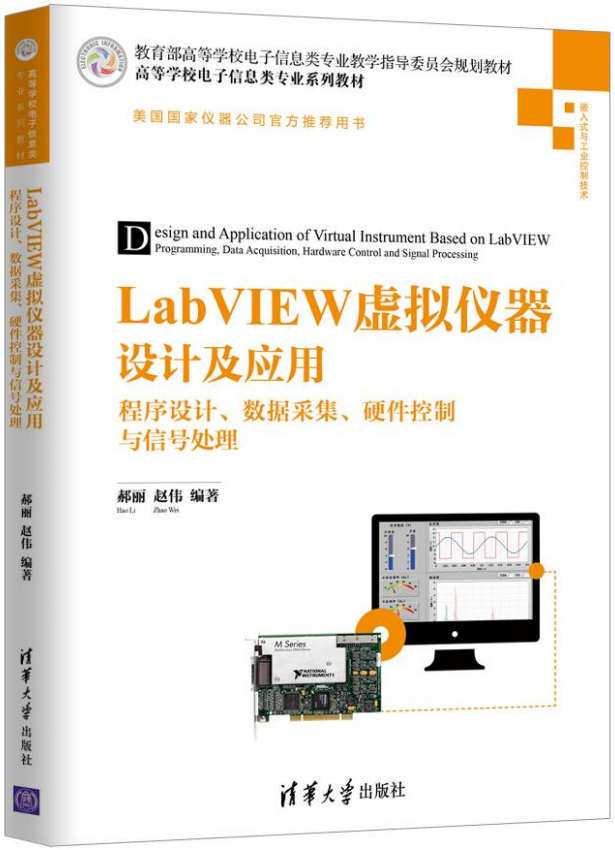


入选教育部高等学校电子信息类专业教学指导委员会规划教材

美国国家仪器公司官方推荐用书

ISBN 978-7-302-50651-5

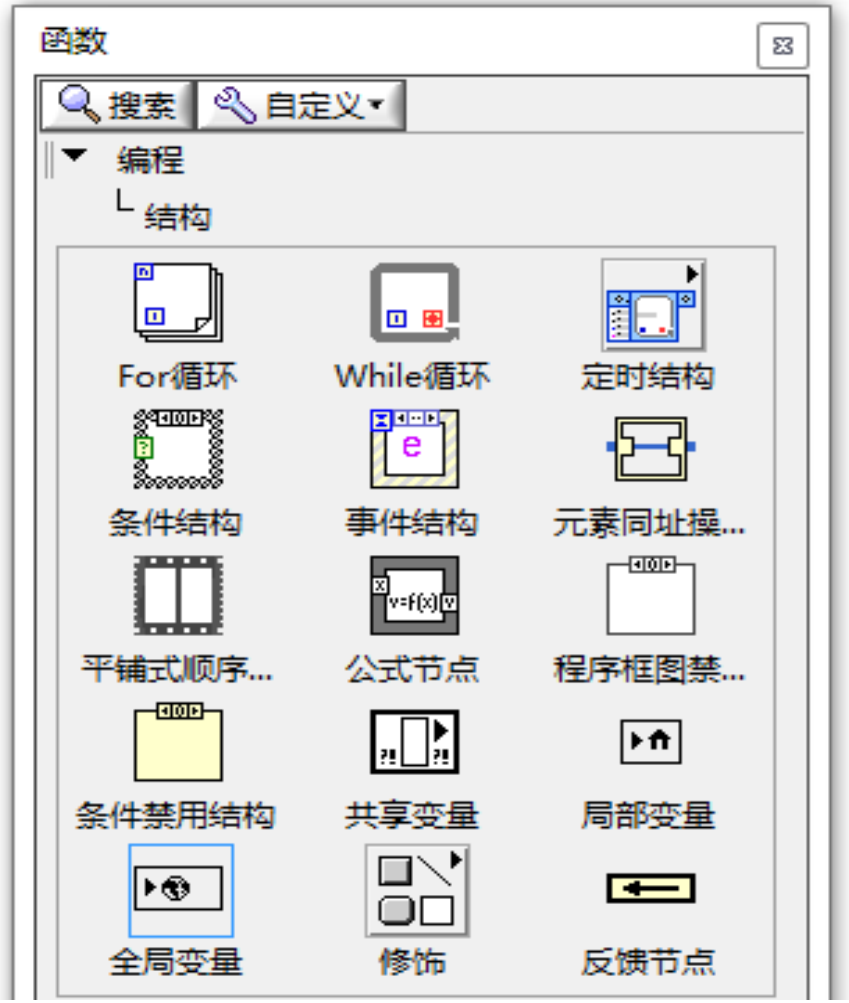
qq交流群:565138476

第4节 程序结构

2018 11

程序结构

- 1. 顺序结构
- 2. 条件结构
- 3. 循环结构
- 4. 事件结构
- 5. 公式节点



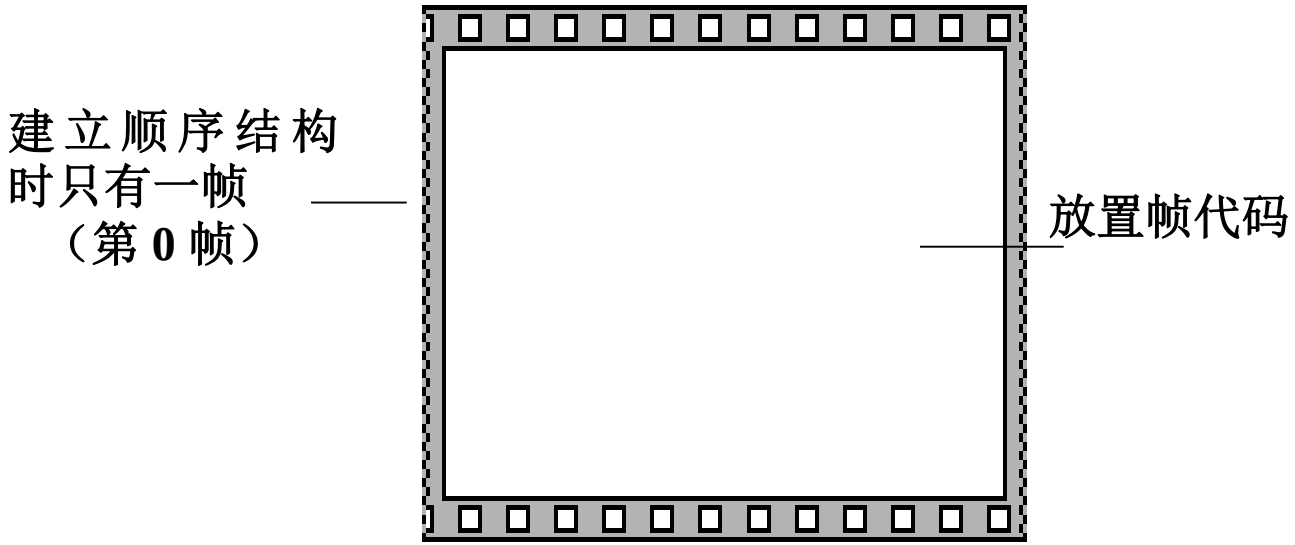
路径：函数选板-> 编程 -> 结构

1. 顺序结构

- **数据流**
  - ✓ 天生并行
  - ✓ 如果想要严格控制程序代码执行的先后，该如何实现呢?

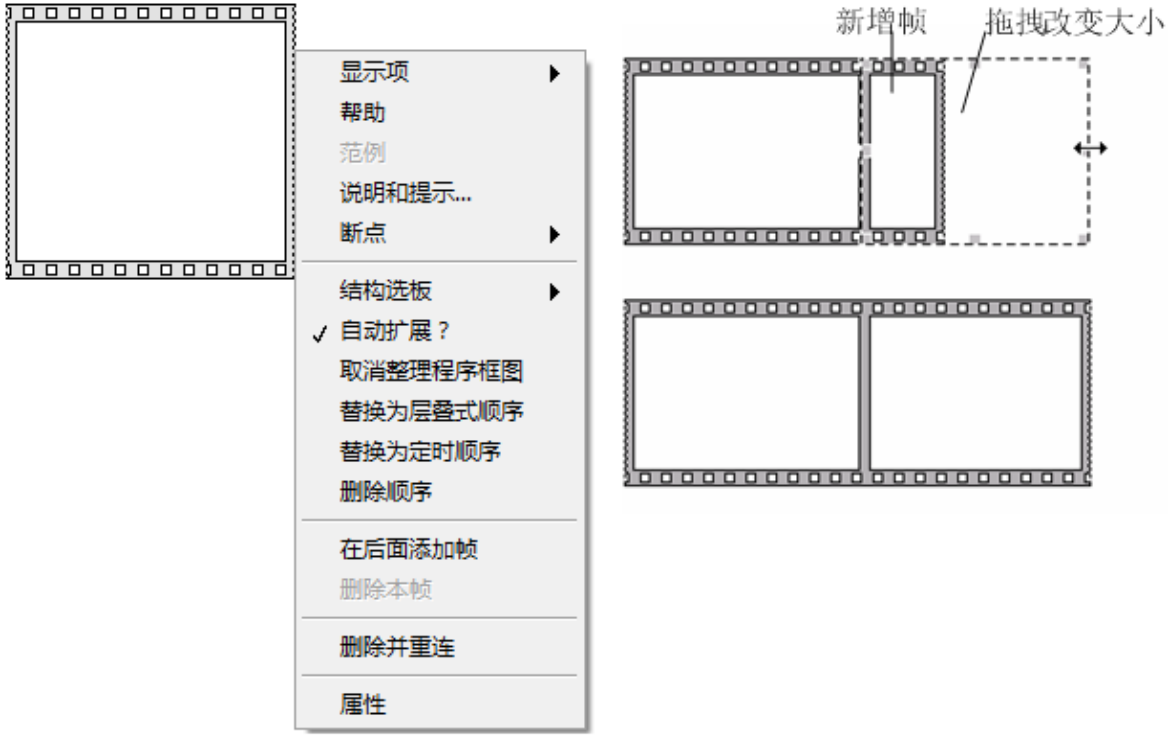
1. 顺序结构

- 创建
  - ✓选中顺序结构，将其拖到程序框图上
- 分类
  - ✓ 层叠式顺序结构、平铺式顺序结构



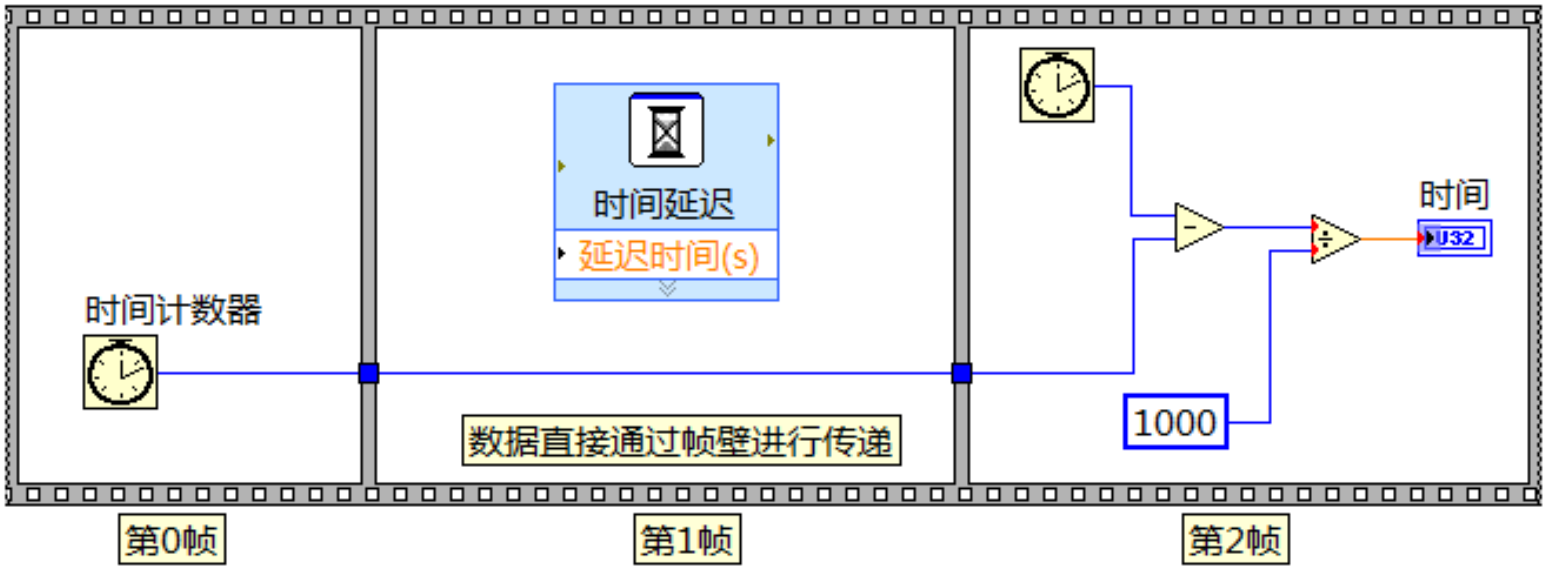
1. 顺序结构

- 添加新帧
  - ✓选中边框，右击鼠标右键，弹出快捷菜单，可在当前桢前后增加桢



1. 顺序结构

- 各个帧之间的数据传递
  - ✓通过连线直接穿过桢壁进行传递
- 例1 计算程序运行的时间
  - ✓ 平铺式： 一目了然

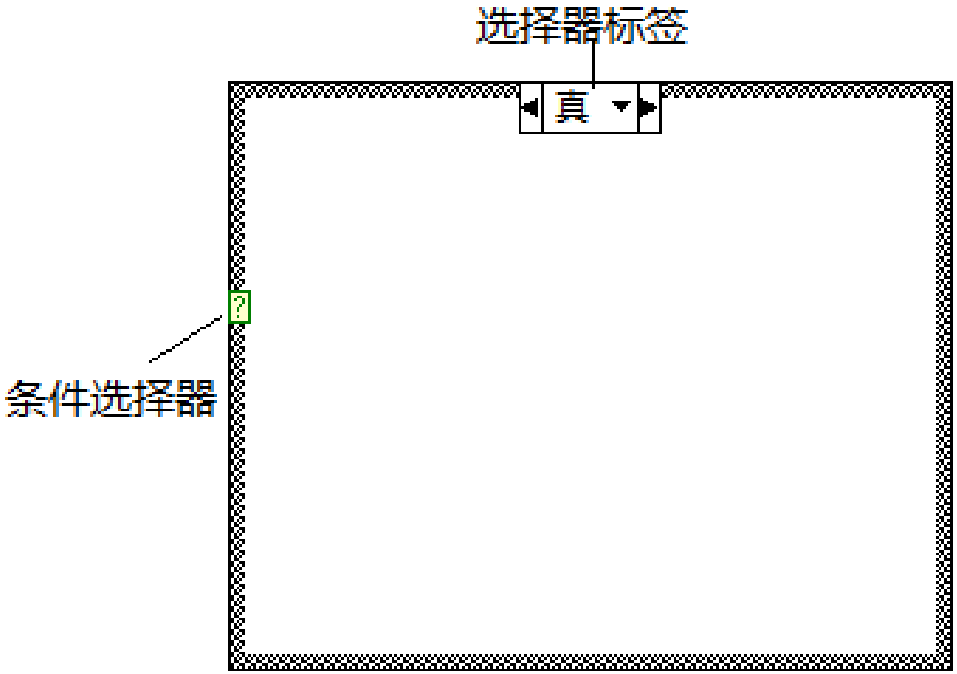


1.顺序结构

- 例1 计算程序运行的时间
  - ✓ 层叠式： 结构紧凑
  - ✓ 层叠式与平铺式可以相互转换

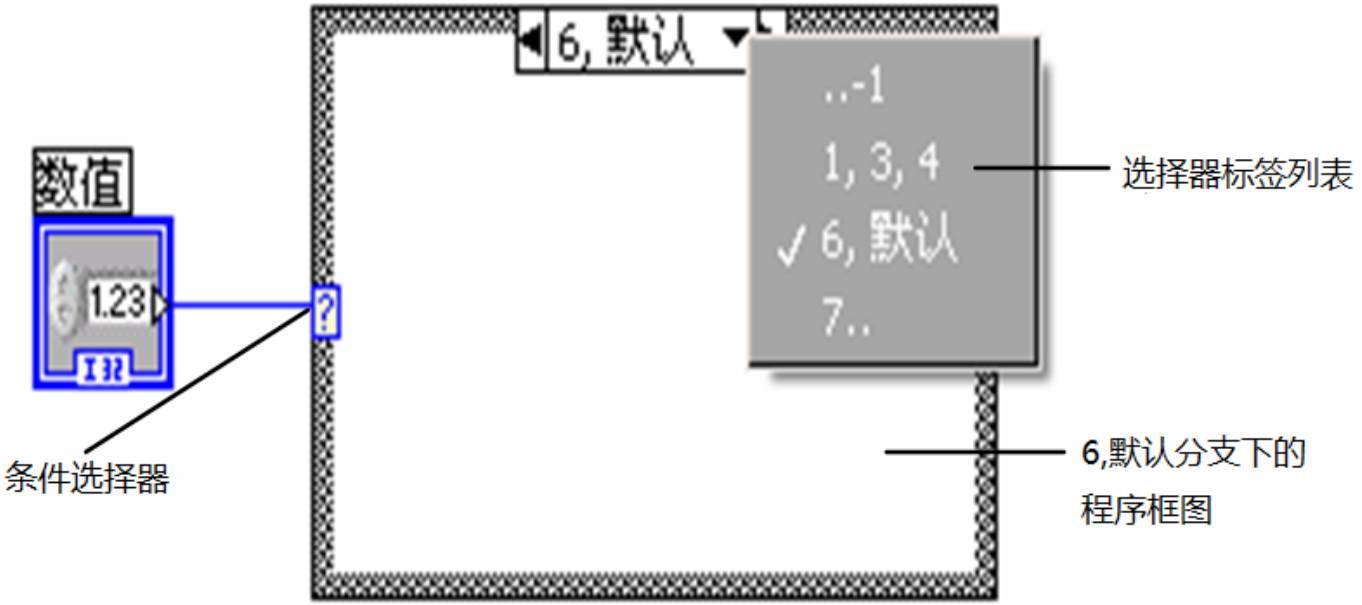
2. 条件结构

- 分支选择器为布尔型
  - ✓真、假两个状态
  - ✓IF语句



2 条件结构

- 分支选择器为非布尔型
  - ✓多分支、switch语句
  - ✓要么在分支选择器标签中列出所有可能的情况；要么必须给出一种缺省(默认)情况

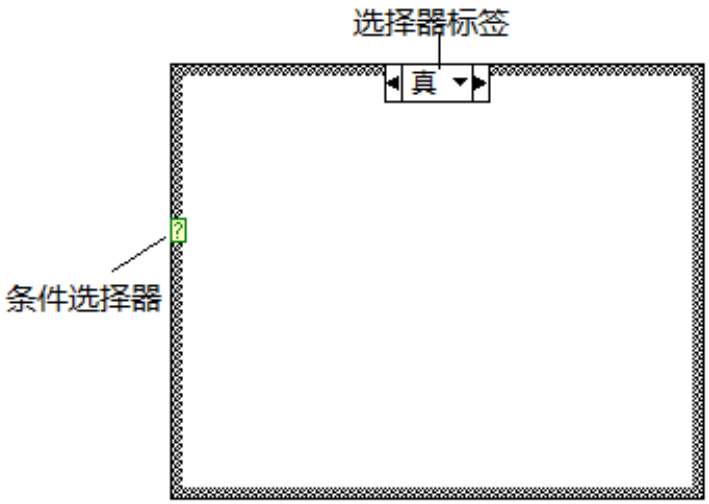


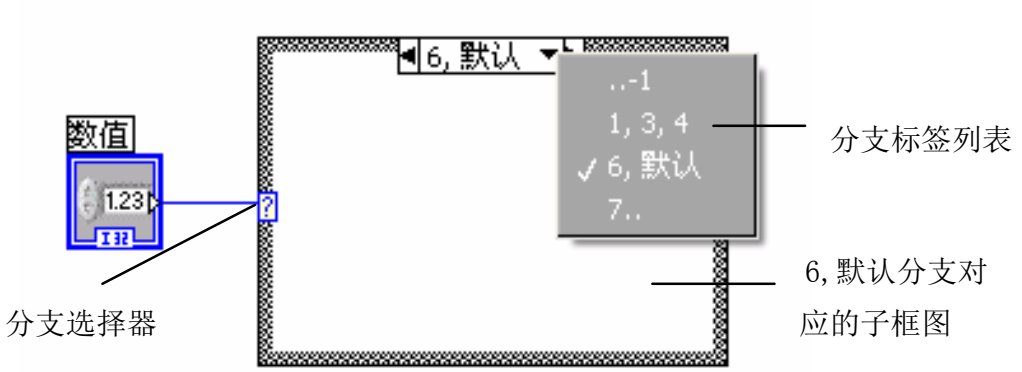
2 条件结构

注意事项：

选择器端子默认为布尔型的，所以上端的增量、减量按钮默认的是真或假。

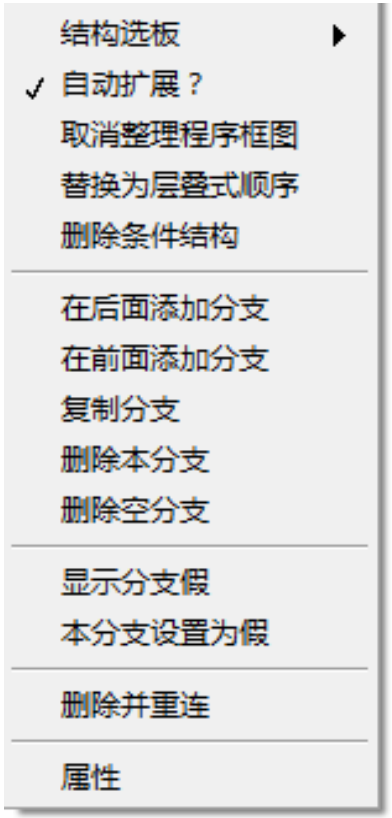
当你在选择器端子上接入一个数值型数据时，增减量按钮会自动变成数值型，这时你可以根据实际需求设置不同的分支。





2 条件结构

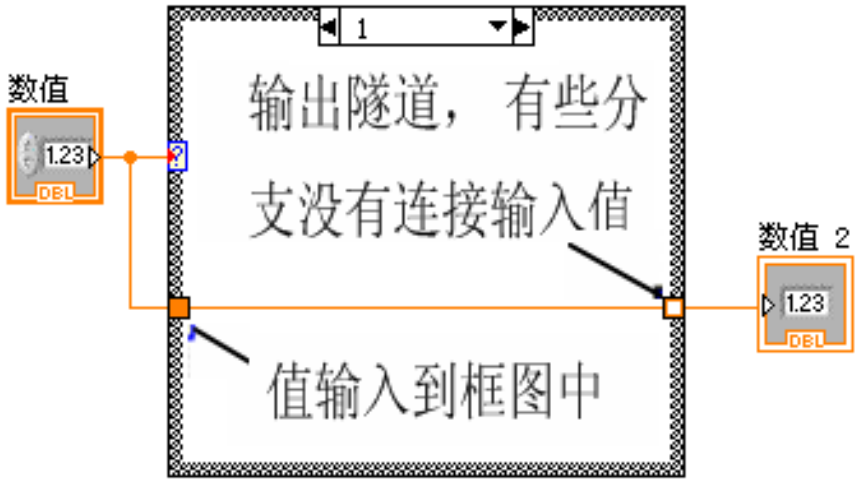
- 条件结构也称分支结构，其快捷菜单在其边框上任意处右击鼠标即弹出；
- 不同分支结构的基本操作相同，部分有关选项如下：

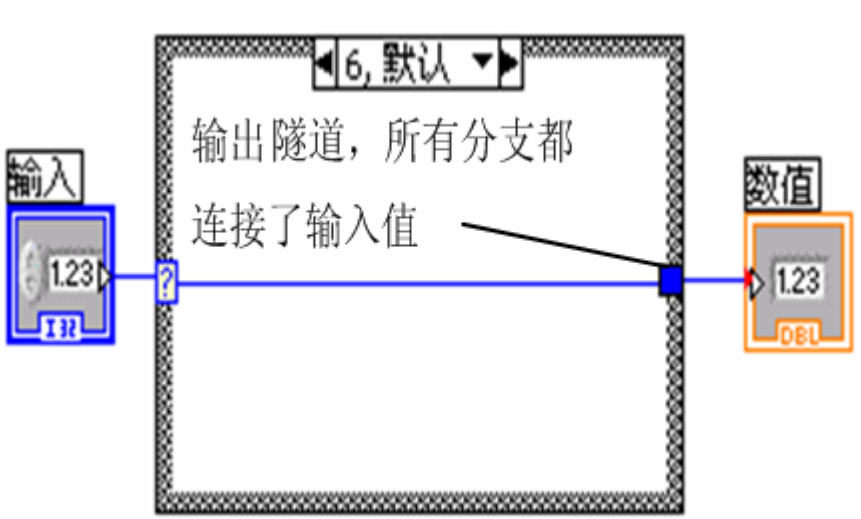


2.条件结构

- 向条件结构内引入连线，或从其内部向外引出连线时，会在其边框上生成隧道；
- 输入隧道在每一个分支中都可以使用；
- 输出隧道必须从每一个分支都得到明确的输入值，否则程序无法运行。

如何实现无else的if语句？



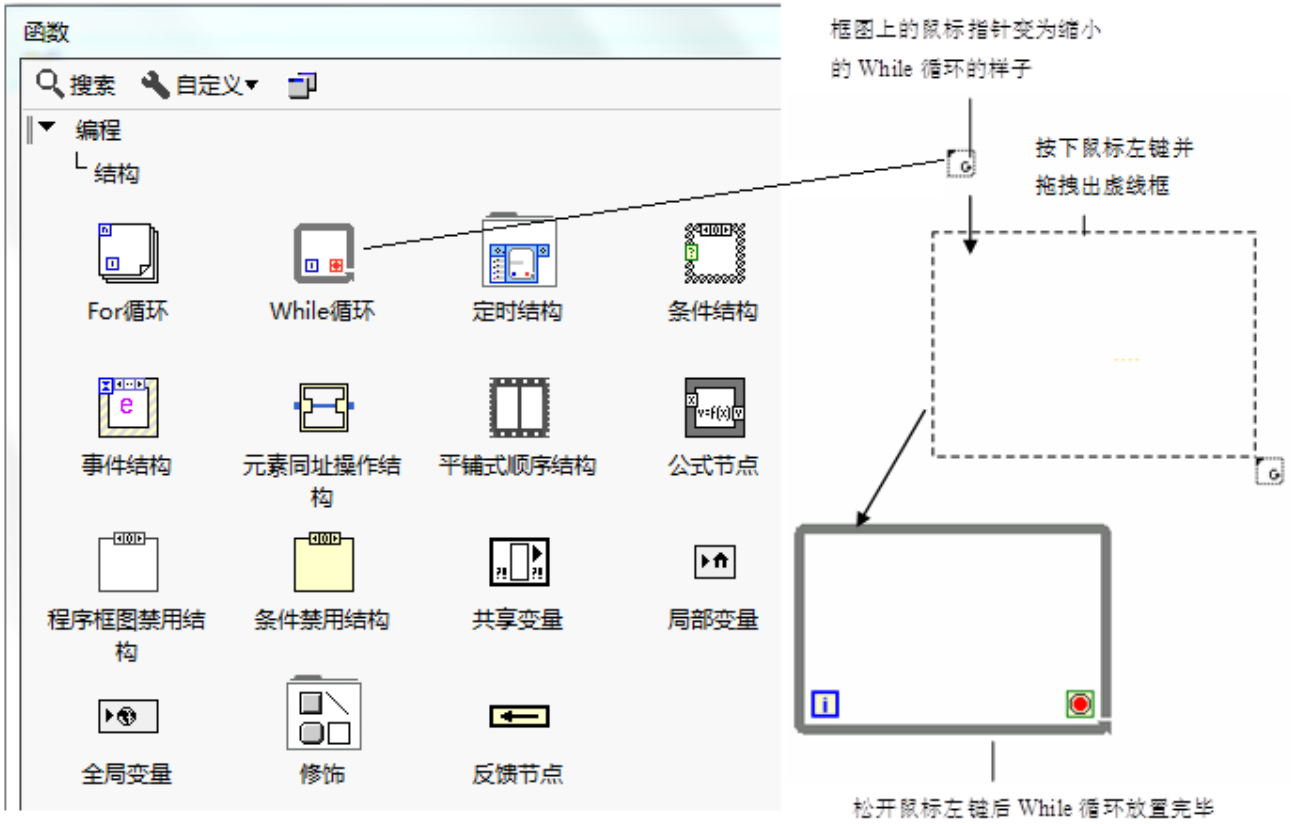


3. 循环结构

- 3.1 While循环
- 3.2 For循环
- 3.3 循环结构内外数据的交换
- 3.4 自动索引
- 3.5 移位寄存器
- 3.6 反馈节点

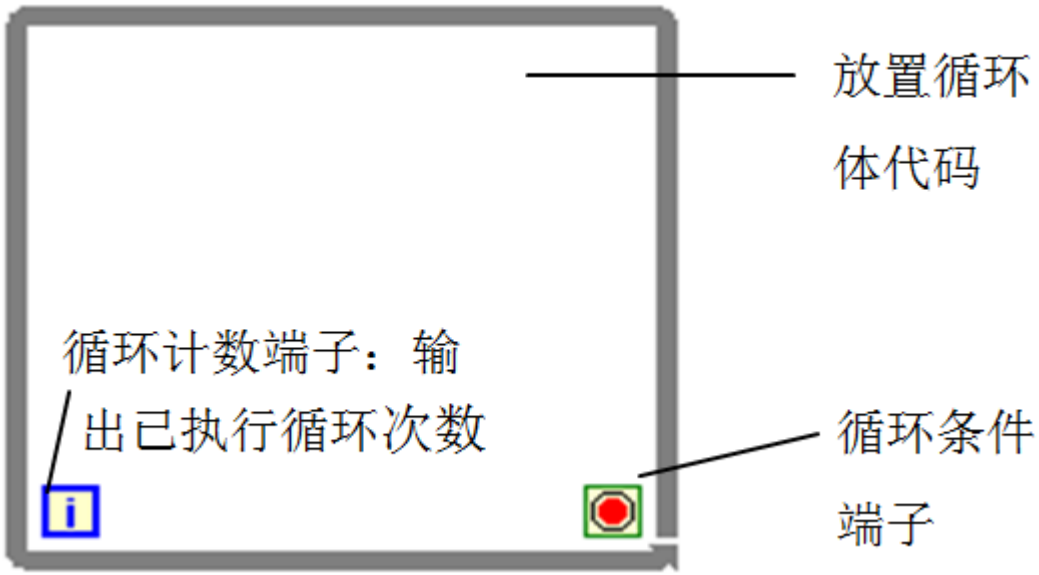
3.1 While循环

- 创建
  - ✓在建立While循环后，向其中添加图形化程序代码。
  - ✓在已有程序外建立While循环（框住程序代码）。



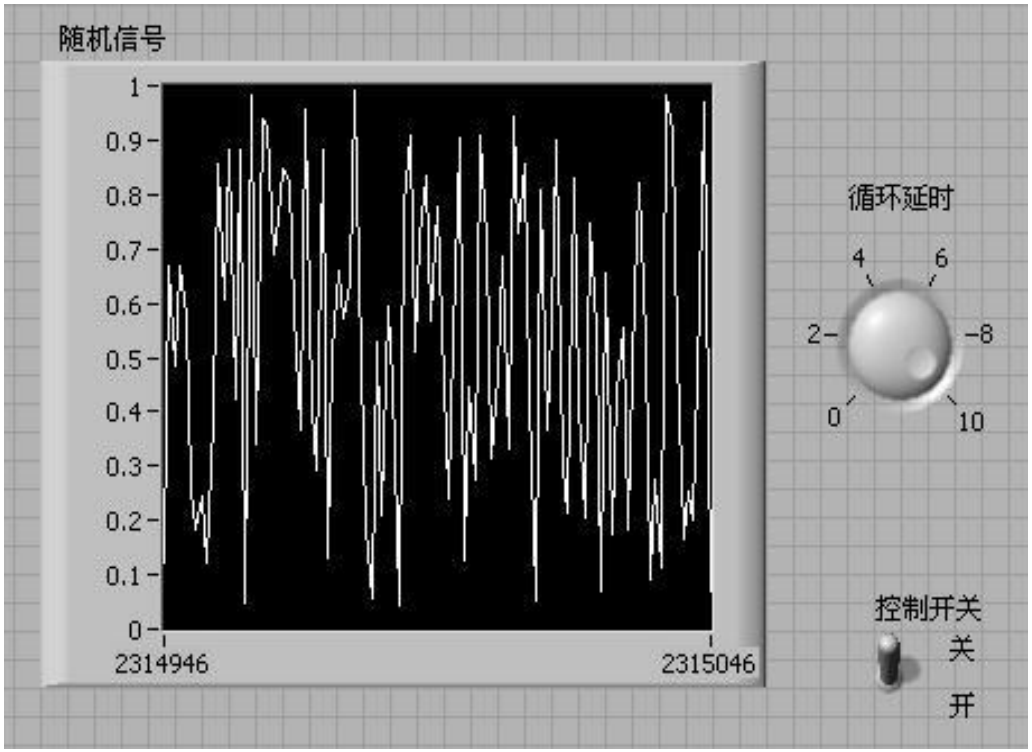
3.1 While循环

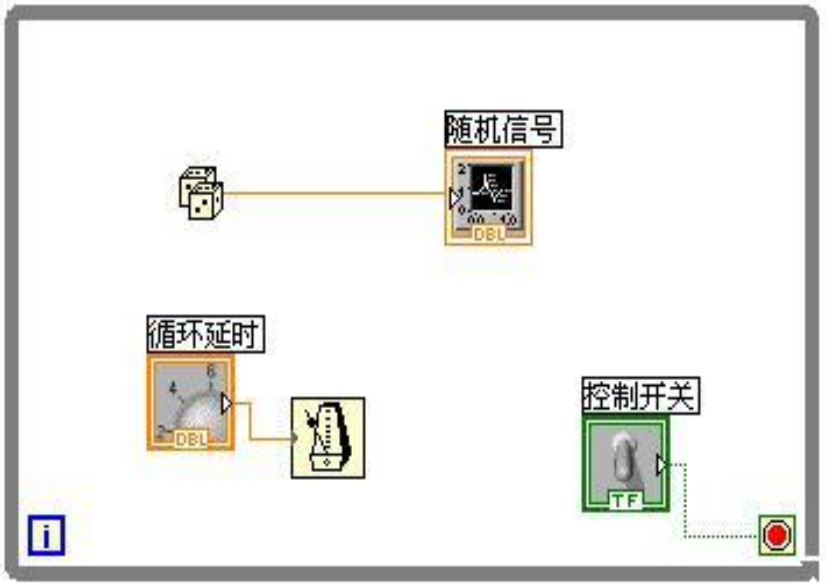
- **运行机理**
  - ✓While循环是先执行、后判断。
  - ✓先执行循环体内的图形化程序代码。
  - ✓循环计数端子加1，循环条件判断，决定是否继续循环。
  - ✓至少执行1次。
  - ✓相当于C++中的Do While循环。



3.1 While循环

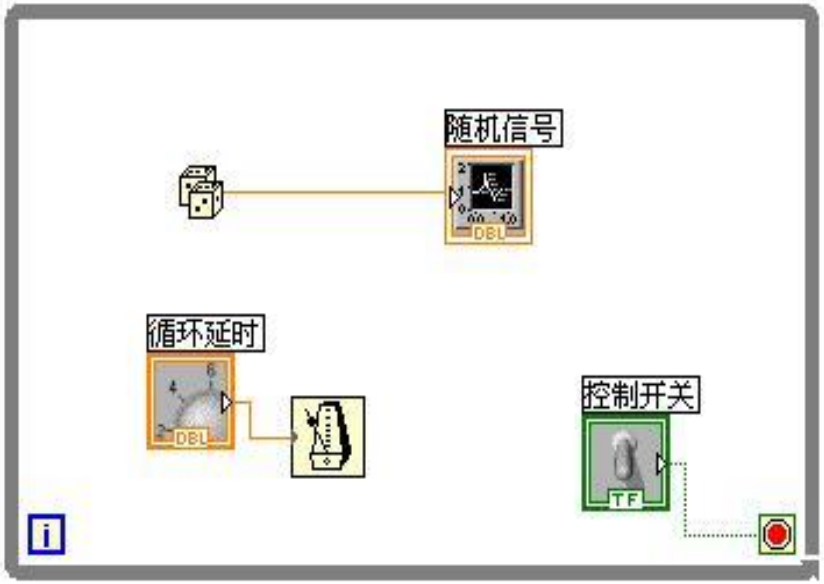
- 例2
  - ✓构建一个可显示随机信号波形的虚拟仪器程序即VI，其速度应可调。





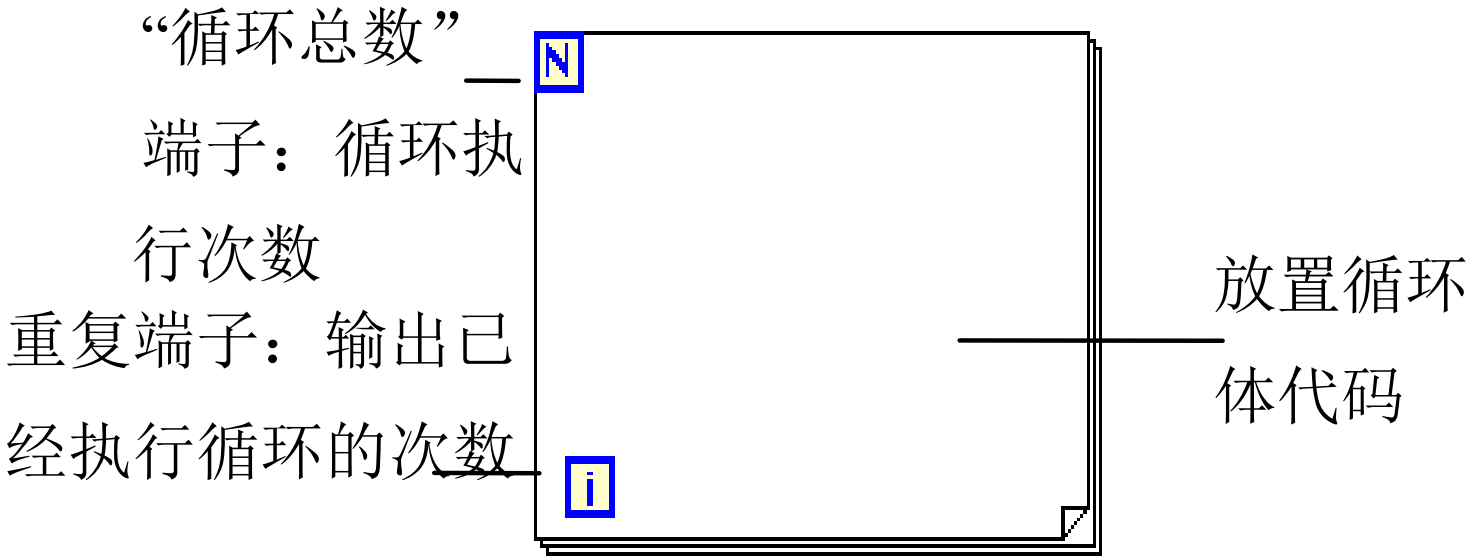
3.1 While循环

- 例2
  - ✓ While循环的条件端子一个常用的接法是接上布尔量控制开关
  - ✓ 至少执行1次
  - ✓ 很有实际意义，此种结构和功能经常会用到



3.2 For循环

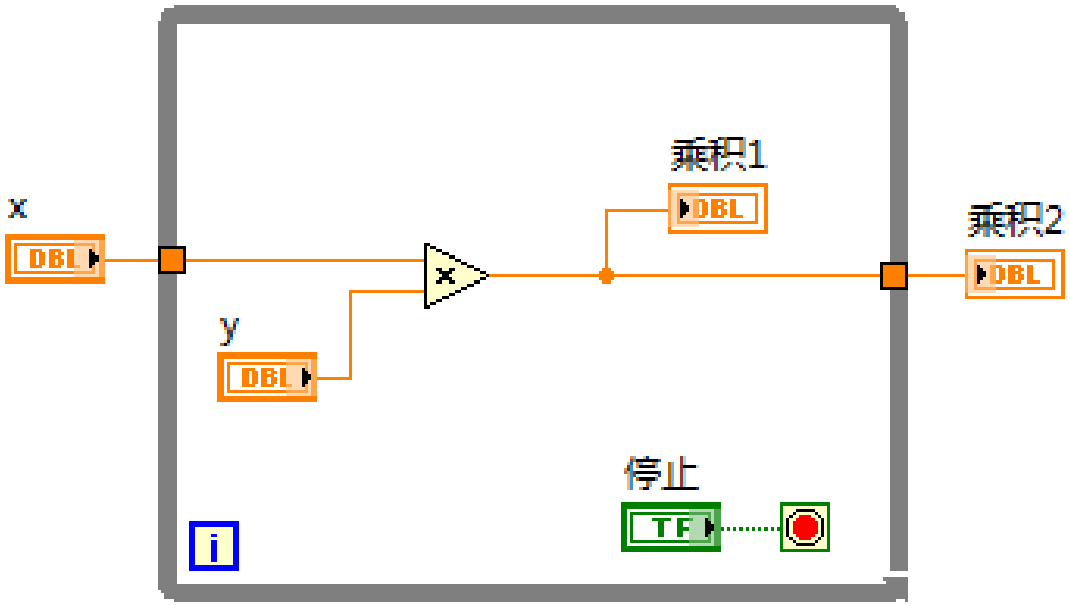
- 运行机理
  - ✓ For循环是先判断、后执行（子框图程序代码）。
  - ✓ 执行循环的次数是确定的。
  - ✓ 判断i是否在 0 到 N-1 的范围内。



3.3 循环结构内外之间的数据交换

- 数据是通过隧道的方式进出循环的；
- 执行循环前，读数一次；循环结束，才输出。

- 例3



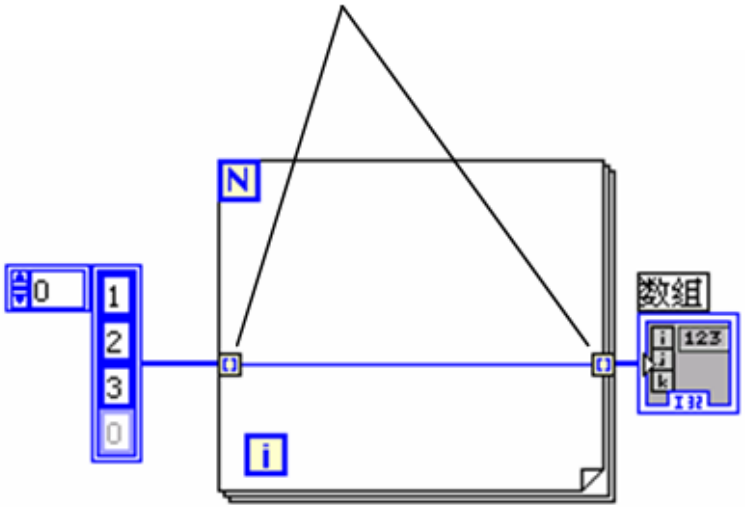
3.4 自动索引

- 当把一个数组连接到循环结构的边框上时，会在边框上生成可流动数据的隧道。
- 生成隧道后，可选择是否打开自动索引功能。
  - ✓ 隧道小方格呈空即“[ ]”，自动索引功能被打开；
  - ✓ 呈实心，则被关闭。
  - ✓ While循环，自动索引被默认关闭；
  - ✓ For循环，自动索引被默认打开.
- 如果打开，则数组将在每次循环中顺序经隧道送过一个元素；该元素在原数组中的索引（地址信息），与当次循环计数端子的值相同。
- 如果关闭，则将整个数组送进循环内。

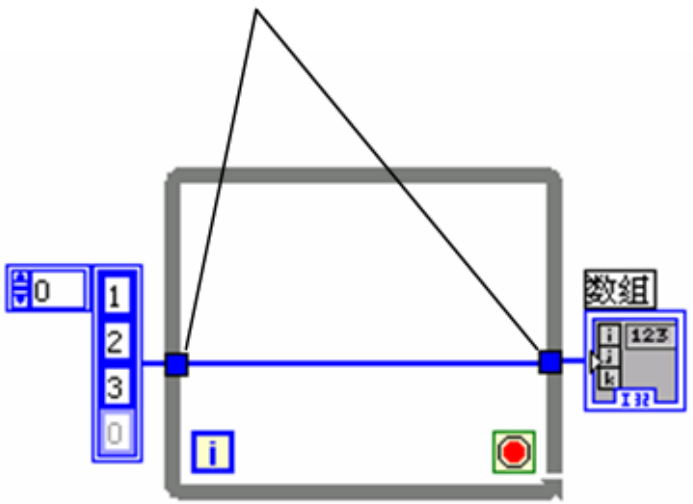
3.4 自动索引

- 循环次数
  - ✓ For循环
    - 接入多个数组且均自动索引，同时循环总数端子也接入一正整型常量，其循环次数取最小值。
  - ✓ While循环
    - 执行次数仍然由条件端子决定。

For 循环的自动索引默认打开

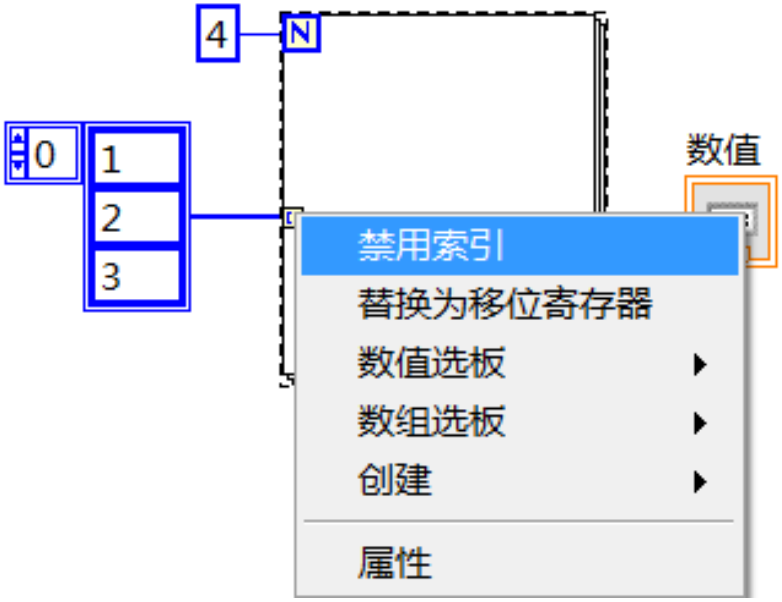


While 循环的自动索引默认关闭



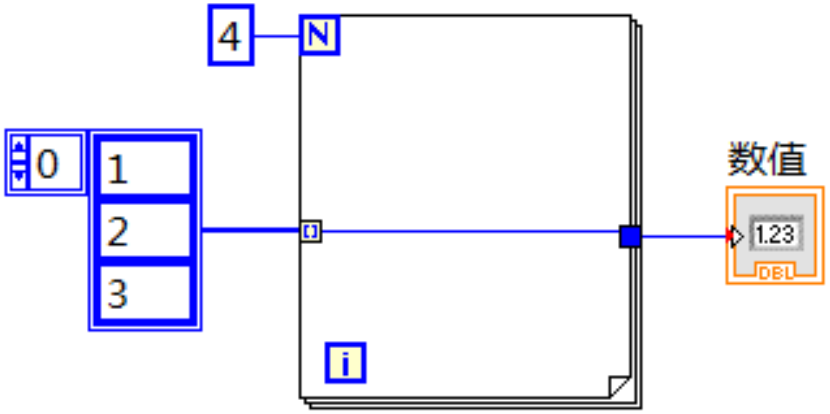
3.4 自动索引

- 自动索引状态转换
  - ✓ 选中隧道，点击鼠标右键，弹出快捷菜单，可以进行启动索引和禁用索引的转换。

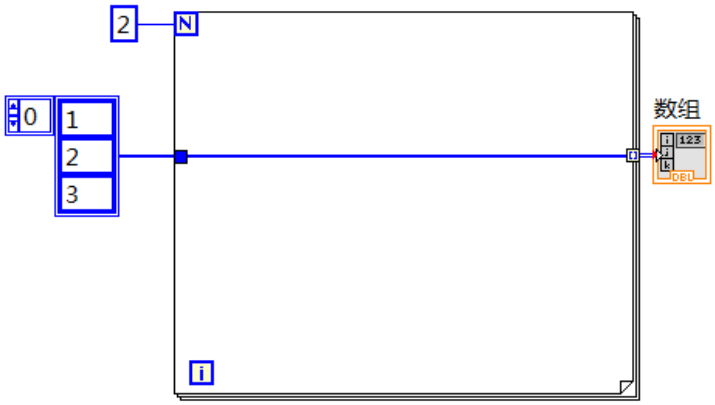


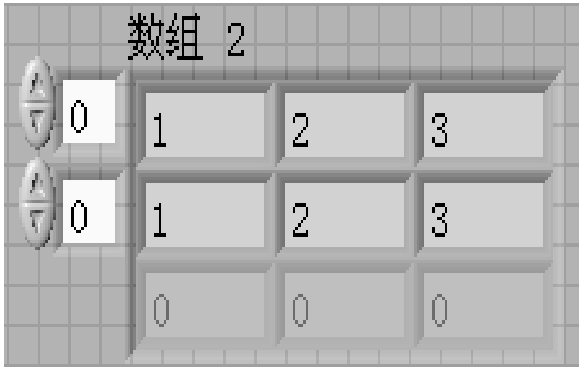
3.4 自动索引

例4 For循环输入隧道打开自动索引，而输出关闭自动索引



例5 For循环输入隧道关闭自动索引，而输出打开自动索引





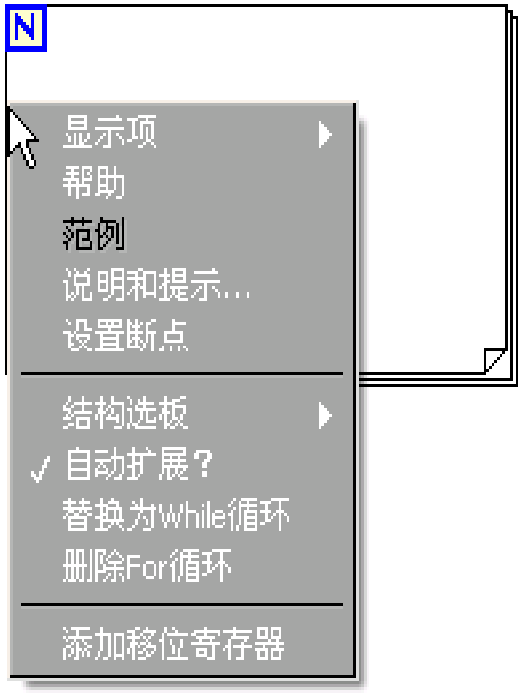
3.5 移位寄存器

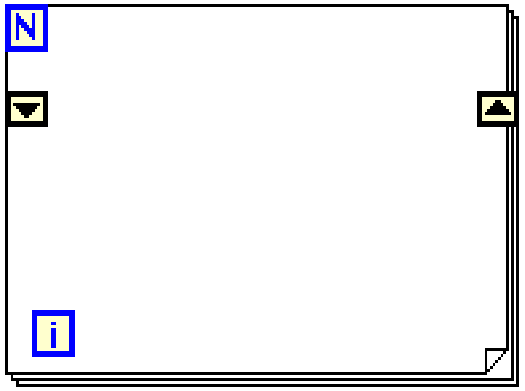
- **迭代运算**
  - ✓变量C的当前值要依靠上次循环的结果得到
  - ✓LabVIEW中如何保存上一次循环的运算结果呢?
- **移位寄存器的功能**
  - ✓ 把当前循环完成时子框图程序代码执行结果的某个数据，传递给下一次循环的开始作输入。

```
c=0;
for  i=1:20
     c=c+1;
end
```

3.5 移位寄存器

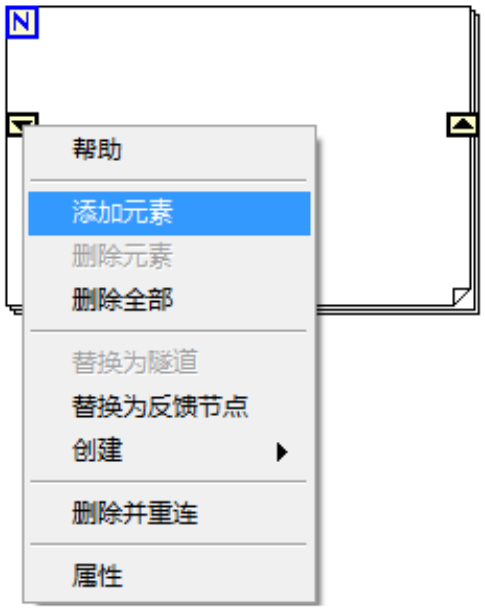
- 创建
  - ✓ 选中循环边框，右击鼠标，选择“添加移位寄存器”
  - ✓ 会在左边边框和右边边框同时生成（可建立一对）

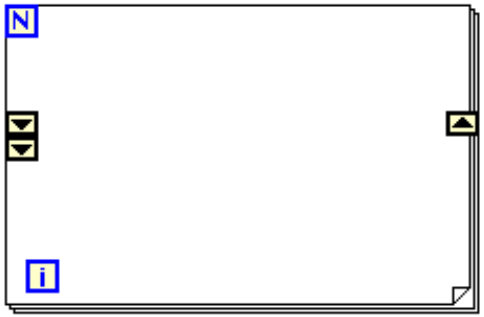




3.5 移位寄存器

- 创建
  - ✓ 要保存前面几次循环的值
  - ✓ 选中移位寄存器，右击鼠标，选择“添加元素”
  - ✓ 输入可多个，但输出只一个



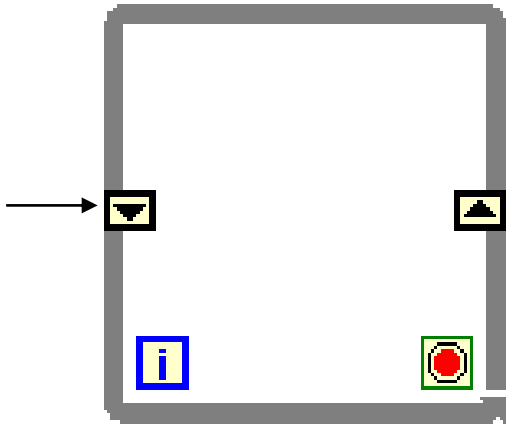


左接线端添加更多元素，保存之前多次循环的值（例：求移动平均值）

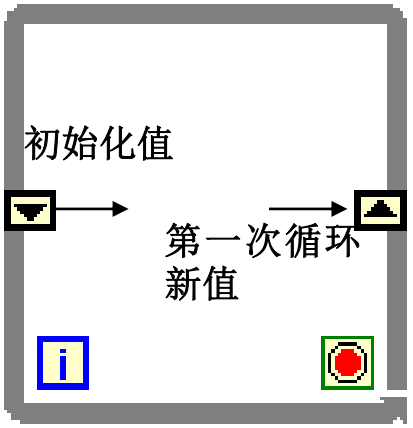
3.5 移位寄存器

工作流程：以While循环为例

1. 循环开始之前



2. 第一次循环

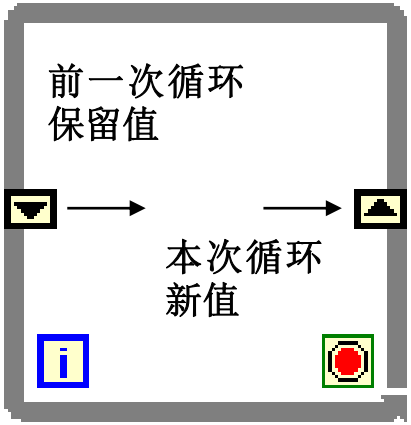


若没有初始化？

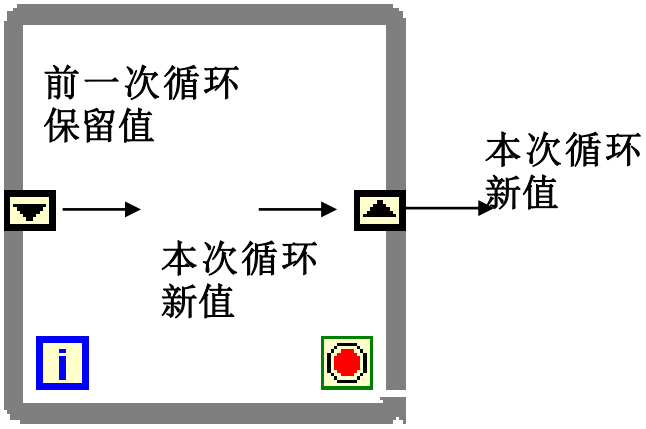
程序运行的第一次移位寄存器会使数据类型的默认值

关闭 VI 前，如果再次运行，移位寄存器会使用上一次储存的值

3. 后面的某次循环



4. 最后一次循环



3.5 移位寄存器

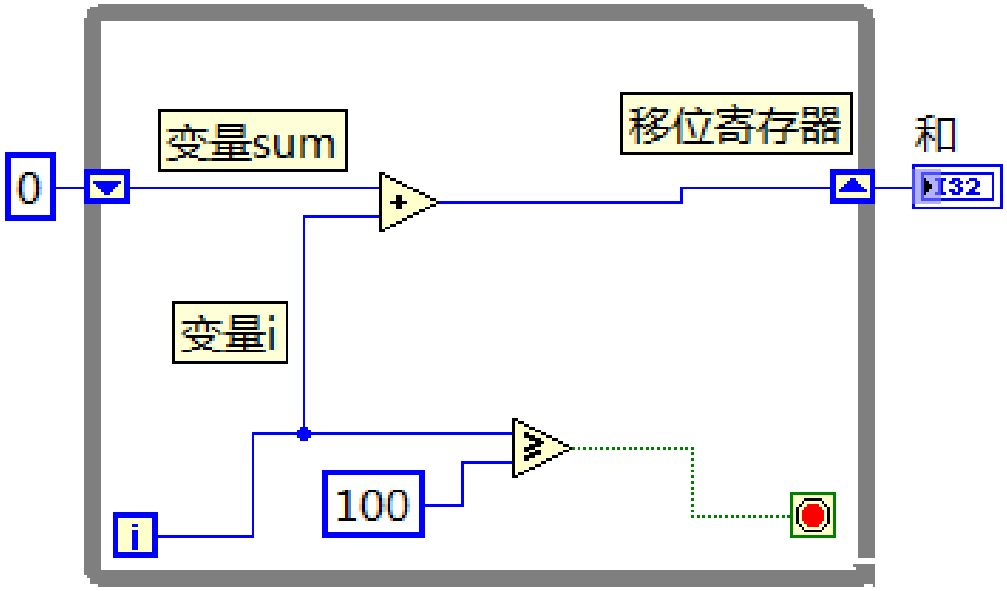
- **使用注意事项**
  - ✓ 一定要进行初始化
  - ✓ 刚生成的移位寄存器是没有数据类型的，其数据类型由连入到移位寄存器左端子或右端子的值来决定。
  - ✓ 只能在循环结构中使用

3.5 移位寄存器

- 例6 求和（n从1到100）。

```
main()
{
   int i,sum=0;
   i=1;
   do
      { sum=sum+i;
        i++;
      }
   while(i<=100);
   printf("%d",sum);
}
```

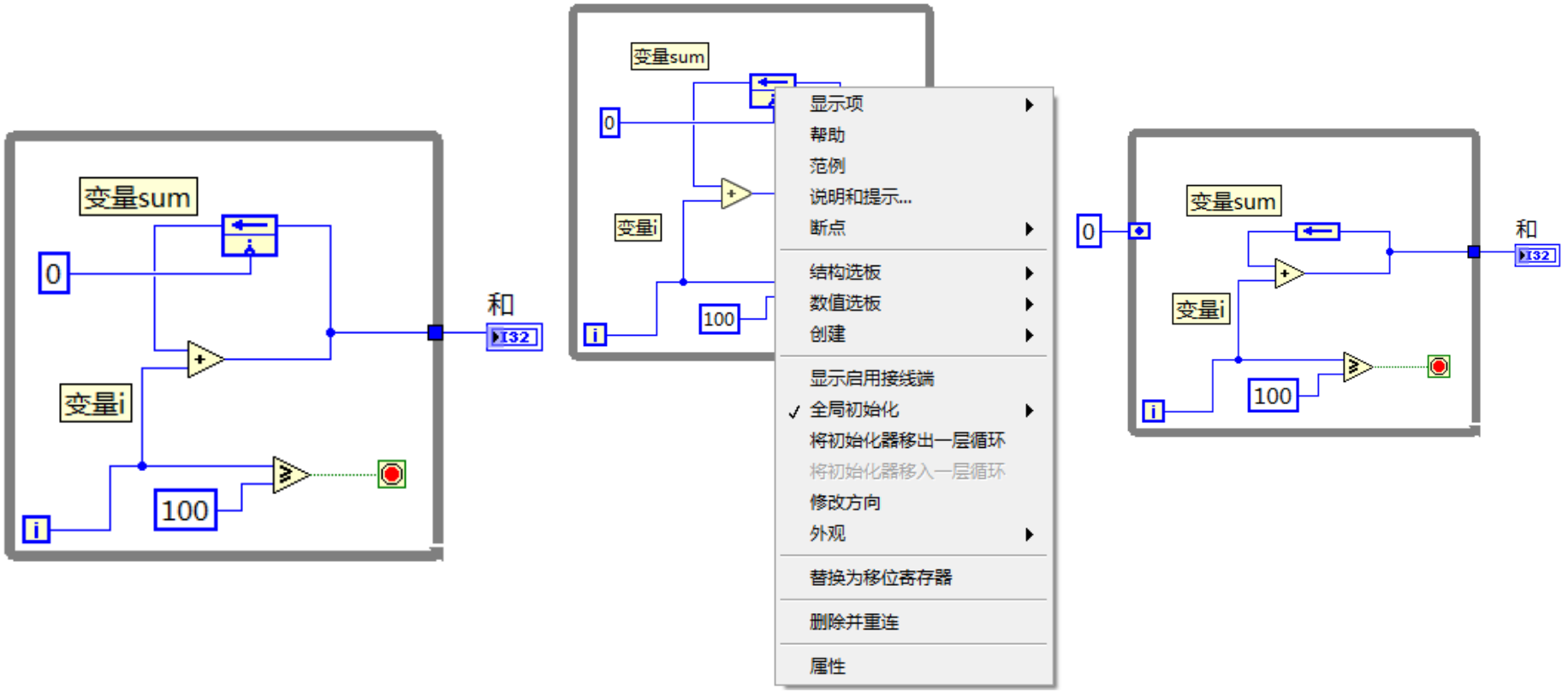
C++语言实现代码



LabVIEW中实现的代码

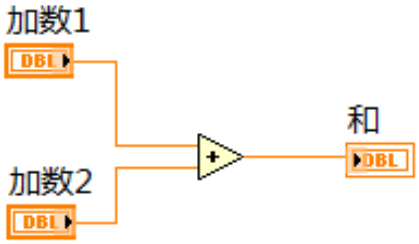
3.6 反馈节点

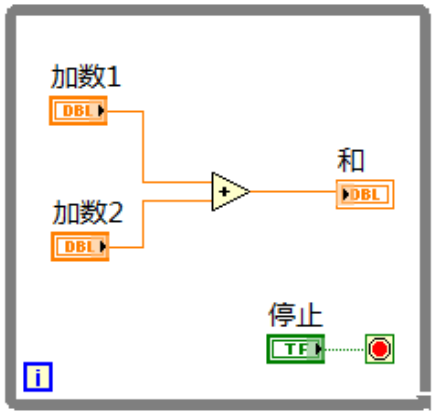
- 创建
  - ✓ 选中反馈节点，将其拖到程序框图中。
  - ✓ 可将反馈节点的初始化移出循环。



4. 事件结构

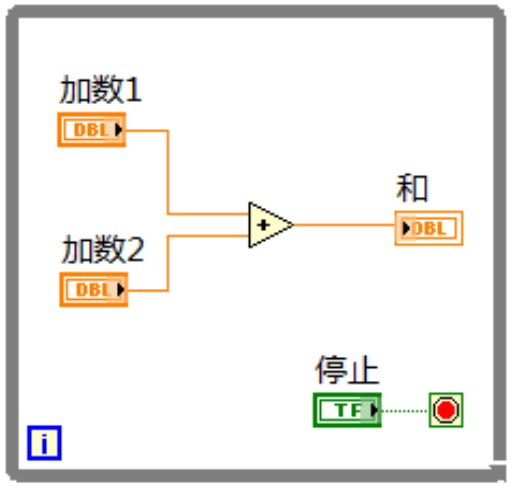
- 例7 制作一个简易加法器
  - ✓ 输入参数有“加数1”和“加数2”，输出结果“和”
  - ✓ 要求一旦VI运行后，在前面板上改变“加数1”或“加数2”的值，结果“和”的值也会跟着改变





4. 事件结构

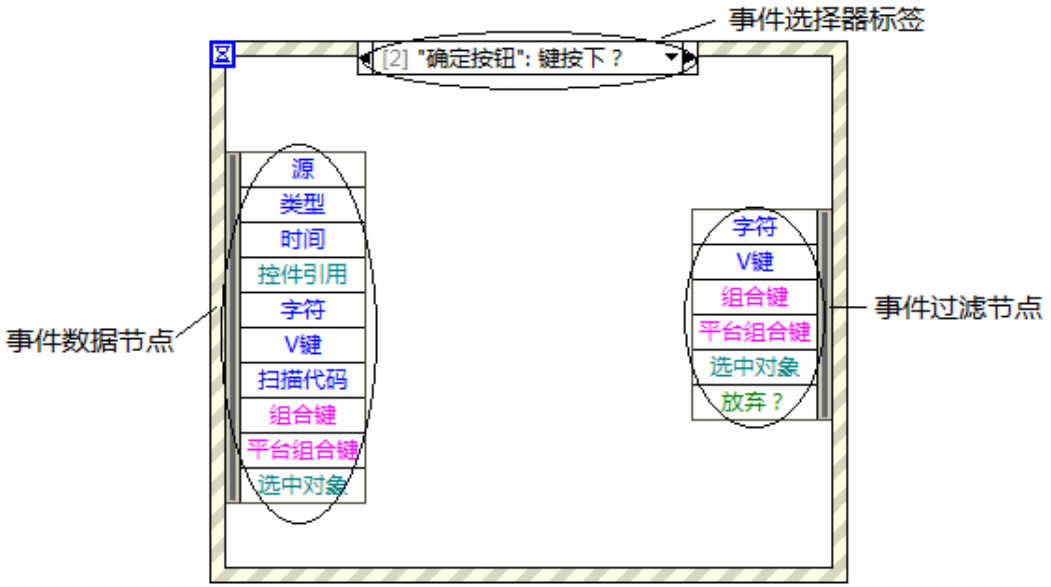
- 轮询
  - ✓ 每一次While循环，都会去读输入控件的值
  - ✓ 大大消耗CPU的使用时间
  - ✓ 如何实现只有当输入控件值改变时，才进行加法运算呢?



4. 事件结构

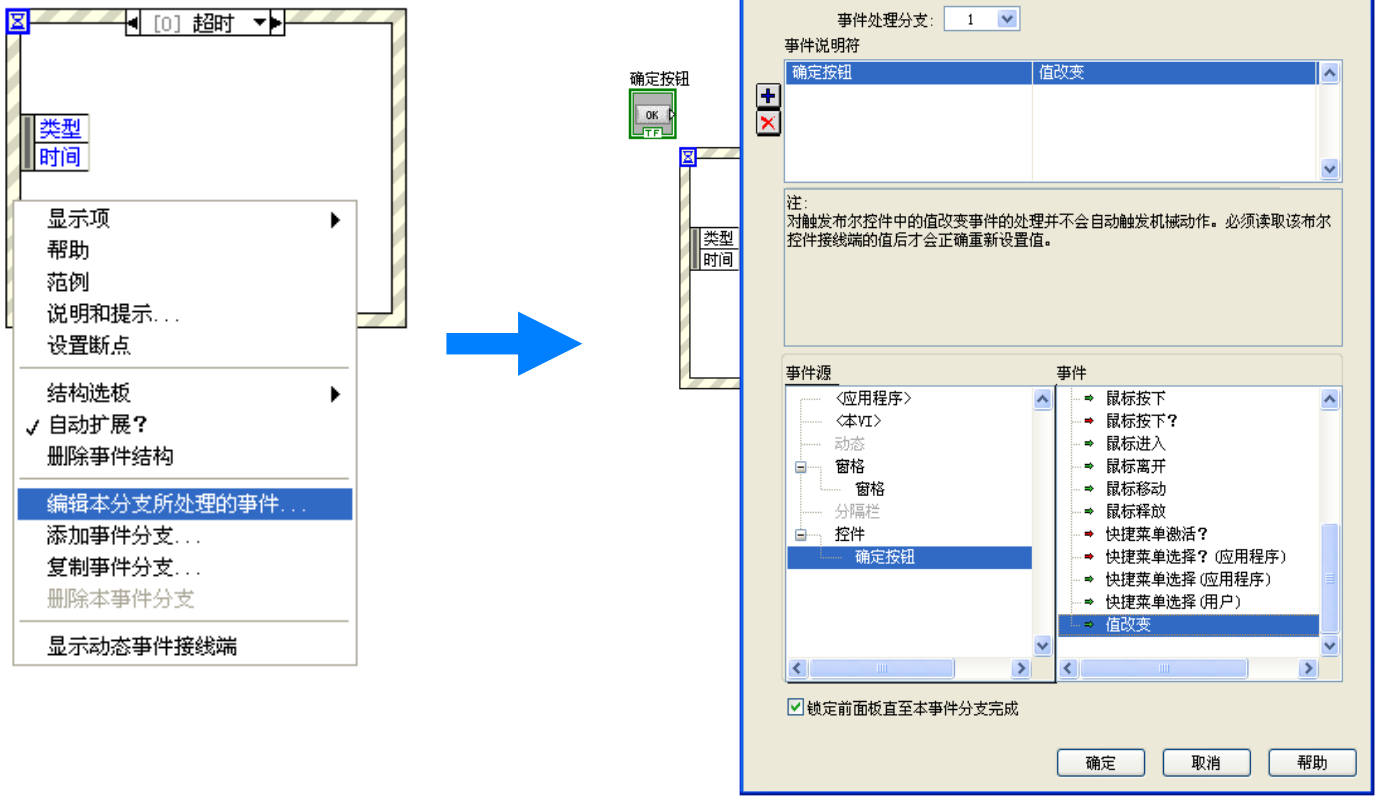
- 与条件结构类似

条件结构	事件结构
由选择器端子状态	根据发生的事件决定执行哪个分支
条件标签可以直接写入	事件标签通过编辑事件对话框来改变。



4. 事件结构

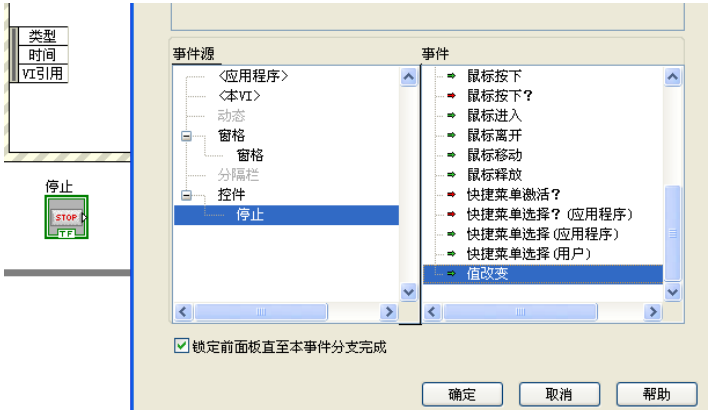
编辑事件对话框



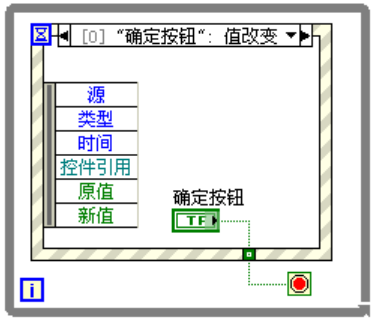
4. 事件结构—事件的种类

- 通知型事件
  - ✓ 不带问号；
  - ✓ 通知型事件是在LabVIEW处理用户操作之后发出的；
  - ✓ 表明某个用户操作已经发生；
  - ✓ 通知事件用于在事件发生且LabVIEW已对事件处理后对事件作出响应 。
- 过滤型事件
  - ✓ 带问号；
  - ✓ 过滤型事件是在LabVIEW处理用户操作之前发出的；
  - ✓ 通知用户LabVIEW在处理事件之前已由用户执行了某个操作，以便用户就程序如何与用户界面的交互作出响应进行自定义；
  - ✓ 使用过滤事件参与事件处理可能会覆盖事件的默认行为。

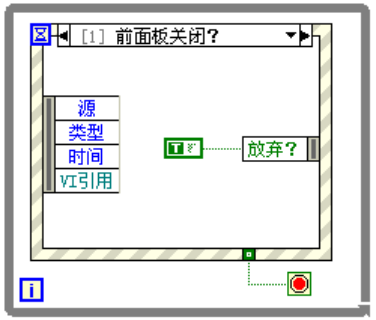
4. 事件结构—事件的种类



通知型事件

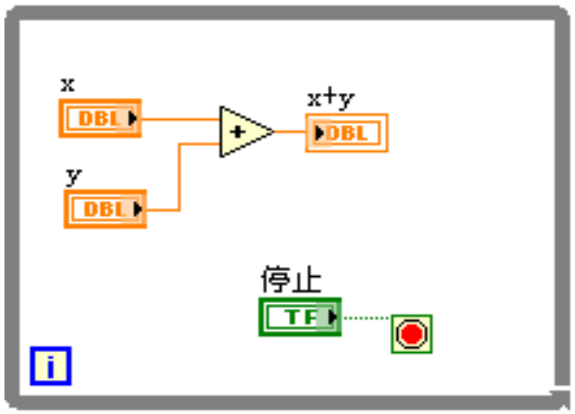


过滤型事件

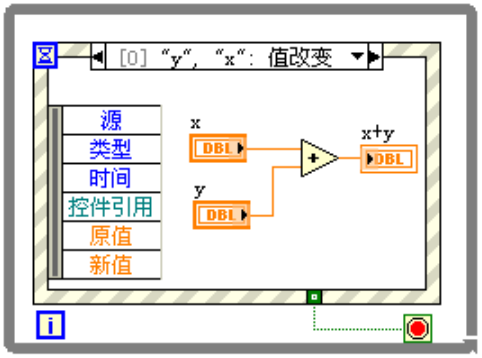


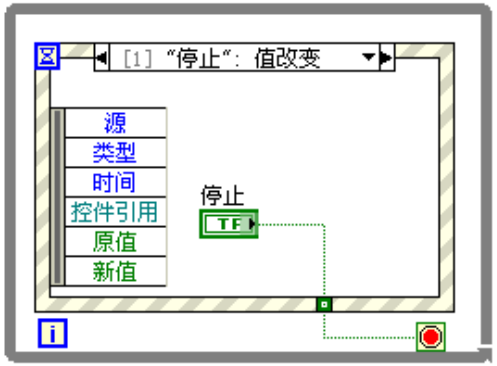
4.事件结构—使用

- 事件结构常会放在一个While循环里。
- 事件结构的优点：将CPU的使用降低到最小，但又不牺牲与用户的交互性。
- 而采用“轮询”方式，会大大消耗CPU的使用时间，不利于处理复杂、多线程的程序。



轮询

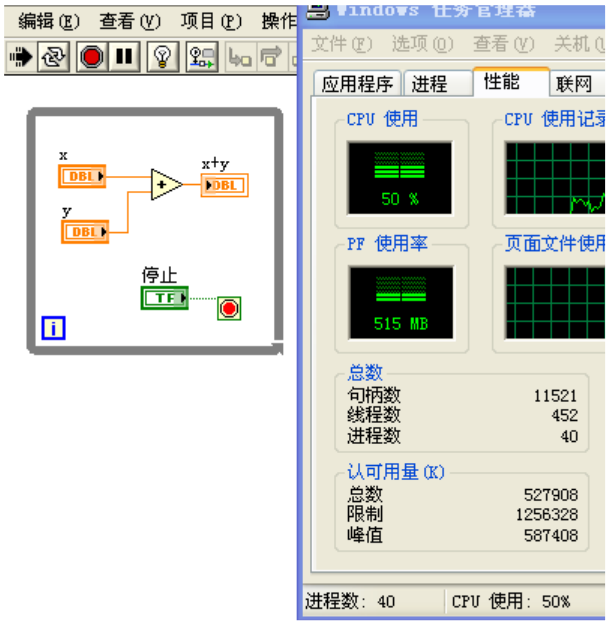




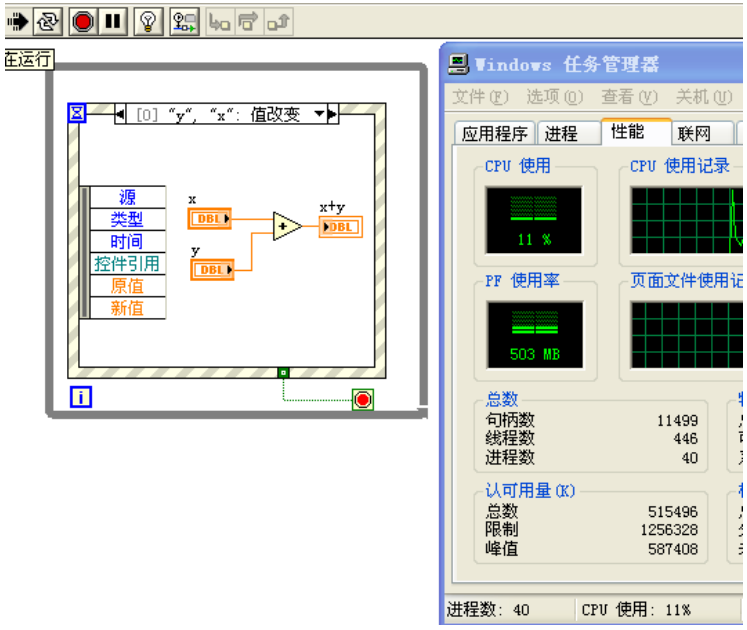
循环事件结构

4. 事件结构—使用

轮询：CPU使用50%



事件结构：CPU使用11%



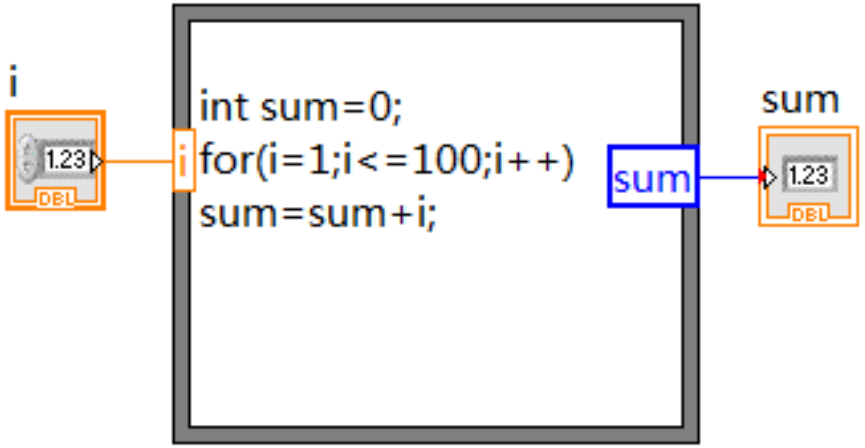
5. 公式节点

- 对拟采用的复杂算法，若完全依赖图形代码进行编程，会过于繁琐。针对此，LabVIEW中专门设立有以文本编辑形式实现程序逻辑的所谓“公式节点”。

  路径：函数选板-> 编程 -> 结构

  特点：公式节点代码文本的语法与C语言十分相似。

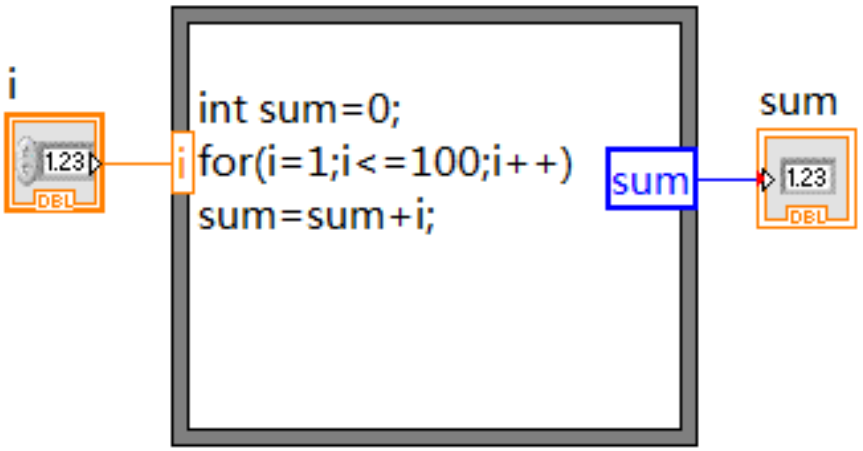
公式节点中，可以直接使用LabVIEW预定义函数和操作符。



5. 公式节点

通过输入、输出端子与外部交换数据

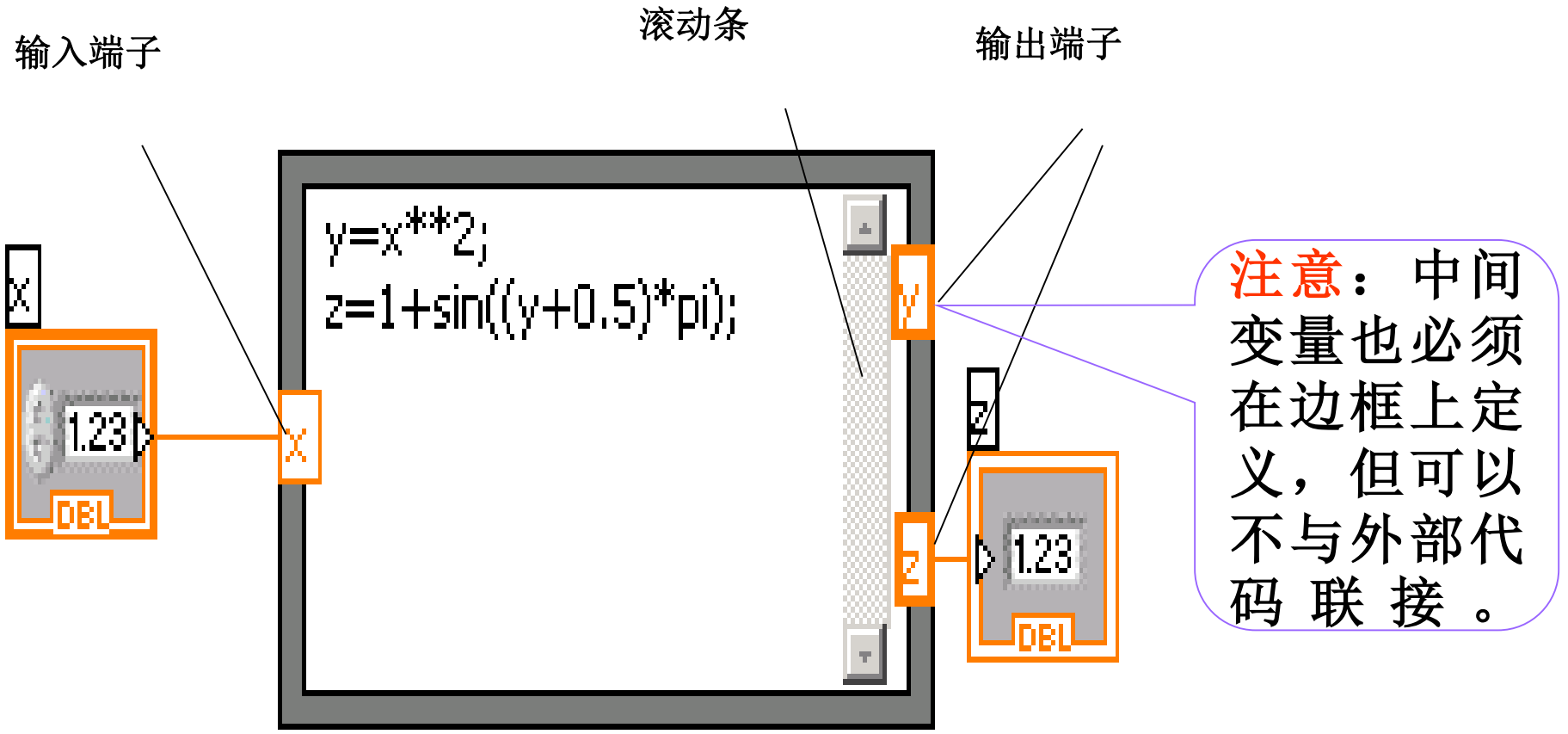
- 在左边框上弹出快捷菜单，选择“添加输入”，添加输入端子；
- 在右边框上弹出快捷菜单，选择“添加输出”，添加输出端子。



注意：端子名称必须与公式节点中的变量名称相同。各输入端子不能重名；各输出端子也不能重名；但输入端子与输出端子可重名。

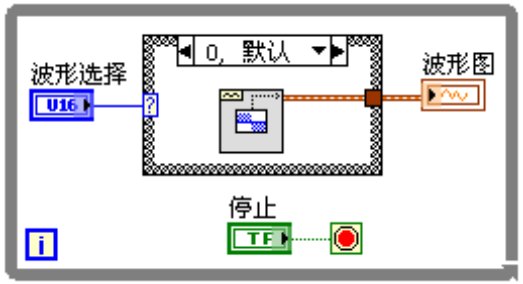
公式节点

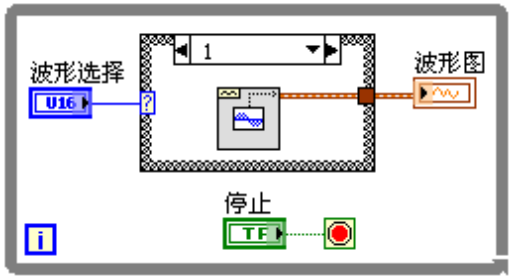
注意：变量名对字母的大小写敏感，故书写要一致。

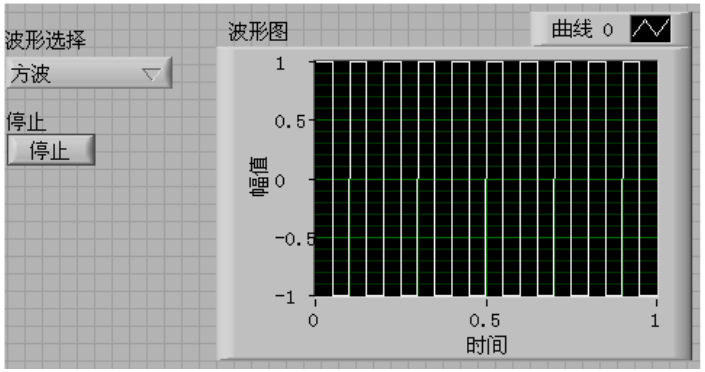


程序结构综合运用

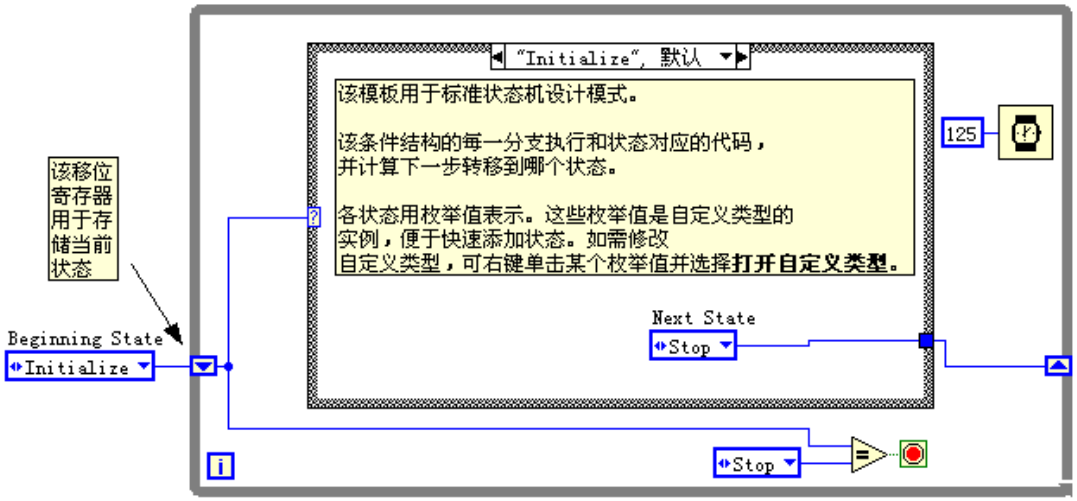
1. 一个While循环结构+条件结构

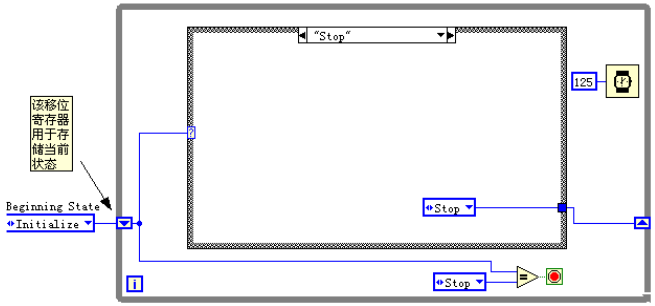






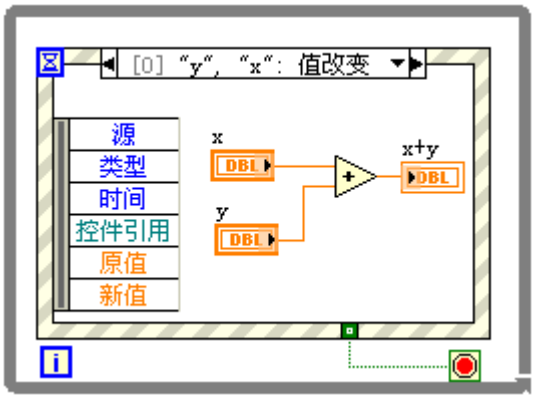
标准状态机

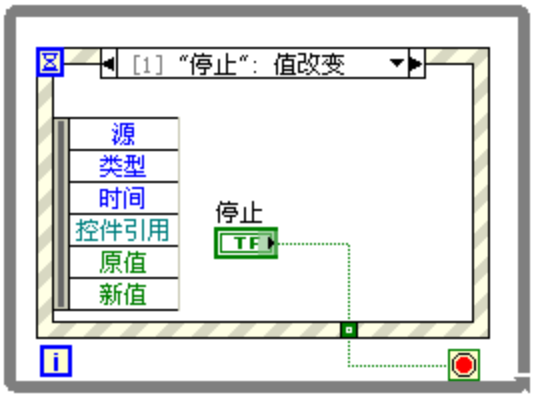




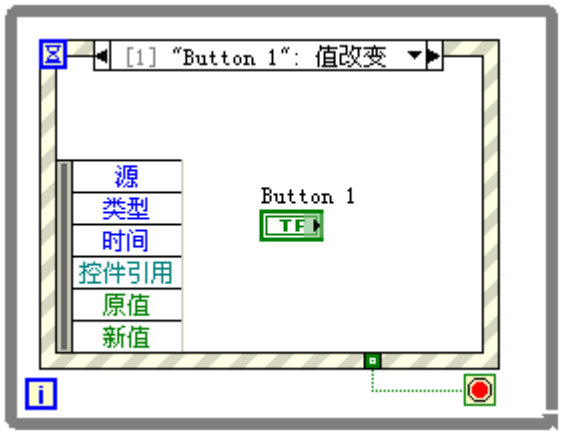
程序结构综合运用

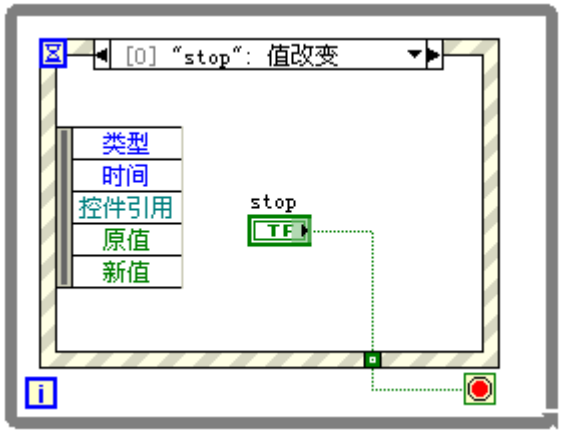
2. 一个While循环结构+事件结构





用户界面事件处理器





练习

1. 数据进出循环时，通过什么方式？

2. While 循环和 For 循环的默认隧道状态是什么？

3. 如果移位寄存器没有初始化，会出现什么结果？

练习

4. 针对以下问题，考虑使用 While循环还是For循环：

一秒钟采集一次温度数据，采集时间是 1 分钟。如果使用 While 循环，如何设置停止条件？ 如果使用 for循环，如何设置循环次数？

练习

5. 利用分支结构，构建一个求平方根的VI，并编辑其图标。

6. 编制一个温度报警程序，当温度值高于30度时就报警；小于-25度时则退出运行状态。

7. 产生一个 1～100 的随机数，当该随机数在一个指定值 ± 3范围内停止，并且输出程序循环的次数。

练习

8. 构建一个简易加法计算器。

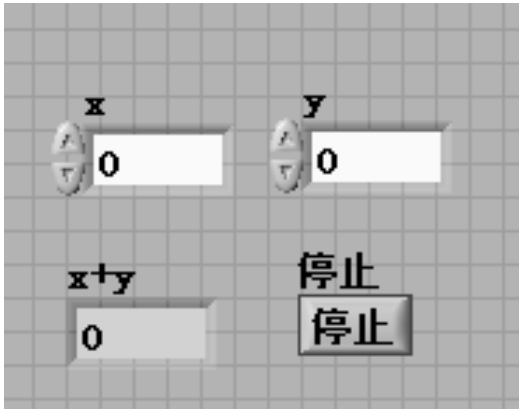
(1)利用“轮询”，循环内不加定时函数；

(2)利用“轮询”，循环内加定时函数。

(3)利用“循环事件结构” 分支为“超时”；

(4)利用“循环事件结构” 分支为“值改变”；

理解四种编程方式，运行程序并打开“任务管理器”查看CPU的使用情况。



谢谢