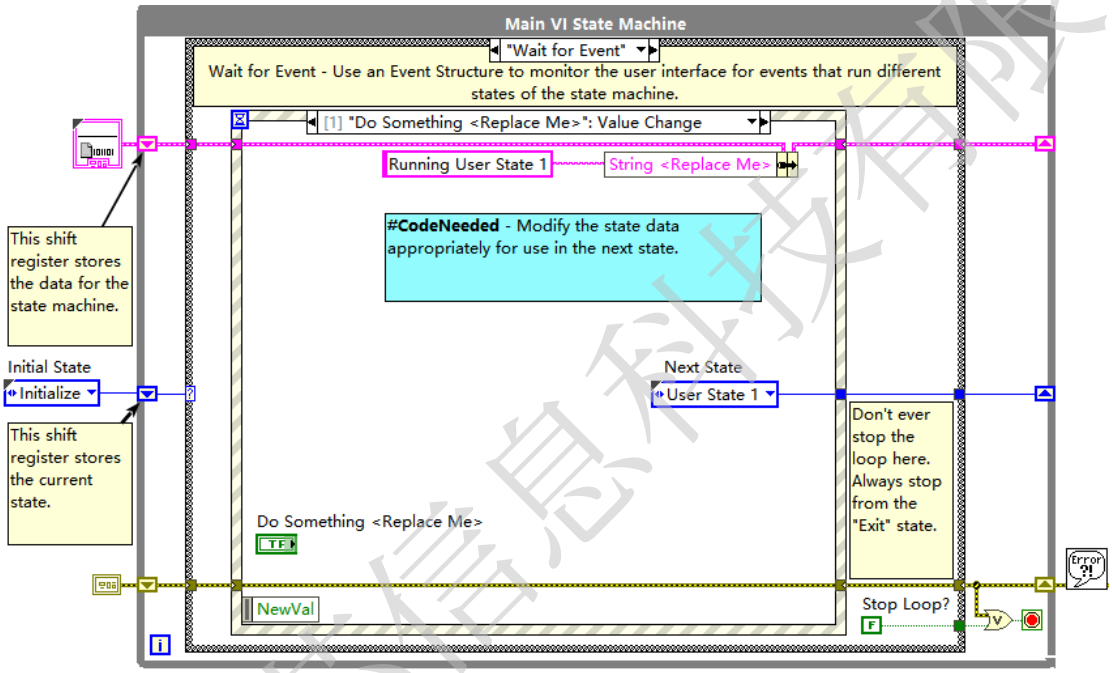
LabVIEWer 的新突破—从面向过程到面向操作者

LabVIEW 作为测控行业非常经典的编程环境与语言，从 1986 年发布至今，迭代了很多经典的设计模式，例如:

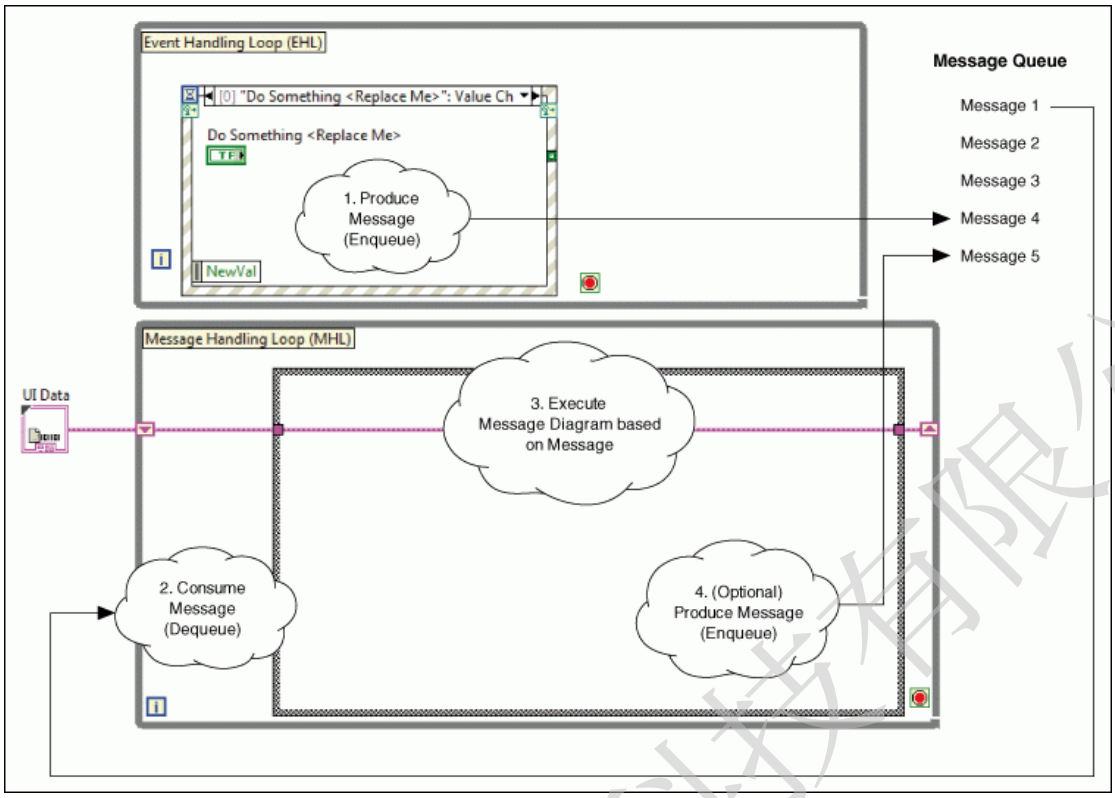
简单状态机

利用该设计模式，可以设计出序列化执行代码，可根据当前执行状态选取下一个执行的步骤，可以方便地插入新的执行步骤，修改执行顺序生成逻辑。其参考代码如下：



QMH

该设计模式在状态机的基础上，添加了消息机制，将状态机的状态跳转渠道，从移位寄存器改为队列，队列中传递的为消息，包含了消息名与消息内容，利用该设计模式，可以进行多线程编程，消息的产生方与消息的接收/执行方分别存在于不同的线程中，其也是生产者/消费者模式的一种，只是 QMH 中，消息的生产者也可以是其他消息渠道的消费者，此处生产者与消费者的定义只针对消息，不针对线程。



很多公司也推出了更加实用的设计参考架构，例如 JKI 状态机，其基本思想与状态机、QMH 基本类似。

上述设计模式能解决大多数小规模的应用需求，例如经典的 SIMON 游戏，小规模数据采集，晚会抽奖程序等，其不管是从概念还是从实现角度，看上去都很直接，易于学习与掌握。

LabVIEW 足够优秀，但在 LabVIEW 圈子中，很经典的一句话就是：LabVIEW 易学不易精。此话可以从两个角度来理解:

一是对编程技巧的精通，例如各种函数功能的掌握。此处体现了 LabVIEW 的强大之处，其以图形化、驱动库的方式集成了且可以继续集成丰富的功能，需要工程师日积月累的勤学苦练

二是对参考架构的渴求。大部分中途放弃 LabVIEW 或者对 LabVIEW 产生质疑的工程师，是因为在多年的使用后，发现永远停留于使用上述设计模式完成简单的应用，或者东拼西凑出稍复杂但日后自己都不想打开的应用程序，在从小白 LV 工程师变为资深 LV 工程师的路上，找不到随心所欲的感觉

小编在与 LabVIEW 相伴多年的过程中，有幸加入了许许多多的 LabVIEW 社区/群，深深感受到 LV 圈中对新突破的渴望，都希望写出真正的模块化、可伸

缩、可扩展、可复用且清晰简洁的代码，在与其他编程语言对比时，不再暗暗自卑。

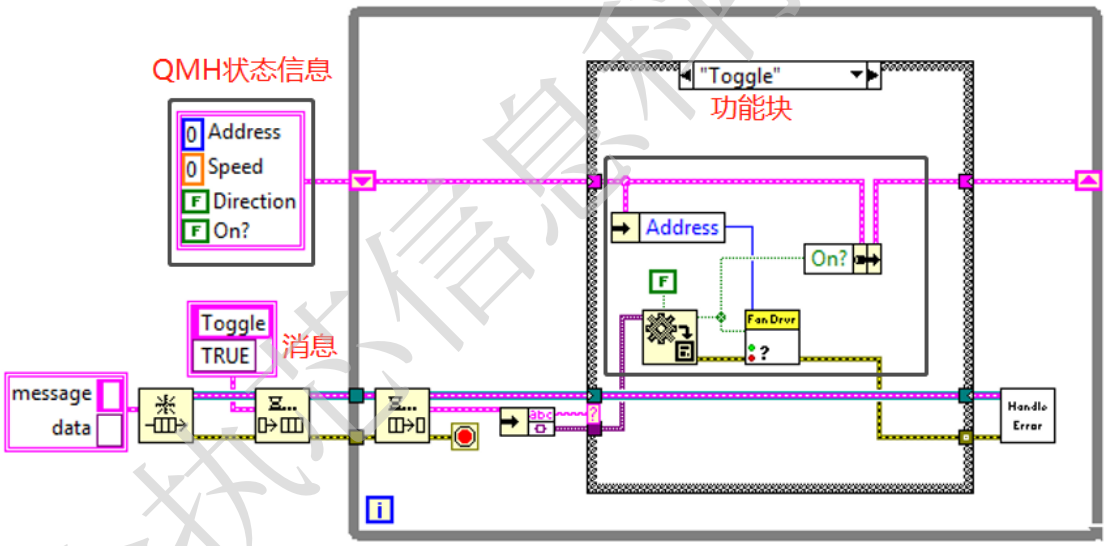
路在何方？本文将抛砖引玉，介绍一个方法：

----以面向对象为思想，以操作者为基石，以专业架构(观察者模式、MVC 模式等）为蓝图，在高效、成熟的编程工具的辅助下开展 LabVIEW 开发。

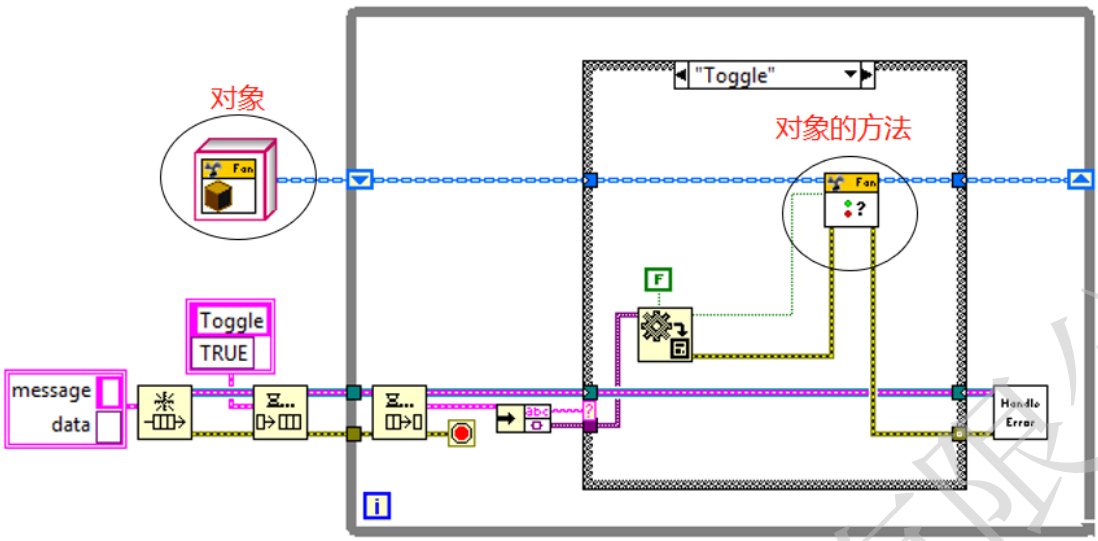
本篇将介绍面向对象的思想与操作者这个基石，这也是 NI 官方推广的两种 LabVIEW 编程方式；下篇将介绍专业架构与高效工具。

如何理解面向对象/操作者与上述设计模式的区别呢？我们以 QMH 为例。

在 QMH 中，消息处理线程如下，我们将从队列中接收消息，根据消息的名称进入不同的 Case 分支，然后取出消息中的数据以及 QMH 消息处理线程移位寄存器缓存的数据，进行特定功能的处理，最后将结果再次存储进线程移位寄存器中。



从复用的角度，该程序中的消息处理部分，只能通过复制代码片段进行复用，或者简单的创建子 VI 模块；但是随着子 VI 越来越多，会发现难于管理，因此有经验的工程师会想到用库的方式进行管理，此举会大大减低子 VI 的管理难度，并提升复用的可能性。但是，此处我们忽略了一个点，那就是数据。我们看到了 QMH 中，消息处理部分用一个移位寄存器存放了线程的状态信息，子 VI 间传递的其实也是经过其处理后的该状态信息。我们如果将数据信息也封装到我们的库中，那就得到了我们的对象，如下图所示：



面向对象思想并不仅仅是将数据存进库中，仅是如此的话，与.lvlib 库并无太大区别。面向对象是：

把一组数据结构与处理他们的方法组成对象

->对象=数据+方法

把具有相同行为的对象归纳为类

->**类**：人　**数据**：性别/年龄/...**方法**：呼吸/吃饭/...**对象**：张三/李四/...

->**类**：汽车 **数据**：长度/品牌/...**方法**：启动/刹车/...**对象**：张三的别克/李四的比亚迪...

通过封装隐藏类的内部细节

->外界无法直接获得对象内部细节，如：外界无法仅凭外貌得知一个人的具体年龄

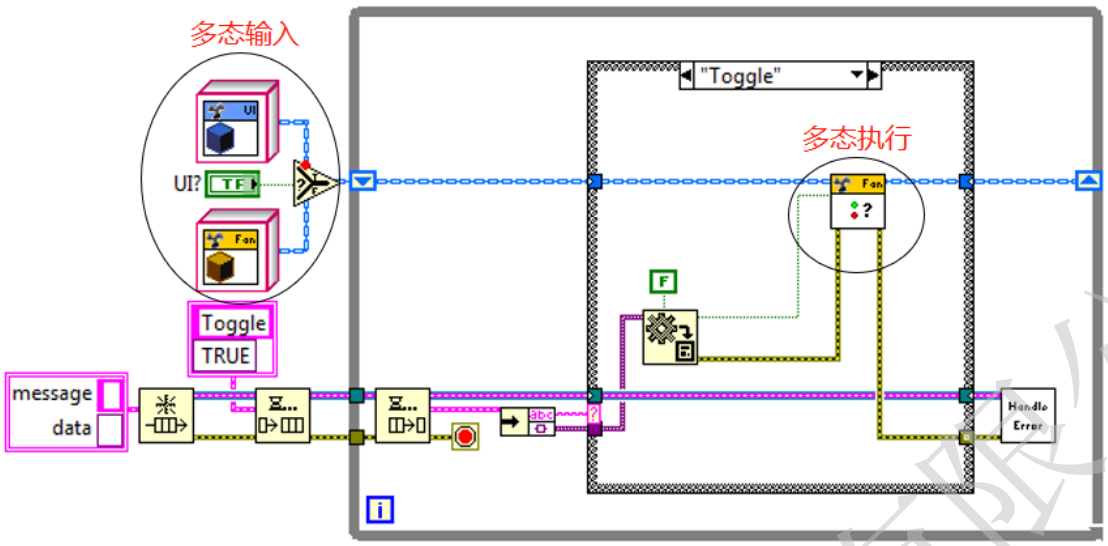
通过继承使类得到泛化

->女性类继承了人类，扩展了生育的方法

通过多态实现基于对象类型的动态派生

->新能源车与燃油车都继承了汽车类，但当他们启动时，使用的能源及启动的方式都是各自独有的

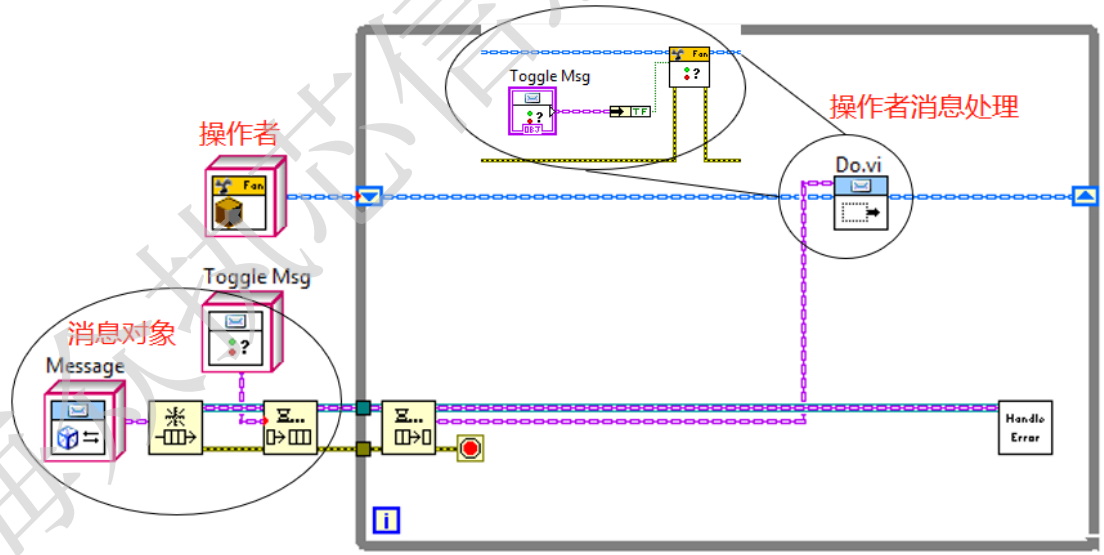
从下图就能看到其好处，其描述的是：如果应用程序需要一个新的版本，其顶层逻辑不变，仅是队列消息具体执行方法及系统状态包含的信息有变化，此时通过面向对象思想的继承与多态，即可在不需要修改主程序 Case 分支及其内部代码的情况下，通过创建一个新的类，在类的数据与方法中，实现需求的变更，在主程序中，通过输入不同的对象，执行方法时会根据对象的类型，动态调用对应的方法。



LabVIEW 面向对象编程足够优秀，但是，在扩展时，如果我们增加新的消息名称，那么就需要对 Case 分支数量进行扩充，还是需要修改主程序，那么有没有更好的方式，在不修改主程序的情况下，完成需求的扩展呢？此时就需要操作者了。

下图为面向操作者的 QMH 消息处理线程，我们发现有两个变化，

1. 将消息簇封装为消息类对象
2. 将 Case 结构，简化为 Do.vi



其优化逻辑为：

1. 消息类中封装了 a. 某个 Case 结构的功能执行代码，命名为 Do.vi，不同 Case 有不同的消息类，这样就可通过消息类名称替代了原消息簇中的消息名，b. 将消息簇数据部分封装进了消息类的数据中

2. Do.vi 为多态 VI，可根据传递的具体消息类，动态调用该类的 Do.vi，这样就通过多态替换了原来 QMH 中的 Case 分支

综上，我们通过

1. 将 QMH 的消息执行部分从面向过程编程替代为面向操作者
2. 将 QMH 的消息簇从簇结构替代为消息类

即完成了从面向过程至面向操作者的编程转换

利用面向操作者的编程方式，我们无需修改主题程序，仅需针对不同的"Case"，编写不同的消息类，这样即可完成程序的扩展，符合模块化，可复用性等特点。

下期文章将介绍从操作者模式至观察者+MVC 模式的演进，敬请期待！