

第一章 LabVIEW 导航

- LabVIEW编程环境
- LabVIEW联机帮助
- 编辑前面板
- 编辑程序框图

第一章 LabVIEW 导航

- VI的运行和调试
- VI子程序
- 数据类型
- 数据运算

1.1 LabVIEW编程环境

1. LabVIEW与虚拟仪器 (VI)

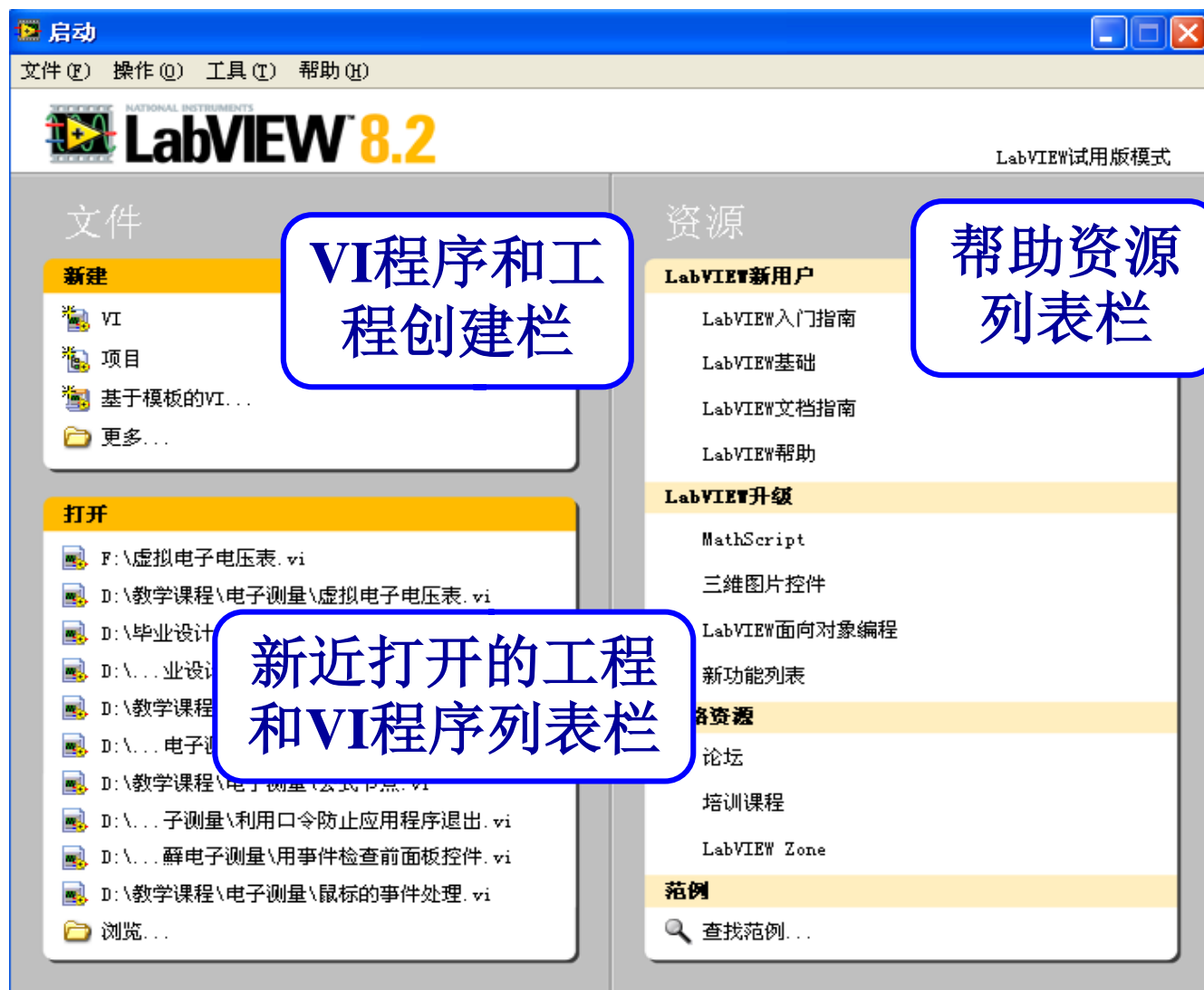
Laboratory **V**irtual **I**nstrument
Engineering **W**orkbench

图形化编程语言

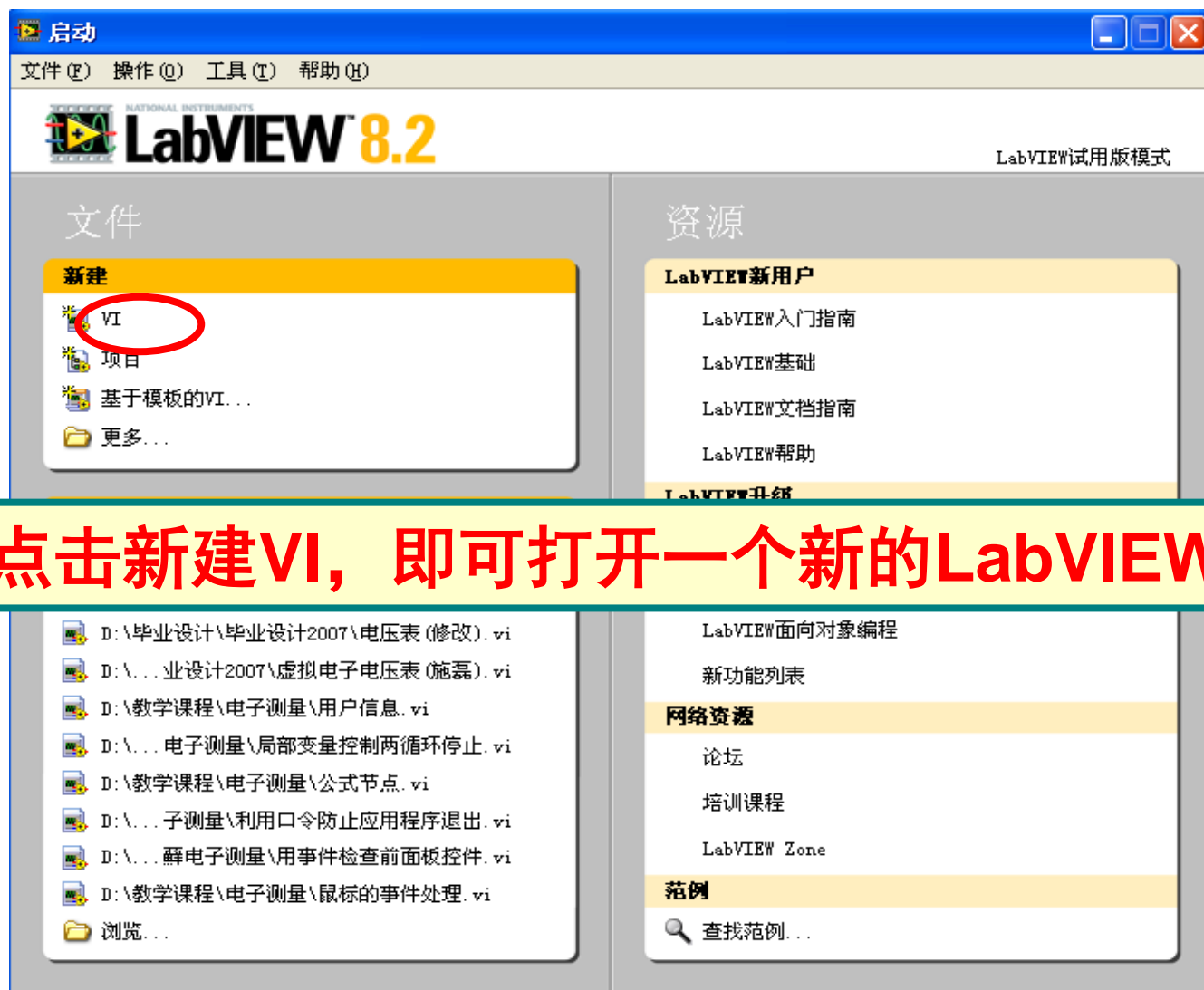
“G” 语言

LabVIEW程序称为虚拟仪器(**Virtual instrument**) 或VI, 扩展名默认为*.vi。

2. LabVIEW 启动界面



2. LabVIEW 启动界面



点击新建VI，即可打开一个新的LabVIEW程序

3. VI的组成

前面板 (front panel)

程序框图 (block diagram)

图标/连接端子(icon/connector)



前面板 窗口

4. 3种选板

控件选板（为前面板添加控件）

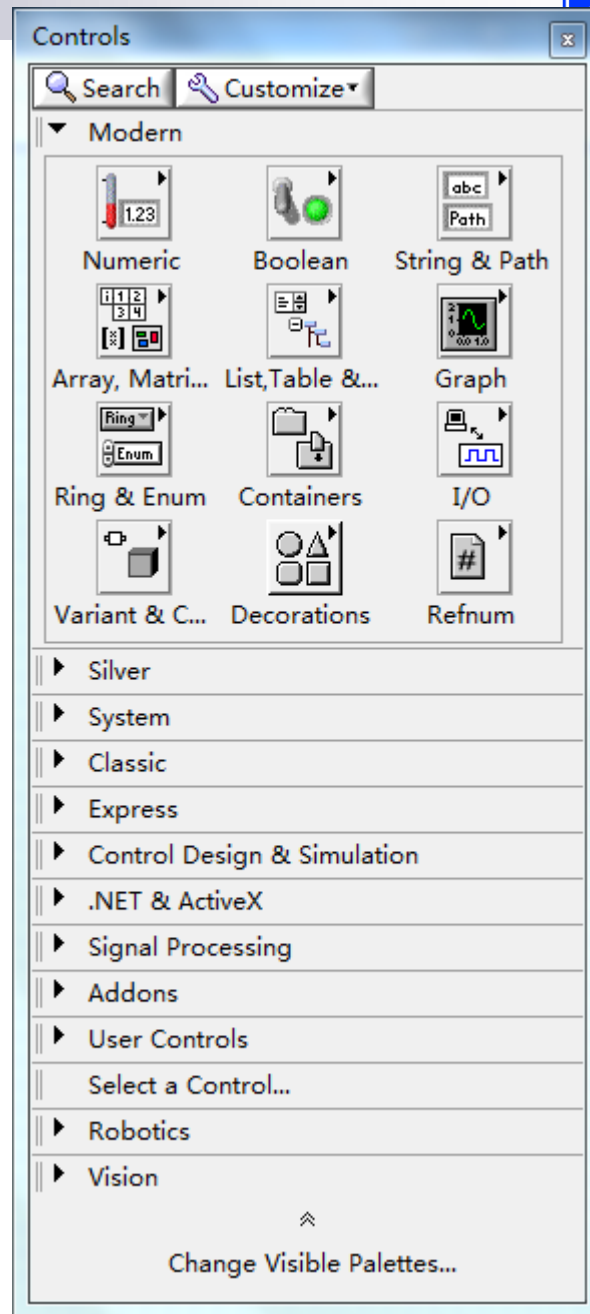
函数选板（在程序框图中添加函数或数据等）

工具选板（选择各种编辑工具，前面板和后面板都要用到）

(1) 控件选板

在前面板显示，它包含创建前面板时可用的全部对象。

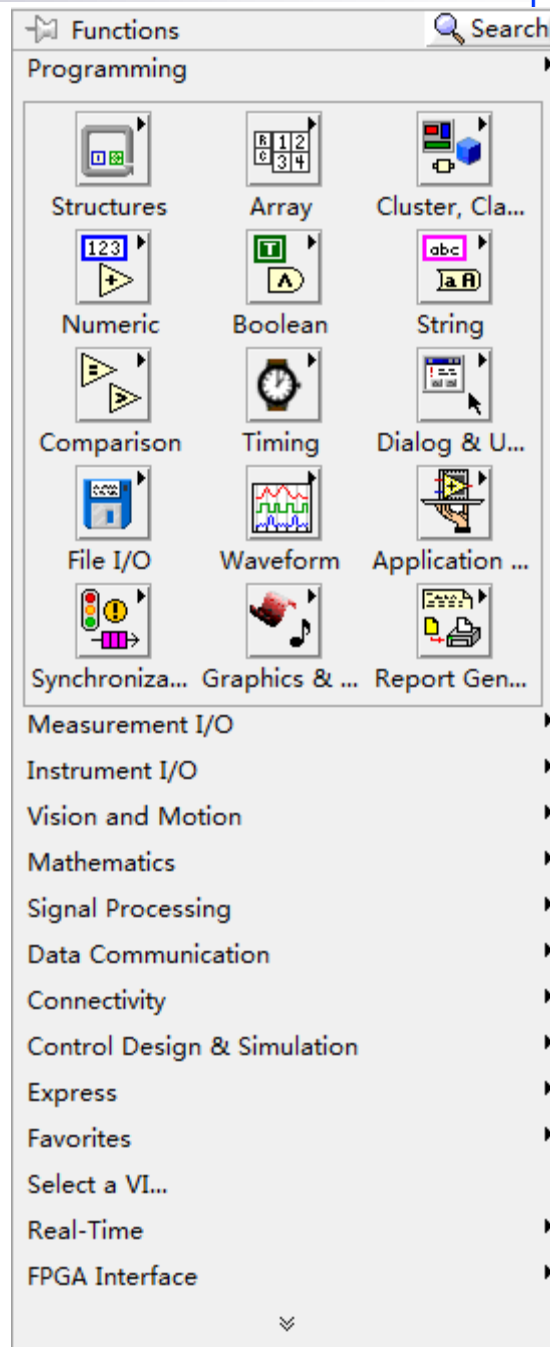
选择主菜单View->Controls
Palette选项或右击前面板空白
处就可以显示控件选板。



(2) 函数选板

只能在编辑程序框图时使用。创建框图程序常用的VI和函数对象都包含在该选板中。

选择View→Functions Palette
或右击框图面板空白处就可以显示函数选板。



(3) 工具选板



在前面板和程序框图中都可以使用工具选板，使用其中不同的工具可以操作、编辑或修饰前面板和程序框图中选定的对象，也可以用来调试程序等。

可以选择View->Tools Palette选项来显示工具选板。

(3) 工具选板



: 自动选择工具



: 操作工具



: 定位/调整大小/选择工具



: 编辑文本工具



: 连线工具



: 对象快捷菜单工具



: 滚动窗口



: 设置/清除断点工具



: 探针工具



: 颜色复制工具



: 着色工具

1.2 LabVIEW联机帮助

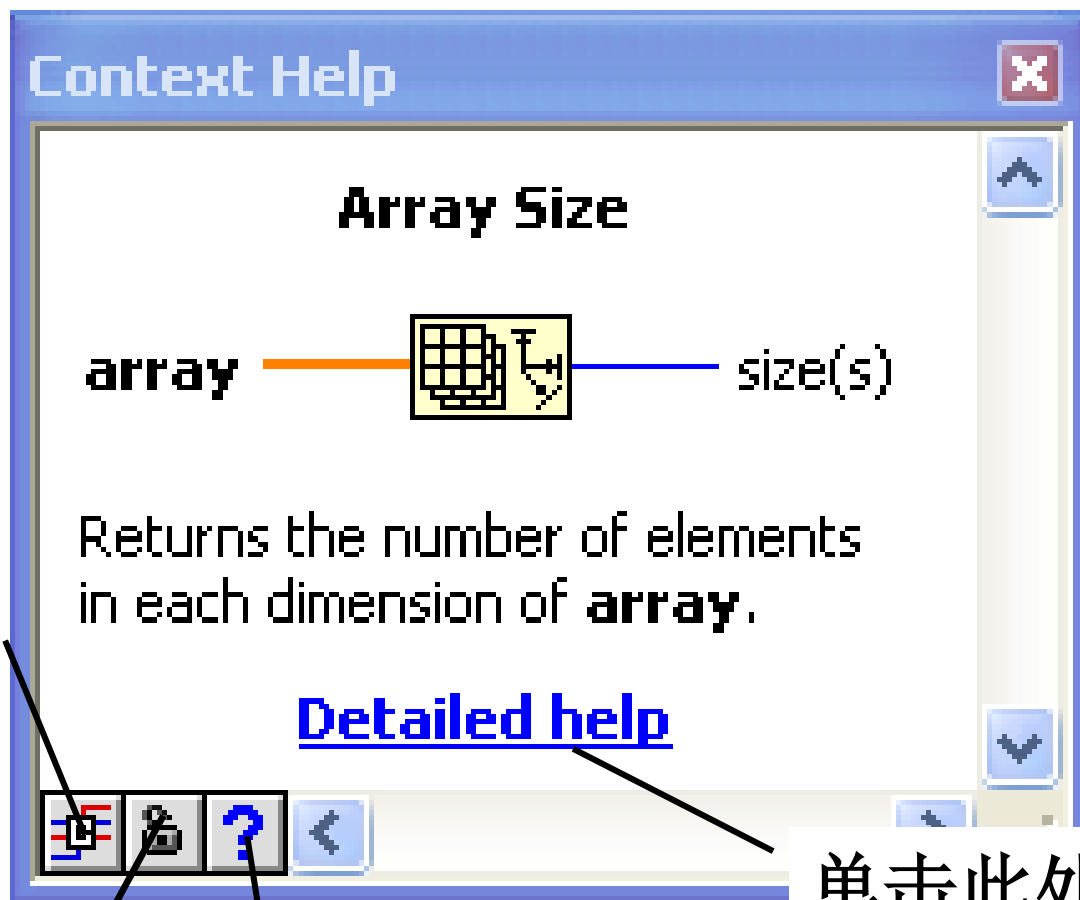
有效地利用帮助信息是快速掌握LabVIEW的一条捷径。

- ▶ 实时上下文帮助 (Show Context Help)
- ▶ 联机帮助
- ▶ LabVIEW范例查找器 (Find Examples)
- ▶ 网络资源 (Web Resources)

1. 实时上下文帮助窗口

选择菜单栏中Help->Show Context Help 选项或按下Ctrl+H，就会弹出Context Help 窗口。

当鼠标移到某个对象或函数上时，上下文帮助窗口就会显示相应的帮助信息。



显示VI
路径

锁定上下文
相关帮助

更多帮助
信息

单击此处访问更
详细的联机帮助

2. 联机帮助文档

当单击Context Help窗口中Detailed help会弹出相应的完整的帮助信息, 包含了LabVIEW全部的帮助信息。

也可以选择主菜单Help->Search the LabVIEW Help 选项打开它。

LabVIEW Help

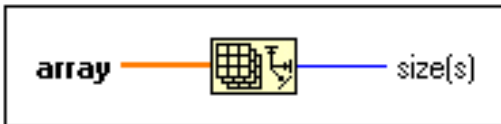
Hide Locate Back Forward Options

Contents Index Search Favorites

- LabVIEW Help
- Finding Example VIs
- Glossary
- LabVIEW 8.2 Features and Changes
- Activating LabVIEW Software
- Using Help
- LabVIEW Documentation Resources
- Getting Started
- Fundamentals
- VI and Function Reference
 - Programming VIs and Functions
 - Application Control VIs and Functions
 - Array Functions
 - Array Constant
 - Array Max & Min
 - Array Size
 - Array Subset

Array Size

Returns the number of elements in each dimension of **array**. The connector pane displays the default data types for this polymorphic function.



☐ Place on the block diagram ☒ Find on the **Functions** palette

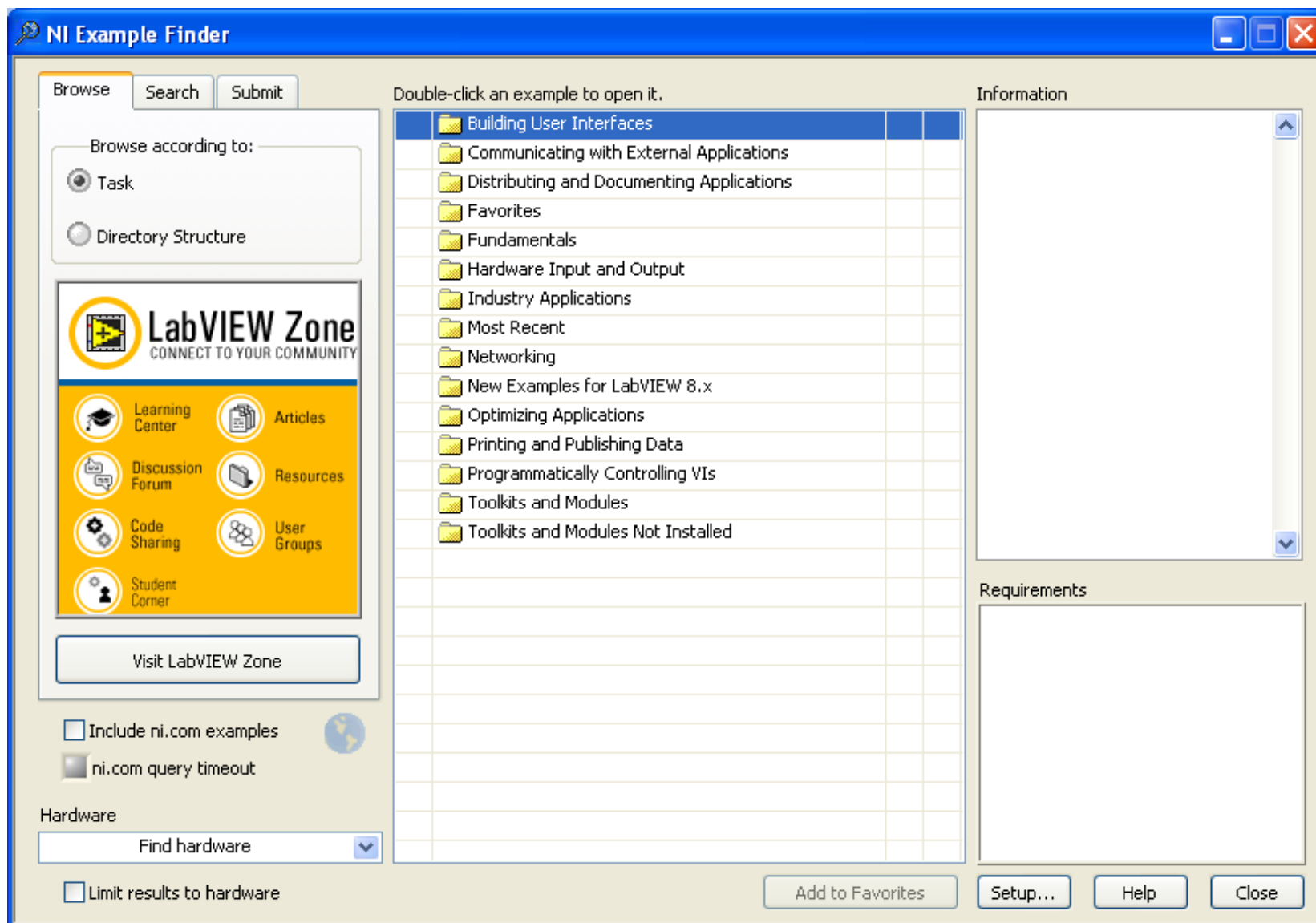
[DBL] **array** can be an n-dimensional array of any type.

[I32] **size(s)** is a 32-bit integer if **array** is one-dimensional (1D). If **array** is multidimensional, the returned value is a 1D array in which each element is a 32-bit integer.

3. 范例查找器

LabVIEW提供了大量的范例，这些范例几乎包含了LabVIEW所有功能的应用实例，并提供了大量的综合应用实例。

在菜单栏中选择Help->Find Examples选项可以打开范例查找器。

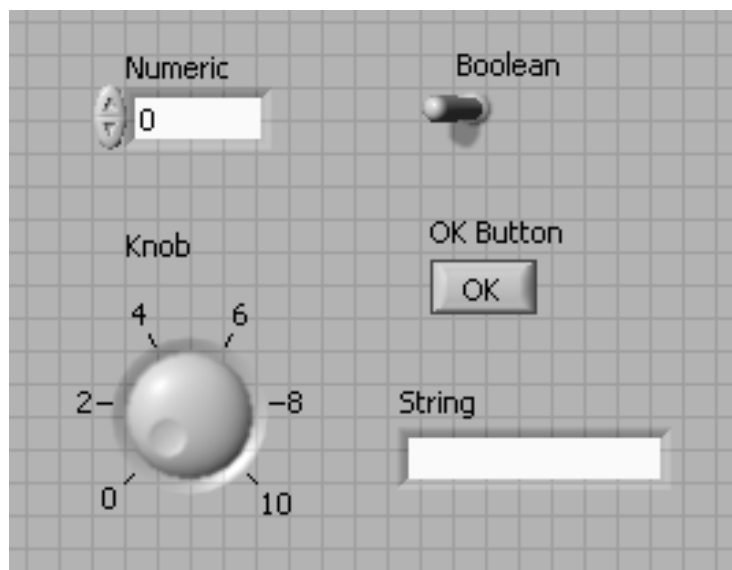


1.3 编辑前面板

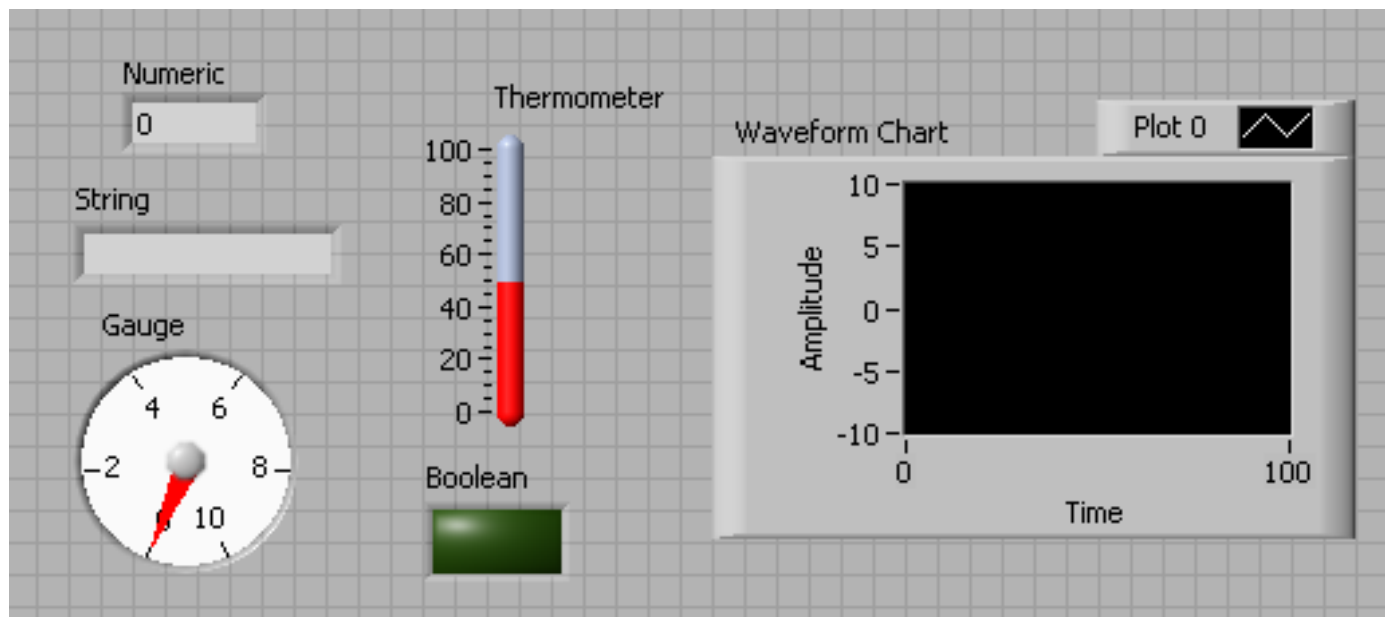
LabVIEW中的前面板是图形化的人机界面，利用控件选项板提供的各种控件可以所见即所得地编辑丰富多彩的人机界面。

1. 输入控件 (Controls) 和显示控件 (Indicator)

利用输入控件可以输入相应的数据，例如数字、布尔量、字符串和文件路径等。



显示控件用来显示数据。显示控件有数字、温度计、LED指示灯、文本、波形图等。



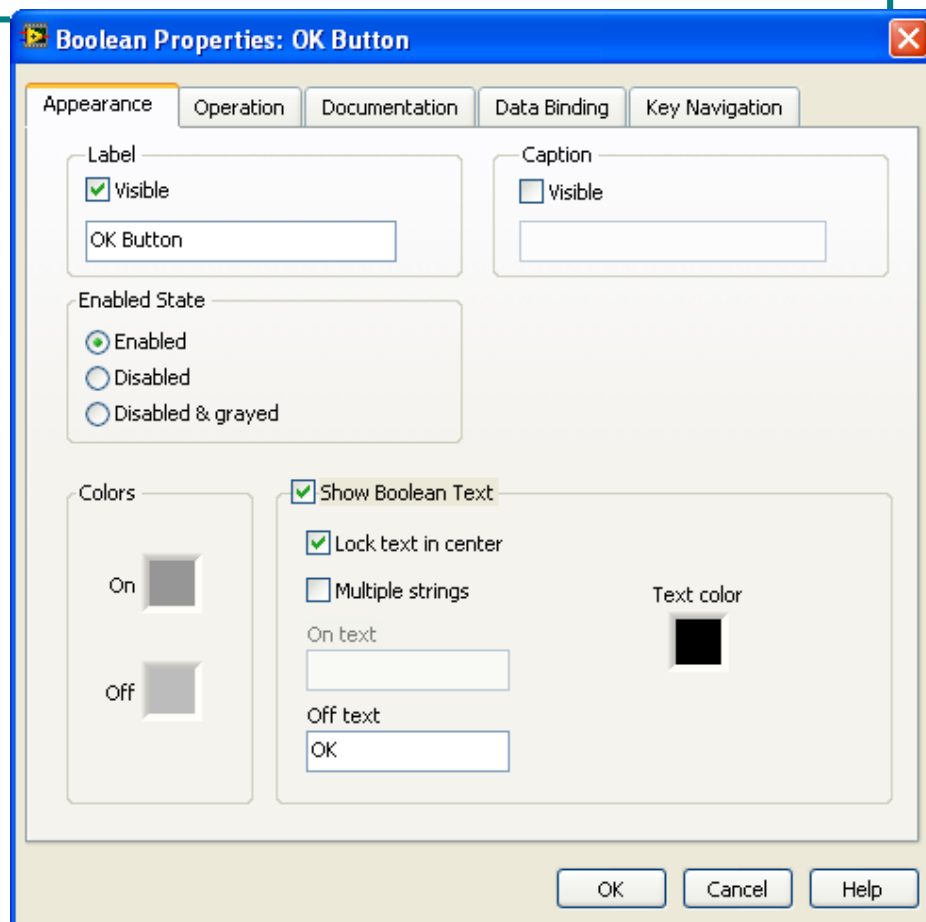
前面板中的一些控件既可以作为输入控件也可以做作为显示控件。

右击控件，选择Change to Indicator或Change to Control可以进行输入控件与显示控件之间的切换。

2. 控件属性

每个控件都有自己的属性，如控件的颜色、最大最小值、显示精度和方式等。

许多属性可以根据不同的需要进行编辑。右击前面板任何一个控件选择Properties选项就可以弹出该控件的属性配置窗口。

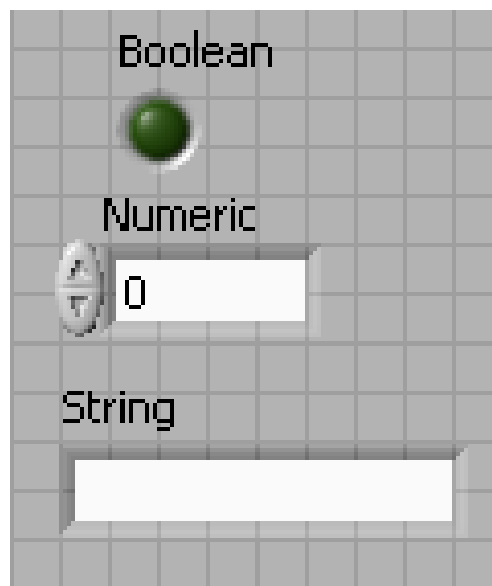


1.4 编辑程序框图

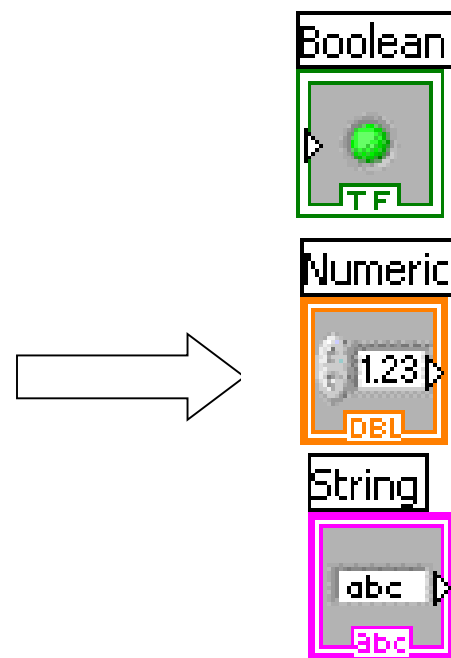
程序框图是图形化源代码的集合，这种图形化的编程语言也称为G语言。

1. 程序框图中的控件对象

程序框图中的控件对象实际上是前面板相应控件的接线端。



前面板控件

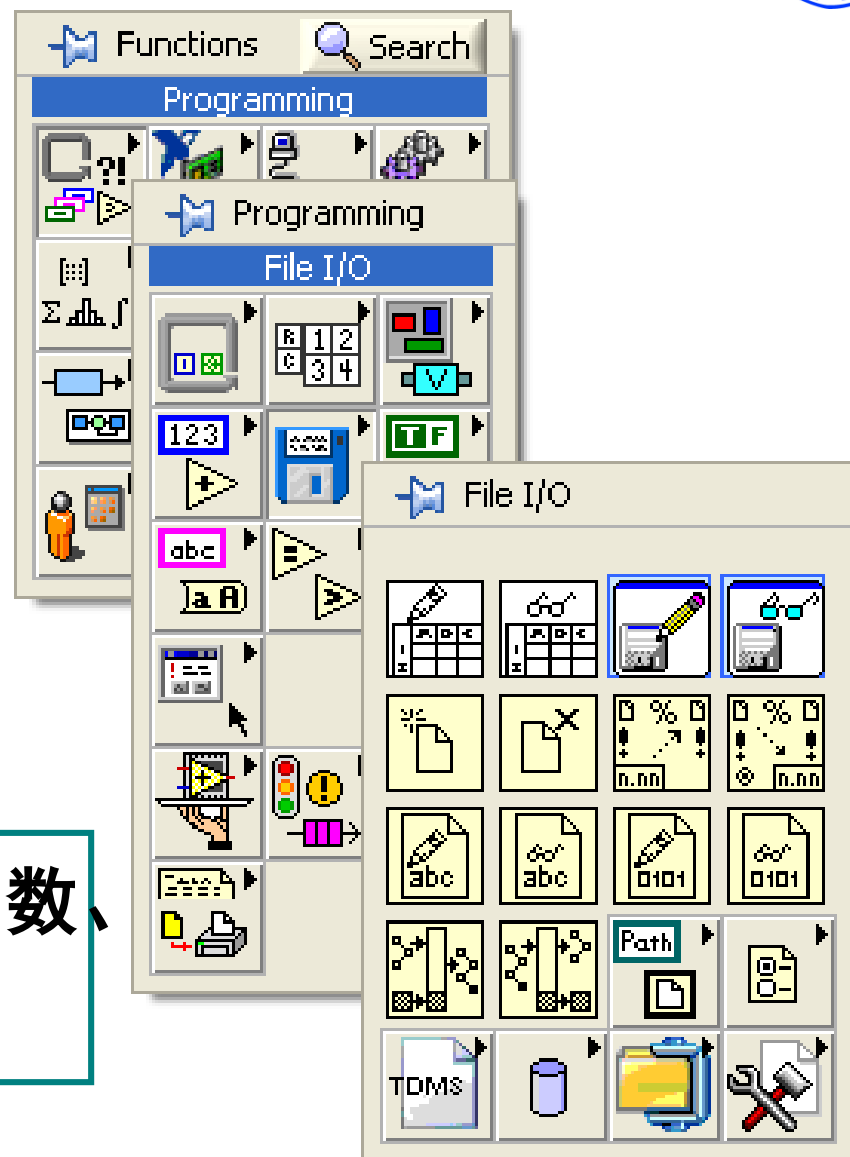


程序框图中
对应的接线端

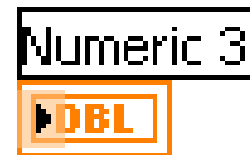
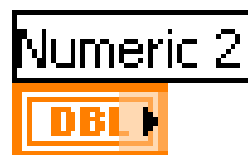
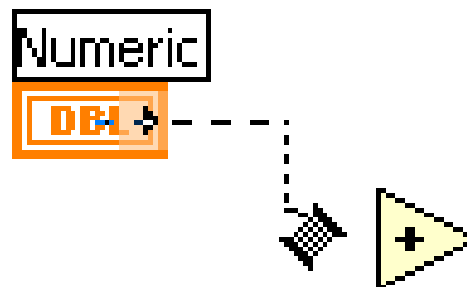
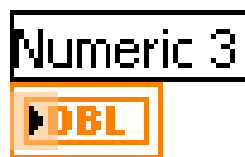
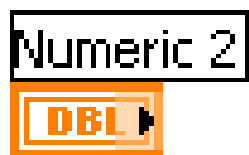
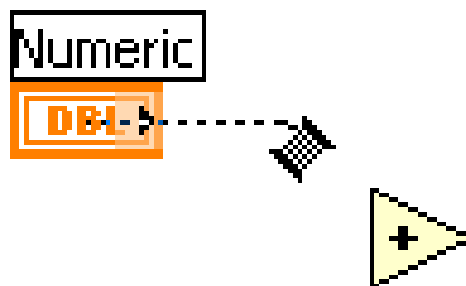
2. 程序框图节点

带有输入和输出接线端的对象，类似文本编程语言中的语句、运算符、函数和子程序。

LabVIEW中的节点主要包括函数、结构、Express VI、子VI等。



3. 对象连线



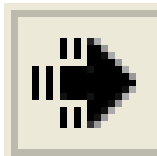
1.5 VI的运行和调试

1. 运行VI

单击前面板或程序框图工具栏中的运行按钮。

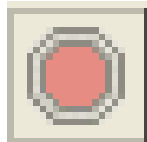


当VI正在运行时，运行按钮状态变为



2. 停止VI运行

当程序运行时，停止按钮由编辑时的状态，



变为可用状态



注意：

如果调试程序时，使程序无意中进入死循环或无法退出时，这个按钮可以**强行结束程序运行**。

3. 调试VI

1) 查找VI不可执行的原因

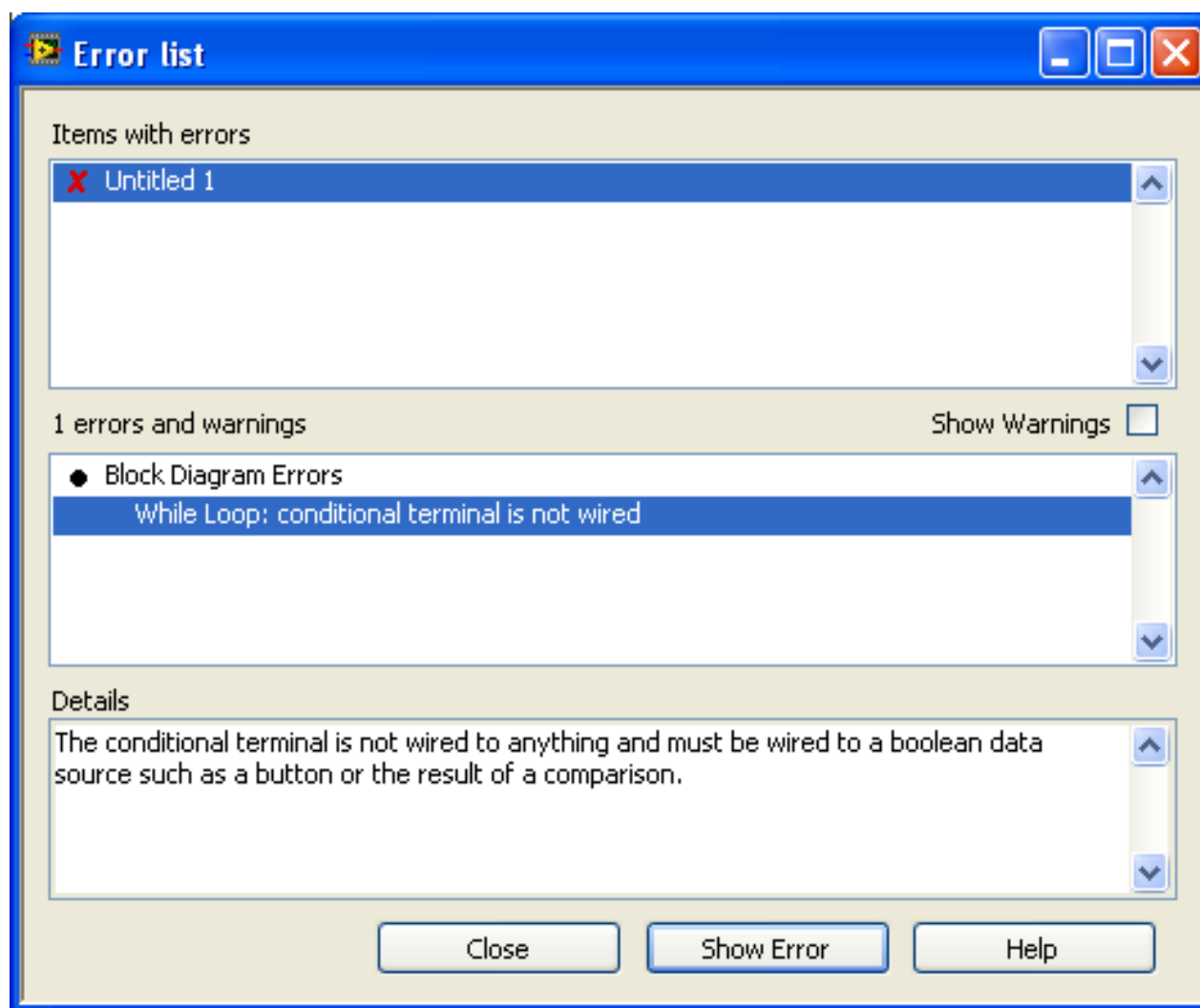
如果在一个VI程序中存在错误时，VI是不能运行的。这时，工具栏中的运行按钮由



变为断裂状态



单击此按钮就会弹出错误列表对话框。



2) VI调试方法

①高亮执行



可以逼真地显示数据的流动过程。

再次单击此按钮，程序又恢复正常运行。

注意：

使用高亮执行方式，将明显降低程序的执行速度。

2) VI调试方法

① 高亮执行

② 单步执行

一个节点一个节点地执行。

③ 探针 

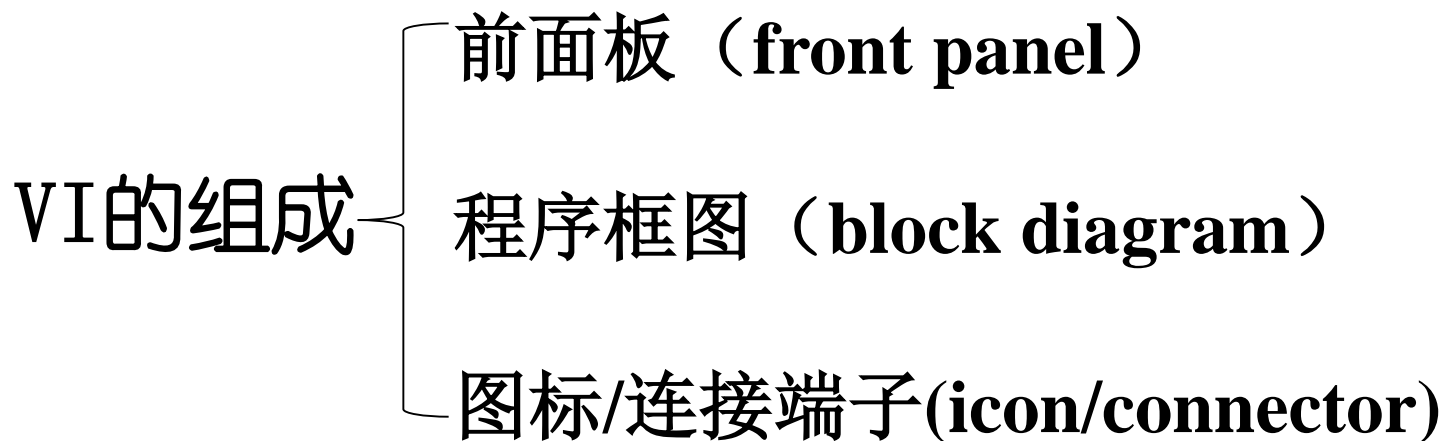
查看运行过程中数据流在该连线上的数据。

④ 断点 

程序运行到该处会暂停执行，再单击暂停按钮程序会继续运行到下一个断点处或直到VI运行结束。

1.6 VI子程序

- ✓ LabVIEW中的子VI (SubVI) 类似于文本编程语言中的函数。
- ✓ 通过子VI, 可以把程序分割为一个个小的模块来实现。

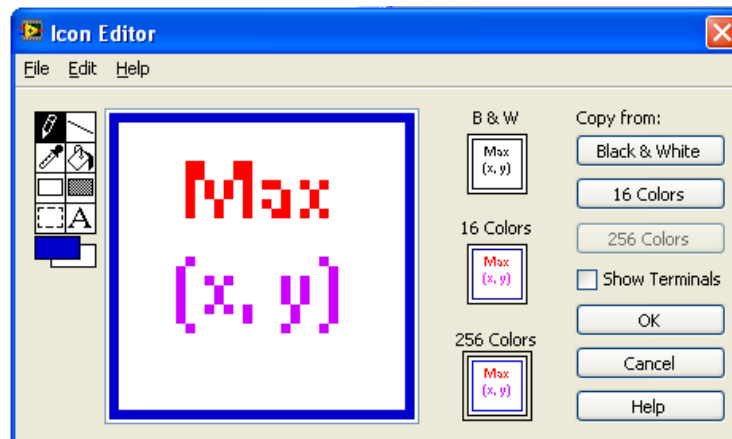


1. 创建子VI

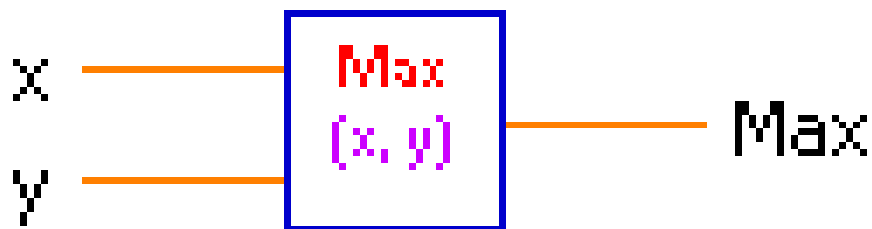
任何VI本身就可以作为子VI被其它VI调用，只是需要在普通VI的基础上多进行两步简单的操作而已：**定义连接端子和图标。**

第一步：新建一个Blank VI，编写其程序框图。

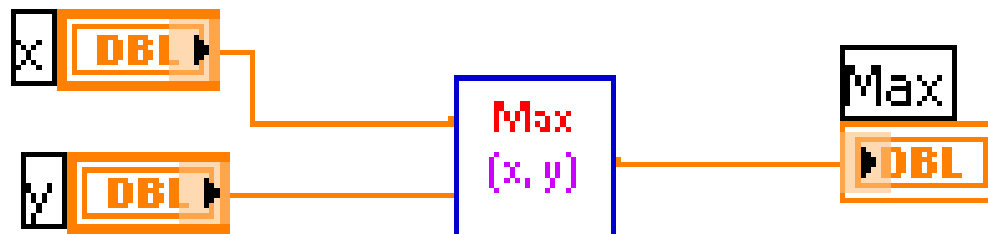
第二步：编辑VI图标。



第三步：建立连接端子。



第四步：保存该VI，将该VI当作子VI调用。

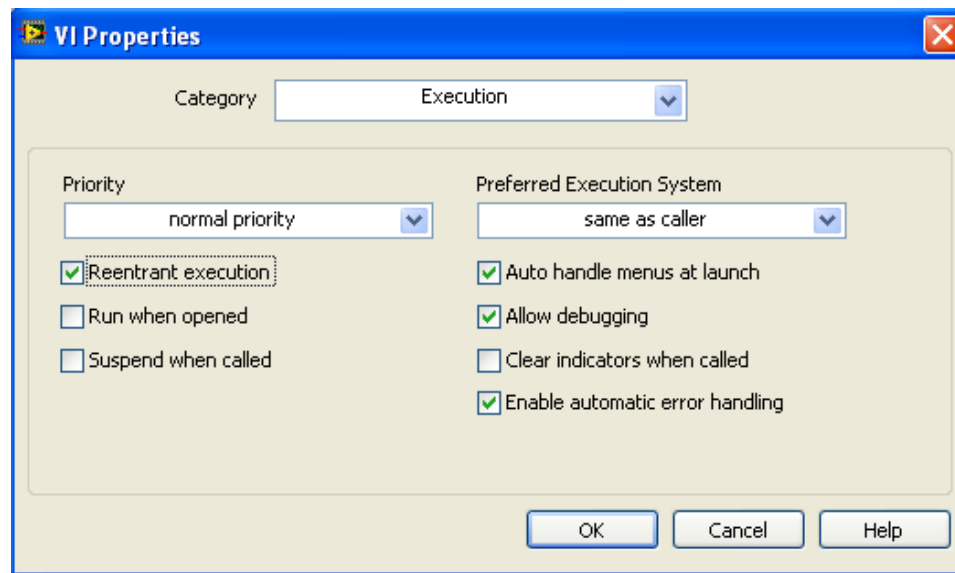


2. 定义子VI属性

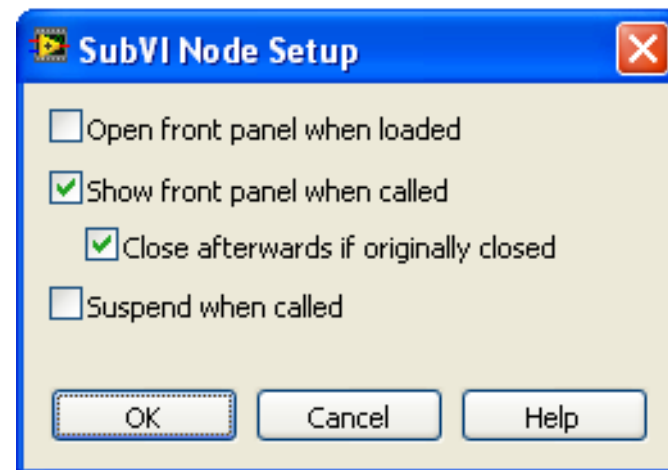
1) 可重入 (Reentrant) 子VI

在缺省情况下，如果有两处程序框图都调用同一个子VI，那么这两处程序框图则不能并行运行。

而在很多情况下，我们都希望不同的调用应该是相互独立的。这时候我们就需要把子VI设为**可重入子VI**。



2) 设置子VI调用属性



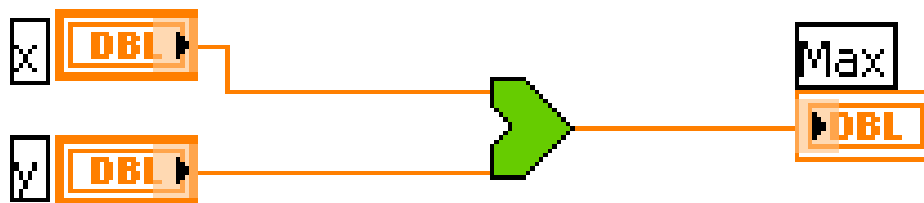
利用显示子VI前面板来实现登录对话框。

3) 自定义子VI图标形状

第一步： 在编辑VI图标界面选择Edit->Clear清除图标。

第二步： 在256 Colors下画一个**封闭**的图形。为了方便与端子对应，可选择Show Terminals复选框显示连接端子。

第三步： 将画好的256色图标分别复制到16 Colors和B&W下，确保三种显示模式所显示的图标形状一致。

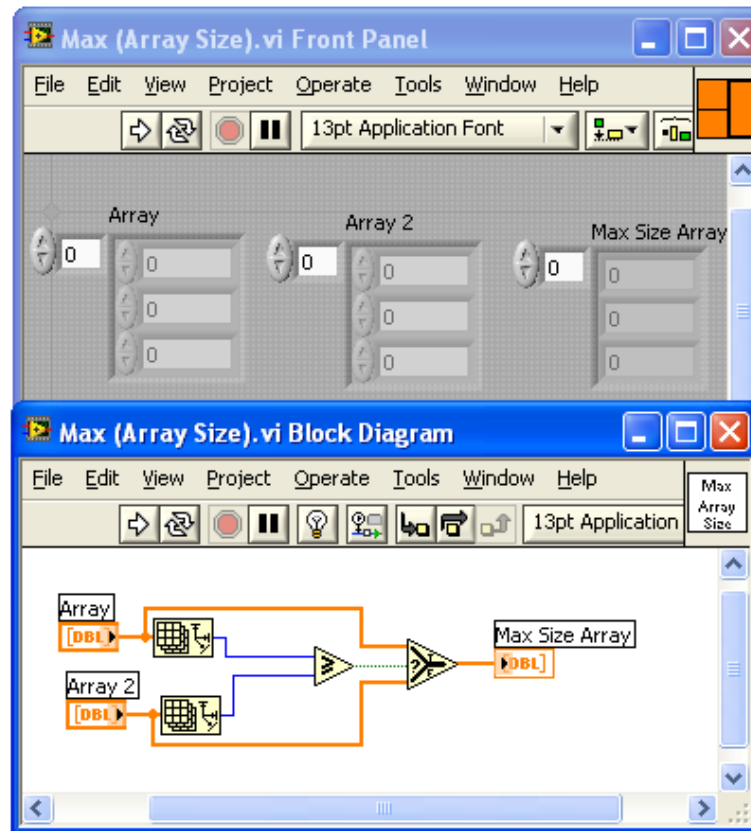
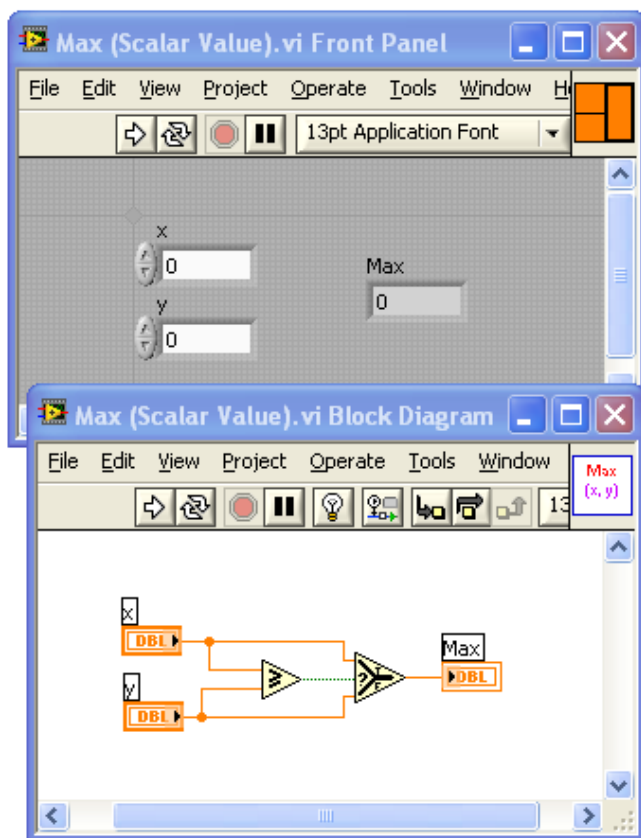


3. 多态 (Polymorphic) VI

LabVIEW中的多态与C++中的多态概念类似，即函数可以根据输入数据的类型自动选择执行内容。

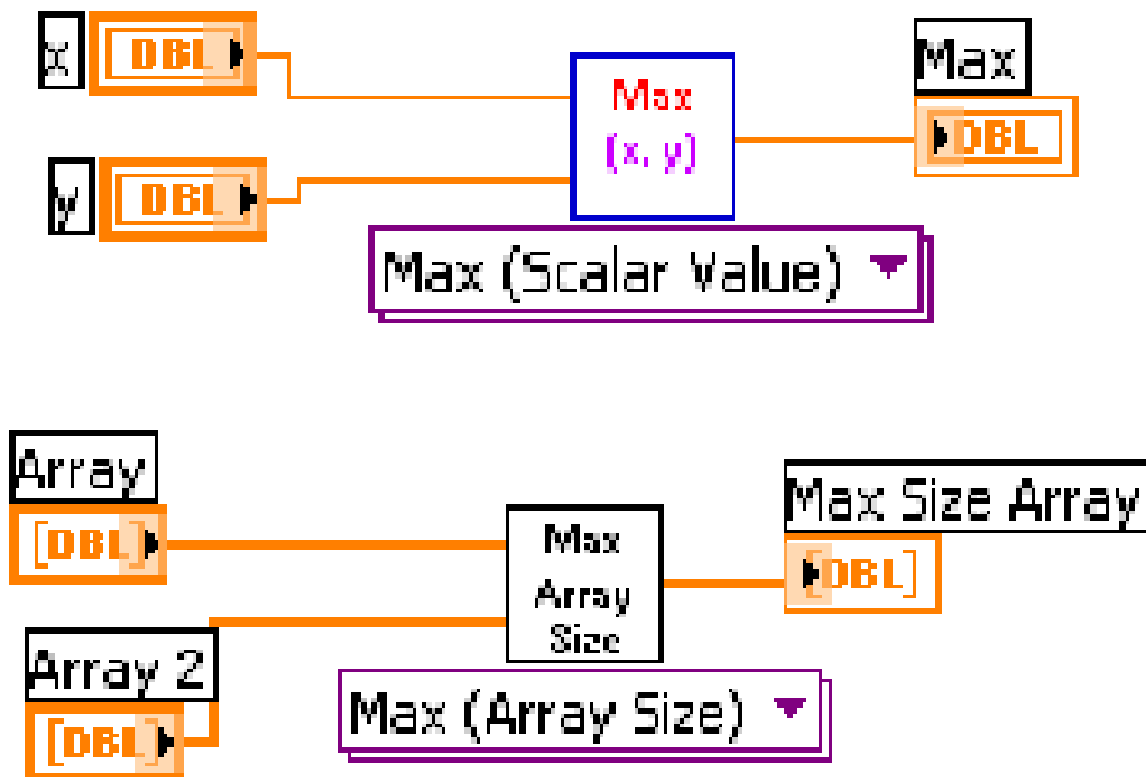
在LabVIEW中你也可以创建自己的多态VI。它实际上是多个VI的集合，这些VI具有相同的端子模式。

✓ 创建多态VI举例



多态VI对应的两个实例VI

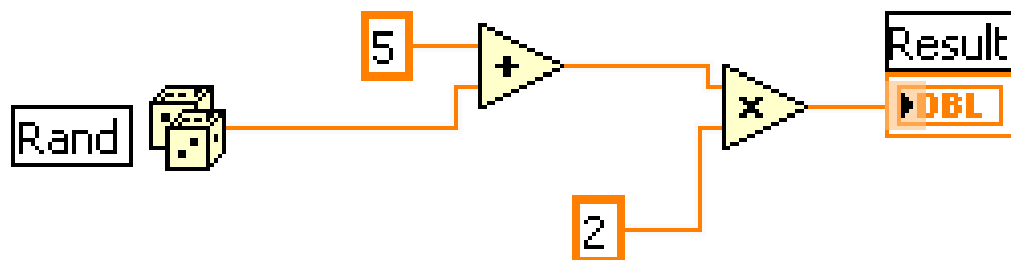
✓ 调用多态子VI



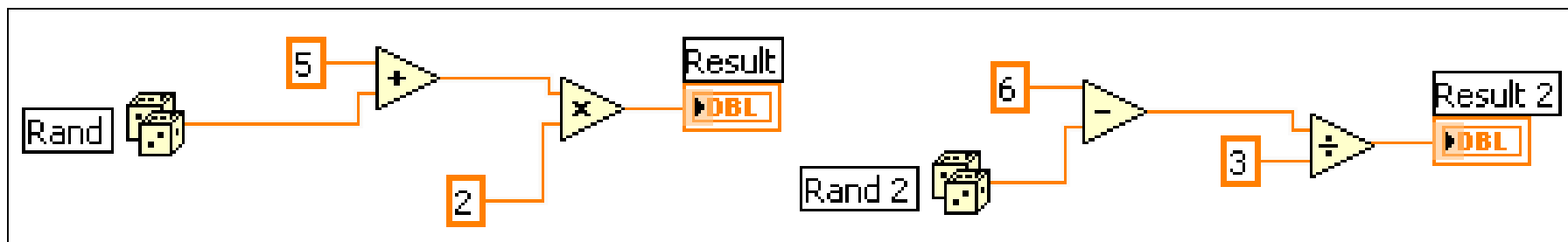
第二章 程序结构

- 顺序结构
- While循环
- For循环
- Case结构
- Event结构

LabVIEW程序的执行顺序



数据流式的编程方法

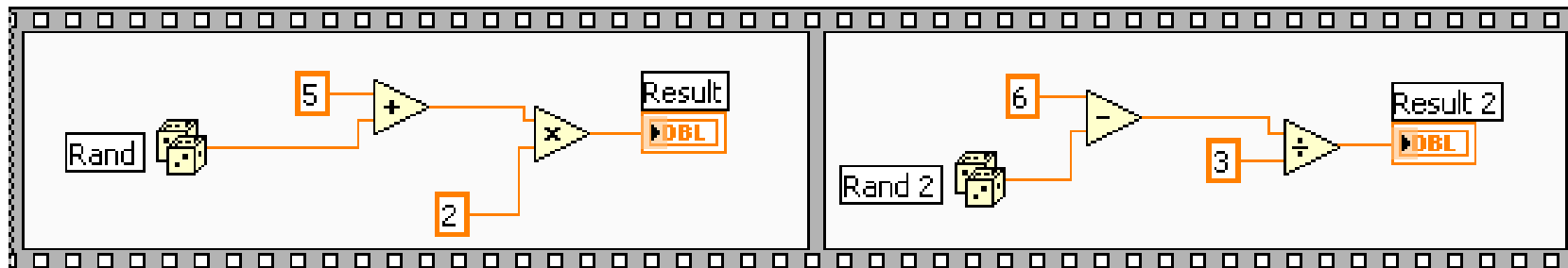


多段代码同时执行

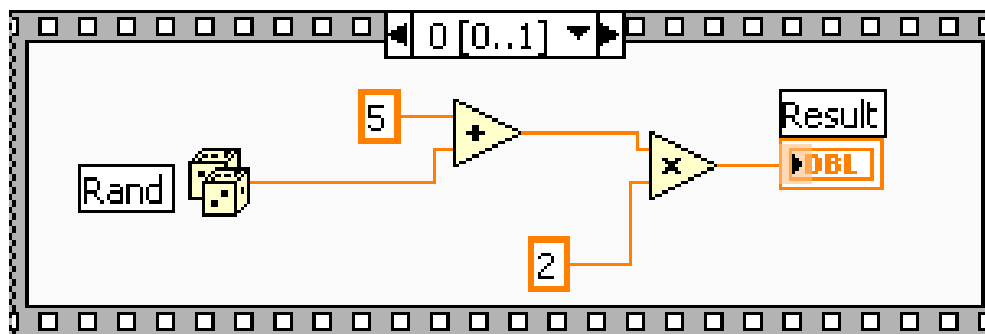
2.1 顺序结构 (Sequence Structure)

1. 种类

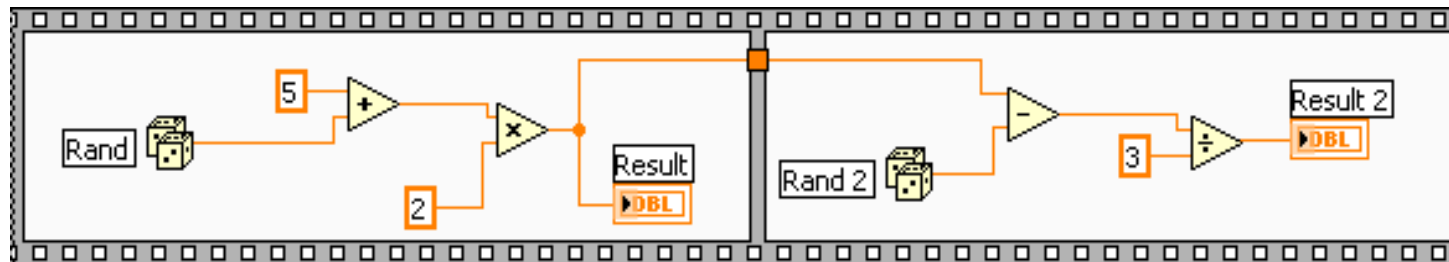
■ Flat Sequence Structure



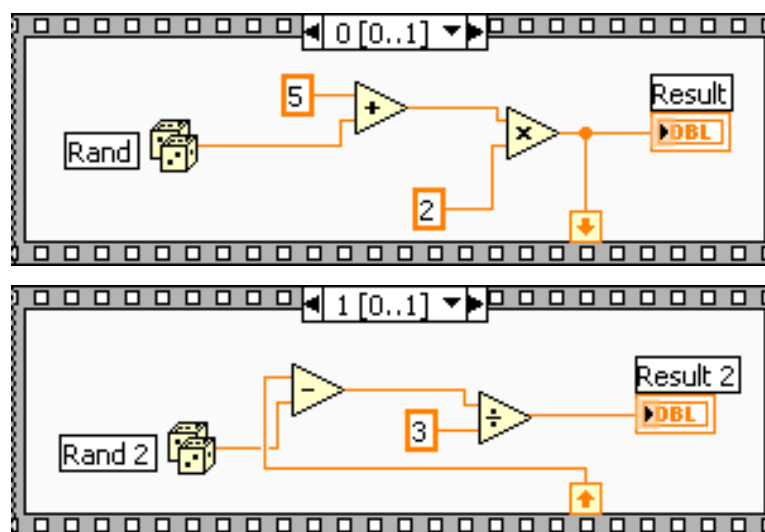
■ Stacked Sequence Structure



2. 在Flat Sequence Structure的Frame间传递数据



3. 在Stacked Sequence Structure的Frame间传递数据

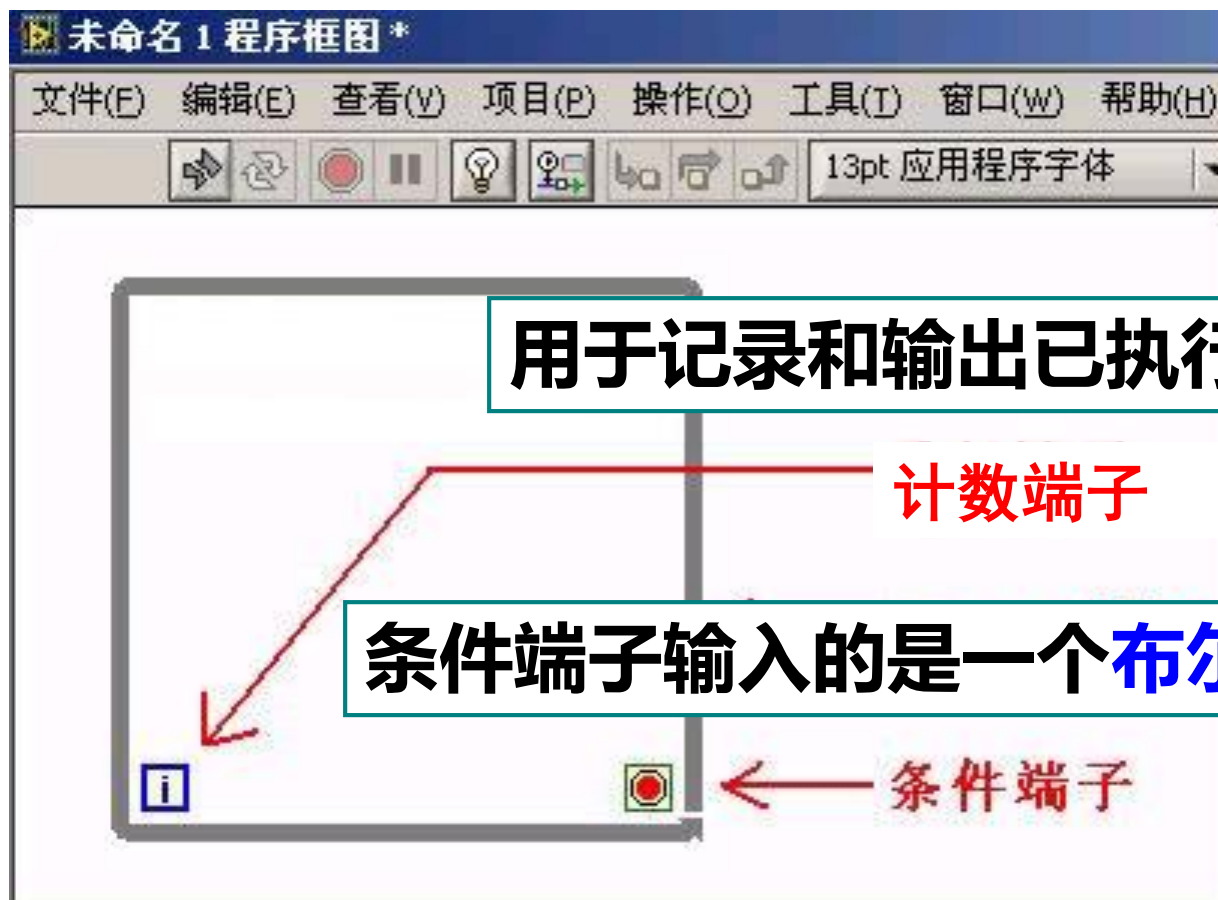


2.2 While循环

While循环用于将某段程序循环执行直到满足某一条件。

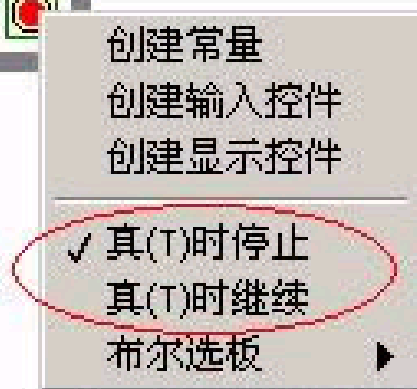
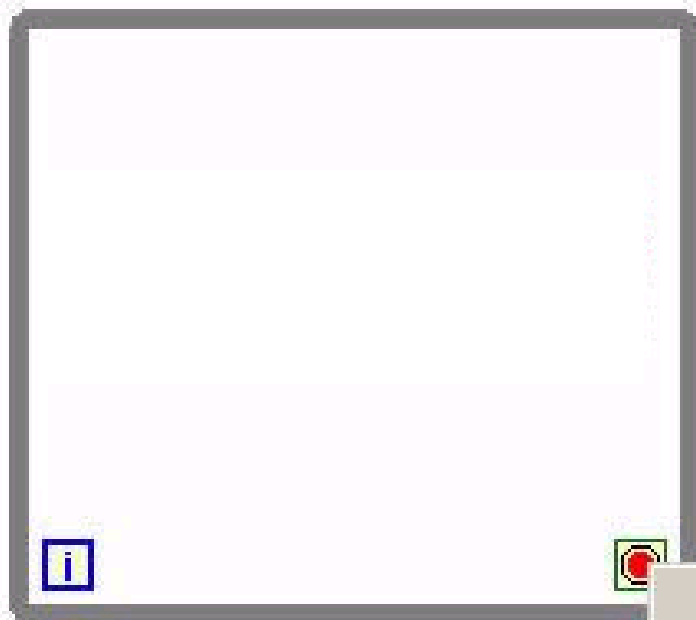
位于编程→结构→While循环。

While循环有一个条件端子和一个重复端子。



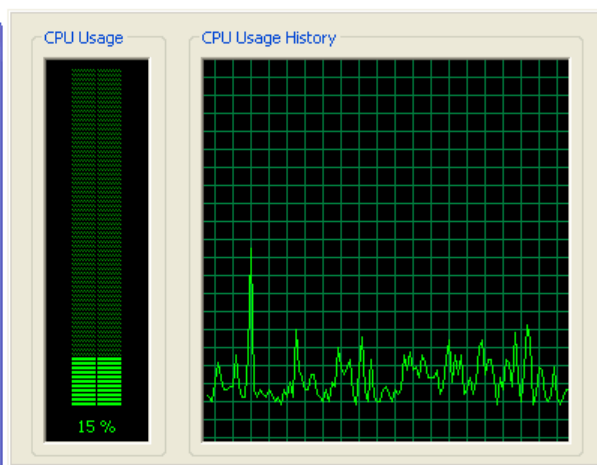
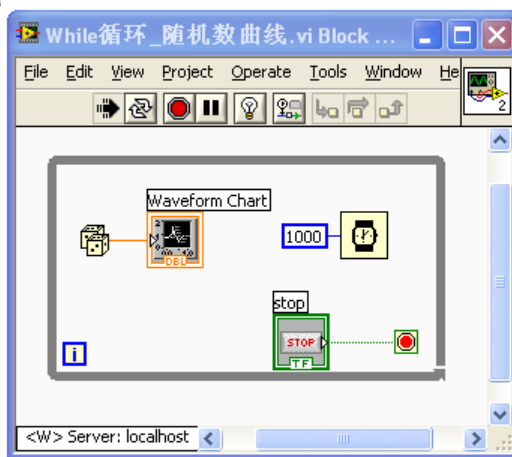
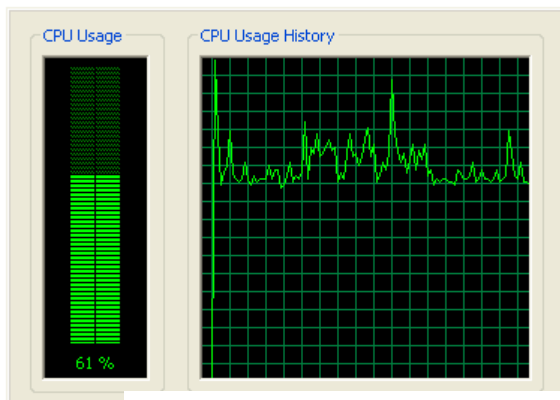
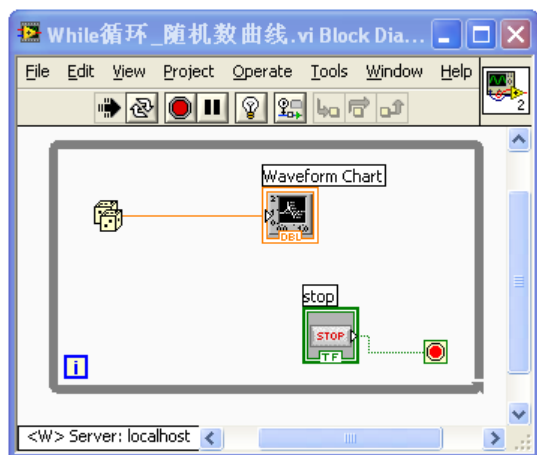
While循环一直执行到连接条件端子上的布尔值满足条件为止。

在条件端子上单击
右键即可进行更改。

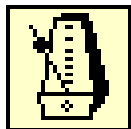


添加定时器

控制循环的频率或定时。处理器也需要定时信息完成其它任务。

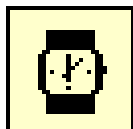


◆ 等待函数



“等待下一个整数倍毫秒” 函数

用于同步各操作。代码执行时间应小于该函数指定的时间。

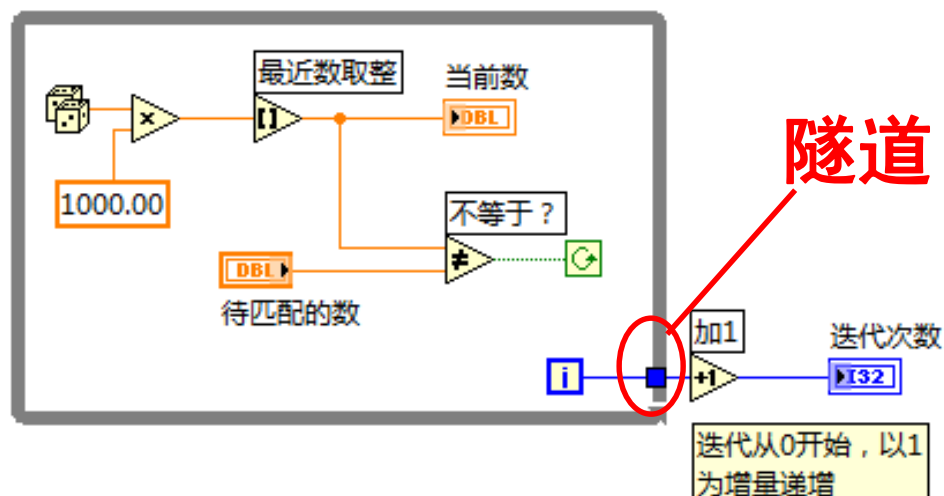


“等待（毫秒） 函数

保证循环的执行速率至少是预先输入的指定值。

◆ 结构隧道

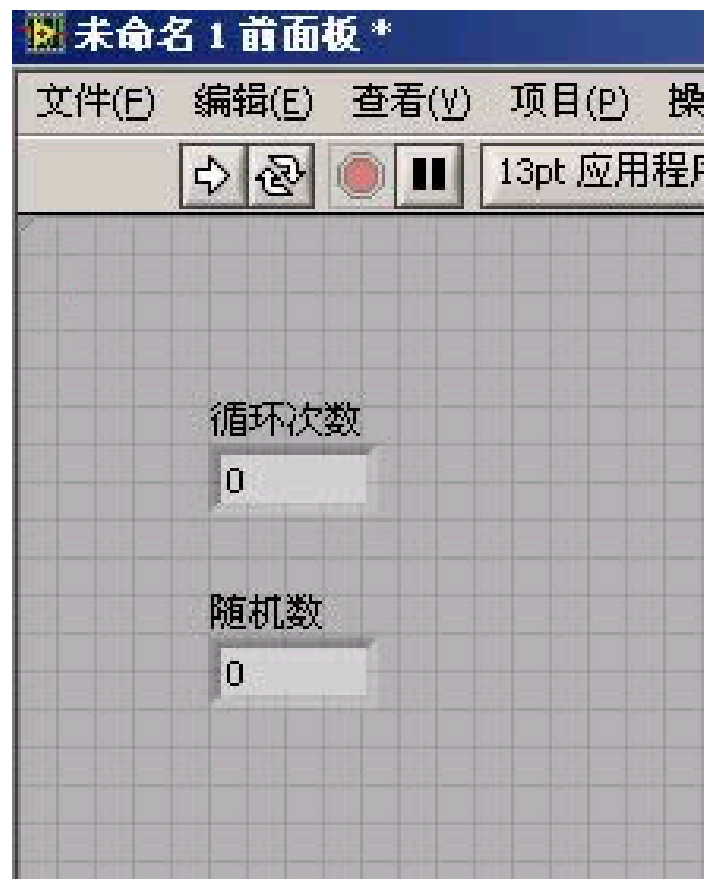
隧道用于接收和输出结构中的数据。



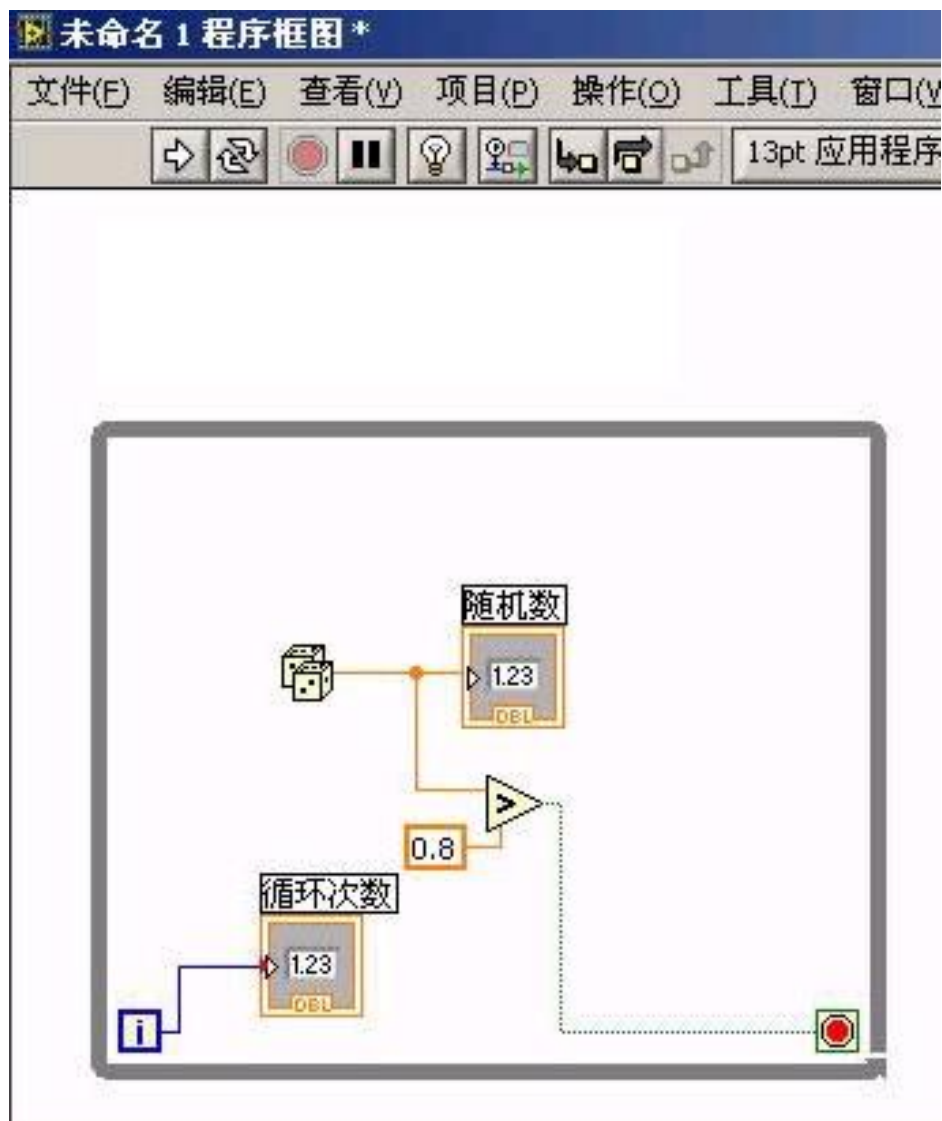
数据输入循环时，只有在数据到达隧道后循环才开始执行；数据输出时，直到循环停止执行后隧道中的值才输出循环。

例：利用While循环产生随机数，当产生的随机数大于0.8时，循环停止。

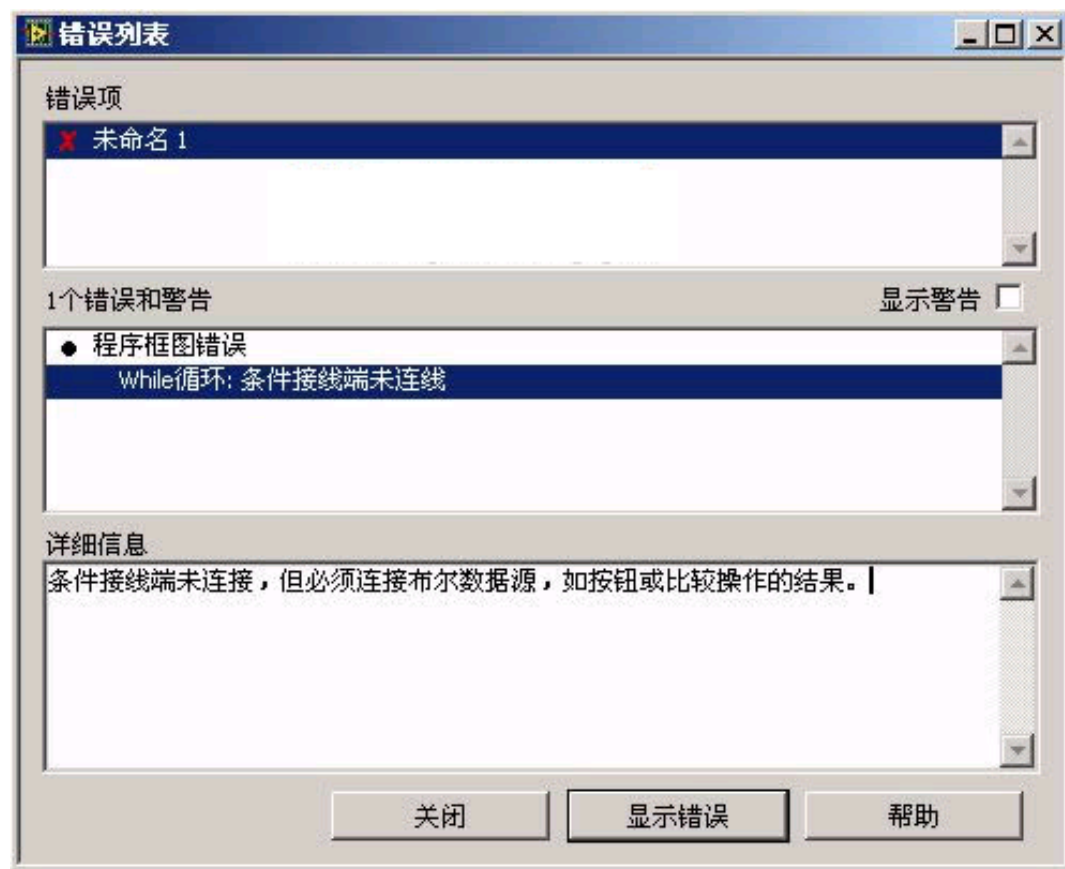
1. 打开前面板，新建两个数值显示控件，一个为循环次数，一个显示最后产生的那个大于0.8的随机数。



2. 切换到程序框图，放置一个While循环， ...

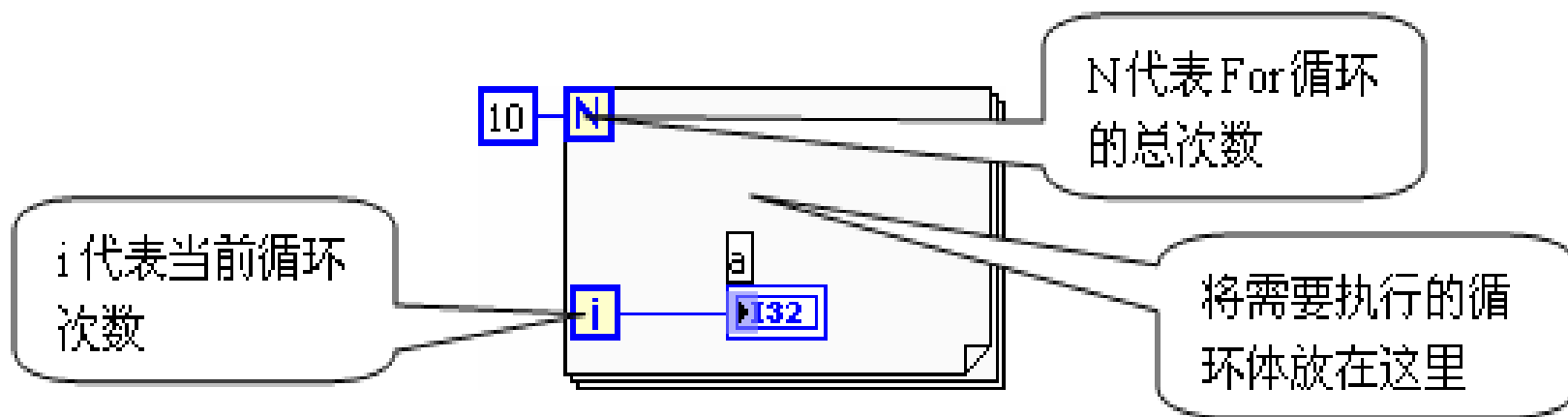


注意：While循环的条件端子一定要有连线，
否则程序无法运行！



2.3 For循环

For循环用于将某段程序循环执行指定的次数。



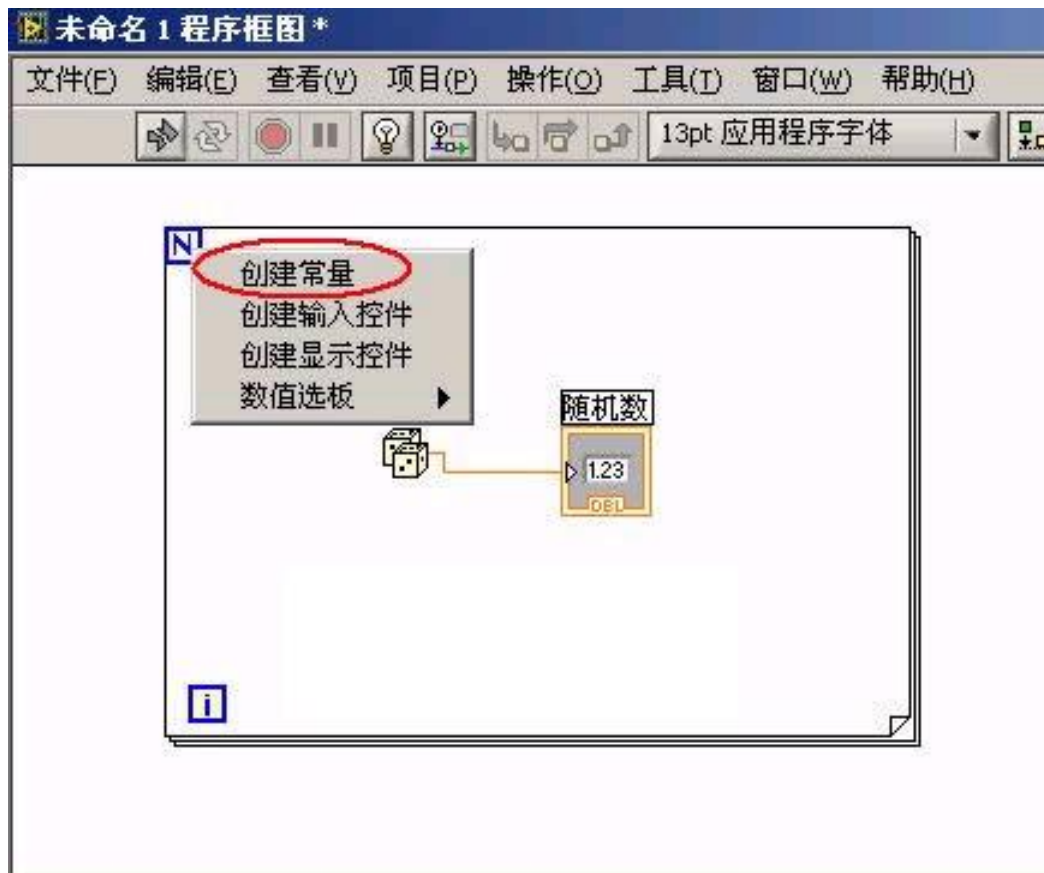
i: 计数从零开始。

例：产生10个随机数，并把最后一个显示出来。



1. 放置一个数值显示控件到前面板，并改名为随机数。
2. 放置For循环到程序框图。

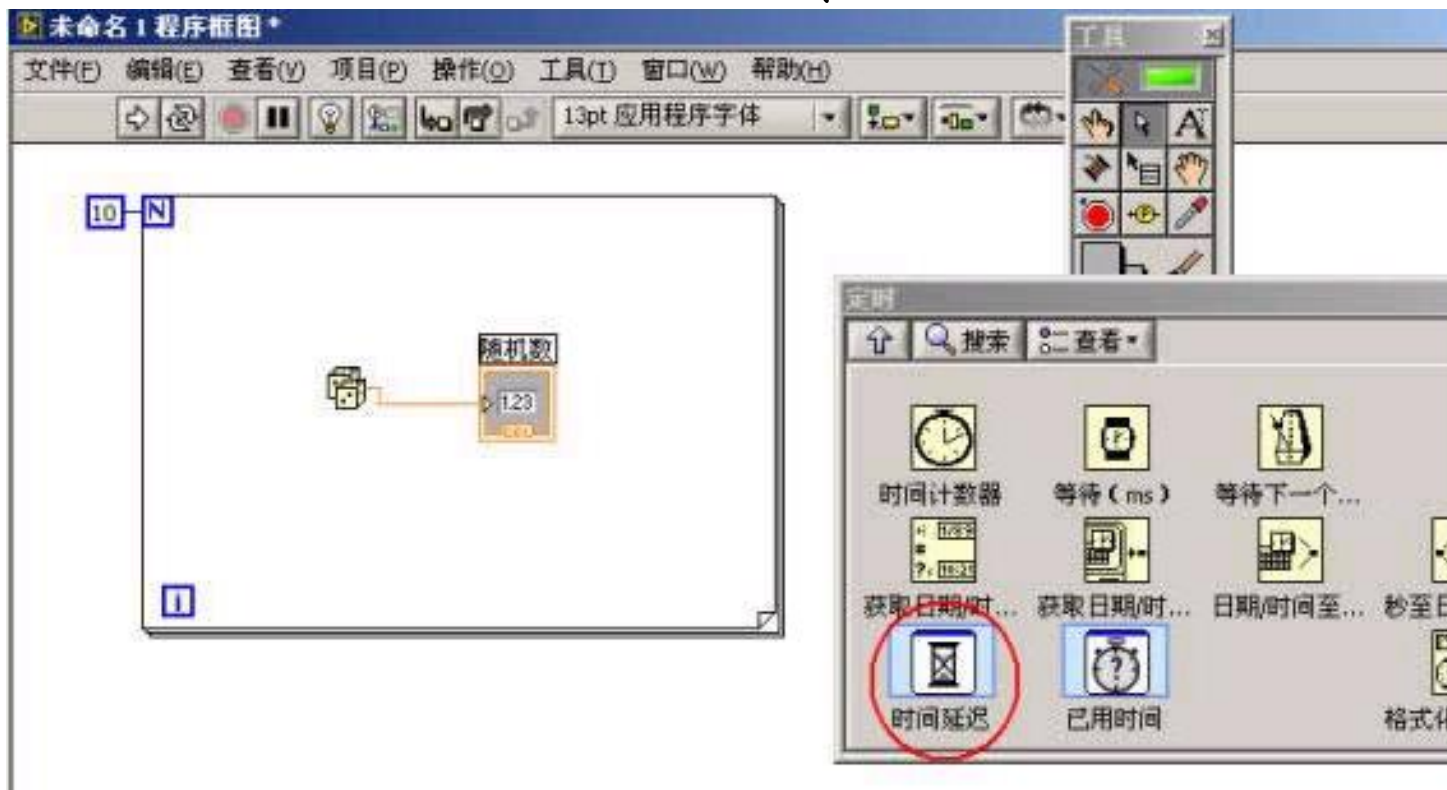
3. 放置随机数函数到程序框图，并连线。



4. 计数端子上单击右键，选择**创建常量**，并输入10。

5. 运行程序。

程序的运行的速度非常快，
无法一个一个地看清所产生的
10个随机数。



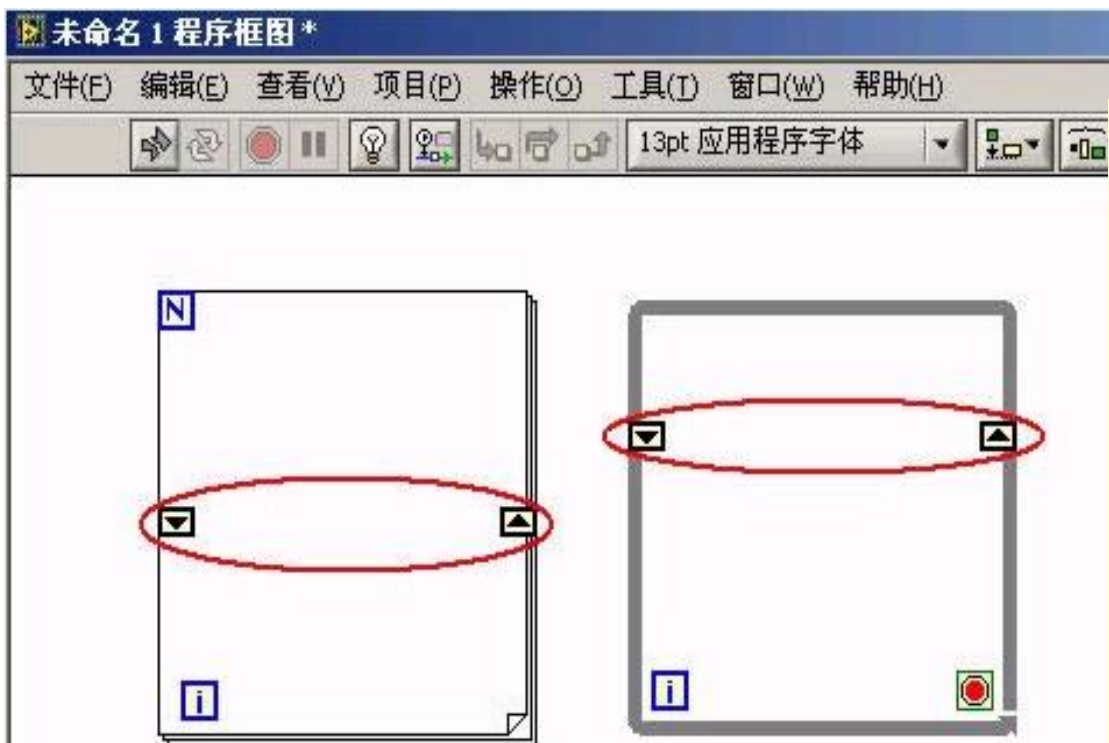
6. 添加时间延迟控件。位于函数→编程→定时→时间延迟。

7. 选择延迟的时间，即两次循环之间的时间间隔。将时间设置为1.000，点击确定，再运行程序。



◆ 移位寄存器 (Shift Register)

用于For循环或While循环中从一个迭代传输数据到下一个迭代。

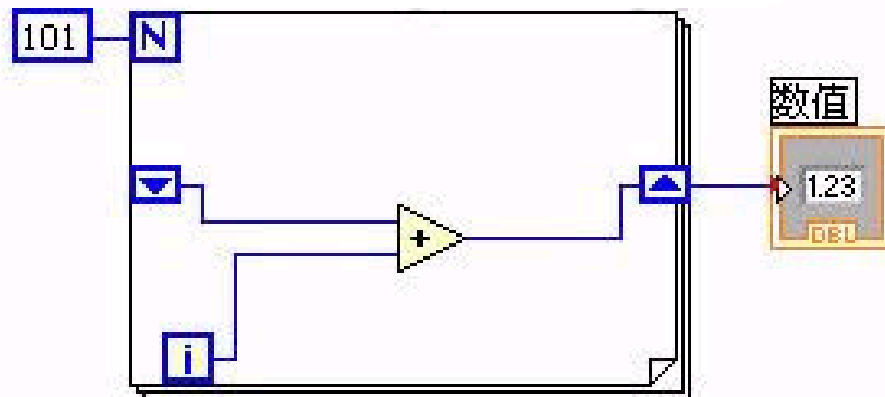


由循环垂直边框上一对**方向**相反的端子组成，在边框上**单击右键**，选择添加移位寄存器，就可进行添加。

右端子（带向上箭头的矩形）在每完成一次迭代后**存储数据**，移位寄存器将上次迭代的存储数据在下一次迭代开始时**移动到左端子**（具有向下箭头的矩形）上。



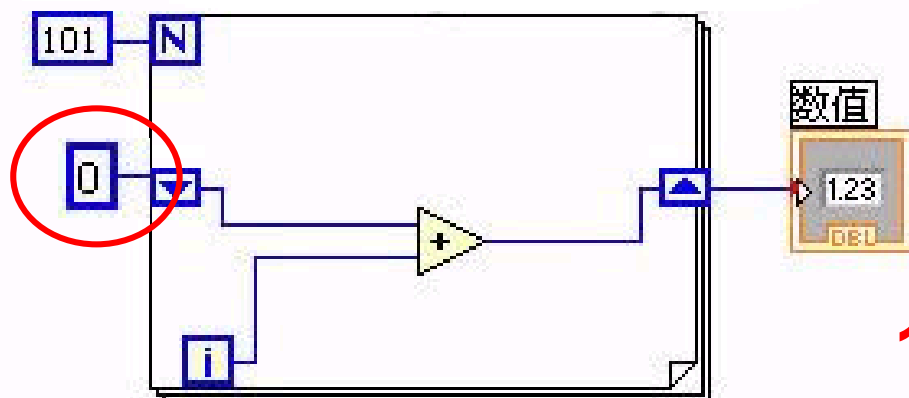
例：1+2+3+4+...+100的程序



说明：进行第一次运算的是 $0+0$ ，For循环中的重复端子是从0开始计数，移位寄存器在没有初始化的情况下，默认的数值是0，所以第一次运算的是 $0+0$ ，第二次运算的是寄存器的0与重复端子的1相加，所以循环要进行101次。而计数端子输出的数据始终是101。



第一次执行时输出：5050



10100

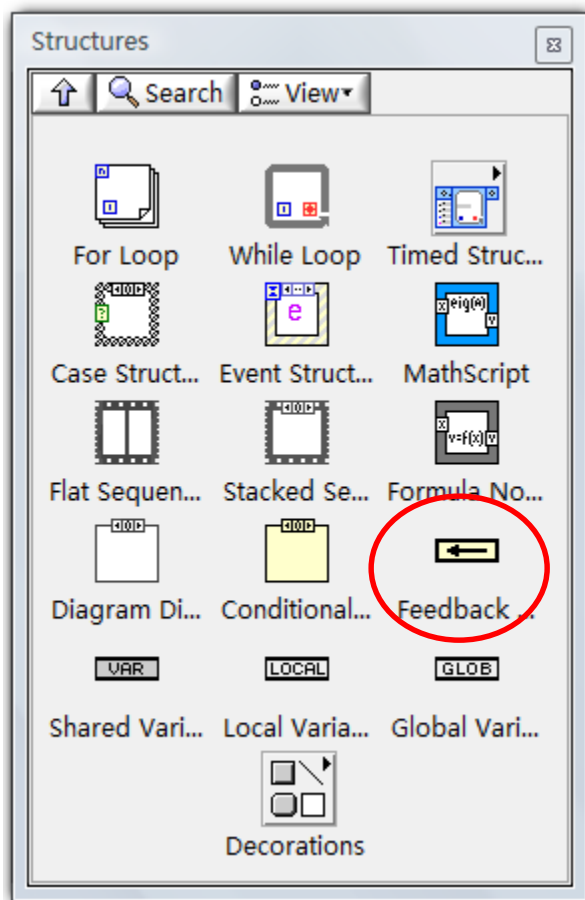
第二次执行时输出：5050

只要用户不退出VI，移位寄存器便可记录上次运算完时的结果。

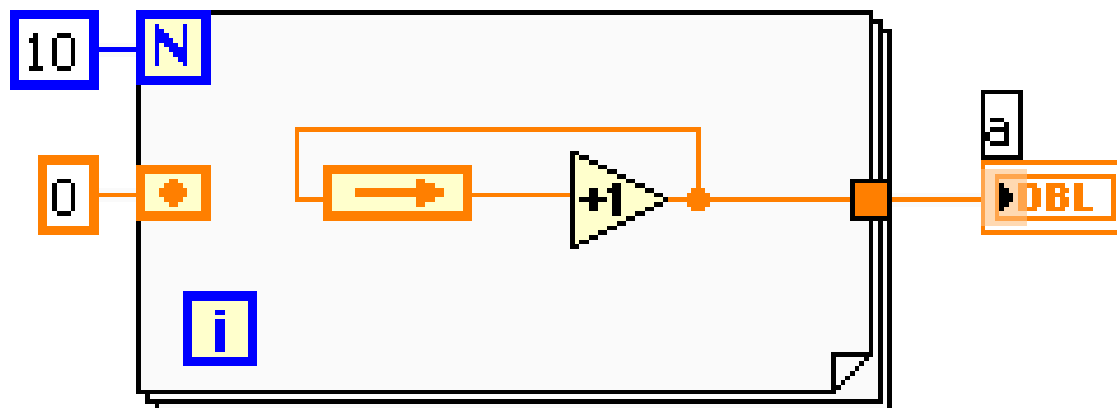
移位寄存器的初始化

◆ 反馈节点 (Feedback Node)

在For循环和While循环中把数据传递到下一次循环。



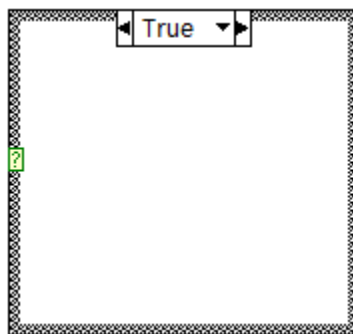
把一个函数或者一个子VI的输出与输入连接，会自动产生一个反馈节点。



通过反馈节点实现a++

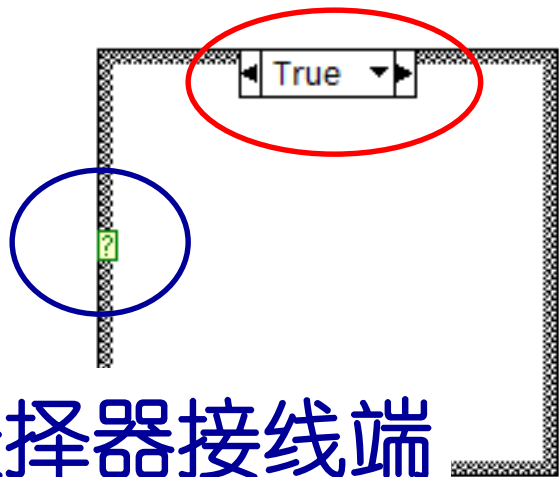
2.4 Case结构

Case结构包括两个或多个条件分支（子程序框图）。



每次只能显示一个子程序框图，每次只能执行一个条件分支。输入值决定执行哪一个子程序框图。

条件选择器标签



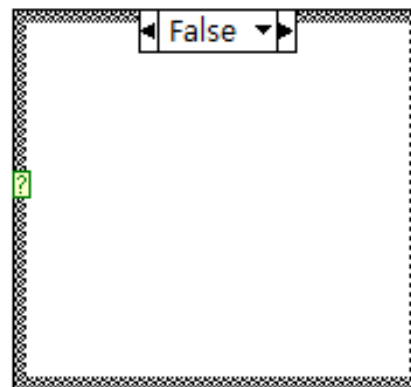
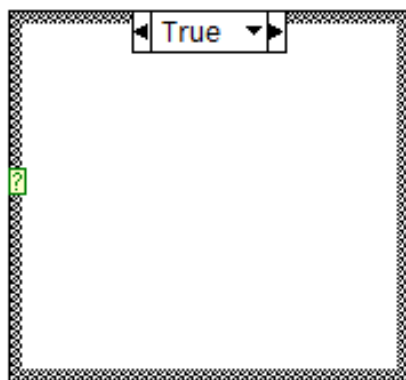
各个条件分支对应的
选择器值的名称；
左侧的递减箭头；
右侧的递增箭头；

将一个输入值或选择器连接到选择器
接线端即可执行所需执行的条件分支。

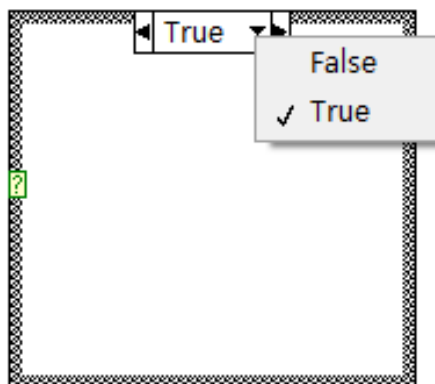
选择器接线端支持的数据类型：**整型、布尔
型、字符串型和枚举型。**

浏览各条件分支

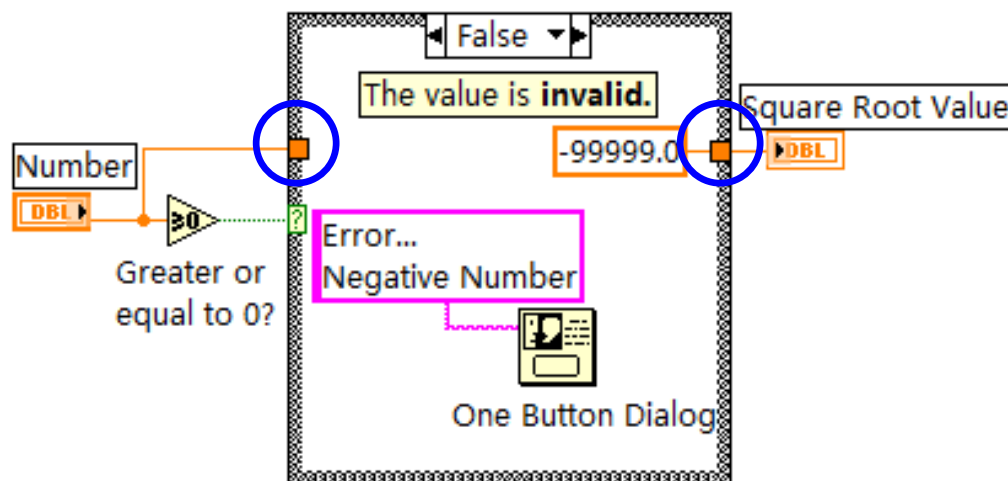
单击递减箭头或递增箭头。



单击向下箭头，在下拉菜单中选择一个条件分支。



◆ 输入和输出隧道



条件分支不一定使用所有输入隧道，但是必须为每个条件分支定义各自的输出隧道。

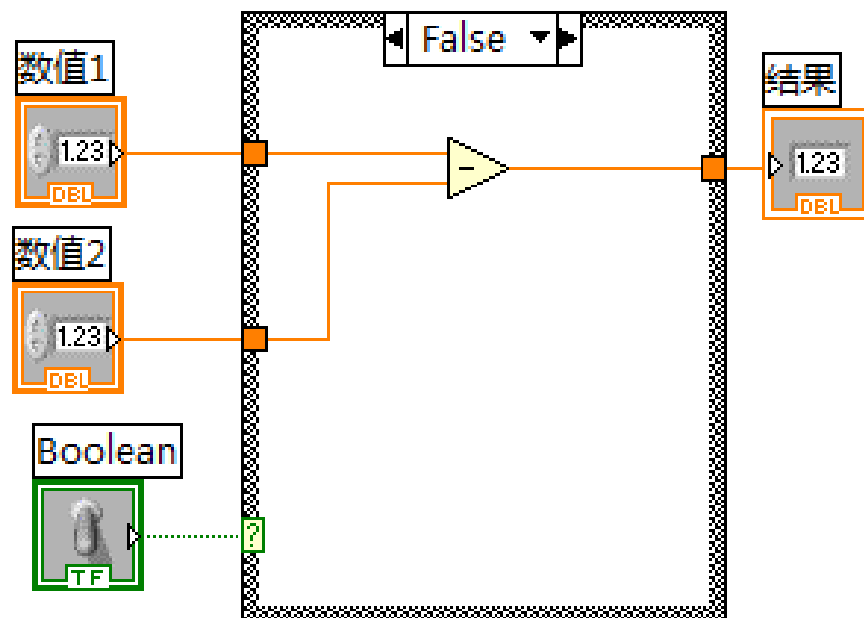
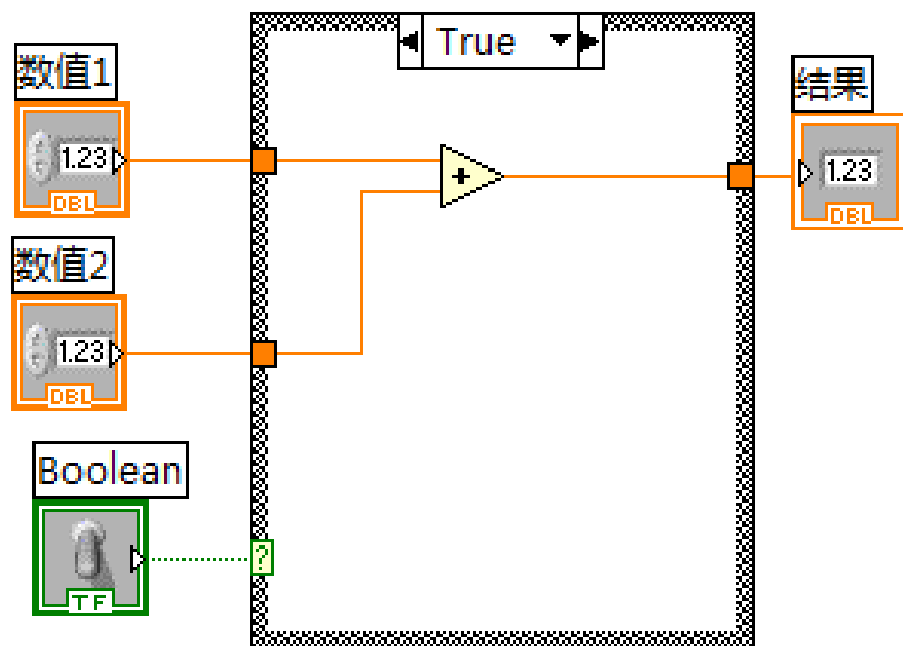
否则，右击输出隧道，快捷菜单中选择“未连接时使用默认值”。（避免使用）

注意：

应指定一个**默认条件分支**，处理超出范围的数值。否则应明确列出所有可能的输入值。

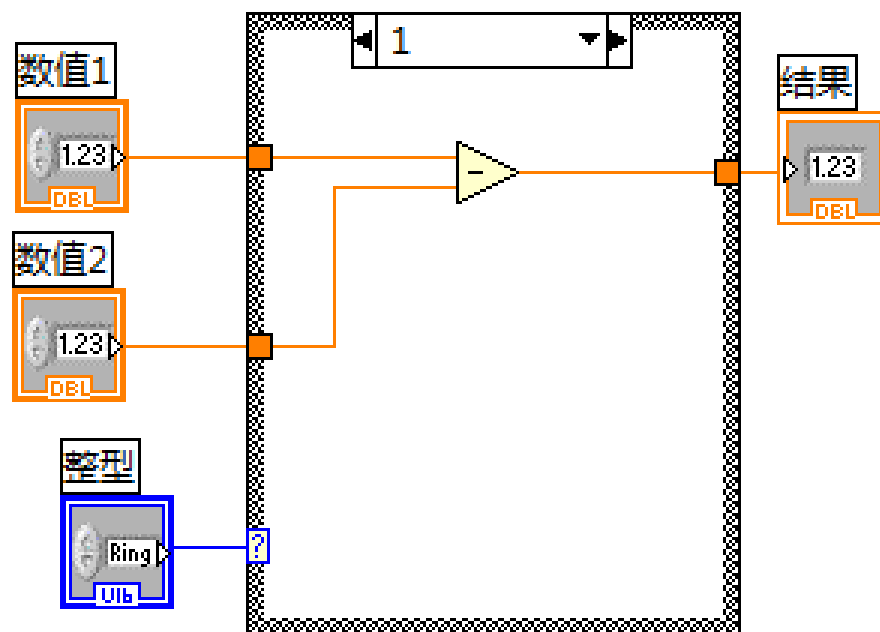
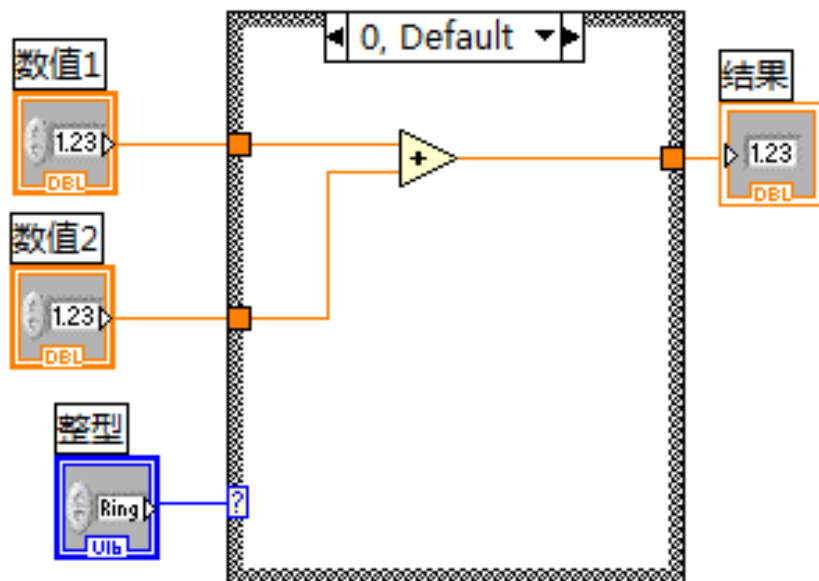
范例（布尔型条件结构）

只有真和假两个条件分支。



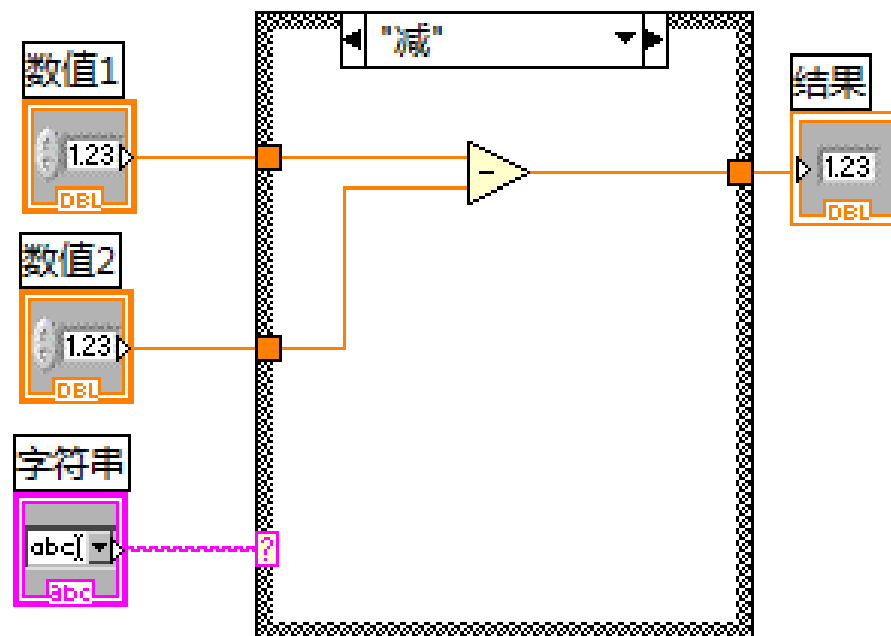
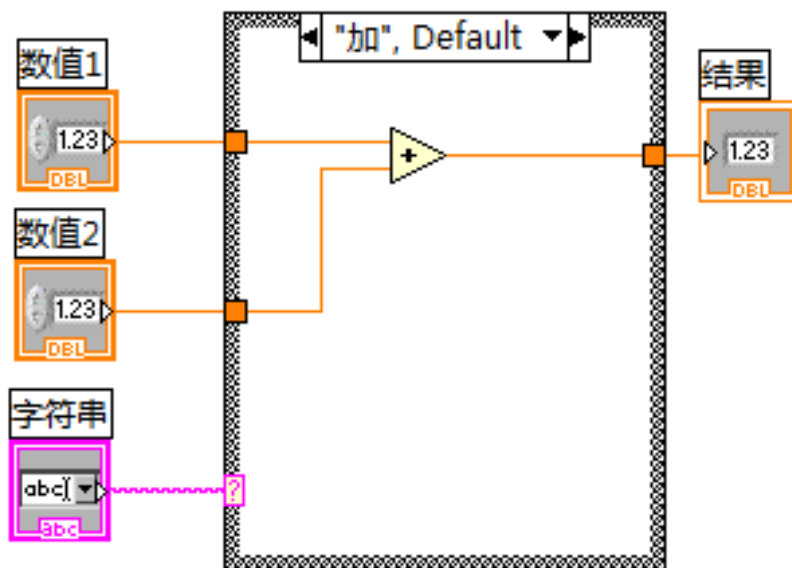
范例（**整型**条件结构）

可以有**多个**条件分支。



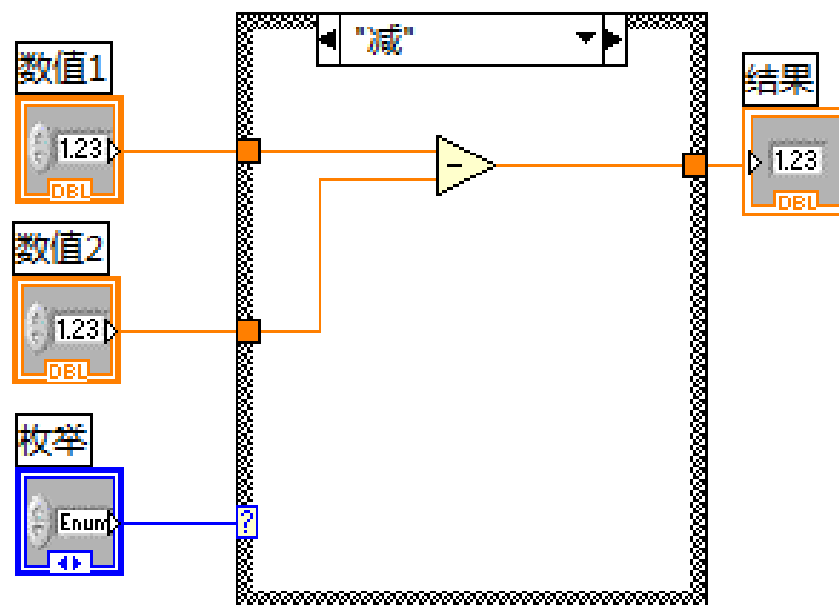
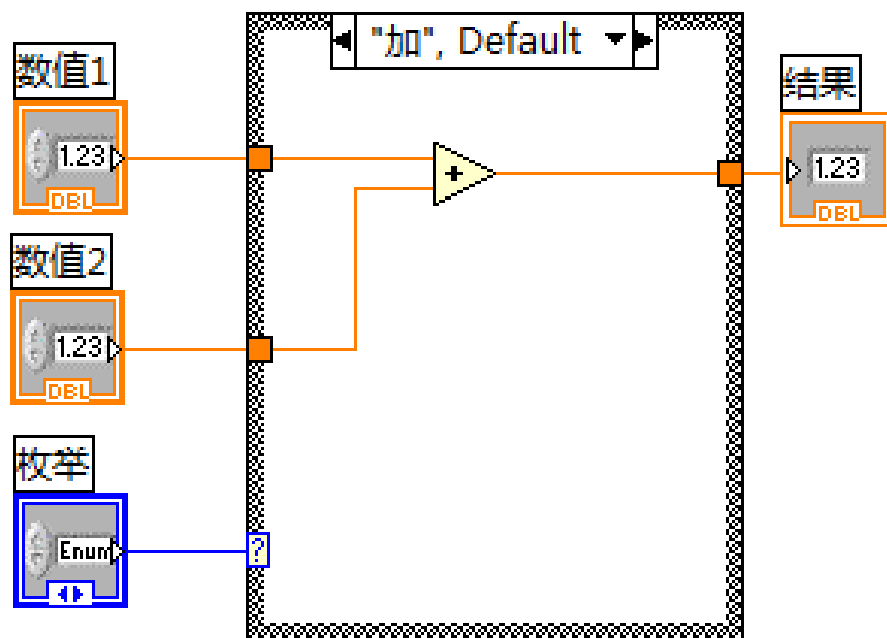
范例（字符串型条件结构）

可以有多个条件分支。



范例（枚举型条件结构）

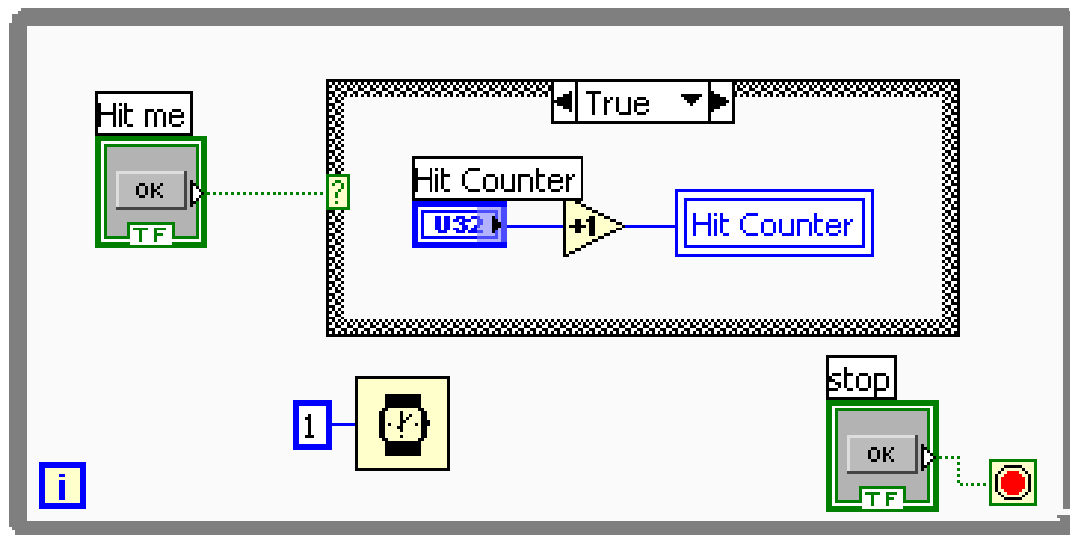
可以有多个条件分支。



2.5 事件结构 (Event Structure)

1. 定义

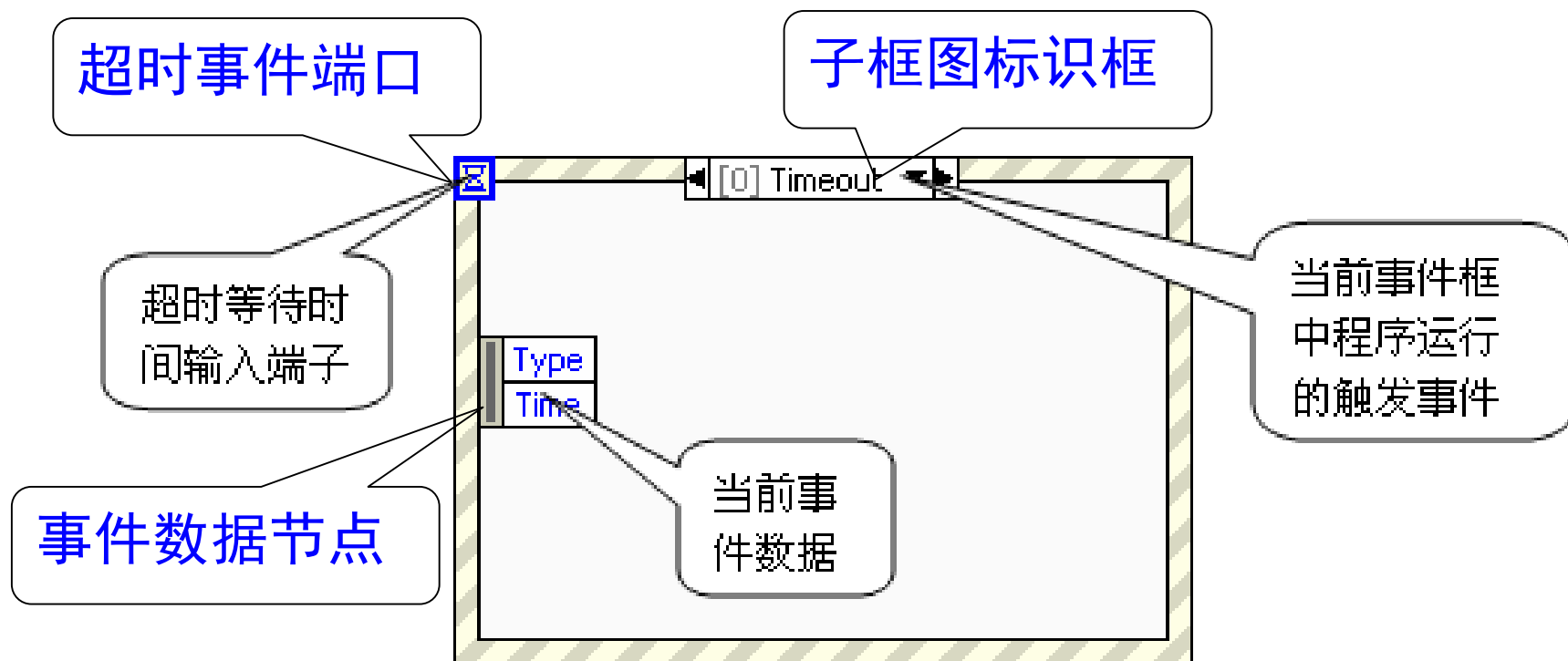
该程序会出现的问题？

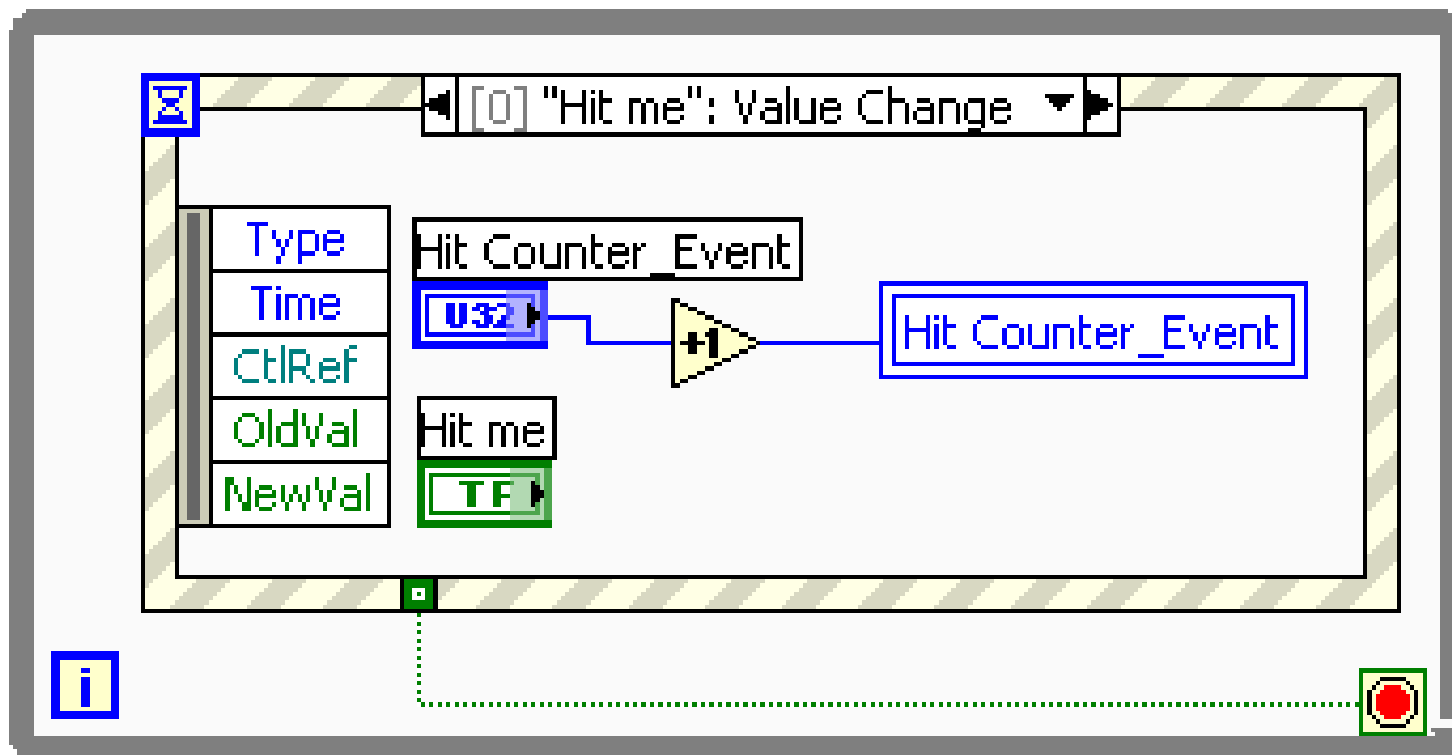


基于While循环和Case结构的单击计数器

在没有用户点击的情况下完全都是“空转”，浪费了大量的CPU资源，而且当“事件”发生太快时可能会被忽略。

因此Labview提供了**事件结构**来解决这个问题，
即仅当“事件”发生时，程序才做相应的响应。

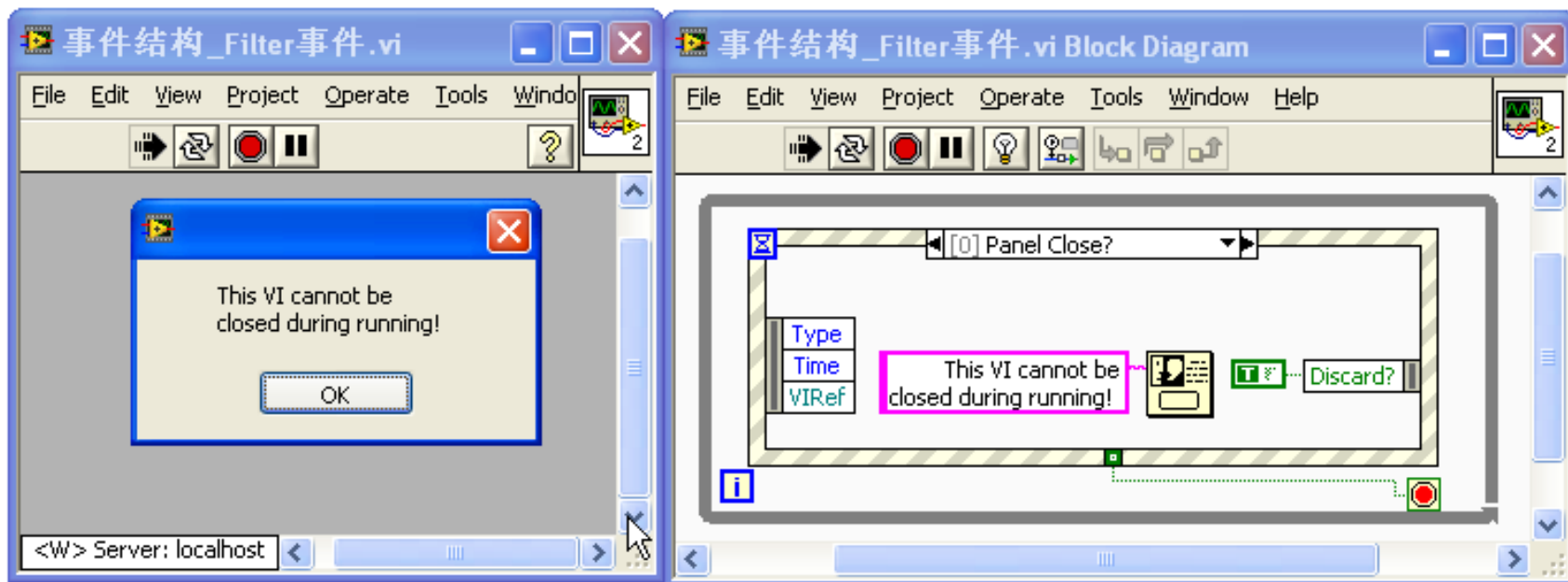




基于事件结构的单击计数器

2. Filter事件

当该事件发生时，用户可以过滤掉甚至完全放弃掉该事件将触发的活动。



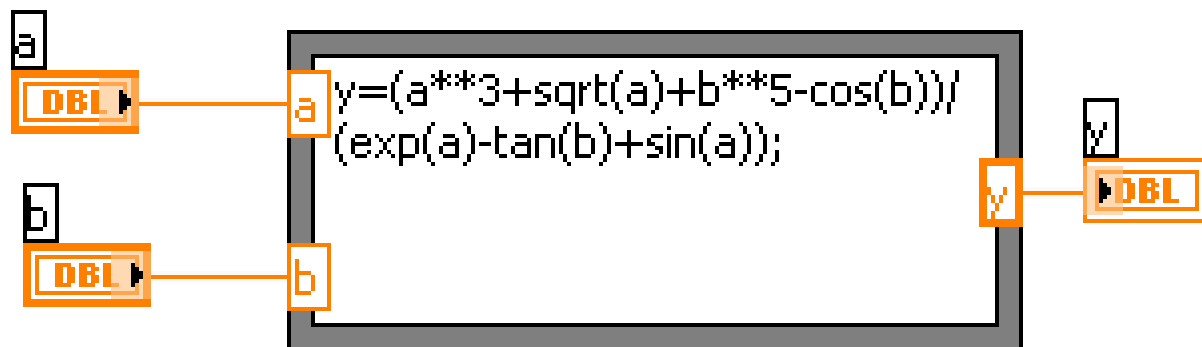
注意：

- ✓ 一般来说，事件只能通过**外在用户**的动作触发，如单击鼠标，键盘等。
- ✓ 如果需要**内部数据**触发事件（例如当 $a > b$ 条件满足时触发一个事件），就需要通过**用户自定义事件**的方法实现。

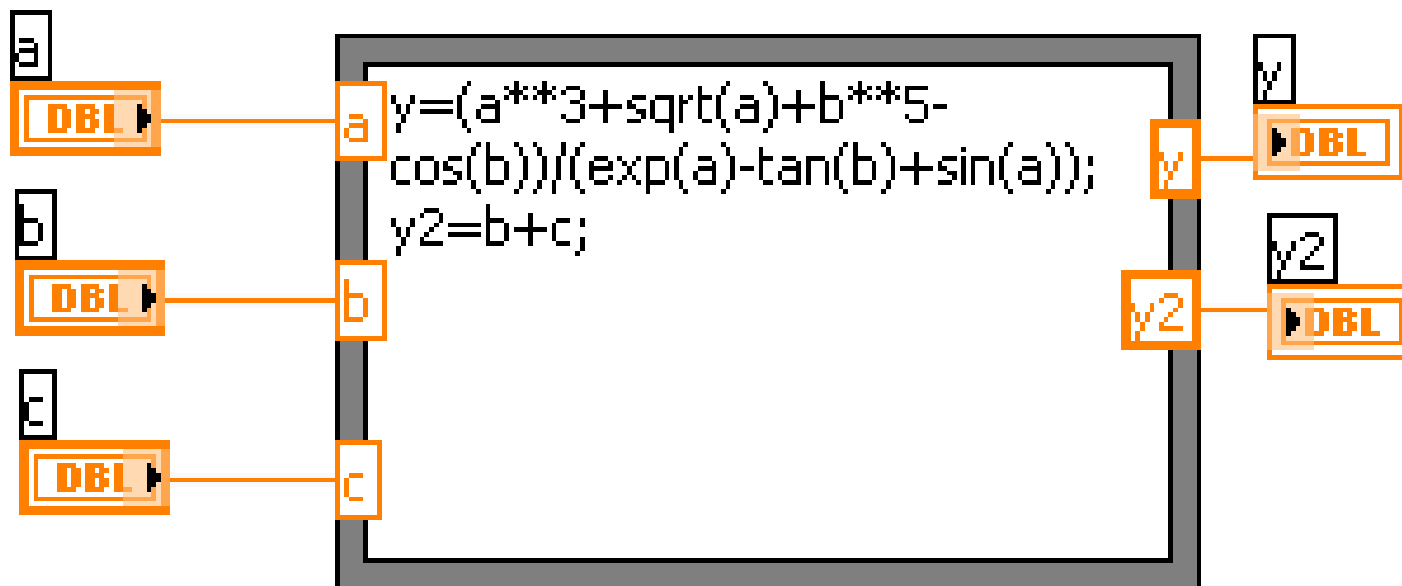
2.6 公式节点 (Formula Node)

通过公式节点，用户不仅可以实现复杂的数学公式，还能通过文本编程写一些基本的逻辑语句，如if...else..., case, while循环之类的语句。

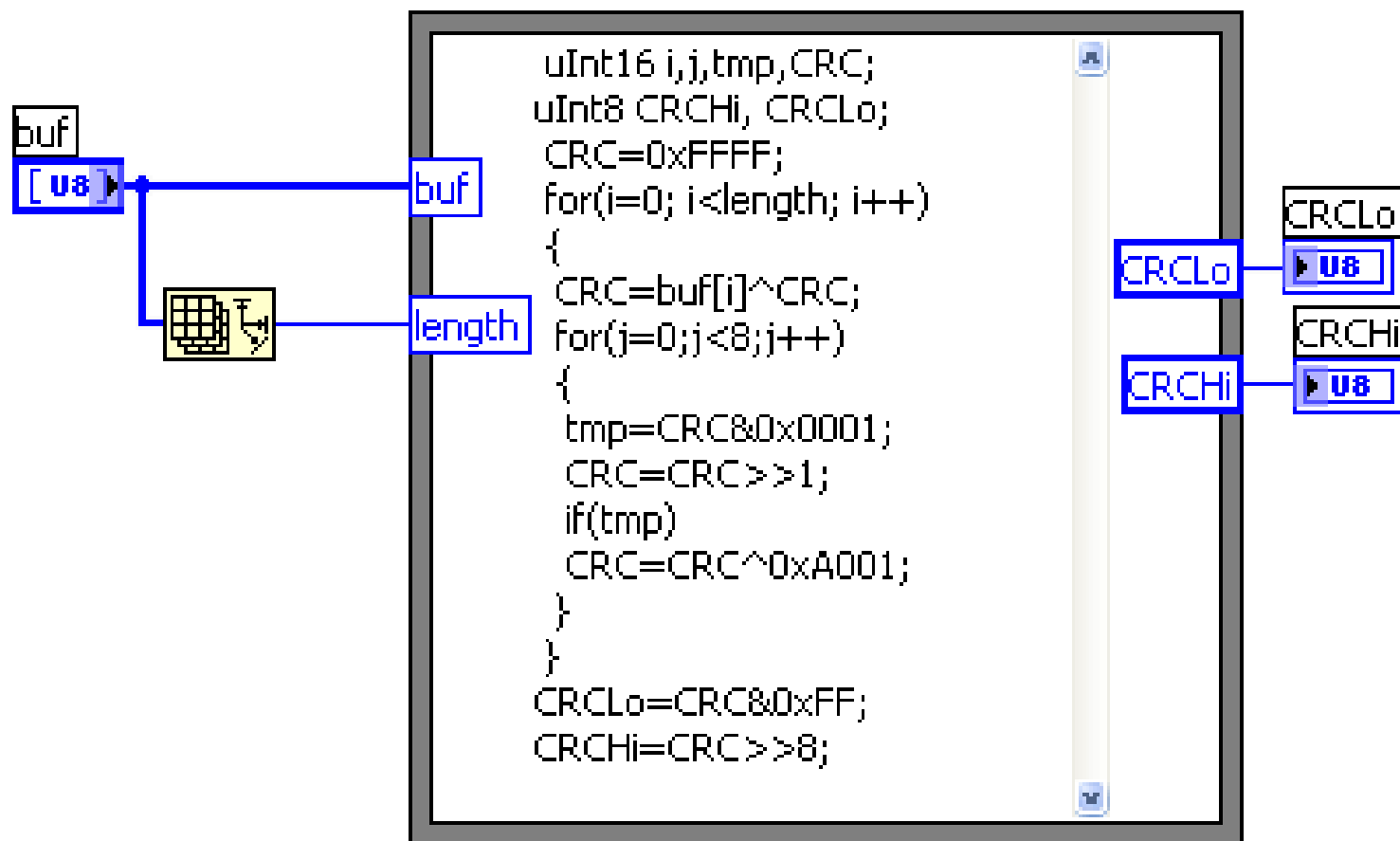
$$y = \frac{a^3 + \sqrt{a} + b^5 - \cos b}{e^a - \operatorname{tg} b + \sin a}$$



公式节点中可以包含任意数量的公式。



文本编程语言的实现



基于公式节点的CRC16算法的实现

第三章 数组和簇

- 数组 (Array)

- 簇 (Cluster)

——LabVIEW中的结构体变量

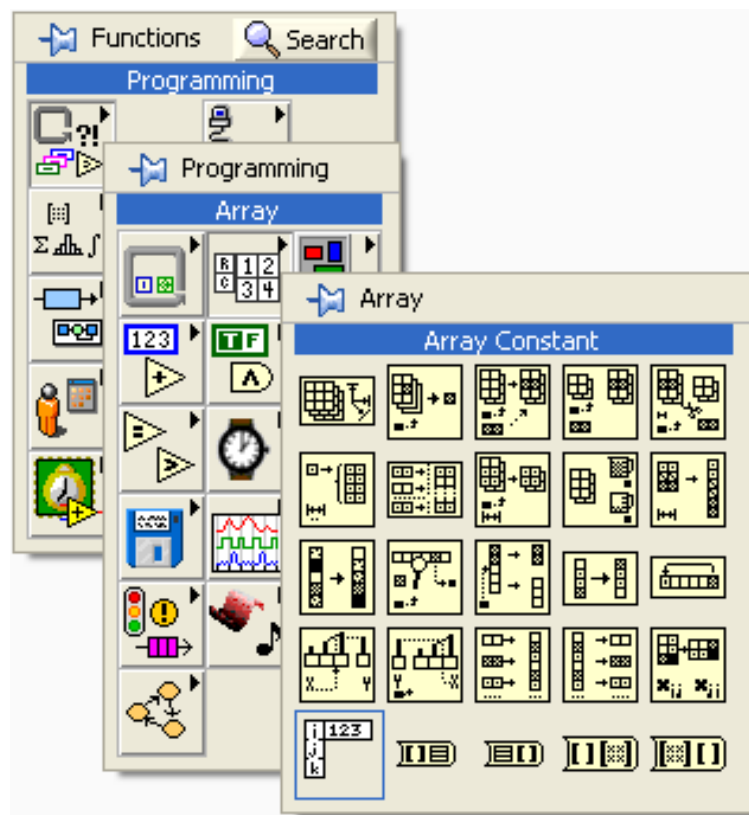
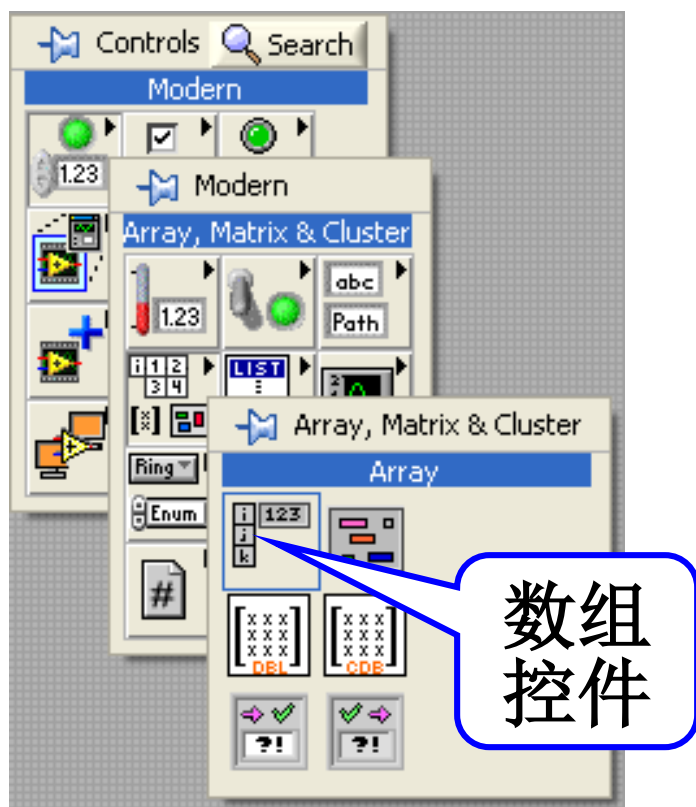
3.1 数组 (Array)

1. 数组的构成

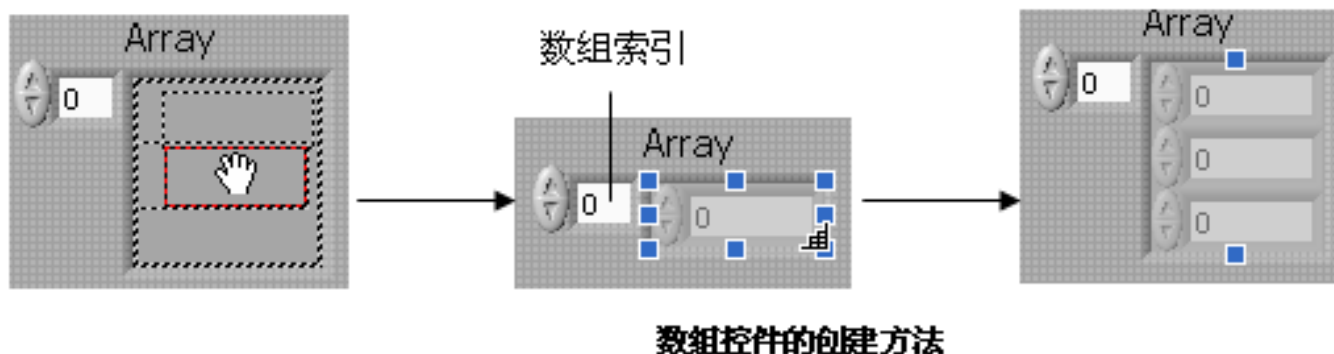
{	元素	组成数组的数据。
	维数	数组的长度、高度或深度。

数组元素是**有序**的。数组使用**索引**，便于访问数组中任意一个特定的元素。

2. 数组控件



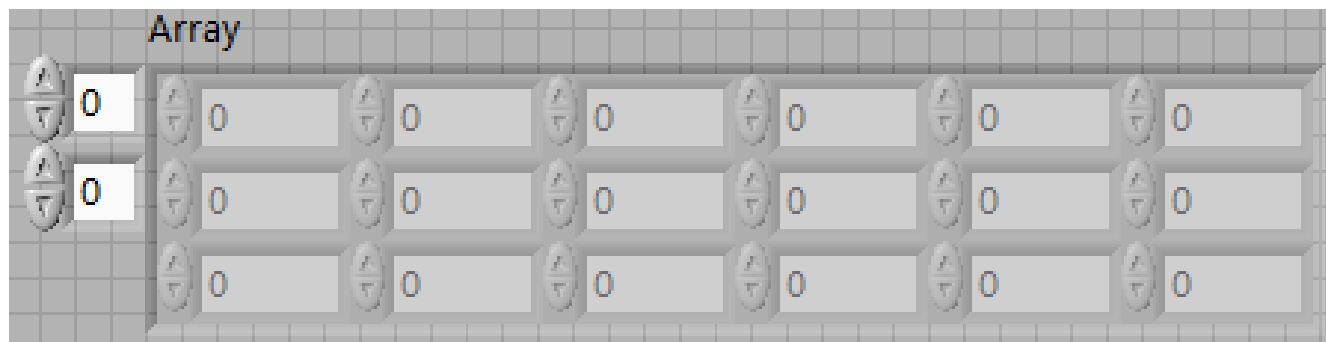
3. 数组控件的创建



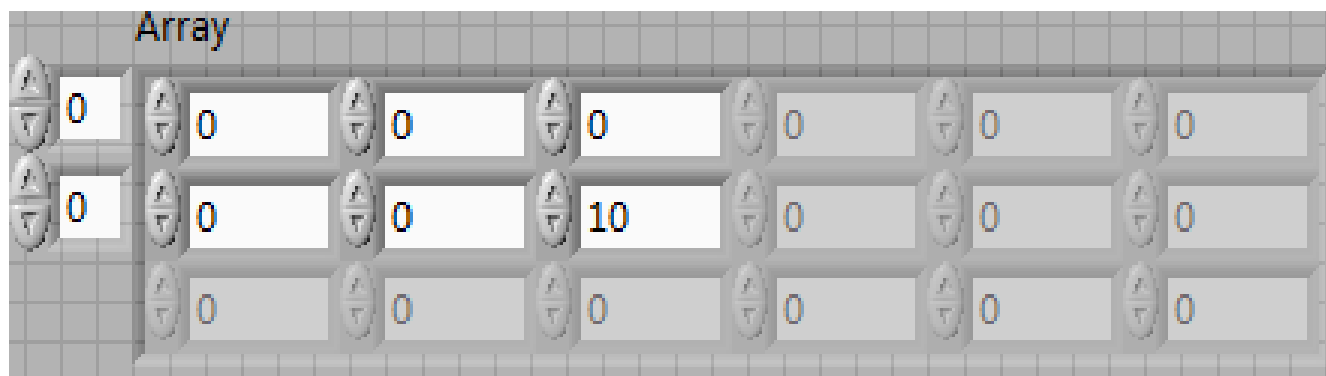
- (1) 前面板放置一个**数组外框**,
- (2) 将一个**数据对象或元素**拖放到该数组外框中。

数值、布尔值、字符串、路径、引用句柄、簇等。

4. 数组的初始化

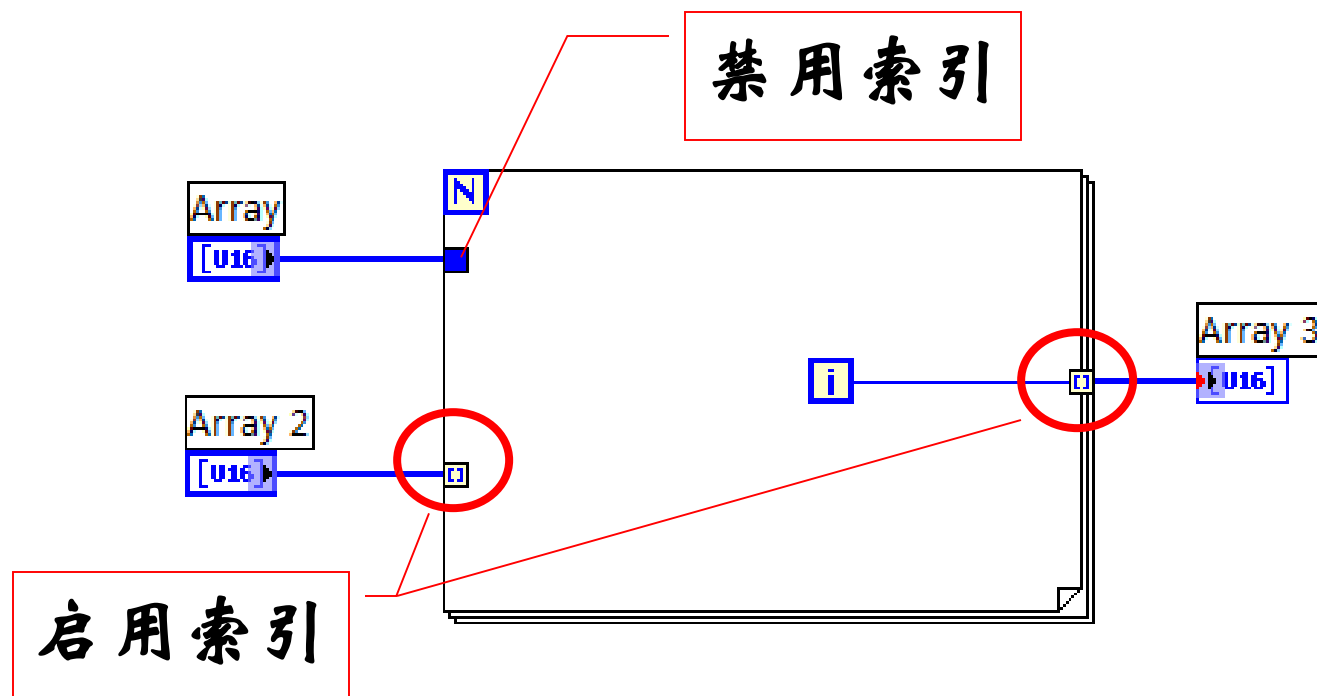


未初始化的二维数组



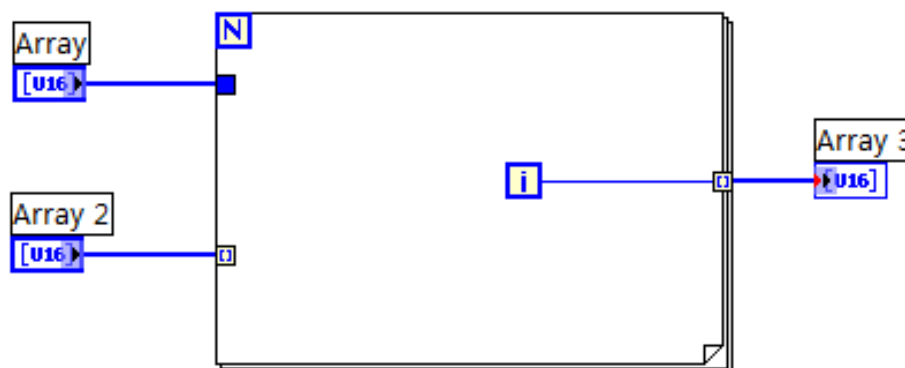
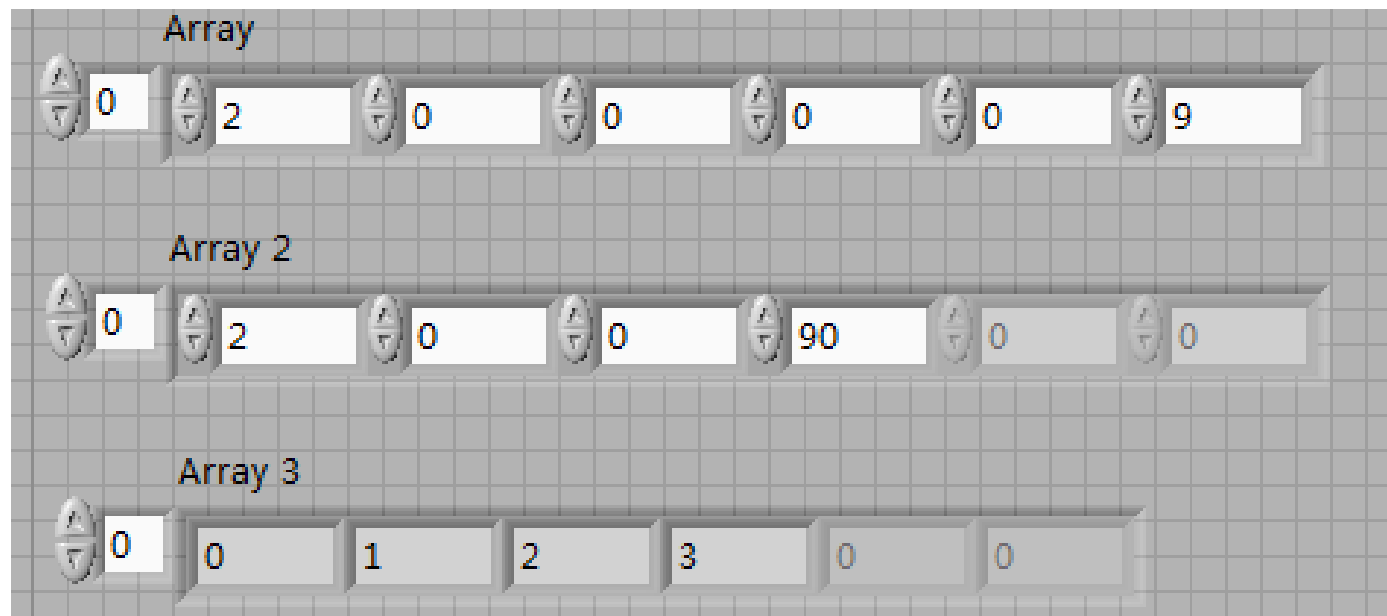
六个元素初始化的二维数组

5. 自动索引

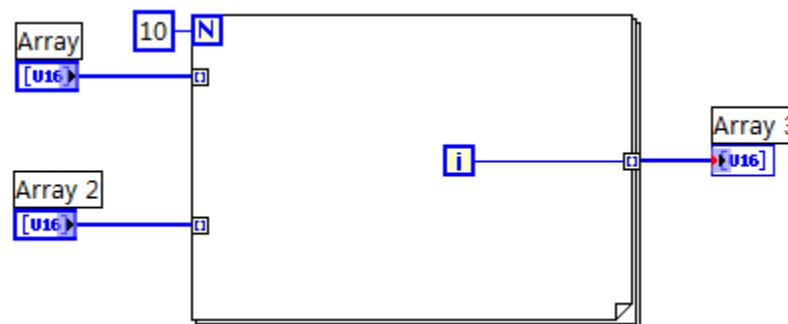
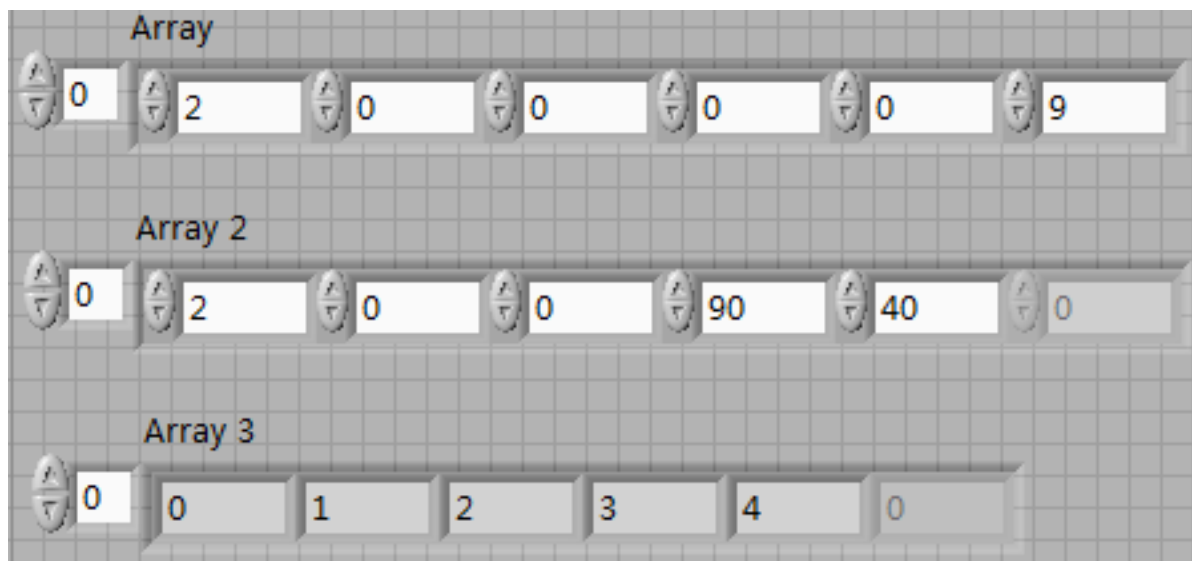


(1) 数组输入

- 计数接线端设置为数组的大小。



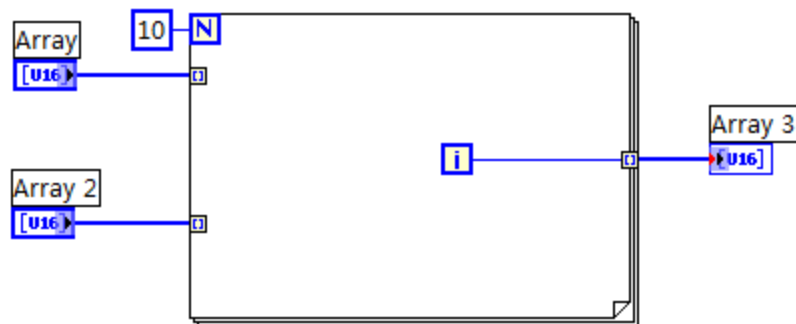
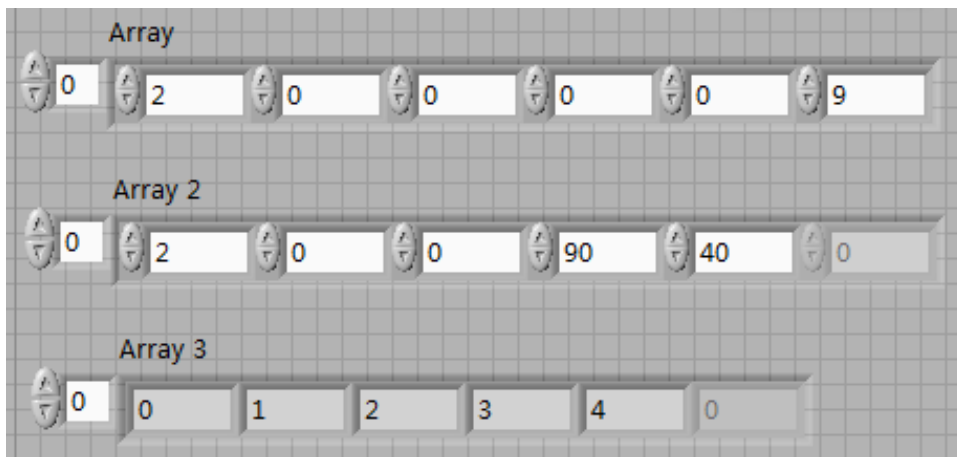
■ 多个隧道启用自动索引或对计数接线端进行连线，计数值取其中较小的值。



(2) 数组输出

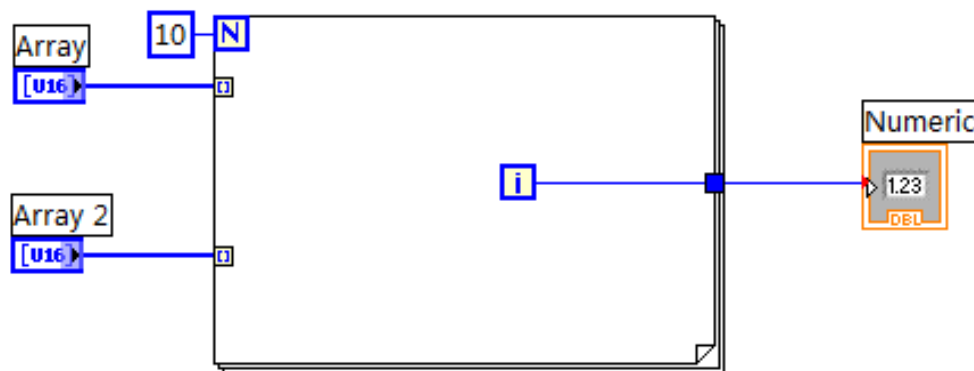
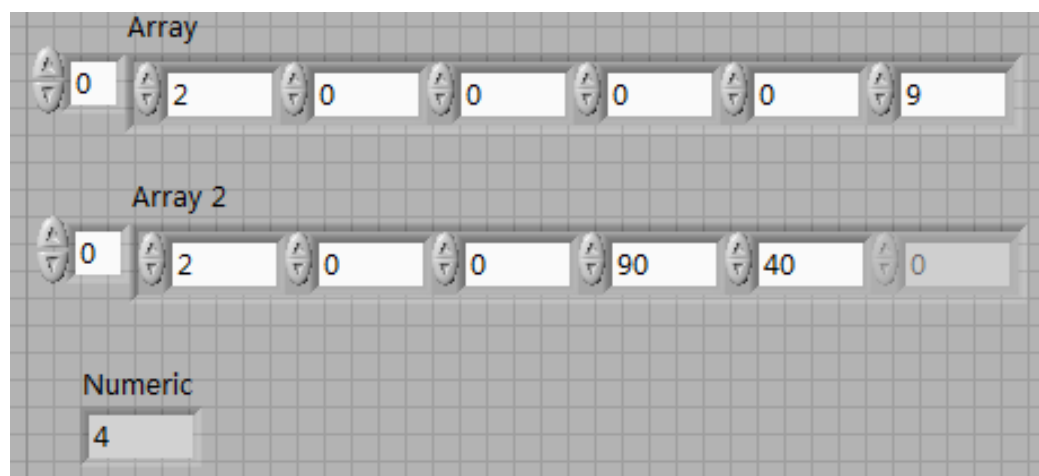
■ 启用自动索引

输出数组从每次循环中接收一个新元素。
数组大小等于循环的次数。



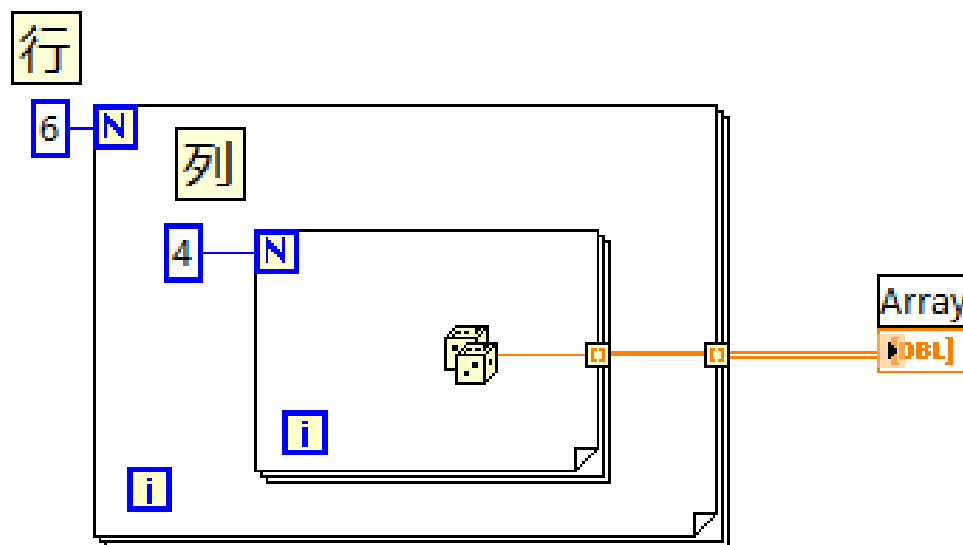
■ 禁用自动索引

只输出最后一个值。



(3) 创建二维数组

- 前面板创建二维数组
- 自动索引创建二维数组



Array

0	0.78263	0.92881	0.88995	0.50223	0
0	0.78566	0.44237	0.12126	0.07334	0
	0.63140	0.07206	0.86102	0.08730	0
	0.42529	0.71893	0.23311	0.79026	0
	0.78599	0.98223	0.45087	0.24051	0
	0.04255	0.29443	0.87270	0.36985	0
	0	0	0	0	0

- 添加/删除行列

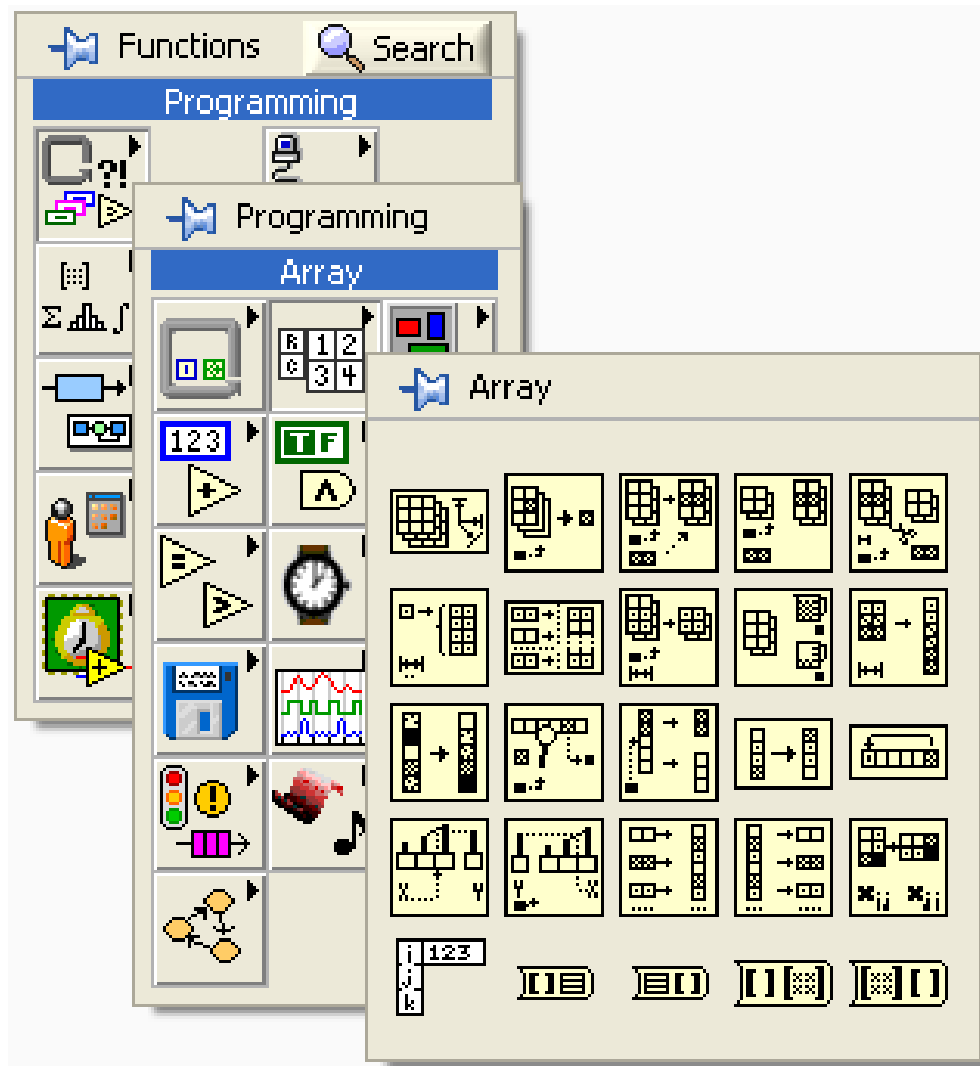
6. 数组之间的算术运算

LabVIEW可以根据输入数据的类型判断算子的运算方法，即自动实现多态。

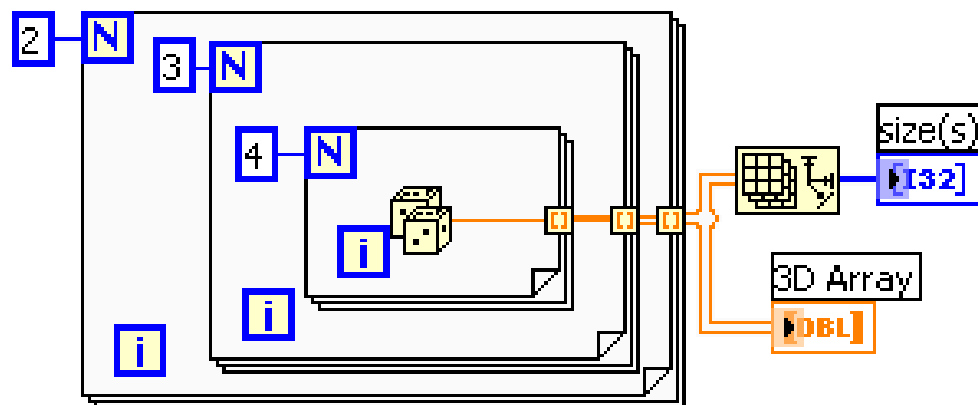
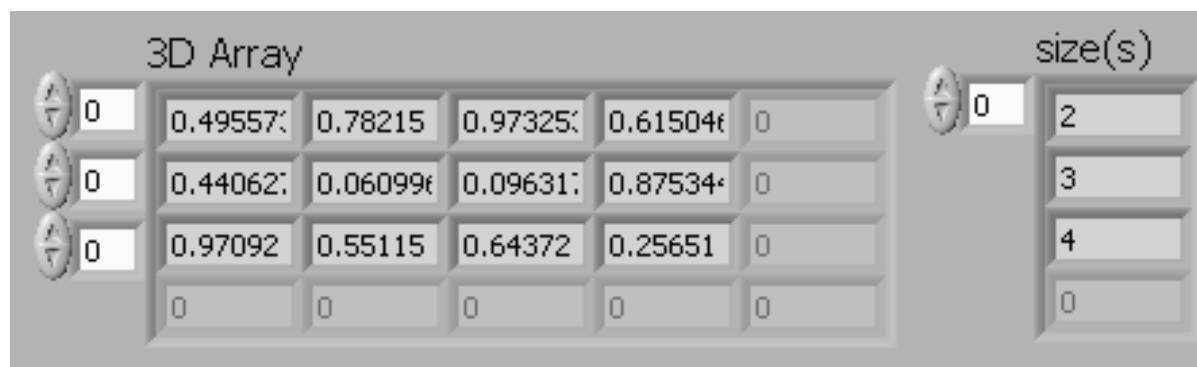
对于加减乘除，数组之间的运算满足下面的规则：

- ✓ 如果两个数组大小一样，则将两个数组中索引相同的元素进行运算形成一个新的数组。
- ✓ 若大小不一样，则忽略较大数组多出来的部分。
- ✓ 如果一个数组和一个数值进行运算，则数组的每个元素都和该数值进行运算从而输出一个新的数组。

7. 数组函数



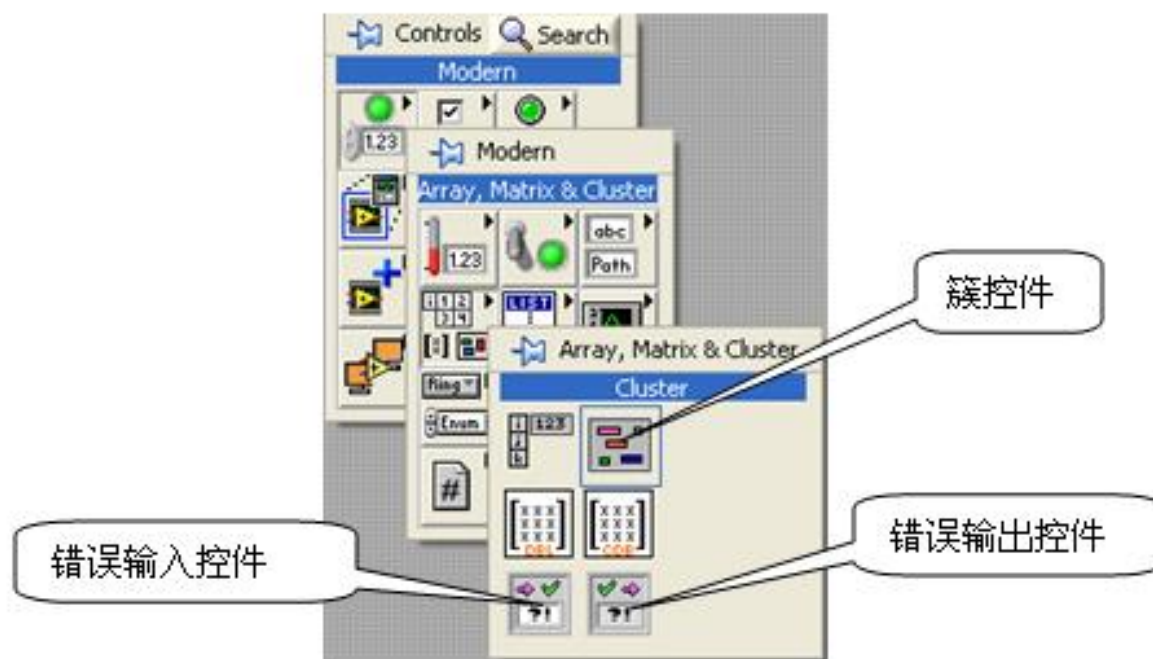
数组函数举例



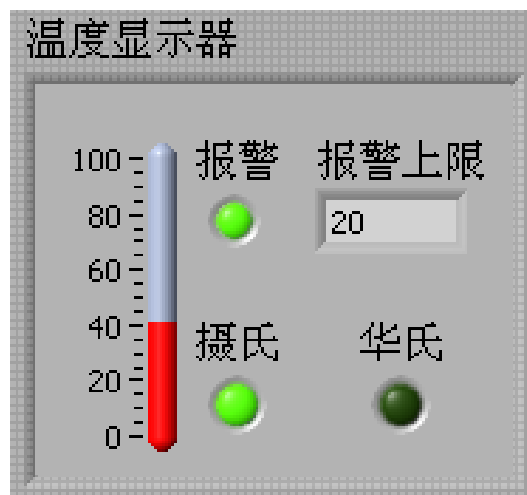
用Array Size 函数获得三维数组大小示例

3.2 簇 (Cluster) ——LabVIEW中的结构体变量

簇是LabVIEW中比较独特的一个概念，但实际上它就对应于C语言等文本编程语言中的结构体变量。



1. 簇的创建



通过簇控件实现的温度显示器

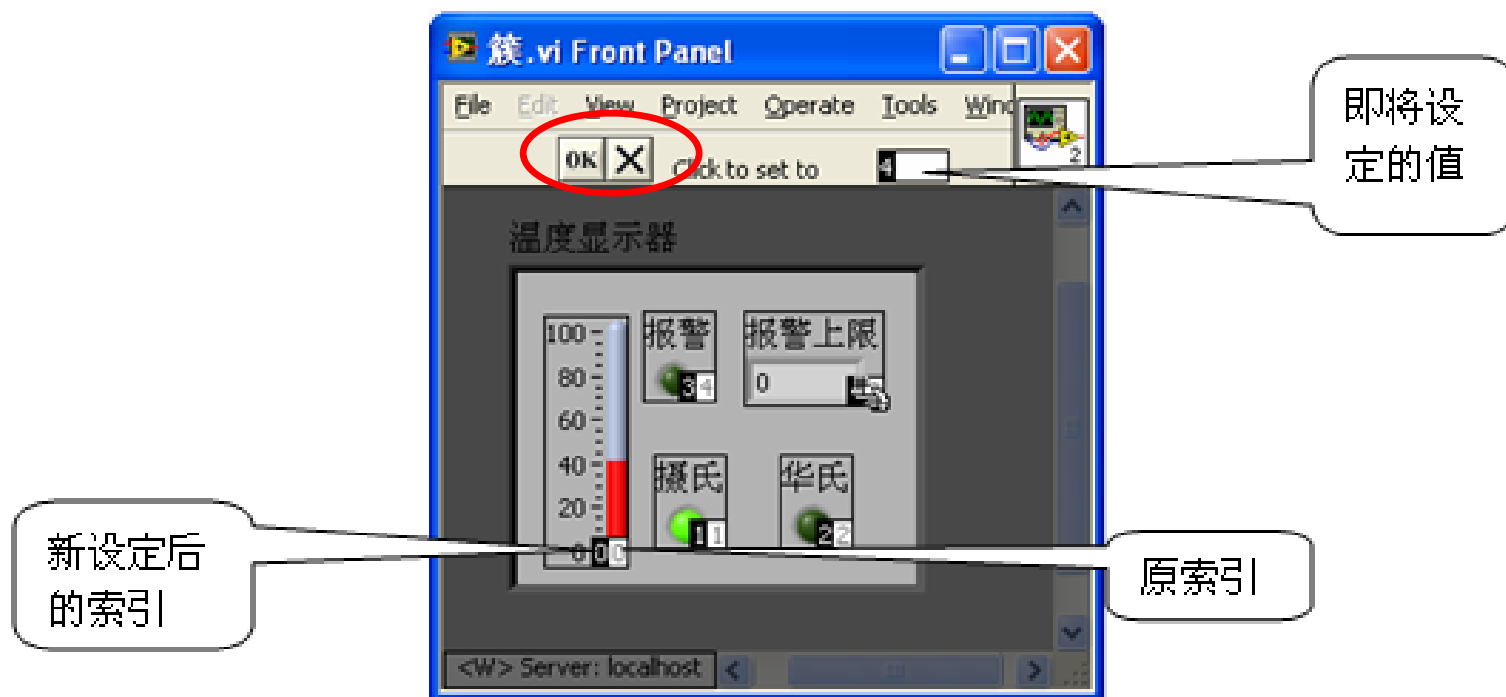
- (1) 前面板放置一个**簇外框**,
- (2) 将一个**数据对象或元素**拖放到该簇外框中。

数值、布尔值、字符串、路径、引用句柄、簇输入（输出）控件等。

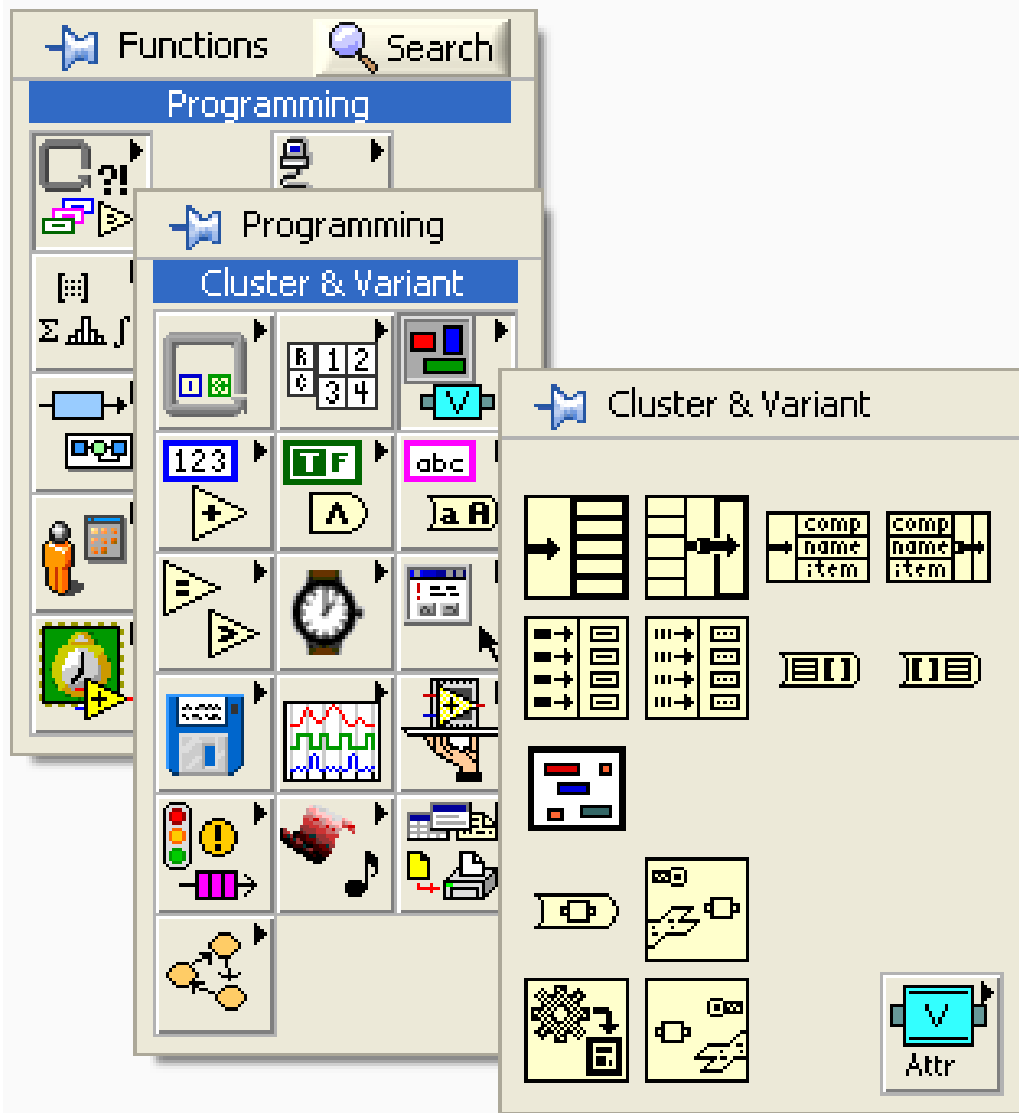
2. 改变簇内部元素控件的索引

簇元素的逻辑顺序由元素放入簇中的顺序决定。

右击簇边框，快捷菜单中选择**重新排序簇中控件**，可查看和修改簇顺序。

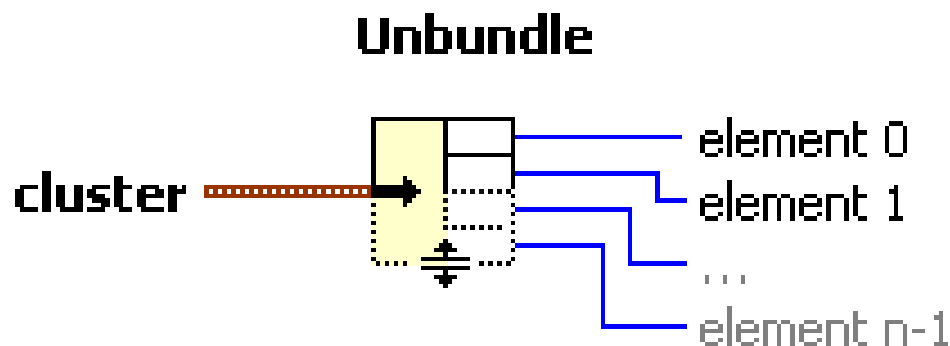


3. 簇操作函数



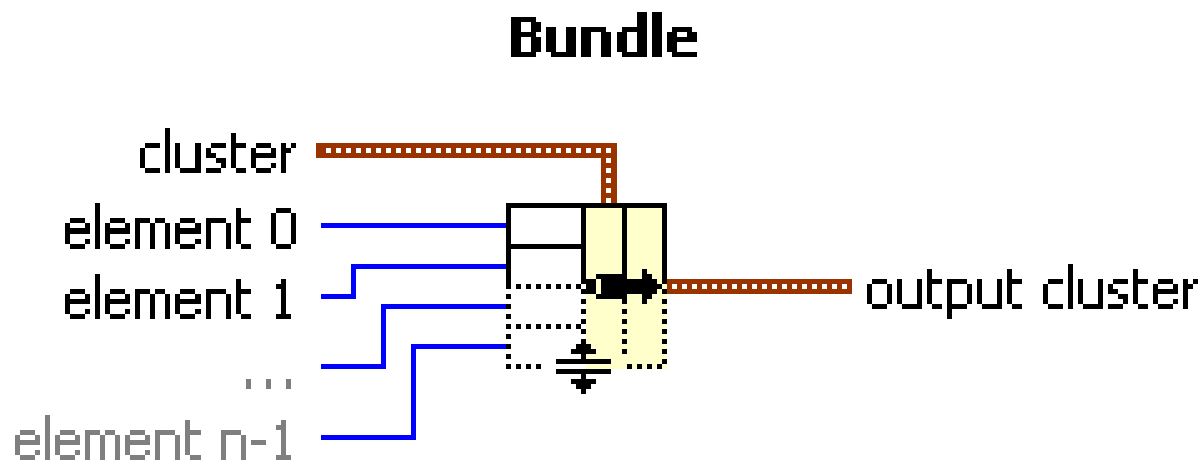
(1) 解包函数 (Unbundle)

将簇解开从而获得簇中各个元素的值。缺省情况下，它会根据输入的簇自动调整输出端子的数目和数据类型，并**按照簇内部元素索引的顺序**排列。



(2) 打包函数 (Bundle)

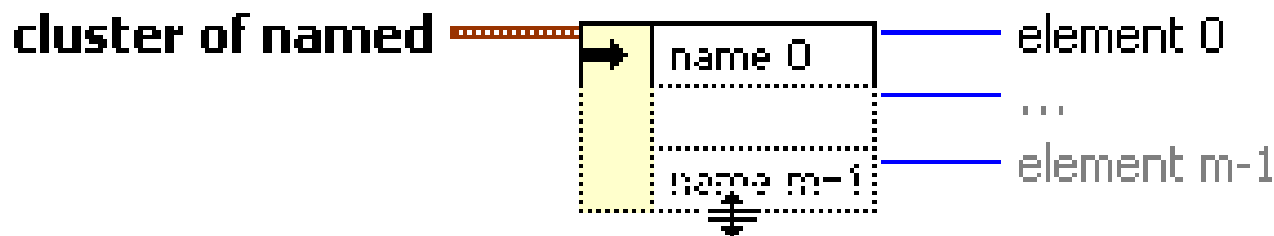
该函数用来为Cluster中各元素赋值。



(3) 按元素名称解包函数 (Unbundle By Name)

普通的解包函数解包后只有将鼠标移到输出端子上才能看到输出元素的名称，程序的可读性不高。该函数可以根据名称有选择的输出簇内部元素。其中元素名称就是指元素的**Label**。

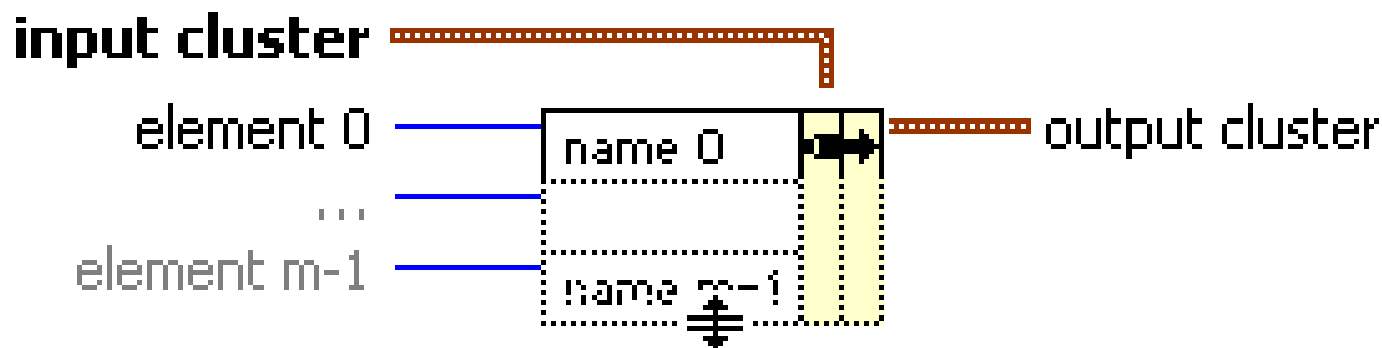
Unbundle By Name



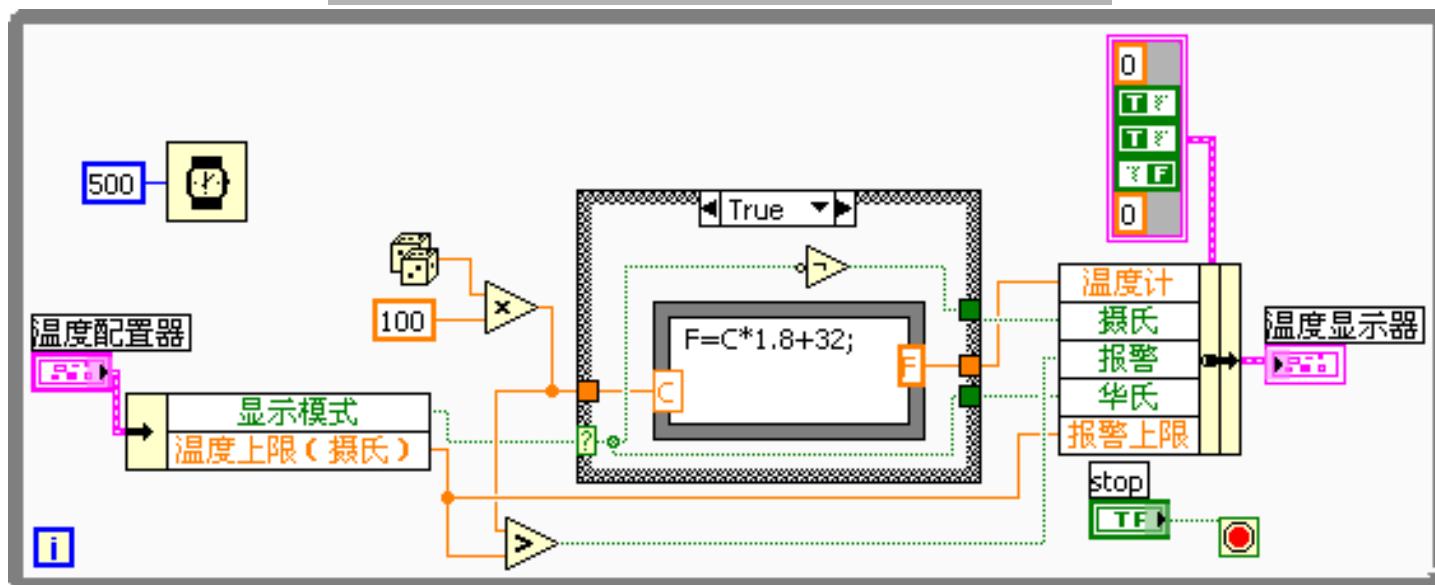
(4) 按元素名称打包函数 (Unbundle By Name)

该函数通过簇内部元素名称来给簇内部元素赋值。参考簇是必须的，该函数通过参考簇来获得元素名称。

Bundle By Name

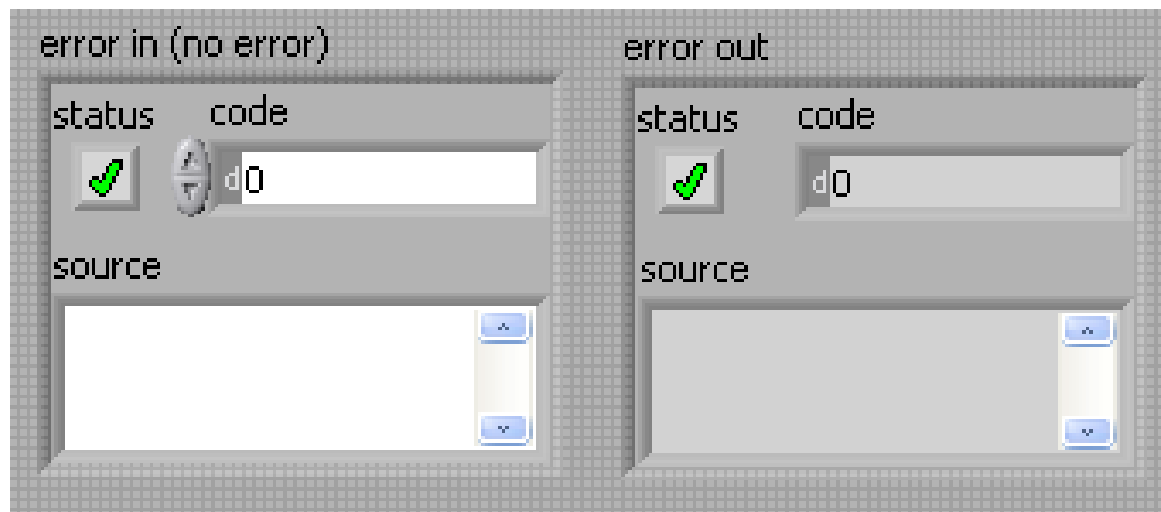


簇操作函数使用示例

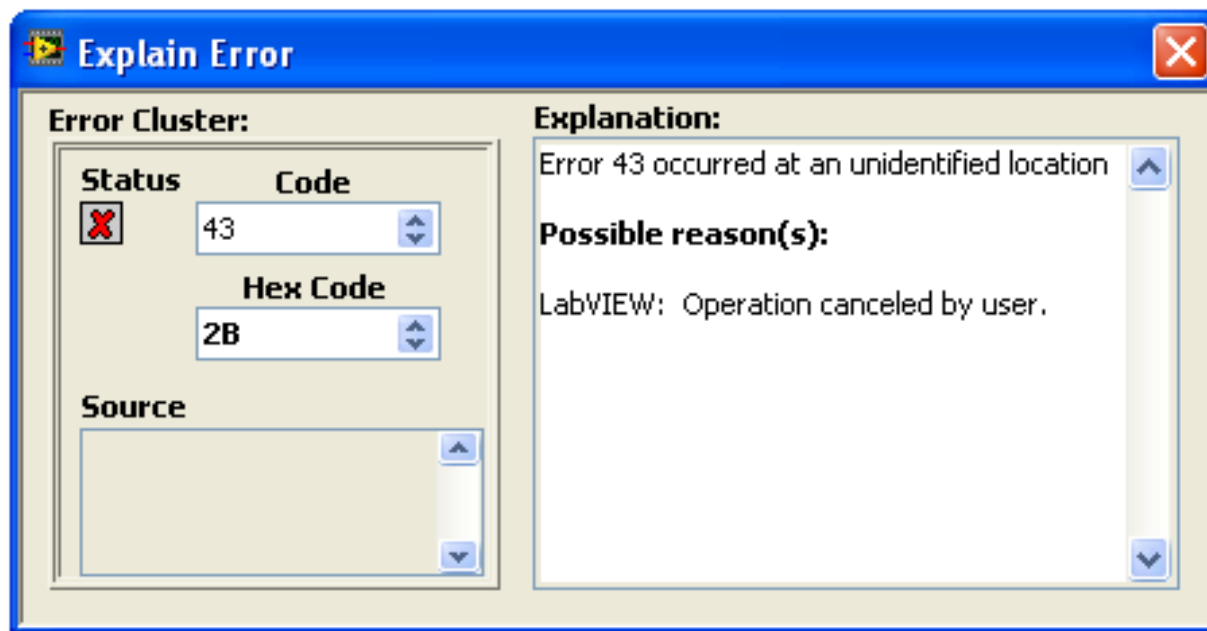


3. error in 和 error out 簇

LabVIEW利用error in 和error out这两个预定义簇来作为传递错误信息的载体。



对于系统错误，code都有预先的定义，可以通过选择Help->Explain Error...打开错误解释框来查找该错误代码的更详细的解释。

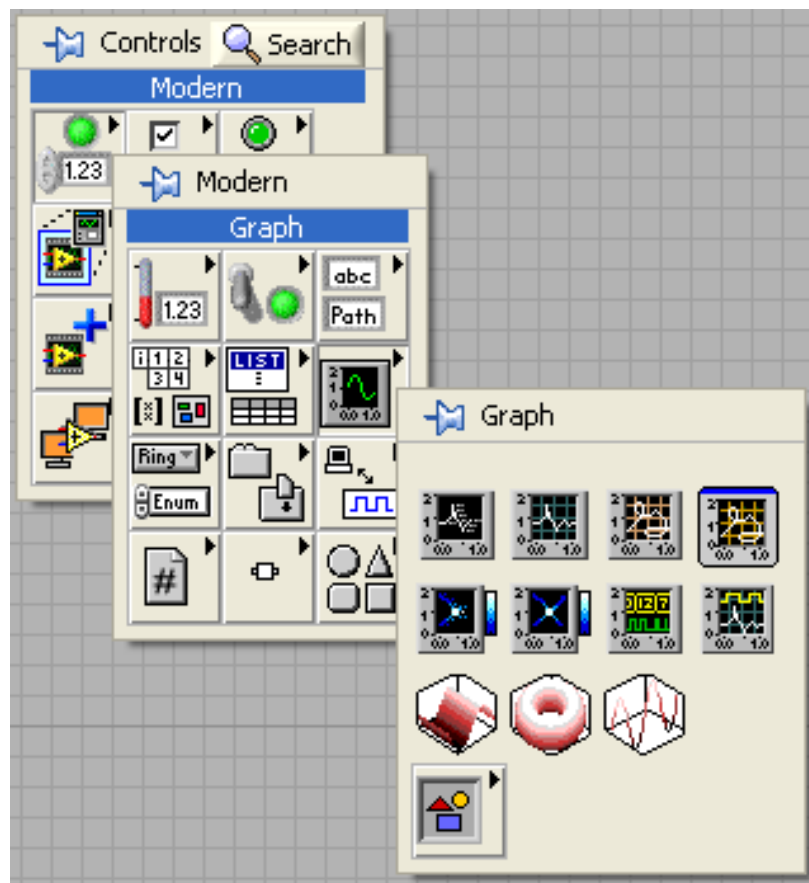


第4章 图形化显示数据 ——图表和图形

- 波形数据 (Waveform)
- Chart趋势图
- Graph图表
- 三维图形 (3D Graph)
- Picture图形控件

引子

LabVIEW很大的一个优势就是它提供了丰富的数据图形化显示控件，而且使用起来极其方便。

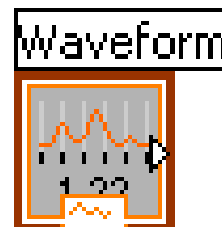


4.1 波形数据(Waveform)

1. 波形数据控件

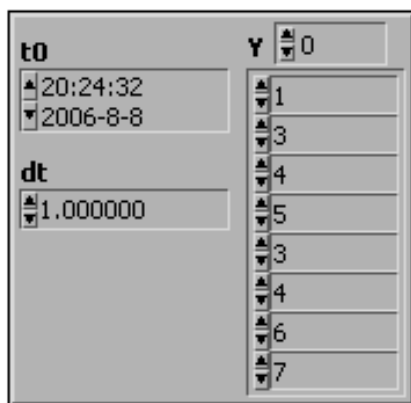
为了方便地显示波形，LabVIEW专门预定义了波形数据类型。它实际上就是按照一定格式预定义的**簇**，在信号采集、处理和分析过程中经常会用到它。

Waveform

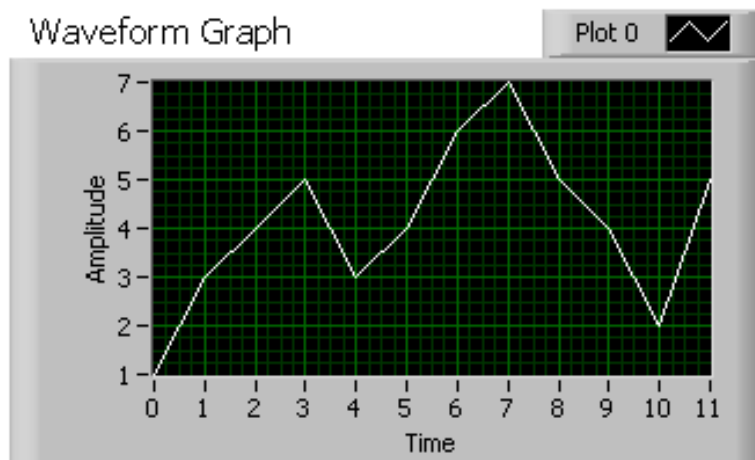


波形数据控件携带的数据包含了时间波形的基本信息，因此可以直接作为Chart和Graph的输入。横坐标代表时间，纵坐标代表Y值。

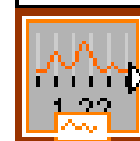
Waveform



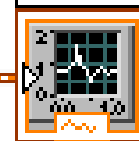
Waveform Graph



Waveform



Waveform Graph

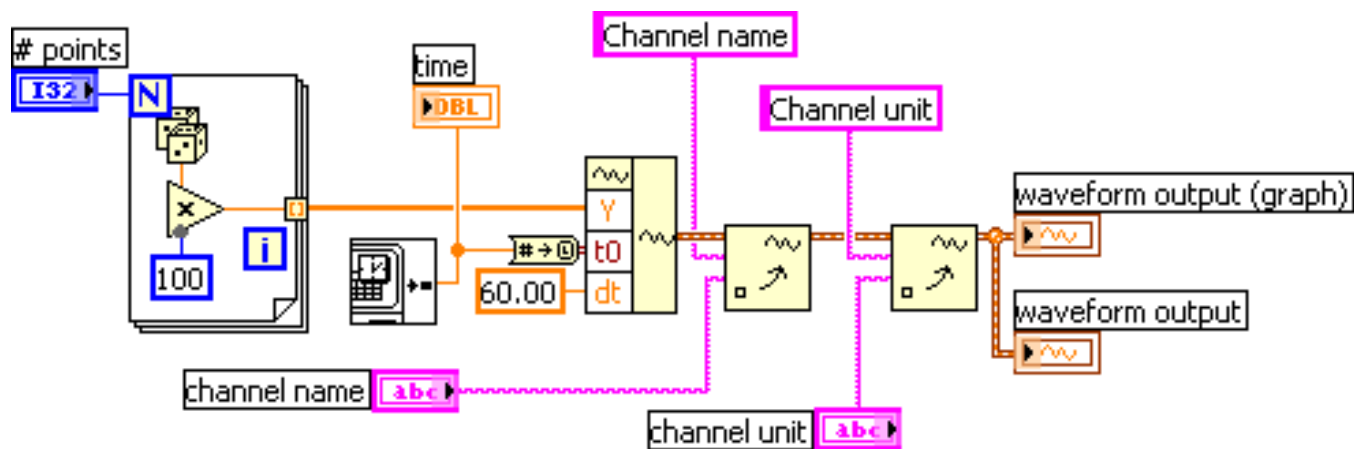
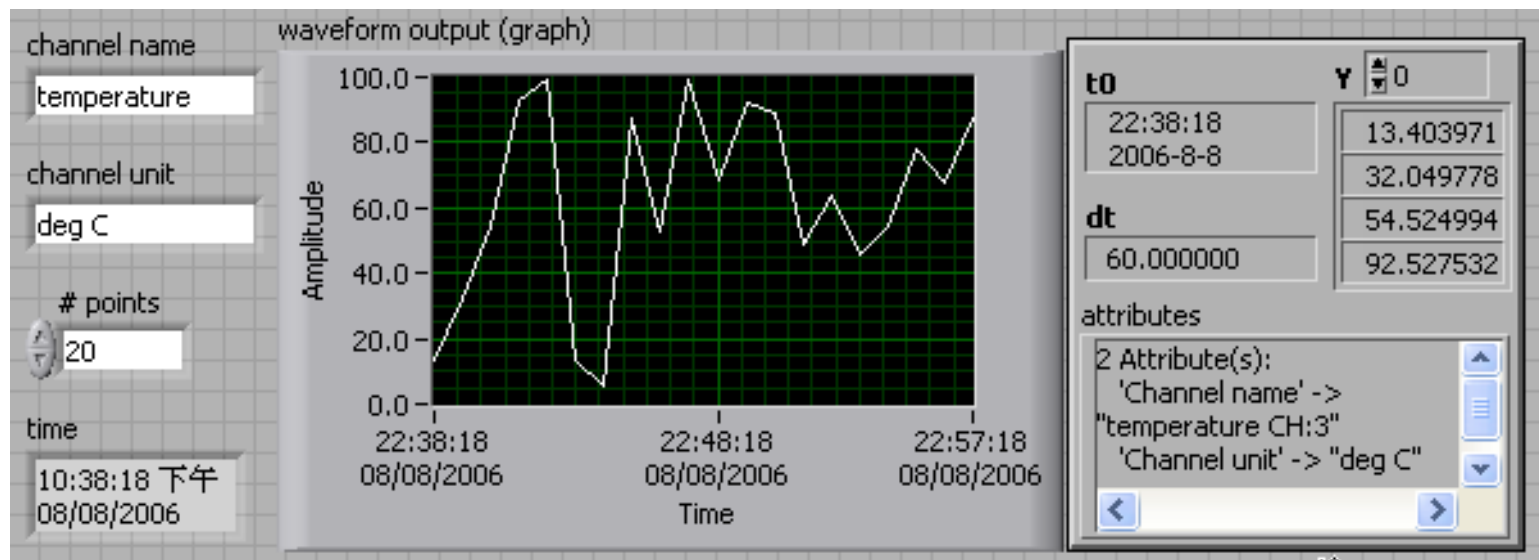


2. 波形数据操作函数

虽然波形数据是一种预定义格式的簇，但是必须用**专用的波形数据操作函数**才能对它进行操作，其中某些操作函数与簇的操作函数非常类似。

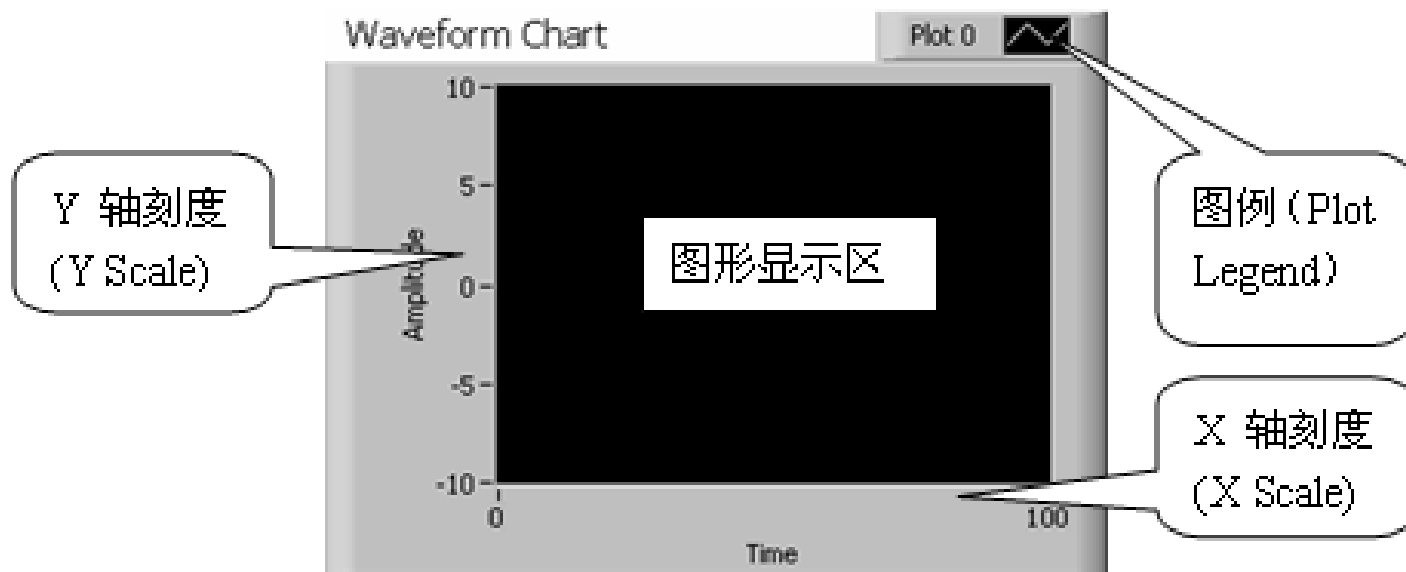
波形数据的操作函数位于Functions Palette 的Programming->Waveform...子模板下。

波形数据操作函数举例



4.2 Waveform Chart (波形图表)

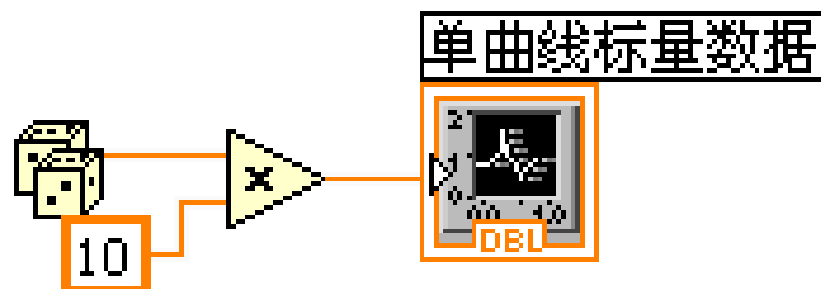
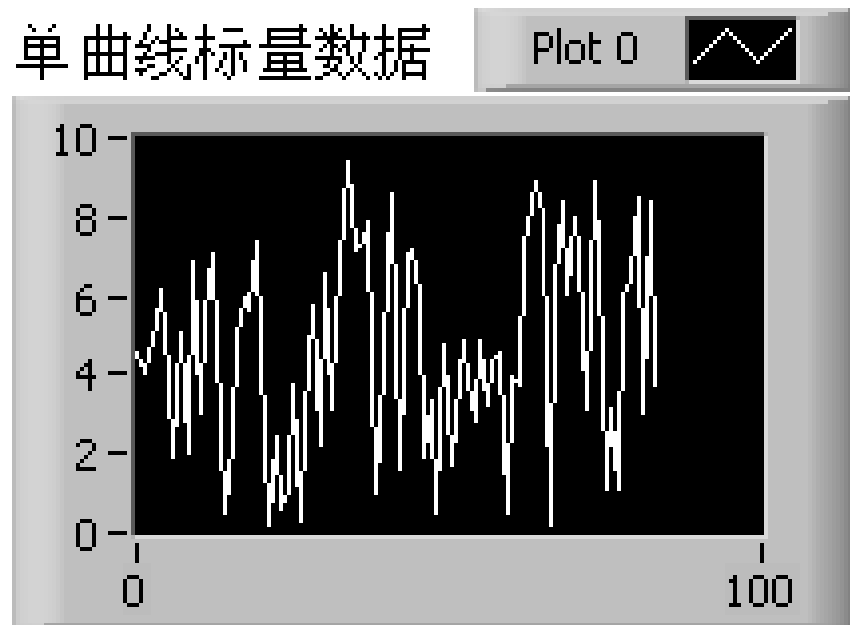
Chart可以将新测得的数据添加到曲线的尾端，从而反映实时数据的变化趋势，它主要用来**显示实时曲线**。



1. 波形图表 (Waveform Chart)

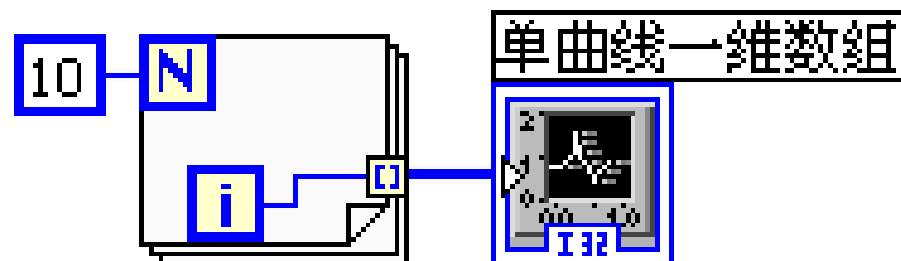
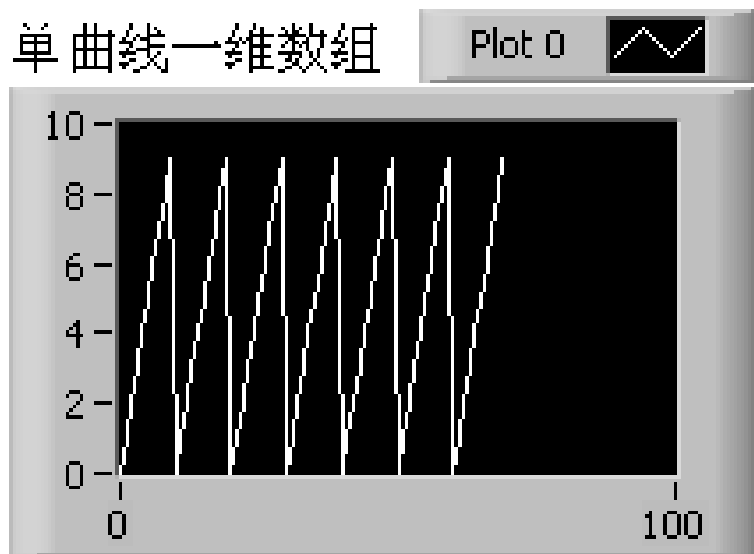
(1) 对于标量数据

Chart图表直接将数据添加在曲线的尾端。



(2) 对于一维数组数据

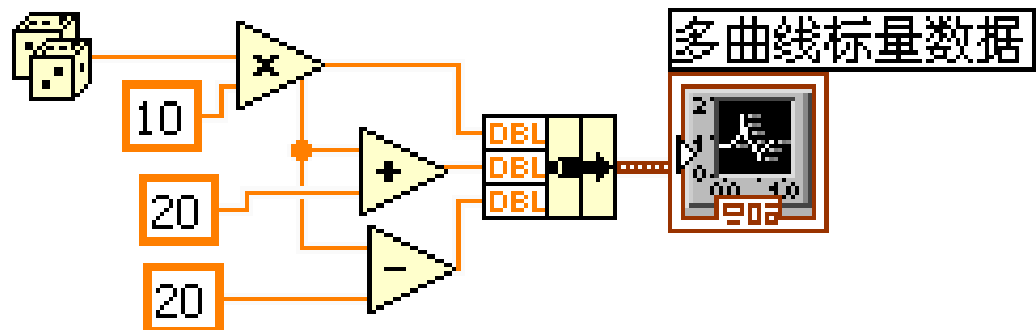
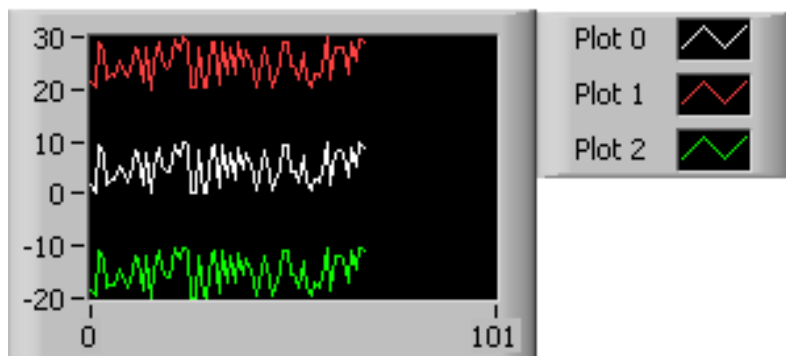
一次性把一维数组的数据添加在曲线末端，
即曲线每次向前推进的点数为一维数组数据的点数。



(3) 显示多条标量曲线

只需要用簇的Bundle函数将它们绑定在一起作为输入即可。

多曲线标量数据



缺省情况下是每一列的数据当作一条一维数组曲线。

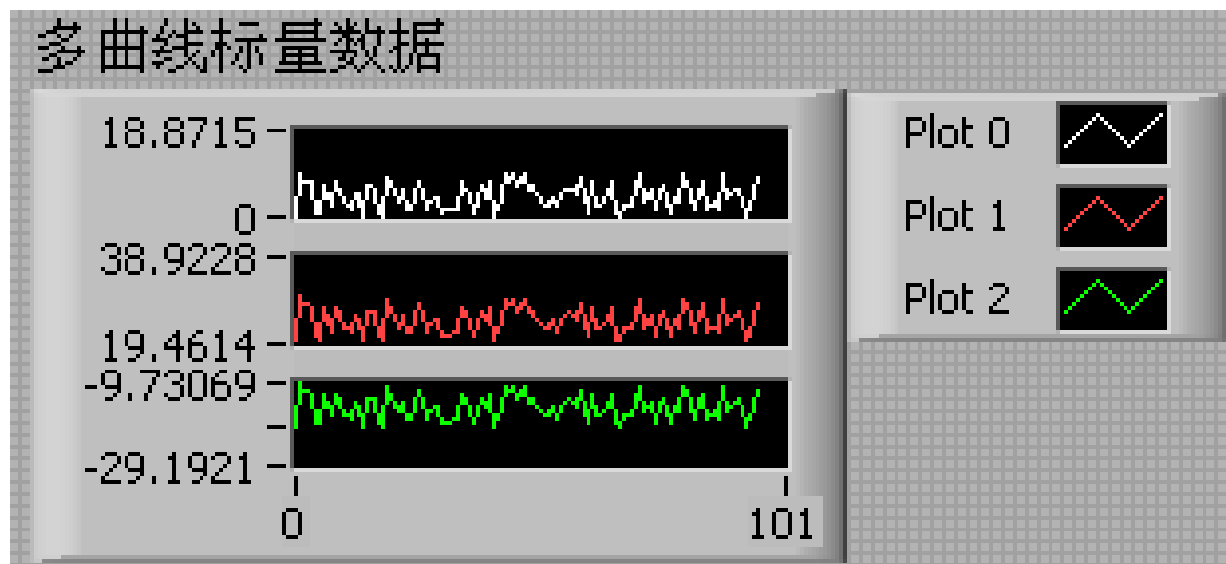


技 巧:

若想将曲线截图，可以通过右击曲线选择Data Operations->Copy Data将曲线图复制到剪切板上。

2. 定制Chart显示样式

(1) 分栏显示多条曲线



右击Chart选择Stack Plots

(2) 设置更新模式

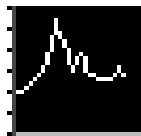
右击Chart选择Advanced->Update Mode...可以设置曲线的更新模式。

Strip Chart模式:



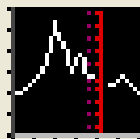
曲线填满显示区后曲线通过左移来更新曲线。

Scope Chart模式:



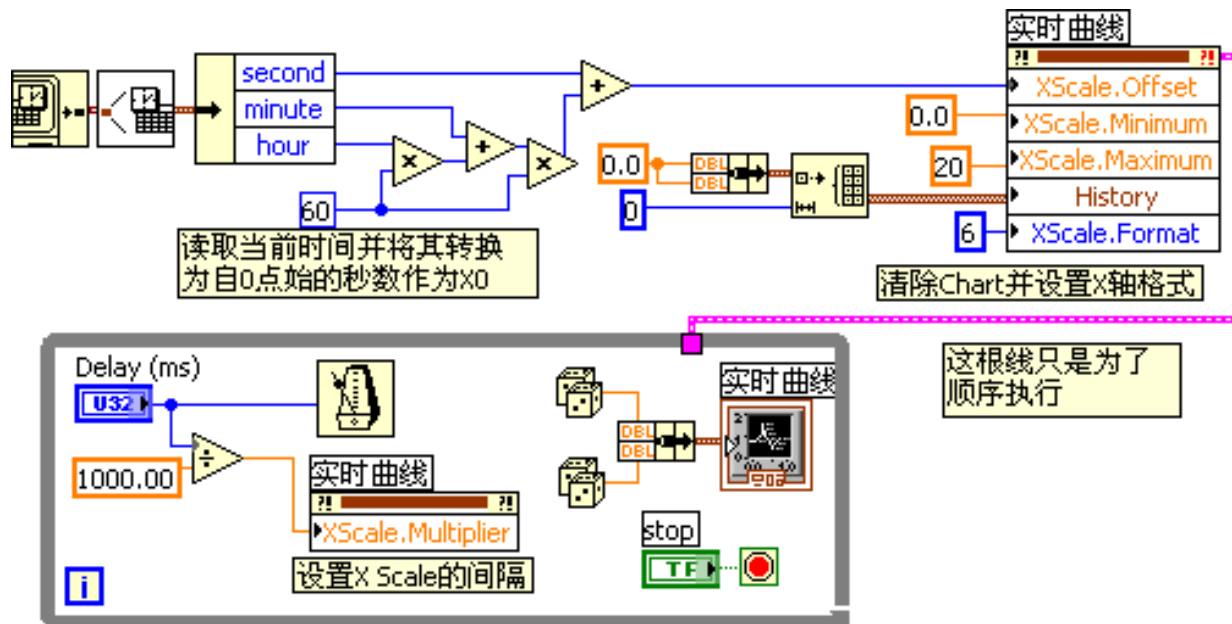
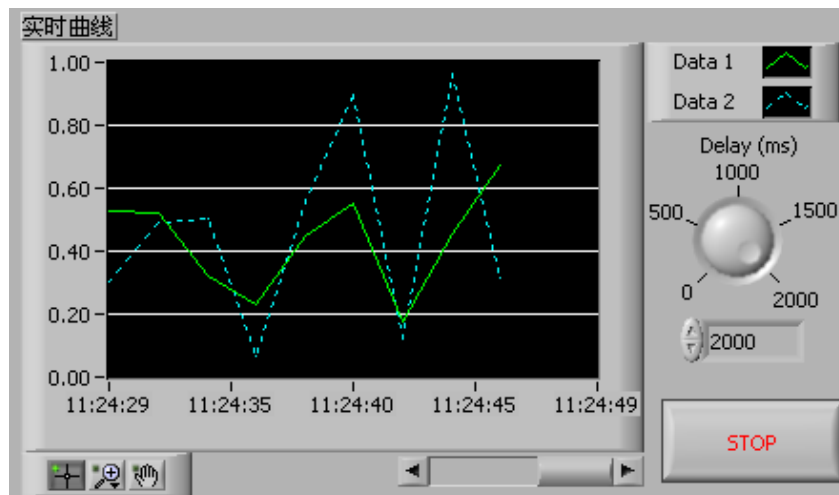
曲线填满显示区后直接清空显示区重新开始从左向右增长曲线。

Sweep Chart模式:



通过一条指示红线从左向右刷新数据。

3. 带时间轴的实时曲线



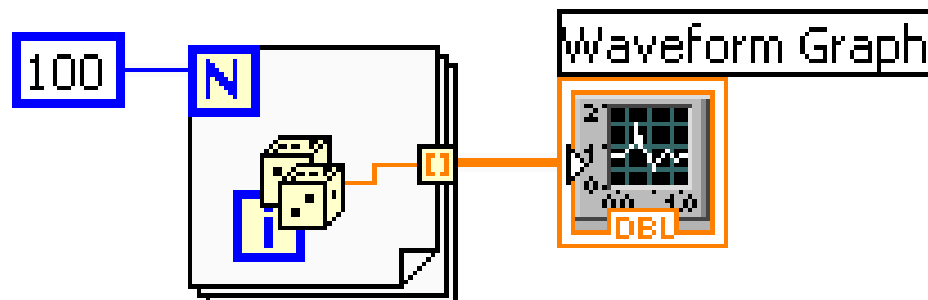
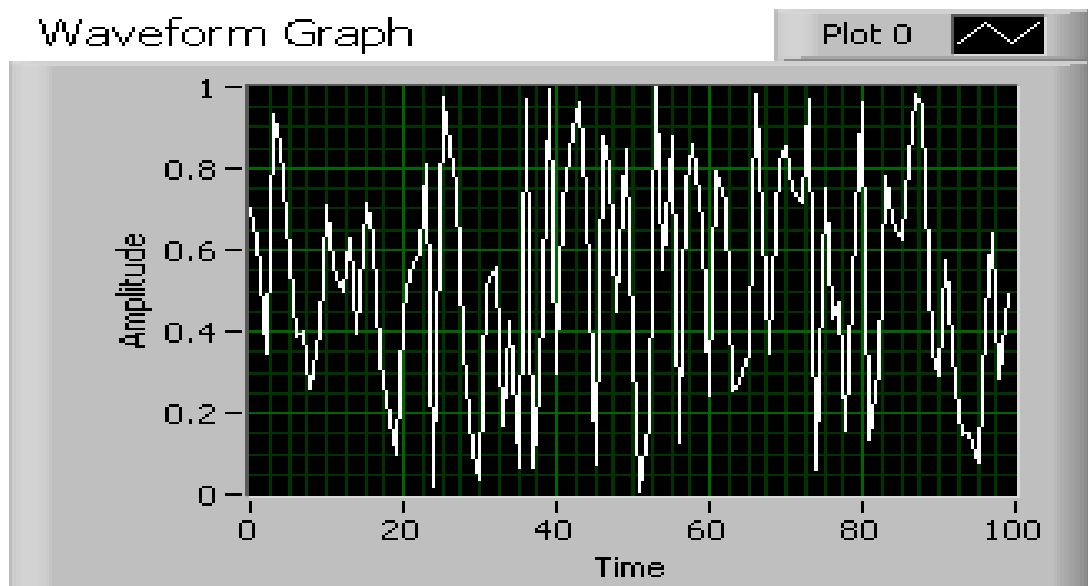
4.3 Graph图

Graph和Chart的**区别**在于Graph是一次性将现有数据绘图，在绘图之前先自动清空图表，而不会将新数据添加到曲线的尾端。

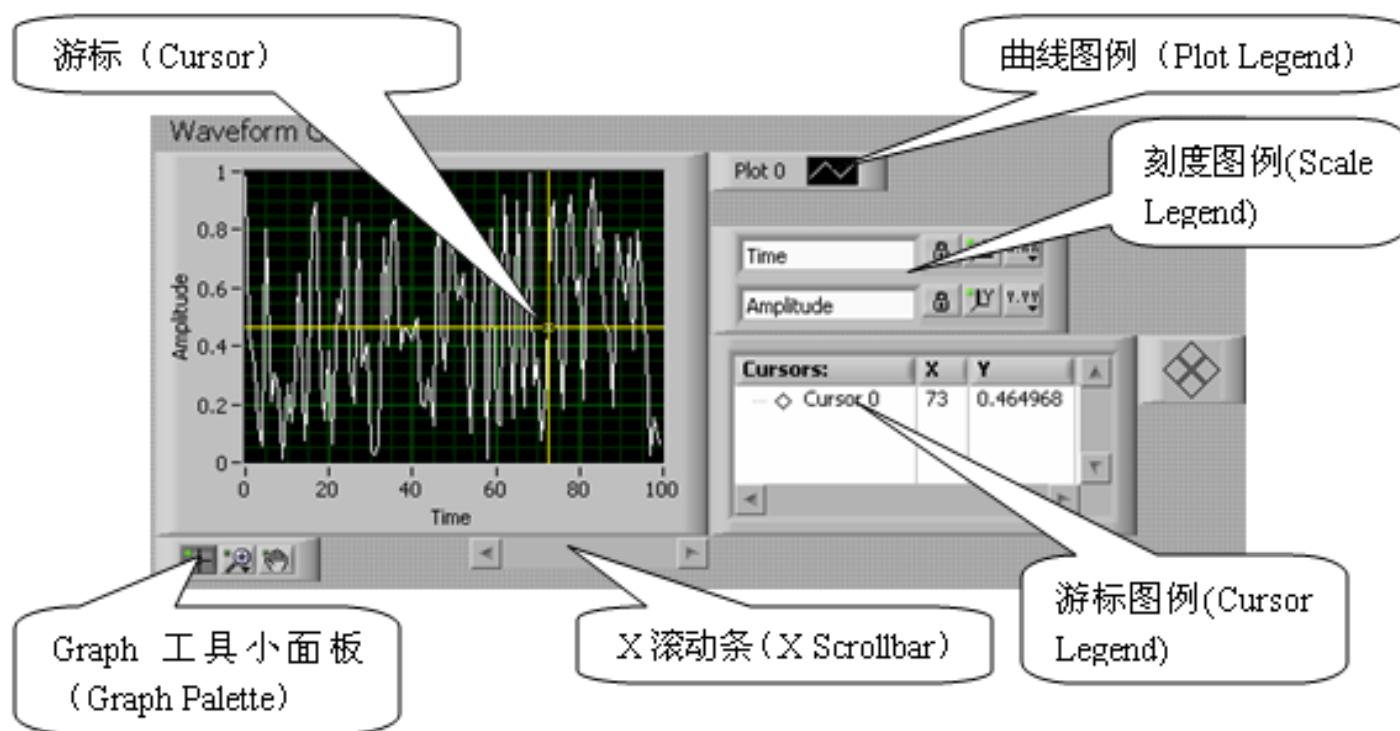
根据显示方法，Graph分为

- ✓ 波形图
- ✓ XY曲线图
- ✓ 强度图
- ✓ 数字时序图
- ✓ 三维图

1. 定制Graph属性



右击Graph控件可以看到该控件有很多属性可以设置。选择Visible Items...可以看到关于该控件的很多辅助选项。选择这些选项可以使它们都可见，如下图所示。



2. Waveform Graph

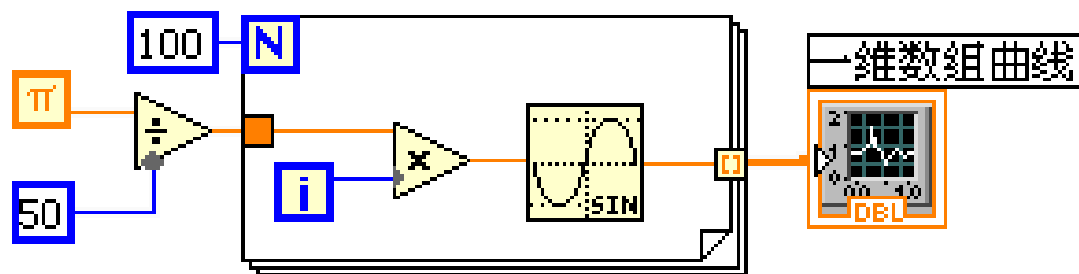
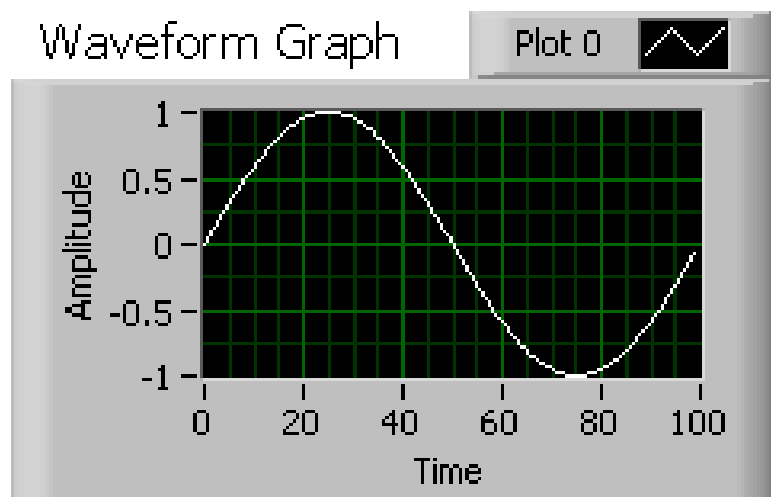
有多种数据输入类型：

- ✓ 一维数组
- ✓ 二维数组
- ✓ 簇
- ✓ 簇数组
- ✓ 波形数据

Waveform Graph在Controls Palette中的位置为Modern->Graph->Waveform Graph。

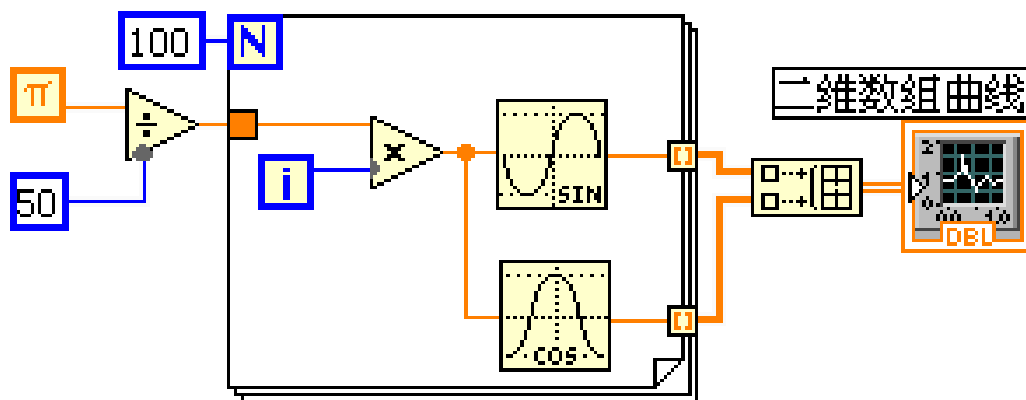
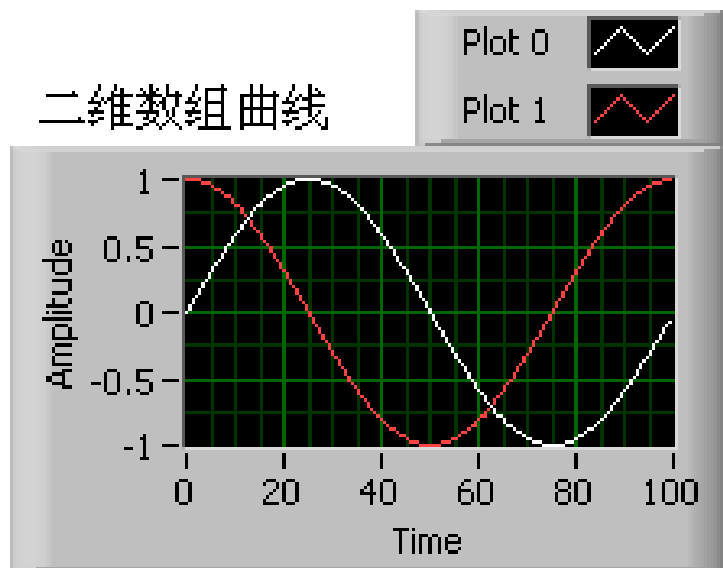
(1) 一维数组作为输入

Waveform Graph直接将一维数组画成一条曲线，纵坐标为数组元素的值，横坐标为数组索引。



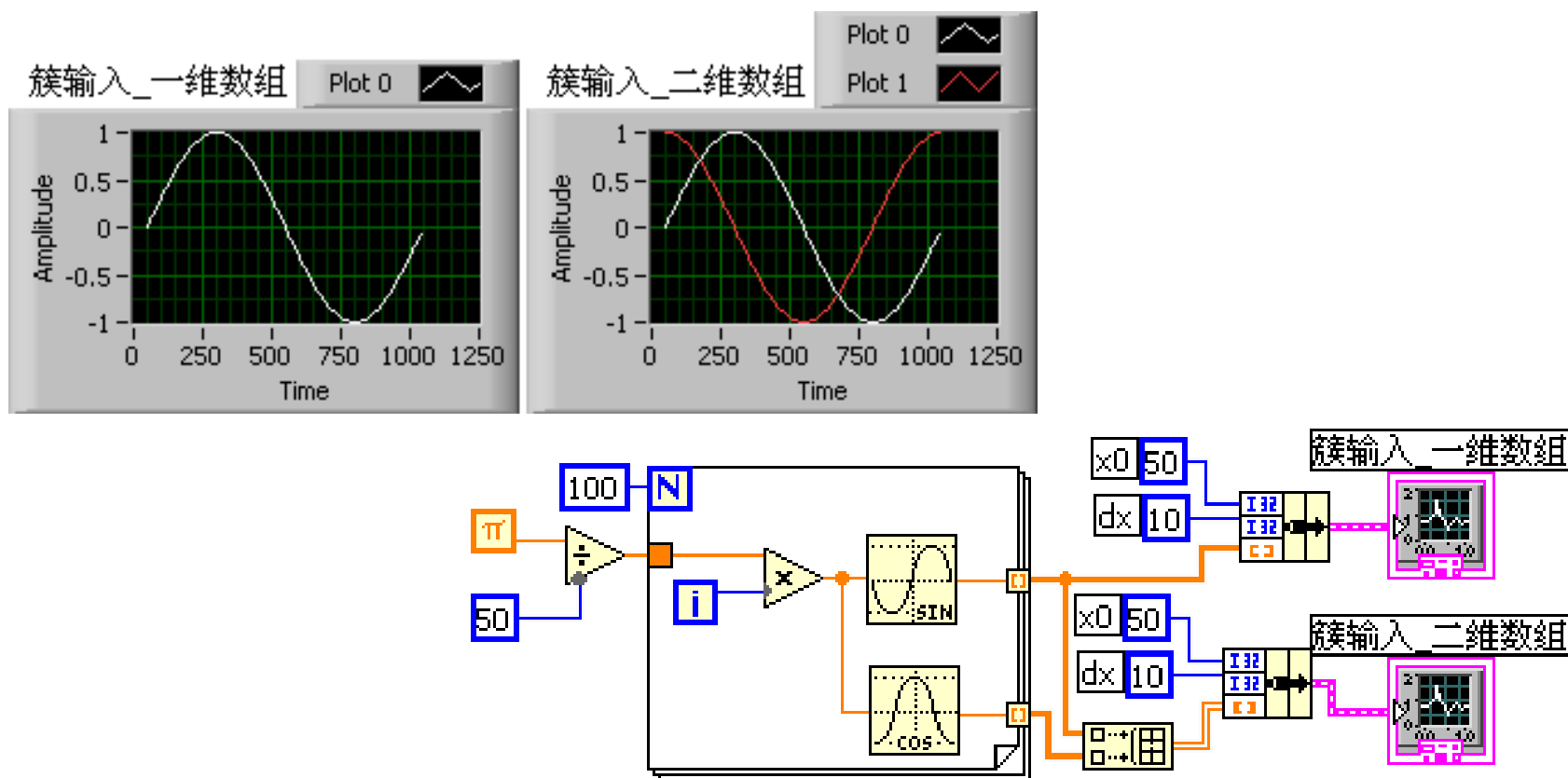
(2) 二维数组作为输入

缺省情况下每一行的数据对应一条曲线，
即曲线的数目和行数相同。



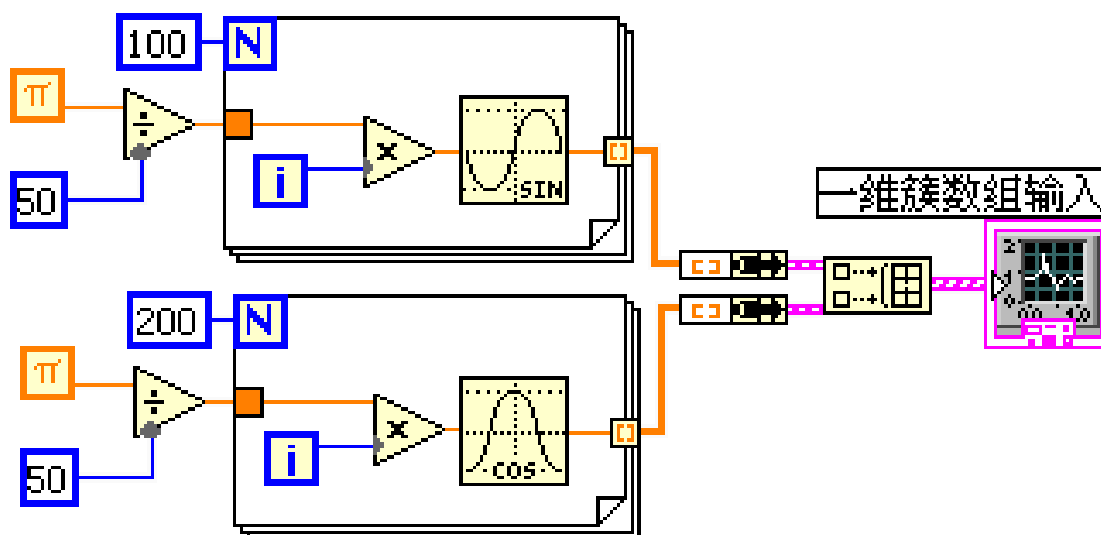
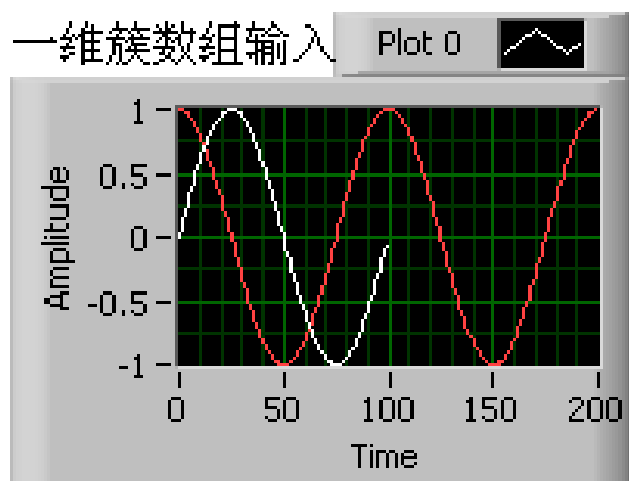
(3) 簇作为输入

簇作为输入时需要指定三个元素：起始位置
 x_0 、数据点间隔 dx 和数组数据。

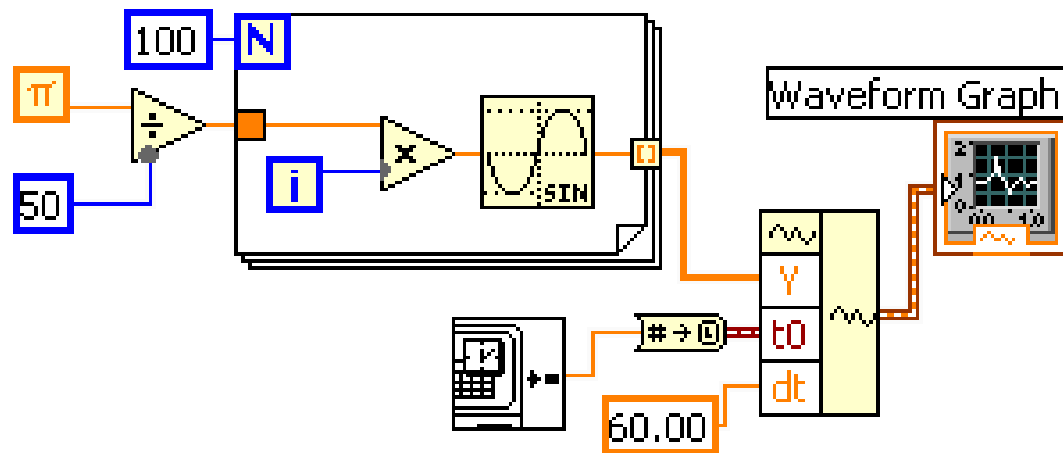


(4) 簇数组作为输入

一维簇数组也可以直接作为Graph的输入，此时相当于 x_0 为0， dx 为1。



由于波形数据所携带的数据横轴为时间，因此需要将Waveform Graph的横轴设为时间轴。



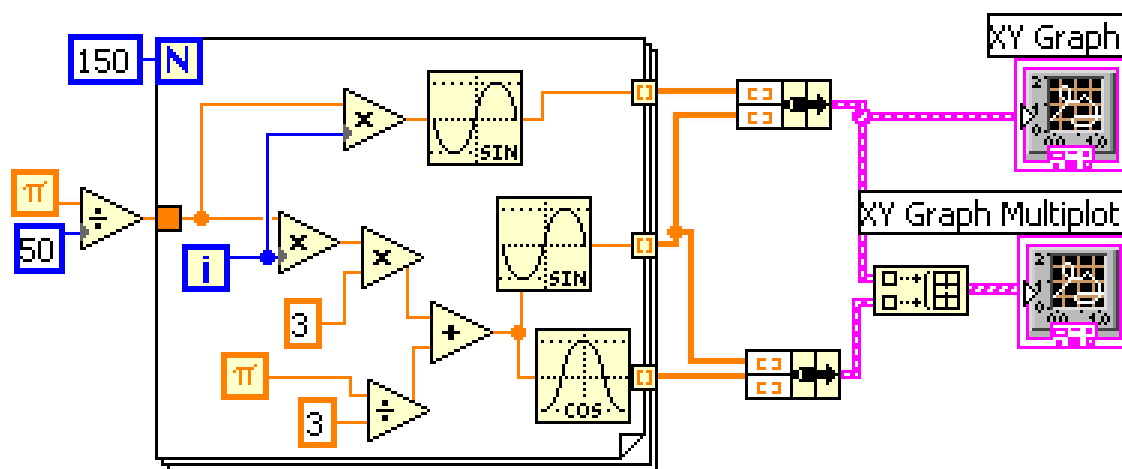
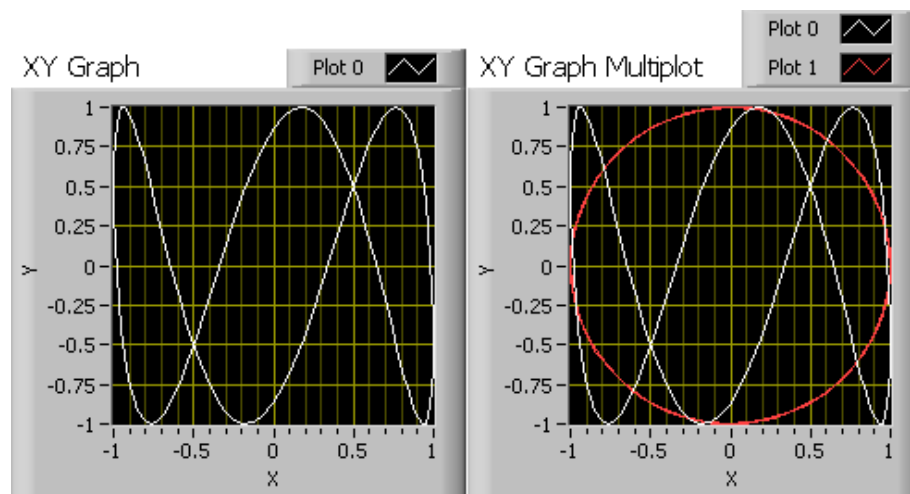
3. XY Graph

当我们需要画的曲线是由 (x, y) 坐标决定的时候，我们就需要采用XY Graph。

其实Waveform Graph在一定意义上也是XY Graph，但是它的X轴必须是等间距的，而且不可控制。

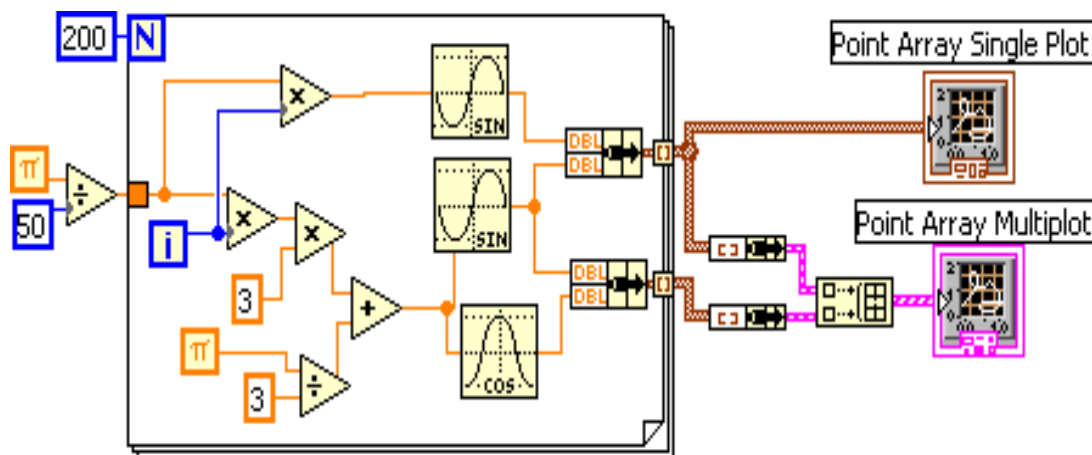
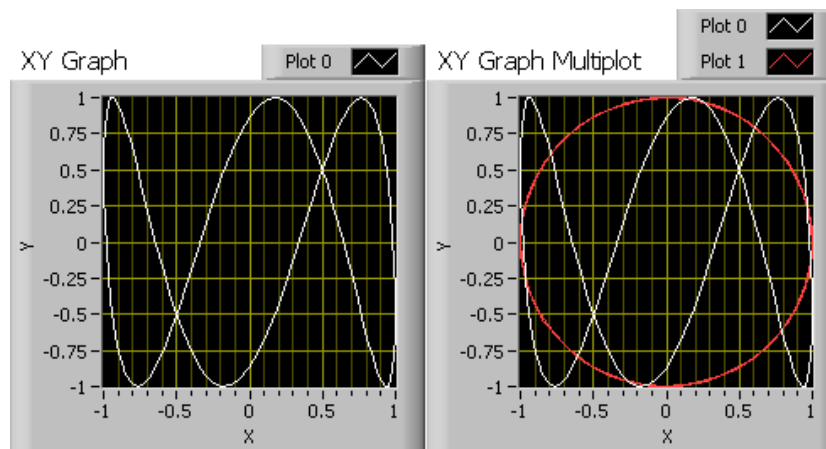
(1) XY两个一维数组绑定为簇作为输入

Bundle函数的输入的第一个数组为X Array，第二个数组为Y Array。绑定为簇后可以直接输入，也可以将多个簇Build为一维数组输入实现多条曲线。

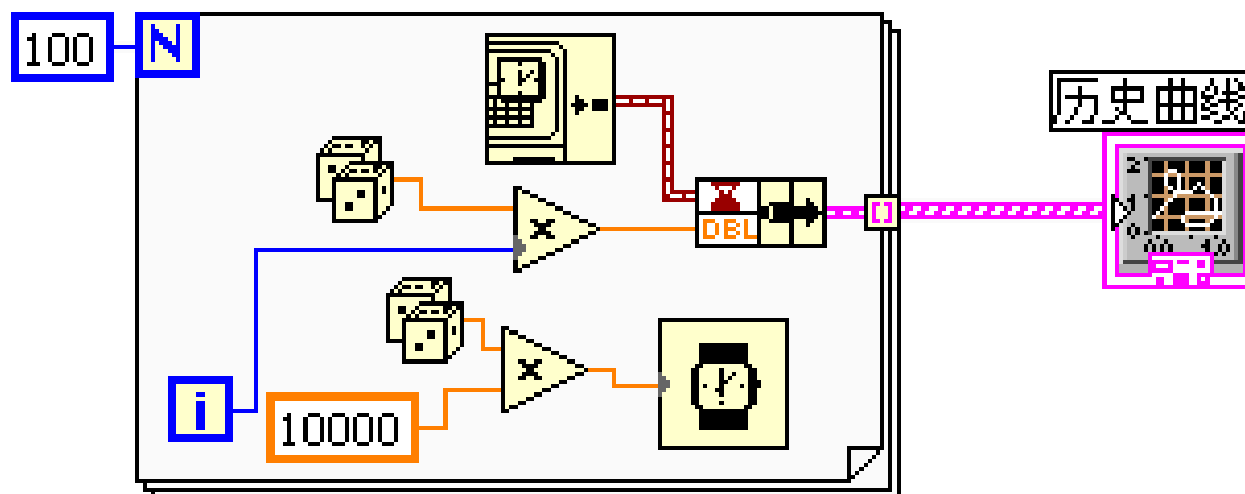
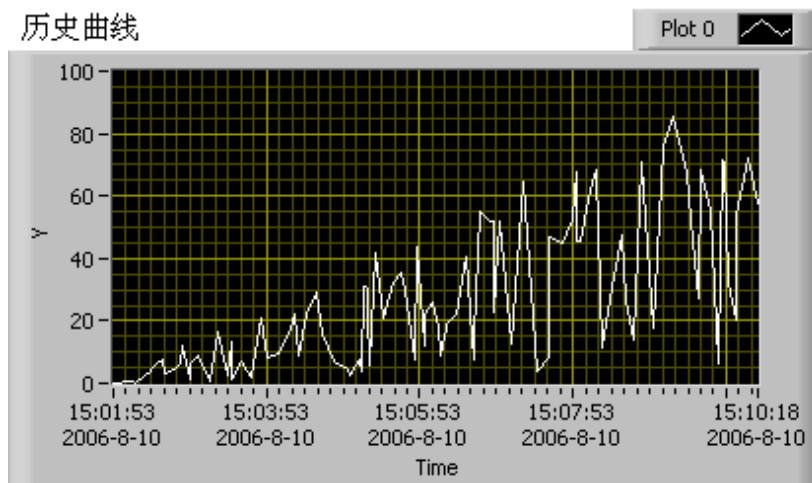


(2) 坐标点簇数组作为输入

将各个点的坐标绑定为簇然后作为簇数组输入，和直接将XY数组绑定为簇输入效果一样。但是后者不一定需要XY数组的大小一致，它会自动将大的数组裁剪。

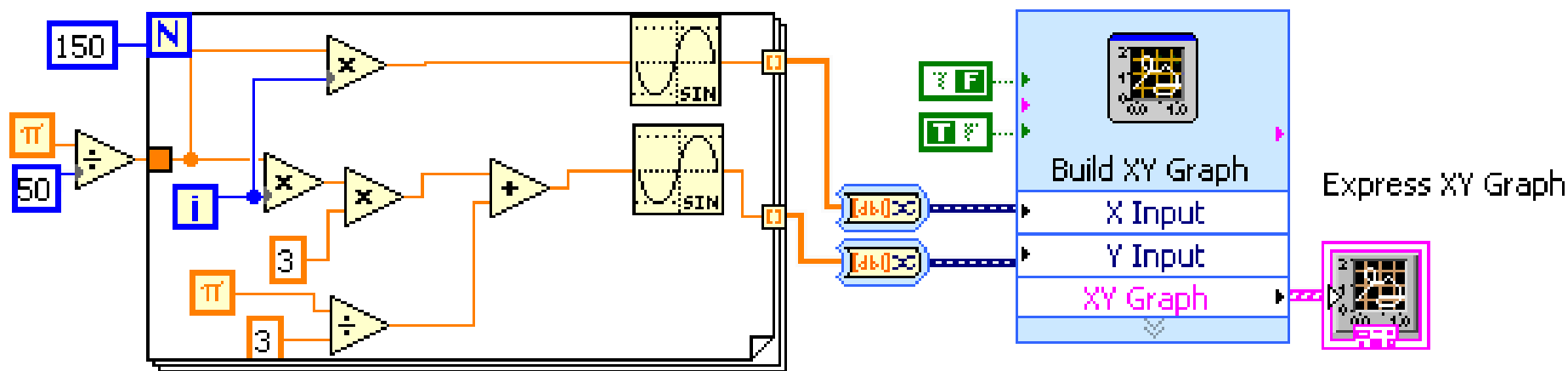


(3) 时间作为X轴——利用XY Graph实现历史曲线

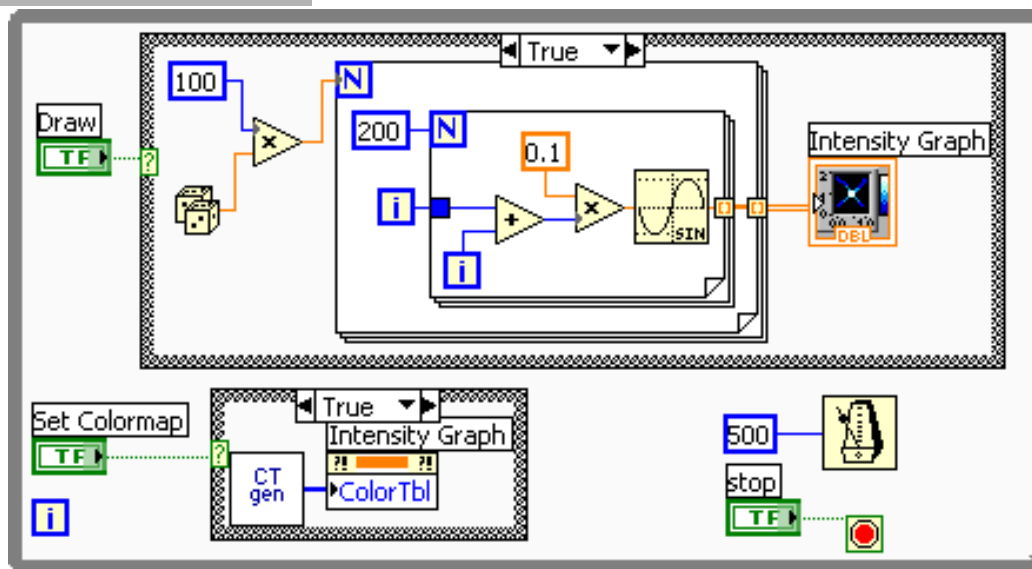
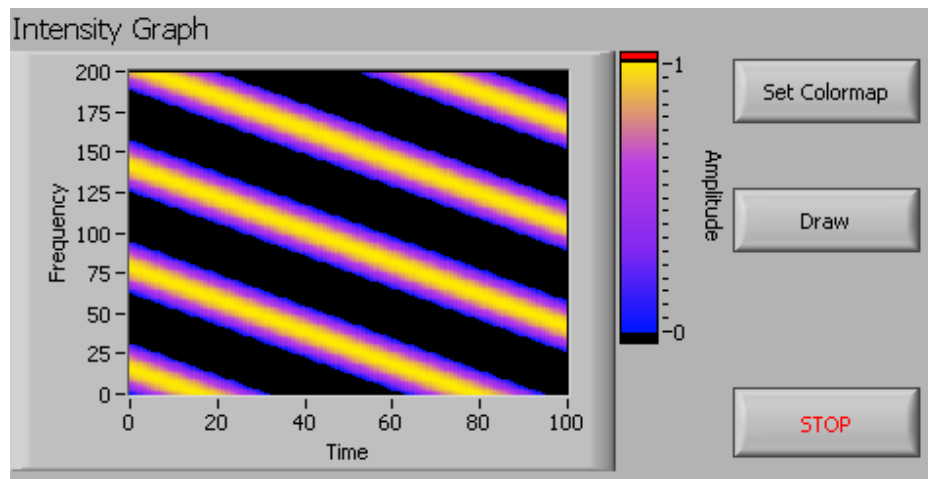


(4) Express XY Graph

Express XY Graph采用了LabVIEW的Express技术，将Express XY Graph放置在前面板上的同时，在程序框图中会自动添加一个VI，它的XY轴数据为动态数据类型。



4. 强度图和强度图表(Intensity Graph & Chart)

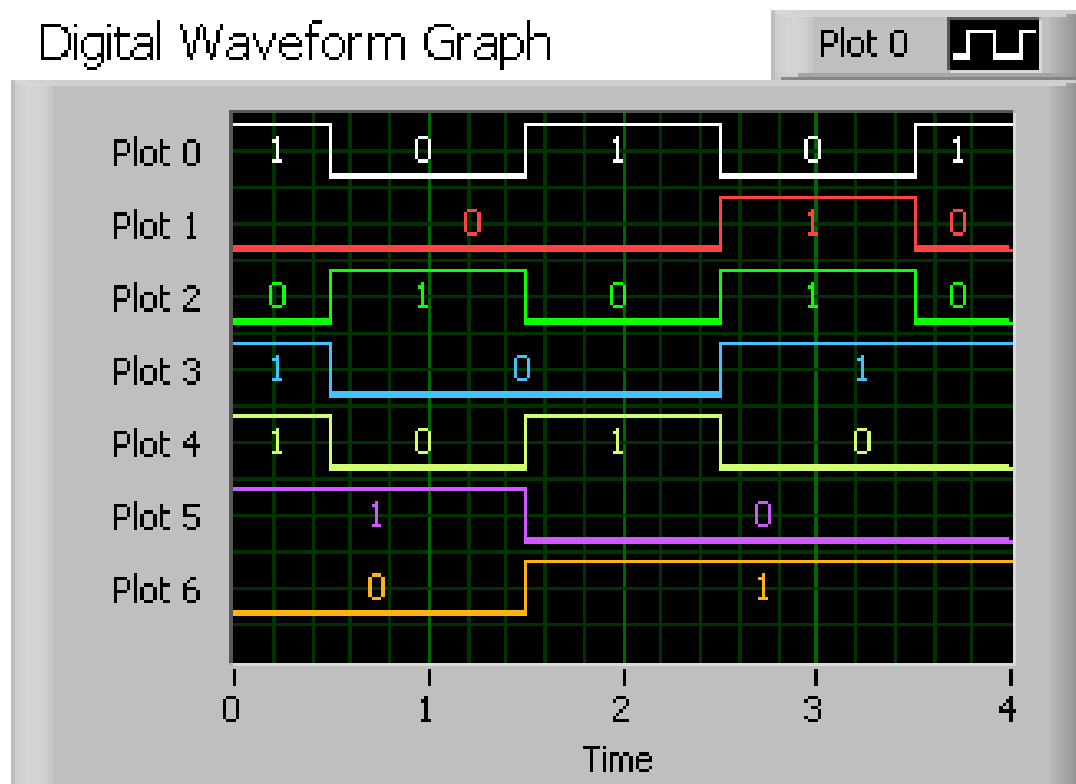


5. 数字波形图(Digital Waveform Graph)

Digital Data

	4	0
0	011	1001
1	010	0100
2	101	0001
3	100	1110
4	100	1001

Digital Waveform Graph

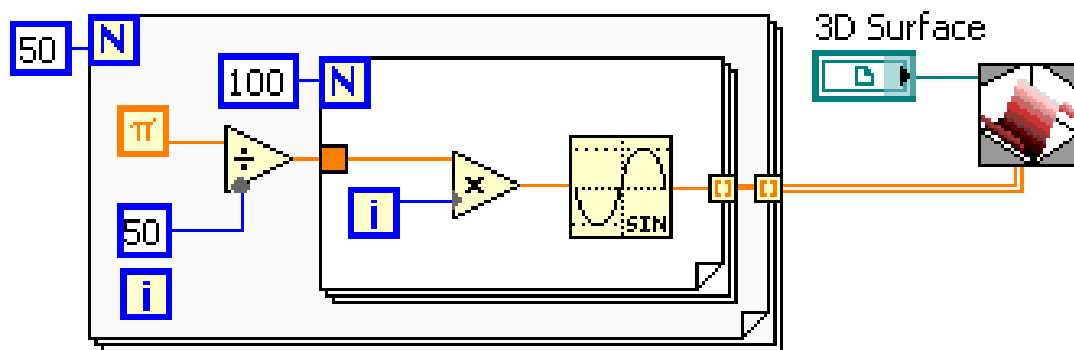
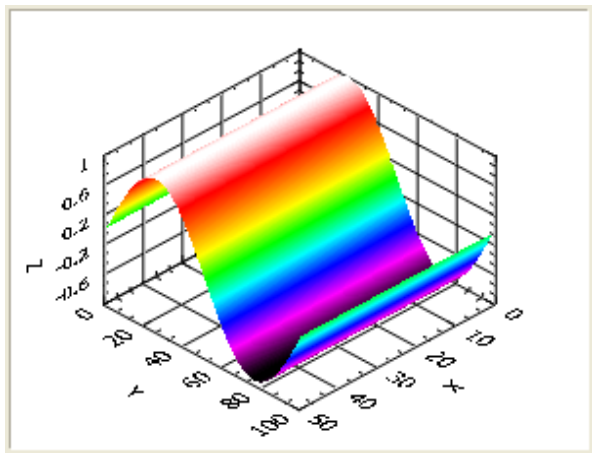


4.4 三维图形 (3D Graph)

1. 三维曲面图 (3D Surface Graph)



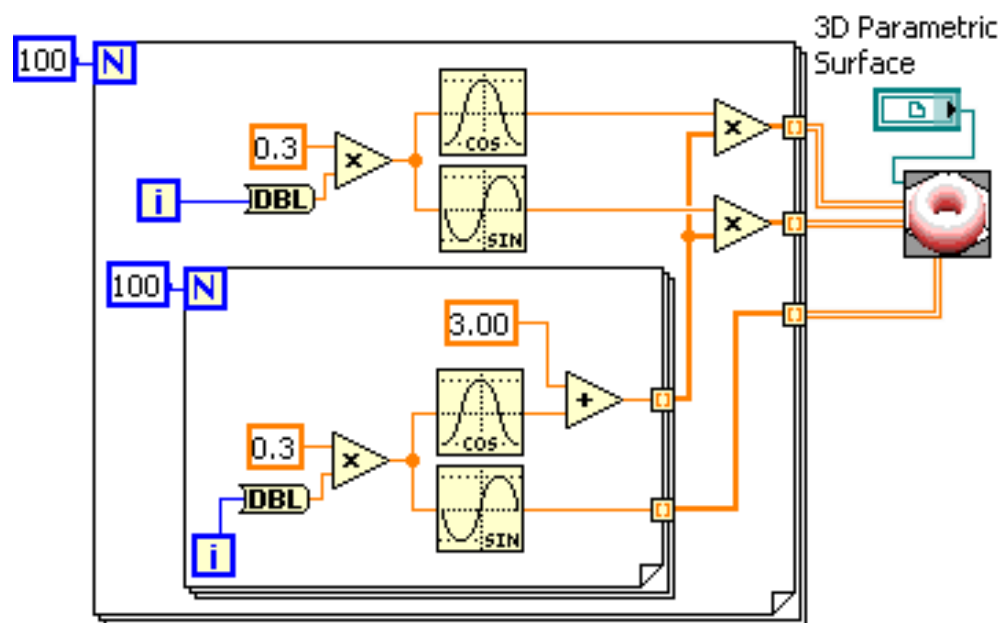
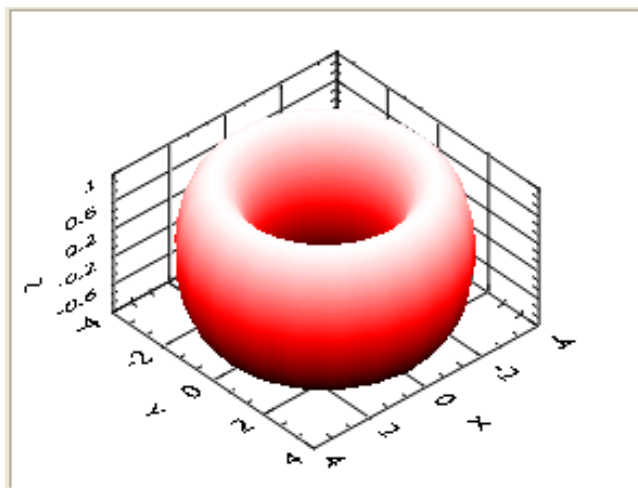
将该控件放置在前面板上的同时，在背面板也会同时出现一个ActiveX控件和一个VI函数3D Surface.vi。



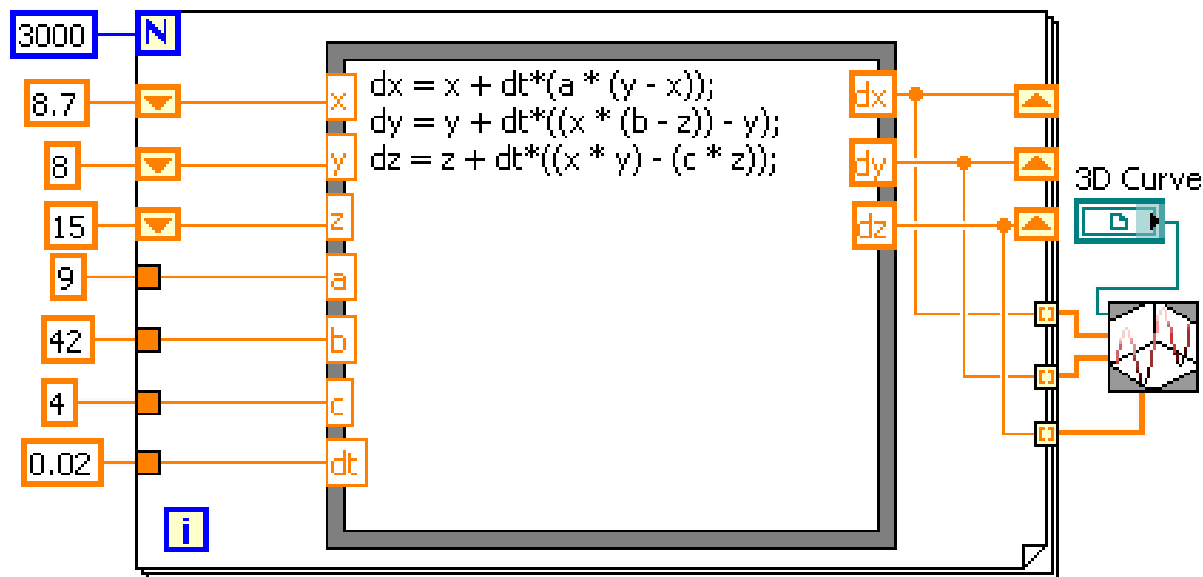
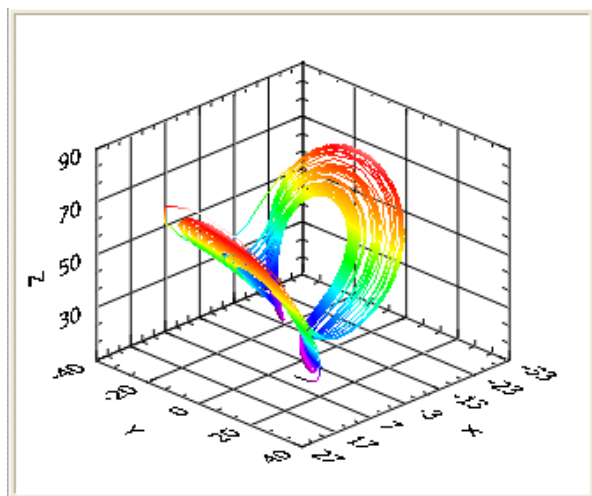
2. 三维参量曲面图 (3D Parametric Graph)



3个轴的数据均为二维数组，分别决定了相对于x平面、y平面和z平面的曲面。

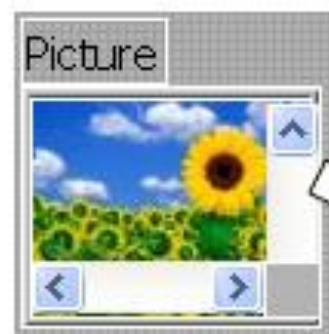


3. 三维曲线图 (3D Curve Graph)



4.5 Picture图形控件

1. 向Picture控件导入图片



2. 利用Picture控件画图

Picture

