从面向操作者到观察者+MVC 组合架构

上篇我们已经介绍了操作者，本篇将在操作者的基础上介绍观察者与 MVC

（请点击网页最后链接，查看上篇内容）

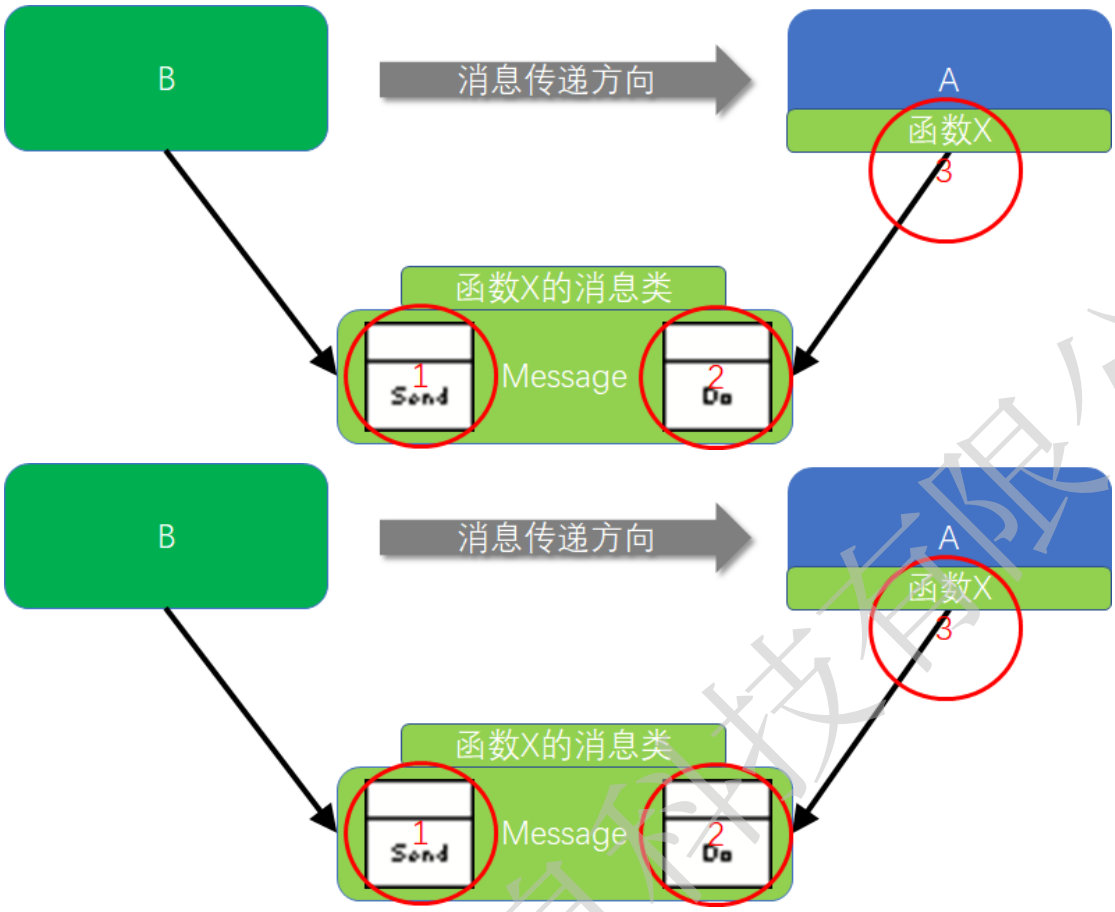
我们可以把操作者描述为一个盲盒，每个盲盒有两个特征

一是有唯一的地址
二是盲盒通过一个个消息类“API”对外开放必要的功能

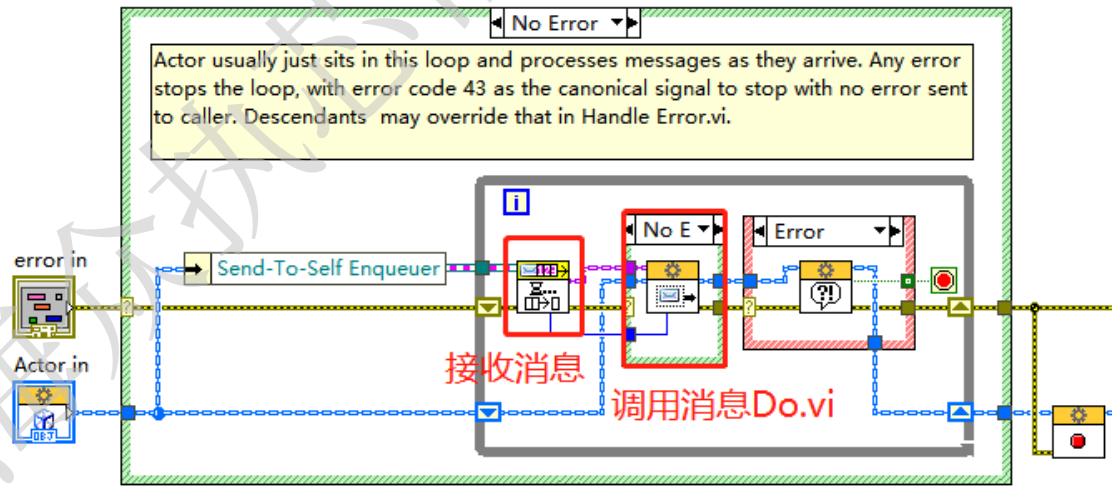
比如，盲盒 A（地址 A）对外开放了调用其内部某个功能 X 的接口（消息类），该接口（消息类）包括两个部分：

1. 供外部调用接口，Send xx Msg.vi，用于外部发消息给盲盒 A，告知它执行功能 X，仅起告知作用
2. 内部执行功能 X 的代码，Do.vi，用于执行功能 X

那么其他盲盒 B（操作者）通过调用 Send xx Msg.vi，向盲盒 A 发送一个消息，请求其执行 Do.vi，从而执行功能 X，如下图所示：



这就是操作者架构的消息机制，LabVIEW 中，在每个操作者启动后，都会执行一个叫做 Actor Core.vi 的代码，如下所示



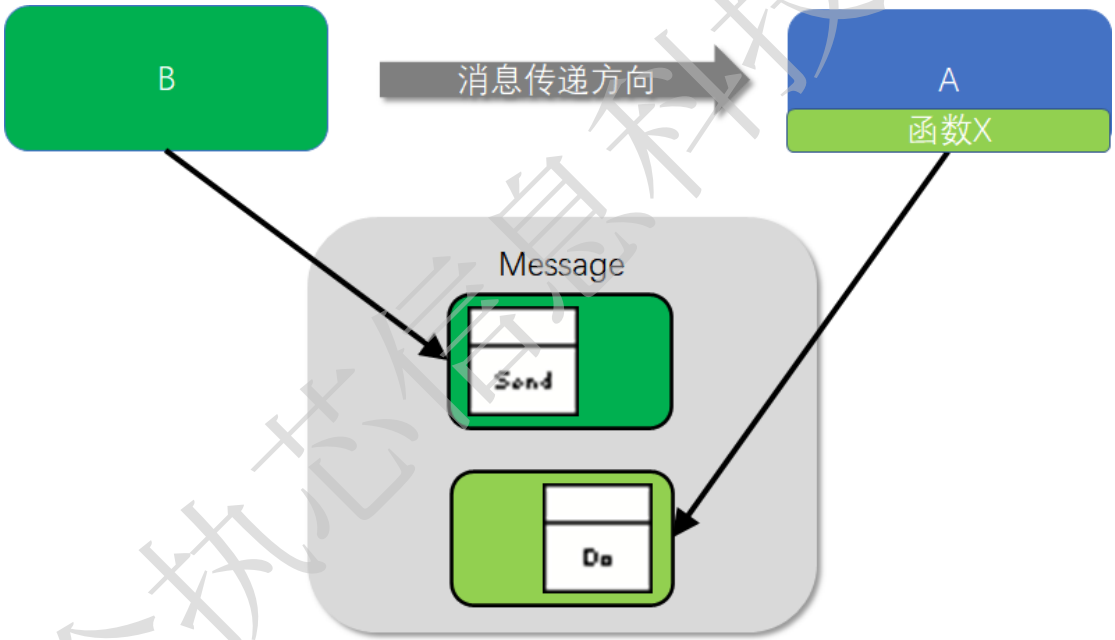
其为一个循环，不断检测并接收其他操作者发送过来的请求消息，不同的消息都将会执行该消息对应的 Do.vi，从而调用不同的功能

该消息机制看上去是没问题的，但是我们在上面的步骤中发现两个限制：

1. 操作者 A 必须知道操作者 B 的地址，才能向操作者 B 发送消息
2. 必须在操作者 B 中调用操作者 A 的消息接口的 vi，人为增加了 AB 之间的耦合

这样的限制会造成代码的复用难题，因为操作者 B 跟操作者 A 是无法完全解耦的，即使把操作者 B 拷到其他项目中复用，其还是需要依赖操作者 A 的，这并不是我们想要的

为此，我们需要将 Send 与 Do 分开，分别放到 B 和 A 中，如下图所示：



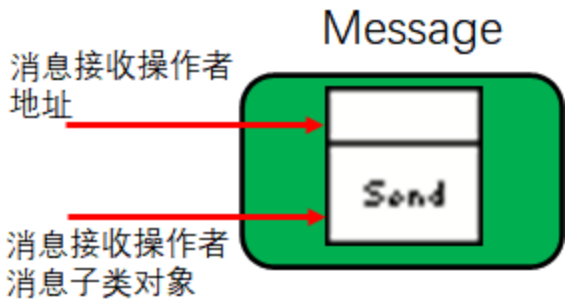
那么，Send 跟 Do 如何绑定，做到 B 发送的消息请求与 A 执行的 Do.vi 正确映射呢？

这里 NI 官方 LabVIEW 操作者架构给出了答案，那就是抽象消息，其设计思路如下：

1. 在操作者 B 中定义一个父类，其类数据为消息中包含的数据，同时包含一个 Send xx Msg.vi；并在操作者 A 中定义一个子类，继承于 B 中的父类，其中包含一个 Do.vi。

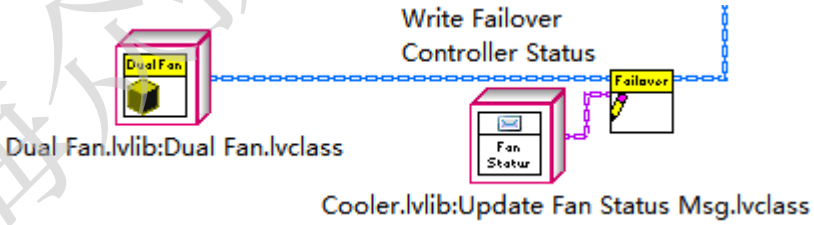
2. 在 B 中，存储 A 类的子消息类对象，需要发送消息请求时，调用 B 类自己的消息父类的 Send xx Msg.vi，其输入为操作者 A 的地址+B 中存储的操作者 A 的子消息类对象

如下所示：



这种方式，在 LabVIEW AF 架构中称作为零耦合。但是，细心的工程师可能已经发现了，这是真正的零耦合吗？

其实并不完全是的，我们说，真正的零耦合，两个操作者之间不但需要实现消息类 Send 方法与 Do 方法的解耦，同时，在消息的发送方 B 中，不能存在 A 的任何其他内容，而我们在上图中还需要在某个操作者的代码中，将消息接收操作者子类对象写入消息发送方属性中，如下图所示：



同时，上述例子中，操作者 B 的消息只能发送给操作者 A，要是还需要发给操作者 C 呢？或者在时刻 1 发给操作者 A，在时刻 2 要发给操作者 D 呢？

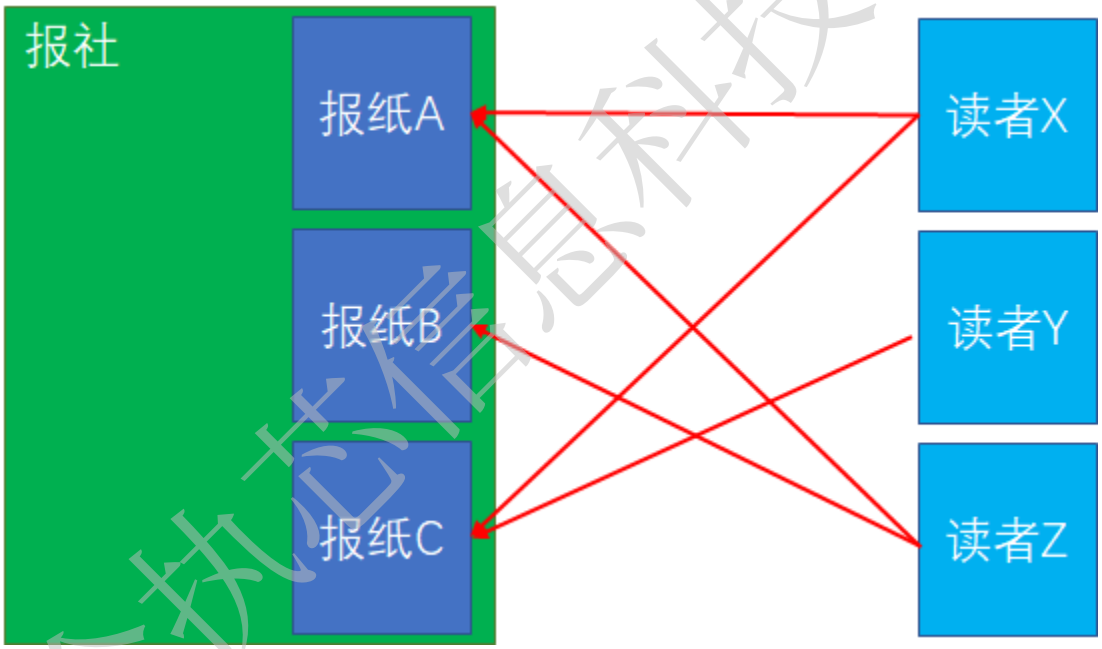
这些都是 LabVIEW AF 本身并未提供的功能，或者需要再添加额外的代码实现，那么我们如何能灵活地实现这些呢？

我们的答案是：观察者模式

观察者模式是一个经典的设计模式，具体的术语解释可自行百度。

本文将做通俗的讲解：

我们可以类比成某个报社，报社出版报纸 A,B,C 等，而社会读者有 X,Y,Z 等，每个读者订阅不同的报纸，例如读者 X 订阅了报纸 A 和报纸 C



实际的执行逻辑抽象如下：

1. 读者给报社发送订阅消息，告知要订阅什么
2. 报社收到订阅消息后，为每个报纸维护一个单独的订阅名单，如：报纸 A 的订阅者有读者 X 和读者 Z

3. 报社在报纸准备好后，将该消息广播给对应的读者收报纸，如：当报纸 A 印刷好后，报社给订阅了报纸 A 的读者 X 和 Z 广播消息，通知报纸 A 已经印刷好的消息
4. 读者收到消息后，取得订阅的报纸，并执行看报纸的动作
5. 若读者想取消订阅某报纸，可以给报社发送取消订阅消息
6. 报社收到取消订阅消息后，更新对应报纸的订阅名单

上述逻辑，在实际生活中很容易理解，那么在观察者模式中，我们需要从里面抽象出什么概念呢？

1. 消息：这里共有两类消息，一是订阅类消息，发送方为读者，接收方为报社，为一对一，二是广播类消息，发送方为报社，接收方为读者，为一对多
2. 订阅名单：由消息发送发维护的一个列表，需要记录某个消息的接收方（地址）与接收方的执行动作列表

再回头看上述两个问题：

1. 有没有实现零耦合？

实现了，消息发送方与接收方没有静态的耦合，相互之间的收发关系是通过运行时发送的订阅消息动态生成的

2. 有没有解决 1 对 1 的限制？

解决了，通过订阅制，可以实现一对多，多对多

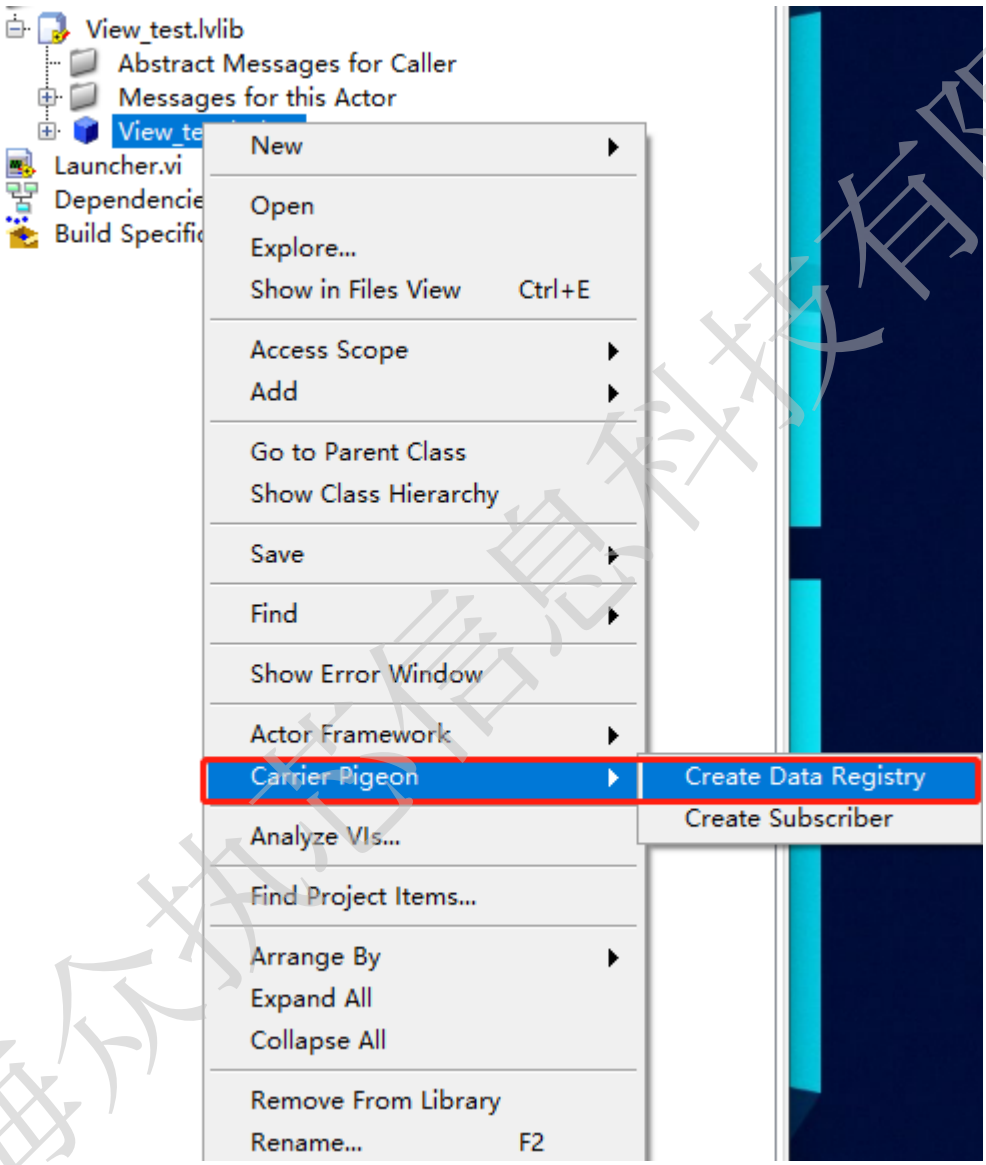
那在 LabVIEW 种如何实现呢？当然，我们可以按照上述概念自行编写，但是在我们新推出的套件中，这些操作很简洁直观，

（请点击网页最后链接，查看新品发布内容）

下方为某个消息的创建与订阅方法。

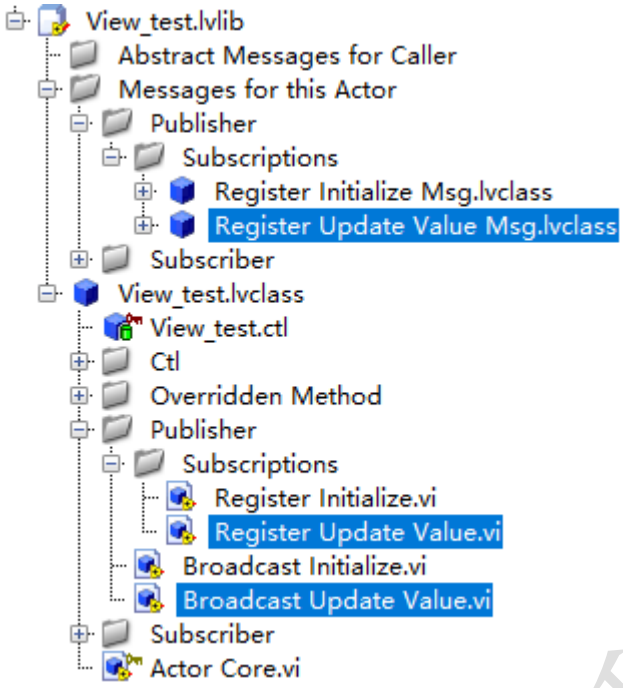
第一步：广播方定义广播内容

即定义有那些报纸，创建其订阅花名册，只需要在广播方类上右键操作，并在弹出的对话框中输入消息名称即可完成

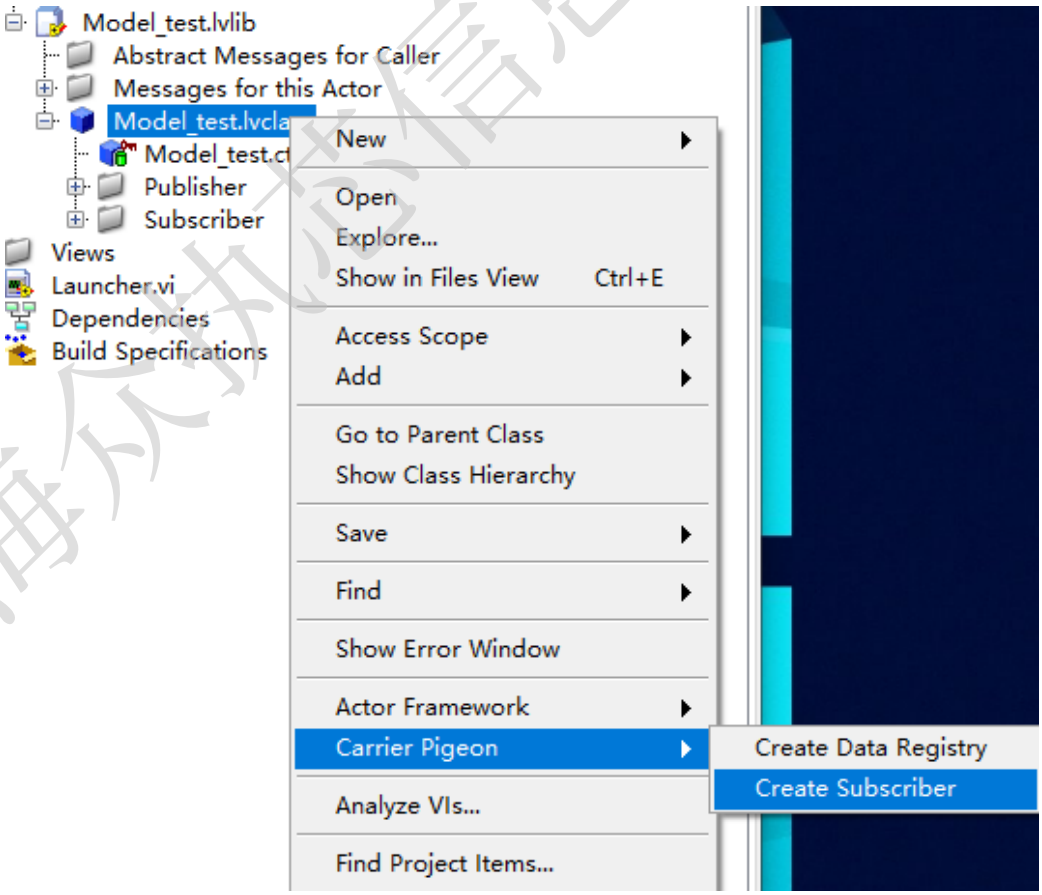


可能读者会关心消息的数据怎么定义？此处我们经过巧妙的设计，实现了针对任意广播消息，都不需要事先定义其消息数据类型，可在实现过程中灵活填充消息数据。

创建完成后的项目如下，在需要发送广播消息时，调用 Broadcast xx.vi 即可，其他 vi 默认放置即可，供工具包脚本自动调用。

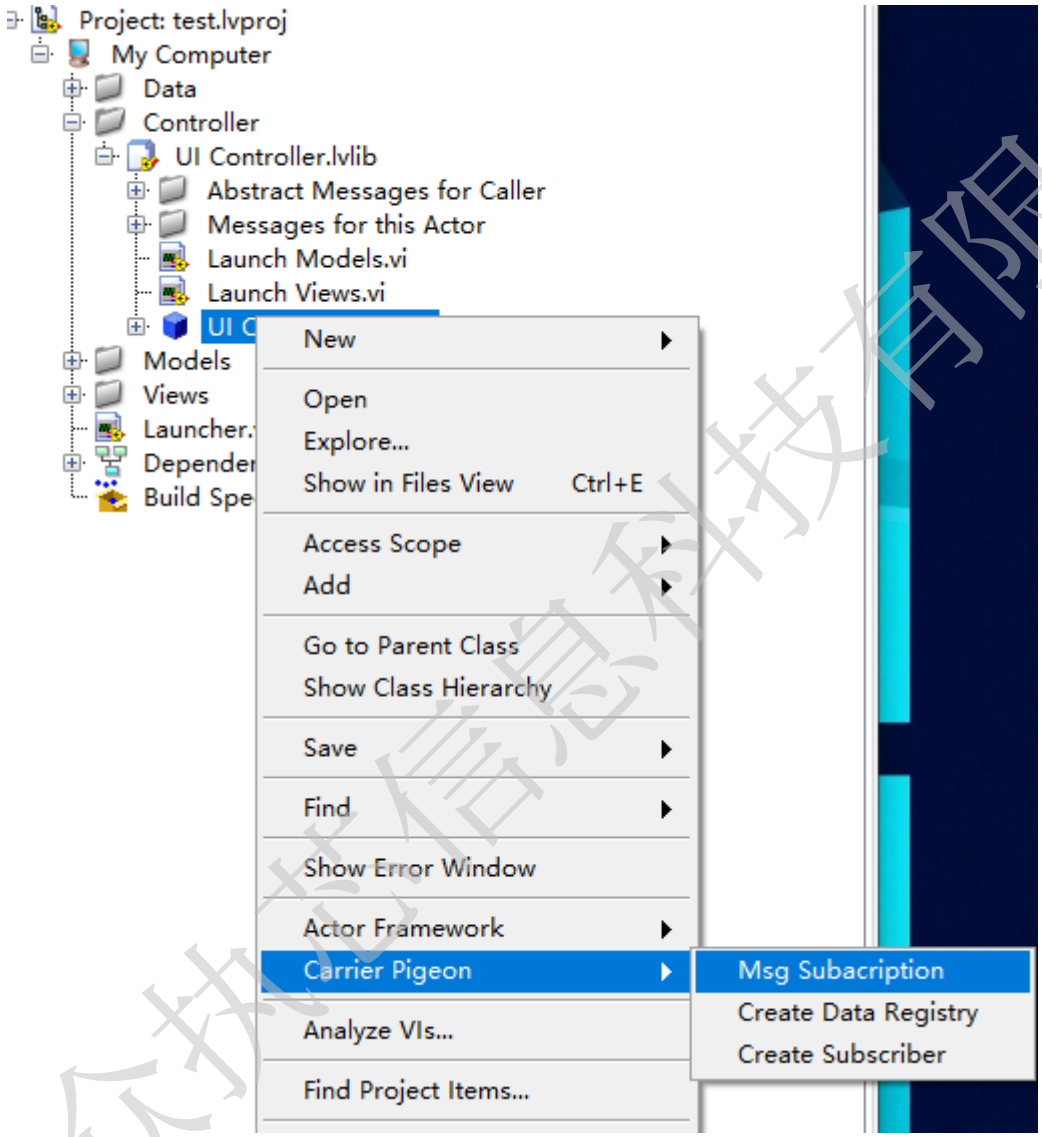


第二步：接收方定义收到广播消息后的执行的动作，在类上右键操作即可，会创建出一个 VI 模板，供用户编写具体的功能

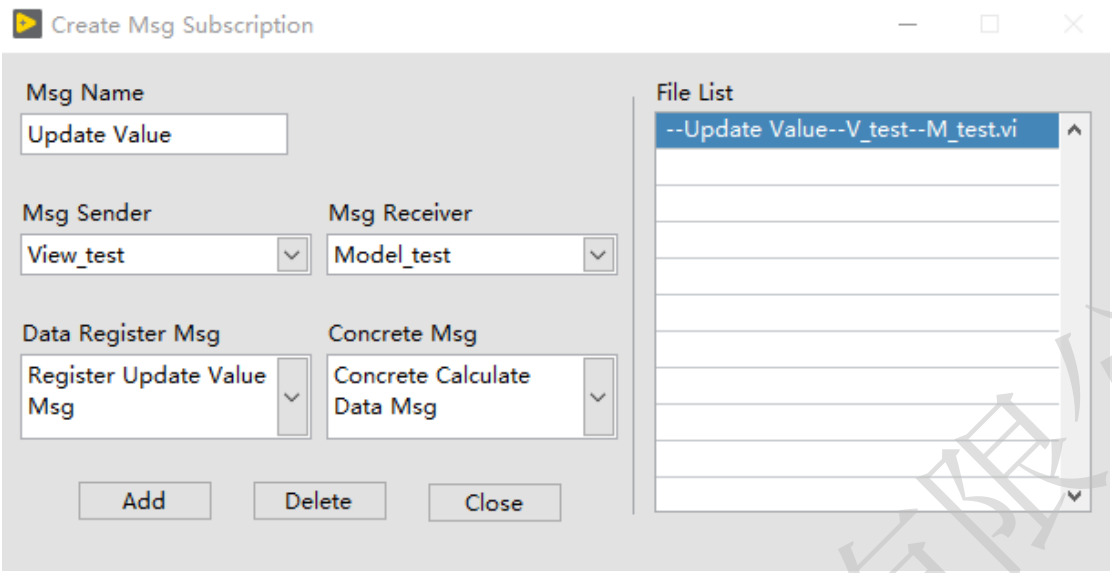


第三步：完成订阅注册，即读者给报社发送订阅/注册消息，

由于需要完成操作者与操作者之间的解耦，我们将此动作放在了工具包中所有操作者共享的 UI Controller 中，在其上右键即可



在弹出的对话框中，可以直观地看到有哪些操作者的具体的广播消息，以及哪些操作者的具体的订阅动作，我们只需要选择发送方与接收方，即可自动完成消息的注册，不需要任何代码编写工作

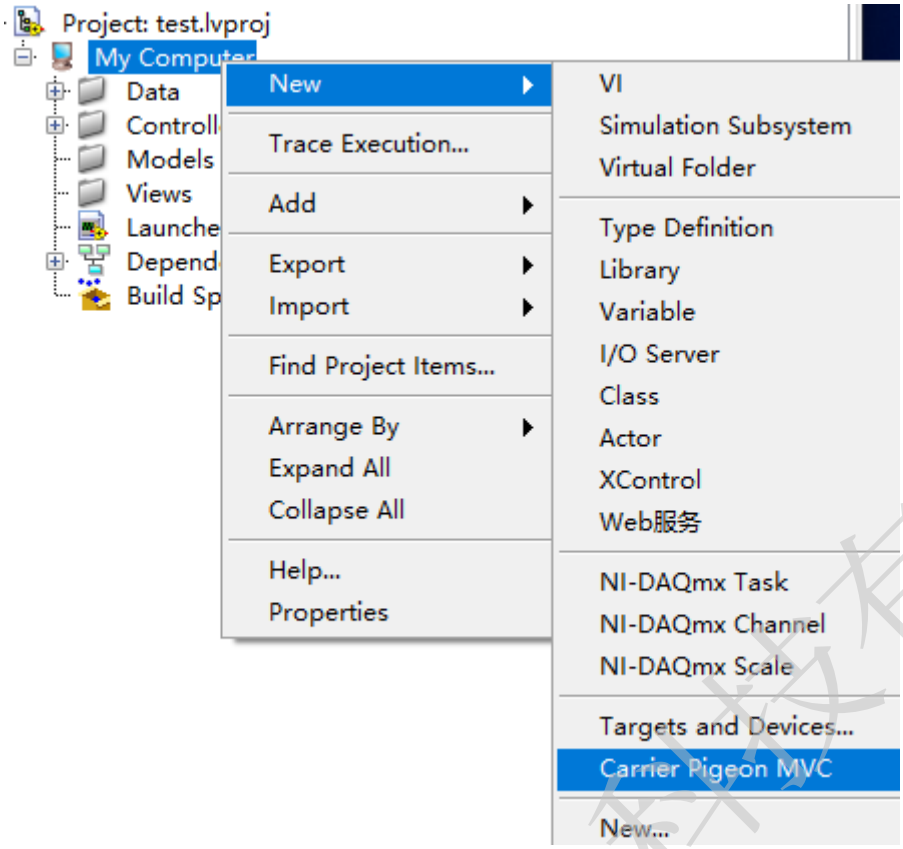


从上面图例中可以发现，CPigeon 套件的消息创建、订阅与广播过程十分易于理解与使用，用户可以很容易的掌握，最终在操作者之间轻松实现想怎么发送消息就怎么发送消息的驾驭感，并且程序还能保持高度的模块化，可扩展，高度复用性等特点，非常有利于团队合作以及应用的长期迭代。

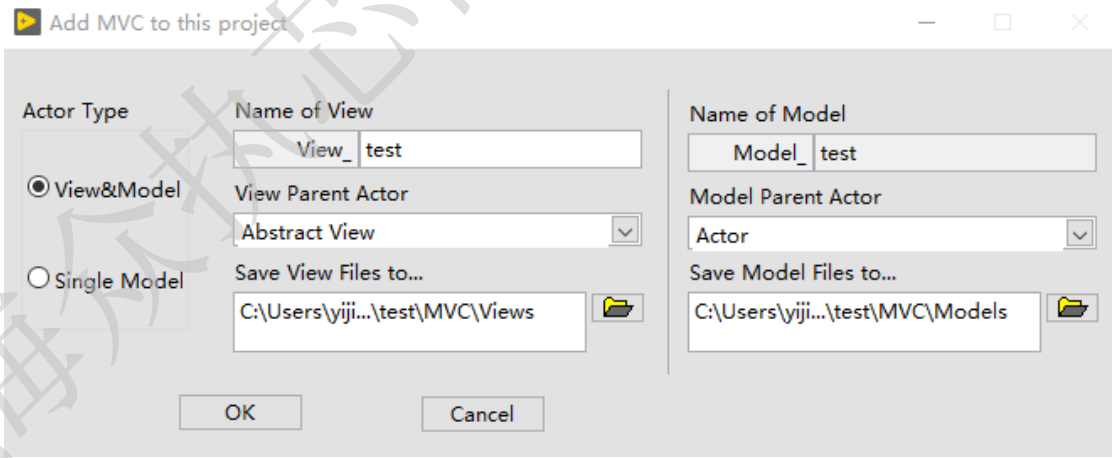
而在上示图例中，您可能已经注意到了我们有三类操作者，分别时 Model，View 以及 Controller，这就是 CPigeon 工具包中包含的第二种设计模式：MVC

通俗来讲，MVC 设计模式即将原本的一个操作根据界面与逻辑的区分，划分为 View 与 Model，View 负责完成界面的操作，即前端，Model 负责完成操作者的逻辑与功能块执行，俗称后端，其间的交互可以通过我们的观察者模式进行。

利用本套件，可以很方便地创建出 Model&View 对，通过右键菜单操作即可，如下图所示：



在弹出的对话框中，输入名称即可完成创建。当然，本套件也支持只创建 Model，因为部分操作者并无界面



每个操作者都有界面，那这些界面如何整合为一个完整应用呢？这就需要 Controller 了，Controller 部分完成整体应用的控制，而在本套件中，我们定义了 UI Controller，负责完成整体应用 UI 的设计，包括但不限于：

1. 操作者的启动
2. 观察者的注册机制
3. 界面的布局与切换
4. ...

同时，针对 UI 设计中比较重要的语言字典库、字体设计等 UI 元素设计，本套件在 MVC 部分中的 View 模块中提供了支持。

综上，本套件借鉴了专业软件开发思路，结合多年的工程经验，大大延申了 LabVIEW 平台的能力，可实现：

开发流程专业化

开发过程轻量化

开发团队协作化

产品持续可扩展

学习成本低廉化

希望通过两篇文章的介绍，让 LabVIEW 工程师对于找到一个适合于自己软件开发的思路。