0115 【万泉河】PLC 编程中的常量

PLC 中的常量，同样是我所不喜欢用的。
因为与我抨击过多次的全局变量和 UDT 一样，它们都是一种全局资源。
然而， 因为这个观点又是在整个 PLC 行业中独一无二的首发，以前未见到其他人发表过，所以势必又会惊掉一大堆人的下巴，所以需要先把这其中的道理掰扯清楚。
以往我发表的很多类似技术观点的文章发表后， 总会出现各种不同的声音。
其中一种是指责我故意制造争论，哗众取宠。
哗众取宠这种罪责从来都是完全没有逻辑的。 技术的问题， 要么对， 要么错。 如果你不能证明错误，而只是因为自己没有相关的认知那么至少暂时你也应该暂时搁置，给自己留个学习认证的时间，而不是第一时间就加以否定， 甚至还什么我的目的是要取宠你。
读者有什么好取宠的呢？除非我能带给读者真正的知识，让他有思考有收获，他会感激我。初次之外，我仅仅惊吓了他们一跳，他凭啥就要宠我，我还取宠呢！

还有一种观点是倒过来质疑我不会用我文章提到的这个技术点。 我讲不用 UDT 的时候怀疑我不会用 UDT，我讲不用循环的时候怀疑我不会用循环，我讲不用 IO 映射的时候怀疑我不会用 IO 映射，我讲不用全局变量的时候怀疑我不会用全局变量。总之，我不用啥都不是因为这个方法有啥缺点，而是因为我不会用。 然后我写出文章来正好撞到了会用的他。

就不想想，我不会的东西可多了去了。 天文地理军事体育，天底下的知识技能包罗万象，其中的大部分都是我不懂的。 那么多我不懂的内容我为啥不去写挨着写文章发表观点， 凭啥就撞到了你特长的枪口上发表观点呢？

真实的情况应该是， 那些我不懂的东西我应该是它们的名字都叫不上来，甚至都不了解它们的存在。 我既然能拿这些作为题目写出一篇小作文来，而且能准确地踩到一大批人的脉搏，大概率当然应该是我会的。就因为我会，我才有能力拿它们开刀开涮。而且所理解的深度比刚刚读到这个观点的你更深刻。

亦或是， 如果你觉得你也可以凭自己不懂，就可以写文章， 那不妨写一两篇出来试试。 我相信你如果愿意写，很快就能写上几万篇文章，积累成专辑出版传世后人。

而且，咱说的这些问题，普遍都是需要掌握的非常低的技能，断然连门槛都算不上。 比如全局变量， 你说会用全局变量还需要什么技能？相反，会不用， 能不用才是技能！

当然，还有一些人，在用上述观点跟我周旋良久占不到便宜后，还会使出最后的绝技：人家西门子， CODESYS， 三菱等系统平台，既然设计了此功能， 你为啥不让我用呢？

这种通常就是属于没有逻辑的人群中的下下限，通常到此为止我就没必要再与这样的人进行技术话题的探讨了。

就好比，我在分享工程师如何做到出差不带编程电缆万用表的经验知识，他来抬杠我家里有，凭什么就不能带？我讲可以出差不带电脑，他来杠不带电脑拿什么干活？讲如何做到乘坐商务仓而不去挤经济舱，他来杠飞机上有经济舱你凭啥不让人坐？要不要你去建议飞机制造商

把经济舱座位全部取消了！

所以一定要注意，我在讲到不使用这个不使用那个的时候，我说的是我自己做到了不用，我从来没有卡住别人的脖子不许别人使用。 甚至连对烟台方法的学员，我都没有这个权力。其次，我在讲这些观点的时候，都是有前提条件的，就是在做标准化架构的模块的时候的最优解。 如果不求最优，或者根本不在模块化范围内，那也是无关的。

要说卡脖子，我也只是卡住了我自己的脖子，并且还秀给别人看：我不用这个材料，也不用那个材料，却仍然做出了标准化架构烟台方法（6 年前）。围观群众纵然没有机会亲自领略其神奇，但可以通过排除法逐渐逼近真相。因为首先一个前提是可以保证的：真实。

因为但凡我有丁点撒谎， 比如如果我明明在自己的项目程序中使用了全局变量或者常量，却还写文章来探讨这个不用那个不用，那就等于把命门暴露给了别人，分分钟等待被学员给揭发，扒掉了底裤。

而从利益的角度，如果我在用的东西，我也完全没必要编瞎话再给自己无端增加事端，我就保持我既有的技术架构，继续出售我的技能知识，都足够。 所以我写这些文章更多的意义在于启发已有的烟台方法的学员，从我交付的资料中发现这些细节的闪光点。 我只是在对自己已经做到的技术方法持续做出总结而已。比如在我写本文发表本文的观点之前， 那些学习了 3-5 年的烟台方法学员，恐怕未必自己能从烟台方法的案例中总结归纳到。

当然，未来的学员提前有了这方面的准备预期，真正入手之后方向也会更明确。

回到文章的开始我把常量和全局变量都比作全局资源，一定有人不能理解。 全局变量 M 是有地址范围的，有可能会用光，不同的 CPU，其 M 空间不一样多。而常量是不需要地址的，怎么也算全局资源呢？

其实，我在最早的讲不用全局变量的文章中，就已经提到过了，比方 AB PLC，它是没有 M 的概念的。 所有的全局变量没有地址，你声明一批 BOOL 或者 INT 等数据类型的变量名，然后就全局使用了。 我们指的是它们，符号化的全局变量。 而平常用 M 来泛指，只是为了表达方便。

所以，即便是符号命名的变量或者常量，或者 UDT，它们的名字也是全局资源，也是可能发生冲突，而只要有发生冲突的可能， 那么不管这个可能性的概率有多低，在制作标准的模块库时都应该尽量避免。 比如我自己，就从来不用。当然读者可以质疑我经历过的项目都还不够复杂， 也确实我目前也遇到的一些功能使用了常量，而我还没找到更好的避免的方法，也仍然在探索中。 文章的最后会提及。

有的人搞不懂我为啥要对冲突如此敏感，尤其是这种可能性极小的冲突。做一个约定，事先规定一个规则，把所有的功能模块都约定在规则之下，不就好了嘛！

比如我在讲不用 M 的时候，就有博主表示不以为然，并大秀智商给了一个高级解决方案：提前规划。 即针对不同的应用和功能，预先分配好不同的 M 地址区域， 不就好了嘛！

然而，有没有想过，如果你通过规划的约束，不管是一个工程项目，还是一套架构方案，那么在规划诞生之前所做的技术储备就全部不能直接拿来使用了。 即便不会被作废，也需要在使用之前做冲突检查。 那所谓的封装和复用的模块化的效果就大打了折扣。

而且，这种规划并没有一个天然的核心，并不是你或者我提出了这种规划大法，我或者你就拥有了天然的主持规划的话语权， 而是所有人都可以主持规划， 那么导致的结果是人人都可以做规划主任， 每个人的规划方案当然会有差异， 那么即便假设有人甘心做小弟，不谋求做规划主任，而只是被动执行规划，也会因为有多个不同的主心骨而无所适从。比如 A 主导了一个平台架构， B 主导了另一个平台架构，而 C 作为小弟，想贡献一套自己做的算法模块的时候，还需要分别调整，根据 AB 各自的规则调整之后，才可以投稿。除非他在做这套算法模块的时候没有使用到任何全局资源，模块是完全内聚的，对资源没有任何要求和约定。 如我一直在号召并践行的。 这就是为什么在 GITHUB 上面关于 PLC 的库分享还完全不成气候的原因。 连规则方法都还没达成一致的行业，怎么可能有共享的生态环境呢？

下面我们在 TIA PORTAL 中演示下使用 CONST 常量会带来的危害。当然顺便也演示使用了 UDT, 用户数据类型。

新建一个 S7-1200 的项目， 在其中建立用户数据类型，就缺省的名字“用户数据类型_1”好了。建立了 2 条数据 DATA1 和 DATA2，分别为 BOOL，假设算法需要。

然后 PLC 变量表中的用户常量的页，建立一个 INT 类型的常量，名字为 MAX， 数值等于 99。通常使用常量的目的就是为了数值可以统一修改。将来需要的时候可以从 99 改为别的数值。

然后建立 FB 块， 在其静态变量中建立变量，一个数组，数组的上限是 MAX， 数据类型即为建立的 UDT。

这不就是你们使用常数的最大目的嘛!需要建立数组做循环,然而循环的上限提前不能确定,导致数组的上限也不能确定,然而几乎所有的编程语言都不允许数组的上限值不确定,不允许变化的变量， 只接受常量。所以就只好用常量来实现了。

后面的程序逻辑就不具体做了。 反正就是有了这样的数据接口和结构,最终在 FB 内实现了特定的算法功能,然后以为可以封装成一个固化的 FB 了,可以无限重复使用了。

然后把整个 PLC 复制一份,并修改自定义数据的结构,比如数据类型改为 2 个 INT,而 UDT 的名称不变。

同样也建立 FB， 定义数据,为了实现另外其它的算法功能。 其中 MAX 常量的数值做了修改,比如 10

好啦!我们现在假设手里有 2 套完全不同的库函数,分别实现了不同的控制算法功能。唯独它们之间好巧不巧使用了相同的名称, FB 的名称、自定义数据名称以及常量名称。当然这个巧合是我人为造出来的， 但你不可否认这种巧合当然存在。

现在,我们假设有第三个项目,需要用到上述的 2 个功能库， 所以需要把它们复制到同一

个 PLC 中。

FB 名称的重复很容易发现，也很容易处理。 复制的时候另外取个名字，也另外分配个 FB 编号即可。 当然 CODESYS 等环境下的 FB 没有编号，不需要。

然而如果重复使用这 2 个 FB 的不是原作者本人， 必然对数据结构和逻辑不够熟悉。复制之后再编译的时候，发现提示 UDT 和常数缺失，然后按照提示去源程序中复制。对于 FB1，没有问题，而对 FB2，则会发现出了问题。 UDT 结构定义变化了， 编译冲突，通不过了。

解决办法是，可以在复制之前，先将 UDT 的名称改掉，改为不重复，然后再复制，就可以不冲突了。
而常量，则并不会有提示，所以，如果你不晓得代码逻辑中有冲突使用的常量，就不会发现其中的数值被偷偷的改掉了，而你一无所知。 ----------这样，必然就留下了 BUG 隐患。

这就是我反对我自己使用全局常量的原因所在。 因为它破坏了封装的独立性和完整性，给原本调试完成的代码留下了一个缺口，有可能随时被干扰，而产生功能的不完整。

假设我是有偿库函数的提供者， 那么我自然要对库的功能完整负有服务义务，而这种因为常量数值变化导致的功能失效，我是没有脸面来指责用户使用不当的。 比如我咋样给自己找理由摆脱责任呢？你们对我的库不熟悉，你们系统里使用了不该用的常量？对方会回答：我们当然不熟悉，我们从头就不打算熟悉，而只想简单使用。我们如果熟悉了就自己做了，谁稀罕用你的库！

这还是在库函数并没有被加密的情况下。 如果指望把库函数加密，然后不给用户看到源代码，用户只能使用而不能盗用专利技术，那么使用了上述 UDT 和常量的库函数都是几乎不可能的。 因为用户随时需要编译，需要使用密码打开程序块。

所以总结：如果这个行业普遍习惯于大量使用全局常量数据来做所谓的标准化封装模块，那就永远不会有成熟稳定的库函数交易市场。 那么所有人，都只能永远自己闭门造车，自己做库自己用。 分工与专业化提高效率就永远没有可能。

有没有人注意到我在建立 FB 的时候，图中，在 FB 的常量区还顺手建立了一个 MAXX 的常量？如果你要使用的常量在这里，我们称之为局部常量数据的话，那是没有问题的。因为 FB 和 FB 之间不会发生冲突，随便使用都可以。 我本文针对的内容是全局常量， 而非局部。

然而，恐怕这又是那些习惯于使用全局常量的同行们所不能接受的。 因为你大概率的是会在多个 FB 中公用这一个常量。 即，你本来使用常量的目的也是为了在不同模块之间分享信息。 而这恰恰是我著本文提醒你需要更改提升方法的所在。

我当然清楚你们的习惯，以及技能方法。 并不是我不知道这种用法，而恰恰是我更清楚这种用法，并非常知晓这种用法的弊端，所以才会提出来主张。

这里也引申出来另一个道理，即一个人， 遇到对自己固有观念和习惯有冲击力的理论和方法的时候，应该持有怎样的态度。

是承认自己的未知状态，遇到的理论方法，哪怕暂时自己还不能接受不能实现，暂时做下记号，留待自己以后有机会时再图谋深入了解， 还是第一时间先否定，先找各种理由为自己做法的合理性做辩解，不管是发出声来还是自己默默内心给自己辩护？这都体现了一个人的学习能力和创新思维能力。对未知事物的拒绝保守态度，本质上是井底之蛙的逻辑。

我曾经研究实现了将 LBP 库函数从 S7-1500 移植到 S7-1200，对其中大量使用的 UDT 头痛不已。其整个系统中用到了 UDT 共有 130 多个，而且是互相耦合的，即要使用哪怕只有 1 个 FB， 所有的其它的 UDT 也必须复制过来，少一个都不行。 所以为了解耦它们，花费了许多的精力。

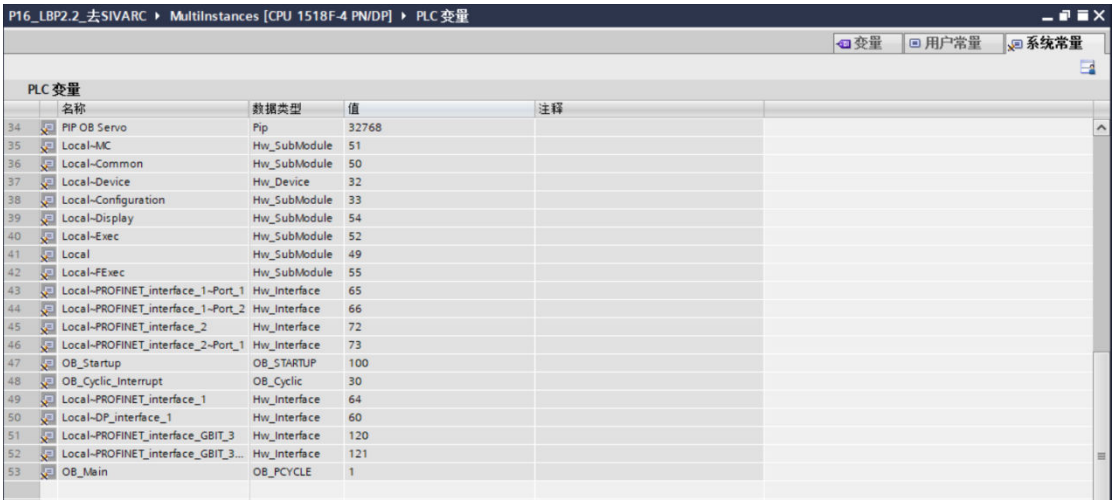
而在 LBP 中，常量的使用则比较少，只有一个：PANELS_NO=2 。
代表的是这台 PLC 通讯的触摸屏的数量等于 2 。 如果下一个项目，触摸屏数量为 3，则需要把这里的数值改为 3。而在整套库函数中， 这个常量到处用到，不管是循环语句还是数组定义的上限值，因为每一个对象的 FB 都需要处理触摸屏通讯数据，并对触摸屏之间的操作指令做出互锁保护。 所以是几乎没办法避免。

```
48
49 REGION read panel request
50   IF "PANELS_NO" > 0 THEN
51       FOR #tempPanelIndex := 1 TO #tempPanelNo DO
52       #statPanels[#tempPanelIndex] := #panels[#tempPanelIndex].mtr;
53       #statIdentNames[#tempPanelIndex] := #panels[#tempPanelIndex].identName;
54       IF #statIdentNames[#tempPanelIndex] <> #statOldIdentNames[#tempPanelIndex] THEN
55         #statFirstCalls[#tempPanelIndex] := true;
56         #statOldIdentNames[#tempPanelIndex] := #statIdentNames[#tempPanelIndex];
57       ELSE
58         #statFirstCalls[#tempPanelIndex] := false;
59       END_IF;
60     END_FOR;
61
62     FOR #tempPanelIndex := 1 TO #tempPanelNo DO
63       IF (NOT #statFirstCalls[#tempPanelIndex]) AND (#statPanels[#tempPanelIndex].settingsHMI <> #statOldPanels[#tempPanelIndex])
64         #statDataMtr.settingsHMI := #statOldPanels[#tempPanelIndex] := #statPanels[#tempPanelIndex].settingsHMI;
65         #statFirstCalls[#tempPanelIndex] := false;
66       END_IF;
67     END_FOR;
68   END_IF;
69
70 END_REGION
```

我能想到的是，如果真的需要封装这批函数，就为每一个数量值分别打包，比如 2 的时候一个包， 3 的时候另一个包。 哪怕触摸屏数量最多到 10， 那也顶多 10 套。

但毕竟，在真实的行业应用中，触摸屏的数量并不会如此多变， 所以暂时，也还只沿用其原有的使用常数的方法了。

最后再补充一点， 在 PLC 变量表中还有一页“系统常量”，通常系统会自动生成一大批，程序中可以使用



有可能会有人误解到我反对用系统常量。 关于系统常量的使用是，你只要不在 FB 内部使用即可。即，你可以使用，但只能把它们当作实参挂到 FB 实例的管脚上，但不可以封装在逻辑内部。

其实我们全文所反对的，也是把全局常量用到 FB 逻辑内部。

万泉河

山东 烟台

扫一扫上面的二维码图案，加我为朋友