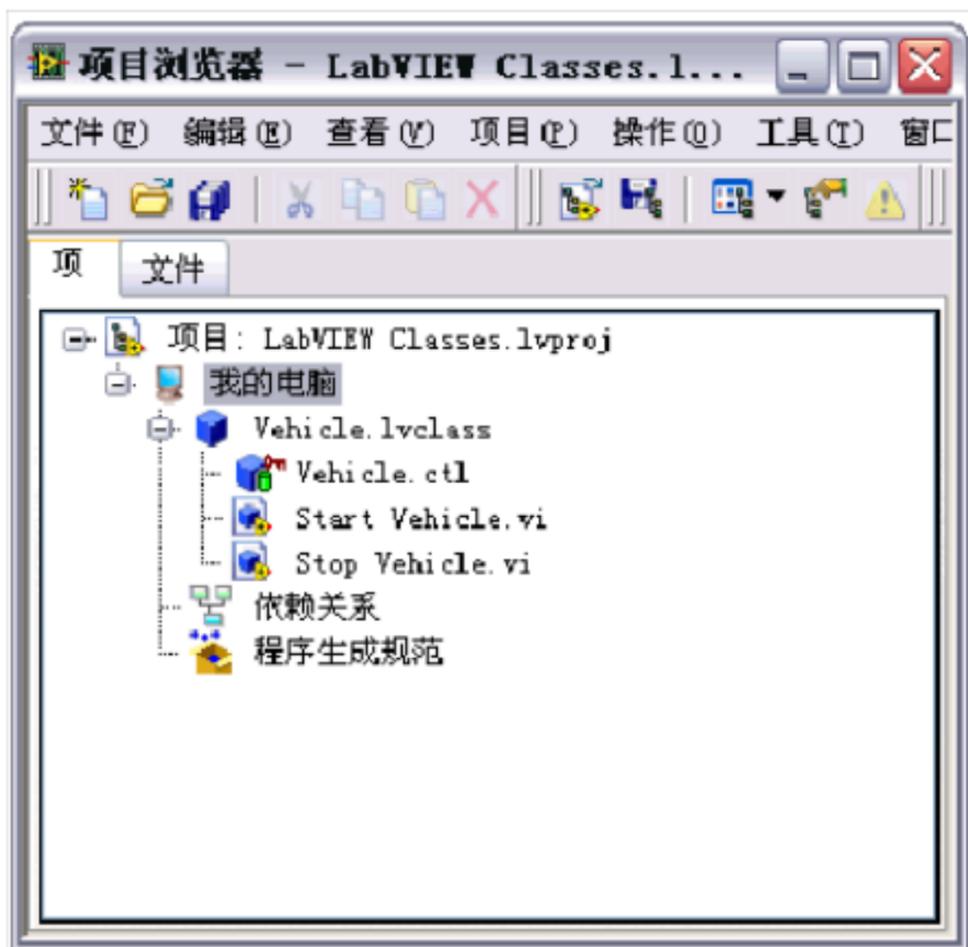


# Labview 面向对象编程快速入门

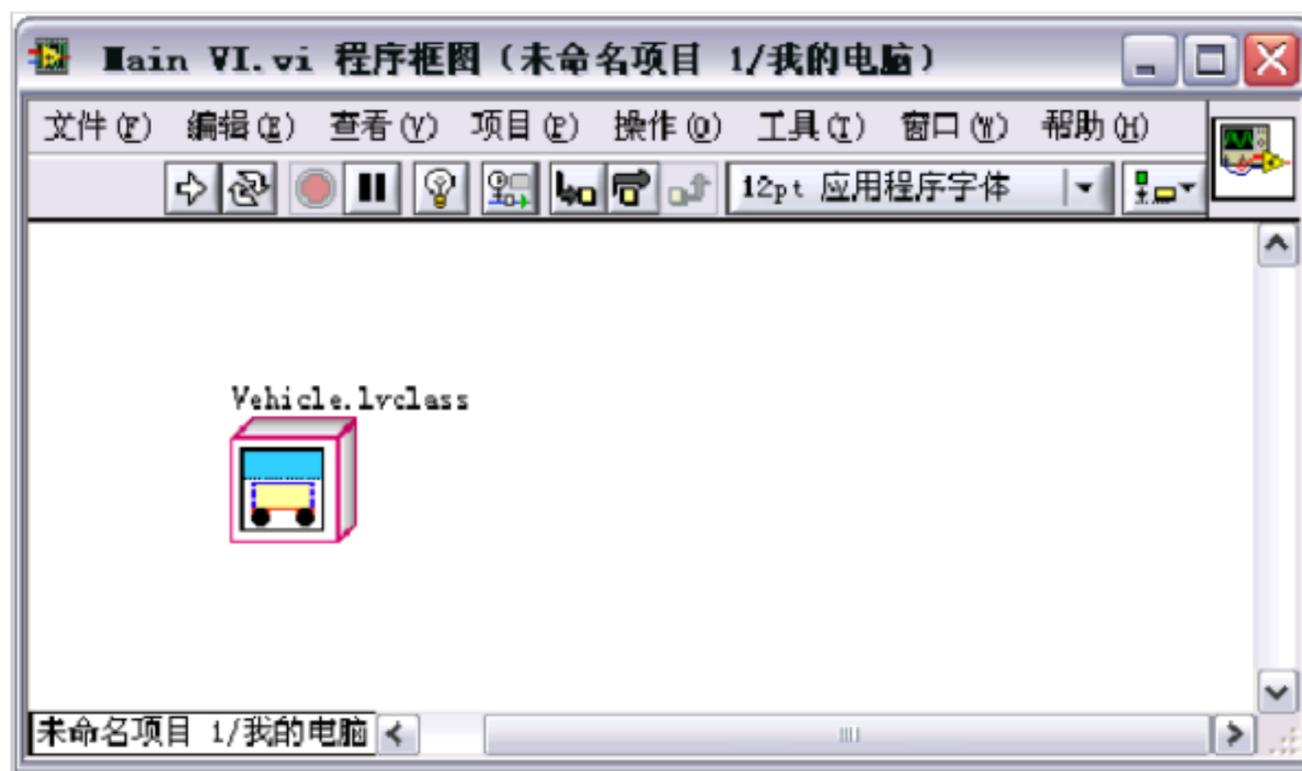
## 一、类和对象的基本概念

在面向对象编程中，类用来表示通用特性。例如，假设有一个描述汽车的类。类中定义了各种汽车的通用特性。对象是类的特定实例。汽车类的一个对象，可以是某一辆特定的汽车。类的定义决定该汽车（对象）的行为。

类中定义了和对象相关的数据和方法。仍以汽车为例。世界上有许多类型的汽车。汽车可分为轿车、卡车、公共汽车等不同类别。汽车的车身上有车门，排挡上有齿轮。车门和齿轮的数量信息都是汽车的相关数据。汽车还可加速或刹车。加速或刹车都是汽车的行为（或称为方法）。汽车相关的数据和方法，构成了汽车类的定义。通过类的创建，可定义对象的数据和方法。下图示范了汽车类在 LabVIEW 中的表示。LabVIEW 将汽车类的数据存储在 Vehicle.ctl 中，将汽车的方法存储在 Start Vehicle VI 和 Stop Vehicle VI 两个 VI 中。



对象是类的特定实例。指定的某一辆汽车是汽车类的一个特定实现，或称为汽车类的一个对象。下列 Main VI 的程序框图中，有一个汽车类的对象。对象包含哪些数据和方法，是通过类来定义的。LabVIEW将类的数据存储在一个控件中，用户创建的成员 VI 即是 LabVIEW类的方法。



为使 LabVIEW面向对象编程的概念更为清晰，可通过如下方式区分 LabVIEW类的不同用户：

- LabVIEW类开发人员 - 开发 LabVIEW类，以供其他开发人员及程序员使用。  
LabVIEW类开发人员适合拥有面向对象编程经验的人员担当。 LabVIEW类开发人员必须理解 LabVIEW类及其机制。
- LabVIEW类用户 - 使用 LabVIEW类开发人员所创建的类。 LabVIEW类用户可以在应用程序中利用面向对象编程的优势，但无需了解类的运行机制， LabVIEW类用户不一定有面向对象编程的经验。 LabVIEW类开发人员发布类之后， LabVIEW类用户可能不具备访问该类内部操作的权限。 LabVIEW类开

开发人员对 LabVIEW类所作的修改，应极少影响 LabVIEW类用户所开发的应用程序。

LabVIEW类用户无需了解如何创建 LabVIEW类，但必须了解应用程序中通过类定义的数据类型应当如何使用，涉及 LabVIEW类的代码有哪些可用于调试的信息，以及 LabVIEW类的新版本将如何影响已经生成的应用程序。在只需使用现有的 LabVIEW类而无需对 LabVIEW类进行开发的情况下，可参考在应用程序中使用 LabVIEW类，学习如何使用其他开发人员所提供的 LabVIEW类。

## 二、在 LabVIEW中创建类

通过创建 LabVIEW类，可在 LabVIEW中创建用户定义的数据类型。LabVIEW类定义了对对象相关的数据和可对数据执行的操作（即方法）。通过封装和继承可创建模块化的代码，使代码更易修改而不影响应用程序中的其它代码。

在 LabVIEW中，类的数据是私有的，也就是说，只有类的成员 VI 才有权限访问该数据。类的数据可在私有数据控件中定义。创建和保存 LabVIEW类时，LabVIEW将创建一个类库文件 (.lvclass)，其中定义了新的数据类型。类库文件记录了私有数据控件和所有被创建的成员 VI 的信息，比如 VI 列表以及 VI 各自的不同属性。类库和项目库 (.lvlib) 相似。不同的是，类库定义了新的数据类型。私有数据控件对应唯一的类库文件，其中为新的数据类型定义了一簇数据，该簇也是类连线上的数据。LabVIEW的私有数据控件并不保存在磁盘上，而是保存在类库文件中。由于在类库文件中保存私有数据，不符合类定义的私有数据一定不会被使用。

提示： 如需保存类库文件、类成员 VI，以及类自定义默认探针，可在磁盘上创建一个和 LabVIEW类同名的目录，将属于该类库的文件保存在该目录中。如同一目录中包含了多个属于不同类库的文件，那么在不同类库中添加相同名称的 VI 时将产生冲突。在开发过程中重写动态成员 VI 将产生命名冲突。

## 2.1 封装

每个 LabVIEW类包括一个数据簇和用于读写该簇的方法。 LabVIEW类的数据是私有的，对于不是该类成员的 VI 来说是隐藏的。如需访问类的私有数据，必须创建方法，即创建该类的成员 VI，通过成员 VI 中的函数对私有数据执行操作。封装就是将数据和方法合并到一个类中，类中数据仅可由类的成员 VI 访问。通过封装可创建模块化代码，有利于方便地更新或修改代码而不影响应用程序中其它部分的代码。

类中的数据是私有的，但成员 VI 却可以按不同的程度公开。方法的设置访问范围选项可以有如下设置：

- 公共 - 任何 VI 都可将该成员 VI 作为子 VI 调用。
- 库内 - 只有同类中的 VI、类的友元或类的友元库中的 VI 可以调用库内成员 VI。在项目浏览器窗口中，库内成员 VI 图标中有一个深蓝色的钥匙符号。
- 保护 - 仅该成员 VI 所在的类及其子类中的 VI 可以调用该成员 VI。在项目浏览器窗口中，受保护的成员 VI 图标中有一个暗黄色的钥匙符号。

- 私有 - 仅该成员 VI 所在类中的 VI 可以调用该成员 VI。在项目浏览器窗口中，私有成员 VI 图标中有一个红色的钥匙符号。
- 未指定 - 仅当选中一个文件夹时，显示该选项。文件夹的访问范围未指定时，其访问范围默认为公共。默认情况下，如未对类中的文件指定访问范围，则这些文件夹的访问范围为公共。

注： 如指定文件夹的访问选项， 则访问设置适用于文件夹下的所有文件，并将覆盖各个文件的原有设置。

注意： 如将动态分配 VI 设置为库内，这些 VI 将无法运行。为友元创建一个静态分配的包装 VI，调用该包装 VI 并将其设置为库内，以此向友元赋予访问受保护动态分配 VI 的访问权限。

## 2.2 分配库的友元

将一个 VI 分配为库的友元，即是给予该 VI 调用库内任何成员 VI 的权限。也可分配一个库作为库的友元。

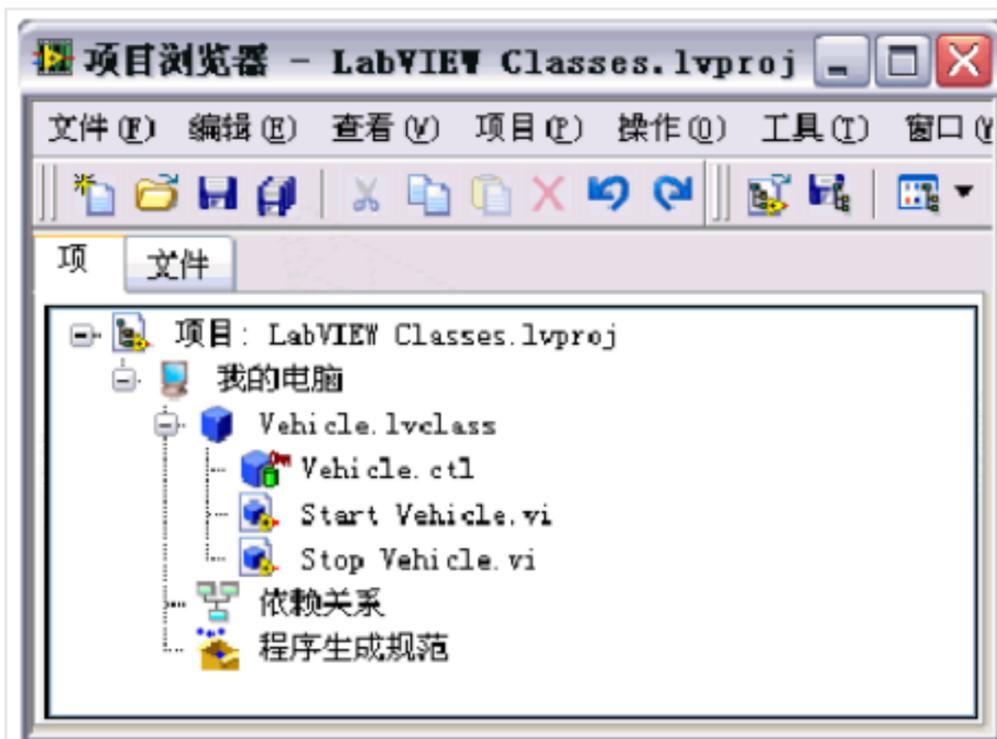
友元关系不具有传递性。例如，如第一个库分配第二个库为友元，第二个库分配第三个库为友元，第三个库不能作为第一个库的友元。除非第一个库将第三个库作为友元，第三个库无法访问第一个库的 VI。如访问权限在库内的库指定某个类为友元，该类的成员 VI 可访问库的 VI，但是友元关系不延展至类的子孙类。

可创建一个在 LabVIEW 类之外的 VI，将公共成员 VI 作为子 VI 在程序框图上使用。公共成员 VI 允许用户操作类的私有数据。用户可在成员 VI 的程序框图上使用私有和受保护的成员 VI 操作 LabVIEW 用户不可见的类私有数据。独立于类

的 VI 可作为类的友元，友元 VI 可调用库内的成员。对类的入口点进行限制，可减少数据引入错误的机会，更便于调试代码。

### 2.3 定义私有数据控件

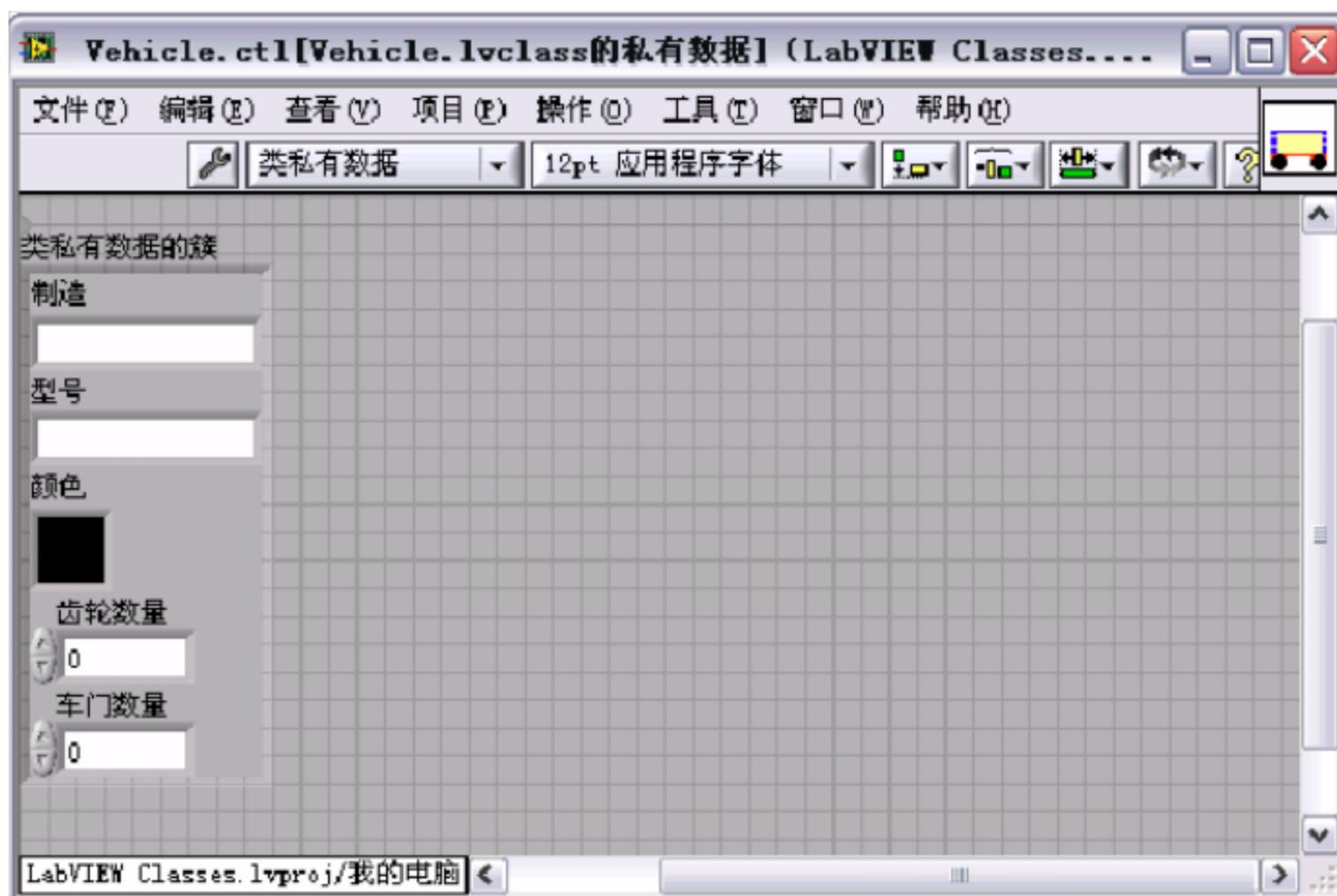
创建 LabVIEW 类时，LabVIEW 将自动创建类的私有数据控件。 请注意在下列项目浏览器窗口中， LabVIEW 类的图标是一个有色立方体。该立方体用于代表一个 LabVIEW 类。私有数据控件的图标是一个带有绿色圆柱体的有色立方体。圆柱体用于代表数据存储。同时，私有数据控件的图标中有一个红色钥匙符号，表示该控件是私有的。



通过控件编辑器窗口可对类的私有数据控件进行自定义。在项目浏览器窗口中双击类的私有数据控件，即可打开“控件编辑器”窗口。可将类私有数据的簇中的输入控件和显示控件放置到 LabVIEW 类的定义私有数据控件中。为放置输入控件和显示控件中的输入控件设置的默认值为类的默认值。

注意：私有数据控件不能包含 XControl 或 XControl 引用句柄。

以下范例中，汽车类的数据类型包含 齿轮数量、车门数量和颜色三个数值，以及制造和型号两个字符串。



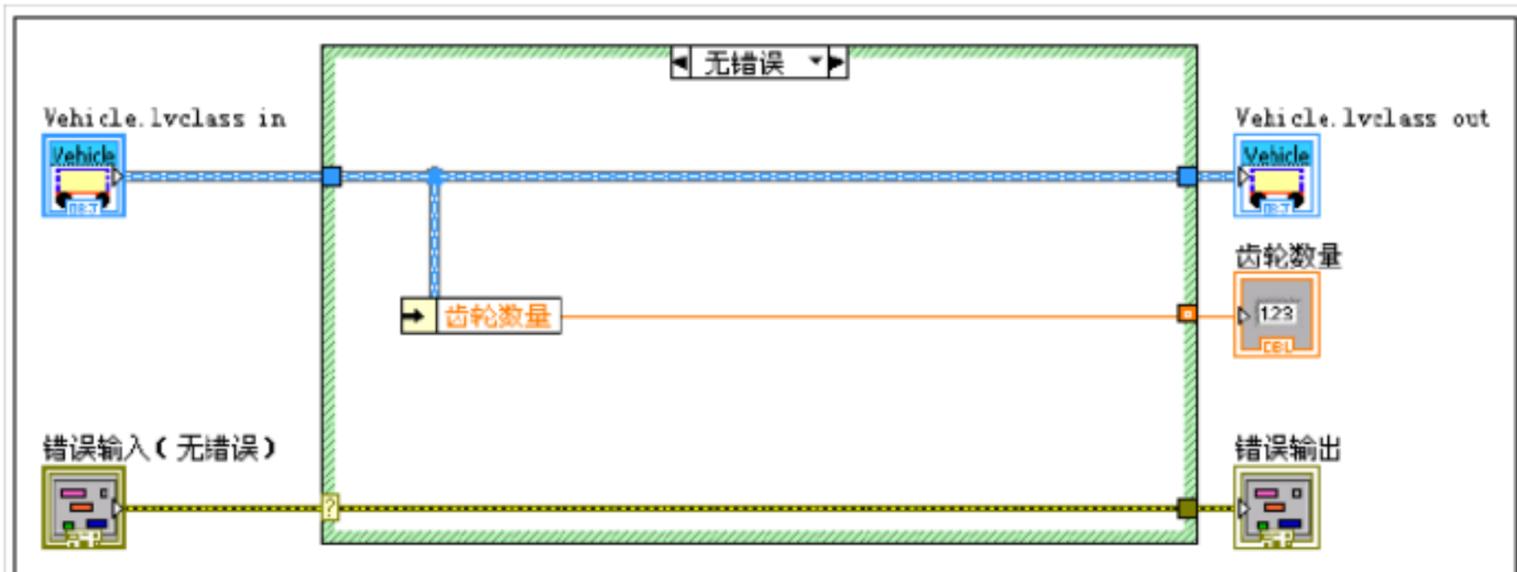
注意：如类中不需要任何私有数据，类私有数据的簇中可以不设定任何数据。

用户可创建在前面板或程序框图上代表类的图标。单击类属性对话框常规设置页的编辑按钮，打开图标编辑器对话框。创建类图标后，LabVIEW将把类图标应用于类的所有对象。修改类中各个对象的图标。

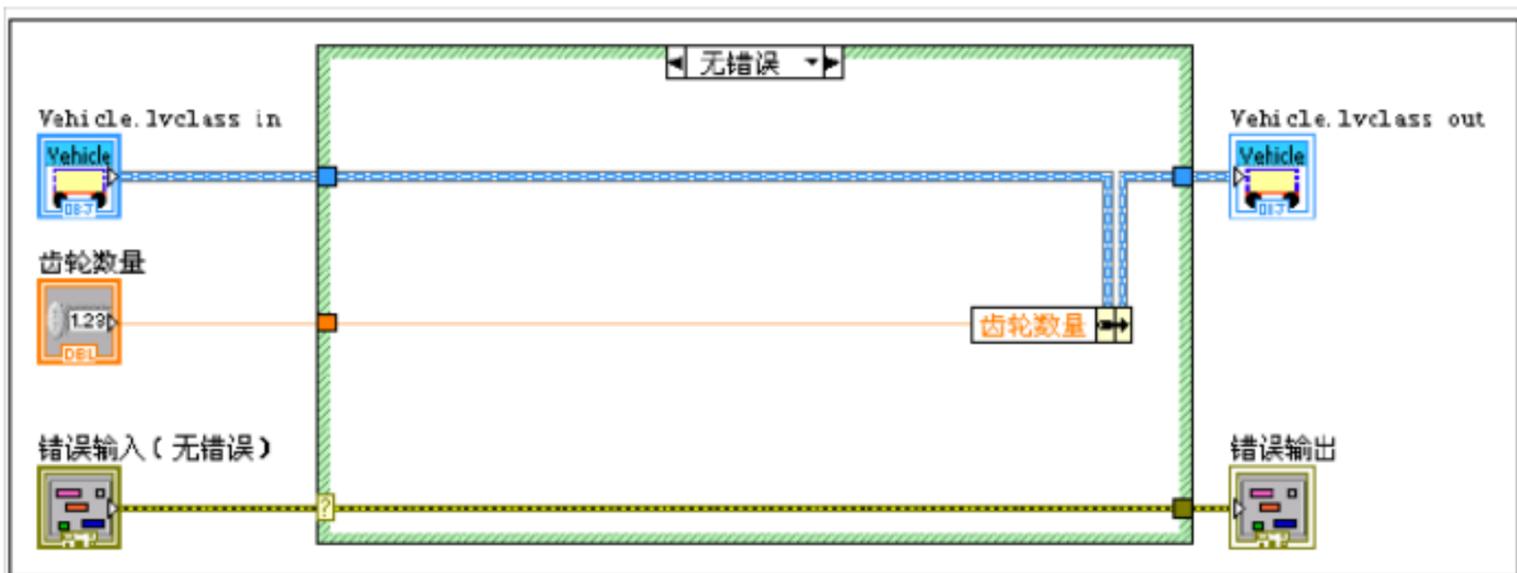
### 2.3 创建成员 VI

通过创建成员 VI（或称创建方法），可对类的私有数据执行操作。成员 VI 是 LabVIEW类的方法。在 LabVIEW类中创建，并在项目浏览器窗口中该类的私有数据控件下出现的 VI，是该类的成员 VI。

LabVIEW将类的数据定义为一个簇。所有成员 VI 都可对类数据的簇进行读写。LabVIEW为创建用于访问簇中各元素的 VI 提供了捷径。这种访问器 VI 是 LabVIEW 类的成员，可对类数据进行读写操作。如创建一个访问器 VI 以读取类数据，LabVIEW将取消对类数据的绑定，如下图所示。



如创建用于写入类数据的访问器 VI，LabVIEW将把新值绑定至类数据，如下图所示。



也可使用解除捆绑或按名称解除捆绑函数，在成员 VI 的程序框图中对类的私有数据解除捆绑。使用捆绑或按名称捆绑函数可在访问和操作私有数据之后，将数据重新捆绑成簇。由于类的数据是私有的，若试图在非该类成员 VI 的程序

框图中通过“捆绑”和“解除捆绑”节点访问该类数据，节点将自动断开无法运行。

注：建议尽量使用“按名称捆绑”和“按名称解除捆绑”函数替代“捆绑”和“解除捆绑”函数，以免在私有数据的簇中插入新元素时 VI 断开。

如写入成员 VI 的操作将取消捆绑某个值，修改该值然后将值捆绑至对象，可使用元素同址操作结构，在结构两边放置解除捆绑和捆绑函数，以实现更高的效率。该结构可保证 LabVIEW 使用了某些内存优化技术。使用常规取消捆绑和捆绑节点时也可使用该内存优化的方法。但是，在复杂 VI 的情况下，LabVIEW 编译器可能会认为优化不够安全而拒绝使用优化算法，导致运行速度变慢。元素同址操作结构保证了这些优化算法的安全性，确保 VI 按优化算法运行。

可通过各种方式创建成员 VI。右键单击类并在以下快捷菜单项中选择：

- 新建 ?VI - 打开一个空的成员 VI。
- 新建 ?属性定义文件夹 - 创建一个属性定义文件夹，可在其中创建或添加现有成员 VI。如 LabVIEW 类包含一个属性定义文件夹，可将 LabVIEW 类连接至属性节点访问私有数据。
- 新建 ?基于动态分配模板的 VI - LabVIEW 将生成一个新成员 VI，该 VI 带有错误输入簇、错误输出簇、一个用于错误处理的条件结构，以及输入 LabVIEW 类和输出 LabVIEW 类。在 VI 连线板上，LabVIEW 将输入和输出接线端都设置为动态。

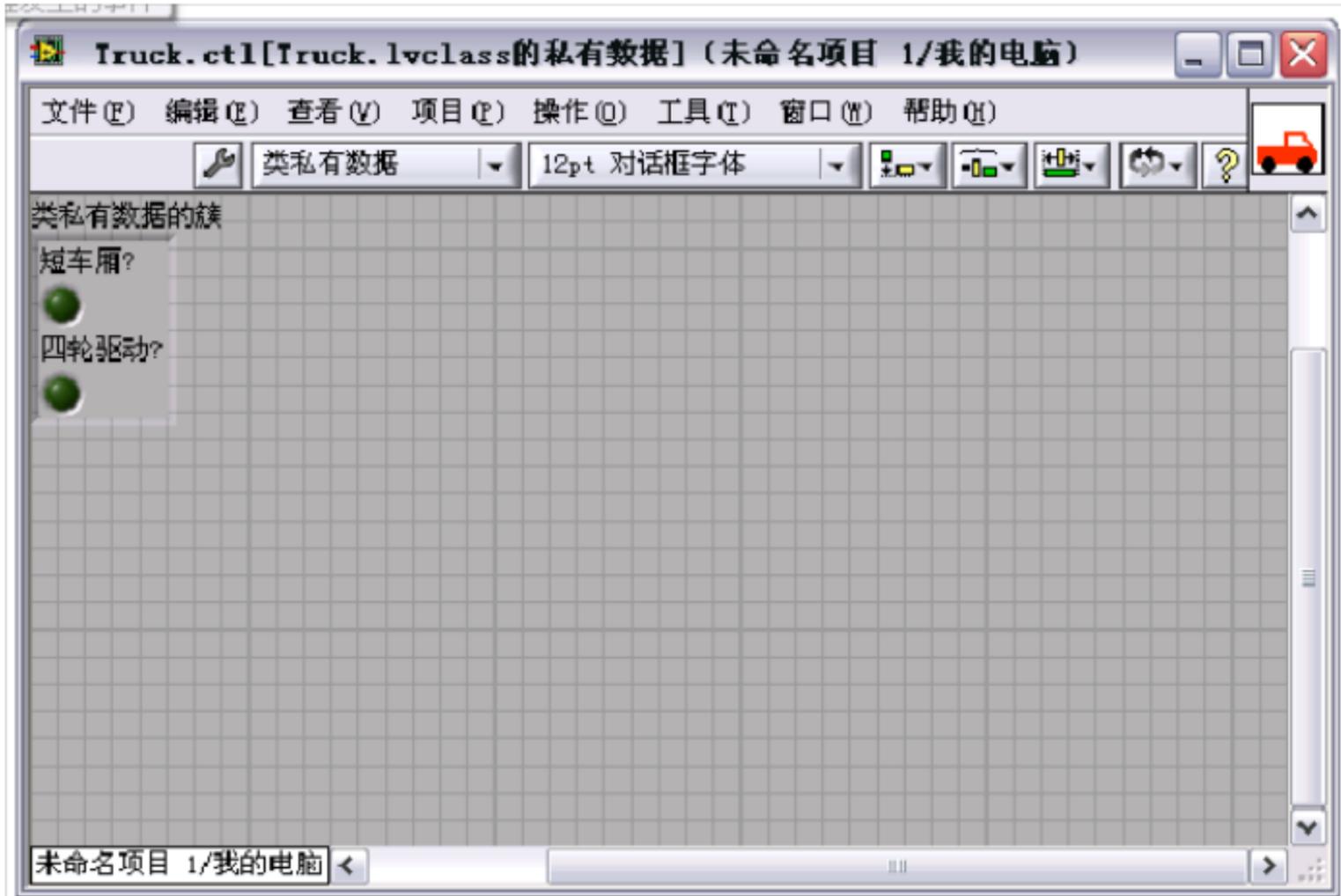
- 新建 ?基于静态分配模板的 VI - LabVIEW将生成一个新成员 VI，该 VI 带有错误输入 簇、错误输出 簇、一个用于错误处理的条件结构，以及输入 LabVIEW类和输出 LabVIEW类。与创建动态分配 VI 相反，LabVIEW不将动态分配 VI 的连线板上的输入和输出接线端设置为动态。
- 新建 ?用于数据成员访问的 VI - 打开创建访问器对话框。通过该对话框快速创建用于访问 LabVIEW类数据的成员 VI。

注： 使用该选项之前必须先保存新建的 LabVIEW类。如未保存新类，LabVIEW将用于数据成员访问的 VI 选项灰暗显示。

- 新建 ?用于重写的 VI - 创建一个重写祖先类成员 VI 的成员 VI。LabVIEW使用父 VI 的图标对子类的图标进行覆盖，创建新 VI 的图标。

注： 若不存在可重写的有效成员 VI，LabVIEW将禁用新建 ?用于重写的 VI 选项。更多关于“动态 VI”和“重写”的信息见继承一节。

右键单击前面板或程序框图上的常量或控件，从快捷菜单中选择显示类库，在项目浏览器窗口高亮显示相关类。如当前类不属于某个 LabVIEW项目，LabVIEW将打开一个类窗口显示该类。



祖先类的数据是私有的，必须使用祖先类提供的函数（成员 VI）才能修改这些数据。子孙类的成员 VI 可以调用祖先类任何“公共”型的成员 VI，就像调用 LabVIEW 中的其它 VI 一样。子孙类的成员 VI 也可以调用祖先类“保护”型的成员 VI。若指定一个祖先类成员 VI 为“保护”型，则其任何子类的成员 VI 可以调用其方法，但该类继承层次结构以外的任何其它 VI 都不能调用其方法。如需访问卡车类从汽车类继承而来的 齿轮数量，可在汽车类中创建一个“公共”型或“保护”型的成员 VI，比如 Get Gears.vi。在 Get Gears.vi 的程序框图中可对汽车类解除捆绑，从而得到 齿轮数量。然后将 齿轮数量 分配到连线板的一个输出接线端，用这种方法，汽车类的子孙类（例如，卡车类）就可访问汽车类的某个私有数据（例如，齿轮数量）了。

在汽车类中创建访问数据成员的成员 VI，即可访问齿轮数量。创建成员 VI 时，勾选创建访问器对话框的通过属性节点实现复选框。然后，将卡车类连接至属性节点，右键单击属性接线端并选择 选择属性 ? 齿轮数量。

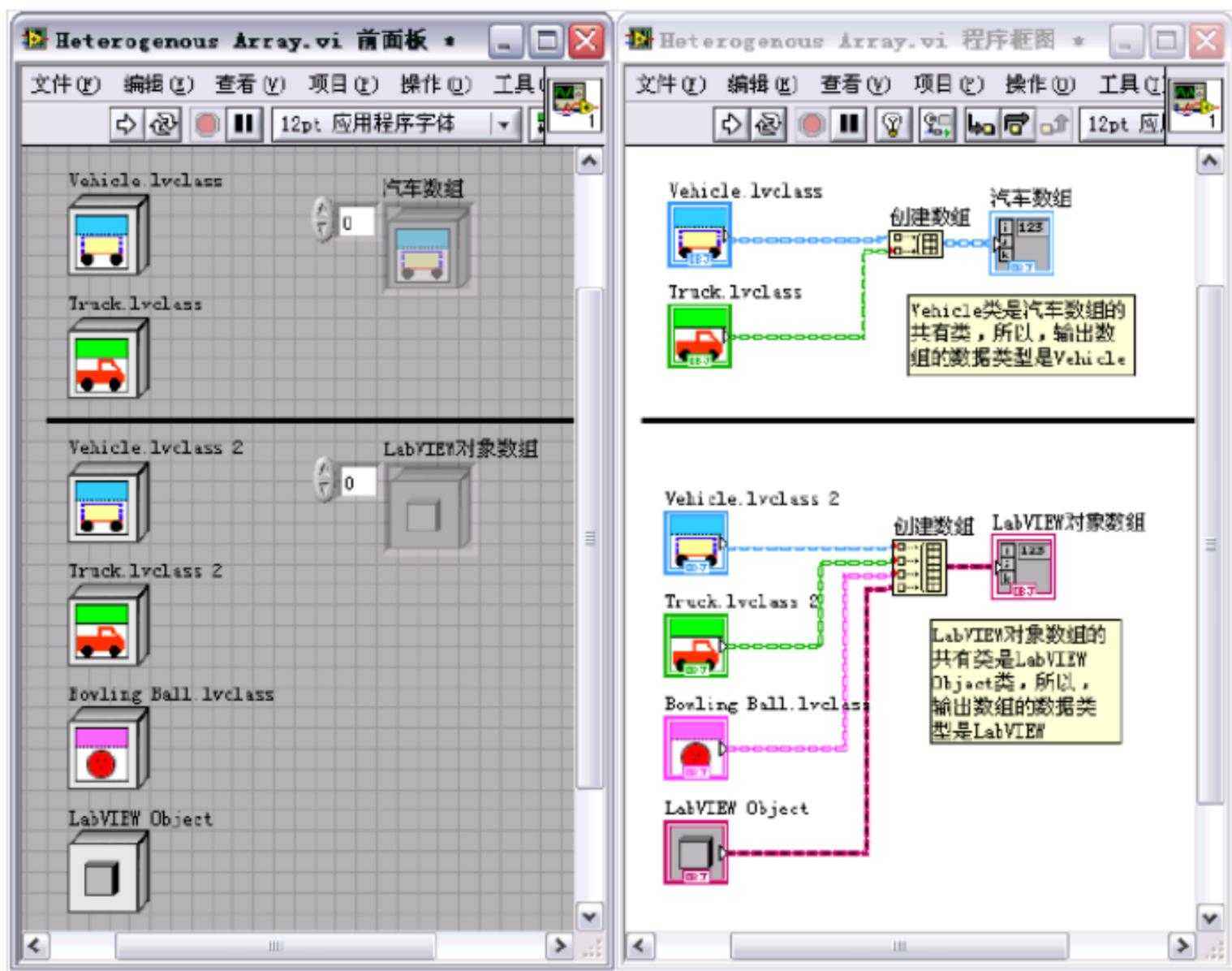
注：LabVIEW类不可调用另一个 LabVIEW类的“私有”型成员 VI，即使是父类，也无法调用其子类的私有成员 VI。类的“私有”成员 VI 只能由这个类的其它成员 VI 在程序框图中调用。

## 2.5 LabVIEW 对象

短语 LabVIEW对象是一个特定的类的名称。LabVIEW面向对象编程中，LabVIEW对象是继承树的根类。默认状态下，所有 LabVIEW类都是从“LabVIEW对象”继承而来的。通过“LabVIEW对象”创建的 VI，能对多个 LabVIEW类执行通用的操作。例如，可创建一个由若干 LabVIEW类构成的数组，该数组的类型是某个基类，数组中的数据可以是这个基类或它的任何子孙类类型的元素，因此数组中的数据是异构的。如果一个数组的类型为“LabVIEW对象”，则该数组可包含汽车类、卡车类和保龄球类。保龄球类并不是从汽车类或卡车类继承而来的，因此 LabVIEW将创建一个通用于这些类的最近的祖先基类，此处便以“LabVIEW对象”为基类。

下图显示了汽车类数组，一个包含汽车类和卡车类的数组。由于卡车类是从汽车类继承而来的，汽车类就是通用于这两个类的最近的祖先基类。该图还显示了包含“LabVIEW对象”类、汽车类、卡车类和保龄球类的 LabVIEW对象数组。保龄球类不是从汽车类或卡车类继承而来的，但所有的这三个类都是从

“LabVIEW对象”这个根类继承而来，因此LabVIEW对象数组的类型是“LabVIEW对象”。



## 2.6 设置继承

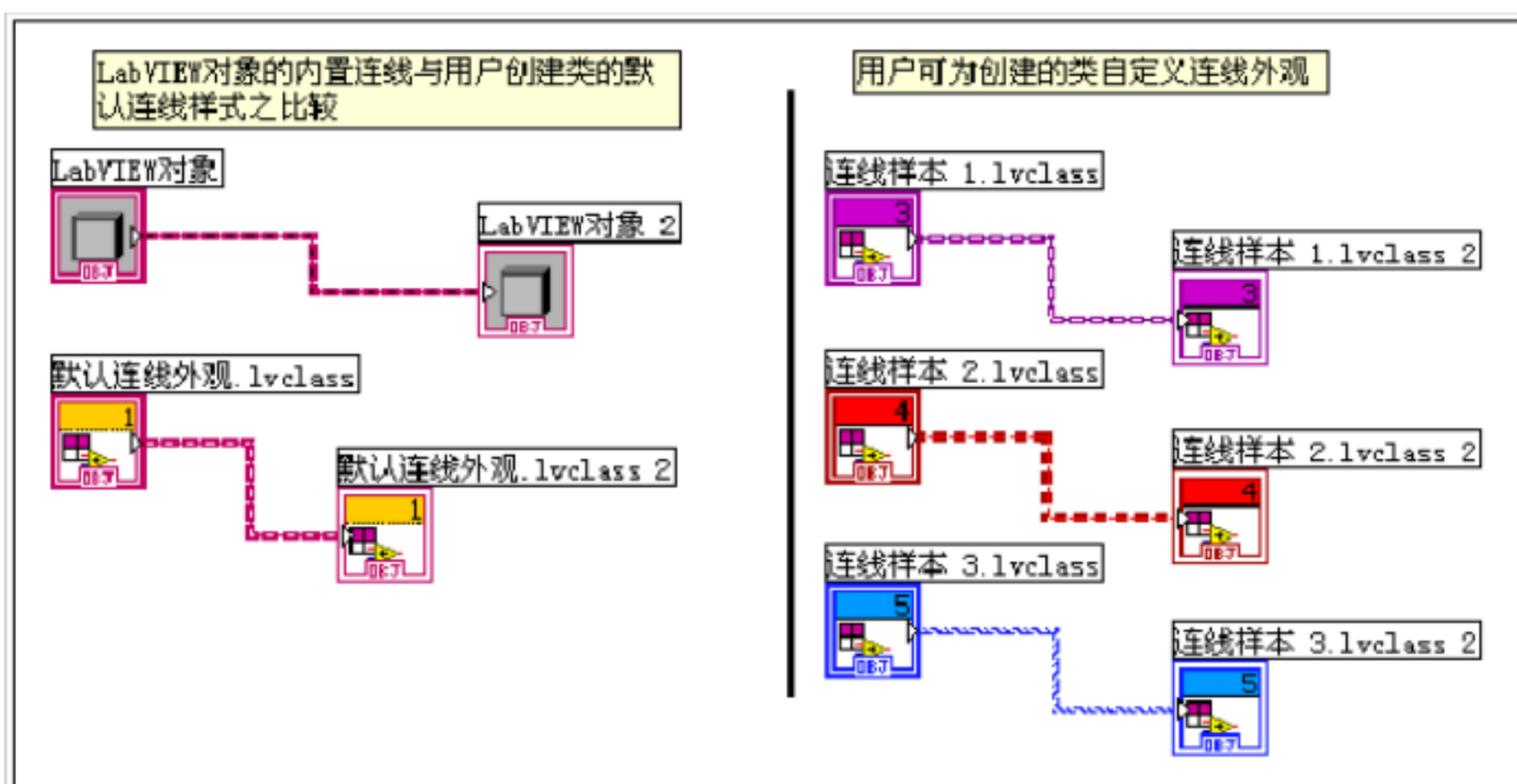
默认状态下，所有LabVIEW类都是从“LabVIEW对象”继承而来的。如果要更改一个类的继承关系，必须在创建该类之后更改继承。通过类属性对话框，可设置类的继承关系和其它选项。在LabVIEW类层次结构窗口中，可查看LabVIEW类的层次结构。类继承的层次结构可包括下列类型的类。

- 父类 - 供其它LabVIEW类继承数据、“公共”型成员VI和“保护”型成员VI的LabVIEW类。

- 子类 - 继承父类的公共和受保护成员 VI 的 LabVIEW类。除非父类提供访问 VI，否则子类不继承父类的私有数据。
- 兄弟类 - 和一个 LabVIEW类继承同一个父类的另一个 LabVIEW类。
- 祖先类 - 一个 LabVIEW类的上一层（父类）、上二层（父类的父类）、上三层等等。“ LabVIEW对象 ” 是所有 LabVIEW类的始祖。
- 子孙类 - 一个 LabVIEW类的下一层（子类）、下二层（子类的子类）、下三层等等。

## 2.7 连线外观

类定义了新的数据类型。在程序框图中，通过类定义的数据类型采用默认的 LabVIEW类连线外观，或者继承父类的连线外观。通过类属性对话框可对 LabVIEW类更改连线外观。适当地更改不同 LabVIEW类的连线外观，可提高程序框图的可读性。而使用过多的连线色彩和连线模式将破坏程序框图的可读性。下图左侧显示了 LabVIEW内置的连线外观，右侧显示了自定义连线外观的样例。



关于在 LabVIEW中避免过多连线和色彩的技巧，见 LabVIEWStyle Checklist 。

## 2.8 动态和静态分配成员 VI

方法是在对象上执行的操作。在 LabVIEW面向对象编程中，方法是用户创建的成员 VI。成员 VI 在 LabVIEW类的数据上进行运算。某些方法可用单个 VI 定义。这些方法称为静态分配方法，因为 LabVIEW每次调用的是同一个 VI。有时也可在类层次结构的多个 VI 中定义同名的方法。这些方法称为动态分配方法，因为直到运行才可确定 LabVIEW调用的是哪一个 VI。动态分配方法和多态 VI 类似：多态 VI 根据连入数据的类型来确定调用哪一个 VI；动态分配方法在运行时根据输入接线端到达的数据确定调用类层次结构中的哪一个 VI。

通过设置成员 VI 的连线板，成员 VI 既可指派为静态，也可指派为动态。若连线板上包含一个动态分配的输入接线端，则该成员 VI 是动态分配方法的一部分。如连线板上没有动态分配输入接线端，则该成员 VI 定义了一个静态分配方法。

一个 LabVIEW类继承另一个 LabVIEW类时，子类将继承父类中定义的所有“公共”和“保护”型的方法。通过在子类中创建和父类成员 VI 相同名称的成员 VI，可定义该方法的子类实现。

由于 LabVIEW通过单个 VI 定义静态分配方法，子类成员 VI 的名称不可与祖先类的静态分配成员 VI 的名称相同。例如，当父类“汽车”中包含了“开门” VI 这一静态分配成员 VI，则子类“卡车”便无法将名为“开门”的 VI 作为其成员 VI。由于“卡车”从“汽车”继承了其成员 VI，故“开门” VI 这一方法已在“卡

车”上定义。如在程序框图中将静态分配方法作为子 VI 调用，则调用这些静态分配方法和调用普通子 VI 没有任何区别。

对一个方法可定义多个动态 VI，可在继承层次结构中的每一层对该方法定义一个动态分配 VI。如动态分配成员 VI 在父类中定义，且也在子类中定义，则子类的执行将覆盖或扩展父类的执行。在以下范例中，“汽车”类和“卡车”类都定义了动态分配方法 Set Make VI。若在程序框图中将一个动态分配 VI 作为子 VI 调用，在编辑状态下，该节点和一般子 VI 调用没有区别。然而如果运行 VI，则流入动态分配输入接线端的数据将决定 LabVIEW 会调用类层次结构中的哪一个动态成员 VI。LabVIEW 类连线可传递本身及任意子类所允许的数据类型，不同的数据类型被定义了不同的成员 VI，该节点将根据连线上的数据类型执行相应的动态分配 VI。阅读下列范例。只有 Set Make VI 的“汽车”类在主 VI 的程序框图上。第一次循环时，由于“汽车”类的数据在类连线上，故 LabVIEW 执行 Set Make VI 的“汽车”类。第二次循环时，由于“卡车”类的数据在类连线上，故 LabVIEW 执行 Set Make VI 的“卡车”类。

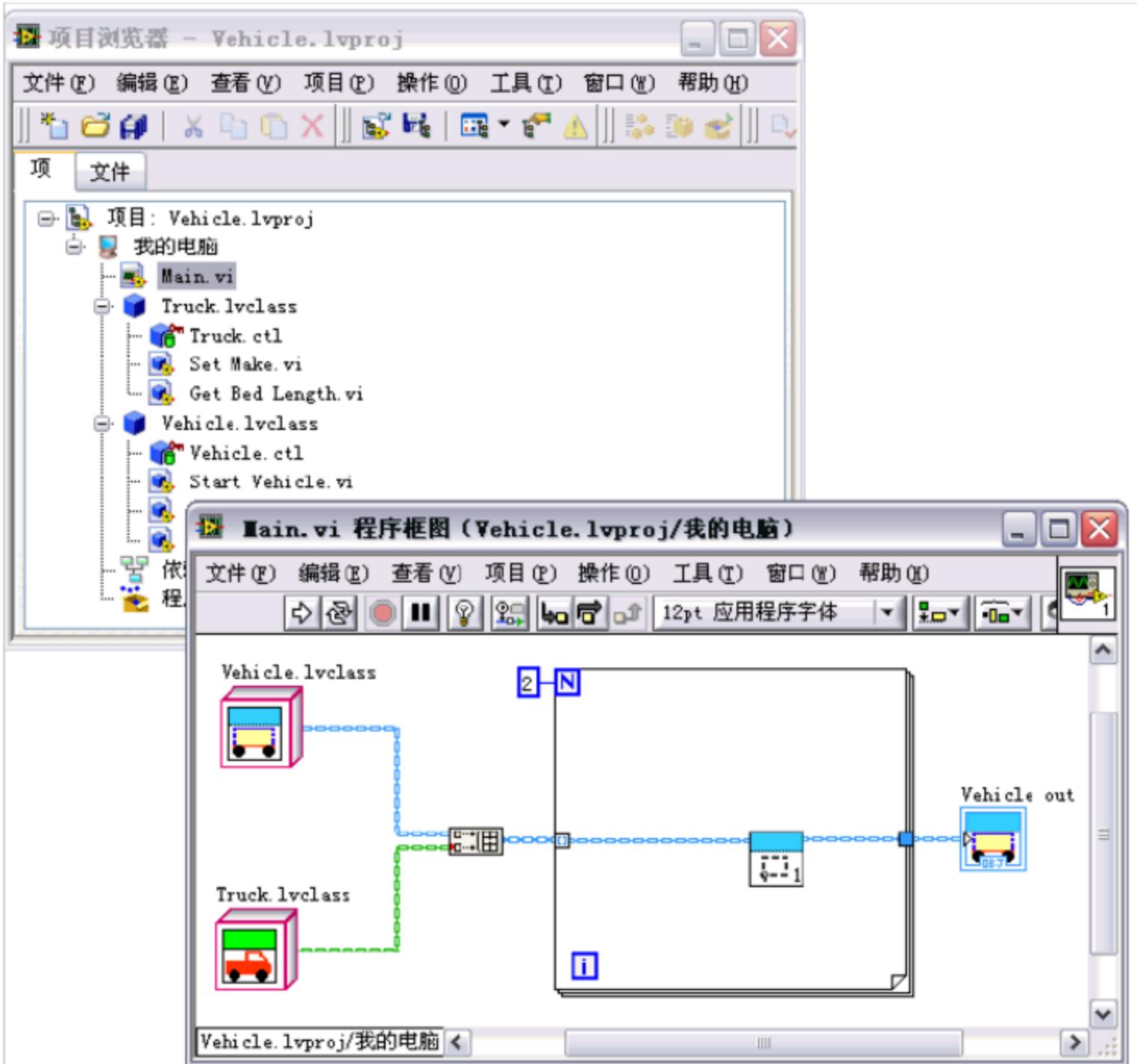
如父类定义了一个动态分配 VI 但是不提供该 VI 的执行，每个子类必须覆盖其父 VI。在许多情况下，用户需提供父类 VI 的有意义的动作。例如，如有一个 Shape 类，定义 Area VI。Area VI 返回 Shape 对象的面积。没有一个公式可计算任意形状的面积。所以，每个继承类必须用相应的面积计算公式覆盖 Area VI。如创建了一个子类 Circle，子类 Circle 就必须提供一个 Area VI，计算  $\pi * \text{radius} * \text{radius}$ 。

如不定义父类 VI 的执行，父 VI 就仅仅是所有覆盖 VI 必须匹配的连线板和 VI 属性。每个子类必须提供一个类似的覆盖 VI。要确保 LabVIEW强制执行该要求，可在父类中标示 VI 为子类必须覆盖的 VI。

一些子类可能无法覆盖成员 VI 的功能。例如，如 Shape类有子类 Quadrilateral，无法提供四边形面积计算公式，除非知道四边形为特定类型的四边形。用户可使 Quadrilateral 类迁移覆盖要求至其子类，以避免在 Quadrilateral 类中创建 Area VI 的空执行。右键单击子类，从快捷菜单中选择属性，打开对象属性对话框。在继承页上，勾选 将全部重写要求传递至子孙类 复选框。LabVIEW将要求 Quadrilateral 的所有子类（例如，梯形和矩形）覆盖 Area VI。

要求子孙类覆盖父类成员 VI 对成员 VI 的运行没有影响。

注：LLB无法包含同名文件。因此，若类层次结构中存在同名的动态成员 VI，这些 VI 不可放置在同一个 LLB中。



注： 如子类的 VI 覆盖了父类的 VI，子类的 VI 必须和父类的 VI 在以下方面吻合：重入设置、首选执行设置、优先级设置、连线板接线端、连线板模式，以及访问范围。

双击程序框图上的一个动态分配子 VI 以显示选择实现对话框。通过该对话框可查看当前内存中动态分配子 VI 的所有实现，然后将该子 VI 的一个或多个实现打开。

如选择 新建?重写 VI 而创建一个 VI 以覆盖一个动态分配成员 VI 的父实现，则将创建另一个动态分配成员 VI，因为重写父类成员 VI 的 VI 与其父类 VI 同名

且具有动态分配接线端。LabVIEW会自动将调用父类方法节点和正确的动态分配输入和输出类接线端及其它匹配祖先类 VI 所需的接线端放在程序框图上。若不存在可重写的祖先类成员 VI，则 LabVIEW将禁用 重写 VI 选项。

## 2.9 动态分配输出

右键单击连线板上的输出接线端并选择 动态分配输出（推荐），可将 LabVIEW 类的输出接线端标记为动态。将一个含有动态分配输出端的 VI 作为子 VI 调用时，动态分配输出端的数据将转换为与动态分配输入端相同的类型。例如，将汽车类连接到一个动态分配输入接线端，则该成员 VI 的输出数据类型将和输入数据类型相同，即为汽车类类型。动态分配输入端和动态分配输出端之间的数据是可修改的。然而，为了确保 LabVIEW类运行时的安全，动态分配输入端的数据必须流入所有动态分配输出端。同时，为了确保 LabVIEW仅从动态分配输入端读取一次，仅向动态分配输出端写入一次，不可将动态的程序框图接线端放置在结构之中。

注： 调试动态分配成员 VI 时，可检查动态分配输入端和动态分配输出端之间的连线是否出错。若一条连线从动态输入出发，且未通过任何能改变运行时数据类型的函数，则该连线的背景颜色将是灰色而不是通常的白色。若该连线通过一个能改变其数据类型的函数，则连线的背景颜色将变成红色。

为使动态输出正常运作，不可改变 LabVIEW类的数据类型。

若已知成员 VI 的程序框图中 LabVIEW类的输出数据类型不同于输入数据类型，则须确保连线板上的 LabVIEW类动态输出接线端设置为 推荐 而不是 动态分配输出（推荐）。例如，若 LabVIEW类的输入是汽车类而输出是卡车类，则必须改变连

线板上 LabVIEW类的默认接线端。此外，也可从空 VI 创建成员 VI，此时可手动设定连线板接线端。

注：如在一个有动态分配输入和动态分配输出的成员 VI 中使用条件结构或事件结构，不能在输出隧道上选择未连线时使用默认选项。若在输出隧道上使用未连线时使用默认，LabVIEW将断开该 VI。必须连接结构中的所有分支。考虑将隧道配置为自动连接输入和输出隧道。

## 2.10 动态分配的内存拷贝优化

如上所述，创建动态分配的方法时，每个子类都会继承父类上定义的所有公共方法和受保护的方法。子类可覆盖或扩展这些成员 VI。VI 调用动态分配的方法时，LabVIEW只有当运行时才知道将调用哪个方法。LabVIEW将优化调用方 VI 的内存分配，并认为子类含有的任何成员 VI 的设置与父类中成员 VI 的设置相同。如父类 VI 的输入为常量，调用方将认为在所有子类 VI 中输入也是常量。如父类成员 VI 的输入返回为输出，调用方 VI 认为所有子类成员 VI 也会有相同的操作。

如上述假设有误，则优化失败。例如，如创建了一个成员 VI，其中包含未改变的输入。这将使调用方 VI 认为所有输入都不改变，即使每次覆盖都会改变部分或所有输入。或者，如未将任何输入连接至父类成员 VI 的输出，调用方 VI 将认为没有任何输出与输入共享内存，即使在相邻的 VI 之间，上一个 VI 的输出可能是下一个 VI 的输入。LabVIEW必须创建代码来处理子类中导致优化失败的非预期行为。

更好的优化是指写入父类 VI 时，在子类 VI 中发生尽可能一致的操作。创建一个动态分配方法，可使用元素同址操作结构明确表示各个接线端的作用。将元

素同址操作结构放在父类的动态分配的 VI 中，然后将元素同址操作结构节点对置于元素同址操作结构中。这些节点对用于通知 LabVIEW 哪些输入应连接至哪些输出，哪些输入应视为常量，哪些输入应视为修改过的量。然后，LabVIEW 开始优化调用方 VI。

### 三、开发 LabVIEW 类

从概念上来说，LabVIEW 面向对象编程和其它面向对象编程语言相似。但由于 LabVIEW 是数据流图形化编程环境，LabVIEW 对类数据的操作和交互，以及 LabVIEW 类代码的调试方法和其它语言有所不同。

LabVIEW 中的对象由值来传递，而不是由引用来传递。LabVIEW 按照簇和数组的操作规则创建对象的副本。

#### 3.1 构造函数和析构函数

构造函数和析构函数在 LabVIEW 面向对象编程中是隐含的。不需要调用构造函数来对 LabVIEW 类数据进行初始化。每当需要对一个类进行初始化时，LabVIEW 会调用一个默认的构造函数。通常情况下，类在前面板的相应控件或程序框图的相应常量中初始化。LabVIEW 用私有数据控件中设定的默认值来对类的值进行初始化。当 LabVIEW 不再需要 LabVIEW 类中的信息时，LabVIEW 将以处理簇和数组同样的方法进行内存释放。如需将类数据设定为其它值，必须创建一个成员 VI 以对类数据设定新值。例如，若在创建汽车类时，将其中 齿轮数量 的默认值设定为 3，而又希望对卡车类从汽车类继承而来的 齿轮数量 指派一个不同的值，这时就必须创建一个成员 VI 以改变汽车类的值。设置新值的另一种方法是：创建一

个没有输入，以类为输出的成员 VI，将需要指派给 齿轮数量 的值设定在输出类中，从而可创建一个以该输出类为数据类型的程序框图常量。

### 3.2 平化和还原数据

LabVIEW以平化数据的形式存储数据。平化至字符串及从字符串还原函数可以处理所有的类数据类型。LabVIEW自动平化和还原数据。所有类型描述符保留了LabVIEW中已平化数据的类型，而LabVIEW类的平化数据本身不但保留了类型信息，而且保留了类的版本信息。由于LabVIEW类保留了用于还原LabVIEW类的相关信息，如果LabVIEW类因被移动或删除而无法找到，LabVIEW将无法还原数据并将显示错误信息。类似于LabVIEW丢失子VI时的情形，打开成员VI时，若某控件的LabVIEW类数据已丢失，该控件将变灰。断开控件所对应的LabVIEW类被加载之后，数据将被还原而控件将不再被断开。

注：也可使用平化至XML和从XML还原函数处理类数据。

### 3.3 实施数据变异

若试图向LabVIEW类控件中写入数据的LabVIEW类比内存中的LabVIEW类版本更新，则被写入的控件将发出警告。当LabVIEW类私有数据控件的数据类型或类的继承关系发生改变时，该类的版本号将被增大。通过类属性对话框，可查看LabVIEW类的版本号。出现以下情况时，LabVIEW将自动增大类的版本号：

- LabVIEW类的继承关系发生改变。

- LabVIEW类的私有数据控件发生改变。私有数据控件的改变包括：添加、替换、重新排序或删除控件；改变控件的表示法；更新已修改的自定义类型。
- 在新版本的 LabVIEW中加载旧版本的 LabVIEW类。
- 通过类属性对话框可手动增大类的版本号。递增的版本号表示类版本的更新。

若对一个类进行重命名，则 LabVIEW将把该类当作一个新类，删除该类的变异历史，并将版本号重新设定为 1.0.0.0。执行以下任意一种操作时，LabVIEW类将被重命名：

- 通过下拉菜单的 文件 ?另存为，对类进行重命名。
- 将类移入所属库。
- 将类移出所属库。
- 重命名磁盘上的 .lvclass 文件。

注：如不作任何改动，将类改回原来的类名，则改回原名的类和原类仍有所不同，因为原类的变异历史已经丢失。例如，假设将汽车类重命名为“汽车#1类”，然后将类名从“汽车 #1”改回“汽车”，则新的汽车类和原来的汽车类不再是同一个类。

对使用 LabVIEW类开发应用程序的 LabVIEW类用户来说，LabVIEW跟踪 LabVIEW类的版本号是很有用的。例如，假设应用程序中有一个 LabVIEW类，该类的私有数据控件使用了无符号 32 位整型。然而，LabVIEW类开发人员发送的某个版本的

LabVIEW类中，私有数据控件中的数值控件变成了双精度浮点型。由于 LabVIEW 跟踪了版本的变化，并能对所有 LabVIEW类的数据进行平化和还原，用户可以直接用新版本替换旧版本，不必作任何修改即可运行程序。

注： 如将数据从某个 LabVIEW类的未来版本还原， LabVIEW将返回错误。

例如，当内存中某个 LabVIEW类的版本号为 1.0.0.2 ，而数据的版本号为 1.0.0.3 时，将可能出现该错误。

LabVIEW采取如下方式处理 LabVIEW类不同版本数据的变异：

- 如从 LabVIEW类层次结构中移除一些类，则 LabVIEW将从原来的类层次结构中删除这些类的平化数据。例如，假设类 C继承了类 B，而类 B继承了类 A。如将类 C更改为直接继承类 A，则 LabVIEW将删除类 C中来源于类 B的平化数据。
- 如在 LabVIEW类层次结构中添加新层，则 LabVIEW将把被添加类的默认数据插入到原来的类层次结构中。例如，假设类 C和类 B都继承了类 A。如将类 C更改为继承类 B，则 LabVIEW将对类 C插入类 B的默认数据。
- 如在类私有数据簇的末尾（按 tab 键顺序）添加一个元素，则 LabVIEW将还原旧的数据并添加类的默认数据。
- 如在类私有数据簇中删除一个元素并且不添加任何新元素，则 LabVIEW将还原旧的数据并删除该元素相关的信息。
- 如改变类私有数据簇的 tab 键顺序，则 LabVIEW将对还原后的相应数据进行重新排序。

- 如在同一次修改中添加、删除、替换或重新排序类的私有数据簇， LabVIEW 将创建并执行一个过程，通过合并以上步骤对数据实施相应的变异操作。

LabVIEW类不同版本的变异总是按顺序（例如从版本 1.0.0.2 到 1.0.0.6 ）实施的。

注： 当类的版本号增大以后，将无法对 LabVIEW类进行恢复。由于没有足够的信息，LabVIEW无法撤消 LabVIEW类的输入和显示控件中数据的更改。如需恢复当前的更改，请不要保存任何更改。卸载该类及所有引用该类的 VI，然后从磁盘重新加载该类。

关于编辑 LabVIEW类时，LabVIEW进行数据变异的详细信息，见 ni.com 的技术支持文档。

### 3.4 强制转换 LabVIEW类

通过转换为通用的类函数可将 LabVIEW类向上转换；通过转换为特定的类函数可将 LabVIEW类向下转换。这些函数还可用于对包含继承层次结构的引用数据类型（例如 VI 服务器控件引用）进行操作，功能是一样的。使用“转换为通用的类”函数并不改变数据，但传递数据的连线类型将被改变。但是，该函数在 VI 运行时不起作用。对那些按照严格代码规范需要避免数据转换的程序员而言，“转换为通用的类”函数提供了一种消除强制转换点的方法。

“转换为特定的类”函数不改变数据，除非出现错误。如运行时连线上的数据不是更为特定的类，则该函数将返回一个错误并且输出数据将是连线类型的默认值。“转换为特定的类”函数的主要用途是对父类的值作类型检查。通常程序

员将同一个父类连接到多个“转换为特定的类”函数，每一个连线导向一个更为特定的类，哪一个“转换为特定的类”函数不报错，便执行该函数之后的代码，从而实现根据类型检查结果运行不同的代码。这种方法相当低效。如需用这种方法作类型检查，并拥有修改父类的权限，可在父类添加一个动态成员 VI，并让每一个子类根据功能需要重写这个动态成员 VI。

如 LabVIEW在运行时检测到用户将子类对象连接至接受父类对象的子 VI，LabVIEW将自动把子 VI 输出向下转换为子类对象。自动向下转换不必使用“转换为特定的类”函数。但是，只有 LabVIEW确保连接至子 VI 的类对象与子 VI 接受的输入兼容时，自动向下转换才发生。例如，如将类对象保存在变体中，然后将变体数据连接至子 VI，LabVIEW不能确保子 VI 包含的数据与变体原本存储的数据为同一种类型。使用保留运行类函数帮助 LabVIEW检查连接至子 VI 的类对象与子 VI 接受的类对象为兼容对象。如两个对象不兼容，函数将返回错误，并将类的输出数据设置为子 VI 接受的父对象。也可将该函数与数据值引用配合使用。数据值引用读取 / 写入元素边框节点必须预留运行类型。可使用“保留运行类”函数检查连接至“数据值引用写入”节点的类对象是否与连接至“数据值引用读取”边框节点的类对象相互兼容。

注：LabVIEW无法自动向下转换递归子 VI。如在动态分配 VI 的程序框图上或在元素同址结构中调用递归子 VI 传递数据值引用时，自动向下转换会引起调用 VI 的断开。在递归子 VI 的程序框图上使用“保留运行类”函数，验证连接至子 VI 的类与子 VI 接受的类是兼容的。如两个类兼容，在递归链之外调用 VI 不会使 VI 断开。

### 3.5 锁定和解除锁定 LabVIEW类

通过锁定 LabVIEW类可阻止 LabVIEW类用户查看那些作为应用程序内部实现的成员 VI。锁定 LabVIEW类可防止对私有成员 VI 的访问，从而预防引入应用程序的错误。类似于项目库，对 LabVIEW类进行密码保护时，类的成员 VI 并未受到密码保护。必须对类的成员 VI 单独设置密码保护。

如一个 LabVIEW类被锁定，则当光标在类连线上移动时，即时帮助窗口中仅显示连线数据类型。当光标在一个未锁定的 LabVIEW类上移动时，即时帮助窗口将显示该类的私有数据，以及该类所有未锁定的父类的私有数据。类似规则也适用于 LabVIEW在“通用探针”上显示的信息。关于对 LabVIEW类使用“通用探针”的相关信息见在 LabVIEW类中使用探针。

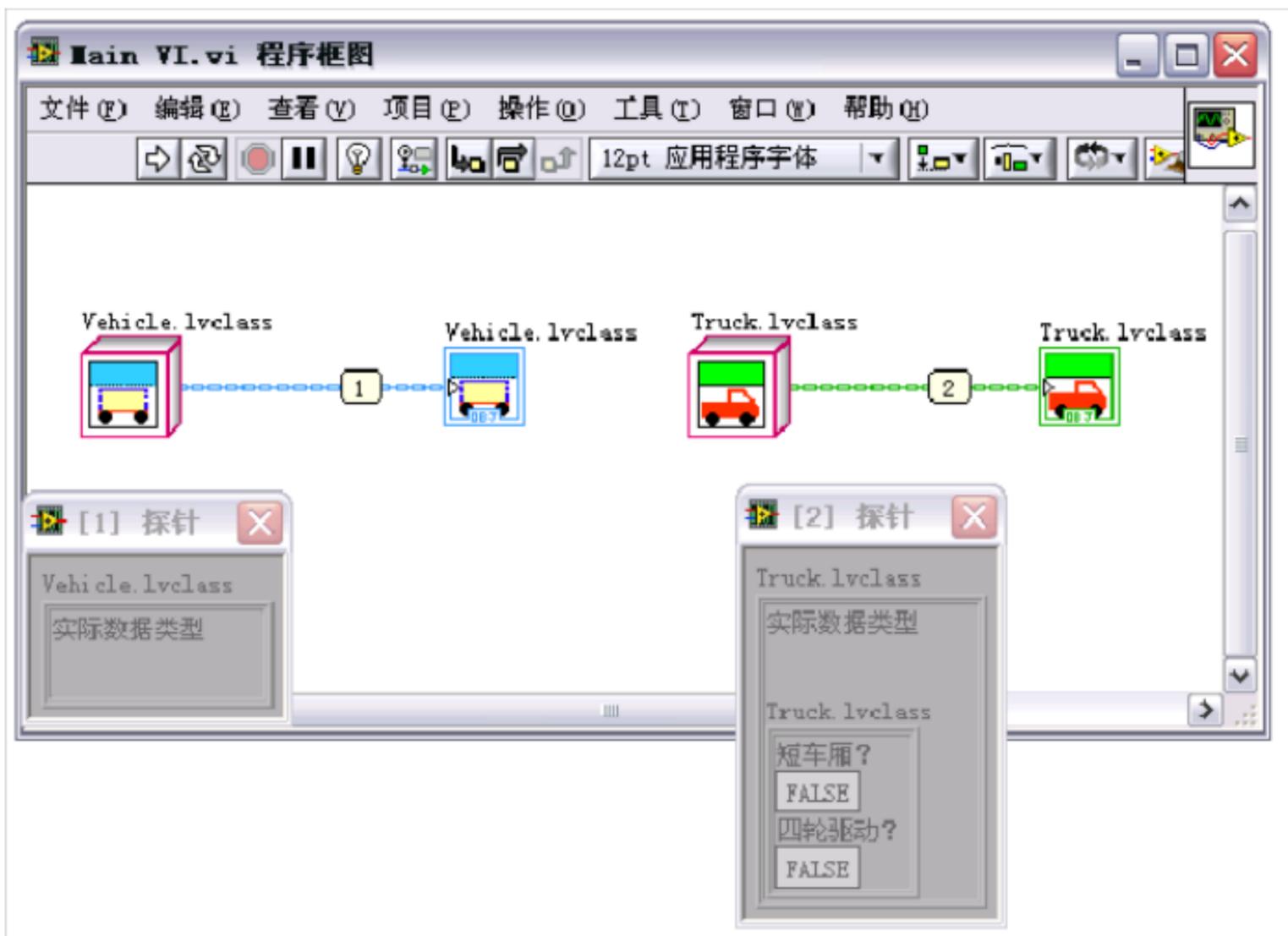
注： 请注意，光标停留在 LabVIEW类连线和停留在 LabVIEW类控件时，即时帮助窗口中显示的信息是不同的。 LabVIEW类控件为最终用户提供类的说明信息，而 LabVIEW类连线则为 LabVIEW类用户提供关于数据类型的补充信息。

LabVIEW类用户仅可查看开发人员提供的信息，因而开发人员须在开发 LabVIEW类时向用户提供足够的文档信息。在类属性对话框中可添加类的说明信息，在成员 VI 各自的 VI 属性对话框中，可添加成员 VI 的说明信息。

### 3.6 在 LabVIEW类中使用探针

通用探针和自定义探针可用于调试 LabVIEW类。对 LabVIEW类连线可使用通用探针。 LabVIEW类连线上的通用探针显示了运行时该类的类名、类的私有数据和该类所有父类的私有数据。如对一个已锁定的 LabVIEW类使用通用探针，探针

将仅显示运行时该类的类名，以及类层次结构中未锁定的私有数据。在下图中，位于左侧的“汽车”类被锁定，LabVIEW不显示类的私有数据：齿轮数量、车门数量、颜色、制造和型号。“卡车”类未被锁定，因而LabVIEW显示该类的私有数据。请注意，LabVIEW此时并不在通用探针上显示卡车类从汽车类继承而来的私有数据，因为汽车类已被锁定。



根据需要，也可为LabVIEW类创建自定义默认探针。仅可对确定类型或祖先类型的LabVIEW类连线使用自定义探针。LabVIEW类开发人员可创建能够直接显示类私有数据，并作为类成员VI的自定义探针。LabVIEW类用户可创建自定义默认探针，用于显示类的“公共”型方法可显示的任何信息。

LabVIEW开发人员可为LabVIEW类用户将自定义探针设置为默认探针。已锁定的LabVIEW类的通用探针不允许LabVIEW类用户访问任何关于数据值的信息，

因为数据是私有的。类开发人员可创建自定义探针，再将它们设定为 LabVIEW类的默认探针以提供给类用户，通过这种方法，开发人员锁定类之后仍可适当地将类中数据显示给类用户。自定义探针必须是 LabVIEW类的成员，LabVIEW类开发人员才可将它设定为默认探针。

### 3.7 向其他开发人员和用户发布 LabVIEW类

LabVIEW类开发人员可向其他类开发人员和类用户发布 LabVIEW类。开发人员可通过多种方法发布 LabVIEW类，可选择最符合要求的方式。可用应用程序生成器来创建 zip 文件以发布一个或多个类。也可在发布之前锁定 LabVIEW类，从而限制 LabVIEW类用户对私有数据和成员 VI 的访问。锁定 LabVIEW类有助于防止将错误引入应用程序中。

提示：也可通过 .NET 互操作程序集访问 LabVIEW类。LabVIEW可为指定的 LabVIEW类生成相应的 .NET 类，可通过 .NET 程序集访问 .NET 类。