工程机械
CODESYS 开发

三天入门 保姆级 速成教材

- PLC 开发
- HMI 开发
- 通信协议规划
- 联调和测试

PROFESSION ENGINEER

目录

前言

本教材内容根据编者在工程机械领域多年以来，使用不同品牌控制器和显示器做了一些项目之后的经验分享。主要面向想从事工程机械领域电液控制（基于codesys平台）的电气工程师，液压工程师，产品经理等。一个快速入门必备的基础知识汇总教材，其中包含有各种不同渠道汇总的资料，也有编者自己经验所得，如有侵权，请联系编者删除。

掌握本教程所包含的知识内容之后，基本可以达到工程机械电控系统软件开发的入门资格。可以独立进行程序的调试和维护，并对程序进行改动和优化。想要达到独立编程和开发的水准，仍需在此基础上找两个项目进行对比实操练手。

本教材所涉及内容，也是编制根据经验所得或者整理，如有纰漏之处，欢迎联系指正。

一、做一个项目你需要具备的 CODESYS 的基本知识

1.1、CODESYS 的基本概念之背景和产品介绍

1.1.1 CODESYS 的背景知识

CODESYS 软件供应商是德国 Smart software solution GmbH（3S）。

CODESYS 是可编程逻辑控制 PLC 的完整开发环境（CODESYS 是 Controlled Development System 的缩写）。

CODESYS 是一种功能强大的 PLC 软件编程工具，它支持 IEC61131-3 标准 IL 、ST、 FBD 、LD、 CFC、 SFC 六种 PLC 编程语言。

目前国内工程机械控制系统使用的控制器和显示器硬件，90%以上都是支持 codesys 编程的平台。

徐工、三一、中联、山河等等工程机械制造厂，应用的自研或者外购控制器国产品牌主要是：徐州威卡，长沙硕博，株洲嘉城，贵阳永清，苏州禾晟等等

活跃在一线的进口品牌包括：芬兰 EPEC、德国 TTC，ifm，IC 等等。

1.1.2 CODESYS 相关产品介绍

CODESYS 相关的产品和服务有很多，具体可以去 CODESYS 官网 www.codesys.com 去了解。

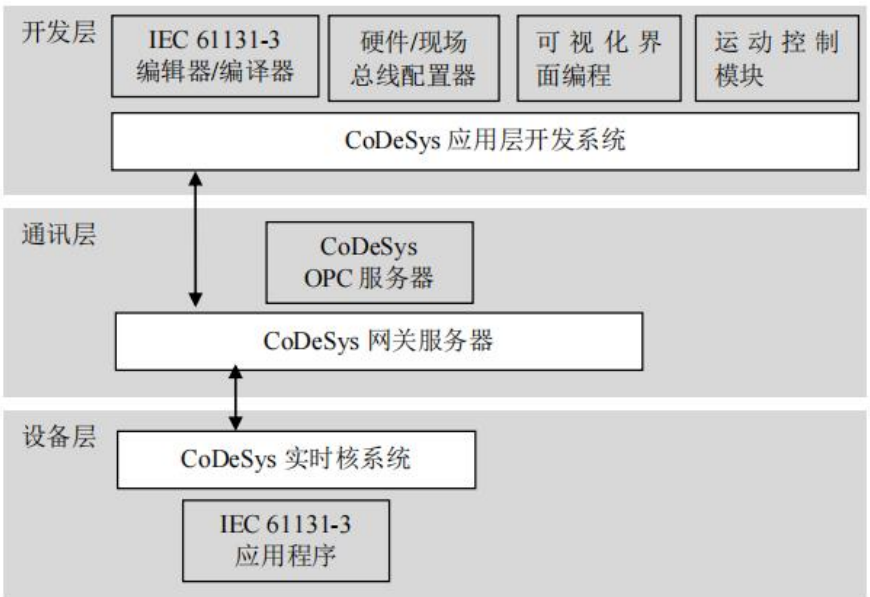


图 1-1 CODESYS 产品架构

本教程针对 CODESYS 的培训只是包含上图 1.1.2 中的应用层级，CODESYS IDE（集成开发环境）部分，这个部分包含了编辑器，配置器，调试器，运动控制，可视化等，是本教程学习的主要目标。目前 IDE 有两个大版本，CODESYSV2.3 版本和 CODESYSV3.5 版本，CODESYS 提供免费的编程软件，在官网可以下载其最新安装包。

1. 开发层

PLC 编程系统，PLC Development System CoDeSys（具有完善的在线编程和离线编程功能）、编译器及其配件组件、可视化界面编程组件等，同时供用户可选的运动控制模块可使其功能更加完整和强大。

IEC61131-3 编辑器：CoDeSys 提供了所有 IEC61131-3 所以定义的五种编程语言：如结构化文本（ST）、顺序功能图（SFC）、功能块图（FBD）、梯形图（LD）和指令表，此外还支持连续功能图（CFC）的编程语言。

编译器：负责将 CoDeSys 中的应用程序转换为机器代码并且优化可编程控制器的性能。当用户输入了错误的应用程序代码时，立刻会接收到编译器发出的语法错误警告及错误信息，让编程人员可以迅速做出相应纠正。

硬件/现场总线配置器：针对不同制造商的硬件设备及不同现场总线协议，该部分负责 CoDeSys 中对相应参数进行设定。

可视化界面编程：直接在 CoDeSys 中即可实现可视化编程（人机界面 HMI），系统已经集成了可视化编辑器。

运动控制模块：运动控制功能已经集成在 CoDeSys 中，形成了 SoftMotion（CNC）软件包。

基于 PLCopen 的工具包可以实现单轴、多轴运动；电子凸轮传动；电子齿轮传动；复杂多轴 CNC 控制等。

2. 通信层

应用开发层和设备层之间的通讯是由 CoDeSys 中的网关服务器来实现的，CoDeSys 网关服务器中安装了 OPC 服务器。

CoDeSys 网关服务器：作用在应用开发层和硬件设备层之间，可以使用 TCP/IP 协议或通过 CAN 等总线实现远程访问，是 CoDeSys 开发工具包不可分割的一部分。

CoDeSys OPC 服务器：对基于 CoDeSys 进行编程的控制器，无需考虑所使用的硬件 CPU，已经集成并实现了 OPC V2.0 规范的多客户端功能，且能同时访问多个控制器。

3. 设备层

使用基于 IEC 61131-3 标准的编辑开发工具 CoDeSys 对一个硬件设备进行操作前，硬件

供应商必须要在设备层预先安装 CoDeSys 的实时核。

IEC 61131-3 应用程序：用户在开发层写完的程序通过以太网或串口下载至设备层中，最终该应用程序中的文件已经被转为二进制存放在目标设备中，根据用户设定的执行方式循环执行对应程序。

1.2、CODESYS 的菜单栏常用功能介绍

1.2.1 CODESYS IDE（集成开发环境）常用菜单区域介绍

CODESYS 常用的菜单按钮位置如下图，涉及联机调试见联调章节。

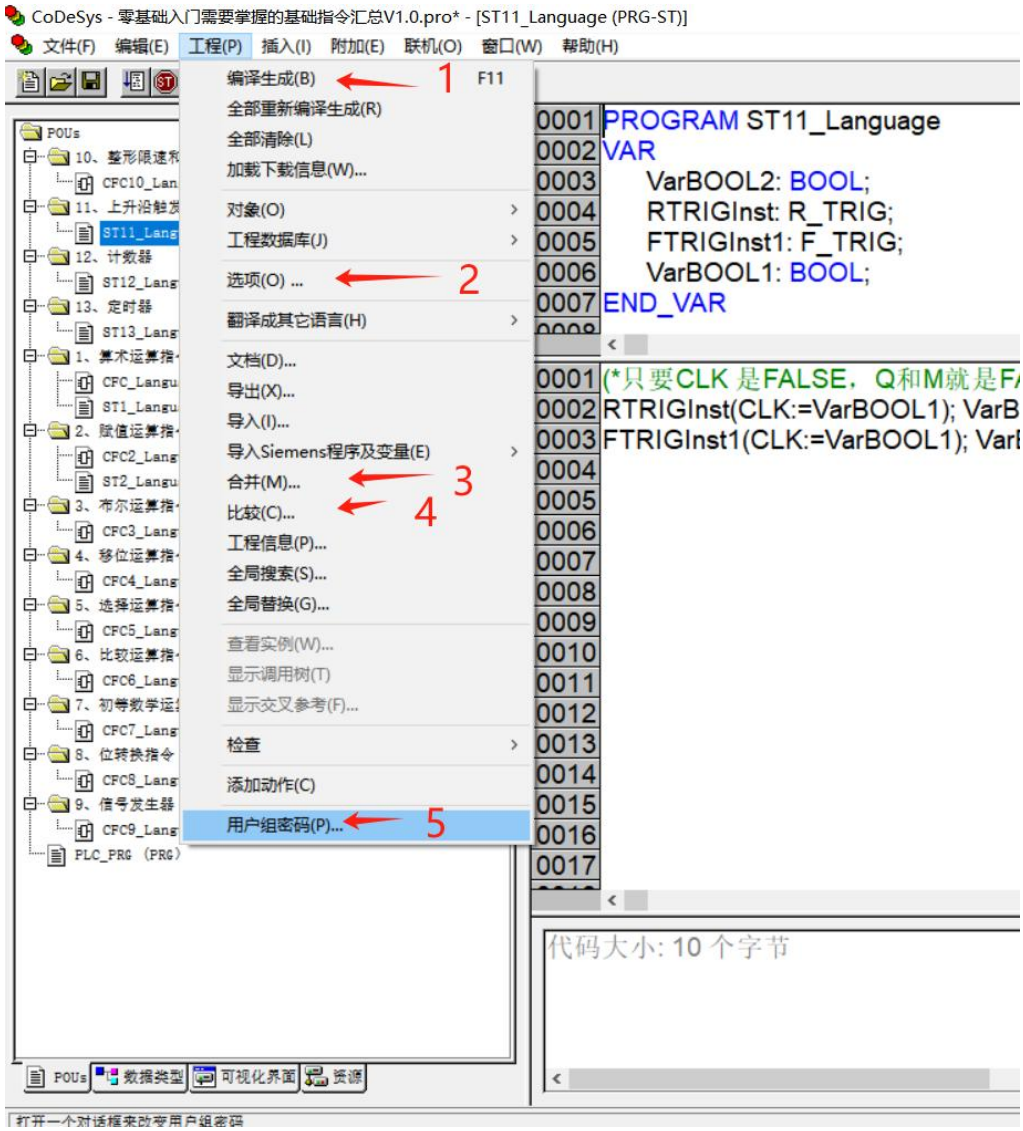


图 1-2 codesys2.3 菜单栏常用按钮

序号 1：编译，程序编写完毕后，或者编写了一部分，都可以通过编译来检查程序的语法或者其他错误。

序号 2：选项，选项中包含的几个主要功能：

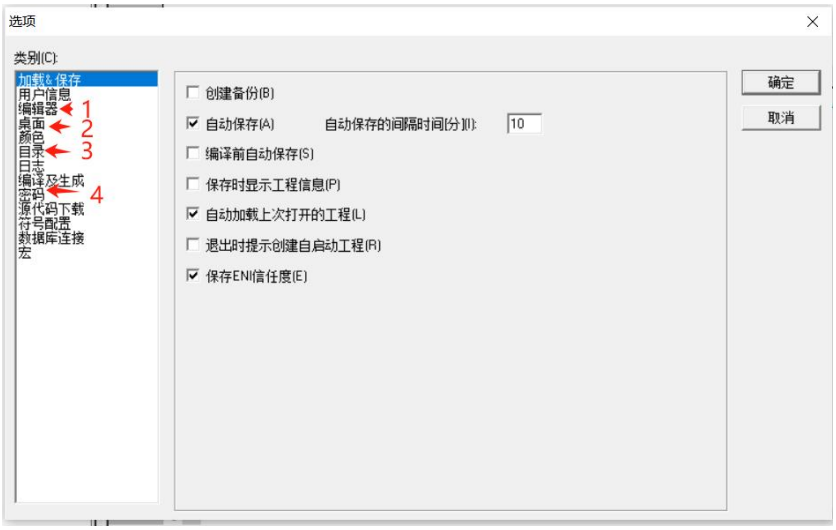


图 1-3 codesys2.3 菜单栏常用按钮

2.1 编辑器

图 1-4 codesys2.3 菜单栏常用按钮

此菜单主要用来设置程序代码的字体格式，大小。以及程序数值是默认显示十进制或者 16 进制。

2.2 桌面

图 1-5 codesys2.3 菜单栏常用按钮

此菜单，主要用来设置 codesys 界面的中英文或者其他语言的切换。

2.3 目录

图 1-6 codesys2.3 菜单栏常用按钮

工程文件从上图中，库文件，编译文件，配置文件等的位置目录去调用该工程文件打开时，对应的库文件，程序打开位置，以及程序的 target 配置文件（cfg 文件）。如果别的电脑转发到你电脑或者拷贝到你电脑的程序，需要在这个目录位置，设置好对应的上述三个文件的路径。

2.4 密码

图 1-7 codesys2.3 菜单栏常用按钮

序号 3：合并

合并的功能，主要用来在实际程序开发过程中，如果碰到相似的程序段，或者类似的程序及功能块，可以通过点击合并，选择对应的想复制的程序路径，选择对应的需要复制的子程序，

确认，就可以把其他项目程序的代码，复制到当前程序中。

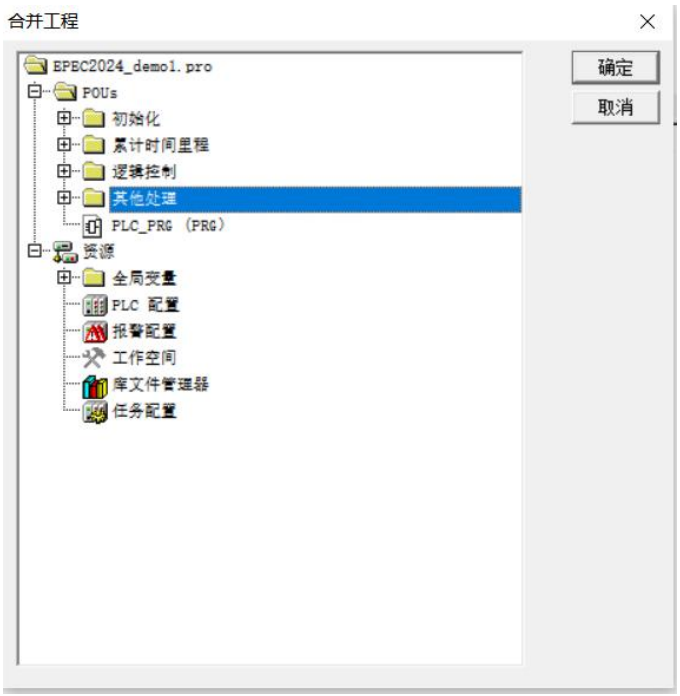


图 1-8 codesys2.3 合并工程

序号 4：比较

比较功能，主要用于程序在调试过程中，接近版本程序之前不同之处的对比，点击比较之后，选择需要对比程序的路径和目录，就可以同时对称打开两个程序，用于比较两个版本程序的不同之处，其不同之处，都会标红。

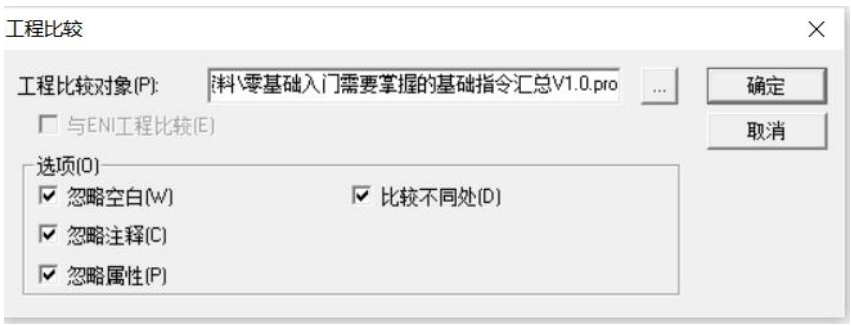
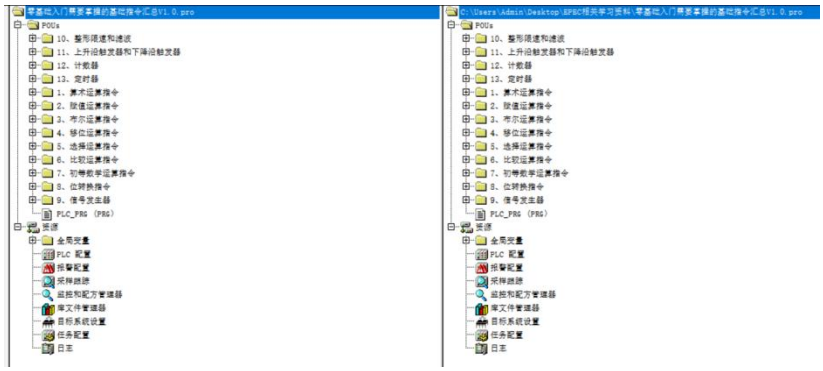


图 1-9 codesys2.3 比较工程



序号 5：用户组和密码

图 1-10 codesys2.3 用户分组

对程序打开的用户进行分组管理，0 级权限最高，7 级最低。

图 1-11 codesys2.3 用户权限

右键程序中任何一段程序，点击属性，可以对不同用户组，分配不同的程序修改，只读，存储等权限。

1.2.2 CODESYS IDE（集成开发环境）常用程序代码开发区域介绍

我们所安装的 V2.3 或者 V3.5codesys IDE，从软件界面的区域划分上可以看出主要分为四个部分：程文件包含 PLC 程序里的所有对象：POUS(program organization units)、数据类型、可视化，资源。

1.2.2.1 POU 的知识

其中 POU（program organization unit） 包括主程序（PRG）、子程序（PRG）、功能模块（FBD）及 函数（FB），每一个 POU 由两部分组成：变量说明部分和程序体。例：

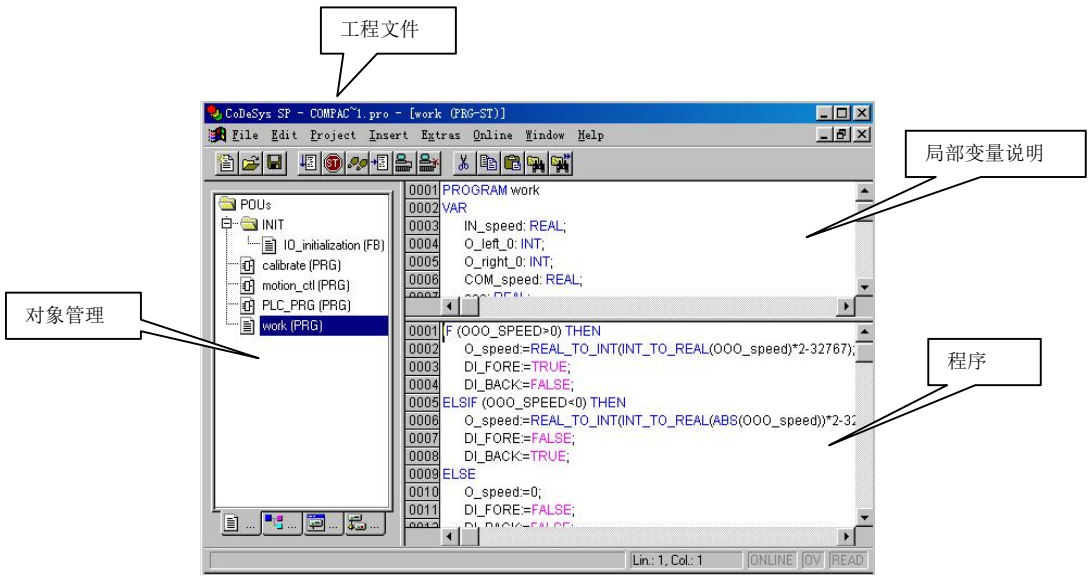


图 1-12 codesys2.3 代码开发区介绍

创建一个工程文件时，打开 CoDeSys 后选择 "File" "New"，这时出现一个对话框：

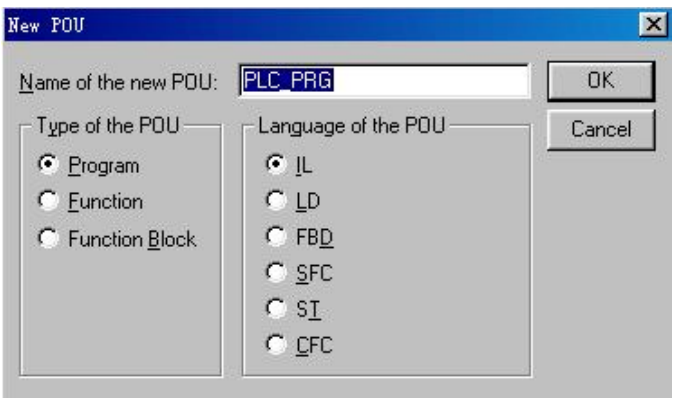


图 1-13 codesys2.3 POU

这是此工程文件的第一个 POU，已经命名为一个默认的名字 PLC_PRG，类型被定义为程序。可以根据自己喜好修改名字，选择一种要编程的语言，创建第一个 POU。因为在任何一个工程文件中，必须要有这一文件，一般地把它作为主程序。

要创建其他的 POU 时，把光标移到对象管理区，按鼠标右键，选择“Add Object”，或选择主菜单 “Project”“Object”“Add”，出现上述菜单，输入文件名、选择 POU 类型及编程语言，按“OK”结束。当你存盘时，系统会提示你输入工程文件名，文件名的命名原则跟 Windows 文件名要求一致。

在工程文件中，按适用范围有两种类型的变量，全局变量、局部变量、输出变量和输入变量。全局变量的说明在“resource”的 “global variable”里：

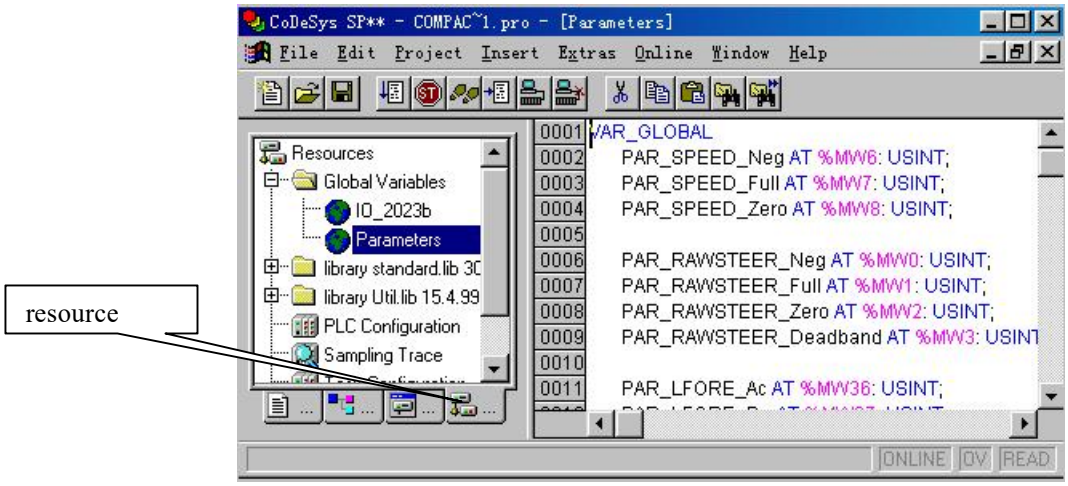


图 1-14 codesys2.3 资源

输入变量、输出变量和局部变量的说明在“程序体”上部的局部变量说明区。 变量说明有两种方式:一种是在变量区进行说明;另一种是自动说明。自动说明是在主菜单里选择“Project”,“Option”,“Edit”,出现以下对话框:

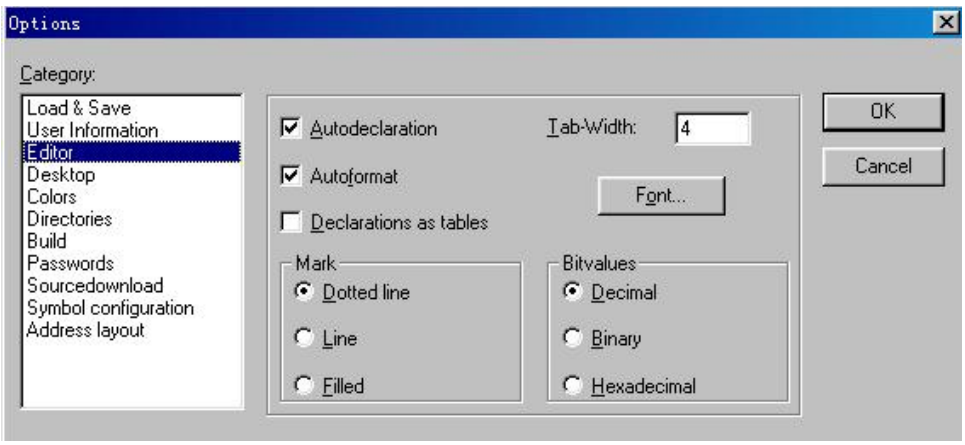


图 1-15 codesys2.3 自动变量说明

选中“Auto declaration”。这样,当编写程序,写到新的变量时,自动弹出对话框:

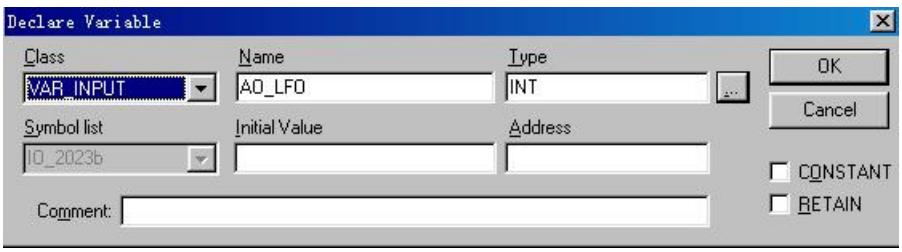


图 1-16 codesys2.3 自动变量说明

输入要定义的变量类型、地址、初始值。局部变量不用指定地址。

1.2.2.2 数据结构

结构(struct)是由一系列具有相同类型或不同类型的数据构成的数据集合,也叫结构体。

例如,一台电机通常都有其对应的信息,如产品型号(Product_ID)、生产厂家(Vendor)、

额定电压（Nominal Voltage）、额定电流（Nominal Current）、极对数（Poles），是否带刹车 （Brake）等信息。这些信息都和这台电机相关联，见表 3-12 所示。可以看出，如果将这些信息分别以独立的变量进行声明，很难反应出它们和电机的内在联系。如果有一种数据类型可以将它们组合起来的话，那就可以解决这个问题，在 CoDeSys 中结构体就能起到这样的功能。

Product_ID	Vendor	Nominal Voltage	Nominal Current	Poles	Brake
11000	FESTO	380	5.2	4	YES

图 1-17 电机结构参数

结构体和其他类型基础数据类型一样，例如 int 类型，char 类型，只不过结构体可以做成你想要的数据类型。以方便以后的使用。

在实际项目中，结构体是大量存在的。工程人员常使用结构体来封装一些属性来组成新的类型。所以在项目中通过对结构体内部变量的操作将大量的数据存储在内存中，以完成对数据的存储和操作。结构体其最主要的作用就是封装。封装的好处就是可以再次利用。

结构体声明的语法如下：

```
TYPE <结构名>:
STRUCT
<变量的声明 1>
.
.
<变量声明 n>
END_STRUCT
END_TYPE
```

<结构名>是一种可以在整个工程中被识别的数据类型，可以像标准数据类型一样使用。

1.2.1.3 可视化

使用'Project’ 'object Add’ 命令，你就能建立一个新的可视化对象。打开'New visualization’对话框，你就能在此输入新的可视化对象的名称。一旦作出有效输入，意味着它不该是已经存在并使用着的名称，也不可使用特殊字符，确定后便可点击 OK 关闭对话框。在窗口打开的情况下，你可以对新的可视化进行编辑。

图 1-18 添加一个可视化视图

可视化元素是一种图形元素，可用来填充一个可视化对象。在 CoDeSys 下拉式菜单中提供有效的元素。每一元素会有各自的组态。

可以将各种几何图形，也可以是 bitmaps 文件，图元文件，按钮和现有的可视化插入你的可视化中。

可使用的几何图形有:矩形、圆导角矩形、椭圆/圆及多边形。进入'Insert’菜单，你可以任意选择下列命令:

图 1-19 可视化元素视图

Codesys 可视化的用法和西门子 wincc 上位机组态软件类似，都是通过添加插入一些元素，图元文件等，然后通过双击这些元素，对其属性进行定义和变量进行关联，这样就会产生一些动态的效果和实时的状态参数显示，从而实现对整个设备状态和运行参数的实时观测。比如，状态告警，实时参数，参数标定等。

目前国内支持 codesys 可视化编程的显示器基本上都是 codesys_V3.5 版本。
其套路和用法和 2.3 类似。一般来说，从所购显示器硬件的厂家去要一些快速上手手册和例程，可以很方便的实现可视化编程。

1.2.1.4 资源

在资源选项卡中，我们可以进行下述操作:

- 定义程序的全局变量
- 库函数管理
- 配置 PLC
- 配置 Target
- 任务配置

图 1-20 资源区域

1、定义全局变量

在工程文件中，按作用域划分，有两种类型的变量：全局变量（Global）、局部变量(local)。

全局变量存在于程序的全部模块中，而局部变量只存在于定义它的那个模块中。全局变量在“Resources”的“Global Variables”中声明：

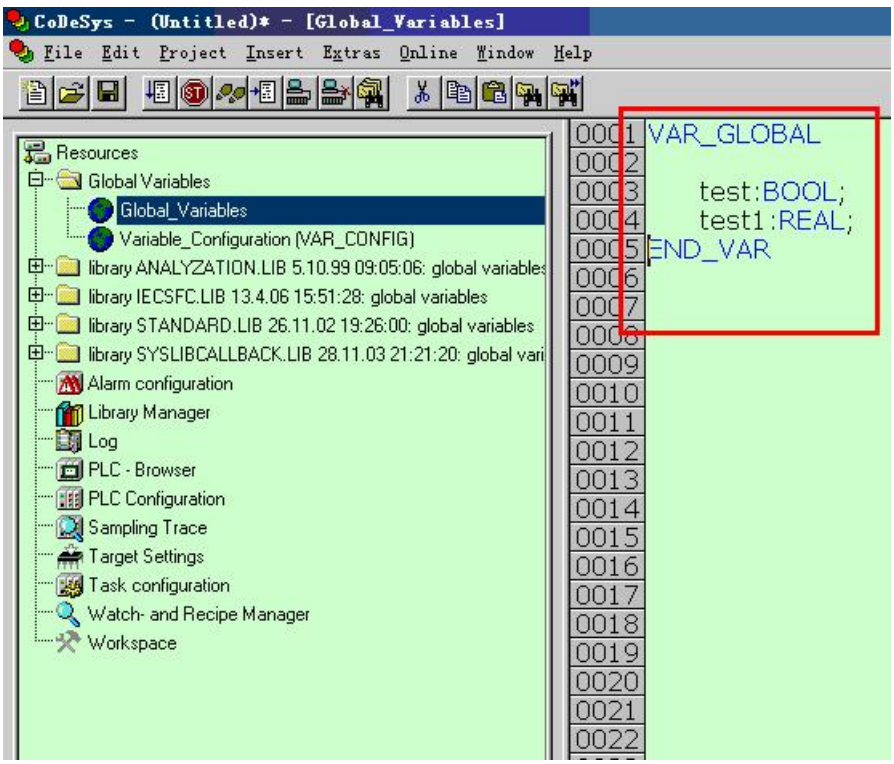


图 1-21 全局变量定义

局部变量在“程序体”上方的变量声明区声明：

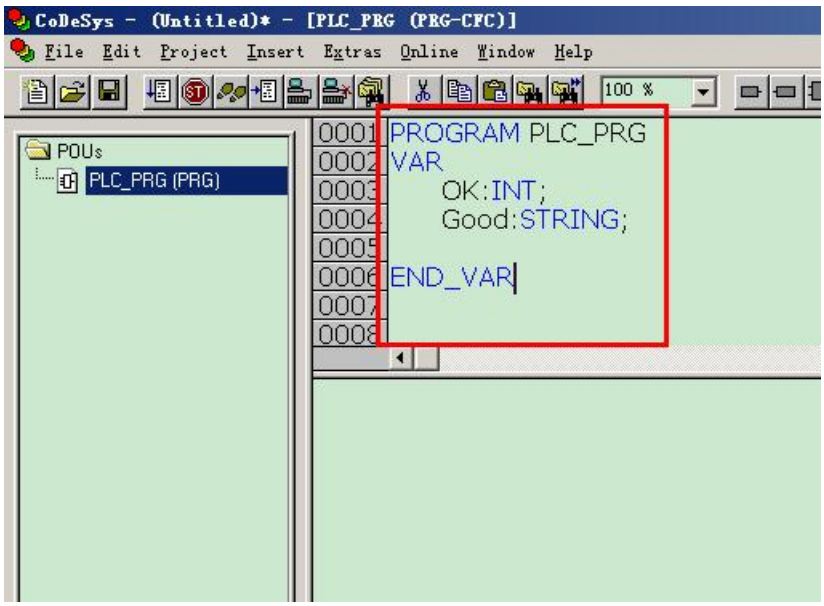


图 1-22 局部变量定义

变量有两种声明方式：一种是手动声明，一种自动声明，声明方法见图 1-15 和图 1-16。

2、PLC 配置

在 Resources 选项卡中，我们可以配置和硬件相关的程序接口。如下图所示，在 Resources 选项卡内双击 PLC Configuration，出现下图所示对话框：

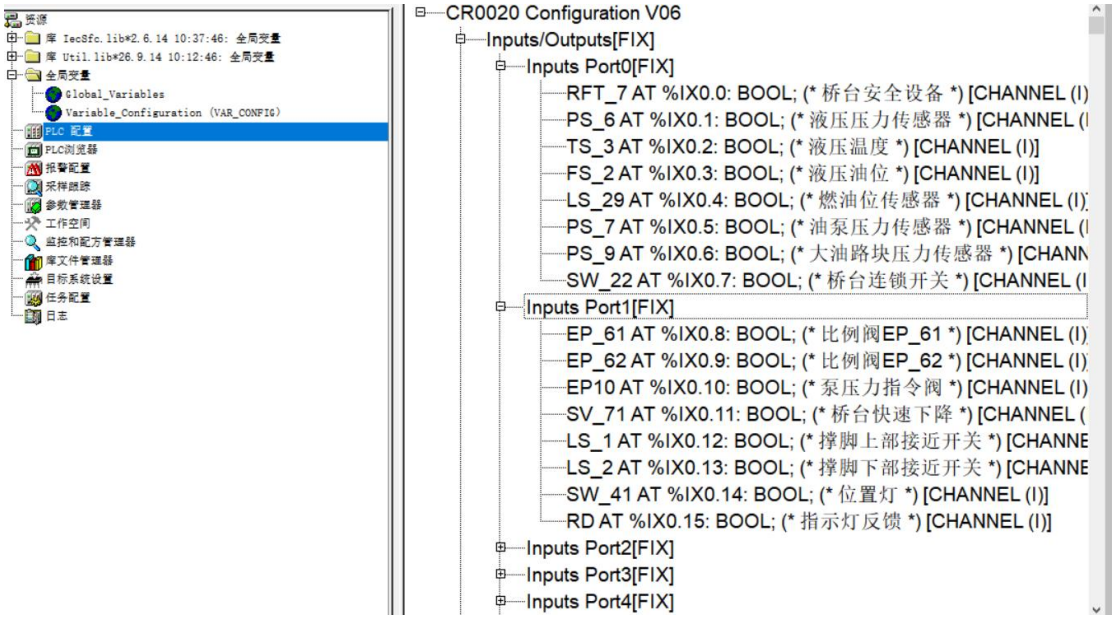


图 1-23 PLC 配置 IO

此端口配置能出现的前提条件为，在 codesys 中加载了 io 配置相关的 CFG 文件。如果打开程序后无法显示该部分 IO 的配置，需要向控制器厂家索要 target 文件或者 CFG 文件，进行安装或者在选项里面进行路径配置。

3、调用库文件

CoDeSys 有大量的库文件（后缀名为 lib）供用户编程时调用。当你要调用某一 操作指令时，把包含该指令的库文件调入当前工程文件的库中。

操作如下：点击主菜单“Window”， “Library Manager”，弹出对话框：

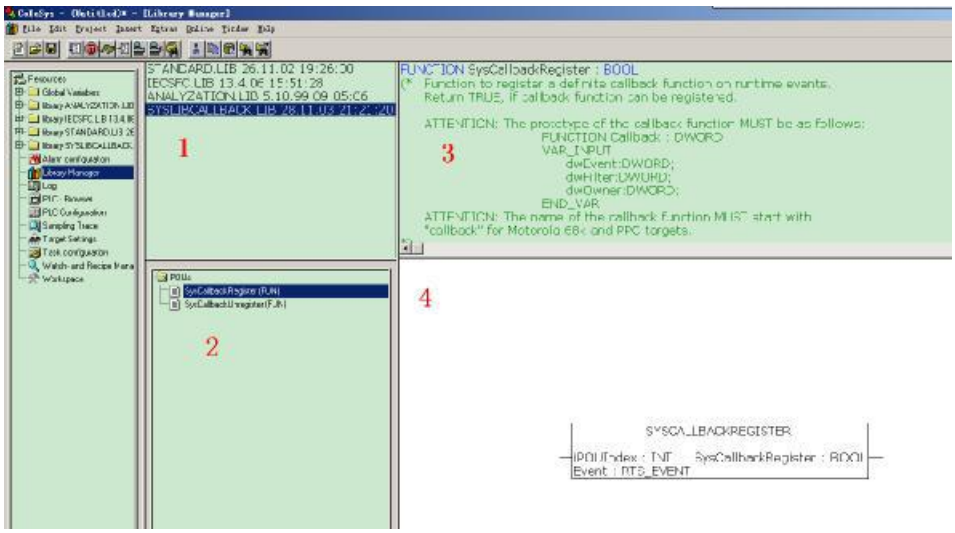


图 1-24 codesys 库文件的添加 1

将光标移到库文件区 1，按右键，选择“Additional Library…Ins”，弹出对话框：

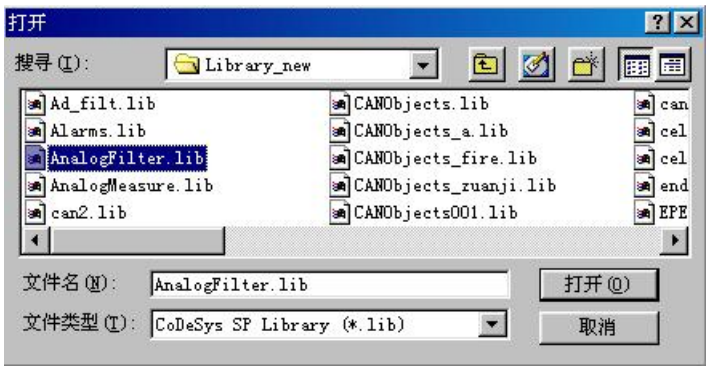


图 1-25 codesys 库文件的添加 2

选中要用的文件即可。关于一些库文件的详细说明，请参看下一章。

删除库函数

选中想要删除的库函数，点右键，选择“Delete Del”菜单项，系统提示：点“是”，即可将库函数删除，点“否”，取消删除操作。也可使用快捷键操作：选中想要删除的库函数，按 delete 键删除。

查看库函数的属性

选中想要查看属性的库函数，点右键，选择“Properties… (Alt + Enter)”就会弹出对话框，显示库函数的一些属性。以 STANDARD.LIB 为例：

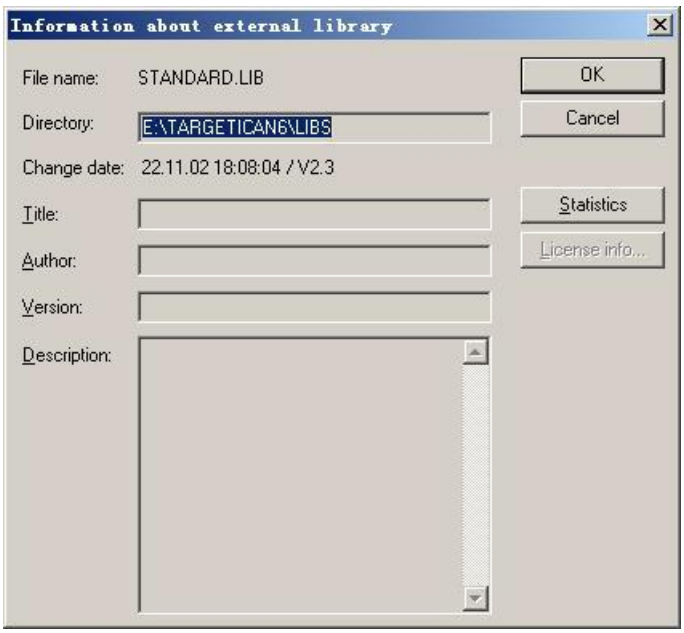


图 1-26 codesys 库文件信息查看

File name: 库函数名称

Directory: 库函数所在文件夹

Change date: 最后修改日期

Title:

Author: 作者

Version: 版本号

Description: 库函数描述

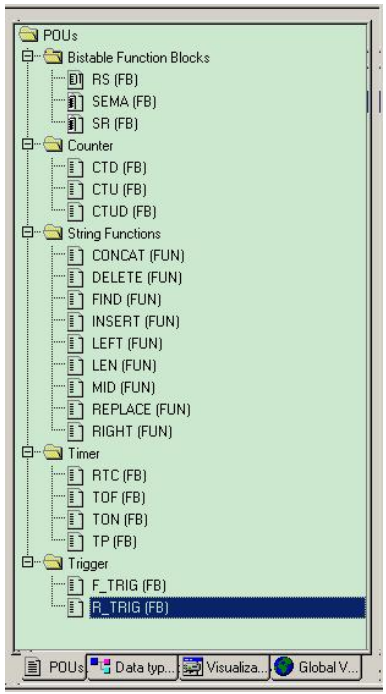


图 1-27 codesys 库文件信息查看

在 3 区域，显示了 2 区域中选中了的函数（功能块）的信息，包括函数（功能块）名称、函数（功能块）信息描述、输入输出变量及描述、中间变量及描述等。

在 4 区域用图形方式显示了函数（功能块）的名称、输入输出变量的名称及类型。有的函数还使用一个小图像形象的描述了函数的功能。例如 R_TRIG 功能块，使用一个小图片说明这个功能块的功能是：当输入为上升沿电平时，输出 TRUE 信号，小灯泡点亮。如下图所示：

图 1-28 codesys 库文件功能块

4、目标系统设置

创建一个新工程时，会弹出下图所示 Target Settings 对话框：

图 1-29 target 设置

在这个对话框中选择我们所使用的 Target：Hirschmann iFLEXC3 Controller，会出现下图所示的对话框：

不同品牌的控制器，需要进行不同的 target 设定。在这里，一般都是选择以其控制器型号命名的控制器型号。

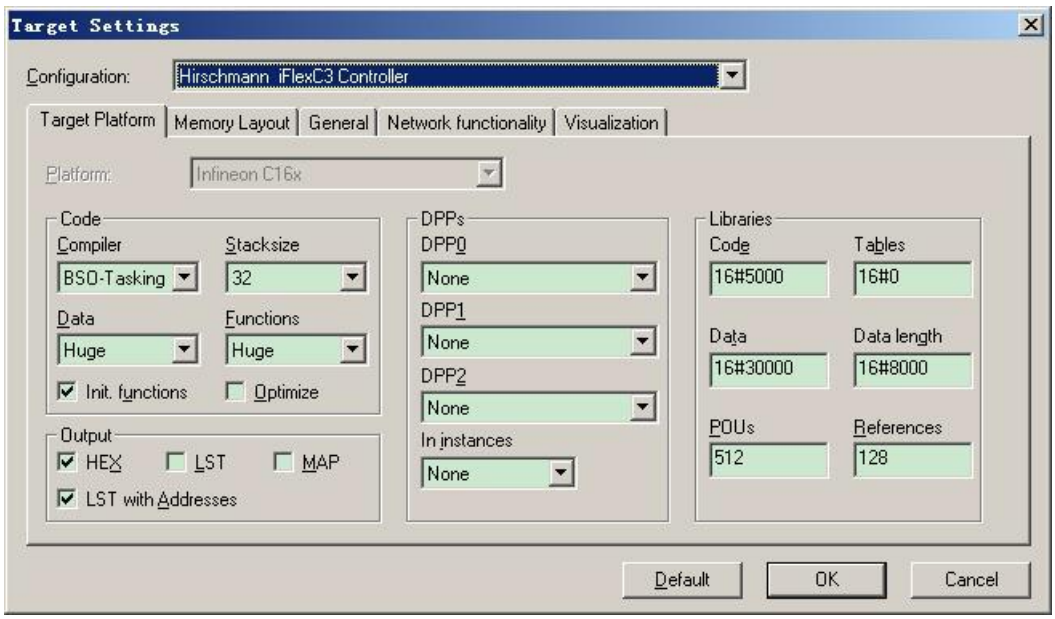


图 1-30 target 设置

◆ 这个对话框也可以通过在 Resources 选项卡中双击 Target Setting 出现。如下图位置所示：

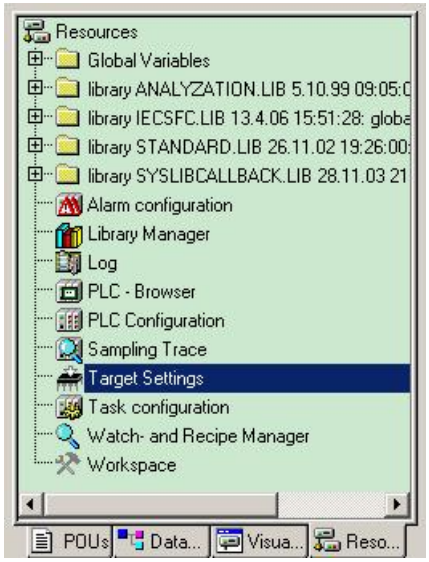


图 1-30 资源-target 设置

该部分可以理解为不同品牌控制器的硬件配置文件，也叫 target 文件。需要在哪个品牌的控制器硬件上进行 codesys 程序开发，就需要安装该品牌控制器的 target 文件和库文件。Target 文件的后缀一般为*.trg;或者*.Tnf,每个厂家的 target 文件配置不一样，需具体咨询各个控制器供应商。

5、任务设置

对每个任务，可以设定一串由任务启动的程序。如果在当前周期内执行此任务，那么这些程序会在一个周期的长度内被处理。优先权和条件的结合将决定任务执行的时序，任务设置界面如图 1-30 所示。

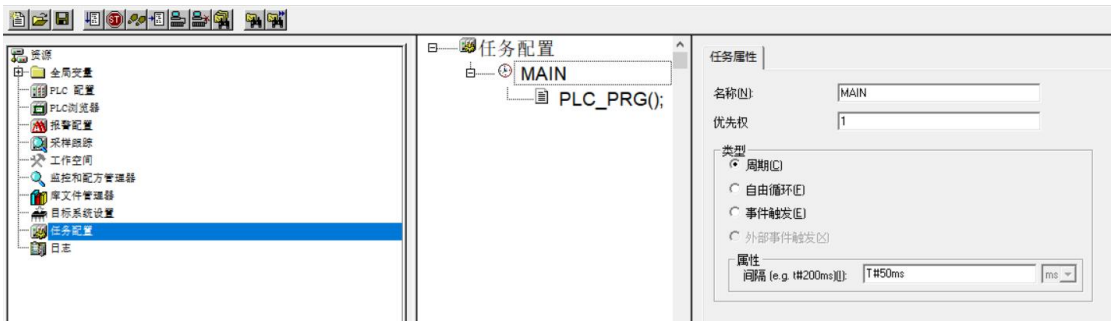


图 1-30 资源-任务设置

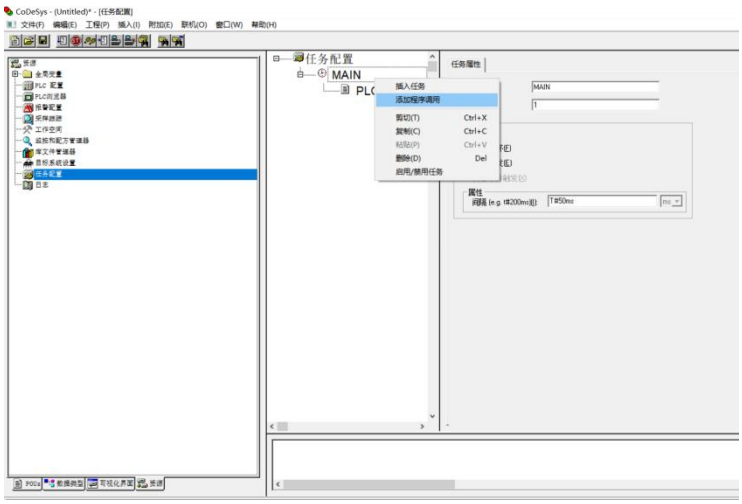


图 1-30 资源-任务设置

点击任务，右键添加程序调用，选择 pou 里面某一个程序，作为主程序。

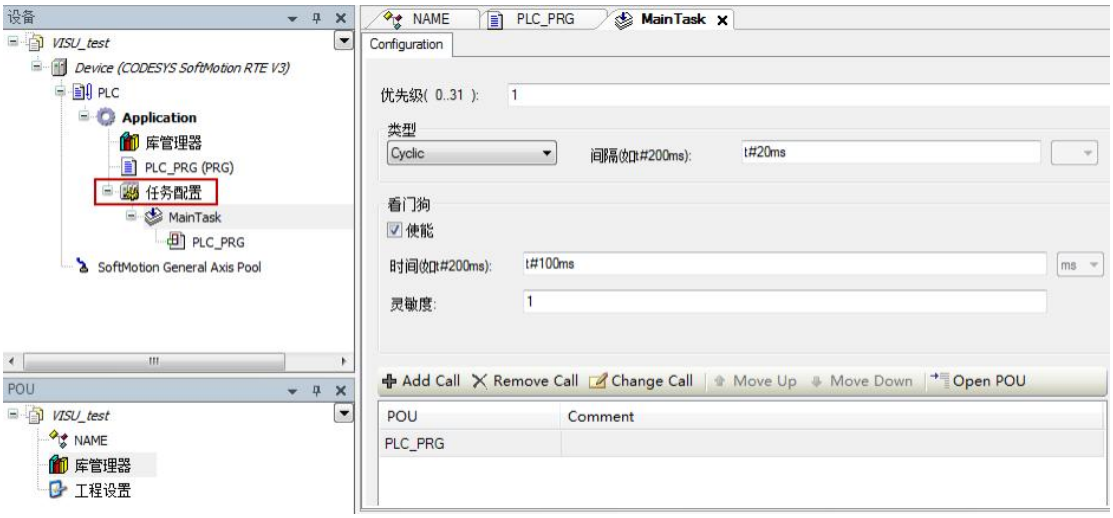


图 1-31 资源-任务设置

由于 CoDeSys V3.x 在任务配置时有如下的属性，编程者需遵循如下规则：

循环任务的最大数为 100；

自由运行任务的最大数为 100；

事件触发任务的最大数为 100；

任务配置中；

处理和调用程序是根据任务编辑器内自上而下的顺序所执行的在任务配置树的最顶端有条目“任务配置”。其中的内容是当前定义的任务，每个通过任务名代表。特定任务的 POUs 调用没有显示在任务配置树中。针对每个独立的任务可以对其进行执行的类型编辑及配置。包括固定周期循环、事件触发、自由运行和状态触发 4 种类型。

图 1-32 资源-任务设置

1) 固定周期循环-Cyclic

根据程序中所使用的指令执行与否，程序的处理时间会有所不同，所以实际执行时间在每个扫描周期都发生不同的变化，执行时间有长有短。通过使用固定周期循环方式，能保持一定的循环时间反复执行程序。即使程序的执行时间发生变化，也可以保持一定的刷新间隔时间。在这里，也推荐大家优先选择固定周期循环任务启动方式。

例如，假设将程序对应的任务设定为固定周期循环方式，间隔时间设定为 10ms 时，实际程序执行的时序图如图 1-33 所示。

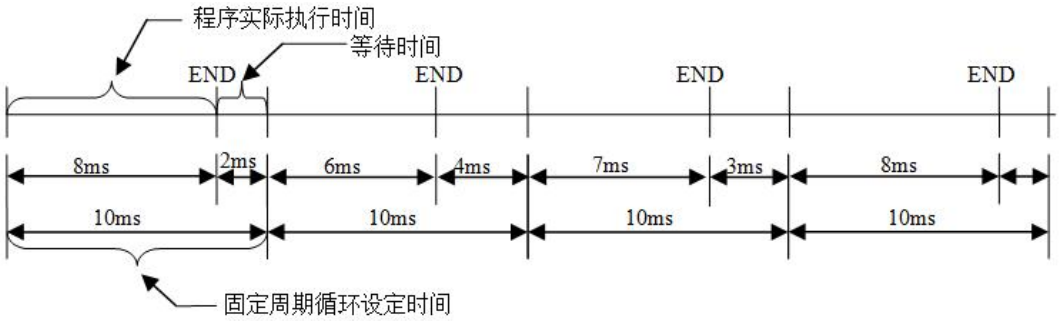


图 1-33 资源-循环任务执行顺序

如果程序实际执行时间在规定的固定周期循环设定时间内执行完，则空余时间用作等待。如应用中还有优先级较低的任务未被执行，则剩下的等待时间用来执行相对低优先级的任务。任务的优先级在后文会有详细的说明。

任务优先级

CoDeSys 中的可以对任务的优先级进行设置，一共可以设 32 个级别（0～31 之间的一个数字，0 为最高优先级，31 为最低优先级）。当一个程序在执行时，优先级高的任务优先于优先级任务低的任务，高优先级任务 0 能中断同一资源中较低优先级的程序执行，使较低优先级程序执行被放缓。

注意：

在任务优先级等级分配时，请勿分配具有相同优先级的任务。如果还存在其他任务视图先于具有相同优先级的任务，则结果可能不确定且不预知。

如果任务的类型为“Cyclic”，则按照“间隔”中的时间循环执行，具体设置如下图 1-34 所示

图 1-34 资源-任务设置

【例】假设有 3 个不同的任务，分别对应三种不同的优先等级，具体分配如下。

任务 1 具有优先级 0 和 循环时间 10 ms ，

任务 2 具有优先级 1 和 循环时间 30 ms ，

任务 3 具有优先级 2 和 循环时间 40 ms 。

在控制器内部，各任务的时序关系如图 2.20 所示，具体说明如下：

0～10ms：先执行任务 1（优先级最高），如在本周期内已将程序执行完，剩余时间执行任务 2 程序。但是如果此时任务 2 没有被完全执行完，但时间已经到了第 10ms 时，由于任务 1 是每 10ms 执行一次的且优先级更高，此时将会打断任务 2 的执行。

10～20ms：先将任务 1 的程序执行完毕，如有剩余时间，再执行上个周期为完成的任务 2。

20～30ms：由于任务 2 是每 30ms 执行一次的，在 10～20ms 之间任务 2 已经全部执行完毕， 此时不需要再执行任务 2，只需将优先级最高的任务 1 执行一次即可。

30～40ms：与之前类似。

40～50ms：此时出现了任务 3，任务 3 的优先级更低，所以只有在确保任务 2 彻底执行完后，才能执行任务 3

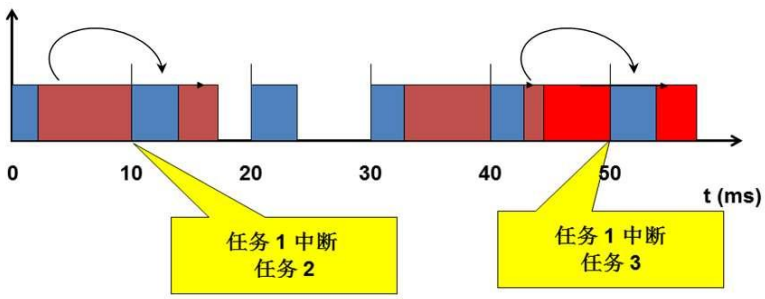


图 1-35 不同优先级任务，循环执行时序

1.3、CODESYS 需要掌握的基础指令

Codesys 的初学者需要**必须掌握**的基础指令如下图，已经整理独立的常用指令文件，供参考学习。这个只要有西门子或者传统 PLC 的指令背景都很好掌握。

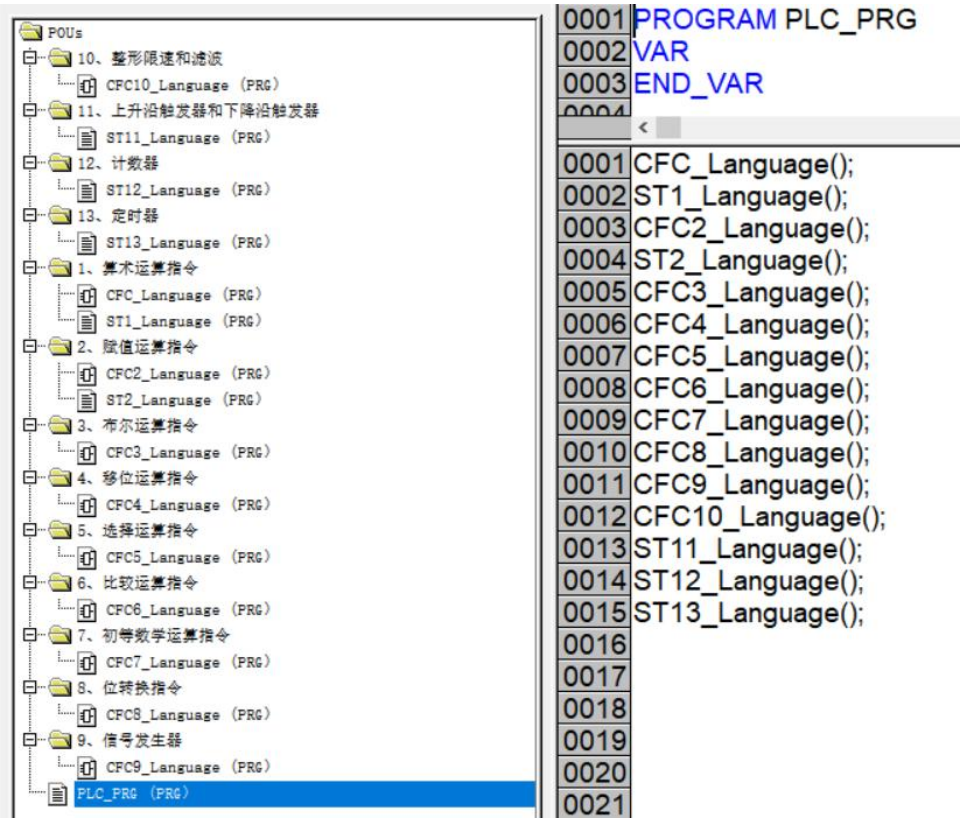


图 1-36 CODESYS 需要掌握的基本指令

掌握了上述基本指令之后，结合本教程的编程思路，基本可以让入门小白达到可以修改，调试程序，简答修改局部程序的能力，之后根据具体项目的实操和实战训练，就可以独立对设备程序进行开发。

1.4、CODESYS 中 POU 和需要掌握的几种编程语言

Codesys 中 POU 的概念

程序组织单元（Program Organization Unit，POU）由声明区和代码区两部分组成，是用户程序的最小软件单元，它相当于传统编程系统中的块（Block），是全面理解新语言概念的基础。按功能分程序组织单元（POU）可分为函数（FUN）、功能块（FB）和程序（PRG）。

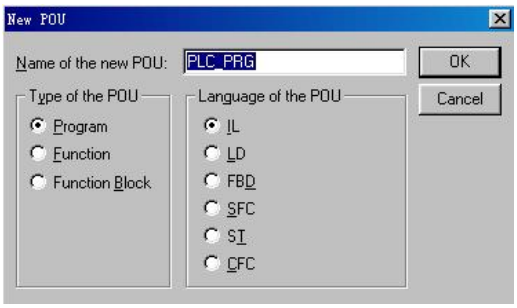


图 1-37 CODESYS 需要掌握的三种 POU

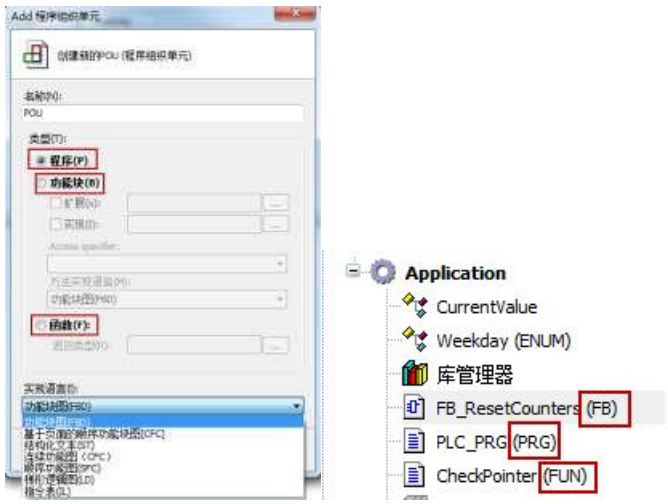


图 1-38 CODESYS 需要掌握的三种 POU（V3.5）

三种 POU，我们常用的是程序 PRG 和 FB 功能块，FUN 函数次之

程序（Program）是规划一个任务的主核心，程序拥有最大的调用权，可以调用功能块及函数。 一般而言分为主程序、子程序，广义上讲，也包含硬件配置、任务配置、通讯配置及目标设置信息。

功能块（Function Block）是把反复使用的部分程序块转换成一种通用部件，他可以在程序中被任何一种编程语言所调用，反复被使用，不仅提高了程序的开发效率，也较少了编程中的错误，从而改善了程序质量。

函数（FUN）是没有内部状态（没有运行时的内存分配）的基本算法单元。也就是说只要给定相同的输入参数，调用函数必定得到相同的运算结果，绝对没有二义性。我们平时使用的各种数学运算函数，如 sin(x)、sqrt(x)等，就是典型的函数类型。

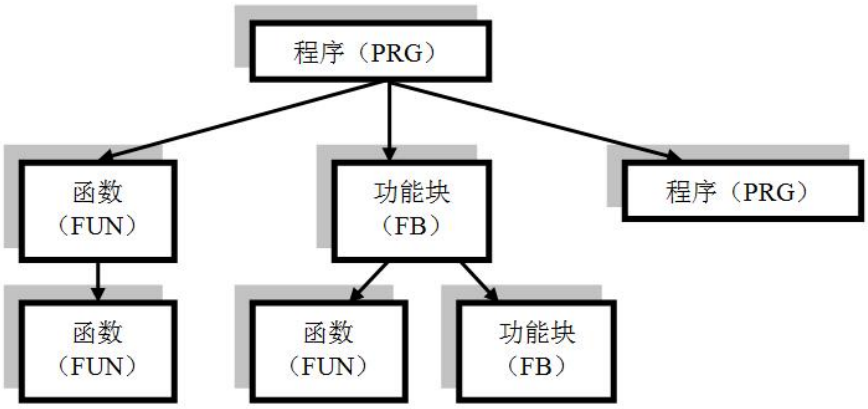


图 1-38 CODESYS 程序调用关系

CODESYS 的几种编程语言

1）梯形图（LD）：梯形图基于图形表示的继电器逻辑，是 PLC 编程中被最广泛使用一种图形化语言。梯形图程序的左、右两侧有两垂直的电力轨线，左侧的电力轨线名义上为功率流

从左向右沿着水平梯级通过各个触点、功能、功能块、线圈等提供能量，功率流的终点是右侧的电力轨线。

每一个触点代表了一个布尔变量的状态，每一个线圈代表了一个实际设备的状态，功能或功能块与 IEC 1131-3 中的标准库或用户创建的功能或功能块相对应。

2）功能块图（FBD）：功能块图用来描述功能、功能块和程序的行为特征，还可以在顺序功能流程图中描述步、动作和转变的行为特征。

功能块用矩形块来表示，每一功能块的左侧有不少于一个的输入端，在右侧有不少于一个的输出端，功能块的类型名称通常写在块内，但功能块实例的名称通常写在块的上部，功能块的输入输出名称写在块内的输入输出点的相应地方。

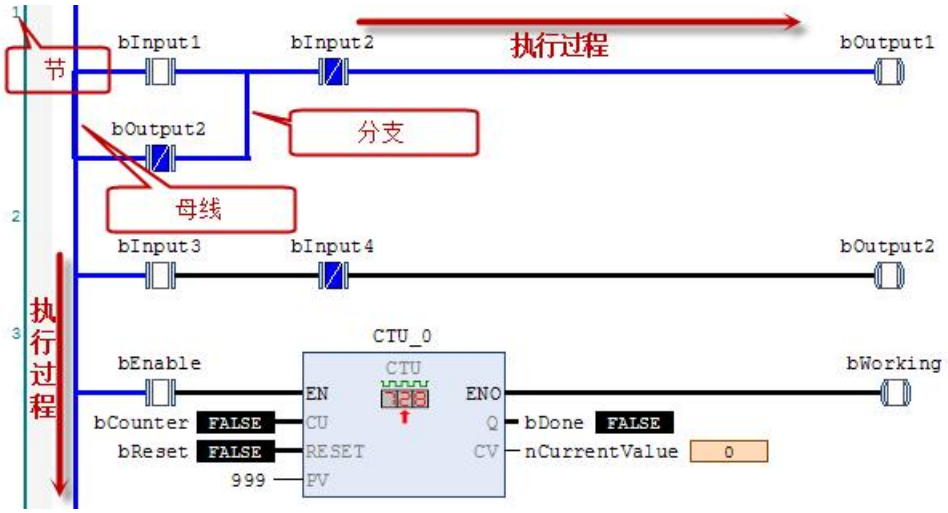


图 1-39 CODESYS 梯形图执行流程

3）指令表（IL）：优点是易于记忆及掌握，与梯形图（LD）有对应关系，便于相互转换和对程序的检查，且编程及调试时不受屏幕大小的限制，输入元素不受限制。缺点和梯形图一样，对复杂系统的程序描述不够清晰。

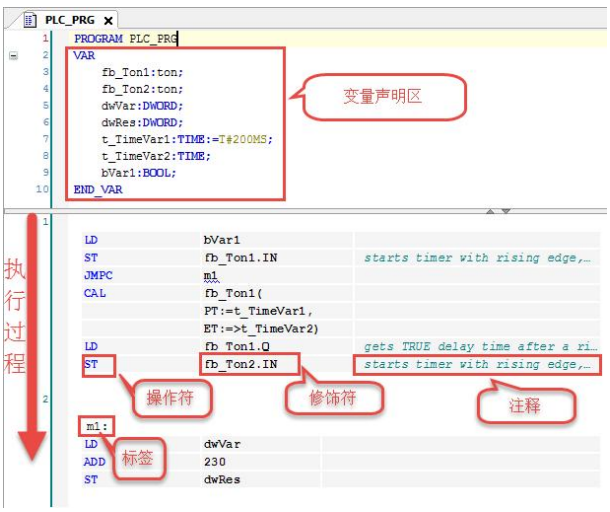


图 1-40 CODESYS IL 指令流程

操作符	修饰符	含义	举例
LD	N	将操作数（取反）载入累加器	LD iVar
ST	N	将累加器中的数（取反）存入操作数变量中	ST iErg
S		当累加器中的数为真，将操作数（布尔型）设为真	S bVar1
R		当累加器中的数为假，将操作数（布尔型）设为真	R bVar1
AND	N, (	累加器中的数和（取反的）操作数逐位进行与运算	AND bVar2
OR	N, (	累加器中的数和（取反的）操作数逐位进行或运算	OR xVar
XOR	N, (	累加器中的数和（取反的）操作数逐位进行异或运算	XOR N, (bVar1, bVar2)
NOT		累加器中的数逐位取反	
ADD	(	将累加器中的数和操作数相加，其结果复制到累加器中	ADD (iVar1, iVar2)
SUB	(	累加器中的数减去操作数，其结果复制到累加器中	SUB iVar2
MUL	(	累加器中的数和操作数相乘，其结果复制到累加器中	MUL iVar2
DIV	(	累加器中的数除以操作数，其结果复制到累加器中	DIV 44
GT	(	检查累加器中的数是否大于操作数，其结果（布尔型）复制到累加器中 ；>	GT 23
GE	(	检查累加器中的数是否大于或等于操作数，其结果（布尔型）复制到累加器中 >=	GE iVar2
EQ	(	检查累加器中的数是否等于操作数，其结果（布尔型）复制到累加器中；=	EQ iVar2
NE	(	检查累加器中的数是否不等于操作数，其结果（布尔型）复制到累加器中；<>	NE iVar1
LE	(	检查累加器中的数是否小于或等于操作数，其结果（布尔型）复制到累加器中；<=	LE 5
LT	(	检查累加器中的数是否小于操作数，其结果（布尔型）复制到累加器中 无条件（有条件）跳转到标签；<	LT cVar1
JMP	CN	无条件（有条件）跳转到标签	JMPN next
CAL	CN	（有条件）调用一个程序或功能块（当累加器中的数为正时）	CAL prog1
RET		从当前 POU 中返回，跳回到调用的 POU 中	RET
RET	C	有条件-仅当累加器中的数为真时，从当前 POU 中返回，跳回到调用的 POU 中	RETC
RET	CN	有条件-仅当累加器中的数为假时，从当前 POU 中返回，跳回到调用的 POU 中	RETCN
)		评估延迟的操作数	

图 1-41 CODESYS 常用 IL 指令列表

4）结构化文本（ST）：结构化文本编程语言是一种高级语言，类似于 Pascal，是一种特别为工业控制应用而开发的一种语言，也是在 CoDeSys 中最常用的一种语言。

图 1-42 CODESYS 结构化文本 ST 程序执行顺序

5) 顺序流程功能图（SFC）：已完成的功能为主线，优点是操作过程条理清楚，便于对程序操作过程的理解和思路；对大型程序可分工设计，采用较灵活的程序结构，节省程序设计时间和调试时间，由于只对活动步进行扫描，因此，可缩短程序执行时间。 EPEC 系列控制器通过 multitool 生成的应用层程序架构。

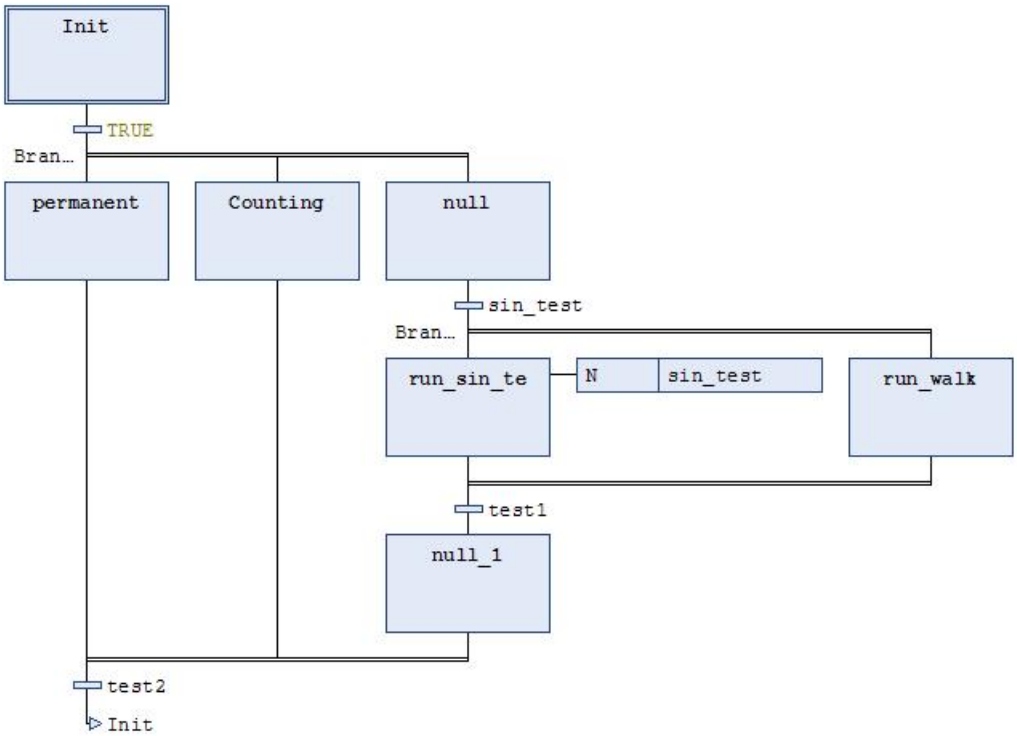


图 1-43 CODESYS SFC 程序执行顺序

6) 连续功能图（CFC）：实际上是功能块图（FBD）的另一种形式。在整个程序中可自定义运算块的计算顺序，易于实现大规模、数量庞大但又不易细分功能的流程运算。在连续控制行业中，得到大量使用。

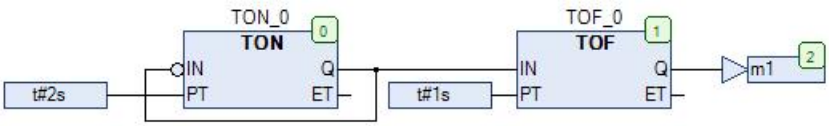


图 1-44 CODESYS SFC 程序执行顺序

7) 在 CFC 语言里元素的右上角的数字，显示了在线模式下 CFC 中元素的执行顺序。执行流程从编号为 0 的元素开始，在每个 PLC 运算周期内，0 号元素令总是第一个被执行。当手动移动该元素时，它的编号仍保持不变。当添加一个新元素时，系统自动按照拓扑序列（从左到右，从上到下）会自动获得一个编号，

如图 5.x 红色部分所示。

图 1-45 CFC 编程语言顺序编号

CFC 语言中运算块、输出、跳转、返回和标签元素的右上角的数字，显示了在线模式下 CFC 中元素的执行顺序。执行流程从编号为 0 的元素开始。考虑到执行顺序会影响到结果，在一定情况下可以改变执行顺序。操作在菜单“CFC”下的“执行顺序”中的子菜单命令可以改变元素的执行顺序。执行顺序包含的命令有：置首、置尾、向上移动、向下移动、设置执行顺序、按数据流排序、按拓扑排序，如图 1-46 所示。

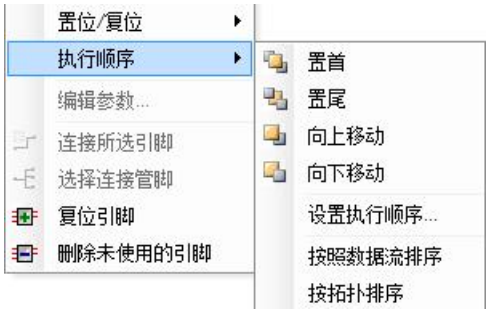


图 1-46 CFC 顺序编排

在实际的工程项目中，涉及到算法部分请选择 ST 语言，编写的程序往往简洁而高效；涉及到流程控制部分，请选择 SFC 语言，编写的程序会条理清晰，逻辑关系不会混乱；涉及到逻辑控制部分，请选择 LD 语言，编写的联锁，互锁等逻辑简单易懂；涉及到功能块部分，请选择 CFC 或者 FBD，编写的程序会形成一个网络清晰的网状电路图，易于读懂。

1.5、CODESYS 的数据类型和常用句式

1.5.1 数据类型

在CoDeSys 环境中，有以下标准数据类型：

BOOL（布尔量）；

SINT（短整型）、INT（整型数）、DINT（双整型数）；

USINT（无符号短整型）、UINT（无符号整型数）、UDINT（无符号双整型数）；

BYTE（位）、WORD（字）、DWORD（双字）；

STRING（字符量）；

REAL（实型数）、LREAL（长实型数）；

TIME（时间量）；

数据大类	数据类型	关键字	位数	取值范围
布尔	布尔	BOOL	1	FALSE(0)或 TEUE(1)
整型	字节	BYTE	8	0～255
	字	WORD	16	0～65535
	双字	DWORD	32	0～4294967295
	长字	LWORD	64	0～(2^64-1)
	短整型	SINT	8	-128～127
	无符号短整型	USINT	8	0～255
	整型	INT	16	-32768～32767
	无符号整型	UINT	16	0～65535
	双整型	DINT	32	-2147483648～2147483647
	无符号双整型	UDINT	32	0～4294967295
	长整型	LINT	64	-2^63～(2^63-1)
实数	实数	REAL	32	1.175494351e-38～3.402823466e+38
	长实数	LREAL	64	2.2250738585072014e-308～ 1.7976931348623158e+308
字符串	字符串	STRING	8*N	
时间数据	时间	TIME	32	T#0ms～T#71582m47s295ms
		TIME_OF_DAY		TOD#0:0:0～TOD#1193:02:47.295
		DATE		D#1970-1-1～D#2106-02-06
		DATE_AND_TIME		DT#1970-1-1-0:0:0 ～DT#2106-02-06-06:28:15

图 1-47 CODESYS 标准数据类型和数据范围

这里对 BOOL 型，WORD 型等的数据的范围和占用存储空间有个概念。因为在程序中定义变量时，需要考虑变量的数据范围，选择对应的数据类型。

在控制器和显示器，控制器和控制器之间通信时，往往会出现带小数点的数据，比如水温，工作小时等。由于总线传输的数据最小单位为 bit，1 或者 0，小数点是无法直接传输的，所以针对这些带小数点的变量，如果需要在总线链路进行传输，我们一般的处理方法是一个小数点的，发送方将数据放大 10 倍，接收方收到后再除以 10；依次类推，2 个小数点的，发送方将数据放大 100 倍，接收方收到后再除以 100。

图 1-48 CODESYS 标准数据类型和数据范围

1.5.2 常用句式

1、ST 写法中的常用句式

结构化文本语句表主要有 5 种类型，即赋值语句、函数和功能块控制语句、选择语句、迭代（循环）语句、跳转语句。表 4-13 列举了所有结构化文本用到的语句。

指令类型	指令语句	举例
赋值语句	:=	bFan:= TRUE;
函数及功能块控制语句	功能块/函数调名（）；	
选择语句	IF	IF <布尔表达式> THEN <语句内容>; END_IF
	CASE	CASE <条件变量> OF <数值 1>: <语句内容 1>; ... <数值 n>: <语句内容 n>; ELSE <ELSE 语句内容>; END_CASE;
迭代语句	FOR	FOR <变量> := <初始值> TO <目标值> [BY <步长>] DO <语句内容> END_FOR;
	WHILE	WHILE <布尔表达式> <语句内容> ; END_WHILE;
	REPEAT	REPEAT <语句内容> UNTIL <布尔表达式> END_REPEAT;
跳转语句	EXIT	EXIT;
	CONTINUE	CONTINUE;
	JMP	<标识符>: ... JMP <标识符>;
返回语句	RETURN	RETURN;
NULL 语句	;	

1

2

图 1-49 CODESYS ST 语言常用句式

1.1. 赋值语句

1）格式及功能

是结构化文本中最常用的语句之一，作用是将其右侧表达式产生的值赋给左侧的操作数（变量或地址），使用“:=”表示。

具体格式如下：

<变量>:=<表达式>;

【例】分别给两个布尔型变量赋值，bFan 置 TRUE，bHeater 置 FALSE。

```
VAR
bFan: BOOL;
bHeater:BOOL;
END_VAR
bFan:= TRUE;
bHeater:= FALSE;
```

通过使用“:=”赋值语句实现上述功能。

2) 使用中的注意事项

a) 数据类型的匹配。如果赋值操作符的两侧数据类型不同，应调用数据类型转换函数。例如:Var1 是Real 实数类型,Var2 是Int 整数类型,当Var1 赋值给Var2时,应调用INT_TO_REAL 的转换函数。例如:

```
Var1:= INT_TO_REAL(Var2);
```

函数和功能块控制语句用于调用函数和功能块。

1.2 函数控制语句

函数调用后直接将返回值作为表达式的值赋值给变量。例如:rVar1:=SIN(rData1);语句中，调用正弦函数SIN，并将返回值赋值给变量rVar1。其语句格式如下:

变量:=函数名（参数表）;

【例】函数控制语句示例。

```
rResult:=ADD (rData1, rData2); //使用 ADD 函数，将 rData1 加 rData2 的结果赋值给变量 rResult。
```

1.3 功能块控制语句

功能块调用采用将功能块名进行实例化实现调用，如Timer 为TON 功能块的实例名，具体格式如下: 功能块实例名:(功能块参数);

如果需要在ST 中调用功能块，可直接输入功能块的实例名称，并在随后的括号中给功能块的各参数分配数值或变量，参数之间以逗号隔开；功能块调用以分号结束。

例如，在结构化文本中调用功能块TON 定时器，假设其实例名为TON1 如下图:

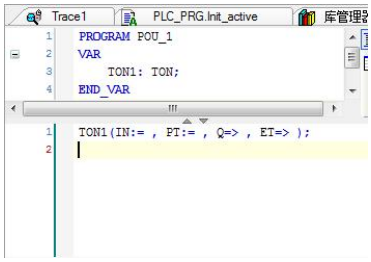


图 1-50 CODESYS 定时器功能块的调用

1.4. 选择语句

选择语句是根据规定的条件选择表达式来确定执行它所组成的语句。从大类上可分为 IF 和 CASE 两类。

1.4.1 IF 语句

用 IF 语句实现单分支选择结构，基本格式如下。

```
IF <布尔表达式> THEN
<语句内容>;
END_IF
```

如果使用上述格式，只有当<布尔表达式>为 TRUE 时，才执行语句内容，否则不执行 IF 语句的<语句内容>。语句内容可以为一条语句或者可以为空语句，也可以并列多条语句，该语句表达式执行流程图如下图所示。

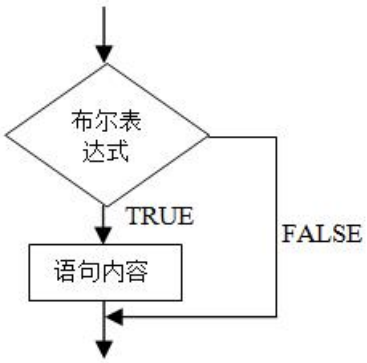


图 1-51 CODESYS if 语句

举个例子

使用 PLC 判断当前发动机温度是否超过了 60 摄氏度，如果超过，始终打开风扇进行散热处理，具体实现代码如下。

```
VAR
nTemp:BYTE; (*当前温度状态信号*)
bFan:BOOL; (*风扇开关控制信号*)
END_VAR
************************************************
IF nTemp>60 THEN
bFan:=TRUE;
END_IF
```

注意：上述写法，在循环任务中，如果温度大于 60 那么风扇开始运转，如果随着风扇的运转温度降低到 60 以下，此时风扇会停止吗？即 bFan 的状态在小于 60 时会编程 false 吗？

1.4.2 IF…ELSE 语句

用 IF 语句实现双分支选择机构，基本格式如下：

```
IF <布尔表达式> THEN
<语句内容 1>;
ELSE
<语句内容 2>;
END_IF
```

如上表达式先判断<布尔表达式>内的值，如果为 TRUE，则执行<语句内容 1>，如为 FALSE，则执行<语句内容 2>，程序执行流程图如图 4.35 所示。

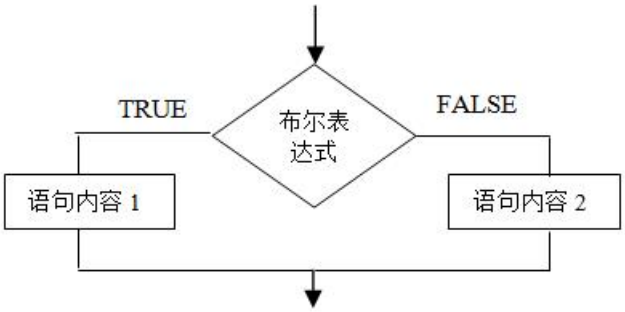


图 1-52 CODESYS if..else 语句

举个例子：使用 PLC 判断当温度小于到 20 摄氏度时，开启加热设备，否则（温度大于等于 20 摄氏度）加热设备断开状态。

```
VAR
nTemp:BYTE; (*当前温度状态信号*)
bHeating:BOOL; (*加热器开关控制信号*)
END_VAR
**********************************************************
IF nTemp<20 THEN
bHeating:=TRUE;
ELSE
bHeating:=FALSE;
END_IF
```

此句式适用于只有两种状态切换的场景，在条件 1 满足时，处于一个状态，在条件 1 不满足时，

处于另一个状态。

1.4.3 if 语句嵌套

当程序的条件判断式不止一个时，此时，需要再一个嵌套的 IF…ELSE 语句，即多分支选择结构，基本格式如下。

```
IF <布尔表达式 1> THEN
IF <布尔表达式 2> THEN
<语句内容 1>;
ELSE
<语句内容 2>;
END_IF
ELSE
<语句内容 3>;
END_IF
```

如上，在 IF…ELSE 中有放入了一个 IF…ELSE 语句，实现嵌套。

如上表达式先判断<布尔表达式 1>内的值，如果为 TRUE，则继续判断<布尔表达式 2>的值，如果<布尔表达式 1>的值为 FALSE，则执行<语句内容 3>，回到<布尔表达式 2>判断，如果<布尔表达式 2>为 TRUE，则执行<语句内容 1>，反之则执行<语句内容 2>。

【例】当设备进入自动模式后，如实际温度大于 50 摄氏度时，才开启风扇并关闭加热器，小于等于 50 摄氏度时关闭风扇打开加热器，如在手动模式时，加热器风扇均不动作。

```
VAR
bAutoMode: BOOL; (*手/自动模式状态信号*)
nTemp:BYTE; (*当前温度状态信号*)
bFan:BOOL; (*风扇开关控制信号*)
bHeating:BOOL; (*加热器开关控制信号*)
END_VAR
*********************************************
IF bAutoMode=TRUE THEN
IF nTemp>50 THEN
bFan:=TRUE;
bHeating:=FALSE;
```

```
ELSE
bFan:= FALSE;
bHeating:= TRUE;
END_IF
ELSE
bFan:= FALSE;
bHeating:=FALSE;
END_IF
```

1.4.4 IF..ELSIF..ELSE 语句

具体格式如下：

```
IF <布尔表达式 1> THEN
<语句内容 1>;
ELSIF <布尔表达式 2> THEN
<语句内容 2>;
ELSIF <布尔表达式 3> THEN
<语句内容 3>;
. . .
ELSE
<语句内容 n>;
END_IF
```

如果表达式<布尔表达式 1>为 TRUE，那么只执行指令<语句内容 1>，不执行其它指令。否则，从表达式<布尔表达式 2>开始进行判断，直到其中的一个布尔表达式为 TRUE，然后执行与此布尔表达式对应语句内容。如果布尔表达式的值都不为 TRUE，则只执行指令<语句内容 n>，程序执行流程图如图 1-53 所示。

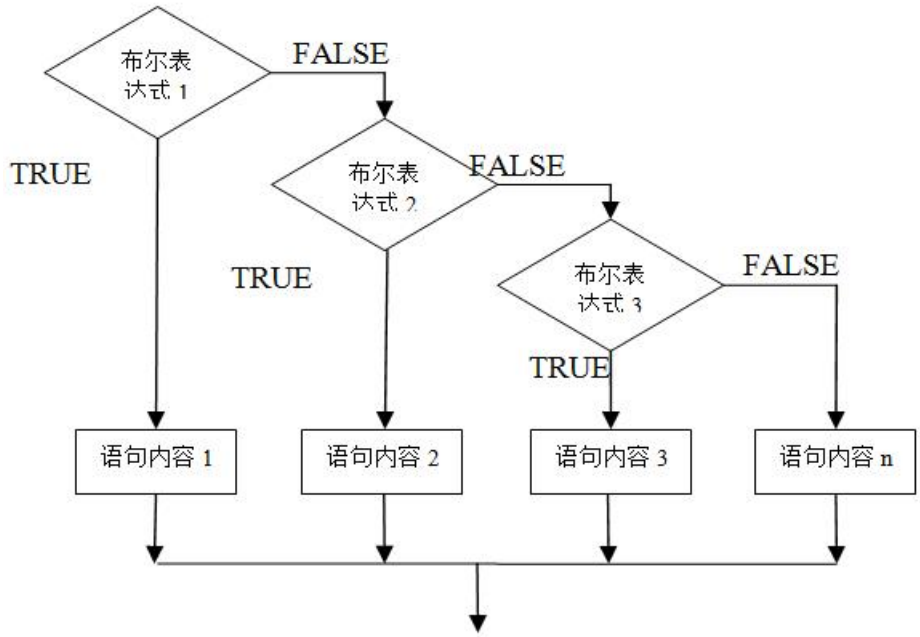


图 1-53 CODESYS if..elsif..else 语句

1.4.5 CASE 语句

CASE 语句是多分支选择语句，他根据表达式的值来使程序从多个分支中选择一个用于执行的分支，基本格式如下。

CASE <条件变量> OF

<数值 1>: <语句内容 1>;

<数值 2>: <语句内容 2>;

<数值 3, 数值 4, 数值 5>: <语句内容 3>;

<数值 6 .. 数值 10>: <语句内容 4>;

...

<数值 n>: <语句内容 n>;

ELSE

<ELSE 语句内容>;

END_CASE;

CASE 语句按照下面的模式进行执行：

如果<条件变量>的值为<数值 i>，则执行指令<语句内容 i>。

如果<条件变量>没有任何指定的值，则执行指令< ELSE 语句内容>。

如果条件变量的几个值都需要执行相同的指令，那么可以把几个值相继写在一起，并且用逗号分开。这样，共同的指令被执行，如上程序第四行。

如果需要条件变量在一定的范围内执行相同的指令，可以通过写入初、终值，以两个点分开。这样，共同的指令被执行，如上程序第五行。

【例】当前状态为 1 或 5 时，设备 1 运行和设备 3 停止；状态为 2 时，设备 2 停止和设备 3 运行；如当前状态在 10 至 20 之间，设备 1 和设备 3 均运行，其他情况时要求设备 1，2，3 均停止，具体实现的代码如下：

```
VAR
nDevice1,nDevice2,nDevice3:BOOL; (*设备 1..3 开关控制信号*)
nState:BYTE; (*当前状态信号*)
END_VAR
*****************************************************************
CASE nState OF
1, 5:
nDevice1 := TRUE;
nDevice3 := FALSE;
2:
nDevice 2 := FALSE;
nDevice 3 := TRUE;
10..20:
nDevice 1 := TRUE;
nDevice 3:= TRUE;
ELSE
nDevice 1 := FALSE;
nDevice 2 := FALSE;
nDevice 3 := FALSE;
END_CASE;
```

CASE 语句流程图见图 4.37 所示，当 nState 为 1 或者 5 时，设备 1 开，设备 3 关；

nState 为 2 时，设备 2 关，设备 3 开；

nState 为 10~20 时，设备 1 关，设备 3 开；

其他情况则设备 1 关，设备 2 关，设备 3 关。

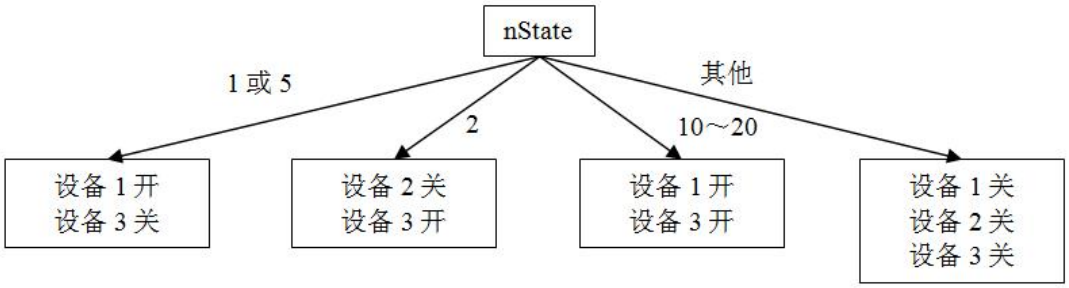


图 1-54 CASE 语句例流程图

上述例程还可以采用另一种变化写法，从而实现 case 语句中，不同情况之间的顺序跳转。

```
CASE nState OF
1, 5:
nDevice1 := TRUE;
nDevice3 := FALSE;
nState : =15;
2:
nDevice 2 := FALSE;
nDevice 3 := TRUE;
nState : =100;
10..20:
nDevice 1 := TRUE;
nDevice 3:= TRUE;
nState : =2;
ELSE
nDevice 1 := FALSE;
nDevice 2 := FALSE;
nDevice 3 := FALSE;
nState : =1;
END_CASE;
```

上述写法可以实现 case 情况下，不同情况之间的按自己的需求进行顺序跳转，可以拷贝到 codesys 看一下效果。

其它：while 语句，for 循环等等，根据工程机械目前做的项目来看用的少一些，自行研究吧。

1.5、FB 功能块的定义和写法

我们在整车的电液系统中，最常见的模拟量传感器，AB 项及总线型编码器等等，例如：温度传感器，油位传感器，压力传感器，编码器等，他们本身都有自己的电气参数和量程范围，比如 4-20Ma 的信号对应 0-600bar 的压力；又或者编码器一圈 500 个脉冲对应 360 度等等。这些信号在进入控制器后均需要通过 AD 转化转换为模拟量输入 ADC 或者计数器的脉冲数 SUM。这个时候我们一般会采用一个自己编写的 FB 来进行封装，方便同一台设备，多个同类型传感器之间直接调用。

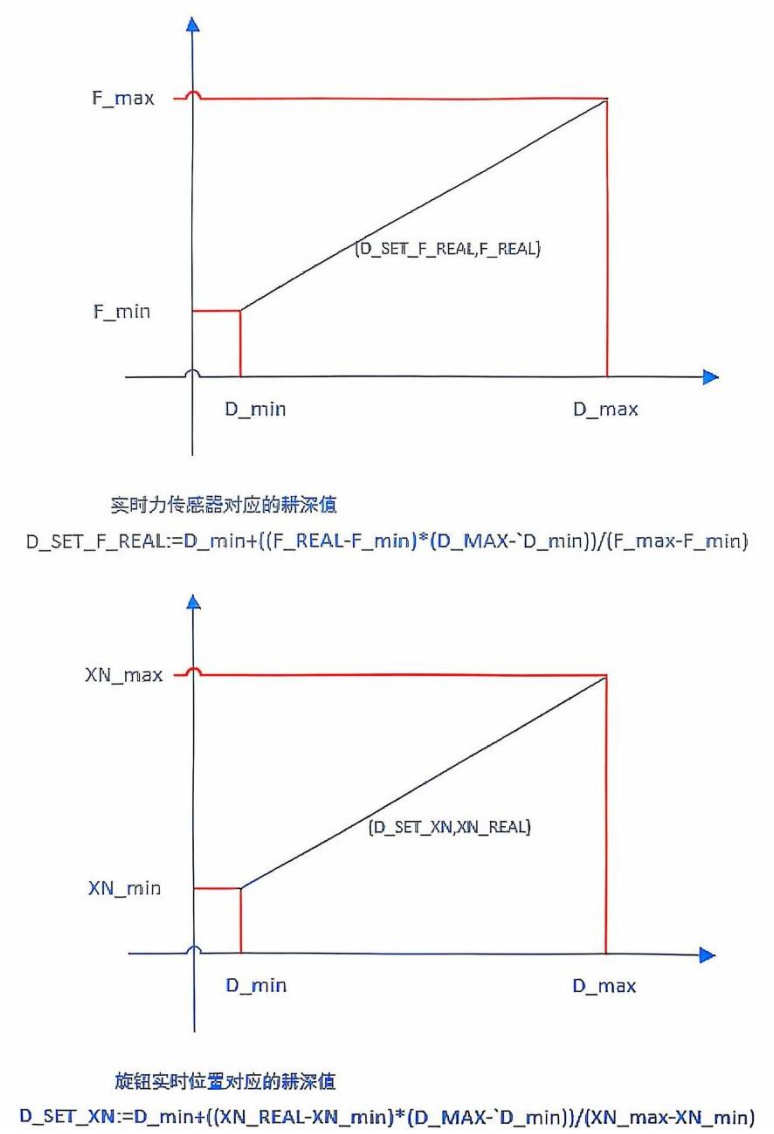


图 1-55 模拟量 FB 线性 ADC 和量程对比图

模拟量读取的本质如上图，上图体现的是信号和传感器物理值之间是线性对应的关系。

其对应的模拟量读取代码举例如下：

```
FUNCTION_BLOCK ANALOG_FB
VAR_INPUT
```

```
    AI_ADC_MIN:WORD:=0; //对应传感器的 ADC 的最小值
    AI_ADC_MAX:WORD:=4096; //对应传感器的 ADC 的最大值
    AI_LC_MIN:INT; //对应传感器的量程最小值
    AI_LC_MAX:INT; //对应传感器的量程最大值
    AI_ADC_REAL:WORD;//ADC 模拟量实时输入值
END_VAR
VAR_OUTPUT
    AI_LC_REAL:WORD;
END_VAR
VAR
END_VAR
// ad 值随着量程增大而线性增大的函数
IF AI_ADC_REAL>AI_ADC_MIN AND AI_ADC_REAL<AI_ADC_MAX THEN
AI_LC_REAL:=((AI_ADC_REAL-AI_ADC_MIN)*((AI_LC_MAX-AI_LC_MIN))/(AI_ADC_MAX-AI_ADC_M
IN))+AI_LC_MIN;
ELSE
AI_LC_REAL:=0;
END_IF
```

以上这种功能块在电液系统传感器的输入处理部分会经常用到，需要掌握。

注意事项：在自己编写一些依据数学的抛物线或者线性函数为公式的算法时，一定要注意输入变量赋初始值，以免程序运行时，由于初始值为 0，而 0 正好被作为除数，导致程序陷入死循环。而这种 BUG 在 codesys 编译过程中是无法发现的。

1.6 针对电液系统同一个 PWM 控比例电磁阀，在不同功能和应用场景下，该电磁阀的控制电流写法一般参考一下句式：

[例]行走（前进），回转（左转），起升（上升）等这些动作时均需要泵控比例电磁阀 Y101 得电，而且不同动作下，Y101 得电情况不同。

那么如果按照一般逻辑顺序去写的话，如下

```
Var:
MOVE_F: BOOL; //行走前进
SLEW_S: BOOL:
```

```
RIASE_UP: BOOL;
Y101: word;
End_VAR
```

常规的入门直观 ST 写法如下：

```
Line1 If Move_F then
Line2 Y101:=800;
Line3 End_if
Line4 If SLEW_S then
Line5 Y101:=700;
Line6 End_if
Line7 If RIASE_UP then
Line8 Y101:=600;
Line9 End_if
```

如果按上述的写法的话 Y101 这个比例电磁阀在一个循环任务中，按行执行，就会出现在某一个周期内，先等于 800，在等于 700，最后等于 600。那针对不同功能下，对电流需求功能的不同，这样写不仅逻辑上不对，更是在同一个周期内，同一个比例阀输出端口给出了三次不同的电流值，而这三个电流值都和需求值不一样。

那么换一种写法：

分两个步骤来写这个流程，第一步分情况给比例阀中间变量电流值，最后统一对比动作优先级，再看比例阀需要输出多少。

```
Var:
MOVE_F: BOOL; //行走前进
SLEW_S: BOOL;
RIASE_UP: BOOL;
Y101: word;
Y101_1, Y101_2, Y101_3:WORD;
End_VAR
```

第一步不同动作下电流值分配

```
Line1 If Move_F then
Line2 Y101_1:=800;
```

```
Line3 End_if
Line4 If SLEW_S then
Line5 Y101_2:=700;
Line6 End_if
Line7 If RIASE_UP then
Line8 Y101_3:=600;
Line9 End_if
```

第二步，根据动作的优先级情况给 Y101 进行最终赋值。

假如是 Move_F 的优先级最高，SLEW_S 次之，RIASE_UP 最后，那么 Y101 最后得电情况如下：

```
If Y101_1>0 then
Y101:=Y101_1;
Elsif Y101_1=0 AND Y101_2>0 then
Y101:=Y101_2;
Elsif Y101_1=0 AND Y101_2=0 AND Y101_3>0 then
Y101:=Y101_3;
End_IF
```

上述这种写法，才能让循环任务执行更加完美。

1.7、手柄开关量按钮的长按短按用法

在工程机械设备中，手柄操纵杆是很多设备普遍采用的设备控制方式，而手柄上面往往会布置一些开关量和模拟量的旋钮。在一些特殊的工程应用中，可能会出现以下两种用法：

1）手柄操纵杆一个开关量按钮第一次按下，执行某一个动作；同一个按钮第二次按下，执行另一个动作；

关于上述开关量的用法，可以采用开关量计数和 SUM 除以 2，MOD（取余）为 1，认为第一次按下；取余为 0，为第二次按下；第三次按下，又是取余为 1；第四次按下，取余为 0；

2）手柄操纵杆一个开关量按钮短按或者点按，执行某一个动作，长按执行另一个动作；

关于长按和短按，我们可以根据程序的循环扫描周期（举例为 t#50ms），认为某一个开关量通电长超过 2s 为长按，低于 50ms 为点按；即：当遇到针对某一个开关量按钮的长按和短按的判断场景，我们可以根据按钮被按下的时间长短作为判断条件进行区分。

1.6、CODESYS 的 CAN 通信

1.6.1 CANBUS 总线网络的概念

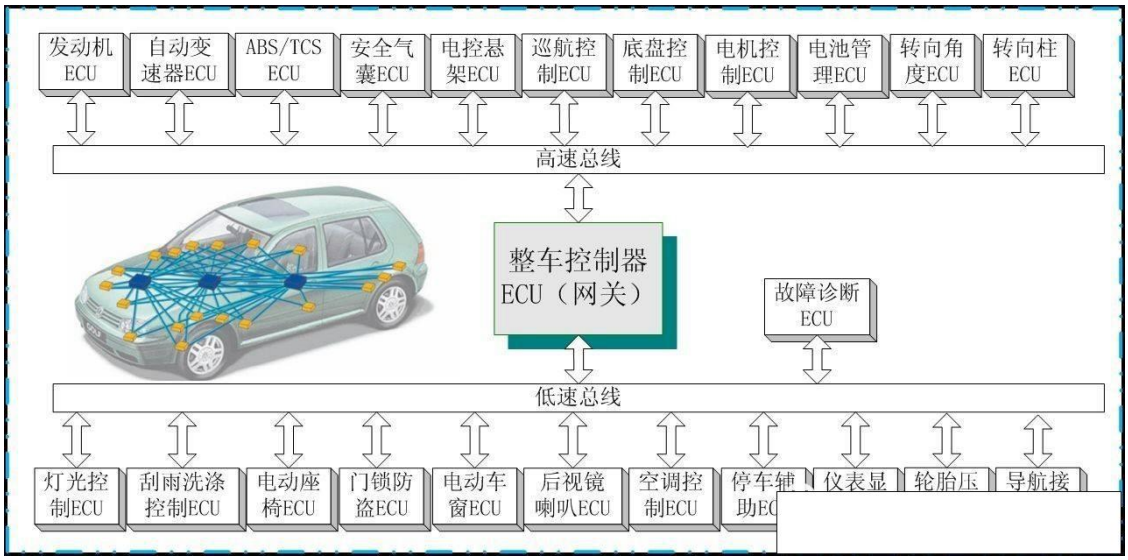


图 1-56 can 网络示例

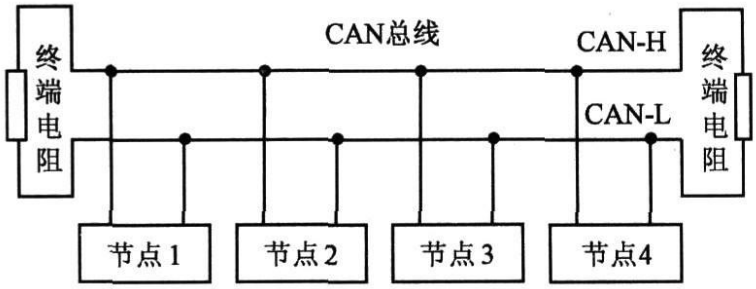


图 1-57 can 网络示例

Can 总线硬件线路参数，当总线通信失联，异常时从以下 4 个维度进行排查硬件问题

1、断电测总线之间电阻，标准 60 欧

2、近端和远端各一个终端 120 欧姆电阻

3、通电测总线 can_H 电压，一般情况 CAN_H 比 CAN_L 高 1V 左右

4、can 总线物理层一般采用屏蔽双绞线，降低干扰

1.6.2 CAN 报文的几个知识点

图 1-58 can 报文结构 1

一个完整的数据帧有 7 部分组成，依次为帧起始(SOF)、仲裁场(Arbitration Field)、控

制场(Control Field)、数据场(Data Field)、CRC 场、应答场(ACK Field)、帧结尾(EOF)。

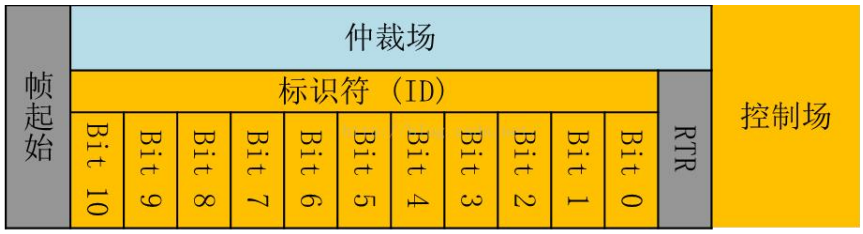


图 1-59 can 报文结构 2

图 1-60 can 报文结构 3

object	*function code (binary)*	*resulting COB-ID*	*Communication Parameters at Index*
NMT	0000	0	-
SYNC	0001	128(80h)	1005h, 1006h, 1007h
TIME STAMP	0010	256(100h)	1012h, 1013h
EMERGENCY	0001	129(81h)-255(FFh)	1014h, 1015h
PDO1(tx)	0011	385(181h) - 511(1FFh)	1800h
PDO1(rx)	0100	513(201h) - 639(27Fh)	1400h
PDO2(tx)	0101	641(281h) - 767(2FFh)	1801h
PDO2(rx)	0110	769(301h) - 895(37Fh)	1401h
PDO3(tx)	0111	897(381h) - 1023(3FFh)	1802h
PDO3(rx)	1000	1025(401h) - 1151(47Fh)	1402h
PDO4(tx)	1010	1153(481h) - 1279(4FFh)	1803h
PDO4(rx)	1100	1281(501h) - 1407(57Fh)	1403h
SDO(tx)	1011	1409(581h) - 1535(5FFh)	1200h
SDO(rx)	1100	1537(601h) - 1663(67Fh)	1200h
NMT Error Control	1110	1793(701h) - 1919(77Fh)	1016h, 1017h

图 1-61 can PDO 的范围

从上图可以看出，can 通信中，标准 COB-ID 的取值范围是从 181H-57FH。

CANOpen 中的实时数据传输是由 PDO 来完成的。共有两种 PDO，TPDO 和 RPDO。

TPDO 用来传输数据，支持 TPDO 的节点都是 PDO 数据的生产者。RPDO 用来接收 PDO 数据，支持 RPDO 的节点是 PDO 数据的消费者。一个节点最多支持 4 个 TPDO

（分别是 180h+NodeID、280h +NodeID、380h +NodeID、480h +NodeID）和 4 个 RPDO（分别是 200h +NodeID、300h +NodeID、400h +NodeID、500h +NodeID）。

每一个 PDO 都对应一些参数，包括通讯参数和映射参数。

Index	*Sub-Index*	*Description*	*Data Type*
1400h to 15FFh 1800h to 19FFh	00h	Number of entries	Unsigned8
	01h	COB-ID	Unsigned32
	02h	Transmission type	Unsigned8
	03h	Inhibit time	Unsigned16
	04h	Reserved	Unsigned8
	05h	Event timer	Unsigned16

图 1-62 通信参数索引和子索引范围

Index	*Sub-Index*	*Description*	*Data Type*
1600h to 17FFh 1A00h to 1BFFh	00h	Number of mapped objects in PDO	Unsigned8
	01h	1st object	Unsigned32
	02h	2nd object	Unsigned32
	03h	3rd object	Unsigned32
	...		
	40h	64th object	Unsigned32

图 1-63 映射参数索引和子索引范围

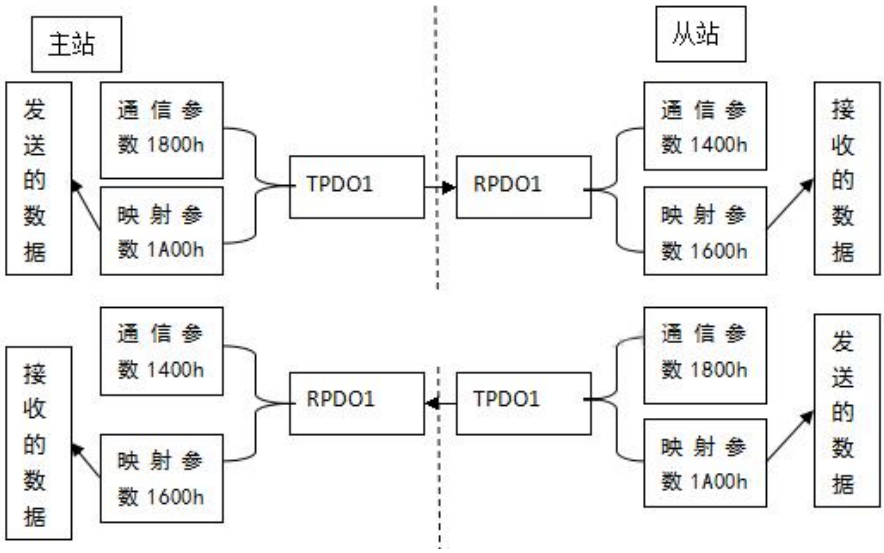


图 1-64 TPDO 的发送和 RPDO 接收过程

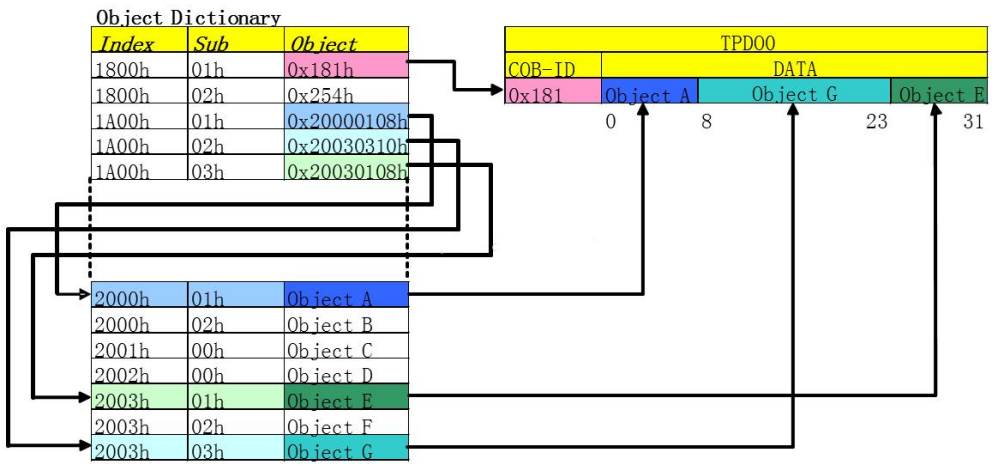


图 1-65 TPDO 的封装过程

通信参数和映射参数到 OD 中提取对应数据，最后封装到 TPDO 后发送出去

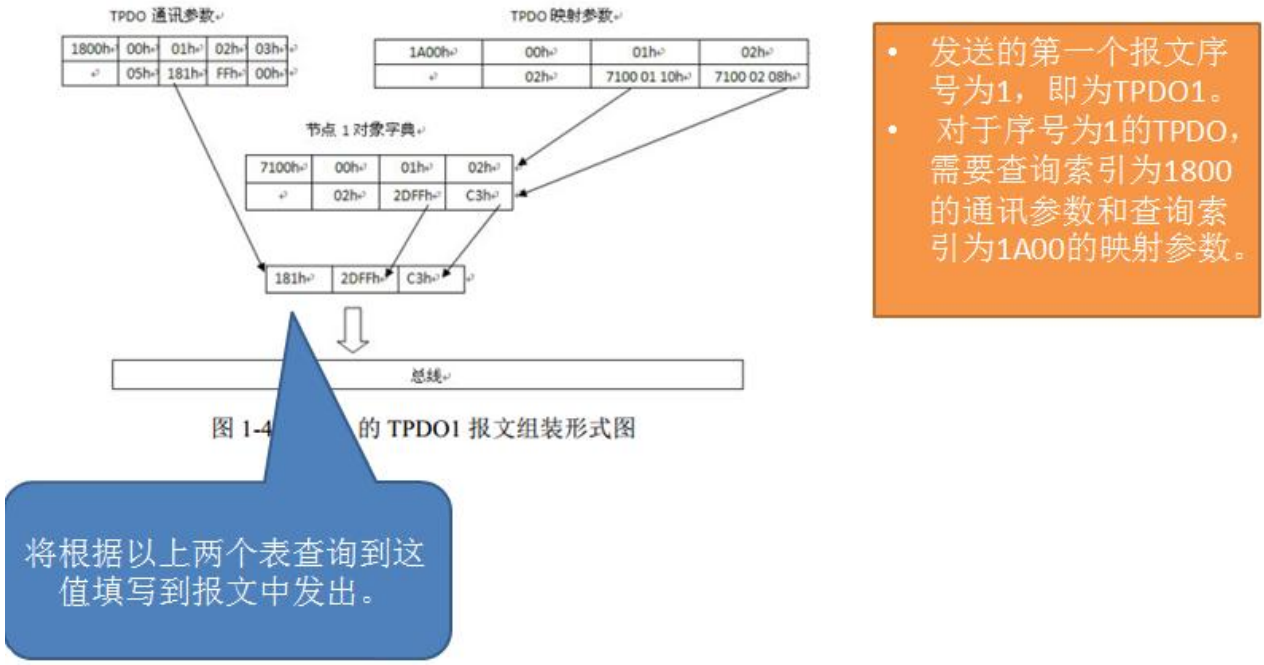


图 1-66 TPDO 的发送过程

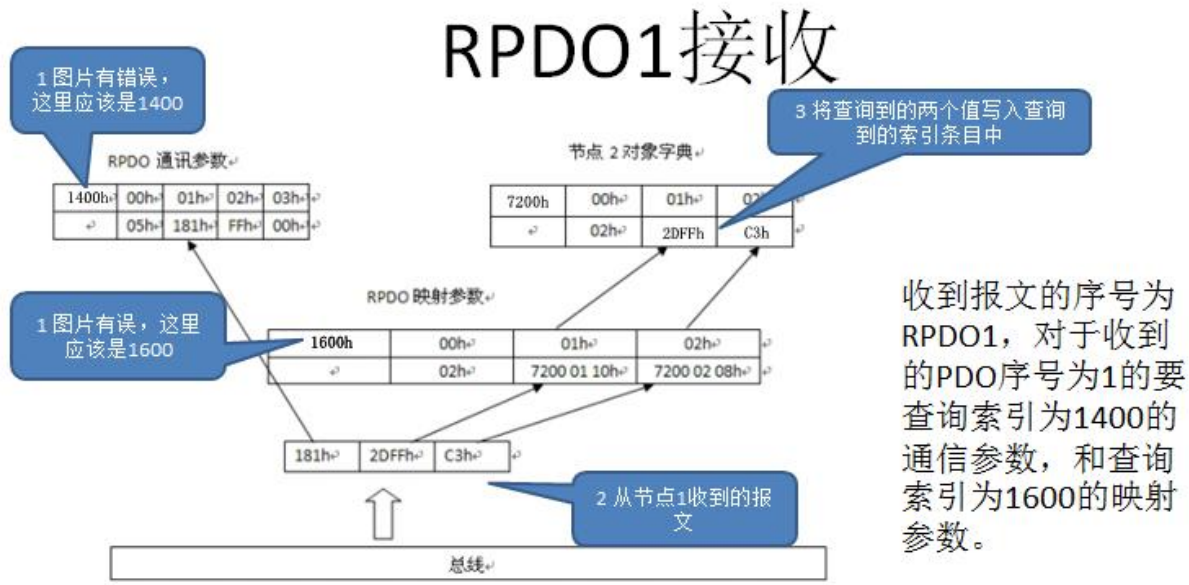


图 1-67 RPDO 的接收过程

了解到通信参数和映射参数就不得不提一下 OD。

对象字典（OD：Object Dictionary）是 CANopen 协议核心部分，是一个有序的对象组；每个对象采用一个 16 位的索引值来寻址，为了允许访问数据结构中的单个元素，同时定义了一个 8 位的子索引，对象字典的结构参照下表。不要被对象字典中索引值低于 0x0FFF 的‘data types’

项所迷惑，它们仅仅是一些数据类型定义。***一个节点的对象字典的有关范围在 0x1000 到 0x9FFF 之间***。

我们可以把 OD 简单的理解为 can 通信中 PDO 所包含和传递信息的在硬件存储中的一个寻址通信协议，就是我们可以通过 OD 的索引和子索引来锁定其具体某一个 ID 所承载的信息。

Index (hex)	*Object*
0000	not used
0001-001F	静态数据类型
0020-003F	复杂数据类型
0040-005F	制造商定义的复杂数据类型
0060-007F	设备子协议定义的静态数据类型
0080-009F	设备子协议定义的复杂数据类型
00A0-0FFF	保留
1000-1FFF	通讯子协议区域
2000-5FFF	制造商定义的子协议区域
6000-9FFF	标准设备子协议区域
A000-BFFF	标准接口子协议区域
C000-FFFF	保留

图 1-68 对象字典索引的范围及定义

从上图可以看出，我们在一个项目程序开发过程中，所编制的通信协议，实际上是定义的索引 1000-1FFF 通讯子协议区域。从 pdo，sdo 指令中我们关心的数据帧到其对应的索引和子索引定义和范围，再到 OD，实际上是相当于 can 通信过程中从物理层到数据链路层再到 OD。

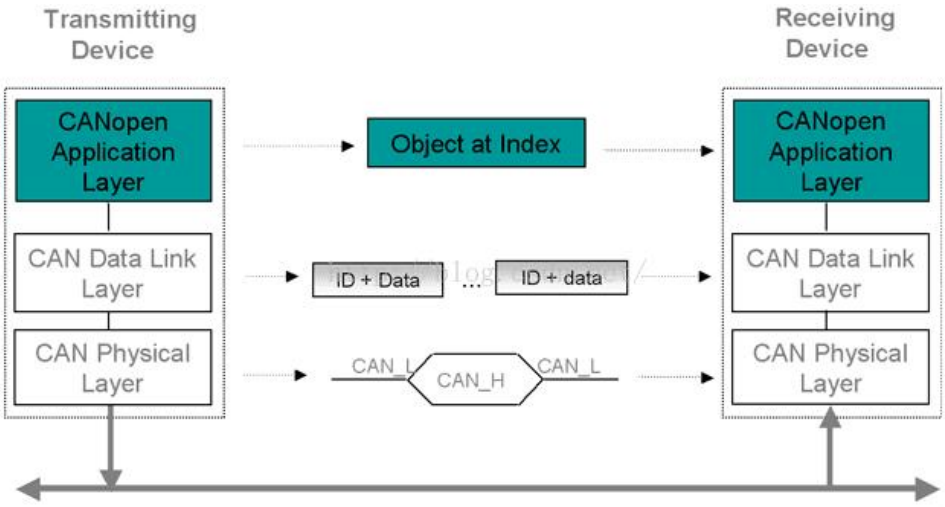


图 1-69 CANOPEN 网络分层及不同节点数据收发对应

由下到上，物理层，数据链路，应用层，分别对应的 can 的信号，报文 PDO，OD。

上图可以可看出在 can 网络中，不同节点之间各层级对应关系。

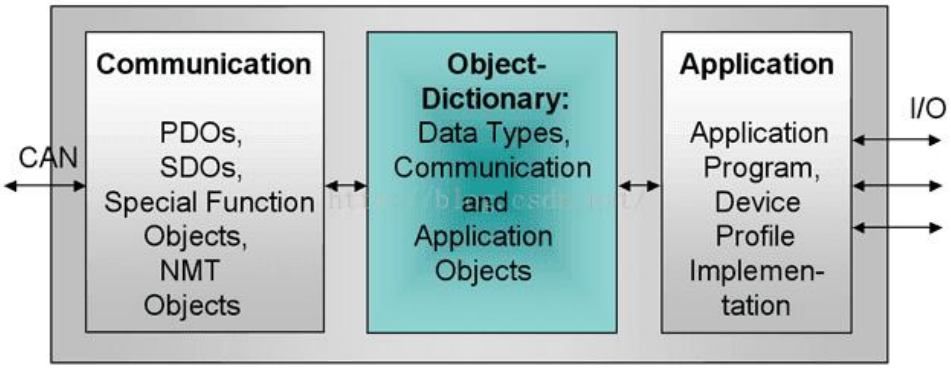


图 1-70 同一设备中从 can 收发到 od 到应用层程序再到 IO

上图中的关系，可以看出单个设备中 can 收发报文印射到 OD 列表以及应用层的对应关系。

OD 定义中，已经明确定义了 1000-1FFF 索引对应的内容，此处可以了解一下，为了方便熟悉芬兰 EPEC 控制器所使用 multitool 软件中 can 配置的 OD 配置。

Index range 索引范围	Description 描述
1000_h to 1029_h	General communication objects 通用通讯对象
1200_h to $12FF_h$	SDO parameter objects SDO 参数对象
1300_h to $13FF_h$	CANopen safety objects 安全对象
1400_h to $1BFF_h$	PDO parameter objects PDO 参数对象
$1F00_h$ to $1F11_h$	SDO manager objects SDO 管理对象
$1F20_h$ to $1F27_h$	Configuration manager objects 配置管理对象
$1F50_h$ to $1F54_h$	Program control object 程序控制对象
$1F80_h$ to $1F89_h$	NMT master objects 网络管理主机对象

图 1-71 1000-1FFF 索引范围对应的数据内容

Index 索引	Object 对象	Name 名字
1000_h	VAR 变量	Device type 设备类型
1001_h	VAR 变量	Error register 错误寄存器
1002_h	VAR 变量	Manufacturer status register 制造商状态寄存器
1003_h	ARRAY 数组	Pre-defined error field 预定义错误场
1005_h	VAR 变量	COB-ID Sync message 同步报文 COB 标识符
1006_h	VAR 变量	Communication cycle period 同步通信循环周期（单位 us）
1007_h	VAR 变量	Synchronous windows length 同步窗口长度(单位 us)
1008_h	VAR 变量	Manufacturer device name 制造商设备名称
1009_h	VAR 变量	Manufacturer hardware version 制造商硬件版本
$100A_h$	VAR 变量	Manufacturer software version 制造商软件版本
$100C_h$	VAR 变量	Guard time 守护时间（单位 ms）
$100D_h$	VAR 变量	Life time factor 寿命因子（单位 ms）
1010_h	VAR 变量	Store parameters 保存参数
1011_h	VAR 变量	Restore default parameters 恢复默认参数
1012_h	VAR 变量	COB-ID time stamp 时间报文 COB 标识符（发送网络时间）

图 1-72 OD 中缺省的标准索引定义 1

Index	Name	Object Code	Data Type	M/O
1008h	manufacturer device name	VAR	Visible String	Optional
1009h	manufacturer hardware version	VAR	Visible String	Optional
100Ah	manufacturer software version	VAR	Visible String	Optional
1018h	Identity Object	RECORD	Identity	Mandatory

图 1-73 OD 中缺省的标准索引定义 2

Sub-Index	Description	M/O	Access	PDO Mapping	Value Range	Default Value
0h	number of entries	Mandatory	ro	No	1…4	No
1h	Vendor ID	Mandatory	ro	No	UNSIGNED32	No
2h	Product code	Optional	ro	No	UNSIGNED32	No
3h	Revision number	Optional	ro	No	UNSIGNED32	No
4h	Serial number	Optional	ro	No	UNSIGNED32	No

图 1-74 OD 中缺省的标准索引定义

1008h 记录设备的名字，1009h 记录设备的硬件版本号，100Ah 记录设备软件版本号。1018h 记录了设备的制造商代码，产品号，序列号等。

1.6.3 CAN 通信在 codesys 中的配置方式

在工程机械领域，国内外不同品牌的控制 PLC 控制器，其 can 通信配置的方式常见的有三种。

- 通过资源里面增加“参数管理”，定义索引和子索引，关联程序变量，最后映射到 PDO，如国内威卡（原赫斯曼）。
- 通过PLC厂家提供的PLC型号为名称的库,库里面一般包含can_init,can_send,can_receive等函数。（V3 版本硕博，嘉城，禾晟等，V2 版本 TTC，PMA）。
- 通过专用的第三方 can 配置软件来进行 can 通信设定，比如：epec 厂家的 multitool 工具；原赫斯曼的 itoo5 工具。

1.6.3.1 威卡-iflexC3-can 配置（举例子用）

iFLEX C3 控制器同时支持 CAN 2.0A 和 CAN2.0B 两种通讯协议。

通过 CoDeSys 软件中的 PLC Configuration 功能，我们能够很方便的添加 CAN 通讯设备。

添加一个支持 CAN2.0A 通讯协议的设备

打开 CoDeSys 软件，进入 PLC Configuration 界面：