤中使用重置操作。

此图显示存储的操作的用法。

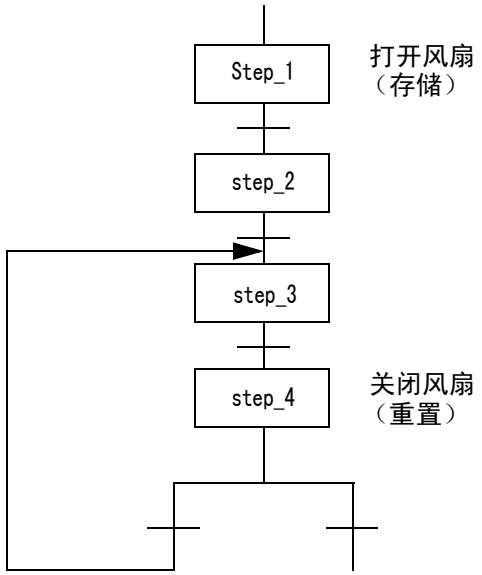


当 SFC 离开存储操作的步骤时，RSLogix 5000 软件继续显示存储的操作为激活。（默认情况下操作周围显示绿色边框。）这使您知道 SFC 正在执行该操作的逻辑。

要使用存储的操作，请使用以下准则：

- 重置操作仅关闭存储的操作。它不自动关闭操作的设备。要关闭设备，请在重置操作后紧跟关闭设备的其他操作。或者使用第 4-33 页上介绍的自动重置选项。
- SFC 到达停止元素前，重置不希望在停止处执行的所有存储操作。即使 SFC 到达停止处，激活的存储操作仍保持激活。
- 在存储操作的步骤和重置操作的步骤间跳跃时请小心。重置操作后，操作仅在您执行存储该操作的步骤时启动。

在此示例中，第 1 - 第 4 步要求风扇打开。在 step_4 的末尾，风扇重置 （关闭）。当 SFC 回跳至 step_3 时，风扇保持关闭。



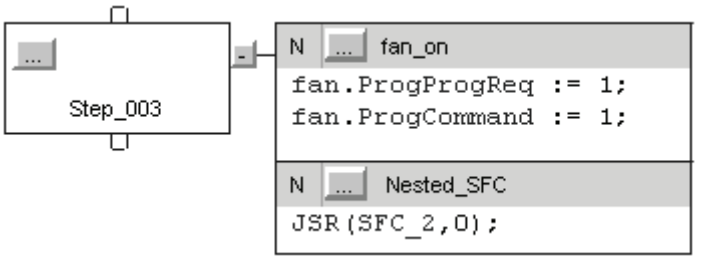
要重新打开风扇，SFC 必须回跳至 step_1。

使用大步骤

如果对多个功能使用一个大步骤，则使用额外逻辑排列功能顺序。一个选项是在大步骤中嵌套 SFC。

在此示例中，一个步骤打开风扇然后调用另一个 SFC。嵌套的 SFC 对步骤的剩余功能排序。在嵌套 SFC 的步骤期间风扇保持打开。

示例 使用大步骤



此操作打开风扇：
- fan.ProgProgReq 使 SFC 命令风扇的状态。
- fan.ProgCommand 打开风扇。

风扇在整个步骤中保持打开。

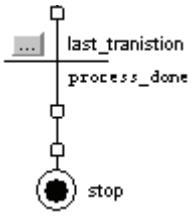
此操作调用另一个 SFC。SFC 对步骤的剩余功能排序。

有关如何嵌套 SFC 的更多信息，请参阅第 4-41 页的 “嵌套 SFC”。

结束 SFC

SFC 完成最后步骤后，不在第一个步骤自动重新开始。您必须通知 SFC 完成最后步骤时进行什么操作。

要：	执行：
自动循环回之前的步骤	将最后一个转变与要跳至的步骤顶连线。 请参阅第 4-14 页上的 “连线上一步”。
停止并等待命令重新开始	使用停止元素。 请参阅第 4-38 页上的 “使用停止元素”。



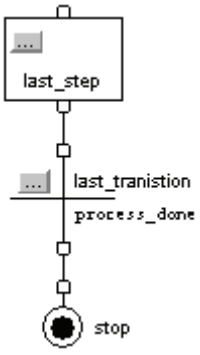
使用停止元素

停止元素使您可以停止整个 SFC 或同步分支的路径的执行并等待重新开始。当 SFC 到达停止元素时：

- 停止元素的 X 位变为 on。这表示 SFC 位于停止元素。
- 存储的操作保持激活。
- 部分或全部 SFC 的执行停止：

如果停止元素位于以下位置末尾：	则：
顺序	整个 SFC 停止
选择分支	
同步分支中的路径	仅该路径停止，而 SFC 的其余部分继续执行。

示例 使用停止元素



当 SFC 到达 last_step 并且 process_done 为 true 时，SFC 的执行停止。

重新开始 （重置）SFC

到达停止元素后，有多种方法重新开始 SFC：

如果 SFC：	并且 *Last Scan of Active Steps*（激活步骤的最后扫描）选项为：	则：
嵌套 （即另一个 SFC 调用此 SFC 作为子例程）	自动重置	在调用嵌套 SFC 的步骤末尾，嵌套的 SFC 自动重置： • 嵌套的 SFC 重置至初始步骤。 • 嵌套 SFC 中停止元素的 X 位清零。
	以编程方式重置 不扫描	1. 使用 SFC Reset （SFR） 指令在所需步骤重新开始 SFC。 2. 使用逻辑清除停止元素的 X 位。
不嵌套 （即没有 SFC 调用此 SFC 作为子例程）	——►	1. 使用 SFC Reset （SFR） 指令在所需步骤重新开始 SFC。 2. 使用逻辑清除停止元素的 X 位。

此示例显示如何使用 SFC Reset （SFR） 指令重新开始 SFC 并清除停止元素的 X 位。

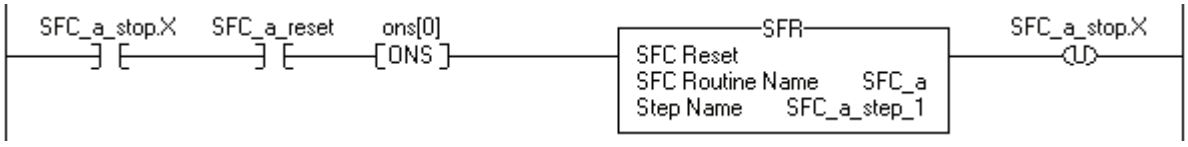
示例

重新开始 （重置）SFC

如果 SFC_a_stop.X = on （SFC_a 在顶部）并且 SFC_a_reset = on （重置 SFC 的时间），则对于一次扫描 （ons[0] = on）：

重置 SFC_a 为 SFC_a_Step_1

SFC_a_stop.X = 0



SFC_STOP 结构

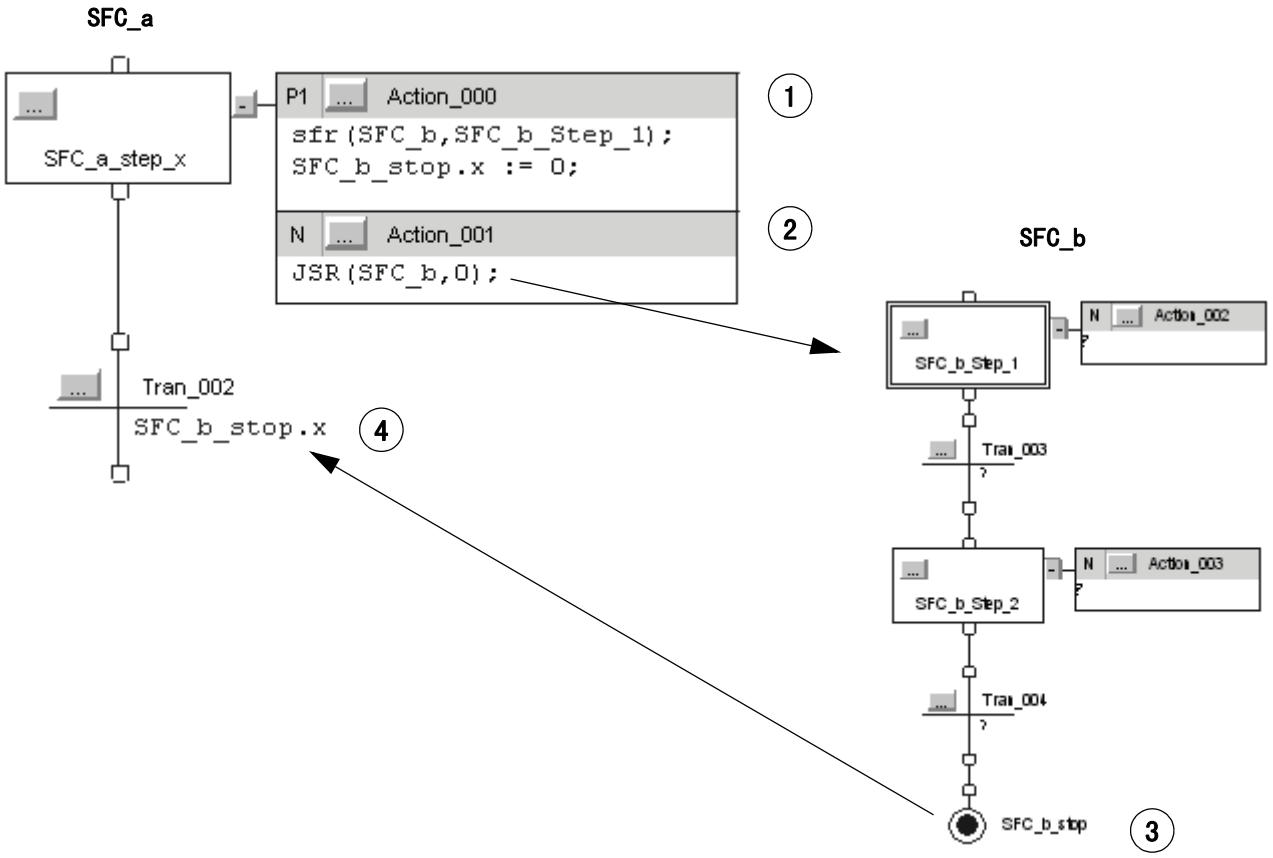
每个停止都使用一个标记提供关于停止元素的信息：

如果希望：	则选中或设置此成员：	数据类型：	详细信息：
决定 SFC 何时位于顶部	X	BOOL	• 当 SFC 到达顶部时，X 位变为 on。 • 如果您配置 SFC 在初始步骤重新开始并且控制器从程序模式更改为运行模式，则 X 位清除。 • 在嵌套 SFC 中，如果您配置 SFC 自动重置并且 SFC 离开调用嵌套 SFC 的步骤，X 位也清除。
决定 SFC Reset (SFR) 指令的目标	重置	BOOL	SFC Reset (SFR) 指令将 SFC 重置为指令指定的步骤或停止。 • Reset 位指示 SFC 将开始再次执行的步骤或停止。 • SFC 执行后，Reset 位清除。
决定停止激活的次数	Count	DINT	这不是停止的扫描计数。 • 每次停止激活时计数递增。 • 仅在停止变为不激活然后再次激活时它递增。 • 仅当您配置 SFC 在初始步骤重新启动时计数重置。 该配置情况下，它将在控制器从程序模式更改为运行模式时重置。
对此停止的各个状态位使用一个标记	状态	DINT	**对于此成员 / 使用此位：** 重置 / 22 X / 31

嵌套 SFC

组织项目的一个方法是创建一个可提供高级别进程视图的 SFC。SFC 的每个步骤调用另一个执行步骤详细步骤的 SFC （嵌套 SFC）。

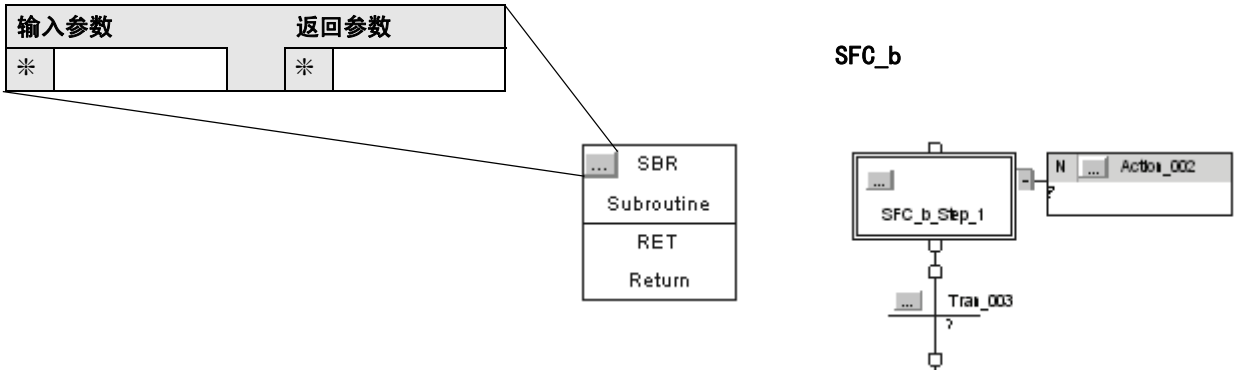
此图显示嵌套 SFC 的方法。在此方法中，SFC 的最后一次扫描选项配置为以编程方式重置或不扫描。如果您配置 SFC 自动重置，则无需第 1 步。



1. 重置嵌套 SFC：
   - SFR 指令在 SFC_b_Step_1 重新开始 SFC_b。每次 SFC_a 离开此步骤然后返回时，必须重置 SFC_b。
   - 操作还清除停止元素的 X 位。
2. 调用 SFC_b。
3. 停止 SFC_b。这将设置停止元素的 X 位。
4. 使用停止元素的 X 位发出信号表示 SFC_b 已完成可以进入下一步。

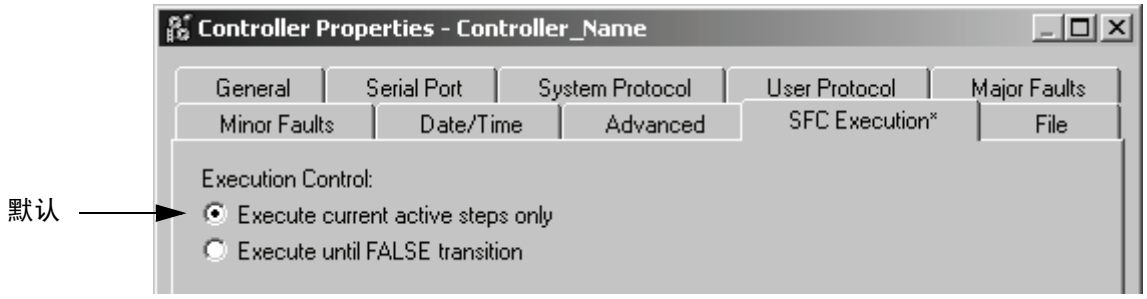
传递参数

要从 SFC 或向 SFC 传递参数，请在 SFC 中放置子例程 / 返回元素。



配置何时返回 OS/JSR

默认情况下，SFC 执行一个步骤或一组同步步骤，然后返回操作系统 (OS) 或调用的例程 (JSR)。



您可以使 SFC 执行直到到达 false 转变。如果多个转变同时为 true，则此选项缩短到达所需步骤的时间。

仅在以下情况使用 Execute until FALSE （执行直至 FALSE）选项：

1. 无需在每个步骤前更新 JSR 参数。仅当 SFC 返回 JSR 时参数更新。请参阅第 4-42 页上的 “传递参数”。

2. 任务的 Watchdog 计数器中发生 false 转变。如果返回 JSR 和完成剩余任务的时间大于 Watchdog 计时器，则发生主故障。

暂停或重置 SFC

有两个可用指令可用于进一步控制 SFC 的执行：

如果希望：	则使用此指令：
暂停 SFC	Pause SFC (SFP)
重置 SFC 至特定步骤或停止	Reset SFC (SFR)

两个指令在梯形逻辑和结构化文本编程语言中都可用。

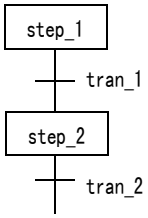
执行图

以下各图显示不同步骤组织或不同执行选项选择的 SFC 的执行情况。

有关图：	请参阅页：
执行顺序	4-44
同步分支执行	4-45
选择分支执行	4-46
参数输入和离开 SFC 时	4-46
执行控制选项	4-47

图 4.5 执行顺序

此…



…执行类似

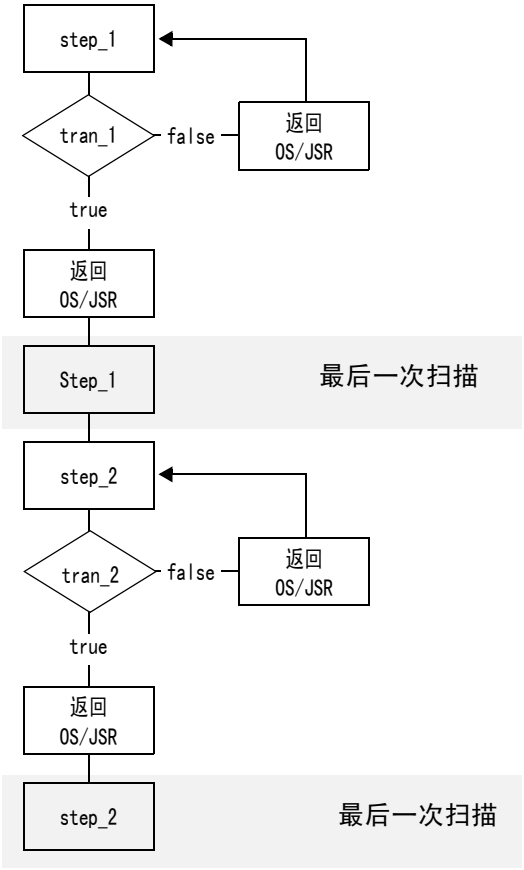


图 4.6 同步分支执行

此… **…执行类似**

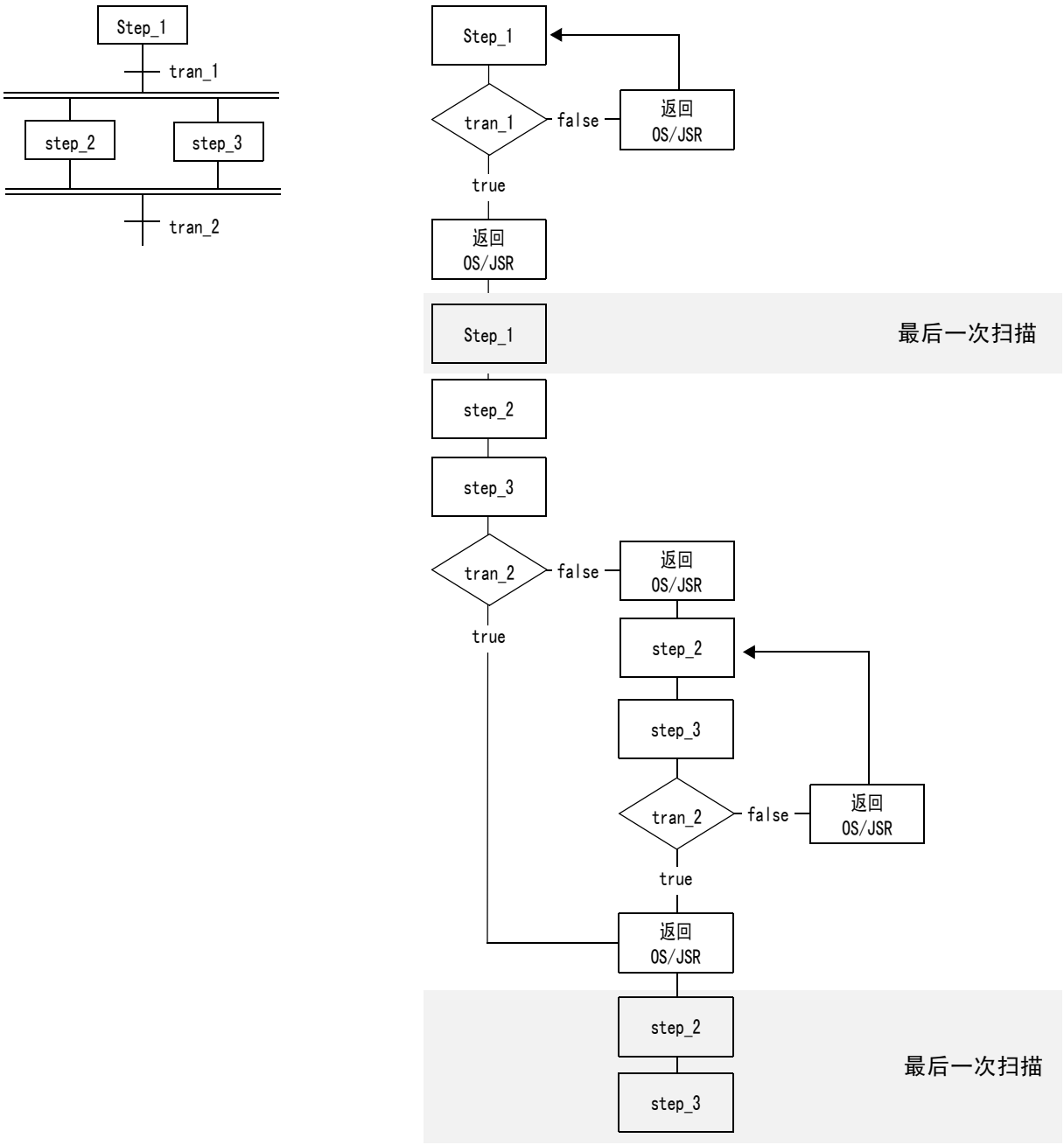


图 4.7 选择分支执行

此… **…执行类似**

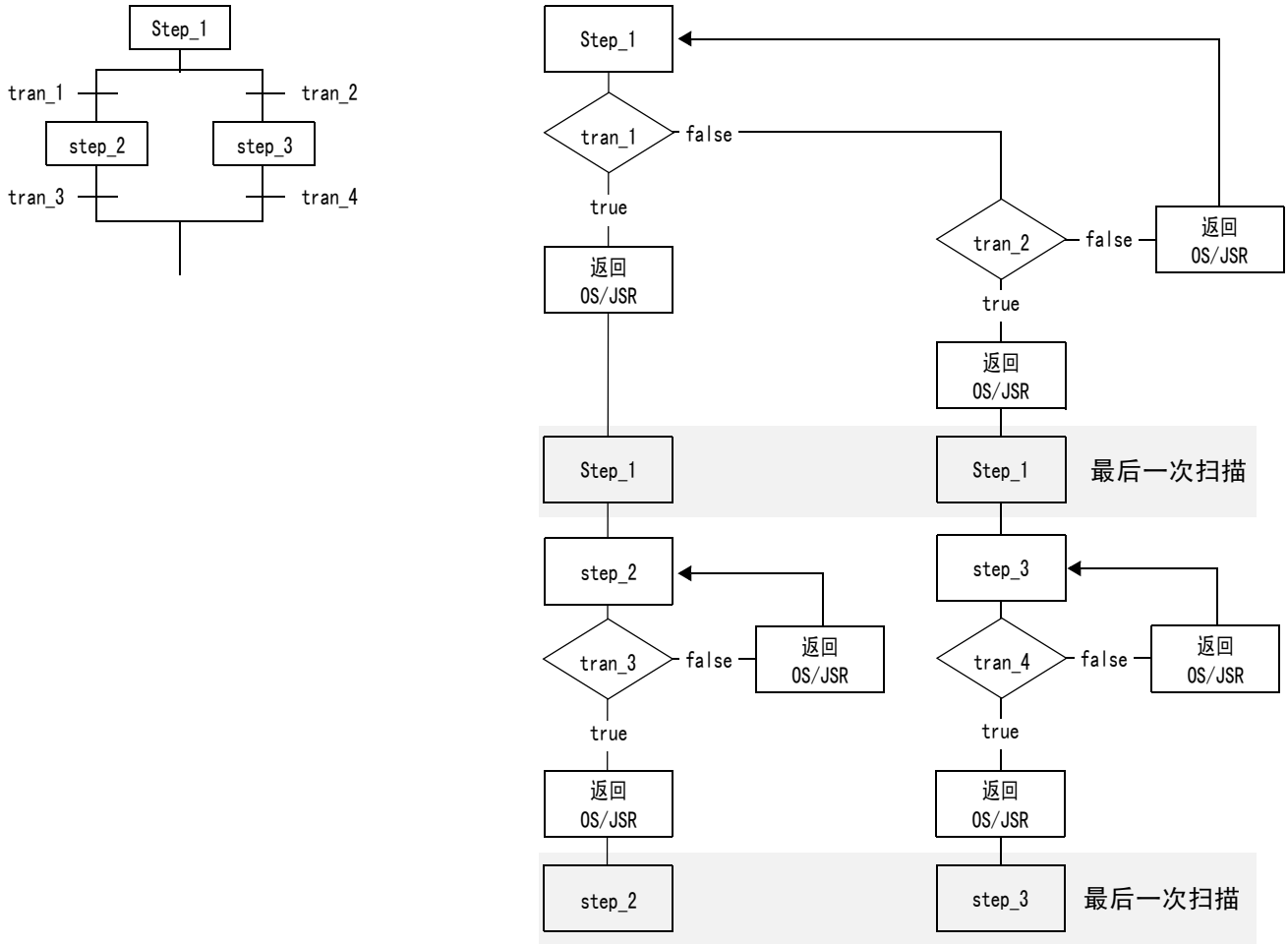


图 4.8 参数输入和离开 SFC 时

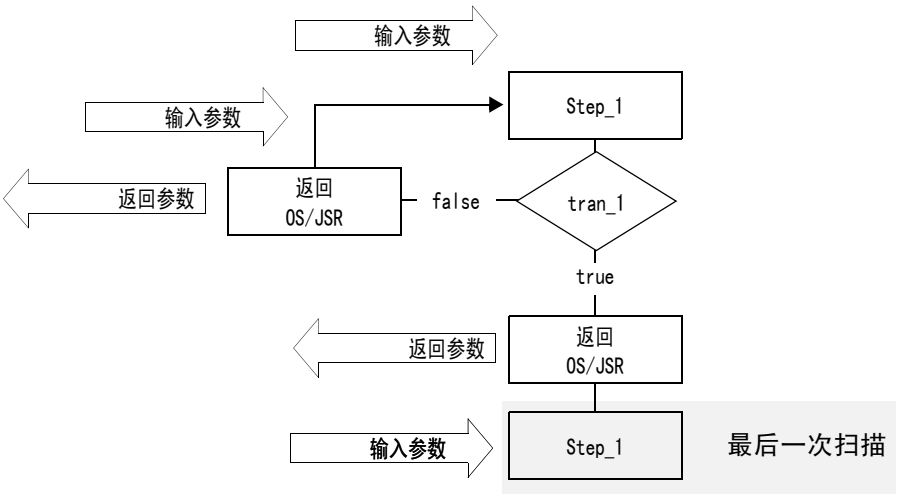
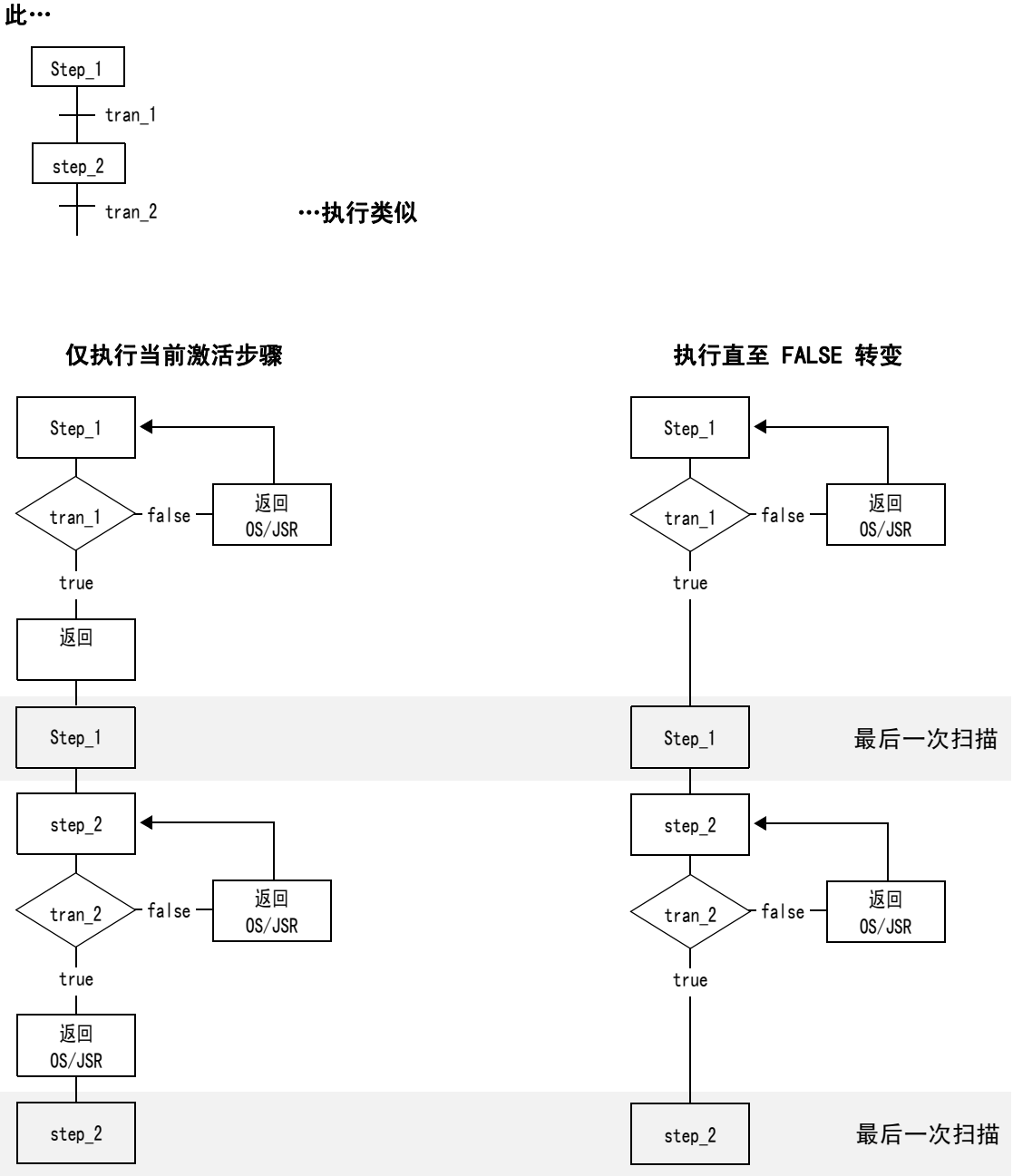


图 4.9 执行控制选项


此…
Step_1
tran_1
step_2
tran_2
…执行类似
仅执行当前激活步骤
执行直至 FALSE 转变
Step_1
tran_1
false
返回
OS/JSR
true
返回
Step_1
step_2
tran_2
false
返回
OS/JSR
true
返回
OS/JSR
step_2
Step_1
tran_1
false
返回
OS/JSR
true
Step_1
最后一次扫描
step_2
tran_2
false
返回
OS/JSR
true
step_2
最后一次扫描

注释：

编程流程图

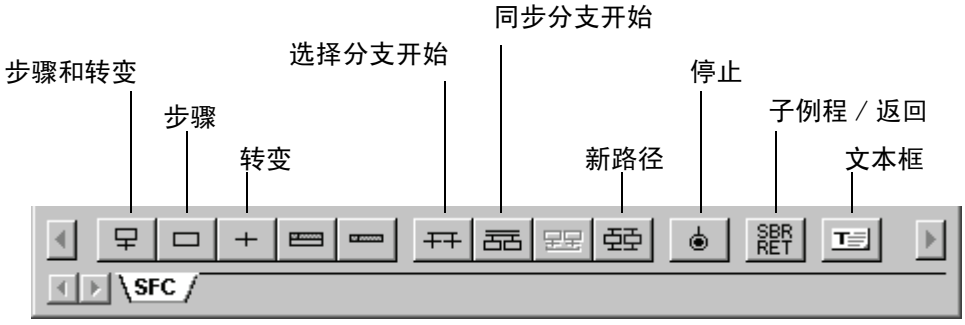
何时使用本章

使用本章将**流程图** (SFC) 输入到 RSLogix 5000 软件中。一边设计 SFC 一边输入。或先设计 SFC 然后输入。要设计 SFC，请参见 第 4-1 页的 “设计顺序流程图”。

编程流程图例程：

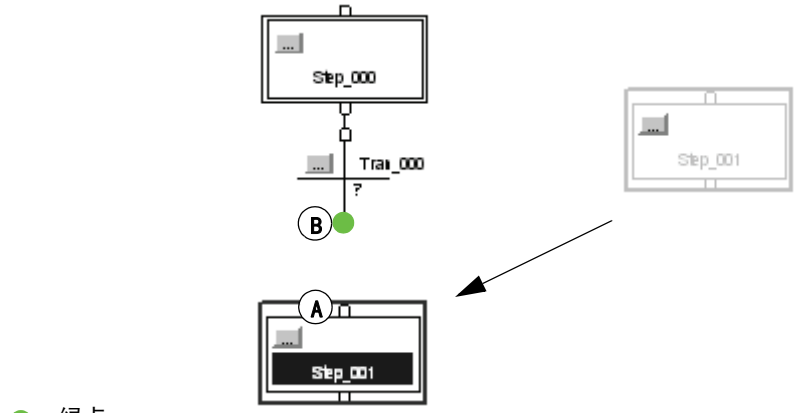
添加 SFC 元素

要添加 SFC 元素，请使用 SFC 工具栏。



添加并手动连接元素

1. 在 SFC 工具栏上单击要添加的项的按钮。
2. 将元素拖动到 SFC 上的所需位置。



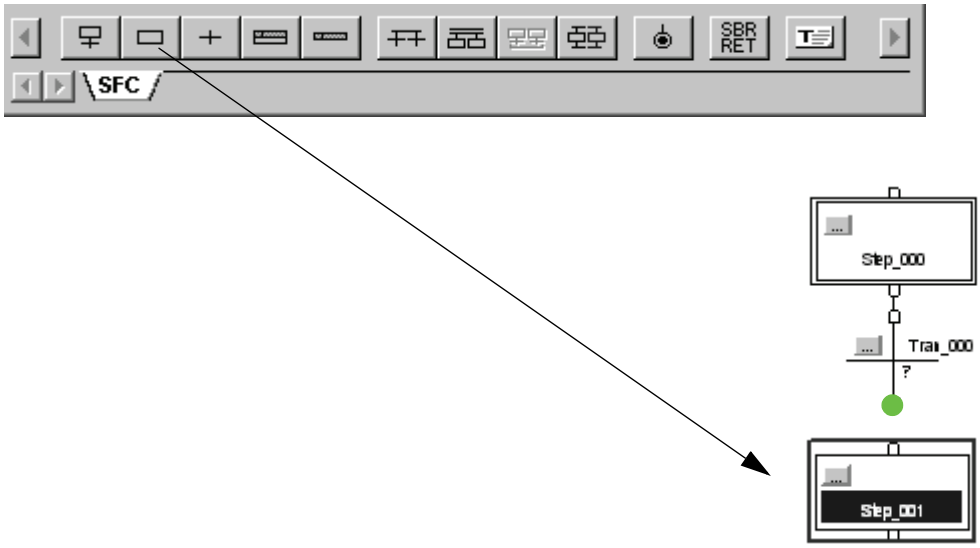
3. 要将两个元素连线 （连接），请单击其中一个元素的引脚 (A) 然后单击另一个元素的引脚 (B)。绿点显示有效连接点。

添加并自动连接元素

1. 选择 （单击）要将新元素连接到的元素。
2. 元素选中时，单击下一个元素的工具栏按钮。

拖放元素

从 SFC 工具栏上将所需元素的按钮拖动到 SFC 上的所需连接点。绿点显示有效连接点。

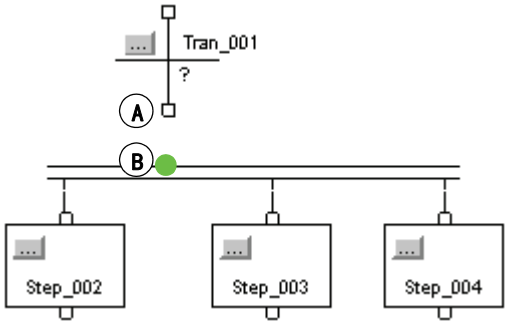


● 绿点

创建同步分支

开始同步分支

1. 在 SFC 工具栏上单击  按钮。将新分支拖动到所需位置。
2. 要将路径添加到分支，请选择 （单击）要添加新路径位置左侧的第一个步骤。单击 

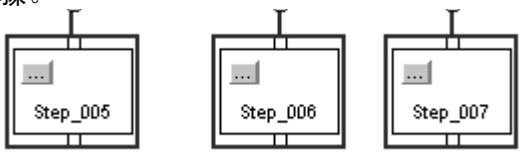


● 绿点

3. 要将同步分支与前面的转变连线，请单击转变的底引脚 (A) 然后单击分支的水平线 (B)。绿点显示有效连接点。

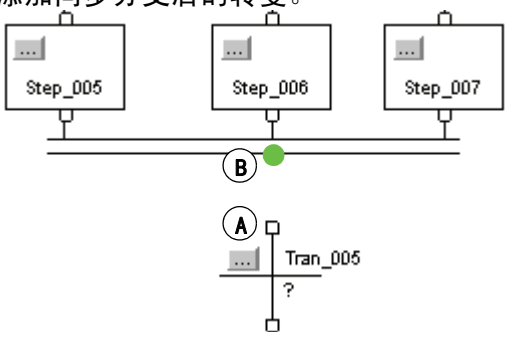
结束同步分支

1. 选择分支中每个路径的最后一步。要选择步骤，您可以：
   - 单击并拖动要选择的步骤周围的指针。
   - 单击第一个步骤。然后按住 [Shift] 单击其余要选择的步骤。



2. 在 SFC 工具栏上单击 

3. 添加同步分支后的转变。



● 绿点

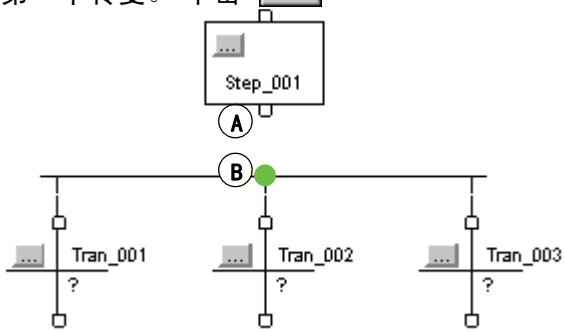
4. 要将同步分支与转变连线，请单击转变的顶引脚 (A) 然后单击分支的水平线 (B)。绿点显示有效连接点。

创建选择分支

开始选择分支

1. 在 SFC 工具栏上单击  按钮。然后将新分支拖动到所需位置。

2. 要将路径添加到分支，请选择 （单击）要添加新路径位置左侧的第一个转变。 单击 



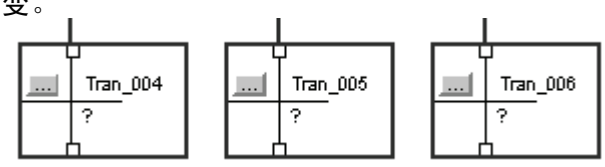
● 绿点

3. 要将选择分支与前面的步骤连线，请单击步骤的底引脚 (A) 然后单击分支的水平线 (B)。 绿点显示有效连接点。

结束选择分支

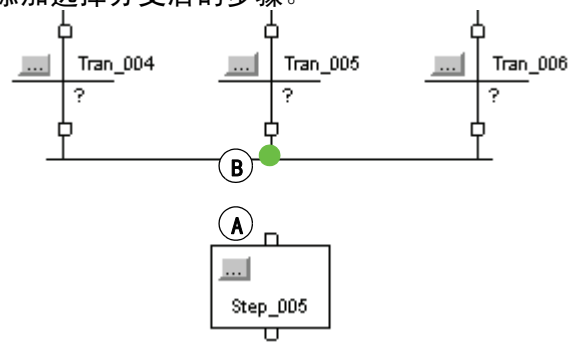
1. 选择分支每个路径的最后一个转变。要选择转变，您可以：

   - 单击并拖动要选择的转变周围的指针。
   - 单击第一个转变。然后按住 [Shift] 单击其余要选择的转变。



2. 在 SFC 工具栏上单击 

3. 添加选择分支后的步骤。



● 绿点

4. 要将选择分支与步骤连线，请单击步骤的顶引脚 (A) 然后单击分支的水平线 (B)。 绿点显示有效连接点。

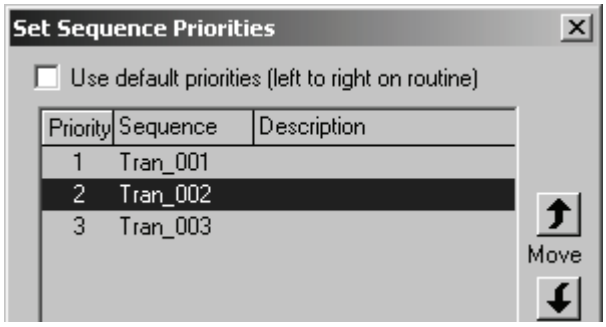
设置选择分支的优先级

默认情况下，SFC 从左到右检查开始每个选择分支的转变。如果希望首先检查不同的转变，请为选择分支的每个路径指定优先级。例如，首先检查错误情况是个好方法。然后检查正常情况。

为选择分支指定优先级：

1. 右击开始分支的水平线，然后选择 Set Sequence Priorities （设置顺序优先级）。

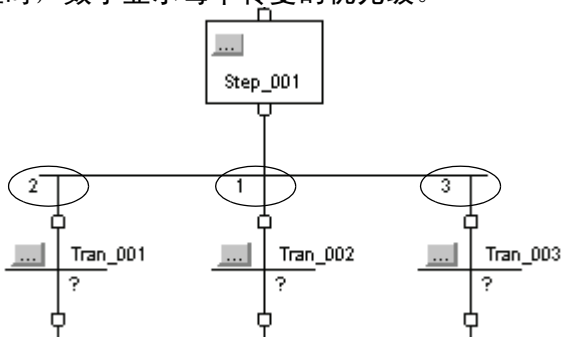
2. 清除 （取消选中）Use default priorities （使用默认优先级）复选框并选择转变。



使用 Move （移动）按钮提高或降低转变的优先级。

3. 所有转变都有所需优先级后，单击 OK

清除 （取消选中）Use default priorities （使用默认优先级）复选框时，数字显示每个转变的优先级。



返回以前的步骤

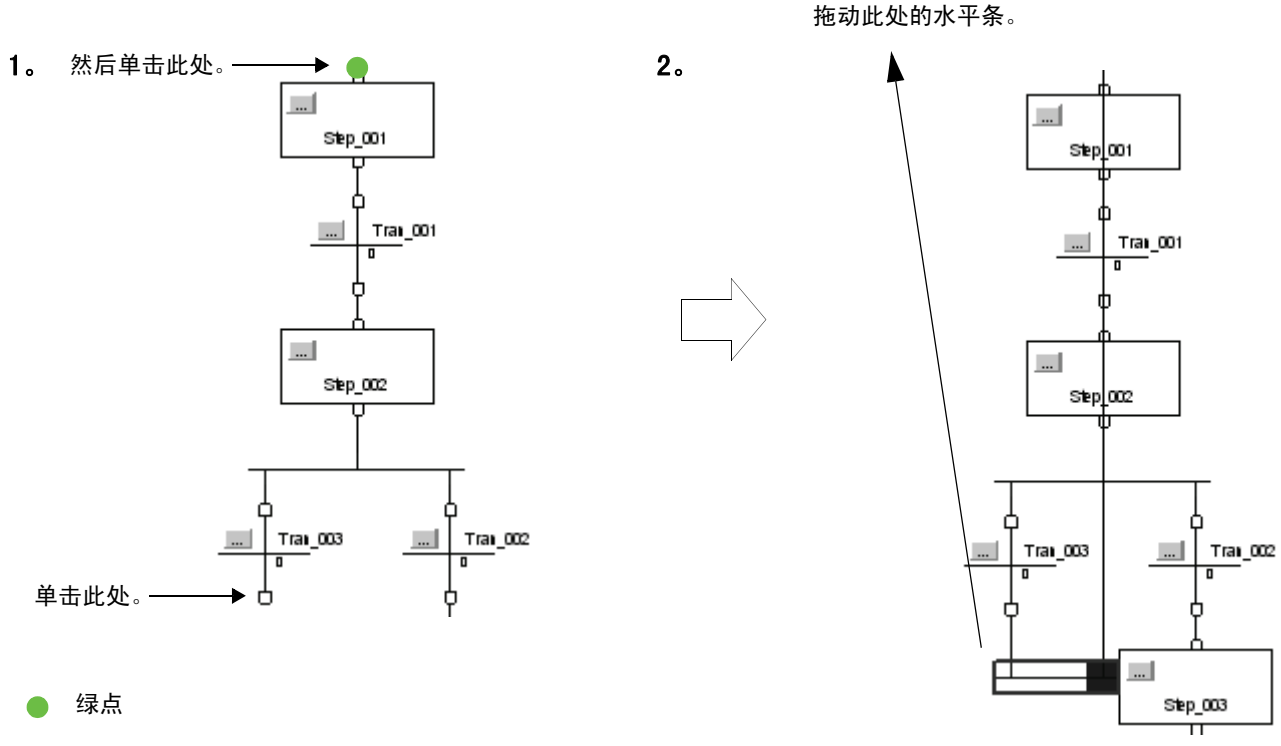
连线步骤

1. 单击发出跳跃信号的转变的底引脚。然后单击要跳至的步骤的顶引脚。绿点显示有效连接点。

   通常，产生的连接将通过流程图中心，难以看见。

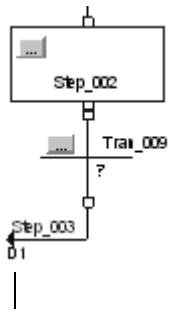
2. 要使跳跃更容易阅读，请拖动其水平条至跳跃到达的步骤上方。可能还需要重新配置一些 SFC 元素。

例如，从 Tran_003 跳至 Step_001：



隐藏线

如果线挡住 SFC 的其他部分，请隐藏线使 SFC 更容易阅读。要隐藏线，请右击线并选择 Hide Wire （隐藏线）。

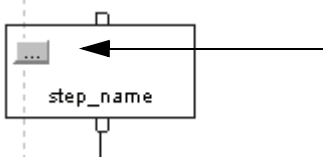


线要到达的位置

要查看线到达的 SFC 元素，请单击线上的网格位置。

配置步骤

为步骤指定预设时间



1. 单击步骤的 ... 按钮。

2. 在 General （一般）选项卡上键入步骤的时间，以毫秒为单位。

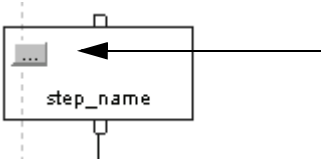
3. 单击 OK

当预设时间步骤激活时 (Timer = Preset)，步骤的 DN 位变为 on。

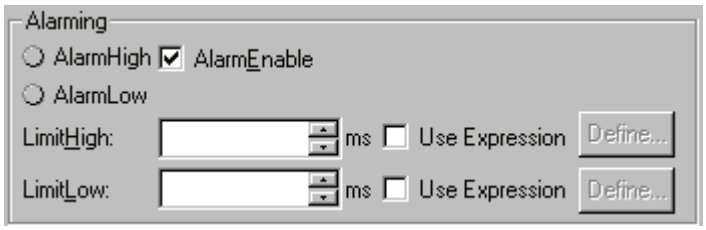
要计算步骤运行时的预设时间，请参见第 5-9 页的 “使用表达式计算时间”。

为步骤配置警告

如果步骤执行过长或时间不够时打开警告：



1. 单击步骤的 … 按钮。
2. 选中 AlarmEnable （启用警告）复选框。



键入高低警告的时间，以毫秒为单位。

3. 单击 OK

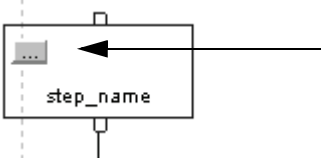
要计算警告运行时的时间，请参见第 5-9 页的 “使用表达式计算时间”。

使用表达式计算时间

要根据项目中的标记计算时间，请以**数值表达式**输入时间。可以使用表达式计算：

- preset
- LimitHigh
- LimitLow

以表达式输入时间：



1. 单击步骤的 … 按钮。
2. 选中 Use Expression （使用表达式）复选框。

3. 单击 Define （定义）按钮并输入表达式。

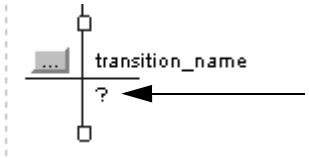
键入定义时间的**数值表达式**。

- 使用对话框旁的按钮帮助完成表达式。
- 有关数值表达式的信息，请参见第 6-4 页的 “表达式”。

4. 单击 OK

5. 要关闭 Step Properties （步骤属性）对话框，请单击 OK

编程转变



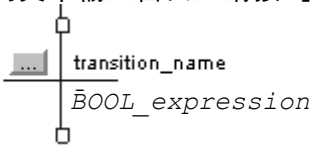
输入 BOOL 表达式

编程转变最简单的方法是将情况作为结构化文本中的 **BOOL 表达式**输入。有关 BOOL 表达式的信息，请参见第 6-4 页的 “表达式”。

1. 双击转变的文本区域。

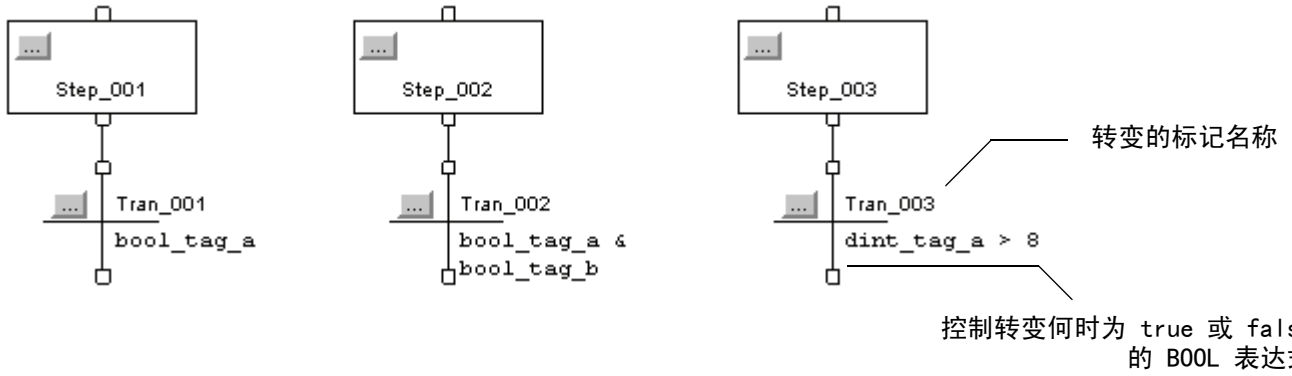
2. 键入决定转变何时为 true 或 false 的 BOOL 表达式。

3. 要关闭文本输入窗口，请按 [Ctrl] + [Enter]。

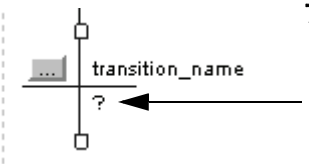


此示例显示三个使用 BOOL 表达式的转变。

示例 输入 BOOL 表达式

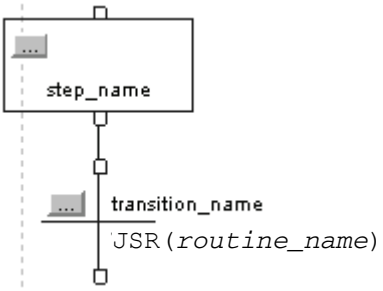


调用子例程



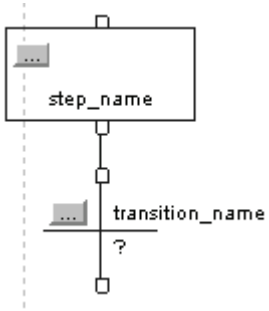
1. 右击转变并选择 Set JSR （设置 JSR）。
2. 选择包含转变逻辑的例程。

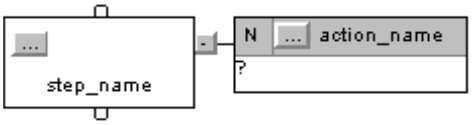
3. 单击 OK



添加操作

要向步骤添加操作，请右击操作执行所在的步骤并选择 Add Action（添加操作）。

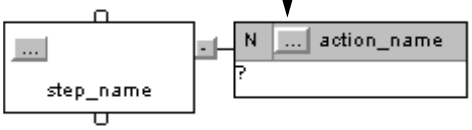




配置操作

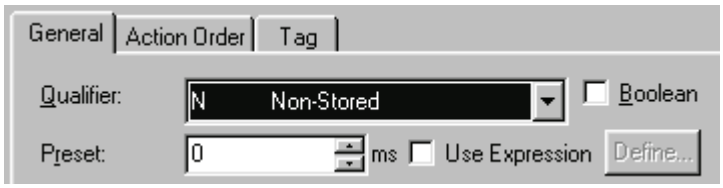
更改操作的限定符

限定符决定操作何时开始和停止。 默认限定符为 N 不存储。操作在步骤激活时开始，步骤取消激活时停止。 有关更多信息，请参见 第 4-19 页的 “选择操作的限定符”。



1. 单击操作的 [...] 按钮。

2. 在 General （一般）选项卡上选择操作的限定符。



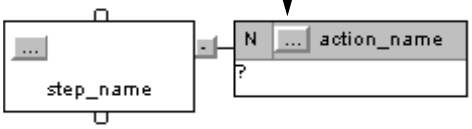
如果选择计时限定符，请键入操作的时间限制或延迟，以毫秒为单位。计时限定符包括：

- L 限时
- SL 存储且时间限制
- D 时间延迟
- DS 延迟且存储
- SD 存储且时间延迟

3. 单击 OK

运行时计算预设时间

要根据项目中的标记计算预设值，请以**数值表达式**输入值。



1. 单击操作的 ... 按钮。

2. 选中 Use Expression （使用表达式）复选框。

3. 单击 Define （定义）按钮。

键入定义预设时间的**数值表达式**。

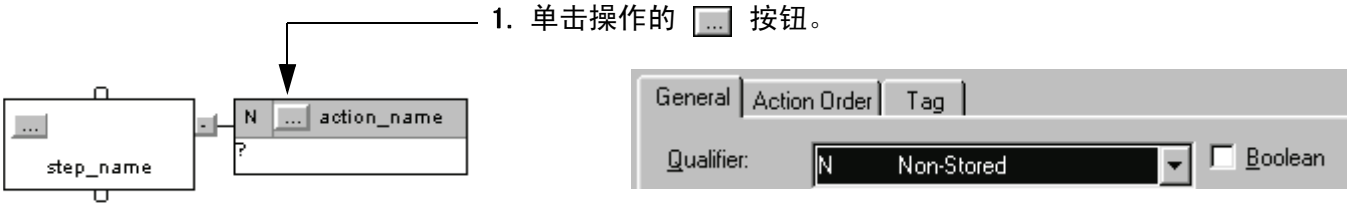
- 使用对话框旁的按钮帮助完成表达式。
- 有关数值表达式的信息，请参见第 6-4 页的 “表达式”。

4. 单击 OK

5. 要关闭 Action Properties （操作属性）对话框，请单击 OK

将操作标记为布尔值奥作

使用布尔值操作在操作执行时仅设置一位。有关更多信息，请参见第 4-17 页的 “使用布尔值操作”。



1. 单击操作的 [...] 按钮。
2. 选中 Boolean （布尔值）复选框。
3. 单击 OK

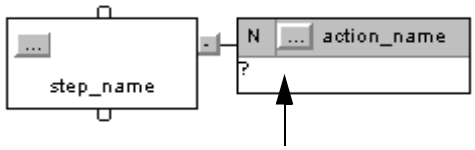
编程操作

要编程操作，您可以：

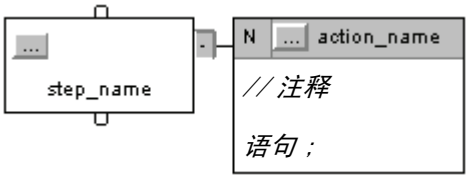
- 输入结构化文本
- 调用子例程

输入结构化文本

最简单的为操作编程的方法是编写逻辑作为操作体中的结构化文本。当操作变为 on 时，控制器执行结构化文本。



1. 双击操作的文本区域。
2. 键入所需的结构化文本。
3. 要关闭文本输入窗口，请按 [Ctrl] + [Enter]。

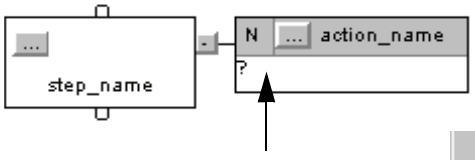


有关结构化文本的信息：

有关此结构化文本信息：	请参见：
关于分配、运算符、函数、指令或注释的一般信息	第 6-1 页的 “结构化文本编程”
有关特定指令的信息	• *Logix5000 控制器基本指令参考手册*，出版物 1756-RM003 • *Logix5000 控制器过程和驱动器指令参考手册*，出版物 1756-RM006 • *Logix5000 控制器运动指令集参考手册*，出版物 1756-RM007

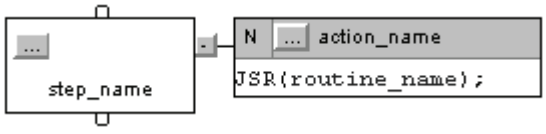
调用子例程

使用 Jump to Subroutine (JSR) 指令在操作激活时执行子例程。



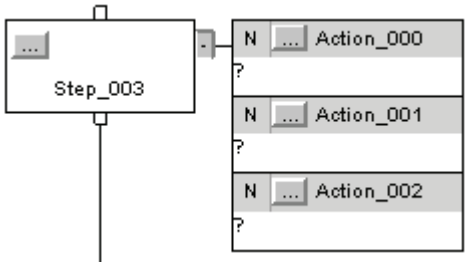
1. 在 SFC 中右击操作的文本输入区域并选择 Set JSR （设置 JSR）。

2. 选择要调用的例程。
3. 要向例程传递参数，请单击一个空的 Input Parameters （输入参数）文本框。然后使用向下箭头选择包含参数的标记。
4. 要从例程接收参数，请单击一个空的 Return Parameters （返回参数）文本框。然后使用向下箭头选择存储来自例程的参数的标记。
5. 单击 OK



指定操作执行顺序

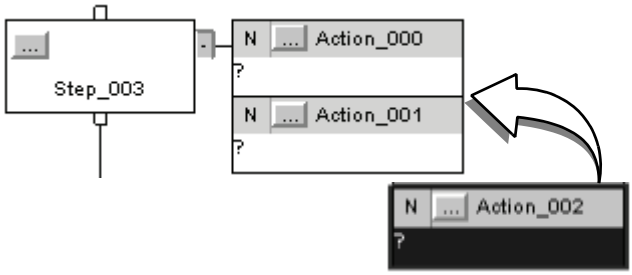
操作按照显示的顺序执行。



当 Step_003 激活时，它的操作按照下面的顺序执行：

1. Action_000
2. Action_001
3. Action_002

要更改操作执行的顺序，请将操作拖动到所需的顺序中的位置。绿色条显示有效放置位置。



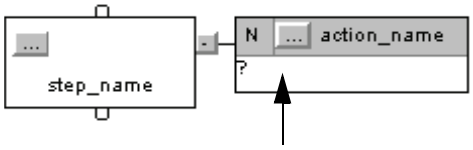
记录 SFC

要记录 SFC，您可以：

要记录：	并且您希望：	执行：
关于 SFC 的一般信息	→	添加文本框
步骤	→	添加文本框 – 或 – 添加标记说明
转变	将文本下载到控制器	添加结构化文本注释
	可以选择显示或隐藏文档记录	添加文本框 – 或 – 添加标记说明
	将文档记录放置在 SFC 中任何位置	
操作	将文本下载到控制器	添加结构化文本注释
停止	→	添加文本框 – 或 – 添加标记说明
其他元素 （例如选择分支）	→	

添加结构化文本注释

嵌入在操作 ST 部分的注释可以下载到控制器内存中，并可用于上载。输入注释：

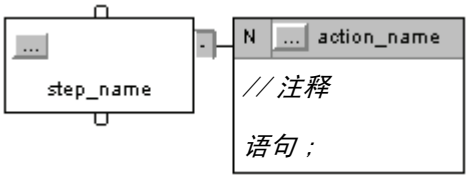


1. 双击操作的文本区域。

2. 键入注释。

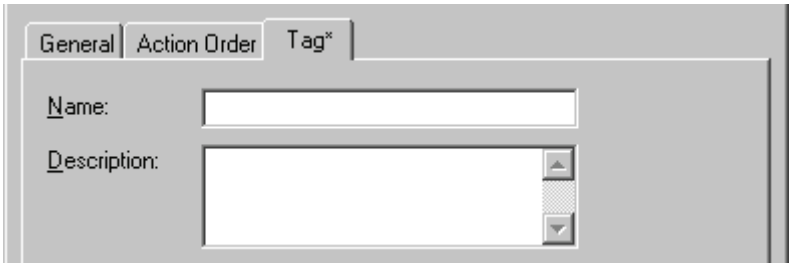
添加注释：	使用以下格式之一：
在一行上	*// 注释*
在一行结构化文本的末尾	(* *注释* *) */* *注释* *\/*
在一行结构化文本中	(* *注释* *) */* *注释* *\/*
分布在多行	(* *注释开始 . . . 注释结束* *) */* *注释开始 . . . 注释结束* *\/*

3. 要关闭文本输入窗口，请按 [Ctrl] + [Enter]。



添加标记说明

1. 单击元素的 ... 按钮。
2. 单击 Tag （标记）选项卡并键入元素的说明。

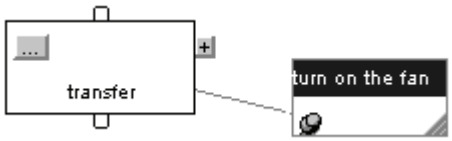


3. 单击 OK
4. 将说明框拖动到 SFC 上的所需位置。

添加文本框

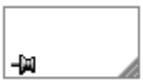
文本框使您可以添加说明 SFC 元素 （步骤、转变、停止等）功能的注释。文本框仅存储在脱机 ACD 项目文件中。文本框不能下载到控制器内存中。

或使用文本框捕获以后将使用的信息。例如：



1. 单击 T

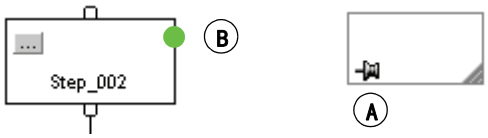
文本框显示。



2. 将文本框拖动到其应用的元素附近的位置。
3. 双击文本框并键入注释。然后按 [Ctrl] + [Enter]。

4. 在 SFC 上移动元素时，您希望文本框做什么操作？

如果您希望文本框：	则：
保留在同一位置	停止。您做到了。
随应用的元素移动	转至第 5 步。



绿点

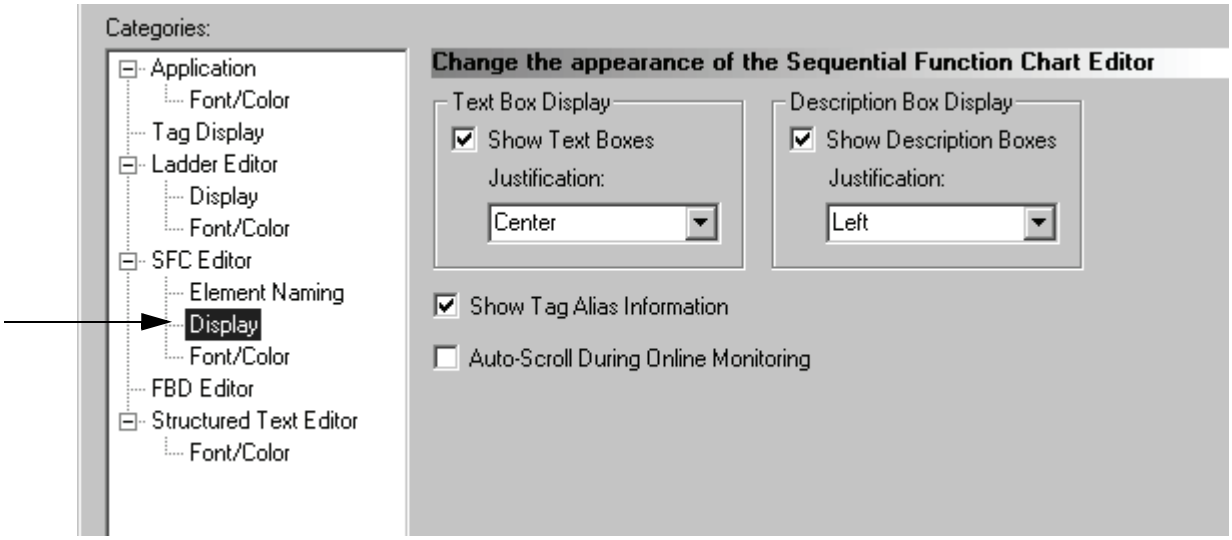
5. 单击文本框中的引脚符号，然后单击要将文本框连接到的 SFC 元素。绿点显示有效连接点。

显示或隐藏文本框或标记说明

您可以显示或隐藏文本框和标记说明。如果您选择显示说明，SFC 窗口仅显示步骤、转变和停止 （不是操作）的说明。

1. 从 Tools （工具）菜单选择 Options （选项）。

2. 在 SFC Editor （SFC 编辑器）下选择 Display （显示）类别。



隐藏单个标记说明

隐藏特定元素的说明而显示其他说明：

1. 单击要隐藏其说明的元素的 ... 按钮。
2. 选中 Never display description in routine （不在例程中显示说明）复选框。

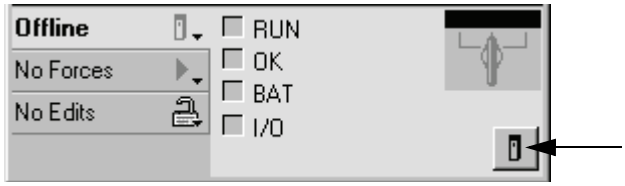
3. 单击 OK

配置 SFC 的执行

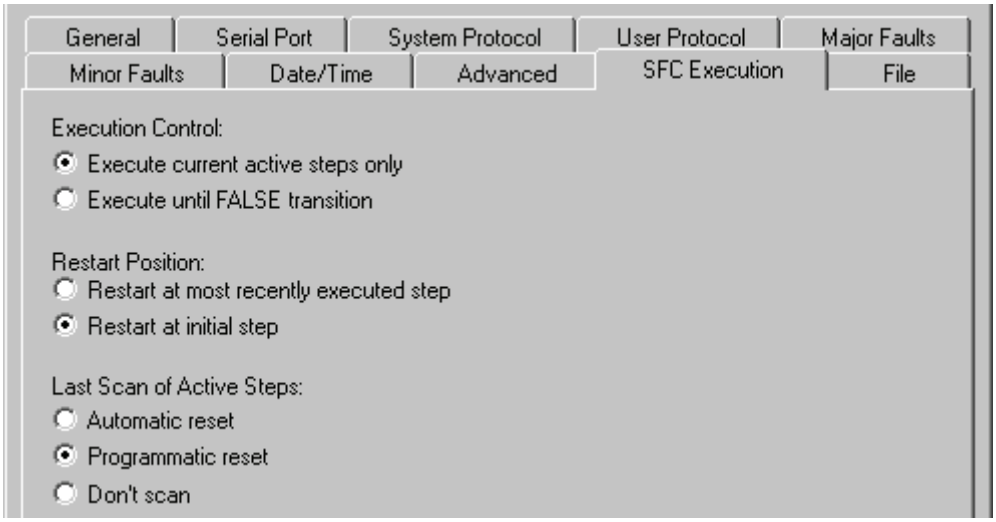
控制器属性的 SFC Execution （SFC 执行）选项卡使您可以配置：

- 当转变为 true 时的操作
- 在转变为运行模式或从断电中恢复后的开始位置
- 步骤最后一个扫描的操作

1. 在 Online （联机）工具栏上单击控制器属性按钮。



2. 选择 SFC Execution （SFC 执行）选项卡。



3. 选择：
   - 转变为 true 时是否返回 OS/JSR。
   - 重新开始 SFC 的位置。

     当控制器断电或离开运行或远程运行模式时，重新开始位置应用。选择：

如果您希望重新开始于：	选择
运行的最后一个步骤	Restart at most recently executed step（在最近执行的步骤处重新开始）
初始步骤	Select Restart at initial step（选择在初始步骤重新开始）

     重新开始位置不会引起主故障。清除主故障后，SFC **始终**在初始步骤重新开始。

   - 步骤最后一个扫描的操作。

4. 单击 OK

检验例程

编程例程时，请定期检验您的工作。

1. 在 RSLogix 5000 窗口最顶部的工具栏上单击 
2. 如果任何错误列在窗口的底部：
   a. 要转至第一个错误或警告，请按 [F4]。
   b. 根据结果窗口的说明更改错误。
   c. 转至第 1. 步
3. 要关闭结果窗口，请按 [Alt] + [1]。

要检查 SFC，您可以：
- 强制转变
- 单步调试 SFC

有关这些调试选项的更多信息，请参见第 13 章强制逻辑单元。

脱机编辑 SFC

固件版本 13 添加了对联机编辑 SFC 的支持。当转换控制器测试或不测试编辑时，控制器重置 SFC 并在初始步骤开始执行。如果您联机编辑 SFC：

- 计时测试或不测试编辑以与执行初始步骤的 SFC 一致。
- 在子例程中放置结构化文本逻辑以最小化联机编辑的影响。
- 使用 SFC 指令以编程方式转移 SFC 执行至所需步骤。

结构化文本编程

何时使用此章节

使用此章节为下面内容写入和输入结构化文本：

- 结构化文本语言例程
- 流程表 (SFC) 操作
- 流程表 (SFC) 转变

结构化文本语法

结构化文本是一种使用语句来定义执行动作的文本化编程语言。

- 结构化文本不区分字母大小写。
- 使用 Tab 标记和回车 （分行）使得结构化文本易于阅读。它们不影响结构化文本的执行。

结构化文本包含以下组成：

术语：	定义：		实例：
赋值 （请参阅第 6-2 页）	使用赋值语句指定标记的数值。 赋值运算符是 :=。 赋值语句以分号 “;” 结束。.		`tag := expression;`
表达式 （请参阅第 6-4 页）	表达式是完整的赋值或结构语句的一部分。表达式计算出数值 （数值表达式）或真假状态 （布尔表达式）。 一个表达式包括：		
	标记	用于存储数据的命名内存区域 (BOOL, SINT, INT, DINT, REAL, string)。	`value1`
	立即数	常数。	`4`
	运算符	在表达式中特指一定操作的符号或助记符。	`tag1 + tag2` `tag1 >= value1`
	函数	一个函数执行后产生一个数值。括号内是其操作数。 函数与指令的语法相似，区别在于函数仅用在表达式中。而指令不能用在表达式中。	`function(tag1)`
指令 （请参阅第 6-11 页）	每条指令是独立的语句。 每条指令使用括号包含其操作数。 根据指令不同，可以为 0、1 或多个操作数。 当指令执行时，产生数据结构的部分值。 指令以分号 “;” 结束。。 指令和函数的语法相似，区别在于指令不能用在表达式中。而函数仅用在表达式中。		*`instruction();`* *`instruction(operand);`* *`instruction(operand1, operand2,operand3);`*

术语：	定义：	实例：
结构 （请参阅第 6-12 页）	用于触发结构化文本代码 （即，其它语句）的条件语句。结构以分号 “;” 结束。	IF...THEN CASE FOR...DO WHILE...DO REPEAT...UNTIL EXIT
注释 （请参阅第第 6-28 页页）	用于解释或阐明一段结构化文本代码功能的文本。 • 使用注释提高结构化文本程序的可读性。 • 注释不影响结构化文本执行。 • 注释可以在结构化文本程序的任意位置。	*//comment* *(*start of comment. . . end of comment*)* */*start of comment. . . end of comment*/*

赋值语句

使用赋值语句改变标记内存储的数值。赋值语句的语法是：

标记 := *表达式* ;

其中：

组成：	说明：	
tag	取得新值的标记 数据类型必须为 BOOL、SINT、INT、DINT 或 REAL	
:=	赋值符号	
expression	赋给标记的新值	
	如果 *tag* 是数据类型：	**使用以下类型表达式：**
	BOOL	布尔表达式
	SINT INT DINT REAL	数值表达式
;	结束赋值	

该*标记*将保持该赋值直至其他赋值语句改变该值。

表达式可以很简单，如立即数或其它标记名，也可以很复杂，包括多个运算符和 / 或函数。详细信息请参阅第 6-4 页上的 “表达式”一节。

提示

I/O 模块数据异步更新逻辑执行。如果在逻辑中多次引用输入，输入可能会在不同的引用间更改状态。如果您需要输入对每个引用都状态相同，则缓存输入值并引用该缓存标记。关于缓存数据的更多信息，请参阅第 1-8 页。

指定非保持赋值语句

非保持赋值语句与上面所描述的常规赋值语句不同，每次控制器进入下一状态，非保持赋值语句中的标记重置为 0：

- 进入运行模式
- 在用户配置 SFC 为*自动重置*的情况下，离开 SFC 的步骤时（仅适用于用户在步骤操作间嵌入赋值语句，或使用操作通过 JSR 指令调用结构化文本例程的情况。）

一个非保持赋值语句的语法：

标记 [:=] *表达式* ;

其中：

组成：	说明：	
tag	取得新值的标记 其数据类型必须为 BOOL、SINT、INT、DINT 或 REAL	
[:=]	非保持赋值符号	
expression	赋给标记的新值	
	如果 *tag* 的数据类型为：	使用以下类型表达式：
	BOOL	BOOL 表达式
	SINT INT DINT REAL	数值表达式
;	结束赋值	

为字符串赋值 ASCII 字符

通过赋值运算符可将 ASCII 字符赋给字符串标记中的 DATA 成员的元素。通过指定字符值或指定标记名称、DATA 成员和字符元素来为字符串赋值。例如：

正确语法：	**错误** 语法。
`string1.DATA[0]:= 65;`	`string1.DATA[0]:= A;`
`string1.DATA[0]:= string2.DATA[0];`	`string1 := string2;`

ASCII 字符串指令用于增加或插入 ASCII 码字符串：

目的：	**所用指令：**
在字符串结尾添加字符	CONCAT
向字符串中插入字符	INSERT

表达式

一个表达式包括标记名、相等和比较关系。使用以下元素写出表达式：

- 存储数值的标记名 （变量）
- 用户直接输入到表达式的数值 （立即数）
- 函数，如：ABS, TRUNC
- 运算符，如：+, -, <, >, And, Or

用户写表达式时必须遵守的规则：

- 表达式中可混合使用大小写字母。例如，三种变体的 “AND”：AND、And 和 and 均可使用。
- 对于较复杂的要求，可在表达式内使用括号集合多个表达式。这样可以提高整个表达式的可读性，并使其按照期望的顺序执行。请参阅第 第 6-10 页 上的 “确定运算顺序。

结构化文本中，用户可以使用以下两种类型的表达式：

布尔表达式：产生 1 （真）或 0 （假）布尔值的表达式。

- 布尔表达式中使用布尔标记、关系运算符和逻辑运算符来比较或检查条件是否为真或假。例如， `tag1>65`。
- 简单的布尔表达式可以是单独 BOOL 标记。
- 一般情况下，用户使用布尔表达式作为其它逻辑执行的条件。

数值表达式：计算整数或浮点数值的表达式。

- 数值表达式使用算术运算符、算术函数和按位运算符。例如，`tag1+5`。
- 通常，用户会将数值表达式嵌套到布尔表达式中。例如，`(tag1+5)>65`。

使用下表为用户表达式选择运算符：

如果用户希望：	那么：
计算算术值	“使用算术运算符和函数”，页 6-6。
比较两数值或字符串	“使用关系运算符”，页 6-7。
检查条件为真或假	“使用逻辑运算符”，页 6-9。
比较数值内各位	“使用按位运算符”，页 6-10。

使用算术运算符和函数

用户可以在算术表达式内使用多个运算符和函数。

使用下面算术运算符计算新值：

目的：	运算符：	最优数据类型：
加	+	DINT, REAL
减 / 非	-	DINT, REAL
乘	*	DINT, REAL
指数 （x 的 y 次幂）	**	DINT, REAL
除	/	DINT, REAL
模数 - 除	MOD	DINT, REAL

算术函数执行数学操作。在函数中需指定常数、非布尔标记或表达式。

目的：	运算符：	最优数据类型：
绝对值	ABS (*numeric_expression*)	DINT, REAL
反余弦	ACOS (*numeric_expression*)	REAL
反正弦	ASIN (*numeric_expression*)	REAL
反正切	ATAN (*numeric_expression*)	REAL
余弦	COS (*numeric_expression*)	REAL
弧度转换成角度	DEG (*numeric_expression*)	DINT, REAL
自然对数	LN (*numeric_expression*)	REAL
以 10 为底的对数	LOG (*numeric_expression*)	REAL
角度转换为弧度	RAD (*numeric_expression*)	DINT, REAL
正弦	SIN (*numeric_expression*)	REAL
平方根	SQRT (*numeric_expression*)	DINT, REAL
正切	TAN (*numeric_expression*)	REAL
截断	TRUNC (*numeric_expression*)	DINT, REAL

例如：

使用该格式：	实例：	
	此情形下：	用户需编写：
value1 operator value2	如果 gain_4 和 gain_4_adj 均为 DINT 型标记且要求：“gain_4 加上 15 并将结果存到 gain_4_adj。”	`gain_4_adj := gain_4+15;`
operator value1	如果 alarm 和 high_alarm 均为 DINT 型标记且要求：“将 high_alarm 取反并将结果存到 alarm。”	`alarm:= -high_alarm;`
function(numeric_expression)	如果 overtravel 和 overtravel_POS 均为 DINT 型标记且要求“计算 overtravel 的绝对值，并将结果存在 overtravel_POS。”	`overtravel_POS := ABS(overtravel);`
value1 operator (function((value2+value3)/2)	如果 adjustment 和 position 均为 DINT 型标记，且 sensor1 和 sensor2 均为 REAL 型标记且要求：计算 sensor1 和 sensor2 平均值的绝对值再加上 adjustment，并将结果存在 position。”	`position := adjustment + ABS((sensor1 + sensor2)/2);`

使用关系运算符

关系运算符将两数值或字符串进行比较，产生一个真或假的结果。关系运算的结果是 BOOL 值：

比较条件为：	结果：
真	1
假	0

使用下列关系运算符：

比较关系：	运算符：	最优数据类型：
等于	=	DINT, REAL, string
小于	<	DINT, REAL, string
小于或等于	<=	DINT, REAL, string
大于	>	DINT, REAL, string
大于或等于	>=	DINT, REAL, string
不等于	<>	DINT, REAL, string

例如：

使用该格式：	实例：	
	此情形下：	用户需编写：
value1 operator value2	如果 temp 是 DINT 标记且要求：“如果 temp 小于 100℃，则…”	`IF temp<100 THEN...`
stringtag1 operator stringtag2	如果 bar_code 和 dest 均为字符串标记且要求：“如果 bar_code 等于 dest，则…”	`IF bar_code=dest THEN...`
char1 operator char2 将 ASCII 字符对应的十进制数值直接输入表达式。	如果 bar_code 是字符串标记且要求：“如果 bar_code.DATA[0] 等于 ’ A’，则…”	`IF bar_code.DATA[0]=65 THEN...`
bool_tag := *bool_expressions*	如果 count 和 length 均为 DINT 标记，done 是 BOOL 型标记，且要求“如果 count 大于或等于 length，计算完成。”	`done := (count >= length);`

如何计算字符串

ASCII 字符对应的十六进制值决定了一个字符串小于或大于另一个字符串。

- 两个字符串按照电话号码簿方式排序时，它们的大小由字符串的顺序决定。

- 如果字符匹配则字符串相等。
- 字符区分大小写。大写 “A”（$41）*不*等于小写 “a”（$61）。

使用逻辑运算符

用户使用逻辑运算符检查多个条件是否为真或假。逻辑操作的结果是 BOOL 值：

比较条件为：	结果：
真	1
假	0

使用下列逻辑运算符：

目的：	运算符：	数据类型：
逻辑与	&, AND	BOOL
逻辑或	OR	BOOL
逻辑异或	XOR	BOOL
逻辑取补	NOT	BOOL

例如：

使用该格式：	实例：	
	此情形下：	用户需编写：
BOOLtag	如果 photoeye 是 BOOL 标记且要求：“如果 photoeye_1 为真，则…”	`IF photoeye THEN...`
NOT *BOOLtag*	如果 photoeye 是 BOOL 标记且要求：“如果 photoeye 为假，则…”	`IF NOT photoeye THEN...`
expression1 & *expression2*	如果 photoeye 是 BOOL 型标记，temp 是 DINT 标记，且要求：“如果 photoeye 为真且 temp 小于 100，则…”.	`IF photoeye & (temp<100) THEN...`
expression1 OR *expression2*	如果 photoeye 是 BOOL 型标记，temp 是 DINT 标记，且要求： “如果 photoeye 为真且 temp 小于 100 ，则…”.	`IF photoeye OR (temp<100) THEN...`
expression1 XOR *expression2*	如果 photoeye1 和 photoeye2 均为 BOOL 标记且要求：“如果： • photoeye1 为真同时 photoeye2 为假或 • photoeye1 为假同时 photoeye2 为真 则…”	`IF photoeye1 XOR photoeye2 THEN...`
BOOLtag := *expression1* & *expression2*	如果 photoeye1 和 photoeye2 均为 BOOL 标记，open 是 BOOL 标记，且要求：“如果 photoeye1 和 photoeye2 同为真，置位 open 为真”。	`open := photoeye1 & photoeye2;`

使用按位运算符

按位运算符基于两个数值处理一个数值中的各位。

目的：	运算符：	最优数据类型：
按位与	&, AND	DINT
按位或	OR	DINT
按位异或	XOR	DINT
按位取补	NOT	DINT

例如：

使用该格式：	实例：	
	此情形下：	用户需编写：
value1 operator value2	如果 input1、input2 和 esult1 均为 DINT 标记，且要求：“计算 input1 和 input2 的按位结果。并将其存到 result1。”	`result1 := input1 AND input2;`

确定运算顺序

运算按预先规定的顺序，而非一定是从左到右来执行写入表达式的运算。

- 同级运算的顺序是从左到右执行。
- 如果表达式中包含多个运算符或函数，可通过圆括号 “( )” 将条件分组。这样可以确保执行顺序正确并使得表达式更具可读性。

顺序：	运算：
1.	( )
2.	function (…)
3.	**
4.	– （取反）
5.	NOT
6.	*, /, MOD
7.	+, – （减）
8.	<, <=, >, >=
9.	=, <>
10.	&, AND
11.	XOR
12.	OR

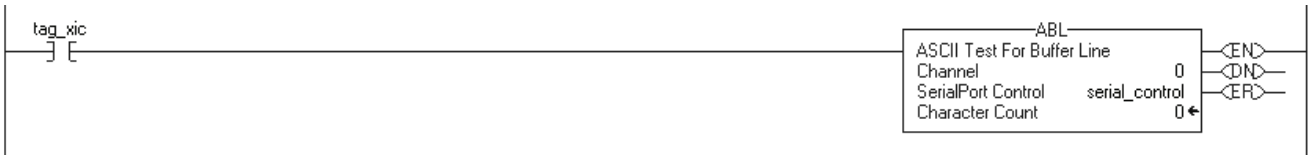
指令

结构化文本编程语句也可以是指令。请参阅本手册开始的指令列表了解结构化文本可用指令。每扫描一次结构化文本程序，指令执行一遍。每次结构条件为真时，结构中结构化文本指令执行。如果结构条件为假，则结构中的语句不被扫描。此时没有梯级条件或状态转变触发指令执行。

与功能块指令的区别在于，功能块指令通过输入使能（EnableIn）触发其执行。而结构化文本指令执行时相当于输入使能（EnableIn）一直置位。

与梯形图的区别在于，梯形图指令通过输入梯级条件触发执行。一些梯形图指令仅当输入梯级条件从假变为真时执行。这些是转变触发型梯形图指令。在结构化文本程序中，除非用户预先规定结构化文本指令的执行条件，否则指令将会在其被扫描时执行。

例如，ABL 指令是转变触发梯形图指令。该例中，当 tag_xic 由清零到置位转变一次，ABL 指令执行一次。当 tag_xic 一直置位或清零时，ABL 指令不执行。



在结构化文本中，如果用户将该例写为：

```
IF tag_xic THEN ABL(0,serial_control);

END_IF;
```

那么，当 tag_xic 置位而不仅是从清零到置位转变时，ABL 指令每次扫描时均执行。

如果用户希望仅当 tag_xic 为从清零到置位转变时 ABL 指令执行，必须编写结构化文本条件指令。使用一次触发条件触发执行。

```
osri_1.InputBit := tag_xic;
OSRI(osri_1);

IF (osri_1.OutputBit) THEN
     ABL(0,serial_control);
END_IF;
```

结构

结构可以单独或与其他结构嵌套编程。

如果用户希望：	使用此结构 ：	可用编程语言：	请参阅页码：
当特定条件发生时，执行某项操作	IF...THEN	结构化文本	6-13
基于数值选择执行的操作	CASE...OF	结构化文本	6-16
根据指定的次数重复执行某项操作，然后再执行其它操作	FOR...DO	结构化文本	6-19
一旦特定条件为真，重复执行某项操作	WHILE...DO	结构化文本	6-22
直到条件为真，否则重复执行某项操作	REPEAT...UNTIL	结构化文本	6-25

保留一些关键词为将来所用

下面的结构不可用：

- GOTO
- REPEAT

RSLogix 5000 软件不允许用户使用这些结构。

IF...THEN

当特定条件发生时，使用 IF…THEN 语句执行某项操作。

操作数：

```
IF bool_expression THEN
    <statement>;
END_IF;
```

结构化文本

操作数：	类型：	格式：	输入：
bool_ expression	BOOL	标记 表达式	计算出布尔值 （布尔表达式）的布尔型标记或表达式

说明： 语法是：

```
IF bool_expression1 THEN
      <statement >;          ◄──── 当 bool_expression1 为真时，执行语句。
                .
                .
                .
可选 ┐ ELSIF bool_expression2 THEN
      <statement >;          ◄──── 当 bool_expression2 为真时，执行语句。
                .
                .
                .
可选 ┐ ELSE
      <statement>;           ◄──── 当两个表达式均为假时，执行语句。
                .
                .
                .
END_IF;
```

根据以下原则使用 ELSIF 或 ELSE。

1. 如果从多个语句组中选择，需添加一个或多个 ELSIF 语句。
   - 每个 ELSIF 表示一个选择路径。
   - 根据用户需要指定 ELSIF 路径数。
   - 控制器执行首个条件为真的 IF 或 ELSIF 语句，并跳过其余的 ELSIF 和 ELSE 语句。
2. 如果要求当所有 IF 或 ELSIF 条件为假时执行语句，需添加一条 ELSE 语句。

下表总结了 IF、THEN、ELSIF 和 ELSE 语句的多种组合。

如果用户要求：	并且：	则使用此结构：
当条件为真时，执行语句	如果条件为假时，不执行语句	IF…THEN
	如果条件为假时，执行其他操作	IF…THEN…ESLE
根据输入条件，从多个语句（语句组）中选择	如果条件为假时，不执行语句	IF…THEN…ELSIF
	如果所有条件为假，分配默认语句	IF…THEN…ELSIF…ELSE

算术状态标志： 不影响

故障条件： 无

实例 1： IF…THEN

如果用户希望：	输入结构化文本程序：

如果 rejects > 3 则
conveyor = off (0)
alarm = on (1)
	IF rejects > 3 THEN
	conveyor := 0;
	alarm:= 1;
	END_IF;

实例 2： IF…THEN…ELSE

如果用户希望：	输入结构化文本程序：

如果传送带方向触点 = 前进 (11)，则
指示灯灭
否则指示灯亮
	IF conveyor_direction THEN
	light := 0;
	ELSE
	light [:=] 1;
	END_IF;

[:=] 会在任何时候控制器进入以下状态时告诉控制器将指示灯清零：

- 进入运行模式
- 在用户配置 SFC 为自动重置的情况下，离开 SFC 的步骤时（仅适用于用户在步骤操作间嵌入赋值语句，或使用操作通过 JSR 指令调用结构化文本例程的情况。）

实例 3： IF…THEN…ELSIF

如果用户希望：	输入结构化文本程序：
如果糖料上下限开关均导通，则 进给阀打开 （导通） 直至糖料上限开关关闭	IF Sugar.Low & Sugar.High THEN Sugar.Inlet [:=] 1; ELSIF NOT(Sugar.High) THEN Sugar.Inlet := 0; END_IF;

[:=] 会在任何时候控制器进入以下状态时告诉控制器将 Sugar.Inlet 清零：

- 进入运行模式
- 在用户配置 SFC 为自动重置的情况下，离开 SFC 的步骤时（仅适用于用户在步骤操作间嵌入赋值语句，或使用操作通过 JSR 指令调用结构化文本例程的情况。）

实例 4： IF…THEN…ELSIF…ELSE

如果用户希望：	输入结构化文本程序：

如果罐温度 > 100
那么设置泵缓慢运行
如果罐温度 > 200
那么设置泵快速运行
其他情况关闭泵
	IF tank.temp > 200 THEN
	pump.fast :=1; pump.slow :=0; pump.off :=0;
	ELSIF tank.temp > 100 THEN
	pump.fast :=0; pump.slow :=1; pump.off :=0;
	ELSE
	pump.fast :=0; pump.slow :=0; pump.off :=1;
	END_IF;

CASE...OF

使用 CASE 语句根据数值选择执行的操作。

操作数：

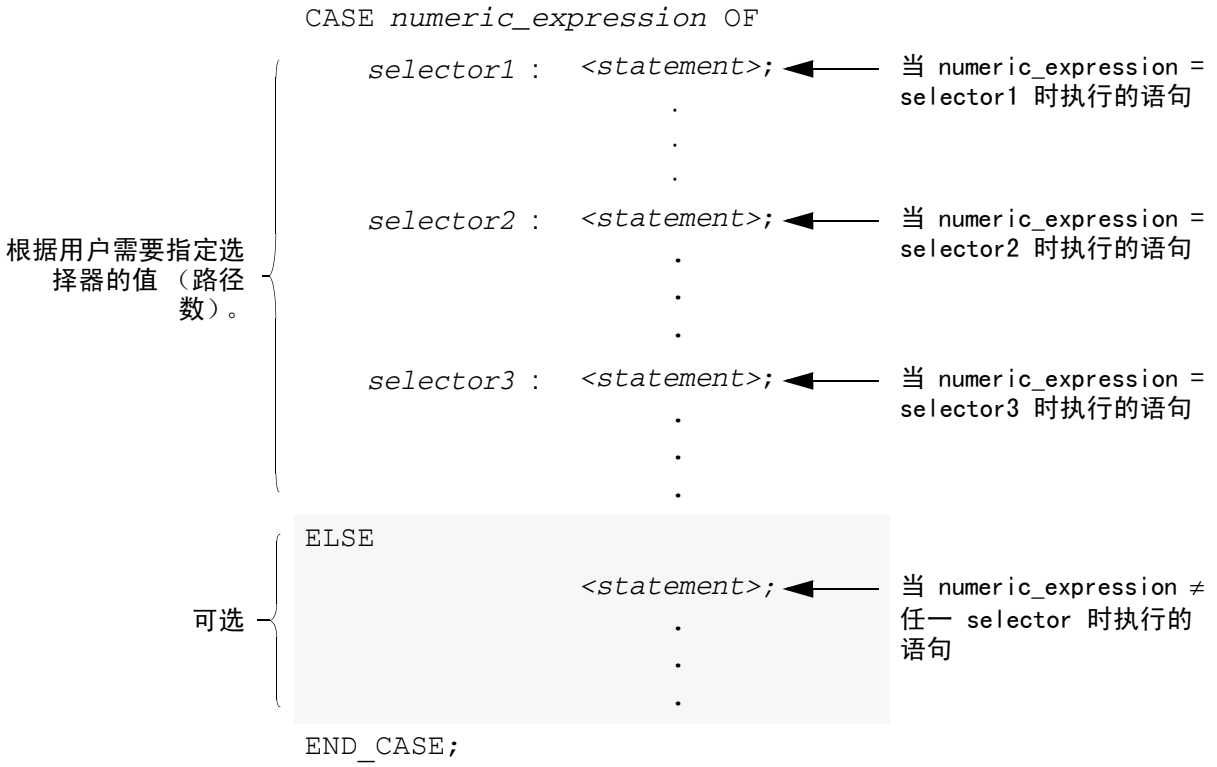
```
CASE numeric_expression OF
     selector1: statement;
     selectorN: statement;
ELSE
     statement;
END_CASE;
```

结构化文本

操作数：	类型：	格式：	输入：
数值表达式	SINT INT DINT REAL	标记 表达式	计算出数值 （数值表达式）的标记或表达式
选择器	SINT INT DINT REAL	立即数	与 *numeric_expression* 类型相同

重要 REAL 数值极有可能在一定的数据范围，而非精确匹配某个特定数值，因此如果用户使用 REAL 数值，必须为选择器设定一个数值范围。

说明： 语法：



有关有效的选择器数值，请参阅下页表格。

输入选择器的语法为：

当选择器为：	输入：
一个值	数值：语句
多个分散的数值	数值 1、数值 2、数值 N：＜语句＞ 使用逗号（,）将每个值分开。
数值范围	数值 1... 数值 N：＜语句＞ 使用两个句点（..）标记范围。
分散的数值加上一个数值范围	数值 a、数值 b、数值 1... 数值 N：＜语句＞

CASE 结构类似于 C 或 C++ 编程语言中的 switch 语句。但是，控制器使用 CASE 结构仅执行与选择器匹配的第一数值相关的语句。选择器语句后的执行一直中断，并转到 END_CASE 语句。

算术状态标志： 不影响

故障条件： 无

实例

如果用户希望：	输入结构化文本程序：
如果配方号 = 1，则	CASE recipe_number OF
成分 A 出口 1 = 打开 (1)	1: Ingredient_A.Outlet_1 :=1;
成分 B 出口 4 = 打开 (1)	Ingredient_B.Outlet_4 :=1;
如果配方号 = 2 或 3，则	2,3: Ingredient_A.Outlet_4 :=1;
成分 A 出口 4 = 打开 (1)	Ingredient_B.Outlet_2 :=1;
成分 B 出口 2 = 打开 (1)	
如果配方号 = 4, 5, 6 或 7，则	4..7: Ingredient_A.Outlet_4 :=1;
成分 A 出口 4 = 打开 (1)	Ingredient_B.Outlet_2 :=1;
成分 B 出口 2 = 打开 (1)	
如果配方号 = 8, 11, 12 或 13，则	8,11..13 Ingredient_A.Outlet_1 :=1;
成分 A 出口 1 = 打开 (1)	Ingredient_B.Outlet_4 :=1;
成分 B 出口 4 = 打开 (1)	
否则所有的出口 = 关闭 (0)	ELSE
	Ingredient_A.Outlet_1 [:=]0;
	Ingredient_A.Outlet_4 [:=]0;
	Ingredient_B.Outlet_2 [:=]0;
	Ingredient_B.Outlet_4 [:=]0;
	END_CASE;

[:=] 会在任何时候控制器进入以下状态时告诉控制器将出口的标记清零：

- 进入运行模式
- 在用户配置 SFC 为自动重置的情况下，离开 SFC 的步骤时（仅适用于用户在步骤操作间嵌入赋值语句，或使用操作通过 JSR 指令调用结构化文本例程的情况。）

FOR…DO

使用 FOR…DO 循环，在执行其他操作前，执行特定次数的操作。

操作数：

FOR *计数值* := *初始值* TO *结束值* BY *增量值* DO
    < *语句* >;
END_FOR;

结构化文本

操作数：	类型：	格式：	说明：
计数值	SINT INT DINT	标记	FOR…DO 结构执行时，标记存储计数值的位置
初始值	SINT INT DINT	标记 表达式 立即数	必须计算一个数 指定计数初始值
结束值	SINT INT DINT	标记 表达式 立即数	指定计数结束值，确定何时退出循环
增量	SINT INT DINT	标记 表达式 立即数	（*可选*）循环一次时计数值的增量 如果用户不指定增量，计数值增量为 1。

重要

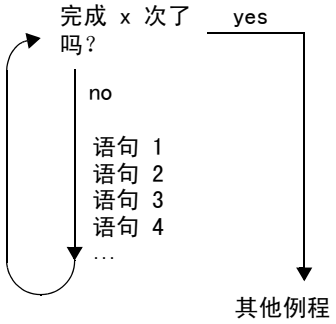
确定在一个扫描周期内重复循环执行的次数不能过多。

- 控制器在完成循环前无法执行例程中其他语句。
- 如果循环完成所用的时间超过了任务监控定时器的值，产生一个主故障。
- 考虑使用不同的结构，如 IF...THEN。

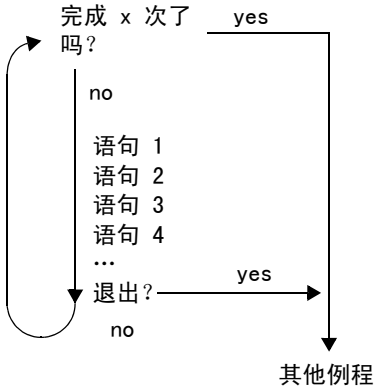
说明： 语法是：

```
FOR count := initial_value
   TO final_value
可选 { BY increment                          如果用户不指定增量，循环增量值为 1。
   DO
   <statement >;
可选 { IF bool_expression THEN
          EXIT;                              如果需要提前退出循环，可使用其他语句，例如采用 IF...THEN 结构作为 EXIT 语句。
       END_IF;
END_FOR;
```

这些图显示了 FOR...DO 循环如何执行，以及如何使用 EXIT 语句提前退出循环。



FOR…DO 循环执行特定次数。



使用 EXIT 语句，在计数值达到最终值前终止循环。

算术状态标志： 不影响

故障条件：

出现主故障的情况：	故障类型：	故障代码：
结构循环太长	6	1

实例 ：

如果用户希望：	输入结构化文本程序：

清除布尔数组的 0 - 31 位：
1. 初始化 subscript 标记为 0。
2. 清除 array[subscript]。 例如，当 subscript = 5，清除 array[5]。
3. subscript 加 1。
4. 如果 subscript ? 31，重复步骤 2 和 3。
否则停止。
	`For subscript:=0 to 31 by 1 do`
	`  array[subscript] := 0;`
	`END_FOR;`

实例 2:

如果用户希望：	输入结构化文本程序：

通过用户自定义数据类型 （结构）存储用户的货物库存信息：
• 货物的条形码 ID 号 （字符串数据类型）
• 货物的库存数量 （DINT 数据类型）
上面的自定义结构数组包含了库存货物的各个元素。用户希望搜索特定产品 （使用条形码），并确定采购数量。
1. 设置 Inventory 数组的大小 （货物数），并将结果存储到 Inventory_Items （DINT 标记）。
2. 初始化 position 标记值为 0。
3. 如果 Barcode 与数组中某条目的 ID 一致，那么：
a. 设置 Quantity 标记 = Inventory[position].Qty。 生成该货物库存量。
b. 结束。
Barcode 是存储货物条形码的字符串标记，用户搜索该项。例如，当 position = 5，比较 Barcode 和 Inventory[5].ID。
4. position 加 1。
5. 如果 position ? (Inventory_Items -1)，重复步骤 3 和步骤 4。由于元素编号起始于 0，最后一个元素编号比数组的元素号少 1。否则停止。
	SIZE(Inventory,0,Inventory_Items);
	For position:=0 to Inventory_Items - 1 do
	If Barcode = Inventory[position].ID then
	Quantity := Inventory[position].Qty;
	EXIT;
	END_IF;
	END_FOR;

WHILE…DO

只要条件为真，则使用 WHILE…DO 循环连续工作。

操作数：

```
WHILE bool_expression DO
    <statement>£a
END_WHILE;
```

结构化文本

操作数：	类型：	格式：	输入：
布尔表达式	BOOL	标记 表达式	计算出布尔值的 BOOL 标记或表达式

重要

确定在一个扫描周期内重复循环执行的次数不能过多。

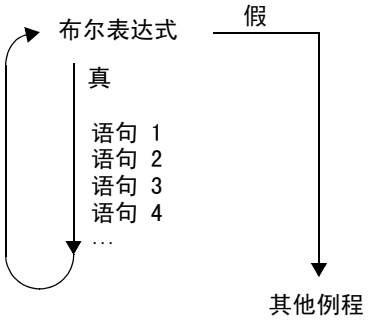
- 控制器在完成循环前无法执行例程中其他语句。
- 如果循环完成所用的时间超过了任务监控定时器的值，产生一个主故障。
- 考虑使用不同的结构，如 IF...THEN。

说明： 语法为：

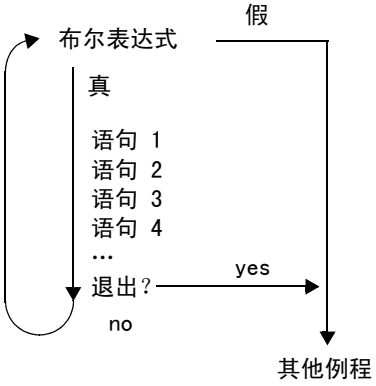
```
WHILE bool_expression1 DO
   <statement>;          <-- 当 bool_expression1 为真时，执行语句。
   IF bool_expression2 THEN
     EXIT;               <-- 如果需要提前退出循环，可使用其他语句，例如采用 IF...THEN 结构作为 EXIT 语句。
   END_IF;
END_WHILE;
```

可选

以下图示了 WHILE...DO 循环如何执行，以及如何使用 EXIT 语句提前退出循环。



当 *bool_expression* 表达式为真，控制器仅执行 WHILE…DO 循环内的语句。



在条件为真前，使用 EXIT 语句退出循环。

算术状态标志： 不影响

故障条件：

出现主故障的情况：	故障类型：	故障代码：
结构循环太长	6	1

实例 1:

如果用户希望：	输入结构化文本程序：

WHILE...DO 循环首先判断其条件。如果条件为真，控制器执行循环内的语句。
与 REPEAT...UNTIL 循环的区别在于 REPEAT...UNTIL 循环执行结构内语句，然后再次执行语句前确定条件是否为真。这样，REPEAT...UNTIL 循环至少执行一次。而 WHILE...DO 循环中的语句可能从不执行。
	`pos := 0;`
	`While ((pos <= 100) & structarray[pos].value <> targetvalue)) do`
	`pos := pos + 2;`
	`String_tag.DATA[pos] := SINT_array[pos];`
	`END_WHILE;`

实例 2:

如果用户希望：	输入结构化文本程序：

将 SINT 数组中 ASCII 字符移入字符串标记。（在 SINT 数组中，每个元素保存一个字符。）当达到回车时停止。
1. 初始化 Element_number 值为 0。
2. 计算 SINT_array 中的元素数 （数组中包含 ASCII 字符），并将结果存入 SINT_array_size （DINT 标记）。
3. 如果 SINT_array[element_number] 的字符为 13 （回车对应的十进制数值），则停止循环。
4. 设置 String_tag[element_number] = SINT_array[element_number] 中字符。
5. element_number 加 1。 控制器检查 SINT_array 中下一个字符。
6. 设置 String_tag 的长度值为 element_number。 （该值记录当前 String_tag 中的字符数。）
7. 如果 element_number = SINT_array_size，则停止循环。（已执行到数组末尾但无回车。）
8. 跳到步骤 3。
	element_number := 0;
	SIZE(SINT_array, 0, SINT_array_size);
	While SINT_array[element_number] <> 13 do
	String_tag.DATA[element_number] := SINT_array[element_number];
	element_number := element_number + 1;
	String_tag.LEN := element_number;
	If element_number = SINT_array_size then
	EXIT;
	END_IF;
	END_WHILE;

REPEAT···UNTIL

REPEAT···UNTIL 循环连续执行语句，直至条件为真。

操作数：

```
REPEAT
    <statement>;
UNTIL bool_expression
END_REPEAT;
```

结构化文本

操作数：	类型：	格式：	输入：
布尔表达式	BOOL	标记 表达式	计算出布尔值（布尔表达式）的布尔型标记或表达式

重要

确定在一个扫描周期内重复循环执行的次数不能过多。

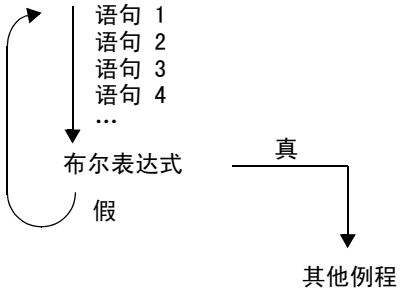
- 控制器在完成循环前无法执行例程中其他语句。
- 如果循环完成所用的时间超过了任务监控定时器的值，产生一个主故障。
- 考虑使用不同的结构，如 IF...THEN。

说明： 语法是：

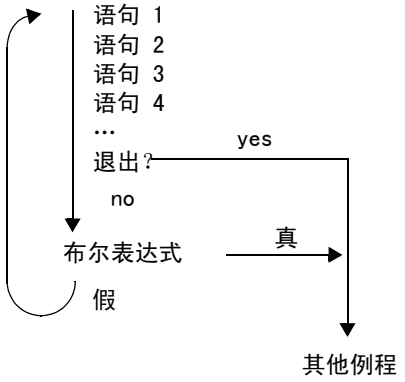
```
REPEAT
   <statement>;          <-- 当 bool_expression1 为假时，执行语句。
   IF bool_expression2 THEN
     EXIT;               <-- 如果需要提前退出循环，可使用其他语句，例如采用 IF...THEN 结构作为 EXIT 语句。
   END_IF;
UNTIL bool_expression1
END_REPEAT;
```

可选

以下图示了 REPEAT...UNTIL 循环如何执行，以及如何使用 EXIT 语句提前退出循环。



当 *bool_expression* 表达式为假，控制器仅执行 REPEAT…UNTIL 循环内的语句。



在条件为假前，使用 EXIT 语句退出循环。

算术状态标志： 不影响

故障条件：

出现主故障的情况：	故障类型：	故障代码：
结构循环太长	6	1

实例 1:

如果用户希望：	输入结构化文本程序：

REPEAT...UNTIL 循环首先执行结构内的语句，在再次执行结构内语句前，确定条件是否为真。
与 WHILE...DO 循环的区别在于 WHILE...DO 循环首先判断条件。如果条件为真，控制器执行循环内的语句。 这样，REPEAT...UNTIL 循环至少执行一次。 而 WHILE...DO 循环中的语句可能从不执行。
	`pos := -1;`
	`REPEAT`
	`  pos := pos + 2;`
	`UNTIL ((pos = 101) OR (structarray[pos].value = targetvalue))`
	`END_REPEAT;`

实例 2:

如果用户希望：	输入结构化文本程序：

将 SINT 数组中 ASCII 字符移入字符串标记。（在 SINT 数组中，每个元素保存一个字符。）当达到回车时停止。
1. 初始化 Element_number 值为 0。
2. 计算 SINT_array 中的元素数（数组中包含 ASCII 字符），并将结果存入 SINT_array_size（DINT 标记）。
3. 设置 String_tag[element_number] = SINT_array[element_number] 中字符。
4. element_number 加 1。控制器检查 SINT_array 中下一个字符。
5. 设置 String_tag 的长度值为 element_number。（该值记录当前 String_tag 中的字符数。）
6. 如果 element_number = SINT_array_size，则停止循环。（已执行到数组末尾但无回车。）
7. 如果 SINT_array[element_number] 的字符为 13（回车的十进制数值），则停止循环。否则，跳转至步骤 3。
	element_number := 0;
	SIZE(SINT_array, 0, SINT_array_size);
	REPEAT
	String_tag.DATA[element_number] := SINT_array[element_number];
	element_number := element_number + 1;
	String_tag.LEN := element_number;
	If element_number = SINT_array_size then
	EXIT;
	END_IF;
	Until SINT_array[element_number] = 13
	END_REPEAT;

注释

为使用户编写的结构化文本程序更具可读性，需添加注释。

- 用户通过注释可以使用清晰的语言来描述结构化文本程序的工作流程。
- 注释不影响结构化文本程序执行。

结构化文本注释下载到控制器内存，且可以上载。在用户结构化文本程序中添加注释：

添加注释：	使用以下格式：
单独一行	*//comment*
结构化文本程序的最后一行	(**comment**) */*comment*/*
一行结构化文本程序内	(**comment**) */*comment*/*
范围超过一行	(**start of comment. . . end of comment**) */*start of comment. . . end of comment*/*

例如：

格式：	实例：

//comment	**行起始**
	`//Check conveyor belt direction`
	`IF conveyor_direction THEN...`
	
	行尾
	`ELSE //If conveyor isn°Øt moving, set alarm light`
	`light := 1;`
	`END_IF;`

(**comment**)	`Sugar.Inlet[:=]1;(*open the inlet*)`
	
	`IF Sugar.Low (*low level LS*)& Sugar.High (*high level LS*)THEN...`
	
	`(*Controls the speed of the recirculation pump. The speed depends on the temperature in the tank.*)`
	`IF tank.temp > 200 THEN...`

*/*comment*/*	`Sugar.Inlet:=0;/*close the inlet*/`
	
	`IF bar_code=65 /*A*/ THEN...`
	
	`/*Gets the number of elements in the Inventory array and stores the value in the Inventory_Items tag*/`
	`SIZE(Inventory,0,Inventory_Items);`

程序梯形图

何时使用本章

使用本章：

- 开发梯形图例程的逻辑
- 在例程中输入逻辑

编程梯形图例程：

有关信息：	请参见页：
定义	7-1
编写梯形逻辑	7-4
输入梯级逻辑	7-7
分配指令操作数	7-9
检验例程	7-11

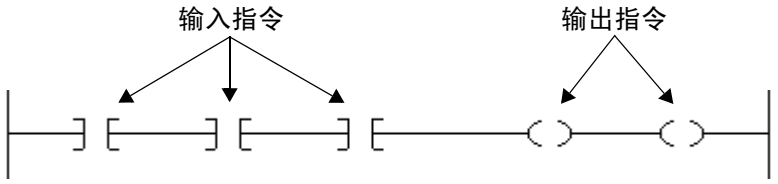
定义

指令

可以将梯形图组织成梯子上的各梯级，在每个梯级上放置指令。有两种基本类型指令：

输入指令：检查、比较或检查机器或进程中特定情况的指令。

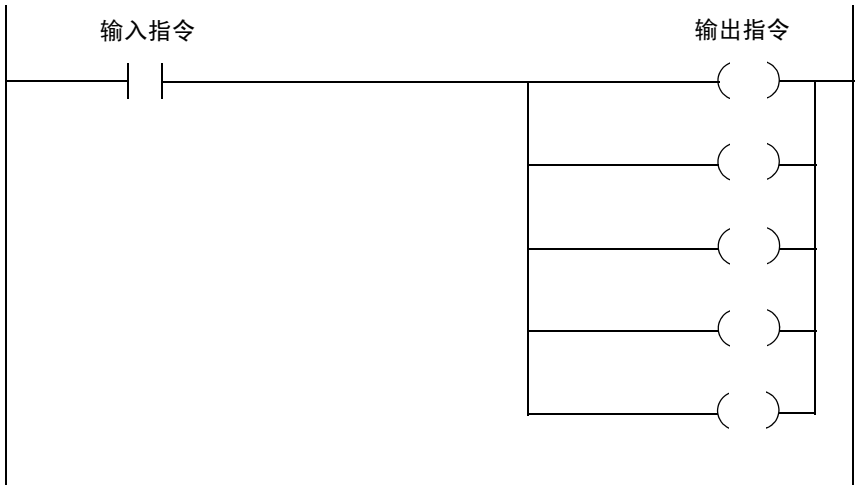
输出指令：采取一定操作的指令，例如打开设备、关闭设备、复制数据或计算值。



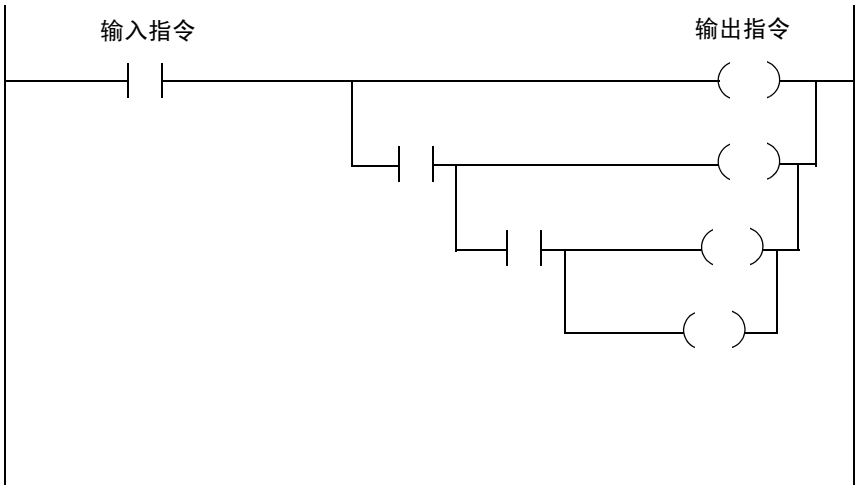
分支

分支是并行的两个或更多指令。

可以输入的并行分支层数没有限制。 此示例显示一个具有五层的并行分支。主层是第一个分支层，后面是其他四个分支。



您可以嵌套最多 6 层分支。此示例显示一个嵌套分支。底部输出指令在一个三层深的嵌套分支上。



具有复杂嵌套分支的大梯级导致必须滚动梯形图编辑器，并且打印逻辑时可能需要扩展多页。为更容易维护，请将逻辑分隔为多个较小的梯级。

梯级条件

控制器根据指令前的梯级条件 （梯级入条件）计算梯级指令。

只有输入指令影响梯级上的后续指令的梯级入条件。

- 如果输入指令的梯级入条件为 true，则控制器计算该指令并设置梯级出条件以匹配计算结果。
  - 如果指令计算结果为 true，则梯级出条件为 true。
  - 如果指令计算结果为 false，则梯级出条件为 false。
- 输出指令不更改梯级出条件。
  - 如果输出指令的梯级入条件为 true，则梯级出条件设置为 true。
  - 如果输出指令的梯级入条件为 false，则梯级出条件设置为 false。

编写梯形逻辑

选择所需指令

1. 分隔条件以检查要采取的操作。
2. 选择每个条件适合的输入指令以及每个操作适合的输出指令。

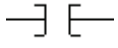
   若要选择特定指令，请参见：

   - *Logix5000 控制器基本指令参考手册*，出版物 1756-RM003
   - *Logix5000 控制器过程和驱动器指令参考手册*，出版物 1756-RM006
   - *Logix5000 控制器运动指令集参考手册*，出版物 1756-RM007

提示

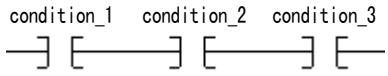
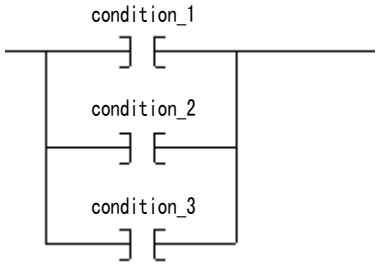
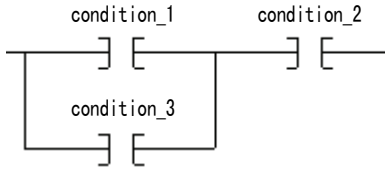
I/O 模块数据更新与逻辑的执行异步进行。如果在逻辑中多次引用一个输入，输入可能在不同引用中更改状态。如果需要输入对每个引用具有相同状态，请缓冲输入值并引用该缓冲标记。有关缓冲数据的更多信息，请参见第 1-8 页。

本章中的示例使用两个简单指令帮助您了解如何编写梯形图逻辑。从这些指令了解的规则适用于所有其他指令。

符号：	名称：	助记符：	说明：	
	检查是否关闭	XIC	关注一个数据位的输入指令。	
			如果该位：	**则指令 （梯级出条件）为：**
			on (1)	true
			off (0)	false
	输出激发	OTE	控制一个数据位的输出指令。	
			如果左边的指令 （梯级入条件）为：	**则指令将该位改为：**
			true	on (1)
			false	off (0)

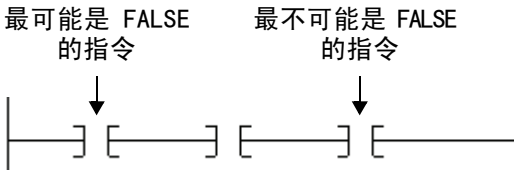
安排输入指令

确定如何在梯级上安排输入指令：

要在以下情况下检查多个输入条件：	安排输入指令：
必须满足所有条件才能行动 例如，if condition_1 AND condition_2 AND condition_3…	串行： 
必须满足任一条件才能行动 例如，if condition_1 OR condition_2 OR condition_3…	并行： 
以上的组合 例如， If condition_1 AND condition_2… or If condition_3 AND condition_2…	组合： 

提示

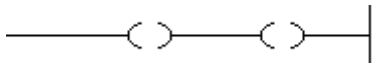
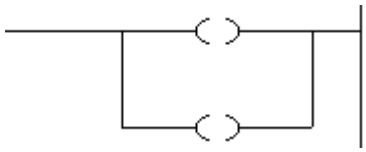
控制器执行梯级上的所有指令，不考虑它们的梯级入条件。要获得一系列指令的最优性能，请安排指令顺序，将最可能 false 的指令放在左边，最不可能 false 的指令放在右边。



控制器找到 false 指令后，将执行序列中的剩余指令，并将它们的梯级入条件设置为 false。通常，指令的梯级入条件为 false 而不是 true 时执行速度更快。

安排输出指令

至少将一个输出指令放在输入指令的右侧。每个逻辑梯级可以输入多个输出指令：

选项：	示例：
将输出指令放在梯级序列中 （串行）。	
将输出指令放在分支中 （并行）。	
将输出指令放在输入指令之间，只要梯级上的最后一个指令是输出指令。	

选择一个标记名称作为操作数

标记名称遵循下列格式：

对于：	指定：
标记	*tag_name*
较大数据类型的位号	*tag_name.bit_number*
结构成员	*tag_name.member_name*
一维数组的元素	*tag_name[x]*
二维数组的元素	*tag_name[x,y]*
三维数组的元素	*tag_name[x,y,z]*
结构内数组的元素	*tag_name.member_name[x]*
数组元素的成员	*tag_name[x,y,z].member_name*

其中

- *x* 是第一维度元素的位置。
- *y* 是第二维度元素的位置。
- *z* 是第三维度元素的位置。

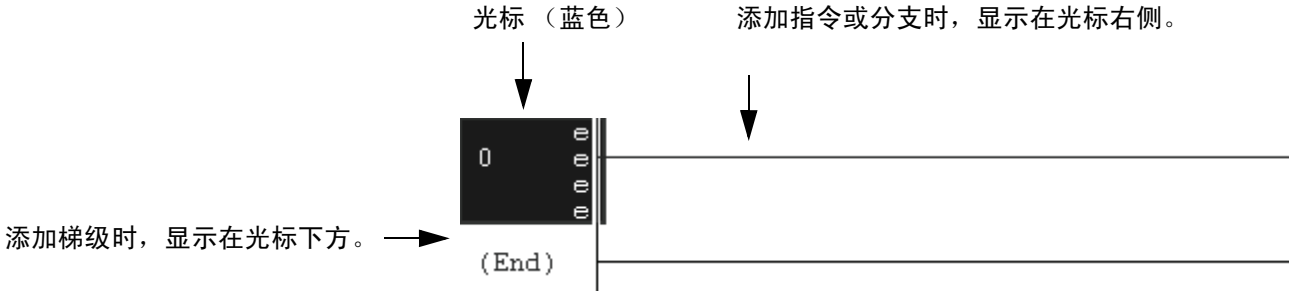
对于结构中的结构，添加一个 *.member_name*。

示例 选择一个标记名称作为操作数

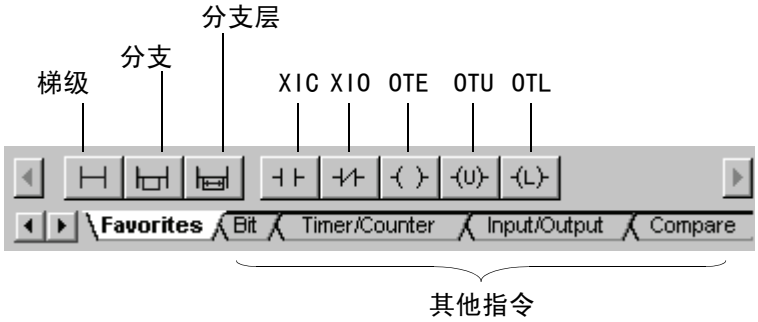
要访问：	标记名称类似：
machine_on 标记	machine_on
one_shots 标记的位号 1	one_shots.1
running_seconds 计时器的 *DN* 成员 （位）	running_seconds.DN
north_tank 标记的 *mix* 成员	north_tank.mix
recipe 数组中的元素 2 和 tanks 数组中的元素 1,1	COP Copy File Source recipe[2] Dest tanks[1,1] Length 1
north_tank 标记中 preset 数组的元素 2	CLR Clear Dest north_tank.preset[2] 0
drill 数组中元素 1 的 part_advance 成员	drill[1].part_advance

输入梯级逻辑

新例程包含准备好指令的梯级。



使用 Language Element （语言元素）工具栏添加梯形图元素。

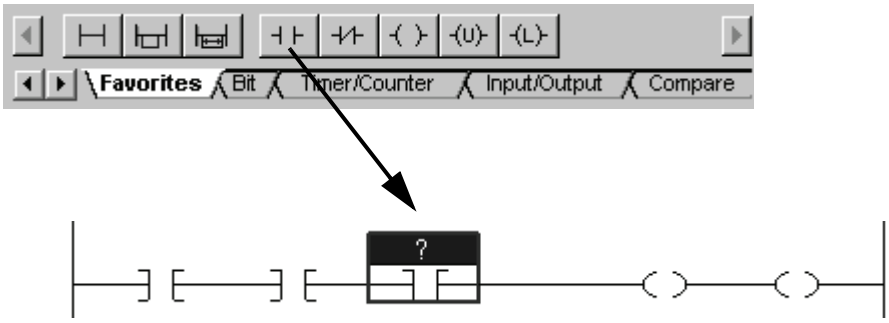


将元素追加到光标位置

1. 单击 （选择）要添加元素上方或左侧的指令、分支或梯级。
2. 在 Language Element （语言元素）工具栏上单击要添加元素的按钮。

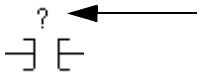
拖放元素

直接将元素的按钮拖动到所需位置。绿点显示有效放置位置 （放置点）。例如



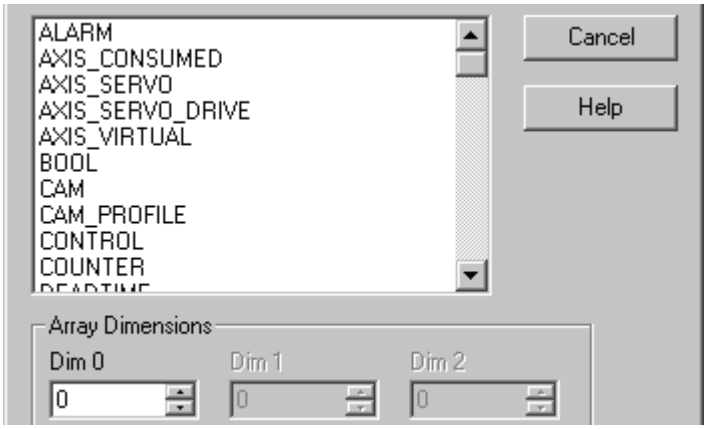
分配指令操作数

创建并分配新标记



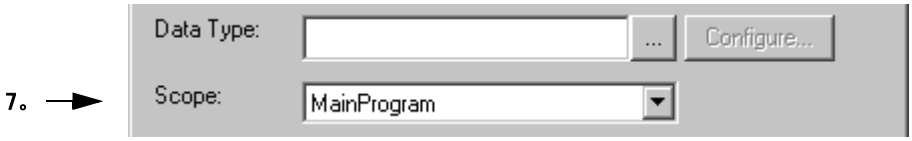
1. 单击指令的操作数区域。
2. 键入标记名称并按 [Enter]。
3. 右击标记名称并选择 New “tag_name”（新建 “标记名称”）。
4. 单击 ... 按钮。

5. 选择标记的数据类型。



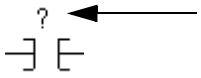
如果要将标记定义为数组，请键入每个维度的元素数量。

6. 单击 OK
7. 选择标记的范围。



8. 单击 OK

选择名称或现有标记



1. 双击操作数区域。

   文本输入框打开。

2. 单击 ▼

3. 选择名称：

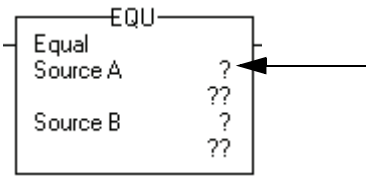
要选择：	执行：
标记、例程名称或类似类型的名称	单击名称。
标记	双击标记名称。
位号	A. 单击标记名称。 B. 在标记名称右侧，单击 ▼ C. 单击所需的位。

4. 按 [Enter] 或单击图上的其他位置。

从标记窗口拖动标记

1. 在标记窗口中找到标记。
2. 单击标记两次或三次直至突出显示。
3. 将标记拖动到指令上的位置。

分配立即 （常数）值



1. 单击指令的操作数区域。
2. 键入值并按 [Enter]。

检验例程

编程例程时，请定期检验您的工作。

1. 在 RSLogix 5000 窗口最顶部的工具栏上单击 
2. 如果任何错误列在窗口的底部：
   a. 要转至第一个错误或警告，请按 [F4]。
   b. 根据结果窗口的说明更改错误。
   c. 转至步骤 1.
3. 要关闭结果窗口，请按 [Alt] + [1]。

注释：

功能块编程

何时使用本章

使用本章：

- 组织功能块例程
- 为例程开发一个或多个功能块程序
- 向例程中输入功能块

要编程功能块例程：

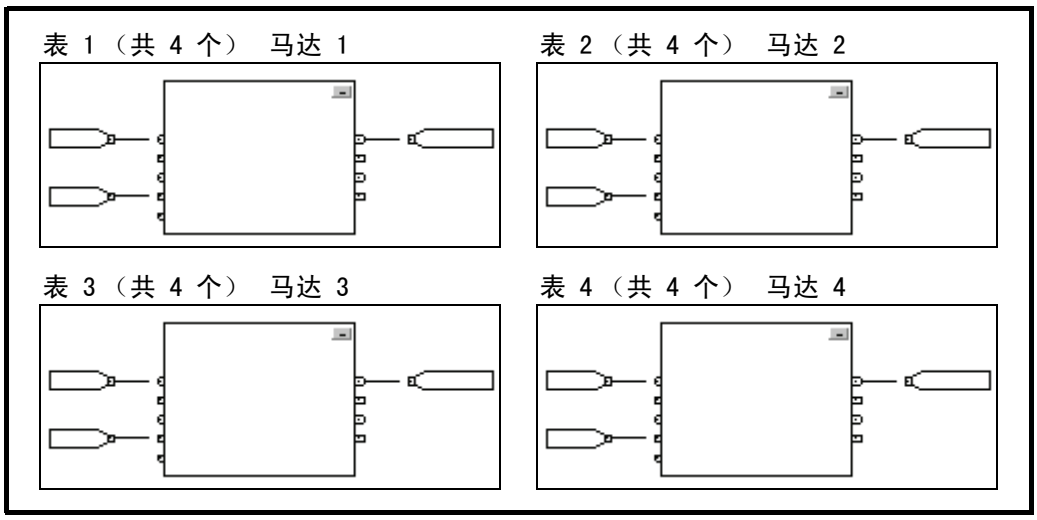
标识例程表

为了便于浏览功能块例程，将例程分成一系列表：

- 表帮助用户组织和查找功能块。它们不会影响功能块执行顺序。
- 执行例程时会执行所有表。
- 一般为一个设备 （马达、阀等）使用一个表

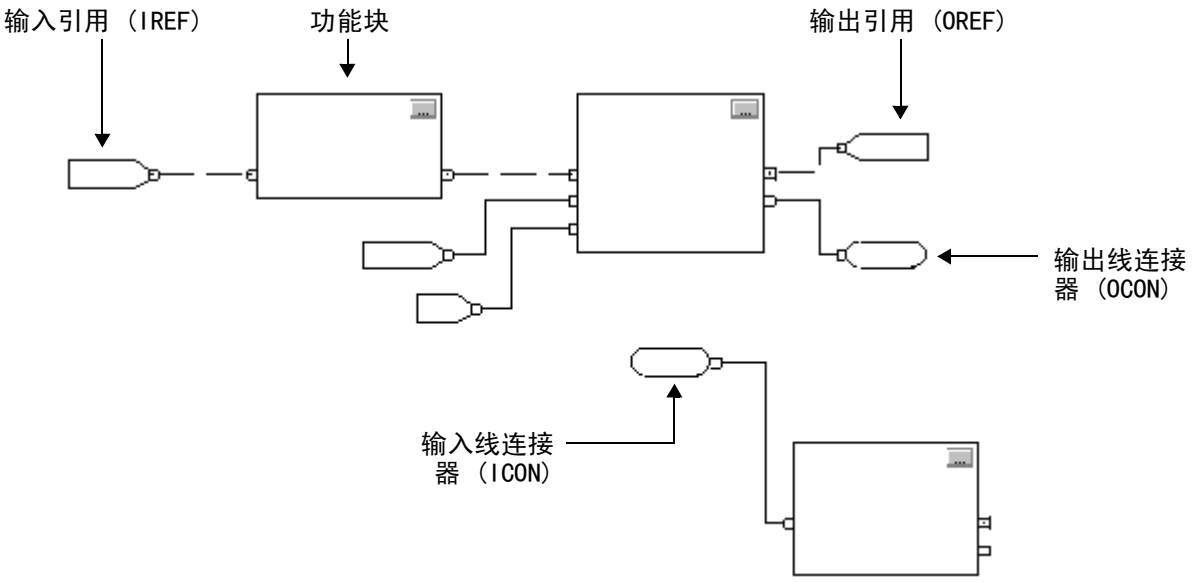
示例 标识例程表

马达控制例程



选择功能块单元

使用以下单元实现对设备的控制：



选择功能块单元：

如果用户要：	则选用：
由输入设备或标记提供值	输入引用 （IREF）
将值发送给输出设备或标记	输出引用 （OREF）
对一个或多个输入值执行操作并生成一个或多个输出值	功能块
在两个功能块间传送数据，当它们： • 位于同一个表内但距离较远 • 位于同一例程的不同表内	输出线连接器 （OCON） 和输入线连接器 （ICON）
分散数据到例程中多个点	单个输出线连接器 （OCON） 和多个输入线连接器 （ICON）

为单元选择标记名

每个功能块使用一个标记存储指令的配置和状态信息。

- 用户添加功能块指令时，RSLogix 5000 软件自动为功能块创建标记。用户可以使用此标记、重命名标记或分配一个不同的标记。
- 对于 IREF 和 OREF，用户必须创建一个标记或分配一个现有标记。

对于：	指定：
标记	*tag_name*
较大数据类型的位数	*tag_name.bit_number*
结构成员	*tag_name.member_name*
一维数组元素	*tag_name[x]*
二维数组元素	*tag_name[x,y]*
三维数组元素	*tag_name[x,y,z]*
结构内数组元素	*tag_name.member_name[x]*
数组元素成员	*tag_name[x,y,z].member_name*

其中：

x 是第一维元素的位置。

y 是第二维元素的位置。

z 是第三维元素的位置。

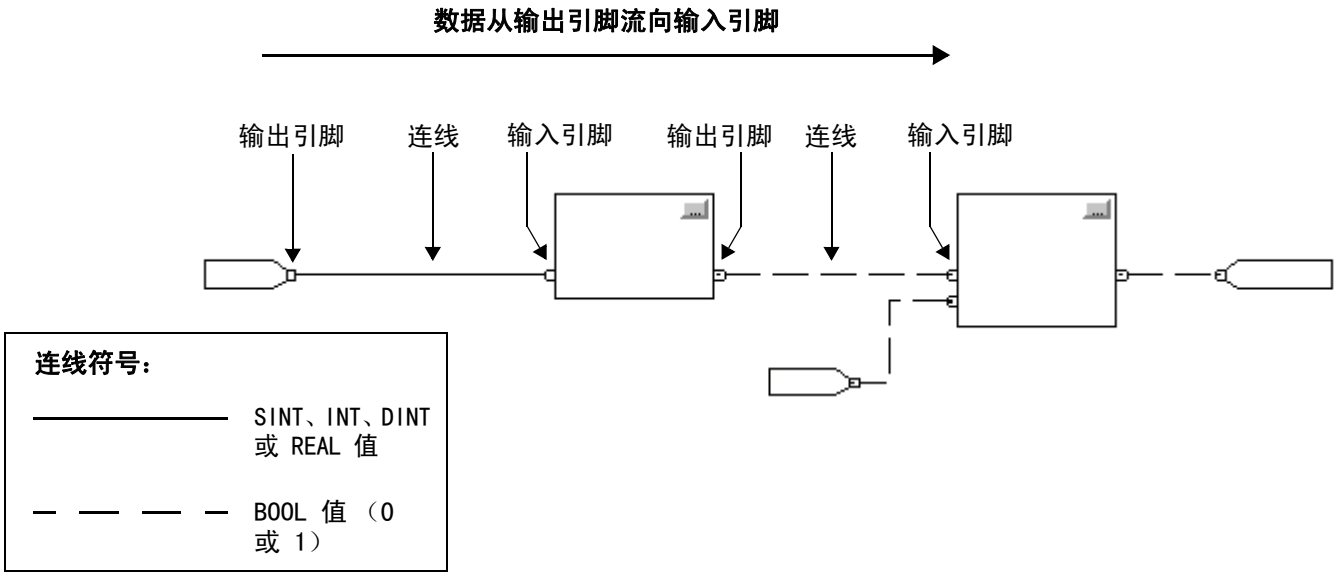
对于结构内的结构，添加一个额外的 *member_name*。

提示

I/O 模块数据更新与逻辑执行异步。 如果在逻辑中多次引用输入，输入可以在各个不同引用间更改状态。 如果您需要输入对每个引用的状态都相同，缓存输入值并引用该缓存标记。 关于缓存数据的更多信息，请参阅页 1-8。

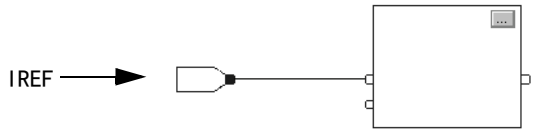
定义执行顺序

如有必要，用户可通过连接单元并指明输入 （反馈）线来指定执行顺序 （数据流）。功能块的位置不影响执行顺序。

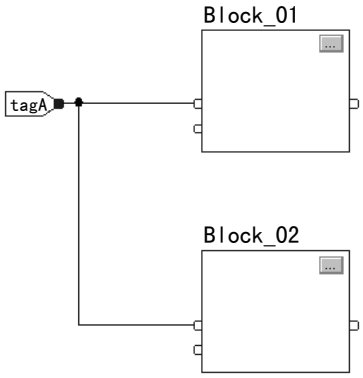


数据锁存

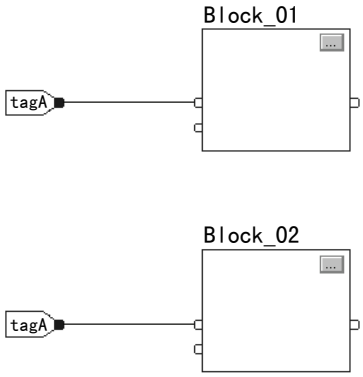
如果用户使用 IREF 指定功能块指令的输入数据，则在功能块例程扫描期间，IREF 内的数据被锁存。IREF 锁存来自程序范围和控制器范围标记的数据。控制器在每次扫描开始时更新所有的 IREF 数据。



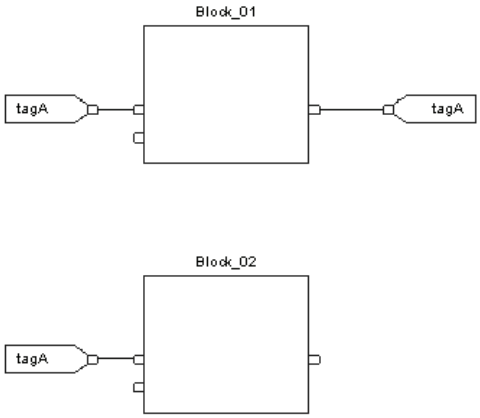
本例中，tagA 的值在开始执行例程时被存储。当 Block_01 执行时，使用该存储值。当 Blcock_02 执行时使用同一存储值。如果在例程执行期间 tagA 值改变，在 IREF 内的 tagA 的存储值直到下一次执行例程时才改变。



本例与上例相同。 tagA 的值只有在开始执行例程时才被存储。 在整个例程内都使用该存储值。



RSLogix 5000 软件版本 11 开始，用户可以在同一例程中的多个 IREF 和一个 OREF 中使用同一标记。由于 IREF 中标记的值在例程扫描过程中被锁存，因此在例程执行过程中，即使 OREF 包含不同的标记值，所有的 IREF 仍使用相同值。该例中，如果例程开始执行扫描时，tagA 值为 25.4，Block_01 将 tagA 的值变为 50.9，但 Block_02 执行此次扫描时，连接到 Block_02 的第二个 IREF 仍然使用 25.4。直至下次扫描开始，tagA 的新值 50.9 才能被此例程中的多个 IREF 使用。



执行顺序

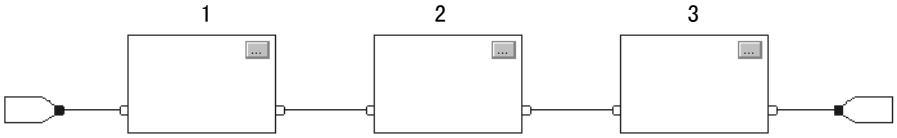
当用户进行如下操作时，RSLogix 5000 编程软件自动确定例程内功能块的执行顺序：

- 校验功能块例程
- 校验含有功能块例程的项目
- 下载含有功能块例程的项目

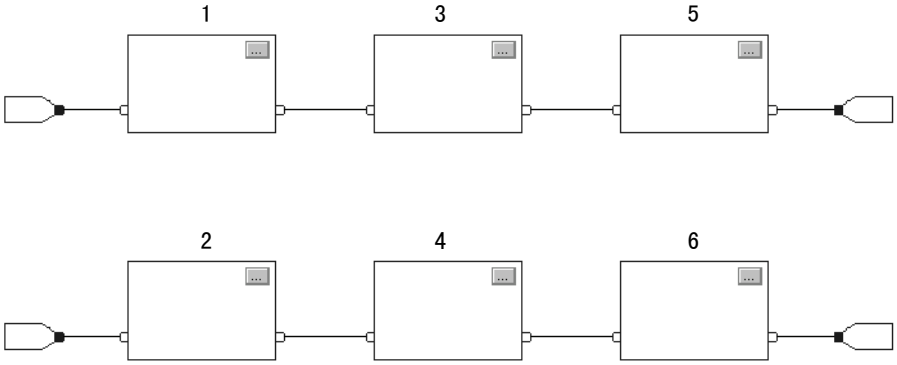
如有必要，用户可通过连接功能块并指定反馈线数据流来指定执行顺序。

如果功能块没有连接在一起，首先执行哪个功能块都没有关系。此时，功能块间无数据流。

如果用户顺序连接功能块，则执行顺序为从输入到输出。功能块的输入要求数据必须在控制器执行该功能块前可用。例如，功能块 2 必须在功能块 3 前执行，因为功能块 2 的输出是功能块 3 的输入。



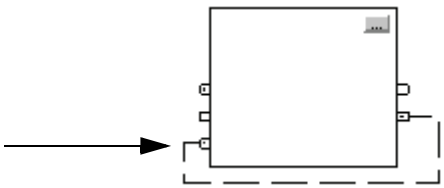
执行顺序只与连接的功能块相关。该实例没有问题，因为两组功能块没有连接在一起。在特定组内的功能块执行顺序只与该组内的功能块有关。



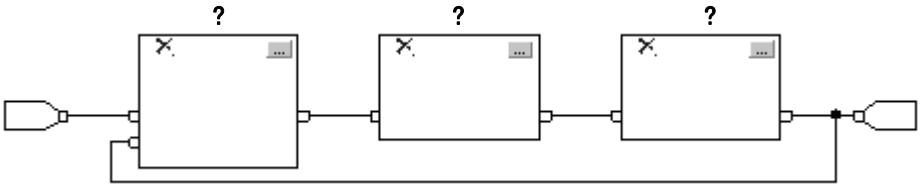
解析回路

创建功能块反馈回路需将块输出引脚连接到同一功能块的输入引脚上。该实例没有问题。该回路仅包含单个功能块，因此不受执行顺序的影响。

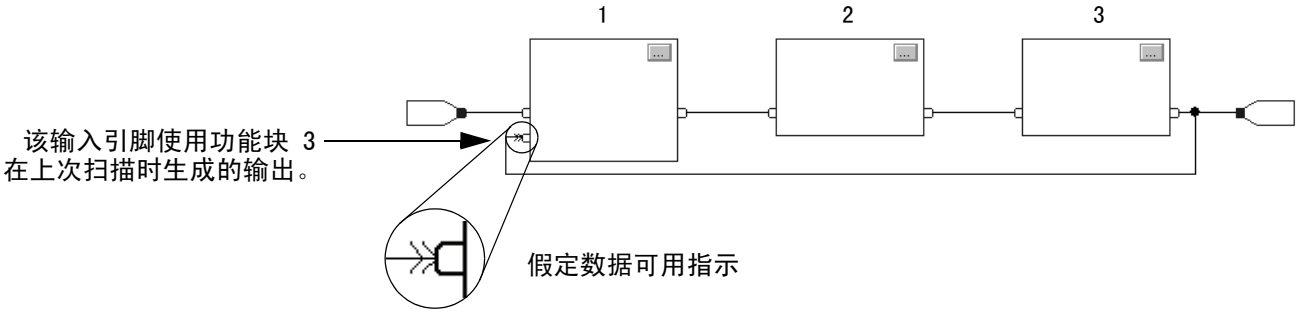
输入引脚使用功能块上次扫描生成的输出值。



如果一组功能块在同一回路内，则控制器无法确定哪个功能块最先执行。也就是说，它无法解析回路。

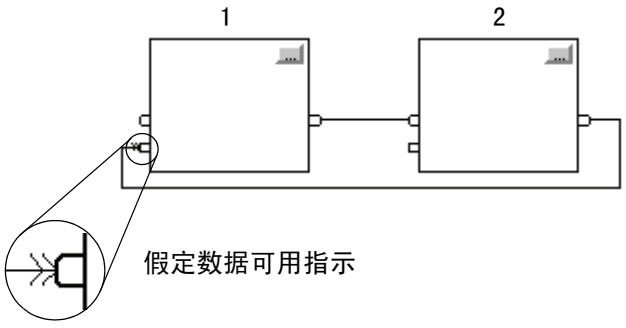
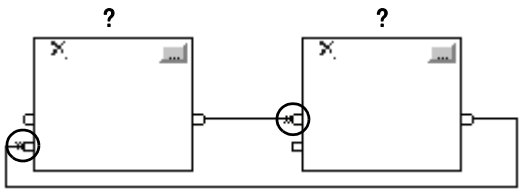


为确定先执行哪个功能块，可使用假定数据可用指示来标记用于创建回路 （反馈线）的输入线。本例中，功能块 1 使用功能块 3 在上次例程执行时生成的输出。



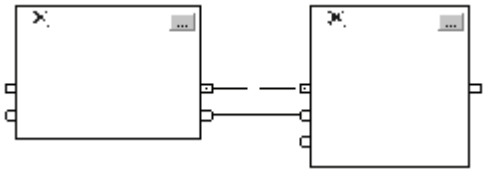
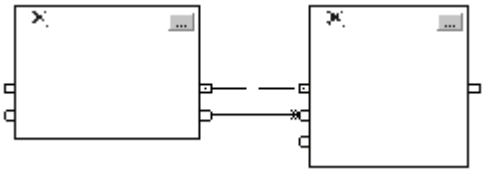
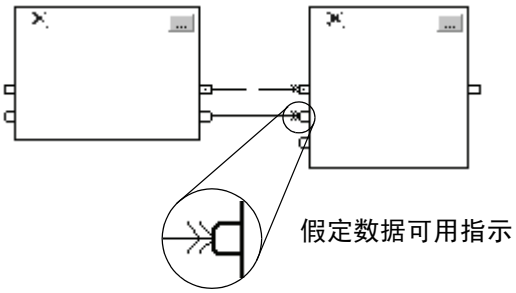
假定数据可用指示确定了回路内数据流向。箭头指示数据充当了回路中首个功能块的输入。

切忌使用假定数据可用指示来标记回路中所有连接线。

正确接法	错误接法
 假定数据可用指示确定了回路内数据流向。	 由于所有连接线均使用假定数据可用指示，控制器无法解析回路。

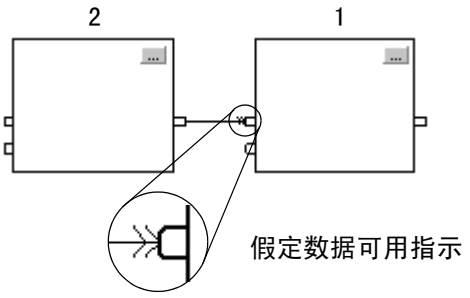
解析两功能块间数据流

如果用户使用两条或更多连接线连接两个功能块，则应为所有连接线使用相同的数据流指示。

正确接法	错误接法
	
所有连接线均不使用假定数据可用指示。	一条连接线使用假定数据可用指示，而另一条不使用。
	
两条连接线均使用假定数据可用指示。	

创建一个扫描周期的延迟

使用假定数据可用指示在两个功能块间生成一个扫描周期延迟。在本例中，功能块 1 首先执行。它使用例程上次扫描时功能块 2 生成的输出数据。



总结

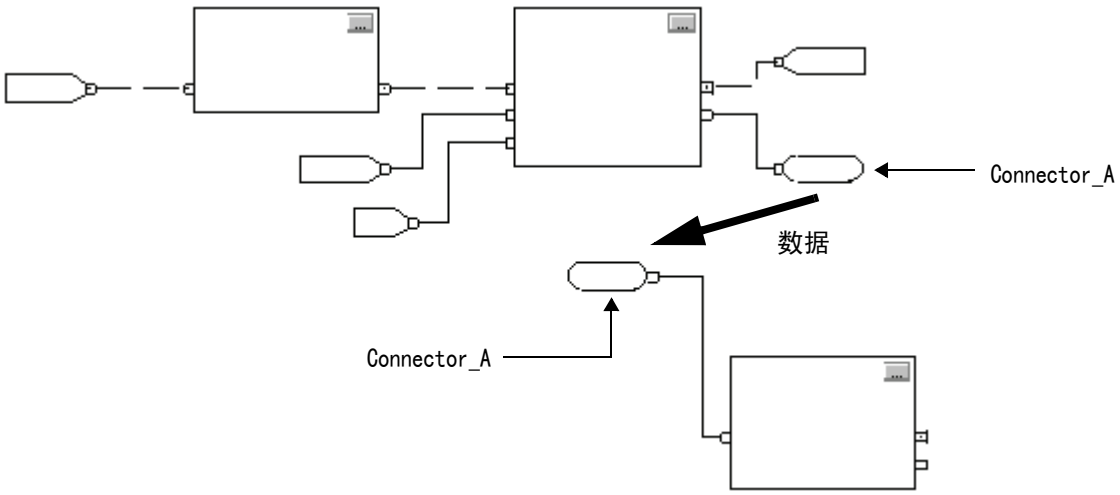
功能块例程按以下顺序执行：

1. 控制器锁存多个 IREF 内的所有数据值。
2. 控制器按顺序 （由功能块接线方式决定）执行其它功能块。
3. 控制器将输出写入所有 OREF。

识别连接器

跟数据线一样，连接器将数据从输出引脚传输到输入引脚。在下列情况下使用连接器：

- 用户要连接的单元位于同一例程不同表内
- 数据线很难与其它数据线或单元路由
- 用户要分散数据到例程中多个点



按下面的规则使用连接器：

- 每个 OCON 都需要唯一的名称。
- 对于每个 OCON，用户至少需要一个 ICON 与之对应 （即 ICON 与 OCON 的名称相同）。
- 多个 ICON 可以引用同一个 OCON。这允许用户分散数据到例程中多个点。

定义程序 / 操作员控制

一些指令支持程序 / 操作员控制。这些指令包括：

- 增强选择 (ESEL)
- 累加器 (TOT)
- 增强型 PID (PIDE)
- 斜坡 / 渗透 (RMPS)
- 离散型 2- 态设备 (D2SD)
- 离散型 3- 态设备 (D3SD)

程序 / 操作员控制允许用户由程序和操作员界面设备同时控制这些指令。在程序控制模式下，指令由程序输入控制；在操作员控制模式下，指令由操作员输入控制。

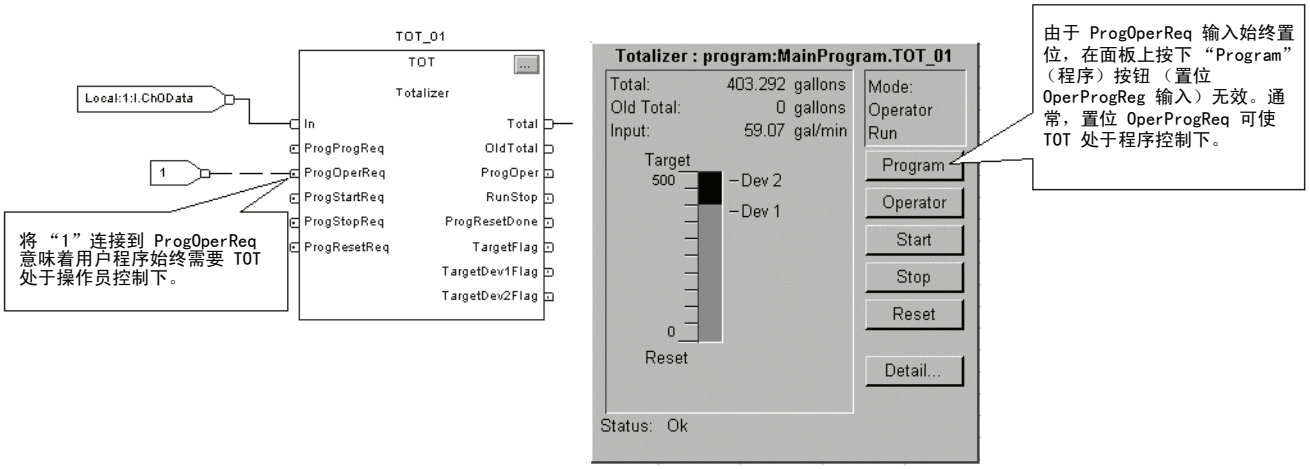
通过以下输入状态确定由程序还是操作员来控制：

输入：	说明：
.ProgProgReq	程序请求进入程序控制模式。
.ProgOperReq	程序请求进入操作员控制模式。
.OperProgReq	操作员请求进入程序控制模式。
.OperOperReq	操作员请求进入操作员控制模式。

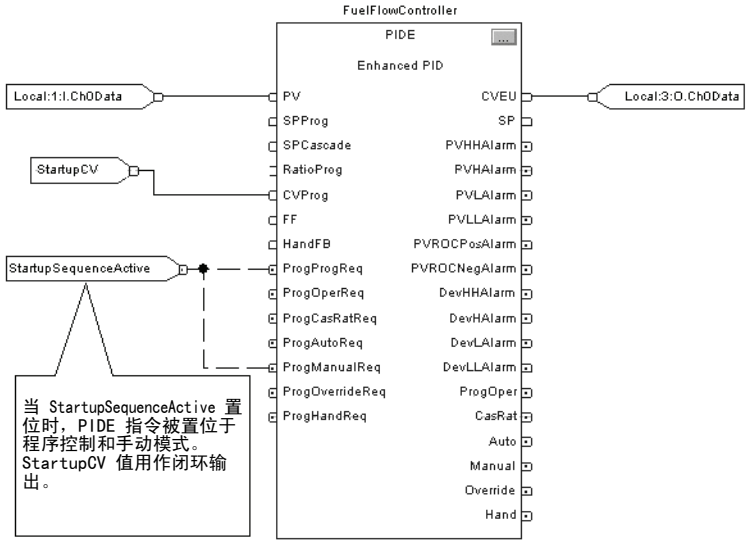
为了确定指令处于程序还是操作员控制模式，可检查 ProgOper 输出。如果 ProgOper 置位，则该指令处于程序控制模式；如果 ProgOper 清零，则该指令处于操作员控制模式。

如果两个输入请求位都置位，则操作员控制优先于程序控制。例如，如果 ProgProgReq 和 ProgOperReq 都置位，则指令进入操作员控制。

程序请求输入优先于操作员请求输入。因此可使用 ProgProgReq 和 ProgOperReq 输入 “锁定”指令在期望的控制模式下。例如，假定一条累加器指令始终处于操作员控制下，并且用户程序始终不能控制该累加器的运行或停止。这种情况下，用户可以将数值 1 连接到 ProgOperReq。这就防止了通过操作员界面设备置位 OperProgReq 将累加器置于程序控制模式。



同样，始终置位 ProgProgReq 能够 “锁定”指令处于程序控制下。这在自动设置起动顺序时很有用，由程序来控制指令的操作，不用担心操作员意外对指令进行控制。在本例中，在起动期间由程序置位 ProgProgReq 输入，然后在起动完成时清零 ProgProgReq 输入。一旦 ProgProgReq 输入清零，指令保持为程序控制直至接收到改变的请求。例如，操作员可以从面板置位 OperOperReq 输入，从而接管该指令的控制。下例给出如何锁定指令处于程序控制。



当指令执行时，该指令的操作员请求输入始终被清零。这样就通过仅设置所需模式请求位来使操作员界面能够使用这些指令。用户不必编程操作员界面复位请求位。例如，如果操作员界面置位 PIDE 指令的 OperAutoReq 输入，当 PIDE 指令执行时，它决定适当的响应并清零 OperAutoReq。

程序请求输入通常不由指令清零，因为它们通常被连接为指令的输入。如果指令清零这些输入，那么输入会再次被接入的输入置位。用户可能会遇到这种情况，在希望程序请求被指令清零的同时，还希望使用其它逻辑置位程序请求。在这种情况下，用户可以置位 ProgValueReset 输入，且在指令执行时，指令始终清零程序模式请求输入。

本例中，另一例程中的梯形图逻辑用于在按下按钮时单步锁存 PIDE 指令的 ProgAutoReq。由于 PIDE 指令自动清零程序模式请求，用户不必编程梯形图逻辑在例程执行后清零 ProgAutoReq，并且每次按下按钮时，PIDE 指令只接收到一个转到 Auto 的请求。

当按下 TIC101AutoReq 按钮时，单步锁存 PIDE 指令 TIC101 的 ProgAutoReq。TIC101 已被配置为 ProgValueReset 输入置位，所以 PIDE 指令执行时自动清零 ProgAutoReq。

添加表

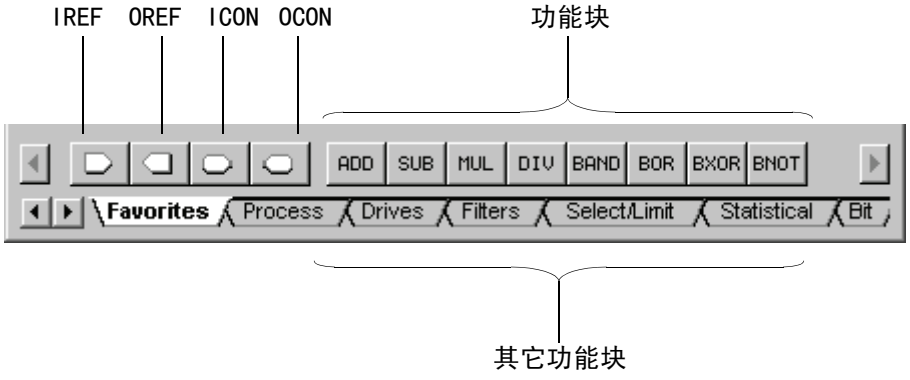
要向功能块例程中添加表：

1. 单击 

2. 输入表的说明 （超过 50 个字符）。

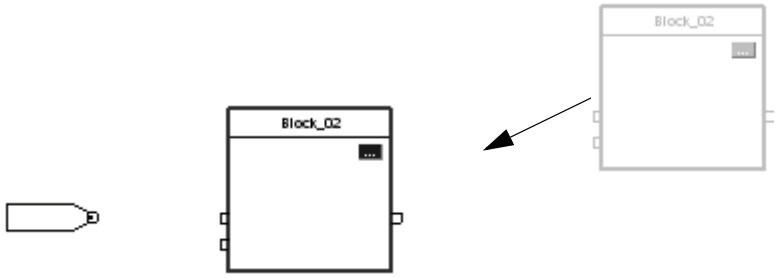
添加功能块单元

使用 Language Element 工具栏向例程中添加功能块。

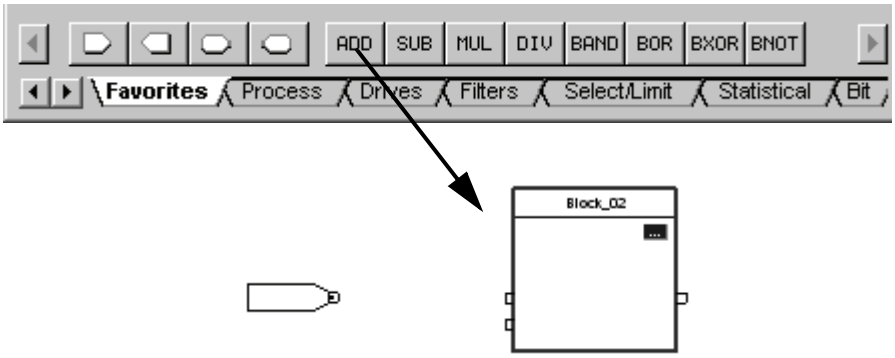


要添加单元，执行下列操作：

1. 在 Language Element 工具栏中，单击用户希望添加的单元。
2. 将单元拖到所需位置。



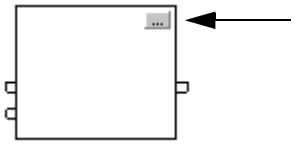
用户也可以直接拖动单元按钮到所需位置。



连接单元

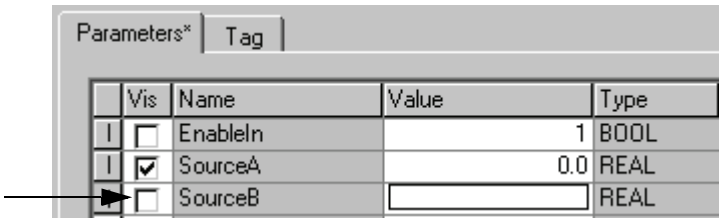
显示或隐藏引脚

用户添加功能块指令时，功能块上出现一系列缺省参数的引脚。其余引脚为隐藏状态。要显示或隐藏一个引脚，执行下列操作；



1. 单击[...]功能块按钮。
2. 清除或选中引脚的 Vis 复选框：

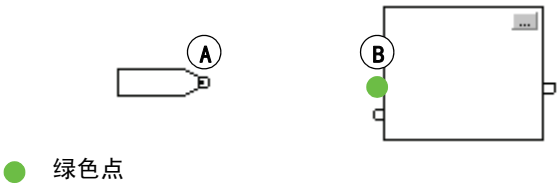
如果用户希望：	则：
隐藏一个引脚	清除 （取消选中）Vis 复选框。
显示一个引脚	选中其 Vis 复选框。



3. 单击 OK

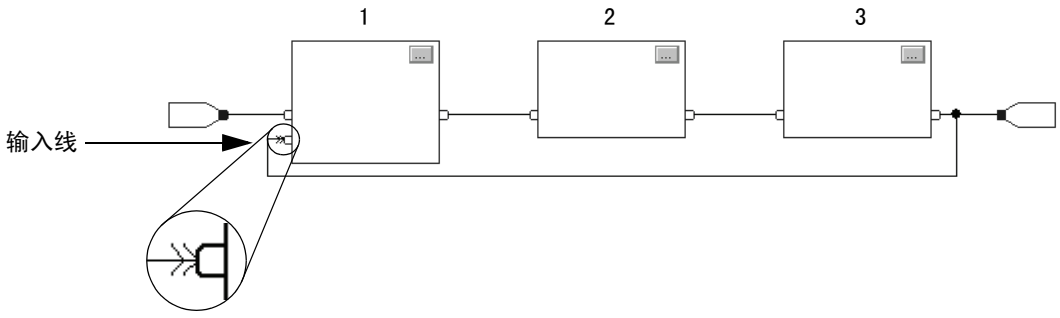
将单元连接在一起

要将两个单元连接在一起，单击第一个单元的输出引脚，然后单击第二个单元的输入引脚。绿色点表明连接点有效。

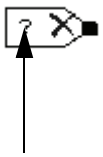


使用假定数据可用指示标记连接

要确定输入连线，右击连线并选择假定数据可用。



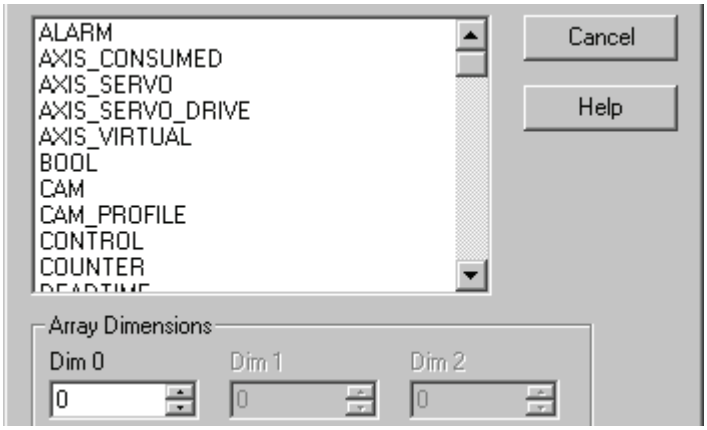
分配一个标记



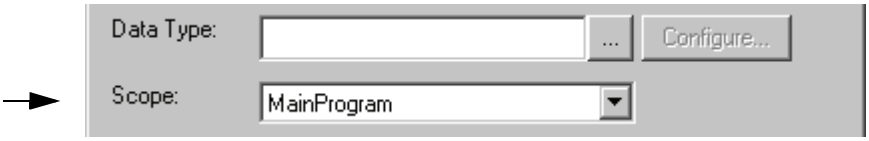
创建并分配一个新标记

1. 双击操作区。
2. 输入标记的名称并按 [Enter] 回车键。
3. 右键单击该标记名，选择 New “tag_name”。

4. 单击 ... 按钮。

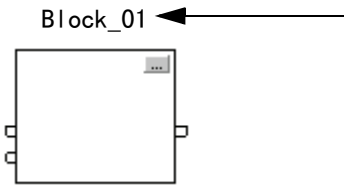


5. 选择标记的数据类型。
6. 如果标记类型为数组，输入每个维的元素个数。
7. 单击 OK
8. 选择标记的范围。



9. 单击 OK

分配现有标记



1. 双击操作区。
2. 单击 ▼
3. 选择标记：

要选择：	请执行下列操作：
标记	双击标记名称。
位号	A. 单击标记名称。 B. 在标记名的右边，单击 ▼ C. 单击所需位。

4. 按 [Enter] 回车键或单击图表上的不同位置。

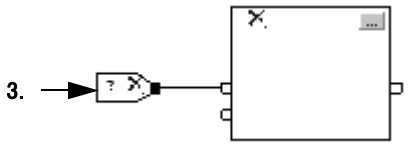
分配一个立即数 （常数）

要向输入参数中分配一个常数值而不是标记值，用户需要作出以下选择：

如果用户希望：	则：
在图表和报告中显示数值	使用一个 IREF
能够不编辑例程在线改变数值	在功能块标记中输入数值

使用一个 IREF

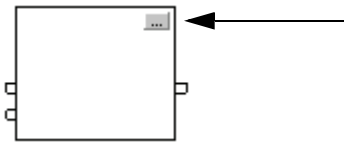
1. 添加一个 IREF。
2. 向获取数值的输入引脚写入 IREF。



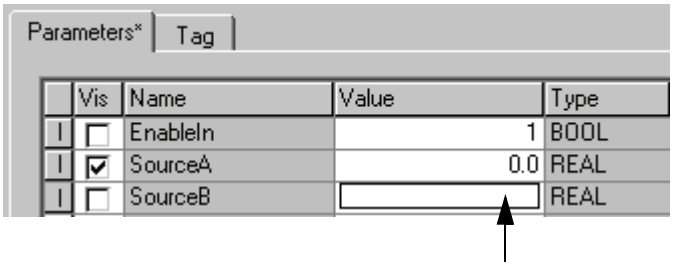
3. 双击 IREF 操作区。
4. 输入数值并按 [Enter] 回车键。

在功能块标记中输入数值

要在接线连接到其引脚时为参数分配数值：



1. 单击 ... 功能块按钮。

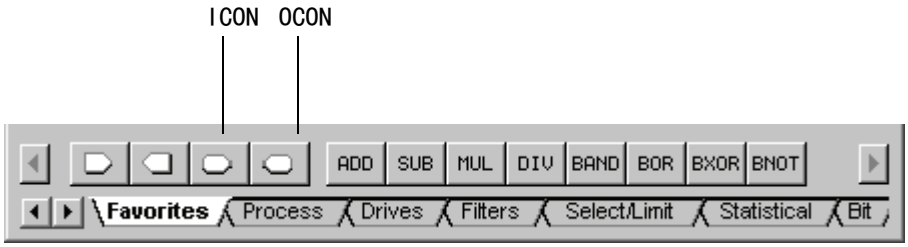


2. 输入数值。

3. 单击 OK

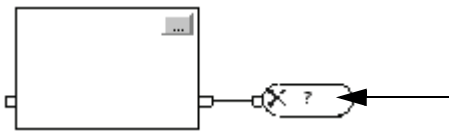
使用 OCON 和 ICON 连接功能块

使用 OCON 或 ICON 在表之间或在接线复杂的情况下传输数据：



添加一个 OCON

1. 添加一个输出线连接器 (OCON)，然后将其置于给出数值的输出引脚附近。

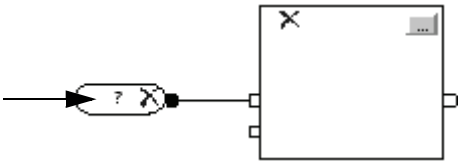


2. 将 OCON 连接到输出引脚。

3. 双击 OCON 操作区。

4. 输入能够标识连接器的名称，然后按 [Enter] 回车键。

添加一个 ICON



1. 添加一个输入连接器 (ICON)，然后将其置于从对应的 OCON 获取数值的输入引脚附近。
2. 将 ICON 连接到输入引脚。
3. 双击 ICON 操作区。
4. 选择向连接器提供数值的 OCON 名称，然后单击图表的空白位置

校验例程

用户编程例程时，应定期校验工作：

1. 在 RSLogix 5000 窗口的最顶端工具栏，单击 
2. 如果在窗口底部列出任何错误：
   a. 按 [F4] 键转到第一个错误或警告。
   b. 按照 Result （结果）窗口中的说明纠正错误。
   c. 转到步骤 1.
3. 按 [Alt] + [1] 关闭 Results （结果）窗口。

与其他设备通信

何时使用此章节

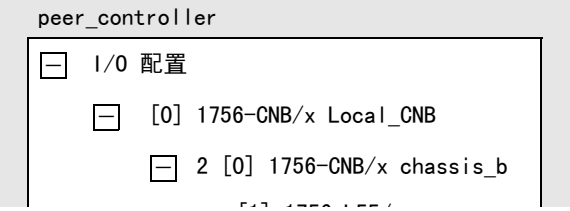
使用此章节规划用户控制器和 I/O 模块或其他控制器之间的通信。

相关信息：	请参阅页：
连接	9-1
生成和使用标记	9-9
执行消息 (MSG) 指令	9-17
获取或设置非连接缓冲器数量	9-23
在 INT 和 DINT 之间转换	9-26

连接

Logix5000 控制器使用**连接** 与其他设备在大多数情况下通信，但不是所有情况。

术语：	定义：
连接	两个设备间的通信链路，如控制器和 I/O 模块、PanelView 终端或其他设备。. 连接是资源分配，与非连接的消息相比，能够为设备提供更可靠的通信。单个控制器的连接数有限。 用户通过配置控制器与其他系统设备之间的通信，间接确定控制器连接。这些通信类型使用以下连接： • I/O 模块 • 生成和使用标记 • 某些类型的消息 (MSG) 指令 （并非所有类型都使用连接）

术语：	定义：
请求数据包间隔 (RPI)	RPI 指定连接中数据更新时间。例如，输入模块以用户分配给模块的 RPI 周期向控制器发送数据。 • 一般情况下，用户按照毫秒 (ms) 配置 RPI。 范围为 0.2 ms （200 微秒）至 750 ms。 • 如果 ControlNet 网络连接设备，RPI 在 ControlNet 网络数据流中保留时槽。 此时槽的计时与精确 RPI 值可能不相符，但是控制系统至少可以按指定的 RPI 速率传输数据。
路径	路径描述连接到达目标的传输路线。 一般情况下，用户在向控制器的 I/O 配置文件夹添加设备时，自动定义连接路径。 

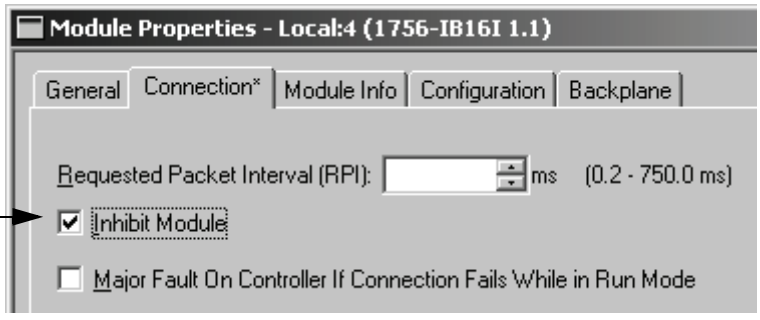
禁止连接

注意

禁止模块会导致与该模块的连接被切断，并阻止 I/O 数据的通信。

在某些情况下，例如：开始运转一个系统时，禁止掉控制系统的一部分，然后当用户连接好整个控制系统时再启用它们是很有用的。控制器允许用户禁止单独的模块或一组模块，阻止控制器与这些模块通信。

禁止与模块通信。



当用户创建一个 I/O 模块时，默认是不被禁止的。用户可以更改一个单独模块的属性来禁止模块。

如果用户希望：	则：
与模块通信	不要禁止模块
禁止与模块通信	禁止模块

用户禁止通信转发模块时，如 1756-CNB 或 1756-DHRIO 模块，控制器关闭与转发模块和所有依赖它的模块之间的连接。禁止通信转发模块使用户能够禁用 I/O 网络的整个分支。

当用户选择禁止模块时，在控制器管理器⚠ 的模块上 会显示一个黄色的提示符号。

如果用户：	并且：	并且：	则：
离线	→	→	禁止状态存储在工程中。当用户下载工程时，模块仍然是禁止的。
在线	与模块连接时禁止模块	→	模块连接会断开。模块输出转到最后一种配置程序模式。
	用户禁止模块时，还没有与它建立连接 （可能由于错误条件或故障）	→	该模块会被禁止。模块状态消息会变成指示模块被禁止了且没有故障。
	取消禁止模块 （清除复选框）	没有故障情况发生	与模块建立连接，而且使用用户为模块创建的配置消息自动识别模块 （如果控制器是所有者控制器）。如果控制器配置为只听，则不可以重新配置模块。
		发生故障情况	不会与模块建立连接）。模块的状态消息会变成指示模块的故障情况。

在逻辑程序中禁止或取消禁止一个模块：

1. 使用 GSV （获取系统值）指令读取该模块的模式属性。
2. 设置或清除位 2：

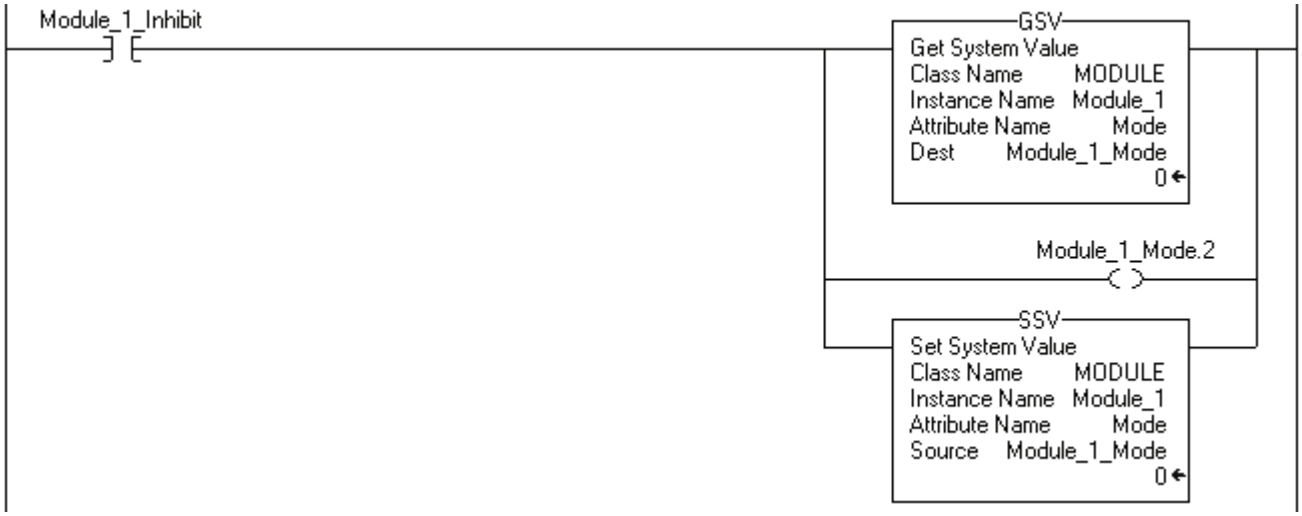
如果用户希望：	则：
禁止模块	设置位 2。
取消禁止模块	清除位 2

3. 使用 SSV （获取系统值）指令把模式属性写回到模块。

示例 禁止连接

如果 Module_1_Inhibit = 1，则禁止名为 Module_1 的 I/O 模块的操作：

1. GSV 指令设置 Module_1_Mode 为模块的模式属性值。
2. OTE 指令设置 Module_1_Mode 的位 2 为 1。意思是禁止连接。
3. SSV 指令设置模块的模式属性为 Module_1_Mode。



管理 连接失败

注意

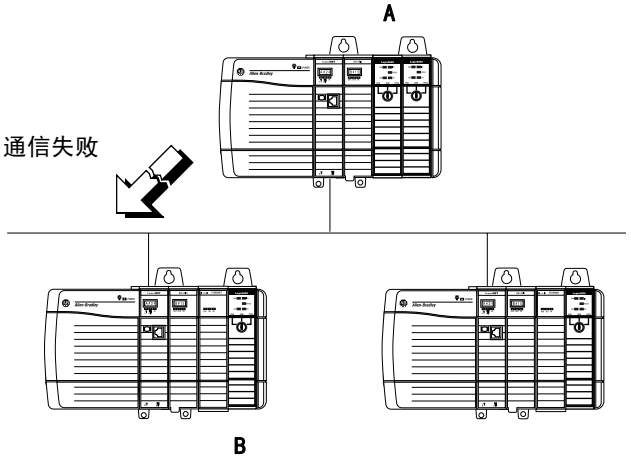
输出与控制输入的最后一次、非故障状态对应。要防止可能的伤亡和机器损坏，确保不会产生不安全的操作。在模块与控制器断开连接时，配置生成控制器主故障的关键 I/O 模块。或者，监视 I/O 模块的状态。

如果控制器断开与模块的通信，则不会更新来自该设备的数据。出现这种情况时，该逻辑确定数据是否正确。

示例

断开通信

控制器 B 需要来自控制器 A 的数据。如果控制器直接的通信失败，则控制器 B 继续操作最后一次从控制器 A 接收的数据。

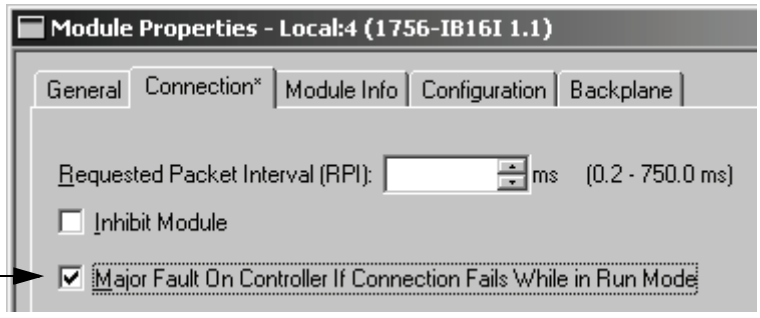


如果与控制器 I/O 配置中设备的通信在 100 ms 内未响应，则通信超时。 如果出现这种情况，用户可以作出以下选择：

如果用户希望控制器：	则：
出现错误 （主故障）	配置出现的主故障
继续操作	监视模块状态

配置出现的主故障

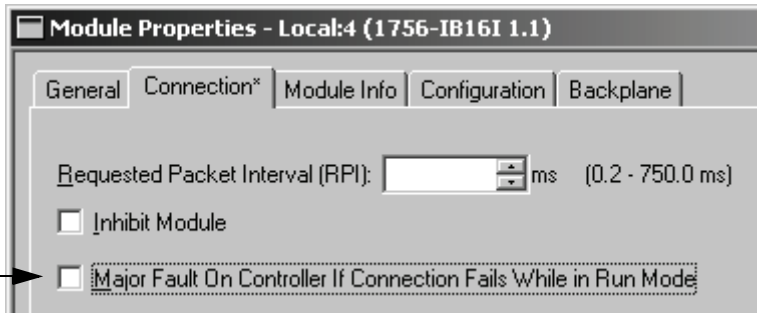
可以配置模块在失去与控制器的连接时在控制器中生成主故障。这会中断逻辑的执行，并执行控制器错误处理程序。 如果控制器错误处理程序未清除错误，则关闭控制器。



如果连接超时，控制器会生成主故障。

监视模块状态

如果用户未配置出现的主故障，则需要监视模块状态。如果一个模块与控制器断开连接，则输出进入所配置的故障状态。控制器和其它 I/O 模块继续基于来自模块的旧数据操作。



如果连接超时，则在不调用控制器主故障情况下继续操作。

如果与模块通信超时，则控制器上出现下面的警告：

- 控制器前面 I/O LED 绿色闪烁。
- ⚠ 在 I/O 配置文件夹和已经超时的其它设备上显示。
- 生成模块故障代码，用户可以通过以下途径访问：
  - 模块的模块属性窗口
  - GSV 指令

要监视连接状态，使用 GSV （获取系统值）指令监视控制器或特定模块的 MODULE 对象：

如果用户希望：	获取该属性：	数据类型：	说明：
确定与任何设备的连接是否超时	LED 状态	INT 为了方便，使用 DINT 作为目标数据类型。	指定控制器前面 I/O LED 的当前状态。 **注释：**用户不输入该属性的实例名。该属性适用于所有模块。
			值： **意思：**
			0 **LED 熄灭：**控制器未配置模块对象 （项目管理器中的 I/O 配置部分没有模块）。
			1 红色闪烁：没有正在运行的模块对象。
			2 绿色闪烁：至少有一个 模块对象不在运行。
			3 恒定绿色： 所有模块对象都在运行。
确定与特定设备的连接是否超时	错误代码	INT 为了方便，使用 DINT 作为目标数据类型。	如果模块出现错误，会有一个指示错误的号码。 在实例名中，选择用户希望监视其连接的设备。确保在工程 I/O 配置文件夹中为设备分配名。

示例 监视模块状态

GSV 指令连续设置 I_O_LED_Status （DINT 标记）为控制器的 I/O LED 状态。

如果 I_O_LED_Status = 2，则至少有一个模块的通信超时 （错误）。GSV 指令设置 Module_3_Fault_Code 为 Module_3 的错误代码。

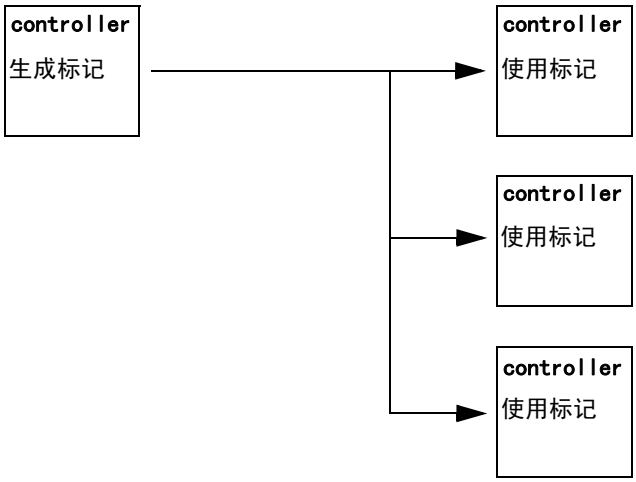
如果 Module_3_Fault_Code. 不等于 0，则与 Module_3 的通信超时 （错误）。 OTE 指令设置 Module_3_Faulted = 1。

生成和使用标记

在控制器之间传输数据 （发送或接收数据），用户可以：

如果数据：	则：	请参阅：
需要在用户指定的间隔内定期传输	生成和使用标记	本部分
在用户应用过程中出现特定条件时发送	执行消息 (MSG) 指令	页 9-17

Logix5000 控制器允许用户生成 （广播）和使用 （接收）系统共享标记。



术语：	定义
生成标记	控制器生成的标记，用于其它控制器使用。允许多个控制器同时使用 （接收）数据。一个生成标记在不使用逻辑的条件下，向一个或多个使用标记（使用者）发送数据 。生成标记按照使用标记的 RPI 速率发送数据。
使用标记	接收生成标记数据的标记。使用标记必须与生成标记具有相同的数据类型（包括数组维数）。使用标记的 RPI 速率确定数据更新的周期。

支持生成标记 / 使用标记的控制器和网络

下面的控制器和网络组合支持生成和使用标记。

该控制器：	可以通过网络生成和使用标记：		
	背板	ControlNet	EtherNet/IP
SLC 500		✔	
PLC-5		✔	
CompactLogix[1]		✔	✔
ControlLogix	✔	✔	✔
DriveLogix		✔	✔
FlexLogix		✔	✔
SoftLogix5800		✔	

[1] ControlNet 使用 1769-L32C 或 1769-L35CR 控制器。EtherNet/IP 使用 1769-L32E 或 1769-L35E 控制器。

对于两个共享生成或使用标记的控制器，必须同时连接到相同的网络（如 ControlNet 或 Ethernet/IP 网络）。用户不能在两个网络上转发生成标记和 使用标记。

生成或使用标记的连接需要

重要 如果一个使用标记的连接失败，那么此远程控制器上使用的所有标记都不会接收新的数据。

生成标记和使用标记都需要连接。随着增加可使用生成标记的控制器的数量，同时也在减少控制器用于其它操作 （如通信和 I/O）的连接数量。

每个生成或使用标记都使用下面的连接：

标记类型：	使用的连接数量
生成标记	*使用者数量* + 1
使用标记	1

示例

生成或使用标记的连接需要

- 一个为 5 个控制器 （使用者）生成标记的 FlexLogix 控制器使用 6 个连接。
- 为一个控制器生成 4 个标记的 ControlLogix 控制器使用 8 个连接：
  – 每个标记使用 2 个连接 （1 使用者 + 1 = 2）。
  – 每个标记 2 个连接 x 4 标记 = 8 个连接
- 从一个控制器使用 4 个标记使用 4 个连接（每个标记的 1 个连接 x 4 个标记 = 4 个连接。

为生成或使用数据组织标记

用户为生成或使用数据 （共享数据）组织标记时，遵循下列原则：

<table>
<tr><th>原则：</th><th colspan="4">详细信息：</th></tr>
<tr><td>创建控制器范围标记。</td><td colspan="4">用户仅可以共享控制器范围标记。</td></tr>
<tr><td>使用下面一种数据类型：
• DINT
• REAL
• DINT 或 REAL 数组
• 用户定义</td><td colspan="4">• 要共享其它数据类型，创建包括所需数据的用户定义数据类型。
• 生成标记与相应的使用标记或标记具有相同的数据类型。</td></tr>
<tr><td rowspan="6">要与 PLC-5C 控制器共享标记，请使用用户定义的数据类型。</td><td>要：</td><td>数据类型：</td><td colspan="2">则：</td></tr>
<tr><td rowspan="3">生成</td><td>整型</td><td colspan="2">创建用户定义数据类型，包括偶数个元素的 INT 数组，如 INT[2]。 （当用户生成 INT 时，必须生成两个或更多。）</td></tr>
<tr><td>只有一个 REAL 值</td><td colspan="2">使用 REAL 数据类型。</td></tr>
<tr><td>超过一个 REAL 值</td><td colspan="2">创建包括 REAL 数组的用户定义的数据类型。</td></tr>
<tr><td rowspan="3">使用</td><td rowspan="3">整型</td><td colspan="2">创建包括下面成员的用户定义数据类型：</td></tr>
<tr><td>数据类型：</td><td>说明：</td></tr>
<tr><td>DINT
INT[x]，其中 x 是来自 PLC-5C 控制器的数据输出大小。（如果用户仅使用一个 INT，则省略 x。）</td><td>状态
PLC-5C 控制器生成的数据</td></tr>
<tr><td>将标记大小限制为 ≤ 500 字节。</td><td colspan="4">• 如果用户必须传输超过 500 字节的标记，则创建逻辑以传输数据包。请参阅第 10 章。
• 如果用户在 ControlNet 网络生成标记，则标记不应超过 500 字节。请参阅第 9-12 页上的 ″调整带宽限制″。</td></tr>
</table>

原则：	详细信息：
使用最高的允许 RPI 速率。	如果控制器在 ControlNet 网络上使用标记，则使用 ControlNet 网络更新时间 (NUT) 的二次乘数。 例如：如果 NUT 为 5 毫秒，则使用 RPI 速率为 5、10、20、40 等 • • 。
组合发送到相同控制器的数据。	如果用户要为相同控制器生成多个标记： • 将数据组合为一种或多种用户定义的数据类型。（这比单独生成每个标记使用的连接数少。） • 按照相同的更新间隔组合数据。（要节约网络带宽，对于非关键数据使用较大的 RPI 速率。） 例如：用户可以为数据创建一个关键标记和一个非关键标记。

调整带宽限制

在 ControlNet 网络上共享标记时，标记必须符合网络带宽要求：

- 随着 ControlNet 网络上的连接数增加，多个连接，包括生成或使用标记，可能需要共享网络更新时间 (NUT)。
- 由于 ControlNet 网络一个 NUT 仅可以传送 500 字节，每个连接的数据必须少于 500 字节以适应 NUT 需要。

按照系统大小，用户可能没有足够的 ControlNet 网络带宽用于 500 字节的标记。如果对于用户的 ControlNet 网络，标记太大，则作下面的一种或多种调整：

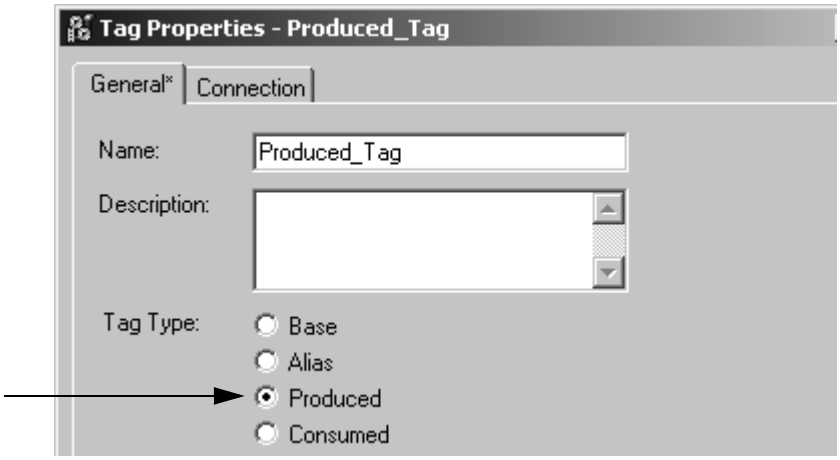
调整：	说明：	
减少用户网络更新时间 (NUT)。	NUT 较快时，只有少量连接需要共享一个更新时槽。	
增加用户的连接请求信息包间隔 (RPI) 。	RPI 较高时，连接可以在更新时槽中轮流发送数据。	
对于远程机架的 ControlNet 转发模块 (CNB)，为该机架选择最有效的通信格式：	**机架中的大多数模块都是不带诊断的数字 I/O 模块吗？**	**则为远程 CNB 模块选择该通信格式：**
	是	机架优化
	否	无
	机架优化格式为机架中的每个槽使用附加的 8 个字节。模拟模块或发送或接收诊断、熔丝、时间戳或确定性数据的模块需要直接连接，且不能利用机架优化格式。 选择 “无”释放每个槽的 8 字节以作它用，如生成或使用标记。	
将标记分成两个或多个小的标记。	1. 按照相同更新率组合数据。例如：用户可以为数据创建一个关键标记和一个非关键标记。 2. 为每个标记分配不同的 RPI。	
创建逻辑以传输较小数据 （包）。	请参阅第 10-1 页上的 ″生成大数组″.	

生成一个标记

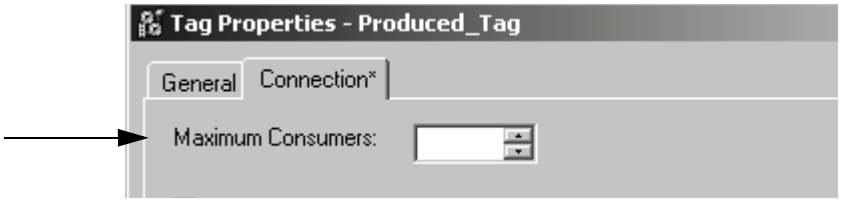
1. 在控制器管理器中，右击 Controller Tags （控制器标记）文件夹，然后选择 Edit Tags （编辑标记）。 （用户仅可以生成控制器范围标记。）

2. 在 Controller Tags （控制器标记）窗口中，右击用户要生成的标记，然后选择 Edit Tag Properties （编辑标记属性）。

3. 单击 Produced option （生成选项）按钮。



4. 选择连接选项卡，然后指定使用 （接收）此标记的控制器数量。



5. 单击 OK

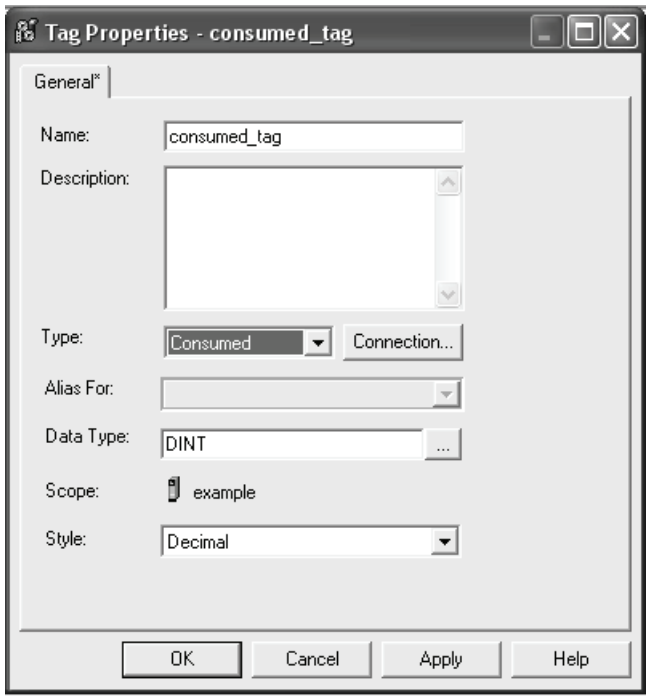
使用其它控制器生成的数据

1. 在控制器管理器的 I/O 配置文件夹中，添加生成数据的控制器（Logix5000 控制器或 PLC-5C 控制器）。

2. 在控制器管理器中，右击 Controller Tags （控制器标记）文件夹，然后选择 Edit Tags （编辑标记）。 （仅控制器范围标记可以使用数据。）

3. 在控制器标记窗口中，右击使用此数据的标记，然后选择 Edit Tag Properties （编辑标记属性）。

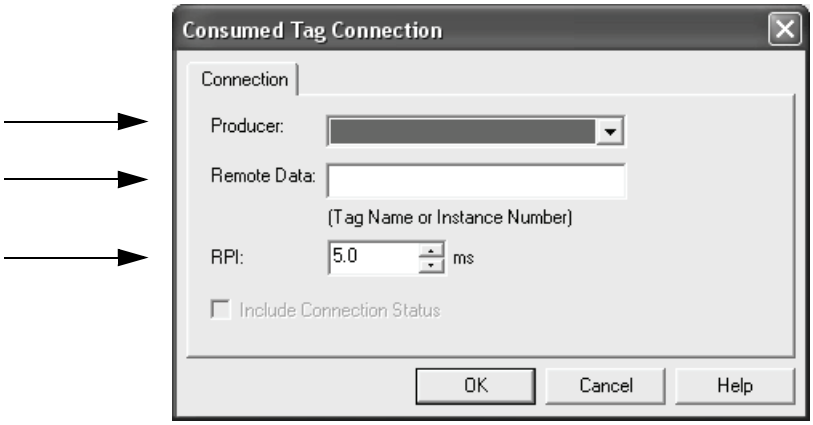
4. 为类型选择 “Consumed”（使用）。



5. 指定数据类型：

如果生成控制器为：	则数据类型必须是：	
Logix5000 控制器	与生成标记数据类型相同	
PLC-5C 控制器	具有以下组成的用户定义的数据类型：	
	数据类型：	**说明：**
	DINT	状态
	INT[*x*]，其中 *x* 是来自 PLC-5C 控制器的数据输出大小。（如果用户仅使用一个 INT，则省略 *x*。）	PLC-5C 控制器生成的数据

6. 单击连接按钮定义使用标记。



- 选择生成数据的控制器。
- 输入远程生成数据的名称或实例。

如果生成控制器为：	则输入或选择：
Logix5000 控制器	生成标记名称
PLC-5C 控制器	来自 PLC-5C 控制器 ControlNet 配置上的消息号

- 为连接输入或选择请求数据包间隔 (RPI)。

7. 单击 OK

8. 如果用户在 ControlNet 网络上使用标记，则使用 RSNetWorx for ControlNet 软件以预定网络。

PLC-5C 控制器的附加步骤

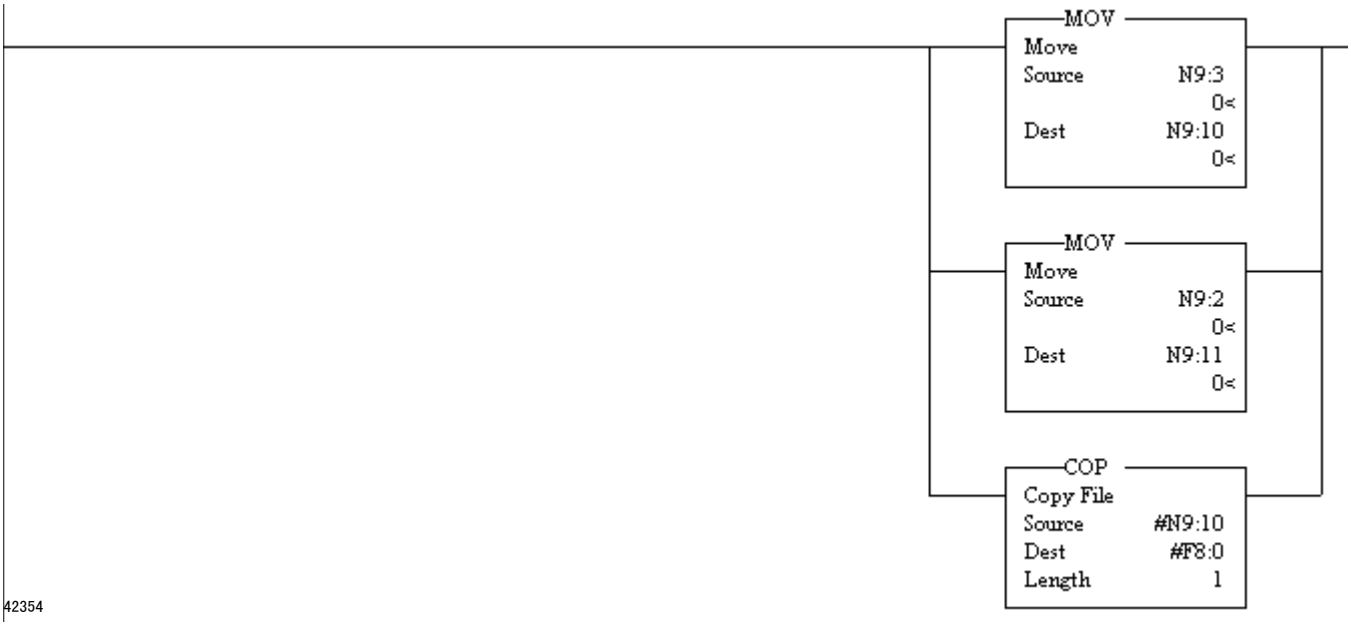
如果用户正在与 PLC-5C 控制器共享数据，则需要：

操作：	详细信息：		
在 PLC-5C 控制器的 ControlNet 配置预定消息。	**如果 PLC-5C：**	**数据类型：**	**则 RSNetWorx 软件：**
	生成	整型	在 PLC-5C 控制器的 ControlNet 配置中插入一条发送预定消息。
	使用	整型	在 PLC-5C 控制器的 ControlNet 配置中： A. 插入一条接收预定消息： B. 在消息大小中，输入生成标记的整数值。
		REAL	在 PLC-5C 控制器的 ControlNet 配置中： A. 插入一条接收预定消息。 B. 在消息大小中，输入生成标记的两倍 REAL 值。 例如：如果生成标记包括 10 个 REAL，则在消息大小中输入 20。
如果 PLC-5C 控制器使用 REAL，则重建值。	在用户为 PLC-5C 控制器生成 REAL （32 位浮点数）时，PLC-5C 以 16 位连续整数存储数据： • 第一个整数包括数值的高位 （最左）。 • 第二个整数包括数值的低位 （最右）。 • 每个浮点数持续使用该样式。 请参阅第 第 9-17 页 页的实例。		

该实例显示如何在 PLC-5C 控制器中重建 REAL （浮点数）

示例 重建一个浮点数

在整数移动到一个新的位置时，两条 MOV 指令颠倒整数顺序。由于 COP 指令的目标是浮点地址，该地址占据两个连续整数 （共 32 位）然后将其转化为单浮点数。



执行消息 (MSG) 指令

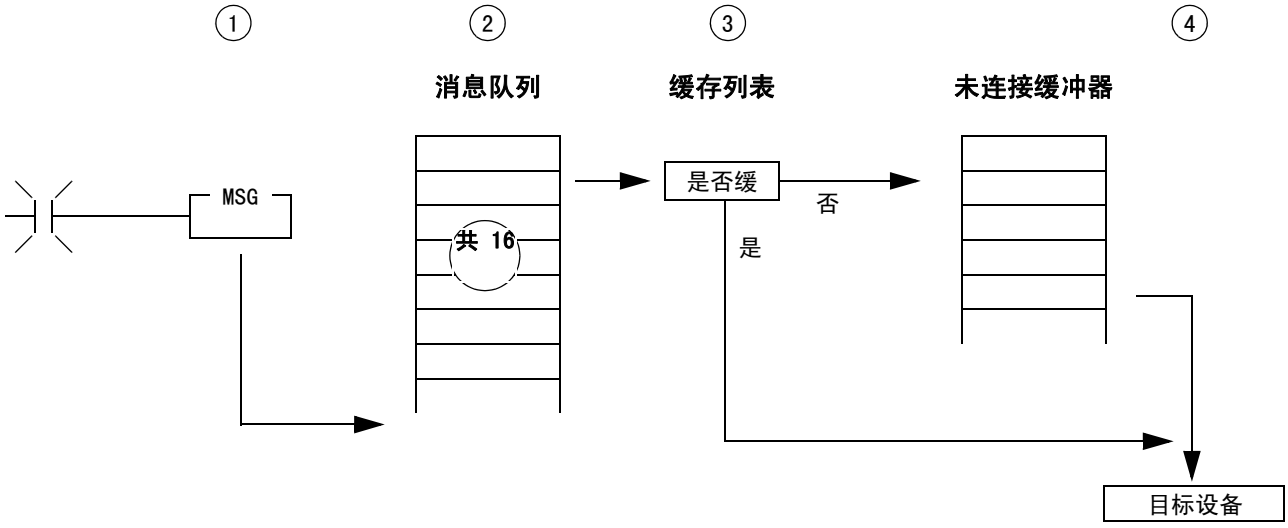
在控制器之间传输数据 （发送或接收数据），用户可以：

如果数据：	则：	请参阅：
需要在用户指定的速率定期传输	生成和使用标记	页 9-9
在用户应用过程中出现特定条件时发送	执行消息 (MSG) 指令	本部分

示例 执行消息 (MSG) 指令

如果 count_send = 1 且 count_msg.EN = 0 （MSG 指令还没有启用），则执行一条 MSG 指令向其它控制器发送数据。

此图表显示控制器如何处理消息指令 (MSG)。

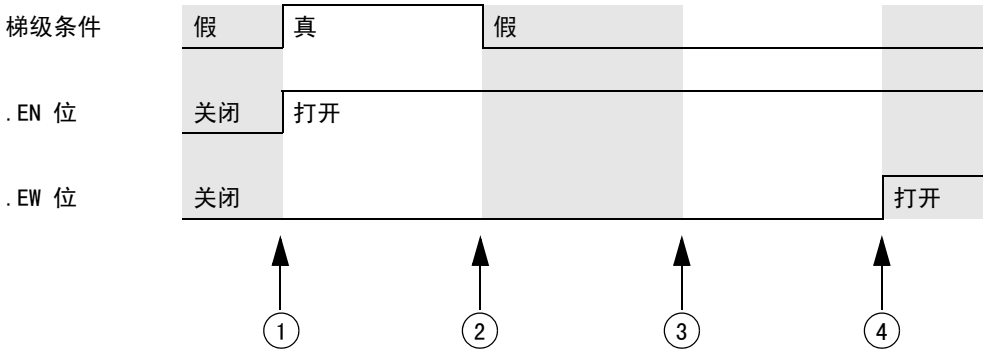


说明：

1	控制器扫描 MSG 指令，且梯级条件为真。 MSG 指令输入消息队列。	
2	MSG 指令离开队列且被处理。	
3	**如果 MSG 指令：**	**则 MSG 指令：**
	不使用连接或连接以前没有缓存。	使用未连接的缓冲器与目标设备通信
	使用连接且连接已缓存	不要使用未连接的缓冲器
4	与目标设备通信。	

消息队列

消息队列可缓存 16 条 MSG 指令，包括配置为块传送读或块传送写的指令。当队列满时，一条指令试图在指令的后续扫描中输入队列，如下面所示：



说明：	
①	控制器扫描 MSG 指令。 MSG 指令的梯级条件为真。 设置 EN 位。 MSG 指令试图进入队列但队列已满 （16 条 MSG 指令已经启用）。 EW 位保持清除。
② & ③	控制器扫描 MSG 指令。 MSG 指令的梯级条件为假。 EN 为保持设置。 MSG 指令试图输入队列但队列已满。 EW 位保持清除。
④	控制器扫描 MSG 指令。 MSG 指令试图进入队列。由于队列有空间，因此向队列输入指令。 设置 EW 位。

缓存列表

按照用户配置 MSG 指令的方式，可能使用连接发送或接收数据。

消息类型：	使用的通信方法：	是否使用连接：
CIP 数据表读或写	→	✓
PLC2、PLC3、PLC5 或 SLC （所有类型）	CIP 带有源 ID 的 CIP	
	DH+	✓
CIP 通用	→	用户选项(1)
块传送读或写	→	✓

(1) 用户可连接 CIP 通用消息。但是对于大多数的应用，我们推荐用户保留 CIP 通用消息为非连接。

如果 MSG 指令使用一个连接，用户可以在发送消息后，选择是保持连接打开 （缓存）还是将它关闭。

如果用户：	则：
缓存连接	执行 MSG 指令后，保持连接打开。这优化执行时间。每次执行消息时打开连接增加执行时间。
不缓存连接	执行 MSG 指令后，关闭连接。这释放连接作为它用。

控制器限制用户可以缓存的连接数：

如果用户有该软件和固件修订版：	则用户可以缓存：
11.x 或更早	• 多达 16 个连接的块传输消息 • 多达 16 个连接的其它类型消息
12.x 或后期版本	多达 32 个连接

如果多个消息传输到相同的设备，则消息可能共享一个连接。

如果 MSG 指令传输到：	且这些指令：	则：
不同设备	——►	每个 MSG 指令使用 1 个连接。
相同设备	同时启用	每个 MSG 指令使用 1 个连接。
	不同时启用	MSG 指令共享一个连接。（即，它们一起作为 1 个连接。）

示例

共享一个连接

如果控制器轮流向同一个模块发送块传送读和块传送写消息，则两个消息一起作为 1 个连接。缓存两个消息作为缓存列表上的 1 个连接。

非连接缓冲器

要建立连接或处理非连接消息，控制器使用一个非连接缓冲器。

术语：	定义
非连接缓冲器	控制器使用的一种内存分配，用以处理非连接通信。控制器在下面情况下执行非连接通信： • 与设备建立一个连接，包括 I/O 模块 • 执行一条不使用连接的 MSG 指令 控制器可以有 10 至 40 个非连接缓冲器。 • 默认数量为 10。 • 要增加非连接缓冲器的数量，执行一条 MSG 指令重新配置非连接缓冲器数。请参阅第 9-23 页上的 获取或设置非连接缓冲器数量。 • 每个非连接缓冲器使用的内存为 1.1K 字节。 • 如果在一条指令离开消息队列时，所有非连接缓冲器都在使用，则指令出错且不传送数据。

如果一条 MSG 指令使用一个连接，则在首次执行该指令建立一个连接时，使用非连接缓冲器。如果用户配置该指令以缓存连接，则一旦连接建立就不再需要非连接缓冲器。

原则

用户规划和编程 MSG 指令时，应遵循下面的原则：：

原则：	详细信息：
1. 对于每个 MSG 指令，创建一个控制标记。	每个 MSG 指令需要自己的控制标记。 • 数据类型 = MESSAGE • 范围 = 控制器 • 标记不能是数组的一部分或用户定义的数据类型。
2. 在控制器范围中保留源和 / 或目标数据。	MSG 指令仅可以访问 Controller Tags （控制器标记）文件夹中的标记（控制器范围）。
3. 如果用户 MSG 传送到使用 16 位整数的设备，则在 MSG 中使用 INT 缓冲器，且整个工程项目中使用 DINT。	如果用户消息传送到使用 16 位整数的设备，如 PLC-5 或 SLC 500 控制器，且其传送整数 （非 REAL），则使用 INT 缓冲器，且整个工程项目中使用 DINT。 这会提高用户工程项目的效率，因为在使用 32 位整型数据 (DINT) 时，Logix5000 控制器执行起来更有效，使用更少的内存。 请参阅第 9-26 页上的 在 INT 和 DINT 之间转换。
4. 缓存频繁执行的连接的 MSG 指令。	为那些频繁执行的 MSG 指令缓存连接，多达用户控制器修订版本中允许的最大量。 由于每次消息执行时不需要打开连接，因此优化执行时间。
5. 如果用户一次要启用的 MSG 指令超过 16 条，则应使用某种管理策略。	如果用户一次要启用的 MSG 指令超过 16 条，则一些 MSG 指令在输入队列时可能需要延迟。要保证执行每条指令，则使用下面选项中的一个： • 按顺序启用每条消息。 • 按组启用消息。 • 编写一条消息与多个设备通信。更多信息，请参阅附录 B。 • 编写逻辑调节消息执行。更多信息，请参阅附录 A。
6. 保持非连接和非缓存 MSG 数量小于非连接缓冲器的数量。	控制器可以有 10 至 40 个非连接缓冲器。 默认数量为 10。 • 如果在一条指令离开消息队列时，所有非连接缓冲器都在使用，则指令出错且不传送数据。 • 用户可以增加非连接缓冲器数量 （最多 40），但应继续原则 5。 • 要增加非连接缓冲器数量，请参阅 第 9-23 页。

获取或设置非连接缓冲器数量

要确定或更改非连接缓冲器数量，请使用一条 MSG 指令。

- 非连接缓冲器数量为 10 至 40。
- 默认数量为 10。
- 每个非连接缓冲器使用的内存为 1.1K 字节。

获取非连接缓冲器数量

要确定控制器当前可用的非连接缓冲器数量，按以下步骤配置一条消息指令 (MSG)：

此选项卡上：	为此项：	键入或选择：
配置	消息类型	CIP 通用
	服务类型	自定义
	服务代码	3
	类别	304
	实例	1
	属性	0
	源元素	*source_array* 的数据类型为 SINT[4] **在该元素中：** **输入：** *source_array*[0] 1 *source_array*[1] 0 *source_array*[2] 17 *source_array*[3] 0
	源长度 （字节）	4 （写入 4 个 SINT。）
	目标	*destination_array* 的数据类型为 SINT[10] （保留所有值为 0。） *destination_array[6]* 为非连接缓冲器当前数量
通信	路径	1, *slot_number_of_controller*

设置非连接缓冲器数量

作为起始值，设置非连接缓冲器数量等于一次启用的非连接和非缓存消息数量，再大概加上 5。附加的 5 个缓冲器提供余量，以防用户低估一次启用的消息数量。

要更改控制器非连接缓冲器数量，按以下步骤配置一条消息指令 (MSG)：

此选项卡上：	为此项：	键入或选择：
配置	消息类型	CIP 通用
	服务类型	自定义
	服务代码	4
	类别	304
	实例	1
	属性	0
	源元素	*source_array* 的数据类型为 SINT[8]
	源长度 （字节）	8 （写入 8 个 SINT。）
	目标	*destination_array* 的数据类型为 SINT[6]（保留所有值为 0。）
通信	路径	1, *slot_number_of_controller*

在该元素中：	输入：
source_array[0]	1
source_array[1]	0
source_array[2]	17
source_array[3]	0
source_array[4]	用户需要的非连接缓冲器数量。
source_array[5]	0
source_array[6]	0
source_array[7]	0

示例 设置非连接缓冲器数量

如果 S:FS （第一次扫描）为 1，则为控制器设置非连接缓冲器数量。

Source_Array[0] = 1

Source_Array[1] = 0

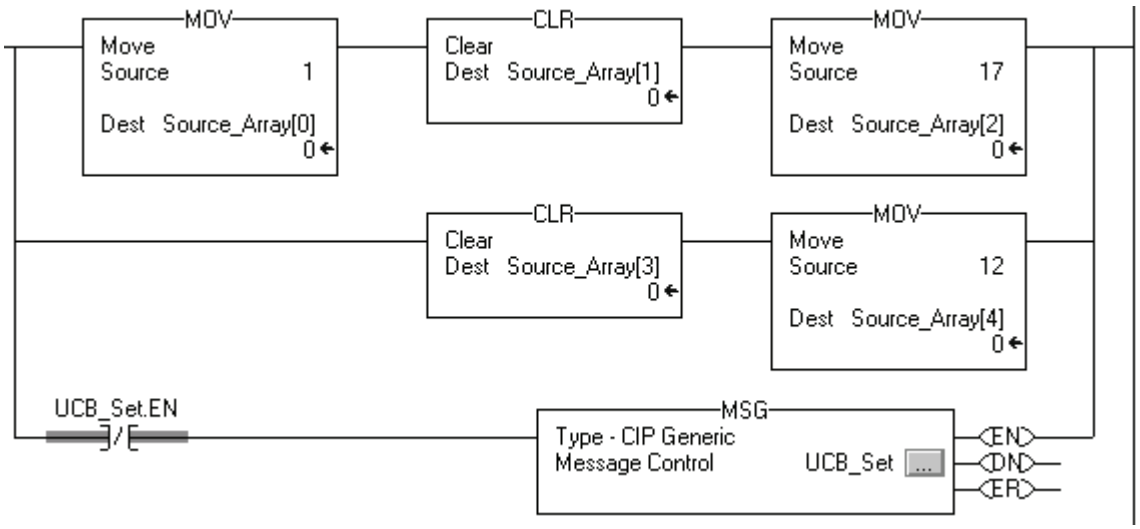
Source_Array[2] = 17

Source_Array[3] = 0

Source_Array[4] = 12 （用户需要的非连接缓冲器数量。在该实例中，用户需要 12 个缓冲器。）

如果 UCB_Set.EN 为 0 （MSG 指令未启用），则：

MSG 指令设置非连接缓冲器数量为 Source_Array[4]



其中：

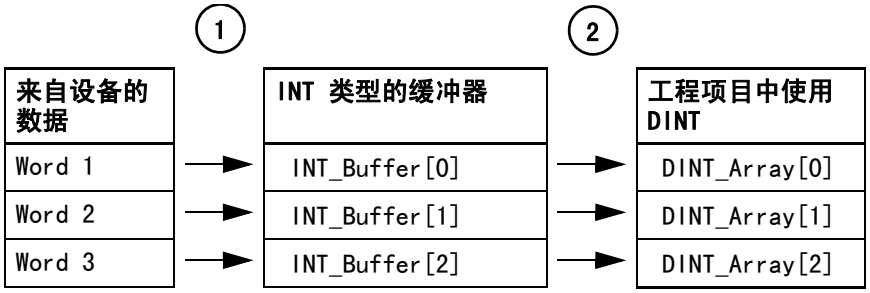
标记名	类型	说明
UCB_Set	MESSAGE	MSG 指令的控制标记。
Source_Array	SINT[8]	MSG 指令的源值，包括用户需要的非连接缓冲器数量。

在 INT 和 DINT 之间转换

在 Logix5000 控制器中，在任何可能情况下为整数使用 **DINT** 数据类型。在使用 32 位整型 (DINT) 时，Logix5000 控制器执行起来更有效，使用更少的内存。

如果用户消息传送到使用 16 位整数的设备，如 PLC-5 或 SLC 500 控制器，且其传送整数 （非 REAL），则使用 INT 缓冲器，且整个工程项目中使用 DINT。 这会提高用户工程项目的效率。

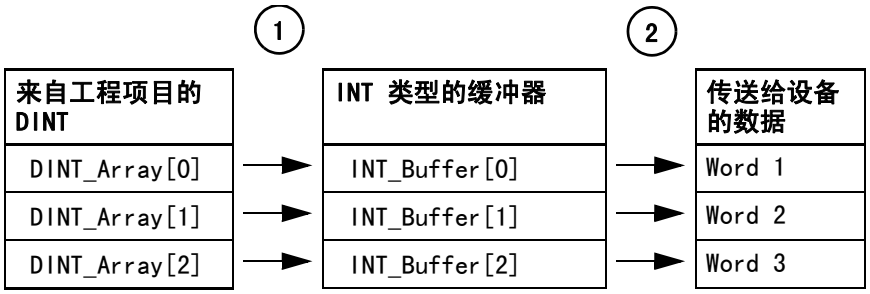
读取 16 位整数



1. 消息指令 (MSG) 从设备读取 16 位整型数据 (INT)，并把它们存储在一个 INT 类型的暂存数组中。

2. 文件算术和逻辑指令 (FAL) 将 INT 类型转化成双整型 (DINT)，目的是能够被用户工程项目中的其它指令使用。

写入 16 位整数



1. FAL 指令将 Logix5000 控制器中的 DINT 转换为 INT。

2. MSG 指令将暂存数组中的 INT 类型写入设备。

示例 读取 PLC-5 控制器的整数值

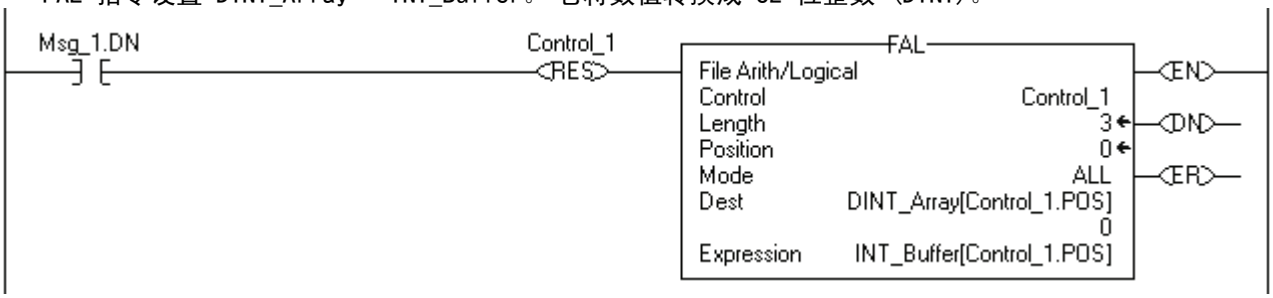
如果 Condition_1 为 1，且 Msg_1.EN 为 0 （MSG 指令未启用），则：

读取 PLC-5 控制器的 3 个整数值，并将它们存入 INT_Buffer (3 INT)

如果 Msg_1.DN =1 （MSG 指令已读取数据），则

重置 FAL 指令。

FAL 指令设置 DINT_Array = INT_Buffer。 它将数值转换成 32 位整数 (DINT)。

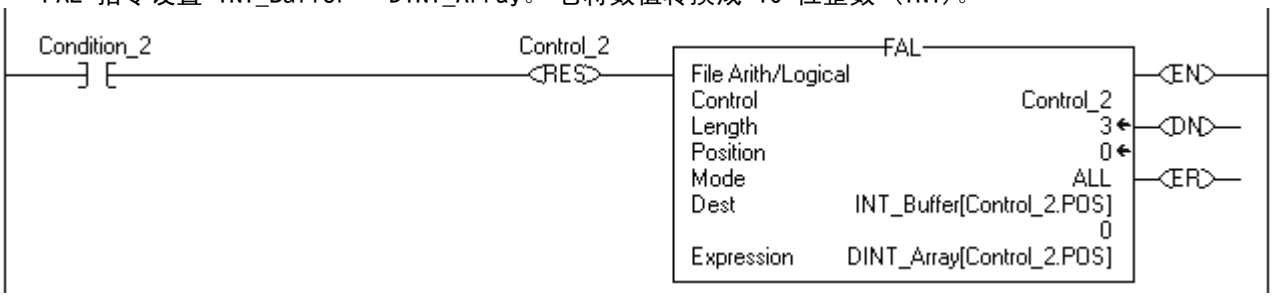


示例 向 PLC-5 控制器写入整数值

如果 Condition_2 为 1，则：

重置 FAL 指令。

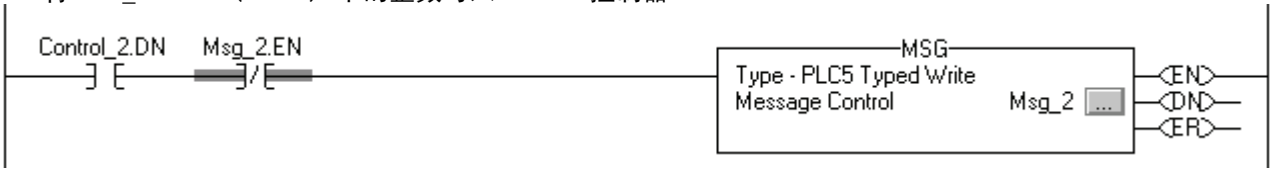
FAL 指令设置 INT_Buffer = DINT_Array。 它将数值转换成 16 位整数 (INT)。



如果 Control_2.DN = 1 (FAL 指令已将 DINT 转换成 INT)

且 Msg_2.EN = 0 （MSG 指令未启用），则：

将 INT_Buffer (3 INT) 中的整数写入 PLC-5 控制器



注释：

生成大数组

何时使用本章

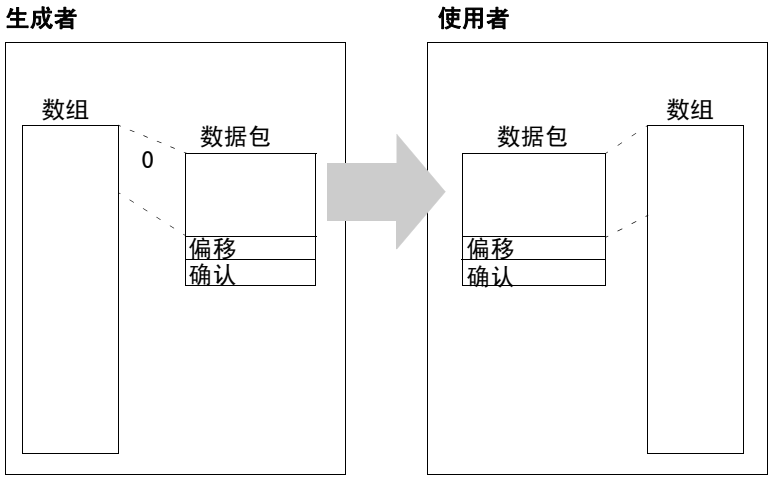
Logix5000 控制器可以通过一个计划的连接发送多达 500 字节数据。这对应于数组的 125 个 DINT 或 REAL 元素。要传输超过 125 个 DINT 或 REAL 的数组，请使用 125 个元素的生成 / 使用标记来创建数据包。然后可以使用数据包将数组片断发送到其他控制器。

因此，以较小数据包发送大数据数组时，必须确保数据包传输完成，才能将数据移动到目标数组。

- ControlLogix 底板上产生的数据以 50 字节片断发送。
- 数据传输与程序扫描异步进行。

此部分包含的逻辑使用确认字以确保数据移动到目标数组前每个数据包包含新数据。逻辑还使用偏移值指示数组中数据包的开始元素。

因为存在偏移和确认元素，每个数据包包含数组的 123 个元素数据，如下所示：



此外，数组必须包含额外 122 个元素。换句话说，它必须比要传输的最大元素数量多 122 个元素。

- 这些元素充当缓冲区。
- 因为每个数据包包含相同数量元素，缓冲区防止控制器在数组边界外复制。
- 如果没有缓冲区，在最后一个数据包有不到 123 个实际数据的元素时，将发生这种边界外复制情况。

生成大数组

1. 在生成数组的控制器项目的 Controller Tags （控制器标记）文件夹中创建下面的标记：

P	标记名称	类型
	array_ack	DINT[2]
✓	array_packet	DINT[125]

2. 将 array_ack 转换为使用标记：.

对于：	指定：
控制器	接收数据包的控制器名称
远程标记名称	array_ack 两个控制器对此数据使用相同名称。

3. 在 Controller Tags （控制器标记）文件夹或将要包含传输逻辑的程序的标记文件夹中创建下面的标记：

标记名称	类型
数组	DINT[x]，其中 x 等于要传输的元素数量加上 122 个元素
array_offset	DINT
array_size	DINT
array_transfer_time	DINT
array_transfer_time_max	DINT
array_transfer_timer	TIMER

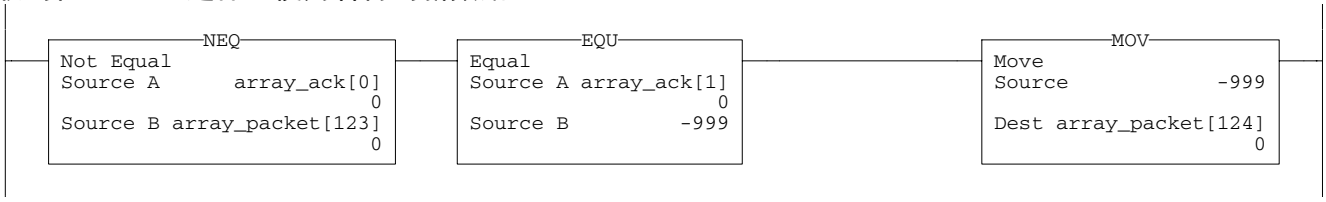
4. 在 array_size 标记中输入实际数据的元素数量。（第 3. 步的 x 值减去缓冲区的 122 个元素。）

5. 为将创建数据包的逻辑创建或打开例程。

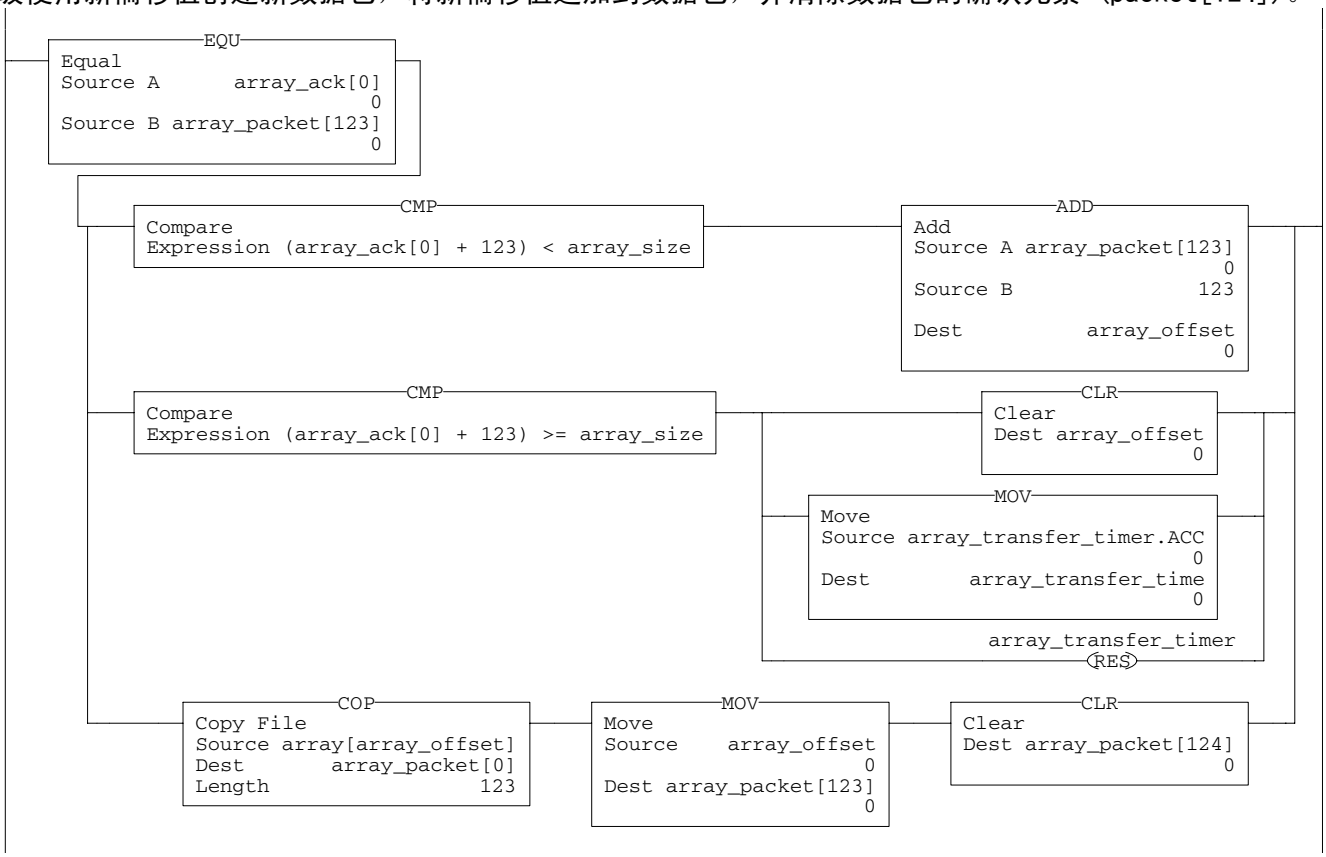
6. 输入此逻辑：

确定发送整个数组的时间

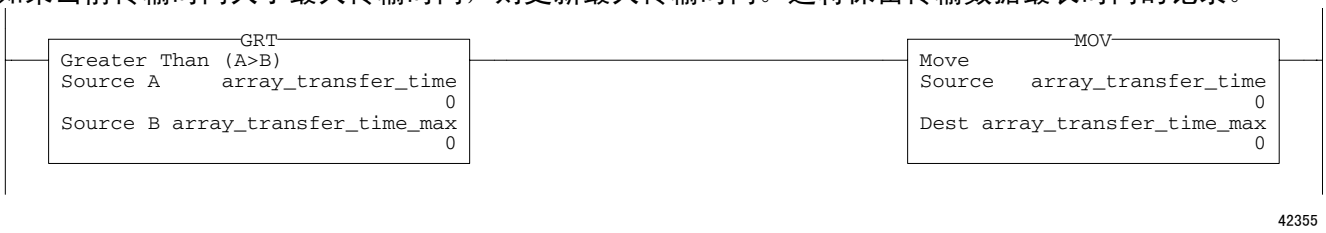
当 array_ack[0] 中的偏移值不等于当前偏移值但 array_ack[1] 等于 -999 时，使用者开始接收新数据包，因此梯级将 -999 移动至数据包的最后一个元素。使用者将数据包复制到数组前，将等待直到接收到值 -999。这保证使用者得到新数据。



当 array_ack[0] 中的偏移值等于当前偏移值时，使用者已将数据包复制到数组，因此梯级检查要传输的更多数据。如果偏移值加上 123 小于数组大小，则有更多数据需要传输，因此梯级增加偏移 123。否则，没有更多要传输的数据，因此梯级重置偏移值，记录传输时间并重置计时器。在任一情况下，梯级使用新偏移值创建新数据包，将新偏移值追加到数据包，并清除数据包的确认元素 (packet[124])。



如果当前传输时间大于最大传输时间，则更新最大传输时间。这将保留传输数据最长时间的记录。



7. 在使用数组的控制器项目的 Controller Tags （控制器标记）文件夹中创建下面的标记：

P	标记名称	类型
✓	array_ack	DINT[2]
	array_packet	DINT[125]

8. 将 array_packet 转换为使用标记：

对于：	指定：
控制器	发送数据包的控制器名称
远程标记名称	array_packet 两个控制器对此数据使用相同名称。

9. 在 Controller Tags （控制器标记）文件夹或将要包含传输逻辑的程序的标记文件夹中创建下面的标记：

标记名称	类型
数组	DINT[x]，其中 x 等于要传输的元素数量加上 122 个元素
array_offset	DINT

10. 为准备将数据从数据包移动到目标数组的逻辑创建或打开例程。

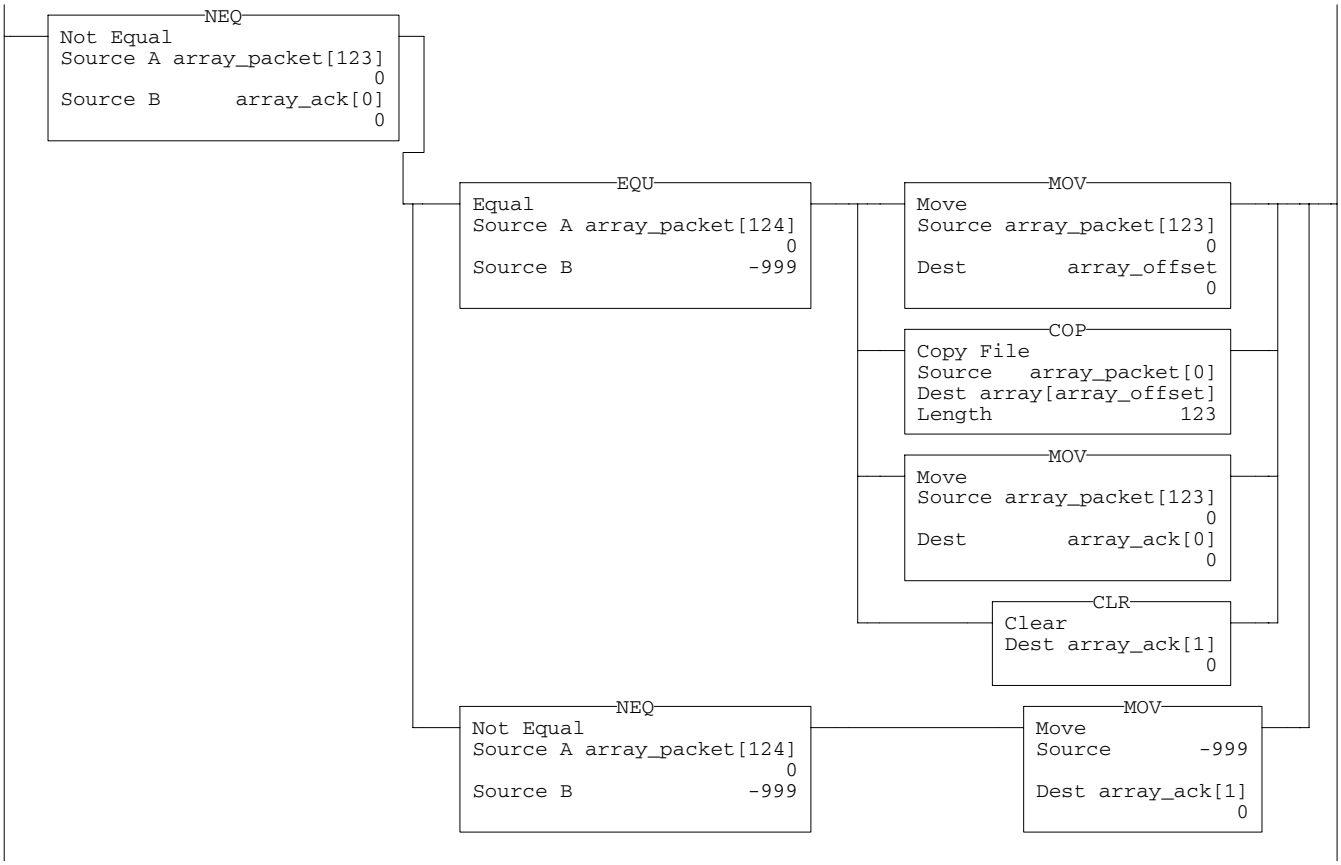
11. 输入此逻辑：

当 array_packet[123] 中的偏移值不同于 array_ack[0] 中的偏移值时，控制器开始接收新数据包，因此梯级检查数据包最后一个元素中的值 -999。

如果数据包的最后一个元素等于 -999，则控制器已接收新数据的整个数据包并开始复制操作：

- 偏移值从数据包移动至 array_offset。
- COP 指令将数据从数据包移动至目标数组，从偏移值开始。
- 偏移值移动至 array_ack[0]，表示复制完成。
- Array_ack[1] 重置为零并等待新数据包到达的信号。

如果数据包的最后一个元素不等于 -999，则向控制器的数据包传输可能没有完成，因此 -999 移动至 array_ack[1]。这将通知生成者在数据包最后一个元素中返回值 -999，以验证数据包的传输。



42356

相比其他传输数据的方法，将大数组作为较小的数据包传输可提高系统性能。

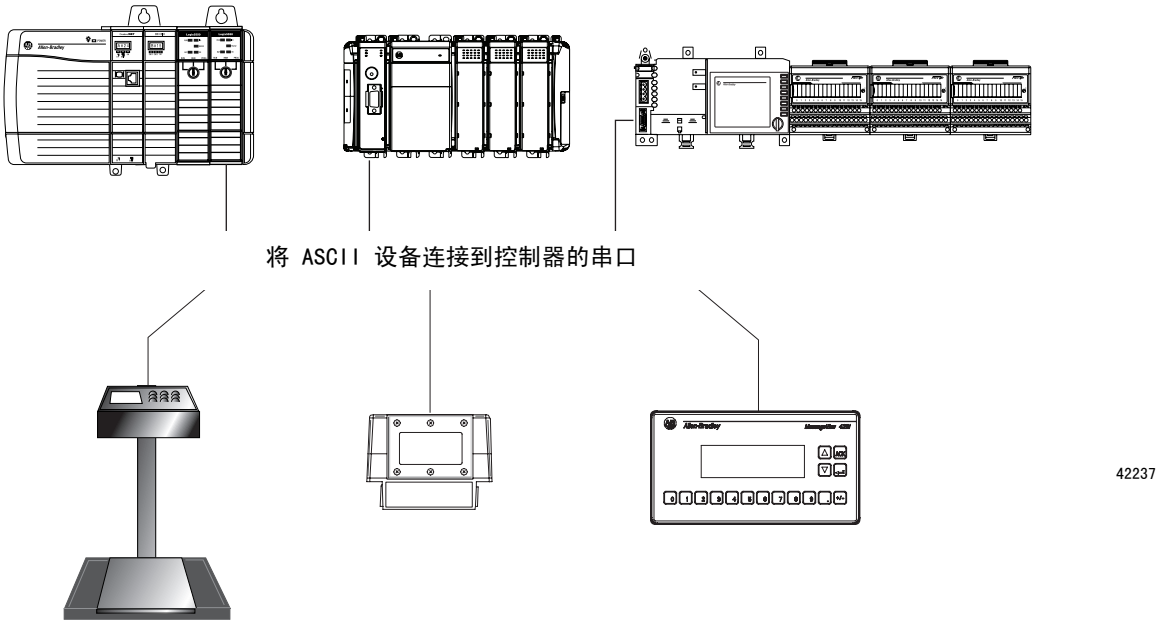
- 相比将数据分散到多个数组并将每个作为生成标记发送的方式，这种方式使用更少的连接。例如，一个具有 5000 个元素的数组将有 40 个使用数组的连接 (5000/125=40)。
- 相比使用消息指令发送整个数组实现更快的传输速度。
  - 消息没有计划，并仅在 Logix5550 执行的“系统开销”部分执行。因此，消息将花相当长的时间完成数据传输。
  - 可以通过增加系统开销时间段来加快传输速度，但将降低连续性任务的性能。

与 ASCII 设备通信

何时使用此章节

使用该程序通过控制器的串口互换设备中的 ASCII 数据。例如，用户可以使用串口执行下面操作：

- 读取来自磅秤模块或条形码阅读器的 ASCII 字符
- 读取和接收来自 ASCII 触发设备的消息，如 MessageView 终端。



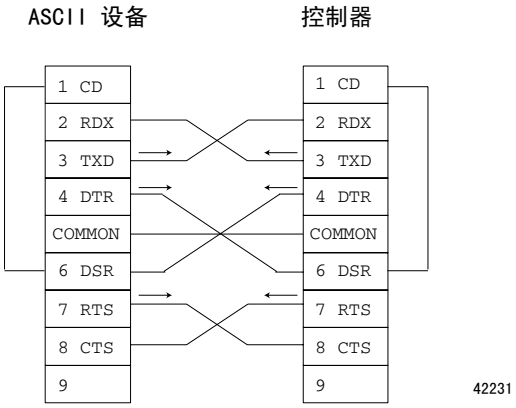
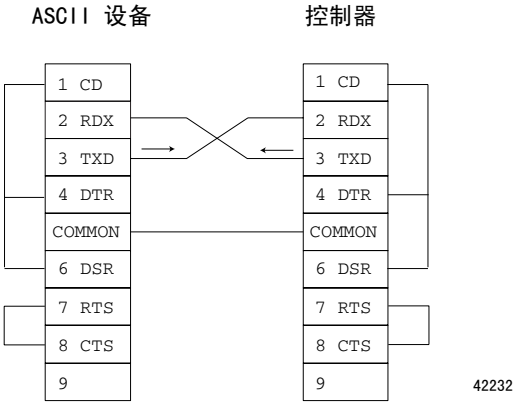
相关信息：	请参阅页码：
连接 ASCII 设备	11-2
配置串口	11-3
配置用户协议	11-4
创建字符串数据类型	11-6
读取设备字符	11-7
向设备发送字符	11-9
输入 ASCII 字符	11-11

除了控制器串口，1756-EWEB EtherNet/IP 网络服务器模块的固件 3.1 和更高版本支持插槽接口，从而 Logix5000 可以使用 TCP 或 UDP 套接字服务来互换 ASCII 数据。请参阅 ***EtherNet/IP 网络服务器用户指南***，出版物 ENET-UM0527，修订版 C 或最新版本。

连接 ASCII 设备

1. 对于 ASCII 设备的串口，确定哪个插针引脚发送信号，哪个插针引脚接收信号。

2. 将发送端插针引脚连接到相应的接收端插针引脚，安装上跳线：

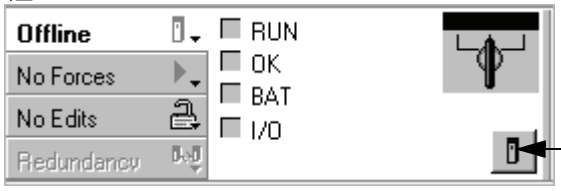
如果通信：	那么按照下面方式连线：
握手	
未握手	

3. 将电缆屏蔽端连接到两个连接器。

4. 将电缆连接到控制器和 ASCII 设备。

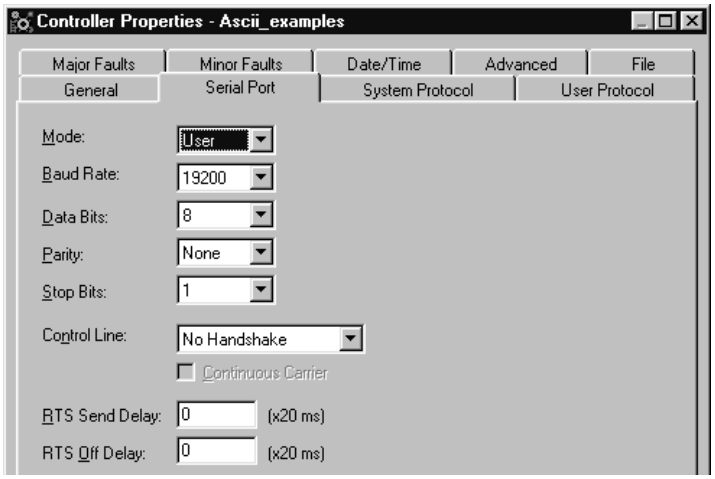
配置串口

1. 在控制器项目的在线工具栏上，单击 controller （控制器）按钮。



2. 选择 Serial Port （串口）选项卡。

3. 选择 User mode （用户模式），输入串口配置设置。



- 选择波特率、数据位、奇偶位和停止位。
- 选择控制行选项：

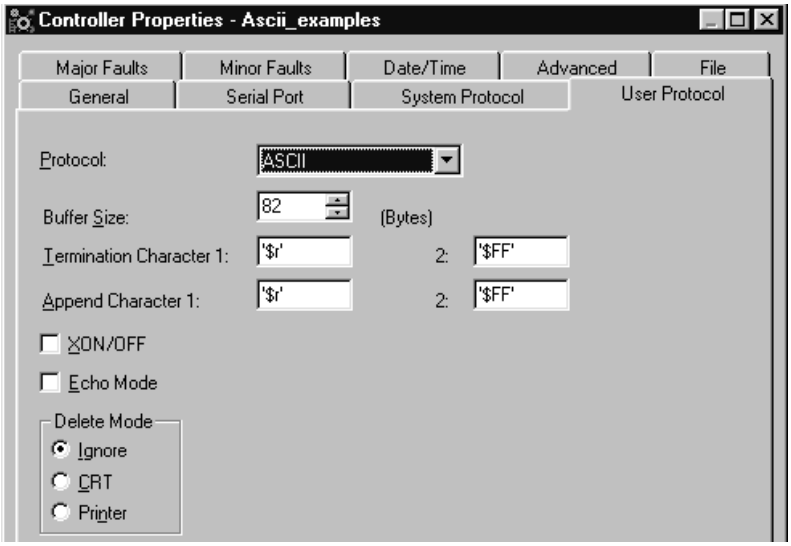
如果：	并且：	且这是：	选择：	则：
用户未使用调制解调器	——→		无握手	
用户正在使用调制解调器	点对点链接的两个调制解调器都是全双工	——→	全双工	
	主调制解调器全双工，而从调制解调器半双工	主控制器	全双工	
		从控制器	半双工	检查连续载波复选框。
	系统所有调制解调器都是半双工	——→	半双工	清除连续载波复选框 （默认）。

- 对于 RTS 发送延迟，输入一个时间延迟 （20 毫秒时段），该计数表示从确定 RTS 信号 （高电平）到开始数据传送之间所用的时间。例如， 数值 4 产生 80 毫秒的延迟。
- 对于 RTS 关断延迟，输入一个时间延迟 （20 毫秒时段），该计数表示从信号传送到关掉 RTS 信号 （低电平）之间所用的时间。

4. 单击 Apply （应用）。

配置用户协议

1. 选择 User Protocol （用户协议）选项卡。



42252

- 输入缓冲区容量大于或等于传输中字符的最大容量。（两倍字符容量是最佳容量。）
- 对于 ABL 或 ARL 指令，输入终止字符指示数据结尾。关于 ASCII 代码，请参阅本手册的背面。

如果设备发送：	则：	注释：
一个终止字符	A. 在终止字符 1 的文本框中，键入第一个字符的十六进制 ASCII 代码。 B. 在终止字符 2 的文本框中，键入 $FF。	对于可打印的字符如 1 或 A，键入该字符。
两个终止字符	在终止字符 1 和 2 的文本框中，键入每个字符的十六进制 ASCII 代码。	

- 对于 AWA 指令，输入附加字符。 对于 ASCII 代码，请参阅本手册的背面。

要附加：	则：	注释：
一个字符	A. 在附加字符 1 的文本框中，键入第一个字符的十六进制 ASCII 代码。 B. 在附加字符 2 的文本框中，键入 $FF。	对于可打印的字符如 1 或 A，键入该字符。
两个字符	在附加字符 1 和 2 的文本框中，键入每个字符的十六进制 ASCII 代码。	

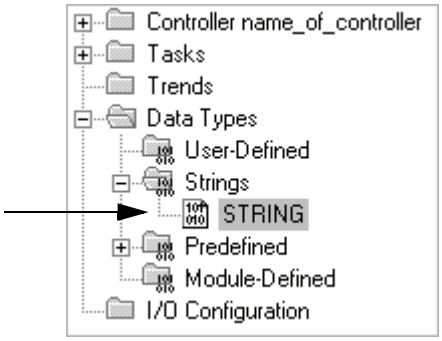
- 如果 ASCII 设备配置为 XON/XOFF 数据流控制，请选择 XON/XOFF 复选框。
- 如果 ASCII 设备为 CRT 或预配置为半双工传输，请选择 Echo Mode （响应模式）复选框。
- 选择删除模式：

如果 ASCII 设备为：	选择：	注释：
CRT	CRT	• DEL 字符 ($7F) 和 DEL 前的字符不 发送到目的地址。 • 如果选择响应模式，并通过 ASCII 指令阅读 DEL 字符，则响应返回三个字符：BACKSPACE SPACE BACKSPACE ($08 $20 $08)。
打印机	打印机	• DEL 字符 ($7F) 和 DEL 前的字符不 发送到目的地址。 • 如果选择响应模式，并通过 ASCII 指令阅读 DEL 字符，则响应返回两个字符：/ ($2F) 后面为被删除的字符。
不属于上面的设备	忽略	将 DEL 字符 ($7F) 与其它任何字符一样对待。

2. 单击 OK （确定）。

创建字符串数据类型

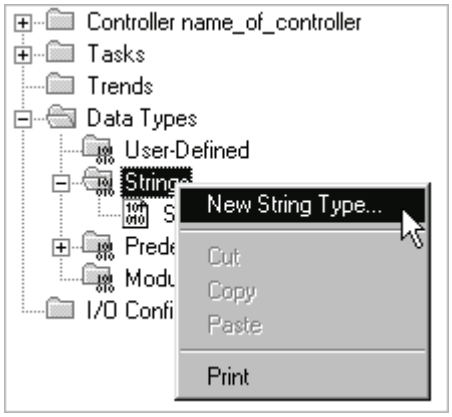
使用**字符串** 数据类型存储标记中的 ASCII 字符。



42811

用户可以使用默认的 STRING 数据类型。该数据类型存储的字符数可达 82。

或



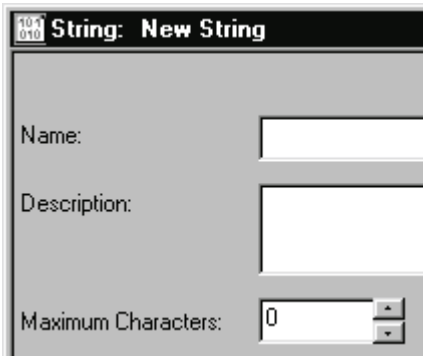
42812

用户可以创建新的 STRING 类型以存储用户定义的字符数。

重要

用户创建新的 STRING 数据类型时要注意。如果用户后来决定更改 STRING 数据类型的容量，用户可能会丢失任何标记中当前使用的数据类型的数据。

如果用户：	则：
创建的 STRING 数据类型太小	• 数据被截断。 • 长度不能更改。
创建的 STRING 数据类型太大	数据和长度重置为零。



42233

1. 在控制器管理器中，右键单击 Strings （字符串）并选择 New String Type （新建字符串类型）…
2. 键入数据类型名。
3. 键入该 STRING 数据类型将存储的最大字符数。
4. 单击 OK （确定）。

读取设备字符

作为通用规范，用户读取缓冲器前使用一条 ACB 或 ABL 指令验证缓冲器是否包括所需字符：

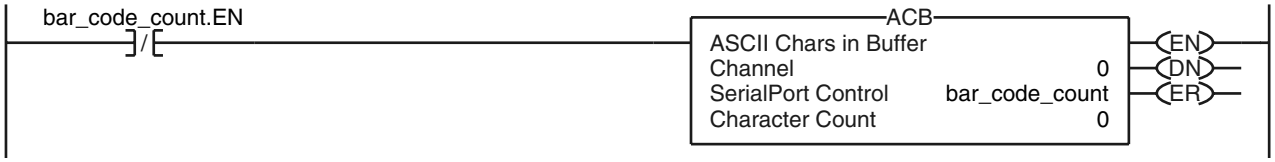
- ARD 或 ARL 指令继续读取缓冲器直到指令读取到所需字符。
- 在 ARD 或 ARL 指令读取缓冲器时，不可以执行其它 ASCII 串口指令 （除了 ACL）。
- 验证缓冲器是否包括所需字符，能够避免在输入设备发送数据时 ARD 或 ARL 继续执行其他 ASCII 串口指令。

关于 ASCII 串口指令的附加信息，请参阅 *Logix5000 基本指令集参考手册*，出版物 1756-RM003。

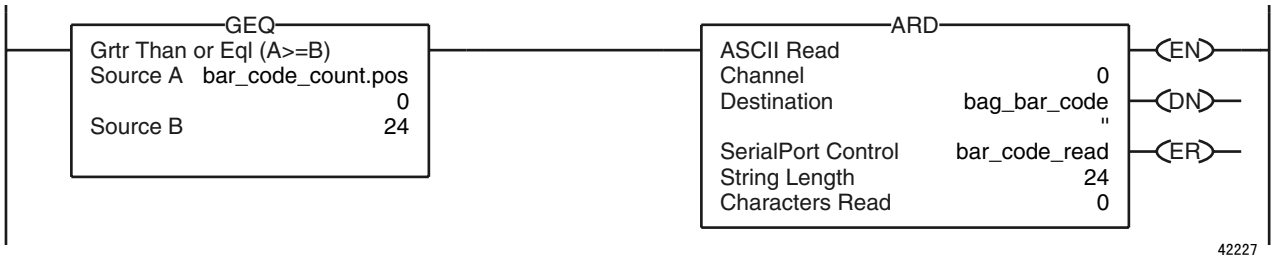
例如，如果设备发送固定字符数，如条形码阅读器。

示例

条形码阅读器向控制器的串口 （通道 0）发送条形码。每个条形码包括 24 个字符。要确定控制器何时接收条形码，ACB 指令连续计算缓冲器中的字符。



在缓冲器至少包括 24 个字符时，控制器接收字符。ARD 指令将条形码移动到 bag_bar_code 标记。



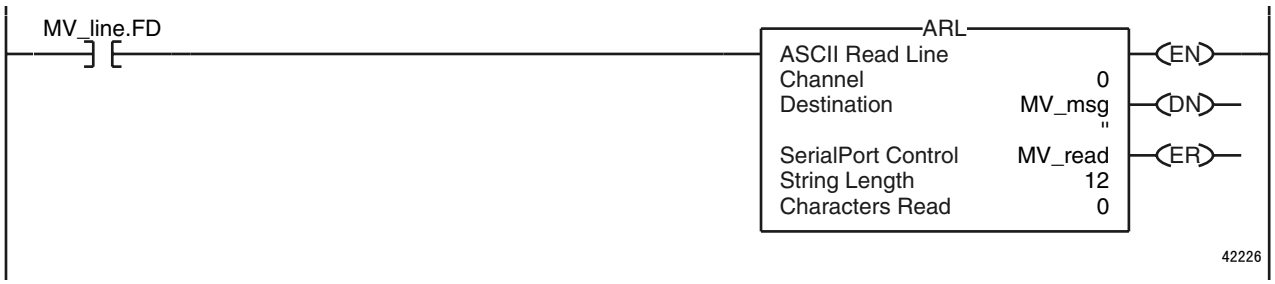
例如，如果设备发送的字符数可变，如信息或显示终端：

示例

继续测试信息缓冲器。

- 由于每条信息以回车键（$0D）结束，回车键在 User Protocol （用户协议）标记 Controller Properties （控制器属性）对话框中被配置为终止字符。
- ABL 找到回车键时设置 FD 位。

ABL 指令找到回车键时 （设置 MV_line.FD），控制器从缓冲器删除字符 （包括回车键），将它们置于 MV_msg 标记。



向设备发送字符

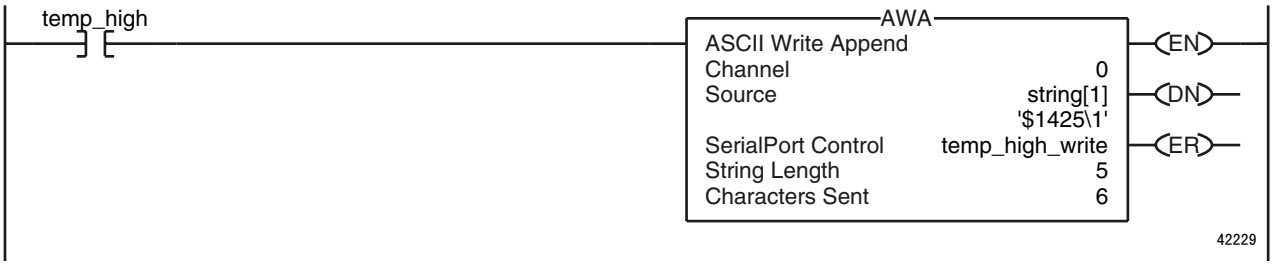
用户向设备发送字符时，需要确定是否每次都发送等量字符，是否需要向数据中附加终止字符。

例如，如果用户发送的字符数总是相等，并要自动向数据末尾附加一到两个字符：

示例

当温度超过上限时 （temp_high 激活），AWA 指令从 string[1] 标记发送五个字符到 MessageView 终端。

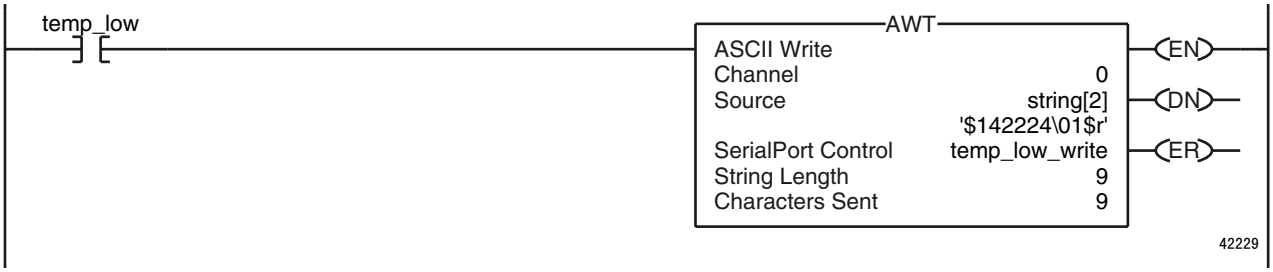
- $14 作为一个字符计算。 它是 Ctrl-T 字符的十六进制代码。
- 该指令也发送 （附加）用户协议中定义的字符。该实例中，AWA 指令发送回车键 ($0D) 表示消息结束。



然后一直发送等量的字符数：

示例

当温度达到下限时 （temp_low 激活），AWA 指令从 string[2] 标记发送九个字符到 MessageView 终端。（$14 作为一个字符计算。它是 Ctrl-T 字符的十六进制代码。）

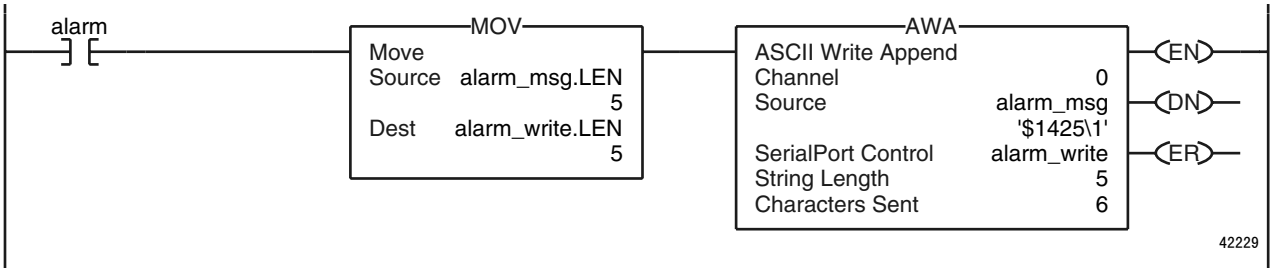


例如，如果用户每次发送的字符数不等，且要自动向数据末尾附加一到两个字符：

示例

当警报打开时，AWA 指令向 alarm_msg 中发送字符，并附加一个终止字符。

- 由于 alarm_msg 中的字符数改变，rung 首先移动 alarm_msg 的长度 (alarm_msg.LEN) 到 AWA 指令的长度 (alarm_write.LEN)。
- 在 alarm_msg 中，$14 作为一个字符计算。它是 Ctrl-T 字符的十六进制代码。

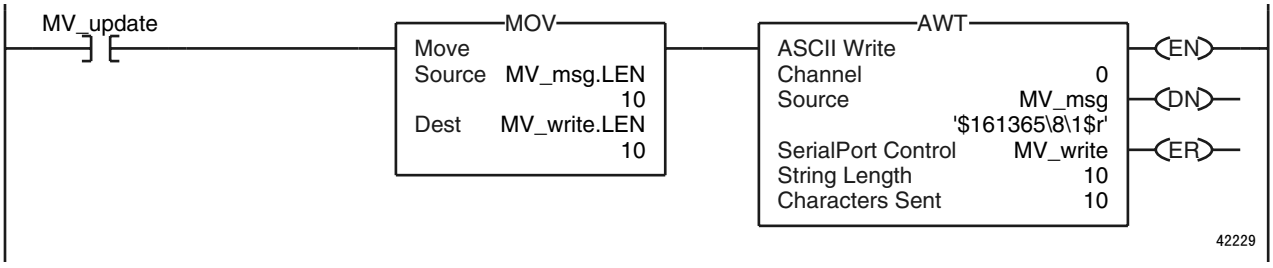


然后每次发送不等量的字符数：

示例

当激活 MV_update 时，AWT 指令向 MV_msg 中发送字符。

- 由于 MV_msg 中的字符数改变，rung 首先移动 MV_msg 的长度 (MV_msg.LEN) 到 AWT 指令的长度 (MV_write.LEN)。
- 在 MV_msg 中，$16 作为一个字符计算。它是 Ctrl-V 字符的十六进制代码。

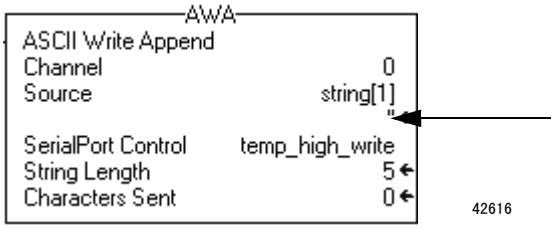


输入 ASCII 字符

如果：	则：
用户要逻辑创建字符串。	转到第 12-1 页上的 “处理 ASCII 字符”。
用户要输入字符。	转到步骤 1.

重要 String Browser （字符串浏览器）窗口显示多达字符标记 LEN 成员的数值。字符串标记可能包括 String Browser （字符串浏览器）窗口不显示的附加数据。

1. 双击源的数值区。



出现文本输入框：

2. 为字符串输入字符。

3. 单击 OK （确定）。

注释：

处理 ASCII 字符

何时使用本章

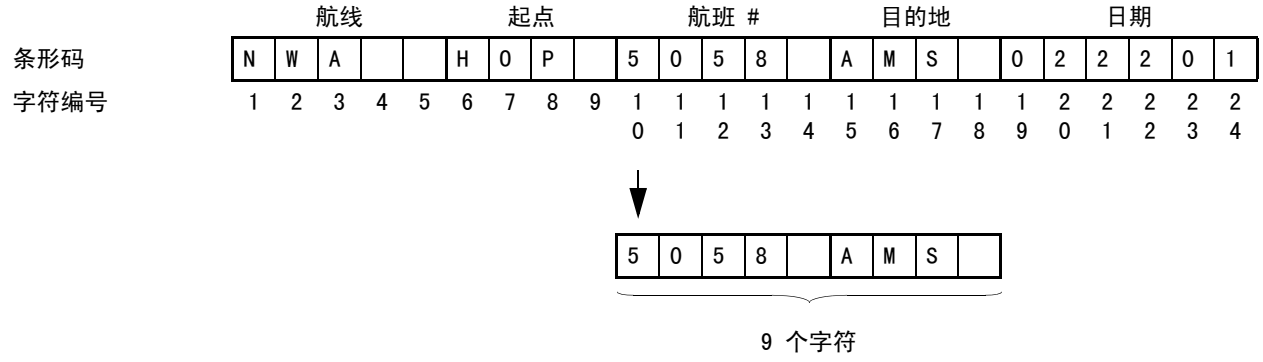
使用本章：

- 解释条形码并根据条形码采取操作
- 当重量作为 ASCII 字符发送时使用来自磅秤的重量
- 从 ASCII 触发设备解码消息，例如操作员终端
- 使用来自应用程序的变量为 ASCII 触发设备生成字符串

有关 ASCII 相关指令的其他信息，请参阅 *Logix5000 控制器基本指令集参考手册*，出版物 1756-RM003。

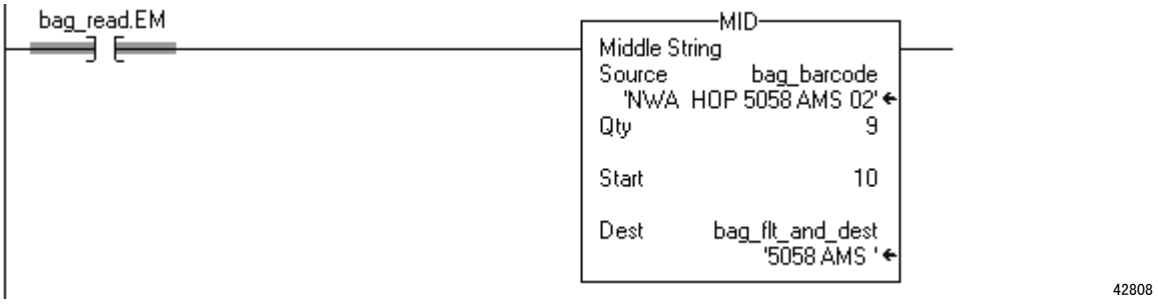
提取部分条形码

例如，条形码可能包含机场传送带上包裹的信息。要检查航班号和包裹目的地，可以提取字符 10 - 18。



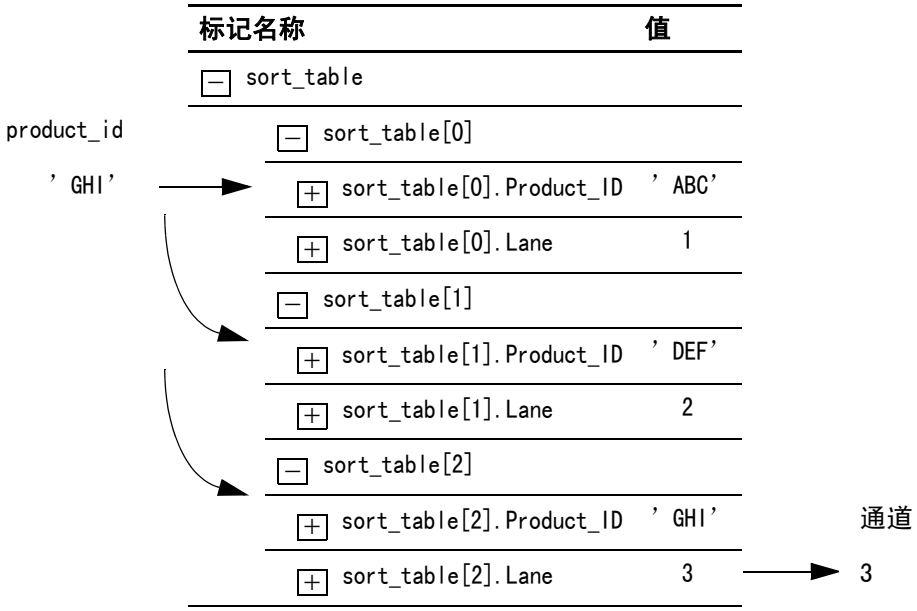
示例

在机场的包裹处理传送带中，每个包裹得到一个条形码。条形码的字符 10 - 18 是航班号和包裹的目的地机场。读取条形码后 （bag_read.EM 为 on），MID 指令将航班号和目的地机场复制到 bag_flt_and_dest 标记。



查找条形码

例如，在分拣操作中，一组用户定义的数据类型创建一个显示每种产品通道号的表。为确定发送产品的通道，控制器在表中搜索产品 ID（条形码上标识产品的字符）。

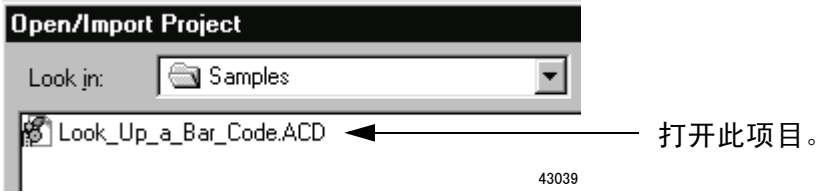


查找条形码：

- 创建 PRODUCT_INFO 数据类型
- 搜索字符
- 标识通道号
- 拒绝错误字符
- 输入产品 ID 和通道号

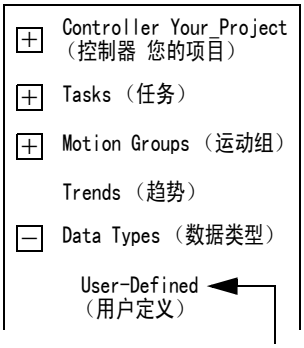
提示

要从示例项目中复制以上组件，请打开 …\RSLogix 5000\Projects\Samples 文件夹。



创建 PRODUCT_INFO 数据类型

创建新数据类型：

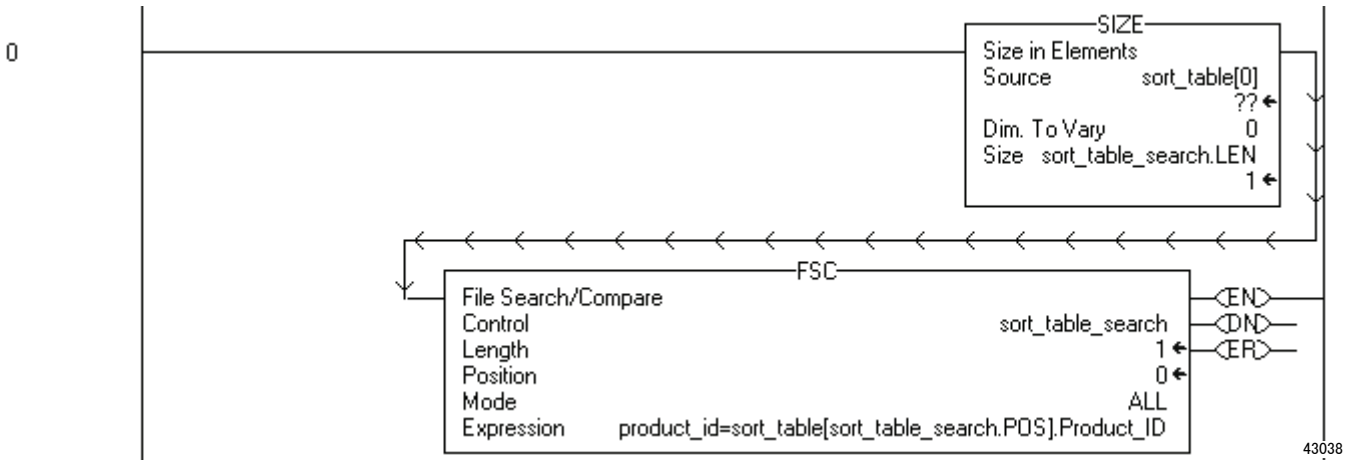


右击并选择 *New Data Type*（新建数据类型）。

创建此用户定义的数据类型。

数据类型：PRODUCT_INFO				
名称	PRODUCT_INFO			
说明	根据标识项目的 ASCII 字符串标识项目的目的地			
成员				
	名称	**数据类型**	**样式**	**说明**
	⊞ product_id	STRING		标识项目的 ASCII 字符
	通道	DINT	十进制	基于 ID 的项目目的地

搜索字符



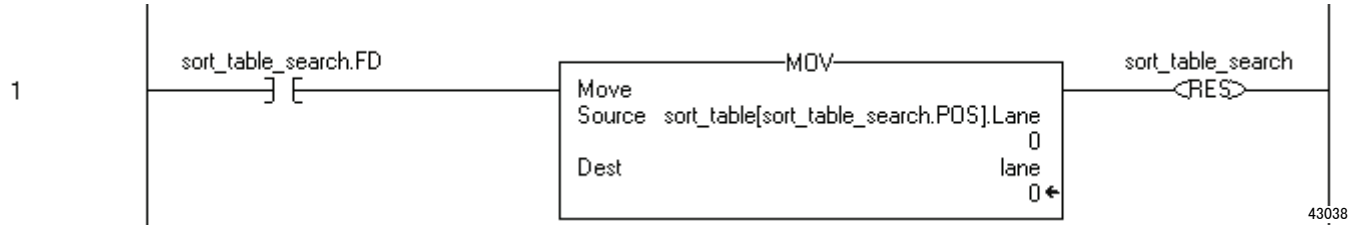
SIZE 指令：

- 计算 sort_table 数组 （类型 PRODUCT_INFO）中的元素数量。此数组包含每个项目的产品 ID 和该项的对应通道号。
- 计算数组维度 0 中的元素数量。在此情况下，这是唯一维度。
- 将后面的 FSC 指令的 Length 设置为 sort_table 数组的大小。

FSC 指令搜索 sort_table 数组中的每个 Product_ID 成员，直至指令找到与 product_id 标记的匹配。

- sort_table_search 标记控制 FSC 指令
- 尽管前面的指令设置此指令的 Length，但是可以输入初始值来检验项目
- product_id 标记包含要查找的条形码字符

标识通道号



当 FSC 指令在 sort_table 数组中找到产品 ID 后，该指令设置 FD 位。POS 成员指示匹配的 sort_table 数组中的元素号。对应的 LANE 成员指示匹配的通道号。

根据 POS 值，MOV 指令将相应的通道号移至通道标记中。控制器使用此标记的值发送项目。

MOV 指令设置通道标记的值后，RES 指令重置 FSC 指令，这样可以搜索下一个产品 ID。

拒绝错误字符

如果 FSC 指令没有在 sort_table 数组中找到产品 ID，指令将设置 DN 位。MOV 指令将 999 移动到通道标记中以通知控制器拒绝或重新发送项目。

MOV 指令设置通道标记的值后，RES 指令重置 FSC 指令，这样可以搜索下一个产品 ID。

输入产品 ID 和通道号

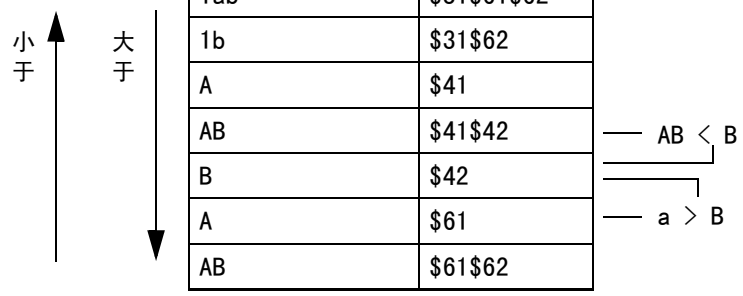
在 sort_table 数组中，输入 ASCII 字符标识每项及该项的对应通道号。

标记名称	值
⊟ sort_table	{…}
⊟ sort_table[0]	{…}
⊞ sort_table[0].Product_ID	标识第一项的 ASCII 字符
⊞ sort_table[0].Lane	项目的通道号
⊟ sort_table[1]	{…}
⊞ sort_table[1].Product_ID	标识下一项的 ASCII 字符
⊞ sort_table[1].Lane	项目的通道号

检查条形码字符

使用比较指令 （EQU、GEQ、GRT、LEQ、LES、NEQ）检查特定字符。

- 字符的十六进制值决定字符串小于还是大于另一个字符串。
- 两个字符串按照电话号码簿方式排序时，它们的大小由字符串的顺序决定。



ASCII 字符	十六进制代码
1ab	$31$61$62
1b	$31$62
A	$41
AB	$41$42
B	$42
A	$61
AB	$61$62

使用以下比较指令之一：

要查看字符串是否：	输入此指令：
等于特定字符	EQU
不等于特定字符	NEQ
大于特定字符	GRT
等于或大于特定字符	GEQ
小于特定字符	LES
等于或小于特定字符	LEQ

例如：

示例

当 bag_flt_and_dest 等于 gate[1] 时，xfer[1] 变为 on。这样将把包裹发送到所需的门。

转换值

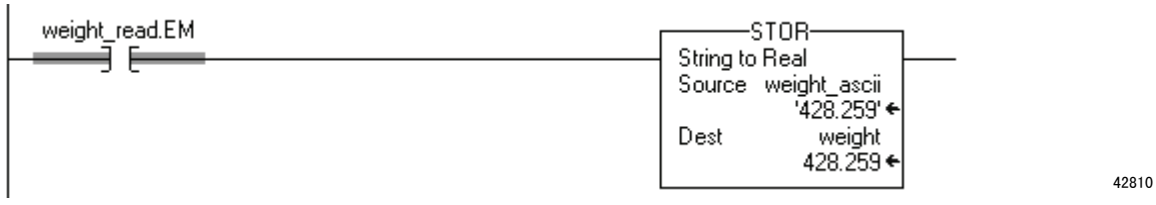
可以将值的 ASCII 表示转换为可在应用程序中使用的 DINT 或 REAL 值。

- STOD 和 STOR 指令跳过任何初始控制字符或非数字字符 （数字前的减号除外）。
- 如果字符串包含多组由分隔符（例如 /）分隔的数字，STOD 和 STOR 指令将仅转换第一组数字。

例如，将 ASCII 字符转换为浮点值：

示例

从磅秤读取重量后 （weight_read.EM 为 on），STOR 指令将 weight_ascii 中的数字字符转换为 REAL 值并将结果存储在 weight 中。



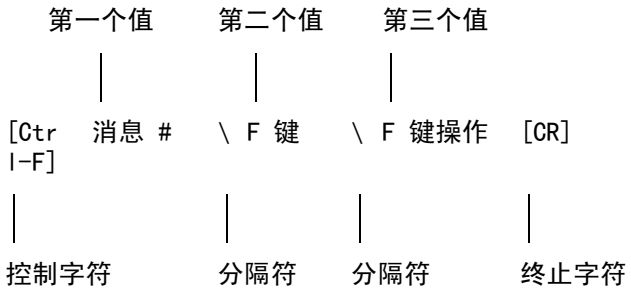
例如，将 ASCII 字符转换为整数值：

示例

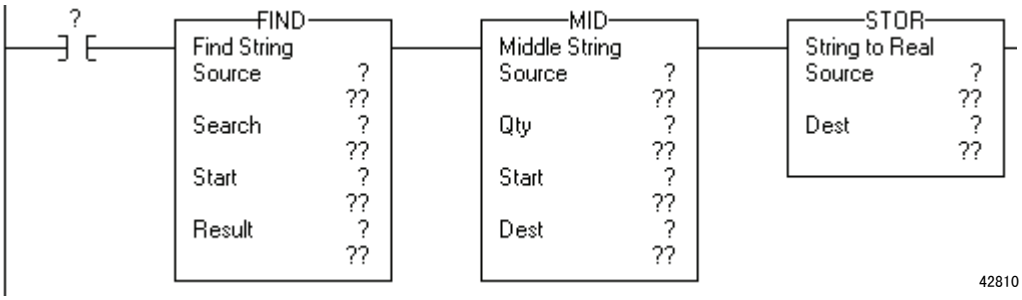
当 MV_read.EM 为 on 时，STOD 指令将 MV_msg 的第一组数字字符转换为整数值。指令跳过初始控制字符（$06）并在分隔符（ \ ）处停止。

解码 ASCII 消息

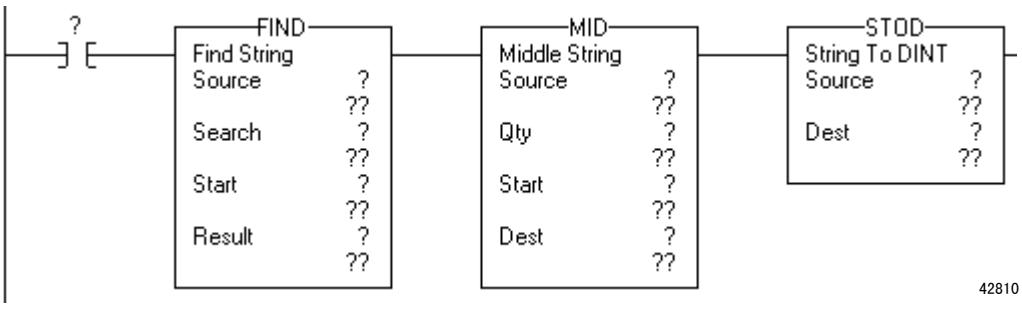
可以从包含多个值的 ASCII 消息中提取并转换值。例如，消息可能如下所示：



梯级 A：查找并转换浮点值



梯级 B：查找并转换整数值



FIND 指令在字符串中定位字符：

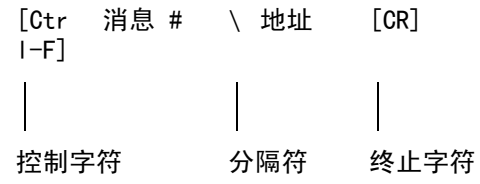
- Source 包含要搜索的字符串标记。
- Result 包含 FIND 指令在其中定位所指定搜索值的位置。

MID 指令标识字符串中的一组字符，并将它们放置在自己的字符串标记中：

- Source 是和 FIND 指令相同的字符串标记。
- 数量值通知 MID 指令要从源提取的字符数量。
- Start 值和 FIND 指令的 Result 值相同。这通知 MID 指令要从 Source 中开始提取字符的位置。
- Destination 包含定位的字符。

生成字符串

此示例生成一个包含两个变量的字符串。例如，操作员终端可能需要一个类似下面的字符串：



- 对于更多变量，请使用额外的 INSERT 或 CONCAT 指令。
- 如果需要发送浮点值，请使用 RTOS 指令代替 DTOS 指令。
- 最终字符串不包括终止字符。发送字符串时，使用 AWA 指令自动追加终止字符。

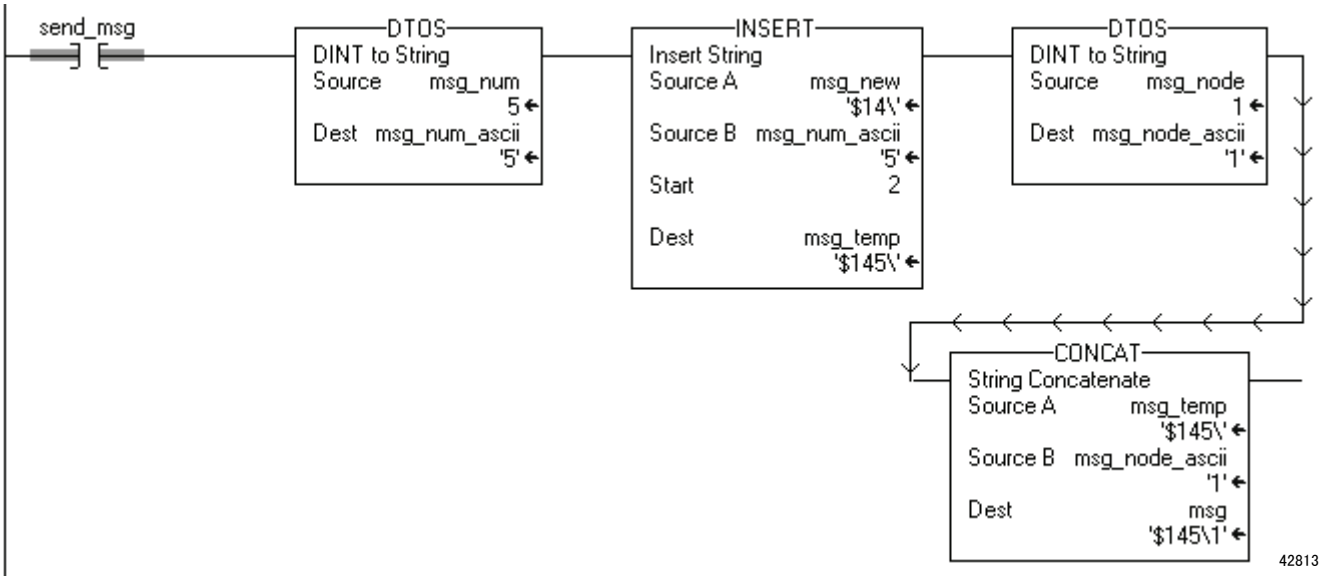
示例

要在 MessageView 终端中触发消息，控制器向终端发送下面格式的消息：[Ctrl-T] 消息 # \ 地址 [CR]

当 send_msg 为 on 时，梯级这样操作：

- 第一个 DTOS 指令将消息号转换为 ASCII 字符。
- INSERT 指令将消息号 （ASCII 格式）插入到控制字符 [Ctrl-T] 后。（Ctrl-T 的十六进制代码为 $14。）
- 第二个 DTOS 指令将终端的节点号转换为 ASCII 字符。
- CONCAT 指令将节点号 （ASCII 格式）放置在反斜线 [\ ] 后并将最终字符串存储在 msg 中。

要发送消息，AWA 指令发送 msg 标记并追加回车符 [CR]。



强制逻辑单元

何时使用此章节

使用强制重写用户逻辑使用或生成的数据。例如，在下面的情况下使用强制：

- 测试和调试用户逻辑
- 检查与输出设备的连接
- 在输出设备出错时，临时保持用户程序运行

强制是一种临时措施。它们不能作为用户应用程序中的永久部分。

预防措施

使用强制时，执行下列预防措施：

注意

强制会引起意想不到的机器运动，可能会造成人员伤亡。用户在使用强制前，确定强制将对机器或程序造成哪些影响，同时应保证人员远离机器。

- 启用 I/O 强制会引起输入值、输出值、生成值或使用值的改变。
- 启用 SFC 强制会引起用户机器或程序转到不同状态或阶段。
- 删除强制仍有可能将强制保留在启用状态。
- 如果强制启用，且用户安装了强制，新强制立即激活。

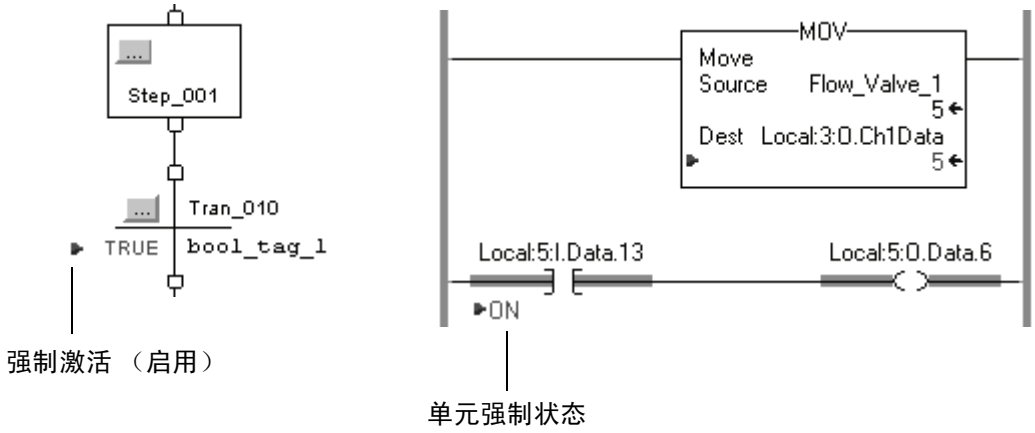
启用强制

要使强制生效，用户应启用强制。用户只能在控制器级别启用或禁用强制。

- 用户既可以分开也可以同时启用 I/O 强制和 SFC 强制。
- 用户不能启用或禁用特定模块、标记集或标记单元强制。

重要 如果用户下载一个启用了强制的项目，编程软件在项目下载完成后，提示用户启用或禁用强制。

强制激活 （启用），▶ 出现在被强制的单元旁。



禁用或删除强制

要停止强制激活并让用户的项目按程序执行，请禁用或删除强制。

- 用户既可以同时也可以分开启用 I/O 强制和 SFC 强制。
- 删除别名标记上的强制同时也删除基础标记上的强制。

注意

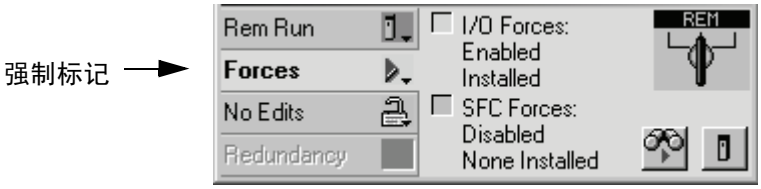
强制的改变会引起意想不到的机器运动，可能会造成人员伤亡。 用户在禁用或删除强制前，确定改变将对机器或程序造成哪些影响，同时应保证人员远离机器。

检查强制状态

用户使用强制前，确定控制器强制状态。用户可以检查强制状态：

要确定状态：	使用下面的操作：
I/O 强制	• 在线工具栏 • FORCE LED • GSV 指令
SFC 强制	在线工具栏

在线工具栏显示强制状态。它分别显示 I/O 强制和 SFC 强制的状态。



此：	意义是：
启用	• 如果项目包含此类型的强制，它们正在 重写用户逻辑。 • 如果用户添加此类型的强制，新的强制立即激活
禁用	此类型强制未被激活。如果项目包含此类型的强制，它们没有 重写用户逻辑。
安装	项目中至少存在一种此类型强制。
未安装	项目中不存在此类型强制。

FORCE LED

如果用户控制器有 FORCE LED，请使用 LED 确定 I/O 强制状态。

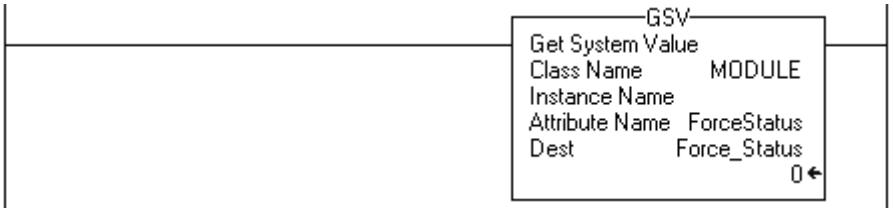
重要 FORCE LED 只显示 I/O 强制状态。不显示 SFC 强制状态。

如果 FORCE LED 为：	则：
关闭	• 没有强制数值的标记。 • I/O 强制没激活 （禁用）。
闪烁	• 至少有一个强制数值的标记。 • I/O 强制没激活 （禁用）。
恒定	• I/O 强制激活 （启用）。 • 强制值可能存在也可能不存在。

GSV 指令

重要 ForceStatus 属性只显示 I/O 强制状态。不显示 SFC 强制状态。

下面的例子显示如何使用 GSV 指令获取强制状态。



其中：

Force_Status 为 DINT 标记。

要确定是否：	检查下列位：	数值为：
安装强制	0	1
*未*安装强制	0	0
启用强制	1	1
禁用强制	1	0

怎样强制

用户可以强制项目的下列元素：

如果用户希望：	则：
重写输入值、输出值、生成标记或使用标记	添加 I/O 强制
重写一次转变条件从活动步骤转到下一步	单步调试转变或路径强制
重写一次并行路径强制，并执行路径各步骤	
在流程图中重写转变条件	添加 SFC 强制
执行流程图中部分但不是全部并行路径分支	

何时使用 I/O 强制

使用 I/O 强制：

- 重写其它控制器的输入值 （即，使用标记）
- 重写输入设备的输入值
- 重写用户逻辑程序并为其它控制器指定输出值 （即，生成标记）
- 重写用户逻辑程序并指定输出设备的状态

重要 强制增加逻辑程序执行时间。强制值越高，执行逻辑程序的时间越长。

重要 控制器控制 I/O 强制，而不是编程工作站。即使编程工作站断开仍保持强制。

用户何时强制 I/O 值：

- 用户可以强制所有 I I/O 数据，除了配置数据。
- 如果标记是数组或结构，如 I/O 标记，则强制 BOOL、SINT、INT、DINT 或 REAL 元素或成员。
- 如果数据值为 SINT、INT 或 DINT 类型，用户可以强制整个值或强制数值的各位。各位都具有强制状态：
  - 没有强制
  - 打开强制
  - 关闭强制
- 用户也可以为 I/O 结构成员、生成标记或使用标记强制别名。
  - 别名标记与基础标记共享同一数据值，因此强制别名标记也强制相关的基础标记。
  - 删除别名标记上的强制即删除相关基础标记上的强制。

强制输入值

强制输入值或使用标记：

- 不管物理设备或生成标记的值，重写值
- 不影响监视输入值或生成标记的其它控制器接收到的值

强制输出值

强制输出或生成标记重写物理设备或其它控制器的逻辑程序。以仅侦听方式监视输出模块的其它控制器也参考强制值。

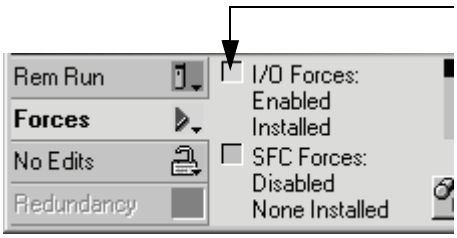
添加 I/O 强制

要重写输入值、输出值、生成标记或使用标记，使用 I/O 强制：

注意

强制可以引起意想不到的机器运动，可能会造成人员伤亡。 用户在使用强制前，确定强制将对机器或程序造成哪些影响，同时应保证人员远离机器。

- 启用 I/O 强制会引起输入值、输出值、生成值或使用值的改变。
- 如果强制启用，且用户安装了强制，新强制立即激活。



1. I/O 强制指示器的状态如何？

如果：	则注意：
关闭	当前不存在任何 I/O 强制。
闪烁	未激活任何 I/O 强制。但用户项目中至少存在一个强制。在用户启用 I/O 强制时，现有的所有 I/O 强制将生效。
恒定	启用 I/O 强制 （激活）。用户安装 （添加）一个强制时，立即生效。

2. 打开包括用户希望强制标记的例程。
3. 右击标记，然后选择 “Monitor”（监视）… 如有必要，展开标记显示用户希望强制的值 （例如：DINT 标记的 BOOL 值）。
4. 安装强制值：

要强制：	请执行下列操作：
BOOL 值	右击标记，然后选择 Force ON （强制打开）或 Force OFF （强制关闭）。
非 BOOL 值	在标记的 Force Mask （强制掩码）栏中输入用户希望强制的标记值。然后按 Enter （回车）键。

5. 是否已启用 I/O 强制？（请参阅步骤 1。）

如果：	则：
无	从 Logic （逻辑程序）菜单，选择 I/O Forcing （I/O 强制）> Enable All I/O Forces （启用所有 I/O 强制）。然后选择 Yes （是）确认。
是	结束。

何时使用单步调试

要重写一次错误转变，然后从当前步骤转到下一步，使用 Step Through （单步调试）选项。使用 Step Through （单步调试）选项：

- 用户不需要添加、启用、禁用或删除强制。
- 下一次 SFC 到达转变时，按照转变条件执行。

该选项也让用户重写一次并行路径的错误强制。用户单步调试强制时，SFC 执行路径步骤。

单步调试转变或路径强制

要单步调试当前步骤的转变或并行路径的强制。

1. 打开 SFC 例程。
2. 右击转变或强制的路径，然后选择 Step Through （单步调试）。

何时使用 SFC 强制

要重写 SFC 逻辑程序，用户需要作出下列选择：

如果用户希望：	则：
每次 SFC 到达转变位置时，重写转变条件	强制转变
阻止并行分支一个或多个路径的执行	强制并行路径

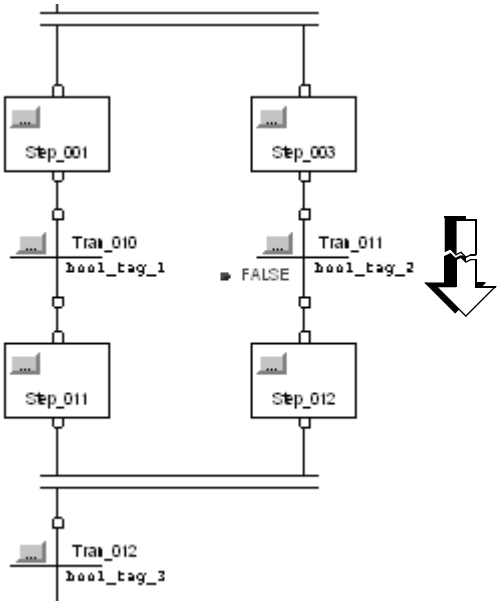
强制转变

要在重复执行 SFC 过程中重写转变条件，请强制转变。强制保留到用户删除或禁用强制。

如果用户希望：	则：
防止 SFC 转到下一步	将转变强制为假
不管转变条件如何，造成 SFC 转到下一步	将转变强制为真

如果用户将并行分支中的转变强制为假，则只要强制激活 （安装并启用），SFC 保持在并行分支。

- 要保留并行分支，每个路径的最后一步必须至少执行一次，且分支下的转变必须为真。
- 将转变强制为假阻止 SFC 到达路径的最后一步。
- 在用户删除或禁用强制时，SFC 可以执行路径中的其它步骤。



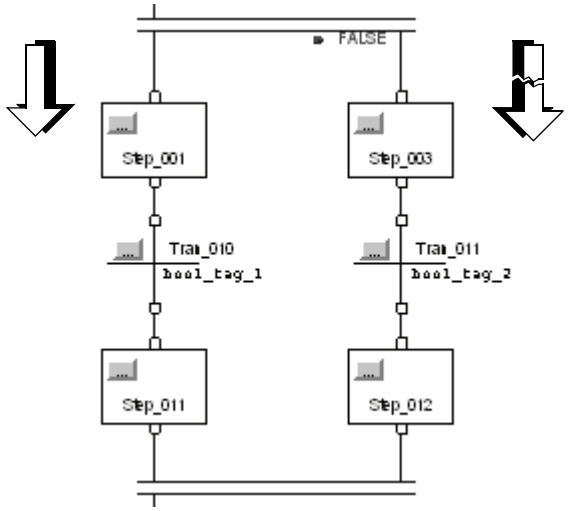
例如：要退出分支，SFC 必须能：

- 至少执行一次 Step_011
- 通过 Tran_011 并至少执行一次 Step_012
- 确定 Tran_012 为真

强制并行路径

要阻止执行并行分支的路径，将路径强制为假。SFC 到达分支时，仅执行非强制路径。

执行该路径。



不执行该路径。

如果用户将并行分支中的路径强制为假，则只要强制激活 （安装并启用），SFC 保持在并行分支。

- 要保留并行分支，每个路径的最后一步必须至少执行一次，且分支下的转变必须为真。
- 将路径强制为假阻止 SFC 输入路径并执行其步骤。
- 在用户删除或禁用强制时，SFC 可以执行路径中的步骤。

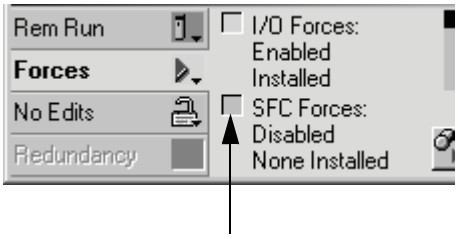
添加 SFC 强制

要重写 SFC 的逻辑程序，使用 SFC 强制：

注意

强制可以引起意想不到的机器运动，可能会造成人员伤亡。 用户在使用强制前，确定强制将对机器或程序造成哪些影响，同时应保证人员远离机器。

- 启用 SFC 强制会引起用户机器或程序转到不同状态或阶段。
- 如果强制启用，且用户安装了强制，新强制立即激活。



1. SFC 强制指示器的状态如何？

如果：	则注意：
关闭	当前不存在任何 SFC 强制。
闪烁	未激活任何 SFC 强制。但用户项目中至少存在一个强制。在用户启用 SFC 强制时，现有的所有 SFC 强制将生效。
恒定	启用 SFC 强制 （激活）。用户安装 （添加）一个强制时，立即生效。

2. 打开 SFC 例程。

3. 右键单击用户希望的转变或并行路径的开始，然后选择 Force TRUE （强制真）或 Force FALSE （强制假）。

4. SFC 强制是否已启用？ （请参阅步骤 。）

如果：	则：
无	从 Logic （逻辑程序）菜单，选择 SFC Forcing （SFC 强制）> Enable All SFC Forces （启用所有 SFC 强制）。 然后选择 Yes （是）确认。
是	结束。

删除或禁用强制

注意

强制的改变会引起意想不到的机器运动，可能会造成人员伤亡。 用户在禁用或删除强制前，确定改变将对机器或程序造成哪些影响，同时应保证人员远离机器。

如果用户希望：	并且：	则：
停止个别强制	保留其它强制处于启用或激活状态	删除个别强制
停止所有 I/O 强制，但保留所有 SFC 强制处于激活状态	保留项目中的 I/O 强制	禁用所有 I/O 强制
	删除项目中的 I/O 强制	删除所有 I/O 强制
停止所有 SFC 强制，但保留所有 I/O 强制处于激活状态	保留项目中的 SFC 强制	禁用所有 SFC 强制
	删除项目中的 SFC 强制	删除所有 SFC 强制

删除个别强制

注意

如果用户删除个别强制，则保持在启用状态的强制和任何新的强制将立即生效。

用户在删除强制前，确定改变将对机器或程序造成哪些影响，同时应保证人员远离机器。

1. 打开包括用户希望删除的强制的例程。
2. 例程使用那些语言？

如果：	则：
SFC	转到步骤 4。
梯形逻辑	转到步骤 4。
功能块	转到步骤 3。
结构文本	转到步骤 3。

3. 右击具有强制的标记，然后选择 Monitor （监视）… 如有必要，展开标记显示用户希望强制的值 （例如：DINT 标记的 BOOL 值）。
4. 右击具有强制的标记或单元，然后选择 Remove Force （删除强制）。

禁用所有 I/O 强制

从 Logic （逻辑程序）菜单，选择 I/O Forcing （I/O 强制） > Disable All I/O Forces （禁用所有 I/O 强制）。然后选择 Yes （是）确认。

删除所有 I/O 强制

从 Logic （逻辑程序）菜单，选择 I/O Forcing （I/O 强制） > Remove All I/O Forces （删除所有 I/O 强制）。 然后选择 Yes （是）确认。

禁用所有 SFC 强制

从 Logic （逻辑程序）菜单，选择 SFC Forcing （SFC 强制）> Disable All SFC Forces （禁用所有 SFC 强制）。然后选择 Yes （是）确认。

删除所有 SFC 强制

从 Logic （逻辑程序）菜单，选择 SFC Forcing （SFC 强制）> Remove All SFC Forces （删除所有 SFC 强制）。 然后选择 Yes （是）确认。

访问状态信息

何时使用此章

Logix5000 控制器与 PLC-5 控制器一样，没有状态文件。要访问状态信息，请使用关键词或访问特定对象。

目的：	参阅页码：
监视状态标志	14-1
获得并设置系统数据	14-2

监视状态标志

控制器支持状态关键词，您可以按您的逻辑程序监视特定事件：

- 状态关键词不区分大小写。
- 由于状态标志更改非常快，RSLogix 5000 软件不能显示标志状态。（即，即使设置状态标志，也不会突出显示引用该标志的指令。）
- 您不能将标记别名定义给关键词。

您可以使用这些关键词：

要确定：	使用：
您存储的数值不适合目标，原因是： • 比目标的最大值大 • 比目标的最小值小 **重要：**每次 S:V 从清除到设置，会产生次故障 （类型 4，代码 4）	S: V
指令目标值为 0	S:Z
指令目标值为负值	S:N
数值运算引起进位或借位，试图使用超出数据类型的位 例如： • 3 + 9 进 1 位 • 25 - 18 借 10	S:C
这是当前程序第一次普通的例程扫描	S:FS
至少已产生一个次故障： • 控制器在由于程序执行出现次故障时设置此位。 • 对于与程序执行无关的次故障，如电池电量低，控制器不设置此位。	S:MINOR

项目中有 SFC 时 S:FS 的状态

S:FS 的状态取决于 SFC 的状态：

- 如果您在流程图（SFC）的操作中使用 S:FS ，则设置 S:FS（on）为每次激活此操作时执行一次扫描。S:FS = *step_name*.FS.

示例

SFC 称为梯形图。

假设 SFC 的多个步骤都调用同一个梯形图例程。假设梯形图使用 S:FS。 每次其中一步被激活，S:FS 开始扫描一次梯形图。

- 如果 SFC 调用一个例程，设置 S:FS（on）为每次调用例程的步骤激活时执行一次扫描。 S:FS = ***step_name***.FS.

  如果 SFC 未调用例程，设置 S:FS（on）为第一次扫描任务。

示例

多个任务但无 SFC

假设您有 2 个使用梯形图的任务。当第一次执行第一个任务时，打开 S:FS 执行一次扫描。扫描后，S:FS 对该任务保持。当第一次执行其它任务时，打开 S:FS 对任务执行一次扫描。 S:FS 继续对第一次执行的任务保持。

关于 S:FS 的更多信息，请参阅第 4-9 页。

获得并设置系统数据

控制器在对象内储存系统数据。如 PLC-5 控制器一样，不存在状态文件。使用 GSV/SSV 指令获取并设置储存在对象内的控制器系统数据。

- GSV 指令检索特定信息，并将其置于目标位置。
- SSV 指令设置来自源位置数据的特定属性。

注意

小心使用 SSV 指令。对象更改会引起意想不到的控制器操作或人员伤亡。

获取或设置系统值：

1. 打开 RSLogix 5000 项目。
2. 从 Help （帮助）菜单，选择 Contents （目录）。
3. 选择索引选项卡。
4. 键入 gsv/ssv 对象，单击 Display （显示）。
5. 选择对象。

获取或设置：	单击：
伺服模块的轴	AXIS
系统开销时间片	CONTROLLER
控制器物理硬盘	CONTROLLERDEVICE
机架中设备的协调系统时间	CST
DF1 串行端口通信驱动程序	DF1
控制器故障历史记录	FAULTLOG
信息指令的属性	MESSAGE
状态、故障和模块模式	MODULE
轴群	MOTIONGROUP
程序的故障信息或扫描时间	PROGRAM
例程的实例号	ROUTINE
串行端口配置	SERIALPORT
任务属性或使用时间	TASK
控制器系统时间	WALLCLOCKTIME

6. 在对象的属性列表中，识别要访问的属性。
7. 为属性值创建标记：

如果属性的数据类型是：	则：
一个元素 （如 DINT）	为属性创建标记。
超过一个元素（如 DINT[7] ）	A. 创建用户定义的数据类型，用于属性且与数据组织匹配。 B. 创建属性标记，从步骤 A. 使用该数据类型。

8. 在梯形逻辑程序例程中，输入合适的指令：

目的：	输入指令：
获取属性值	GSV
设置属性值	SSV

9. 为指令分配所需操作数：

为这些操作数：	选择：
类名	对象名
实例名	特定对象名 （如，所需 I/O 模块名、任务名或信息名） • 并非所有对象都需要这些输入。 • 要指定当前任务、程序或例程，请选择 THIS。
属性名	属性的名称
目标 (GSV)	存储检索值的标记 • 如果标记是用户定义的数据类型或数组。请选择第一个成员或元素。
源 (SSV)	存储设置值的标记 • 如果标记是用户定义的数据类型或数组。请选择第一个成员或元素。

此示例获取当前日期和时间。

示例 获取系统值

第一次扫描时，获取 WALLCLOCKTIME 对象的 DateTime 属性，并存储到 wall_clock 标记中，该标记基于用户定义的数据类型。

42370

更多信息，请参阅 *Logix5000 一般指令集参考手册*，出版物 1756-RM003。

处理主故障

使用本章

使用本章开发可处理特定故障情况的逻辑程序。

有关信息：	请参阅页：
开发故障例程	15-1
以编程方式清除主故障	15-5
预扫描期间清除主故障	15-7
测试故障例程	15-10
创建用户定义的主故障	15-11
主故障代码	15-14

开发故障例程

如果发生足以导致控制器关闭的故障情况，控制器将生成**主故障**并停止执行逻辑。

- 根据应用程序，您可能不希望所有主故障关闭您的整个系统。
- 在这些情况下，可以使用故障例程清除特定故障并至少使部分系统继续运行。

示例

使用故障例程

在将配方号用作间接地址的系统中，类型错误的编号可能产生主故障，例如类型 4 代码 20。

要使整个系统不关机，故障例程清除任何类型 4 代码 20 主故障。

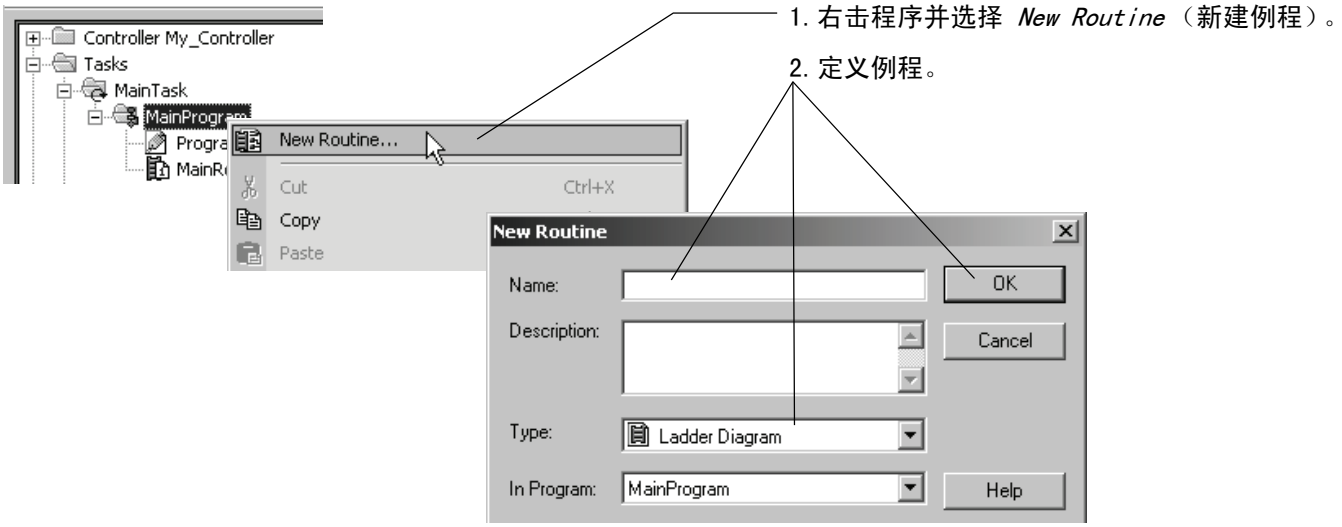
选择放置故障例程的位置

故障例程使程序逻辑在故障后采取特定操作，例如清除故障并继续执行。放置例程的位置取决于要处理的故障类型：

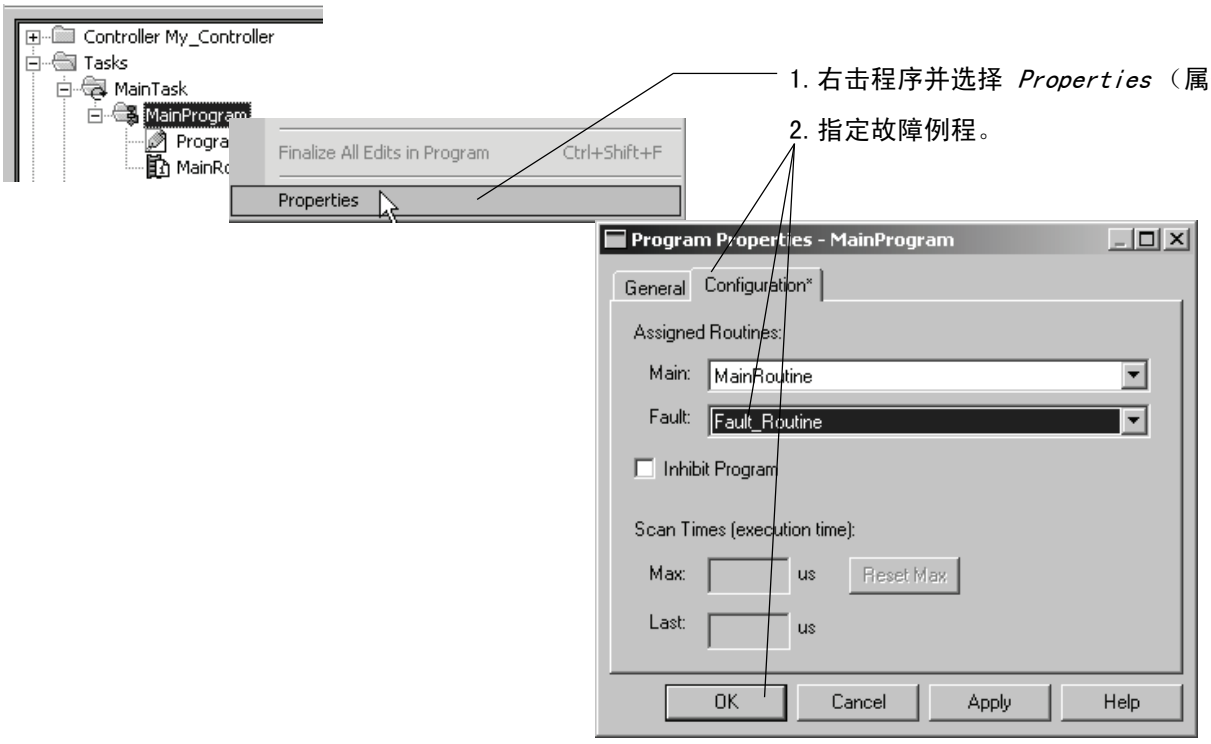
如果希望在以下情况下采取特定操作 / 清除故障：		执行：	请参阅页：
情况：	故障类型：		
指令执行出现故障	4	为程序创建故障例程	15-2
与 I/O 模块的通信失败	3	为控制器故障处理程序创建例程	15-3
任务的 Watchdog 时间超时	6		
项目正在下载到控制器时，按键开关放置在 RUN	8		
运动轴发生故障	11		
控制器以运行 / 远程运行模式启动	1	为启动处理程序创建例程	15-4

为程序创建故障例程

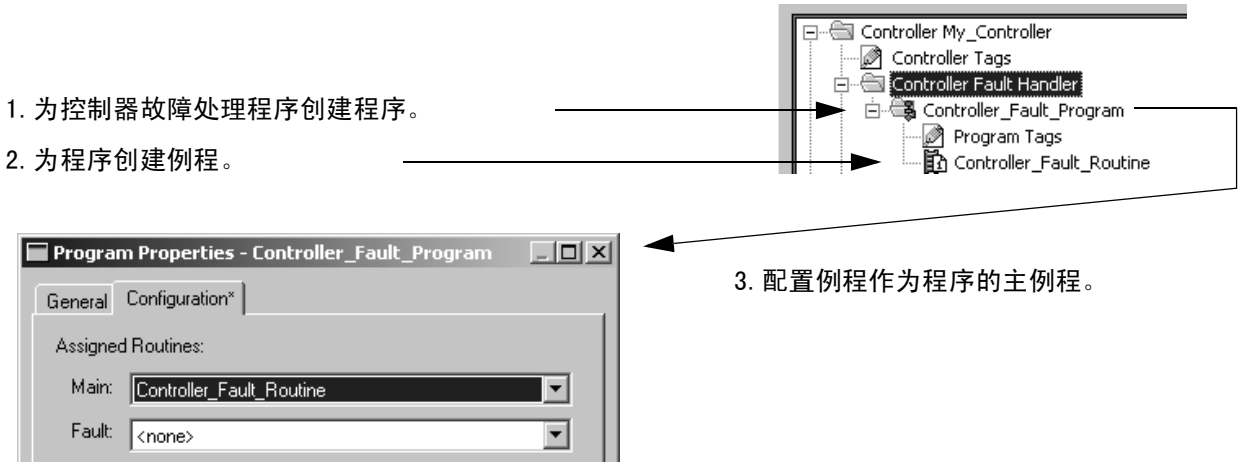
创建例程



将例程指定为故障例程



为控制器故障处理程序创建例程

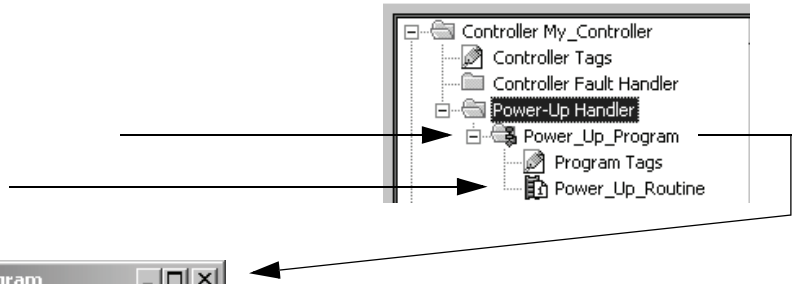


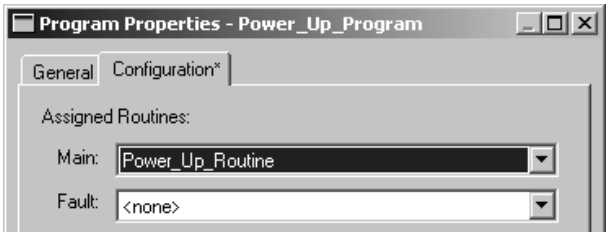
为启动处理程序创建例程

通电处理程序是控制器以运行 / 远程运行模式启动时执行的可选任务。当您希望在断电并恢复后完成以下任一操作时使用启动处理程序：

要：	执行：
防止控制器返回运行 / 远程模式	将例程的 Power-Up Handler （启动处理程序）条目留空。恢复电源后，主故障 （类型 1 代码 1）发生并且控制器进入故障模式。
恢复电源时，采取特定操作然后恢复正常运行	在启动处理程序的例程中： 1. 清除主故障 （类型 1 代码 1）。 2. 输入操作的逻辑。

1. 为启动处理程序创建程序。
2. 为程序创建例程。





3. 配置例程作为程序的主例程。

以编程方式清除主故障

若要清除执行项目期间发生的主故障，请在合适的例程中完成以下操作。（请参阅第 15-2 页上的选择放置故障例程的位置）。

- 创建存储故障信息的数据类型
- 获得故障类型和代码
- 检查特定故障
- 清除故障

创建存储故障信息的数据类型

Logix5000 控制器在对象中存储系统信息。与 PLC-5 或 SLC 500 控制器不同，没有状态文件。

- 若要访问系统信息，请使用获得系统值 (GSV) 或设置系统值 (SSV) 指令。
- 对于程序的状态信息，可访问 PROGRAM 对象。
- 对于故障信息，可访问 PROGRAM 对象的这些属性。

属性：	数据类型：	指令：	说明：
MajorFaultRecord	DINT[11]	GSV SSV	记录此程序的主故障 指定程序名称以确定所需的 PROGRAM 对象。（或指定 THIS 访问包含 GSV 或 SSV 指令的程序的 PROGRAM 对象）。

若要简化对 MajorFaultRecord 属性的访问，可创建下面的用户定义的数据类型：

创建新数据类型：

右击并选择 *New Data Type*（新建数据类型）。

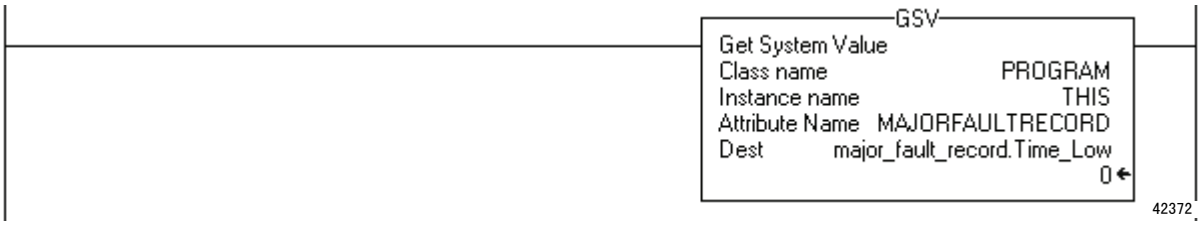
数据类型：FAULTRECORD

名称	FAULTRECORD
说明	存储 PROGRAM 对象的 MajorFaultRecord 属性或 MinorFaultRecord 属性。

成员

	名称	数据类型	样式	说明
	Time_Low	DINT	十进制	故障时间戳值的低 32 位
	Time_High	DINT	十进制	故障时间戳值的高 32 位
	类型	INT	十进制	故障类型 （程序、I/O 等）
	代码	INT	十进制	故障唯一代码
	信息	DINT[8]	十六进制	故障特定信息

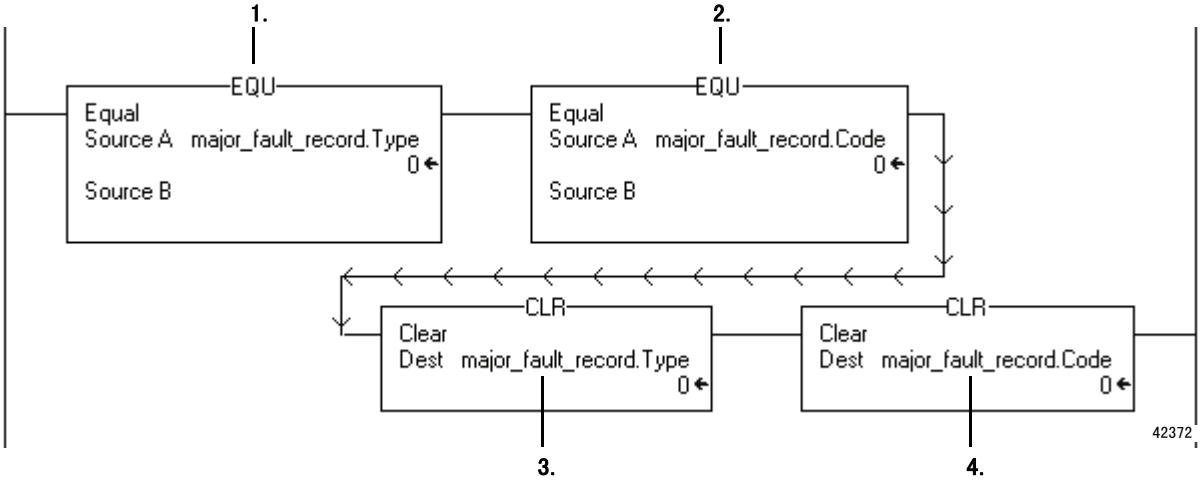
获得故障类型和代码



1. GSV 指令访问此程序的 MAJORFAULTRECORD 属性。此属性存储故障的相关信息。

2. GSV 指令在 major_fault_record 标记 （FAULTRECORD 类型）中存储故障信息。输入基于结构的标记时，请输入标记的第一个成员。

检查特定故障



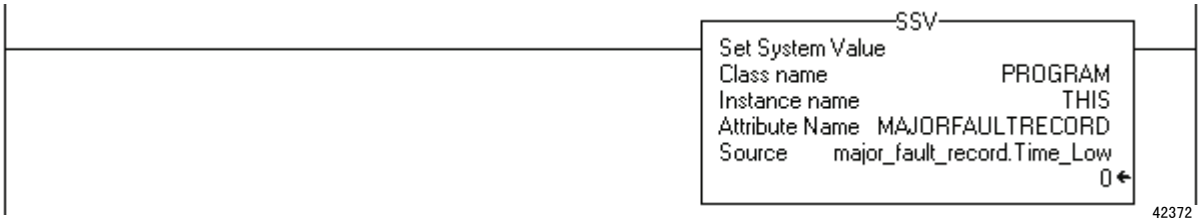
1. 第一个 EQU 指令检查特定类型故障，例如程序、I/O。在源 B 中，输入要清除的故障类型的值。

2. 第二个 EQU 指令检查特定故障代码。在源 B 中，输入要清除的代码的值。

3. 第一个 CLR 指令将 major_fault_record 标记中故障类型的值设置为零。

4. 第二个 CLR 指令将 major_fault_record 标记中故障代码的值设置为零。

清除故障



1. SSV 指令向此程序的 MAJORFAULTRECORD 属性写入新值。
2. SSV 指令写入包含在 major_fault_record 标记中的值。由于 Type 和 Code 成员设置为零，故障清除并且控制器继续执行。

预扫描期间清除主故障

如果将控制器切换为运行模式后控制器立刻发生故障，则检查**预扫描**操作寻找故障。根据控制器的版本不同，预扫描期间超过数组范围的数组下标可能产生故障。

如果控制器版本为：	则：
11.x 或更低	在预扫描期间，超过数组范围的数组下标产生主故障。
12.x	请参阅控制器固件发布声明。
13.0 或更高	在预扫描期间，控制器自动清除任何因超过数组范围的数组下标引起的故障。

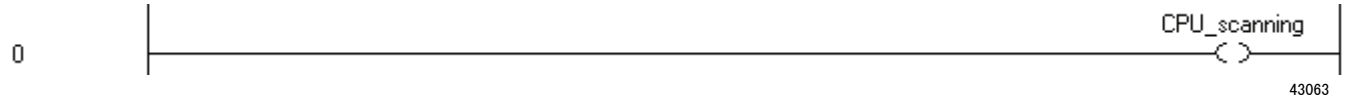
清除预扫描期间发生的主故障：

- 确定控制器何时处于预扫描
- 获得故障类型和代码
- 检查特定故障
- 清除故障

重要 清除故障前检查特定故障是一个良好的编程行为。

确定控制器何时处于预扫描

在程序主例程中，将此梯级作为程序主例程的第一个梯级输入：



此程序的故障例程使用该位的状态来确定故障是在预扫描期间还是逻辑的正常扫描期间发生的：

- 预扫描期间，该位为 off。（预扫描期间，控制器重置 OTE 指令引用的所有位。）
- 控制器开始执行逻辑后，CPU_scanning 位始终为 on。

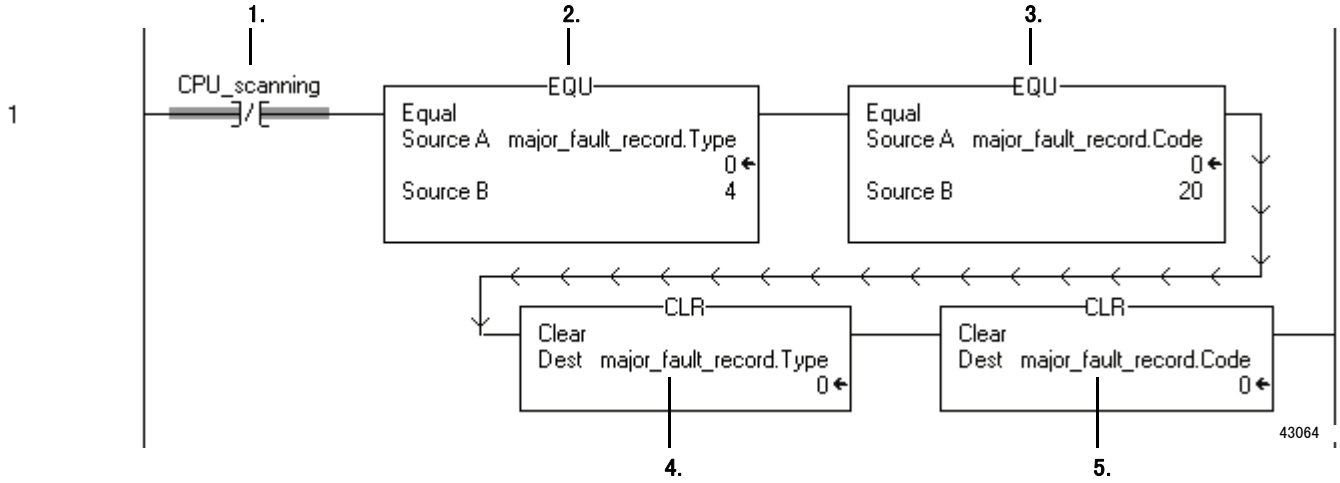
获得故障类型和代码

在程序的故障例程中输入此梯级：

1. GSV 指令访问此程序的 MAJORFAULTRECORD 属性。 此属性存储故障的相关信息。
2. GSV 指令在 major_fault_record （FAULTRECORD 类型）标记中存储故障信息。输入基于结构的标记时，请输入标记的第一个成员。

检查特定故障

在程序的故障例程中输入此梯级：



1. 预扫描期间，所有 OTE 指令的位都关闭，此指令为 true。控制器开始执行逻辑后，此指令始终为 false。

2. 第一个 EQU 指令检查类型 4 故障，这表示此程序中的指令引起故障。

3. 第二个 EQU 指令检查代码 20 故障，这表示数组下标过大，或者 CONTROL 结构的 POS 或 LEN 值无效。

4. 第一个 CLR 指令将 major_fault_record 标记的故障类型的值设置为零。

5. 第二个 CLR 指令将 major_fault_record 标记的故障代码的值设置为零。

清除故障

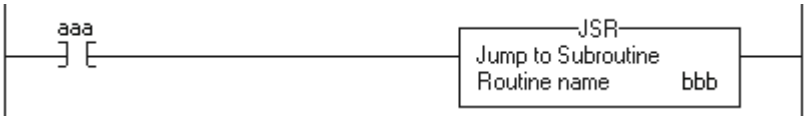
在程序的故障例程中输入此梯级：

1. 预扫描期间，所有 OTE 指令的位都关闭，此指令为 true。 控制器开始执行逻辑后，此指令始终为 false。
2. SSV 指令向此程序的 MAJORFAULTRECORD 属性写入新值。
3. SSV 指令写入包含在 major_fault_record 标记中的值。 由于 Type 和 Code 成员设置为零，故障清除并且控制器继续执行。

测试故障例程

可以使用 JSR 指令测试程序的故障例程，无需创建错误 （即模拟故障）：

1. 创建将用于启动故障的 BOOL 标记。
2. 在程序的主例程或子例程中，输入此梯级：

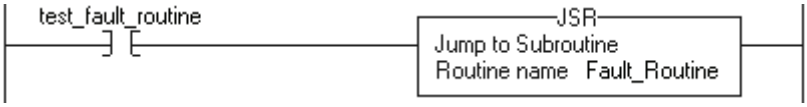


其中：	为：
aaa	将用于启动故障的标记
bbb	程序的故障例程

3. 若要模拟故障，请设置输入条件。

示例 测试故障例程

当 test_fault_routine 为 on 时，主故障发生并且控制器执行 Fault_Routine。



创建用户定义的主故障

如果要根据应用程序中的情况暂停 （关闭）控制器，请创建用户定义的主故障。用户定义的主故障：

- 故障类型 = 4。
- 定义故障代码的值。选择 990 至 999 之间的值。这些代码为用户定义的故障保留。
- 控制器将这些故障与其他主故障一样处理：
  - 控制器更改为**故障模式** （主故障）并停止执行逻辑。
  - 输出设置为故障模式配置的状态或值。

示例 用户定义的主故障

当 Tag_1.0 = 1 时，产生主故障并生成故障代码 999。

创建用户定义的主故障：

- 为程序创建故障例程
- 配置程序使用故障例程
- 跳转至故障例程

为程序创建故障例程

是否已存在程序的故障例程？

如果：	则：
是	转至第 15-12 页的 “跳转至故障例程”。
否	为程序创建故障例程：

1. 在控制器组织器中，右击程序并选择 New Routine （新建例程）。
2. 在名称框中键入故障例程的名称。
3. 从 Type （类型）下拉列表中选择 Ladder （梯级）。
4. 单击 OK （确定）。

配置程序使用故障例程

1. 在控制器组织器中，右击程序并选择 New Routine （新建例程）。
2. 选择 Configuration （配置）选项卡。
3. 从 Fault （故障）下拉列表中选择故障例程。
4. 单击 OK （确定）。

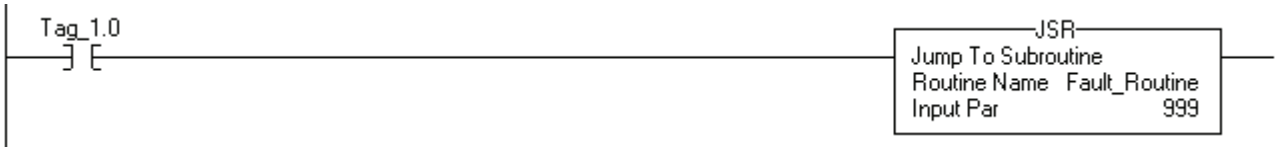
跳转至故障例程

在程序的主例程中输入此梯级：

其中：	为：
Fault_Routine	程序的故障例程的名称
999	故障代码的值

示例 创建用户定义的主故障

当 Tag_1.0 = 1 时，执行跳转至 name_of_fault_routine。主故障发生，控制器进入故障模式。输出转为故障状态。Controller Properties （控制器属性）对话框 Major Faults （主故障）选项卡显示代码 999。



主故障代码

类型和代码对应于显示在以下位置的类型和代码：

- Controller Properties （控制器属性）对话框 Major Faults （主故障）选项卡
- PROGRAM 对象 MAJORFAULTRECORD 属性

表 15.1 主故障类型和代码

类型：	代码：	原因：	恢复方法：
1	1	控制器以运行模式启动。	执行断电处理程序。
1	60	对于没有 安装 CompactFlash 卡的控制器，控制器： • 检测到不可恢复的故障 • 从内存清除项目	1. 清除故障。 2. 下载项目。 3. 更改为远程运行 / 运行模式。 如果问题继续： 1. 关闭控制器电源并重新启动前，记录 OK （确定）和 RS232 LED 的状态。 2. 联系 Rockwell Automation 支持。请参阅本出版物的封底。
1	61	对于安装有 CompactFlash 卡的控制器，控制器： • 检测到不可恢复的故障 • 向 CompactFlash 卡写入诊断信息 • 从内存清除项目	1. 清除故障。 2. 下载项目。 3. 更改为远程运行 / 运行模式。 如果问题继续存在，请联系 Rockwell Automation 支持。请参阅本出版物的封底。
3	16	必需的 I/O 模块连接出现故障。	检查 I/O 模块是否在机箱中。检查电子键控要求。 查看控制器属性 Major Fault （主故障）选项卡和模块属性 Connection （连接）选项卡了解故障的更多信息。
3	20	ControlBus 机架可能存在的问题。	不可恢复 - 更换机架。
3	23	进入运行模式前至少有一个必需连接没有建立。	更改为运行模式前等待控制器 I/O 灯变绿。
4	16	遇到未知指令。	删除未知指令。这可能是因为程序转换过程而产生。
4	20	数组下标过大，控制结构 .POS 或 .LEN 无效。	调整该值在有效范围内。不要超过数组大小或定义的维度。
4	21	控制器结构 .LEN 或 .POS < 0。	调整该值使其 > 0。
4	31	JSR 指令的参数不匹配相关 SBR 或 RET 指令的参数。	传递合适数量的参数。如果传递过多参数，多出的参数将忽略，不产生错误。
4	34	计时器指令具有负预设或累积值。	修正程序，不将负值加载到计时器预设或累积值。
4	42	JMP 至不存在或已删除的标签。	纠正 JMP 目标或添加丢失的标签。
4	82	流程图 (SFC) 调用子例程，子例程尝试跳转回调用 SFC。当 SFC 使用 JSR 或 FOR 指令调用子例程时发生。	删除向调用 SFC 的跳转。
4	83	测试的数据不在所需限制内。	修改值使其在限制内。
4	84	堆栈溢出。	减小子例程嵌套级别或传递的参数数量。
4	89	在 SFR 指令中，目标例程不包含目标步骤。	纠正 SFR 目标或添加缺少的步骤。

表 15.1 主故障类型和代码 （续）

类型：	代码：	原因：	恢复方法：
6	1	任务 Watchdog 超时。 用户任务没有在指定时间内完成。程序错误导致无限循环，或者程序太复杂以至于无法尽快执行，或者较高优先级任务使此任务无法完成。	增加任务 Watchdog，缩短执行时间，提高此任务的优先级，简化较高优先级任务，或者将部分代码转移到其他控制器。
7	40	向非易失性内存的存储失败。	1. 再次尝试向非易失性内存存储项目。 2. 如果项目无法存储到非易失性内存，请更换内存板。
7	41	从非易失性内存的加载因控制器类型不匹配而失败。	更改为正确类型的控制器或下载项目并存储在 CompactFlash 卡上。
7	42	从非易失性内存的加载失败，原因是非易失性内存中项目的固件版本不匹配控制器的固件版本。	将控制器固件更新到非易失性内存中项目相同的版本级别。
7	43	从非易失性内存的加载因错误的校验和而失败。	联系 Rockwell Automation 支持。 请参阅本出版物的封底。
7	44	无法还原处理器内存。	联系 Rockwell Automation 支持。 请参阅本出版物的封底。
8	1	下载期间尝试用按键开关将控制器设为运行模式。	等待下载完成并清除故障。
11	1	实际位置超过超行程正限制。	向负方向移动轴直到位置在超行程限制内，然后执行 Motion Axis Fault Reset （运动轴故障重置）。
11	2	实际位置超过超行程负限制。	向正方向移动轴直到位置在超行程限制内，然后执行 Motion Axis Fault Reset （运动轴故障重置）。
11	3	实际位置超过位置错误容差。	将位置移动到容差内，然后执行 Motion Axis Fault Reset （运动轴故障重置）。
11	4	编码器通道 A、B 或 Z 连接断开。	重新连接编码器通道，然后执行 Motion Axis Fault Reset （运动轴故障重置）。
11	5	检测到编码器噪声事件，或编码器信号不在区间内。	修正编码器布线，然后执行 Motion Axis Fault Reset （运动轴故障重置）。
11	6	驱动器故障输入被激活。	清除驱动器故障，然后执行 Motion Axis Fault Reset （运动轴故障重置）。
11	7	同步连接引起故障。	首先执行 Motion Axis Fault Reset （运动轴故障重置）。如果不起作用，将伺服模块拉出再重新插入。如果都不起作用，则更换伺服模块。
11	8	伺服模块检测到严重硬件故障。	更换模块。
11	9	异步连接引起故障。	首先执行 Motion Axis Fault Reset （运动轴故障重置）。 如果不起作用，将伺服模块拉出再重新插入。如果都不起作用，则更换伺服模块。
11	32	运动任务发生重叠。	组的进程更新速率过高以至于无法维持正确运行。清除组故障标记，提高组更新速率，然后清除主故障。

监视次故障

何时使用本章

如果发生不足以使控制器关闭的故障情况，控制器生成**次故障**。

- 控制器继续执行。
- 无需清除次故障。
- 若要优化执行时间并确保程序准确度，应监视和纠正次故障。

监视次故障

使用梯级逻辑获取次故障的信息：

检查：	执行：
定期任务重叠	1. 输入获取 FAULTLOG 对象 MinorFaultBits 属性的 GSV 指令。 2. 监视位 6。
从非易失性内存加载	1. 输入获取 FAULTLOG 对象 MinorFaultBits 属性的 GSV 指令。 2. 监视位 7。
串行端口问题	1. 输入获取 FAULTLOG 对象 MinorFaultBits 属性的 GSV 指令。 2. 监视位 9。
电量不足	1. 输入获取 FAULTLOG 对象 MinorFaultBits 属性的 GSV 指令。 2. 监视位 10。
指令问题	1. 创建存储故障信息的用户定义的数据类型：命名数据类型 FaultRecord 并指定以下成员：

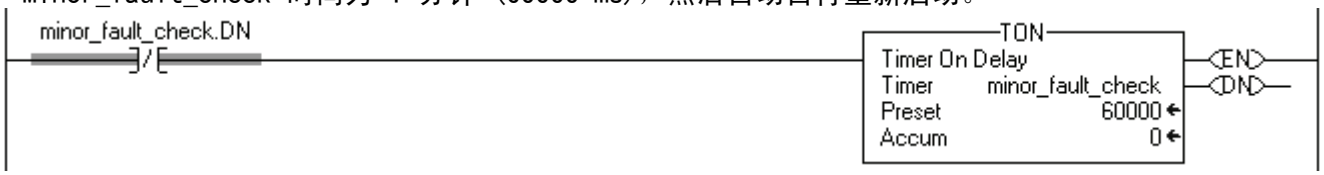
名称：	数据类型：	样式：
TimeLow	DINT	十进制
TimeHigh	DINT	十进制
类型	INT	十进制
代码	INT	十进制
信息	DINT[8]	十六进制

2. 创建将存储 MinorFaultRecord 属性值的标记。选择第 1. 步的数据类型。
3. 监视 S:MINOR。
4. 如果 S:MINOR 为 on，则使用 GSV 指令获得 MinorFaultRecord 属性值。
5. 如果希望检测其他指令引起的次故障，请重置 S:MINOR。（S:MINOR 保持设置直到扫描结束。）

此示例检查电量不足警告。

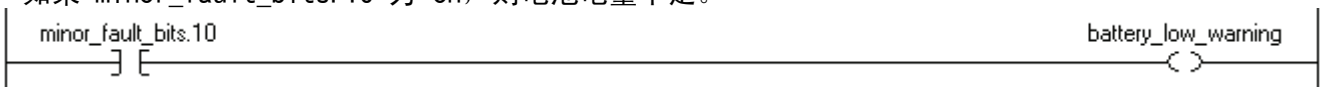
示例 检查次故障

Minor_fault_check 时间为 1 分钟 (60000 ms)，然后自动自行重新启动。



minor_fault_check.DN 每分钟为变为 on 执行一次扫描。发生此情况时，GSV 指令获得 FAULTLOG 对象 MinorFaultBits 属性值，并存储到 minor_fault_bits 标记中。因为 GSV 指令每分钟仅执行一次，所以大部分扫描的扫描时间减少。

如果 minor_fault_bits.10 为 on，则电池电量不足。



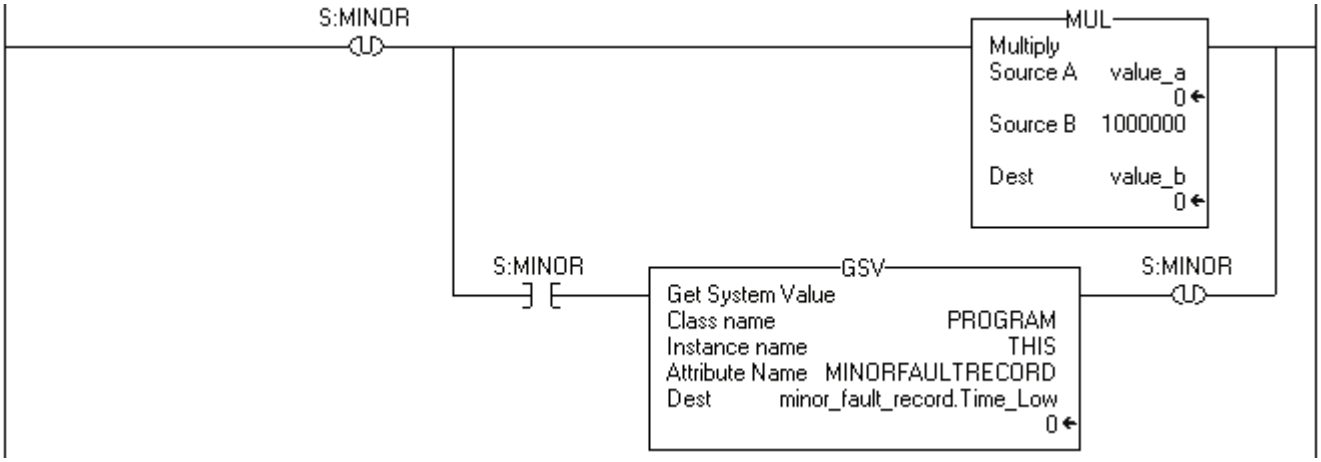
42373

此示例检查特定指令引起的次故障。

示例 检查指令引起的次故障

将 value_a 乘以 1000000 并检查次故障，例如数学溢出：

- 为确保之前的指令不产生故障，梯级首先清除 S:MINOR。
- 然后梯级执行乘指令。
- 如果指令产生次故障，则控制器设置 S:MINOR。
- 如果 S:MINOR 已设置，则 GSV 指令获得故障的信息并重置 S:MINOR。



42373

次故障代码

类型和代码对应于显示在以下位置的类型和代码：

- Controller Properties （控制器属性）对话框 Minor Faults （次故障）选项卡
- PROGRAM 对象 MINORFAULTRECORD 属性

表 16.1 次故障类型和代码

类型：	代码：	原因：	恢复方法：
4	4	指令中发生算术溢出。	通过检查算术运算 （顺序）或调整值来修正程序。
4	5	在 GSV/SSV 指令中，找不到指定实例。	检查实例名称。
4	6	在 GSV/SSV 指令中： • 指定的 Class （类）名称不受 支持 • 指定的 Attribute （属性）名称无效	检查 Class （类）名称和 Attribute （属性）名称。
4	7	GSV/SSV 目标标记过小而无法保存所有数据。	修改目标使其具有足够空间。
4	35	PID 时间增量 ≤ 0。	调整 PID 时间增量使其 > 0。
4	36	PID 设置点超出范围	调整设置点使其在范围内。
4	51	字符串标记的 LEN 值大于字符串标记的 DATA 大小。	1. 检查确保没有指令向字符串标记的 LEN 成员写入。 2. 在 LEN 值中，输入字符串包含的字符数量。
4	52	输出字符串大于目标。	创建对于输出字符串足够大的新字符串数据类型。使用新字符串数据类型作为目标的数据类型。
4	53	输出数字超过目标数据类型的限制。	或者： • 减小 ASCII 值的大小。 • 对目标使用更大的数据类型。
4	56	Start （开始）或 Quantity （数量）值无效。	1. 检查 Start （开始）值是否在 1 和 Source （源）的 DATA 大小之间。 2. 检查 Start （开始）值和 Quantity （数量）值的和是否小于等于 Source （源）的 DATA 大小。
4	57	AHL 指令因串行端口设置为不握手而无法执行。	或者： • 更改串行端口的 Control Line （控制线）设置。 • 删除 AHL 指令。
6	2	定期任务重叠。 定期任务在再次执行时尚未完成。	简化程序，或者延长时间，或者提高相对优先级等。
7	49	项目从非易失性内存加载。	
9	0	服务串行端口时发生未知错误。	联系 GTS 工作人员。

表 16.1 次故障类型和代码 （续）

类型：	代码：	原因：	恢复方法：
9	1	CTS 线对于当前配置不正确。	断开并重新连接串行端口电缆与控制器。 确保电缆正确连线
9	2	轮询列表错误。 检测到 DF1 主轮询列表问题，例如指定站数超过文件大小，指定超过 255 站，尝试索引经过列表结尾，或者轮询广播地址 (STN #255)。	检查轮询列表中的以下错误： • 总站数大于轮询列表标记中的空间 • 总站数大于 255 • 当前站指针大于轮询列表标记结尾 • 遇到大于 254 的站编号
9	5	DF1 从轮询超时。 从轮询监视超时。主轮询没有在指定时间量内轮询此控制器。	确定并纠正轮询延迟。
9	9	调制解调器连接丢失。 DCD 和 / 或 DSR 控制线没有以正确顺序和 / 或状态接收。	纠正与控制器的调制解调器连接。
10	10	没有检测到电池或需要更换电池。	安装新电池。

注释：

使用非易失性内存存储和加载项目

何时使用本章

重要

当您存储项目时，非易失性内存存储用户内存的内容。

- 存储项目后所做的更改不影响非易失性内存。
- 如果对项目更改但不存储这些更改，在从非易失性内存加载项目时将覆盖更改。如果发生此情况，必须上载或下载项目以联机。
- 如果希望存储更改，例如联机编辑、标记值或者 ControlNet 网络调度计划，请在更改后再次存储项目。

使用此步骤利用控制器的**非易失性内存存储**或**加载**项目。

- 如果控制器断电或没有足够的电池电量，将丢失用户内存中的项目。
- 非易失性内存使您在控制器上保留项目的副本。控制器无需电源即可保留此副本。
- 可以将副本从非易失性内存加载到控制器的用户内存中。
  - 每次通电时
  - 控制器中没有项目而通电时
  - 通过 RSLogix 5000 软件时

使用非易失性内存前

存储或加载具有以下参数：

参数：	存储：	加载：
存储或加载需要多少时间？	如果控制器不 使用 1784-CF64 Industrial CompactFlash 卡，存储最多可能要 3 分钟。如果控制器使用 CompactFlash 卡，存储速度显著提高 （不足一分种）。	数秒
可以以哪些控制器模式存储或加载项目？	程序模式	
存储或加载时可以与控制器联机吗？	否	
存储或加载时 I/O 状态是什么？	I/O 保留编程模式的配置状态。	

选择具有非易失性内存的控制器

以下 Logix5000 控制器具有用于项目存储的非易失性内存。

控制器类型：	类别 #：	固件版本：	需要 1784-CF64 Industrial CompactFlash 内存卡：
CompactLogix	1769-L31	13.x 或更高版本	是
	1769-L32E	13.x 或更高版本	是
	1769-L35CR	13.x 或更高版本	是
	1769-L35E	12.x 或更高版本	是
ControlLogix	1756-L55M22	10.x 或更高版本	否
	1756-L55M23	8.x 或更高版本	否
	1756-L55M24	8.x 或更高版本	否
	1756-L60M03SE	13.x 或更高版本	是
	1756-L61	12.x 或更高版本	是
	1756-L62	12.x 或更高版本	是
	1756-L63	11.x 或更高版本	是
DriveLogix	5720	10.x 或更高版本	否
	5730	13.x 或更高版本	是
FlexLogix	1794-L34/B	11.x 或更高版本	否

防止加载时出现主故障

如果非易失性内存中项目的主次版本不匹配控制器的主次版本，则加载期间可能发生主故障。

如果控制器：	则：
不 使用 CompactFlash 卡	确保非易失性内存中项目的主次版本匹配控制器的主次版本。 控制器的非易失性内存仅存储项目。它不存储控制器的固件。
使用 CompactFlash 卡	CompactFlash 卡为项目 ≥ 12.0 存储固件。根据控制器的当前版本，可以使用 CompactFlash 卡更新控制器固件并加载项目。 请参见第 第 17-4 页 页上的 “确定如何处理固件更新”。

格式化 CompactFlash 卡

向 1784-CF64 Industrial CompactFlash 内存卡存储项目时，控制器格式化卡 （如果需要）。

如果项目版本为：	则：	
11.x	CompactFlash 卡使用特别格式。 • 仅使用 Logix5000 控制器在 CompactFlash 卡上存储项目。 • 仅存储单个 Logix5000 项目，CompactFlash 卡上没有其他数据。 • 在 CompactFlash 卡上存储项目时，覆盖卡上的全部内容。换句话说，丢失卡上当前的所有内容。	
≥ 12.0	CompactFlash 卡使用 FAT16 文件系统。	
	如果卡：	**则控制器：**
	已经为 FAT16 文件系统格式化	• 留下现有数据。 • 为项目和固件创建文件夹和文件。
	没有 为 FAT16 文件系统格式化	• 删除现有数据。 • 为 FAT16 文件系统格式化卡。 • 为项目和固件创建文件夹和文件。
	为 FAT16 文件系统格式化 CompactFlash 卡后： • CompactFlash 卡可以存储多个项目和关联固件。 • 如果 CompactFlash 卡已经包含具有同名的项目，存储将覆盖 CompactFlash 卡上的该项目。 • CompactFlash 卡加载最近存储的项目。 使用版本 ≥ 12.0，还可以使用 CompactFlash 读取器读取和操作 CompactFlash 卡上的文件。请参见第 17-14 页上的 “使用 CompactFlash 读取器”。	

确定如何处理固件更新

下表列出更新具有非易失性内存的控制器固件的选项和预防措施。

<table>
<tr><th>如果：</th><th>则：</th></tr>
<tr><td>满足以下全部 条件：
❑控制器使用 1784-CF64 Industrial CompactFlash 卡。
❑CompactFlash 卡上的项目版本为 ≥ 12.0。
❑CompactFlash 卡上的项目的 Load Image（加载映像）选项 = On Power Up（通电时）或 On Corrupt Memory（破坏内存时）。
❑如果控制器为 1756-L63 控制器，则固件版本为：
❑对于新开箱的控制器，版本 ≥ 1.4。（在控制器侧部或箱外寻找 F/W REV.。）
❑对于服务中的控制器，版本 ≥ 12.0。</td><td>使用以下任一内容更换固件：
• CompactFlash 卡
• RSLogix 5000 软件
• ControlFlash 软件
使用 CompactFlash 卡更新固件并加载项目：
1. 将卡安装到控制器中。
2. 如果 Load Image （加载映像）选项 = On Corrupt Memory （破坏内存时）并且控制器包含项目，请从控制器上断开电池。
3. 打开或关闭控制器再重新打开。
如果使用 RSLogix 5000 软件或 ControlFlash 软件更新固件：
1. 更新过程中，控制器将 CompactFlash 卡的 Load Image （加载映像）选项设置为 User Initiated （用户启动）。为避免此情况，从控制器取下卡。
2. 更新固件后，再次将项目存储到非易失性内存。这样确保非易失性内存中的项目版本匹配控制器的版本。</td></tr>
<tr><td>不 满足以上列出的全部 条件：</td><td>使用以下任一内容更换固件：
• RSLogix 5000 软件
• ControlFlash 软件
采取预防措施：
1. 更新固件前：
<table><tr><th>如果控制器：</th><th>则：</th></tr><tr><td>不使用 CompactFlash 卡</td><td>将项目保存到脱机文件。更新控制器固件时，擦除非易失性内存的内容 （版本 10.x 或更高）。</td></tr><tr><td>使用 CompactFlash 卡</td><td>或者：
• 从控制器取下 CompactFlash 卡。
• 检查 CompactFlash 卡的 Load Image （加载映像）选项。如果设置为 On Power Up （通电时）或 On Corrupt Memory（破坏内存时），首先在 Load Image （加载映像）选项设为 User Initiated （用户启动）的情况下存储项目。
否则，可能在更新控制器固件时得到主故障。产生的原因是 On Power Up （通电时）或 On Corrupt Memory （破坏内存时）选项导致控制器从非易失性内存加载项目。然后加载后的固件不匹配导致主故障。</td></tr></table>2. 更新固件后，再次将项目存储到非易失性内存。 这样确保非易失性内存中的项目版本匹配控制器的版本。</td></tr>
</table>

选择加载映像的时机

对于何时 （何种条件下）将项目加载回控制器的用户内存 (RAM) 中有多个选项：

如果希望加载它：	则选择：	注释：

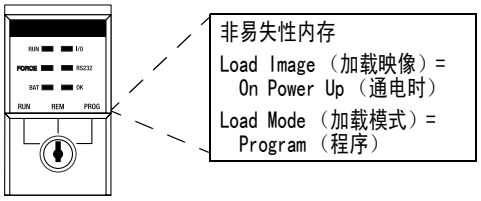
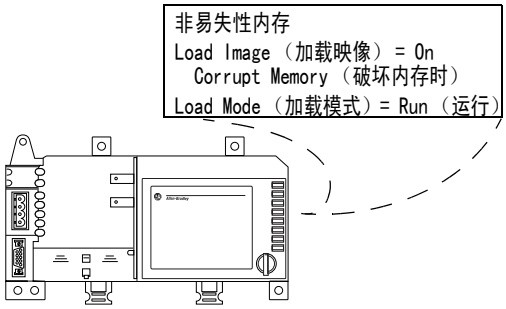
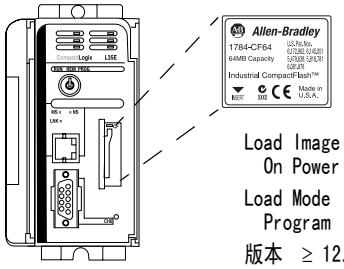
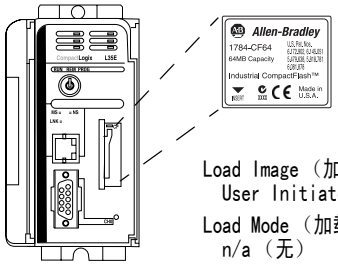
在打开或关闭机架电源再重新打开时	On Power Up
	（通电时）
		• 在关闭电源并重新打开的过程中，将丢失没有存储在非易失性内存中的所有联机更改、标记值和网络计划。
		• 1784-CF64 Industrial CompactFlash 卡也将更改控制器固件。
		  • 如果 CompactFlash 卡上项目的版本和控制器固件版本都为 ≥ 12.0 将发生此情况。
		  • 有关更多信息，请参见第 第 17-4 页 页上的 “确定如何处理固件更新”。
		• 始终可以使用 RSLogix 5000 软件加载项目。

当控制器中没有项目并且打开或关闭机架电源再打开时	On Corrupt Memory
	（破坏内存时）
		• 例如，如果电池电量用光或者控制器断电，项目将从内存中清除。恢复电源时，此加载选项将项目加载回控制器中。
		• 1784-CF64 Industrial CompactFlash 卡也将更改控制器固件。
		  • 如果 CompactFlash 卡上项目的版本和控制器固件版本都为 ≥ 12.0 将发生此情况。
		  • 有关更多信息，请参见第 第 17-4 页 页上的 “确定如何处理固件更新”。
		• 始终可以使用 RSLogix 5000 软件加载项目。

仅通过 RSLogix 5000 软件	User Initiated （用户启动）	

示例

下面是一些使用不同加载选项的示例用途：

示例：	说明：
1. 	1. 将控制器固件更新为所需版本。 2. 在非易失性内存中为控制器存储项目。 3. 在安装后打开控制器电源时，项目加载到控制器中。 4. 控制器保留程序模式。
2. 	1. 在非易失性内存中为控制器存储项目。 （控制器中固件的主次版本匹配非易失性内存中项目的主次版本。） 2. 如果控制器电池完全用光电量而且控制器的电源中断，项目将从控制器内存中清除。 3. 电源恢复时，项目自动加载到控制器中并且控制器返回运行模式。
3. 	1. 控制器出现故障。 2. 取下 CompactFlash 卡。 3. 用新的控制器更换出现故障的控制器。 4. 更换 CompactFlash 卡。 5. 打开电源时，固件和项目都加载到控制器中。控制器保留程序模式。
4. 	1. 希望将不同项目加载到控制器中。 2. CompactFlash 卡包含所需项目。 3. 当控制器中安装有 CompactFlash 卡时，使用 RSLogix 5000 软件将项目加载到控制器中。

存储项目

在此任务中，在控制器的非易失性内存中存储项目。

注意

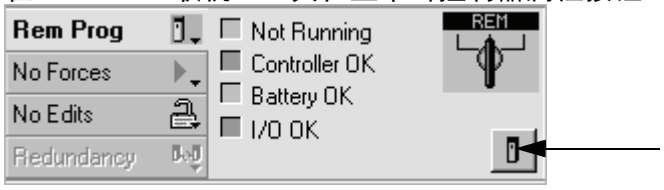
存储过程中，所有活动伺服轴关闭。存储项目前，确保这不会导致轴的意外移动。

存储项目前：

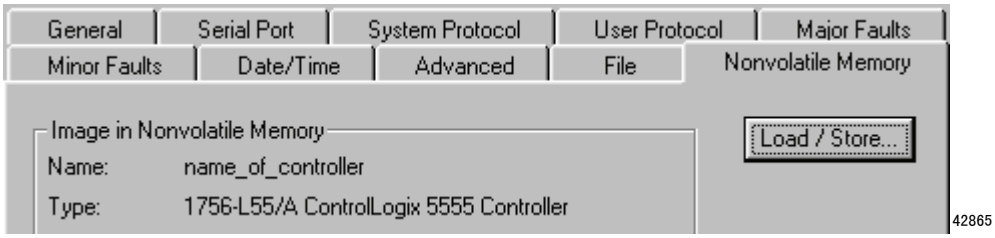
- 对逻辑进行所有必要的编辑
- 将项目下载到控制器
- 安排 ControlNet 网络计划

存储项目：

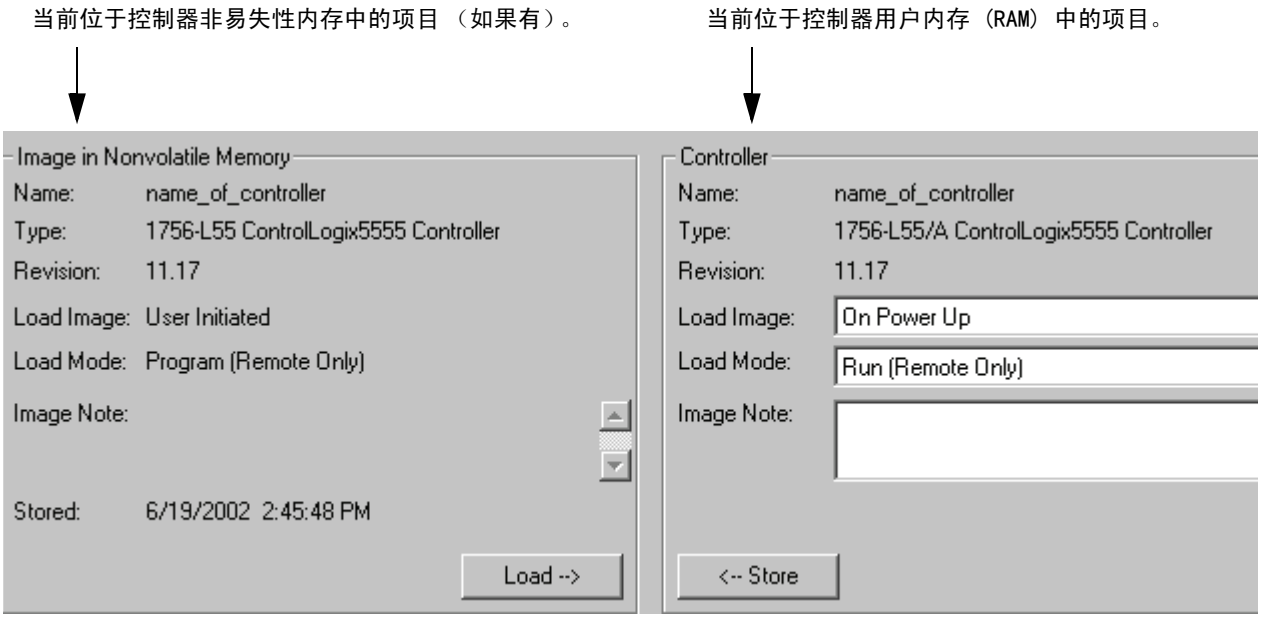
1. 与控制器联机。
2. 将控制器设为程序模式 （远程程序或程序）
3. 在 Online （联机）工具栏上单击控制器属性按钮。



4. 选择 Nonvolatile Memory （非易失性内存）选项卡并单击 Load/Store （加载 / 存储）。



5. 选择何时 （何种条件下）将项目加载回控制器的用户内存 (RAM) 中。



如果选择 On Power Up （通电时）或 On Corrupt Memory （破坏内存时），必须还选择在加载后希望控制器进入的模式：

- Remote Program （远程程序）
- Remote Run （远程运行）

6. 单击 <- Store （存储）。

对话框将提示您确认存储。

7. 若要存储项目，请选择 Yes （是）。

存储过程中，发生以下事件：

- 在控制器前部，OK LED 显示下面的序列：闪烁绿色 ⇒ 恒定红色 ⇒ 恒定绿色
- RSLogix 5000 软件转为脱机。
- 对话框将通知您正在进行存储。

8. 单击 OK （确定）。

存储完成后，保持脱机。

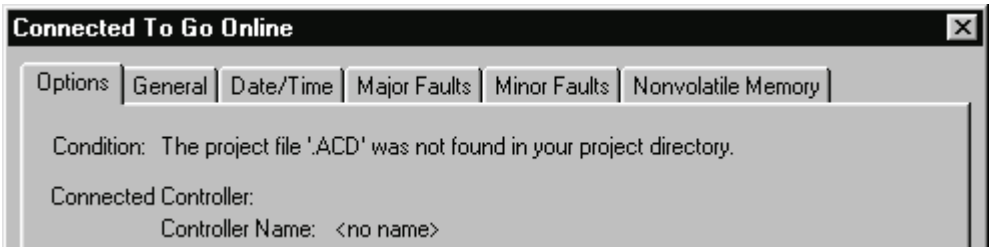
加载项目

使用 RSLogix 5000 软件从非易失性内存中加载项目：

注意

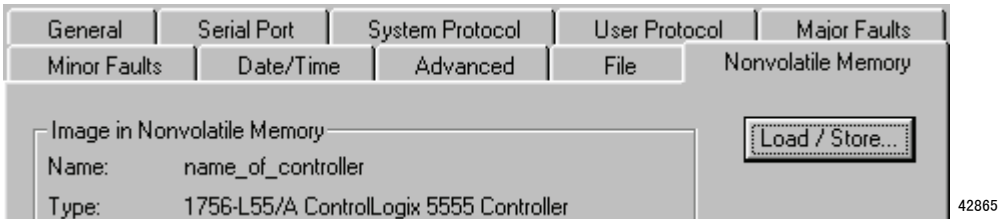
下载过程中，所有活动伺服轴关闭。 加载项目前，确保这*将不* 导致轴的意外移动。

1. 与控制器联机。
2. 下面的对话框打开？

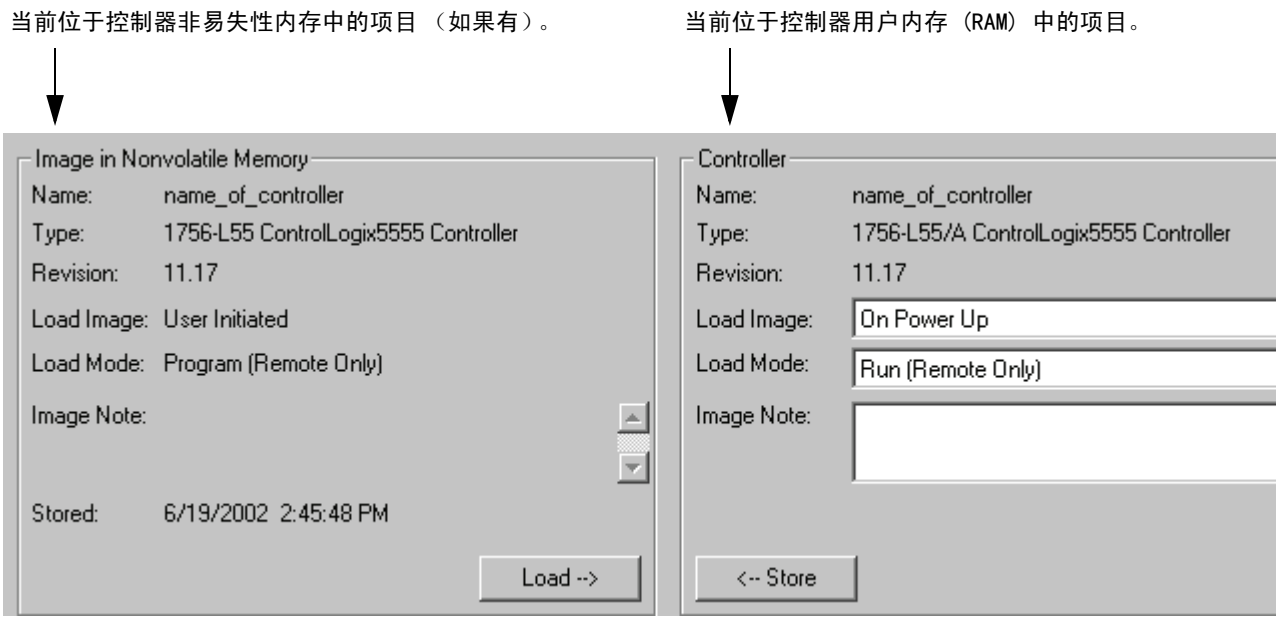


如果：	则：
否	a. 将控制器设为程序模式 （远程程序或程序）。 b. 在 Online （联机）工具栏上单击控制器属性按钮。
是	将控制器设为程序模式 （远程程序或程序）。使用： • Connected To Go Online （连接以联机）对话框的 General （一般）选项卡。 • 控制器前部的按键

3. 选择 Nonvolatile Memory （非易失性内存）选项卡并单击 Load/Store （加载 / 存储）。



4. 单击 Load -> （加载）。



对话框将提示您确认加载。

5. 若要从非易失性内存加载项目，请单击 Yes （是）。

加载过程中，发生以下事件：

- 在控制器前部，OK LED 显示下面的序列：

如果加载：	则 OK LED 显示：
不包括固件	恒定红色 > 恒定绿色
包括固件	闪烁红色 > 恒定红色 > 恒定绿色

- RSLogix 5000 软件转为脱机。

加载完成后，保持脱机。

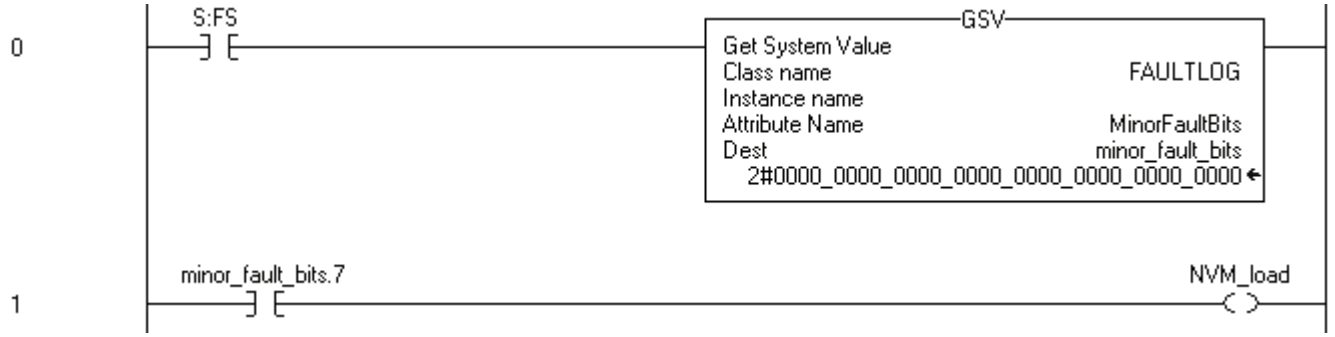
检查加载

控制器从非易失性内存加载项目时，提供以下信息：

- 记录次故障 （类型 7 代码 49）
- 设置 FAULTLOG 对象 MinorFaultBits 属性位 7

如果希望项目标记其从非易失性内存中加载，请使用下面的逻辑：

在项目的第一次扫描时 （*S:FS* 为 on），GSV 指令获得 FAULTLOG 对象 MinorFaultBits 属性，并存储 *minor_fault_bits* 中的值。如果位 7 为 on，则控制器从其非易失性内存加载项目。



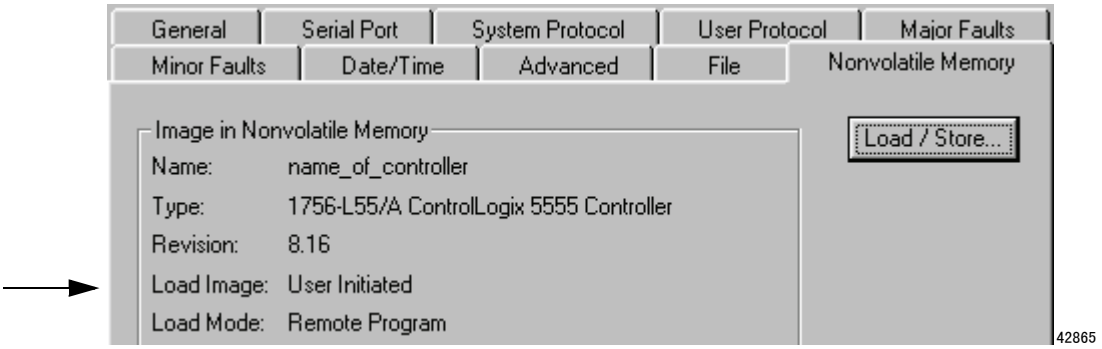
位置：	为：
minor_fault_bits	存储 FAULTLOG 对象 MinorFaultBits 属性的标记。数据类型为 DINT。
NVM_load	指示控制器从其非易失性内存加载项目的标记。

清除非易失性内存

从非易失性内存清除项目：

1. 与控制器联机。
2. 在 Online （联机）工具栏上单击控制器属性按钮。

3. 选择 Nonvolatile Memory （非易失性内存）选项卡。



4. Load Image （加载映像）选项是否设置为 User Initiated （用户启动）？

如果：	则：
否	转至第 17-12 页的 “更改 Load Image （加载映像）选项”。
是	转至第 17-13 页的 “从控制器清除项目”。

更改 Load Image （加载映像）选项

1. 单击 Load/Store （加载 / 存储）。
2. 在 Load Image （加载映像）下拉列表中选择 User Initiated （用户启动）。
3. 单击 <- Store （存储）。

   对话框将提示您确认存储。
4. 若要存储项目，请选择 Yes （是）。

   对话框将通知您正在进行存储。
5. 单击 OK （确定）。
6. 等待直到控制器前部的 OK LED 恒定绿色。这表示存储完成。

从控制器清除项目

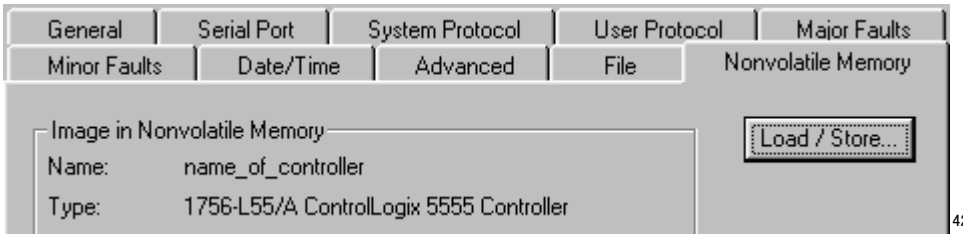
1. 从控制器断开电池。
2. 关闭机架电源再打开。
3. 重新将电池连接到控制器。

存储空映像

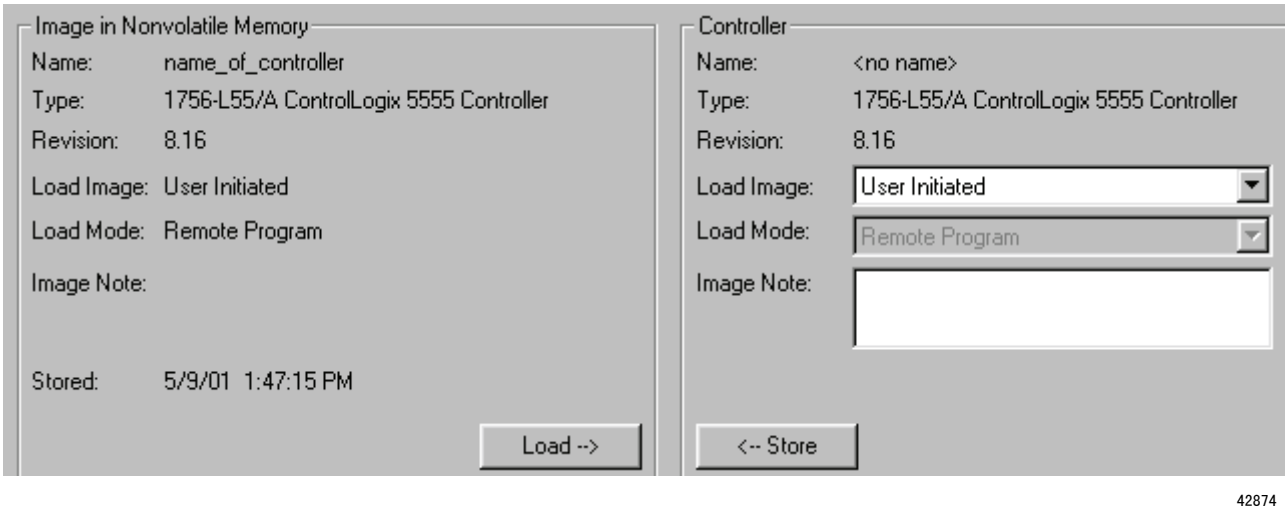
1. 与控制器联机。

   Connected To Go Online （连接以联机）对话框打开。

2. 单击 Nonvolatile Memory （非易失性内存）选项卡并单击 Load/Store （加载 / 存储）



3. 选择 <– *存储*。



对话框将提示您确认存储。

4. 要存储项目，请选择 *Yes（是）*。

   存储过程中，发生以下事件：

   - 在控制器前部，OK LED 显示下面的序列：
     闪烁绿色 > 红色 > 绿色
   - RSLogix 5000 软件转为脱机。
   - 对话框将通知您正在进行存储。

5. 选择 *OK（确定）*。

存储完成后，保持脱机。

使用 CompactFlash 读取器

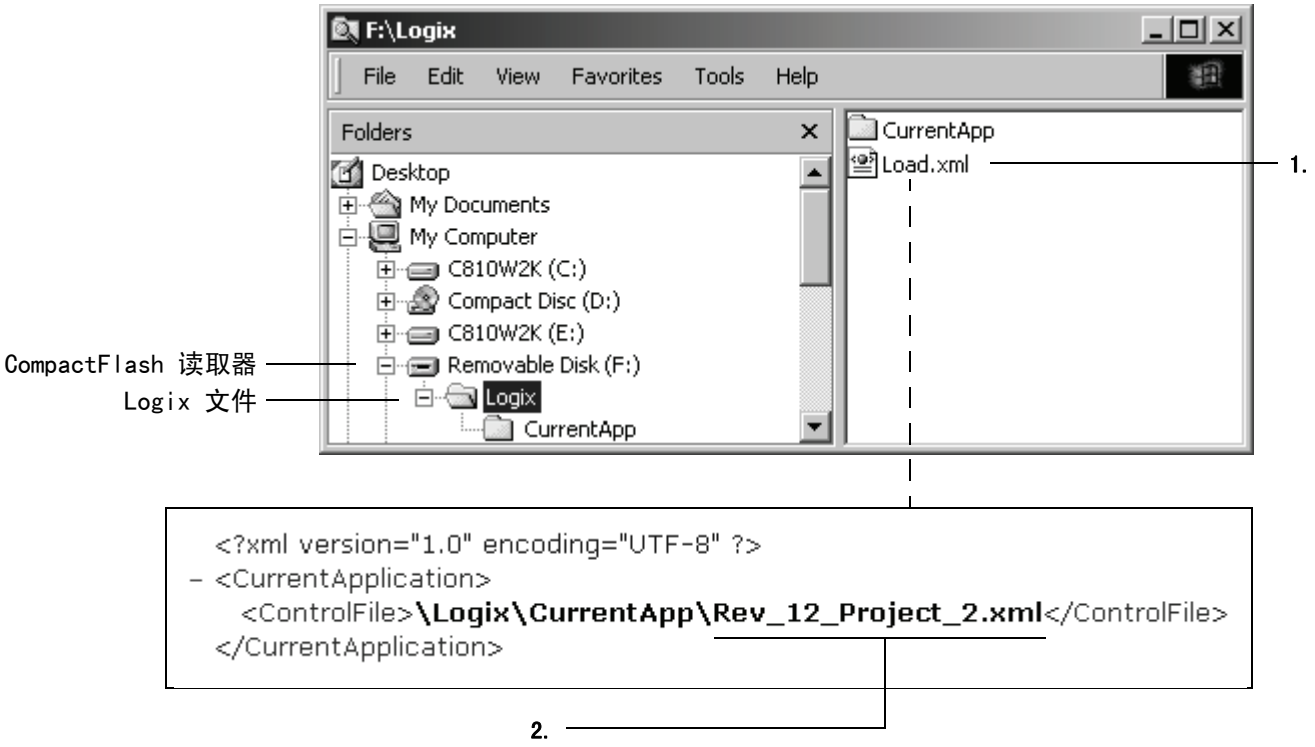
如果 CompactFlash 卡上项目的版本为 ≥ 12.0，则该卡使用 FAT16 文件系统格式化。

读取和写入 CompactFlash 卡的示例控制器项目随 RSLogix 5000 Enterprise 编程软件提供。从 RSlogix 5000 软件选择 Help（帮助）→ Vendor Sample Projects（供应商示例项目）显示可用示例项目的列表。

手动更改将从 CompactFlash 卡加载的项目

CompactFlash 卡存储多个项目。默认情况下，控制器根据项目的加载选项加载最近存储的项目。

要指定从 CompactFlash 卡加载不同项目，请编辑卡上的 Load.xml 文件。



1. 要选择从卡加载的项目，请打开 *Load.xml*。选择文本编辑器打开文件。
2. 编辑要加载的项目名称。
   - 使用 CurrentApp 文件夹中的 XML 文件的名称。
   - 在 CurrentApp 文件夹中，项目由 XML 文件和 P5K 文件组成。

手动更改项目的加载参数

向非易失性内存存储项目时，定义：

- 何时加载项目 （通电时，破坏内存时，用户启动）
- 设置控制器的模式 （如果按键在 REM 并且加载模式不是 User Initiated 用户启动）

要指定从 CompactFlash 卡加载不同项目，请编辑卡上的 Load.xml 文件。

1. 若要更改项目的加载参数，请打开和项目同名的 XML 文件。使用文本编辑器打开文件。

```
<?xml version="1.0" encoding="UTF-8" ?>
- <Controller>
  - <ExecutiveLoadOption>
      <ExecFile>\Logix\CurrentApp\Executive.bin</ExecFile>
    </ExecutiveLoadOption>
  - <ProgramLoadOption>
2.    <ProgramLoadMode>CORRUPT_RAM</ProgramLoadMode>
      <LoadFile>\Logix\CurrentApp\Rev_12_Project_2.p5k</LoadFile>
    </ProgramLoadOption>
  - <ControllerModeOption>
3.    <ControllerMode>RUN</ControllerMode>
    </ControllerModeOption>
  </Controller>
```

2. 编辑项目的 Load Image （加载映像）选项。

如果希望将 Load Image （加载映像）选项设置为：	则输入：
On Power Up （通电时）	ALWAYS
On Corrupt Memory （破坏内存时）	CORRUPT_RAM
User Initiated （用户启动）	USER_INITIATED

3. 编辑项目的 Load Mode （加载模式）选项 （如果加载映像选项不是*用户启动* 则不适用）。

如果希望将 Load Mode （加载模式）选项设置为：	则输入：
Program (Remote Only)（程序，仅远程）	PROGRAM
Run (Remote Only)（运行，仅远程）	RUN

CompactFlash 卡的其他用途

从版本 13 开始，除了存储控制器项目，还可以使用 CompactFlash 卡存储数据。例如：

- PanelView 终端更改控制器项目中的标记值。如果控制器断电（并且控制器没有备用电池），控制器中运行的程序以及 PanelView 终端更改的所有值都将丢失。使用 CompactFlash 文件系统和项目中的逻辑在标记值更改时存储它们。当项目从 CompactFlash 卡重新加载时，它可以检查 CompactFlash 卡中任何保存的标记值并重新加载到项目中。

- 在 CompactFlash 卡上存储配方集合。需要配方时，编程控制器从 CompactFlash 卡读取新配方的数据。

- 编程控制器以特定时间间隔写入数据日志。

还可以使用 CompactFlash 卡读取器读写 CompactFlash 卡。此方法以二进制写入标记值。可以使用任何文本编辑器读取数据，但数据显示为二进制数据的对等 ASCII。

有关更多信息，请参见 RSLogix 5000 Enterprise 编程软件提供的示例项目。从 RSlogix 5000 软件选择 Help （帮助） → Vendor Sample Projects （供应商示例项目）显示可用示例项目的列表。 还可以从 MySupport 知识库查看 G82433612 技术声明“在 Logix5000 控制器上使用 CompactFlash 文件系统”。从 www.ab.com 选择 Knowledgebase 链接以访问此数据库

注释：

确保项目安全

何时使用本章

以下选项可确保项目安全：

要：	请参阅页：
使用例程源程序保护	18-1
使用 RSI Security Server 保护项目	18-11

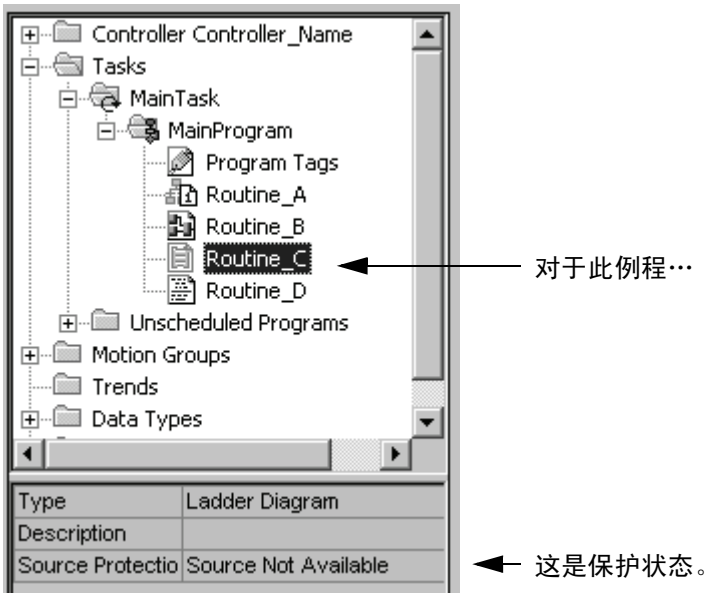
可以同时使用两个选项。

使用例程源程序保护

要限制访问例程的对象，请使用 RSLogix 5000 软件向例程分配**源密钥**（保护例程）。

- 要保护例程，必须首先启用 RSLogix 5000 软件的这项功能。
- 保护例程后，计算机需要源程序密钥才能编辑、复制或导出例程。
- 可以选择使例程在没有源程序密钥的情况下可查看还是不可查看。
- 无论源程序密钥是否可用，始终可以下载项目并执行所有例程。
- 可以通过以下方法之一获得访问：
  - 将源程序密钥的名称手动输入创建或已经存在的源文件中。
  - 将 RSLogix 5000 软件指向源程序密钥文件的位置。

控制器组织器显示例程的保护状态：



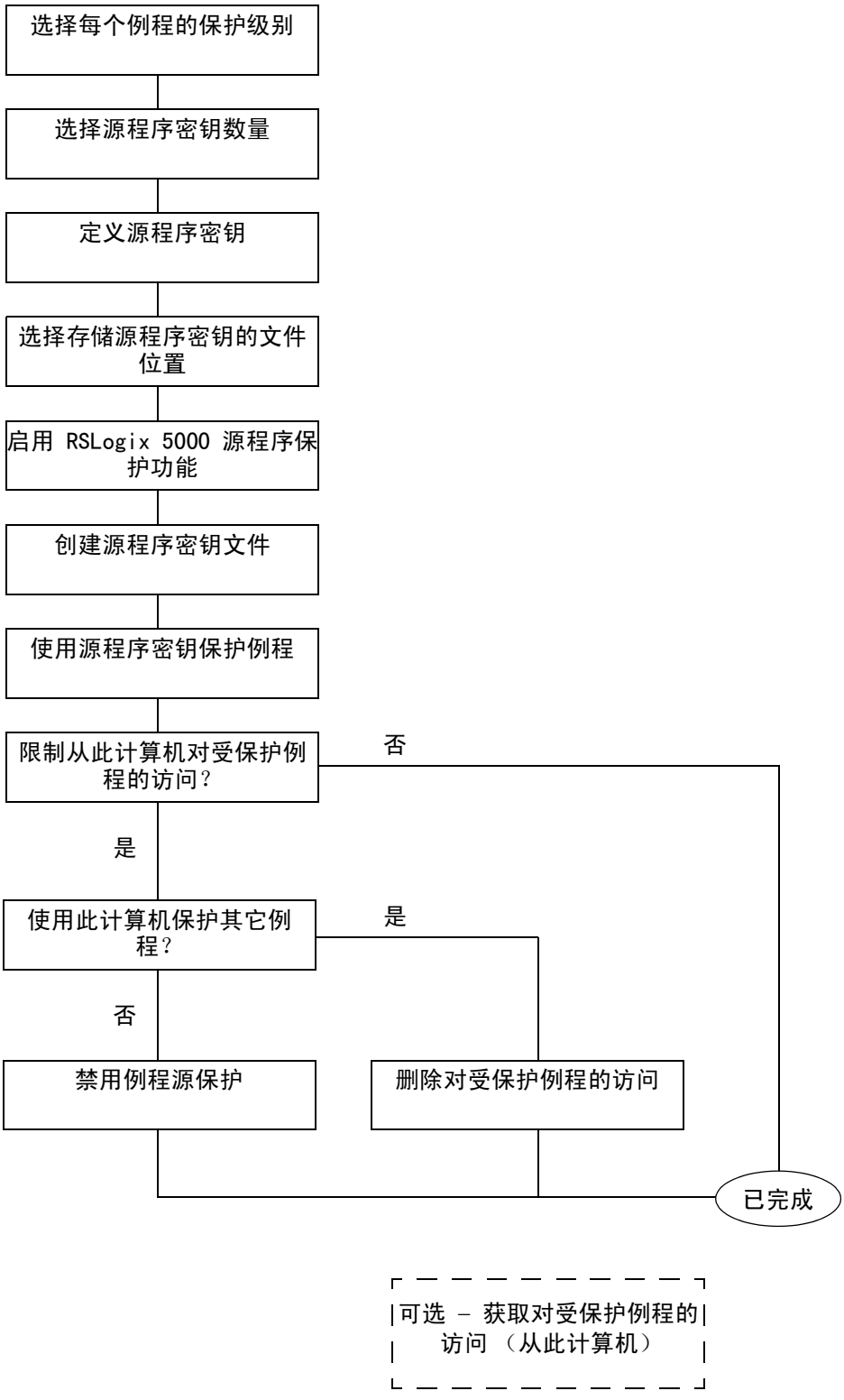
如果控制器组织器显示：	则：
Source Not Available（源程序不可用）	• 源程序程序密钥已分配给例程。 • 要打开例程，计算机需要例程的源程序密钥。
Source Not Available (Viewable)（源程序不可用，可以查看）	• 源程序程序密钥已分配给例程。 • 只能打开和查看例程。 • 不能更改或复制例程的任何内容。
Source Available（源程序可用）	• 源程序密钥已分配给例程。 • 具有例程的完全访问权。
Source Available (Viewable)（源程序可用，可以查看）	• 源程序密钥已分配给例程。 • 具有例程的完全访问权。 • 没有源程序密钥的人也可以查看例程。
以上都不是	• 没有为例程分配源程序密钥。 • 具有例程的完全访问权。

重要

如果例程的源程序不可用，请不要导出项目。

- 导出文件（.L5K）仅包含源代码在其中可用的例程。
- 如果导出源代码在其中不是对所有例程可用的项目，将不能还原整个项目。

要分配和管理源程序密钥，请：



选择每个例程的保护级别

源程序保护以例程级别保护项目。

表 18.1 例程保护选项

如果希望：	**并且：**	**则：**	
		保护例程？	**允许查看？**
防止其它人： • 编辑例程 • 更改例程的属性 • 导出例程	还防止其它人： • 打开 （显示）例程 • 搜索例程 • 在例程中交叉引用 • 打印例程	是	否
	没有其它限制	是	是
让所有人都具有对例程的完全访问权	→	否	

选择源程序密钥数量

要保护例程，请为例程指定一个**源密钥**。可以根据需要重新使用源程序密钥，如下所示。

此：	**给您：**
所有项目一个源程序密钥	最少数量的源程序密钥 （更容易管理但保护性更差）
每个项目一个唯一源程序密钥	
每个项目中每个例程一个唯一源程序密钥	最多数量的源程序密钥 （更难以管理但保护性更强）

在保护需求和想承担的源程序密钥管理水平间选择平衡的源程序密钥数量。

定义源程序密钥

源程序密钥和 RSLogix 5000 组件 （如例程、标记和模块）遵循相同的命名规则。 源程序密钥：

- 必须以字母字符 （A-Z 或 a-z）或下划线 （_） 开始
- 只能包含字母字符、数字字符和下划线
- 最多可有 40 个字符
- 不能连续出现下划线 （_） 或以下划线结尾
- 不区分大小写

选择存储源程序密钥的文件位置

源程序密钥文件 （sk.dat） 存储源程序密钥。 源程序密钥文件与 RSLogix 5000 项目文件 （.acd） 分开。可以将源程序密钥文件存储在任何所选的文件夹中。

启用 RSLogix 5000 源程序保护功能

要使用 RSLogix 5000 软件的例程源程序保护功能，请编写下面的注册表项以启用该功能：

键：	值条目：		
	名称：	类型：	数据：
HKEY_CURRENT_USER\Software\Rockwell Software\RSLogix 5000\ProtectedRoutine	PTCRoutine	DWORD	1

制作注册表项：

1. 获得 RSLogix 5000 软件 CD。
2. 从 CD 执行下面的文件：

语言 \Tools\Source Protection Tool\Enable Protected Routine Config.reg

其中：

语言 是软件的语言。例如，对于英文软件，打开 ENU 文件夹。

Enable Protected Routine Config.reg 文件制作所需的注册表项

创建源程序密钥文件

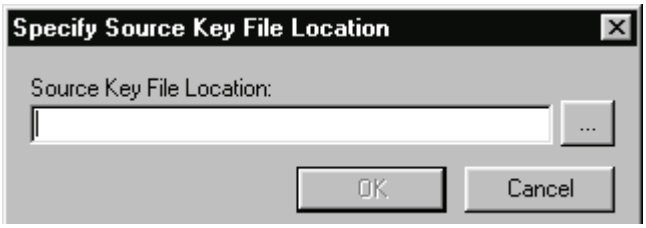
1. 打开要保护的 RSLogix 5000 项目。

2. 从 Tools （工具）菜单选择 Security （安全）> Configure Source Protection （配置源程序保护）。

3. RSLogix 5000 软件提示您指定源程序密钥文件的位置？

如果：	则：
否	计算机已经有源程序密钥文件。转至第 第 18-7 页 页的 “使用源程序密钥保护例程。
是	转至第 4 步。

4. 选择 Yes （是）。



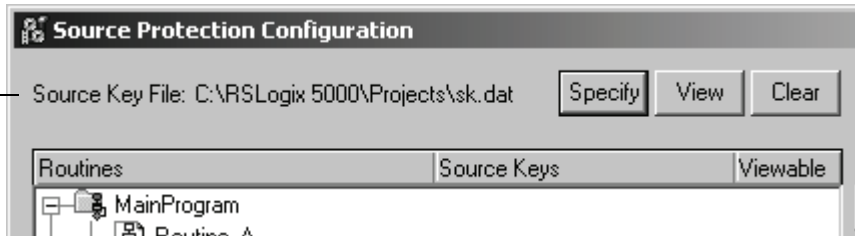
5. 单击 ... 并浏览至要存储文件的文件夹。

6. 单击 *OK （确定）*。

   对话框询问您是否要创建源程序密钥文件 （sk.dat）。

7. 选择 Yes （是）。

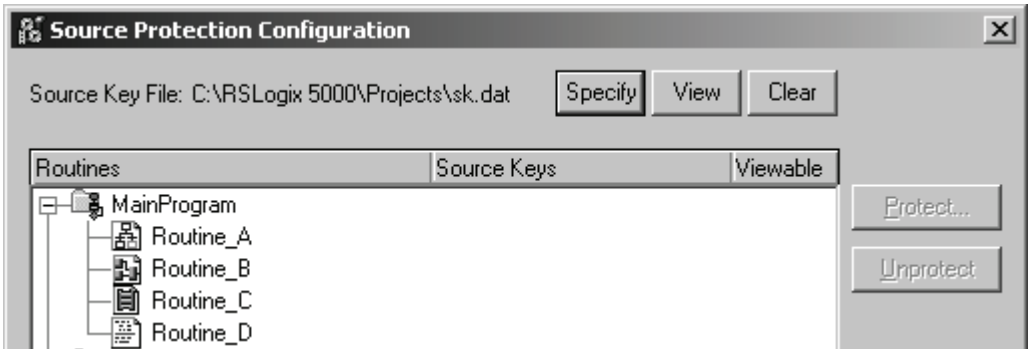
源程序密钥文件 (sk.dat) 的位置



使用源程序密钥保护例程

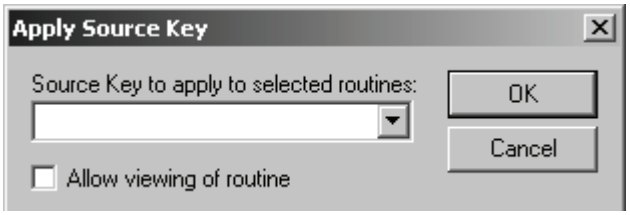
1. 打开要保护的 RSLogix 5000 项目。

2. 从 Tools （工具）菜单选择 Security （安全） >Configure Source Protection （配置源程序保护）。



3. 选择要保护的例程。

4. 单击 Protect （保护）。



- 键入要用作源程序密钥的**名称**。或者从下拉列表选择现有源程序密钥。
- 如果有人没有源程序密钥，是否希望允许他们打开和查看例程？

如果：	则：
否	清除 （取消选中）*Allow viewing of routine*（允许查看例程）复选框 （默认）。
是	选中 *Allow viewing of routine*（允许查看例程）复选框。

5. 单击 OK （确定）。

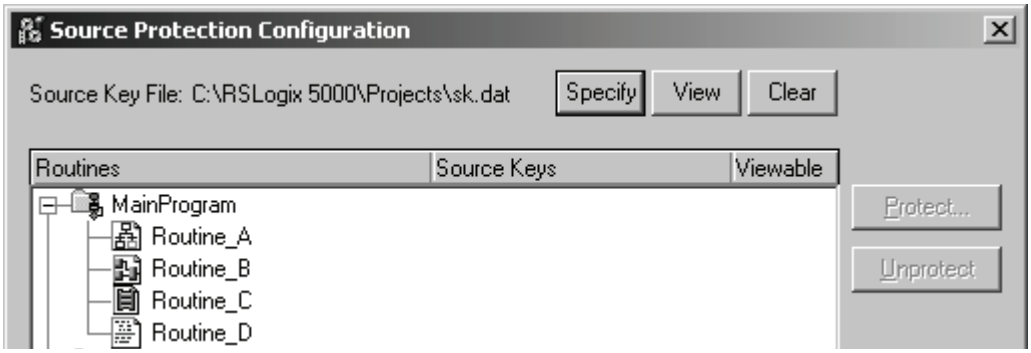
6. 向项目分配所需源程序密钥后，单击 Close （关闭）。

7. 从 File （文件）菜单选择 Save （保存）。

删除对受保护例程的访问

重要 从计算机删除源程序密钥文件（sk.dat）前，请记下源程序密钥或复制文件并存储在安全位置。

1. 打开受保护的 RSLogix 5000 项目。
2. 从 Tools （工具）菜单选择 Security （安全）>Configure Source Protection （配置源程序保护）。



3. 单击 Clear （清除）。

   对话框将询问您是否要删除源程序密钥文件（sk.dat）。

4. 是否希望从计算机删除源程序密钥文件 （防止以后继续访问文件）？

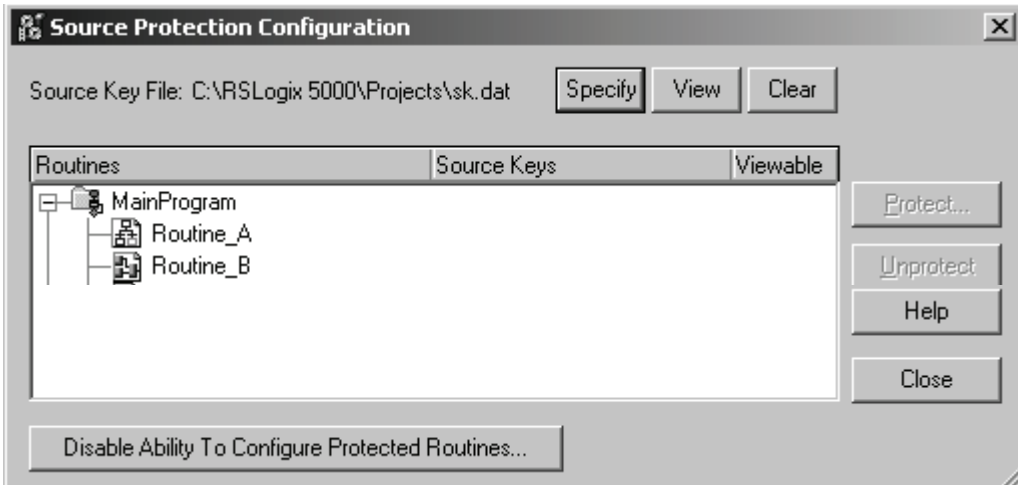
如果：	则：
是	选择 Yes （是）。
否	选择 No （否）。

禁用例程源保护

重要 从计算机删除源程序密钥文件（sk.dat）前，请记下源程序密钥或复制文件并存储在安全位置。

1. 打开受保护的 RSLogix 5000 项目。

2. 从 Tools （工具）菜单选择 Security （安全）>Configure Source Protection （配置源程序保护）。



3. 单击 Disable Ability To Configure Protected Routines （禁用配置受保护例程的功能）。

   对话框将提示您确认此操作。

4. 选择 *Yes （是）*。

   对话框将询问您是否要删除源程序密钥文件（sk.dat）。

5. 是否希望从计算机删除源程序密钥文件 （防止以后继续访问文件）？

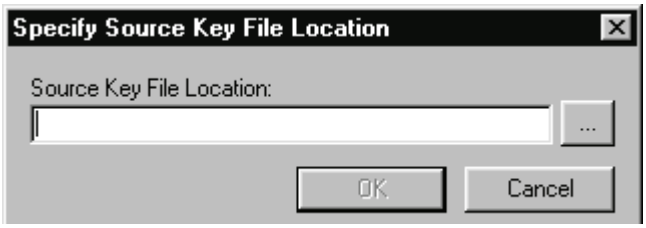
如果：	则：
是	选择 Yes （是）。
否	选择 No （否）。

获取对受保护例程的访问

1. 打开包含受保护例程的 RSLogix 5000 项目。
2. 从 Tools （工具）菜单选择 Security （安全） >Configure Source Protection （配置源程序保护）。
3. RSLogix 5000 软件提示您指定源程序密钥文件的位置？

如果：	则：
否	转至第 7 步。
是	转至第 4 步。

4. 选择 Yes （是）。

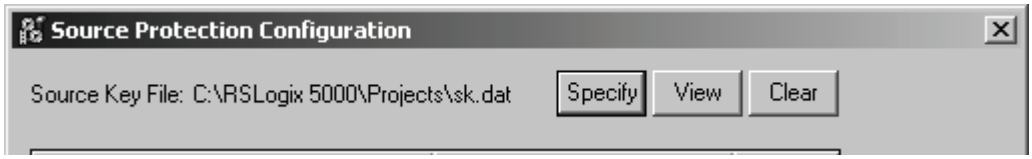


5. 单击 ...

6. 计算机是否已有源程序密钥文件 (sk.dat)？

如果：	则：
是	A. 选择包含文件的文件夹并单击 OK （确定）。 B. 单击 *OK*（确定）。
否	A. 选择存储新文件的文件夹并选择 OK （确定）。 对话框询问您是否要创建源程序密钥文件 (sk.dat)。 B. 选择 Yes （是）。

7. 单击 View （查看）。



- 如果提示您选择用于打开文件的程序，请选择一个字处理程序，例如记事本。
- sk.dat 文件打开。

8. 键入源程序密钥的名称。要输入多个密钥，请将每个密钥键入在不同的行。

sk.dat - Notepad
key1
key2
key3

9. 保存并关闭 sk.dat 文件。

使用 RSI Security Server 保护项目

RSI Security Server 软件使您可以控制用户对 RSLogix 5000 项目的访问权。使用此软件，可以根据下面的内容自定义对项目的访问权：

- 当前登录到工作站的用户
- 用户正在访问的 RSLogix 5000 项目
- 用户从中访问 RSLogix 5000 项目的工作站

为 RSLogix 5000 项目使用 Security Server 软件前，请设置软件：

- 安装 RSI Security Server 软件
- 设置 DCOM
- 启用 Security Server for RSLogix 5000 软件
- 导入 RSLogix5000Security.bak 文件
- 为用户定义全局操作
- 为用户定义项目操作
- 添加用户
- 添加用户组
- 向 RSLogix 5000 软件分配全局访问权
- 向新 RSLogix 5000 项目分配项目操作

Security Server 软件为 RSLogix 5000 项目设置后，完成以下操作以保护项目：

- 确保 RSLogix 5000 项目安全
- 向 RSLogix 5000 项目分配访问权
- 刷新 RSLogix 5000 软件，如果需要

安装 RSI Security Server 软件

重要 如果安装 Security Server 软件时 RSLogix 5000 软件已经位于计算机上，请在提示时启用 RSLogix 5000 软件的安全。

请参阅随 RSI Security Server 软件提供的 *Getting Results with Rockwell Software's Security Server (Standalone Edition)*（使用 Rockwell Software 的 Security Server 标准版 ）。

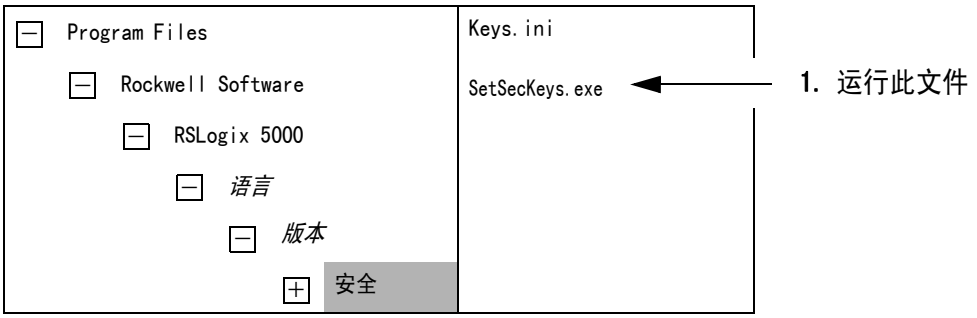
设置 DCOM

请参阅随 RSI Security Server 软件提供的 *Getting Results with Rockwell Software's Security Server (Standalone Edition)*（使用 Rockwell Software 的 Security Server 标准版 ）。

启用 Security Server for RSLogix 5000 软件

安装 RSLogix 5000 软件前是否安装了 Security Server？

如果：	则：
是	转至第 1 步。
否	转至第 18-13 页的 “导入 RSLogix5000Security.bak 文件。



其中：	表示：
语言	软件的语言。例如，对于英文软件，打开 ENU 文件夹。
版本	软件的版本，例如 v10

Locate Project File（定位项目文件）对话框打开。Keys.ini 文件默认应选中。

2. 选择 Open （打开）。

3. 选中 RSLogix 5000 复选框并单击 OK （确定）。

43073

重要 打开安全功能后，将永久保持打开。

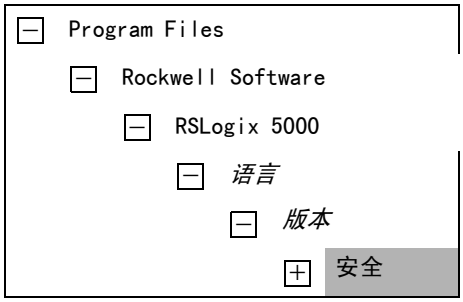
导入 RSLogix5000Security.bak 文件

RSLogix5000Security.bak 文件提供 Security Server 与 RSLogix 5000 软件工作时所需的配置。

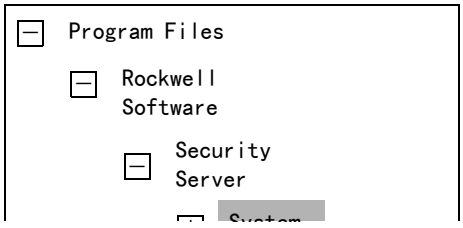
1. 启动 Security Configuration （安全性配置）浏览器。

2. 从 File （文件）菜单选择 Import Database （导入数据库）。

3. 正在使用的 Security Server 软件版本：

如果：	则：
2.00	查看此文件夹： 

其中：	表示：
语言	软件的语言。 例如，对于英文软件，打开 ENU 文件夹。
版本	软件的版本，例如 v10

2.01	查看此文件夹： 

4. 选择 RSLogix5000Security.bak 文件并单击 Open （打开）。

为用户定义全局操作

全局操作是没有绑定到具体项目的任务，例如创建新项目或更新控制器固件。这些全局操作适用于 RSLogix 5000 软件。

表 18.2 全局操作

要使用户：	则授予以下操作的访问权：
确保任何未受保护的控制器的安全	确保控制器安全
创建新 RSLogix 5000 项目	新建项目
在 RSLogix 5000 软件中打开 .L5K 文件，该文件创建项目	
将 PLC 或 SLC 项目转换为 .L5K 文件	
使用 RSLogix 5000 软件启动 ControlFLASH 软件并更新控制器固件	更新固件
更改工作站设置	工作站：修改选项
更改打印设置	打印：修改选项
更改工具栏	工具栏：配置

为用户定义项目操作

项目操作使您对特定项目或项目组执行特定任务。

- 为 RSLogix 5000 项目启用安全功能或在安全功能打开情况下创建新项目时，项目成为 New RSLogix 5000 Resources 组的成员。
  - 使用该组中项目的用户需要相应访问权。
  - 建议对任何具有创建项目权限的人授予完全访问权限。
- 要自定义项目的访问权，请将其移出 New RSLogix 5000 Resources 组并授予特定于该项目的权限。

Security Server 支持四种特定于 RSLogix 5000 软件的内置操作组。这些操作组将相关操作组合在组中以更容易选择。这些操作组是 View Project、Go Online、Maintain Project 和 Full Access。向操作组授予访问权将向该组中的相关操作授予访问权。每个组还在前一组基础上构建。例如，向 Maintain Project 授予访问权也将向 Go Online 和 View Project 授予访问权。“Go Online”（联机）和“View Project”（查看项目）操作。

版本 15 的 RSLogix 5000 软件添加了向操作组内各个操作授予访问权的功能。

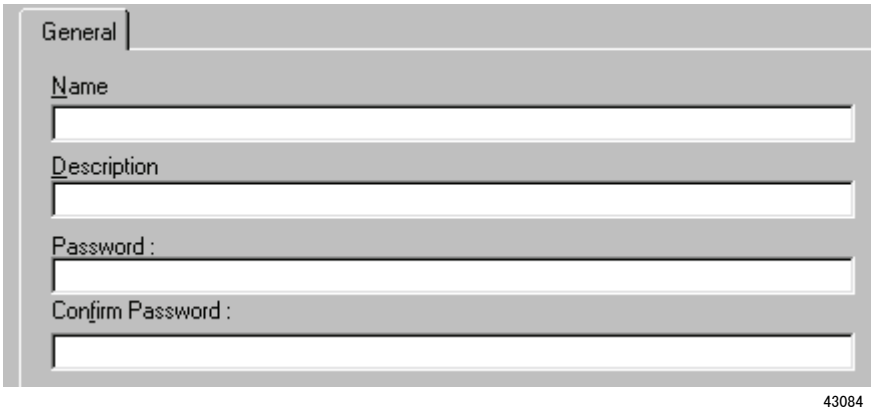
这些操作分组及各个操作适用于 RSLogix 5000 软件：

要使用户：	**授予对此项目的访问权（将授予对所有相关操作的访问权）：**	**或授予对与项目相关的各个操作的访问权：**
打开项目的（只读）版本。 用户必须具有打开和查看项目的权限才能对项目进行其它操作。	查看项目	项目：打开
以 View Project 权限联机	联机	项目： 联机
访问和修改项目中的控制器信息	维护项目	控制器：清除故障 控制器： 修改模式 控制器： 修改版本 控制器： 修改类型
访问和修改项目中的模块信息	维护项目	模块：修改属性
打印报告	维护项目	打印： 报告
访问和修改项目	维护项目	项目：压缩 项目： 下载 项目： 导出 项目： 联机 项目： 修改路径 项目： 修改路径 项目： 保存 项目： 另存为 项目： 上载
访问和修改标记	维护项目	标记：强制 标记： 修改值
访问和修改趋势	维护项目	趋势：创建 趋势：删除 趋势： 修改属性 趋势： 运行
更新控制器固件	维护项目	更新固件
提供控制器的完全访问权	完全访问	控制器： 锁定 / 取消锁定 控制器： 修改属性
提供对控制器组织器中模块的完全访问权	完全访问	模块： 创建 模块： 删除 模块： 维护高 模块： 维护低

要使用户：	授予对此项目的访问权 （将授予对所有相关操作的访问权）：	或授予对与项目相关的各个操作的访问权：
提供对运动配置和命令的完全访问权	完全访问	运动：命令轴 运动： 修改配置
提供对控制器非易失性内存的完全访问权	完全访问	非易失性内存：加载 非易失性内存： 存储
提供对设备相位的完全访问权	完全访问	相位：创建 相位： 删除 相位： 手动控制 相位： 修改属性
提供映射 PLC/SLC 标记的完全访问权	完全访问	PLC/SLC：修改标记映射
提供对程序的完全访问权	完全访问	程序：创建 程序： 删除 程序： 修改属性
提供对例程的完全访问权	完全访问	例程：创建 例程： 删除 例程： 手动控制 例程： 修改逻辑 例程： 修改属性
提供对标记的完全访问权	完全访问	标记： 创建 标记： 删除 标记： 修改属性
提供对任务的完全访问权	完全访问	任务：创建 任务： 删除 任务： 修改属性
提供对用户定义的标记的完全访问权	完全访问	用户定义的类型：创建 用户定义的类型： 删除 用户定义的类型： 修改
取消保护受保护的控制器	不可用	取消保护控制器

添加用户

1. 右击并选择 New （新建）。

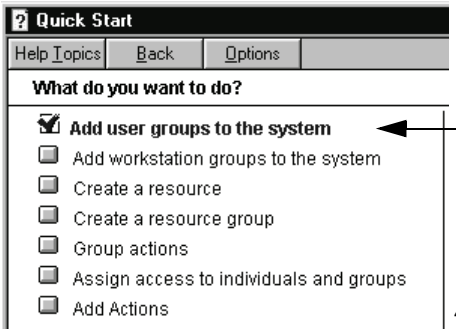


2. 键入用户信息并单击 OK （确定）。

添加用户组

组使您可以管理多个需要类似权限的用户。

1. 从 Help （帮助）菜单选择 Quick Start （快速开始）。



2. 遵循此任务的步骤。

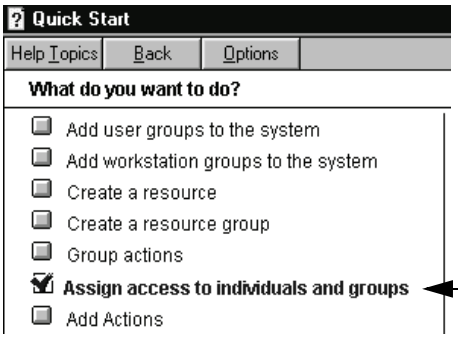
向 RSLogix 5000 软件分配全局访问权

允许用户执行全局操作：

1. 在 Configuration （配置）浏览器中选择 RSLOGIX 5000 组。

43077

2. 从 Help （帮助）菜单选择 Quick Start （快速开始）。



43076

3. 遵循此任务的步骤。

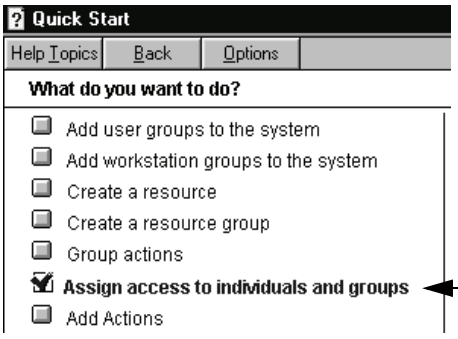
向新 RSLogix 5000 项目分配项目操作

允许用户对 New RSLogix 5000 Resources 组中的项目执行操作：

1. 在 Configuration （配置）浏览器中选择 New RSLogix 5000 Resources 组。

43075

2. 从 Help （帮助）菜单选择 Quick Start （快速开始）。



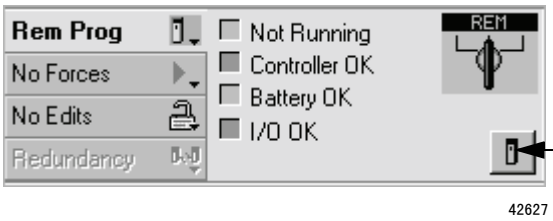
43076

3. 遵循此任务的步骤。

确保 RSLogix 5000 项目安全

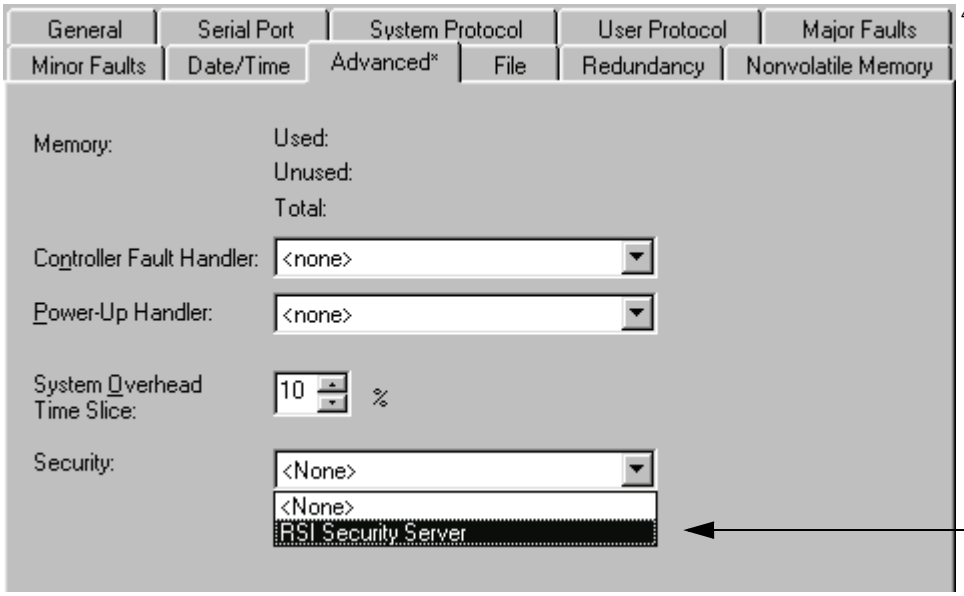
对于新项目，安全选项在创建项目时可用。要使 Security Server 软件保护现有项目，请为该项目启用安全功能。

1. 打开 RSLogix 5000 项目。



2. 单击控制器属性按钮。

3. 选择 Advanced （高级）选项卡。



4. 选择 RSI Security Server。

5. 依次单击 OK （确定）和 Yes （是）。

在 Security Server 软件中，项目作为 New RSLogix 5000 Resources 组的成员出现。如果 Security Server 软件已打开，则从它的 View （查看）菜单选择 Refresh （刷新）。

向 RSLogix 5000 项目分配访问权

当项目在 New RSLogix 5000 Resources 组中时，该组的访问控制列表决定用户可以在项目上执行的操作。要自定义项目的访问权，请将其移出该组并分配特定操作：

1. 在 Configuration （配置）浏览器中选择 New RSLogix 5000 Resources 组。

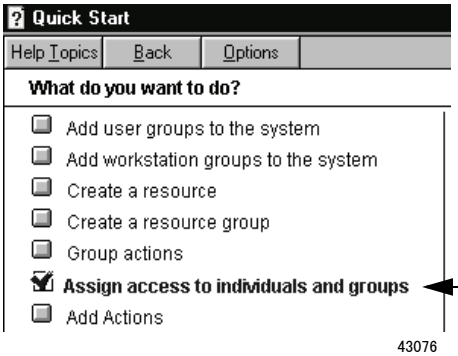
2. 选择 Group Members （组成员）选项卡。

3. 在 Member items （成员项）列表中选择项目并单击 << 按钮。

4. 单击 Apply （应用）。

5. 在 Configuration （配置）浏览器中选择该项目。

6. 从 Help （帮助）菜单选择 Quick Start （快速开始）。



7. 遵循此任务的步骤。

刷新 RSLogix 5000 软件，如果需要

如果 RSLogix 5000 项目已打开并且 RSI Security Server 软件中进行了影响项目的更改，请刷新 RSLogix 5000 软件：

从 Tools （工具）菜单选择 Security （安全） > Refresh Privileges （刷新权限）。

确定控制器内存信息

何时使用本章

使用本章获得 Logix5000 控制器内存的信息。

确定要获得的内存信息

根据您的控制器类型，控制器内存可分为以下几个区域：

如果您有此类型控制器：	则它存储：	在此内存中：
ControlLogix	I/O 标记	I/O 内存
	生成标记	
	使用标记	
	通过消息 (MSG) 指令进行的通信	
	与工作站的通信	
	与使用 RSLinx 软件的轮询 (OPC/DDE) 标记进行的通信 (1)	
	I/O 标记、生成标记或使用标记以外的标记	数据和逻辑内存 (2)
	逻辑例程	
	与使用 RSLinx 软件的轮询 (OPC/DDE) 标记进行的通信 (1)	
• CompactLogix • FlexLogix • DriveLogix • SoftLogix5800	这些控制器不划分它们的内存。它们在一个公共内存区域存储所有元素。	

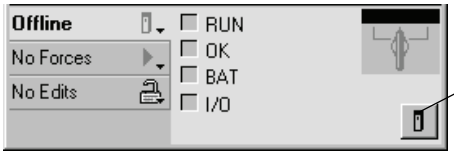
(1) 若要与轮询标记通信，控制器既使用 I/O 和数据也使用逻辑内存。

(2) 1756-L55M16 控制器具有用于逻辑的额外内存区。

脱机估计内存信息

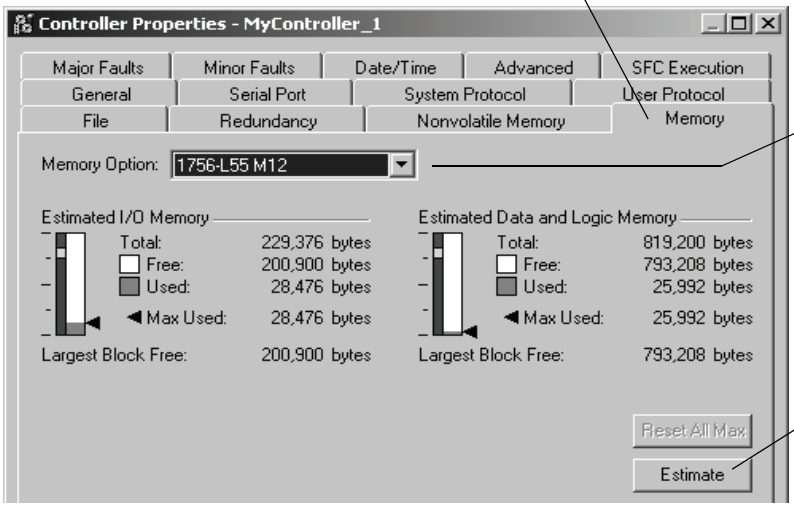
若要估计项目所需的控制器内存量，请使用控制器属性对话框的 Memory （内存）选项卡。对于控制器的每个内存区域，它允许您估计以下内存的字节数：

- 可用 （未使用）内存
- 使用的内存
- 最大可用连续内存块



1. 单击控制器属性按钮。

2. 选择 Memory （内存）选项卡。



3. 对于具有不同内存选项的控制器，请选择内存大小 （例如 M12）。

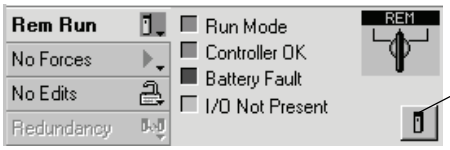
4. 查看自上次估计后的内存信息。

5. 估计控制器内存量。

查看运行时内存信息

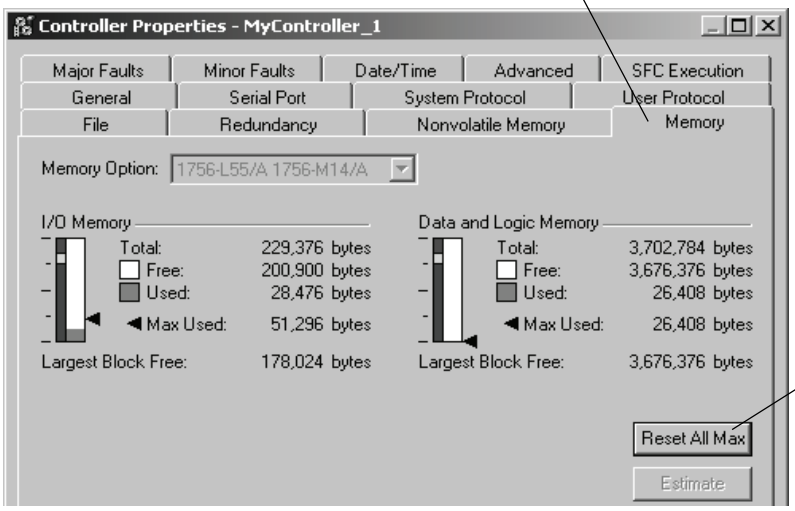
与控制器联机时，Memory （内存）选项卡显示控制器的实际内存使用。控制器运行时，使用额外内存用于通信。所需量取决于通信状态。

控制器的 Memory （内存）选项卡为每种内存都提供一个 Max Used （最大使用量）条目。Max Used （最已使用量）值显示通信发生时内存使用的峰值。



1. 单击控制器属性按钮。

2. 选择 Memory （内存）选项卡。



3. 查看内存信息。

4. 若要重置 *Max Used （最大使用量）*值，请单击此处。

写入逻辑以获得内存信息

使用逻辑获得控制器的内存信息：

- 从控制器获得内存信息
- 选择需要的内存信息
- 将 INT 转换为 DINT

从控制器获得内存信息

若要从控制器获得内存信息，请执行如下配置的消息 (MSG) 指令：

在此选项卡上：	对于此项：	键入或选择：	表示：
配置	消息类型	CIP Generic	执行控制与信息协议命令。
	服务类型	Custom	创建下拉列表中没有的 CIP Generic 消息。
	服务代码	3	读取控制器的特定信息 （GetAttributeList 服务）。
	类	72	从用户内存对象获得信息。
	实例	1	此对象仅包含 1 个实例。
	属性	0	空值
	源元素	*source_array*，类型 SINT[12]	
		在此元素中： **输入：**	**表示：**
		source_array[0] 5	获得 5 个属性
		source_array[1] 0	空值
		source_array[2] 1	获得可用内存
		source_array[3] 0	空值
		source_array[4] 2	获得总内存
		source_array[5] 0	空值
		source_array[6] 5	获得额外可用逻辑内存的最大连续块
		source_array[7] 0	空值
		source_array[8] 6	获得可用 I/O 内存的最大连续块
		source_array[9] 0	空值
		source_array[10] 7	获得可用数据与逻辑内存的最大连续块
		source_array[11] 0	空值
	源长度	12	写入 12 个字节 (12 SINT)。
	目标	*INT_array*，类型 INT[29]	
通信	路径	1, *slot_number_of_controller*	

选择需要的内存信息

MSG 指令将以下信息返回到 INT_array （MSG 的目标标记）：

重要

- 控制器返回 32 位字表示的值。若要按字节查看值，请将其乘以 4。
- 如果控制器不划分其内存，则值显示为 I/O 内存。
- 对于 1756-L55M16 控制器，MSG 指令为每个逻辑内存类别返回两个值。若要确定 1756-L55M16 控制器的可用或总逻辑内存，请为类别添加这两个值。

如果需要：	则复制以下数组元素：	说明：
可用 I/O 内存量 （32 位字）	*INT_array*[3]	32 位值的低 16 位
	INT_array[4]	32 位值的高 16 位
可用数据与逻辑内存量 （32 位字）	*INT_array*[5]	32 位值的低 16 位
	INT_array[6]	32 位值的高 16 位
仅 1756-L55M16 控制器讯钊饪捎寐呒 — 额外可用逻辑内存量 （32 位字）	*INT_array*[7]	32 位值的低 16 位
	INT_array[8]	32 位值的高 16 位
总 I/O 内存量 （32 位字）	*INT_array*[11]	32 位值的低 16 位
	INT_array[12]	32 位值的高 16 位
总数据与逻辑内存大小 （32 位字）	*INT_array*[13]	32 位值的低 16 位
	INT_array[14]	32 位值的高 16 位
仅 1756-L55M16 控制器讯钊饴呒 — 额外逻辑内存 （32 位字）	*INT_array*[15]	32 位值的低 16 位
	INT_array[16]	32 位值的高 16 位
仅 1756-L55M16 控制器讯钊饪捎寐呒 — 额外可用逻辑内存最大连续块 （32 位字）	*INT_array*[19]	32 位值的低 16 位
	INT_array[20]	32 位值的高 16 位
可用 I/O 内存最大连续块 （32 位字）	*INT_array*[23]	32 位值的低 16 位
	INT_array[24]	32 位值的高 16 位
可用数据与逻辑内存最大连续块 （32 位字）	*INT_array*[27]	32 位值的低 16 位
	INT_array[28]	32 位值的高 16 位

将 INT 转换为 DINT

MSG 指令将每个内存值返回为两个分离的 INT。

- 第一个 INT 表示值的低 16 位。
- 第二个 INT 表示值的高 16 位。

若要将分离的 INT 转换为一个可用的值，请使用复制 (COP) 指令，其中：

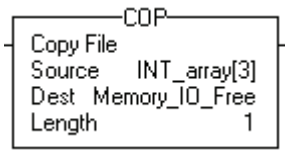
在此操作数中：	指定：	表示：
源	2 元素对的第一个 INT （低 16 位）	以低 16 位开始
目标	存储 32 位值的 DINT 标记	将值复制到 DINT 标记。
长度	1	复制一次 Destination （目标）数据类型中的字节数。在此情况下，指令复制 4 个字节 （32 位），将低 16 位和高 16 位组合到一个 32 位值中。

在下面的示例中，COP 指令以 32 位字产生表示可用 I/O 内存量的 32 位值。

示例

将 INT 转换为 DINT

- INT_array 的元素 3 是可用 I/O 内存量的低 16 位。元素 4 是高 16 位。
- Memory_IO_Free 是存储可用 I/O 内存量值的 DINT 标记 （32 位）。
- 若要复制全部 32 位，请指定 Length （长度）为 1。这将使指令复制一次 Destination （目标）的大小 （32 位）。此操作复制元素 3 （16 位）和元素 4 （16 位）并将 32 位结果放置在 Memory_IO_Free 中。



管理多个消息

何时使用此附录

此附录描述如何使用梯形逻辑程序顺序发送消息组 (MSG) 指令。如果需要控制大量 MSG 的执行，请使用此附录。

- 要继续，每个 MSG 指令都必须输入消息队列。
- 每个队列持有 16 个 MSG。
- 如果一次允许发送的 MSG 超过 16 个，则队列中没有空间发送下一个 MSG。
- 如果出现这种情况，MSG 必须等到队列中存在空间，此时控制器才能发送 MSG。每次对 MSG 进行后续扫描时，都要检查队列是否存在空间。

此附录中的消息管理器逻辑程序控制一次允许发送的 MSG 数目，并按顺序发送 MSG。通过这种方式，MSG 按顺序输入并退出队列，无需等到队列中的可用空间。

如何使用此附录

在此附录中，消息管理器逻辑程序发送三组 MSG。

- 为了使得示例易于进行，每组仅包含 2 个 MSG。
- 在您的项目中，每组使用多个 MSG，如 5 个。
- 按需要使用多个组来包含所有 MSG。

Msg_Group 标记控制 MSG 启用。

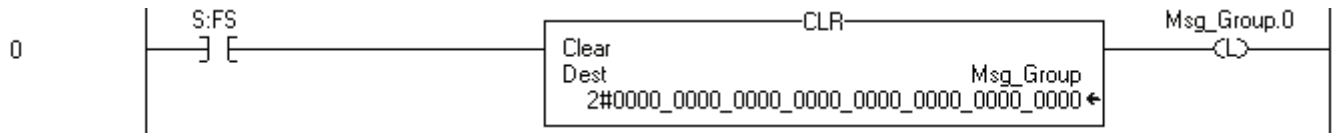
- 标记使用 DINT 数据类型。
- 标记的每位与一组 MSG 对应。
- 例如，Msg_Group.0 启用和禁用第一组 MSG （组 0）。

消息管理器逻辑程序 初始化逻辑程序

如果 S:FS = 1 （第一次扫描），然后初始化 MSG。

Msg_Group = 0，禁用所有 MSG。

Msg_Group.0 =1，启用第一组 MSG。



如有必要，重新开始序列

如果当前启用第 2 组 （最后一组）中的 MSG (Msg_Group.2 = 1)

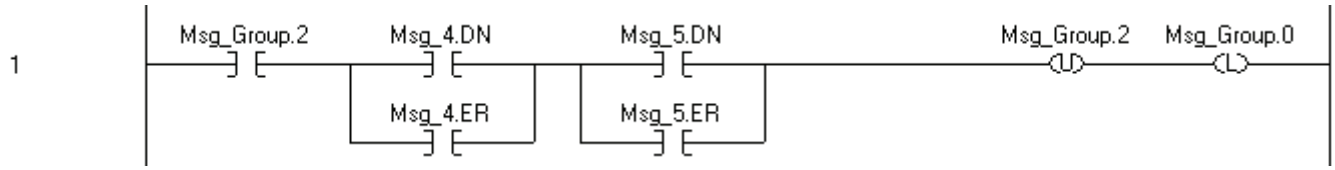
Msg_4 已完成或出错

Msg_5 已完成或出错

则重新开始第一组中的 MSG 序列：

Msg_Group.2 = 0。禁用最后一组 MSG。

Msg_Group.0 = 1。启用第一组 MSG。



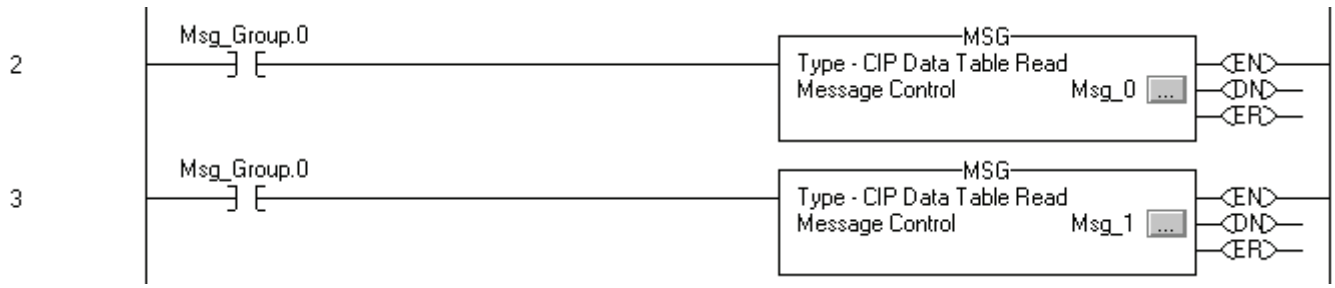
发送第一组 MSG

如果 Msg_Group.0 从 0 变为 1，则

发送 Msg_0。

发送 Msg_1。

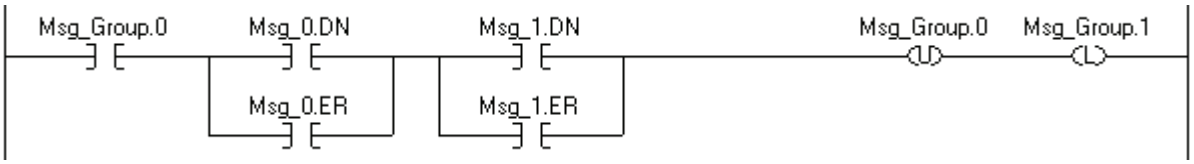
由于 MSG 指令是转换指令，所以此指令仅在运行条件从假变为真时执行。



启用下一组 MSG

如果当前启用第 0 组中的 MSG (Msg_Group.0 = 1)
Msg_0 已完成或出错
Msg_1 已完成或出错
则
Msg_Group.0 = 0。禁用当前 MSG 组。
Msg_Group.1 = 1。启用下一组 MSG。

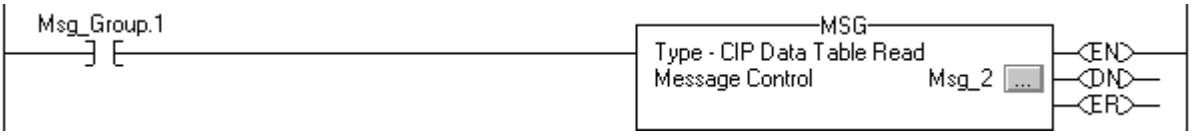
4



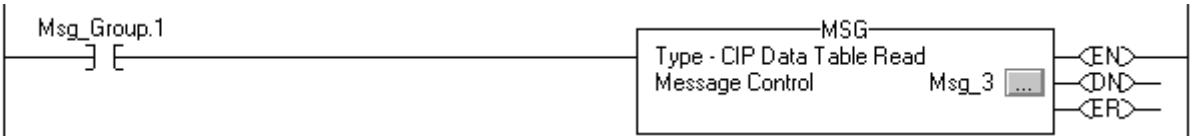
发送下一组 MSG

如果 Msg_Group.1 从 0 变为 1，则
发送 Msg_2。
发送 Msg_3。

5

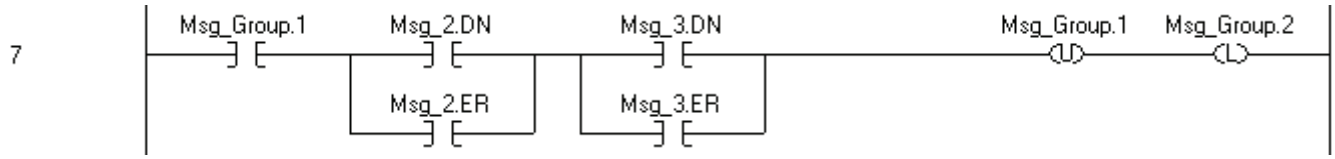


6



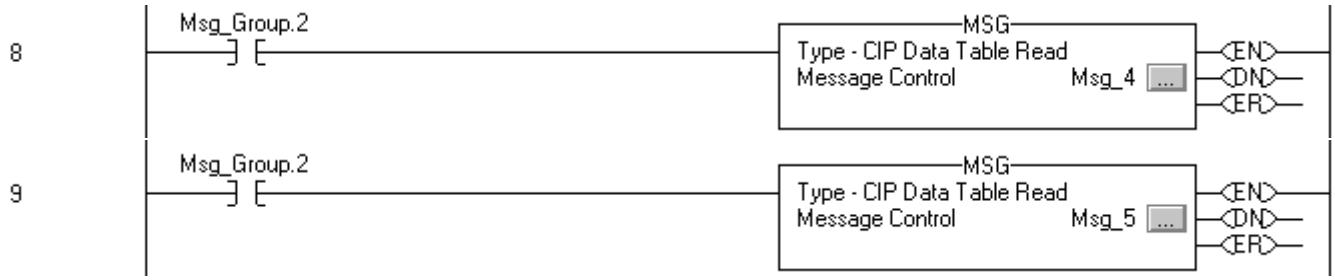
启用下一组 MSG

如果当前启用第 1 组 MSG (Msg_Group.1 = 1)
Msg_2 已完成或出错
Msg_3 已完成或出错
则
Msg_Group.1 = 0。禁用当前 MSG 组。
Msg_Group.2 = 1。启用下一组 MSG。



发送下一组 MSG

如果 Msg_Group.1 从 0 变为 1，则
发送 Msg_2。
发送 Msg_3。



向多个控制器发送消息

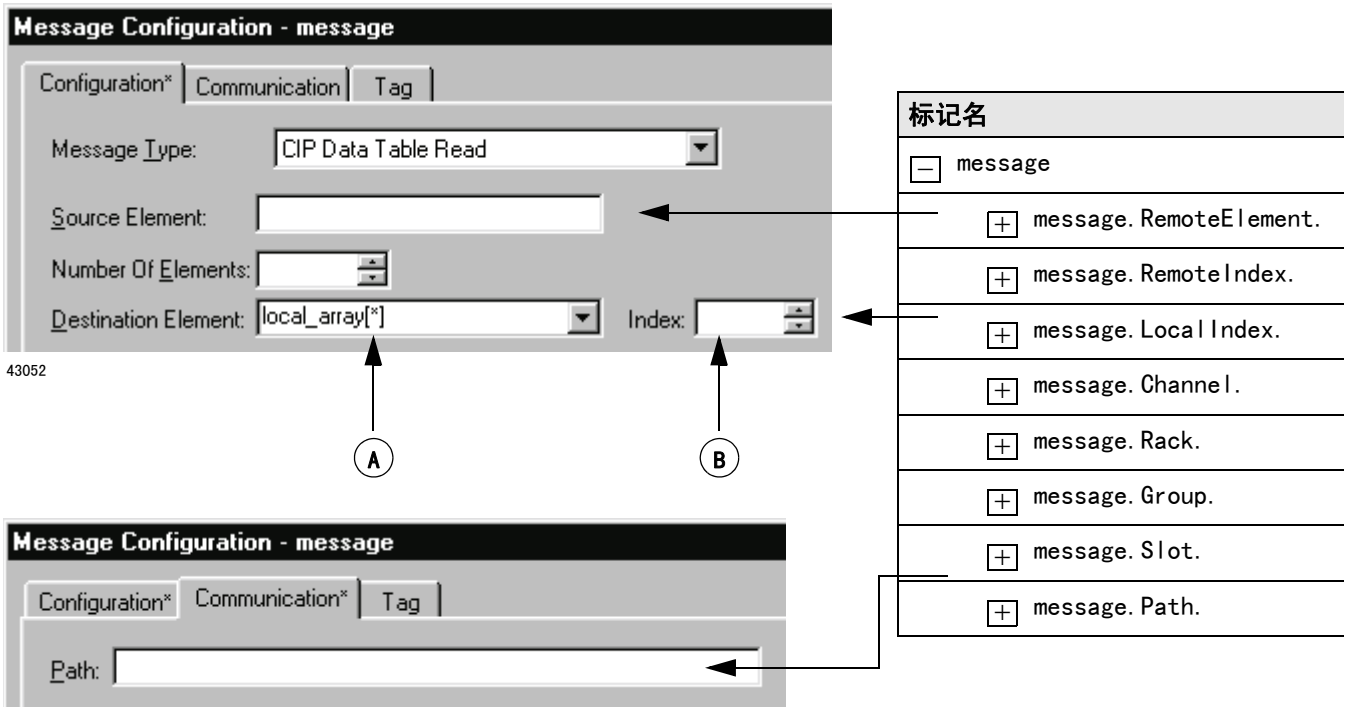
何时使用此附录

使用此附录编制单条消息指令来与多个控制器通信。为了在运行时重新配置 MSG 指令，为 MESSAGE 数据类型成员编写新值。

重要

在 MESSAGE 数据类型中，RemoteElement 成员在接收消息的控制器中存储数据标记名或地址。

如果消息：	则 RemoteElement 是：
读取数据	源元素
写入数据	目标元素



A. 如果您使用星号 [*] 指定数组的元素号码，则 (B) 中的数值提供元素号码。

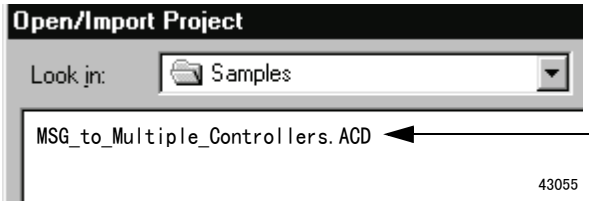
B. 只有当您在源元素或目标元素中使用 [*] 时，索引框才可用。指令替代星号 [*] 的索引值。

向多个控制器发送消息：

- 设置 I/O 配置
- 定义源和目标元素
- 创建 MESSAGE_CONFIGURATION 数据类型
- 创建配置数组
- 获取本地数组的大小
- 为控制器加载消息属性
- 配置消息
- 下一控制器步骤
- 重新启动序列

提示

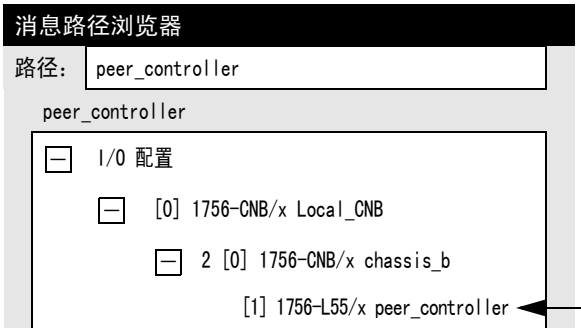
要从示例项目中复制上面的组件，请打开 …\RSLogix 5000\Projects\Samples 文件夹。



设置 I/O 配置

虽然不是必需的，但是我们建议您向控制器的 I/O 配置中添加通信模块和远程控制器。这便于定义每个远程控制器的路径。

例如，一旦您添加本地通信模块、远程通信模块和目标控制器，Browse （浏览器）按钮就可供您选择目标。



定义源和目标元素

此过程中，数组存储从远程控制器读取或写入远程控制器的数据。数组中的每个元素都和不同的远程控制器对应。

1. 使用下面的工作表组织本地控制器和远程控制器上的标记名：

远程控制器名：	远程控制器上的标记或数据地址：	此控制器上的标记：
		local_array[0]
		local_array[1]
		local_array[2]
		local_array[3]

2. 创建 local_array 标记存储此控制器上的数据。

标记名	类型
local_array	*data_type* [*长度*] 地址： *data_type* 是消息发送或接收的数据类型，如 DINT、REAL 或 STRING。 *长度* 是局部数组的元素数目。

创建 MESSAGE_CONFIGURATION 数据类型

此过程中，您创建用户定义的数据类型来向每个控制器存储消息配置变量。

- 数据类型需要的一些成员使用字符串数据类型。
- 默认 STRING 数据类型存储 82 个字符。
- 如果路径或远程标记名或地址使用的字符少于 82 个，您可以选择创建新的字符串类型存储较少的字符。这样可以节约内存。
- 要创建新的字符串类型，选择 File（文件）> New Component（新组件）> String Type（字符串类型）…
- 如果您创建新的字符串类型，在此过程中使用它来代替 STRING 数据类型。

要创建新的数据类型：

右击并选择 *New Data Type*（新建数据类型）。

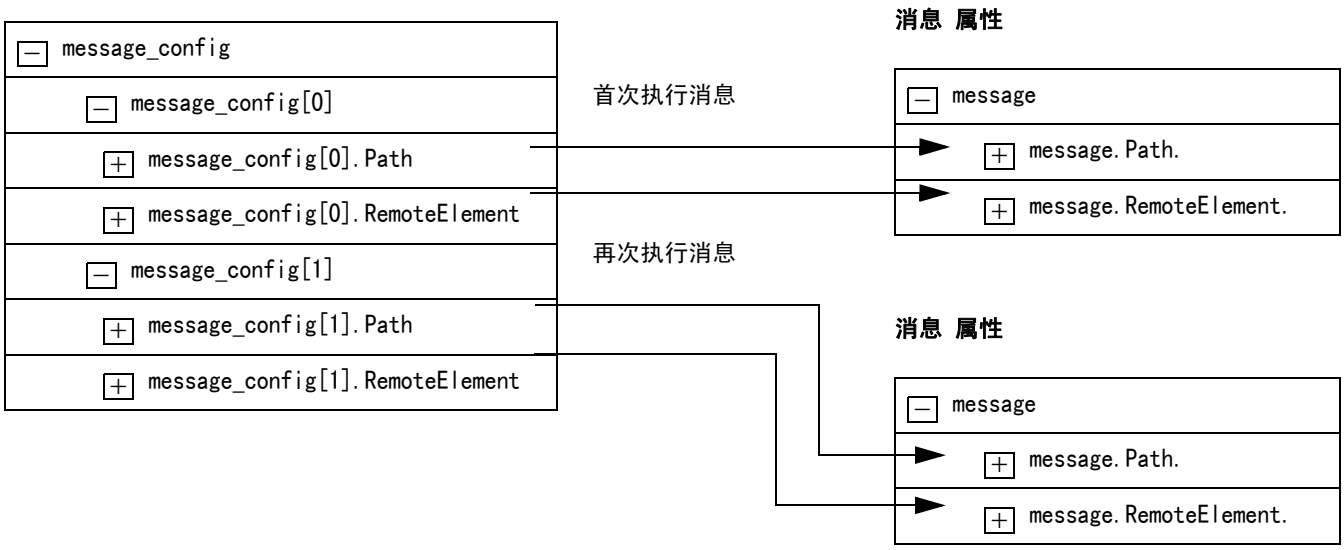
要向每个控制器中存储消息配置变量，请创建下面用户定义的数据类型。

数据类型：MESSAGE_CONFIGURATION				
名称	MESSAGE_CONFIGURATION			
说明	为另一控制器配置消息属性			
成员				
	名称	**数据类型**	**样式**	**说明**
	路径	STRING		
	RemoteElement	STRING		

创建配置数组

在此过程中，以数组的形式为每个控制器存储配置属性。在执行每条 MSG 指令前，您的逻辑程序向指令中加载新的属性。这会向不同控制器发送消息。

图 2.1 向 MSG 指令中加载新的配置属性



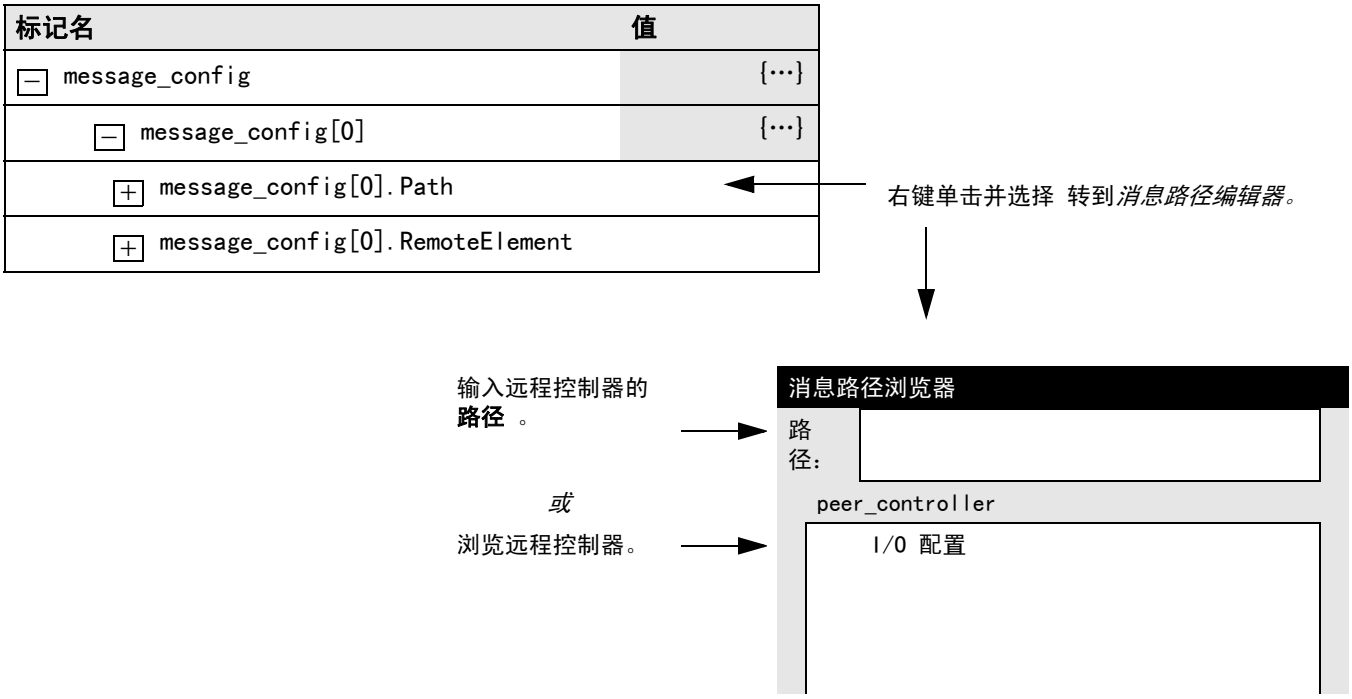
1. 要存储消息配置属性，请创建下面的数组：

标记名	类型	范围
message_config	MESSAGE_CONFIGURATION[*number*]	任何

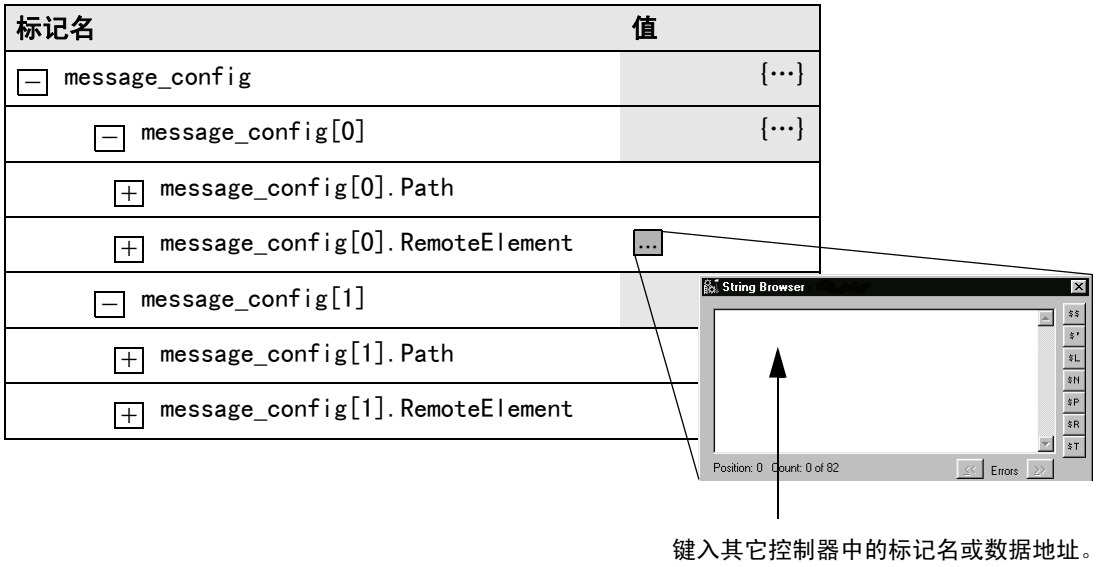
其中：

number 是接收消息的控制器数目。

2. 在 message_config 数组中，向第一个接收消息的控制器输入路径 。



3. 在 message_config 数组中，输入第一个控制器中的标记名或数据地址以便于接收消息。



4. 为每个附加控制器输入路径和远程元素：

标记名	值
⊟ message_config	{…}
⊟ message_config[0]	{…}
⊞ message_config[0].Path	
⊞ message_config[0].RemoteElement	
⊟ message_config[1]	{…}
⊞ message_config[1].Path	←
⊞ message_config[1].RemoteElement	←

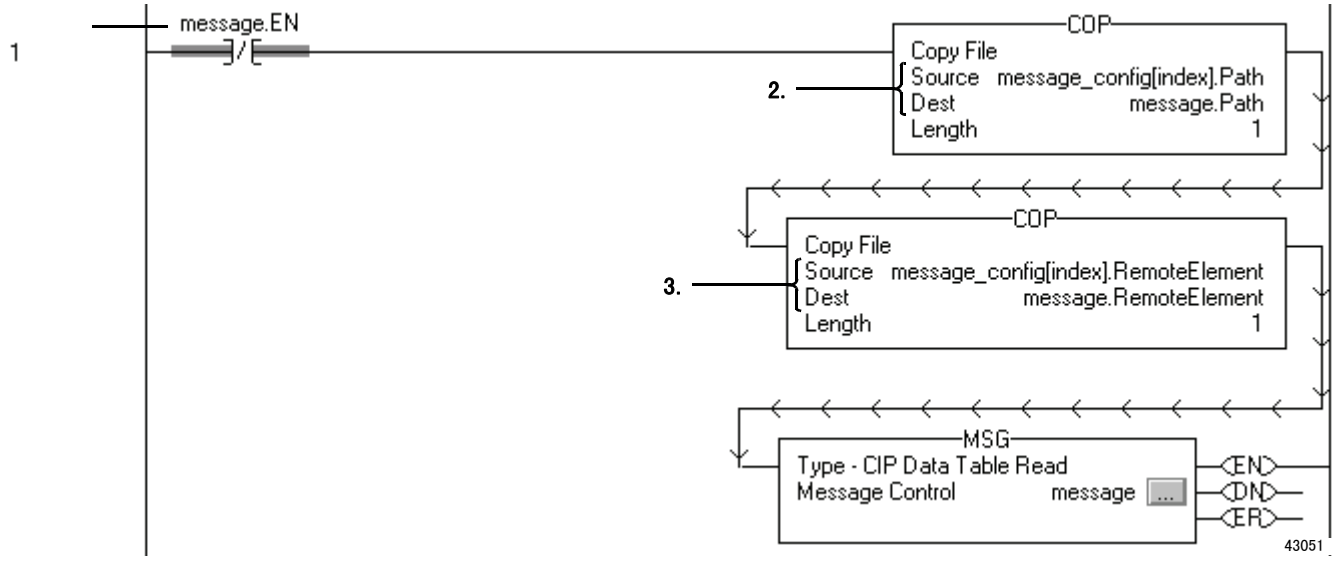
获取本地数组的大小

1. SIZE 指令：

- 计算 local_array 中元素数目。
- 计算维数为 0 数组的元素数目。这种情况下，这是唯一维数。

2. Local_array_length (DINT) 存储 local_array 的大小 （元素数目）。 消息发送到所有控制器并再次以第一个控制器开始时，此数值告知后续语句行。

为控制器加载消息属性



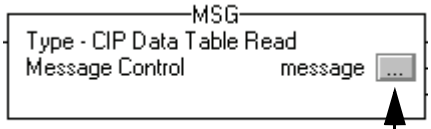
1. XIO 指令为连续发送消息提供条件。

2. 首次 COP 指令加载消息路径。索引值决定指令从 message_config 加载的元素。请参阅第 B-5 页上的图 2.1.

   指令从 message_config 加载 1 个元素。

3. 第二个 COP 指令向接收消息的控制器中加载标记名或数据地址。索引值决定指令从 message_config 加载的元素。 请参阅第 B-5 页上的图 2.1.

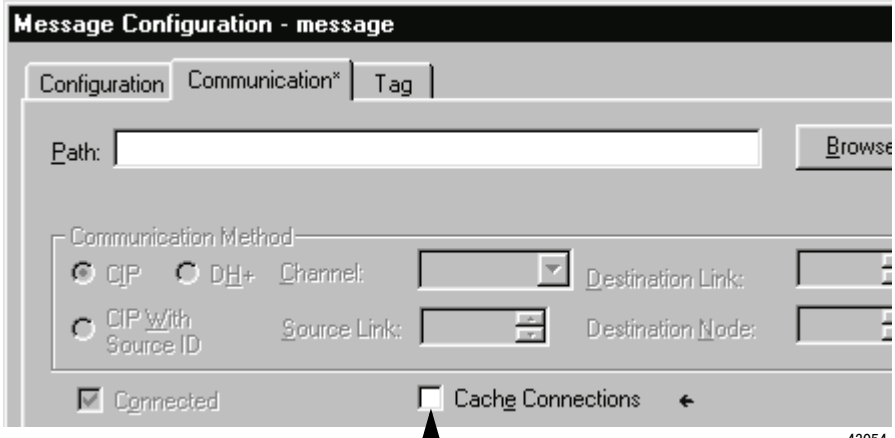
   指令从 message_config 加载 1 个元素。



配置消息

尽管逻辑程序控制远程元素和消息路径，不过还存在初始配置。

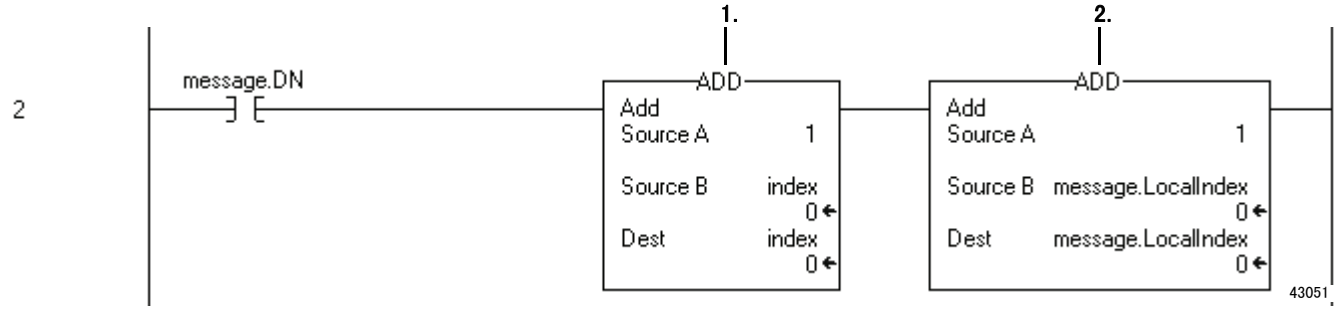
重要



取消选中 *Cache Connection （缓存连接）*复选框

此选项卡上：	如果您要：	对于此项：	键入或选择：
配置	从其它控制器读取 （接收）数据	消息类型	与其它控制器对应的读取类型
		源元素	包含第一个控制器中数据的标记或地址
		元素数目	1
		目标标记	local_array[*]
		索引	0
	向其它控制器写入 （发送）数据	消息类型	与其它控制器对应的写入类型
		源标记	local_array[*]
		索引	0
		元素数目	1
		目标元素	包含第一个控制器中数据的标记或地址
通信	→	路径	第一个控制器路径
		Cache Connections	取消选中 Cache Connection （缓存连接）复选框。由于此过程不断更改消息路径，清除此复选框更有效。

下一控制器步骤

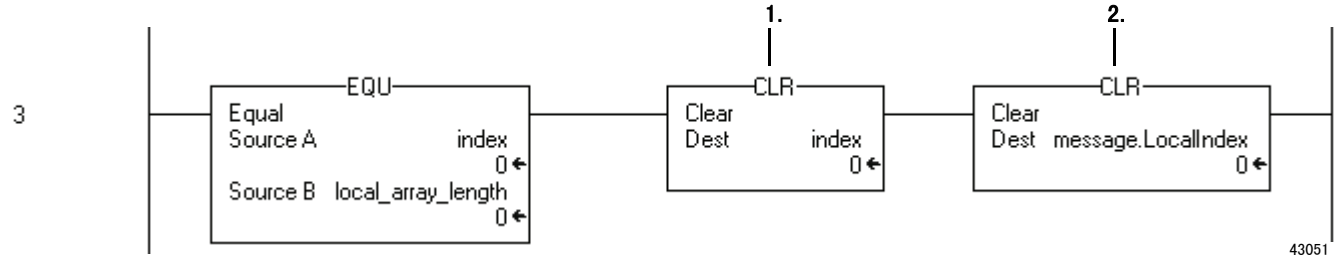


MSG 指令发送消息后…

1. 第一条 ADD 指令使索引递增。这允许逻辑程序为下一控制器的 MSG 指令加载配置属性。

2. 第二条 ADD 指令使 MSG 指令中的 LocalIndex 成员递增。这允许逻辑程序从下一控制器向 local_array 的下一元素加载数值。

重新启动序列



当索引等于 local_array_length，控制器已向所有其它控制器发送消息。

1. 第一条 CLR 指令设置索引等于 0。这要求逻辑程序为第一个控制器的 MSG 指令加载配置属性，并重新启动消息序列。

2. 第二条 CLR 指令设置 MSG 指令的 LocalIndex 成员等于 0。这要求逻辑程序从第一个控制器为 local_array 的第一个元素加载值。

符合 IEC61131-3 标准

使用此附录

介绍

国际电工委员会（IEC）已为可编程控制器开发一套程序规范。这些规范旨在促进控制业设备和编程语言的国际统一性。这些标准为 Logix5000 控制器和 RSLogix 5000 编程软件提供基础。

IEC 可编程控制器规范分解为五个独立的部分，每一部分集中强调控制系统不同的方面：

- 第 1 部分： 概况
- 第 2 部分： 设备和要求测试
- 第 3 部分： 编程语言
- 第 4 部分：用户指南
- 第 5 部分： 信息服务规范

控制业作为一个整体，集中强调第 3 部分 （IEC61131-3），编程语言，因为它为其它标准的实现提供支柱，并通过减少培训成本为最终用户提供最显著的利益。由于这些，这里仅强调 IEC61131-3 。

IEC61131-3 编程语言规范强调可编程控制器多个方面的内容，包括操作系统执行、数据定义、编程语言和指令集。按规范要求，IEC61131-3 规范的组成被分为可选或扩展。通过这样划分，IEC61131-3 规范提供最小功能集，可以对其扩展以满足最终用户应用需要。该方法的不利一面是每个可编程控制系统供应商都可实现规范的不同组件或提供不同的扩展。

操作系统

Logix5000 控制器的强占式多任务操作系统 (OS) 符合 IEC61131-3 定义。在 IEC61131-3 标准中，可编程控制器 OS 可以包含零任务或多任务，可以执行一个或多个程序，每个程序都包含一个或多个功能或例程。按照 IEC61131-3 标准，每个组成的数量与具体实现相关。Logix5000 控制器提供多个任务，每个任务包含多个程序和无限的功能和例程。

IEC61131-3 提供创建多任务执行分类选项。任务可以配置为连续的、周期的和基于事件的。由于连续任务在其它任务休眠时利用任何剩下的时间，所以不需要提前计划。周期任务基于重新出现的时间周期提前计划操作。IEC61131-3 未为周期任务的配置指定时基。基于 IEC61131-3 事件的任务在检测到所配置输入的上升沿时触发。Logix5000 控制器同时支持连续任务和周期任务。另外，周期任务的周期起点可配置为低于 1 毫秒 (ms)。

数据定义

IEC61131-3 规范通过创建命名变量来提供访问内存的能力。IEC61131-3 变量名最少由六个字母组成 （RSLogix5000 编程软件支持最少 1 个字符），首字母为下划线 “_” 或字母字符 (A-Z)，后面是一个或多个字符，包括 “_”、字母字符 (A-Z) 或数字 (0-9)。 可选地，只要不区分大小写 (A = a, B = b, C = c …)，也支持小写字母字符 (a-z)。Logix5000 控制器完全符合此定义，支持小写选项，扩展为支持长达 40 字符的名称。

IEC61131-3 中的数据变量可定义为可被源或控制器中所有程序访问，或限制为仅可被单个程序中的功能或例程来访问。要在多个资源或控制器间传递数据，可配置访问路径来定义系统中数据的位置。Logix5000 控制器通过程序范围、控制器范围数据来符合标准并允许使用生成 / 使用数据来配置访问路径。

IEC61131-3 中变量的内存解释使用基本数据类型或可选派生数据类型 （由一组多数据类型创建）来定义。Logix5000 控制器支持 BOOL （1 位）、SINT （8 位整数）、INT （16 位整数）、DINT （32 位整数）和 REAL （IEEE 浮点数）等基本数据类型的使用。另外，通过创建用户定义的结构和数组支持可选派生数据类型。

编程语言

IEC61131-3 规范定义五种 (5) 不同的编程语言和一套公用语言元素。所有语言都定义为可选，但是必须至少支持一种语言，以便于符合规范要求。IEC61131-3 编程语言组成定义如下：

- 公用语言元素
- 公用图形元素
- 指令集 (IL) 语言元素
- 结构化文本语言 (ST) 元素
- 梯形图 (LD) 语言元素
- 流程图 (SFC) 语言元素
- 方框图 (FBD) 语言元素

Logix5000 控制器和 RSLogix5000 支持共用语言元素和结构化文本、梯形图、流程图和方框图语言选项。另外，环境利用基于结构化文本语言的 ASCII 导入 / 导出格式。指令集和程序文件交换功能在下面部分详细讨论。

指令集

IEC61131-3 指定的指令集全部可选。规范列出有限的指令，如果实现，必须遵守规定的执行和可视化表示方法。但是，IEC61131-3 并不局限于在规范中列出的那些指令集。每家 PLC 供应商都可自由以指令形式实现规范所列以外的附加功能。在执行诊断、PID 循环控制、运动控制和数据文件操作时所需的指令就属于这类扩展指令。由于 IEC61131-3 规范未定义扩展指令，不保证不同 PLC 供应商实现的指令间相互兼容。因此，利用这些指令时应避免供应商间的逻辑移动。

Logix5000 控制器和 RSLogix5000 提供了一整套指令，按照 IEC61131-3 规范定义的方式执行。这些指令的外在形式保持了其在现有系统中的样式，以便减少与环境相关的培训成本。除了符合 IEC61131-3 的兼容指令，已将现有产品中的各种指令引入环境中，因此不会失去任何功能。

IEC61131-3 程序可移植性

最终用户在符合 IEC61131-3 的环境中创建程序的一个目标是保证程序在不同供应商开发的控制器之间的移动性和可移植性。这是 IEC61131-3 的一个弱项，因为规范未定义文件交换格式这意味着，如果程序是在一个供应商环境中创建的，要将它移到其它供应商系统就需要采取额外的处理。

为了尽量减少在各个供应商之间实现可移植性的努力，控制器的 RSLogix 5000 编程软件提供了一个全面的 ASCII 导出和导入实用工具。另外，此工具使用的文件格式融合了 IEC61131-3 结构化文本语言定义。控制器操作系统和数据定义均遵循相应的 IEC61131-3 格式。还实现了一系列扩展以将梯形图逻辑转换为 ASCII 文本，因为这在 IEC61131-3 中并未定义。

关于 RSLogix 5000 编程软件 ASCII 导出和导入实用工具的更多信息，请参阅 *Logix5000 控制器导入／导出参考手册*，出版物 1756-RM084。

IEC 法规遵守表

Logix5000 控制器和 RSLogix5000 符合 IEC61131-3 要求，提供下面的语言功能：

表号：(1)	功能号：	功能描述：	扩展和实现注释：
1	2	小写字母	无
1	3a	编号标记 (#)	用于指定立即数数据类型
1	4a	美元标记 ($)	用于描述和字符串控制字符
1	6a	下标分隔符 ([ ])	数组下标
2	1	使用大写字母和数字的标识符	任务、程序、例程、结构和标记名
2	2	使用大写字母、数字和嵌入式下划线的标识符	任务、程序、例程、结构和标记名
2	3	使用大小写字母、数字和嵌入式下划线的标识符	任务、程序、例程、结构和标记名
3	1	注释	ST Comments 注释，也支持 /* Comment */ 和 // End of line 注释。
4	1	整数	12, 0, -12
4	2	实数	12.5, -12.5
4	3	带指数的实数	-1.34E-12, 1.234E6
4	4	二进制数	2#0101_0101
4	5	八进制数	8#377
4	6	十六进制数	16#FFE0
4	7	布尔值零和一	0, 1
5	1A	空字符串''	描述和字符串编辑器
5	1B	包含字符 'A' 的一个长度的字符串	描述和字符串编辑器
5	1C	包含空格 ' ' 的一个长度的字符串	描述和字符串编辑器
5	1D	包含单引号 '$'' 的一个长度的字符串	描述和字符串编辑器
5	1E	包含双引号 '"' 的一个长度的字符串	描述和字符串编辑器
5	1F	包含字符 CR 和 LF 的两个长度的字符串	描述和字符串编辑器
5	1G	包含 LF 字符 '$0A' 的一个长度的字符串	描述和字符串编辑器
5	1H	使用 '$$1.00' 显示为 "$1.00" 的 5 个长度的字符串	描述和字符串编辑器
5	1I	长度为 2 的等效字符串 'AE' 和 '$C4$CB'	描述和字符串编辑器
6	2	字符串美元标记 '$$'	描述和字符串编辑器

表号：(1)	功能号：	功能描述：	扩展和实现注释：
6	3	字符串单引号 '$''	描述和字符串编辑器
6	4	字符串换行符 '$L' 或 '$l'	描述和字符串编辑器
6	5	字符串换行字符 '$N' 或 '$n'	描述和字符串编辑器
6	6	字符串换页字符 '$P' 或 '$p'	描述和字符串编辑器
6	7	字符串回车字符 '$R' 或 '$r'	描述和字符串编辑器
6	8	字符串 Tab 键 '$T' 或 '$t'	描述和字符串编辑器
6	9	字符串双引号 $″	描述和字符串编辑器
10	1	BOOL 数据类型	标记变量定义
10	2	SINT 数据类型	标记变量定义
10	3	INT 数据类型	标记变量定义
10	4	DINT 数据类型	标记变量定义
10	10	REAL 数据类型	标记变量定义
10	12	时间	标记变量定义，TIMER 结构
10	16	STRING 数据类型	8 位
11	1	数据类型层次结构	无
12	1	从基本类型直接派生	用户定义的数据类型结构
12	4	数组数据类型	标记变量定义
12	5	结构化数据类型	用户定义的数据类型结构
13	1	BOOL、SINT、INT, DINT 初始值为 0	标记变量定义
13	4	REAL、LREAL 初始值为 0.0	标记变量定义
13	5	时间初始值为 T#0s	标记变量定义，重置 (RES) 指令
13	9	空字符串''	描述和字符串
14	1	直接派生数据类型初始化	导入 / 导出
14	4	数组数据类型初始化	导入 / 导出
14	5	结构化类型元素初始化	导入 / 导出
14	6	派生的结构化数据类型初始化	导入 / 导出
19a	2a	上下文调用，非正式	可用于 ST 中
20	1	使用 EN 和 ENO	在 LD 中存在的功能但未标记。可用于 FBD 中。
20	2	不使用 EN 和 ENO	可用于 FBD 中
20	3	使用 EN 而不使用 ENO	可用于 FBD 中
20	4	不使用 EN 而使用 ENO	可用于 FBD 中
21	1	重载函数 ADD(INT, DINT) 或 ADD(DINT, REAL)	所有支持的重载类型都在每个指令中说明
22	1	_TO_ 转换函数	RAD, DEG 指令弧度 / 小数转换。字符串数字转换 STOD、STOR、RTOS 和 DTOS。由于指令重载，不需要其它

表号：(1)	功能号：	功能描述：	扩展和实现注释：
22	2	截断转换函数	LD 中的 TRN 指令和 ST 中的 TRUNC 函数
22	3	BCD 转换为 INT	LD 中的 FRD 指令
22	4	INT 转换为 BCD	LD 中的 TOD 指令
23	1	绝对值	ABS 指令
23	2	平方根	LD 和 FBD 中的 SQR 指令，ST 中的 SQRT 函数。
23	3	自然对数	LN 指令
23	4	以 10 为底的对数	LOG 指令
23	6	以弧度为单位之正弦函数	SIN 指令 / 函数
23	7	以弧度为单位之余弦函数	COS 指令 / 函数
23	8	以弧度为单位之正切函数	TAN 指令 / 函数
23	9	反正弦函数主值	LD 和 FBD 中的 ASN 指令，ST 中的 ASIN 函数
23	10	反余弦函数主值	LD 和 FBD 中的 ACS 指令，ST 中的 ACOS 函数
23	11	反正切函数主值	LD 和 FBD 中的 ATN 指令，ST 中的 ATAN 函数
24	12	算术加	LD 和 FBD 中的 ADD 指令，ST 中的 + 指令。
24	13	算术乘	LD 和 FBD 中的 MUL 指令，ST 中的 * 指令。
24	14	算术减	LD 和 FBD 中的 SUB 指令，ST 中的 - 指令。
24	15	算术除	LD 和 FBD 中的 DIV 指令，ST 中的 / 指令。
24	16	模数	LD 和 ST 中的 MOD 指令
24	17	乘方	LD 和 FBD 中的 XPY 指令，ST 中的 ** 指令。
24	18	值传送	LD 中的 MOV 指令，和 ST 中的 := 。
25	1	左移位	LD 中 BSL 指令包含的功能，用于移动 1 位
25	2	右移位	LD 中 BSR 指令包含的功能，用于移动 1 位
25	3	循环左移位	LD 中 BSL 指令包含的功能，用于移动 1 位
25	4	循环右移位	LD 中 BSR 指令包含的功能，用于移动 1 位
26	5	与	FBD 中的 BAND 指令，ST 中的 "&" 运算符
26	6	或	FBD 中的 BOR 指令
26	7	异或	FBD 中的 BXOR 指令

表号：(1)	功能号：	功能描述：	扩展和实现注释：
26	8	非	FBD 中的 BNOT 指令
27	1	选择	FBD 中的 SEL 指令
27	2a	最大选择 MAX	FBD 和 ST 中的 ESEL 指令包含的功能
27	2b	最小选择 MIN	FBD 和 ST 中的 ESEL 指令包含的功能
27	3	上限 / 下限 LIMIT	FBD 和 ST 中的 HLL 指令
27	4	多路复用器 MUX	FBD 中的 MUX 指令
28	5	大于	LD 和 FBD 中的 GRT 指令，ST 中的 >。
28	6	大于或等于	LD 和 FBD 中的 GRT 指令，ST 中的 >= 。
28	7	等于	LD 和 FBD 中的 EQU 指令，ST 中 =。
28	8	小于	LD 和 FBD 中的 LES 指令，ST 中的 <。
28	9	小于或等于	LD 和 FBD 中的 LEQ 指令，ST 中 <=。
28	10	不等于	LD 和 FBD 中的 NEQ 指令，ST 中 <>。
29	1	字符串长度 LEN	作为 STRING 数据类型的参数
29	4	抽取字符串 MID	LD 和 ST 中的 MID 指令
29	5	字符串连接 CONCAT	LD 和 ST 中的 CONCAT 指令
29	6	字符串插入 INSERT	LD 和 ST 中的 INSERT 指令
29	7	字符串删除 DELETE	LD 和 ST 中的 DELETE 指令
29	9	查找字符串 FIND	LD 和 ST 中的 FIND 指令
32	1	输入读取	FBD 和 ST
32	2	输入写入	FBD 和 ST
32	3	输出读取	FBD 和 ST
32	4	输出写入	FBD 和 ST
34	1	置位优先	FBD 和 ST 中的 SETD 指令
34	2	复位优先	FBD 和 ST 中的 RESD 指令
35	1	上升沿触发	LD 中的 OSR 指令，FBD 和 ST 中的 OSRI 指令
35	2	下降沿触发	LD 中的 OSF 指令，FBD 和 ST 中的 OSFI 指令
36	1B	加计数	LD 中 CTU 和 RES 指令包含的功能，以及 FBD 和 ST 中 CTUD 指令包含的功能
37	2a	延时导通计时器	LD 中的 TON 指令包含的功能，以及 FBD 和 ST 中 TONR 指令包含的功能
37	3a	延时断开计时器	LD 中的 TOF 指令包含的功能，以及 FBD 和 ST 中 TOFR 指令包含的功能
38	2	延时接通	LD 中的 TON 指令包含的功能，以及 FBD 和 ST 中 TONR 指令包含的功能

表号：(1)	功能号：	功能描述：	扩展和实现注释：
38	3	延时断开	LD 中的 TOF 指令包含的功能，以及 FBD 和 ST 中 TOFR 指令包含的功能
40	1A	SFC 步骤	
40	1B	SFC 初始步骤	
40	2a	SFC 步骤文本化	导入 / 导出，使用格式 ″操作数 := 步骤名″ 指定步骤名
40	2b	SFC 初始步骤文本化	导入 / 导出，使用 ″初始步骤 ″ 参数，使用格式″操作数 := 步骤名″ 指定步骤名
40	3a	SFC 步骤标志的一般形式	步骤支持标记
40	4	步骤消耗时间的一般形式	步骤支持标记
41	1	使用 ST 转换	
41	5	转换文本形式	使用不同格式导入 / 导出
41	7	转换名	转换支持标记
41	7a	LD 设置转换	转换支持标记
41	7b	FBD 设置转换	转换支持标记
41	7d	ST 设置转换	转换支持标记
42	1	操作布尔值	操作支持标记
42	3s	操作文本表示形式	导入 / 导出
43	1	步骤操作关联	
43	2	具有级联操作的步骤	
43	3	文本步骤体	使用不同格式导入 / 导出
43	4	操作体字段	嵌入式 ST
44	1	操作块限定符	
44	2	操作块名	
44	3	操作指示器标记	扩展以支持 BOOL 外的 DINT、INT、SINT 或 REAL 数据类型
44	5	使用 ST 操作	在 ST 例程同时支持内嵌式 ST 和 JSR
44	6	使用 LD 操作	在 LD 例程使用 JSR
44	7	使用 FBD 操作	在 FBD 例程使用 JSR
45	1	无操作限定符	明确输入为无时，默认为 N
45	2	操作限定符 N - 未存储	
45	3	操作限定符 R - 重置	
45	4	操作限定符 S - 设置 / 存储	
45	5	操作限定符 L - 限时	
45	6	操作限定符 D - 延时	
45	7	操作限定符 P - 脉冲	
45	8	操作限定符 SD - 存储和延时	

表号：(1)	功能号：	功能描述：	扩展和实现注释：
45	9	操作限定符 DS - 延时和存储	
45	10	操作限定符 SL - 存储和限时	
45	11	操作限定符 P1 - 脉冲上升沿	
45	12	操作限定符 P0 - 脉冲下降沿	
45a	1	操作控制	
45a	2	操作控制	
46	1	SFC 单个程序	
46	2a	SFC 程序选择发散	使用线路连接与星号
46	2b	具有执行顺序的 SFC 程序选择发散。	
46	3	SFC 程序选择汇聚	
46	4a	SFC 同时程序发散	
46	4b	SFC 同时程序汇聚	
46	5a, b, c	跳过 SFC 程序	
46	6a, b, c	SFC 程序循环	
46	7	SFC 循环定向箭头	线路隐藏时
47	1	SFC 图形表示形式	
47	4	SFC 图形表示形式	
48	1	符合 SFC 最小步骤要求	参阅上面各个表的注释。
48	2	符合 SFC 最小转换要求	参阅上面各个表的注释。
48	3	符合 SFC 最小操作要求	参阅上面各个表的注释。
48	4	符合 SFC 最小操作体要求	参阅上面各个表的注释。
48	5	符合 SFC 最小操作限定符要求	参阅上面各个表的注释。
48	6	符合 SFC 最小分支要求	参阅上面各个表的注释。
48	7	符合 SFC 最小块连接要求	参阅上面各个表的注释。
55	1	ST 插入语 （表达式）	
55	2	ST 函数计算	为内置函数使用非正式形式调用。ST 语言中使用 JSR 来调用用户开发的代码。
55	3	ST 乘方 **	
55	4	ST 负 -	
55	5	ST 负 NOT	
55	6	ST 乘 *	
55	7	ST 除 /	
55	8	ST 模 MOD	
55	9	ST 加 +	
55	10	ST 减 -	
55	11	ST 比较 <、>、<=、>=	

表号：(1)	功能号：	功能描述：	扩展和实现注释：
55	12	ST 等号 =	
55	13	ST 不等号 <>	
55	14	ST 布尔与 &	
55	15	ST 布尔与	
55	16	ST 布尔异或	
55	17	ST 布尔或	
56	1	ST 赋值 :=	
56	2	ST 功能块调用	
56	3	ST RETURN	RET( ) 带有多个参数
56	4	ST IF / ELSIF / ELSE/ END_IF	
56	5	ST CASE OF / ELSE / END_CASE	
56	6	ST FOR / END_FOR	
56	7	ST WHILE DO / END_WHILE	
56	8	ST REPEATE / UNTIL / END_REPEAT	
56	9	ST EXIT	
56	10	ST 空语句 ;	
57	1, 2	水平线	LD 编辑器、FBD 编辑器
57	3, 4	垂直线	LD 编辑器、FBD 编辑器
57	5, 6	水平连接 / 垂直连接	LD 编辑器、FBD 编辑器
57	7, 8	线路交叉而未连接	FBD 编辑器
57	9, 10	连接和无连接角	LD 编辑器、FBD 编辑器
57	11, 12	连接块	LD 编辑器、FBD 编辑器
57	13,14	连接器	FBD 编辑器
58	2	无条件跳转	LD 中的 JMP 指令
58	3	跳转目标	LD 中的 LBL 指令
58	4	有条件跳转	LD 中的 JMP 指令
58	5	有条件返回	LD 中的 RET 指令
58	8	无条件返回	LD 中的 RET 指令
59	1	左侧供电导轨	LD 编辑器
59	2	右侧供电导轨	LD 编辑器
60	1	水平链接	LD 编辑器
60	2	垂直链接	LD 编辑器
61	1, 2	常开接点 --| |--	LD 中的 XIC 指令
61	3, 4	常闭接点 --| / |--	LD 中的 XIO 指令
61	5, 6	正转换感应接点 -| P |-	LD 中的 ONS 指令
62	1	线圈 --( )--	LD 中的 OTE 指令
62	3	设置 （闩锁）线圈	LD 中的 OTL 指令包含的功能

表号：[1]	功能号：	功能描述：	扩展和实现注释：
62	4	重置 （非闭锁）线圈	LD 中的 OTU 指令包含的功能
62	8	正转换感应线圈	LD 中的 OSR 指令
62	9	负转换感应线圈	LD 中的 OSF 指令

[1] 已略过与结构化文本、流程图、梯形图和功能块图以外语言关联的表。

A

alias 标记 引用其他标记的标记。alias 标记可以引用其他 alias 标记或 base 标记。alias 标记还可以通过引用结构成员、数组元素或者标记或成员中的位来引用其他标记的组成部分。请参见 **base 标记**。

ASCII 用于表示数字字母字符、标点符号和控制代码字符的 7 位代码 （带有一个可选校验位）。有关 ASCII 代码的列表，请参见本手册的封底。

B

base 标记 实际定义存储数据元素的内存的标记。请参见 **alias 标记**。

BOOL 存储单个位状态的数据类型，其中：

- 0 表示 off
- 1 表示 on

BOOL 表达式 在结构化文本中，产生 BOOL 值 1 (true) 或 0 (false) 的表达式。

- bool 表达式使用 bool 标记、关系运算符和逻辑运算符比较值或检查条件为 true 或 false。例如 `tag1>65`。
- 简单 bool 表达式可以是单个 BOOL 标记。
- 通常，使用 bool 表达式决定其他逻辑执行的条件。

C

CIP 请参见**控制与信息协议**。

COUNTER 包含计数器指令的状态和控制信息的结构数据类型

D

DINT 存储 32 位 （4 字节）有符号整数值 （-2,147,483,648 至 +2,147,483,647）的数据类型。在 Logix5000 控制器中，对整数使用 DINT：

- 处理 32 位整数 (DINT) 而不是 16 位整数 (INT) 或 8 位整数 (SINT) 时，Logix5000 控制器执行效率更高，使用更少的内存。

- 通常，执行期间指令将 SINT 或 INT 值转换为**最优数据类型**（通常为 DINT 或 REAL 值）。因为这需要额外时间和内存，所以请尽量减少 SINT 和 INT 数据类型的使用。

I

INT 存储 16 位 （2 字节）整数值 （-32,768 至 +32,767）的数据类型。尽量减少此数据类型的使用：

- 通常，执行期间指令将 SINT 或 INT 值转换为**最优数据类型**（通常为 DINT 或 REAL 值）。因为这需要额外时间和内存，所以请尽量减少 SINT 和 INT 数据类型的使用。

R

REAL 存储 32 位 （4 字节）IEEE 浮点值的数据类型，范围如下：

- $-3.40282347E^{38}$ 至 $-1.17549435E^{-38}$ （负值）
- 0
- $1.17549435E^{-38}$ 至 $3.40282347E^{38}$ （正值）

REAL 数据类型还存储 ±无穷、± NAN 和 -IND，但软件根据显示格式不同而显示不同。

显示格式：	等价于：	
实数	+ 无穷	1.$
	- 无穷	-1.$
	+NAN	1.#QNAN
	-NAN	-1.#QNAN
	- 不定	-1.#IND
指数	+ 无穷	1.#INF000e+000
	- 无穷	-1.#INF000e+000
	+NAN	1.#QNAN00e+000
	-NAN	-1.#QNAN00e+000
	- 不定	-1.#IND0000e+000

软件还存储和显示 IEEE 低于正常范围：

- $-1.17549421E^{-38}$ 至 $-1.40129846E^{-45}$ （负值）
- $1.40129846E^{-45}$ 至 $1.17549421E^{-38}$ （正值）

S

SINT 存储 8 位 （1 字节）有符号整数值 （-128 至 +127）的数据类型。尽量减少此数据类型的使用：

- 通常，执行期间指令将 SINT 或 INT 值转换为**最优数据类型**（通常为 DINT 或 REAL 值）。因为这需要额外时间和内存，所以请尽量减少 SINT 和 INT 数据类型的使用。

八进制 以基数 8 显示和输入的整数值 （每个数字表示 3 位）。以 8# 为前缀。 填充至布尔值或整数值的长度 （1、8、16 或 32 位）。显示时，每组三个数位由下划线分隔以保持可读性。请参见**二进制**、**十进制**、**十六进制**。

标记 控制器内存中存储数据的命名区域。

- 标记是分配内存、从逻辑引用数据以及监视数据的基础机制。
- 标记的最小内存分配为 4 字节。
  - 创建存储 BOOL、SINT 或 INT （小于 4 字节）的标记时，控制器分配 4 字节，但数据只填充所需的部分。
  - 用户定义的数据类型和数组在连续内存中存储数据，并将较小的数据类型封装在 32 位字中。

以下示例显示各种标记的内存分配：

– start，使用 BOOL 数据类型：

内存分配	**位**	
	31 ... **1**	**0**
分配	不使用	start

– station_status，使用 DINT 数据类型：

内存分配：	**位**	
	31 ... **0**	
分配	station_status	

– mixer，使用用户定义的数据类型：

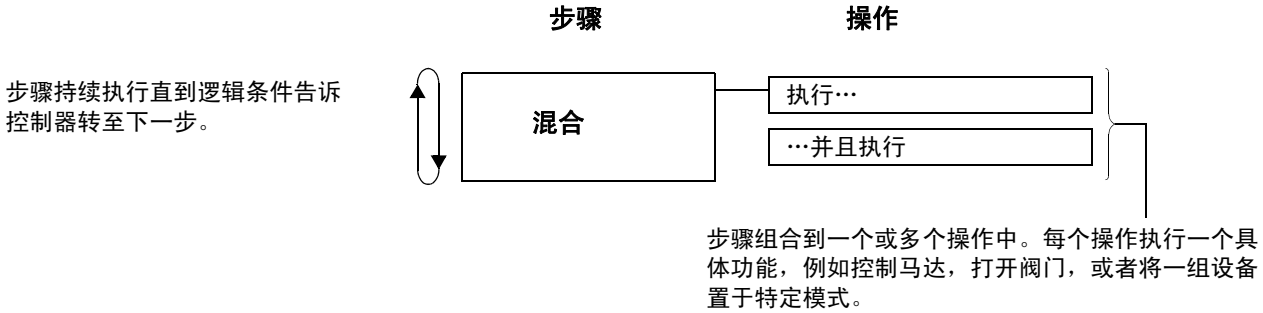
内存分配	位			
	31 24	23 16	15 8	7 0
分配 1	mixer.pressure			
分配 2	mixer.temp			
分配 3	mixer.agitate_time			
分配 4	未使用	未使用	未使用	位 0 mixer.inlet 位 1 mixer.drain 位 2 mixer.agitate

– temp_buffer，包含四个 INTS （INT[4] ） 的数组：

内存分配：	位	
	31 16	0
分配 1	temp_buffer[1]	temp_buffer[0]
分配 2	temp_buffer[3]	temp_buffer[2]

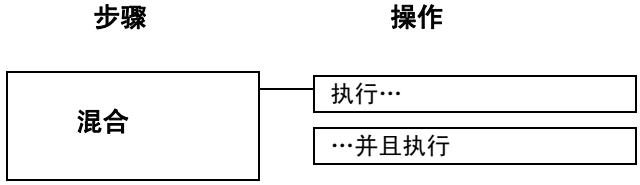
请参见 ***alias 标记***、**base 标记**、**使用标记**。

步骤 在流程图 (SFC) 中，步骤表示过程的一个主要功能。它包含特定时间、阶段或位置发生的事件。



请参见**操作**、**流程图**、**转变**。

操作 在流程图 (SFC) 中，操作表示步骤的一个流程分支。多个操作组成一个步骤。每个操作执行一个具体功能，例如控制马达，打开阀门，或者将一组设备置于特定模式。



每个操作包含一个限定符。当步骤激活 （正在执行）时，限定符决定操作何时开始和停止。

请参阅**流程图**、**步骤**、 **限定符**。

产品定义结构 自动由软件和控制器定义的结构数据类型。通过配置 I/O 模块，可以为该模块添加产品定义结构。

成员 具有自己的数据类型和名称的结构元素。

- 成员也可以是结构，创建嵌套结构数据类型。
- 结构中的每个成员可以是不同数据类型。
- 若要引用结构中的成员，可使用下面的格式：

  tag_name.member_name

  例如：

此地址：	引用：
timer_1.pre	timer_1 结构的 PRE 值。
input_load 作为数据类型 load_info input_load.height	用户定义的 input_load 结构的 height 成员

- 如果结构嵌入在其他结构中，请使用结构的最高级别标记名称，后接子结构标记名称和成员名称。

  tag_name.substructure_name.member_name

  例如：

此地址：	引用：
input_location 作为数据类型位置 input_location.load_info.height	input_location 结构中 load_info 结构的 height 成员。

- 如果结构定义数组，请使用数组标记，后接数组中的位置以及任何子结构和成员名称。

  array_tag[position].member

  或

  array_tag[position].substructure_name.member_name

例如：

此地址：	引用：
conveyor[10].source	conveyor 数组中第 11 个元素的 source 成员 （数组元素从零开始）。
conveyor[10].info.height	conveyor 数组的第 11 个元素中 info 结构的 height 成员 （数组元素从零开始）。

请参见*结构*。

程序 一组相关例程和标记。

- 每个程序包含程序标记、主可执行例程、其他例程和一个可选故障例程。
- 若要执行程序中的例程，可以将该程序分配 （规划）给任务：
  - 触发任务时，任务内计划的程序从头到尾执行至完成。
  - 任务执行程序时，程序的主例程首先执行。
  - 而主例程可以使用 JSR 指令执行子例程。
- Unscheduled Programs （非计划程序）文件夹包含没有分配给任务的程序。
- 如果程序中的逻辑产生主故障，则执行跳转至为程序配置的故障例程。
- 程序中的例程可访问以下标记：
  - 程序的程序标记
  - 控制器标记
- 例程不能访问其他程序的程序标记。

请参见*例程*、**任务。**

程序范围 仅在当前程序中可访问的数据。每个程序包含一组只能由该程序中的例程和 alias 标记引用的标记。请参见*控制器范围*。

持续任务 持续运行的任务。

- 持续任务在后台运行。任何没有分配给其他操作 （例如运动、通信和定期任务）的 CPU 时间都用于执行持续任务内的程序。
- 持续任务在其最后一个程序完成后重新开始。
- 项目不需要持续任务。
- 如果使用，只能有一个持续任务。
- 所有定期任务中断持续任务。
- 创建项目时，默认 MainTask 为持续任务。您可以保留此任务原样，或者更改其属性 （名称、类型等）。

请参见*定期任务*。

次版本 1756 模块系列具有主版本和次版本指示器。 每次有不影响模块功能或界面的模块更改时更新次版本。请参见**电子密钥**、**主版本**。

次故障 ***不*** 足以使控制器关闭的故障情况：

如果发生：	控制器：
指令问题	1. 设置 S:MINOR 2. 将故障信息记录到 PROGRAM 对象 MinorFaultRecord 属性 3. 设置 FAULTLOG 对象 MinorFaultBits 属性第 4 位
定期任务重叠	设置 FAULTLOG 对象 MinorFaultBits 属性第 6 位
串行端口问题	设置 FAULTLOG 对象 MinorFaultBits 属性第 9 位
电量不足	设置 FAULTLOG 对象 MinorFaultBits 属性第 10 位

清除次故障：

1. 在控制器组织器中，右击控制器 *name_of_controller* 文件夹并选择 Properties （属性）。
2. 单击 Minor Faults （次故障）选项卡。
3. 使用 Recent Faults （最近故障）列表中的信息纠正故障起因。参考第 16-4 页上的″次故障代码″。
4. 单击 Clear Minors （清除次故障）按钮。

请参见***主故障***。

存储 将项目复制到控制器的非易失性内存。这将覆盖非易失性内存中的当前项目。请参见**加载**、**非易失性内存**。

单向连接 数据单向流动的连接：从发生处到接收处。请参见**连接**、**单向连接**。

电子密钥 1756 I/O 线的一项功能，可请求模块执行电子检查以确保物理模块与软件配置的模块一致。使用户通过软件防止误用不正确模块或不正确版

定期任务 操作系统在重复时间段内触发的任务。

- 对需要准确或确定性执行的功能使用定期任务。
- 到时间后，触发任务并执行其程序。
- 任务中的程序建立的数据和输出保留它们的值，直到下次执行任务或被其他任务操作。
- 可以将时间段配置在 1 ms 至 2000 s。默认值为 10 ms。

注意

确保时间段不长于分配给任务的所有程序的总执行时间。如果控制器检测到定期任务触发器对正在运行的任务触发，则将发生次故障。

- 定期任务始终中断持续任务。
- 根据优先级，定期任务可能中断控制器中的其他定期任务。

请参见**持续任务**。

本的模块。请参见**兼容模块**、**禁用密钥**、**精确匹配**。

定期任务重叠 当一个任务正在执行而再次触发此同一任务时发生的情况。任务的执行时间大于为该任务配置的定期率。请参见***定期任务***。

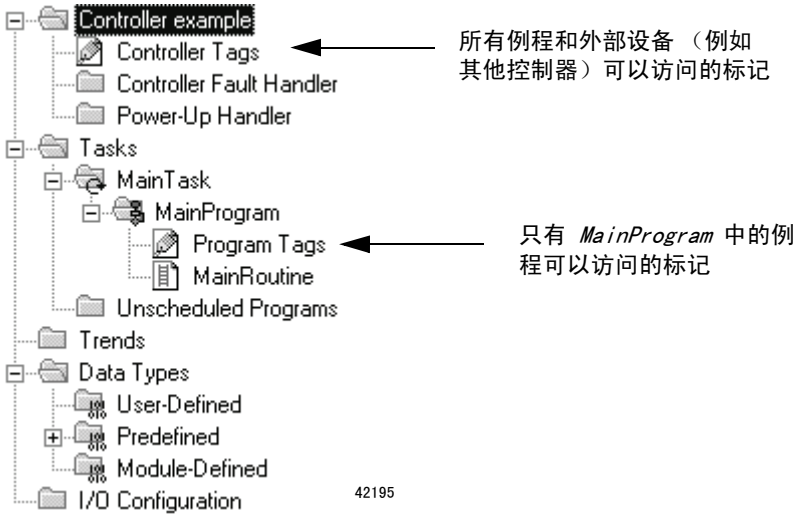
对象 存储状态信息的数据结构。输入 GSV/SSV 指令时，您指定要访问的对象及其属性。在某些情况下，同一类型对象存在多个实例，所以可能还需要指定对象名称。例如，应用程序中可能有多个任务。每个任务都有各自的可按照任务名称访问的 TASK 对象。

多播 模块可用于在网络上发送数据的机制，可由多个听众同时接收。介绍 ControlLogix I/O 线支持多个控制器同时从同一 I/O 模块接收输入数据的功能。

多所有者 一种配置设置，其中多个控制器具有完全相同的配置信息以同时拥有同一输入模块。

二进制 以 2 为基数显示和输入的整数值 （每个数位表示一位）。以 2# 为前缀。填充至布尔值或整数值的长度 （1、8、16 或 32 位）。显示时，每组四个数位由下划线分隔以保持可读性。请参见**十进制**、**十六进制**、**八进制**。

范围 定义可以访问一组特定标记的区域。创建标记时，您将其分配 （指定范围）作为控制器标记或特定程序的程序标记，如下所示。



可以使用多个具有相同名称的标记：

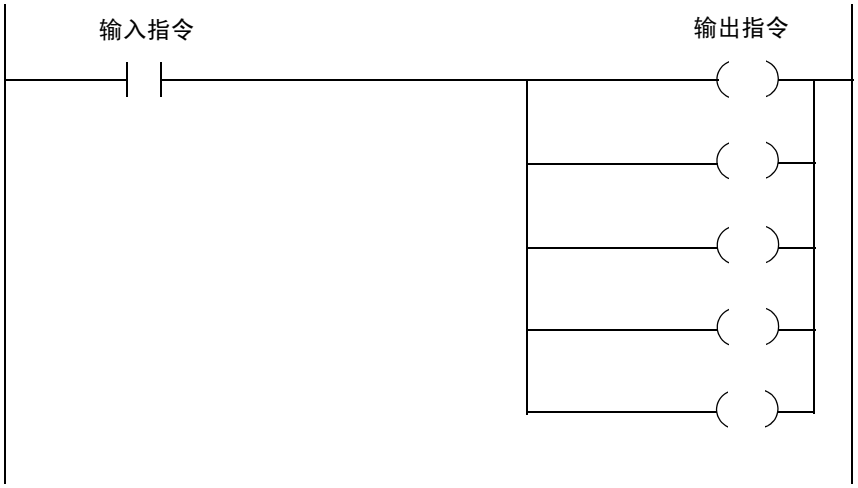
- 每个标记必须具有不同范围。例如，其中一个标记可以是控制器标记，而其他标记可以是其他程序的程序标记。或者，每个标记可以是不同程序的程序标记。
- 在程序中，如果程序的程序标记与控制器标记名称相同，则不能引用控制器标记。

请参见*控制器范围*、**程序范围**。

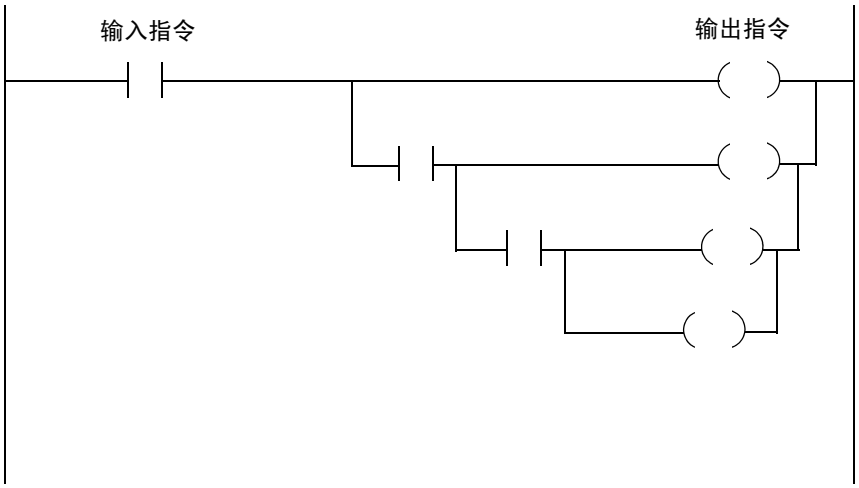
非缓存连接 利用 MSG 指令，非缓存连接指示控制器在完成 MSG 指令时关闭连接：清除连接使其可供其他控制器使用。请参见**连接**、**缓存连接**。

非易失性内存 在控制器没有电源或电池时保存其内容的控制器内存。请参见**加载**、**存储**。

分支 可以输入的并行分支层数没有限制。下图显示具有五层的并行分支。主层是第一个分支层，后面是其他四个分支。



您可以嵌套最多 6 层分支。下图显示一个嵌套分支。底部输出指令在一个三层深的嵌套分支上。



浮点 以浮点格式显示和输入的实数值。根据数字的大小，小数点左边的位数不同。请参见***样式***。

故障模式 控制器生成主故障，不能清除故障，已关闭。请参见***主故障***。

后扫描 控制器的一项功能，将在禁用程序前检查程序内的逻辑以重置指令和数据。

缓冲 根据配置 MSG 指令的方式，可以使用连接发送或接收数据。

消息类型：	通信方法：	使用连接：
CIP 数据表读写	→	3
PLC2、PLC3、PLC5 或 SLC （所有类型）	CIP 使用源 ID 的 CIP	
	DH+	3
CIP Generic	→	自行选择 (1)
块传输读写	→	✓

(1) 您可以连接 CIP generic 消息。但对于大部分应用程序，我们建议保留 CIP generic 消息不连接。

如果 MSG 指令使用连接，您可以选择在消息完成传输时使连接打开（缓存）或关闭连接。

如果您：	则：
缓存连接	MSG 指令完成后连接保持打开。这将优化执行时间。每次消息执行时打开连接将增加执行时间。
不缓存连接	MSG 指令完成后连接关闭。这将释放连接用于其他用途。

如果多个消息到达同一设备，则消息可能共享一个连接。

如果 MSG 指令到达：	并且它们：	则：
不同设备	→	每个 MSG 指令使用 1 个连接。
同一设备	同时启用	每个 MSG 指令使用 1 个连接。
	不同时启用	MSG 指令共享连接。（即，共同计数为 1 个连接。）

请参见**连接**、**非缓存连接**。

机架优化连接 对于数字 I/O 模块，可以选择机架优化通信。 机架优化连接加强控制器和机架 （或 DIN 轨道）中所有 I/O 模块间的连接使用。不是每个 I/O 模块都有单独的直接连接，而是整个机架 （或 DIN 轨道）有一个连接。请参见**直接连接**。

加载 将项目从非易失性内存复制到控制器的用户内存 (RAM)。这将覆盖控制器上的任何当前项目。请参见**非易失性内存**、**存储。**

兼容模块 电子密钥保护模式，该模式要求供应商、目录号以及物理模块和软件中配置的模块的主版本属性匹配，以建立与模块的连接。请参见**禁用密钥**、**精确匹配**。

监视 指定触发主故障前任务可以运行的时间。

- 每个任务都有一个监视计时器，用于监视任务的执行。
- 监视时间在 1 ms 至 2,000,000 ms （2000 秒）。默认值为 500 ms。
- 任务启动时监视计时器开始计时，任务中的所有程序都执行后计时器停止。
- 如果任务耗时超过监视时间，则发生主故障：（时间包括其他任务引起的中断。）
- 如果任务执行时再次触发 （定期任务重叠），也将发生监视超时故障 （主故障）。如果较高优先级的任务中断较低优先级的任务，从而推迟较低优先级任务完成，可发生此情况。
- 可以使用控制器故障处理程序清除监视故障。如果同一逻辑扫描期间同一监视故障再次发生，则控制器进入故障模式，无论控制器故障处理程序是否清除监视故障。

注意

如果监视计时器到达可配置的预设值，则发生主故障。根据控制器故障处理程序，控制器可能关闭。

更改任务的监视时间：

1. 打开 RSLogix 5000 项目。
2. 在控制器组织器中，右击 **name_of_task** 并选择 Properties （属性）。
3. 单击 Configuration （配置）选项卡。
4. 在 Watchdog （监视）文本框中，键入监视时间。
5. 单击 OK （确定）。

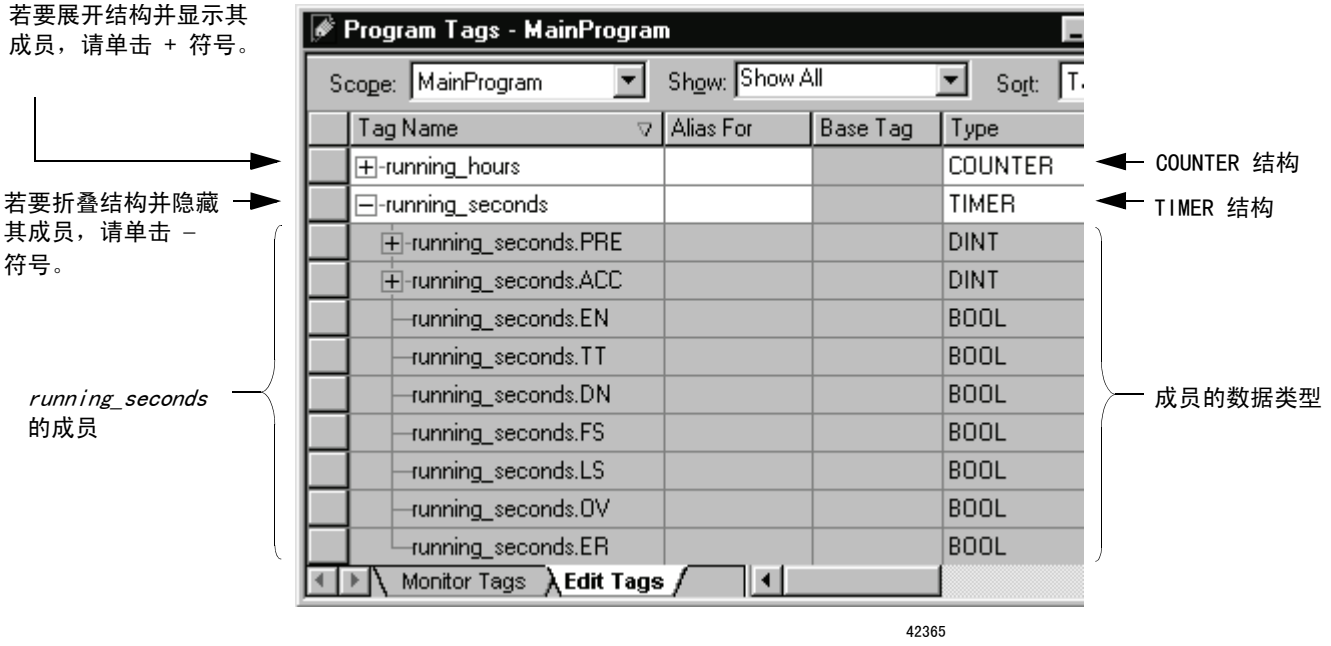
接口模块 （IFM） 预连线 I/O 现场布线臂。

结构 一些数据类型是结构。

- 一个结构存储一组数据，每个数据可以是不同数据类型。
- 结构中，每个数据类型称为一个**成员**。
- 和标记类似，成员具有名称和数据类型。

- 您使用各个标记的任意组合和其他结构创建您自己的结构，称为**用户定义的数据类型**。
- 若要将数据复制到结构，请使用 COP 指令。请参见 *Logix5000 控制器基本指令集参考手册*，出版物 1756-RM003。

COUNTER 和 TIMER 数据类型是常用的结构的示例。



请参见***成员***、**用户定义的数据类型**。

仅侦听连接 其他控制器在其中拥有 / 提供 I/O 模块的配置数据的 I/O 连接。使用仅侦听连接的控制器不写入配置数据，仅在所有者控制器活动控制 I/O 模块时维持与 I/O 模块的连接。请参见**所有者控制器**。

禁用密钥 一种电子密钥保护模式，不需要物理模块和软件中配置的模块的属性匹配，并仍建立与模块的连接。请参见**兼容模块**、**精确匹配**。

经过的时间 执行单个任务内配置的所有操作所需的总时间。

- 如果控制器配置为运行多个任务，则经过的时间包括执行其他操作的其他任务使用 / 共享的任何时间。
- 联机时，可以使用 ***Task Properties （任务属性）*** 对话框查看当前任务的最大扫描时间和上次扫描时间，以 ms 为单位。这些值是经过的时间，包括等待更高优先级任务所花的时间。

请参见**执行时间**。

精确匹配 一种电子密钥保护模式，要求物理模块和软件中配置的模块的所有属性 （供应商、目录号、主版本和次版本）一致，以建立到模块的连接。

控制器范围 可在控制器内任意位置访问的数据。控制器包含可由任何程序中的例程和 alias 标记以及控制器范围内的其他假名引用的一组标记。请参见*程序范围*。

控制器故障处理程序 控制器故障处理程序是以下情况下执行的可选任务：

- 主故障不是指令执行故障
- 程序故障例程：
  - 不能清除主故障
  - 出错
  - 不存在

只能为控制器故障处理程序创建一个程序。创建该程序后，必须配置一个例程为主例程。

- 控制器故障程序***不*** 执行故障例程。
- 如果为控制器故障程序指定故障例程，控制器将不再执行该例程。
- 您可以创建其他例程并从主例程调用它们。

控制与信息协议 Allen-Bradley 的控制设备 Logix5000 系列使用的消息协议。ControlNet 网络上使用的本机通信协议。

立即值 实际的 32 位有符号实数或整数值。不是存储值的标记。

例程 以单个编程语言编写的一组逻辑指令，例如梯级程序。

- 例程为控制器中的项目提供可执行代码 （类似 PLC 或 SLC 控制器中的程序文件）。
- 每个程序具有一个主例程：
  - 当控制器触发相关任务并执行关联程序时，首先执行主例程。
  - 若要调用程序中的其他例程，请在主例程中输入 JSR 指令。
- 还可以执行可选程序故障例程。
  - 如果关联程序中的例程产生主故障，则控制器执行程序故障例程。

请参见*程序*、**任务**。

连接 两个设备间的通信链接，例如控制器和 I/O 模块、PanelView 终端或其他控制器之间。

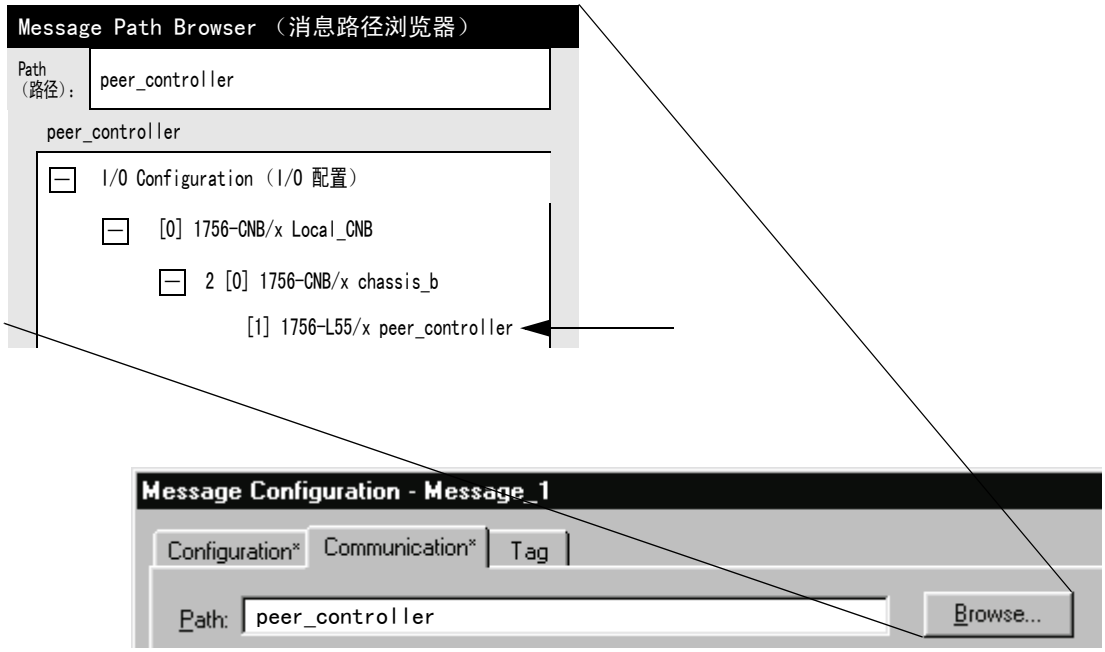
- 连接是在设备间提供比不连接的消息更可靠的通信的资源分配。
- 单个控制器的连接数有限。
- 您可以通过配置控制器与系统中其他设备通信来间接决定控制器使用的连接数。

联机 查看和编辑控制器中的项目。请参见*脱机*。

流程图 流程图（SFC）类似于过程图。它使用步骤和转变来控制机器或过程。

请参见**操作**、**步骤**、**转变**。

路径 路径说明消息到达目标所采取的路线。如果控制器的 I/O 配置包含目标设备，请使用 Browse （浏览）按钮选择设备。这将自动定义路径。



如果 I/O 配置*不* 包含目标设备，则使用下面的格式键入到目标的路径：

port,address,port,address

位置：	表示：	为：
port	任何 1756 控制器或模块的底板	1
	Logix5000 控制器的 DF1 端口	2
	1756-CNB 模块的 ControlNet 端口	
	1756-ENBx 或 -ENET 模块的以太网端口	
	1756-DHRIO 模块通过通道 A 的 DH+ 端口	
	1756-DHRIO 模块通过通道 B 的 DH+ 端口	3
address	ControlLogix 底板	插槽号
	DF1 网络	静态地址 (0-254)
	ControlNet 网络	节点号 （1-99 十进制）
	DH+ 网络	8# 后接节点号 （1-77 八进制） 例如，要指定八进制节点地址 37，键入 8#37。
	EtherNet/IP 网络	可以使用以下任意格式指定 EtherNet/IP 网络上的模块： IP 地址 （例如 130.130.130.5） IP 地址：端口 （例如 130.130.130.5:24） DNS 名称 （例如 tanks） DNS 名称：端口 （例如 tanks:24）

请参见**连接**。

名称 名称标识控制器、任务、程序、标记、模块等。名称遵循 IEC-1131-3 标识符规则并且：

- 必须以字母字符 （A-Z 或 a-z）或下划线 (_) 开始
- 只能包含字母字符、数字字符和下划线
- 最多可有 40 个字符
- 不能连续出现下划线 (_) 或以下划线结尾
- *不* 区分大小写
- 下载到控制器

内存 构建在控制器中用于保存程序和数据的电子存储介质。

请求数据包间隔 (RPI) 在网络上通信时，这是后续生成输入数据间的最大时间。

- 通常，间隔配置以微秒为单位。
- 实际产生数据限制在小于所选 RPI 的网络更新时间的最大倍数。
- 使用 ControlNet 网络更新时间 (NUT) 的偶数积。

  例如，NUT 为 5 ms，则速率类型为 5、10、20、40 ms 等。

请参见***网络更新时间 (NUT)***。

任务 执行程序的规划机制。

- 默认每个新项目文件包含一个预定义的持续任务。
- 根据需要配置额外定期任务。
- 任务为根据特定标准执行的一组一个或多个程序提供计划和优先级信息。
- 任务触发 （激活）后，所有分配 （规划）给该任务的程序按照其在控制器组织器中显示的顺序执行。
- 一次只能为一个任务分配一个程序。

请参见***持续任务***、**定期任务**。

扫描时间 请参见***经过的时间***、**执行时间。**

上载 将控制器的内容传输到工作站上的项目文件中的过程。

如果没有控制器的项目文件，则可以从控制器上载并创建项目文件。但是，控制器不能访问项目文件中存储的所有内容。如果从控制器上载，新项目文件将不包含：

- 梯级注释
- 标记、任务、程序、例程、模块或用户定义的结构的说明
- 假名链 （指向其他假名的假名）

  假名链不完全从控制器重建。如果数据项有多个可用名称，固件和软件将选择一个不影响在原始项目中指定假名的最合适的假名。

请参见**下载**。

生成标记 控制器提供给其他控制器使用的标记。生成标记始终在控制器范围内。请参见***使用标记***。

十进制 以 10 为基数显示和输入的整数值。没有前缀。不扩展到整数长度。请参见**二进制**、**十六进制**、**八进制**。

十六进制 以基数 16 显示和输入的整数值 （每个数字表示 4 位）。以 16# 为前缀。填充至布尔值或整数值的长度 （1、8、16 或 32 位）。显示时，每组四个数位由下划线分隔以保持可读性。 请参见**二进制**、**十进制**、**八进制**。

时间戳 一个 ControlLogix 过程，记录输入数据的更改以及更改发生时间的相对时间参考。

使用标记 接收生成标记在 ControlNet 网络或 ControlLogix 底板上广播的数据的标记。使用标记必须是：

- 控制器范围
- 和远程标记 （生成标记）数据类型相同 （包括任何数组维度）

请参见***生成标记***。

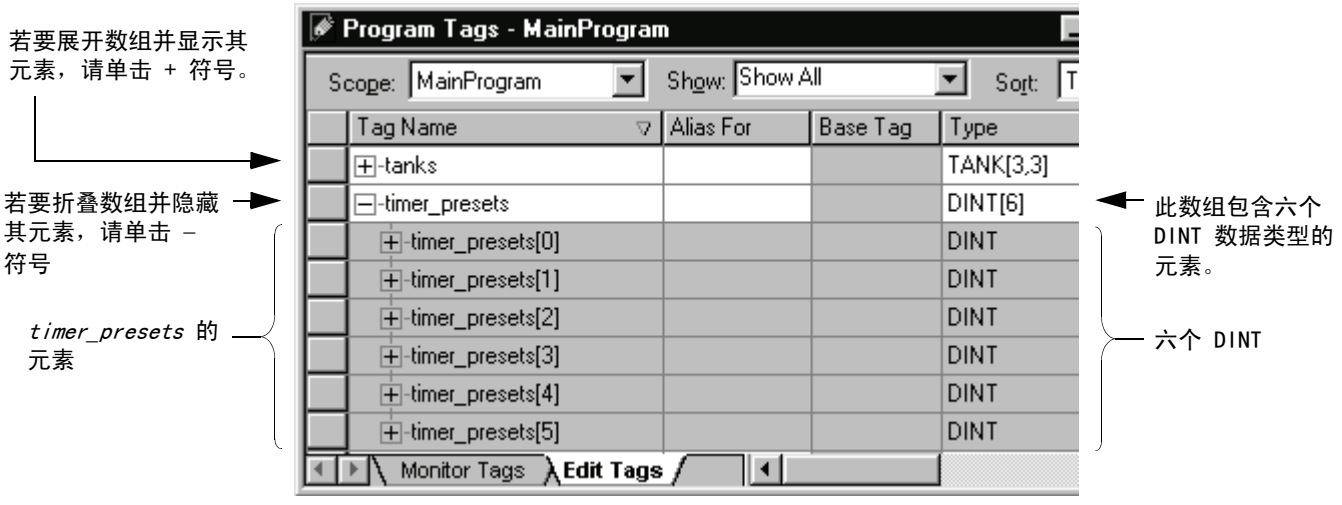
数据类型 创建数据类型的标记时将分配的内存大小和布局的定义。

数值表达式 以结构化文本表示的计算整数值或浮点值的表达式。

- 数值表达式使用算术运算符、算术函数和按位运算符。例如 `tag1+5`。
- 您常在 bool 表达式中嵌套数值表达式。例如 `(tag1+5)>65`。

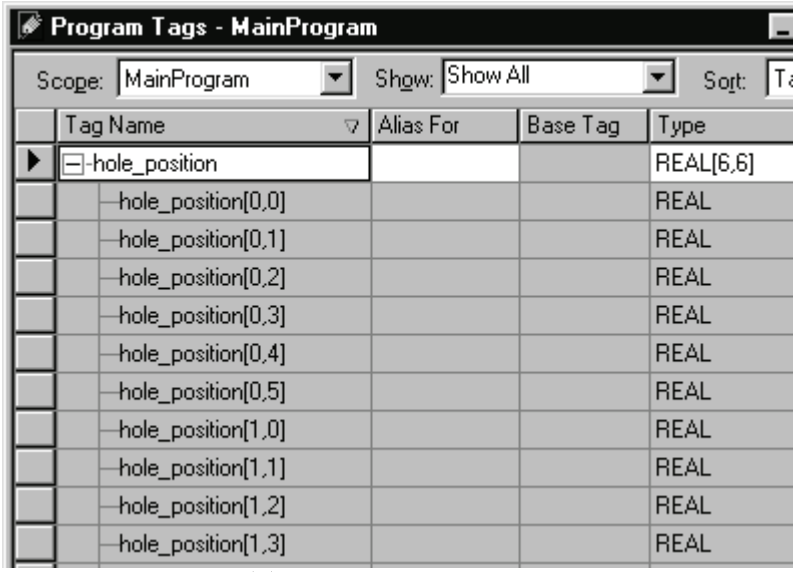
数组 数组让您将数据 （具有相同数据类型）分组在一个公共名称下。

- 数组类似于文件。
- 下标标识数组中的各个**元素**。
- 下标从 0 开始，至元素数减一的位置 （从零开始）。



- 数组标记占据控制器中的一个连续内存块，每个元素顺序排列。
- 您可以使用数组和定序器指令操作或索引数组中的元素。
- 数组可以具有多达三个维度。这使您可以使用一个、两个或三个下标 （坐标）灵活标识元素。

• 在二维或三维数组中，最右的维度在内存中首先递增。

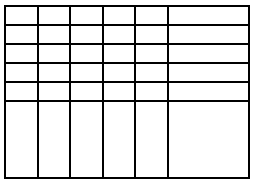
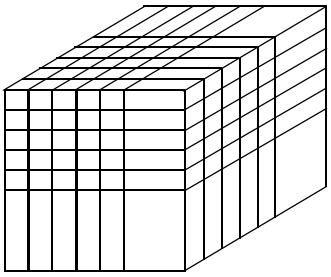


此数组包含一个二维元素网格，6 x 6 个元素。

最右的维度递增至其最大值后重新开始计数。

当最右的维度重新开始计数时，该维度左边的维度递增一。

- 数组元素总数是每个维度大小的乘积，如以下示例所示：

此数组：	存储数据：	例如：				
一个维度		标记名称：	类型	维度 0	维度 1	维度 2
		one_d_array	DINT[7]	7	--	--
		总元素数 = 7				
		有效下标范围 DINT[x] 其中 x=0–6				
两个维度		标记名称：	类型	维度 0	维度 1	维度 2
		two_d_array	DINT[4,5]	4	5	--
		总元素数 = 4 * 5 = 20				
		有效下标范围 DINT[x,y] 其中 x=0–3; y=0–4				
三个维度		标记名称：	类型	维度 0	维度 1	维度 2
		three_d_array	DINT[2,3,4]	2	3	4
		总元素数 = 2 * 3 * 4 = 24				
		有效下标范围 DINT[x,y,z] 其中 x=0–1; y=0–2, z=0–3				

- 脱机编程时您可以修改数组维度而不丢失标记数据。联机编程时您不能修改数组维度。

双向连接 数据双向流动的连接：从发生处到接收处，以及从接收处到发送处。请参见**连接**、**单向连接**。

说明 可用于进一步记录应用程序的可选文本。

- 可以使用任何可打印字符，包括回车符、制表符和空格。
- 说明不下载到控制器。它们保留在脱机项目文件中。
- 说明具有以下长度限制：
  - 对于标记，可以使用最多 120 个字符。
  - 对于其他对象 （任务、程序、模块等），可以使用最多 128 个字符。

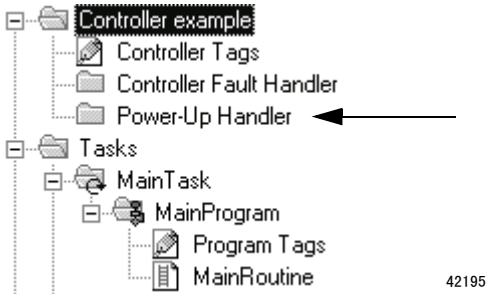
速率 对于定期任务，控制器执行任务的速率，范围在 1 ms 至 2,000,000 ms （2000 秒）。默认值为 10 ms。

所有者控制器 创建主配置和到模块的通信连接的控制器。所有者控制器写入配置数据并可以建立与模块的连接。请参见**仅侦听连接**。

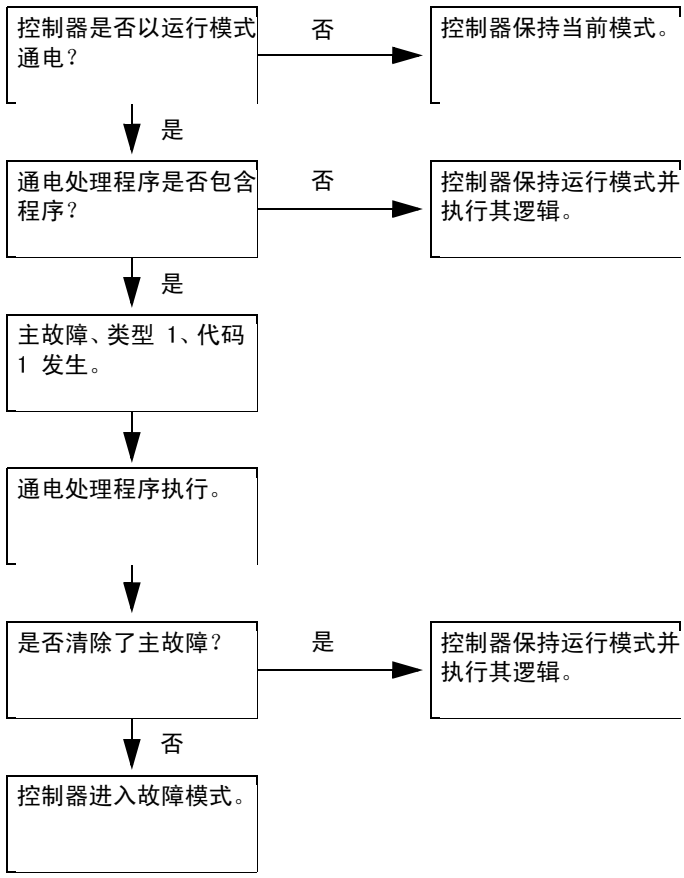
索引 用于指定数组中元素的引用。

通电插拔 (RIUP) 允许用户在机架通电时安装或取下模块的 ControlLogix 功能。

通电处理程序 控制器以运行模式通电时执行的一项可选任务。若要使用通电处理程序，必须创建一个通电程序并关联主例程。



通电处理程序执行方式如下：



通信格式 定义 I/O 模块与控制器通信的方式。选择通信格式定义：

- 编程软件中可用的配置选项卡
- 标记结构和配置方法

脱机 查看和编辑位于工作站硬盘上的项目。请参见***联机***。

网络更新时间 (NUT) 可在 ControlNet 网络上发送数据的重复时间间隔。网络更新时间范围在 2ms-100ms。

维度 数组大小的规格。数组可以具有多达三个维度。请参见***数组***。

位 二进制数位。最小内存单位。由数字 0 （清除）和 1 （设置）表示。

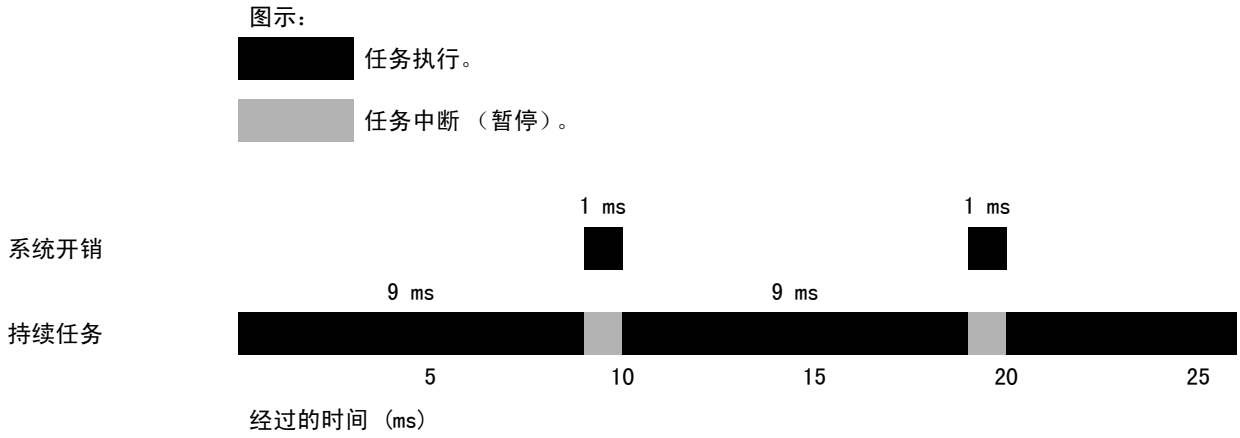
系统开销时间片 指定用于通信和后台功能 （系统开销）的控制器时间 （不包括定期任务的时间）的比例。

- 控制器一次执行系统开销功能最多 1 ms。
- 如果控制器在 1 ms 内完成开销功能，则继续持续任务。
- 通信和后台功能包括：
  - 与编程和 HMI 设备 （例如 RSLogix 5000 软件）的通信
  - 响应消息
  - 发送消息，包括块传输
  - 重新建立和监视 I/O 连接 （例如 RIUP 情况）；这***不*** 包括程序执行期间发送的正常 I/O 通信。
  - 通过 ControlLogix 底板从控制器的串行端口到其他 ControlLogix 设备的转发通信
- 如果通信完成速度不够快，请增加系统开销时间片。

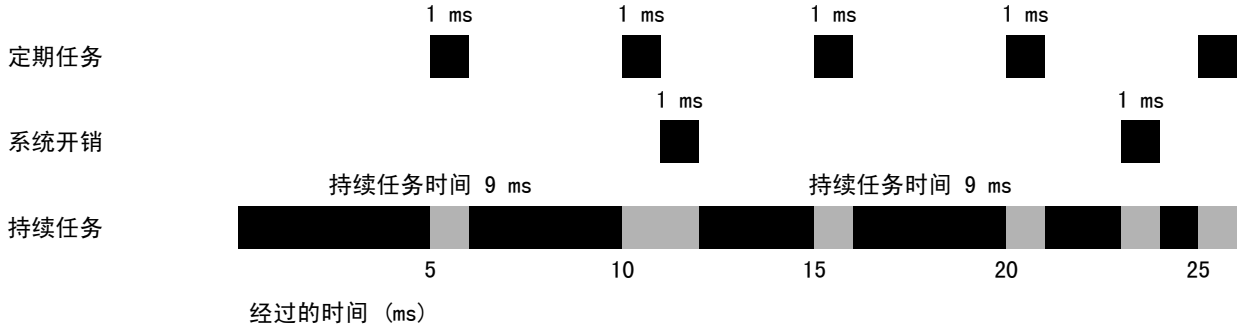
此表显示持续任务和系统开销功能间的比例：

此时间段：	持续任务运行：	然后开销进行最多：
10%	9 ms	1 ms
20%	4 ms	1 ms
33%	2 ms	1 ms
50%	1 ms	1 ms

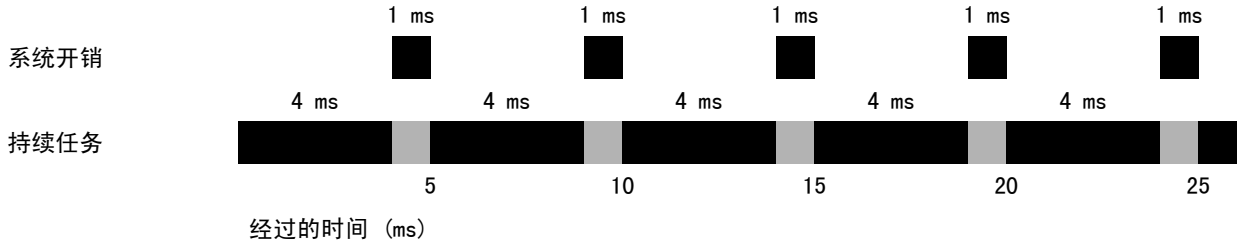
在默认时间片 10 % 时，系统开销每 9 ms （持续任务时间）中断一次持续任务。



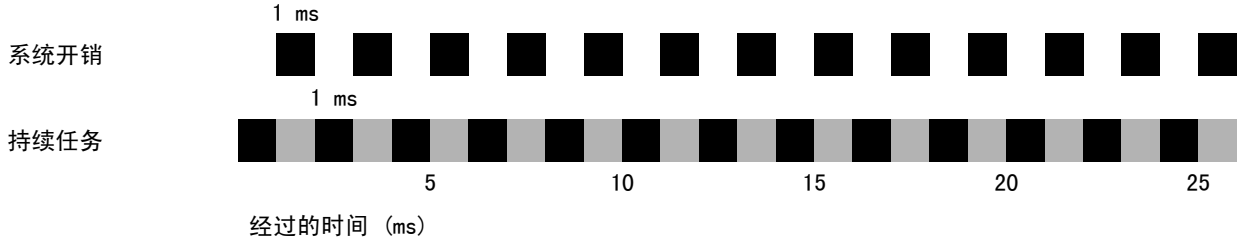
定期任务的中断增加系统开销的执行之间经过的时间 （时钟时间）。



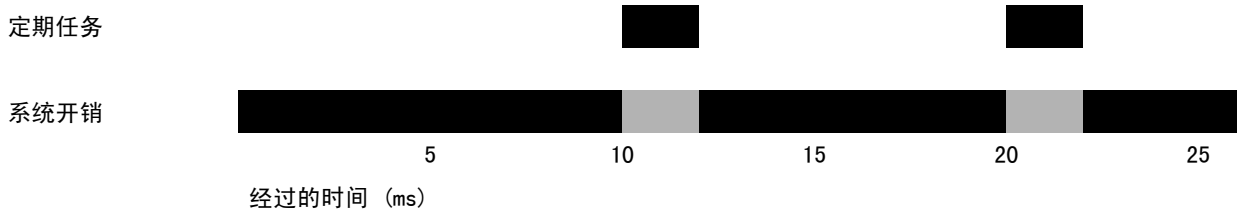
如果将时间片增至 20 %，系统开销每 4 ms （持续任务时间）中断一次持续任务。



如果将时间片增至 50 %，系统开销每 1 ms （持续任务时间）中断一次持续任务。



如果控制器只包含一个定期任务，系统开销时间片值不起作用。定期任务没有运行时系统开销都将运行。



更改系统开销时间片：

1. 打开 RSLogix 5000 项目。
2. 在控制器组织器中，右击控制器 *name_of_controller* 文件夹并选择 Properties （属性）。
3. 单击 Advanced （高级）选项卡。
4. 在 System Overhead Time Slice （系统开销时间片）文本框中，键入或选择开销时间的比例 (10 -90%)。
5. 单击 *OK*（确定）。

下载 将工作站上的项目内容传输到控制器的过程。请参见**上载**。

限定符 在流程图 (SFC) 的操作中，限定符定义操作何时开始和停止。请参见**操作**、**流程图**、**步骤**。

项目 位于工作站 （或服务器）上的存储控制器逻辑、配置、数据和文档的文件。

- 项目文件扩展名为 .ACD。
- 创建项目文件时，文件名称为控制器名称。
- 控制器名称与项目文件名称无关。如果将当前项目文件保存为其他名称，则控制器名称不更改。

- 如果控制器名称与项目文件名称不同，则 RSLogix 5000 软件的标题栏同时显示两个名称。

请参见**应用程序**。

协调系统时间 (CST) 表示自 CST 主控制器开始计数后微秒数的 64 位值。

- CST 值存储为 DINT[2] 数组，其中：
  - 第一个元素存储低 32 位
  - 第二个元素存储高 32 位
- 您可以使用 CST 时间戳比较数据采样间的相对时间。

样式 数值显示的格式。请参见 ***ASCII***、**二进制**、**十进制**、**指数**、**浮点**、**十六进制**、**八进制**。

异步 彼此独立发生并缺乏常规模式的操作。在 Logix5000 控制器中，I/O 值对以下逻辑执行异步更新：

- 任务内的程序直接从控制器范围内的内存访问输入和输出数据。
- 任何任务中的逻辑都可修改控制器范围内的数据。
- 数据与 I/O 值是异步的，可在任务执行过程中更改。
- 任务执行开始时引用的输入值在以后引用时可以不同。

注意

注意确保数据内存在任务执行过程中包含合适的值。您可以在扫描开始时复制或缓冲数据来为逻辑提供引用值。

应用程序 用于定义单个控制器运行的 I/O 配置的例程、程序、任务以及 I/O 配置的组合。请参见**项目**。

用户定义的数据类型 还可以创建您自己的**结构**，称为用户定义的数据类型 （通常也指用户定义的结构）。用户定义的数据类型将不同类型数据组合到单个命名实体中。

- 在用户定义的数据类型中，定义**成员**。
- 和标记类似，成员具有名称和数据类型。
- 可以包含数组和结构。
- 创建用户定义的数据类型后，可以创建使用该数据类型的一个或多个标记。
- 尽量减少以下数据类型的使用，因为它们通常增加逻辑的内存要求和执行时间：
  - **INT**
  - **SINT**

例如，一些系统值使用 SINT 或 INT 数据类型。如果创建用户定义的数据类型来存储这些值，则使用相应的 SINT 或 INT 数据类型。

- 如果包括表示 I/O 设备的成员，必须使用梯级逻辑在结构中的成员和相应 I/O 标记间复制数据。请参见第 1-8 页上的“缓存 I/O”。
- 使用 BOOL、SINT 或 INT 数据类型时，将使用相同数据类型的成员放置在序列中：

更高效

BOOL
BOOL
BOOL
DINT
DINT

更低效

BOOL
DINT
BOOL
DINT
BOOL

- 可以使用单维数组。
- 只能在脱机编程时创建、编辑和删除用户定义的数据类型。
- 如果修改用户定义的数据类型并更改其大小，则使用该数据类型的任何标记的现有值设置为零 (0)。
- 若要将数据复制到结构，请使用 COP 指令。请参见 *Logix5000 控制器基本指令集参考手册*，出版物 1756-RM003。

请参见***结构***。

优先级 指定同时触发两个任务时首先执行哪个任务。

- 具有较高优先级的任务首先执行。
- 优先级范围为 1-15，1 具有最高优先级。
- 较高优先级任务将中断任何较低优先级任务。
- 如果同时触发两个具有相同优先级的任务，控制器将每毫秒在两个任务间切换。

预定义结构 存储特定指令相关信息的结构数据类型，例如计时器指令的 TIMER 结构。预定义结构始终可用，与系统硬件配置无关。请参见**产品定义结构**。

预扫描 预扫描是转换为运行模式期间的中间扫描。

- 当您从程序模式更改为运行模式时控制器执行预扫描。
- 预扫描检查所有程序和指令并根据结果初始化数据。
- 一些指令在预扫描期间和正常扫描期间执行方式不同。

元素 作为更大数据单元的子单元的可寻址数据单元。数组的单个单元。

- 可以按其下标指定数组中的元素：

对于此数组：	指定：
一个维度	*array_name[subscript_0]*
两个维度	*array_name[subscript_0, subscript_1]*
三个维度	*array_name[subscript_0, subscript_1, subscript_2]*

请参见**数组**。

源密钥 限制哪些人可以查看例程的机制。

- 可以对一个或多个例程指定源密钥。
- 源密钥和 RSLogix 5000 组件 （如例程、标记和模块）遵循相同的命名规则。
- 若要为例程指定密钥 （保护例程），请使用 RSLogix 5000 软件。（必须首先启用该工具）。
- 源密钥文件 （sk.dat） 存储源密钥。源密钥文件与 RSLogix 5000 项目文件 （.acd） 分开。
- 若要查看受源密钥保护的例程，必须具有源密钥。
- 没有源密钥无法打开例程。RSLogix 5000 软件显示 Source Not Available （源不可用）。
- 无论源密钥是否可用，始终可以下载项目并执行所有例程。

请参见**名称。**

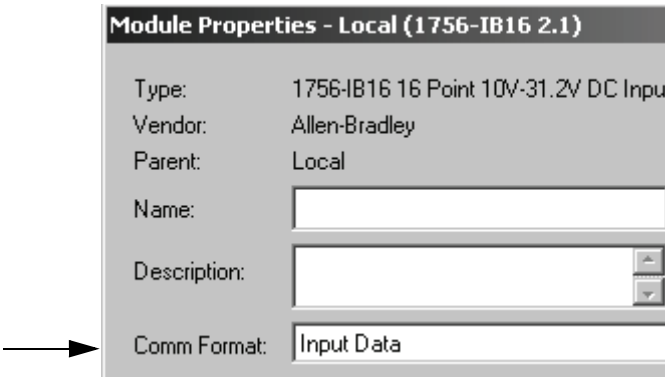
执行时间 执行单个程序所需的总时间。

- 执行时间仅包括单个程序使用的时间，不包括执行其他操作的其他任务中的程序共享 / 使用的时间。
- 联机时，使用 Program Properties （程序属性）对话框查看当前程序的最大扫描时间和上次扫描时间，以 ms 为单位。这些值是程序的执行时间，不包括等待其他程序或更高优先级任务所花的时间。

请参见***经过的时间***。

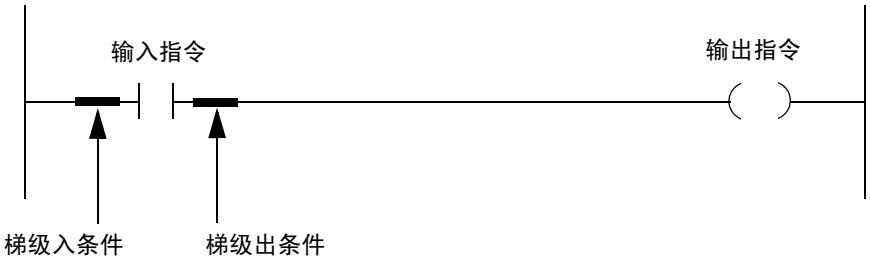
直接连接 直接连接是控制器和 I/O 模块间的实时数据传输链接。控制器维护并监视与 I/O 模块的连接。任何连接中断，如模块故障或通电时取下模块，都将在与模块相关的数据区域设置故障位。

直接连接是不使用机架优化通用格式的连接。



请参见**机架优化连接**。

指令 控制器根据指令前的梯级条件 （梯级入条件）计算梯级指令。



只有输入指令影响梯级上的后续指令的梯级入条件。

- 如果输入指令的梯级入条件为 true，则控制器计算该指令并设置梯级出条件以匹配计算结果。
  - 如果指令计算结果为 true，则梯级出条件为 true。
  - 如果指令计算结果为 false，则梯级出条件为 false。
- 输出指令不更改梯级出条件。
  - 如果输出指令的梯级入条件为 true，则梯级出条件设置为 true。
  - 如果输出指令的梯级入条件为 false，则梯级出条件设置为 false。

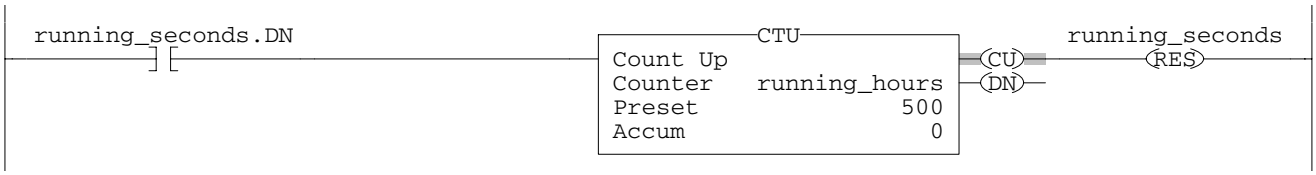
在 Logix5000 控制器中，每个逻辑梯级可以输入多个输出指令。可以输入输出指令：

- 按梯级上的顺序 （顺序）
- 在输入指令之间，只要梯级上的上个指令是输出指令

此示例在梯级上使用多个输出。

示例 将多个输出放置在梯级上

当 *running_seconds.DN* 为 on 时，*running_hours* 递增 1，*running_seconds* 重置。



当 *machine_on* 为 on 时，*drill_1_on* 变为 on。当 *machine_on* 和 *drill[1].part_advance* 都为 on 时，*conveyor_on* 变为 on。

42362

指数 以科学记数法或指数格式显示和输入的实数值。数字通常显示方式：小数点左一个数位，之后是小数部分，紧接是指数。请参见***样式***。

重叠 当任务仍在执行上一个触发器时触发任务 （定期或条件性）的情况。

主 (CST) 在单个机架内，必须只有一个控制器指定为协调系统时间 (CST) 主。机架中的其他模块必须与 CST 主同步它们的 CST 值。

主版本 1756 模块系列具有主版本和次版本指示器。出现对模块的功能更改时都会更新主版本。请参见**电子密钥**、**次版本**。

主故障 如果不清除该情况，将足以使控制器关闭的故障情况。发生主故障时，控制器将：

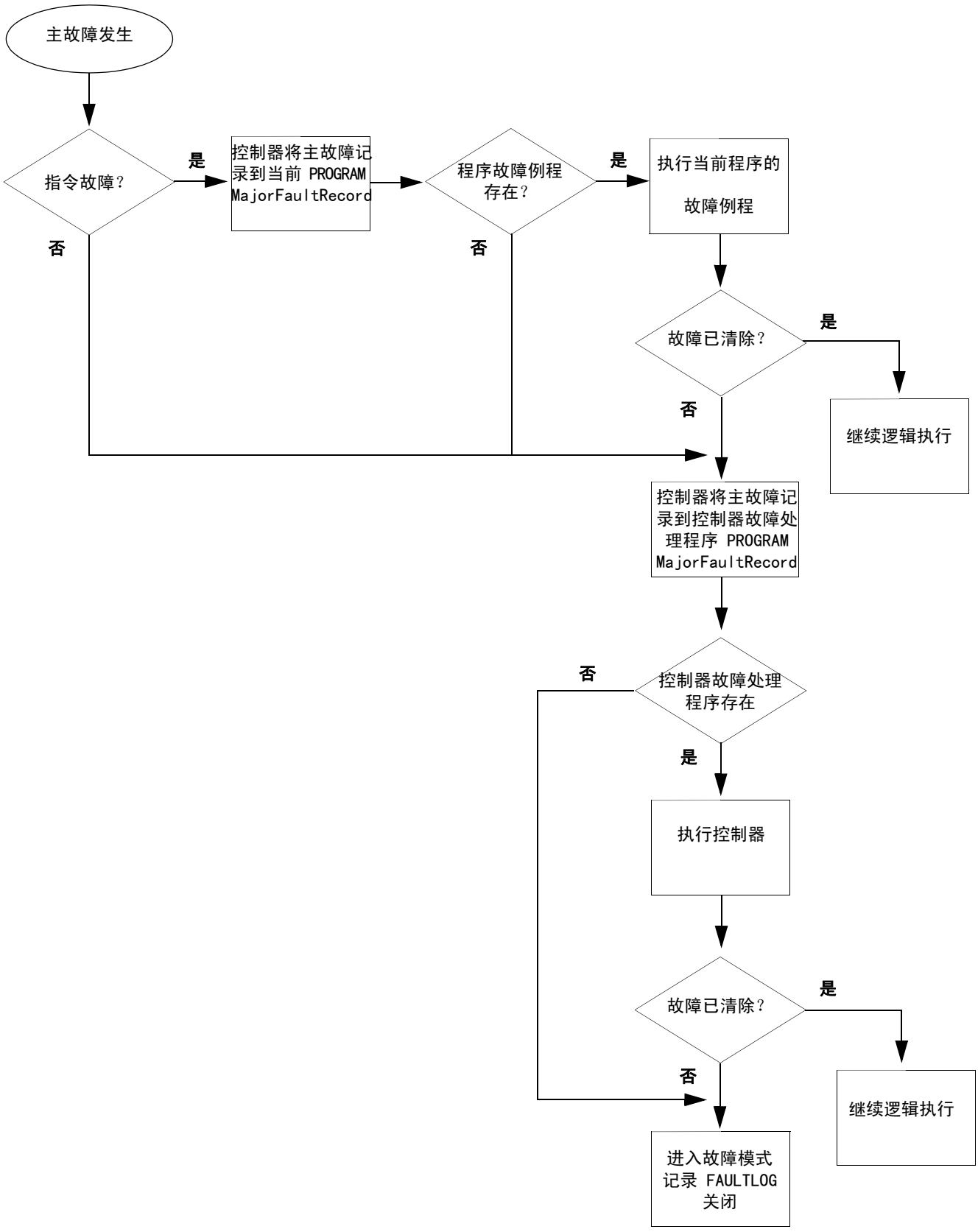
1. 设置主故障位
2. 运行用户提供的故障逻辑，如果有
3. 如果用户提供的故障逻辑不能清除故障，则控制进入故障模式
4. 根据程序模式中的输出状态设置输出
5. OK LED 闪红灯

控制器支持两种处理主故障的级别：

- 程序故障例程：
  - 每个程序都有自己的故障例程。
  - 发生指令故障时控制器执行程序的故障例程。
  - 如果程序的故障例程不能清除故障或程序故障例程不存在，控制器将继续执行控制器故障处理程序 （如果已定义）。
- 控制器故障处理程序：
  - 如果控制器故障处理程序不存在或无法清除主故障，则控制器进入故障模式并关闭。此时，FAULTLOG 更新。（请参见下页。）
  - 所有非指令故障 （I/O、任务监视等）直接执行控制器故障处理程序。（不调用程序故障例程。）

没有清除的故障以及最多两个尚未清除的其他故障记录在控制器故障日志中。

请参见**故障状态**、**次故障**。

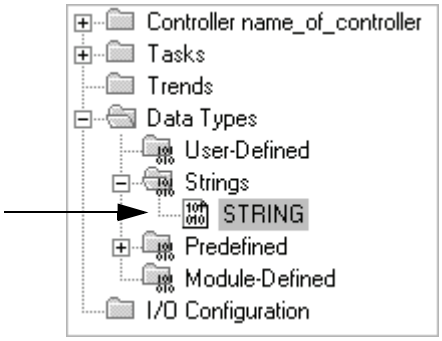

主故障发生
指令故障？
是
否
控制器将主故障记录到当前 PROGRAM MajorFaultRecord
程序故障例程存在？
是
否
执行当前程序的故障例程
故障已清除？
是
否
继续逻辑执行
控制器将主故障记录到控制器故障处理程序 PROGRAM MajorFaultRecord
控制器故障处理程序存在
否
是
执行控制器
故障已清除？
是
否
继续逻辑执行
进入故障模式
记录 FAULTLOG
关闭

主例程 程序执行时首先执行的例程。使用主例程调用 （执行）其他例程（子例程）。

转变 在流程图 (SFC) 中，转变是决定何时进入下一步的 true 或 false 条件。

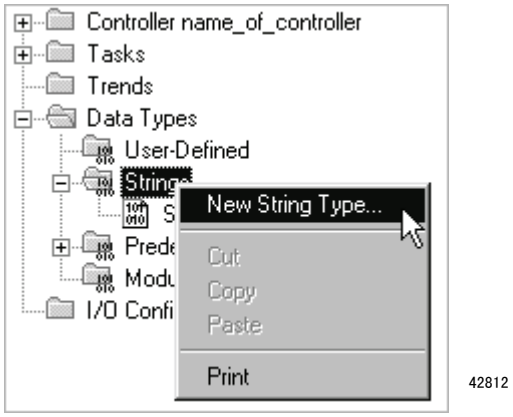
状态更改 (COS) I/O 模块上的一个点或一组点状态的任何更改。

字符串 一组存储 ASCII 字符的数据类型。



可以使用默认 STRING 数据类型。它存储最多 82 个字符。

或



可以创建新字符串数据类型来存储定义数量的字符。

每个字符串数据类型包含下列成员：

名称：	数据类型：	说明：	注释：
LEN	DINT	字符串中的字符数	当您进行下面操作时，LEN 自动更新为新的字符计数： • 使用 String Browser （字符串浏览器）对话框输入字符 • 使用读取、转换或操作字符串的指令 LEN 显示当前字符串的长度。DATA 成员可能包含 LEN 计数中不包括的其他旧的字符。
DATA	SINT 数组	字符串的 ASCII 字符	• 若要访问字符串的字符，请寻址标记的名称。 例如，要访问 string_1 标记的字符，请输入 string_1。 • DATA 数组的每个元素包含一个字符。 • 可以创建存储更多或更少字符的新字符串数据类型。

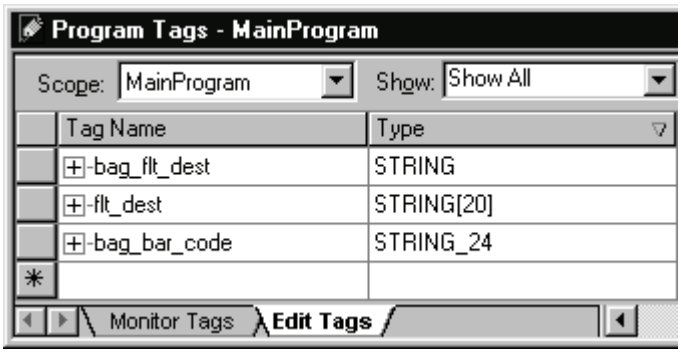
新字符串数据类型在以下场合有用：

- 如果您有大量具有小于 82 字符的固定大小的字符串，可以通过创建新字符串数据类型来节约内存。
- 如果您必须处理字符数超过 82 的字符串，可以创建新字符串数据类型来适合所需的字符串数。

重要 创建新字符串数据时请小心。如果以后决定更改该字符串数据类型的大小，可能丢失当前使用该数据类型的任何标记中的数据。

如果您：	则：
缩小字符串数据类型	• 数据被截断。 • LEN 不改变。
放大字符串数据类型	数据和 LEN 重置为零。

此示例显示 STRING 数据类型和一个新字符串数据类型。



此标记使用默认 STRING 数据类型。

此标记为默认 STRING 数据类型的 20 元素数组。

此标记使用新字符串数据类型。

- 用户命名字符串数据类型为 STRING_24。
- 新字符串数据类型仅存储 24 个字符。

字节 8 位组成的内存单元。

最优数据类型 Logix5000 指令实际使用的数据类型 （通常为 DINT 和 REAL 数据类型）。

- 在指令集参考手册中，**粗体**数据类型表示最优数据类型。
- 如果指令的所有操作数都使用下面内容，指令执行速度更快，所需内存更少：
  - 相同数据类型
  - 最优数据类型
- 如果您混合数据类型并使用不是最优数据类型的标记，则控制器根据以下规则转换数据

– 是否任何操作数为 REAL 值？

如果：	则输入操作数 （例如源、表达式中的标记、限制）转换为：
是	REAL
否	DINT

– 指令执行后，结果 （DINT 或 REAL 值）转换为目标数据类型，如果必要。

- 因为转换数据需要额外时间和内存，所以可通过下面的方式提高程序效率：

  – 在指令中使用相同数据类型
  – 尽量减少 SINT 或 INT 数据类型的使用

  换句话说，在指令中全部使用 DINT 标记或 REAL 标记以及立即值。

• 此表汇总控制器在数据类型间转换数据的方式：

<table>
<tr><th>转换：</th><th>结果：</th></tr>
<tr><td>较大的整数至较小的整数</td><td>控制器截断较大整数的高部分并生成溢出。

例如：
<table><tr><th></th><th>十进制</th><th>二进制</th></tr><tr><td>DINT</td><td>65,665</td><td>0000_0000_0000_0001_0000_0000_1000_0001</td></tr><tr><td>INT</td><td>129</td><td>0000_0000_1000_0001</td></tr><tr><td>SINT</td><td>-127</td><td>1000_0001</td></tr></table></td></tr>
<tr><td>SINT 或 INT 至 REAL</td><td>不丢失数据精确度</td></tr>
<tr><td>DINT 至 REAL</td><td>可能丢失数据精确度。两种数据类型都以 32 位存储数据，但 REAL 类型使用部分 32 位存储指数值。如果丢失精确度，控制器从 DINT 最不重要的部分补偿。</td></tr>
<tr><td>REAL 至整数</td><td>控制器四舍五入小数部分并截断非小数部分的高位。如果丢失数据，控制器将设置溢出状态标志。

数字四舍五入如下：
• 不是 x.5 的数字舍入为最接近的数字。
• X.5 舍入为最接近的偶数。

例如：
<table><tr><th>REAL （源）</th><th>DINT （结果）</th></tr><tr><td>-2.5</td><td>-2</td></tr><tr><td>-1.6</td><td>-2</td></tr><tr><td>-1.5</td><td>-2</td></tr><tr><td>-1.4</td><td>-1</td></tr><tr><td>1.4</td><td>1</td></tr><tr><td>1.5</td><td>2</td></tr><tr><td>1.6</td><td>2</td></tr><tr><td>2.5</td><td>2</td></tr></table></td></tr>
</table>

注释：

ASCII 字符代码

字符	十进制	十六进制
[ctrl-@] NUL	0	$00
[ctrl-A] SOH	1	$01
[ctrl-B] STX	2	$02
[ctrl-C] ETX	3	$03
[ctrl-D] EOT	4	$04
[ctrl-E] ENQ	5	$05
[ctrl-F] ACK	6	$06
[ctrl-G] BEL	7	$07
[ctrl-H] BS	8	$08
[ctrl-I] HT	9	$09
[ctrl-J] LF	10	$l ($0A)
[ctrl-K] VT	11	$0B
[ctrl-L] FF	12	$0C
[ctrl-M] CR	13	$r ($0D)
[ctrl-N] SO	14	$0E
[ctrl-O] SI	15	$0F
[ctrl-P] DLE	16	$10
[ctrl-Q] DC1	17	$11
[ctrl-R] DC2	18	$12
[ctrl-S] DC3	19	$13
[ctrl-T] DC4	20	$14
[ctrl-U] NAK	21	$15
[ctrl-V] SYN	22	$16
[ctrl-W] ETB	23	$17
[ctrl-X] CAN	24	$18
[ctrl-Y] EM	25	$19
[ctrl-Z] SUB	26	$1A
ctrl-[ESC	27	$1B
[ctrl-\] FS	28	$1C
ctrl-] GS	29	$1D
[ctrl-^] RS	30	$1E
[ctrl-_] US	31	$1F

字符	十进制	十六进制
SPACE	32	$20
!	33	$21
“	34	$22
#	35	$23
$	36	$24
%	37	$25
&	38	$26
‘	39	$27
(	40	$28
)	41	$29
*	42	$2A
+	43	$2B
,	44	$2C
-	45	$2D
.	46	$2E
/	47	$2F
0	48	$30
1	49	$31
2	50	$32
3	51	$33
4	52	$34
5	53	$35
6	54	$36
7	55	$37
8	56	$38
9	57	$39
:	58	$3A
;	59	$3B
<	60	$3C
=	61	$3D
>	62	$3E
?	63	$3F

字符	十进制	十六进制
@	64	$40
A	65	$41
B	66	$42
C	67	$43
D	68	$44
E	69	$45
F	70	$46
G	71	$47
H	72	$48
I	73	$49
J	74	$4A
K	75	$4B
L	76	$4C
M	77	$4D
N	78	$4E
O	79	$4F
P	80	$50
Q	81	$51
R	82	$52
S	83	$53
T	84	$54
U	85	$55
V	86	$56
W	87	$57
X	88	$58
Y	89	$59
Z	90	$5A
[	91	$5B
\	92	$5C
]	93	$5D
^	94	$5E
_	95	$5F

字符	十进制	十六进制
‘	96	$60
a	97	$61
B	98	$62
C	99	$63
d	100	$64
E	101	$65
F	102	$66
G	103	$67
H	104	$68
I	105	$69
J	106	$6A
K	107	$6B
L	108	$6C
m	109	$6D
n	110	$6E
O	111	$6F
P	112	$70
Q	113	$71
R	114	$72
s	115	$73
T	116	$74
U	117	$75
V	118	$76
W	119	$77
X	120	$78
Y	121	$79
Z	122	$7A
{	123	$7B
|	124	$7C
}	125	$7D
~	126	$7E
DEL	127	$7F

Rockwell Automation 支持

Rockwell Automation 在网站上提供可帮助您使用其产品的技术信息。您可以在 http://support.rockwellautomation.com 上找到技术手册、常见问题解答知识库、技术和应用程序说明、示例代码以及指向软件服务包的链接，另外还提供一项 MySupport 功能，您可以通过该功能执行自定义设置，从而最优利用这些工具。

为了提供有关安装、配置和故障排除的更高一级的电话支持，我们提供了 TechConnect Support 方案。有关更多信息，请联系当地分销商或 Rockwell Automation 代表，或者访问 http://support.rockwellautomation.com。

安装帮助

如果您在安装的 24 小时以内遇到硬件模块问题，请查看本手册中提供的信息。您也可以拨打入门帮助客户支持专线，以修复模块问题并使之运行：

美国	1.440.646.3223 星期一到星期五，早 8 点到晚 5 点 （美国东部时间）
美国境外	任何技术支持问题，请联系当地 Rockwell Automation 代表。

新产品满意退货

Rockwell 会对其所有产品进行测试，以保证其产品在从制造车间发出时能够充分运作。但是，如果您因产品不能使用而需要退货：

美国	联系分销商。您必须向分销商提供客户支持帐户编号 （参见上面的电话号码获取一个），以便完成退货程序。
美国境外	请联系当地 Rockwell Automation 代表，了解退货手续。

www.rockwellautomation.com

Corporate Headquarters
Rockwell Automation, 777 East Wisconsin Avenue, Suite 1400, Milwaukee, WI, 53202-5302 USA, Tel: (1) 414.212.5200, Fax: (1) 414.212.5201

Headquarters for Allen-Bradley Products, Rockwell Software Products and Global Manufacturing Solutions
Americas: Rockwell Automation, 1201 South Second Street, Milwaukee, WI 53204-2496 USA, Tel: (1) 414.382.2000, Fax: (1) 414.382.4444
Europe: Rockwell Automation SA/NV, Vorstlaan/Boulevard du Souverain 36-BP 3A/B, 1170 Brussels, Belgium, Tel: (32) 2 663 0600, Fax: (32) 2 663 0640
Asia Pacific: Rockwell Automation, 27/F Citicorp Centre, 18 Whitfield Road, Causeway Bay, Hong Kong, Tel: (852) 2887 4788, Fax: (852) 2508 1846

Headquarters for Dodge and Reliance Electric Products
Americas: Rockwell Automation, 6040 Ponders Court, Greenville, SC 29615-4617 USA, Tel: (1) 864.297.4800, Fax: (1) 864.281.2433
Europe: Rockwell Automation, Brühlstraße 22, D-74834 Elztal-Dallau, Germany, Tel: (49) 6261 9410, Fax: (49) 6261 17741
Asia Pacific: Rockwell Automation, 55 Newton Road, #11-01/02 Revenue House, Singapore 307987, Tel: (65) 351 6723, Fax: (65) 355 1733

出版物 1756-PM001H-ZH-P - 3 月 2004 PN 957974-87
以前的出版物 1756-PM001F-EN-P - 2003 年 6 月

Allen-Bradley

Logix5000™ 控制器常用过程

编程手册