

第4章 LabVIEW的循环与结构

LabVIEW中除了拥有**C**语言中所有的程序结构外，还有一些特殊的程序结构，如事件结构、公式节点等，通过这些可以方便快捷地实现任何复杂的程序结构。

第4章 LabVIEW的循环与结构



图4-1 “结构”子选板界面

第4章 LabVIEW的循环与结构

LabVIEW中的结构放置在程序框图中，其外形一般是一个大小可以缩放的边框，当它与其他节点的连线有数据传递时，边框内的一段代码将反复执行或有条件执行或按某一定的顺序执行。结构内的该段代码则被称为**子框图**。

结构框图可以看成是个**代码容器**。容器内的代码按照某种条件反复执行。

4.1 For循环

4.1.1 For循环的组成

LabVIEW中的循环与结构位于程序框图的“函数”选板下的“结构”子选板中，如图4-1所示。

4.1 For循环

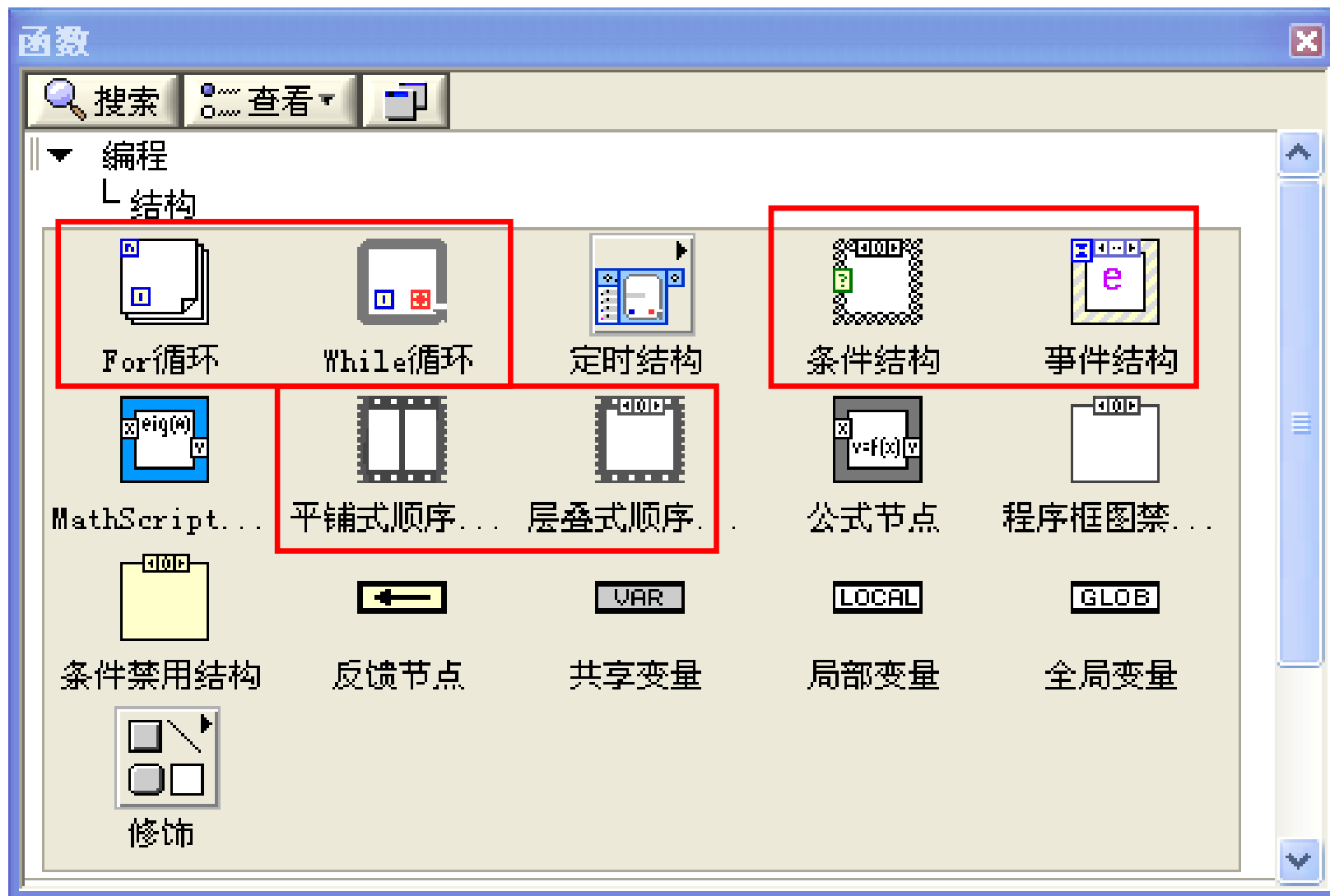


图4-1 “结构”子选板界面

4.1 For循环

For循环相当于C语言中的下列程序代码：

```
for (i=0; i<N; i++)  
{  
    };
```

N:总循环次数，**i**: 循环变量，初值为**0**，增值为**1**，循环体内为代码

4.1 For循环

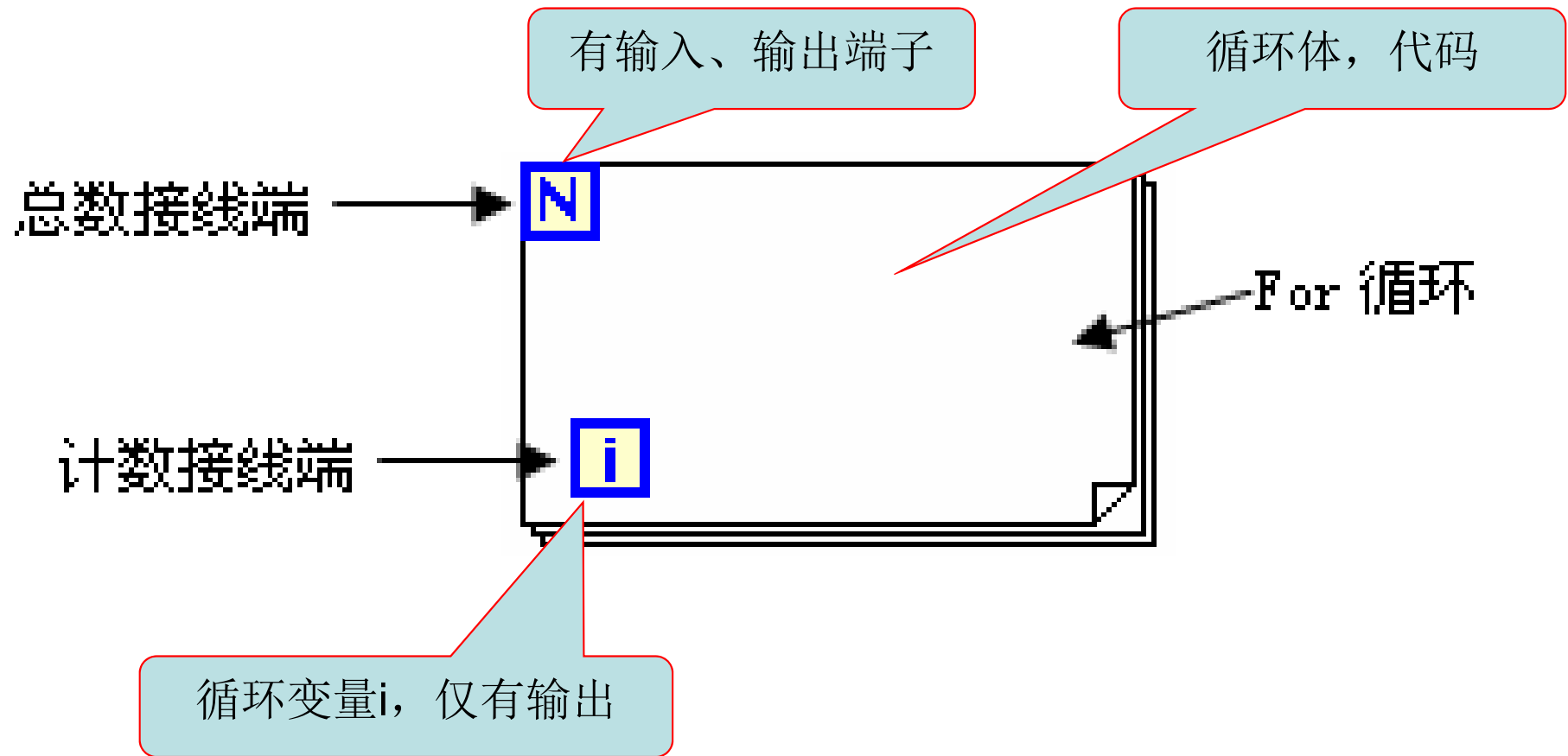
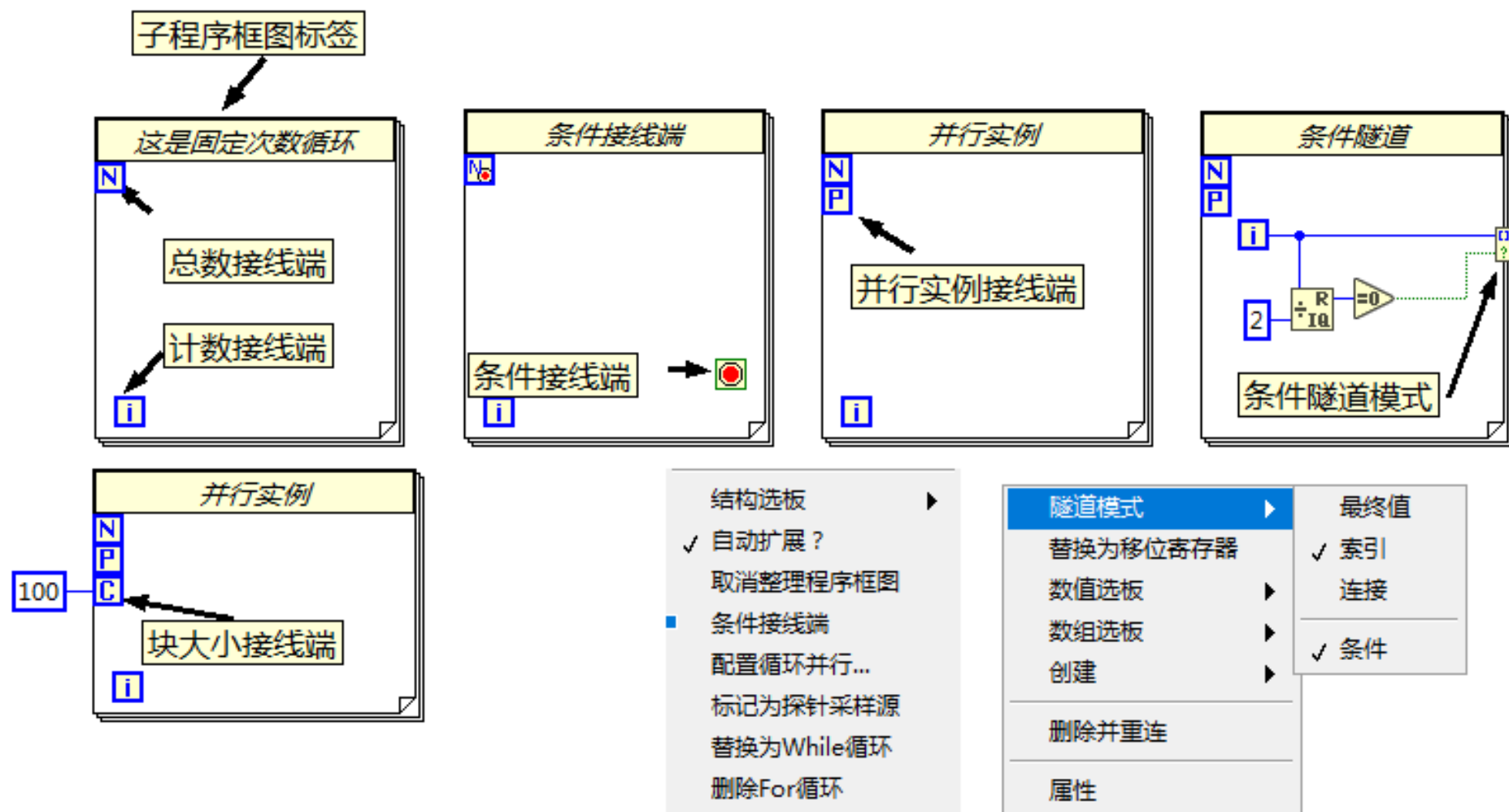


图4-2 For循环界面

最基本的**For**循环由循环框架、总数接线端（输入端）、计数接线端（输出端）组成。

4.1.1 For循环组成



4.1.2 简单For循环应用示例

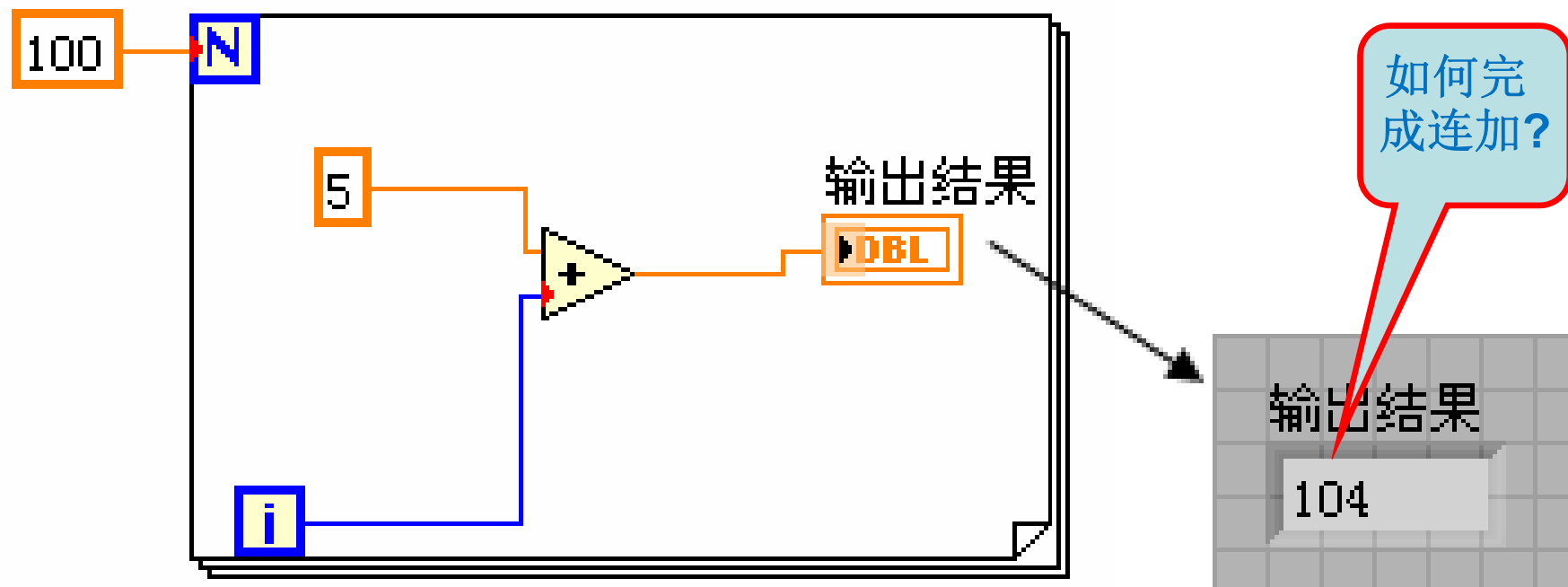


图4-3 简单For循环示例

FOR循环简单示例

4.1.3 移位寄存器在For循环内的应用

在**LABVIEW**中，不支持变量的自赋值。只能通过移位寄存器解决。

移位寄存器是**LabVIEW**的循环结构中的一个附加变量对象，其功能是将当前循环完成的某个数据临时寄存后，传递给下一个循环中相应的变量。

一般来说，移位寄存器可以存储任何类型的数据，但是连接在**同一个寄存器**两个端子上的数据必须是同一类型的。

4.1.3 移位寄存器在For循环内的应用

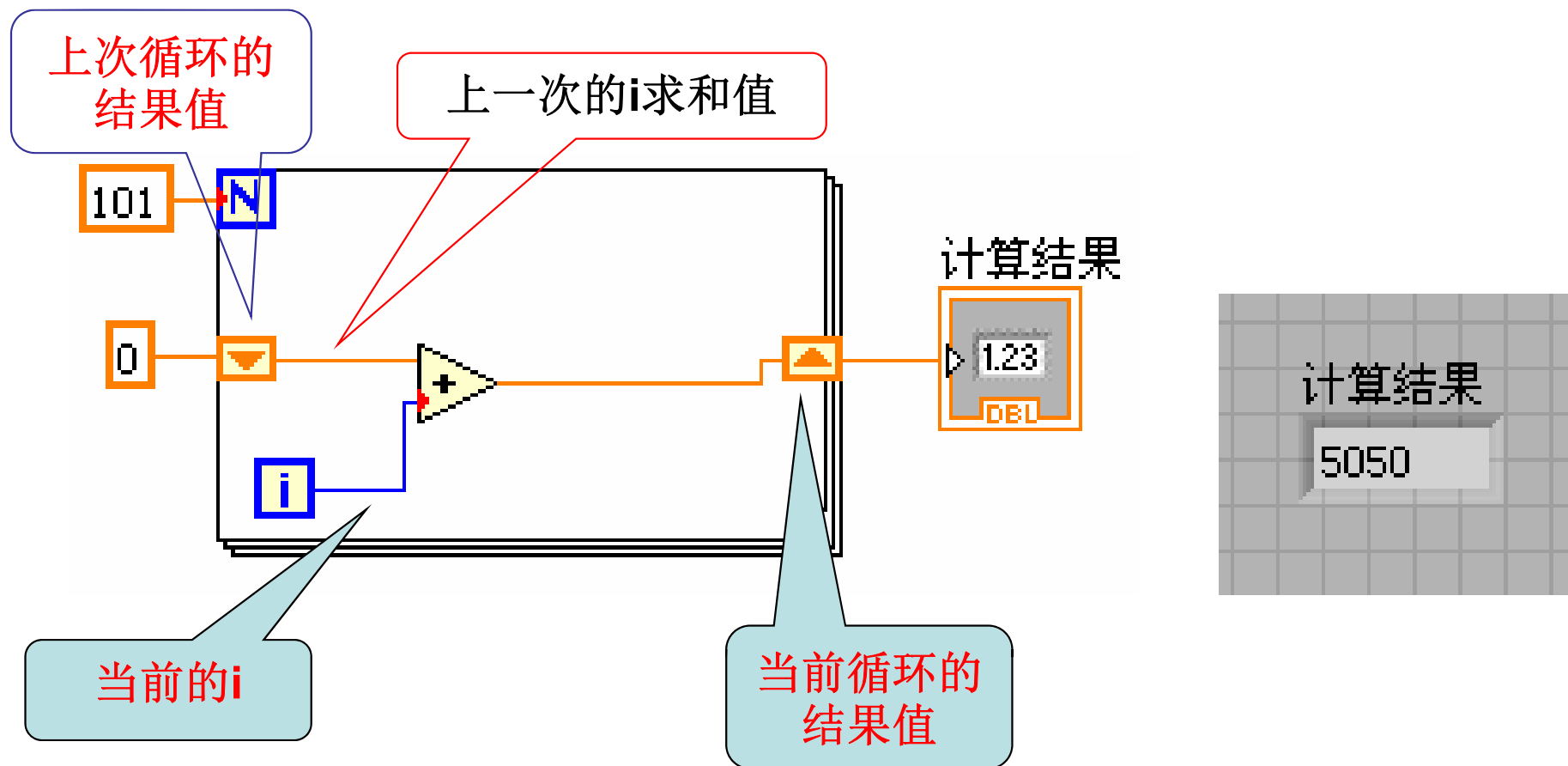


图4-5 For循环移位寄存器示例

FOR循环移位寄存器示例

4.1.4 反馈节点在For循环内的应用

和移位寄存器一样，反馈节点也是用来实现数据在前后两次循环中的传递。但与移位寄存器相比，使用反馈节点有时能让程序更加简洁易懂。

循环中一旦连线构成反馈，就会自动出现反馈节点的符号。反馈节点符号由两部分构成，分别为初始化端子和反馈节点箭头。

反馈节点在结构选板中，移位寄存器直接添加。

4.1.4 反馈节点在For循环内的应用

初始化端子  既可位于**For**循环框图内，也可位于**For**循环框图外，默认为位于**For**循环框图内。

反馈节点箭头  表示连线上的数据流动方向，它可以是正向的，也可以是反向的。

4.1.4 反馈节点在For循环内的应用

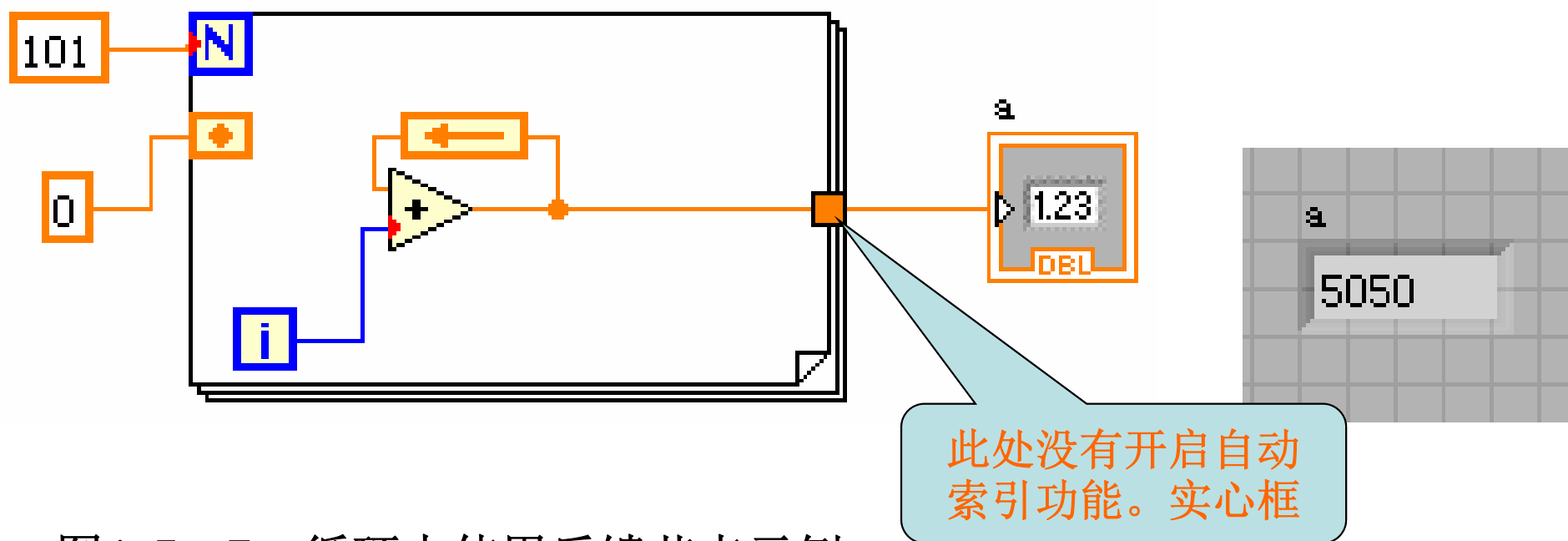


图4-7 For循环中使用反馈节点示例

反馈节点的使用

4.1.5 自动索引在For循环内的应用

自动索引的功能是使循环框外面的数组成员逐个进入循环框内，或使循环框内的数据累加成一个数组输出到循环框外面。

循环内--》循环外，元素--》数组，一维---》二维，
循环外--》循环内，数组--》元素，二维---》一维，

自动索引的值，是把每次循环的值作为一个数组元素，在完成循环后才一次性输出整个数组。

For循环的索引可通过鼠标右键单击循环边框的数据通道来启动。

4.1.5 自动索引在For循环内的应用

尽管**For**循环和**While**循环都支持自动索引功能，但其主要区别在于：**For**循环的数组**默认为能自动索引**，如不需要索引，可在数组进入循环的通道上单击鼠标右键弹出快捷菜单选择“禁用索引”选项；而**While**循环中的数组**默认为不能自动索引**，如果需要索引，可在循环的通道上单击鼠标右键弹出快捷菜单选择“启用索引”选项。另外，在创建二维数组时一般使用**For**循环而不使用**While**循环。

4.1.5 自动索引在For循环内的应用

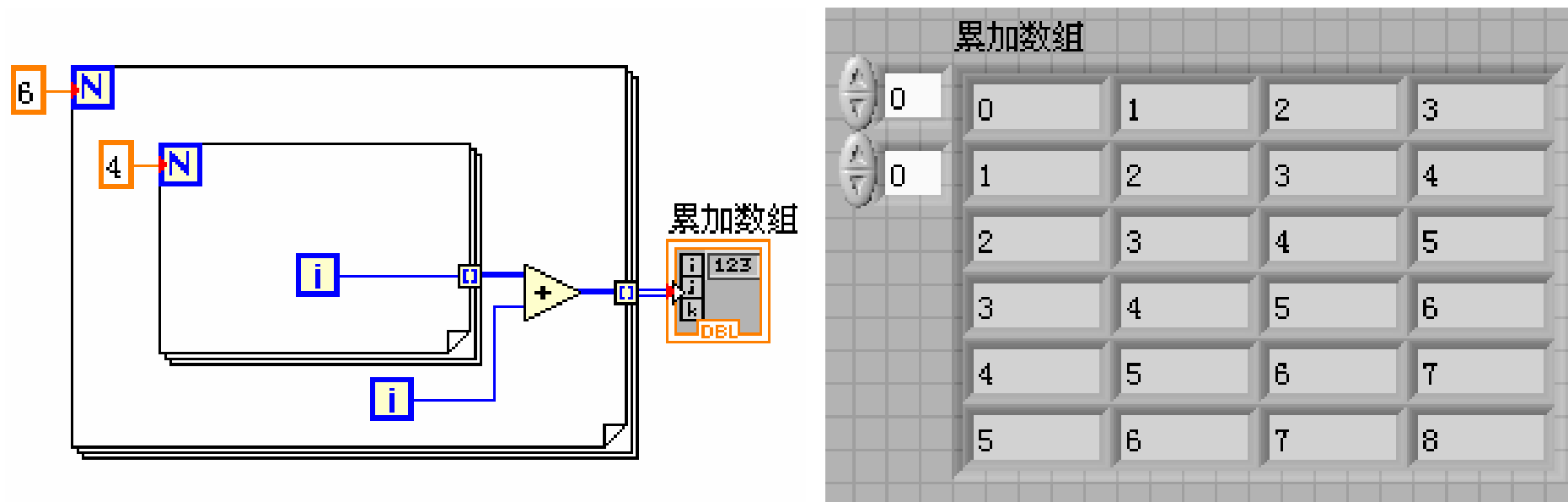


图4-4 For循环自动索引示例

自动索引示例

4.2 While循环

4.2.1 While循环的组成

在如图4-1所示的界面中找到**While**循环后，用鼠标左键单击**While**循环后会发现鼠标箭头变成一个表示**While**循环的小图标，此时用户可在程序框图上用鼠标拖放一个任意大小和位置的**While**循环边框，如图4-8所示。

4.2 While循环



图4-8 While循环界面

最基本的**While**循环由循环框架、条件接线端（输入端）和计数接线端（输出端）组成。

4.2 While循环

与**For**循环的计数接线端一样，**While**的计数接线端也是输出循环已执行次数的数字输出端子。**While**的条件接线端是一个布尔变量，需要输入一个布尔值。

条件接线端用于控制循环是否继续执行时，有两种使用状态：默认状态的条件接线端属性为“真（T）时停止”，此时的图标是一个方框圈住的实心的红色圆点，如图4-8右下角所示，这表示当条件为真时循环停止。



4.2 While循环

当在条件接线端图标上单击鼠标右键选择“真（T）时继续”，则图标变成如图4-9所示，此时表示当条件为真时循环继续。当每一次循环结束时，**条件端口检测通过数据连线输入的布尔值和其使用状态决定是否继续执行循环。**

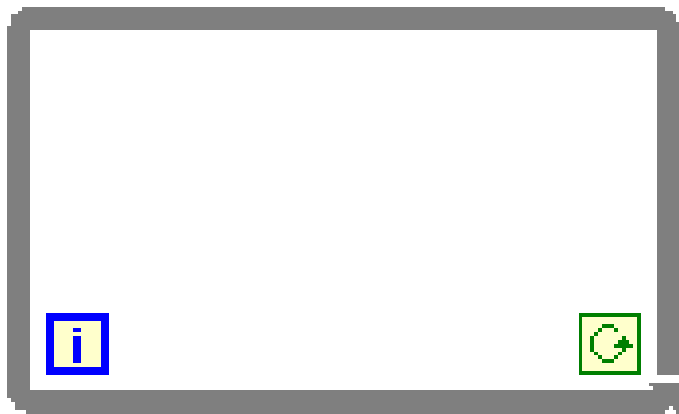


图4-9
条件端子变换后的
While循环界面

4.2 While循环

与**For**循环是在执行前检查是否符合条件不同，**While**循环是在执行后再检查条件端子。因此，**While**循环至少执行一次。

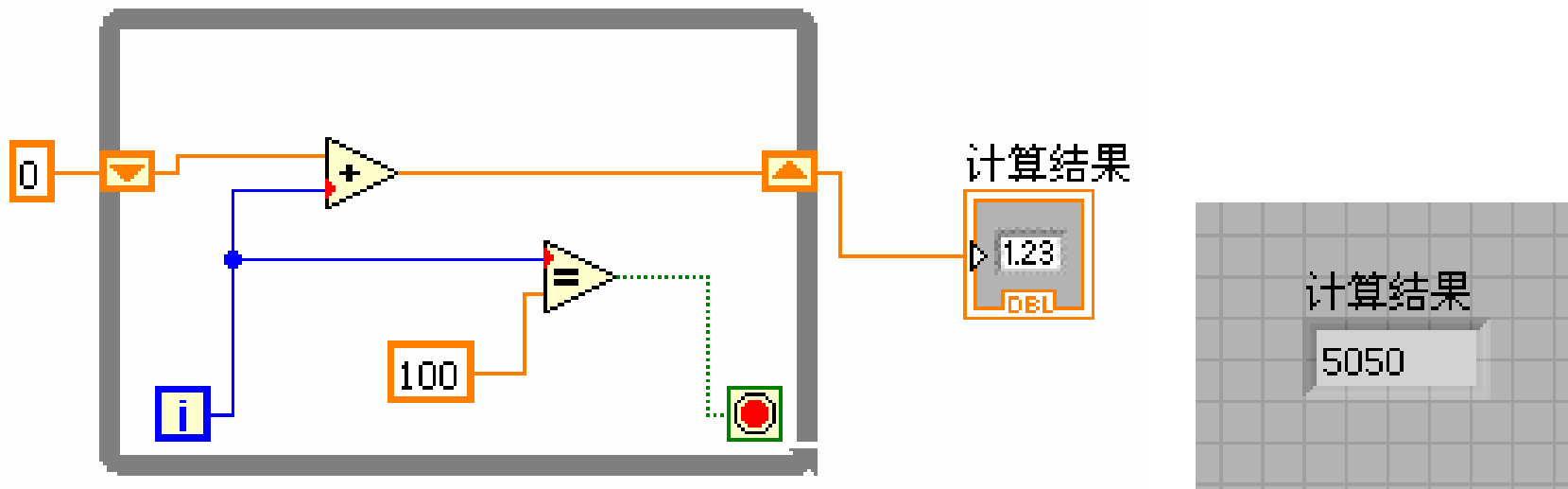


图4-10 简单While循环示例

4.2 While循环

1到100连加

While循环示例1

随机波形图

While循环示例2

温度监测

While循环示例3

持续运行

While-for示例

循环的输入输出

建立数组

反馈节点的使用

4.3 条件（CASE）结构

条件结构同样位于“函数”选板下的“结构”子选板中。与创建循环的方法类似，我们可以从结构选板中选择条件结构，用鼠标在程序框图上任意位置拖放任意大小的条件结构图框。**Case**条件结构由结构框架、条件选择端口、选择器标签及递增/减按钮组成，如图4-11所示。

4.3 条件结构

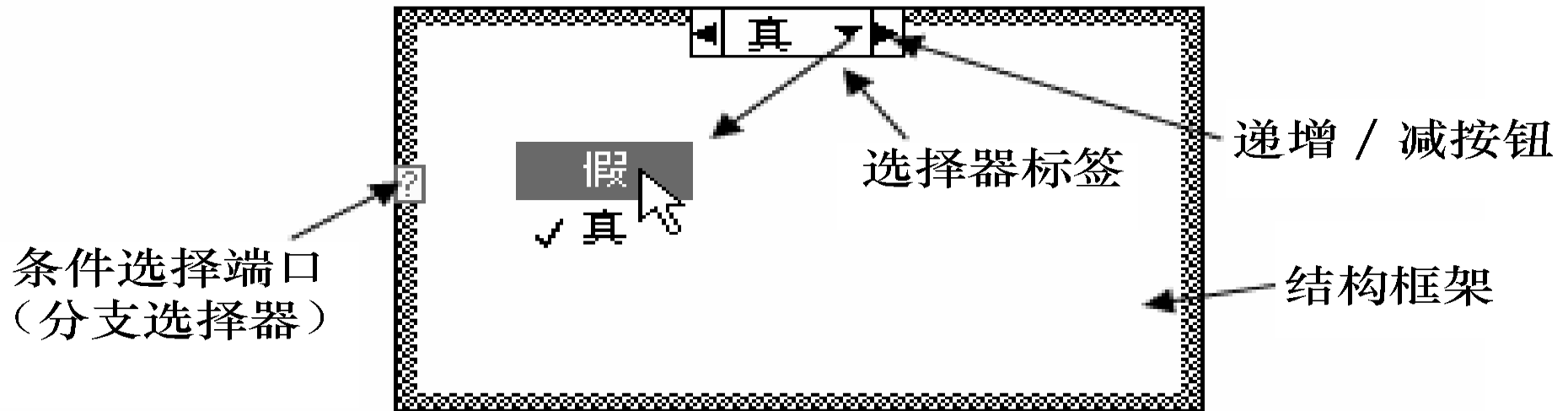


图4-11 条件结构界面

如果要将分支选择器的端口数据类型从数字型改成布尔型，则对应的0和1分支会分别改变成假和真。选择器标签中也可以输入单个值、数值列或数值范围。

4.3.1 添加、删除与排序分支

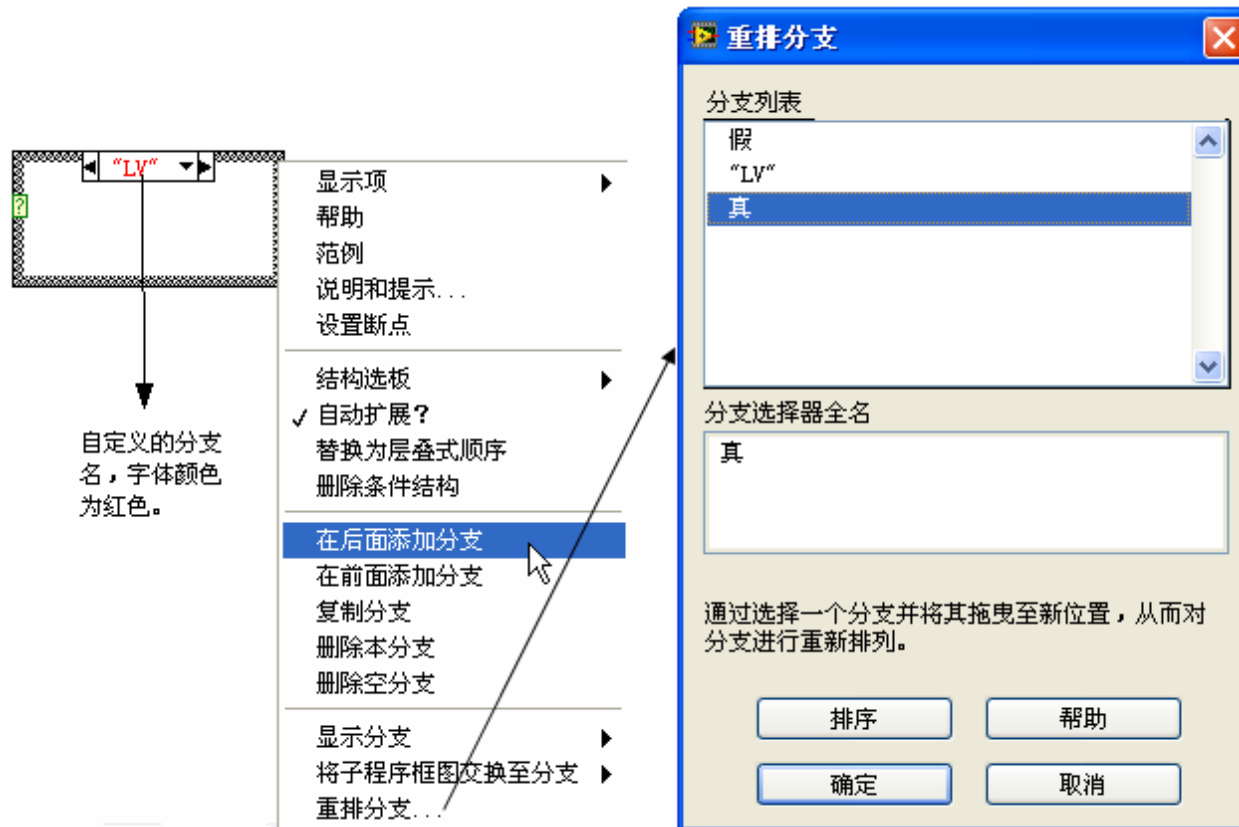


图4-12 分支的添加和排序

在条件结构框架上单击鼠标右键，在弹出的快捷菜单中选择“在后面添加分支”菜单项用户就可以为条件结构添加新的分支，如图4-12所示。

4.3.1 添加、删除与排序分支

添加完新分支后可在快捷菜单中选择“重排分支”菜单项打开重排分支对话框，在对话框的分支列表中用鼠标拖动列表项可以对分支重新排序。通常，排序按钮以第一个选择值为基准对选择器标签值进行排序。删除分支的操作与添加分支相同。

创建新的分支后可以为新分支添加分支名。

建立选择结构

求开平方

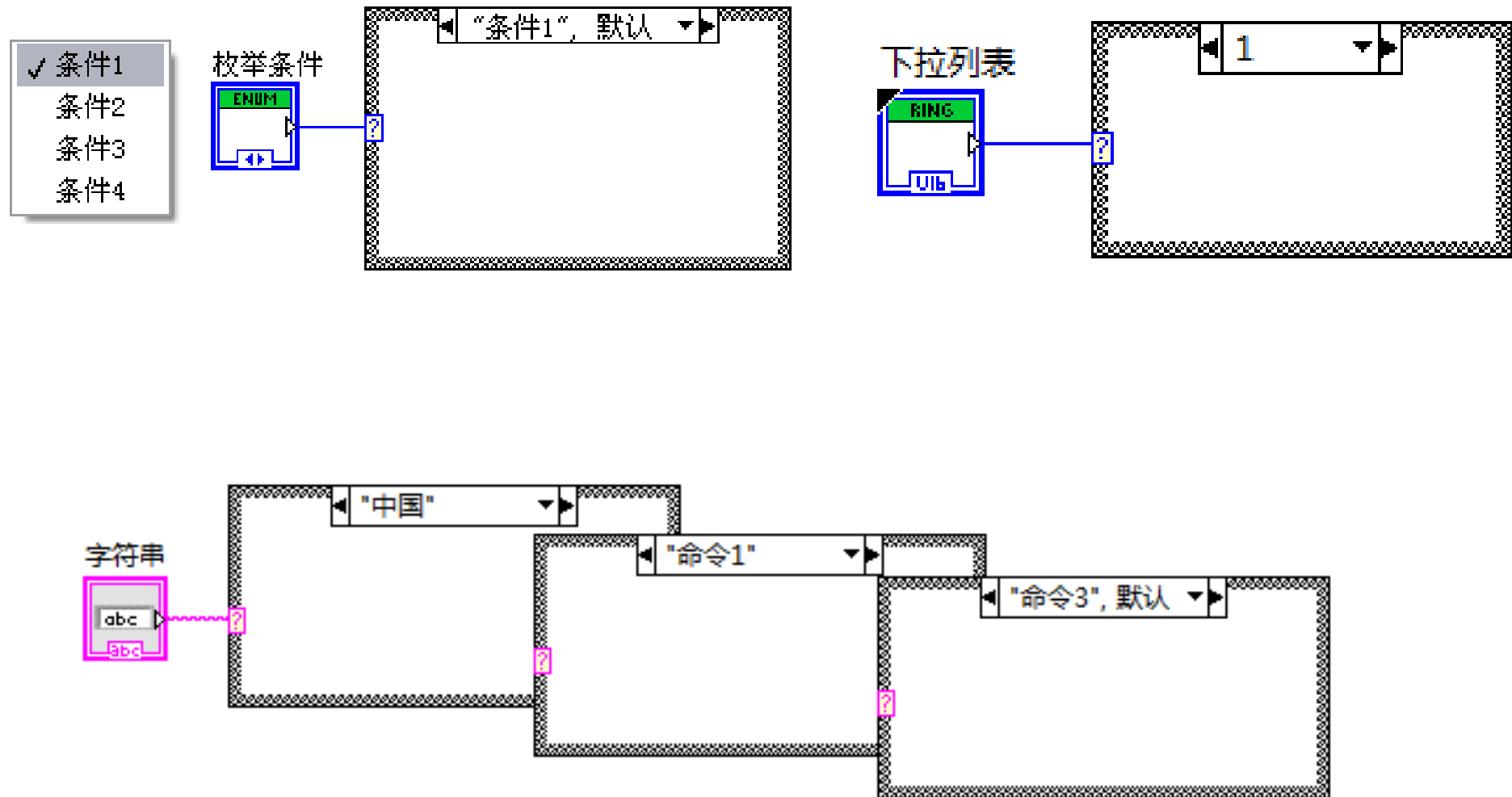
建立选择结构2

枚举列表

建立多选结构

枚举列表

4.3.2 条件结构的条件类型

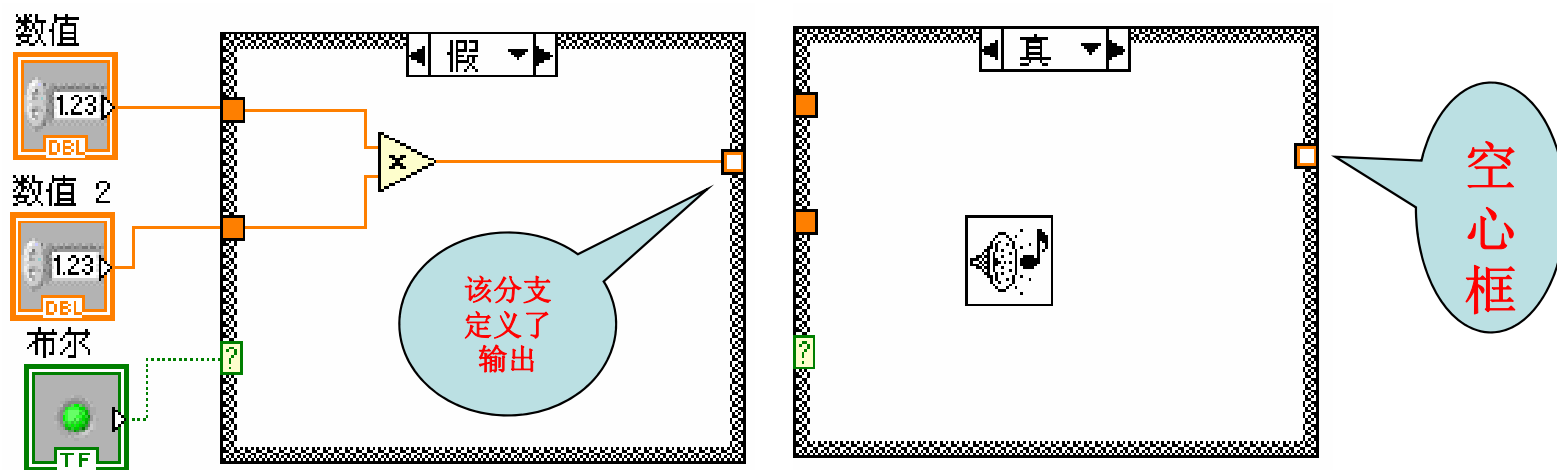


4.3.3 连接数据的输入与输出

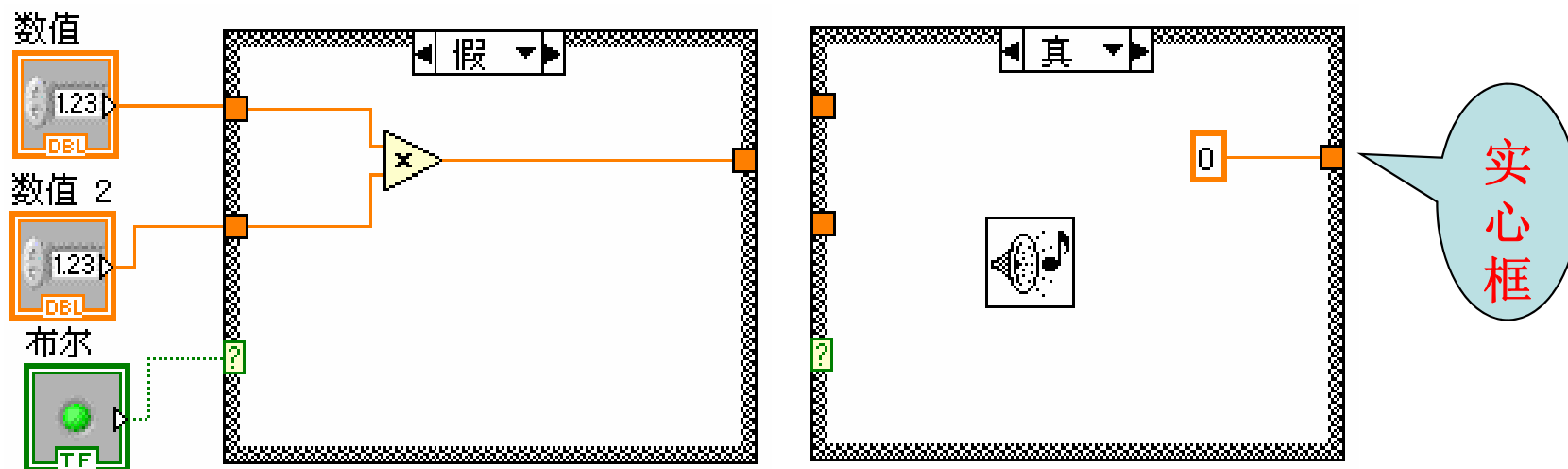
条件结构的所有**输入端子**（包括隧道和选择端子）的数据对所有分支都可以通过连线使用，甚至不用连线也可使用。隧道即是指结构上的数据出入口，表现为以矩形框出现在结构的边框上。分支不一定要使用输入数据或提供输出数据，但是**如果任一分支有输出数据，则其他所有的分支也必须在该数据通道有数据输出，否则将可能导致编程中的代码错误。**

(可以设定其它帧选用缺省值的方式设置)

4.3.3 连接数据的输入与输出



(a) 不正确的连接一边框上的数据通道为中空状态



(b) 正确的连接一边框上的数据通道为实心状态

图4-13 连接数据的输入与输出

4.3.4 Case条件结构示例

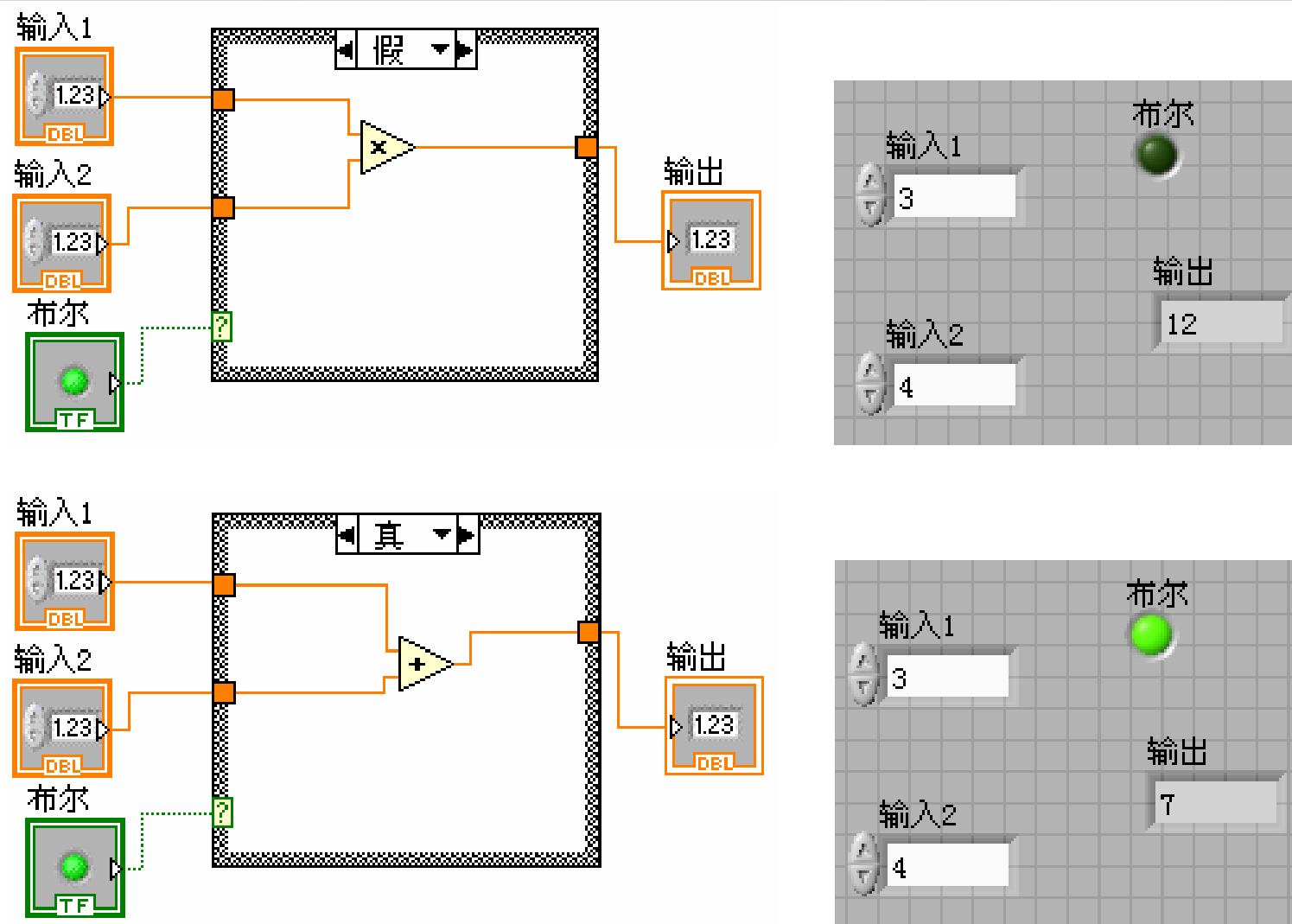


图4-14 执行两个数相乘或相加运算的Case条件结构示例

4.3.4 Case条件结构示例

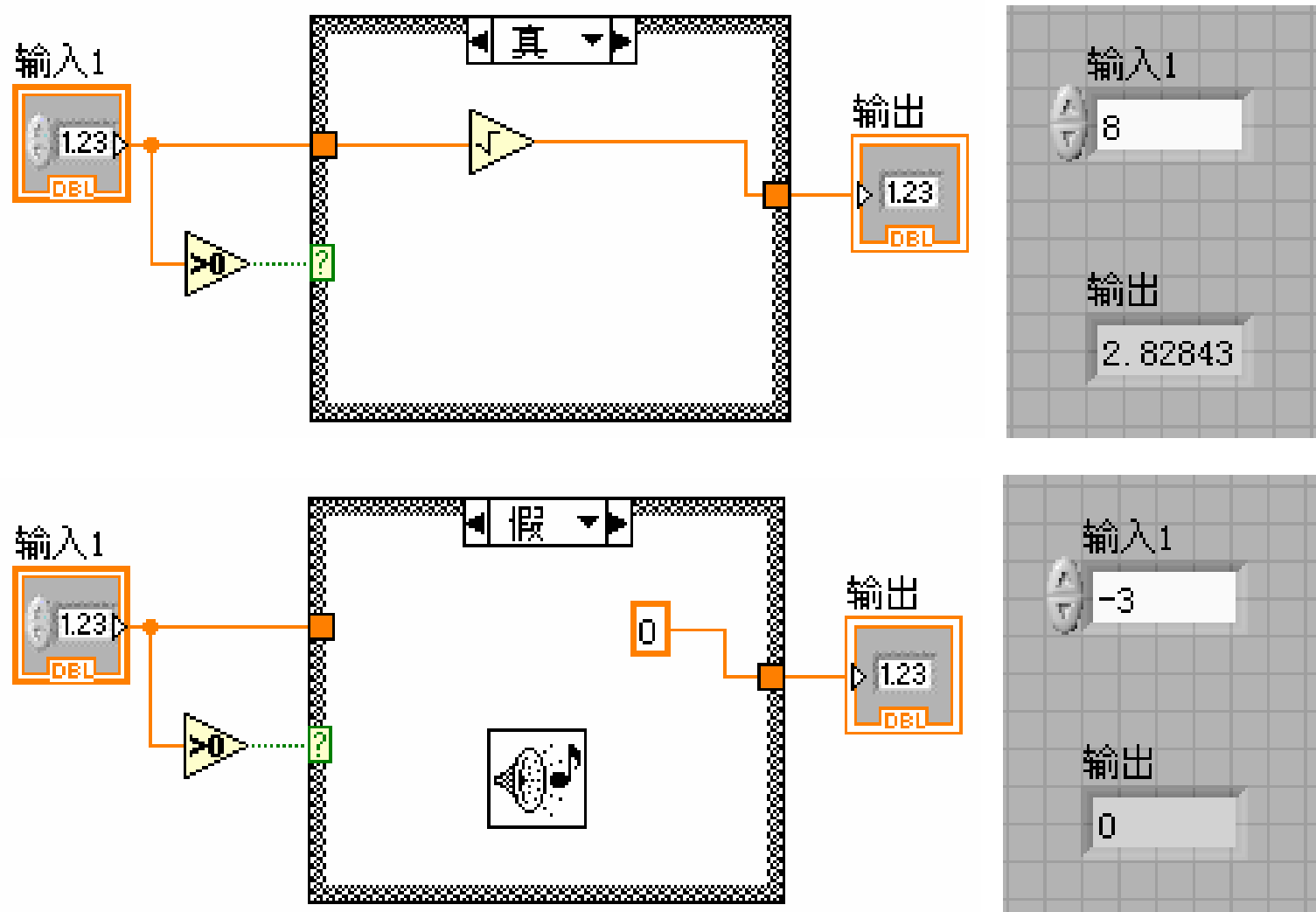


图4-15 执行一个数开根号运算的Case条件结构示例

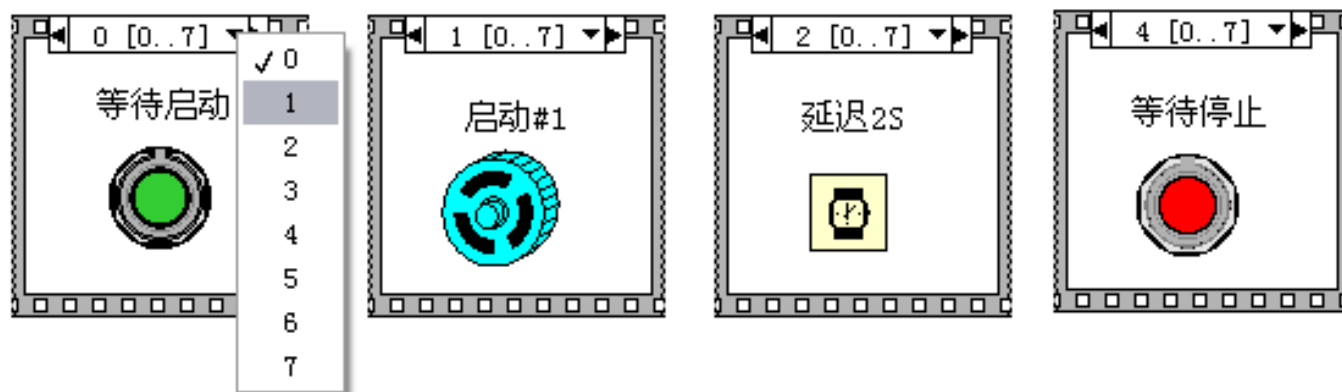
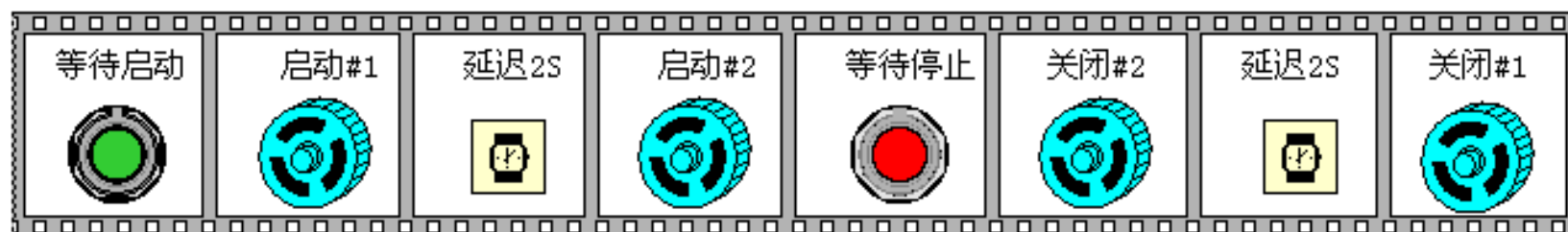
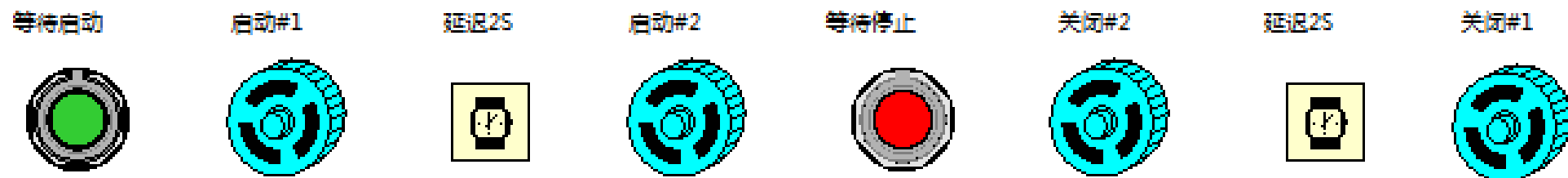
4.3.4 Case条件结构示例

字符串输入

CASE结构示例1
运算符

在用字符串框做输入时，要注意字符串框内的完整内容是“输入”。

4.4 顺序结构



4.4 顺序结构

顺序结构共有两种类型：

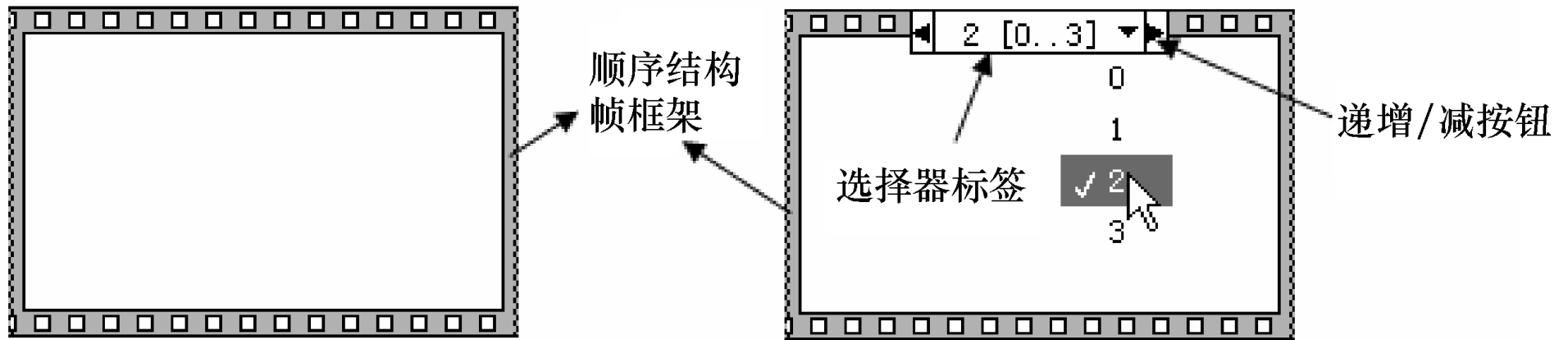
层叠式顺序结构 平铺式顺序结构。

顺序结构顺序地执行子框图，而这些子框图看起来就像一帧帧的电影胶片，因此称之为**帧**。层叠式顺序结构和平铺式顺序结构都位于“函数”选板下的“结构”子选板中。

4.4.1 层叠式顺序结构

与创建其他数据结构的方法类似，用户可以从结构选板中选择顺序结构，然后用鼠标在程序框图上任意位置拖放任意大小的顺序结构图框，此时的顺序结构只有一帧，如图4-16（a）所示。在层叠式顺序结构的边框上单击鼠标右键，从弹出的快捷菜单中选择“在后面添加帧”菜单项就可以添加新的帧。每一个帧都有一个帧编号，编号从0开始。

4.4.1 层叠式顺序结构



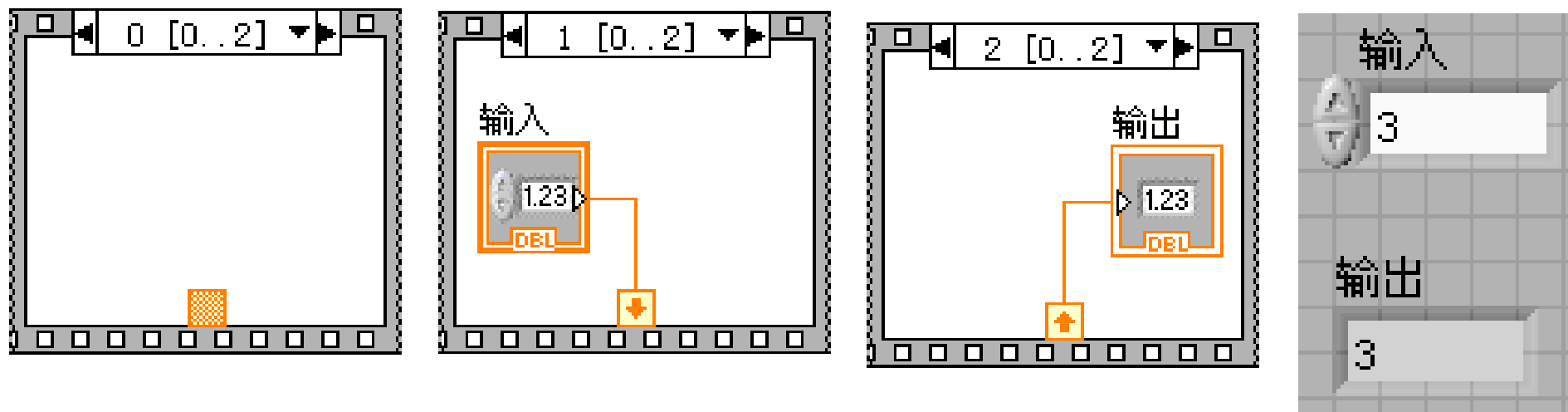
(a) 原始层叠式顺序结构界面

(b) 添加新帧后的层叠式顺序结构

图4-16 层叠式顺序结构界面

层叠式顺序结构中的数据要借助于顺序结构变量来传递。

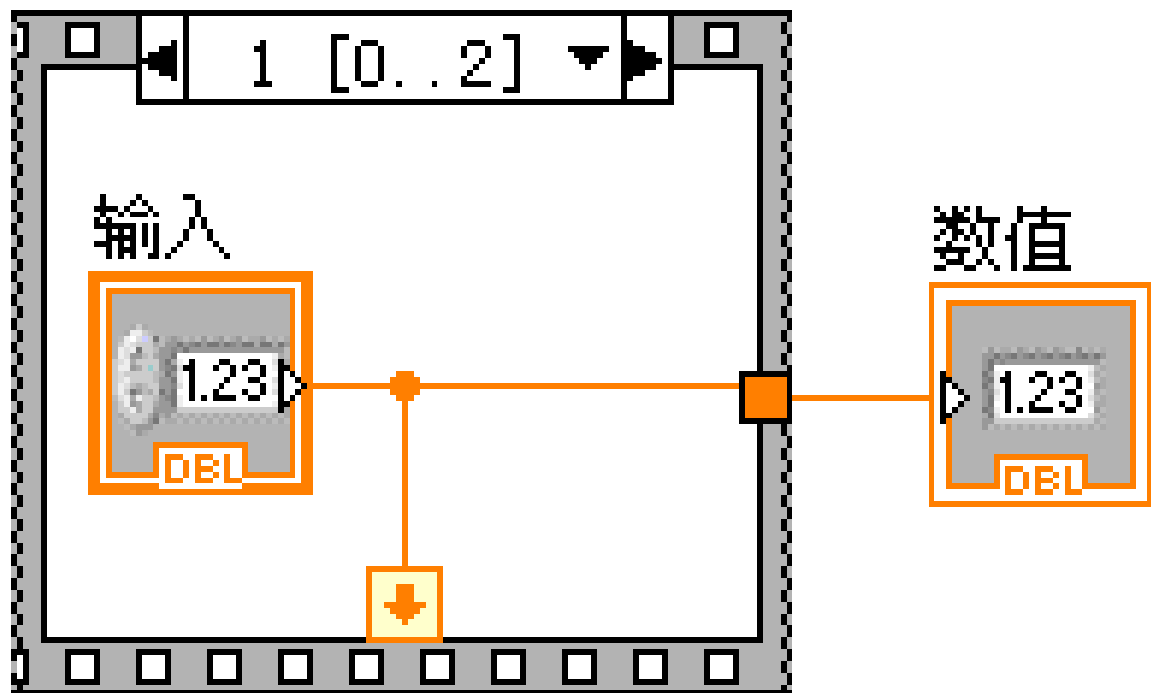
4.4.1 层叠式顺序结构



(a) 未赋值的局部变量 (b) 连接数据 (c) 赋值后的局部变量 (d) 结果

图4-17 顺序局部变量的创建与使用

4.4.1 层叠式顺序结构



层叠式顺序结构

图4-18 顺序结构中的数据通道

4.4.2 平铺式顺序结构

多框架平铺式顺序结构的一个鲜明的特点是它的多个框架不是层叠在一起，而是自左至右平铺，并按**从左至右**的顺序执行。

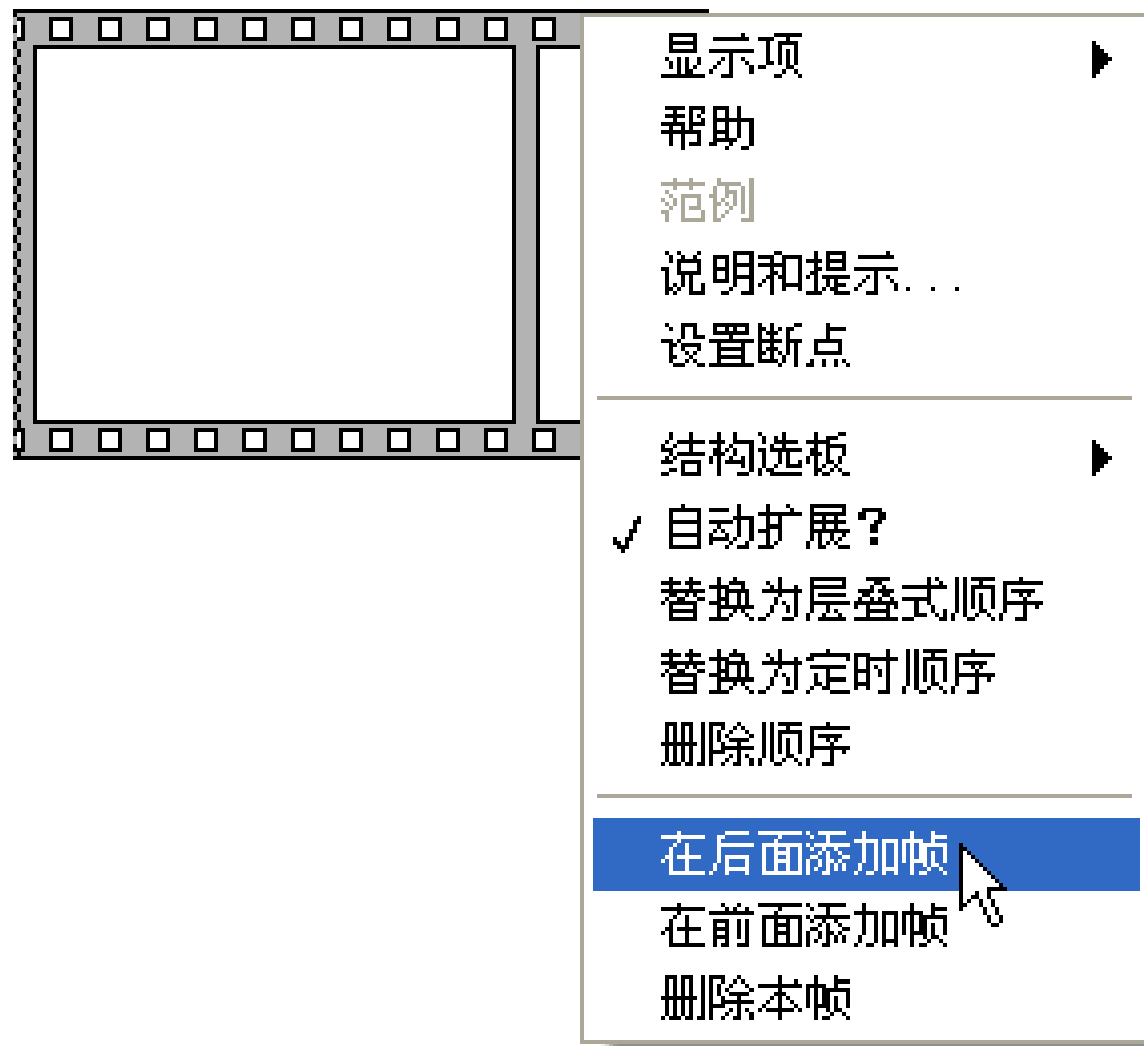


图4-19 平铺式顺序结构界面

4.4.2 平铺式顺序结构

层叠式顺序结构与平铺式顺序结构的功能完全相同。他们的主要区别在于平铺式顺序结构的所有框架在一个平面上，视觉上较为直观，不需要用户在框架之间的切换；层叠式顺序结构使框图中程序更加简洁。层叠式顺序结构和平铺式顺序结构之间是可以互相切换的。在顺序框架的右键选单中按需要选择相应选项即可。

4.4.2 平铺式顺序结构

相比于层叠式顺序结构，平铺式顺序结构各帧之间同样可以传输数据，而且平铺式顺序结构传递数据的方式与层叠式顺序结构相比较而言更加简单和直观，**只需直接在两帧间连线就可以自动创建一个隧道传输数据，如图4-20所示。**

4.4.2 平铺式顺序结构

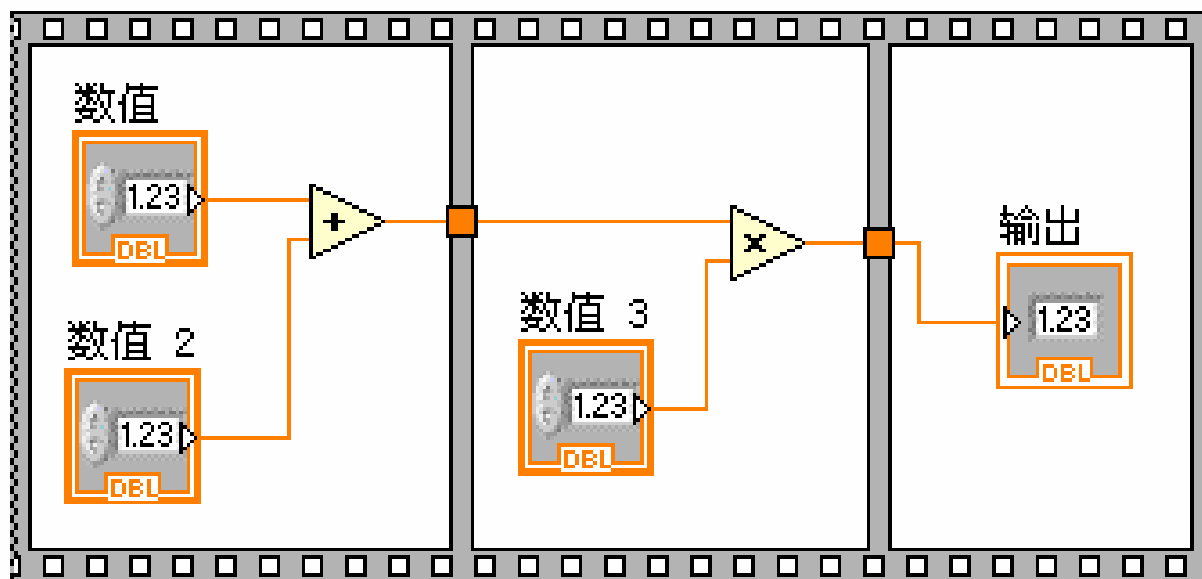


图4-20 平铺式顺序结构的数据传输

平铺式顺序结构

4.5 事件结构

所谓**事件**，是指对程序活动发生的**异步通知**。事件可以来自于用户界面、外部**I/O**或其他方式。**用户界面事件**包括鼠标点击、键盘按键等动作，**外部I/O事件**则指诸如数据采集完毕或发生错误时硬件触发器或定时器发出信号。**LABVIEW**中的事件结构也是一种特殊的选择结构。

4.5 事件结构

LabVIEW支持用户界面事件和通过编程生成的事件，但不支持外部I/O事件。

LabVIEW中的事件结构也是一种能改变数据流执行方式的结构，使用事件结构可以实现用户在前面板的操作（事件）与程序执行的互动。

4.5 事件结构

Labview支持下列5种事件

1. **应用程序事件**（<Application>），这类事件主要反映整个应用程序状态的变化，例如：程序是否关闭，是否超时等。
2. **VI事件**（<This VI>），这类事件反映当前VI状态的改变。例如：当前VI是否被关闭，是否选择了菜单中的某一项等等。
3. **动态事件**（Dynamic），用于处理用户自己定义的或在程序中临时生成的事件。
4. **区域事件**（Pane）和分割线事件（Splitter）是LabVIEW 8中新添加的特性。LabVIEW 8中，用户可以把一个VI的前面板分割成几份，这两类事件用来处理用户对某个区域或区域分割线的状态改变。
5. **控件事件**（Control）是最常用的一种事件，用于处理某个控件状态的改变。例如，控件值的改变，或者鼠标键盘的操作。

在右键关联菜单中，只要选定了某一个事件产生源，其相应的所有事件均排列在右侧**events**框中。

4.5.1 事件结构的创建与简单示例

一个标准的事件结构由**框架、超时端子、事件数据节点、递增/减按钮、选择器标签**组成，如图**4-21**所示。和条件结构相似，事件结构也可以由多层框架组成，但与条件结构不同的是，**事件结构虽然每次只能运行一个框图**，但可以**同时**响应几个事件。

4.5.1 事件结构的创建与简单示例

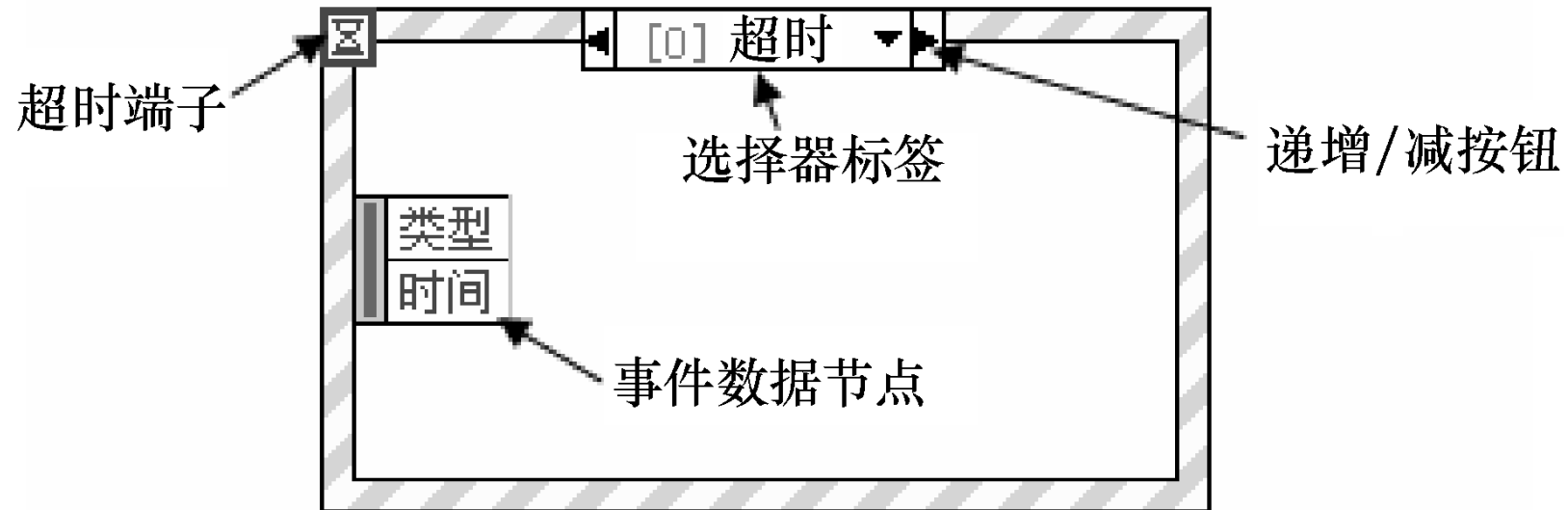
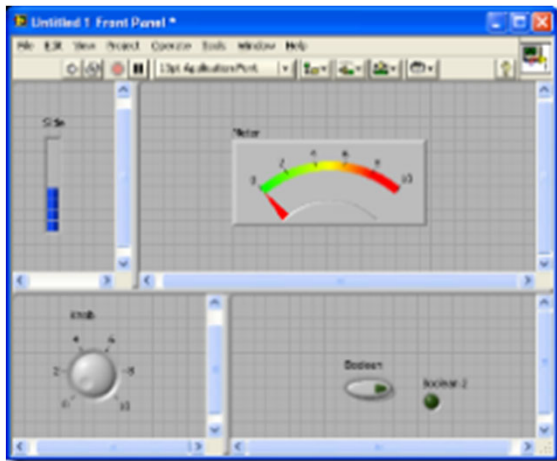


图4-21 事件结构的基本构成



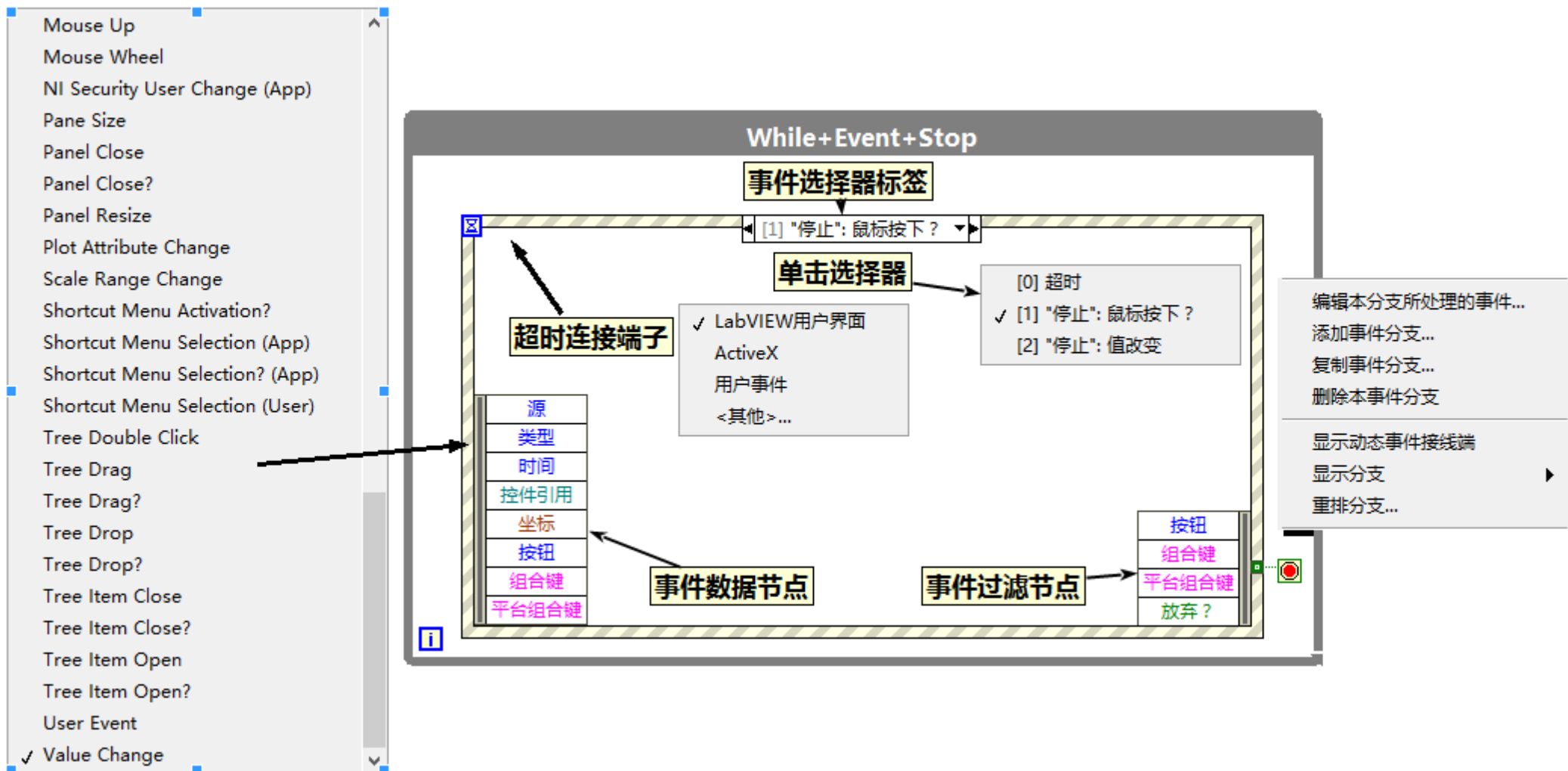
4.5.1 事件结构的创建与简单示例

超时端子用来设定超时时间，其接入数据是以毫秒为单位的整数值。**-1为关闭超时事件。**

事件数据节点由若干个事件数据端子构成，数据端子的增减可以通过拖拉事件数据节点来进行，也可以通过单击鼠标右键从弹出的快捷菜单中选择“添加/删除元素”选项进行。

事件结构同样支持隧道。

4.5.1 事件结构的创建与简单示例



4.5.1 事件结构的创建与简单示例

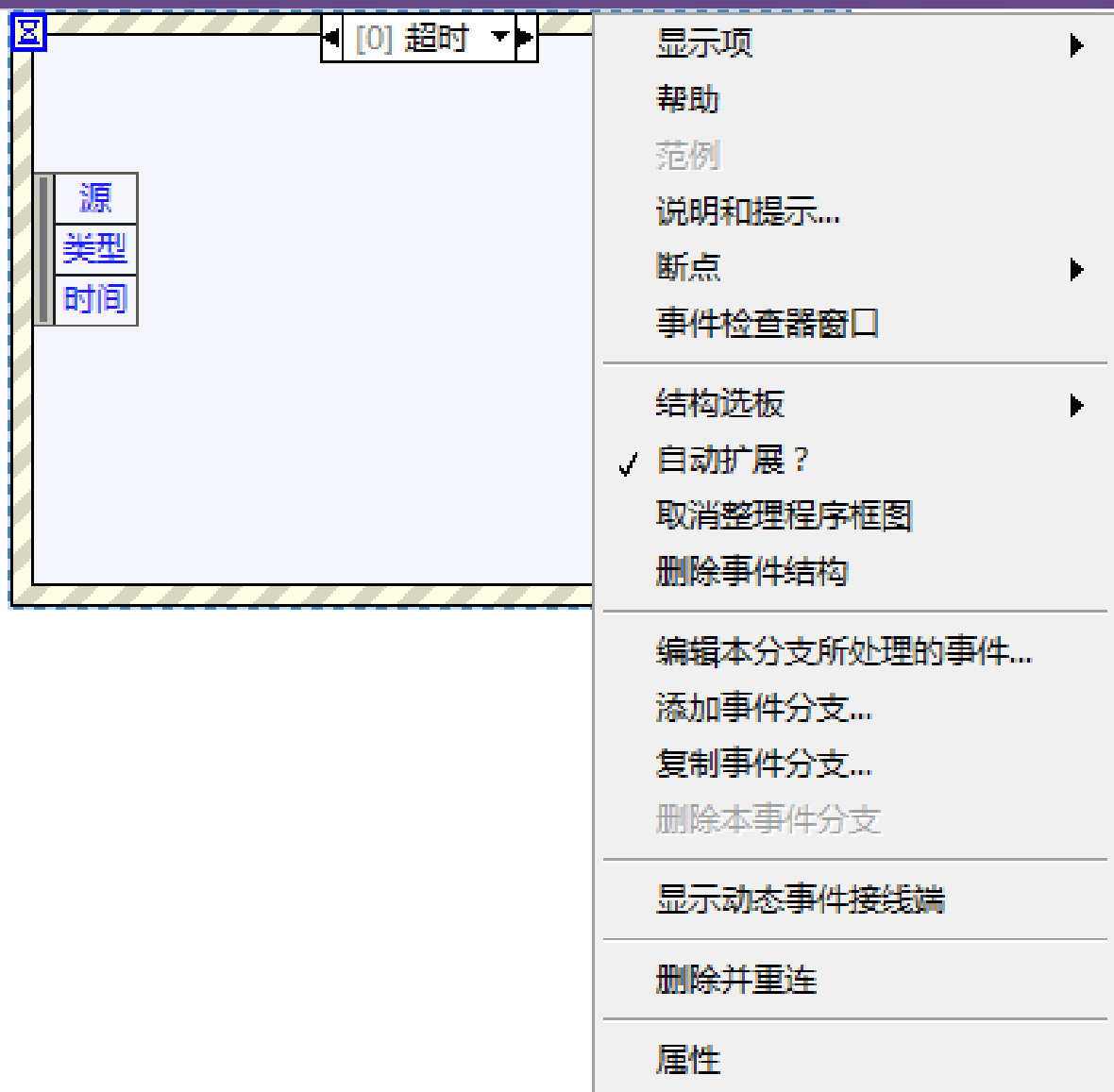


图4-22 添加事件分支

4.5.1 事件结构的创建与简单示例

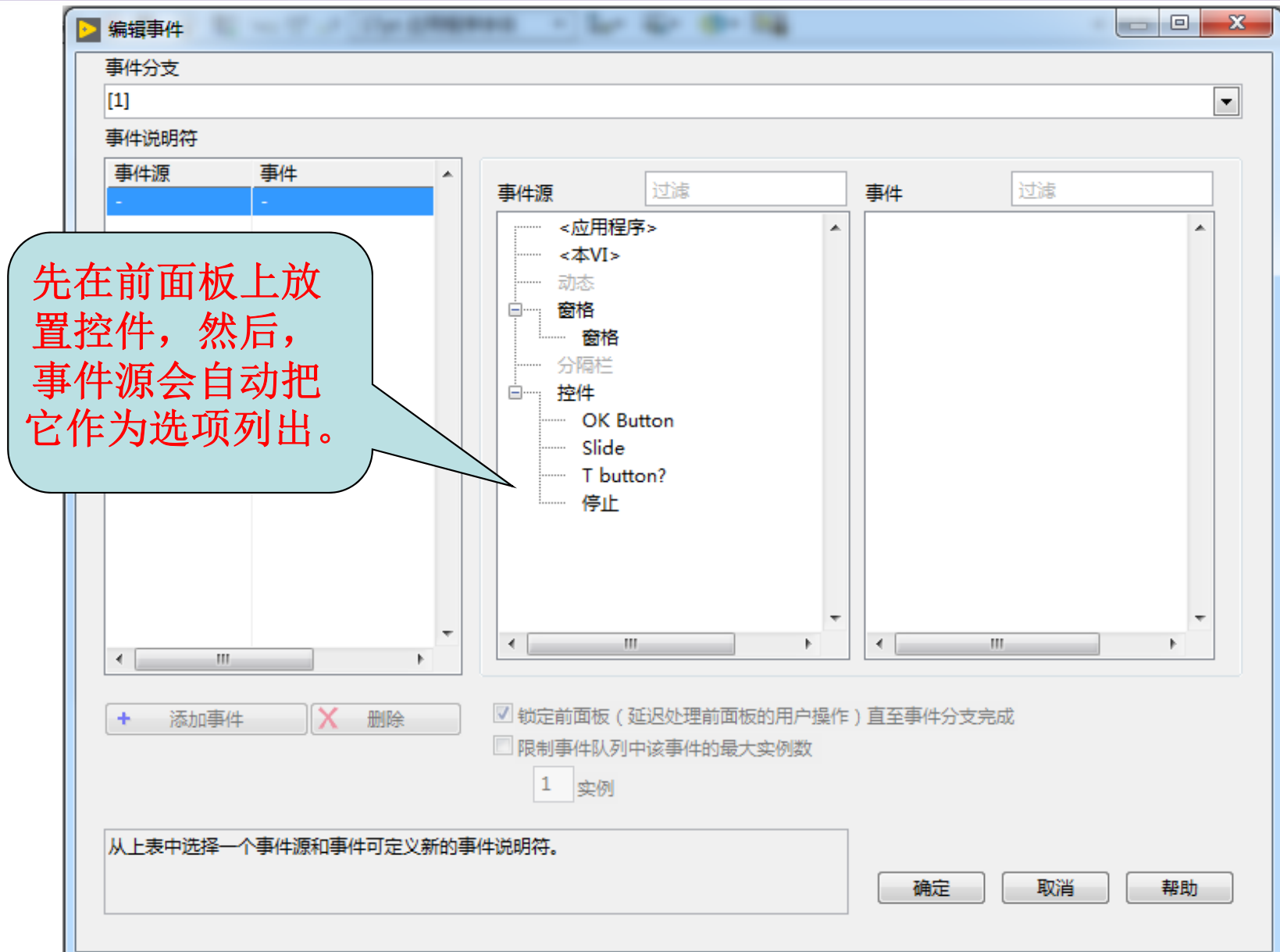


图4-23 编辑事件对话框

4.5.1 事件结构的创建与简单示例

在编辑事件对话框中首先要选择一个事件处理分支作为对象，然后在事件源中选择合适的**事件源**。

编辑事件对话框中的事件列表中放置的是选中事件源对应的所有可能的事件名称，使用鼠标左键选择希望的动作选项就可以为**事件源**创建事件。

4.5.1 事件结构的创建与简单示例

在编辑事件对话框中还有一个添加事件按钮，它可以在一个事件框中添加多个事件，当事件框中某个事件发生时，事件框中的程序就会运行。事件的删除则通过单击添加事件按钮下方的删除事件按钮来操作。

应该避免在同一VI上设立多个事件结构容器，用一个事件结构容器处理所有的事件。

如果没有把事件结构放在循环结构中，则事件结构框架执行一次，就被认为是执行完毕。这点和选择结构一样。如果放在循环里面，则事件必须执行一次才能够算执行一次循环体。所以，循环中的事件，往往要有超时事件，不然，可能导致循环的无休止等待。

4.5.1 事件结构的创建与简单示例

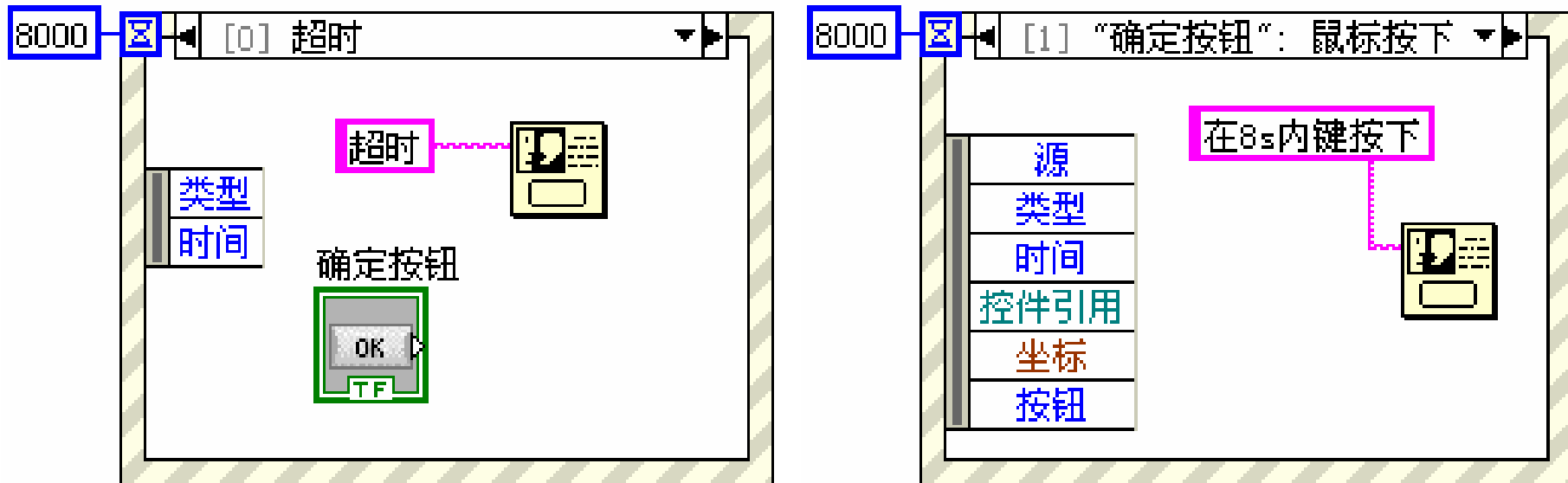


图4-24 事件结构示例

事件结构简单示例1

8s内按键
While 停止控制

事件结构简单示例2

分割栏

停止按钮处理方法

事件结构简单示例3

窗格、滑动杆

4.5.2 过滤事件和通知事件

按照事件的发出时间来区分，**LabVIEW**的事件可分为**通知型事件（Notify Event）**和**过滤型事件（Filter Event）**。

通知型事件是在**LabVIEW**处理完用户操作之后发出的，比如用户利用键盘操作改变了一个字符串，**LabVIEW**在改变了该控件的值之后，发出一个**值改变（Value Changed）**通知型事件，告诉事件结构，控件的值被改变了。如果事件结构内有处理该事件的框架，则程序转去执行该框架。

过滤型事件是在**LabVIEW**处理用户操作之前发出的，并等待相对应的事件框架执行完成之后，**LabVIEW**再处理该用户操作。**这类事件的名称之后都有一个问号**。例如**键盘按下？事件（Key Down? Event）**，当用户处理该事件时，控件的值还没有被改变，因此，用户可以在该事件对应的事件框架内决定是否让**LabVIEW**先处理该事件，或改变键盘按下的值之后再让**LabVIEW**继续处理该事件。

过滤型事件比相应的通知型事件要先发出。

4.5.2 过滤事件和通知事件

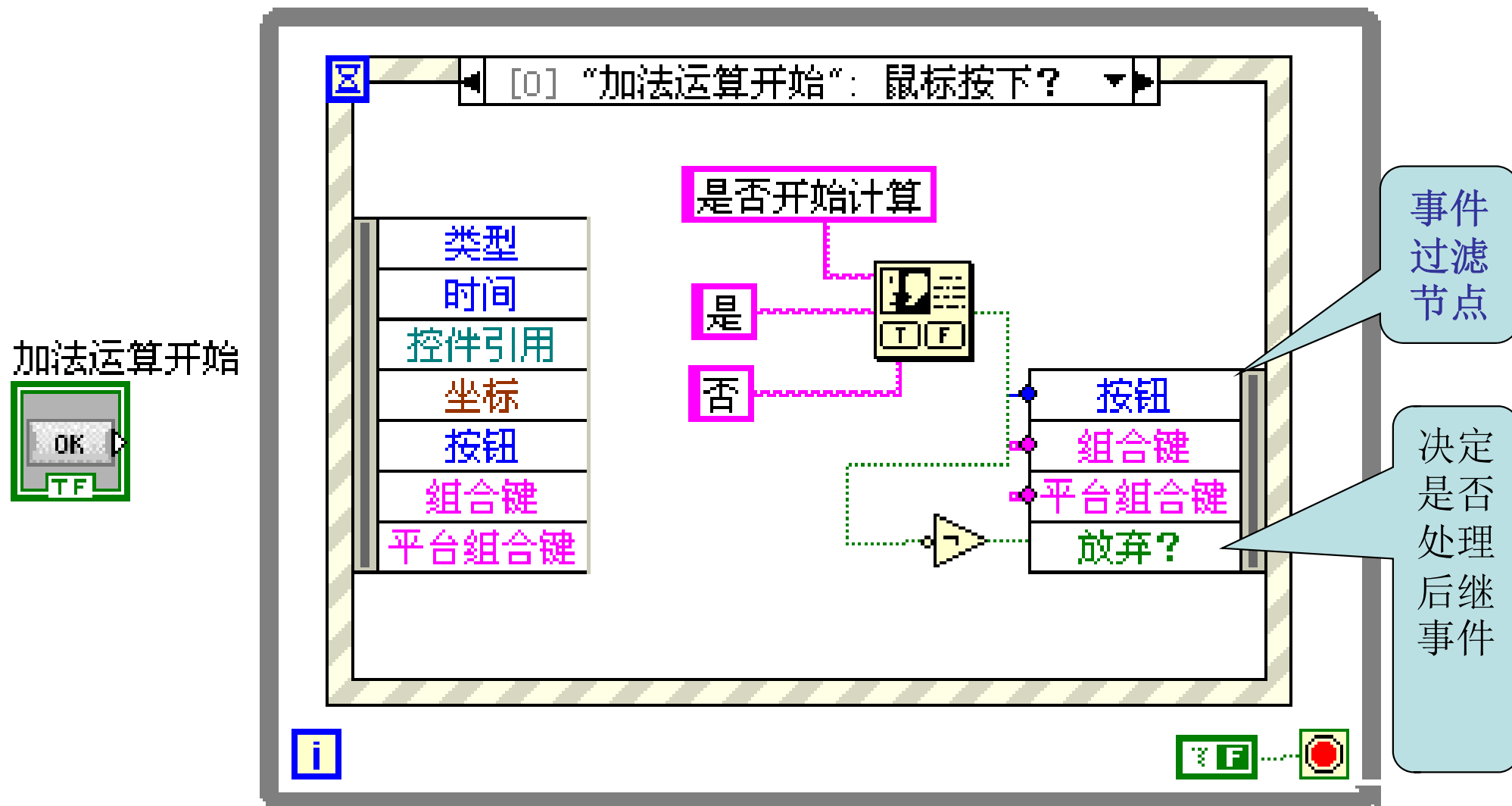


图4-25 事件结构框架0（过滤事件）

4.5.2 过滤事件和通知事件

加法运算开始

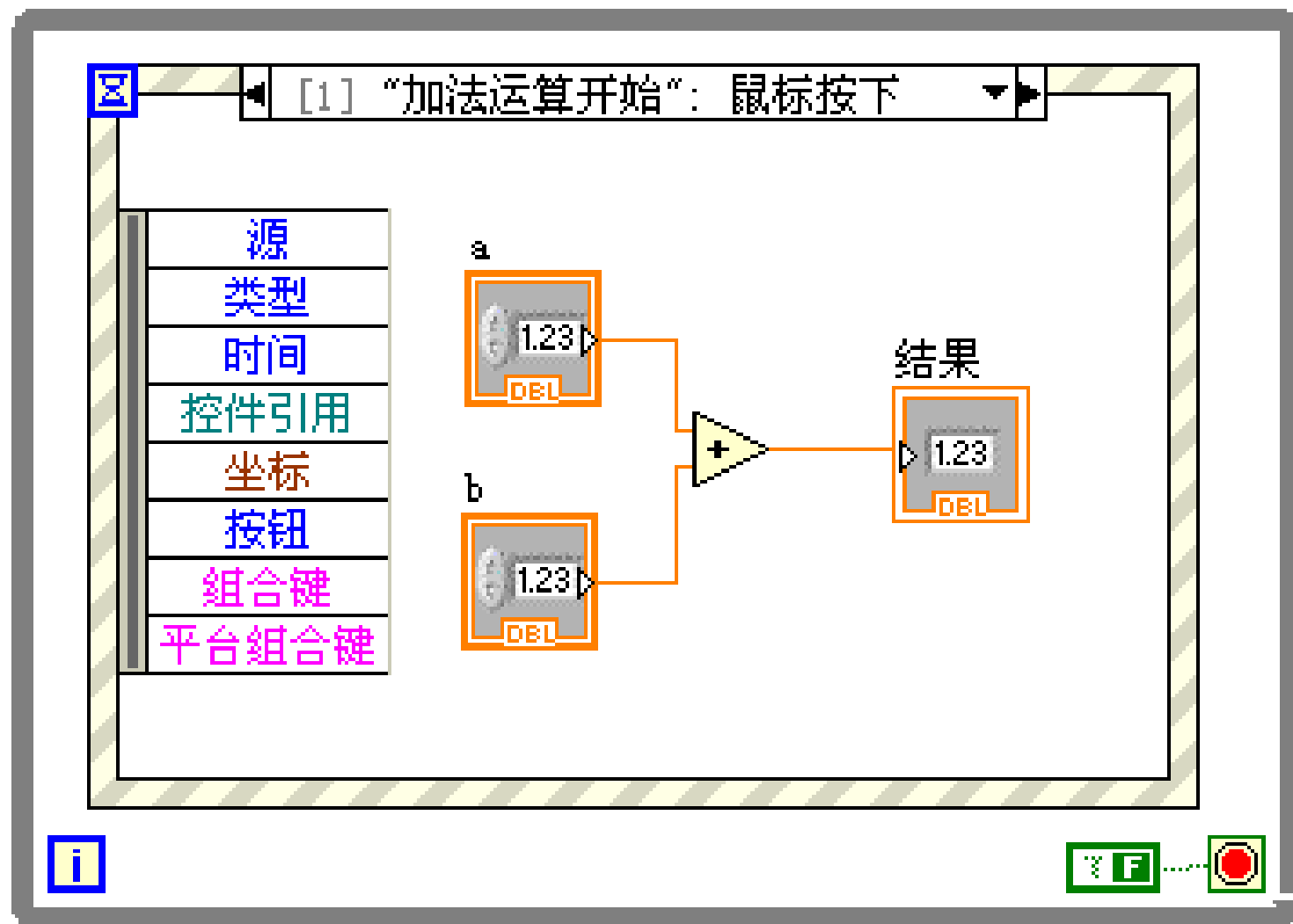
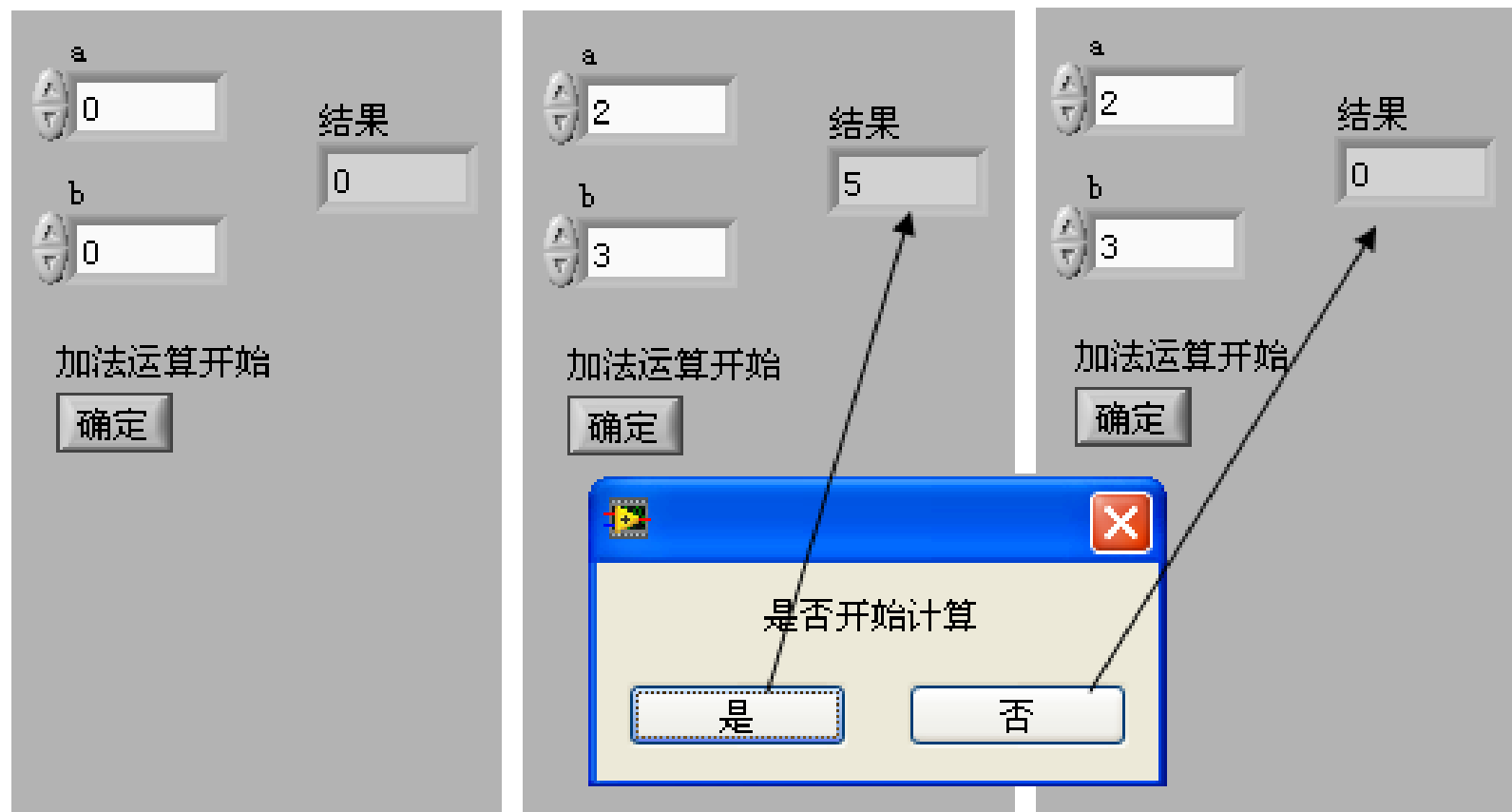


图4-26 事件结构框架1（通知事件）

4.5.2 过滤事件和通知事件



(a) “确定”按钮动作前 (b) 选择“是”的计算结果 (c) 选择“否”的计算结果

图4-27 静态事件结构运行结果

事件结构简单示例3

加法计算

事件结构简单示例4

两个控件对应同一个事件

4.5.2 过滤事件和通知事件

- 由于过滤事件一定比同类的通知事件要先发生，故对于同一个控件，可以在事件结构中，并存过滤事件、通知事件，且二者常常结合使用。

鼠标事件发生顺序

事件的过滤与转发

过滤事件的作用

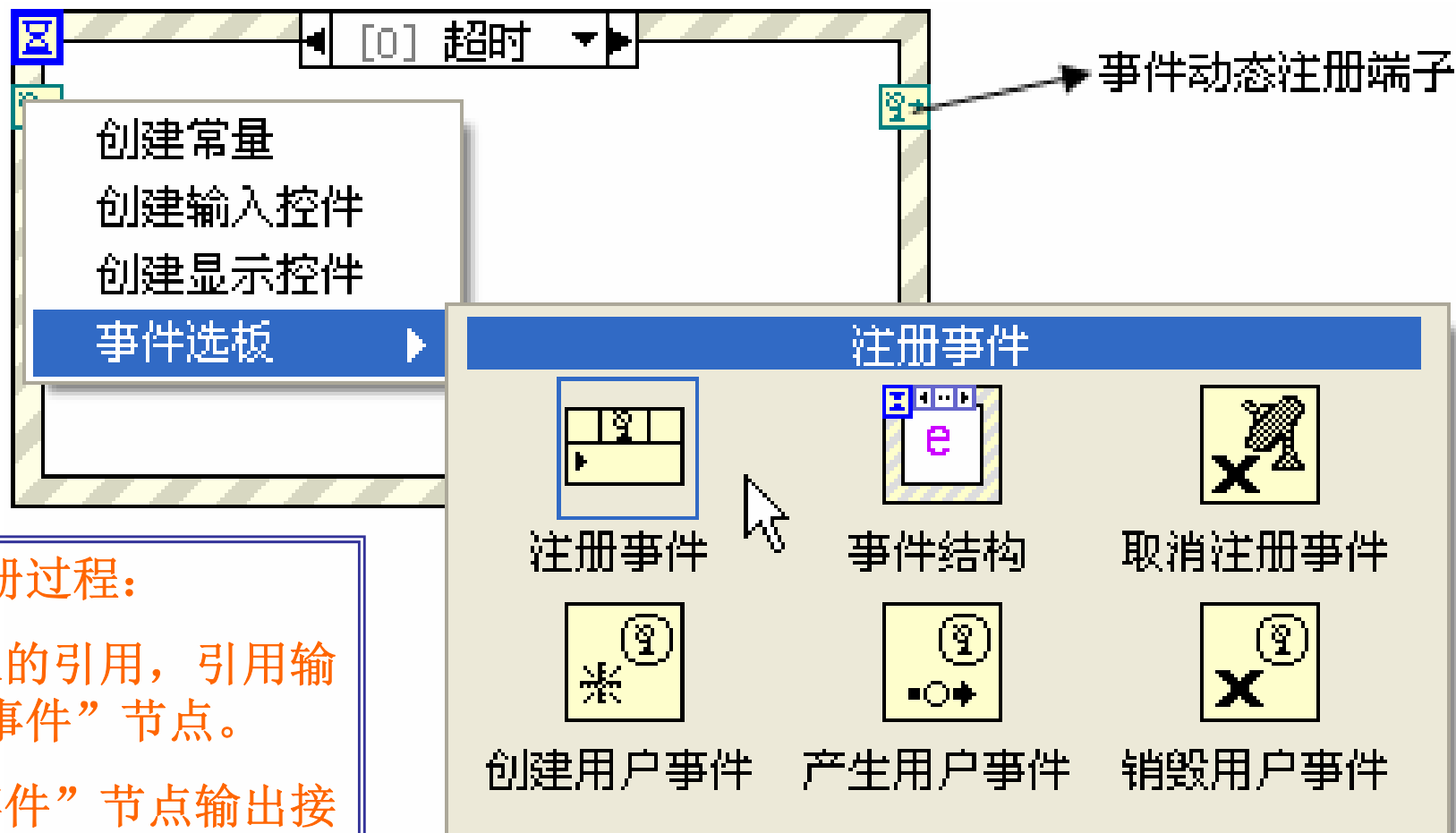
4.5.3 静态事件和动态事件

事件结构分为静态和动态两种。如果只需对前面板对象进行操作判断，使用静态事件结构就完全可以实现；但如果需要实时改变注册内容或将程序中的数据作为事件的发生条件，将事件注册的过程限制在一段代码内等特殊情况时就要用到动态事件结构：动态注册事件源、动态端子连接、配置动态响应事件、在结构外部注销事件。

4.5.3 静态事件和动态事件

动态事件结构的创建就需要使用注册事件节点注册事件（指定事件结构中事件的事件源和事件类型的过程称为注册事件），再将结果输出到事件结构动态事件注册端子上。若要创建一个事件动态注册端子，可以在事件结构框图上单击鼠标右键，在弹出的快捷菜单中选择“显示动态事件接线端”选项即可。

4.5.3 静态事件和动态事件



动态事件注册过程:

- 1、获得对象的引用，引用输入到“注册事件”节点。
- 2、“注册事件”节点输出接到动态端子上。
- 3、“配置”动态事件。
- 4、在结构外，加入“取消注册事件”节点。

图4-28 事件函数选板

4.6 局部变量与全局变量

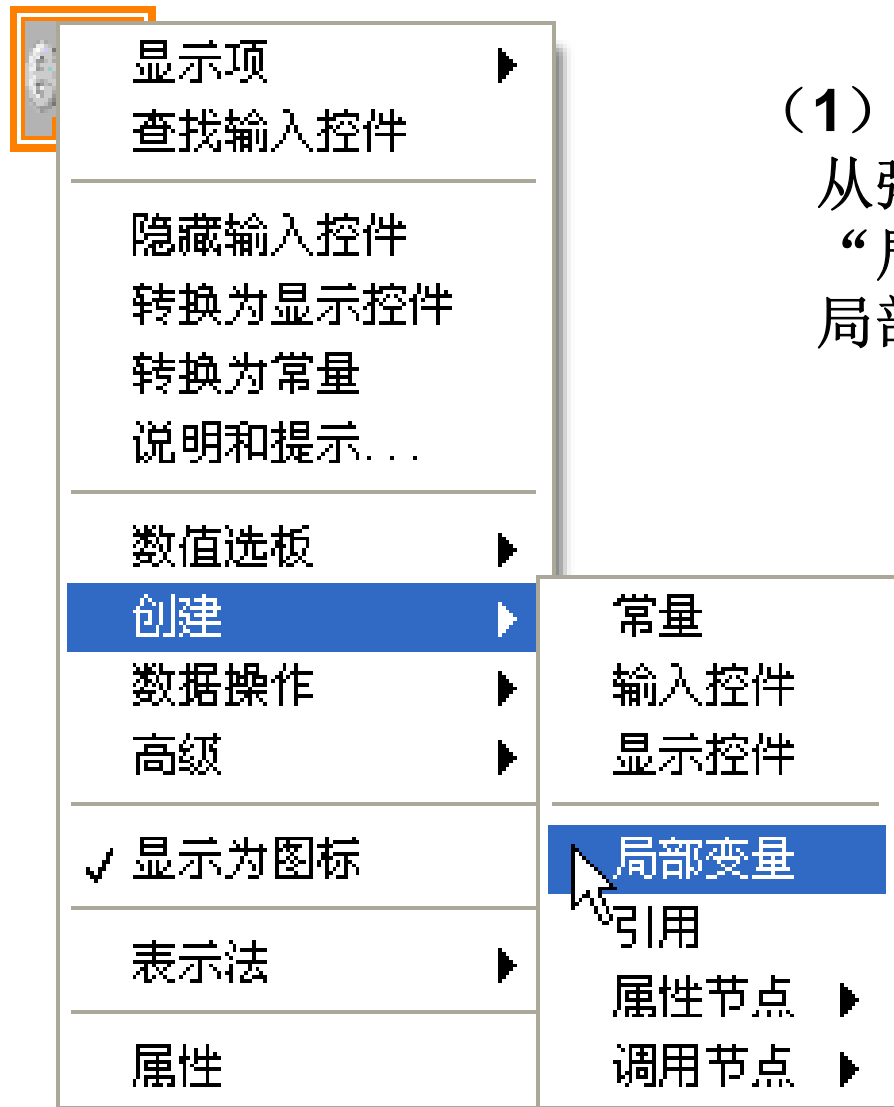
4.6.1 局部变量的创建及使用

使用局部变量可对前面板上的输入控件或显示控件进行数据读写。写入一个局部变量相当于将数据传递给其他接线端。同时，通过局部变量，前面板对象既可作为输入访问也可作为输出访问。局部变量可从一个VI的不同位置访问前面板对象，并将无法用连线连接的数据在程序框图上的节点之间传递。

顺序结构中的局部变量、循环结构中的移位寄存器都属于局部变量。

4.6 局部变量与全局变量

数值



局部变量的创建方式有两种。

(1) 鼠标右键单击一个前面板中已有的对象，从弹出的快捷菜单中选择“创建”选项下的“局部变量”选项，便可创建为该对象一个局部变量，如图4-29所示。

(2) 从如图4-1所示的“结构”子选板中选择“局部变量”并将其拖放到程序框图上。（数据通信 选板）

图4-29 直接单击前面板中对象创建局部变量

4.6 局部变量与全局变量

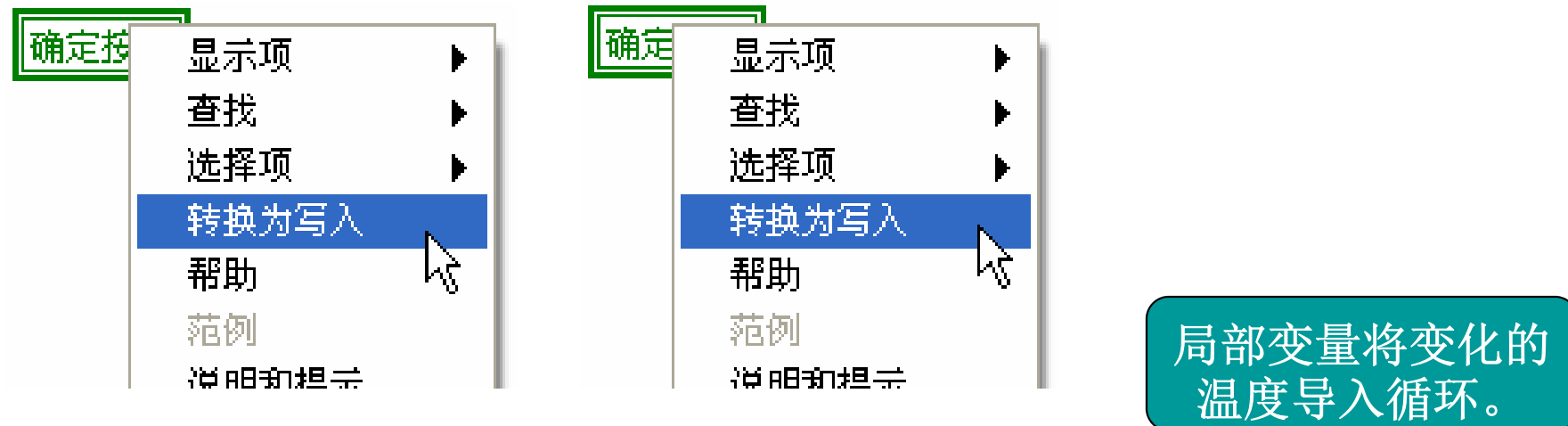


图4-31 局部变量“读/写”属性的切换

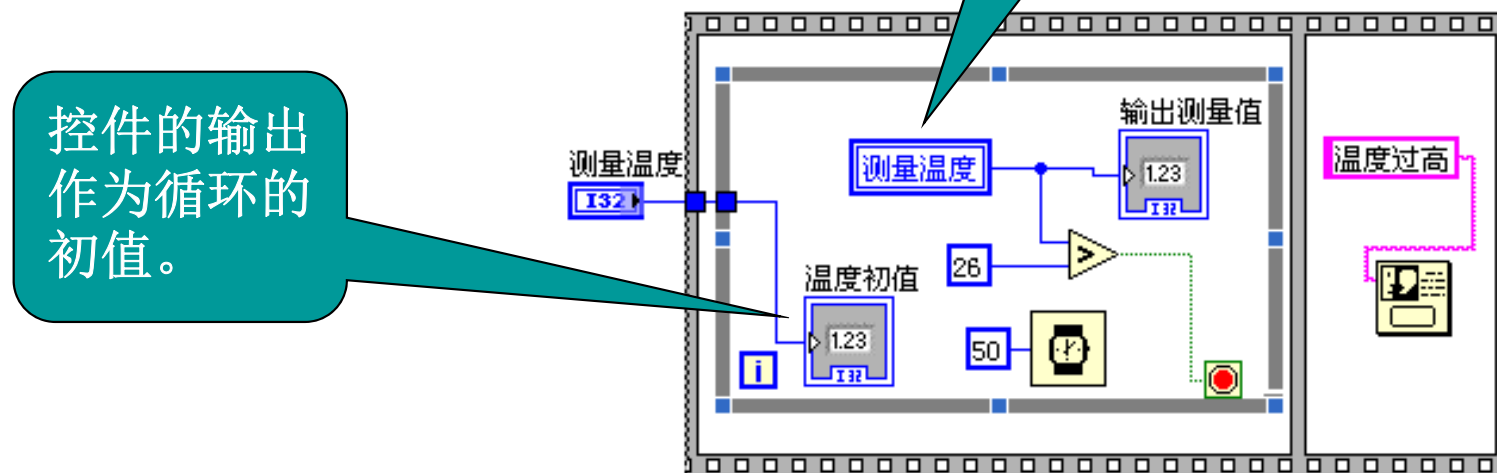


图4-32 含有局部变量的程序框图

4.6 局部变量与全局变量

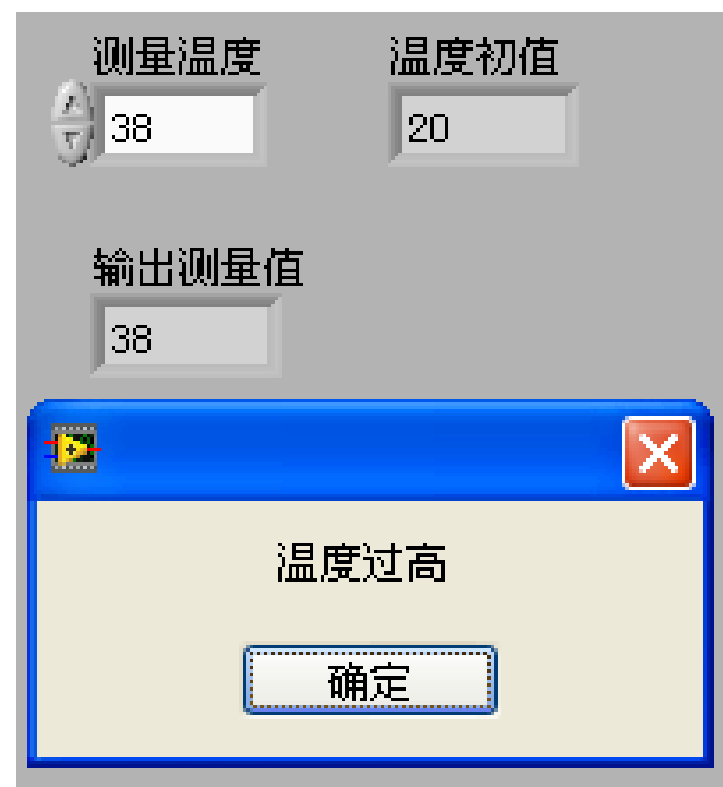
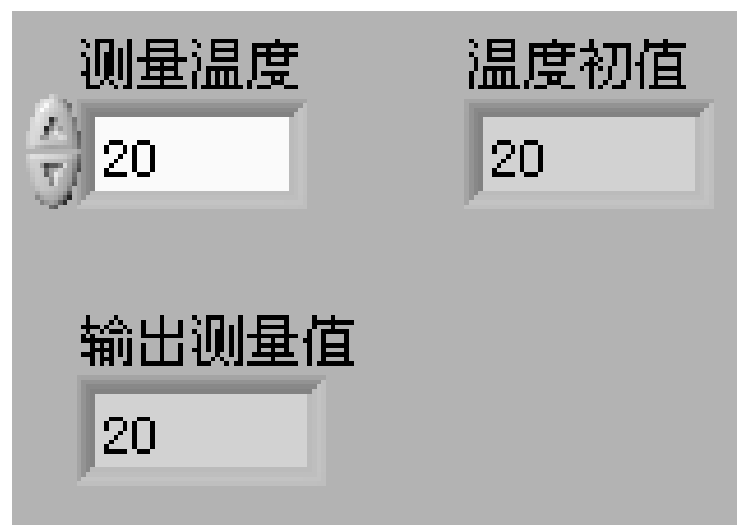


图4-33 程序运行结果

4.6 局部变量与全局变量

LabVIEW通过**自带标签**将局部变量和前面板对象相关联，因此必须用描述性的自带标签对前面板控件和显示控件进行标注。

局部变量有**读**和**写**两种属性。

顺序局部变量

局部变量示例

循环中局部变量

局部变量示例2

循环中局部变量

局部变量示例3

4.6.2 全局变量的创建及使用

局部变量主要用于在程序内部传递数据，但是不能实现程序之间进行数据传递。局部变量的这个缺陷可以通过全局变量来实现，它可以同时在运行的多个VI或子VI之间访问和传递数据。**LabVIEW**中的全局变量与传统语言中的全局变量相类似。

4.6.2 全局变量的创建及使用

若要创建一个全局变量，可以从如图4-1所示的“结构”子选板中选择“全局变量”并将其拖放到程序框图上，得到如图4-34所示的全局变量的图标。

对于空的全局变量，可以通过“双击”图标，打开相对应的“全局变量文件”，添加各种变量到文件中，保存后，就可以使用。

4.6.2 全局变量的创建及使用

全局变量图标，
双击打开

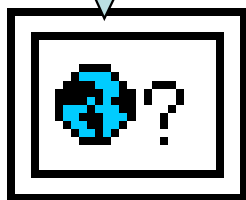


图4-34 全局变量

全局变量文件



图4-35 全局变量界面

4.6.2 全局变量的创建及使用

另一种方法创建全局变量是在**LabVIEW**前面板的菜单栏中选择“**文件**”下拉菜单并选择“**新建**”选项，则将打开一个如图**4-36**所示的窗口。在窗口中选择“全局变量”并确定同样可以创建一个全局变量。使用时，在程序框图界面，通过使用“**选择Vi**”来调入全局变量的**Vi**文件。

4.6.2 全局变量的创建及使用

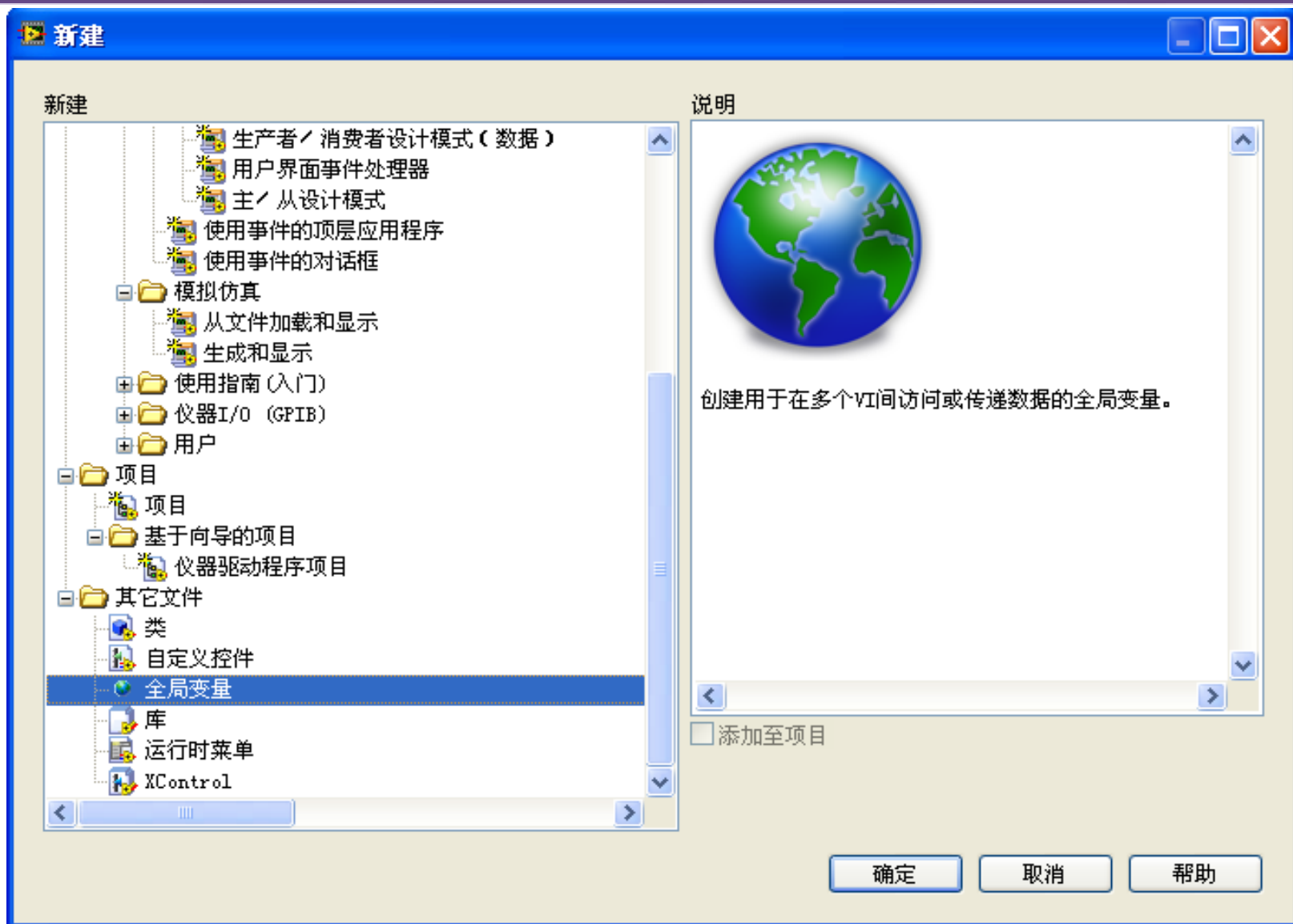


图4-36 从“文件”下拉菜单中创建全局变量

4.6.2 全局变量的创建及使用

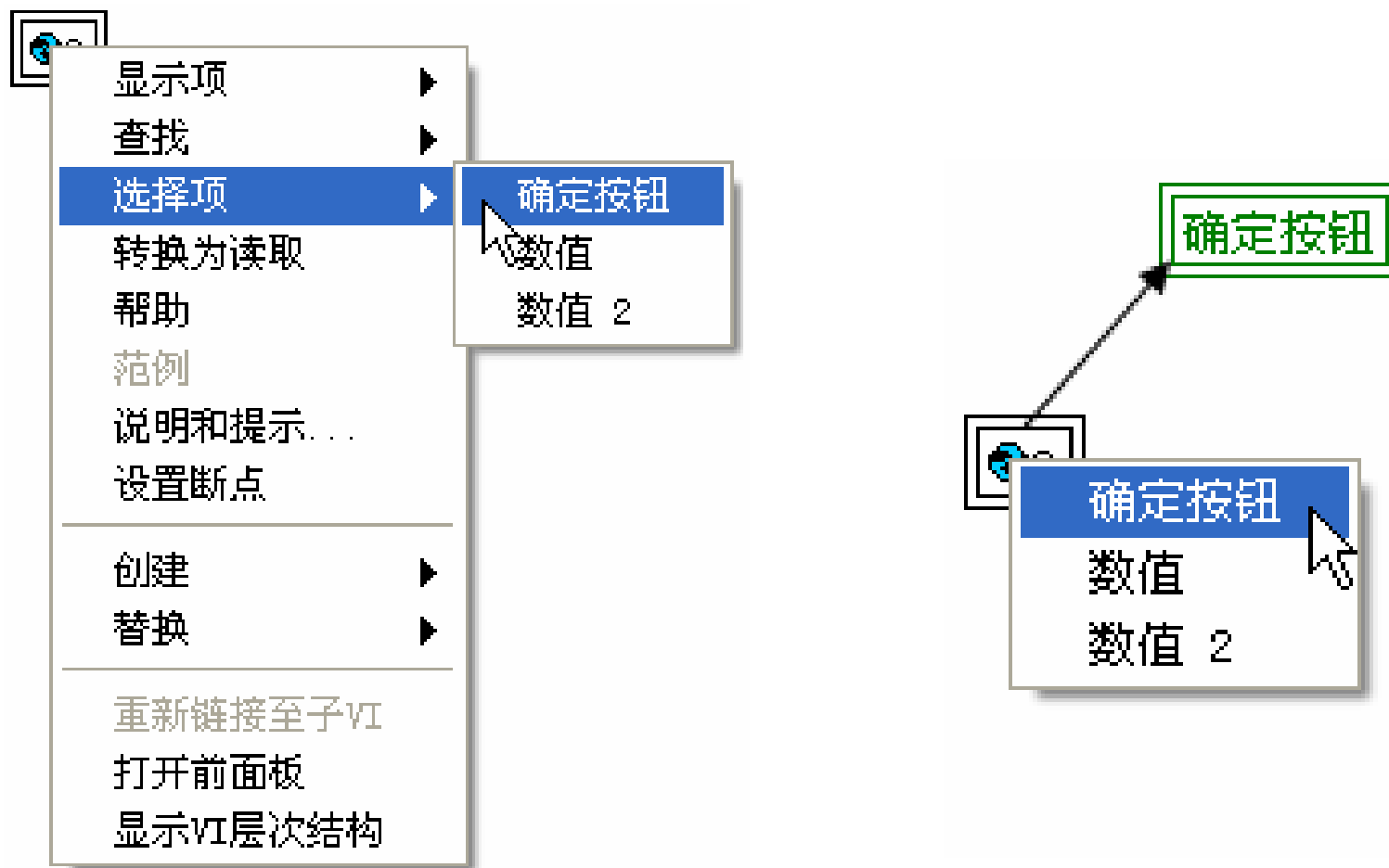
LabVIEW以自带**标签**区分全局变量，因此应当对前面板中的输入控件和显示控件使用描述性的自带标签来进行标注。可创建多个仅含有一个前面板对象的全局变量，也可创建一个含有多个前面板对象的全局变量从而将相似的变量归为一组。

全局变量定义文件

全局变量发送端

全局变量接收端

4.6.2 全局变量的创建及使用



(a) 通过鼠标右键关联全局变量 (b) 通过鼠标左键关联全局变量

图4-37 全局变量的关联方式

4.6.2 全局变量的创建及使用

- **【例4-3】** 图4-38所示的是一个温度调节过程。

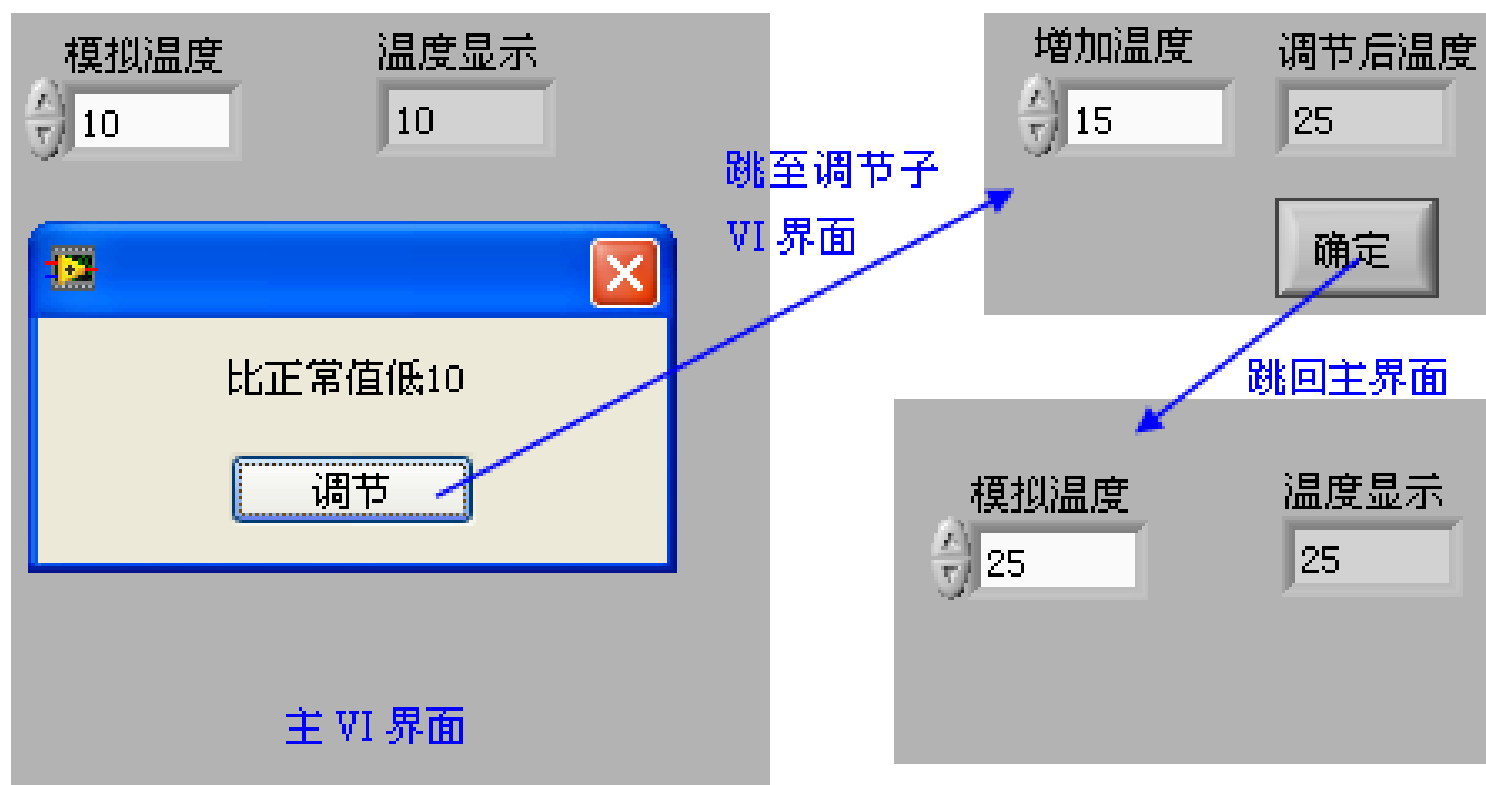


图4-38 程序运行过程与结果

4.6.2 全局变量的创建及使用

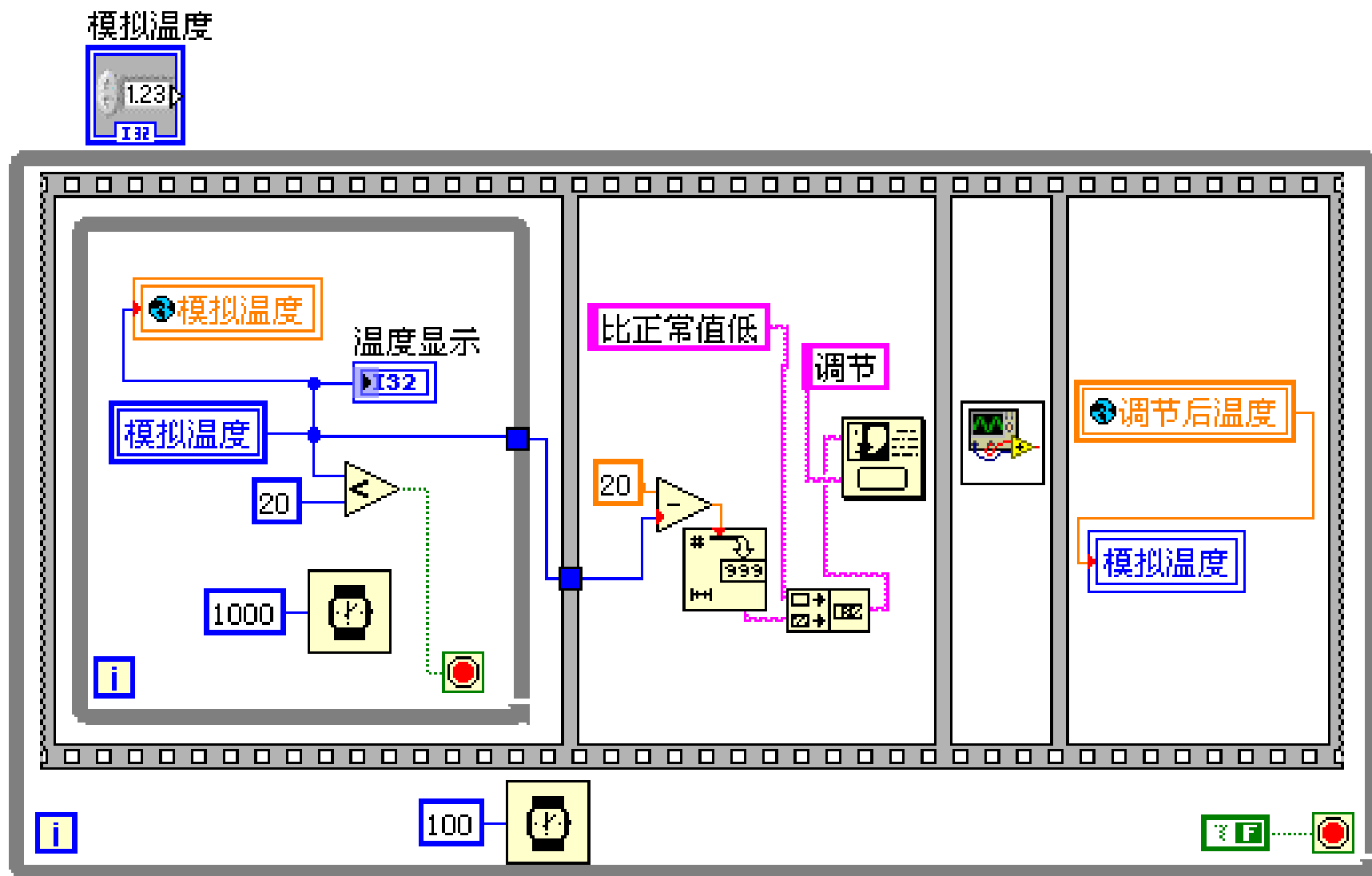


图4-39 温度调节系统程序框图

4.6.2 全局变量的创建及使用

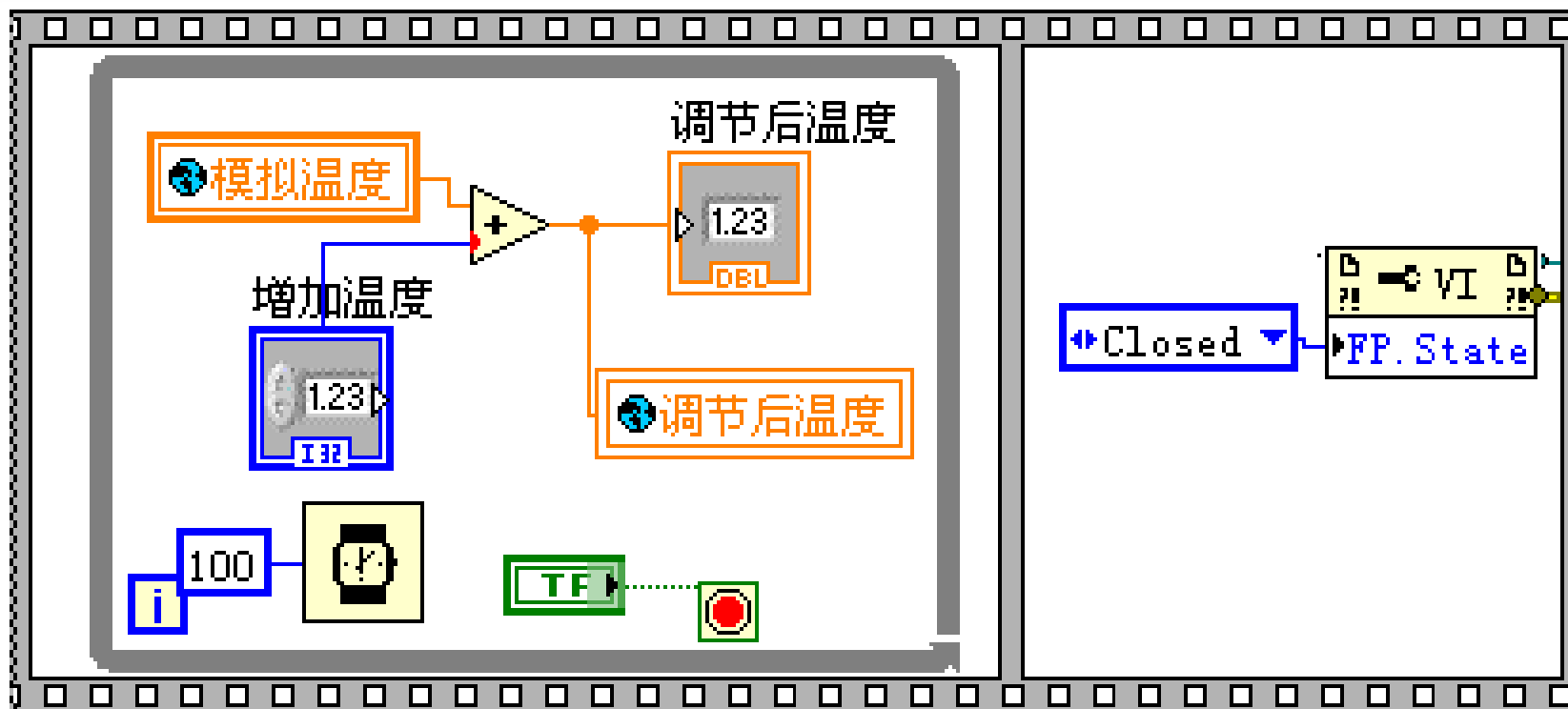


图4-40 温度调节子VI框图

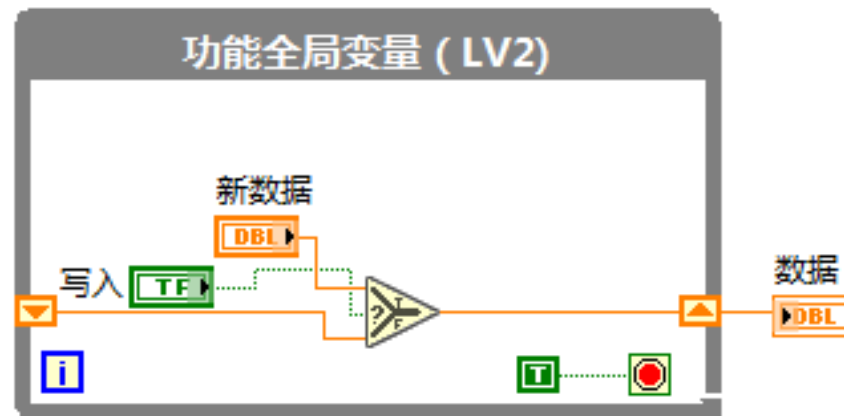
全局变量示例

全局变量服务器

客户端/服务器模式 全局变量停止多个VI

4.6.3 函数全局变量

通过特殊的函数实现。具体来讲，是通过未初始化的移位寄存器实现的。



函数全局变量示例

函数全局变量不存在数据竞争的问题。

函数全局变量不存在数据复制问题，效率更高。

4.7 公式节点

公式节点是一种便于在程序框图上执行数学运算的文本节点。公式节点的引入，使**LabVIEW**的编程更加灵活，无需使用任何外部代码或应用程序，且创建方程时无需连接任何基本算术函数，采用公式节点实现计算公式在一定程度上减少了编程的工作量。

“表达式节点”只适合于单变量的计算节点。

4.7 公式节点

除接受文本方程表达式外，公式节点还接收为C语言中的**if**语句、**while**循环、**for**循环和**do**循环。需要注意的是出现在公式节点框架上的所有变量必须声明为输入或输出。

4.7.1 公式节点的创建

也可以在程序框图中单击鼠标右键选择“**函数→结构→公式节点**”并将其拖放至程序框图中。



图4-41 公式节点界面

4.7.2 公式节点的使用

在公式节点框图的左边或者右边的边框上单击鼠标右键，用户从弹出的快捷菜单中选择“添加输入”或者“添加输出”，就可得到如图4-42所示的带有输入输出变量端口的公式节点框图。

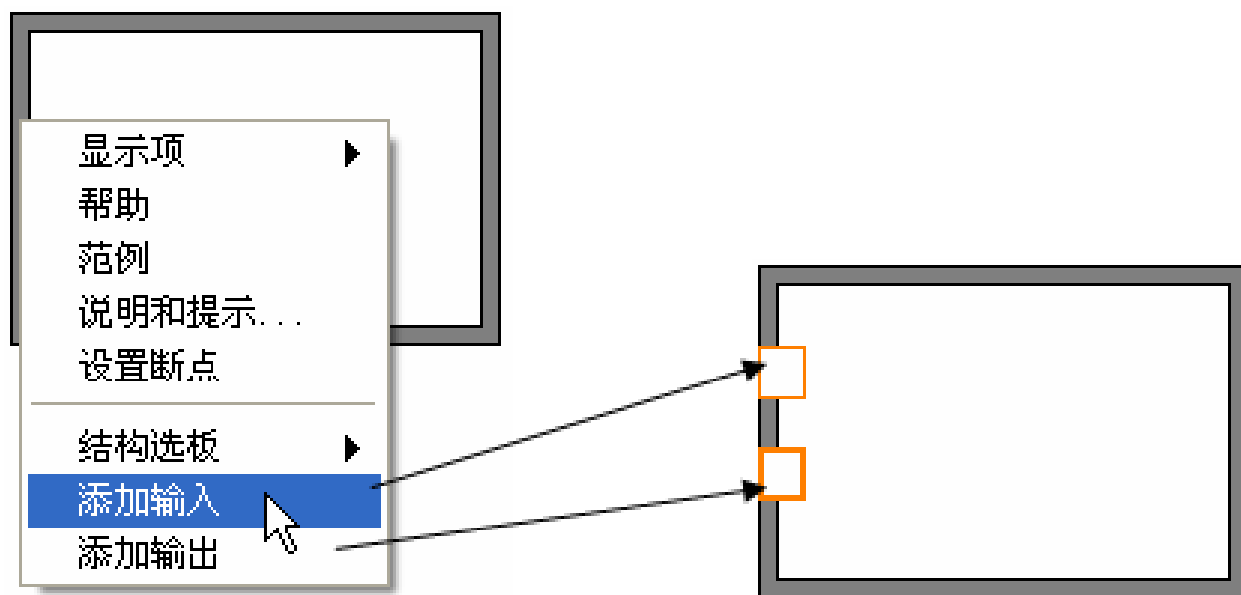


图4-42 带有输入输出变量端口的公式节点框图

4.7.2 公式节点的使用

我们可以在公式节点中输入需要运算的公式，然后在输入端口和输出端口中输入相应的输入变量和输出变量的变量名。当把鼠标放在相应的端口上变成小箭头时，我们双击鼠标左键，就可以往端口中输入相应标签了。

为了可以方便的操作和显示输入变量和输出变量的值，可以创建数值输入控件和数值输出控件。

4.7.2 公式节点的使用

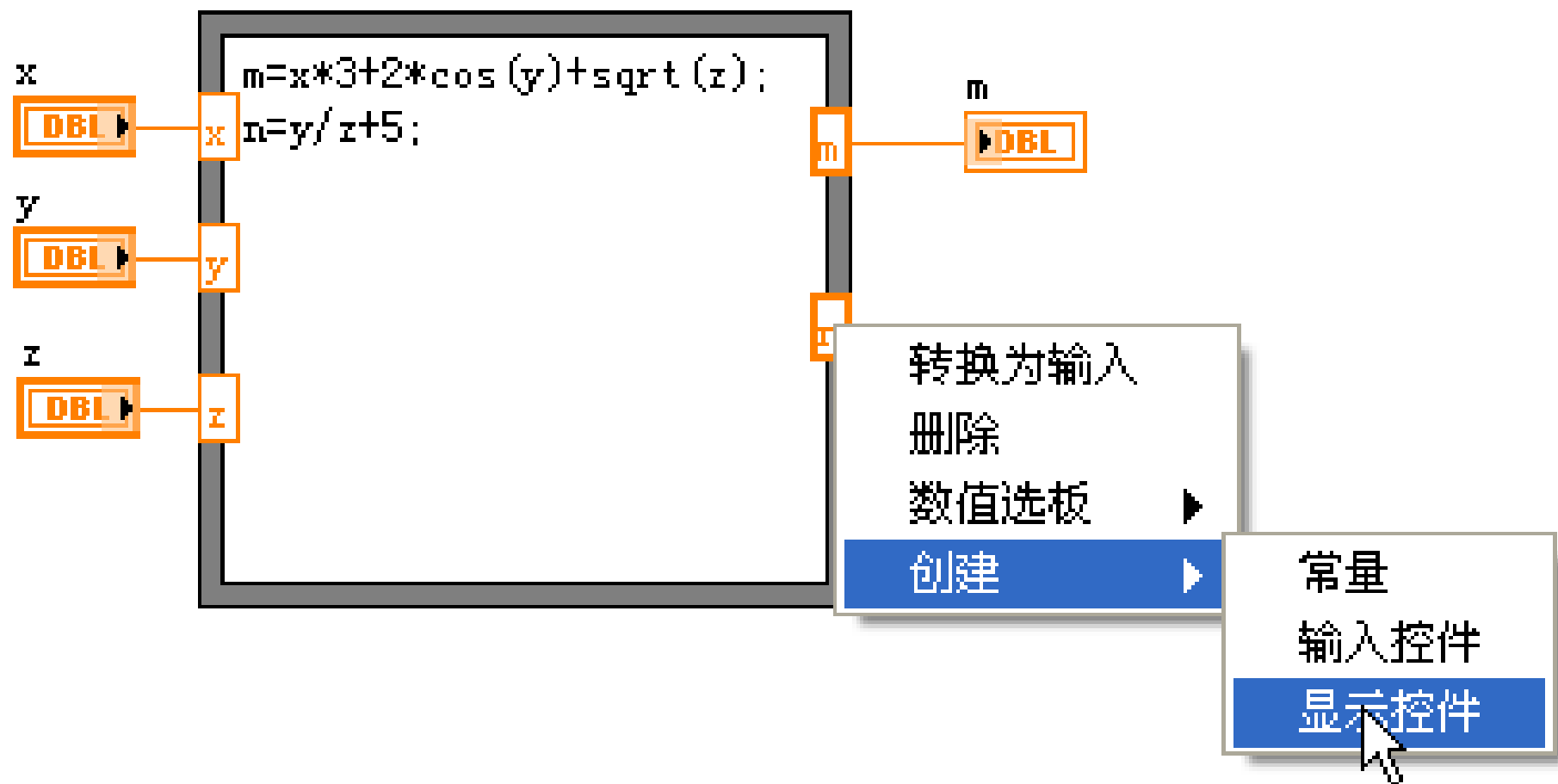


图4-43 创建显示控件

4.7.2 公式节点的使用

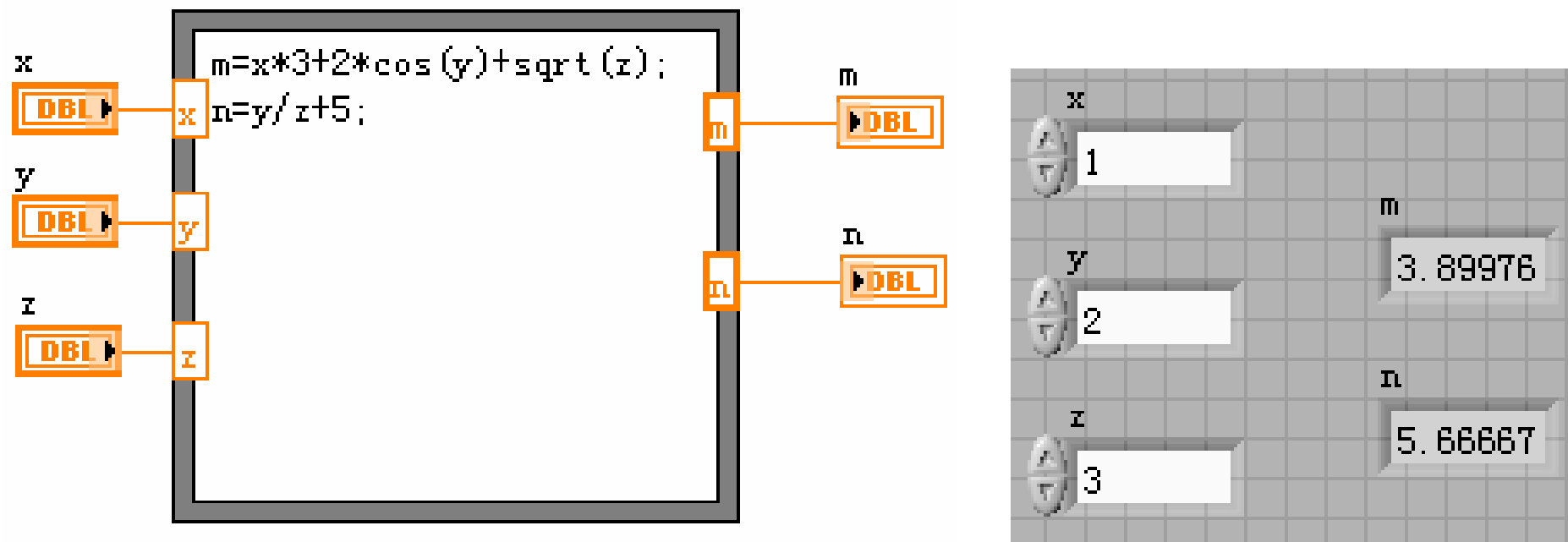
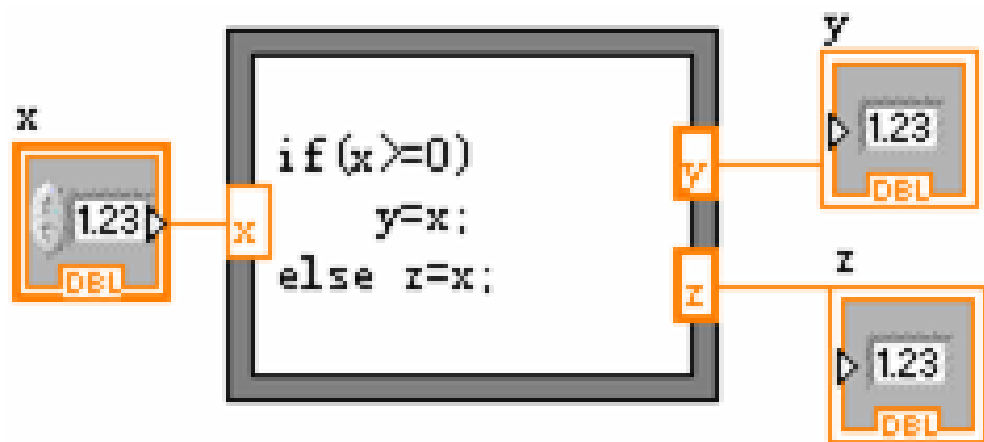
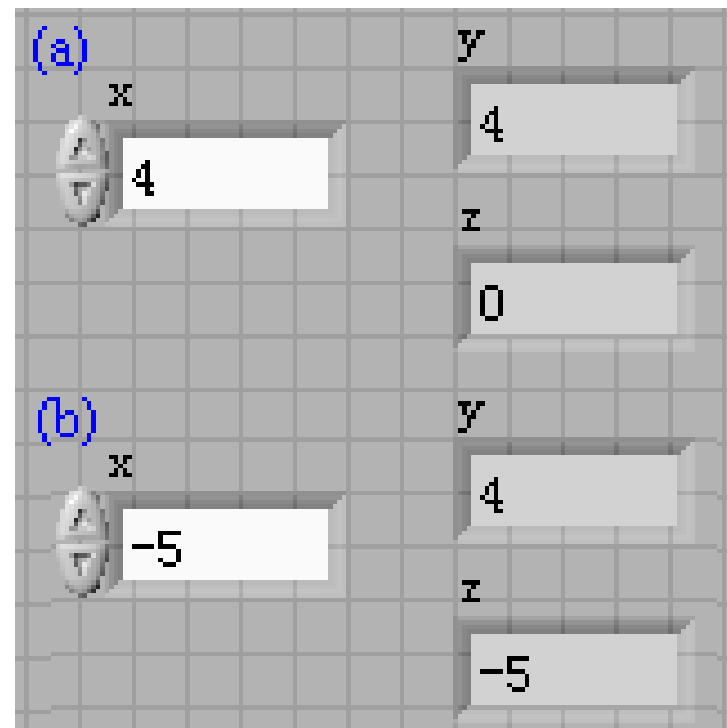


图4-44 在公式节点中实现多个计算公式

4.7.2 公式节点的使用



公式节点语法



公式节点不支持字符、字符串定义

图4-45 在公式节点中实现if-else功能

公式节点示例1

公式节点示例2

公式节点示例3

4.8 MathScript节点和MATLAB脚本节点

4.8.1 MathScript节点的创建

LabVIEW MathScript是一种直接可以用于编写函数和脚本的文本语言。

按照**MATLAB**语法编写的脚本通常可在**LabVIEW MathScript**中运行。虽然**MathScript**引擎可执行**MATLAB**脚本，但不支持某些**MATLAB**软件所支持的函数。运行**MathScript**脚本，不需**MATLAB**软件的支持。

4.8 MathScript节点和MATLAB脚本节点

MathScript节点的创建有两种方法：
一是直接在如图4-1所示的“**结构**”选板中选择**MathScript**节点并拖放到程序框图中，
二是在程序框图中单击鼠标右键选择“函数→数学→脚本与公式→ **MathScript节点**”
然后拖放至程序框图中。

两种方法创建的**MathScript**节点完全一样，其界面如图4-46所示。

4.8 MathScript节点和MATLAB脚本节点

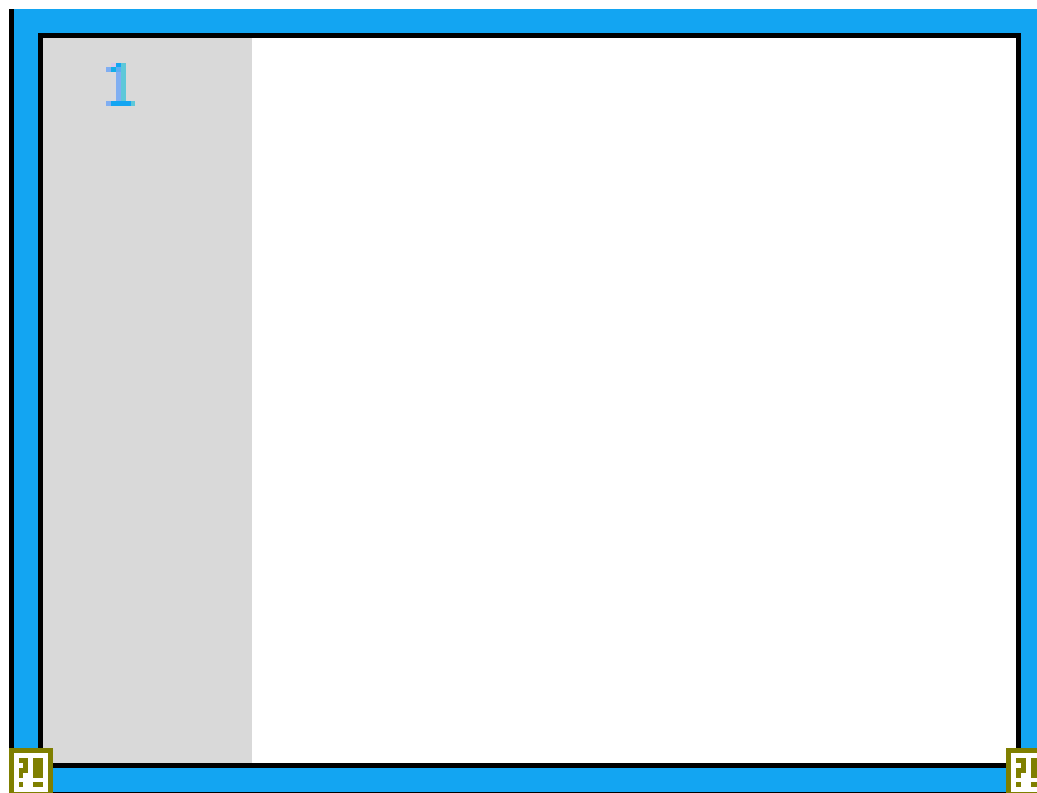
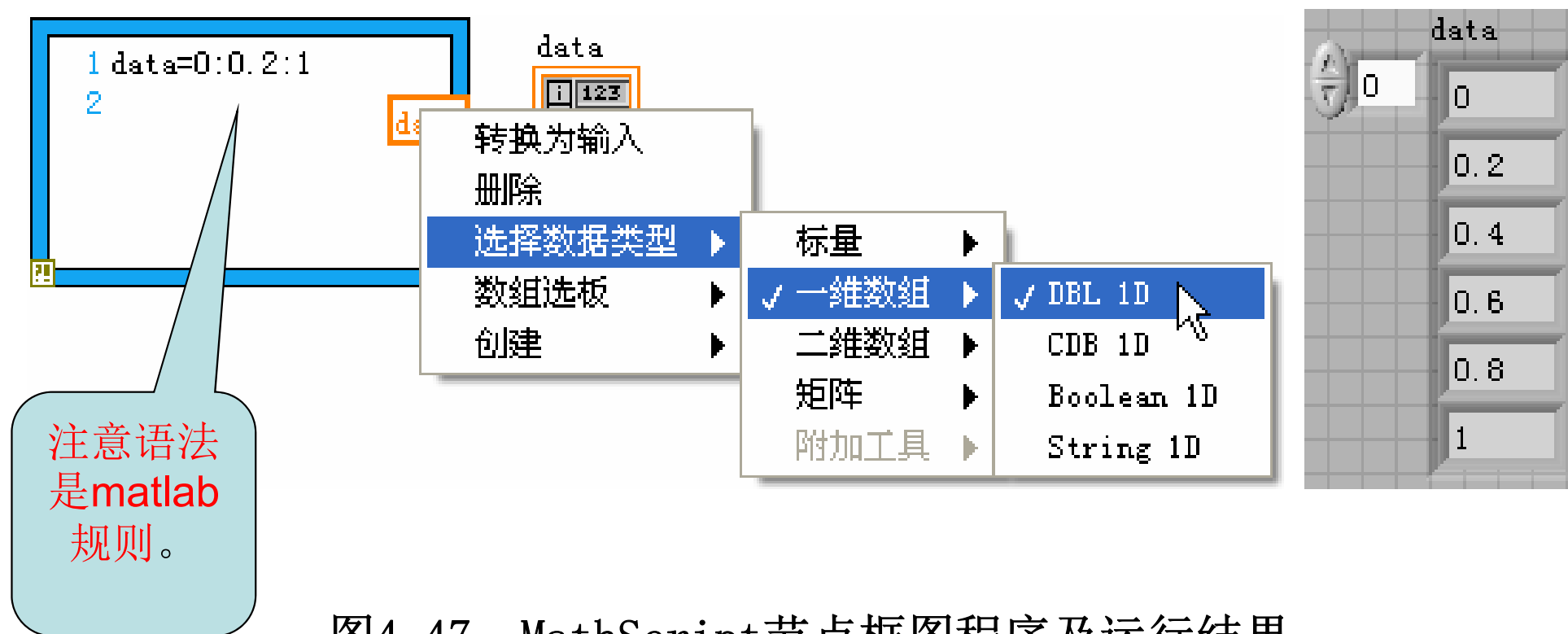


图4-46 MathScript节点界面

4.8 MathScript节点和MATLAB脚本节点

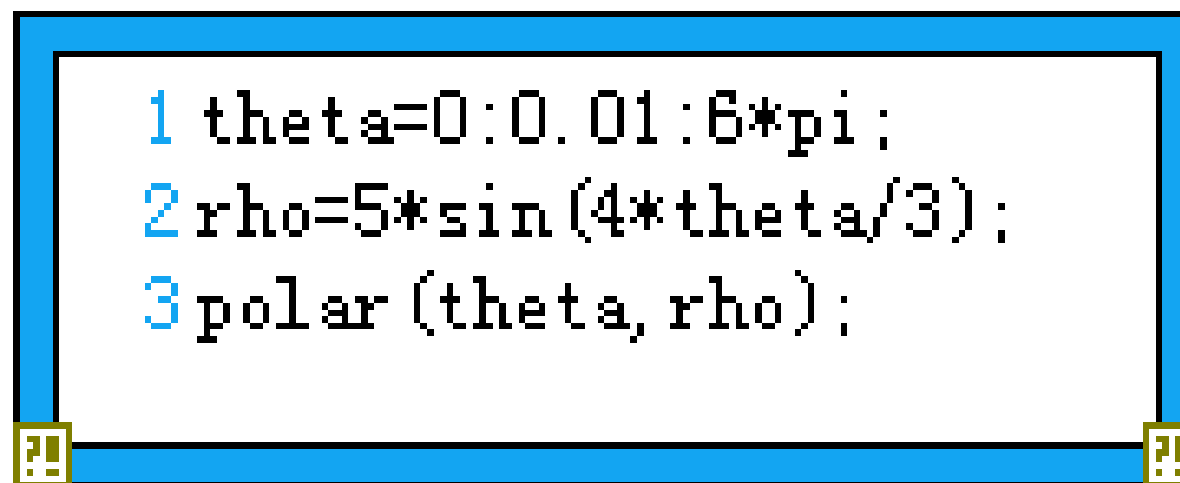


4.8.2 MathScript节点示例分析

【例4-4】 下面通过使用**MathScript**节点实现绘制一个下式所示的曲线。

$$\rho = 5 \sin\left(\frac{4\theta}{3}\right) \quad 0 \leq \theta \leq 6\pi$$

4.8.2 MathScript节点示例分析



```
1 theta=0:0.01:6*pi;  
2 rho=5*sin(4*theta/3);  
3 polar(theta,rho);
```

图4-48 MathScript节点框图程序

4.8.2 MathScript节点示例分析

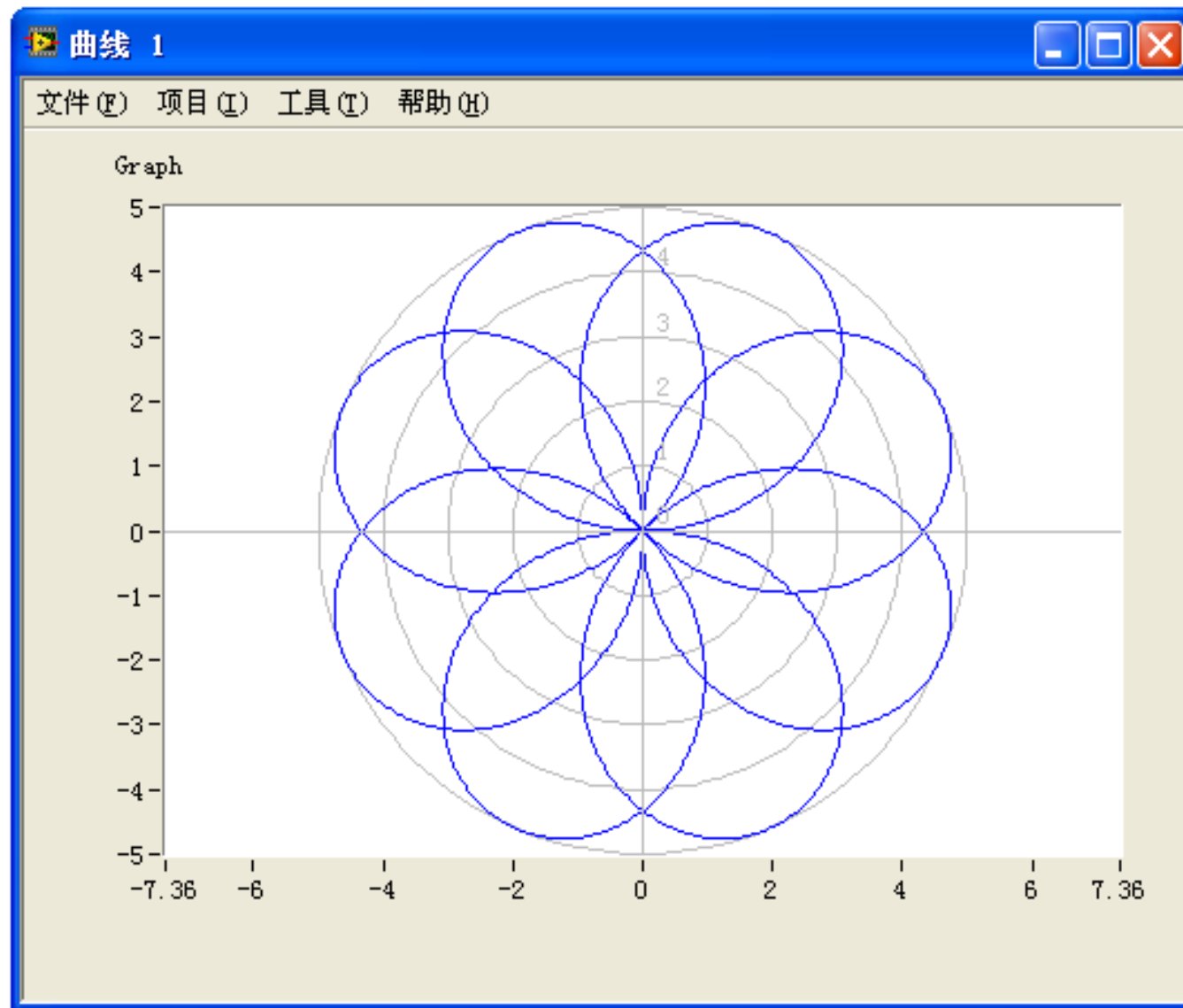


图4-49 MathScript节点框图程序的运行结果

4.8.2 MathScript节点示例分析

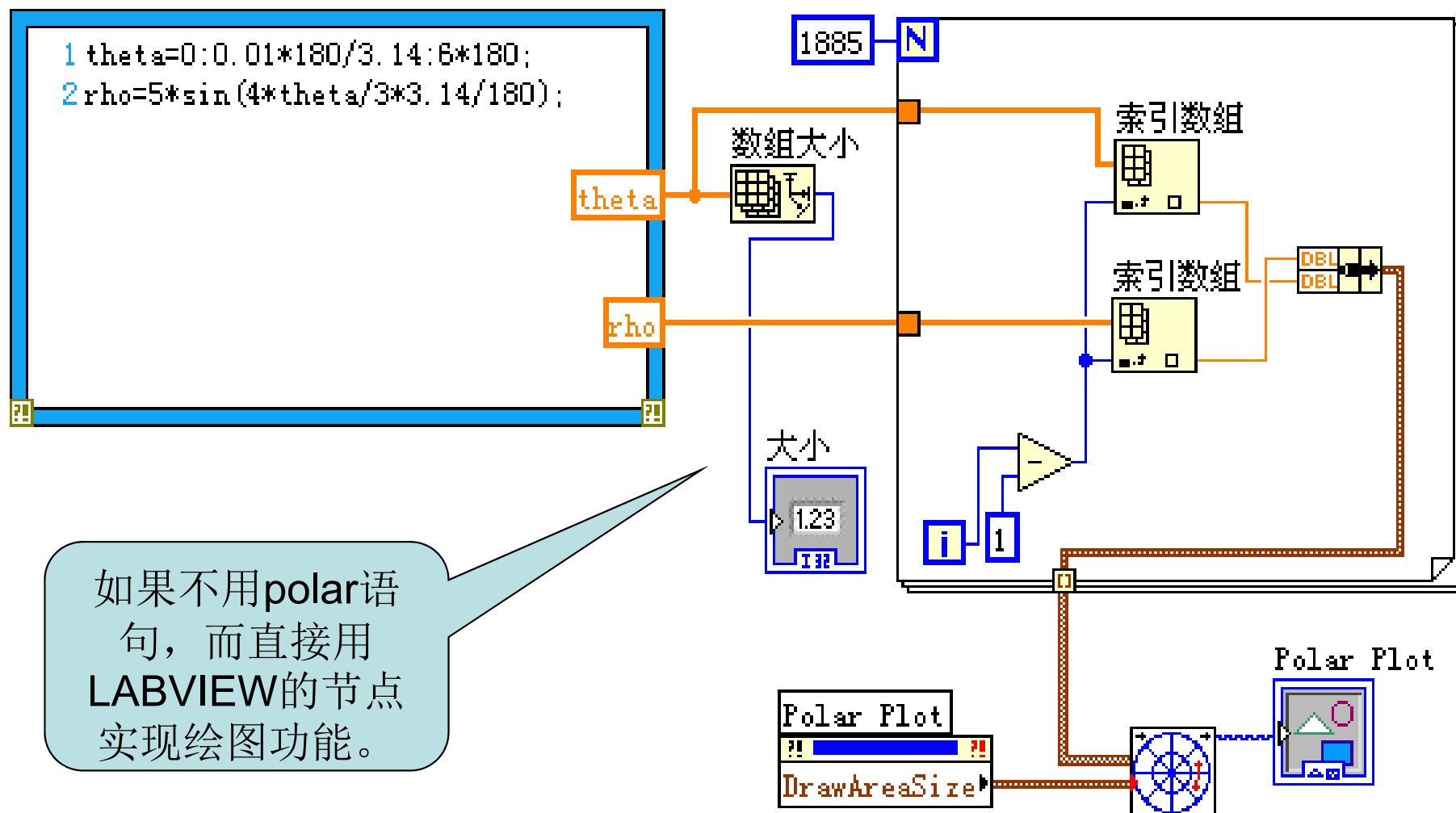
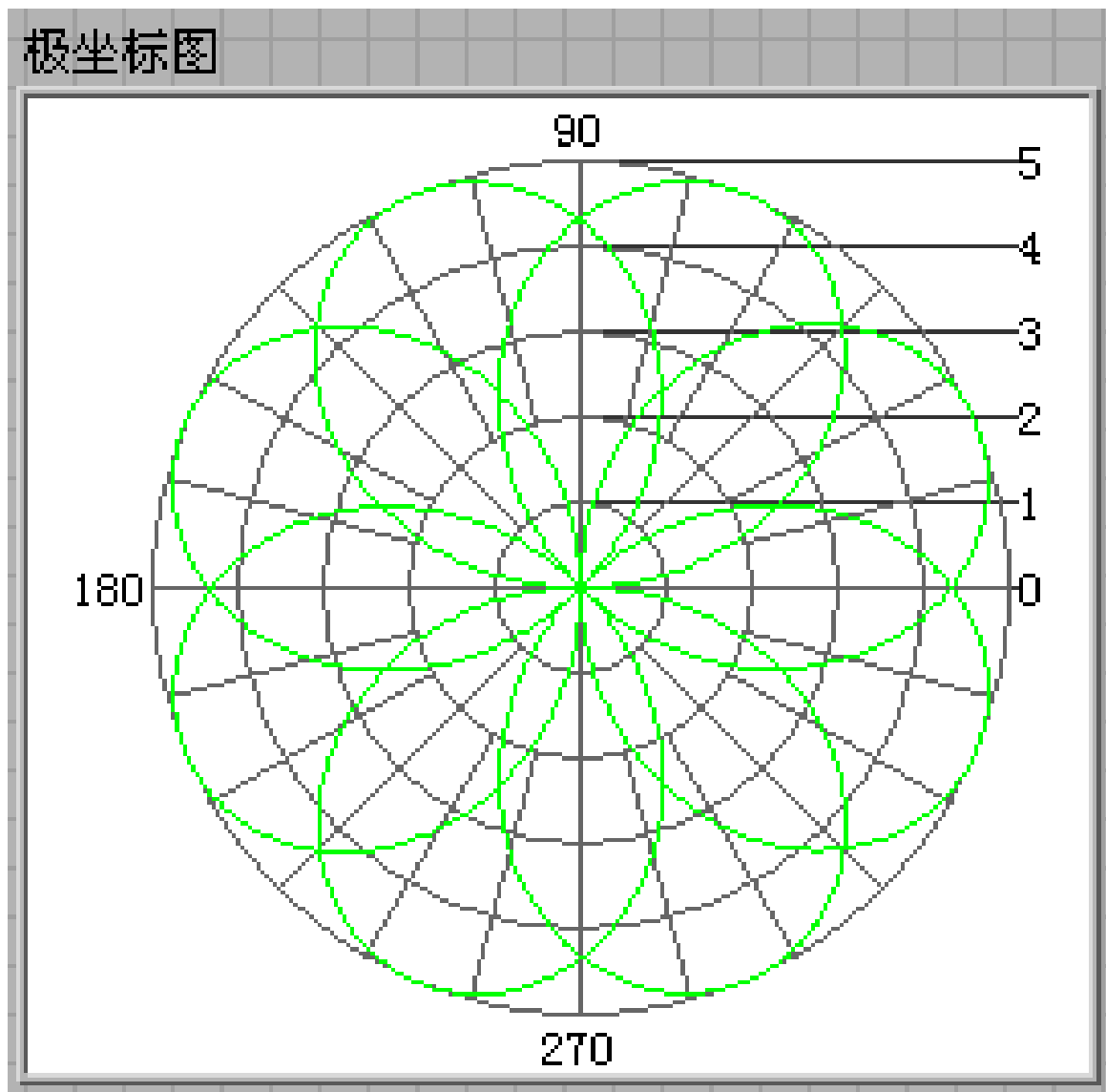


图4-50 MathScript节点框图程序

4.8.2 MathScript节点示例分析



MathScript
节点示例

输入变量的定义

MathScript
节点示例

MathScript
节点示例

图4-51 MathScript节点框图程序的运行结果

4.8.3 MATLAB脚本节点

MATLAB节点的创建方法与**Mathscript**节点的第二种创建方法相同，可在程序框图中单击鼠标右键选择“函数→数学→脚本与公式→脚本节点→**MATLAB**脚本”然后拖放到程序框图中来创建一个**MATLAB**脚本节点。**MATLAB**脚本节点和**MathScript**节点的框图与结构很相似，如图4-52所示。虽然也可以选择输出变量的数据类型，但是要注意到选项的差别。

MATLAB节点运行需**MATLAB**软件支持。

4.8.3 MATLAB脚本节点

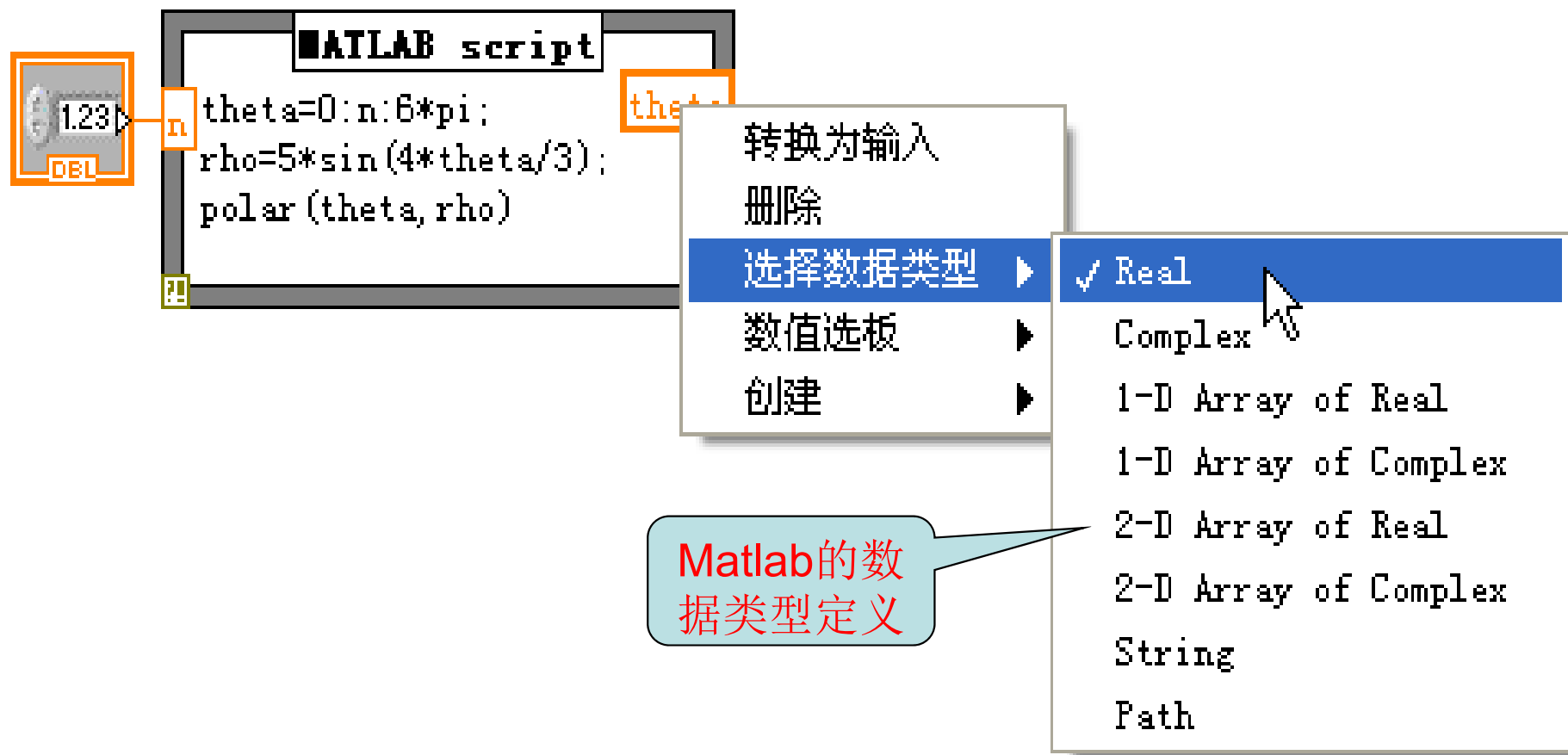


图4-52 MATLAB脚本节点框图

4.8.3 MATLAB脚本节点

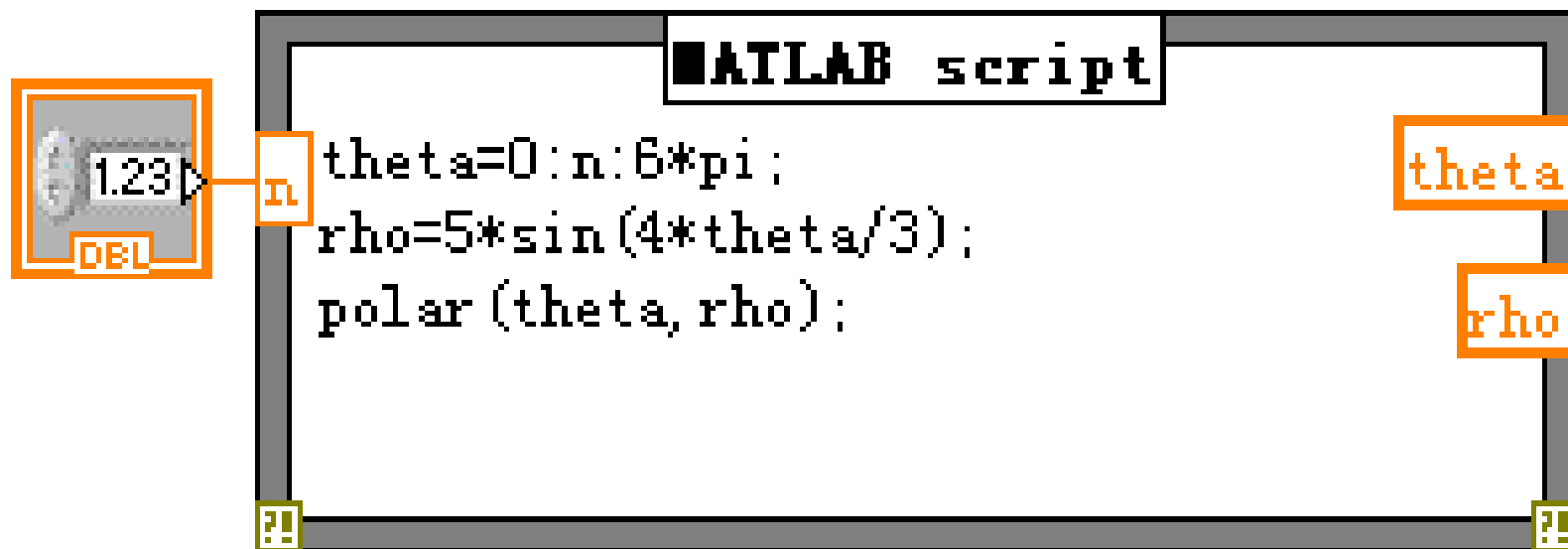


图4-53 MATLAB脚本节点的程序框图

MATLAB脚本节点示例

MATLAB脚本节点示例

4.8.3 MATLAB脚本节点

在
matlab
中完成。

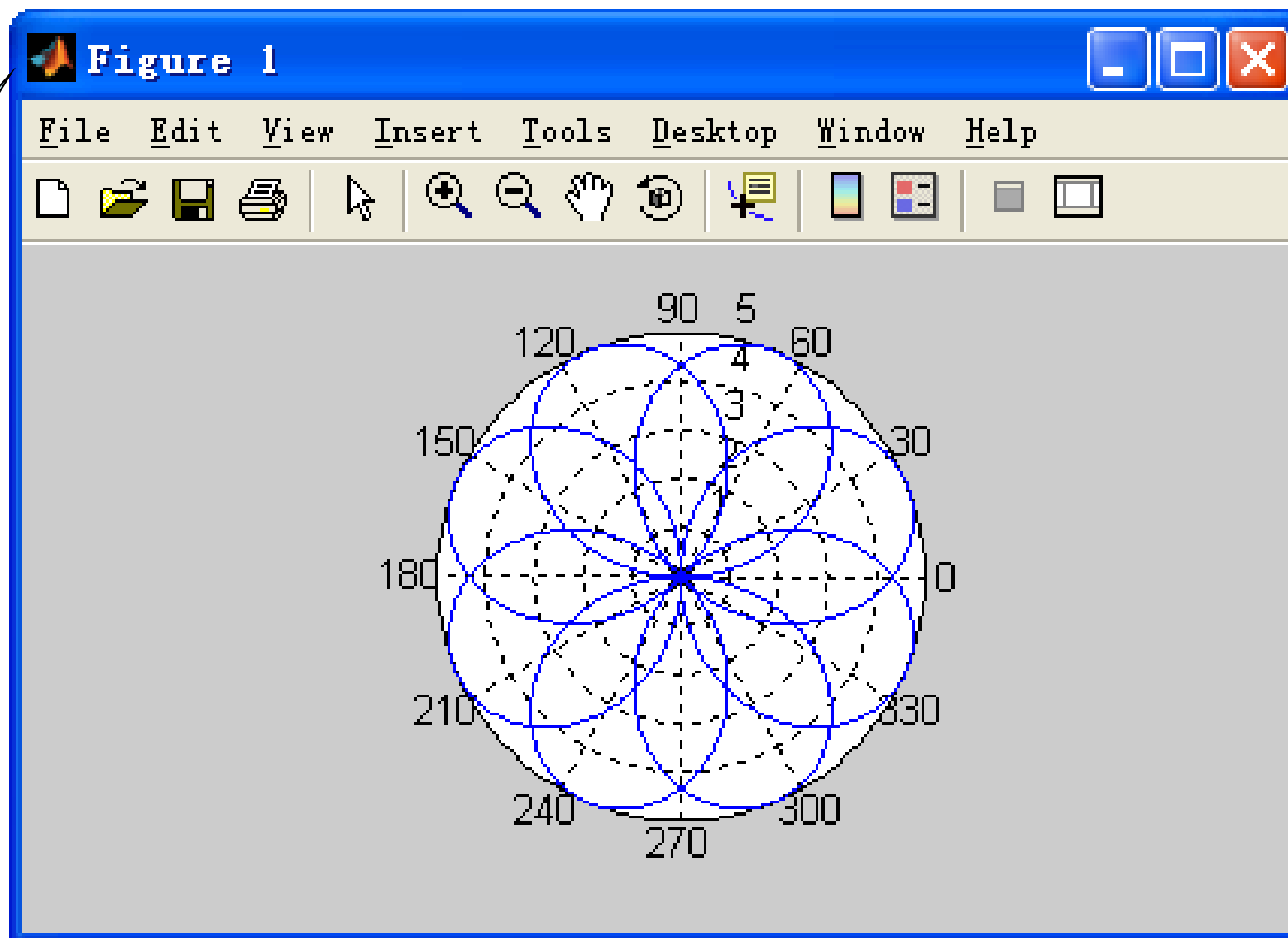


图4-54 MATLAB脚本节点的程序框图

4.9 属性节点

LabVIEW为前面板中的大部分控件都预定义了各种可用属性，其中包含了前面板控件的外观、值和功能行为，如定义控件的可见性、闪烁状态及数据操作的边界、文本的宽度等。但前面板的控件通常默认为只有输入和输出显示功能，**需要通过创建属性节点去获取并设置控件隐含的属性。**

4.9 属性节点

一般来说，属性节点的创建方法有两种：一种方法是从函数选板中获取没有属性标识的**空属性节点**放置于程序框图中，然后为属性节点配置相应的类和属性，使用**引用节点**去指派需要配置属性的前面板控件对象，完成属性节点的创建；另一种方法是在框图中的控件上单击鼠标右键弹出快捷菜单，用户在快捷菜单中选择**创建属性节点的菜单项**直接创建属性节点，然后在菜单项中选择要创建的属性。

4.9.1 属性节点的直接创建法

在控件上单击鼠标右键，从弹出的快捷菜单上选择“**创建→属性节点**”选项，在“属性节点”选项中选择所要建立的控件属性，这就能为控件直接创建属性节点。

4.9.1 属性节点的直接创建法

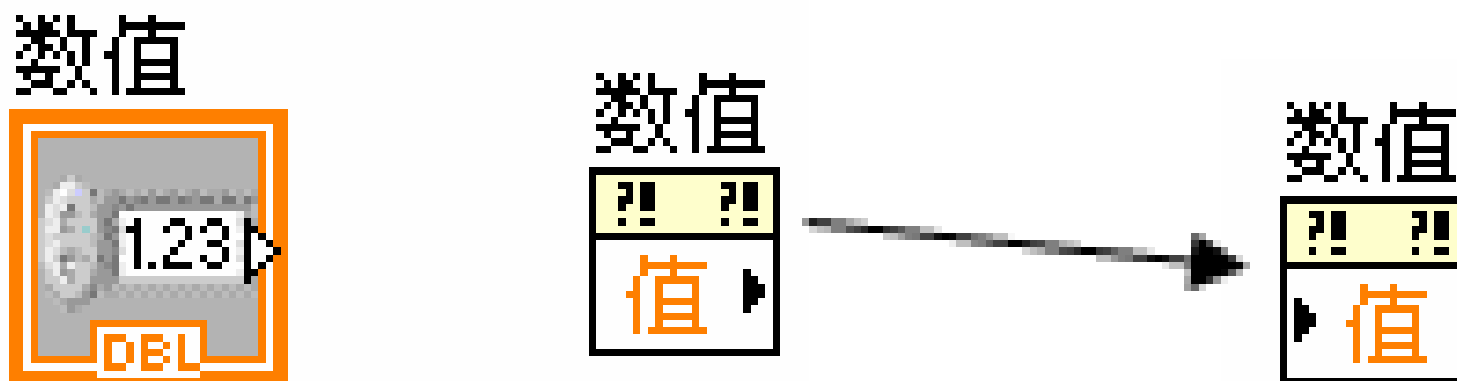


图4-55 属性节点的创建

4.9.1 属性节点

下面介绍几种常用的属性节点及其使用方法。

(1) 键选中属性

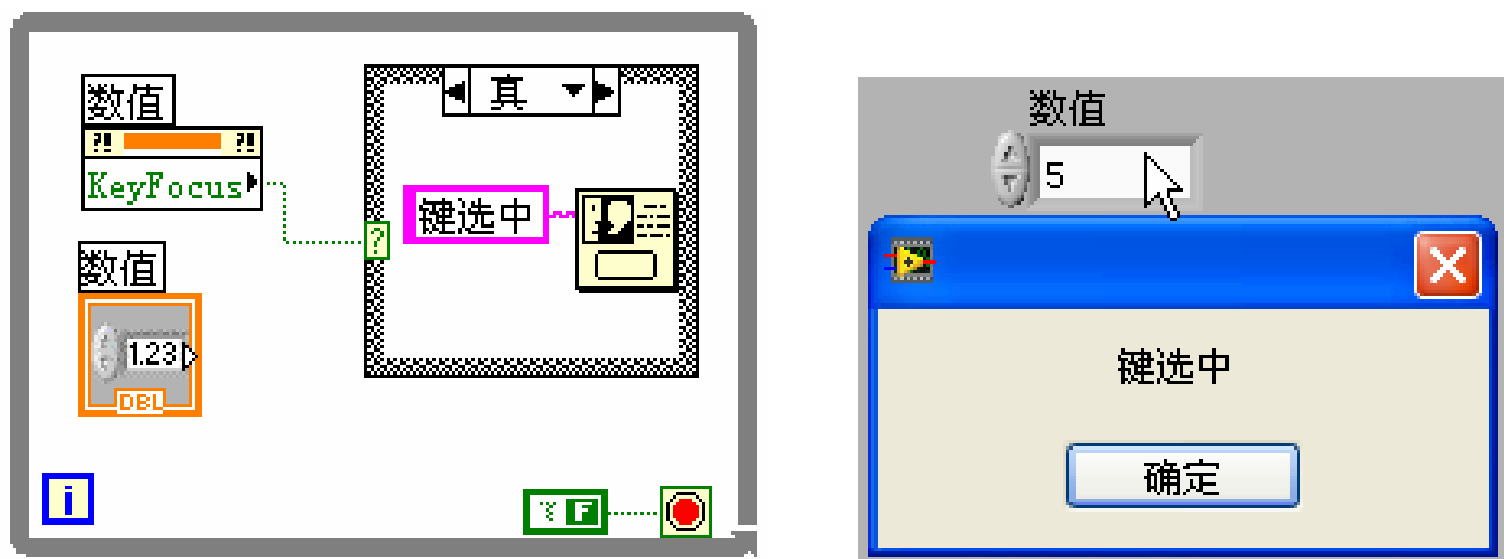


图4-56 键选中属性

(2) 禁用属性

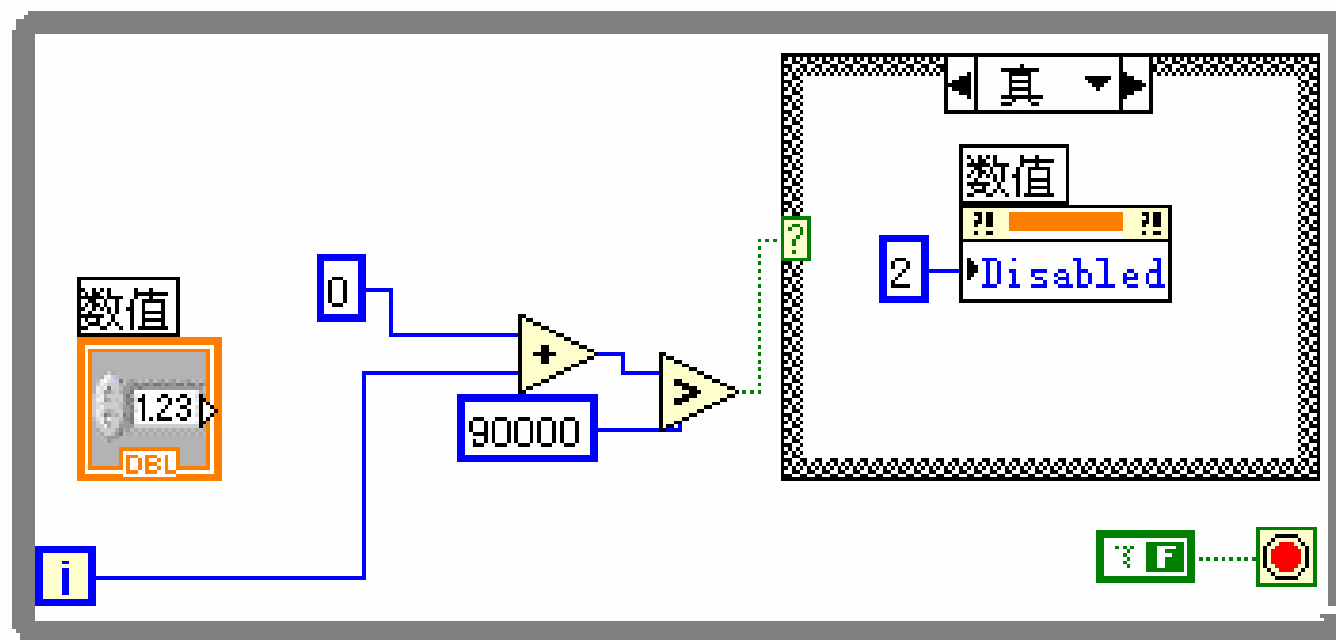


图4-57 禁用属性

(3) 可见属性

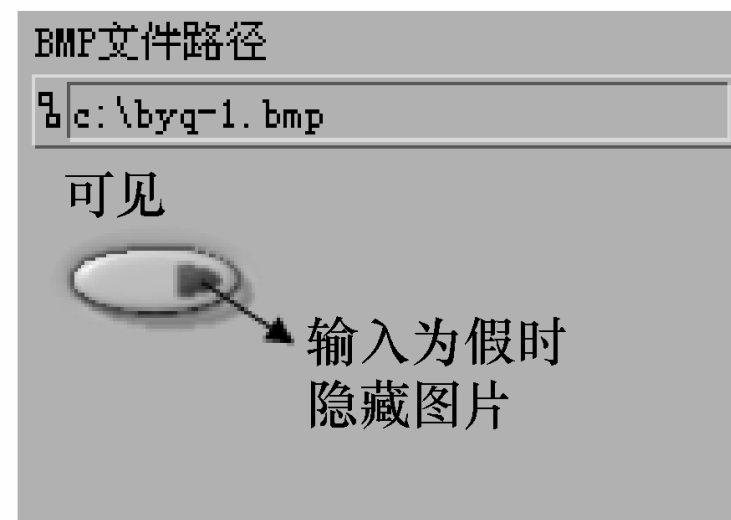
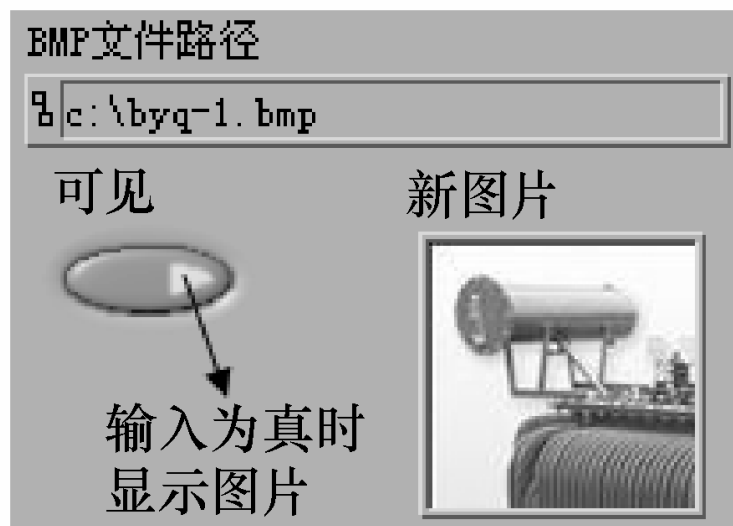
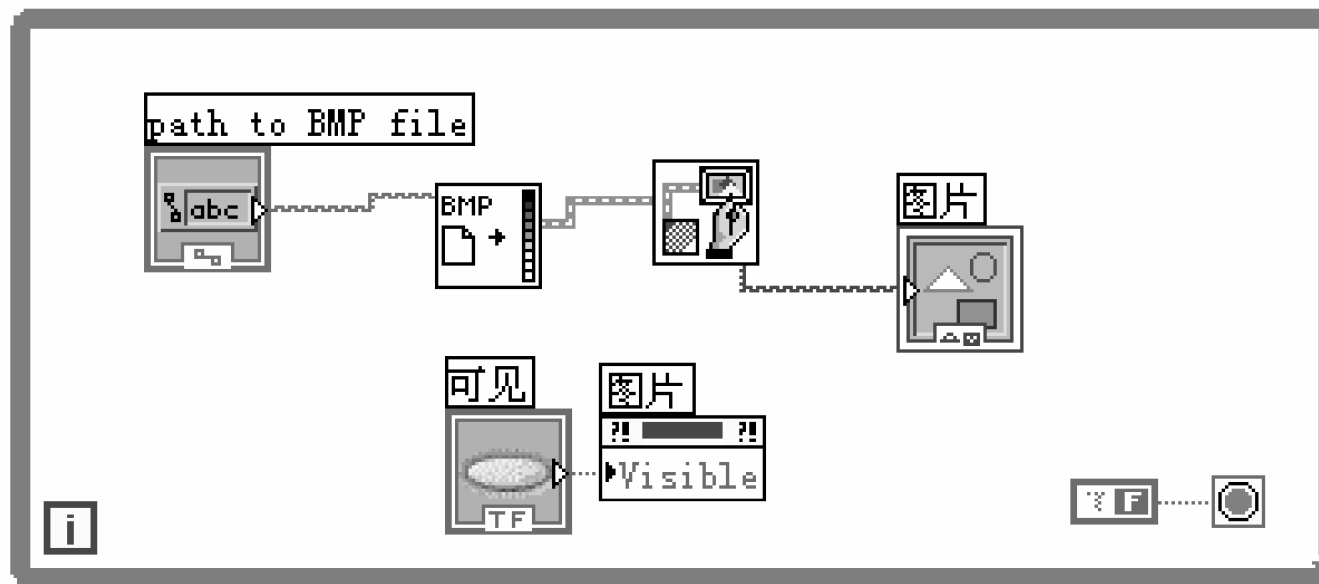


图4-58 可见属性

(4) 衬底颜色属性

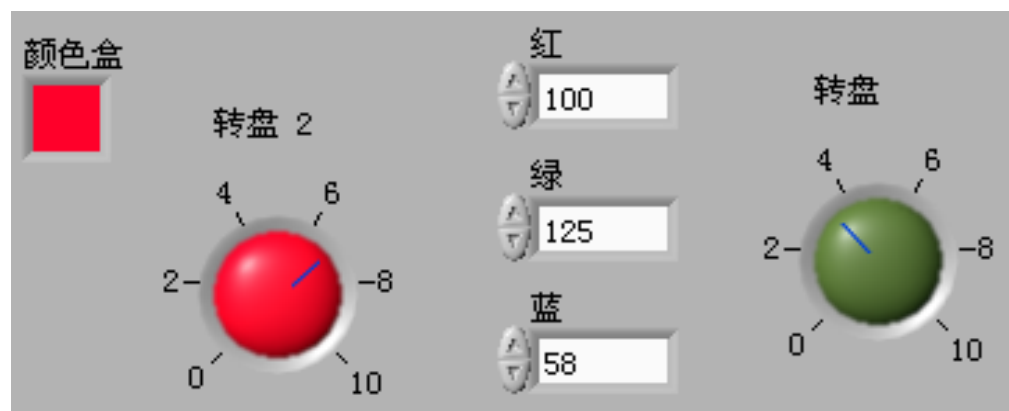
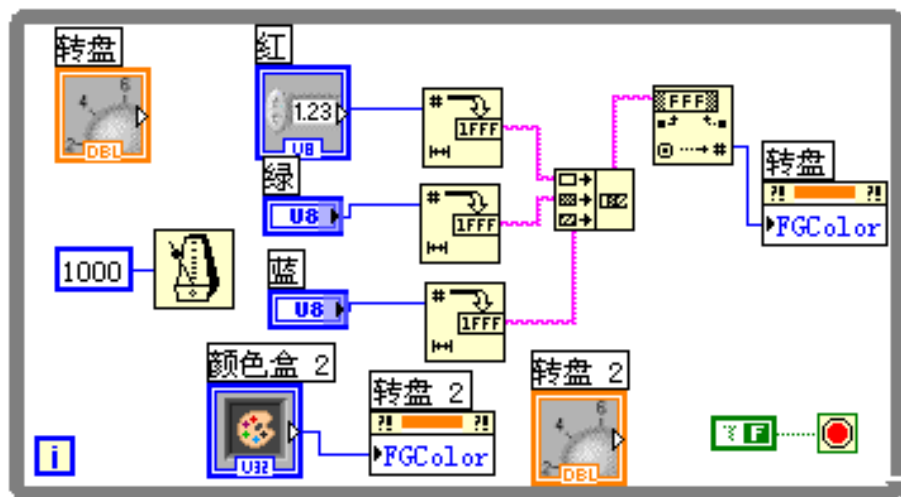
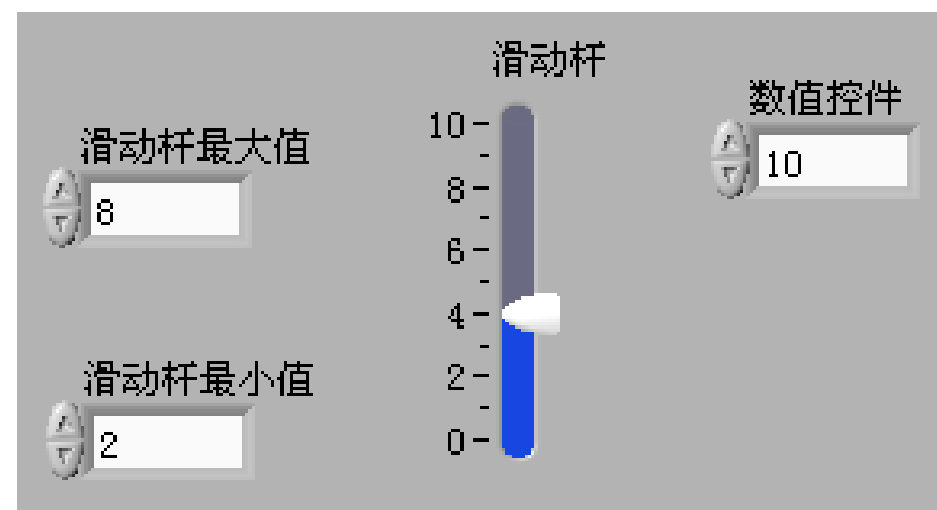
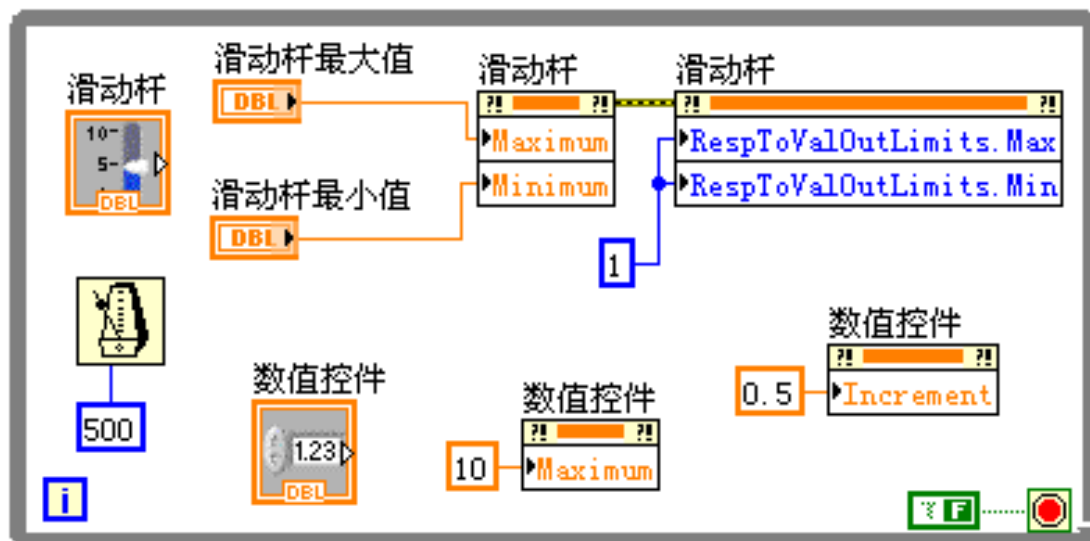


图4-60 修改衬底颜色属性的运行结果

图4-59 衬底颜色属性

(5) 数据范围属性



(a) 数据范围属性的使用

(b) 程序运行结果

图4-61 数据范围属性示例

(5) 数据范围属性

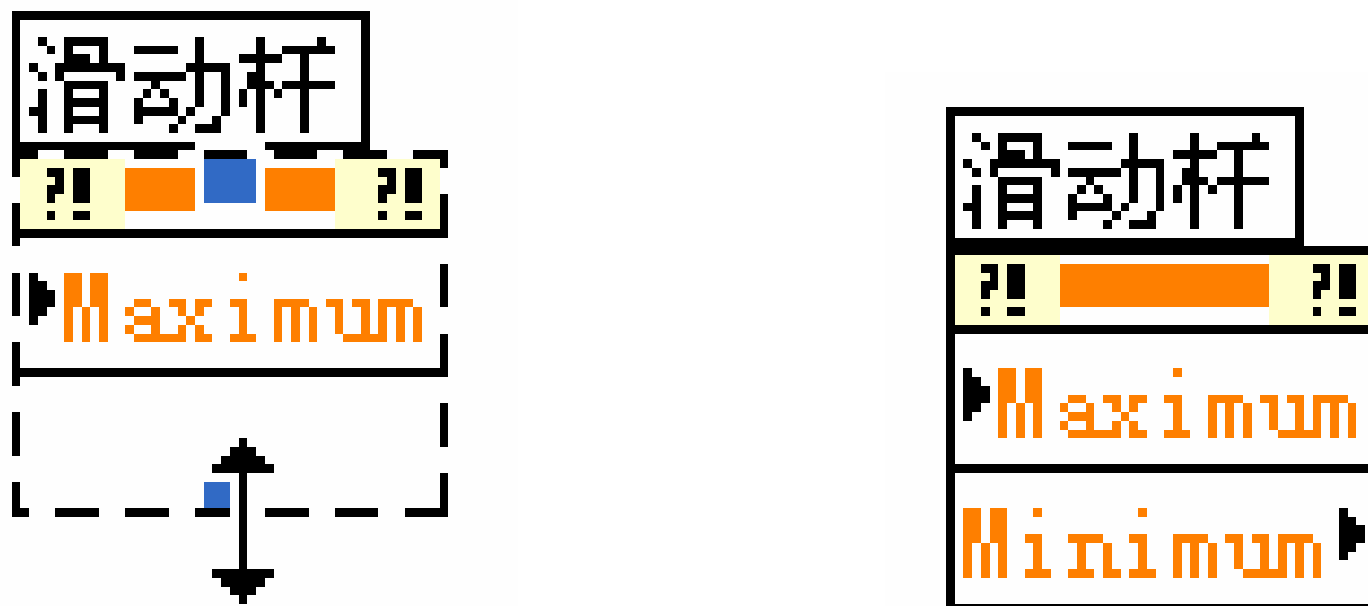


图4-62 多属性节点端子的创建

4.9.2 属性节点的编程创建法

用编程方法创建属性节点就是先获取空属性节点，再为其配置对象并选择对象属性。空属性节点即没有连接任何对象的属性节点，其位于“函数”选板下“**应用程序控制**”子选板中。

4.9.2 属性节点的编程创建法

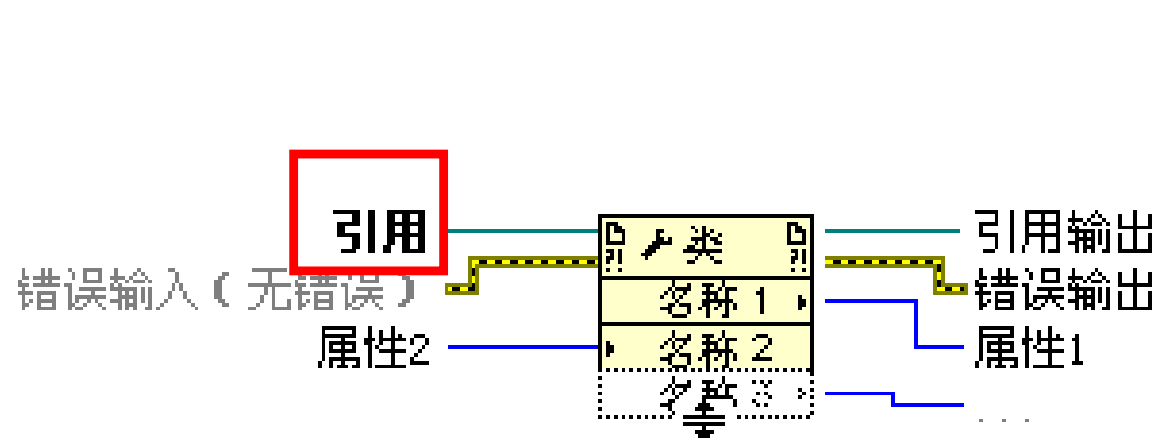


图4-63 属性节点接线端子

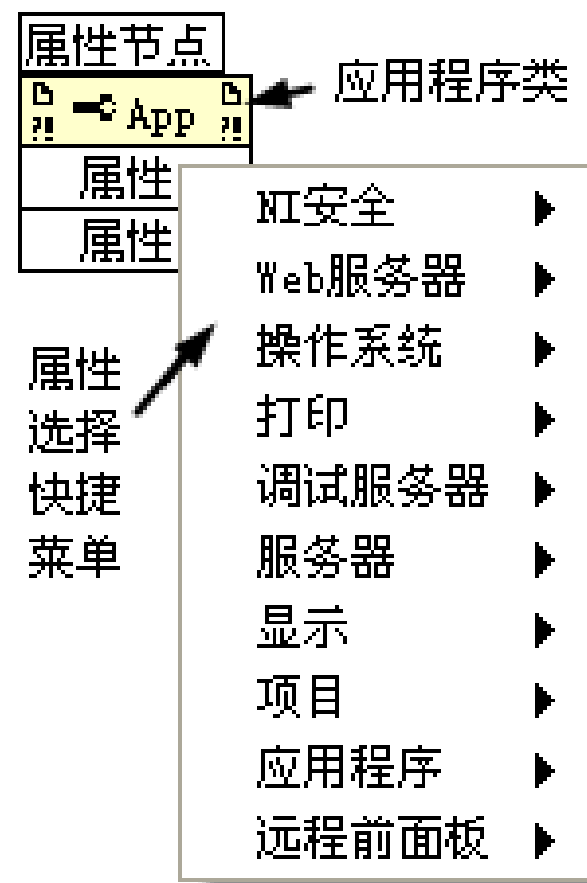


图4-64 属性列表菜单

4.9.2 属性节点的编程创建法

使用编程方法为控件添加属性时，首先要获取控件的引用节点以获取该控件的属性。创建引用节点的方法是在控件上单击鼠标右键，在快捷菜单中选择“创建→引用”菜单项，创建后的引用节点如图4-65所示。

4.9.2 属性节点的编程创建法

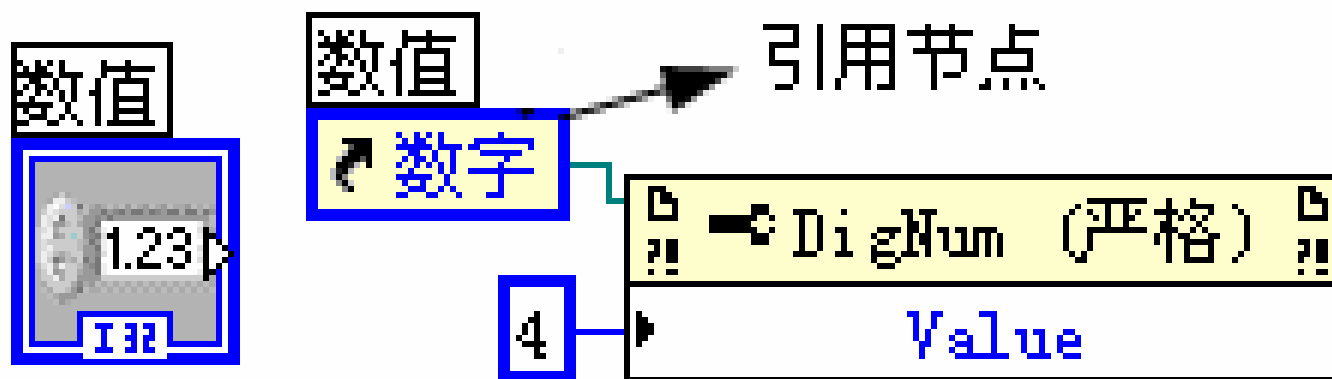


图4-65 引用节点

滑动杆

编程创建
示例

可见属性示例

属性运用示例

禁用属性示例

控件颜色示例

本章小结

本章主要介绍了**LabVIEW**的**2**循环（**For**循环、**While**循环）和**3**结构（条件结构、顺序结构、事件结构）。**For**循环和**While**循环主要用于重复执行位于循环内部的程序。

条件结构和顺序结构主要用于控件数据流。事件结构主要用于对来自于用户界面、外部**I/O**或其他方式事件的异步通知。

本章小结

循环与结构是**LabVIEW**的重点，是学习**LabVIEW**的基础。必须学会熟练的使用**For**循环、**While**循环、事件结构等常用编程语句。