

LabVIEW

程序设计与应用(第2版)

<http://www.phei.com.cn>

 电子工业出版社
PUBLISHING HOUSE OF ELECTRONICS INDUSTRY

杨乐平 李海涛 杨磊 编著



附光盘



LabVIEW

程序设计与应用(第2版)

本书以最新LabVIEW 7 Express版本为对象,系统介绍了LabVIEW程序设计的基本概念、关键技术和实际应用的专门知识。本书内容分为三大部分,第一部分介绍虚拟仪器的基本概念、图形化编程语言基本原理与特点、LabVIEW编程环境;第二部分系统介绍LabVIEW程序设计的语法规则、程序结构和基本编程技巧;第三部分介绍LabVIEW在数据采集、仪器控制和通信等方面的应用。本书结构编排合理,运用大量实例阐述基本概念与编程难点,突出内容的系统性与实用性。为方便读者学习查阅,本书附带光盘按章节编排,提供了本书所有编程例子,并且列出了LabVIEW程序错误代码表,供读者参考。

本书可作为大、中专院校相关专业教材或教学参考书,也可供有关工程技术人员和软件工程师参考。

ISBN 7-121-00588-3



9 787121 005886 >



责任编辑:刘志

封面设计:张

本书贴有激光防伪标志,凡没有防伪标志者,属盗版图书。

ISBN 7-121-00588-3 定价:42.00元(含光盘1张)

LabVIEW 程序设计与应用

(第2版)

杨乐平 李海涛 杨 磊 编著

电子工业出版社

Publishing House of Electronics Industry

北京·BEIJING

内 容 简 介

本书以最新 LabVIEW 7 Express 版本为对象,系统介绍了 LabVIEW 程序设计的基本概念、关键技术和实际应用的专门知识。本书内容分为三大部分,第一部分介绍虚拟仪器的基本概念、图形化编程语言基本原理与特点、LabVIEW 编程环境;第二部分系统介绍 LabVIEW 程序设计的语法规则、程序结构和基本编程技巧;第三部分介绍 LabVIEW 在数据采集、仪器控制和通信等方面的应用。本书结构编排合理,运用大量实例阐述基本概念与编程难点,突出内容的系统性与实用性。为方便读者学习查阅,本书附带光盘技章节编排,提供了本书所有编程例子,并且列出了 LabVIEW 程序错误代码表,供读者参考。

本书可作为大、中专院校相关专业教材或教学参考书,也可供有关工程技术人员和软件工程师参考。

未经许可,不得以任何方式复制或抄袭本书之部分或全部内容。
版权所有,侵权必究。

图书在版编目(CIP)数据

LabVIEW 程序设计与应用 / 杨乐平, 李海涛, 杨磊编著. —2 版. —北京: 电子工业出版社, 2005.1

ISBN 7-121-00588-3

I. L... II. ①杨...②李...③杨... III. 软件工具, LabVIEW—程序设计 IV. TP311.56

中国版本图书馆 CIP 数据核字(2004)第 120494 号

责任编辑: 刘志红 特约编辑: 张莉

印 刷: 北京民族印刷厂

出版发行: 电子工业出版社

北京市海淀区万寿路 173 信箱 邮编 100036

经 销: 各地新华书店

开 本: 787×1092 1/16 印张: 29.5 字数: 756 千字

印 次: 2005 年 1 月第 1 次印刷

印 数: 4000 册 定价: 42.00 元(含光盘 1 张)

凡购买电子工业出版社的图书,如有缺损问题,请向购买书店调换。若书店售缺,请与本社发行部联系。联系电话:(010) 68279077。质量投诉请发邮件至 zllts@phei.com.cn, 盗版侵权举报请发邮件至 dbqq@phei.com.cn。

前 言

自 20 世纪 90 年代以来,随着计算机技术的迅猛发展,虚拟仪器技术在数据采集、自动测试和测量仪器领域得到广泛应用,促进和推动测试系统和测量仪器的设计方法与实现技术发生了深刻的变化。“软件就是仪器”已经成为测试与测量技术发展的重要标志。美国国家仪器公司(National Instruments,简称 NI)是虚拟仪器技术的主要倡导者和贡献者,其创新软件产品 LabVIEW(Laboratory Virtual Instrument Engineering Workbench,简称 LabVIEW)自 1986 年问世以来,已经成为虚拟仪器软件开发平台事实上的工业标准,在研究、制造和开发的众多领域得到广泛应用。从简单的仪器控制、数据采集到尖端的测试和工业自动化,从大学实验室到工厂,从探索研究到技术集成,人们都可以发现 LabVIEW 应用的成果和开发的产品。LabVIEW 所创新的图形化语言编程方法成为虚拟仪器技术应用与发展的重要基础,得到工业界和学术界的广泛关注。

最近几年,随着 LabVIEW 在国内普及和应用的推广,陆续出版了一批有关 LabVIEW 程序设计和应用开发方面的教材和专著,培养了一批 LabVIEW 的忠实用户和程序设计员。由本书编著者编写,电子工业出版社 2001 年 7 月出版的《LabVIEW 程序设计与应用》是国内第一本系统介绍 LabVIEW 程序设计的入门书。该书体系结构设计突出了图形化编程语言的概念、方法与特点,论述深入浅出,编排结构和编写风格采用寓教于例、图文结合的形式,得到了广大 LabVIEW 用户的认同和欢迎。该书已多次重印,国内多所高等院校已将该书选为教材。但是由于该书是以当时的 LabVIEW 5.1 版本为蓝本组织内容,而目前 NI 公司已经推出的最新 LabVIEW 7 Express 版本,在开发环境、编程手段、应用管理等方面增加了许多先进功能,因此有必要对该书内容进行更新、完善和扩展,以适应技术发展的要求。

本书在保持原书风格、特色与体系基本不变的基础上,以 LabVIEW 7 Express 版本为对象,结合读者反馈意见和我们最新的研究成果,对原书内容进行了较大的增加、修订和调整。增加的内容主要包括 LabVIEW 7 Express 版本在编程环境、数据类型、模板设置等方面新增的功能,如 Express VI、动态数据、波形数据、事件结构、反馈节点、逐点数字信号处理等;也包括在数据采集、仪器控制和通信等方面的应用,增加或扩展了 VPP 和 IVI 仪器驱动器、DataSocket 编程、远程操作面板、PLC 通信等内容。在修订和调整时,主要对原书的一些论述和结构顺序进行了完善和优化,如将字符串运算从原第 8 章“字符串与文件 I/O”调整到第 4 章“数据操作”,原第 8 章则加大了文件操作与管理的论述与编程实例等。总之,本书力求在保持原书特色的基础上,内容更加系统完整,结构更加科学合理,应用更加全面深入,为广大 LabVIEW 用户提供一本易懂实用的入门教材。

参加本书编写工作的有国防科技大学杨乐平教授、李海涛博士和杨磊博士,全书由杨乐平教授统稿。本书可供相关专业高年级本科生和研究生作为教材使用,也可供从事测试计量、仪器设计、过程控制及数据处理方面工作的工程技术人员参考。

由于编著者水平有限,书中难免有疏漏和错误之处,恳请广大读者批评指正。

编著者

2004 年 10 月于国防科技大学

目 录

第 1 章 绪论	(1)
1.1 LabVIEW 概述	(1)
1.1.1 LabVIEW 起源	(1)
1.1.2 LabVIEW 概念创新	(2)
1.2 G 语言与虚拟仪器	(3)
1.3 LabVIEW 应用解决方案	(4)
1.4 LabVIEW 7 Express 新特性	(6)
第 2 章 LabVIEW 编程环境	(11)
2.1 LabVIEW 系统安装	(11)
2.2 LabVIEW 启动	(11)
2.3 LabVIEW 模板	(15)
2.4 VI 库	(21)
2.5 定制 LabVIEW 环境	(22)
2.5.1 环境参数设置	(22)
2.5.2 模板设置	(23)
第 3 章 LabVIEW 编程入门	(26)
3.1 基本概念	(26)
3.1.1 前面板	(27)
3.1.2 框图程序	(29)
3.1.3 使用数据连线	(33)
3.1.4 图标/连接端口	(41)
3.2 LabVIEW 术语	(42)
3.3 创建和编辑 VI	(43)
3.3.1 创建 VI	(44)
3.3.2 编辑 VI	(49)
3.4 运行和调试 VI	(60)
3.4.1 运行 VI	(60)
3.4.2 调试 VI	(61)
3.5 创建和调用 SubVI	(64)
3.5.1 创建 SubVI	(64)
3.5.2 调用 SubVI	(67)
3.6 Express VI	(69)

3.6.1	Express VI 的特点	(69)
3.6.2	Express VI 的使用方法	(72)
3.7	获取帮助	(74)
3.7.1	实时上下文帮助	(74)
3.7.2	VI 及功能模块帮助	(75)
3.7.3	LabVIEW 例程	(75)
3.7.4	LabVIEW 书架	(76)
3.7.5	LabVIEW 网络资源	(77)
第 4 章	数据操作	(78)
4.1	数据类型	(78)
4.1.1	数字型	(80)
4.1.2	布尔型	(85)
4.1.3	字符串型与路径	(86)
4.2	数学运算	(90)
4.2.1	数字常量	(91)
4.2.2	基本数学运算节点	(91)
4.2.3	类型转换节点	(93)
4.2.4	三角函数节点	(96)
4.2.5	对数节点	(97)
4.2.6	复数节点	(98)
4.2.7	附加常数节点	(99)
4.3	布尔运算	(100)
4.4	字符串运算	(102)
4.4.1	字符串常量	(102)
4.4.2	基本字符串运算	(103)
4.4.3	字符串/数字转换	(107)
4.4.4	字符串/数组/路径转换	(110)
4.4.5	附加字符串运算	(112)
4.5	比较运算	(115)
第 5 章	变量、数组、簇与波形数据	(118)
5.1	本地变量	(118)
5.1.1	本地变量的创建	(118)
5.1.2	本地变量的使用	(120)
5.1.3	本地变量的特点	(121)
5.2	全局变量	(122)
5.2.1	全局变量的创建	(122)
5.2.2	全局变量的使用	(123)
5.2.3	全局变量的特点	(124)

5.3	数组	(125)
5.3.1	数组的组成与创建	(125)
5.3.2	数组的使用	(127)
5.3.3	数组的特点	(149)
5.4	簇	(149)
5.4.1	簇的组成与创建	(149)
5.4.2	簇的使用	(151)
5.4.3	簇的特点	(161)
5.5	波形数据	(163)
5.5.1	波形数据的组成	(163)
5.5.2	波形数据的使用	(167)
5.5.3	波形数据的特点	(178)
第6章	结构与属性	(179)
6.1	For 循环	(179)
6.1.1	For 循环的组成	(180)
6.1.2	For 循环的使用	(182)
6.1.3	For 循环的特点	(186)
6.2	While 循环	(186)
6.2.1	While 循环的组成	(186)
6.2.2	While 循环的使用	(187)
6.2.3	While 循环的特点	(188)
6.3	顺序结构	(189)
6.3.1	顺序结构的组成	(190)
6.3.2	顺序结构的使用	(192)
6.3.3	顺序结构的特点	(194)
6.4	选择结构	(196)
6.4.1	选择结构的组成	(196)
6.4.2	选择结构的使用	(198)
6.4.3	选择结构的特点	(198)
6.5	事件结构	(199)
6.5.1	事件结构的组成	(200)
6.5.2	事件结构的使用	(201)
6.5.3	事件结构的特点	(203)
6.6	基本公式节点	(205)
6.6.1	基本公式节点的创建	(205)
6.6.2	基本公式节点的使用	(206)
6.6.3	基本公式节点的特点	(209)
6.7	属性节点	(209)

6.7.1	属性节点的创建	(209)
6.7.2	属性节点的使用	(211)
6.7.3	属性节点的特点	(214)
第 7 章	波形显示	(215)
7.1	事后记录波形图	(215)
7.1.1	事后记录波形图的组成	(216)
7.1.2	使用事后记录波形图	(217)
7.1.3	定制事后记录波形图的外观	(223)
7.2	实时趋势图	(231)
7.2.1	使用实时趋势图	(231)
7.2.2	定制实时趋势图的外观定制	(233)
7.3	XY 波形图	(235)
7.4	密度图形显示控件 (Intensity Graph)	(237)
7.4.1	使用密度图	(237)
7.4.2	定义密度图的颜色	(238)
7.4.3	设置密度图的外观	(239)
7.5	密度趋势图	(240)
7.6	三维曲面图	(241)
7.6.1	使用三维曲面图	(241)
7.6.2	设置三维曲面图的外观	(244)
7.7	三维参数曲面图	(247)
7.7.1	使用三维参数曲面图	(248)
7.7.2	设置三维参数曲面图的外观	(249)
7.8	三维曲线图	(249)
7.8.1	使用三维曲线图	(249)
7.8.2	设置三维曲线图的外观	(251)
7.9	极坐标图	(251)
7.9.1	使用极坐标图	(251)
7.9.2	设置极坐标图的外观	(252)
第 8 章	文件操作与管理	(253)
8.1	基本概念及术语	(253)
8.1.1	路径	(253)
8.1.2	标识号	(254)
8.1.3	文件 I/O 的出错管理	(256)
8.1.4	文件 I/O 操作流程控制	(256)
8.2	文件操作	(256)
8.2.1	文件定位与文件对话框	(257)
8.2.2	文件操作	(260)

8.3	文件管理	(267)
8.3.1	文件的删除、移动和复制	(268)
8.3.2	获取文件、目录的信息	(268)
8.3.3	路径、目录操作	(269)
8.4	数据存储与读取	(272)
8.4.1	LabVIEW 数据文件类型	(272)
8.4.2	数据文件存储与读取	(273)
第 9 章	数学分析与信号处理	(284)
9.1	数学分析	(285)
9.1.1	公式运算节点	(286)
9.1.2	函数计算与微积分	(289)
9.1.3	概率统计与曲线拟合	(295)
9.1.4	矩阵与数组运算	(300)
9.1.5	最优化与零点求解	(304)
9.1.6	数值函数	(309)
9.2	数字信号处理	(310)
9.3	波形测量	(328)
9.3.1	波形测量节点	(328)
9.3.2	波形测量应用实例	(331)
9.4	信号调理	(332)
9.4.1	信号调理节点	(332)
9.4.2	信号调理应用实例	(333)
9.5	波形监测	(335)
9.5.1	波形监测节点	(335)
9.5.2	波形监测应用实例	(336)
9.6	逐点信号分析	(338)
9.6.1	逐点信号分析的特点	(338)
9.6.2	逐点信号分析节点	(338)
9.6.3	逐点信号分析应用实例	(339)
第 10 章	LabVIEW 程序设计	(340)
10.1	人机交互界面	(340)
10.1.1	定制前面板对象	(340)
10.1.2	选单的编辑与响应	(341)
10.1.3	子面板的使用	(343)
10.1.4	界面装饰	(345)
10.2	定时与对话框	(346)
10.2.1	定时器	(347)
10.2.2	对话框	(347)

10.2.3	错误处理节点	(351)
10.3	LabVIEW 环境参数设置	(352)
10.3.1	新特性	(353)
10.3.2	路径与性能	(355)
10.3.3	编程界面	(356)
10.3.4	模板与调试	(359)
10.3.5	属性设置	(361)
10.3.6	VI Server 与 Web Server	(365)
10.4	VI 属性设置	(368)
10.4.1	一般设置	(369)
10.4.2	存储空间	(370)
10.4.3	帮助与编辑	(371)
10.4.4	版本历史与安全	(372)
10.4.5	窗口与运行	(374)
10.5	文件管理	(376)
10.6	创建应用程序	(378)
10.7	培养良好的编程风格	(381)
第 11 章	数据采集	(384)
11.1	数据采集基础	(384)
11.1.1	DAQ 功能	(384)
11.1.2	DAQ 节点的组织与结构	(385)
11.1.3	DAQ VIS 的组织结构	(386)
11.1.4	DAQ 节点常用参数简介	(388)
11.2	DAQ 设备的安装与配置	(391)
11.2.1	安装 PCI-1200 数据采集卡	(392)
11.2.2	配置 PCI-1200 数据采集卡	(392)
11.2.3	配置数据采集虚拟通道	(396)
11.3	DAQ 编程	(399)
11.3.1	简易 DAQ 编程	(399)
11.3.2	扩展 DAQ 编程	(402)
11.3.3	高级 DAQ 编程	(408)
第 12 章	仪器控制	(412)
12.1	仪器驱动器	(412)
12.1.1	VPP 仪器驱动器	(412)
12.1.2	IVI 仪器驱动器	(415)
12.2	VISA 标准	(416)
12.3	VISA 编程	(418)
12.3.1	VISA 节点	(419)

12.3.2	VISA 编程实例	(423)
12.4	VPP 驱动程序转换与编程	(424)
12.4.1	VT1432A 数字化仪简介	(425)
12.4.2	VPP 驱动程序转换	(425)
12.4.3	VPP 驱动程序编程实例	(428)
第 13 章	通信	(431)
13.1	串行通信	(431)
13.1.1	串口简介	(431)
13.1.2	串行通信节点	(434)
13.1.3	串行通信编程举例	(437)
13.2	网络通信	(439)
13.2.1	TCP 协议简介	(439)
13.2.2	TCP 节点	(440)
13.2.3	TCP 通信编程实例	(441)
13.3	DataSocket 通信	(443)
13.3.1	DataSocket 基本概念	(443)
13.3.2	DataSocket 节点	(446)
13.3.3	DataSocket 编程举例	(449)
13.4	远程面板	(452)
13.4.1	配置 LabVIEW Web Server	(453)
13.4.2	在 LabVIEW 环境中操作 Remote Panel	(454)
13.4.3	通过网页浏览器在网页中操作 Remote Panel	(456)
参考文献		(458)

第1章 绪 论

1.1 LabVIEW 概述

1.1.1 LabVIEW 起源

LabVIEW 是实验室虚拟仪器集成环境 (Laboratory Virtual Instrument Engineering Workbench) 的简称, 是美国国家仪器公司 (NATIONAL INSTRUMENTS™, 简称 NI) 的创新软件产品, 也是目前应用最广、发展最快、功能最强的图形化软件开发集成环境。

数据采集、仪器控制、过程监控和自动测试是实验室研究和工业自动化领域广泛存在的实际任务。在 20 世纪 80 年代初个人计算机出现之前, 几乎所有拥有程控仪器的实验室都采用贵重的仪器控制器来控制测试系统, 这些功能单一、价格昂贵的仪器控制器通过一个集成通信口来控制 IEEE-488 总线仪器 (也称为 GPIB 程控仪器)。后来, 随着 PC 的出现, 工程师和科学家们找到了一种通过性能价格比高的通用 PC 控制台式仪器的方法, 各种基于 PC 的接口板卡产品迅速地打开了市场, NI 公司也应运而生。1983 年, NI 公司已经成为世界上 PC GPIB 接口卡最主要的供应商。

到 1983 年, GPIB 总线事实上已经成为连接仪器和计算机的通用标准接口。除了不同仪器制造商对 IEEE-488 标准的个别解释不同之外, 用户在物理上配置仪器和仪器系统基本上已没有问题。不过, 仪器控制软件的发展, 仍然存在许多问题。当时几乎所有的仪器控制程序都是由 BASIC 语言编写的。虽然与可读性差、编程专业性要求更高的机器语言和汇编语言相比, BASIC 语言已经具有许多优势 (如简单、可读性强的命令集和交互能力), 但与其他基于文本的高级语言一样, 它也存在一个根本问题, 即要求使用仪器的科学家、工程师和技术人员成为程序员。这些用户必须将他们关于仪器和应用的知识转化成一行行的程序代码, 以形成测试程序。这个过程经常是费时费力的苦差事, 尤其是对当时那些很少编程或基本没有编程经验的测试工程师更是如此。

NI 公司有一支用 BASIC 语言开发仪器程控软件的程序员专门队伍, 因此, 它十分敏锐地感觉到程控仪器编程为工程师和科学家带来的负担, 清楚地意识到需要开发一个用于程控仪器编程的软件工具。NI 公司的创始人杰姆·特鲁查德博士、杰夫·柯德斯凯博士和他们的好友杰克·麦克里森组成了一个小组, 开始研究开发这个新的软件工具, 希望这个新的软件工具能够改变工程师和科学家从事测试开发的方式。他们首先想到的软件工具模型是电子表格软件, 电子表格软件解决了特鲁查德博士、柯德斯凯博士和麦克里森三人希望解决的共同问题: 使计算机更容易被非程序员的计算机用户使用。当然, 电子表格软件主要是为财务人员设计的, 而特鲁查德博士小组设想的软件工具是为从事测试和仪器控制的工程师和科学家服务的。

1984 年, 当时财政实力还相对较弱的 NI 公司决定投资启动该软件工程项目, 特鲁查

德博士负责研究工作，并成立了一个基金会，任命柯德斯凯为项目实施人。为了克服办公室日常事务干扰，为创新和灵感创造一个安静环境，柯德斯凯在得州首府奥斯汀得州大学附近安营扎寨，这样一方面可以利用得州大学的图书馆资源，另一方面也便于雇用学生程序员。测试与仪器领域一个重要事件就此拉开序幕。

1.1.2 LabVIEW 概念创新

LabVIEW 的概念雏形来源于特鲁查德和柯德斯凯两人 20 世纪 70 年代末期在 ARL (Applied Research Laboratory, 应用研究实验室) 完成的一个大型测试系统。该系统主要用于测试美国海军的声呐探测器，研究人员也可用该系统开展水声学实验研究。这套测试系统的应用十分灵活，因为为各层次用户提供了不同的交互接口。技术人员可以在某些预先确定的限制条件下，操作测试系统完成指定的测试任务；水声工程师可以进入低层设备设计测试过程；而研究人员权限最大，他们可以进入系统所有可编程硬件，配置需要的测试系统。该测试系统也存在两个缺点：一是系统需要超过 18 人一年；二是用户必须理解选单上复杂的命令缩语。

通过几年的时间，柯德斯凯把从该测试系统得到的启示发展到测试系统软件由多层虚拟仪器 (Virtual Instruments, 简称 VI) 构成的新概念。一个 VI 可以由更低层的多个 VI 组成，就像真实仪器由印制电路板组成，而印制电路板又由集成电路 (IC) 组成一样。底层 VI 代表了最基本的软件功能——计算与输入/输出 (I/O)，操作。柯德斯凯特别重视多层软件的互连与嵌套，创造性地提出了各层 VI 都有相同结构形式的思想。在硬件领域，将 IC 装配为电路板和将电路板装配为仪器的技术是完全不同的；在软件领域，由语句组成子程序和子程序组成程序也有差别，更不用说由多个程序构成一个系统。所以柯德斯凯提出的所有层次 VI 一致性的结构和接口模型，极大地简化了软件结构，是对传统结构化软件设计思想的一个新发展。

虚拟仪器模型的另一个主要特征是每一个 VI 都有一个用户接口组件 (以下称 VI 前面板)，也就是与实际仪器面板相对应的软面板。在当时传统的编程语言里，即使是简单的命令行用户接口，在核心程序完成后也还要增加一系列复杂的输入输出语句，何况是设计复杂的仪器操作面板，但是 VI 前面板概念模型的提出改变了这一状况。VI 前面板接口已成为整个软件模型不可分割的一部分，用户通过打开 VI 前面板，就可以在系统的任何层次上与 VI 交互，并且前面板对象的设计与修改不涉及程序结构和源代码的变化，使得在开发过程中一步步调试软件模块和定位编程错误更为方便。

柯德斯凯是一个 UNIX 系统程序员，当 1983 年 Apple 公司支持图形化操作界面的 Macintosh 个人计算机问世后，柯德斯凯灵机一动，他心目中可以模拟真实仪器前面板的 VI 图形化操作面板可以实现了。应该说，图形化操作系统的出现，为 LabVIEW 的实现奠定了技术基础，然而，VI 仅仅有容易操作的图形化前面板还不够，还必须在编程技术上有大的突破，柯德斯凯再次想到了电子表格软件。电子表格软件采用财务人员最熟悉的数据表格和公式来实现财务软件编程，广大工程师又是如何设计软件的呢？最基本的办法是首先按照系统要求设计流程图，再将流程图转化为具体的程序代码，当然，这个转化过程需要许多编程技巧。柯德斯凯设想构造设计一种基于框图的编程方法，既便于用户概念设计，其功能和灵活性又足以作为编程语言使用。

在分析比较了几种框图编程方法的优劣后,柯德斯凯决定采用数据流图作为编程工具。数据流图长期以来一直被认为是顶层软件设计的有效工具,但一般的数据流图并没有提供循环结构、顺序结构和条件结构等程序设计的基本要素。柯德斯凯扩展了数据流图功能,使它能够处理循环、顺序和条件等程序控制,并在此基础上提出了结构化数据流程图模型。1990年他的结构化数据流图和虚拟仪器面板获得了两项美国专利。

在VI模型、图形界面和结构化数据流图编程等核心技术确定后,编程实现相对容易多了。柯德斯凯用4个月时间组织了一支软件开发队伍,在Macintosh上开始编程工作。在开发过程中,麦克里森表现了卓越的项目组织管理能力,提出了整个软件设计的关键数据结构与模块关系,加快了整个软件开发进程。柯德斯凯领导的开发小组克服了程序溢出和内存不足等困难,于1986年5月推出LabVIEW Beta测试版,又经过几个月的反馈修改,于1986年10月正式发布了LabVIEW 1.0版。LabVIEW最初吸引的仅仅是没有任何编程语言经验的用户,这些用户相信采用LabVIEW就能实现有经验的程序员也难完成的应用程序。

有效的内存管理是使图形化编程语言优于普通解释语言的关键。由于数据流解释需要大量分配内存,因此,内存重用对数据流图编程效率至关重要,寻找内存重用的有效算法成为提高LabVIEW性能的关键。LabVIEW 1.1版解决了算法问题,随后改进的LabVIEW 1.2版是可靠性和鲁棒性很强的产品,但由于内在体系局限,其性能与C语言程序相比,仍然有较大差距。为了解决这个问题,1988年开始的LabVIEW 2.0采用了最新的面向对象编程(OOP)技术。当1990年1月LabVIEW 2.0发送给第一个热心用户时,LabVIEW程序在执行速度和灵活性等方面的改进令人惊叹。与普通编程语言采用编译、连接等步骤生成应用程序不同,LabVIEW 2.0编译器是整个软件集成和不可见的部分,编译速度极快。

LabVIEW 2.0以前的版本都是运行在Macintosh平台上的,在Windows 3.0操作系统出现,32位Windows应用程序设计成为可能后,LabVIEW才实现了从Macintosh平台到Windows平台的移植。1992年8月,跨平台的LabVIEW 2.5问世。1993年1月,增加了大量新特性的LabVIEW 3.0正式发行,这些新特性包括全局与局部变量、属性节点和执行动画。从LabVIEW 3.0版本开始,LabVIEW作为一个完整优异的图形化软件开发环境得到了工业界和学术界的认可,并开始迅速占领市场,赢得了广大用户的青睐。

1.2 G语言与虚拟仪器

从LabVIEW研制开发的过程可以看到,虽然LabVIEW本身是一个功能比较完整的软件开发环境,但它是为替代常规的BASIC或C语言而设计的,LabVIEW是编程语言而不仅仅是一个软件开发环境。作为编写应用程序的语言,除了编程方式不同外,LabVIEW具备语言的所有特性,因此又称之为G语言。

G语言是一种适合应用于任何编程任务,具有扩展函数库的通用编程语言。和BASIC或C语言一样,G语言定义了数据模型、结构类型和模块调用语法规则等编程语言的基本要素,在功能完整性和应用灵活性上不逊于任何高级语言,同时,G语言丰富的扩展函数库还为用户编程提供了极大的方便。这些扩展函数库主要面向数据采集、GPIB和串行仪器控制,以及数据分析、数据显示和数据存储。G语言还包括常用的程序调试工具,比如允许设置断点、单步调试、数据探针和动态显示执行程序流程等功能。G语言与传统高级编程语言最大的差别在于编程方式,一般高级语言采用文本编程,而G语言采用图形化编程方式。

G 语言编写的程序称为虚拟仪器 VI (Virtual Instruments), 因为它的界面和功能与真实仪器十分相像, 在 LabVIEW 环境下开发的应用程序都被冠以 VI 后缀, 以表示虚拟仪器的含义。一个 VI 由前面板、数据流框图程序和图标连接端口组成, 各部分功能如下:

1. 前面板

前面板是 VI 的交互式用户接口, 与真实物理仪器面板相似。前面板可以包含旋钮、刻度盘、开关、图表和其他界面工具, 允许用户通过键盘或鼠标获取数据显示结果。

2. 数据流框图程序

VI 从数据流框图程序中接收指令, 框图程序是一种解决编程问题的图形化方法, 实际上是 VI 的程序代码。

3. 图标连接端口

VI 图标和连接端口的功能就像一个图形化参数列表, 可在 VI 与 SubVI 之间传递数据。一个 VI 既可以作为上层独立程序, 也可以作为其他程序 (或子程序) 的子程序。当一个 VI 作为子程序时, 称做 SubVI。

正是基于 VI 的上述特性, G 语言最佳地实现了模块化编程思想。用户可以将一个应用分解为一系列任务, 再将每个任务细分, 将一个复杂的应用分解为一系列简单的子任务, 为每个子任务建立一个 VI, 然后, 把这些 VI 组合在一起完成最终的应用程序。因为每个 SubVI 可以单独执行, 所以很容易调试。进一步而言, 许多低层 SubVI 可以完成一些常用功能, 因此, 用户可以开发特定的 SubVI 库, 以适用一般的应用程序。

G 语言是 LabVIEW 的核心, 熟练掌握 G 语言的编程要素和语法规则, 是开发高水平 LabVIEW 应用程序最重要的基础。换句话说, 要真正掌握 LabVIEW 开发工具, 必须把它作为一个编程语言, 而不仅仅是一个编程环境来学习, 这正是本书着力强调并贯穿于全过程的重点内容。

虚拟仪器概念是 LabVIEW 的精髓, 也是 G 语言区别于其他高级语言最显著的特征。正是由于 LabVIEW 的成功, 才使虚拟仪器的概念为学术界和工程界广泛接受; 反过来也正是因为虚拟仪器概念的延伸与扩展, 才使 LabVIEW 的应用更加广泛。

1.3 LabVIEW 应用解决方案

LabVIEW 自 1986 年正式推出至今不到 19 年的时间内, 已经从最初单一图形化编程功能、单一运行平台发展到目前以最新版本 LabVIEW 7 Express 为核心, 包括控制与仿真、高级数字信号处理、统计过程控制、模糊控制和 PID 控制等众多附加软件包, 运行于 Windows NT/2000、Linux、Macintosh、Sun 和 HP-UX 等多种平台的工业标准软件开发环境。在美国, 许多工科大学已将 LabVIEW 作为课堂或实验室教学内容, 作为工程师素质培养的一个方面。不同领域的科学家和工程师都借助这个易用的软件包来解决工作中的各种应用课题。

LabVIEW 在包括航空、航天、通信、汽车、半导体和生物医学等世界范围的众多领

域内得到了广泛应用,从简单的仪器控制、数据采集到尖端的测试和工业自动化,从大学实验室到工厂,从探索研究到技术集成,都可以发现应用 LabVIEW 的成果和开发产品。

1. LabVIEW 应用于测试与测量

LabVIEW 已成为测试与测量领域的工业标准,通过 GPIB、VXI、PLC、串行设备和插卡式数据采集板可以构成实际的数据采集系统。它提供了工业界最大的仪器驱动程序库,同时还支持通过 Internet、ActiveX、DDE 和 SQL 等交互式通信方式实现数据共享,它提供的众多开发工具使复杂的测试与测量任务变得简单易行。

2. LabVIEW 应用于过程控制和工业自动化

LabVIEW 强大的硬件驱动、图形显示能力和便捷的快速程序设计,为过程控制和工业自动化应用提供了优秀的解决方案。对于更复杂更专业的工业自动化领域,在 LabVIEW 基础上发展起来的 BridgeVIEW 是更好的选择。

3. LabVIEW 应用于实验室研究与自动化

LabVIEW 为科学家和工程师提供了功能强大的高级数学分析库,包括统计、估计、回归分析、线性代数、信号生成算法、时域和频域算法等众多科学领域,可满足各种计算和分析需要。即使在联合时域分析、小波和数字滤波器设计等高级或特殊分析场合,LabVIEW 也为此提供了专门的附加软件包。

LabVIEW 是一个具有高度灵活性的开发系统,用户可以根据自己的应用领域和开发要求选择 LabVIEW 系统配置。NI 公司为不同层次用户提供了 3 种系统配置:

(1) LabVIEW 基本版

LabVIEW 基本版是用于开发数据采集和仪器控制系统的最小 LabVIEW 配置,包括 VISA、GPIB、RS-232、DAQ 和基本分析库,同时还包括支持 ActiveX、TCP/IP 和 DDE 等标准程序的接口。

(2) LabVIEW 完整版 (FDS)

除了基本版的功能外,FDS 还包括完整的高级分析库。

(3) LabVIEW 专业版 (PDS)

LabVIEW 专业版除了 FDS 的功能外,还具有专业程序员开发时所需要的全部工具,包括可执行文件生成工具、源代码控制、复杂矩阵分析、软件工程文档管理、质量控制标准文档、图形差异比较和大型软件项目管理文档工具等。

对一般用户而言,采购 LabVIEW 完整版,并根据实际应用选取专门的 LabVIEW 工具套件是最佳选择。表 1.3.1 列出了需单独购买的 LabVIEW 主要工具包。

除了表 1.3.1 列出的 LabVIEW 工具包以外,还有许多第三方软件开发商设计开发的大量定制 VI 供用户选择。实际上 LabVIEW 已经成为工业标准,形成了广泛的用户群体和专业开发人员,有力地促进了 LabVIEW 技术本身的发展与进步。从 LabVIEW 4.01 版本更新到 LabVIEW 7 版本,几乎一年更新一个版本就是 LabVIEW 技术发展和广泛应用最显著的标志。

表 1.3.1 需单独购买的主要 LabVIEW 工具包

LabVIEW 工具包名称	功 能
Signal Processing Toolkit	包括数字滤波器与联合时频分析工具包, 小波及滤波器组设计工具箱
Sound and Vibration Toolkit	声音和震动信号分析工具包
Order Analysis Toolkit	阶次分析工具包
Modulation Toolkit	调制工具包
PID Control Toolkit	提供 P、PI、PD 和 PID 控制算法并直接驱动硬件输出
Express VI Development Toolkit	Express VI 开发工具包
State Diagram Toolkit	状态图工具包, 支持从状态图直接生成代码
VI Analyzer Toolkit	VI 分析工具包, 提供自动测试工具以改进 VI 的设计
Motion Control Toolkit	运动控制工具包
Simulation Interface Toolkit	仿真与控制工具包
Database Connectivity Toolkit	数据库访问工具
Internet Toolkit	Internet 访问工具
SPC Toolkit	SPC 分析, 控制流程图, Pareto 分析
Vision and Image Processing Toolkit	高级图像分析工具
Report Generation Toolkit	报表生成工具

本书内容组织以 LabVIEW 7 Express 版本为基础。与之前的老版本相比, LabVIEW 7 Express 版本增加了许多先进功能和新特性, 下面首先对这些新功能、新特性作一概要性描述, 以便读者更好地理解本书增加的内容。

1.4 LabVIEW 7 Express 新特性

LabVIEW 7 Express 新增功能与特性概括起来可以归纳为: 进一步改进提高了编程功能与运行效率; 进一步简化了程序开发与管理; 进一步扩展了应用领域。总之, 这些新功能、新特性使得 LabVIEW 提供了创新的开发环境、全新的编程工具、交互式测量与运行方式选择, 同时具备强大的功能和方便的使用性能。这些新功能、新特性主要包括:

- 改进的 LabVIEW 启动界面, 如图 1.4.1 所示。单击 New 按钮将打开新建对话框, 该对话框包含一系列 VI 模板, 可创建基于特定模板的 VI。单击 Configure 按钮可以打开 Measurement & Automation Explorer (MAX)、创建 DAQmx 任务或创建 DAQmx 通道。

- Express VIs。又称快速 VI, 是针对常见的测试和测量应用而定制设计的 VI。一个 Express VI 通常综合了多个传统 VI 的功能于一身, 极大地简化和改进了用户的开发过程。Express VI 没有属性结点, 所有属性都通过对话框设置, 方便快捷。

- VI 模板。从主菜单中选择 File→New, 打开新建对话框, 对话框列出了内建的模板, 可以选择并创建基于这些模板的 VI。



图 1.4.1 LabVIEW 启动界面

- 增强的控件模板和功能模板。单击控件或功能模板的工具条上的 Options 按钮，可以选择模板的视图类型。用户也可以建立自己的模板视图，方法是从主选单中选择 Tools → Advanced → Edit Palette Views，然后对视图进行修改。
- 改进的断线提示功能。LabVIEW 将在错误连线上放置错误标识“×”，选择连线工具，将光标定位在存在错误的连线上，将显示错误提示信息，如图 1.4.2 所示。

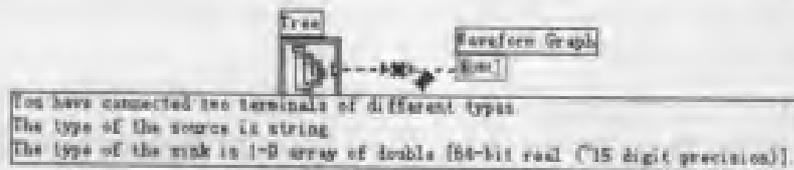


图 1.4.2 断线以及错误说明

- 属性对话框。LabVIEW 7 Express 前面板控件新增了属性设置对话框。可以通过属性对话框设置控件的外观和行为。方法是在控件的右键弹出选单中选择 Properties。
- 自动错误处理。当处于默认设置时，LabVIEW 在检测到错误的时候将挂起正在运行的程序，高亮显示出错的子 VI，并弹出一个错误对话框。
- 用户可以设置前面板对象的端口在框图中的显示方式。显示方式有两种：图标 (Icon) 和传统数据类型端口 (Data Type Terminal)，如图 1.4.3 所示。在端口图标的右键弹出选单中选择 View As Icon，可以在这两种方式之间切换。



图 1.4.3 前面板控件在框图中的两种显示方式

- 自动布线功能。用户无须考虑两个 VI 的相对位置，直接使用连线工具连接相应端口即可。数据线在连线途中遇到其他 VI 会自动绕行。
- 结构 VI 的自动缩放功能。在使用 While 循环、For 循环等结构时，如果结构内部的 VI 过于靠近结构的边框，结构的边框将自动放大。当手动缩小结构的边框时，边框将不能覆盖内部对象，而不像以前那样可以随意缩小结构。
- 平铺式顺序结构。不同于传统的层叠式顺序结构，平铺式顺序结构同时显示所有

的帧,它的外观看上去就像一卷展开的胶片,如图 1.4.4 所示。平铺式顺序结构的优点是不用创建顺序结构本地变量 (Sequence Local),并且更容易阅读和理解,当然在框图程序中它会占用更多的面积。



图 1.4.4 平铺式顺序结构

- NI 实例查找器 (NI Example Finder)。从主菜单中选择 Help → Find Examples 可以打开 NI 实例查找器,通过 NI 实例查找器可以查找已安装和 Web 上的 LabVIEW 例程。

- LabVIEW 数据目录。当安装 LabVIEW 时,安装程序将自动创建 LabVIEW 数据目录,用以存储系统产生的数据文件。

- DAQ 助手 (DAQ Assistant)。DAQ 助手通过图形界面配置 NI-DAQmx 测试任务,可以通过单击对话框中的 Configure 按钮打开 DAQ 助手,也可以通过 DAQ Assistant 节点打开,该节点是 Express VI,位于 Functions 模板 → All Functions 子模板 → NI Measurements 子模板 → DAQmx-Data Acquisition 子模板 → DAQ Assistant。

- 仪器 I/O 帮手 (Instrument I/O Assistant)。仪器 I/O 帮手用于与串口、以太网或 GPIB 接口仪器通信。启动仪器 I/O 帮手的方法是将 Instrument I/O Assistant Express VI 放置到框图程序中。Instrument I/O Assistant 位于 Functions 模板 → Input 子模板。

- I/O 控件。使用 I/O 模板中的控件涵盖了 Motion 设备、FieldPoint 设备和 NI-DAQmx 设备。

- .NET 函数。使用 .NET 函数可以创建、设置、操作 .NET 对象。

- 网格线。无论前面板窗口还是框图程序窗口,都可以显示网格线用以对齐不同的对象。

- 改进的工具自动选择功能。单击工具模板中的工具可以取消自动选择功能。

- 单位标签 (Unit Label)。对于一个数字控件,可以在显示数据的同时显示其单位。若要编辑一个单位标签,首先显示它,然后在其右键弹出菜单中选择 Build Unit String。显示了单位的数字,在计算时将自动进行单位的换算,如图 1.4.5 所示。



图 1.4.5 带有单位的计算过程

- 改进的打印和报告生成 VI。使用 VI Documentation VIs 模板中的 VI 设置报告。

使用 Query Available Printers 节点列出可用的打印机。

- 树型控件。使用树型控件可以显示层级列表结构（如目录树），树型控件的外观如图 1.4.6 所示。
- 子面板。使用子面板可以在当前 VI 的前面板内显示另一个 VI 的前面板，如图 1.4.7 所示。

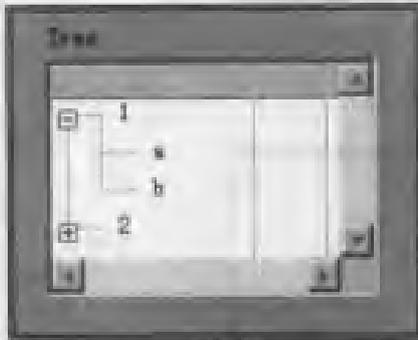


图 1.4.6 树型控件



图 1.4.7 子面板

- 改进的下拉列表框 (Ring Control)。右键单击下拉列表框，从快捷菜单中选择 Edit Items 可以编辑下拉列表框的内容，可以为每个项目指定独立的数值。
- 组合框 (Combo Box)。组合框是新增的控件类型，主要用于提供字符串的列表。
- 动态注册事件和用户事件。新的事件结构 (Event Structure) 可以处理动态注册事件和用户自定义事件。
- 升级的多态 VI。多态 VI 类似于其他编程语言的多态函数。创建多态 VI 的方法是，从主菜单中选择 New...，然后从对话框中选择 Other Document Types → Polymorphic VI。
- 时间标识控件 (Time Stamp Control)。时间标识控件可以使用、查看、存储高精度的时间数据。以秒为单位，时间标识数据类型可以拥有 15 位精度的整数部分和 15 位精度的小数部分。
- 自定义探针。如果觉得系统提供的探针不够方便，可以设计自己的探针。方法是在数据线的右键弹出菜单中选择 Custom Probe。图 1.4.8 显示了一个自定义的探针，该探针显示信号的 FFT 功率谱。

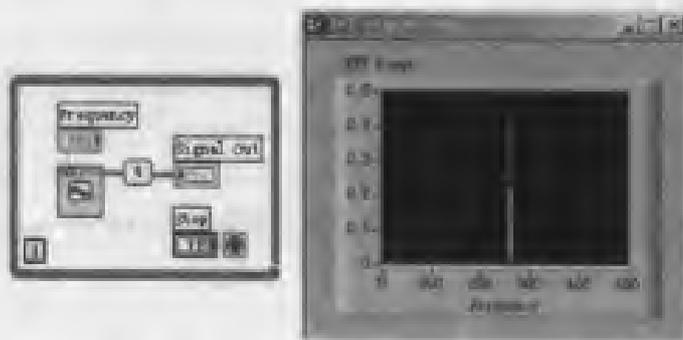


图 1.4.8 使用自定义探针显示信号 FFT 功率谱

- 改进的图片控件 (Picture Control)。使用 Create Mask 节点可以创建图片遮罩，使

用 Picture to Pixmap 节点可以将图片数据转换为图像数据簇 (Image Data Cluster), 从而可以进行特定操作, 如保存图片。使用 Get Image Subset 节点可以获取图像的一部分内容。

- 改变光标的外观。使用光标节点可以设置前面板中光标的外观。
- 反馈节点。在以前版本的 LabVIEW 中, 一个 VI 的输出端口和输入端口是不能直接相连的。有了反馈节点, 在 For 循环或 While 循环中, 就可以通过反馈节点将子 VI 的输出端口和输入端口连接起来了, 如图 1.4.9 所示。反馈节点的功能类似于移位寄存器, 保存前一次循环的数据, 并将它传递到下一次循环。通过使用反馈节点, 连线将更加容易, 看起来也更加直观。

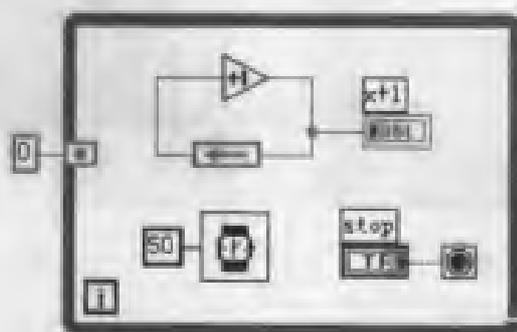


图 1.4.9 利用反馈节点在循环间传递数据

- 通过电子邮件发送数据。通过 SMTP E-mail 节点可以将数据和文件以电子邮件的方式在网络上传递。
- 缓存 DataSocket 数据。如果需要读出 DataSocket Server 发布的所有数据而不是最新的数据, 客户端必须缓存数据。
- 通过编程打开或关闭 DataSocket 连接。使用 DataSocket Open 和 DataSocket Close 节点在程序内打开 DataSocket 连接。通过 DataSocket 属性, 如 Buffer Maximum Bytes, Buffer Maximum Packets, Buffer Utilization (Bytes), Buffer Utilization (Packets) 设置 DataSocket 缓存。通过 Connection Status 属性确认 DataSocket 连接状态。通过 URL 属性读取 DataSocket URL。
- 处理 ActiveX 事件。LabVIEW 并没有提供 ActiveX 事件节点。要使用 ActiveX 事件必须用 Register Event Callback 节点注册和处理 ActiveX 事件。
- 快速参考卡片。提供了有关快捷键、数据类型等相关信息。
- Application Builder 用户手册。提供了有关使用 Application Builder 的一些建议。
- 新的 VI Server 属性和方法。新增了多项 VI Server 的属性和方法。
- 输入设备控制节点。使用输入设备控制节点以获得来自鼠标、键盘、游戏摇杆等输入设备的信息。输入设备控制节点位于 Functions 模板 → All Functions 子模板 → Advanced 子模板 → Input Device Control 子模板中。

第 2 章 LabVIEW 编程环境

本章概要介绍了 LabVIEW 系统的组成、LabVIEW 的组成

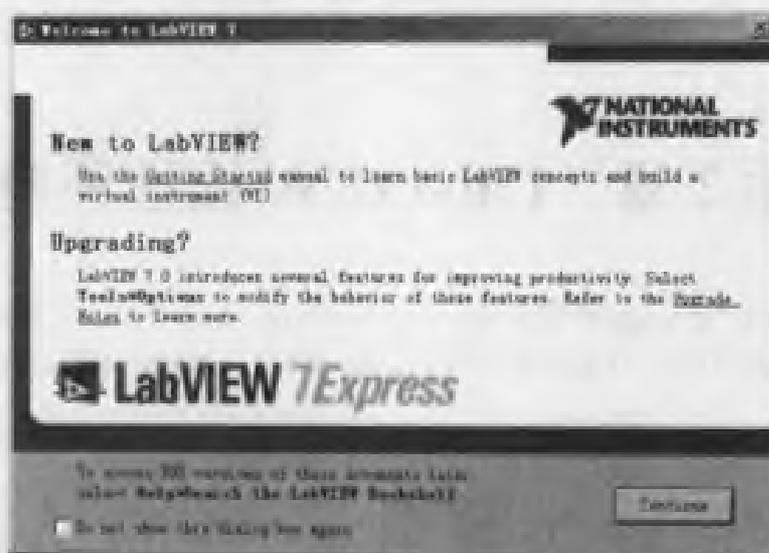


图 2.2.1 LabVIEW 7 Express 的启动界面 1

单击 Continue 按钮进入第二个启动界面, 如图 2.2.2 所示。



图 2.2.2 LabVIEW 7 Express 的启动界面 2

界面右侧有 4 个按钮, 每个按钮都包含按钮主体和下拉选单。单击按钮主体则弹出相应的对话框, 单击按钮主体右边的下拉按钮则弹出下拉选单。各按钮功能如表 2.2.1 所示。

表 2.2.1 LabVIEW 启动界面上的按钮功能表

按 钮	功能说明
New...	创建一个空白 VI 或者从模板生成一个 VI。LabVIEW 7 Express 提供了几种常用的 VI 模板, 如果选择从模板创建 VI, 那么将直接生成包含基本框架的 VI
Open...	打开一个最近操作过的 VI 或者打开一个例程
Configure...	设置 Measurement and Automation Explorer 或 LabVIEW
Help...	查看包括 VI 说明、查找例程、错误代码说明、网络资源等帮助信息

当用户单击 New VI 按钮右边的下拉按钮, 然后在下拉选单中选择 Blank VI, LabVIEW 会生成一个空 VI。空 VI 包括两个无标题 (Untitled) 窗口, 一个是前面板窗口, 用于编辑和显示前面板对象; 另一个是框图程序窗口, 用于编辑和显示框图程序 (程序代码)。两个窗口选单条设计基本一样, 除了框图程序窗口增加了 4 个用于程序调试的工具按钮以外, 两个窗口工具条的设计也是一致的。以下将以框图程序窗口为例, 说明窗口工具条和窗口弹出式选单的布局与功能。

1. 窗口工具条

窗口工具条中各图标的功能如表 2.2.2 所示。

表 2.2.2 窗口工具条功能一览表

图 标	名 称	功 能 说 明
	执行按钮	单击此按钮运行 VI
	中断按钮	当执行按钮消失, 错误列表按钮出现时, 表明 VI 有错, 不能编译运行, 单击该按钮, 可弹出 Error List 对话框, 为用户提示 VI 中的错误
	连接运行按钮	单击此按钮, 可重复运行 VI
	停止运行按钮	单击此按钮, 可强行停止 VI 执行
	暂停按钮	单击此按钮, 可暂停 VI 执行, 再单击此按钮, VI 又继续执行
	指示灯按钮	单击此按钮, 可动态显示 VI 执行时数据流动的动画
	单步 (入) 按钮	单击此按钮, 不仅按节点顺序单步执行, 而且在节点内也单步执行
	单步 (跳) 按钮	单击此按钮, 按节点顺序单步执行, 不进入循环, SubVI 等节点内部执行单步运行
	单步 (出) 按钮	单击此按钮, 退出循环, SubVI 等节点内部执行单步模式
	对齐列表框	分布列表框, 为选定的两个或多个对象提供左、右、上、下对准选项, 以美化界面设计
	间隔列表框	为选定的两个或多个对象提供间隔排列控制
	重新排序列表框	为选定的对象重新设定在窗口中的前后次序
	帮助按钮	显示实时在线帮助

2. 窗口主选单

LabVIEW 窗口的选单条包括文件 (File)、编辑 (Edit)、操作 (Operate)、工具 (Tools)、浏览 (Browse)、窗口 (Window) 和帮助 (Help) 6 大项。

(1) File

除了与常规 Windows 应用程序一样的文件操作和打印设置选项外, 下面有几条是 LabVIEW 独有的, 如表 2.2.3 所示。

表 2.2.3 LabVIEW 窗口 File 选单中特有的选项

选单选项	功能说明
Print Window...	打印当前窗口
Save with Options...	有选择的保存 VI, 可以对 VI 加密码、删除框图程序、转化为 LabVIEW6.1 文件格式或将 VI 保存为模板
VI Properties...	设置当前 VI 的各种属性。包括优先级、安全保护、窗口外观等

(2) Edit

Edit 选单中除了一般的编辑功能外, 需要特别介绍的是一些如表 2.2.4 所示的选单功能。

表 2.2.4 LabVIEW 窗口 Edit 选单中的一些选项

选单选项	功能说明
Customize Control ...	激活前面板控件编辑器, 允许用户修改控件并代替默认标准控件
Scale Object with Panel	使控件前面板尺寸随着窗口尺寸成比例变化
Set Tabbing Order...	设置使用 Tab 键切换焦点的次序
Import Picture from File...	从文件导入图片
Remove Broken Wircs	删除框图中所有错误连接线
Create SubVI	将框图中选中的对象转化为 SubVI
Run-time Menu...	编辑运行时选单。定制用户需要的的选单项

(3) Operate

Operate 选单中除了一目了然的 Run 和 Stop 选项外, 其他运行控制项如表 2.2.5 所示。

表 2.2.5 LabVIEW 窗口 Operate 选单中的一些选项

选单选项	功能说明
Suspend When Called	当 VI 被调用时暂停执行
Print at Completion	VI 运行完后打印 VI 前面板
Log at Completion	VI 运行完后将数据记录写入文件
Data Logging...	选择数据记录选项
Make Current Values Default	改变前面板对象的当前值称为默认值
Reinitialize All to Default	重新恢复前面板对象的默认值
Enable Alignment Grid on Panel/Diagram	在前面板或框图上显示网格线
Change to Run Mode	在运行和编辑模式之间切换
Connect to Remote Pannel	连接到远程面板

(4) Tools

LabVIEW6.1 中有 Project 选单, LabVIEW 7 Express 取消了 Project 选单, 将 Project 选单中的若干内容放在了 Tools 选单中。除了通用的查找指定对象功能外, Tools 选单还提供了一些专门功能, 如表 2.2.6 所示。

表 2.2.6 LabVIEW 窗口 Tools 选单中的一些选项

选单选项	功能说明
Instrumentation	仪器驱动程序的更新和导入
Compare	比较两个 VI, 显示差异或进行层级结构比较
Source Code Control	提供多种源代码控制功能
VI Revision History	VI 版本修改记录
User Name...	用户名设定
Build Application or Shared Library (DLL)	建立应用程序或动态连接库。此选项只有在安装了 Application Builder 工具包之后才会出现
VI Library Manager...	VI 库管理器
Find VIs on Disk	查找磁盘上的 VI
Edit VI Library...	VI 库编辑器
Remote Panel Connection Manager...	管理远程面板的连接
Web Publishing Tool...	网络发布工具
Advanced	包括 Mass Compile、错误代码编辑、ActiveX 控件导入、ActiveX 控件属性浏览等
Options...	多种选项设定

(5) Browse

Browse 选单提供了多种浏览 VI 信息的方法, 具体说明见表 2.2.7。

表 2.2.7 LabVIEW 窗口 Browse 选单中的一些选项

选单选项	功能说明
Show VI Hierarchy...	以层级图的方式显示 VI 的构成
This VIs SubVIs...	显示 VI 包含的 SubVI
This VIs Callers...	显示 VI 的调用者
Unopened SubVIs...	显示没有打开的 SubVI
Unopened Type Defs...	显示没有打开的类定义
Break Points	查看 VI 中所设置的断点

(6) Windows 和 Help

Windows 和 Help 选单所列功能意义明确, 主要用于 LabVIEW 环境下各种窗口的管理和帮助信息的获取, 这里不再详述。

2.3 LabVIEW 模板

与一般 Windows 程序相比, LabVIEW 提供了 3 个浮动的图形化模板, 分别是工具模板、控件模板和功能模板。这 3 个模板功能强大、使用方便、表示直观, 是用户编程的主要工具。

1. 工具模板



图 2.3.1 工具模板

工具模板包含了一系列的工具, 主要用于 VI 的创建、修改和调试, 以及操作前面板对象。

在 LabVIEW 窗口主选单中选择 Windows → Show Tools Palette, 可以打开工具模板 (Tools Palette), 如图 2.3.1 所示。

用鼠标拖动模板窗口的标题栏可以移动工具模板的位置。将鼠标放置在某一个工具的图标上, LabVIEW 会自动显示该工具的功能说明。LabVIEW 7 Express 的工具模板还提供了工具自动切换按钮。在操作时可以根据鼠标相对于控件的位置自动选择合适的工具。表 2.3.1 总结了工具模板中各个工具的名称和功能。

表 2.3.1 工具模板功能一览表

图 标	名 称	功 能
	Automatic Test Selection	自动选择工具, 根据鼠标相对于控件的位置自动选择合适的工具
	Operation Value	数据操作工具, 用于操作前面板对象的数据, 或选择对象内的文本或数据
	Position/Size/Select	对象操作工具, 用于选择对象, 移动对象或缩放对象的大小
	Edit Text	文本编辑工具, 用于在对象中输入文本或在窗口中创建注释
	Connect Wire	连线工具, 用于在框图程序中节点端口之间连线, 或定义 SubVI 的端口
	Object Shortcut Menu	弹出选单工具, 用于弹出右键快捷选单, 与单击鼠标右键作用相同
	Scroll Window	滚动窗口工具, 同时移动窗口内所有的对象
	Set/Clear Breakpoint	断点工具, 用于在框图程序内设置或清除断点
	Probe Data	探针工具, 用于在框图程序内的数据连线上设置数据探针
	Get Color	颜色拷贝工具, 获取对象上某一点的颜色
	Set Color	颜色工具, 利用在颜色选择对话框中选择的颜色, 或由颜色拷贝工具获得的颜色给对象上色

2. 控件模板

控件模板 (Controls Palette) 包含了 LabVIEW 中所有的前面板对象, 例如, 数字控件、温度计、指示灯、按钮、开关等。用户可以通过控件模板 (Controls Palette) 创建前面板对象, 控件模板按前面板对象的数据类型分类, 包含一系列控件子模板。

控件模板位于 VI 的前面板窗口中, 在前面板窗口的主选单中选择 Windows → Show Controls Palette, 可以弹出控件模板, 在 VI 前面板窗口的空白处单击鼠标右键, 也可以弹出控件模板, 如图 2.3.2 所示。



图 2.3.2 控件模板

LabVIEW 7 Express 的控件模板与较早版本的 LabVIEW 相比,最大的特色就是增加了一些 Express 前面板对象,这些 Express 前面板对象按照其数据类型和操作属性被分散到 8 个 Express 子模板中。

Express 的含义是快速,即用户可以一种极为快捷的方式创建和使用 LabVIEW 前面板对象或框图程序中的功能。有关 Express 的内容,将在本书第 3 章进行介绍。

Controls 模板中第 1 层的前 8 个子模板为 Express 子模板,只放置 Express 前面板对象。表 2.3.2 总结了控件模板中的 Express 子模板的主要功能。

表 2.3.2 控件模板中的 Express 子模板功能一览表

图 标	名 称	功 能
 Num Cnls	Numeric Controls	数字控制子模板,存放 Express 数字控制前面板对象,例如,滚动条、按钮、旋钮盘
 Num Inds	Numeric Indicators	数字指示子模板,存放 Express 数字指示前面板对象,例如,进程条、表头、水银、温度计
 Buttons	Buttons & Switches	按钮和开关子模板,存放 Express 布尔控制前面板对象,例如,翘臂开关、滑动开关、拨动开关、按钮开关
 LEDs	LEDs	LED 指示灯子模板,存放 Express 布尔指示前面板对象,例如,方形 LED 指示灯、圆形 LED 指示灯
 Text Cnls	Text Controls	文本控制子模板,存放 Express 字符串控制前面板对象,例如,字符串控制、路径控制
 Text Inds	Text Indicators	文本指示子模板,存放 Express 字符串指示前面板对象,例如,字符串指示、路径指示
 Graph Inds	Graph Indicators	波形图指示子模板,存放 Waveform Chart、Waveform Graph 和 Express Waveform XY Graph 等 3 个 Express 前面板对象
 User Cnls	Express User Controls	Express 用户控件子模板,存放用户定制的 Express 前面板对象
 All Controls	All Controls	所有控件子模板,存放 LabVIEW 所有的前面板对象,这个子模板不是 Express 子模板,不包含 Express 前面板对象

Controls 模板中的第 9 个子模板,即 All Controls 子模板中存放了 LabVIEW 所有的前面板对象,如图 2.3.3 所示。

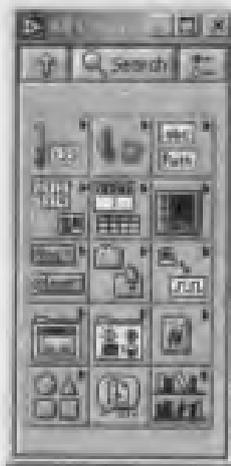


图 2.3.3 All Controls 子模板

表 2.3.3 总结了 All Controls 子模板各部分的主要功能。

表 2.3.3 控件模板中的 All Controls 子模板功能一览表

图 标	名 称	功 能
	Numeric	数字子模板, 提供各种表示数字量的控制与指示对象, 包括数字量、温度计、刻度盘和按钮等多种形式
	Boolean	布尔子模板, 提供各种表示布尔量的控制与指示对象, 包括各种类型的开关、按钮和指示灯等
	String & Path	字符串与文件路径子模板, 提供表示字符串、路径与组合列表框控件。
	Array & Cluster	数组与簇子模板, 提供表示数组与簇的控制与指示对象
	List & Table	列表框和表格子模板, 提供包括列表框、多列列表框、树型控件内的多种控件
	Graph	图形子模板, 提供各种形式的图形显示对象, 包括实时趋势图、事后记录图、XY图和密度图等形式
	Ring & Enum	下拉列表框和枚举控件子模板, 提供文本下拉列表框、选单形式的下拉列表框、枚举列表框、图形列表框以及图形和文本组合列表框
	Containers	容器子模板, 提供页框控件、子面板控件和 ActiveX 容器控件
	IO	提供与仪器 I/O 相关的控件, 包括波形控件、数字波形控件、数字数据控件、传统 DAQ 通信控件、DAQmx 名称控件、VISA 资源名称控件、IVI 逻辑名称控件、现场总线 I/O 节点控件、DMAQ 回调控件、Motion 资源控件
	Dialog Controls	对话框子模板, 提供各种 Windows 标准对话框控件
	Classic Controls	经典控件模板, LabVIEW 7 Express 版本以前的 LabVIEW 控件均可在此子模板中找到, 方便了喜欢使用经典控件的用户。
	Refnum	标识号子模板, LabVIEW 使用标识号来区别各种对象, 标识号的概念类似于 C 语言的句柄
	Decorations	修饰子模板, 于修饰和定制前面板的图形对象
	User Controls	用户控件子模板, 存放用户定制的前面板对象
	Select Control	选择控件子模板, 提供控件选择对话框, 以便装载用户定制的前面板对象

3. 功能模板

功能模板 (Functions Palette) 包含了 LabVIEW 中所有的功能节点, 这些节点用于创建 LabVIEW 的框图程序, 功能模板位于 VI 的框图程序窗口中。LabVIEW 框图编程所需的所有功能节点按照功能分类, 分布在功能模板的各个子模板里。

在框图程序窗口的主菜单中选择 Windows → Show Functions Palette, 可以弹出功能模板, 如图 2.3.4 所示。在 VI 前面板窗口的空白处单击鼠标右键, 也可以弹出功能模板。注意, 功能模板与控件模板有一点区别在于控件模板是在前面板窗口, 而功能模板是在框图程序窗口。图 2.3.4 所示的是 LabVIEW 7 Express 的 Express 风格的功能模板。



图 2.3.4 功能模板

功能模板中第一层前 7 个子模板是 Express 风格的子模板, 用于存放各种 Express 节点。表 2.3.4 总结了功能模板中的 Express 子模板的主要功能, 有关 Express 子模板中的 Express 节点的使用, 将在本书第 3 章中进行介绍。

表 2.3.4 功能模板中的 Express 子模板功能一览表

图 标	名 称	功 能
	Input	Express 子模板, 存放用于控制各种仪器输入的 Express 节点
	Signal Analysis	Express 信号分析子模板, 存放用于对数字信号进行各种分析的 Express 节点
	Output	Express 子模板, 存放用于控制各种仪器输出的 Express 节点
	Execution Control	Express 运行控制子模板, 存放用于控制 VI 运行的各种结构, 延时的 Express 节点
	Arithmetic & Comparison	Express 运算和比较子模板, 存放用于数学运算, 布尔运算及比较运算的 Express 节点
	Signal Manipulation	Express 信号操纵子模板, 存放用于对波形数据进行操纵的 Express 节点。
	Express User Libraries	Express 用户 VI 库子模板, 存放用户定制的 Express VI
	All Functions	所有功能子模板, 存放 LabVIEW 所有的功能节点、VI, 这个子模板不是 Express 子模板

功能模板中的第 8 个子模板, 即 All Functions 子模板存放了 LabVIEW 所有的功能节点, 如图 2.3.5 所示。

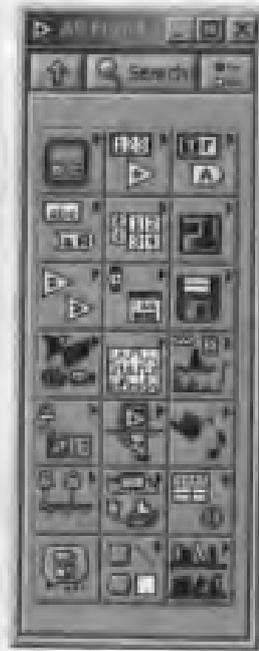


图 2.3.5 All Functions 子模板

表 2.3.5 总结了功能模板各个子模板的功能与作用, 至于每个子模板内的节点及其使用, 则是 LabVIEW 编程最基本最重要的内容, 本书将在后续章节中详细讲述。

表 2.3.5 功能模板中的 All Functions 子模板功能一览表

图 标	名 称	功 能
	Structures	结构子模板, 提供循环、条件、顺序结构、公式节点、全局与局部变量等 LabVIEW 编程要素
	Numeric	数值子模板, 提供数学运算、标准数学函数、各种常量和数据类型变换等 LabVIEW 编程基础模块
	Boolean	布尔子模板, 提供包括布尔运算符和布尔常量在内的编程元素
	String	字符串模板, 提供字符串运算、字符常量和特殊字符在内的编程元素
	Array	数组子模板, 提供数组运算和变换的功能模块
	Cluster	簇子模板, 提供簇运算和变换
	Comparison	比较子模板, 提供用于数字量、布尔量和字符串变量比较运算的功能模块
	Time & Dialog	时间与对话框子模板, 提供各种定时、时间数据处理、对话框和错误出口处理模块
	File I/O	文件 I/O 子模板, 提供文件管理、变换和读/写操作模块

续表

图 标	名 称	功 能
	NI Measurements	提供各种与数据采集相关的 VI, 需要单独安装
	Waveform	波形生成子模板, 提供波形数据生成 VI, 包括波形数据创建、通道信息设置、波形提取、波形存储等 VI
	Analyze	分析子模板, 包括信号分析和数学两部分, 信号分析部分包括时域、频域、滤波器设计和信号发生、逐点分析等功能模块; 数学计算部分包括线性性和非线性方程求解、微积分、统计分析、优化设计和线性代数等高级分析功能
	Instrument I/O	仪器 I/O 子模板, 提供用于串行、GPIB 和 VISA 仪器控制的 VI 模块以及仪器驱动 VI, LabVIEW 7 Express 中, 将仪器驱动 VI 子模板放在仪器 I/O 子模板中
	Application Control	应用程序控制子模板, 提供外部程序或 VI 调用和打印选项, 帮助管理等辅助功能
	Graphics & Sound	图形和声音子模板, 进行 3D 图形处理、绘图及声音的处理
	Communication	通信子模板, 提供支持 TCP, UDP, DDE, OLE, Active X 和启动外部程序的模块
	Report Generation	报表生成子模板, 用于生成各种报表
	Advanced	高级子模板, 提供库函数调用、代码接口节点、数据管理、内存管理和程序标志管理等高级功能
	User Libraries	用户库子模板, 存放用户定制的 VI
	Decorations	修饰子模板, 提供文字注释、箭头、线条等工具, 可用于在框图中添加注释说明
	Select a VI	选择 VI 子模板, 插入从对话框中选择的 SubVI 或者全局变量

为方便讲述 LabVIEW, 本书后续的内容在涉及到上述 3 个模板时, 统一采用名称“Tools 模板”、“Controls 模板”和“Functions 模板”。

2.4 VI 库

在 LabVIEW 环境下编写的 VI 既能以单独文件形式存放在指定任意的目录下, 也可以采用 VI 库形式保存。VI 库是指包含一系列 VI 文件的集合, 其后缀采用 .LLB。与普通文件保存方式相比, VI 库方式具有以下特点。

- VI 库中的文件支持 255 个字符的标准文件名。
- VI 库中的文件由于压缩可以节省磁盘空间 (LabVIEW 打开 VI 库中文件时会自动解压)。
- 多个 VI 集中在一个文件中, 便于程序的移植, 尤其是将构成一个系统的多个 VI 存放在一个库里, 对程序管理维护也是十分有利的。
- VI 库不支持树状层次结构, 即在一个 VI 库内只能存放 VI 而不能创建新的 VI 库。
- 在文件系统中保存和打开 VI 比在 VI 库中保存和打开 VI 要快。

要创建一个新的 VI 库, 在选择 Save、Save as ... 或 Save a Copy as ... 保存 VI 时, 单击文件保存窗口的 New VI Library 按钮, 然后在指定文本框输入中 VI 库文件名即可。VI 库的操作与文件夹完全一样, 利用 LabVIEW 窗口选单 Tools → Edit VI Library... 和 Tools → VI Library Manager... 选项, 可以对 VI 库进行管理, 在本书的第 10 章将详细讲述其用法。

2.5 定制 LabVIEW 环境

当 LabVIEW 安装完毕第一次启动后, LabVIEW 环境的所有窗口、模板和系统运行参数都按照默认设置, 用户可以根据需要, 以最适合的形式和要求设置 LabVIEW 的环境参数和编程工具。

2.5.1 环境参数设置

在 LabVIEW 窗口主选单中选择 Tools → Options, 弹出 LabVIEW 的 Options 对话框, 如图 2.5.1 所示, 在一个下拉选单中, 列出各种选项, 选中任何一个选项, 系统都会自动列出一系列可选设置或参数供给用户选择。

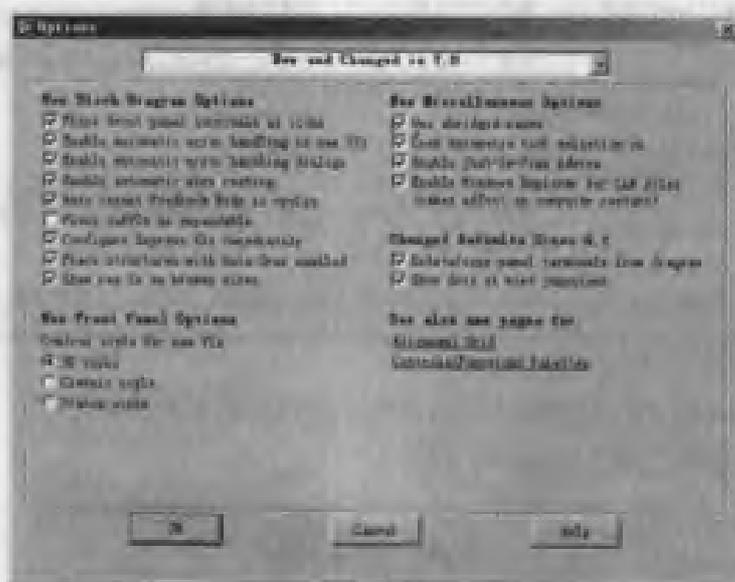


图 2.5.1 Options 对话框

表 2.5.1 按照 Options 对话框上部的下拉选单中的选项简单介绍 Options 对话框的功能, 这些功能的使用将在本书的第 10 章进行介绍。

表 2.5.1 Options 对话框功能一览表

Options 下拉选单选项	功 能
New and Changed in 7.0	列出 LabVIEW 7.0 版中新添加的功能
Paths	设置 LabVIEW 的临时文件和临时 VI 库的路径, 设置 LabVIEW 搜索 VIs 的文件夹顺序
Performance and Disk	设置内存的分配
Front Panel	设置前面板对象的各种选项
Block Diagram	设置框图程序窗口中对象的各种选项
Alignment Grid	设置所有 VIs 窗口中的对齐网格选项
Controls/Functions Palettes	设置 Controls 模板和 Functions 模板的各种选项
Debugging	设置用于框图程序调试的各种选项
Colors	设置 LabVIEW 环境的各种颜色选项
Fonts	设置应用程序字体 (application font)、对话框字体 (dialog font) 和系统字体 (system font) 的默认值
Printing	设置 LabVIEW 的打印参数
Revision History	设置历史记录窗口的运行参数。在 LabVIEW 窗口的主选单中选择 Tools→VI Revision History, 可以打开历史记录窗口
Miscellaneous	设置前面板窗口和框图程序窗口中对象的其他杂项参数
VI Server : Configuration	配置 VI Server
VI Server : TCP/IP Access	指定可以通过 VI Server 访问本地 VIs 的外地计算机的 TCP/IP 地址
VI Server : Exported VIs	指定可以通过 VI Server 访问的 VIs
Web Server : Configuration	激活、配置 Web Server.
Web Server : Browser Access	设置网络浏览器访问 Web Server 的权限
Web Server : Visible VIs	指定在网络上可见的 VIs

2.5.2 模板设置

1. 在 User 子模板中添加内容

在 Controls 模板和 Functions 模板中分别有一个 User Controls 子模板和 User Libraries 子模板, 这两个子模板用于存放用户定制的前面板对象和 VI。在 LabVIEW 安装完之后, 这两个子模板是空的, 没有任何内容, 需要用户添加。

LabVIEW 规定这两个子模板中的内容存放在 LabVIEW 安装目录中 User.lib 文件夹中, 因此, 只要将用户定制的前面板对象和 VI 存放到 User.lib 文件夹中, 就可以在这两个子模板中找到并调用它们。

2. 使用模板编辑器

在主选单中选择 Tools→Advanced→Edit Palette Views, 可以弹出模板编辑器对话框, 如图 2.5.2 所示。首先从模板集中选择指定的模板类型, LabVIEW 提供了 Express, Data Acquisition 和 Advanced 等 3 种形式的模板, 在对话框的 Palette View 栏的下拉选单中可以在这 3 种形式的模板之间切换。这 3 种形式的模板中包含的内容完全一样, 只是其中的子

模板的摆放顺序不同。

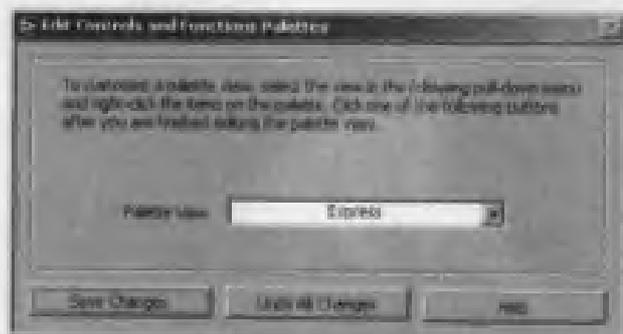


图 2.5.2 模板编辑器对话框

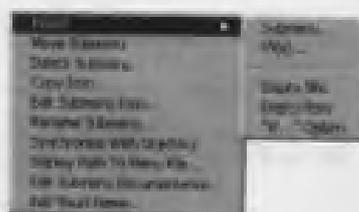


图 2.5.3 子模板的右键弹出选单

另外, Palette View 栏的下拉选单中还有一个选项: New Setup...。利用这个选项,用户可以定制自己的模板。在用户定制的模板中,可以任意编辑子模板,例如,插入子模板、移动子模板、删除子模板、拷贝子模板图标、编辑子模板图标及更改子模板名称等,这些功能都可以通过选择子模板的右键弹出选单中的选项实现,如图 2.5.3 所示。

3. 设定模板的显示方式

在主选单中选择 Tools→Options..., 在弹出的 Options 对话框的 Controls/Functions Palettes 页面中可以设定模板的显示方式,如图 2.5.4 所示。

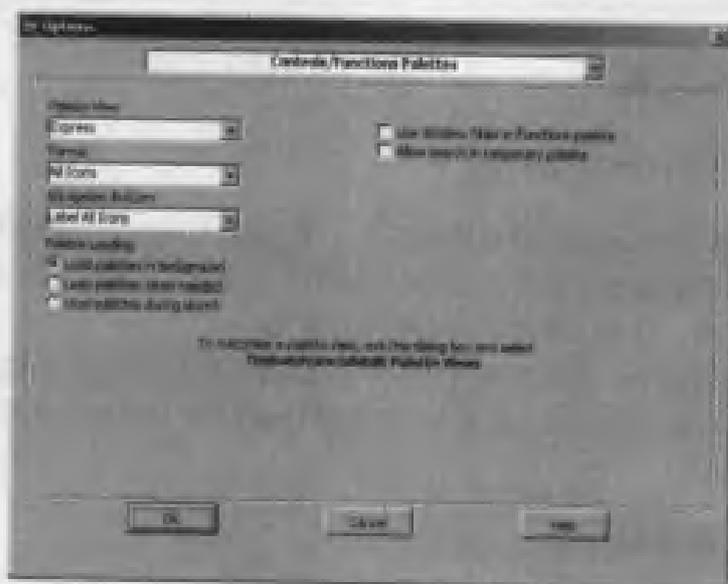
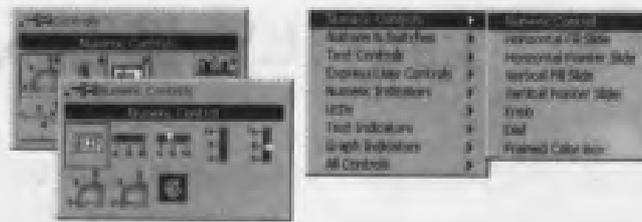


图 2.5.4 Options 对话框的 Controls/Functions Palettes 页面

在 Controls/Functions Palettes 页面中可以设定 Controls 模板和 Functions 模板的显示方式为 Express 方式或 Advanced 方式,也可以设定模板中显示的内容为以下格式: Standard、

Standard (Icons and Text)、All Icons、All Text 或 Icons and Text。例如, All Icons 显示模式下的 Controls 模板和 All Text 显示模式下的 Controls 模板如图 2.5.5 所示。



(a) All Icons 显示模式

(b) All Text 显示模式

图 2.5.5 All Icons 显示模式和 All Text 显示模式下的 Controls 模板

第3章 LabVIEW 编程入门

本章主要介绍 LabVIEW 的一些基本概念和术语, 创建和编辑 VI, 创建和调用 SubVI, 以及运行和调试 VI 的基本方法和如何获取帮助等, 为初学者提供一个基本的编程思路和简单指导, 为深入学习 LabVIEW 编程原理和技巧打下基础。

3.1 基本概念

在深入学习 LabVIEW 之前, 有必要先介绍一些 LabVIEW 的基本概念, 这是理解与学习 LabVIEW 的基础。

LabVIEW 程序称为虚拟仪器 (Virtual Instrument) 程序, 简称 VI。一个最基本的 VI 由 3 个部分组成: 前面板 (Panel)、框图程序 (Diagram Programme) 和图标/连接端口 (Icon/Terminal), 如图 3.1.1 所示。

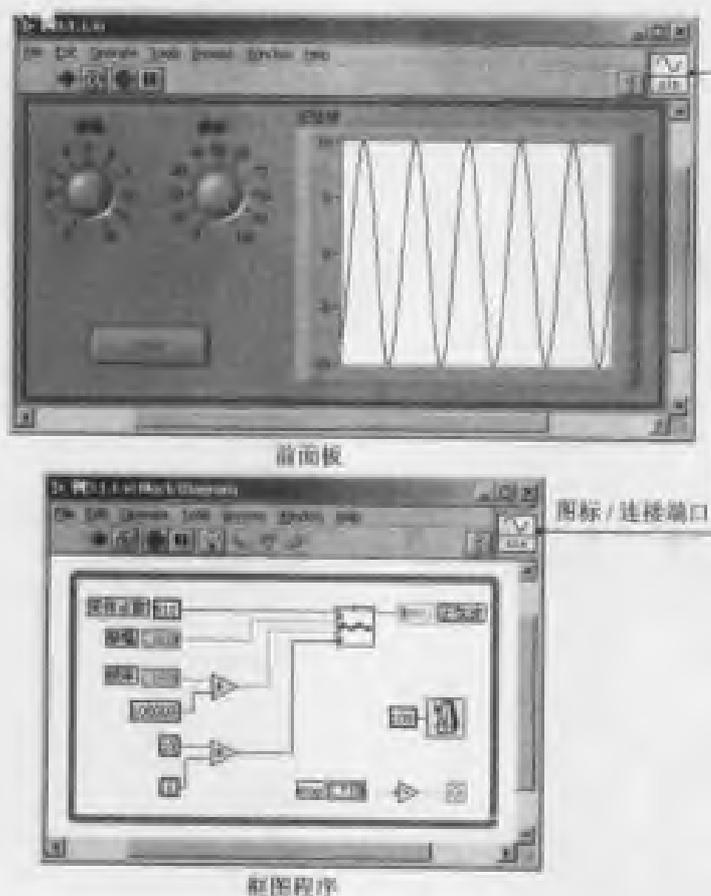


图 3.1.1 VI 的组成



3.1.1 前面板

前面板就是图形化用户界面，用于设置输入数值和观察输出量，可以模拟真实仪器的前面板。前面板由控制（Control）、指示（Indicator）和装饰（Decoration）构成。为了方便起见，本书将前面板中的控制和指示统称为前面板对象或控件。

1. 控制（Control）

控制是用户设置和修改 VI 程序中输入量的接口，控制在某种意义上相当于 C 语言中的输入语句 `scanf`。

在 LabVIEW 中，控制以图形化的图标形式出现，例如数字控制、旋钮、开关、按钮、滑动条等，如图 3.1.2 所示。用户可以通过鼠标或键盘更改控制中的值。

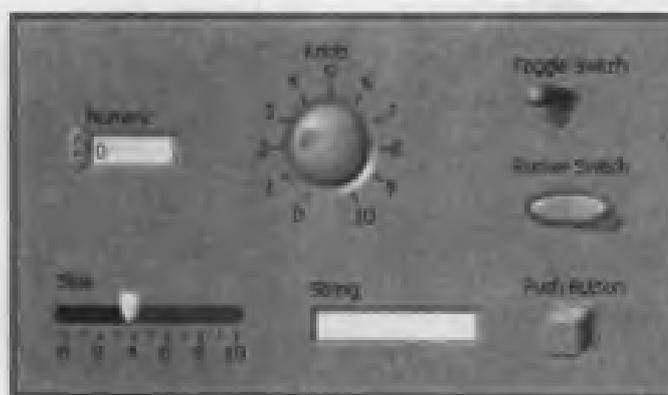


图 3.1.2 LabVIEW 中的控制

2. 指示（Indicator）

指示用于显示由 VI 程序运行产生的输出数据，指示某种意义上相当于 C 语言中的输出语句 `printf`。

在 LabVIEW 中，指示也是以图形化的图标形式出现的，例如速度表、温度计、水箱、LED 指示灯、进程条、波形图等，如图 3.1.3 所示。

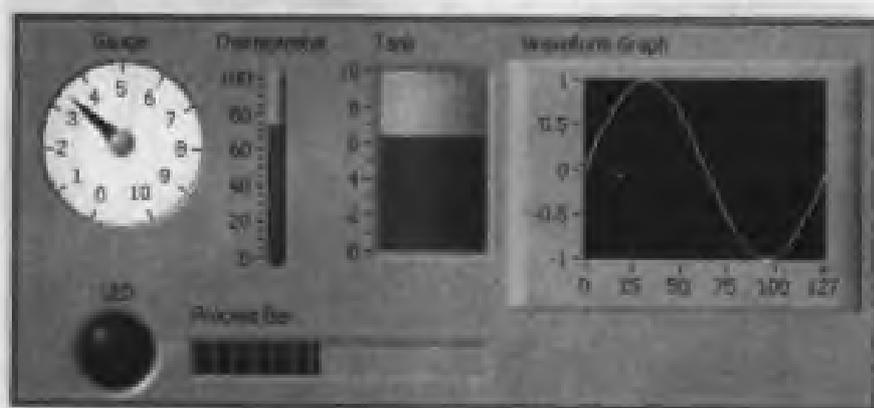


图 3.1.3 LabVIEW 中的指示

注意, 指示只能用于将 VI 程序运行产生的输出数据在前面板窗口中以不同的形式显示出来, 在 VI 处于运行状态时, 用户不能通过鼠标或键盘更改指示中的值。

任何一个前面板对象都有控制和指示两种属性, 在前面板对象的右键弹出菜单中选择 Change to Indicator 或 Change to Control, 可以在这两种属性之间切换。请注意, 如果用于输入的前面板对象被设置为指示, 或用于输出的前面板对象被设为控制, 则 LabVIEW 会报错。

3. 装饰 (Decoration)

装饰的作用仅是将前面板点缀得更加美观, 并不能作为 VI 的输入或输出使用。在 Controls 模板中专门有一个 Decorations 子模板, 子模板中含有各种装饰图形, 例如线条、箭头、矩形、圆形、三角形等, 如图 3.1.4 所示。当然, 用户也可以直接将外部图片 (BMP 或 JPEG 格式) 粘贴到前面板中作为装饰。



图 3.1.4 LabVIEW 中的装饰

注意, 当 VI 处于编辑状态时, 只可以对 LabVIEW 中的装饰进行改变大小和颜色等两项操作, 不能对其进行编程; 当 VI 处于运行状态时, 不能对装饰进行任何操作。

在前面板中, 用户可以使用各种前面板对象, 如旋钮 (Knob)、按钮 (Button)、开关 (Switch)、实时趋势图 (Waveform Chart) 和事后记录图 (Waveform Graph) 及 Windows 标准控件等, 创建一个模拟的仪器面板, 就像真实的仪器面板一样。图 3.1.5 所示的就是一个简单的正弦波发生器的前面板。

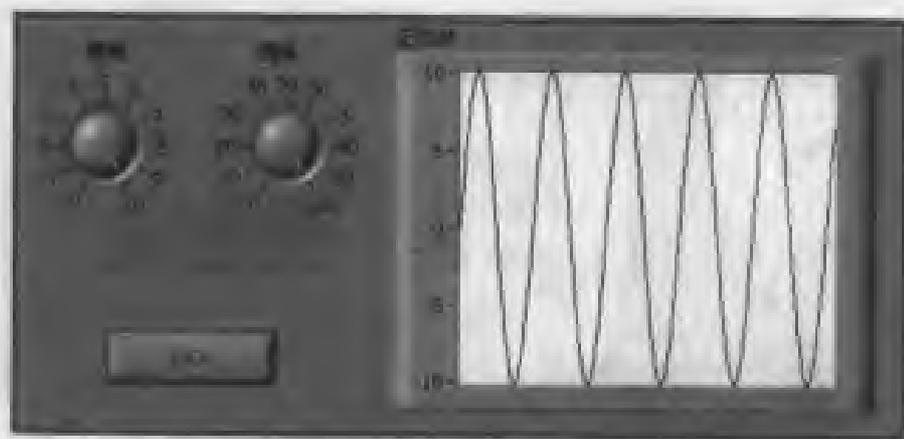


图 3.1.5 正弦波发生器的前面板

3.1.2 框图程序

每一个前面板都有一个框图程序与之对应。框图程序用图形化编程语言（G 语言）编写，可以把它理解成传统编程语言程序中的源代码。用图形而不是用传统的文本代码进行编程是 LabVIEW 最大的特色。

框图程序由节点（Node）、端口（Terminal）和数据连线（Wire）组成。

1. 节点

节点是 VI 程序中的执行元素，类似于文本编程语言程序中的语句、函数或者子程序。节点之间由数据连线按照一定的逻辑关系相互连接，定义框图程序内的数据流动方向。上述正弦波发生器的框图程序就是一个典型的例子，如图 3.1.6 所示。

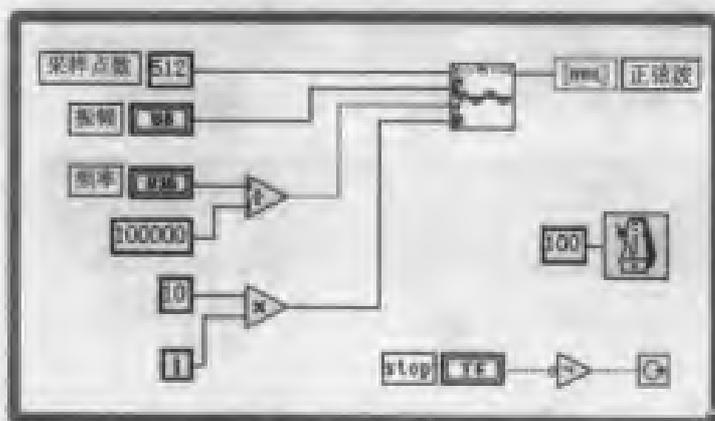


图 3.1.6 正弦波发生器框图程序

LabVIEW 共有 4 种类型的节点，如表 3.1.1 所示。

表 3.1.1 LabVIEW 节点类型表

节点类型	节点功能
功能函数 (Functions)	LabVIEW 内置节点，提供基本的数据与对象操作，例如数学运算、布尔运算、比较运算、字符串运算、文件 I/O 操作等
结构 (Structures)	用于控制程序执行方式的节点，包括顺序结构、选择结构、循环结构、事件结构及公式节点等
外部代码接口节点	LabVIEW 与外部程序的接口，包括调用库函数节点 (CLF)、代码接口节点 (CIN)、动态数据交换节点 (DDE) 等
子 VI (SubVI)	将一个已存在的 VI 以 SubVI 的形式调用，相当于传统编程语言中子程序的调用，通过 Functions 模板中的 Select a VI 子模板可以创建一个 SubVI 节点

2. 端口

节点与节点之间、节点与前面板对象之间是通过数据端口和数据连线来传递数据的。数据端口是数据在前面板对象与框图程序之间交互数据的接口，是数据在框图程序内节点之间传输的接口。

LabVIEW 中有两种类型的数据端口：前面板对象的端口和节点的端口。

(1) 前面板对象的端口

前面板对象的端口是前面板对象与框图程序交互数据的接口, 前面板对象的端口又分为控制端口和指示端口两种类型。

控制端口是控制在框图程序中的端口, 当 VI 程序运行时, 从控制输入的数据通过控制端口传递到框图程序中, 供其中的节点使用。

指示端口是指示在框图程序中的端口, VI 程序运行产生的输出数据, 通过指示端口传输到前面板中对应的指示中显示。

在框图程序中, 每一个前面板对象都有一个惟一的端口, 端口的名称与其相对应的前面板对象的名称相同, 如图 3.1.7 所示。当在前面板窗口中创建、删除控制或指示时, LabVIEW 会自动在框图程序中创建、删除与之相对应的控制端口或者指示端口。



图 3.1.7 前面板控制对象和端口

从图 3.1.7 可以看出, 控制端口和指示端口在外观上略有不同, 控制端口的边框用粗实线表示, 端口右侧有一个向右的箭头, 表示输出数据; 而指示端口的边框用细实线表示, 表示输入数据, 这样用户可以很容易地区分它们。

图 3.1.7 所示的端口外观是 LabVIEW 传统的表现形式, 从 LabVIEW 7 Express 版本开始, 出现了一种图标形式的端口外观, 如图 3.1.8 所示。



图 3.1.8 前面板控制对象和图标形式端口

在端口的右键弹出选单中选择 View As Icon，可以在这两种外观形式之间进行切换，如图 3.1.9 所示。

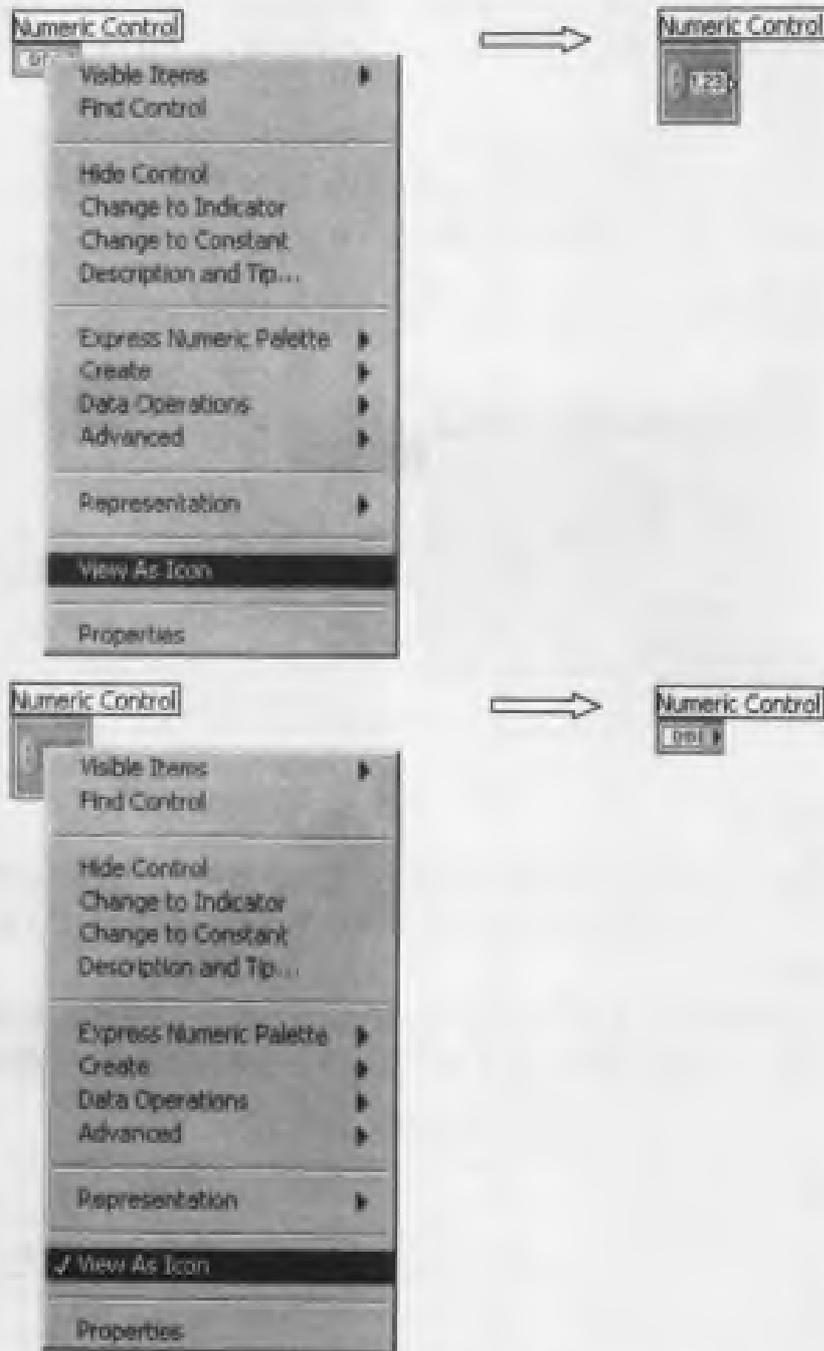


图 3.1.9 在端口的两种外观形式之间切换

若要设置所有新创建的控件的显示方式，可以从主选单中选择 Tools→Options，然后在弹出的 Options 对话框的下拉列表框中选择 Block Diagram，在 Block Diagram 页面中选中 Place front panel terminals as icons 选项。

从功能上讲，这两种外观形式的端口图标完全相同，其使用方法也完全相同，为了与以前版本的 LabVIEW 兼容，本书的实例一律采用传统的端口表现形式。

(2) 节点的端口

为了与外界交换数据, 框图程序中的每个节点都有一个或数个数据端口用以输入或输出数据, 节点的端口相当于传统文本编程语言中函数的参数。例如, 正弦函数 \sin 节点共有 2 个端口, 其中端口 x 为输入端口, 端口 $\sin(x)$ 为输出端口, 端口 x 相当于 C 语言中的函数 $\sin(x)$ 中的参数 x 。

在默认状态下, 节点的端口是不显示的, 在节点的右键弹出选单中选择 Visible Items \rightarrow Terminals, 可以将节点的所有端口显示出来, 如图 3.1.10 所示。在通常情况下, 节点左侧的端口为输入端口, 节点右侧的端口为输出端口。

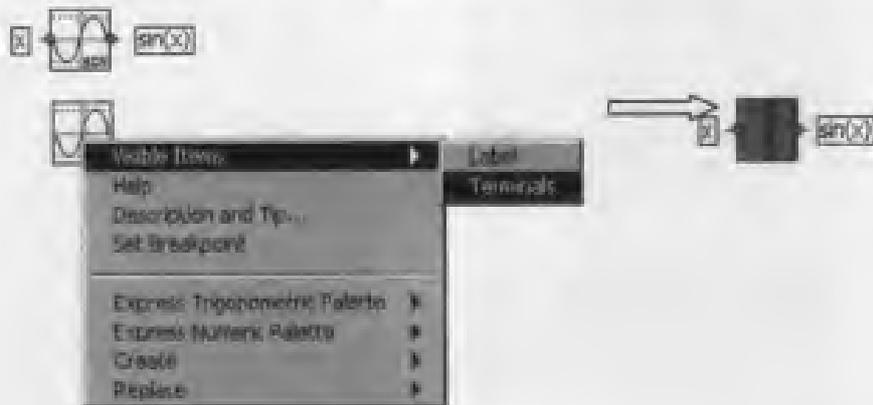


图 3.1.10 节点的端口

3. 数据连线

数据连线是端口与端口之间的数据传输通道, 它将数据从一个端口中传送到另一个与之相连的端口中。数据连线中的数据是单向流动的, 从源端口 (输出端口) 流向一个或多个目的端口 (输入端口)。

LabVIEW 利用数据连线传递各种不同类型的数据, 通过不同的线型和颜色来区分不同的数据类型。表 3.1.2 所示的是 LabVIEW 中几种常用的数据类型所使用的线型和颜色。

表 3.1.2 常用线型与颜色表

数据类型	标量	一维数组	二维数组	颜色
数字量	——	——	——	橙色 (浮点数) 蓝色 (整数)
布尔量	——	——	——	绿色
字符串	——	——	——	紫色
路径	——	——	——	青色
簇	——	——	——	紫色
波形数据	——	//	//	棕色
标识	——	——	——	青色
图片	——	——	——	蓝色

由于框图程序中的数据是沿数据连线并按照程序中的逻辑关系流动的, 因此, LabVIEW 的编程又可称为“数据流编程”, LabVIEW 的程序运行方式又可称为“数据流”。

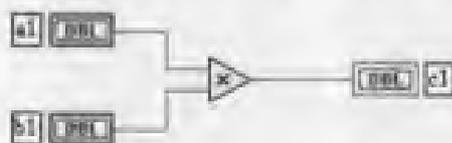
控制。对一个节点而言，只有当它所有输入端口所需要的数据都被提供以后，它才能够执行。节点程序运行完毕后，会把结果数据送到其输出端口中，这些数据将很快地通过数据连线送至目的端口。“数据流”与常规编程语言中的“控制流”相类似，相当于控制程序语句一步一步地执行。

如图 3.1.11 所示，这个 VI 程序把数字量控制 a 和 b 中的数值相加，然后再乘以 100，运算结果在数字量指示 c 中显示，即 $c = (a+b) \times 100$ 。



图 3.1.11 采用“数据流”控制的框图程序

在这个程序中，框图程序从左向右执行，但这个执行次序不是由其对象的摆放位置来确定的，而是由其逻辑关系来确定的。由于乘法节点的一个输入量是加法节点的运算结果，所以只有当加法节点的相加运算完成并把结果送到乘法节点的输入端口后，乘法节点才能执行下去。请记住，一个节点只有当其输入端口的所有数据均有效地到达后才能执行，而且只有它执行完成后，才会把结果送到输出端口。



再看另一个 VI 的框图程序，如图 3.1.12 所示。请判断哪一个节点先执行，是乘法节点还是除法节点。答案是这两个节点并行执行。这是由于这两个节点之间没有数据依赖关系，是相互独立的，它们的输入数据同时到达，同时满足上述节点执行的条件，因此，它们并行执行。并行执行是 LabVIEW 有别于传统的编程语言的特点之一。



图 3.1.12 并行执行的框图程序

但是，如果根据某种要求，需要先执行乘法运算，后执行除法运算，可以用顺序结构解决这个问题，这一点将在第 6 章详细介绍。

3.1.3 使用数据连线

在 LabVIEW 中，数据连线有一定的使用方法和规则，正确地使用数据连线是 LabVIEW 编程的基本要求之一，数据连线也是初学者进行 LabVIEW 编程时容易出错的地方。因此，下面详细介绍数据连线的使用方法、使用规则和一些应当注意的内容。

1. 手动连接数据连线

使用连线工具  可以在两个端口之间连接数据连线，手动连接数据连线通常按照下列步骤进行。

第一步，将端口或节点放到框图程序中。

第二步, 移动鼠标到一个节点的某一个端口上, 此时该端口会处于闪烁状态, 并且出现一个黄色小标识框 (tip strip), 显示该端口的名称, 如图 3.1.13 所示。黄色小标识框用于帮助用户将数据连线正确地连接到目的端口上, 该功能在连接一个有多个端口的节点时显得很方便。

第三步, 在该端口上单击鼠标左键, 然后移动鼠标, 此时会有一条从该端口引出的虚线跟随鼠标移动, 如图 3.1.14 所示。



图 3.1.13 连线工具位于 Add 节点输入端口上的情形

图 3.1.14 单击鼠标左键后移动鼠标时的情形



图 3.1.15 完成了一条数据连线的创建

第四步, 将鼠标移动到另外一个节点的端口上, 该端口处于闪烁状态, 并出现一个黄色小标识框, 单击该端口, 此时, 跟随鼠标的虚线会变成一条实线, 至此, 就完成了一条数据连线的创建, 如图 3.1.15 所示。

另外, 若需要从一条已经存在的数据连线引出一条数据连线分支, 有两种方法可以实现:

- 将连线工具移动到这条数据连线上, 当数据连线闪烁时, 单击鼠标左键, 然后拖动鼠标, 在鼠标的单击处就会引出一条数据连线分支。
- 将鼠标移动到这条数据连线上合适的位置, 在线上单击鼠标右键, 在弹出的快捷菜单中选择 Create Wire Branch, 在鼠标的单击处就会引出一条数据连线分支。

图 3.1.16 所示的就是一条引出的数据连线分支。

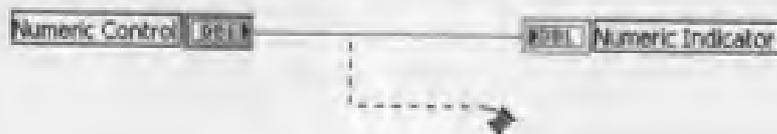


图 3.1.16 引出的数据连线分支

2. 自动连接数据连线

在 LabVIEW 7 Express 默认的编程环境中, 自动连线功能处于激活状态。此时, 当将一个节点放至框图程序中, 且其输入 (或输出) 端口与另外一个节点的同数据类型的输出 (或输入) 端口在位置上比较靠近时, LabVIEW 会自动利用数据连线将两个端口连接在一起。若两个节点的端口的属性相同, 即都为输入端口或都为输出端口, 或者两个端口的数据类型不同时, 即使这两个端口的位置靠得再近, LabVIEW 也不会将它们连接在一起。使用 LabVIEW 自动连线功能的步骤如图 3.1.17 所示。

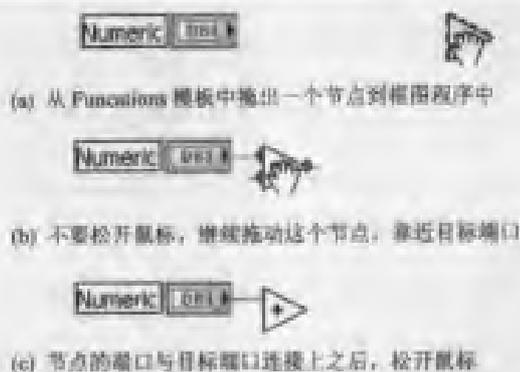


图 3.1.17 使用自动连线功能的步骤

注意，自动数据连线的功能只有在节点创建时有效，若移动一个在框图程序中已经存在的节点，则该功能无效。当节点处于图 3.1.18 (a) 所示的状态时，若按下空格键，则自动数据连线功能失效，此时节点图标变为一个虚线轮廓，如图 3.1.18 (b) 所示。

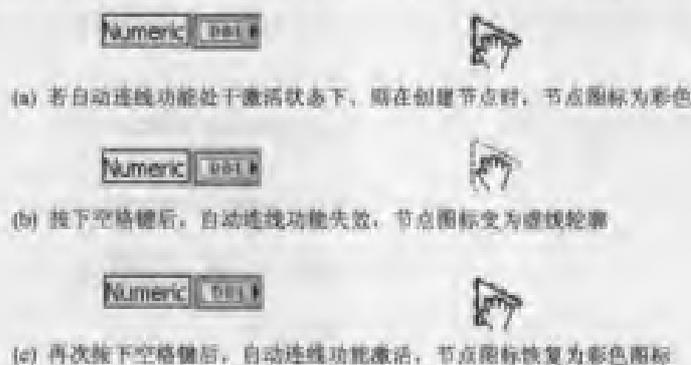


图 3.1.18 自动连线功能的失效与激活

3. 数据连线的有关操作

(1) 选中数据连线

首先将鼠标切换到对象操作工具状态，移动鼠标到目标数据连线上，然后按照下列 4 种方法之一进行操作。

单击鼠标左键，选中一段数据连线，操作结果如图 3.1.19 所示。

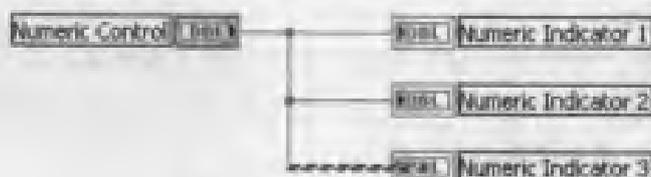


图 3.1.19 选中一段数据连线

双击鼠标左键，选中一个数据连线分支，操作结果如图 3.1.20 所示。

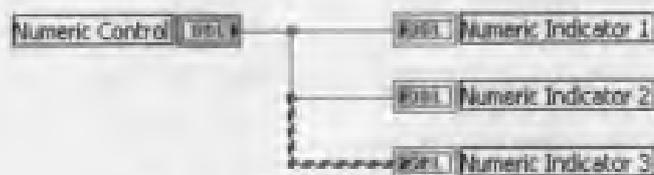


图 3.1.20 选中一段数据连线分支

三击鼠标左键，选中整个数据连线，操作结果如图 3.1.21 所示。

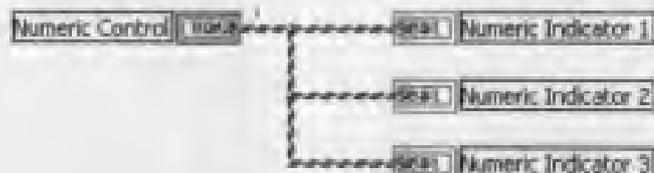


图 3.1.21 选中整个数据连线

按住 Shift 键的同时选中其他数据连线，可以同时选中多段数据连线，操作结果如图 3.1.22 所示。

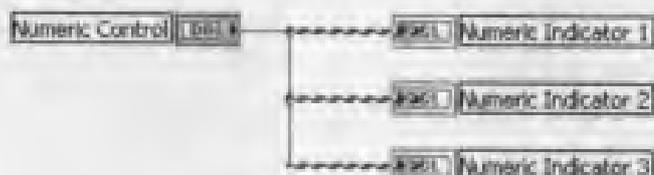


图 3.1.22 选中多段数据连线

(2) 删除数据连线

首先选中数据连线，然后按照下列三种方法之一操作：

- 按下 Delete 键或 Backspace 键；
- 在框图程序窗口主选单中选择 Edit→Clear；
- 在数据连线的右键弹出选单中选择 Delete Wire Branch。

另外，使用组合键“Ctrl+B”可以同时删除框图程序中所有的错误数据连线。请慎用这种方法，因为这种方法有时可能会删除用户所不希望删除的错误数据连线，从而造成一些不必要的麻烦。

(3) 移动数据连线

首先选中数据连线，然后单击鼠标左键，拖动数据连线到指定的位置。注意，当利用这种方式移动数据连线时，与被移动的数据连线相连接的其他数据连线分支会自动调整其位置，以适应被移动的数据连线位置的变化。

(4) 数据连线的路线

当用连线工具创建了一条数据连线时，在鼠标拖动数据连线的过程中，按下空格键可以改变数据连线的路线，如图 3.1.23 所示。

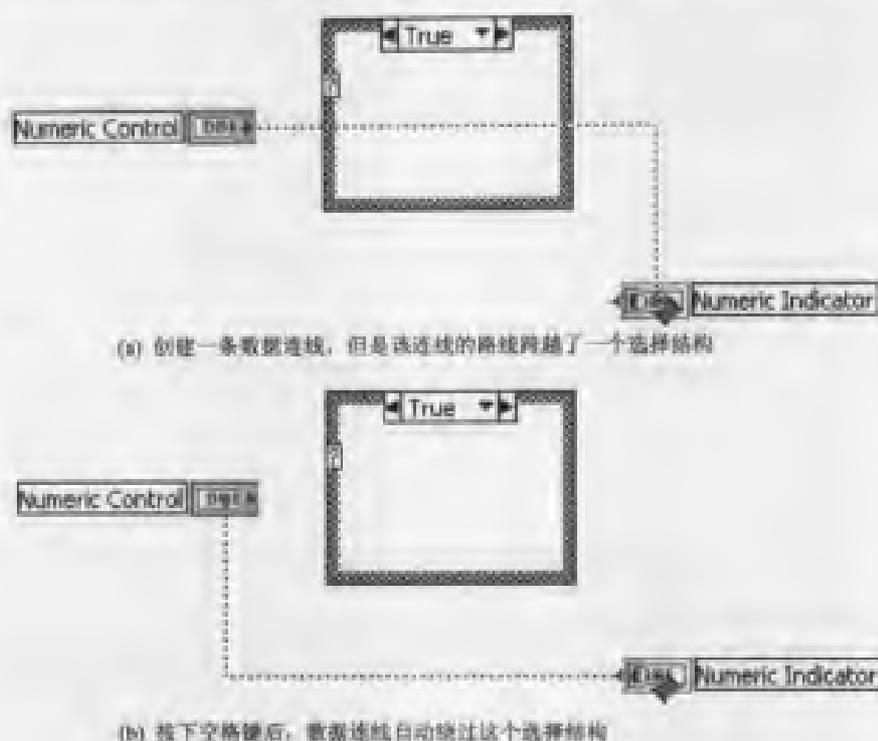


图 3.1.23 利用空格键改变数据连线的路线

LabVIEW 还有一种自动寻找路线的功能，当创建了一条数据连线，且它处于图 3.1.24 (a) 所示的状态时，按下键盘上的 A 键，LabVIEW 会自动为这条数据连线寻找最佳的路线，如图 3.1.24 (b) 所示。



图 3.1.24 利用 A 键寻找最佳路线

4. 数据连线的使用规则

在使用数据连线时，LabVIEW 规定了一些必须遵守的规则，若不遵守这些规则，则

LabVIEW 会报错, VI 无法运行。

(1) 数据连线的两端必须与端口相连接

图 3.1.25 所示的是在数据连线的一端或两端都没有连接端口时, 该数据连线的状态。

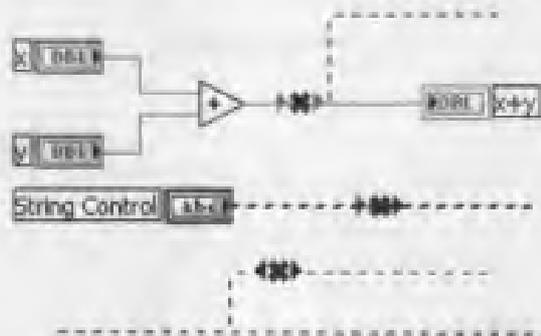


图 3.1.25 数据连线的一端或两端都没有连接端口

(2) 只能在相同数据类型的端口之间连接数据连线

图 3.1.26 所示为端口的数据类型相匹配和不相匹配时, 该数据连线的状态。



图 3.1.26 端口的数据类型相匹配和不相匹配时数据连线的状态

(3) 只能在源端口 (输出端口) 与目的端口 (输入端口) 之间连接数据连线

图 3.1.27 所示为输入输出端口相匹配和不相匹配时, 该数据连线的状态。



图 3.1.27 输入输出端口相匹配和不相匹配时数据连线的状态

(4) 数据连线从源端口 (输出端口) 出发后有多个分支

图 3.1.28 所示为数据连线从源端口出发后有多个分支时的情形, 在这种情况下, 每一个分支的末端可以连接一个目的端口 (输入端口)。

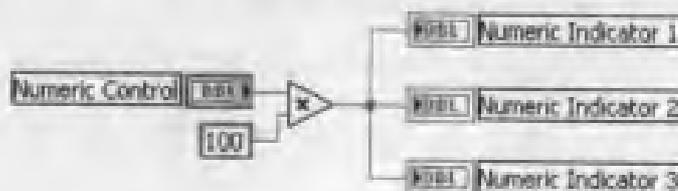


图 3.1.28 数据连线从源端口出发后有多分支时的情形

(5) 一条数据连线必须连接到一个源端口（输出端口）上

图 3.1.29 所示为数据连线有多个源端口和没有源端口时，数据连线的状态。



图 3.1.29 数据连线有多个源端口和没有源端口时的状态

(6) 不能利用数据连线将同一个节点的输入端口和输出端口相连接

图 3.1.30 所示为将同一个节点的输入端口和输出端口相连接时，数据连线的状态。



图 3.1.30 将同一个节点的输入端口和输出端口相连接时，数据连线的状态

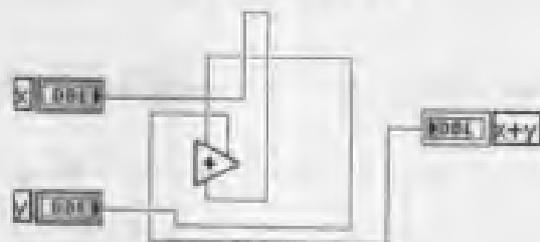
5. 使用数据连线时应当注意的内容

(1) 数据连线的分布应整齐美观

框图程序中的数据连线应当整齐美观，混乱的数据连线在逻辑上可能没有错误，但是在用户阅读框图程序时，可能会很困难，如图 3.1.31 (b) 所示。若出现数据连线分布混乱的情况，在数据连线的右键弹出选单中选择 Clean Up Wire，启用 LabVIEW 的自动整理数据连线功能，可以将混乱的数据连线自动整理成图 3.1.31 中 (a) 的状态。



(a) 整齐美观的数据连线



(b) 混乱的数据连线

图 3.1.31 分布简洁美观和分布混乱的数据连线

(2) 避免数据连线从对象的底下通过

应当避免数据连线从对象底下通过, 否则, 可能会使用户在阅读框图程序时, 误认为数据连线连接到了该对象上, 从而引起逻辑上的混乱, 如图 3.1.32 所示。

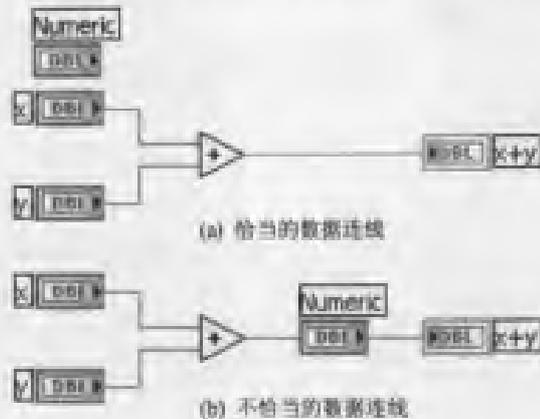


图 3.1.32 正确的数据连线分布和数据连线从对象的底下通过

(3) 避免隐藏数据连线

避免将数据连线隐藏到一个结构框架的下面, 否则会使用户在阅读框图程序时找不到该数据连线。用户在扩大一个结构的框架时, 常常会引起数据连线隐藏的情况。如图 3.1.33 所示, 数据连线仅仅是隐藏在所选结构的框架下面, 虽然程序上没有错误, 但会导致用户在读程序时找不到该段数据连线。

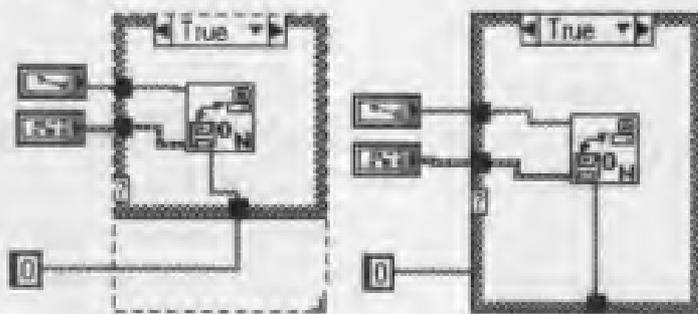


图 3.1.33 扩大一个选择结构框架时, 不小心隐藏了部分数据连线

(4) 避免错误的数据连线

当试图将数据连线连接到一个错误的端口上时, 连线工具会由正常状态  变为警告状态 , 警告用户这种连接会产生一条错误的的数据连线, 如图 3.1.34 所示。



图 3.1.34 连线工具在将要发生错误时变为警告状态

用户可以在 VI 处于编辑状态时创建一些错误的数据连线, 但 LabVIEW 在具有错误的的数据连线的状态下不能运行, 只有当清除了所有的错误之后, LabVIEW 才会运行。

(5) 为节点的端口连接有效的数据

LabVIEW 的一些节点在使用时,必须为它的某些指定必须输入数据的端口都连接上数据连线,即在这些端口上输入数据之后,该节点才能正常运行,若这些端口中有一个没有连接的数据连线,LabVIEW 就会报错,此时 LabVIEW 不能运行。

在 LabVIEW 窗口的主选单中选择 Help→Show Context Help, 会弹出 Context Help 窗口,窗口中会实时显示鼠标所指向的节点帮助信息,包括图标、输入输出端口的定义和节点功能说明,如图 3.1.35 所示。

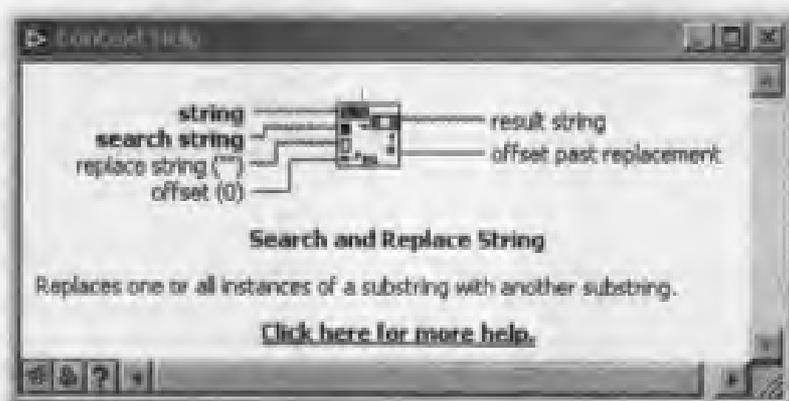


图 3.1.35 在 Context Help 窗口中获得节点帮助信息

节点的某些输入端口名称采用的字体是粗体,而有些输入端口的名称采用的字体是细体。凡是采用粗体名称的输入端口,例如图 3.1.35 中的端口 string 和端口 search string,必须为其连接数据连线,即为其输入数据;而采用细字体名称输入端口,可以根据需要不用连接数据连线,此时,该输入端口采用其默认值,在一般情况下,这些端口的默认值都会在端口名称上体现,例如图 3.1.35 中的端口 offset (0),圆括号中的数字 0 就是该端口的默认值。

3.1.4 图标/连接端口

图标/连接端口可以把 VI 变成一个对象 (SubVI, 即 VI 子程序),然后像子程序一样在其他 VI 中调用。图标作为 SubVI 的直观标记,在被其他 VI 调用时,图标代表 SubVI 中的所有框图程序,而连接端口表示该 SubVI 与调用它的 VI 之间进行数据交换的输入输出端口,就像传统编程语言子程序的参数端口,与 SubVI 中前面板上的控制和指示对应,图 3.1.36 所示的是正弦波发生器 VI 的图标和连接端口。



图标



连接端口

图 3.1.36 正弦波发生器 VI 的图标和连接端口

连接端口通常是隐藏在图标中的,图 3.1.36 的左侧是 SubVI 的图标。在 SubVI 图标的右键弹出选单中选择 Show→Terminals, 可以将图标切换到连接端口状态,即图 3.1.36 的右侧的图标,图标和连接端口都是由用户在编制 VI 时根据实际需要创建的,关于创建图标和连接端口的具体方法将在 3.5 节中介绍。

LabVIEW 的强大功能归因于它的层次化结构,用户可以把创建的 VI 程序当做 SubVI 来调用,以创建更加复杂的 VI,并且这种调用的递阶次数是无限制的。

3.2 LabVIEW 术语

由于 LabVIEW 是一种与常规编程语言不同的图形化编程语言, 其中用了大量的新术语, 为方便本书以后的介绍, 并且便于读者易于看懂本书后续部分内容, 特将 LabVIEW 中一些常用的术语列出, 如表 3.2.1 所示。

表 3.2.1 LabVIEW 常用术语中、英文对照表

英文术语	中文术语	英文术语	中文术语
VI, Virtual Instrument	虚拟仪器	Enable Indexing	有索引
SubVI	子 VI	Read Local	本地读
LLBs	VI 库	Write Local	本地写
Objects	对象	Read Global	全局读
Panel	前面板	Write Global	全局写
Block Diagram, Diagram Programme	框图程序	Legend	图例
Control	控制型控件	Cursor	光标
Indicator	指示型控件	Bounds	边界范围
Control 与 Indicator 的合称	前面板对象、控件	Data Acquisition (DAQ)	数据采集
Palette	模板	Atomic Data Type	原子类型
Functions Palette	功能模板	Encapsulation	封装性
Controls Palette	控件模板	Information hiding	信息隐藏性
Tools Palette	工具模板	Implementation independence	独立执行性
Terminal	端口	Abstraction	抽象性
Wires	数据连线	Flattened Data	平滑数据
Bad Wires	错误数据连线	Overhead	系统开销
Node	节点	Run time Stack	运行堆栈
Property Node	属性节点	mod	取模
Frame	框架	Modulus	模数
Tunnel	框架通道	Mechanical Action	机械动作
Index	索引	Enable	激活
Shift Register	移位寄存器	Disable	禁止
Label	标签	Latch	触发器
Chart	实时趋势图	Container	包容器
Graph	事后记录图	Servers	服务器
Functions	功能	Clients	客户机
Structures	结构	In-Process	进程内
Cluster	簇	Out-of-Process	进程外
Bundle	打包	Early Binding	早期绑定
Unbundle	解包	Late Binding	晚期绑定
RefNum	标识号、标识符	Automation	自动化
Local Variable	本地变量	Checkbox	复选框
Global Variable	全局变量	Copy of Data	数据复本
Constant	常量	State Machine	状态机
Disable Indexing	无索引	NI Test Execute State Machine	NI 测试执行状态机
Reference	指针	Polling Loop	轮询循环

3.3 创建和编辑 VI

在 3.1 节中已经提到，一个完整的 VI 是由前面板、框图程序和图标/连接端口组成的。例 3.3.1 就是一个简单完整的 VI。

例 3.3.1 计算两数之和。

如图 3.3.1 所示，左侧的窗口是 VI 的前面板窗口，右侧的窗口是框图程序窗口，图标位于前面板窗口及框图程序窗口的右上角，连接端口隐藏在图标之中。在前面板窗口中的图标的右键弹出选单中选择 Show Connector，可将图标状态切换到连接端口状态。有关连接端口创建的内容将在 3.5 节进行详细介绍。

例 3.3.1 的框图程序相当于 C 语言中的语句：

```
c=a+b;
```

下面介绍这个 VI 中各种对象的功能。

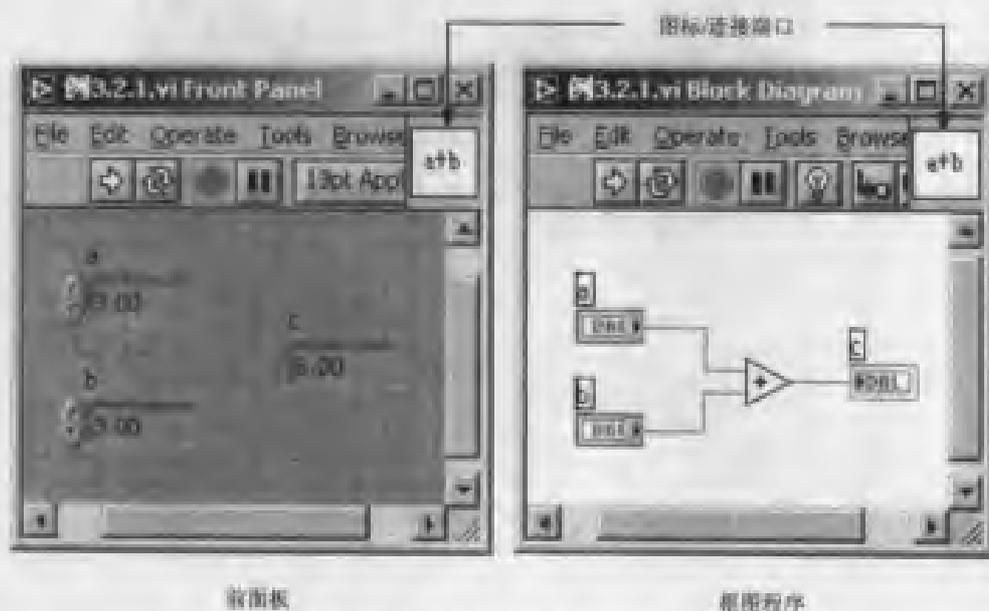


图 3.3.1 VI 的前面板及框图程序

数字量控制 (Digital Control)，数据源，用户可将数据输入到这两个控制中，相当于 C 语言中的变量 a 、 b 。向控制 a 和控制 b 输入数据的过程，就相当于执行 C 语言中的语句 “scanf (“%f”, &a);” 和语句 “scanf (“%f”, &b);”。

数字量指示 (Digital Indicator)，数据输出，显示运算结果，相当于 C 语言中的变量 c 。指示 c 显示计算结果的过程就相当于执行 C 语言中的语句 “printf (“e=%f”, c);”。

Add，算术运算节点，相当于 C 语言中的算术运算符 +。

Wires，数据连线，用于在节点间传递数据。

另外值得注意的是，LabVIEW 允许前面板对象没有名称，甚至允许重名。比如，两个数字量控制的名称都可以是 a 。这一点在传统的编程语言中是不允许的，这也是 LabVIEW 的编程特点之一。当然，一般情况下为便于程序的阅读理解，应当给每一个前面板对象赋予一个惟一的、一目了然的名称。另外，LabVIEW 支持汉字名称。

3.3.1 创建 VI

VI 的创建是 LabVIEW 编程应用中的基础, 下面以例 3.3.1 为例介绍如何创建 VI。VI 的创建通常按以下步骤进行。

1. 选择创建一个新的 VI

在 LabVIEW 主窗口中选择 New... 按钮中的 Blank VI, 或在一个已打开的 VI 的主选单中选择 File→New VI, 会出现如图 3.3.2 所示的 VI 窗口。前面是 VI 的前面板窗口, 后面是 VI 的框图程序窗口, 在两个窗口的右上角是默认的 VI 图标/连接端口。



图 3.3.2 New VI 窗口

2. 创建 VI 前面板



图 3.3.3 Controls 模板

(1) 创建控制量 a 和 b

在 VI 前面板窗口的空白处单击鼠标右键, 或者在窗口主选单 Windows 中选择 Show Controls Palette, 弹出 Controls 模板, 如图 3.3.3 所示。

在 Controls 模板→All Controls 子模板→Numeric 子模板中选择数字量控制 (Numeric Control), 如图 3.3.4 所示。

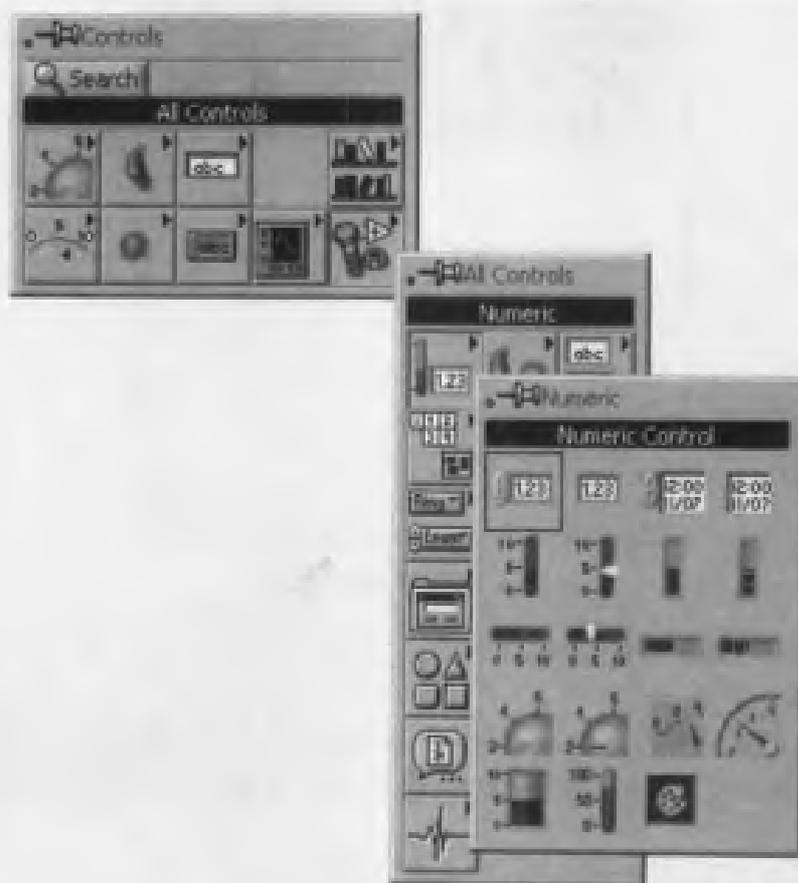


图 3.3.4 Numeric 子模板中选择数字量控制

将数字量控制放在前面板窗口中适当的位置上,用文本编辑工具  单击数字量控制的标签,把名称修改为 a,如图 3.3.5 所示。



图 3.3.5 创建数字量控制 a

此时,在框图程序中就会出现一个名称为 a 的端口图标与控制量 a 相对应,如图 3.3.6 所示。



图 3.3.6 数字量控制 a 的端口

以同样的方式创建控制量 b。

(2) 创建指示量 c

在 Controls 模板 → All Controls 子模板 → Numeric 子模板中选择数字量指示 (Numeric Indicator)，将其放在前面板窗口中适当的位置上，用文本编辑工具单击数字量指示的标签，把名称修改为 c。

此时，就完成了 VI 前面板的创建，如图 3.3.7 所示。

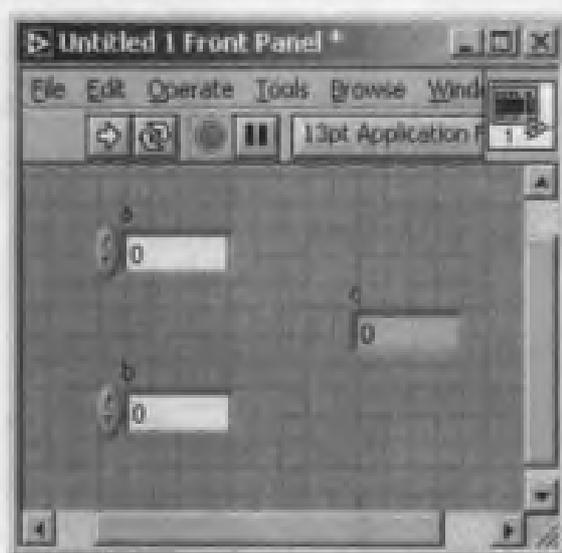


图 3.3.7 VI 前面板

3. 创建框图程序

在前面板窗口的主选单 Windows 中选择 Show Block Diagram，将前面板窗口切换到框图程序窗口，此时会看到在框图程序中有 3 个名称分别为 a, b, c 的端口，如图 3.3.8 所示。这 3 个端口与前面板上的 3 个对象一一对应。

(1) 创建加法算术运算节点

在框图程序窗口中的空白处单击鼠标右键，或在框图程序窗口的主选单 Windows 中选择 Show Functions Palette，弹出 Functions 模板，如图 3.3.9 所示。



图 3.3.8 前面板对象的端口



图 3.3.9 Functions 模板

然后在 Functions 模板中选择 All Functions 子模板→Numeric 子模板→Add 节点，如图 3.3.10 所示。

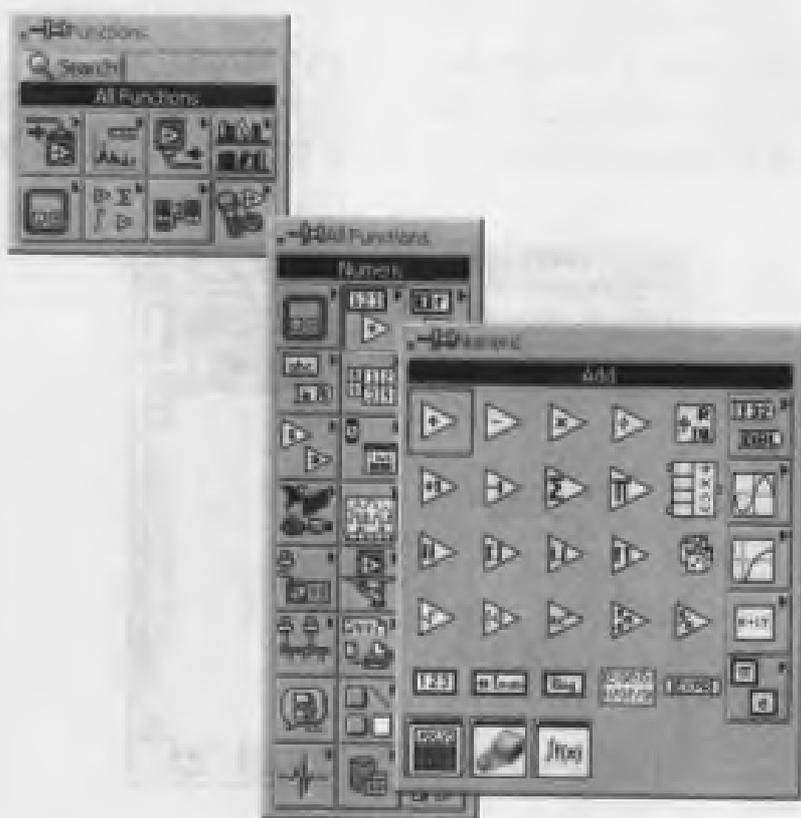


图 3.3.10 选择 Add 节点

最后用鼠标将所需的 Add 节点图标拖到框图程序窗口中适当的位置。至此，就完成了 Add 节点的创建，如图 3.3.11 所示。



图 3.3.11 创建 Add 节点

(2) 连接框图程序窗口内的节点与端口

完成了框图程序所需端口和节点的创建之后, 下面的工作就是用数据连线将这些端口和节点的图标连接起来, 形成一个完整的框图程序。

用连线工具将端口 a、b 分别连接到 Add 节点的两个输入端口 x 和 y 上, 将端口 c 连接到 Add 节点的输出端口 x+y 上。完成了数据连线的创建后, 将鼠标切换到对象操作工具状态, 适当调整各图标及数据连线的位置, 使之整齐美观。至此, 就完成了框图程序的创建, 如图 3.3.12 所示。

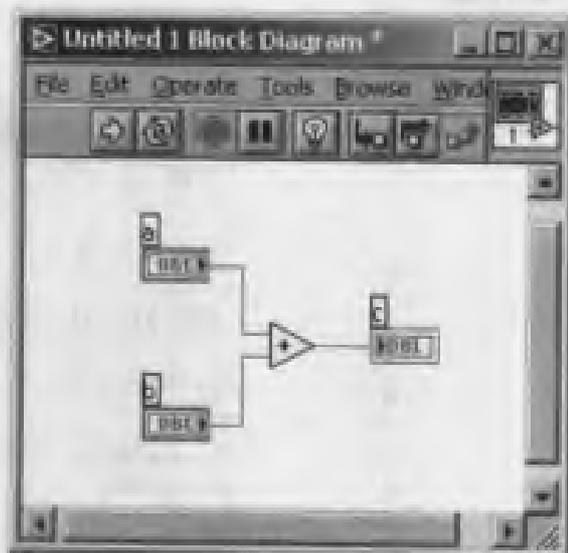


图 3.3.12 完整的 VI 框图程序

4. 创建 VI 图标

双击前面板窗口或框图程序窗口右上角的 VI 图标, 或在 VI 图标处单击鼠标右键并在弹出的选单中选择 Edit Icon..., 会弹出一个图标编辑器 (Icon Editor), 如图 3.3.13 所示。



图 3.3.13 图标编辑器

在图标编辑器中可创建用户自己的图标。图标编辑器的用法与 Windows 操作系统中的画图工具软件类似，这里不再详细介绍其用法。图 3.3.14 所示的是一个包含用户自定义图标的前面板窗口。

5. 保存 VI

在前面板窗口或框图程序窗口的主选单中选择 File→Save，然后在弹出的保存文件对话框中选择适当的路径和文件名保存该 VI。如果一个 VI 在修改后没有存盘，那么在 VI 前面板窗口和框图程序窗口的标题栏中就会出现一个“*”，提示用户注意存盘，参见图 3.3.14。

至此，就完成了—个 VI 的创建。在控制量 a 和 b 中分别输入适当的数字值，然后单击前面板窗口工具条中的 Run 按钮 ，就可以在指示量 c 中得到计算结果，参见图 3.3.1。



图 3.3.14 包含用户自定义图标的前面板窗口

3.3.2 编辑 VI

创建 VI 后，还需要对 VI 进行编辑，使 VI 的图形化交互式用户界面更加美观、友好而易于操作，使 VI 框图程序的布局 and 结构更加合理，易于理解、修改。

1. 选择对象

在工具模板中将鼠标切换为对象操作工具。

当选择单个对象时，直接用鼠标左键单击需要选中的对象；如果需要选择多个对象，则要在窗口空白处拖动鼠标，使拖出的虚线框包含要选择的多个目标对象，或者按住 Shift 键，用鼠标左键单击多个目标对象，如图 3.3.15 所示。

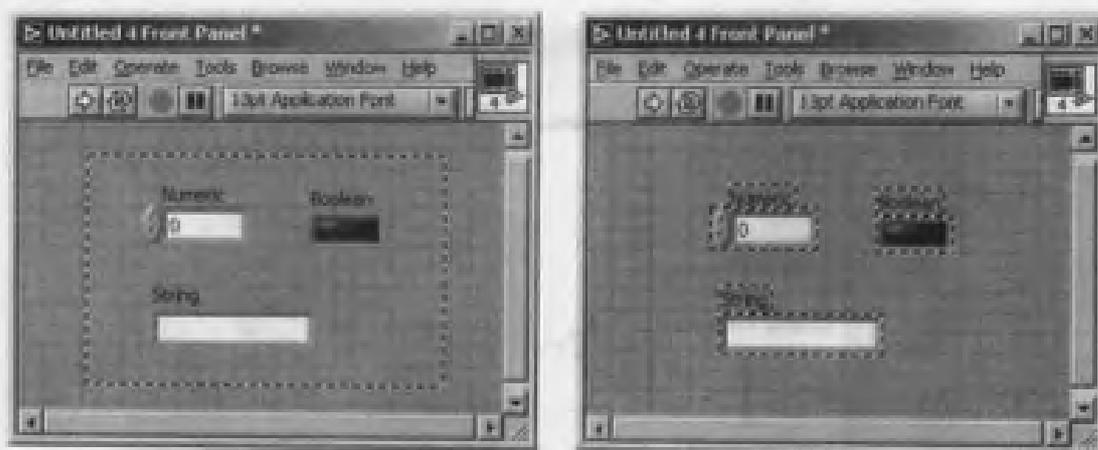


图 3.3.15 选择对象

2. 删除对象

选中目标对象，按键盘上的 Delete 键，或在窗口主选单中选择 Edit→Clear，即可删除对象。其结果如图 3.3.16 所示。



图 3.3.16 删除对象

3. 变更对象位置

用对象操作工具拖动目标对象到指定位置，如图 3.3.17 所示。

注意：在拖动对象时，窗口中会出现一个黄色的文本框，实时显示对象移动的相对坐标。

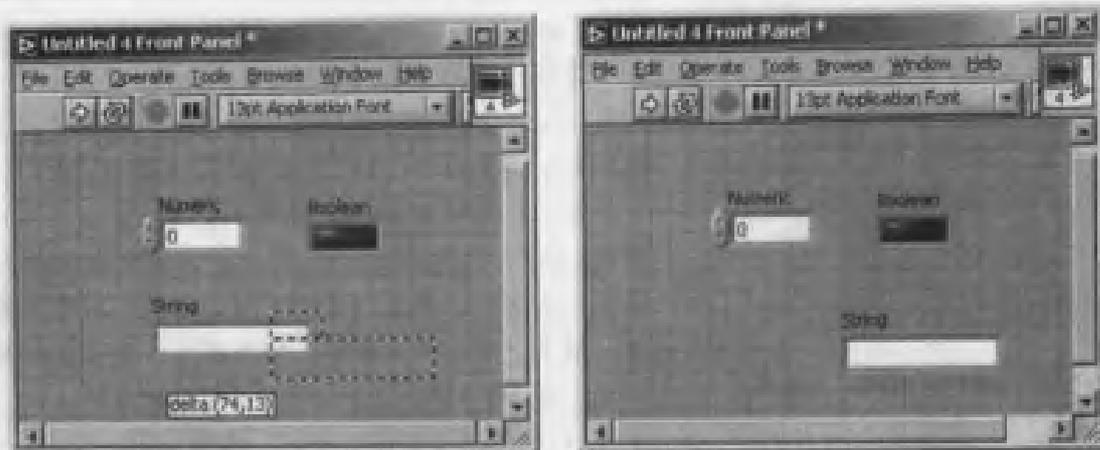


图 3.3.17 变更对象位置

4. 改变对象的大小

几乎每一个 LabVIEW 对象都有 8 个尺寸控制点，当对象操作工具位于对象上时，这 8 个尺寸控制点会显示出来。用对象操作工具拖动某一个尺寸控制点，可以改变对象在该位置的尺寸，如图 3.3.18 所示。注意，有些对象的大小是不能改变的，例如框图程序中的控制端口或指示端口、Functions 模板中大部分节点的图标和 SubVI 的图标等。

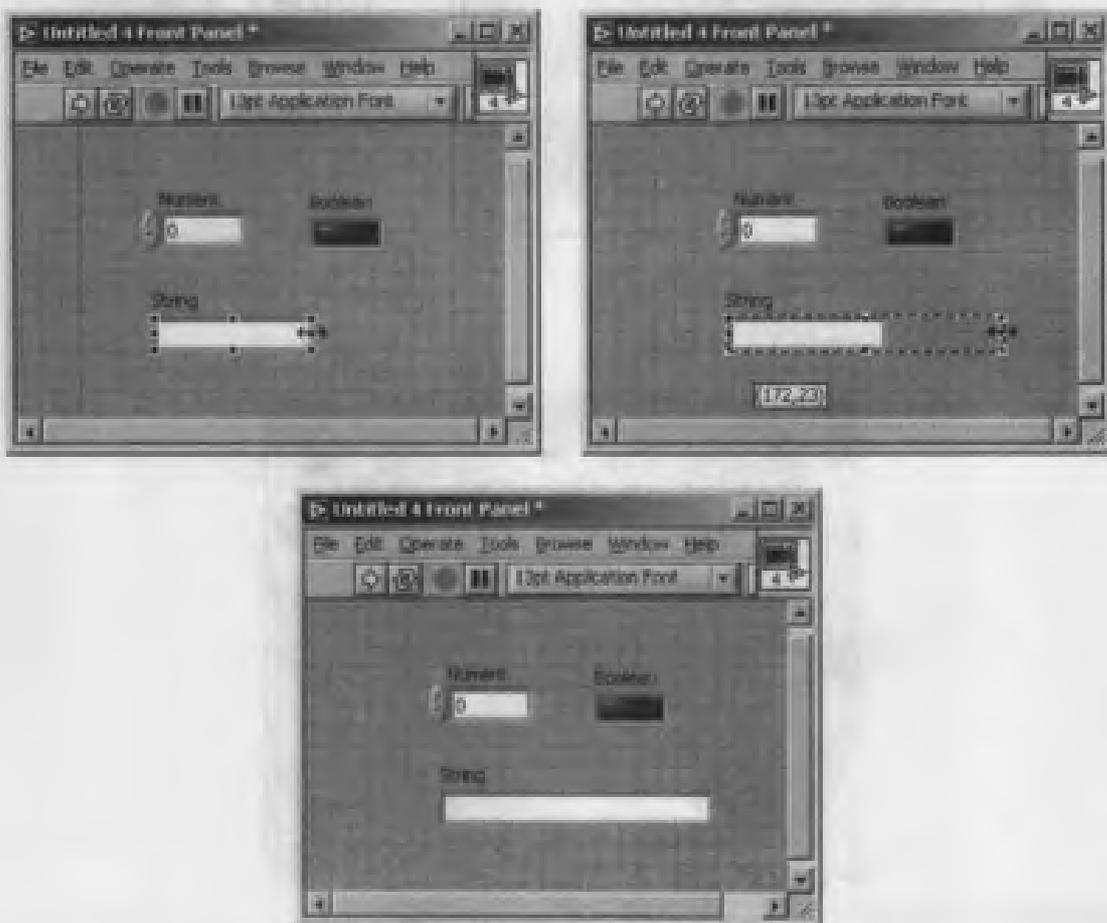


图 3.3.18 改变对象的大小

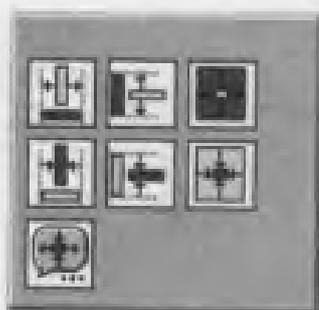


图 3.3.19 Resize Objects 下拉选单

注意，在拖动对象的边框时，窗口中也会出现一个黄色的文本框，实时显示对象的相对坐标。

另外，LabVIEW 的前面板窗口的工具条上还提供了一个“Resize Objects”按钮，用鼠标单击该按钮，弹出一个图形化下拉选单，如图 3.3.19 所示。

利用该选单中的工具可以统一设定多个对象的尺寸，包括将所选中的多个对象的长度设为这些对象的最大宽度、最小宽度、最大高度、最小高度、最大宽度和高度、最小宽度和高度以及指定的宽度和高度等。

例如，将前面板上所有的对象的宽度设为这些对象的最大宽度，步骤如下。

第一步，选中目标对象，如图 3.3.20 所示。

第二步，在 Resize Objects 下拉选单中选择 Maximum Width，如图 3.3.21 所示。

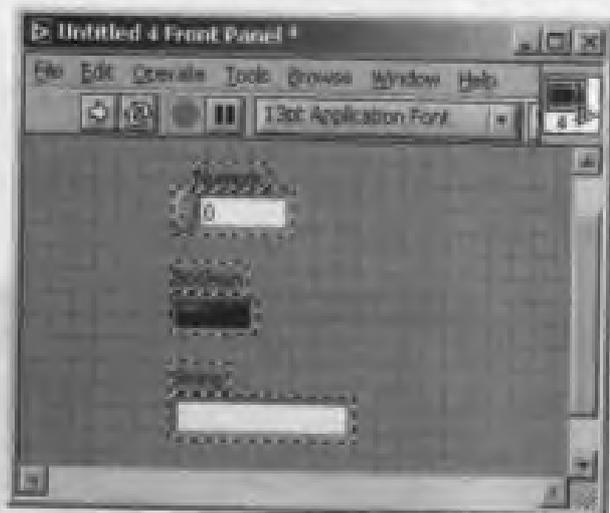


图 3.3.20 选中目标对象



图 3.3.21 Resize Objects 下拉选单

统一宽度后的对象如图 3.3.22 所示。



图 3.3.22 统一宽度后的对象

若在 **Resize Objects** 下拉菜单中选择 **Set Width and Height...**，如图 3.3.23 所示，则会弹出一个 **Set Width and Height** 对话框，用户可以在该对话框中指定控件的宽度或高度，如图 3.3.24 所示。

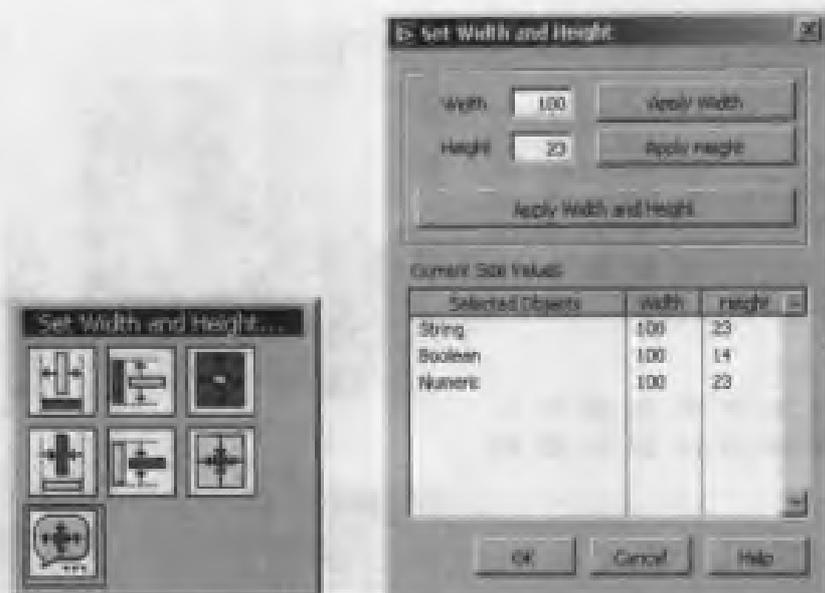


图 3.3.23 Resize Objects 下拉菜单

图 3.3.24 Set Width and Height 对话框

5. 改变对象颜色

在工具模板中将鼠标切换为颜色工具，如图 3.3.25 所示。

在图 3.3.25 所示的颜色工具的图标中，有两个上下重叠的颜色框，上面的颜色框代表对象的前景色或边框色，下面的颜色框代表对象的背景色。单击其中一个颜色框，就可以在弹出的颜色对话框中为其选择需要的颜色，如图 3.3.26 所示。



图 3.3.25 工具模板

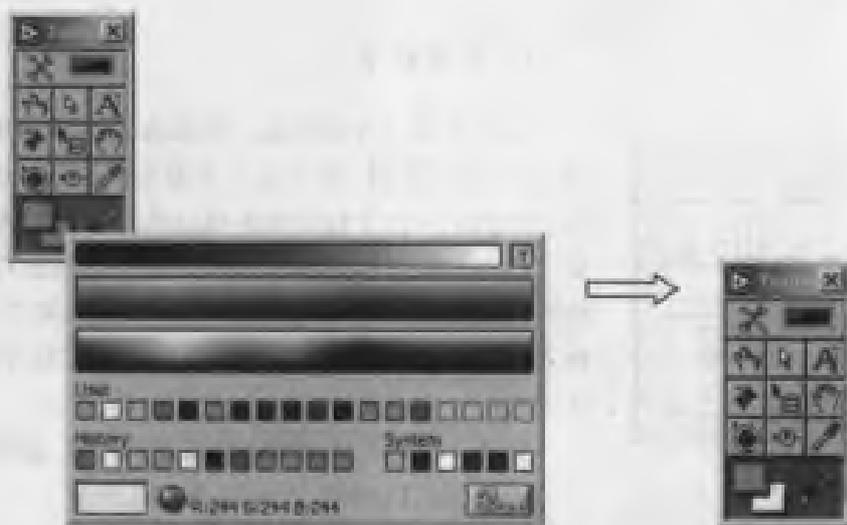


图 3.3.26 颜色对话框

若颜色对话框中没有所需的颜色, 可以单击颜色对话框中的“More Colors”按钮, 此时系统会弹出一个 Windows 标准颜色对话框, 如图 3.3.27 所示, 在这个对话框中可以选择预先设定的各种颜色, 或直接设定 RGB 三原色的数值, 更加精确地选择颜色。



图 3.3.27 Windows 标准颜色对话框

完成颜色的选择后, 用颜色工具单击需要改变颜色的对象, 即可将对象改为指定的颜色, 如图 3.3.28 所示。



图 3.3.28 改变对象颜色

6. 对齐对象

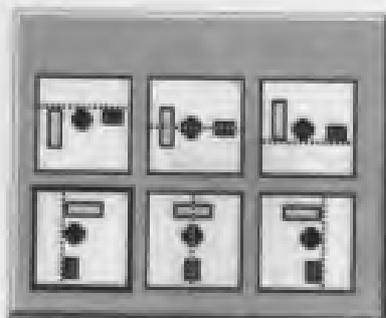


图 3.3.29 Align Objects 下拉菜单

选中需要对齐的对象, 然后在工具条中单击 Align Objects 按钮 , 会出现一个图形化的下拉选单, 如图 3.3.29 所示, 在下拉选单中可以选择各种对齐方式。选单中的各种图例很直观地表示了各种不同的对齐方式, 共有左边缘对齐、右边缘对齐、上边缘对齐、下边缘对齐、水平中轴线对齐以及垂直中轴线对齐等 6 种方式可选。

例如, 要将几个对象按左边缘对齐, 步骤如下。

- ① 选中目标对象, 如图 3.3.30 所示。

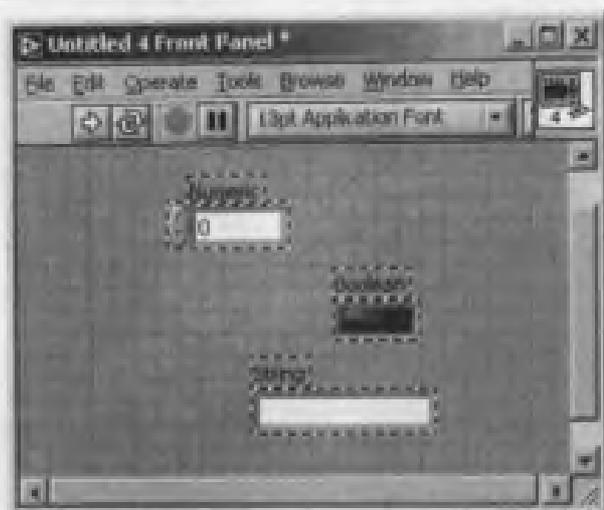


图 3.3.30 选中目标对象

② 在 Align Objects 下拉菜单中选择 Left Edges，如图 3.3.31 所示。左边缘对齐后的对象如图 3.3.32 所示。



图 3.3.31 Align Objects 下拉选单

图 3.3.32 左边缘对齐后的对象

7. 分布对象

选中对象，在工具条中单击 Distribute Objects 按钮 ，会出现一个图形化的下拉选单，如图 3.3.33 所示，在选单中可以选择各种分布方式。选单中的各种图例很直观地表示了各种不同的分布方式。

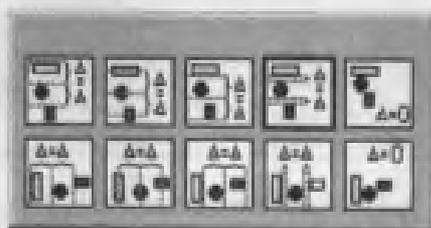


图 3.3.33 Distribute Objects 下拉选单

例如, 要将对象按照等间隔垂直分布步骤如下。

① 选中目标对象如图 3.3.34 所示。

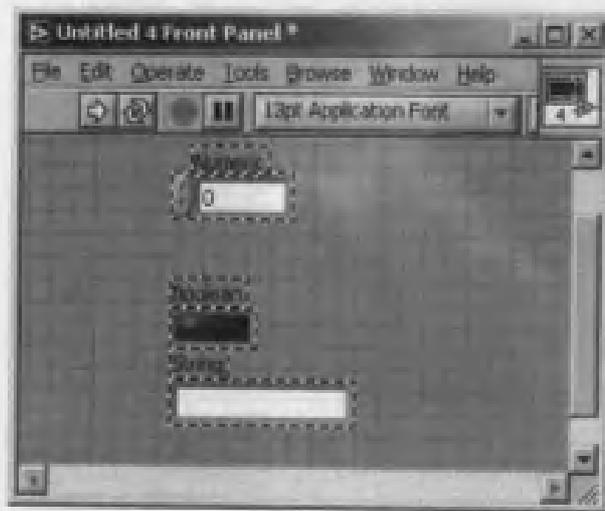


图 3.3.34 选中目标对象

② 在 Distribute Objects 下拉菜单中选择 Vertical Gap, 如图 3.3.35 所示。

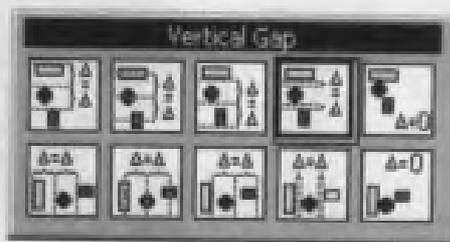


图 3.3.35 在 Distribute Objects 下拉菜单中选择 Vertical Gap

等间隔垂直分布的对象如图 3.3.36 所示。



图 3.3.36 等间隔垂直分布的对象

8. 改变对象在窗口中的前后次序

选中对象，在工具条中单击 Reorder 按钮 ，可以在下拉选单中改变对象在窗口中的前后次序。下拉选单如图 3.3.37 所示。

Move Forward 是指将对象向上移动一层；Move Backward 是指将对象向下移动一层；Move to Front 是指将对象移动至窗口的最顶层；Move to Back 是指将对象移动至窗口的最底层。

例如，要将一个对象从窗口的最顶层移动至窗口的最底层，具体操作步骤如下。

第一步，选中目标对象，如图 3.3.38 所示。

第二步，在 Reorder 下拉选单中选择 Move To Back，如图 3.3.39 所示。

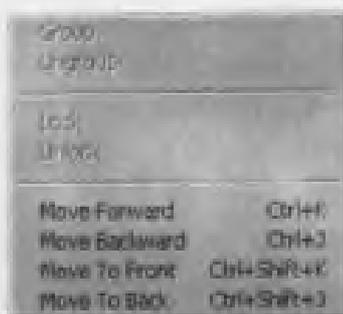


图 3.3.37 Reorder 下拉选单



图 3.3.38 选中目标对象

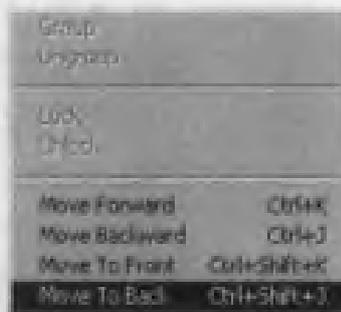


图 3.3.39 在 Reorder 下拉选单中选择 Move To Back

改变次序后的对象如图 3.3.40 所示。



图 3.3.40 改变次序后的对象

9. 组合与锁定对象

在 Reorder 下拉菜单中还有几个选项, 它们分别是 Group 与 Ungroup, Lock 与 Unlock。

Group 的功能是将几个选定的对象组合成一个对象组, 对象组中的所有对象形成一个整体, 它们的相对位置和尺寸都相对固定。当移动对象组或改变对象组的尺寸时, 对象组中所有的对象会同时移动相同的距离或改变相同的尺寸。注意, Group 的功能仅仅是将数个对象按照其位置和尺寸简单的组合在一起形成一个整体, 并没有在逻辑上将其组合, 它们之间在逻辑上的关系并没有因为组合在一起而得到改变。Ungroup 的功能是解除对象组中对象的组合, 将其还原为独立的对象。

Lock 的功能是将几个选定的对象组合成一个对象组, 并且锁定该对象组的位置和大小, 用户不能改变锁定的对象的位置和尺寸。当然, 用户也不能删除处于锁定状态的对象。Unlock 的功能是解除对象的锁定状态。

当用户已经编辑好一个 VI 的前面板时, 建议用户利用 Group 或 Lock 功能将前面板中的对象组合并锁定, 防止由于误操作而改变了前面板对象的布局。

例如, 将几个前面板对象组合在一起, 其步骤如下。

第一步, 选中目标对象, 如图 3.3.41 所示。



图 3.3.41 选中目标对象

第二步, 在 Reorder 下拉菜单中选择 Group, 如图 3.3.42 所示。



图 3.3.42 在 Reorder 下拉菜单中选择 Group

组合后的对象如图 3.3.43 所示。

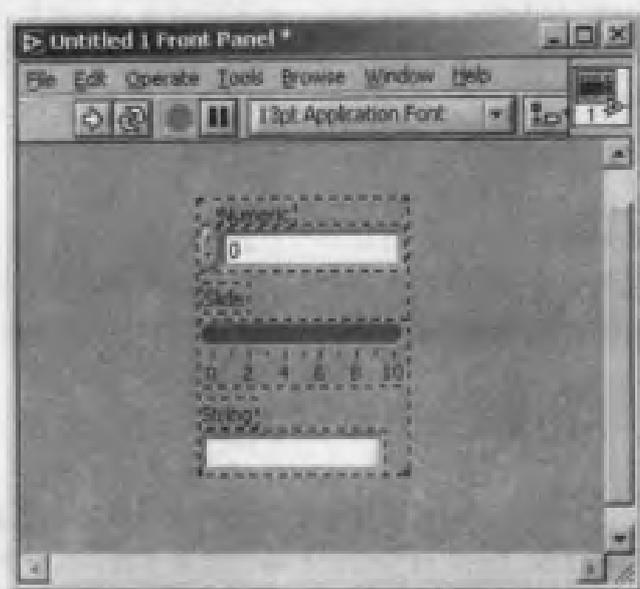


图 3.3.43 组合后的对象

10. 设置对象的字体

选中对象，在工具条中的 Text Settings 下拉列表框 **13pt Application Font** 中选择 Font Dialog...，弹出字体设置对话框后可设置对象的字体、大小、颜色、风格及对齐方式，如图 3.3.44 所示。

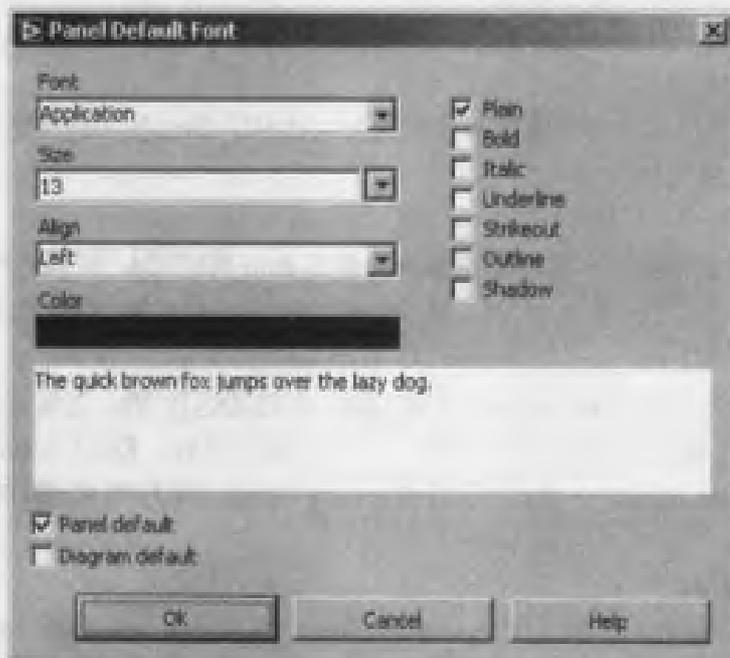


图 3.3.44 字体设置对话框

Text Settings 下拉列表框中的其他选项只是将字体设置对话框中的内容分别列出，若只

改变字体的某一个属性,可以方便地在这些选项中更改,而无须在字体对话框中更改。

另外,还可以在 Text Settings 下拉列表框中将字体设置为系统默认的字體,包括:应用程序字体 (Application Font)、系统字体 (System Font)、对话框字体 (Dialog Font) 及当前字体 (Current Font) 等。

11. 在窗口中添加标签

将鼠标切换至文本编辑工具状态,在窗口空白处中的适当位置单击鼠标,就可在窗口中创建一个标签,然后根据需要键入文字,改变其字体或颜色。该工具也可用于改变对象的 Label、Caption、布尔量控件的 Boolean Text 和数字量控件的刻度值,等等。

3.4 运行和调试 VI

运行和调试程序是用任何一种编程语言编程最重要的一步。通过这一步,编程者可以查找出程序中存在的各种错误,根据这些错误和运行结果修改、优化程序,最终得到一个正确、可靠的程序,提交给用户使用。传统编程语言在运行程序之前,大都必须先编译源代码,然后再连接形成可执行代码。LabVIEW 系统编译器是内置的后台编译,VI 运行时可自动完成编译并执行,系统的编译连接过程是不可见的。LabVIEW 还专门设计了一个需要单独购买的应用程序生成器 (Application Builder Tools),为生成可以脱离 LabVIEW 环境运行的 Windows 可执行程序提供编译环境,这也是 LabVIEW 有别于其他图形化编程环境的特点之一。

下面介绍如何在 LabVIEW 中运行和调试 VI。

3.4.1 运行 VI

在 LabVIEW 中,用户可以通过两种方式来运行 VI,即运行和连续运行。下面介绍这两种运行方式的使用方法。

(1) 运行 VI

在前面板窗口或框图程序窗口的工具条中单击 Run 按钮 ,可以运行 VI。使用这种方式运行 VI,VI 只运行一次,当 VI 正在运行时,Run 按钮会变为  状态。

(2) 连续运行 VI

在工具条中单击 Run Continuously 按钮 ,可连续运行 VI。连续运行的意思是指一次 VI 运行结束后,继续重新运行 VI。当 VI 正在连续运行时,Run Continuously 按钮会变为  (Running Continuously) 状态。单击 Running Continuously 按钮  可停止 VI 的连续运行。

(3) 停止运行 VI

当 VI 处于运行状态时,在工具条中单击 Abort Execution 按钮 ,可强行终止 VI 的运行。这项功能在程序的调试过程中非常有用,当不小心使程序处于死循环状态时,用该按钮可安全地终止程序的运行。当 VI 处于编辑状态时,Abort Execution 按钮处于  状态,此时的按钮是不可操作的。

(4) 暂停 VI 运行

在工具条中单击 Pause 按钮 ，可暂停 VI 的运行，再次单击该按钮，可恢复 VI 的运行。

3.4.2 调试 VI

LabVIEW 编译环境提供了多种调试 VI 程序的手段，除了具有传统编程语言支持的单步运行、断点和探针等调试手段之外，还添加了一种特有的调试手段——实时显示数据流动画。这种调试手段的出现，可以使用户更加清楚地观察程序运行的每一个细节，为查找错误、修改和优化程序提供了有效的手段和依据。下面介绍 LabVIEW 中各种调试手段的具体用法。

1. 单步执行 VI

单步执行 VI 与传统编程语言中的单步执行程序类似。所不同的是，传统编程语言中的单步运行是指按照程序中语句的逻辑顺序逐条语句地执行程序，而单步执行 VI 则是在框图程序中，按照节点之间的逻辑关系，沿数据连线逐个节点地执行 VI。在 LabVIEW 中有两种单步执行 VI 的方式。

(1) 单步（入）执行

按节点顺序单步执行，遇到循环或 SubVI 时，跳入循环或 SubVI 内部继续逐步运行程序。单击工具条上的单步（入）按钮 ，就可进入单步（入）执行 VI 状态。单击一次该按钮，程序执行一步。

(2) 单步跳执行

按节点顺序单步执行，但是遇到循环或 SubVI 时，不跳入其内部逐条执行其中的内容，而是将其作为一个整体节点执行。单击工具条上的单步跳（Step over）按钮 ，就可进入单步（跳）执行 VI 状态。单击一次该按钮，程序执行一步。

(3) 单步（出）

在框图程序的工具条中选择单步（出）按钮 ，可跳出单步执行 VI 的状态，进入暂停运行状态。

当 VI 进入单步执行状态时，将鼠标移动到单步（入）按钮或单步（跳）按钮上，会出现一个黄色的小标注框，显示下一步将要执行的节点。这一点对于有效地调试 VI 很有帮助，也是 LabVIEW 的特色之一。

2. 设置断点

在工具模板中将鼠标切换至断点工具状态，如图 3.4.1 所示。

单击框图程序中需要设置断点的地方，就可完成一个断点的设置。当断点位于某一个节点时，该节点图标的边框就会变红；当断点位于某一条数据连线时，数据连线的中央就会出现一个红点，如图 3.4.2 所示。



图 3.4.1 工具模板

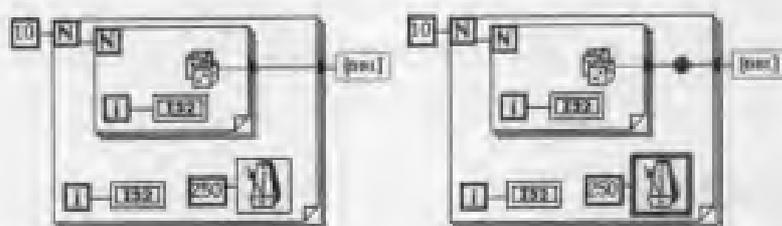


图 3.4.2 设置断点

当程序运行到该断点时, VI 会自动暂停, 此时断点处的节点会处于闪烁状态, 提示用户程序暂停的位置。用鼠标单击 Pause 按钮, 可以恢复程序的运行。用断点工具再次单击断点处, 或在该处的右键弹出选单中选择“Clear Breakpoint”, 就会取消该断点。

3. 设置探针

在工具模板中将鼠标切换至探针工具状态, 如图 3.4.1 所示。

用鼠标单击需要查看的数据连线, 或在数据连线的右键弹出选单中选择 Probe, 会弹出一个探针对话框。当 VI 运行时, 若有数据流过该数据连线, 对话框就会自动显示这些流过的数据, 如图 3.4.3 所示。同时, 在探针处会出现一个黄色的内含探针数字编号的小方框。

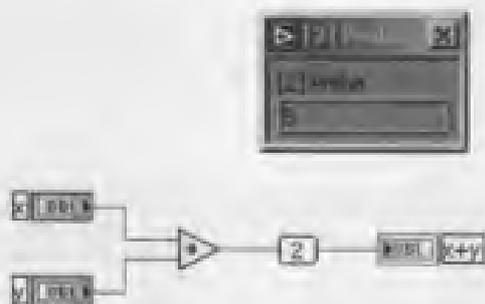


图 3.4.3 设置探针

利用探针工具弹出的探针对话框是 LabVIEW 默认的探针对话框, 有时候并不能满足用户的需求, 若在数据连线的右键弹出选单中选择 Custom Probe, 用户可以自己定制所需的探针对话框。

4. 显示数据流动画

当运行 VI 时, 在框图程序窗口的工具条中单击 Highlight Execution 按钮 , LabVIEW 会在框图程序上实时地显示程序执行的进程, 以及实时显示每一条数据连线和每一个端口中流过的数据。再次单击该按钮, VI 会恢复到正常执行状态。注意, 在使用该功能时, VI 的执行速度会明显降低, 以保证用户能够清楚地看到 VI 程序执行过程的每一个细节。

综合运用以上各种调试 VI 的方法, 可以方便、有效地编制出逻辑严谨、结构合理的 VI。这种图形化的编程方式及调试方式可以大大提高编程效率。

有经验的用户可以体会到, 使用文本编程语言时, 在程序的调试过程中, 查找并修改语法错误是一个令人十分头痛的问题, 并且这一工作占用了用户相当一部分的工作时间。

而采用 LabVIEW 的图形化编程方式就不存在这样的语法问题, 用户可将大部分的精力放在程序的结构、逻辑以及功能实现上, 大大提高了编程的效率。根据统计, 采用 LabVIEW 进行编程, 有经验的用户可以提高 80% 甚至 100% 的工作效率, 这一点是 LabVIEW 功能强大的又一体现。

5. 查找 VI 中的错误

用户在编程过程中难免会出现各种错误, 例如, 将两个不同数据类型的端口连接在了一起, 或没有给某一个节点的端口连接数据, 如图 3.4.4 所示。



图 3.4.4 存在错误的 VI 框图程序

在一个 VI 存在错误的情况下, 该 VI 是不能运行的, 此时 VI 窗口工具条上的 Run 按钮  会变为 List Errors 按钮 。用鼠标单击 List Errors 按钮, 会弹出 Error List 对话框, 如图 3.4.5 所示。

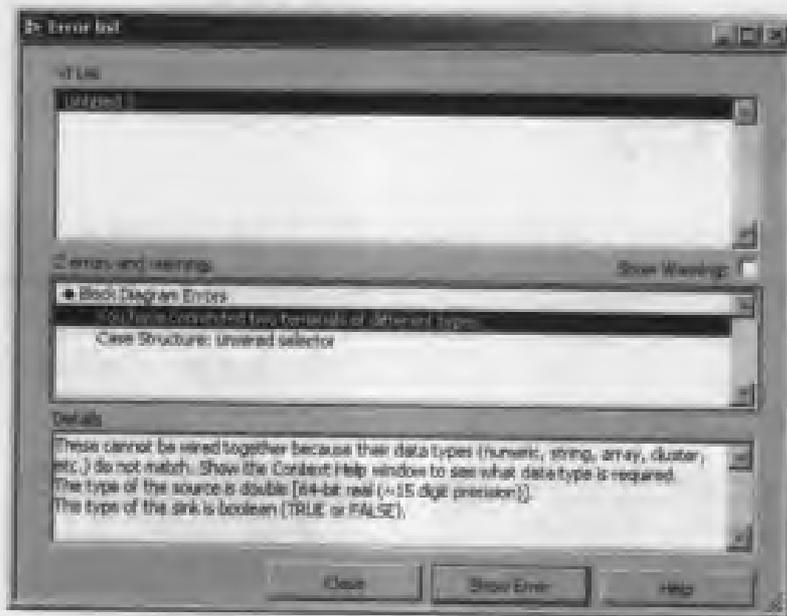


图 3.4.5 Error List 对话框

Error List 对话框中又分为 3 个栏目: VI List, Errors and Warnings 和 Details。表 3.4.1 列出了这 3 个栏目和对话框中 3 个按钮的功能。

表 3.4.1 窗口工具条功能一览表

栏目/按钮名称	功能说明
VI List	列出出现错误的 VI 的文件名
Errors and Warnings	列出 VI 中出现的错误和警告信息
Details	显示 Errors and Warnings 栏中指定的错误或警告的信息
Close 按钮	关闭 Error List 对话框
Show Error 按钮	定位到框图程序中 Errors and Warnings 栏中指定的错误
Help 按钮	在 LabVIEW Help 窗口中显示 Errors and Warnings 栏中指定的错误或警告的详细信息

当 VI 框图程序较为复杂, 且存在的错误较多时, 利用 Error List 对话框可以很方便地查找到每一个错误, 并得到详细的错误信息和修改建议, 建议用户在编程过程中有效地利用 Error List 对话框, 这样可以大大提高编程效率。

3.5 创建和调用 SubVI

SubVI 相当于常规编程语言中的子程序, 在 LabVIEW 中, 用户可以把任何一个 VI 当做 SubVI 来调用。因此在使用 LabVIEW 编程时, 也应与其他编程语言一样, 尽量采用模块化编程的思想, 有效地利用 SubVI, 简化 VI 框图程序的结构, 使其更加简单, 易于理解, 以提高 VI 的运行效率。

在本章第 2 节中曾经提到, VI 由 3 部分组成, 其中一部分是图标/连接端口, SubVI 利用连接端口与调用它的 VI 交换数据。实际上, 创建完成一个 VI 后, 再按照一定的规则定义好 VI 的连接端口, 该 VI 就可以作为一个 SubVI 来使用了。

下面以例 3.3.1 为例, 介绍如何创建和调用 SubVI。

3.5.1 创建 SubVI

在完成一个 VI 的创建后, 将其作为 SubVI 调用的主要工作就是定义 VI 的连接端口。

在 VI 前面板或框图程序面板的右上角图标的右键弹出菜单中选择 Show Connector, 原来图标的位置就会显现一个连接端口, 如图 3.5.1 所示。连接端口由输入端口和输出端口组成, 在某种意义上, 输入端口就相当于 C 语言子程序中的虚参, 而输出端口就相当于 C 语言子程序中 return () 语句括号中的参数。



图 3.5.1 VI 的连接端口

第一次打开连接端口时, LabVIEW 会自动根据前面板中的控制和指示建立相应个数的端口。当然, 这些端口并没有与控制或指示建立起关联关系, 但需要用户自己定义。但通常情况下, 用户并不需要把所有的控制或指示都与一个端口建立关联, 与外部交换数据, 因而需要改变连接端口中端口的个数。

LabVIEW 提供了 2 种方法来改变端口的个数:

第 1 种方法是在连接端口右键弹出选单中选择 Remove Terminal 或 Add Terminal, 逐个删除或添加连接端口。这种方法较为灵活, 但也比较烦琐。

第 2 种方法是在连接端口右键弹出选单中选择 Patterns, 会出现一个图形化下拉选单, 选单中会列出 36 种不同的连接端口, 一般情况下可以满足用户的需要, 如图 3.5.2 所示。这种方法较为简单, 但是不够灵活, 有时不能满足需要。

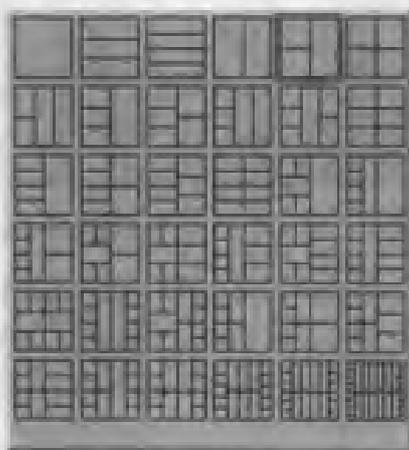


图 3.5.2 Patterns 下拉选单

通常的做法是, 先用第 2 种方法选择一个与实际需要比较接近的连接端口, 然后再用第 1 种方法对选好的连接端口进行修正。

完成了连接端口的创建之后, 下面的工作就是定义前面板中的控制和指示与连接端口中各输入输出端口的关联关系。具体步骤如下:

- ① 在工具模板中将鼠标变为连线工具状态, 如图 3.5.3 所示。
- ② 用鼠标在控制 a 上单击一下, 选中控制 a, 此时控制 a 的图标周围会出现一个虚框, 如图 3.5.4 所示。



图 3.5.3 工具模板



图 3.5.4 选中控制 a

③ 将鼠标移动至连接端口的一个端口上，单击这个端口。

此时这个端口就建立了与控制 a 的关联关系，端口的名称为 a，颜色为棕色，如图 3.5.5 所示。当其他 VI 调用这个 SubVI 时，从这个连接端口输入的数据就会输入到控制 a 中，然后程序从控制 a 在框图程序中所对应的端口中将数据取出，进行相应的处理。

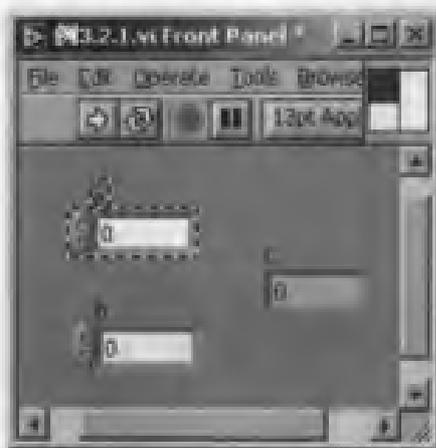


图 3.5.5 建立端口与控制 a 的关联关系

注意，端口的颜色是由与之关联的前面板对象的数据类型来确定的，不同的数据类型对应不同的颜色，例如，与布尔量相关联的端口的颜色是绿色。

建立前面板中其他控制或指示与端口连接关系的方法与此相同。定制好的 VI 连接端口如图 3.5.6 所示。



图 3.5.6 定制好的 VI 连接端口

注意，按照 LabVIEW 的定义，与控制相关联的连接端口都作为输入端口。在 SubVI 被其他 VI 调用时，只能向输入端口中输入数据，而不能从输入端口中向外输出数据。当某一个输入端口没有连接数据连线时，LabVIEW 就会将与该端口相关联的那个控制中的数据默认值作为该端口的数据输入值。相反，与指示相关联的连接端口都作为输出端口，只能向外输出数据，而不能向内输入数据。

在编辑调试 VI 过程中，用户有时会根据实际需要断开某些端口与前面板对象的关联，具体做法是：在需要断开的端口的右键弹出选单中选择 Disconnect This Terminal。若在选

单中选择 Disconnect All Terminals，则会断开所有端口的关联。

3.5.2 调用 SubVI

在完成了连接端口的定义之后，这个 VI 就可以当做 SubVI 来调用了。下面介绍如何在一个主 VI 中将例 3.3.1 作为 SubVI 来调用，具体步骤如下：

① 选择 SubVI。

选择 Functions 模板中的 All Functions→Select a SubVI...子模板，会弹出一个名为 Choose the VI to open 的对话框，如图 3.5.7 所示。

在对话框中找到需要调用的 SubVI，选中后单击“打开 (O)”按钮。

② 将 SubVI 的图标放至主 VI 框图程序窗口中。



图 3.5.7 Choose the VI to open 对话框

用户选择了一个 SubVI 后，此时，在鼠标上会出现这个 SubVI 的图标，将其移动到框图程序窗口中的适当位置上，单击鼠标左键，将图标加入到主 VI 的框图程序中，如图 3.5.8 所示。

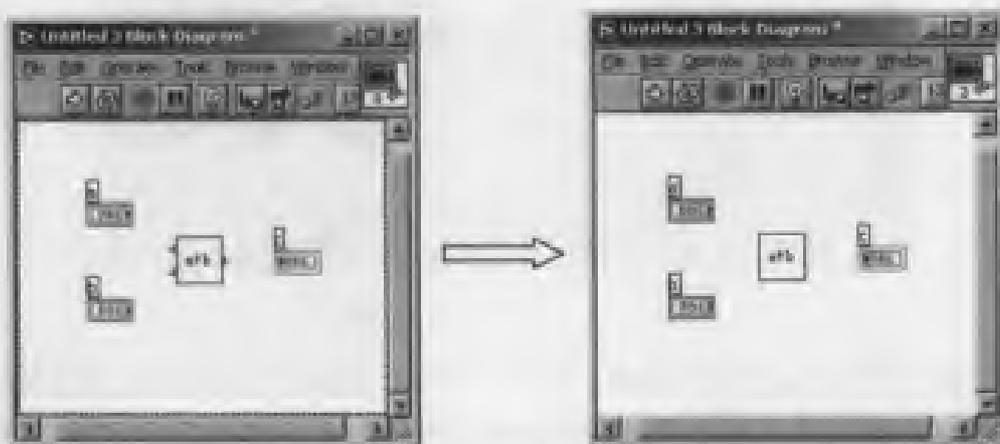


图 3.5.8 添加 SubVI

③ 用连线工具将 SubVI 的各个连接端口与主 VI 中的其他节点按照一定的逻辑关系连接起来。

至此,就完成了 SubVI 的调用。主 VI 的前面板及框图程序如图 3.5.9 所示。

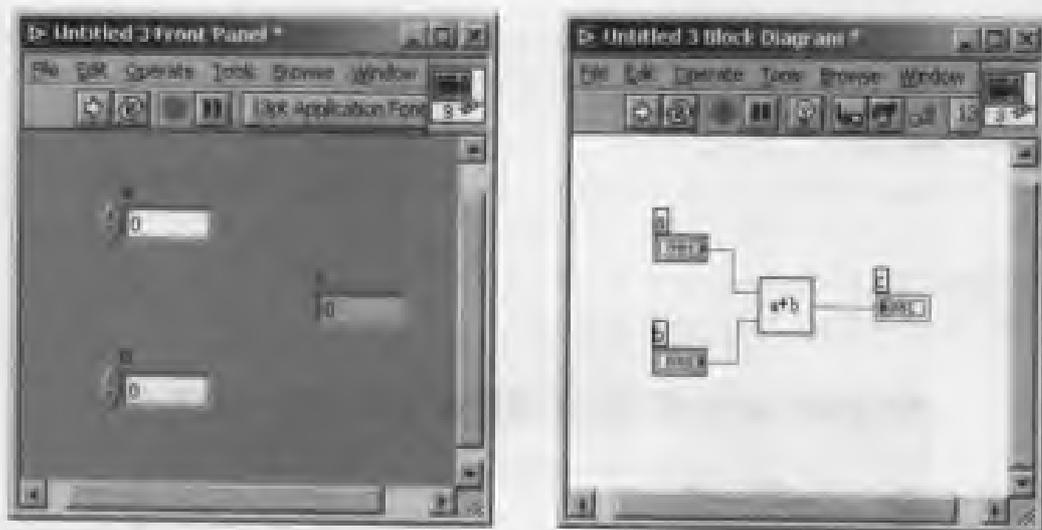


图 3.5.9 主 VI 的前面板及框图程序

采用上述的 SubVI 调用方式来调用一个 SubVI,只是将其作为一般的计算模块来使用,程序运行时并不显示其前面板。如果要将 SubVI 的前面板作为弹出式对话框来使用,则需要改变一些 VI 的属性设置。

在 SubVI 前面板窗口右上角图标的右键弹出菜单中选择 VI Properties..., 会出现一个 VI Properties 对话框,在对话框的 Category 栏中选择 Window Appearance, 将对话框页面切换到窗口显示属性页面,如图 3.5.10 所示。



图 3.5.10 VI Properties 对话框

在对话框中单击 Customize...按钮,弹出 Customize Window Appearance 对话框,如图 3.5.11 所示。在该对话框中选中 Show front panel when called 和 Close afterwards if originally closed, 单击 OK 按钮关闭对话框。

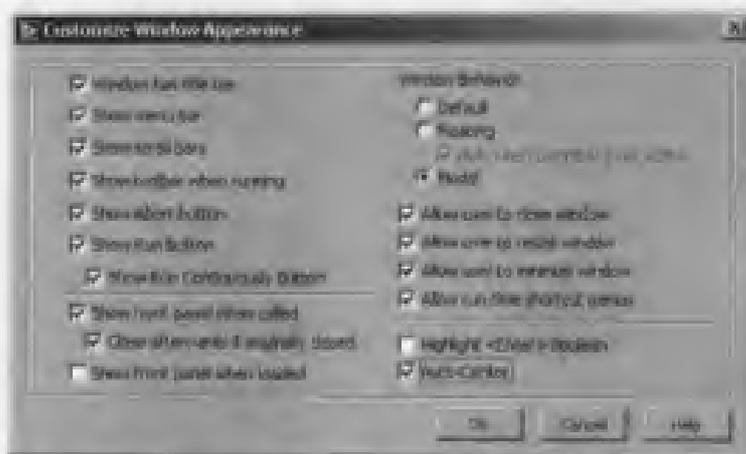


图 3.5.11 Customize Window Appearance 对话框

选中 Show front panel when called 后，当程序运行到这个 SubVI 时，其前面板就会自动弹出来。若再选中 Close afterwards if originally closed，则当 SubVI 运行结束时，其前面板会自动消失。

在 VI Setup 对话框中可以设定 VI 执行时的各种其他属性，本书的第 10 章将对此进行详细介绍。

3.6 Express VI

Express VI 是 LabVIEW 7 Express 的一个新特性，凭借 Express VI，LabVIEW 将编程效率提高到了一个新的水平。Express VI 是专门针对常见的测试与测量而创建的，它将常用的测量功能集成于一些简单易用、交互式的 VI 中。在使用时，只要双击 Express VI 就可以通过属性设置对话框配置采集、分析和显示功能。

3.6.1 Express VI 的特点

Express VI 共有 40 多个。在 Express 风格的模板视图中，Functions 模板中的 VI 主要是 Express VI，所有的 VI（包括标准 VI）都可以在 All Function 子模板中找到。更改模板视图风格可以从主菜单中选择 Tools→Options，在弹出的 Options 对话框上部的下拉列表框中选择 Controls/Functions Palette，然后在 Controls/Functions Palette 页面的 Palette View 下拉列表框中选择 Express。

Express VI 无论在外观上还是在使用上与 LabVIEW 的标准 VI 和节点都有很大不同。

1. 认识 Express VI

从外观上看，Express VI 的图标很大。以 Spectral Measurements 为例，图 3.6.1 给出了 Spectral Measurements 的图标。可以看到，整个图标被深蓝色的边框包围，背景是淡雅的淡蓝色，图标中心是一个小图标和 VI 的名称，小图标的两侧有代表输入和输出端口的三角箭头，在名称的下



图 3.6.1 Spectral Measurements 的图标

方有下拉箭头 .

图 3.6.1 所示的 Express VI 并没有显示端口名称, 如果要显示端口名称, 可以用鼠标(对象操作工具状态)拖动图标下边缘的尺寸控制点, 将端口名称显示出来, 名称旁边小箭头的位罝指明了端口的输入输出属性, 如图 3.6.2 所示。

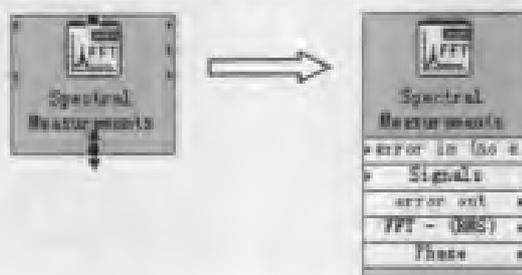


图 3.6.2 显示 Express VI 的端口

如果端口的名称过长, 不能完全显示, 可以在图标的右键弹出菜单中选择 Size To Text, Express VI 将根据端口名称的长度自动调整 VI 的宽度, 如图 3.6.3 所示。

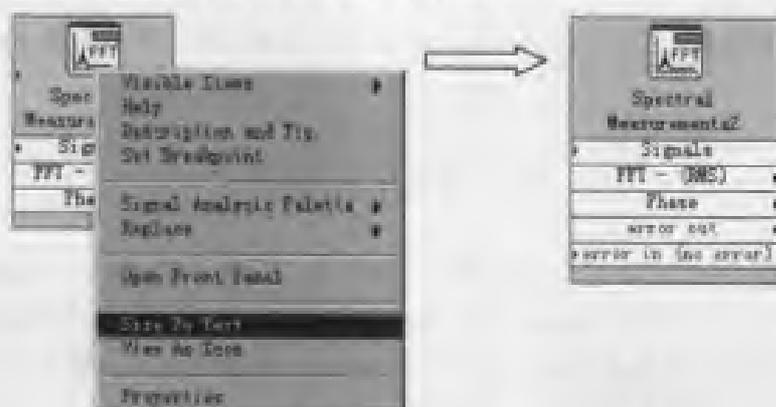


图 3.6.3 自动调整 Express VI 的宽度

如果觉得 Express VI 的图标太大, 也可以将其显示为小图标, 方法是在图标的右键弹出菜单中选择 View As Icon, 使该选项处于选中状态, 如图 3.6.4 所示。缩小后的图标依然以淡蓝色为背景, 并且四周带有导角。

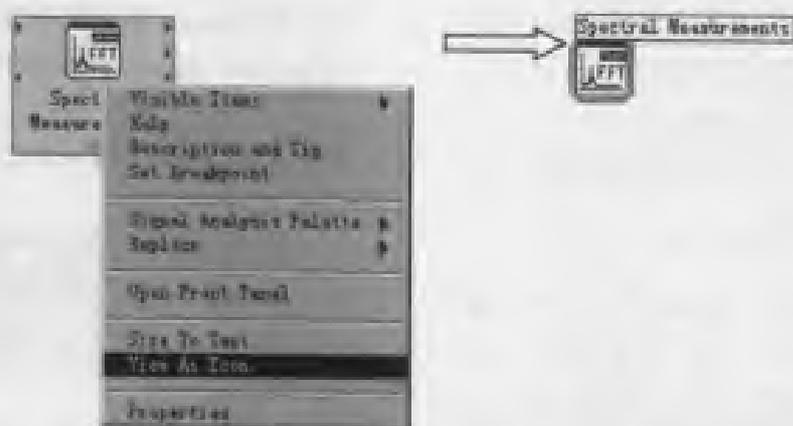


图 3.6.4 Express VI 的小图标

2. 标准 VI 的扩展模式

在 LabVIEW 7 Express 中, 标准 VI 可以显示为扩展模式 (Expandable), 扩展模式图标的外观与 Express VI 的图标有点类似, 但是边框为传统的黑色, 底色为淡黄色, 图标中也没有 VI 的名称。图 3.6.5 所示为 Basic Averaged DC-RMS.vi 扩展模式的图标。若要将标准 VI 的图标显示为扩展模式, 可以从选单中选择 Tools→Options, 在弹出的 Options 对话框上部的下拉列表框中选择 New and Changed in 7.0, 然后在 New and Changed in 7.0 页面中选中 Place subVIs as expandable 项, 则新添加到框图程序中的 Express VI 将显示为扩展图标。这只是外观上的区别, 更重要的是使用方法上的区别。

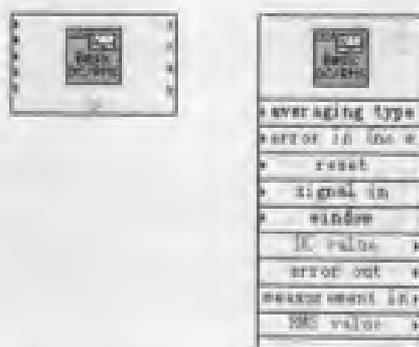


图 3.6.5 Basic Averaged DC-RMS.vi 扩展模式的图标

3. 动态数据类型 (Dynamic Data Type)

在标准 VI 中, 一个端口的数据类型是确定的, 如果连接了不同类型的数据 (如将一个布尔量连接到字符串类型的端口), LabVIEW 将会报错。而在 Express VI 中, 大部分数据端口可以连接多种数据类型, 这一功能的实现得益于动态数据类型。

除了以紫红色显示错误的信息输入、输出端口, Express VI 的端口数据类型只有两种: 布尔量和动态数据类型。布尔量的颜色为绿色, 动态数据类型的颜色为深蓝色。动态数据类型相当于一种中性数据结构, 很多种 LabVIEW 数据类型均可以转换成动态数据类型, 包括各种格式的实数、波形数据、布尔量及相应的数组, 并且这个过程是可逆的。在将不同类型的数据连接到 Express VI 的数据端口时, Express VI 将自动将其转换为动态数据类型, 这样就可以用一个端口连接多种数据类型, 转换过程由 LabVIEW 自动完成, 用户无须考虑。

4. 将 Express VI 转换为标准 VI

对于 Express VI, 无法查看其前面板和框图程序, 双击 Express VI 的图标会打开 Express VI 的属性设置对话框。若要查看 Express VI 的前面板和框图程序, 必须先将其转换为标准 VI。方法是在 Express VI 的图标上单击鼠标右键, 在弹出的快捷选单中选择 Open Front Panel, LabVIEW 将弹出提示对话框, 询问用户是否要将 Express VI 转换成标准 VI。单击 Convert 按钮即可进行转换, 如图 3.6.6 所示。转换后的图标如图 3.6.7 所示, 转换后的标准 VI 是可以编辑的。转换时将生成 Express VI 的副本, 而不会影响原 Express VI。

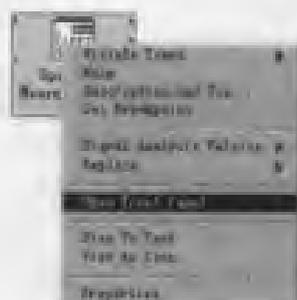


图 3.6.6 将 Express VI 转换为标准 VI

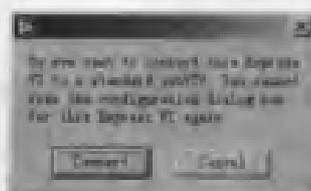


图 3.6.7 转换后的标准 VI

3.6.2 Express VI 的使用方法

Express VI 使用起来较之标准 VI 更为方便。

当 Express VI 处于默认设置时, 在将 Express VI 放置到框图程序中, 将弹出 Express VI 的属性设置对话框 (可以通过 LabVIEW 环境设置以禁止自动弹出属性设置对话框, 设置的方法见 3.6.1 节)。在编程时, 也可以双击 Express VI 的图标打开属性设置对话框。下面通过一个例子来看一下 Express VI 的使用方法。

例 3.6.1 使用 Express VI 进行频谱分析。

本例首先产生一个虚拟信号, 然后分析信号频谱。这里要用到两个 Express VI, 一个是 Simulate Signal, 另一个是 Spectral Measurements。Simulate Signal 与 Spectral Measurements 都位于 Functions 模板 → All Functions 子模板 → Input 子模板中, 如图 3.6.8 和图 3.6.9 所示。



图 3.6.8 Simulate Signal 的位置



图 3.6.9 Spectral Measurements 的位置

第一步, 将 Simulate Signal 放于框图程序中, 这时 LabVIEW 将自动打开 Simulate Signal 属性设置对话框。在对话框中进行如下设置:

- 在 Signal type 下拉列表框中选择正弦信号 Sine;
- 在 Frequency (Hz) 一栏中将频率设为 102 Hz;
- 选中 Add noise 选择框, 添加噪声;
- 在 Noise type 下拉列表框中选择白噪声 Uniform White Noise;
- 在 Noise amplitude 一栏中设置噪声幅度为 0.1。

在更改设置的时候, 可以从右上角 Result Preview 区域中观察当前设置的信号的波形。其他项保持默认设置, 完成后的设置如图 3.6.10 所示。单击 OK 按钮, 退出属性设置对话框。

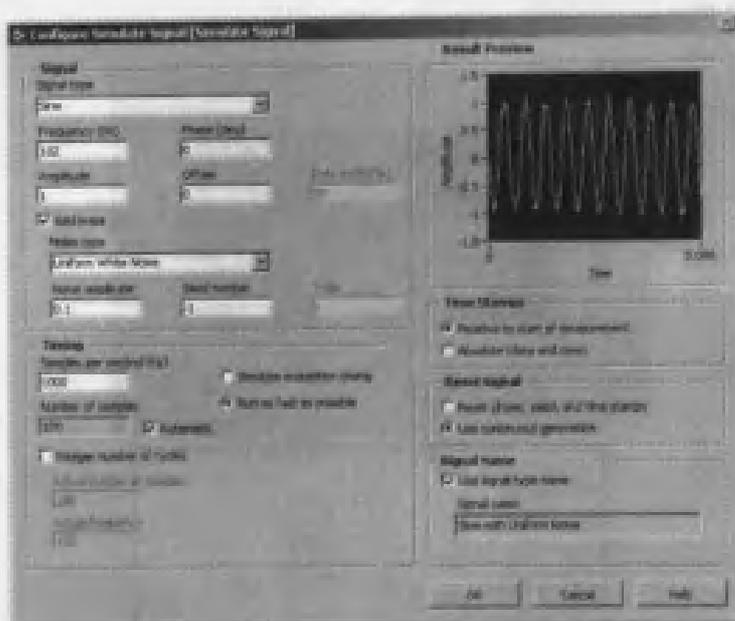


图 3.6.10 Simulate Signal 的属性设置对话框

第二步，将 Spectral Measurements 放于框图程序中，LabVIEW 将自动打开 Spectral Measurements 的属性设置对话框。在对话框中进行如下设置：

- 在 Spectrum Measurement 一栏中选择 Magnitude (RMS)；
- 在 Windows 下拉列表框中选择窗函数为 Hanning 窗；
- 选中 Averaging 选择框；
- 在 Mode 一栏中选择平均方式为 RMS。

保持其他属性为默认设置。完成后的设置如图 3.6.11 所示。

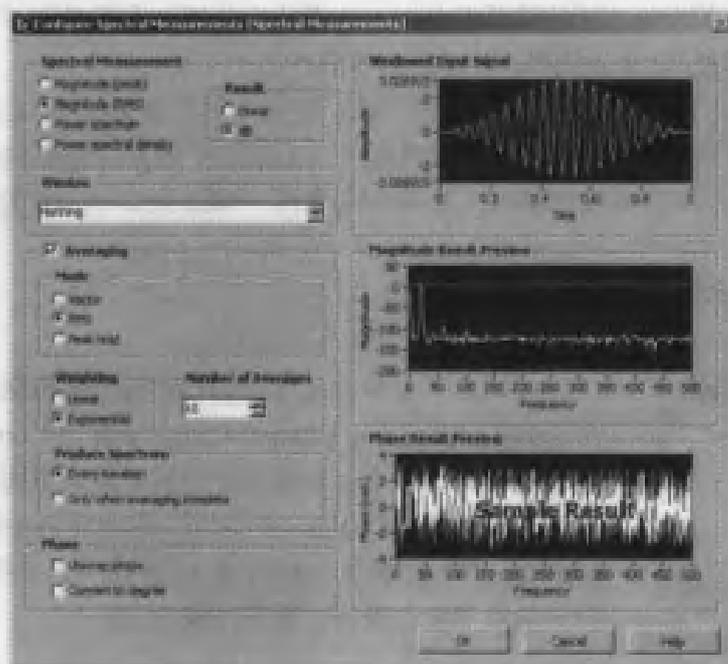


图 3.6.11 Spectral Measurements 的属性设置对话框

程序最终的前面板和框图程序如图 3.6.12 所示。

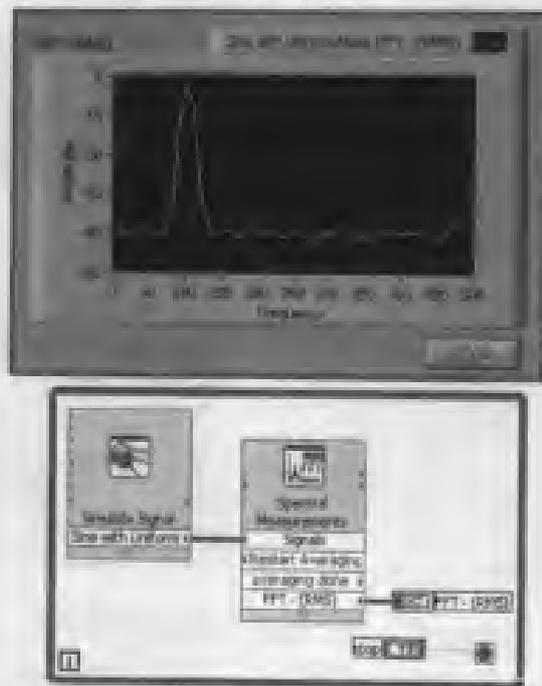


图 3.6.12 例 3.6.1 的前面板和框图程序

3.7 获取帮助

有效地获取帮助信息是快速掌握 LabVIEW 的一条捷径, LabVIEW 为用户提供了较为全面的帮助信息, 利用这些帮助信息, 用户可以获得 LabVIEW 中的模板、选单、工具、功能模块(节点)的使用方法及其功能介绍, 并能够逐步深入系统地学习 LabVIEW 编程环境的使用、编程技巧、LabVIEW 高级编程等内容。

LabVIEW 提供了各种获取帮助信息的方法, 包括实时上下文帮助(Context Help)、VI 及功能模块帮助(VI, Function&How-To Help)、LabVIEW 例程(LabVIEW Examples)、LabVIEW 书架(LabVIEW Bookshelf)以及网络资源(Web Resources)等。

3.7.1 实时上下文帮助



图 3.7.1 Context Help 窗口

实时上下文帮助是 LabVIEW 提供的快捷帮助, 在主选单中选择 Help→Show Context Help, 会弹出 Context Help 窗口, 窗口中实时显示鼠标所指的前面板对象或框图程序中功能模块的简短帮助信息, 如图 3.7.1 所示。单击窗口中的“Click here for more help”, 会弹出 LabVIEW Help 窗口, 如图 3.7.2 所示, 用户可以从该窗口获得完整的帮助信息。

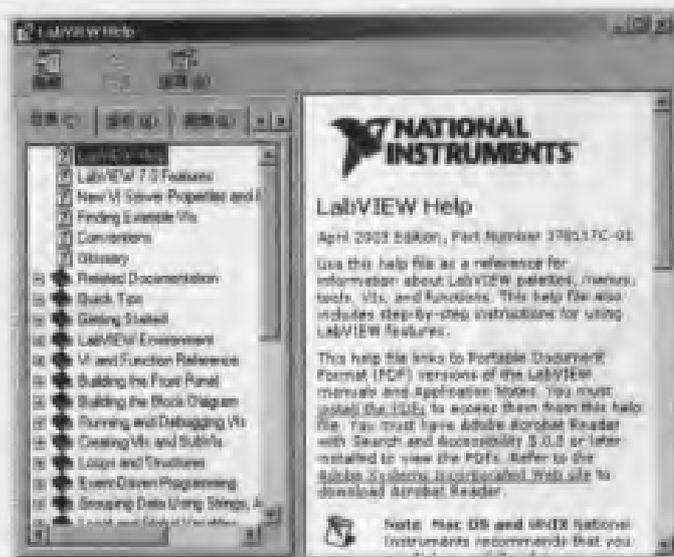


图 3.7.2 LabVIEW Help 窗口

3.7.2 VI 及功能模块帮助

VI 及功能模块帮助提供了 LabVIEW 全部的帮助信息，在主选单中选择 Help→VI, Function&How-To Help..., 会弹出 LabVIEW Help 窗口，这是一个 Windows 标准风格的帮助窗口，包含了 LabVIEW 全部的帮助信息，参见图 3.7.2。用户可以利用窗口中的目录、索引或搜索查找所需的帮助信息。

3.7.3 LabVIEW 例程

为方便用户快速掌握 LabVIEW 中各种功能模块的用法，LabVIEW 提供了大量的例程，这些例程几乎包含了 LabVIEW 所有功能模块的应用实例，并提供了大量的综合应用实例，利用这些实例来学习 LabVIEW 功能模块的用法是通向 LabVIEW 殿堂的捷径。在主选单中选择 Help→Find Examples..., 会弹出 NI Example Finder 窗口，利用该窗口中的目录，用户可以访问 LabVIEW 所提供的所有例程，如图 3.7.3 所示。

建议用户在学习 LabVIEW 过程中逐一研究、学习这些 LabVIEW 例程，并且最后熟悉这些例程，这可大大缩短用户学习 LabVIEW 的学习时间。另外，值得一提的是，由于这些例程包含了大量的综合应用实例，用户编程过程用所需的一些基本功能甚至一些复杂的功能几乎都能够在这类例程中找到，将这些例程中的框图程序直接拷贝到用户的应用程序中，可以为用户节约大量的编程时间。

若 LabVIEW 提供的这些例程还不能满足用户的需求，可以单击 NI Example Finder 窗口中的“Visit LabVIEW Zone”按钮，访问 NI 公司官方网站上的 LabVIEW 乐园，可以获得更多的 LabVIEW 实例。

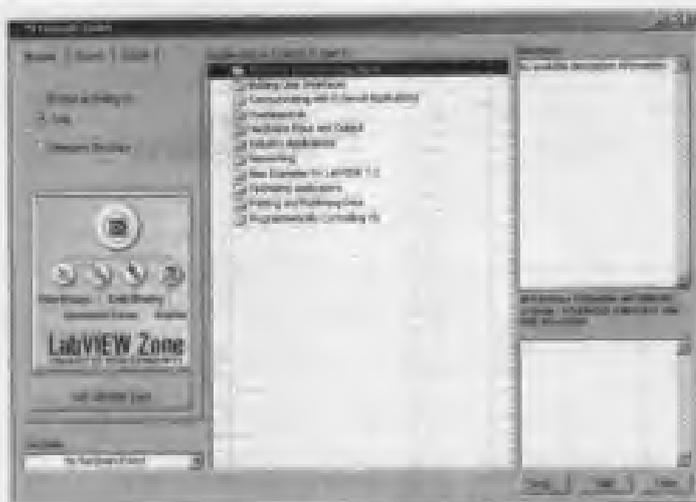


图 3.7.3 NI Example Finder 窗口

3.7.4 LabVIEW 书架

上述三种帮助方式为用户提供的都是分散、独立的帮助信息，用户并不能从中系统地学习 LabVIEW 的编程环境、编程方法、编程技巧以及一些高级的编程知识。为此，LabVIEW 提供了大量的 PDF 文档，称之为 LabVIEW 书架。从 LabVIEW 书架中，用户可以系统地学习 LabVIEW 的相关内容。在主选单中选择 Help→Search the LabVIEW Bookshelf...，会弹出 LabVIEW Bookshelf 窗口（需要安装 Acrobat Reader 5.0 或更高版本），如图 3.7.4 所示。

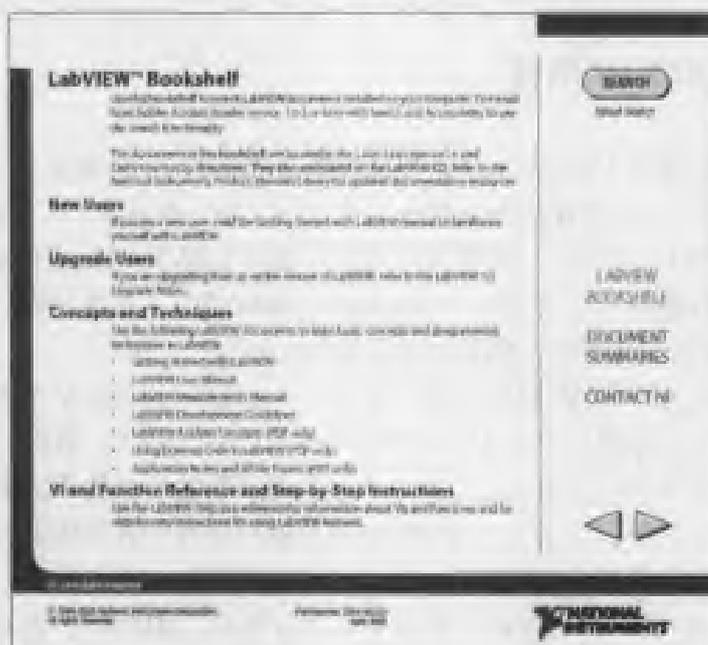


图 3.7.4 LabVIEW Bookshelf 窗口

在该窗口中用户可以选择各种 PDF 文档，包括 LabVIEW 入门（Getting Started with LabVIEW）、LabVIEW 用户手册（LabVIEW User Manual）、LabVIEW 测试手册（LabVIEW Measurements Manual）、LabVIEW 开发指南（LabVIEW Development Guidelines）、LabVIEW

分析概念(LabVIEW Analysis Concepts)、在 LabVIEW 中使用外部代码(Using External Code in LabVIEW)、LabVIEW 应用笔记和白皮书 (Application Notes and White Papers) 等。

3.7.5 LabVIEW 网络资源

若上述帮助内容并不能满足用户需求, 用户还可以访问互联网上的 LabVIEW 网络资源。在主菜单中选择 Help→Web Resources..., 会通过 Internet Explorer 连接到 NI 公司的官方网站 <http://www.ni.com>, 该网站提供了大量的 LabVIEW 网络资源和相关链接, 包括大量的用户应用笔记、例程、LabVIEW 论坛和各种其他的 LabVIEW 资源等, 可以在这些 LabVIEW 网络资源中寻找所需的帮助信息。

第4章 数据操作

LabVIEW 支持所有的数据类型。数字可以是各种精度的浮点数或整数，布尔型、字节、字符串及数字可以自由的联合到一种数据结构中，满足程序的需要。另外，LabVIEW 还有一种特殊的功能，称之为数据多态性 (Polymorphism)，数据多态性是指 LabVIEW 可以自动适应各种类型的输入数据。有了数据多态性的支持，即使是初学者，也可以在 LabVIEW 中很好的使用各种复杂的数据类型。LabVIEW 大多数的内部功能是数据多态性的，不过用户自己写的 VIs 可能不支持数据多态性，也许它们在两个不同的数字类型之间可以适应，但在两个不同的数据类型，如字符串到数字之间就可能不适应。在大多数情况下，LabVIEW 都能够适应用户提供的各种数据类型。例如一个数字可以直接加到一个数组的每一个元素上，这在其他大多数语言中，则必须利用一个循环才能完成。当然，LabVIEW 的数据多态性也有一定的限制，例如不能将一个布尔值加到一个字符串上，因为其结果很难定义，所以，LabVIEW 不允许这种操作。

本章将主要介绍 LabVIEW 中常用的数据类型，与这些数据类型相关的前面板对象，以及 Functions 模板中与之相关的数据运算。将要介绍的数据运算包括数学运算、布尔运算、比较运算以及字符串运算等。

4.1 数据类型

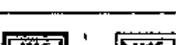
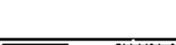
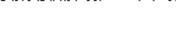
LabVIEW 的数据类型与传统编程语言中的数据类型基本类似，除了具有一般的数据类型之外，还有一些独特的数据类型。

表 4.1.1 总结了 LabVIEW 所支持的所有数据类型及其相对应的前面板对象的默认值及其端口图标。每种数据类型的端口图标都有一种颜色，以示区别。控制端口图标的边框为粗实线，端口右侧有一个向右的箭头，表示输出数据，指示端口的图标的边框为细实线，端口左侧有一个向左的箭头，表示输入数据。

表 4.1.1 LabVIEW 所支持数据类型表

数据类型	默认值	端口图标	备注
单精度浮点型 (Single-precision, floating-point numeric)	0.0	 	
双精度浮点型 (Double-precision, floating-point numeric)	0.0	 	
扩展精度浮点型 (Extended-precision, floating-point numeric)	0.0	 	
复数单精度浮点型 (Complex single-precision, floating-point numeric)	0.0 + 0.0i	 	
复数双精度浮点型 (Complex double-precision, floating-point numeric)	0.0 + 0.0i	 	

续表

数据类型	默认值	端口图标	备注
复数扩展精度浮点型 (Complex extended-precision, floating-point numeric)	0.0 + 0.0i		
有符号 8 位整数 (8-bit signed integer numeric)	0		
有符号 16 位整数 (16-bit signed integer numeric)	0		
有符号 32 位整数 (32-bit signed integer numeric)	0		
无符号 8 位整数 (8-bit unsigned integer numeric)	0		
无符号 16 位整数 (16-bit unsigned integer numeric)	0		
无符号 32 位整数 (32-bit unsigned integer numeric)	0		
<64.64>位时间标识 (<64.64>-bit time stamp)	当地时间日期		
枚举 (Enumerated type)	--		
布尔 (Boolean)	FALSE		
字符串 (String)	空字符串		
数组 (Array)	-		端口图标的颜色根据数组中包含元素的数据类型确定
簇 (Cluster)	—		可以包含多种不同的数据类型。如果簇内的元素都为数字类型，则端口图标的颜色为棕色；若包含其他的数据类型，则端口图标的颜色为粉红色
路径 (Path)	空路径		
动态数据 (Dynamic)	--		主要用于 Express Vis，包括数据及其相关的属性，例如：信号名称、时间、日期等
波形数据 (Waveform)	—		一种特殊类型的簇，包含一个波形的数据值、开始时间和两个数据点之间的时间间隔 Δt
数字波形 (Digital waveform)	—		一种特殊类型的簇，包含一个二进制数字波形的串二进制数字值、开始时间、两个数据点之间的 Δx 以及二进制数字波形的属性
二进制数字 (Digital)	—		包含数字信号 (digital signals) 的数据
标识 (Reference number (refnum))	—		
变体 (Variant)	—		包含控制或指示的名称、数据类型以及数据本身
I/O 名称 (I/O name)	—		将用户配置的资源传递给 I/O Vis，用来与一台仪器或一个测量设备通信
图片 (Picture)	—		显示一副图片，包括线、圆、文本或其他格式 (BMP, JPG 等) 的图形、图片

本章将主要介绍上述数据类型中的数字型数据、布尔型数据和字符串型数据, 以及与这些数据类型有关的一些功能节点的详细用法, 其他的数据类型将在后续的相关章节中介绍。

4.1.1 数字型

在 LabVIEW 中, 数字型是一种基本的数据类型, 其分类比较详细, 如表 4.1.2 所示。

表 4.1.2 LabVIEW 数字类型表

详细分类	图标	磁盘存储所占位数	小数位数	范围
无符号 8 位整型		8	2	0~255
无符号 16 位整型		16	4	0~65 535
无符号 32 位整型		32	9	0~4 294 967 295
有符号 8 位整型		8	2	-128~127
有符号 16 位整型		16	4	-32 768~32 767
有符号 32 位整型		32	9	-2 147 483 648~2 147 483 647
单精度浮点型		32	6	-Inf ~ +Inf
双精度浮点型		64	15	-Inf ~ +Inf
扩展型		128	15~33	-Inf ~ +Inf
单精度复数		64	6	无
双精度复数		128	15	无
扩展型复数		256	15~33	无
128 位时间标记	无	<64.64>	15	5.421 010 862 427 522 170 037 264 004 349 7e -20 s~9 223 372 036 854 775 808 s

在 LabVIEW 中, 数据类型是隐含在控制、指示及常量之中的。因此, 有必要介绍一下数字类型的前面板对象。数字类型的前面板对象包含在 Controls 模板→All Controls 子模板→Numeric 子模板和 Controls 模板→All Controls 子模板→Classic Controls 模板→Classic Numeric 子模板中, 如图 4.1.1 所示。

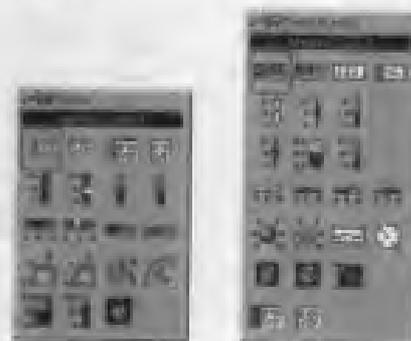


图 4.1.1 Numeric 子模板

传统编程语言中的数据分为变量和常量两种。在某种意义上, LabVIEW 中的数据也可以这么分, Numeric 模板中的前面板对象就相当于传统编程语言中的数字变量, 而 LabVIEW 中的数字常量是不出现在前面板窗口中的, 只存在于框图程序窗口中, 在 Functions 模板→Numeric 子模板中有一个名为 Numeric Constant 的节点, 这个节点就是 LabVIEW 中的一种数字常量, 如图 4.1.2 所示。注意, LabVIEW 的前面板对象(变量)中的数据可以在程序运行时由用户通过键盘或鼠标改变或者由程序动态赋值, 但常量只能在编程时设定, 一旦程序运行, 常量的值就是一个常数, 不能改变, 并且所有数据类型的常量都是如此。



图 4.1.2 Numeric 子模板中的 Numeric Constant 节点

Numeric 子模板包括多种不同形式的控制和指示, 它们的外观各不相同, 有数字量、滚动条、水箱、温度计、旋钮、表头、刻度盘以及颜色框等。这些前面板对象的本质是完全相同的, 都是数字型, 只是外观不同而已。LabVIEW 的这一特点为创建虚拟仪器的前面板提供了很大的方便。只要理解并掌握了其中一个的用法, 就可掌握其他全部数字类型的前面板对象的用法。

下面以 Numeric 子模板中的 Digital Control 为例, 介绍如何设置该对象的属性。

首先在 VI 前面板窗口中创建一个 Digital Control, 然后在 Digital Control 的右键弹出菜单中选择 Properties, 可以弹出 Numeric Properties 对话框, 如图 4.1.3 所示。Numeric Properties 对话框共包括 4 个属性页面: Appearance (外观)、Data Range (数据范围)、Format and Precision (格式和精度)、Documentation (文档) 等。

(1) Appearance 属性页面

在 Appearance 属性页面中可以设置 Digital Control 的外观属性, 包括 Label (标签、名称)、Caption (标题)、Enabled State (激活状态)、Show radix (显示基数)、Show increment/decrement button (显示递增/递减按钮) 等, 如图 4.1.3 所示。



图 4.1.3 Numeric Properties 对话框 (Appearance 属性页面)

(2) Data Range 属性页面

在 Data Range 属性页面中可以设定 Digital Control 的数字类型、默认值、最大最小值

和递增量等属性,如图 4.1.4 所示。注意,在设定最大最小值时,不能超出该数字类型的数据范围,否则设定值无效。



图 4.1.4 Numeric Properties 对话框 (Data Range 属性页面)

在 Data Range 属性页面中单击 Representation, 出现一个图形化下拉选单, 在选单中可以设定表 4.1.2 中所列的数据类型, 如图 4.1.5 所示。注意, 表 4.1.2 中所列的数字类型“128 位时间标记”是一种特殊的数字类型, 与其他的数字类型之间不能直接转换, 需要利用框图程序窗口中的 Functions 模板→All Functions 子模板→Numeric 子模板→Conversion 子模板中的 To Time Stamp 节点实现转换。



图 4.1.5 Representation 下拉选单

(3) Format and Precision 属性页面

在 Format and Precision 属性页面中可以设定 Digital Control 的格式和精度, 如图 4.1.6 所示。

(4) Documentation 属性页面

在 Documentation 属性页面中可以设定 Digital Control 的 Description (描述) 信息和 Tip strip (提示条) 信息, 如图 4.1.7 所示。



图 4.1.6 Numeric Properties 对话框 (Format and Precision 属性页面)



图 4.1.7 Numeric Properties 对话框 (Documentation 属性页面)

另外，滚动条、水箱、温度计、旋钮、表头和刻度盘等前面板对象还有一种特殊的属性，叫做 Text Labels。当这些前面板对象处于 Text Labels 状态时，其刻度值可以是字符串，此时，每一个刻度值对应一个惟一的数字值，该数字值就是这些前面板对象的数值，其数字类型是无符号 32 位整数。在这些前面板对象的右键弹出选单中选择 Text Labels，对象就会变为 Text Labels 状态，如图 4.1.8 所示。

图 4.1.8 所示是一个处于 Text Labels 状态的水平滚动条，其默认的 Text Labels 只有 2 个：min 和 max。

- 当滑块处于 min 位置时，水平滚动条的值为 0；
- 当滑块处于 max 位置时，水平滚动条的值为 1。

在水平滚动条右侧的是它的 Text Display，Text Display 中显示的 Text 值是与滑块的位置对应的。在 Text Display 的右键弹出选单中选择 Edit Items...，在弹出的 Slide Properties 对话框的 Text Labels 属性页面中可以编辑滚动条的 Text 条目，如图 4.1.9 所示。

在 Text Labels 属性页面中插入一个 Text 条目 middle，如图 4.1.10 所示。



图 4.1.8 处于 Text Labels 状态的水平滚动条

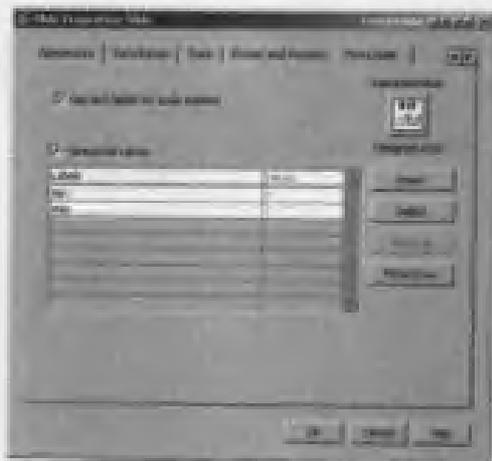


图 4.1.9 Slide Properties 对话框的 Text Labels 属性页面



图 4.1.10 Slide Properties 对话框的 Text Labels 属性页面



图 4.1.11 添加一个 Text 条目的水平滚动条

修改 Text 条目后的水平滚动条如图 4.1.11 所示, 此时, 水平滚动条上的 Text 条目与水平滚动条数值的对应关系如下:

- 当滚动条上的滑块处于 min 位置时, 水平滚动条的值为 0;
- 当滑块处于 middle 位置时, 水平滚动条的值为 1;
- 当滑块处于 max 位置时, 水平滚动条的值为 2。

在水平滚动条的 Text Display 的右键弹出选单中选择 Disable Item, 在程序运行时, 这个 Text 条目将被禁用, 如图 4.1.12 所示。

Text Labels 是一种非常有用的功能, 可以用为仪器面板上的档位开关, 例如电风扇的风速调节档位开关, 如图 4.1.13 所示。LabVIEW 的这种功能可以帮助用户设计更加友好的虚拟仪器前面板。



图 4.1.12 禁用一个 Text 条目



图 4.1.13 电风扇风速调节档位开关

4.1.2 布尔型

布尔型即逻辑型，它的值为真 (True) 或假 (False)，或者为 1 或 0。在 LabVIEW 中，布尔型体现在布尔型前面板对象中。布尔型前面板对象包含在 Controls 模板 → All Controls 子模板 → Boolean 子模板和 Controls 模板 → All Controls 子模板 → Classic Controls 模板 → Classic Boolean 子模板中，如图 4.1.14 所示。



图 4.1.14 Boolean 子模板

可以看到，上述模板中有各种不同的布尔型前面板对象，如不同形状的按钮、指示灯和开关等，这都是从实际仪器的开关、按钮、指示灯演化来的，十分形象。利用这些布尔按钮，用户可以设计出很逼真的虚拟仪器前面板。与数字类型的前面板对象类似，这些不同的布尔控制也是外观不同，内涵相同，都是布尔型，只有 0 和 1 两个值。

与传统编程语言中的逻辑量不同的是，这些布尔型前面板对象有一个独特的属性，叫做 Mechanical Action，这是模拟真实继电器机械开关触点开 / 闭特性的一种专门开关控制特性。在布尔控制的右键弹出菜单中选择 Mechanical Action，会出现一个图形化的下拉菜单，菜单中有 6 种不同的 Mechanical Action 属性：Switch When Pressed、Switch When Released、Switch Until Released、Latch When Pressed、Latch When Released 和 Latch Until Released，如图 4.1.15 所示。

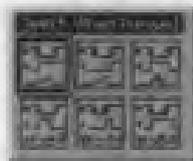


图 4.1.15 Mechanical Action 下拉菜单

对于一个按钮，例如 Push Button，在弹起状态  时它的值为 0，在按下的状态  时它的值为 1。Mechanical Action 属性定义了用鼠标单击按钮时，按钮的值在什么时刻由 0 阶跃为 1，在什么时刻由 1 阶跃为 0。这一点对于真实的仪器按钮来说是非常重要的。由于 LabVIEW 是用来设计虚拟仪器的，因此这一点也显得很重要。灵活使用按钮的这种属性，对于能否开发出优秀的虚拟仪器具有一定的意义。Mechanical Action 下拉菜单中的图标和文字说明很直观地表示出了鼠标的点按动作与按钮 0,1 值的变化关系，在此不再赘述。

与数字量类似，Boolean 子模板中的布尔型前面板对象相当于传统编程语言中的布尔变量，LabVIEW 中的布尔常量则存在于框图程序中。在 Functions 模板 → All Functions 子模板 → Boolean 子模板中有两个名为 True Constant 和 False Constant 的节点，这两个节点就是 LabVIEW 中的布尔常量。如图 4.1.16 所示。



图 4.1.16 Boolean 子模板中的 True Constant 和 False Constant 的节点

4.1.3 字符串型与路径



图 4.1.17 String & Path 子模板

字符串是 LabVIEW 中一种基本的数据类型。LabVIEW 提供了功能强大的字符串控件和字符串运算功能函数，但其使用却十分简单。路径也是一种特殊的字符串，专门用于对文件路径的处理。布尔型前面板对象包含在 Controls 模板 → All Controls 子模板 → String & Path 子模板和 Controls 模板 → All Controls 子模板 → Classic Controls 模板 → Classic String & Path 子模板中，如图 4.1.17 所示。

String & Path 子模板中共有三种对象：字符串 (String)、组合框 (Combo Box) 和文件路径 (File Path)。下面对这三种对象进行介绍。

1. 字符串对象

字符串对象用于处理和显示各种字符串。用数据操作工具或文本编辑工具单击字符串对象的显示区，即可在对象显示区的光标位置进行字符串的输入和修改。字符串的输入修改操作与常见的文本编辑操作几乎完全一样，LabVIEW 的一个字符串对象就是一个简单的文本编辑器。用户可以通过双击鼠标并拖动鼠标来选定一部分字符，对已选定的文字进行剪切、拷贝和粘贴等操作，还可改变选定文字的大小、字体和颜色等属性。同样，常用的文本编辑功能键在输入字符串时同样有效，如光标键、换页、退格键和删除键等。有 3 种方法可以用来结束字符串的输入：输入硬回车键 (Ctrl+Enter)、单击 LabVIEW 工具条上的 Enter Text 按钮 或在字符串控件外的任意位置单击鼠标左键。

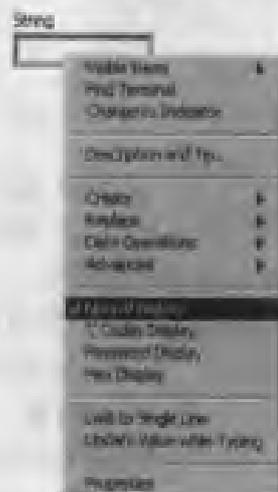


图 4.1.18 在字符串对象的右键弹出菜单中切换显示模式

对于不可见的控制字符，如 Tab 和 Esc 等，其输入要依赖于其他输入方法，而不能直接在控件中键入。当字符输入完毕后，可以在其右键弹出菜单中选择

对于不可见的控制字符，如 Tab 和 Esc 等，其输入要依赖于其他输入方法，而不能直接在控件中键入。

当字符输入完毕后，可以在其右键弹出菜单中选择

Data Operations → Make Current Value Default 项保存, 当下次重新启动该 VI 时, 字符串的内容将保持不变。

字符串对象支持 ASCII 码字符, 有些是可以显示的普通字符, 有些是不可以显示的特殊控制字符。字符串对象显示字符串时共有四种显示模式: 普通显示模式 (Normal Display)、“\”代码显示模式 (“\” Codes Display)、密码显示模式 (Password Display) 和十六进制显示模式 (Hex Display)。在字符串对象的右键弹出选单中可以对这四种显示模式进行切换, 如图 4.1.18 所示。

(1) Normal Display: 正常显示

在这种显示模式下, 字符串控件显示键入的所有字符, 但是有些字符在这种模式下是不可显示的, 如制表符、Esc、声音 (BEEP) 等。

(2) “\” Codes Display: 控制码显示

在这种显示模式下, 字符串控件除了显示普通的字符外, 还以 “\” 的方式显示不可显示的特殊控制字符。在程序调试、向仪器或其他外设传输控制字符时经常使用这种模式。特殊控制字符的命令格式及含义如表 4.1.3 所示。

表 4.1.3 特殊字符表

格 式	含 义
\00-FF	斜线后接两位十六进制整数, 显示一个以为 ASCII 值该值的字符, 斜线后表示十六进制 ASCII 值的字符必须大写
\b	退格符 (Backspace, ASCII BS, 相当于\08)
\f	进格符 (Formfeed, ASCII FF, 相当于\0C)
\n	换行符 (Linefeed, ASCII LF, 相当于\0A)
\r	回车符 (Carriage return, ASCII CR, 相当于\0D)
\t	制表符 (Tab, ASCII HT, 相当于\09)
\s	空格符 (Space, 相当于\20)
\\	反斜线 (Backslash, ASCII \, 相当于\5C)

在 LabVIEW 中, 如果反斜线后接的是大写字母, 并且是一个合法的十六进制整数(00-FF), 则把它理解为一个字符的十六进制 ASCII 码值, 如 \BFWARE, 它被认为是一个 ASCII 码值为 BF 的字符再接单词 WARE。如果反斜线后接的是小写字母, 而且是表 4.1.3 中的一个命令字符, 则把它理解为一个控制字符, 如 \bFWARE, 则认为它是一个退格符 (\b) 再接单词 FWARE。如果输入为 \Bfware, Bf 不是一个合法的十六进制数, 但 B 是十六进制整数的一个合法的组成字符, 在这种情况下, LabVIEW 在 B 前自动加上一个 0, 使其成为一个合法的十六进制数, 即 LabVIEW 把 \Bfware 理解为一个 ASCII 码为 0B 的字符再接单词 fware。如果反斜线后既不是合法的十六进制数, 也不是表 4.1.3 所示特殊字符中的任何一个, 则忽略反斜线的作用, 如把 \LabVIEW 理解为单词 LabVIEW, 把 \labview 理解为单词 labview。

(3) Password Display: 口令显示

这种模式主要用来输入口令或密码。在这种模式下键入字符的显示均以 “*” 字符代替。拷贝或剪切操作时, 出现在剪贴板上的也是 “*” 字符。

(4) Hex Display: 十六进制显示

在这种模式下, 将显示输入字符对应的十六进制 ASCII 码值。这种模式在程序调试及与仪器进行通信时是很有用的。

下面以一个字符串为例介绍这四种显示模式，这个字符串共包括“abcdefg”、一个空格、“12345”和一个换行符等 14 个字符，如图 4.1.19 所示。



图 4.1.19 一个字符串的四种显示模式实例

字符串对象有单行和多行等两种存储字符串的方式：



图 4.1.20 在字符串对象的右键弹出选单中切换字符串存储模式

- 字符串对象处于单行存储方式时，字符串对象内只能存放一行字符串，不接受回车换行；
- 字符串对象处于多行存储方式时，字符串对象内可以存放多行字符串。

在字符串对象的右键弹出选单中选择 Limit to Single Line 可以对这两种存储模式进行切换，如图 4.1.20 所示。

若字符串对象中的字符串数量较多，那么字符串对象在有限的空间内不能将所有的字符串同时显示出来，此时，可以为字符串对象添加一个滚动条，通过滚动条就可以访问到字符串对象中所有的字符。添加滚动条的方法是在字符串对象的右键弹出选单中选择 Visible Items→Scrollbar，如图 4.1.21 所示。

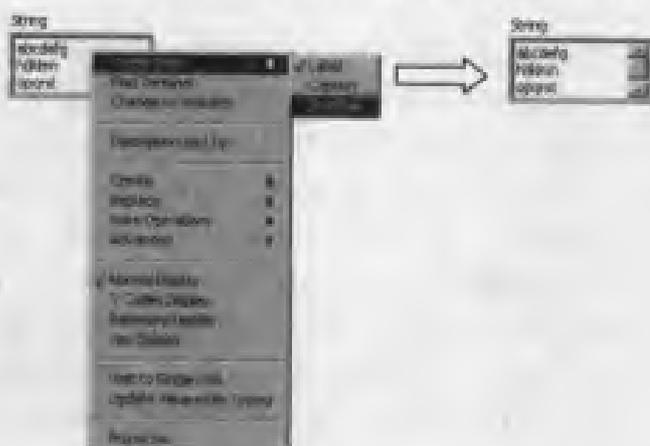


图 4.1.21 在字符串对象中显示滚动条

另外，字符串对象还有一种数据更新模式称之为 Update Value while Typing，其含义是在用户通过键盘键入字符时实时更新字符串对象本身的值，并同时通过框图程序中的端口输出。若字符串对象不处于这种模式，只有当用户在通过键盘输入字符完毕之后才会更新

字符串对象本身的值，并通过框图程序中的端口输出。在字符串对象的右键弹出选单中选择 Update Value while Typing 可以对这两种模式进行切换，如图 4.1.22 所示。

2. 组合框对象

组合框对象是一种特殊的字符串对象，除了具有字符串对象的功能之外，还添加了一个字符串列表，在字符串列表中，可以预先设定几个预定的字符串，供用户选择，如图 4.1.23 所示。单击组合框对象右侧的下拉按钮 ，会出现一个下拉选单，选单中列出了预先设定的字符串选项，用户可以任意选择。

在组合框对象的右键弹出选单中选择 Edit Items...，会弹出一个 Edit Items 对话框，用于编辑、预设组合框对象中可选择的字符串条目，如图 4.1.24 所示。

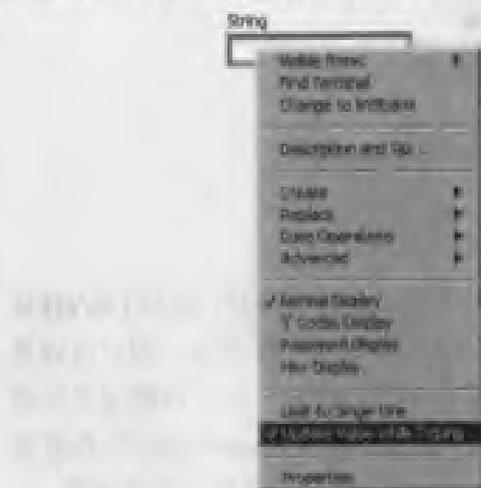


图 4.1.22 在字符串对象的右键弹出选单中切换数据更新模式

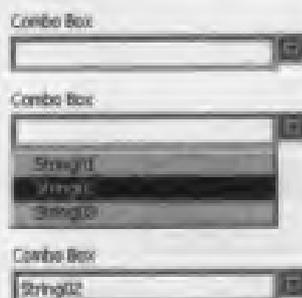


图 4.1.23 组合框对象



图 4.1.24 组合框对象的 Edit Items 对话框

另外，在组合框对象的右键弹出选单中还有一个选项 Allow Undefined Strings，意思是允许未定义的字符串，如图 4.1.25 所示。若选中该选项，组合框对象允许用户通过键盘输

入任意的字符串；若未选中该选项，组合框对象不允许用户通过键盘输入任意的字符串，只能通过鼠标在对象的下拉选单中选择预定的字符串。

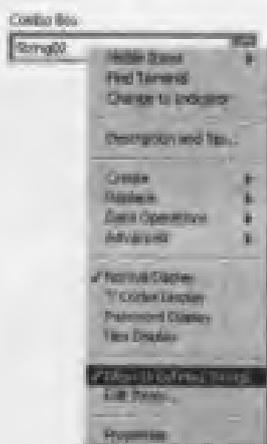


图 4.1.25 组合框对象的 Allow Undefined Strings 选项

3. 文件路径对象

文件路径对象也是一种特殊的字符串对象，专门用于处理文件的路径，可与 LabVIEW 的文件 I/O 节点配合使用，如图 4.1.26 所示。用户可以直接在文件路径对象中输入文件的路径，也可以通过单击右侧的 Browse 按钮  打开一个 Windows 标准文件对话框。在 Windows 标准文件对话框中寻找所需要的文件。



图 4.1.26 文件路径对象

4.2 数学运算

数学运算是编程语言中的基本运算之一，LabVIEW 中的数学运算主要由 Functions 模板 → All Functions 子模板 → Numeric 子模板中的节点完成，如图 4.2.1 所示。

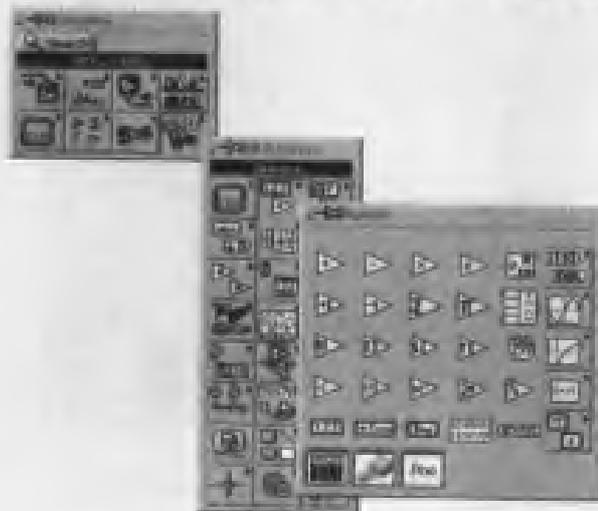


图 4.2.1 Numeric 子模板

Numeric 子模板由数字常量、基本数学运算节点、类型转换 (Conversion) 节点、三角函数 (Trigonometric) 节点、对数 (Logarithmic) 节点、复数 (Complex) 节点、附加数字常数 (Additional Numeric Constant) 节点和三个 Express 节点组成。

4.2.1 数字常量

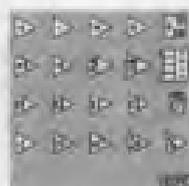
数字常量如图 4.2.2 所示, 包括 Numeric Constant, Enum Constant, Ring Constant, Time Stamp Constant 等 4 个不同类型的数字常量, 这些常量从本质上讲都是数字类型, 只是其表现形式不同。



图 4.2.2 数字常量

4.2.2 基本数学运算节点

基本数学运算节点如图 4.2.3 所示, 主要实现加、减、乘、除等基本数学运算。



基本数学运算节点支持数字类型的数据, 但与一般编程语言提供的运算符相比, LabVIEW 中数学运算节点功能更强, 使用更灵活, 它不仅支持单一的数值量输入, 还可支持处理不同类型的复合型数值量, 比如由数字量构成的数组、簇或簇数组等。数字类型的数据包括浮点数、整数和复数。

对于有两个输入的节点, 可以使用下列输入组合。

- 相同结构 (Similar)。两个输入的数据结构相同, 节点输出数据的数据结构与输入的数据结构相同。
- 单标量 (One Scalar)。一个输入是数值标量, 另一个输入是数组或簇, 节点的输出是数组或簇。
- 簇数组 (Array of Clusters)。一个输入是簇, 另一个输入可以是簇构成的数组, 运算结果是簇数组。

各种输入组合如图 4.2.4 所示。

这就是 LabVIEW 的数据多态性 (Polymorphism) 表现之一, 下面以一个实例进一步说明这一点。

例 4.2.1 将一个数字加到一个数组中。

本例将一个无符号 32 位整数数组和一个双精度浮点数相加, 其结果是一个双精度浮点数数组, VI 的前面板和框图程序如图 4.2.5 所示。

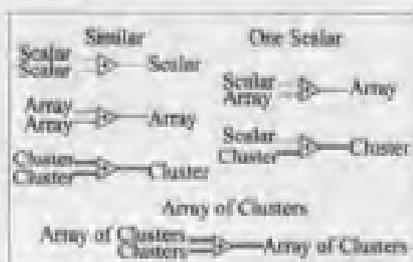


图 4.2.4 各种输入组合

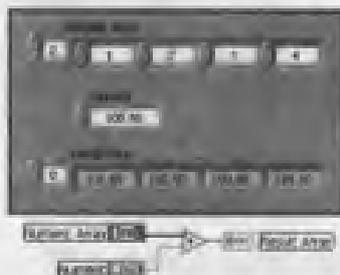
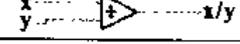
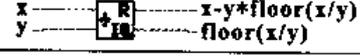
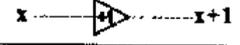
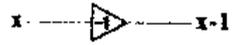
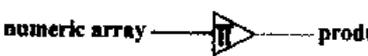
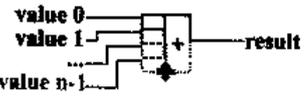
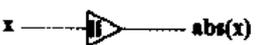
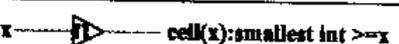
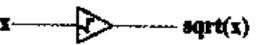
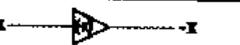


图 4.2.5 例 4.2.1 的前面板和框图程序

基本运算节点的用法如表 4.2.1 所示。

表 4.2.1 基本运算节点用法表

节点名称	图标及端口	功能	备注
Add		加	当一个输入是数字标量, 另一个输入是一个数组时, 节点将数组中的每一个元素与这个数字标量相加。其他节点与此类似
Subtract		减	
Multiply		乘	不能进行矩阵相乘。若输入是两个矩阵, 该节点仅将两个矩阵第一行的第一个元素相乘
Divide		除	当 y=0 时, 返回值为 Inf
Quotient & Remainder		整除	
Increment		递增	
Decrement		递减	
Add Array Elements		求数组所有元素之和	输入数组的维数不限
Multiply Array Elements		求数组所有元素之积	输入数组的维数不限
Compound Arithmetic		复合运算	在节点图标的右键弹出选单中选择 Change Mode, 出现的下拉选单中有加、乘、与、或、异或等 5 种运算模式可选。用对象操作工具拖动节点图标下边缘 (或上边缘) 上的尺寸控制点, 或在输入端口的右键弹出选单中选择 Add Input, 可添加输入端口。在输入端口或输出端口的右键弹出选单中选择 Invert, 可使该端口对数据进行“非”操作
Absolute Value		绝对值	
Round To Nearest		求整	四舍五入
Round To -Infinity		求整	返回小于等于 x 的最大整数
Round To +Infinity		求整	返回大于等于 x 的最小整数
Square Root		开方	当 x<0 时, 返回值为 NaN
Negate		求负	
Scale By Power Of 2		求 x 与 2 的 n 次幂的积	

续表

节点名称	图标及端口	功能	备注
Sign	number  -1, 0, 1	求符号	
Reciprocal	x  1/x	求倒数	当 $x=0$ 时, 返回值为 Inf
Random Number (0-1)	 number(0 to 1)		
Expression Node	input  $2+x*\log(x)$  output	使用表达式来计算包含一个变量的公式或方程式。该节点允许使用非整数的数字类型	允许下列函数在表达式中使用: abs, acos, acosh, asin, asinh, atan, atanh, ceil, cos, cosh, cot, csc, exp, expm1, floor, getexp, getman, int, intrz, ln, lnpl, log, log2, max, min, mod, round, sec, sec, sign, sin, sinh, sqrt, tan, tanh

4.2.3 类型转换节点

LabVIEW 中的数据由两部分组成: 数据本身和类型描述。类型描述是不可见的, 它在内部指导 LabVIEW 处理与之相联系的数据, 这也是多态性功能是如何知道所连接的数据是何种类型的原因。一个数据的类型描述确定了它的数据类型 (如双精度浮点数组) 和字节数。当一个数据转化为另一数据类型时, 它的数据内容及类型描述都将被改变, 例如, 一个 4 字节 32 位符号整数转化为一个双精度浮点数 (8 字节 64 位浮点数), 包含在这 4 字节 32 位符号整数中的数值将转化为一个特征值和一个指数, 其类型描述也得到了相应的转换, 只是新数据类型会多占用一些内存。两个标量数字类型之间的转换很简单, 通常仅需要一个 CPU 指令。若转换包含聚合的数据类型 (如字符串、簇等), 则会多花费一些时间。这是因为首先要运行一个特殊的转换程序来解释数值, 其次, 新数据类型会或多或少地需要内存, 因此还需要呼出系统的内存管理器。

在 LabVIEW 中可以利用类型转换节点在各种不同的数据类型之间进行转换。所有的类型转换节点都包含在 Conversion 子模板中, 如图 4.2.6 所示。



图 4.2.6 Conversion 子模板

除了 Byte Array to String, String to Byte Array, Covert Unit 和 Cast Unit Bases 等节点外,

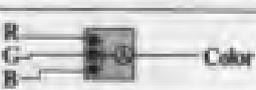
模板中的其他节点的输入可以是多样化的数据类型, 包括标量值、标量数组、标量值的簇和标量值的簇的数组等。

各节点的用法见表 4.2.2。

表 4.2.2 类型转换节点用法表

节点名称	图标及端口	功能	备注
To Byte Integer	number →  8bit integer	将输入的 number 转换为 8 位整数	范围: -128 ~ 127
To Word Integer	number →  16bit integer	将输入的 number 转换为 16 位整数	范围: -32768 ~ 32767
To Long Integer	number →  32bit integer	将输入的 number 转换为 32 位整数	范围: $-2^{31} \sim 2^{31}-1$
To Unsigned Byte Integer	number →  unsigned 8bit integer	将输入的 number 转换为 8 位无符号整数	范围: 0 ~ 255
To Unsigned Word Integer	number →  unsigned 16bit integer	将输入的 number 转换为 16 位无符号整数	范围: 0 ~ 65535
To Unsigned Long Integer	number →  unsigned 32bit integer	将输入的 number 转换为 32 位无符号整数	范围: $0 \sim 2^{31}-1$
To Time Stamp	number →  time stamp	将输入的 number 转换为时间标记, 时间起点为 1904 年 1 月 1 日 8 时 00 分 00 秒, 以秒为单位进行转换	例如, 当 number=1 时, 对应的 time stamp=1904 年 1 月 1 日 8 时 00 分 01 秒; 当 number=120 时, 对应的 time stamp=1904 年 1 月 1 日 8 时 02 分 00 秒
To Single Precision Float	number →  single precision float	将输入的 number 转换为单精度浮点数	
To Double Precision Float	number →  double precision float	将输入的 number 转换为双精度浮点数	
To Extended Precision Float	number →  extended precision float	将输入的 number 转换为扩展型浮点数	
To Single Precision Complex	number →  single precision complex	将输入的 number 转换为单精度复数	
To Double Precision Complex	number →  double precision complex	将输入的 number 转换为双精度复数	
To Extended Precision Complex	number →  extended precision complex	将输入的 number 转换为扩展型复数	
Number To Boolean Array	number →  Boolean array	将整数转换为布尔数组	布尔数组的每一个元素对应 number 相应二进制数的每一位。数组的长度为 8、16 或 32。当输入的 number 为浮点数时, 节点先将其转换为 32 位无符号整数, 将该整数转换为布尔数组

续表

节点名称	图标及端口	功能	备注
Boolean Array To Number	  number	将布尔数组转换为 32 位无符号整数	
Boolean To (0, 1)	  0, 1	将布尔值转换为 16 位整数 0 或 1。	输入为 False 时，输出为 0；输入为 True 时，输出为 1
String To Byte Array	  unsigned byte array	将字符串转换为 8 位无符号整数数组	每一个字符对应数组中的一个元素，输出的数字是字符在 ASCII 表中的编号
Byte Array To String	  string	将 8 位无符号整数数组转换为字符串	整数与字符的对应关系同上
Convert Unit	 x  y	将一个物理数字（一个数字加一个单位）转换为一个纯数字（不带单位的数字），或将一个纯数字转换为一个物理数字	在节点图标的右键弹出菜单中选择 Unit...，在弹出的对话框中可以设定输入输出的单位
Cast Unit Bases	 unit (none)  x  x	转换基础单位	将与输入 x 相关联的基础单位转换为由 unit (none) 端口输入的基础单位。比如，将与 x 相关联的长度单位转换为时间单位
Color to RGB	 Color  Resolved Color  R  G  B	将一个输入的颜色值转换为这个颜色相对应的红绿蓝分量	
RGB to Color	 R  G  B  Color	将一个颜色的红绿蓝分量转换为这个颜色的颜色值	

用类型转换节点可以进行数据类型间的转换，但用户在编程时通常不使用这些节点，而会在不同的数字类型之间直接连线，如图 4.2.7 所示。



图 4.2.7 两个数字类型直接连线

注意，图 4.2.5 中的指示量 Numeric Indicator 的端口上出现了一个灰点，这称为强制转换点（Coercion），这种用法与用一个明确的类型转换节点一样，也可以完成不同数据类型间的转换。但在使用类型转换节点或强制转换功能时，必须警惕精度的丢失。

一个双精度浮点型数字量或扩展型浮点数字量的数值可以达到 10^{+257} ，若将一个如此大的数转换为一个范围仅为 0~255 的 8 位无符号整数，很明显源数值会丢失，例如，图 4.2.6 所示的转换结果如图 4.2.8 所示。



图 4.2.8 双精度浮点型数字量转换为无符号 8 位整数数字量

所以，如果用户使用类型转换节点或强制转换功能不当，很可能会造成一些极为隐蔽

的逻辑错误。所有编程语言都存在这类问题，LabVIEW 也不例外。

4.2.4 三角函数节点

Trigonometric 子模板中的节点可实现各种三角函数运算，如图 4.2.9 所示。

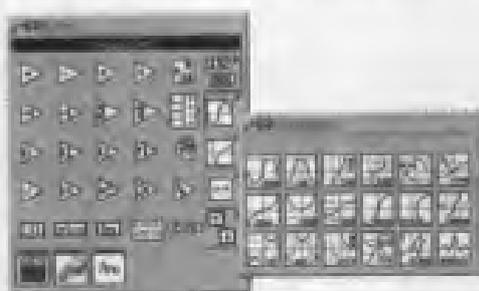


图 4.2.9 Trigonometric 子模板

注意，该模板中的节点均以弧度为单位。节点的输入可以是数字标量、数字量的数组或簇、数字量的簇的数组等。

各节点的用法见表 4.2.3。

表 4.2.3 三角函数节点用法表

节点名称	图标及端口	功 能	备 注
Sine	$\sin(x)$	$\sin(x)$	
Cosine	$\cos(x)$	$\cos(x)$	
Tangent	$\tan(x)$	$\tan(x)$	
Inverse Sine	$\arcsin(x)$	$\arcsin(x)$	$-1 < x < 1$ ，否则，返回值为 NaN
Inverse Cosine	$\arccos(x)$	$\arccos(x)$	$-1 < x < 1$ ，否则，返回值为 NaN
Inverse Tangent	$\arctan(x)$	$\arctan(x)$	
Hyperbolic Sine	$\sinh(x)$	$\sinh(x)$	
Hyperbolic Cosine	$\cosh(x)$	$\cosh(x)$	
Hyperbolic Tangent	$\tanh(x)$	$\tanh(x)$	
Inverse Hyperbolic Sine	$\operatorname{arsinh}(x)$	$\operatorname{arsinh}(x)$	
Inverse Hyperbolic Cosine	$\operatorname{arcosh}(x)$	$\operatorname{arcosh}(x)$	$x \geq 1$ ，否则，返回值为 NaN
Inverse Hyperbolic Tangent	$\operatorname{artanh}(x)$	$\operatorname{artanh}(x)$	$-1 < x < 1$ ，否则，返回值为 NaN

续表

节点名称	图标及端口	功能	备注
Cosecant	 $1/\sin(x)$	$1/\sin(x)$	
Secant	 $1/\cos(x)$	$1/\cos(x)$	
Cotangent	 $1/\tan(x)$	$1/\tan(x)$	
Sine & Cosine	 $\sin(x)$ $\cos(x)$	$\sin(x)$ 与 $\cos(x)$	
Inverse Tangent (2 input)	 $\text{atan2}(x, y)$	$\tan^{-1}(x/y)$	
Sinc	 $\sin(x)/x$	$\sin(x)/x$	

4.2.5 对数节点

对数节点用于各种对数运算，包含在 Logarithmic 子模板中，如图 4.2.10 所示。



图 4.2.10 Logarithmic 子模板

各节点的用法见表 4.2.4。

表 4.2.4 对数节点用法表

节点名称	图标及端口	功能	备注
Exponential	 $\exp(x)$	$\exp(x)$	
Power Of 10	 10^x	10^x	
Power Of 2	 2^x	2^x	
Power Of x	 x^y	x^y	当 y 为非整数时， x 须为复数，或 $x > 0$ ，否则，返回值为 NaN
Exponential (Arg) -1	 $\exp(x)^{-1}$	$\exp(x)^{-1}$	

续表

节点名称	图标及端口	功能	备注
Natural Logarithm	 $\ln(x)$	$\ln(x)$	$x > 0$ $x = 0$ 时, 返回值为 $-\text{Inf}$ $x < 0$ 时, 返回值为 NaN
Logarithm Base 10	 $\lg(x)$	$\lg(x)$	$x > 0$ $x = 0$ 时, 返回值为 $-\text{Inf}$ $x < 0$ 时, 返回值为 NaN
Logarithm Base 2	 $\log_2(x)$	$\log_2(x)$	$x > 0$ $x = 0$ 时, 返回值为 $-\text{Inf}$ $x < 0$ 时, 返回值为 NaN
Logarithm Base x	 $\log_x(y)$	$\log_x(y)$	$x > 0$ 且 $y > 0$ $y = 0$ 时, 返回值为 $-\text{Inf}$ 当 x, y 为非复数, 且 $x \leq 0$ 或 $y \leq 0$ 时, 返回值为 NaN
Natural Logarithm (Arg + 1)	 $\ln(x+1)$	$\ln(x+1)$	$x > -1$ $x = -1$ 时, 返回值为 $-\text{Inf}$ $x < -1$ 时, 返回值为 NaN 当 x 接近 0 时, 用该节点比直接将 x 加到 1 上, 再用 Natural Logarithm 节点计算要精确得多

4.2.6 复数节点

Complex 子模板中的节点用于复数运算, 如图 4.2.11 所示。



图 4.2.11 Complex 子模板

节点的输入可以是数字标量、数字量的数组或簇、数字量的簇的数组等, 各节点的用法见表 4.2.5。

表 4.2.5 复数运算节点用法表

节点名称	图标及端口	功能
Complex Conjugate	 $x+iy$ \rightarrow $x-iy$	求共轭
Polar To Complex	 r θ \rightarrow $r e^{j\theta}$	将幅角和幅值转换为相应的复数
Complex To Polar	 $r e^{j\theta}$ \rightarrow r θ	将复数转换为相应的幅角和幅值
Re/Im To Complex	 x y \rightarrow $x+iy$	将两个实数 x, y 组成复数 $x+iy$
Complex To Re/Im	 $x+iy$ \rightarrow x y	将复数 $x+iy$ 分解为两个实数 x, y

4.2.7 附加常数节点

为方便编程计算, LabVIEW 还提供了各种常用的常数, 如 π 、 e 、万有引力常数等, 这些常数均在 Additional Numeric Constants 子模板中, 如图 4.2.12 所示。



图 4.2.12 Additional Numeric Constants 子模板

注意, 子模板中的 Color Box Constant, Listbox Symbol 和 Ring Error Ring 这 3 个节点在 VI 处于编辑状态时可以设定其常量值, 在运行状态时就不能设定了。

各种常数节点见表 4.2.6。

表 4.2.6 常数节用法点表

节点名称	图标	功能	备注
Color Box Constant		颜色常量	用数据操作工具单击图标, 在弹出的颜色对话框中可选择颜色
Listbox Symbol Ring Constant		列表框符号环	用数据操作工具单击图标, 在弹出的图形化选单中可选择不同的列表框符号
Error Ring Constant		错误环, 环中包括了 LabVIEW 中的各种可能出现的错误	用操作工具单击图标, 在出现的下拉选单中可以选择各种类型的错误, 如内存使用、网络、打印和文件 I/O 等方面的错误
Pi		π	$3.1415926535897932e+0$
Pi Multiplied By 2		2π	$6.2831853071795865e+0$
Pi Divided By 2		$\pi/2$	$1.5707963267948966e+0$
Reciprocal Of Pi		$1/\pi$	$3.1830988618379067e-1$
Natural Logarithm Of Pi		$\ln \pi$	$1.1447298858494002e+0$
Natural Logarithm Base		自然对数基 e	$2.7182818284590452e+0$
Reciprocal Of e		$1/e$	$3.6787944117144232e-1$
Base 10 Logarithm Of e		$\log_{10} e$	$-4.3429448190325183e-1$
Natural Logarithm Of 10		$\ln 10$	$2.3025850929940597e+0$
Natural Logarithm Of 2		$\ln 2$	$6.9314718055994531e-1$
Negative Infinity		$-Inf$	负无穷

续表

节点名称	图标	功能	备注
Positive Infinity		+inf	正无穷
Planck Constant (J/Hz)		普朗克常数	$6.62607550000000125e-34$ (J/Hz)
Elementary Charge (C)		基本电荷	$1.602177330000000000e-19$ (C)
Speed Of Light In Vacuum (m/s)		真空中的光速	$2.99792458000000000e+8$ (m/s)
Gravitational Constant (N m ² /kg ²)		重力常数	$6.67259000000000000e-11$ (N m ² /kg ²)
Avogadro Constant (1/mol)		阿伏伽德罗常数	$6.02213670000000000e+23$ (1/mol)
Rydberg Constant (1/m)		里德伯常数	$1.09737315340000000e+7$ (1/m)
Molar Gas Constant (J/(mol K))		摩尔气体常数	$8.3145100000000000e+0$ (J/(mol K))

4.3 布尔运算

布尔运算相当于传统编程语言中的逻辑运算。传统编程语言使用逻辑运算符将关系表达式或逻辑量连接起来, 形成逻辑表达式, 逻辑运算符包括与 and、or、not 等。在 LabVIEW 中这些逻辑运算符是以图标的形式出现的。布尔运算节点包含在 Functions 模板 → All Functions 子模板 → Boolean 子模板中, 如图 4.3.1 所示。

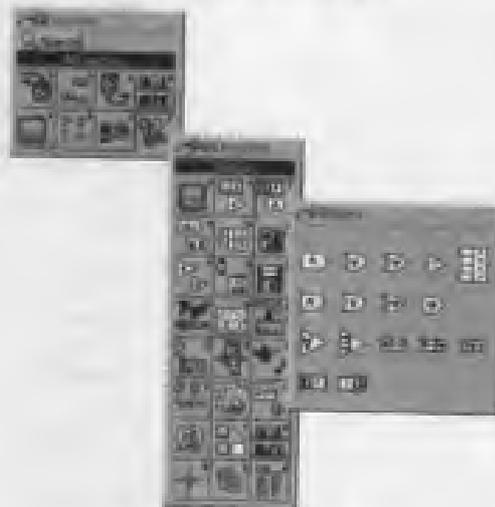
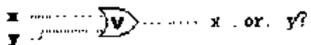
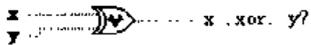
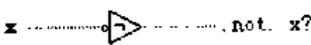
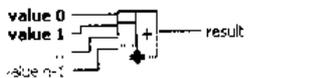
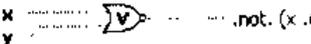


图 4.3.1 Boolean 子模板

图 4.3.1 中, 布尔运算节点的图标与集成电路常用逻辑符号一致, 可以使用户很方便的使用这些节点而无须重新记忆。

布尔运算节点的用法见表 4.3.1。

表 4.3.1 布尔运算节点用法表

节点名称	图标及端口	功能	备注
逻辑与 (And)		与	x, y 的数据类型可以是布尔型、整型、元素为布尔型或整型的数组及簇。当 x, y 为整型时, 节点将其转化为相应的二进制数, 然后将这两个二进制数的每一位相与, 最后的输出结果为相与后的十进制数
逻辑或 (Or)		或	x, y 的数据类型与 and 节点相同
异或 (Exclusive Or)		异或	x, y 的数据类型与 and 节点相同
逻辑非 (Not)		非 (Not)	x 的数据类型与 and 节点相同
复合运算 (Compound Arithmetic)		复合运算	用法与 Numeric 模板中的 Compound Arithmetic 节点相同
与非 (Not And)		与非	x, y 的数据类型与 and 节点相同
或非 (Not Or)		或非	x, y 的数据类型与 and 节点相同
同或 (Not Exclusive Or)		同或	x, y 的数据类型与 and 节点相同
蕴含 (Implies)		蕴含	x=F, y=F =>T x=T, y=F =>F x=F, y=T =>T x=T, y=T =>T x, y 的数据类型与 and 节点相同
数组与 (And Array Elements)		数组与	当输入数组中的所有元素均为 True 时, 节点的输出为 True, 否则为 False; 输入可以是任意维数组
数组或 (Or Array Elements)		数组或	当输入数组中的所有元素均为 False 时, 节点的输出为 False, 否则为 True; 输入可以为任意维数组
数字转换为布尔数组 (Number To Boolean Array)		数字转换为布尔数组	用法与 Numeric 模板中的 Number To Boolean Array 节点相同
布尔数组转换为数字 (Boolean Array To Number)		布尔数组转换为数字	用法与 Numeric 模板中的 Boolean Array To Number 节点相同
布尔值转换为 0 或 1 (Boolean To (0, 1))		布尔值转换为 0 或 1	用法与 Numeric 模板中的 Boolean To (0, 1) 节点相同
布尔常数 (Boolean Constant)		布尔常数	在编辑状态下, 用数据操作工具单击节点图标可改变布尔常数的值

4.4 字符串运算

LabVIEW 主要用于测控软件的开发。因此,经常需要同各种仪器进行通信,需要处理各种不同的文本命令,而这些命令主要是由字符串组成,对字符串进行合成、分解、变换是测试软件开发人员经常遇到的问题,因此,LabVIEW 为用户提供了丰富的功能强大并且简单易用的字符串运算节点。字符串运算如同数学运算、布尔运算一样,是 LabVIEW 中一类最基本的运算。在 LabVIEW 中这些字符串运算节点也是以图标形式的出现,字符串运算节点包含在 Functions 模板→All Functions 子模板→String 子模板中,如图 4.4.1 所示。

String 子模板由字符串常量 (String Constant)、基本字符串运算节点、字符串/数字转换 (String/Number Conversion) 节点、字符串/数组/路径转换 (String/Array/Path Conversion) 节点和附加字符串运算 (Additional String Functions) 节点组成。



图 4.4.1 String 子模板

4.4.1 字符串常量

字符串常量如图 4.4.2 所示,包括 String Constant, Empty String Constant (空字符串), Carriage Return Constant (回车符), Line Feed Constant (换行符), End of Line Constant (回车换行符) 和 Tab Constant (制表符) 等 6 个不同类型的字符串常量,这些常量从本质上讲都是字符串,只是其表现形式和用法稍有不同。



图 4.4.2 字符串常量

表 4.4.1 列出了这些常量的功能和用法。

表 4.4.1 字符串常量用法表

节点名称	图标及端口	功 能	备 注
String Constant		字符串常量	当 VI 处于编辑状态时, 用鼠标 (数据操作工具或文本编辑工具状态) 单击字符串常量的图标, 即可利用键盘输入任意字符
Empty String Constant		字符串常量-空字符	
Carriage Return Constant		字符串常量-回车符 (\r)	
Line Feed Constant		字符串常量-换行符 (\n)	
End of Line Constant		字符串常量-回车换行符 (\r\n)	
Tab Constant		字符串常量-制表符 (\t)	

4.4.2 基本字符串运算

基本字符串运算节点如图 4.4.3 所示, 主要实现对字符串的一些最基本的运算, 例如求字符串的长度、合成字符串、字符串的大小写转换等。

基本字符串运算节点的用法见表 4.4.2。

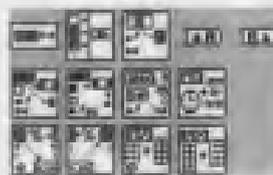
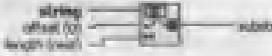
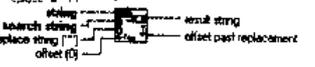
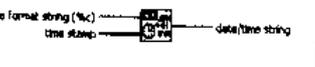
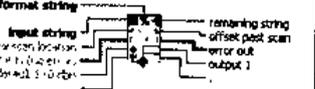
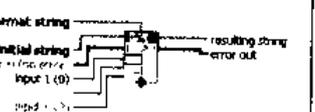


图 4.4.3 基本字符串运算节点

表 4.4.2 基本字符串运算节点用法表

节点名称	图标及端口	功 能	备 注
String Length		返回字符串中字符的个数, 以字节为单位	若连接到 String 端口上的数据是一个 String 数组, 则 Length 端口输出的是一个相同维度的数字数组, 数组中的每一个元素值表示 String 数组中相应位置元素的字符串长度
Concatenate Strings		将输入的字符串 String1, String2, ..., String n-1 按照前后次序连接成一个新的字符串。若某一个输入端口所输入的是一个 String 数组, 则节点首先将该数组内的元素按照从前到后的顺序连接成一个字符串, 然后再将这个字符串与其他输入端口所输入的字符串相连接	用鼠标拖动节点图标下边缘 (或上边缘) 上的尺寸控制点, 或在图标输入端口的右键弹出菜单中选择 Add Input, 可增加节点的输入端口
String Subset		返回输入字符串中的指定的子字符串, 子字符串的起始位置由 offset 端口确定, 长度由 length 端口确定	
To Upper Case		将输入字符串内的英文字母转换为大写字母	如果字符串中含有非英文字母的字符, 则节点不对这些字符进行任何处理

续表

节点名称	图标及端口	功能	备注
To Upper Case		将输入字符串内的英文字母转换为小写字母	如果字符串中含有非英文字母的字符, 则节点不对这些字符进行任何处理
Replace Substring		在输入字符串中指定的位置插入、删除或替换一个子字符串。	若输入到 length 端口中的数字为 0, 则节点会将输入到 substring 端口中的子字符串插入到由 string 端口输入的字符串中, 插入位置由 offset 端口指定; 若输入到 substring 端口中的是一个空字符串, 则节点会从 offset 端口指定的位置删除由 length 端口指定长度的字符; 若输入到 substring 端口中的子字符串不是一个空字符串, 且输入到 length 端口中的数字大于 0, 则节点会用这个子字符串在 offset 端口指定的位置处替换由 length 端口所指定长度的字符串
Search and Replace String		将一个或所有指定的子字符串替换为另一个子字符串。节点从 offset 端口指定的位置开始搜索 search string 端口所指定的子字符串, 然后将搜索到的第一个 (或全部, 如果 replace all? 端口的输入值为 True) 子字符串替换为由 replace string 端口所输入的字符串	该节点的搜索是一种最为简单的搜索, 若用户需要模糊匹配搜索或更加复杂的搜索, 则需要使用 Search and Replace Pattern VI 或 Match Pattern 节点
Match Pattern		在 string 端口中输入的字符串中搜索由 regular expression 端口输入的一个正规表达式, 搜索开始位置由 offset 端口指定。正规表达式的语法请见表 4.4.3	该节点的功能与 Search and Replace Pattern VI 类似
Format Date/Time String		按照用户指定的格式将一个时间标记值或一个数字值作为时间显示。	时间输出格式请见表 4.4.4
Scan From String		扫描从 input string 端口输入的字符串, 并将其转换为由 format string 端口指定的格式。	当对输入的字符串的格式非常明确时, 可以使用该节点, 若需从一个文件中扫描字符串, 可以使用 Scan From File 节点。format string 格式的语法与 C 语言相同, 具体的格式定义请见表 4.4.5, 表 4.4.6 列出了一些 format string 格式的实例
Format Into String		将字符串、数字、路径或布尔量格式化为文本, 文本的格式由 format string 端口指定, format string 格式的定义参见表 4.4.5	若需将数据格式化为文本, 并将其写入一个文件, 可以使用 Format Into File 节点。表 4.4.7 列举了该节点在输入不同的 format string 时对应的输出情况

续表

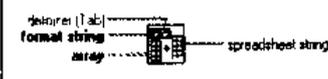
节点名称	图标及端口	功能	备注
Spreadsheet String To Array		将 spreadsheet string 端口输入的表单格式的字符串数据转换为一个数组，数组的数据格式由 array type 端口指定	
Array To Spreadsheet String		将一个任意维数的数组转换为一个字符串格式的表格，这个表格包括制表符、列的分隔符、行的终止符 EOL，对于 3 维或更高维数的数组，这个表格还包括分页标识	

表 4.4.3 Match Pattern 节点的正规表达式 (Regular Expression) 语法

需要进行匹配搜索的字符串	正规表达式
Volts	Volts
Volts 的所有大写和小写形式，例如 VOLTS、Volts、volts 等	[Vv][Oo][Ll][Tt][Ss]
空格“ ”、加号“+”、减号“-”	[+-]
需要进行匹配搜索的字符串	正规表达式
一个数字序列 (包含一个或多个数字)	[0-9]+
0 个或多个空格	\s* 或 * (即一个空格跟一个星号)
一个或多个空格、制表符、换行符或回车符	[\t\r\n\s]+
除了数字外还有一个或多个字符	[~0-9]+
单词 Level，恰好在字符串的 offset 端口所指定的位置	^Level
单词 Volts，恰好在字符串的末尾	Volts\$
圆括号内的长字符串	(.*)
圆括号内的长字符串，且该字符串不包含任何圆括号	([~()])*
字符[]	[[]]
cat、dog、cot、dot、cog 等	[cd][ao][tg]

表 4.4.4 Format Date/Time String 节点的时间输出格式表

格式	含义	格式	含义
%d	显示日期的天值	%a	显示星期值
%m	显示月值	%H	显示 24 小时制的小时值
%y	显示二位的年值	%I	显示 12 小时制的小时值
%Y	显示四位的年值	%M	显示时间分值
%x	按本国习惯显示日期	%S	显示时间秒值
%X	按本国习惯显示时间	%P	显示 AM/PM 标志
%c	按本国习惯显示日期/时间	<digit>	显示小数形式的秒值

表 4.4.5 Scan From String 节点 format string 格式定义

格 式	含 义
%f	搜索十进制的浮点数, 如: 12.3, 12.3e1
%d	搜索十进制的整数, 如: 1234
%x	搜索十六进制的整数, 如: 6AF
%o	搜索八进制的整数, 如: 701
%s	搜索二进制的整数, 如: 10011
%[A-Za-z]	搜索一个字符串
%[^A-Za-z]	搜索一个由英文字母组成的字符串
abcd	搜索一个由非英文字母组成的字符串
%%	搜索百分号%
%, :%?:	设定十进制浮点数的小数点的形式

表 4.4.6 Scan From String 节点的 format string 格式的应用实例

input string 端口输入	format string 端口输入	default(s) 端口输入	output(s) 端口输出	remaining string 端口输出
abc, xyz 12.3+56i 7200	%3s, %s%f%2d		abc	00
			xyz	
		0.00 +0.00 i	12.30 +56.00 i	
			72	
Q+1.27E-3 tail	Q%f t		1.27E-3	ail
0123456789	%3d%3d		12.00	6789
			345	
X:9.860 Z:3.450	X:%fY:%f	100 (I32)	10	Z: 3450
		100.00 (DBL)	100.00	
set49.4.2	set%d		49	.4.2
color: red	color: %s	blue (enum {red, green, blue})	red	
abcd012xyz3	%[a-z]%d%[a-z]%d		abcd	
			12	
			xyz	
			3	
welcome to LabVIEW, John Smith	%[^,]%s		welcome to LabVIEW	Smith
			John	

表 4.4.7 Format Into String 节点 format string 格式的应用实例

Format String 端口输入	Input 端口输入	Resulting String 端口输出
Score=%2d%%	87	Score=87%
Level=ln-7.2eV	0.03642	Level=3.642e-2V
Name:%s, %s	Smith John	Name=Smith, John
Temp:%05.1f%s	96.793 Fahrenheit	Temp:096.8Fahrenheit
String:%10.5s	Hello, world	____Hello

4.4.3 字符串/数字转换

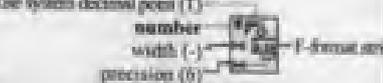
LabVIEW 与仪器通信的过程中,经常需要进行字符串与相对应的数字之间的转换,例如,将从仪器输出缓存中读出的字符串(ASCII 码字符)“1024”转换为相对应的数字 1024,或将数字 10 转换为相对应的字符串“10”,然后发送给仪器。String 子模板中的 String/Number Conversion 子模板提供了 12 个字符串/数字转换节点,如图 4.4.4 所示。



图 4.4.4 String/Number Conversion 子模板

利用这些节点,可以实现字符串和数字之间的相互转换,这些节点的功能强大,但其使用却十分简单,其用法请见表 4.4.8。

表 4.4.8 类型转换节点用法表

节点名称	图标及端口	功能	备注
Number To Decimal String		将由 number 端口输入的数字转换为十进制整数字符串,字符串的宽度由 width 端口指定。节点使用实例请见表 4.4.6	若 number 端口输入的数字是一个浮点数,则节点首先将这个数字转换为一个 32 位的整数后,然后再将这个整数转换为字符串。节点使用实例请见表 4.4.9
Number To Hexadecimal String		将由 number 端口输入的数字转换为十六进制整数字符串,字符串的宽度由 width 端口指定	数字 A~F 在转换后输出的字符串中始终为大写;若 number 端口输入的数字是一个浮点数,则节点首先将这个数字转换为一个 32 位的整数后,然后再将这个整数转换为字符串。节点使用实例请见表 4.4.10
Number To Octal String		将由 number 端口输入的数字转换为八进制整数字符串,字符串的宽度由 width 端口指定	若 number 端口输入的数字是一个浮点数,则节点首先将这个数字转换为一个 32 位的整数后,然后再将这个整数转换为字符串。节点使用实例请见表 4.4.11
Number To Fractional String		将由 number 端口输入的数字转换为小数格式(F-format)的浮点数字符串,字符串的宽度由 width 端口指定	如果输入到 number 端口的数字为无穷大,或者不是一个数字,则 F-format string 端口输出是 inf, -inf 或 NaN。节点使用实例请见表 4.4.12

续表

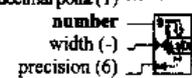
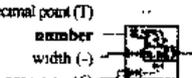
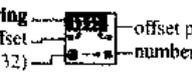
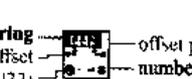
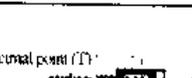
节点名称	图标及端口	功能	备注
Number To Exponential String		将由 number 端口输入的数字转换为指数格式 (E-format) 的浮点数字符串, 字符串的宽度由 width 端口指定	节点使用实例请见表 4.4.13
Number To Engineering String		将由 number 端口输入的数字转换为工程格式 (engineering format) 的浮点数字符串, 字符串的宽度由 width 端口指定	工程格式与指数格式基本相同, 但其指数为 3 的倍数, 例如 -3, 0, 3, 6... 节点使用实例请见表 4.4.14
Format Value		将由 number 端口输入的数字按照 format string 端口指定的格式转换为一个字符串	
Decimal String To Number		将 string 端口输入的字符串中的数字字符转换为十进制整数, 转换起始位置由 offset 端口指定	字符串中的字符“0~9”作为十进制整数数字。节点使用实例请见表 4.4.15
Hexadecimal String To Number		将 string 端口输入的字符串中的数字字符转换为十六进制整数, 转换起始位置由 offset 端口指定	字符串中的字符“0~9”、“A~F”及“a~f”作为十六进制整数数字。节点使用实例请见表 4.4.16
Octal String To Number		将 string 端口输入的字符串中的数字字符转换为八进制整数, 转换起始位置由 offset 端口指定	字符串中的字符“0~7”作为八进制整数数字。节点使用实例请见表 4.4.17
Fract/Exp String To Number		将 string 端口输入的字符串中的数字字符转换为浮点数, 转换起始位置由 offset 端口指定	字符串中的字符“0~9”、“+”、“-”、“e”、“E”以及“.”作为浮点数。节点使用实例请见表 4.4.18
Scan Value		将从 string 端口输入的字符串转换为数字, 转换起始位置为字符串的开始处, 转换后的数字类型由 default 端口输入的数字确定, 数据格式由 format string 端口指定, value 端口返回转换后的数字, output string 返回字符串中剩余的部分字符	

表 4.4.9 Number To Decimal String 节点 Width 端口使用实例

Number 端口输入	Width 端口输入	Decimal integer string 端口输出	备注
9	1	9	
5.6	2	_6	将浮点数四舍五入为整数
8.0	4	___8	如果 Width 端口所指定的输出字符串的长度大于所需要的长度, 则在转换后的字符串的左侧添加相应个数的空格

续表

Number 端口输入	Width 端口输入	Decimal integer string 端口输出	备注
-1024	3	-1024	如果 Width 端口所指定的输出字符串的长度小于所需要的长度, 则在按照实际所需要的长度输出转换后字符串

注: 上表中的下划线“_”表示空格。

表 4.4.10 Number To Hexadecimal String 节点使用实例

Number 端口输入	Width 端口输入	hex integer string 端口输出	备注
3	4	0003	如果 Width 端口所指定的输出字符串的长度大于所需要的长度, 则在转换后的字符串的左侧添加相应个数的字符“0”
42	3	02A	
-4.2	3	FFFFFFFC	首先将 -4.2 四舍五入为 32 位整数 -4, 然后再将这个整数转换为十六进制整数字符串“FFFFFFFC”, 但 Width 端口所指定的输出字符串的长度 3 小于所需要的长度 8, 则在按照实际所需要的长度输出转换后字符串“FFFFFFFC”

表 4.4.11 Number To Octal String 节点使用实例

Number 端口输入	Width 端口输入	octal integer string 端口输出	备注
3	4	0003	如果 width 端口所指定的输出字符串的长度大于所需要的长度, 则在转换后的字符串的左侧添加相应个数的字符“0”
42	3	052	
-4.2	3	3777777774	首先将 -4.2 四舍五入为 32 位整数 -4, 然后再将这个整数转换为十六进制整数字符串“3777777774”, 但 Width 端口所指定的输出字符串的长度 3 小于所需要的长度 11, 则在按照实际所需要的长度输出转换后字符串“3777777774”

表 4.4.12 Number To Fractional String 节点使用实例

Number 端口输入	Width 端口输入	Precision 端口输入	F-format string 端口输出	备注
4.911	6	2	__4.91	将从 number 端口输入的数字按照 Precision 端口指定的精度转换后, 再利用空格在输出字符串的左侧补足长度不够的部分
.003926	8	4	__0.0039	同上
-287.3	5	0	__287	同上

注: 上表中的下划线“_”表示空格。

表 4.4.13 Number To Exponential String 节点使用实例

Number 端口输入	Width 端口输入	Precision 端口输入	E-format string 端口输出	备注
4.911	5	2	4.91e0	将从 Number 端口输入的数字按照 Precision 端口指定的精度转换后输出, 但 Width 端口所指定的输出字符串的长度 5 小于所需要的长度 6, 则在按照实际所需要的长度输出转换后字符串“4.91e0”
.003926	10	2	___3.93e-3	将从 Number 端口输入的数字按照 Precision 端口指定的精度转换后, 再利用空格在输出字符串的左侧补足长度不够的部分
216.01	5	0	__2e2	同上

注: 上表中的下划线“_”表示空格。

表 4.4.14 Number To Engineering String 节点使用实例

Number 端口输入	Width 端口输入	Precision 端口输入	Engineering string 端口输出	备 注
4.93	10	2	____4.93e0	将从 Number 端口输入的数字按照 Precision 端口指定的精度转换后, 再利用空格在输出字符串的左侧补足长度不够的部分
.49	10	2	____490e-3	同上
61.96	8	1	__62.0e0	同上
1789.32	8	2	__1.79e3	同上

注: 上表中的下划线“_”表示空格。

表 4.4.15 Decimal String To Number 节点使用实例

String 端口输入	Offset 端口输入	Default 端口输入	offset past number 端口输出	Number 端口输出	备 注
13ax	0	0	2	13	
-4.8bede conversion	0	0	2	-4	该节点的功能是转换整数, 由于已经转换了一个整数 -4, 所以在第 3 个字符“.”处停止转换, 最终输出转换结果 -4
a49b	0	-9	0	-9	若字符串的 Offset 处没有数字, 则输出由 default 端口所指定的默认值

表 4.4.16 Hexadecimal String To Number 节点使用实例

String 端口输入	Offset 端口输入	Default 端口输入	offset past number 端口输出	Number 端口输出	备 注
f3g	0	0	2	243	字符“g”不是一个有效的十六进制字符, 转换在该字符处停止
-30	0	0	0	0	十六进制数字不支持负数

表 4.4.17 Octal String To Number 节点使用实例

String 端口输入	Offset 端口输入	Default 端口输入	offset past number 端口输出	Number 端口输出	备 注
92	0	0	0	0	字符“9”不是一个有效的八进制字符
071a	0	0	3	57	字符“a”不是一个有效的八进制字符, 转换在该字符处停止

表 4.4.18 Fract/Exp String To Number 节点使用实例

String 端口输入	Offset 端口输入	Default 端口输入	offset past number 端口输出	Number 端口输出	备 注
-4.7e-3x	0	0	7	-0.0047	字符“x”不是一个有效的浮点数字符, 转换在该字符处停止
+5.3.2	0	0	4	5.3	第 2 个小数点“.”无效, 转换在该字符处停止

4.4.4 字符串/数组/路径转换

字符串/数组/路径转换节点主要用于处理字符串与路径之间以及字符串与字符相对应

的 ASCII 码编号数组之间的转换，字符串/数组/路径转换节点位于 String/Array/Path Conversion 子模板中，如图 4.4.5 所示。



图 4.4.5 String/Array/Path Conversion 子模板

字符串/数组/路径转换节点的使用十分简单，其用法请见表 4.4.19。

表 4.4.19 字符串/数组/路径转换节点用法表

节点名称	图标及端口	功能	备注
Path to Array of Strings		将从 path 端口输入的文件路径转换为一个字符串数组，并同时通过 relative 端口返回这个路径是否是一个相对路径。路径中的每一层对应一个字符串数组中的元素，其转换顺序从路径的最高层开始	例如转换路径“C:/WINDOWS”得到的字符串数组为 
Array of Strings to Path		将从 array of strings 端口输入的字符串数组转换为一个相对或绝对的路径。若数组中由一个元素为空字符串，则转换从这个元素后面的元素开始	
Path To String		将从 path 端口输入的文件路径转换为一个字符串	例如转换路径“C:/WINDOWS/winhelp.exe”得到的字符串为：“C:/WINDOWS/winhelp.exe”
String To Path		将从 string 端口输入的字符串转换为文件路径	
String To Byte Array		将从 string 端口输入的字符串转换为一个无符号 8 位整数数组。数组中的每一个元素是字符串中相应位置的字符所在 ASCII 码表中的编号	
Byte Array To String		将从 unsigned byte array 端口输入的无符号 8 位整数数组中每一个元素转换为所对应的 ASCII 字符，并将这些字符连接成一个字符串后输出	

4.4.5 附加字符串运算

除了上述的各种字符串运算节点之外, LabVIEW 还提供了一些附加字符串运算节点, 利用这些节点可以完成一些较为复杂的字符串运算。例如, 将字符串分成两段, 将字符串中的字符按照从后到前的顺序倒置, 更加复杂的搜索等运算。附加字符串运算节点位于 Additional String Functions 子模板中, 如图 4.4.6 所示。

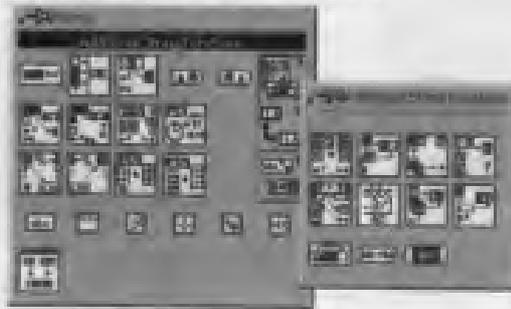


图 4.4.6 Additional String Functions 子模板

附加字符串运算节点的用法见表 4.4.20 所示。

表 4.4.20 附加字符串运算节点用法表

节点名称	图标及端口	功能	备注
Search/Split String		把输入的字符串从特定的位置分离成两个子字符串。分离位置由 Search String/char 端口与 offset 端口共同决定	分离得到的两个子字符串分别从 substring before match 端口和 match + rest of string 端口输出。若节点没有搜索到由 Search String/char 端口指定的字符串, 则 offset of match 端口返回-1, substring before match 端口返回整个字符串, match + rest of string 端口返回一个空字符串
Pick Line		从 multi-line string 端口中输入的多行字符串中提取一指定行, 并把该行加到字符串 string 后, 组成一个新的字符串输出。指定行的位置由 line index 端口确定	
Match First String		将从 string 端口输入的搜索字符串与从 string array 端口输入的字符串数组中的每一个元素进行逐一比较, 若发现搜索字符串与数组中的某一个元素相匹配, 则从 index 端口返回该元素的索引值。若没有发现相匹配的元素, 则返回-1, 并从 output string 端口返回这个搜索字符串	

续表

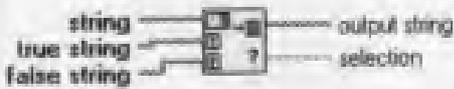
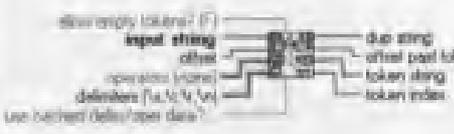
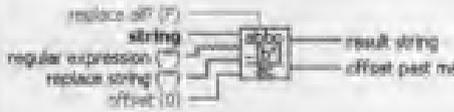
节点名称	图标及端口	功能	备注
Match True/False String		检查从 string 端口输入的字符串,看这个字符串的起始处是否与端口或 false string 端口输入的字符串相匹配,并从 selection 端口返回检查结果	若与 true string 相匹配,则返回 True, 若与 false string 相匹配,则返回 False, 若与这两个字符串都不匹配,也返回 False
Scan String For Tokens		在从 input string 端口输入的字符串中搜索特征字符串,并将特征字符串之间的子字符串通过 token string 端口输出,特征字符串一般是关键字、数字或运算符,由 operators 端口和 delimiters 端口指定,搜索的起始位置由 offset 端口指定	节点各端口的用法较为特殊,见表 4.4.21。 Scan String For Tokens 节点的应用实例请见例 8.5.1
Search and Replace Pattern		在从 string 端口输入的字符串中搜索与从 regular expression 端口输入的正则表达式相匹配的子字符串,并利用 replace string 端口输入的字符串替换搜索到子字符串	该节点的功能与 Match Pattern 节点的功能类似
Index String Array		从 string array 端口输入的字符串数组中取出一个指定的元素,并将其与 string 端口输入的字符串合并成一个新的字符串,元素的位置由 index 端口指定	
Append True/False String		根据 selector 端口的输入,将字符串“FALSE”或“TURE”添加到 string 端口输入的字符串中	
Rotate String		将从 string 端口输入的字符串中的第一个字符放到该字符串的最末尾,其他所有的字符前移一位	例如,输入字符串为“abcdefg”,则输出字符串为“bdefgca”
Reverse String		将从 string 端口输入的字符串中的字符按照从后至前的倒序顺序输出	
Trim Whitespace		删除从 string 端口输入的字符串头部或尾部的空格、回车符,和换行符,具体的删除位置由 location 端口指定	location 端口输入值的含义: 0: 从头部和尾部删除 1: 仅删除头部 2: 仅删除尾部

表 4.4.21 Scan String For Tokens 节点各个端口用法表

端口名称	端口类型 / 数据类型	使用说明	备注
allow empty tokens?	输入端口 / 布尔型	若该端口的输入为 True, 则当输入字符串中有两个相接的分隔符时, token string 端口返回一个空值; 若该端口的输入为 False, 则当输入字符串中有两个或多个相接的分隔符时, 节点将这些分隔符当作一个分隔符使用。其默认值为 False	
input string	输入端口 / 字符串	输入字符串, 被搜索的对象	
offset	输入端口 / 整数	搜索起始位置, 默认值为 0	
operators	输入端口 / 字符串数组	列举特征字符串。如果输入字符串中有一部分子字符串与 operators 中的一个元素相匹配, 则这个元素就是一个特征字符串	Operators 端口输入的字符串除了具体的字符或字符串外, 可以包含一些特殊的通配符, 如表 4.4.22 所示。如 input string="input 10V DC", 而 operators 中含有通配符 "%d", 则字符串 "10" 被认为是一个特征字符串
delimiters	输入端口 / 字符串数组	提供些分隔符。在输入字符串中的两个分隔符之间的子字符串就是一个特征字符串	delimiters 的默认值为 %s、%t、%r、%n (空格、制表符、换行、回车), 也可以自己定义任意字符为分隔符。delimiters 与 operators 是同时起作用的
use cached delim/oper data?	输入端口 / 布尔型	当该端口输入为 True 时, 节点将用一个缓冲区来暂时保存 delimiters 端口及 operators 端口中输入的内容, 所以在一次分析的过程中, 如果 delimiters 或 operators 中的内容有了改变, 并不会立即反映到正在进行的分析过程。但是, 在一次分析过程的开始一定要注意必须保证这个输入为 False, 否则它将仍然使用缓冲区里的 delimiters 及 operators, 这样, 输出结果就有可能有是错的	
dup string	输出端口 / 字符串	将输入字符串原样输出	
offset past token	输出端口 / 整数	输出上一个特征子字符串结束的位置, 即搜索下一个特征子字符串的起始位置。如果 offset 端口输入的数值小于 0 或者大于输入字符串的长度, 或者已经搜索到输入字符串的末尾, 该端口返回 -1	
token string	输出端口 / 字符串	返回搜索到的特征字符串, 这个特征字符串可能是 operators 端口所输入的数组中的一个元素, 也可能是输入字符串中两个分隔符之间的子字符串	
token index	输出端口 / 整数	如果找到一个特征子字符串, 且这个子字符串是 operators 中列举的, token index 端口将输出该子字符串在 operators 数组中的序号, 依此可以用来判断这个特征字符串是什么; 如果找到一个特征子字符串但该字符串并不是在 operators 列举的, 而是处在 delimiters 中的分隔符之间的, 则 token index 输出 -1; 如果已搜索到输入子字符串尾, 则 token index 输出 -2。从这几种情况可以知道, 如果 token index 的输出不是 -2 的话, 则一定会有一个特征子字符串输出	

表 4.4.22 Scan String For Tokens 节点的 Operators 端口中可用的通配符表

格式	含义
%d	十进制整数
%o	八进制整数
%x	十六进制整数
%b	二进制整数
%e,%f,%g	实数
%s	%字符

例 8.5.1 Scan String For Tokens 节点使用实例

搜索字符串 “This is a testLabVIEWstring. DC=10V”，特征字符串为：“LabVIEW”、“%d”、“=”和“.”。VI 的前面板和框图程序如图 4.4.7 所示。

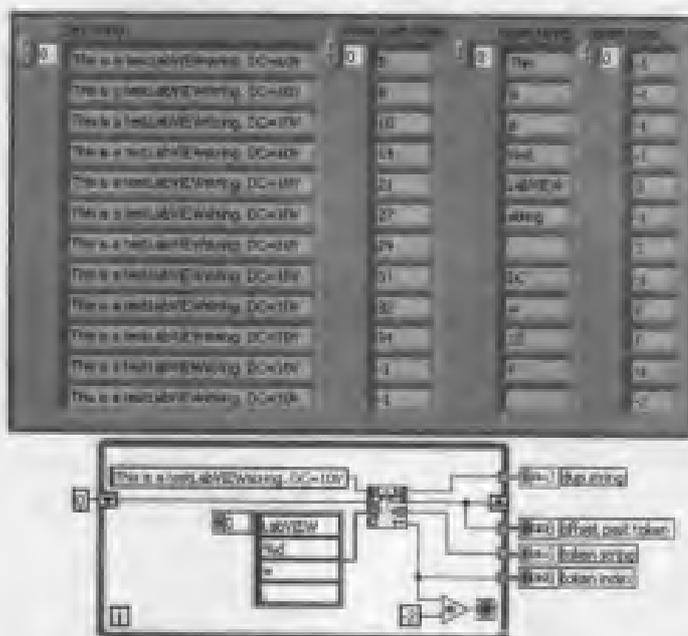


图 4.4.7 例 8.5.1 的前面板和框图程序

4.5 比较运算

比较运算也就是通常所说的关系运算，比较运算节点包含在 Functions 模板→All Functions 子模板→Comparison 子模板中，如图 4.5.1 所示。

在 LabVIEW 中，可以进行以下几种类型的比较：数字值的比较、布尔值的比较、字符串的比较及簇的比较。

(1) 数字值的比较

比较节点在比较两个数字值时，会先将其转换为同一类型的数字。当一个数字值和一个非数字 (NaN) 相比较时，比较节点将返回一个表示二者不相等的值。

(2) 布尔值的比较

两个布尔值相比较时，True 值比 False 值大。

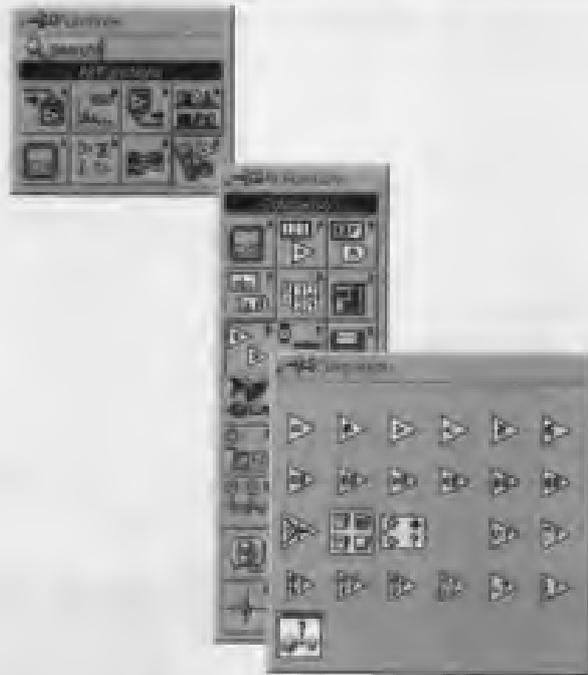


图 4.5.1 Comparison 子模板

(3) 字符串的比较

字符串的比较是按照字符在 ASCII 表中的等价数字值进行比较的, 例如字符 a (在 ASCII 表中的值为 97) 大于字符 A (在 ASCII 表中的值为 65); 字符 A 大于字符 0 (48)。当两个字符串进行比较时, 比较节点会从这两个字符串的第一个字符开始逐个比较, 直至有两个字符不相等为止, 并按照这两个字符输出比较结果。例如, 比较字符串 abcd 和字符串 abef, 比较会在 e 停止, 而字符 c 小于字符 e, 所以字符串 abcd 大于字符串 abef。当一个字符串中存在某一个字符, 而在另一个字符串中这个字符不存在时, 前一个字符串大。例如, 字符串 abcd 大于字符串 abc。

(4) 簇的比较

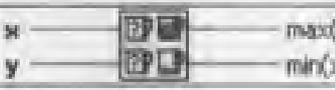
簇的比较与字符串的比较类似, 比较时从簇的第 0 个元素开始, 直至有一个元素不相等为止。簇中元素的个数必须相同, 元素的数据类型和顺序也必须相同。

对于数组和簇而言, 某些比较节点有两种比较模式: Compare Aggregates 和 Compare Elements。在节点图标的右键弹出菜单中可设定这两种模式。当比较节点的模式为 Compare Aggregates 时, 比较节点会返回单个值; 当比较节点的模式为 Compare Elements 时, 比较节点会逐个比较数组或簇中的元素, 并返回一个布尔数组或簇。比较节点的用法见表 4.5.1。

表 4.5.1 比较节点用法表

节点名称	图标及端口	功能	备注
Equal?		$x = y?$	
Not Equal?		$x \neq y?$	
Greater?		$x > y?$	
Less?		$x < y?$	

续表

节点名称	图标及端口	功能	备注
Greater Or Equal?		$x \geq y?$	$x \geq y?$
Less Or Equal?		$x \leq y?$	$x \leq y?$
Equal To 0?		$x = 0?$	$x = 0?$
Not Equal To 0?		$x \neq 0?$	$x \neq 0?$
Greater Than 0?		$x > 0?$	$x > 0?$
Less Than 0?		$x < 0?$	$x < 0?$
Greater Or Equal To 0?		$x \geq 0?$	$x \geq 0?$
Less Or Equal To 0?		$x \leq 0?$	$x \leq 0?$
Select		$x? t:f$	$x?t:f$ 选择输出, 当输入 s 值为 True 时, 节点将输入端口 t 中的内容送至输出端口; 当输入 s 值为 False 时, 节点将输入端口 f 中的内容送至输出端口
Max & Min		$\max(x,y)$ $\min(x,y)$	求 $\max(x,y)$ 和 $\min(x,y)$
In Range and Coerce		$\text{coerced}(x)$ In Range?	$\text{lower limit} < x < \text{upper limit?}$ 判断 x 是否位于由输入端口 upper limit 和 lower limit 指定的上限和下限范围内, 若 x 不在指定的范围内, 会将 x 值强制转换到这个范围内。
Not A Number/Path/Refnum?		$\text{NaN?}/\text{path?}/\text{Refnum?}$	判断输入是否是一个 NaN/Path/Refnum 值
Empty String/Path?		empty?	判断输入的字符串或者路径值是否为空
Decimal Digit?		digit?	判断输入的字符是否为 10 进制
Hex Digit?		hex?	判断输入的字符是否为 16 进制
Octal Digit?		octal?	判断输入的字符是否为 8 进制
Printable?		printable ASCII?	判断输入的字符是否为可打印 ASCII 码
White Space?		$\text{space, \n, \t, \r, \f, \c?}$	判断输入的字符是否为 space, \n, \t, \r, \f, \c 等非打印字符
Lexical Class		class number	判断输入的字符是否为 class number

第5章 变量、数组、簇与波形数据

本地变量 (Local Variable) 和全局变量 (Global Variable) 是 LabVIEW 为改善图形化编程灵活性局限而专门设计的两个特殊节点, 主要解决数据和对象在同一 VI 程序中的复用和在不同的 VI 程序中的共享问题。LabVIEW 中的本地变量和全局变量与 C 语言中的局部变量与全局变量在有效作用范围上意义相同, 但是 LabVIEW 中的本地变量与全局变量的定义与使用更加复杂, 稍有不慎, 便容易引起程序隐性逻辑错误, 因此, 这两类变量是学习和掌握 LabVIEW 编程的难点之一。

数组、簇和波形数据是 LabVIEW 中三类比较复杂的数据类型。数组与其他编程语言中的数组概念是相同的, 簇相当于 C 语言中的结构数据类型, 波形数据则是 LabVIEW 为数据采集和处理而提供的一种专门的数据结构。LabVIEW 为上述三种数据类型的创建和使用提供了大量灵活便捷的工具, 使编程效率得到很大提高。可以说数组、簇和波形数据是学习 LabVIEW 编程必须掌握并且能够灵活运用数据类型。

5.1 本地变量

本地变量相当于传统编程语言中的局部变量, 可以在同一个程序内部使用。但由于 LabVIEW 的特殊性, 本地变量与局部变量又有所不同。

在 LabVIEW 中, 前面板上的每一个控制或指示在框图程序上都有一个对应的端口, 控制通过这个端口将数据传送给框图程序的其他节点, 框图程序也可以通过这个端口为指示赋值。但是, 这个端口是惟一的, 一个控制或一个指示只有一个端口。而用户在编程时, 经常需要在同一个 VI 框图程序中的不同位置多次为指示赋值, 或多次从控制中取出数据, 或者是为控制赋值, 从指示中取出数据。显然, 这时仅用一个端口是无法实现这些操作的。这就不同于传统的编程语言, 如定义一个变量 a , 在程序的任何地方需要这个变量时, 写一个 a 就可解决问题。本地变量的引入, 巧妙地解决了这个问题。

5.1.1 本地变量的创建

本地变量的创建有两种方式。第一种方式是在 Functions 模板 → All Functions 子模板 → Structures 子模板中选择 Local Variable, 然后将其图标放在框图程序中, 如图 5.1.1 所示。

注意, 在本地变量的图标上有一个问号, 此时的本地变量没有任何用处, 因为它还没有与前面板上的控制或指示相关联。用操作工具单击图标, 会出现一个下拉选单, 选单列出了前面板上所有控制或指示的名称, 选择所需要的名称, 就完成了前面板对象的一个本地变量的创建。如图 5.1.2 所示。

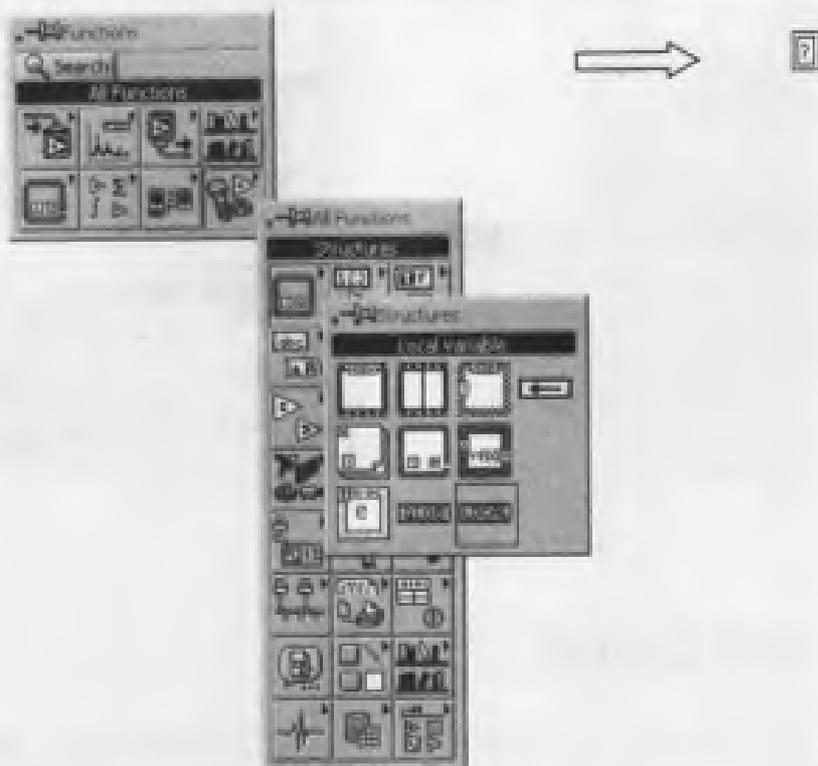


图 5.1.1 本地变量的创建方法一

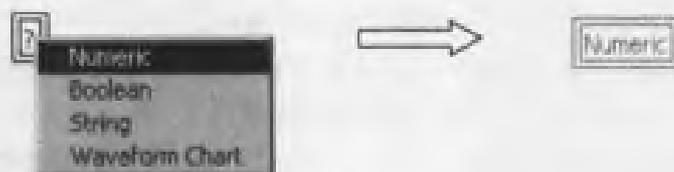


图 5.1.2 选择前面板对象的名称方法一

也可以在图标的右键弹出菜单中选择 **Select Item**，会出现一个与图 5.1.2 同样的下拉菜单，功能完全相同，如图 5.1.3 所示。

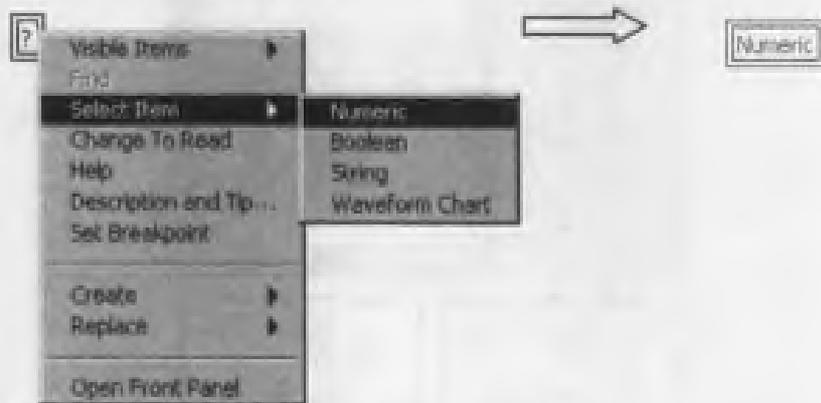


图 5.1.3 选择前面板对象的名称方法二

创建本地变量的第二种方法是在前面板对象以及在框图程序中与之相应端口的右键弹出菜单中选择 **Create**→**Local Variable**，直接创建该前面板对象的本地变量，如图 5.1.4 所示。

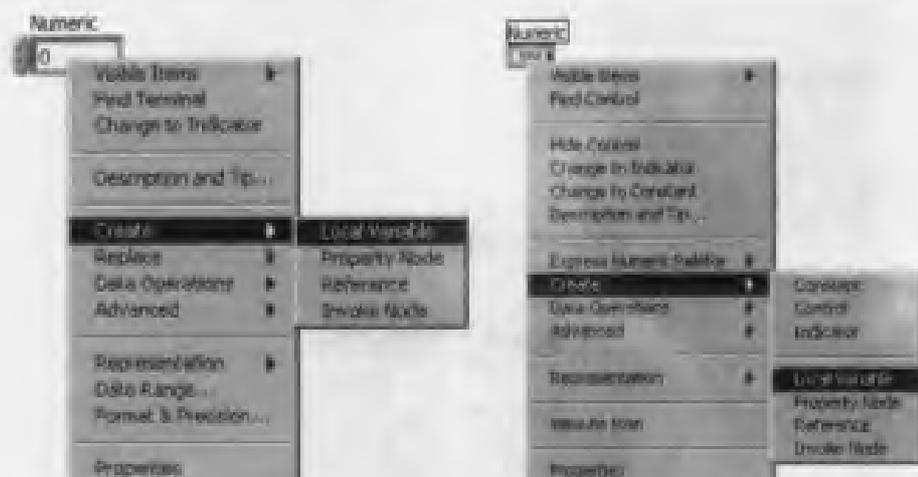


图 5.1.4 本地变量的创建方法二

5.1.2 本地变量的使用

如前所述，使用本地变量可以在框图程序的不同位置访问前面板对象。前面板对象的本地变量相当于其端口的一个拷贝，它的值与该端口同步，也就是说，两者所包含的数据是相同的。

例 5.1.1 用一个布尔开关同时控制两个 While 循环。

本例使用两个 While 循环（详细用法见第 6 章）分别产生两个随机波形。如图 5.1.5 所示，将布尔开关的端口连接到一个 While 循环的条件端口上，将布尔开关的本地变量连接到另一个 While 循环的条件端口上，这样，只用一个布尔开关就可以同时控制两个 While 循环。

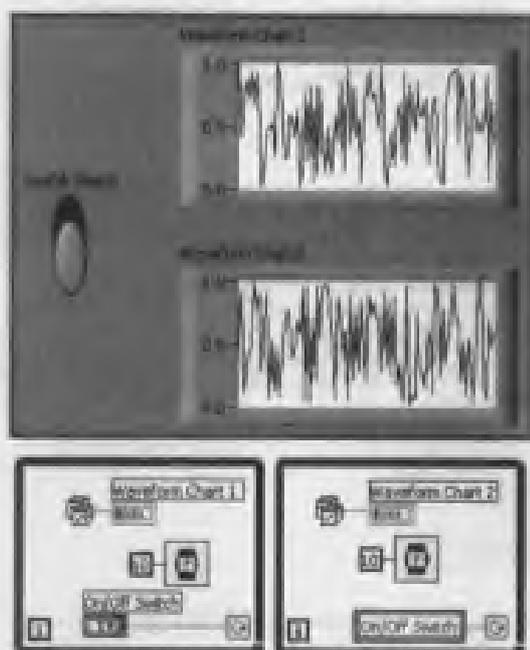


图 5.1.5 例 5.1.1 的前面板及框图程序

用户经常需要为一个控制赋值或从一个指示中读出数据，但通过前面板对象的端口不能为控制赋值，也不能从指示中读出数据，如图 5.1.6 所示。



图 5.1.6 访问前面板对象的错误方法

利用本地变量就可以解决这个问题。本地变量有 Write 和 Read 两种属性：当属性为 Read 时，可以从本地变量中读出数据；当属性为 Write 时，可以给这个本地变量赋值。利用这种方法，就可以给控制赋值或从指示中读出数据。

例 5.1.2 利用本地变量给一个控制赋值，并从一个指示中读出数据，如图 5.1.7 所示。



图 5.1.7 例 5.1.2 的前面板及框图程序

在本地变量的右键弹出选单中选择 Change to Read 或 Change to Write，可改变本地变量的属性，如图 5.1.8 所示。注意，当本地变量的属性为 Read 时，本地变量图标边框用粗线来强调；当本地变量的属性为 Write 时，本地变量图标边框则用细线强调。通过本地变量图标边框线条的粗细，可以很容易地区分出该本地变量的读写属性，为用户编程提供了很大的方便。

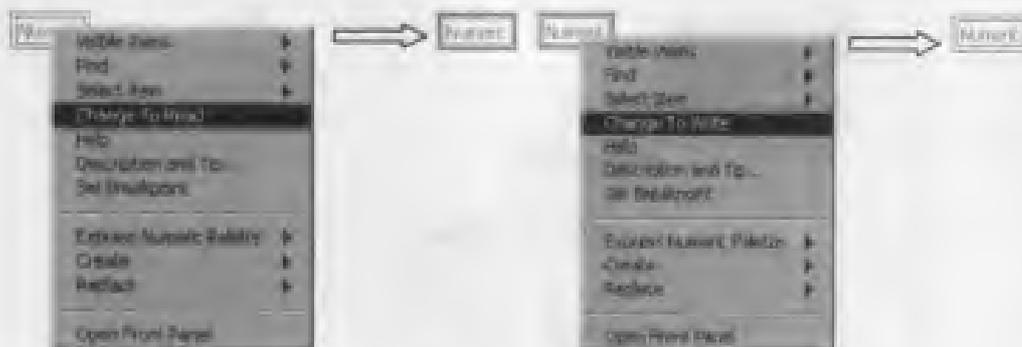


图 5.1.8 改变本地变量的属性

5.1.3 本地变量的特点

本地变量具有许多特点，了解这些特点，有助于用户更加有效地使用 LabVIEW 编程。

一个本地变量就是其相应前面板对象的一个数据拷贝,要占用一定的内存。所以,应该在程序中控制本地变量的数量,特别是对于那些包含大量数据的数组,若在程序中使用多个这种数组的本地变量,那么这些本地变量就会占用大量的内存,从而降低程序运行的效率。

LabVIEW 是一种并行处理语言,只要节点的输入有效,节点就会执行。当程序中有多个本地变量时,就要特别注意这一点,因为这种并行执行可能造成意想不到的错误。例如,在程序的某一处,用户从一个控制的本地变量中读出数据,在另一处,根据需要又为这个控制的另一个本地变量赋值。如果这两过程恰好是并行发生的,这就有可能使读出的数据不是前面板对象原来的数据,而是赋值后的数据。这种错误不是明显的逻辑错误,很难发现,因此,在编程过程中要特别注意,尽量避免这种现象的出现。

本地变量的另外一个特点与传统编程语言中的局部变量相似,就是它只能在同一个 VI 中使用,不能在不同的 VI 之间使用。若需要在不同的 VI 间进行数据传递,则要使用全局变量。

另外,关于本地变量的使用,还有一点需要提醒读者,当布尔控件的 Mechanical Action 属性处于 Latch When Pressed 、Latch When Released 和 Latch Until Released 状态时,不能创建和使用该布尔控件的本地变量,否则,LabVIEW 会报错。

5.2 全局变量

全局变量是 LabVIEW 中的一个对象,通过全局变量,可以在不同的 VI 之间进行数据传递。LabVIEW 中的全局变量与传统编程语言中的全局变量类似,但也有独特之处。

5.2.1 全局变量的创建

全局变量的创建较为复杂,通常按以下步骤进行。

第一步,在 Functions 模板→All Functions 子模板→Structures 子模板中选择 Global Variable,并将其图标放至框图程序中,如图 5.2.1 所示。

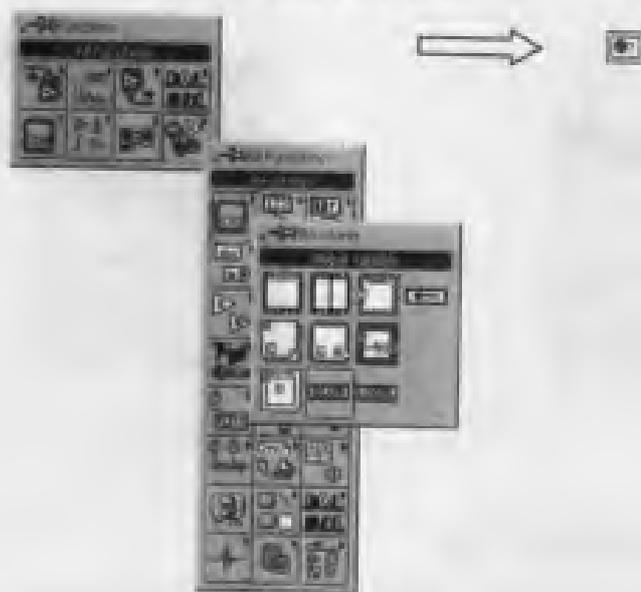


图 5.2.1 全局变量的创建

第二步，双击全局变量图标，打开其前面板，如图 5.2.2 所示。

第三步，在 Controls 模板中选择需要的前面板对象，放入全局变量的前面板中，如图 5.2.3 所示。

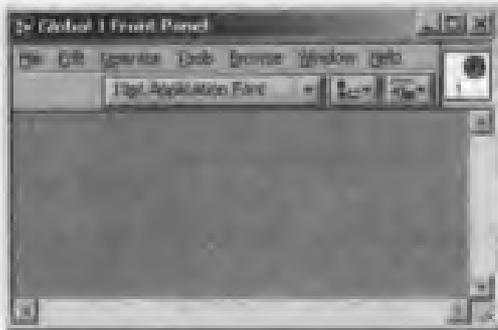


图 5.2.2 全局变量的前面板



图 5.2.3 全局变量的前面板

第四步，在主选单中选择 File→Save，保存这个全局变量。然后关闭全局变量的前面板窗口，至此，就完成了全局变量的创建。

第五步，将鼠标切换至数据数据操作工具状态，单击第一步所创建的全局变量的图标，或在其右键弹出选单中选择 Select Item，弹出的子选单列出了全局变量所包含的所有对象的名称，根据需要选择相应的对象，如图 5.2.4 所示。

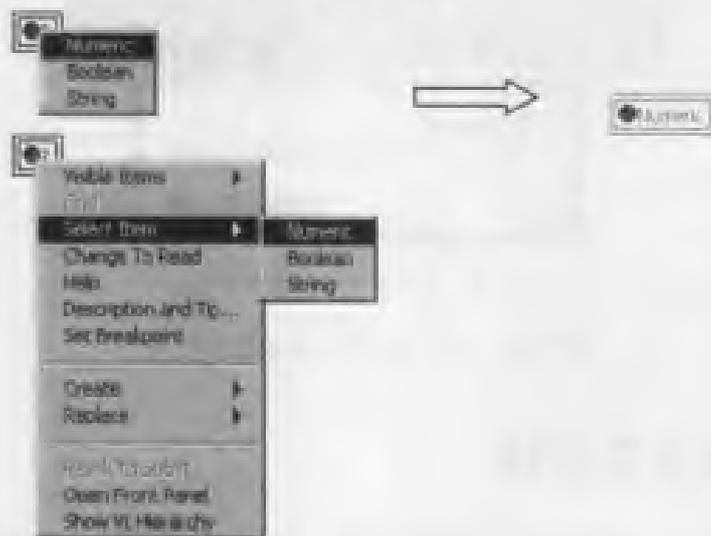


图 5.2.4 选择一个全局变量

5.2.2 全局变量的使用

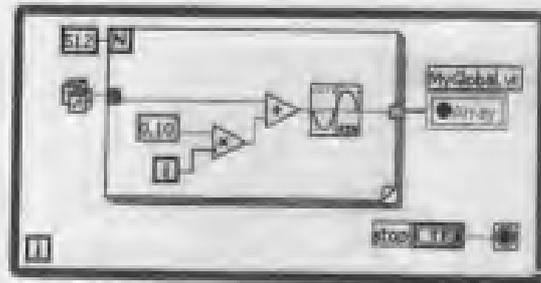
例 5.2.1 利用全局变量在 VI 之间传递数据。

本例创建了 1 个全局变量和 2 个 VI。当 2 个 VI 同时运行时第 1 个 VI 产生 1 个正弦波，送至全局变量中；第二个 VI 从全局变量中将波形数据读出，并送至前面板上的 Waveform Graph 指示中，将正弦波显示出来。全局变量以及 2 个 VI 的框图程序如图 5.2.5 所示。

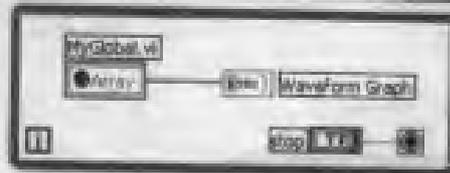
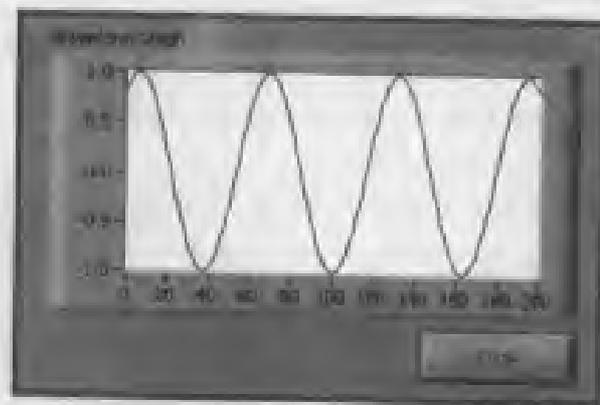
全局变量也有 Read 和 Write 两种属性,其用法和属性切换方法与本地变量相同,在此不再赘述。



(a) 全局变量前面板



(b) 第一个 VI 的框图程序



(c) 第二个 VI 的前面板及框图程序

图 5.2.5 例 5.2.1 的前面板及框图程序

5.2.3 全局变量的特点

LabVIEW 中的全局变量与传统编程语言中的全局变量相比有很大的不同。在传统编程语言中,全局变量只能是一个变量,一种数据类型,而 LabVIEW 中的全局变量比较灵活,它以独立文件的形式存在,并且在一个全局变量中可以包含多个对象,拥有多种数据类型。

全局变量与 SubVI 的不同之处在于它不是一个真正的 VIs,不能进行编程,只能用于简单的数据存储与数据传递。但全局变量的速度是其他大多数数据类型的 10 倍。全局变量的另一个优点是可将所有的 Global 数据放入一个全局变量中,并且在程序执行时分别访问。LabVIEW 全局变量的这些特点,使得全局变量的功能非常强大,并且使用方便,易于管理。

通过全局变量在不同的 VI 之间进行数据交换只是 LabVIEW 中 VI 之间数据交换的方

式之一，通过 DDE（动态数据交换）也可以进行数据交换。

注意，在一般情况下，不能利用全局变量在两个 VI 之间传递实时数据。其原因是，通常情况下两个 VI 对全局变量的读写速度不能保证严格一致。例如，若例 5.2.1 中的第一个 VI 向全局变量中写数据的速度高于第二个 VI 从全局变量中读数据的速度，则会造成部分数据丢失，第二个 VI 不能读出所有的数据；若第一个 VI 向全局变量中写数据的速度低于第二个 VI 从全局变量中读数据的速度，则会造成第二个 VI 从全局变量中重复读出部分的数据。若需要在两个 VI 之间传递实时数据，可以使用 DataSocket 技术（将在第 13 章介绍）或 LabVIEW 同步控制技术。关于 LabVIEW 同步控制技术，有兴趣的读者可以参见清华大学出版社出版的《LabVIEW 高级程序设计》一书。

5.3 数 组

当有一串数据需要处理时，它们很可能是一个数组（Array），大多数的数组是一维数组（1D，列或向量），少数是二维数组（2D，矩阵），极少数是三维或多维数组。在 LabVIEW 中可以创建数字类型的、字符串类型的、布尔型以及其他任何数据类型的数组（数组的数组除外）。数组经常用一个循环来创建，其中 For 循环是最佳的，因为 For 循环的循环次数是预先指定的，在循环开始前它已分配好了内存；而 While 循环却无法做到这一点，这是因为 LabVIEW 无法知道 While 循环将循环多少次。

数组是 LabVIEW 中常用的数据类型之一，与其他编程语言相比，LabVIEW 中的数组更加灵活多变，独具特色。

5.3.1 数组的组成与创建

LabVIEW 是图形化编程语言，因此，LabVIEW 中数组的表现形式与其他语言有所不同。数组由 3 部分组成：数据类型、数据索引（Index）和数据，如图 5.3.1 所示。



图 5.3.1 数组的组成

在数组中，数组元素位于右侧的数组框架中，按照元素索引由小到大的顺序从左至右或从上至下排列，图 5.3.1 仅显示了数组元素由左至右排列时的情形。数组左侧为索引显示（Index Display），其中的索引值是位于数组框架中最左侧或最上侧元素的索引值，这样做是由于数组中能够显示的数组元素个数是有限的，用户通过索引显示可以很容易地访问到数组中的任何一个元素。

数组的创建分两步进行。

第一步，从 Controls 模板 → All Controls 子模板 → Array & Cluster 子模板中创建数组框

架,如图 5.3.2 所示。注意,此时创建的只是一个数组框架,不包含任何内容,对应框图程序中的端口只是一个如图 5.3.2 所示的黑色中空矩形图标。

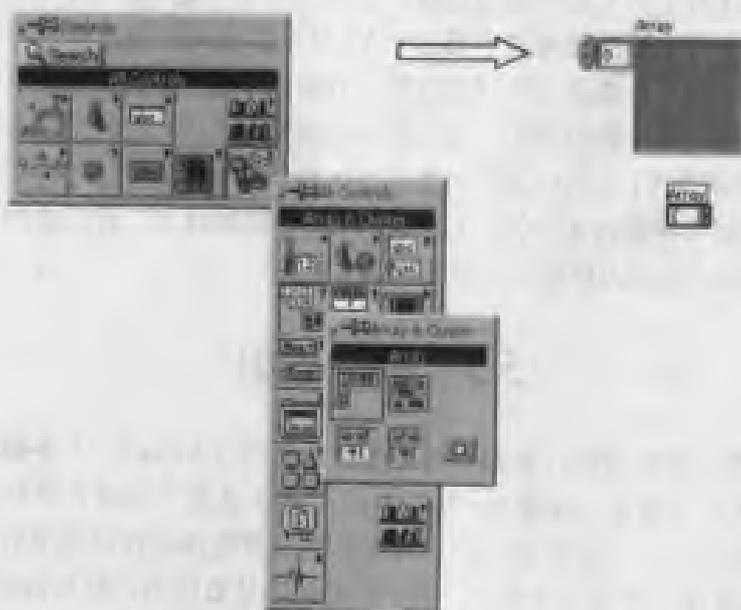


图 5.3.2 数组的创建第 1 步

第二步,根据需要将相应数据类型的前面板对象放入数组框架中。图 5.3.3 所示的是将一个数字量控制放入数组框架,这样就创建了一个数字类型数组(数组的属性为控制)。从图 5.3.3 中可以看出,当数组创建完成之后,数组在框图程序中相应的端口就变为相应颜色和數據类型的图标了。

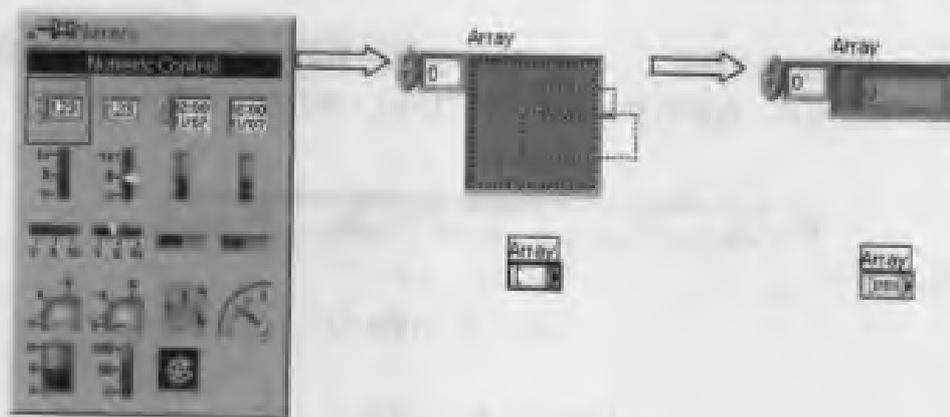


图 5.3.3 数组的创建第 2 步

另外,数组在创建之初都是一维数组,如果要用到二维以上的数组,用鼠标(对象操作工具状态)在数组索引显示边框下边缘的尺寸控制点上向下拖动,或者在数组的右键弹出菜单中选择 Add Dimension 即可添加数组的维数,如图 5.3.4 所示。注意,在框图程序中数组相对应的端口的图标中有一个方括号,方括号线条的粗细与数组的维数成正比,数组的维数越高,方括号线条越粗。

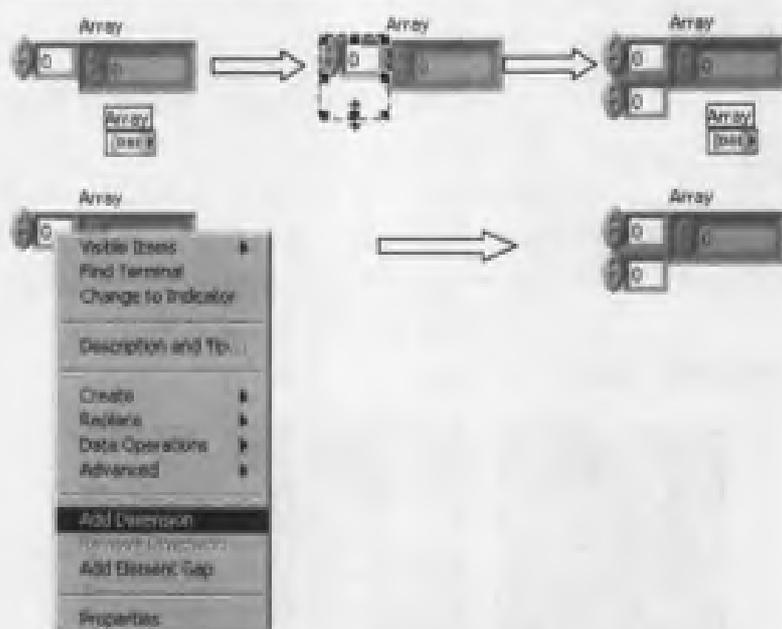


图 5.3.4 添加数组的维数

5.3.2 数组的使用

对一个数组进行操作，无非是求数组的长度、对数据排序、取出数组中的元素、替换数组中的元素或初始化数组等各种运算。传统编程语言主要依靠各种数组函数来实现这些运算，而在 LabVIEW 中，这些函数是以功能函数节点的形式来表现的，本小节将详细介绍 Functions 模板 → All Functions 子模板 → All Functions 子模板 → Array 子模板（见图 5.3.5）中各节点的用法。

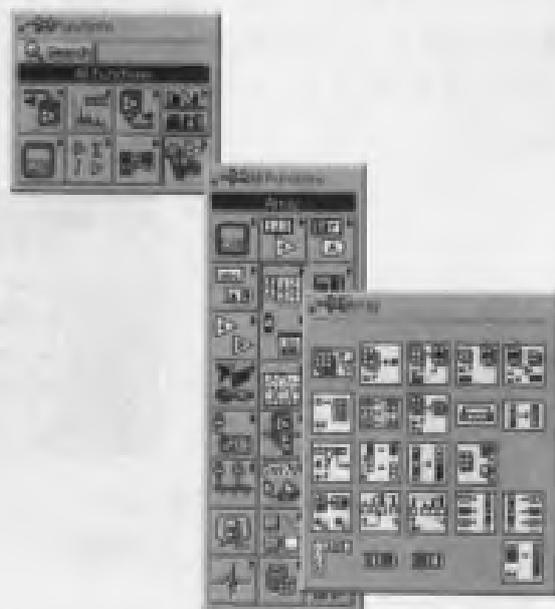


图 5.3.5 Array 子模板

1. Array Size



图 5.3.6 Array Size 节点的图标及其端口定义

返回输入数组的长度, 节点的图标及其端口定义如图 5.3.6 所示。节点的输入为一个 n 维数组, 输出为该数组各维包含元素的个数。当 $n=1$ 时, 节点的输出为一个标量; 当 $n>1$ 时, 节点的输出为一个一维数组, 数组的每一个元素对应输入数组中每一维的长度。

例 5.3.1 求一个一维数组和一个二维数组的长度。

VI 的前面板及框图程序如图 5.3.7 所示。

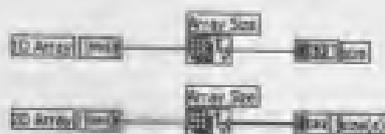


图 5.3.7 例 5.3.1 的前面板及框图程序

2. Index Array

返回输入数组中由输入索引 index 指定的元素, 节点的图标及其端口定义如图 5.3.8 所示。

例 5.3.2 将一个一维数组中的第二个元素取出。

VI 的前面板及框图程序如图 5.3.9 所示, 本例在 C 语言中相当于语句 “Output_Array=1D_Array[2];”。

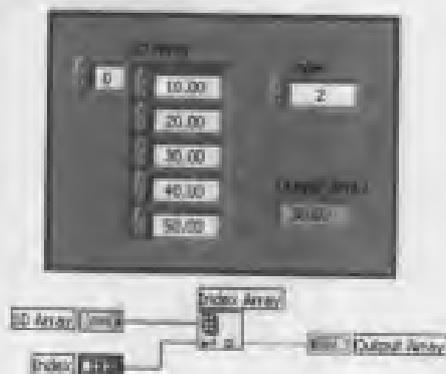


图 5.3.8 Index Array 节点的图标及其端口定义

图 5.3.9 例 5.3.2 的前面板及框图程序

用鼠标 (对象操作工具状态) 向下拖动 Index Array 节点边框下边缘的尺寸控制点, 可

添加多组“Index-Element / Subarray”端口，如图 5.3.10 所示。

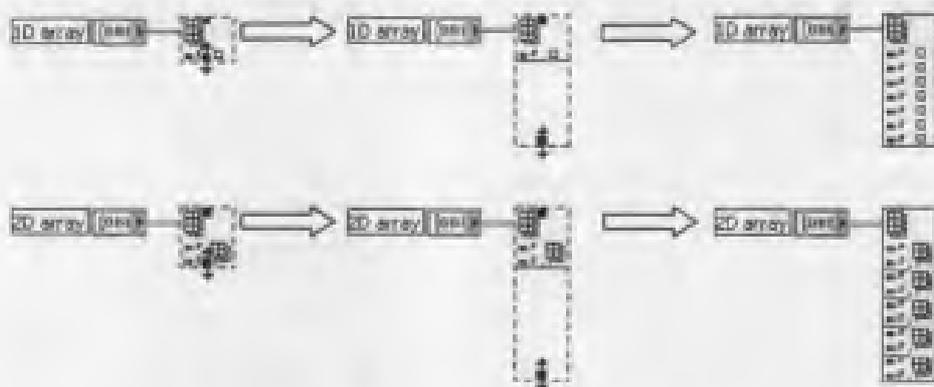


图 5.3.10 添加“Index-Element/Subarray”端口

对于一维数组 1D Array[], 左侧的 Index 端口只有一个，右侧的 Element 端口输出值为一个标量。若 Index 端口没有输入，则 Element 端口按照从上到下的顺序依次输出数组的元素 1D Array[0]、1D Array[1]、1D Array[2]、1D Array[3] …。

对于二维数组 2D Array[], 左侧的 Index 端口有两个，第一个 Index 端口为二维数组“行(Row)”的索引，第二个 Index 端口为二维数组“列(Column)”的索引，右侧的 Subarray 端口输出值为一个一维数组。若 Index 端口没有输入，则 Subarray 端口按照从上到下的顺序依次输出二维数组的第 0 行子数组、第 1 行子数组、第 2 行子数组、第 3 行子数组…。

故可利用 Index Array 节点这种功能实现例 5.3.2, Index Array 节点的第 2 组“Index-Element / Subarray”端口的 Element 端口所输出的就是一维数组的第二个元素 1D Array[2]，如图 5.3.11 所示。

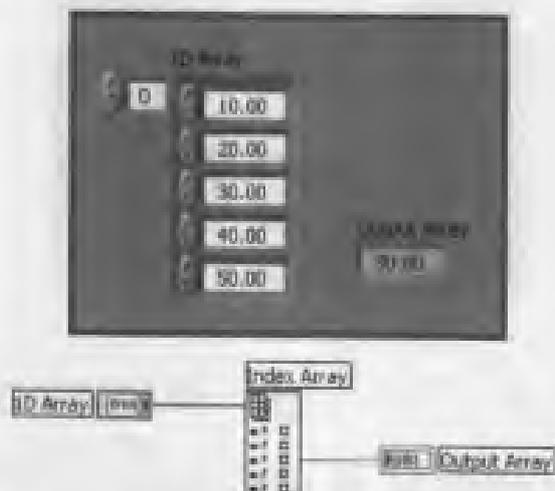


图 5.3.11 利用另一种方法实现例 5.3.2 的 VI 的前面板及框图程序

上述两种方法各有优缺点，第一种方法较为灵活，只要改变 Index 的数值，就可以获得数组中的任意一个元素，但编程较为烦琐；第二种方法编程简单，但不够灵活，程序一旦运行，只能获得预先指定的数组中的元素，不能获得其他的元素。用户可以根据自己的需要选用合适的方法。为进一步说明并比较这两种方法，请看例 5.3.3。

例 5.3.3 取出一个一维数组中所有的元素。

本例利用两种方法实现,框图程序如图 5.3.12 和 5.3.13 所示,结果如图 5.3.14 所示。

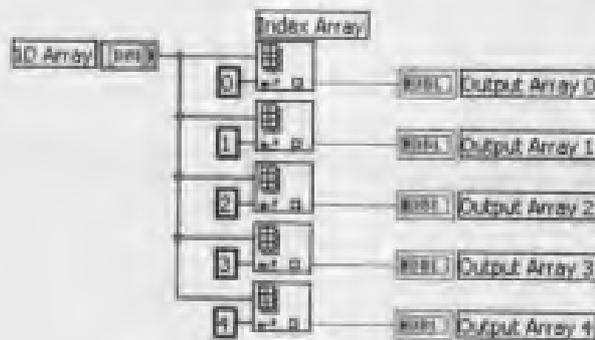


图 5.3.12 例 5.3.3 方法 1 的框图程序

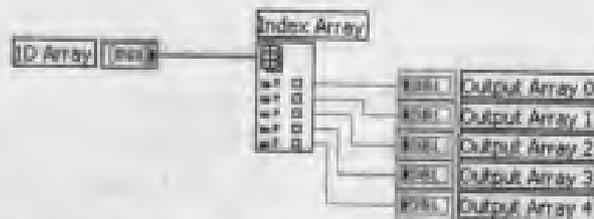


图 5.3.13 例 5.3.3 方法 2 的框图程序



图 5.3.14 例 5.3.3 的结果

从例 5.3.3 可以看出,方法 2 的编程较为简单,建议用户在编写类似功能的 VI 时利用方法 2。

下面以二维数组为例来说明在输入数组是多维数组时,如何使用 Index Array 节点。

例 5.3.4 从二维数组中取出一个元素。

VI 的前面板及框图程序如图 5.3.15 所示。

图 5.3.14 所示的 Index Array 节点有两个 Index 端口,第一个 Index 端口输入的是行索引;第二个 Index 端口输入的是列索引。该例在 C 语言中相当于语句:

```
Output_Array=2D_Array[2][3];
```

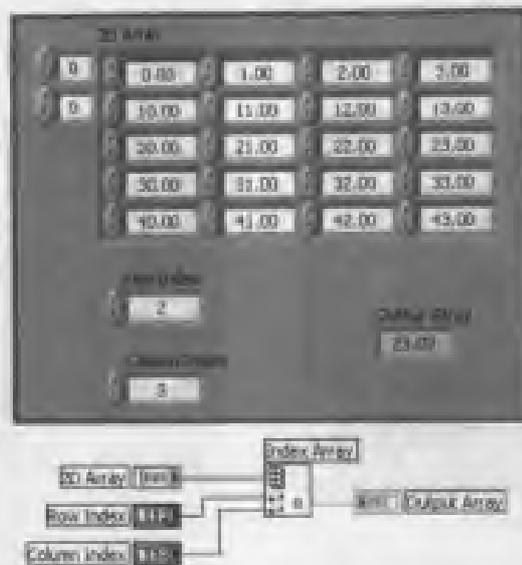


图 5.3.15 例 5.3.4 的前面板及框图程序

注意，在例 5.3.3 中得到的结果仅是二维数组中的一个元素，如果需要得到二维数组中的某一行或某一列中所有的元素又该如何处理？一种方法是用一个循环将元素一个一个取出，但这种方法较为烦琐。最简捷的方法是只为其中一个 Index 端口赋值，就能直接得到数组中的某一行或某一列中所有的元素。

当只为第一个 Index 端口赋值时，节点返回的是输入到第一个 Index 端口的索引值所指定的二维数组中的那一行元素；当只为第二个 Index 端口赋值时，节点返回的是输入到第二个 Index 端口的索引值所指定的二维数组中的那一列元素。

例 5.3.5 从二维数组中取出某一列中所有的元素。

VI 的前面板及框图程序如图 5.3.16 所示。从图中可以看出，程序只为 Index Array 节点的第二个 Index 端口赋值，而没有为第一个 Index 端口赋值。Index Array 节点返回的是一个一维数组，这个一维数组中的元素是由 Column Index 端口所指定的二维数组中的第 2 列元素。

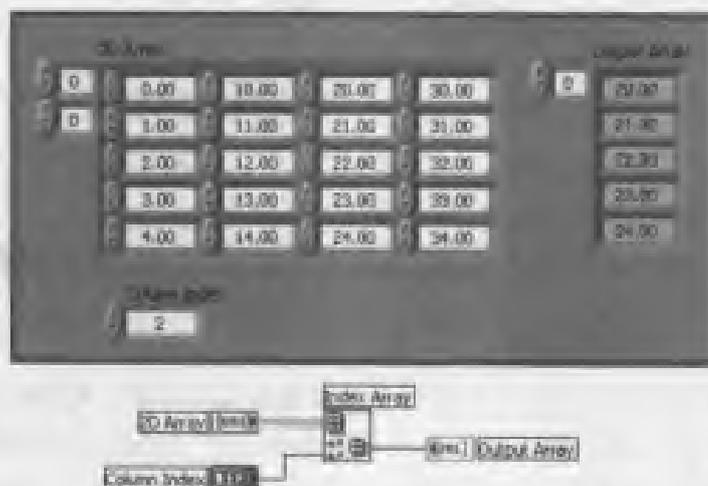


图 5.3.16 例 5.3.5 的前面板及框图程序

例 5.3.6 从二维数组中取出某一行中所有的元素。

该例可由两种方法实现,这两种方法的前面板和框图程序如图 5.3.17 和图 5.3.18 所示。

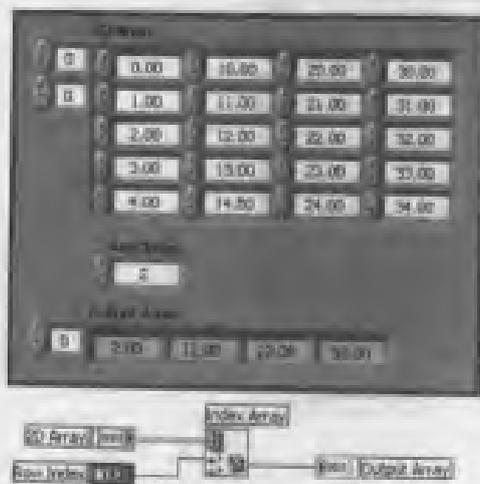


图 5.3.17 例 5.3.6 方法 1 的前面板及框图程序

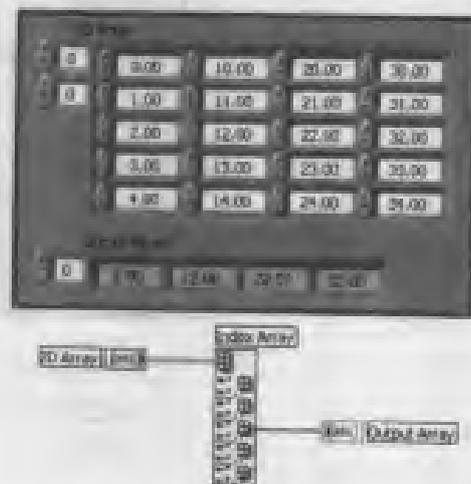


图 5.3.18 例 5.3.6 方法 2 的前面板及框图程序

注意,若需要使用方法 2 取出二维数组中的某一系列元素,需要首先利用一个 Transpose 2D Array 节点将该二维数组转置后,再利用 Index Array 节点将转置后的二维数组的第二行元素取出即可。

3. Replace Array Subset

替换输入数组中的一个元素,节点的图标及其端口定义如图 5.3.19 所示。注意,输入到 Replace Array Subset 节点上 new element/subarray 端口的数据的数据类型必须与输入数组的数据类型一致。



图 5.3.19 Replace Array Subset 节点的图标及其端口定义

例 5.3.7 替换二维数组中的某一个元素。

VI 的前面板及框图程序如图 5.3.20 所示。

利用 Replace Array Subset 节点可以一次性地替换二维数组中整行或整列的元素,其实现方法请见例 5.3.8。

例 5.3.8 替换二维数组中的某一行元素。

VI 的前面板及框图程序如图 5.3.21 所示。从图中可以看出,程序只为 Replace Array Subset 节点的第一个 Index 端口赋值,并没有为第二个 Index 端口赋值,并且在 new element/subarray 端口上连接了一个一维数组。若需要替换二维数组中的某一系列元素,则需要为第二个 Index 端口赋值,而不能为第一个 Index 端口赋值。

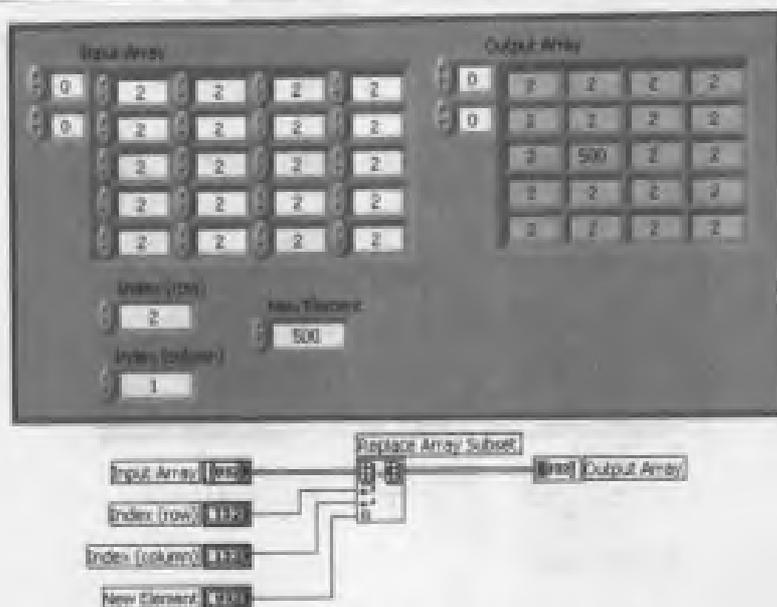


图 5.3.20 例 5.3.7 的前面板及框图程序

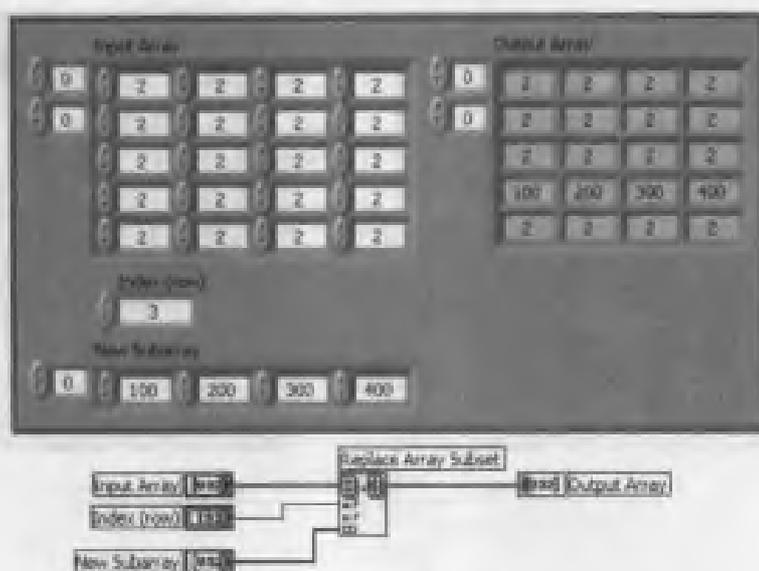


图 5.3.21 例 5.3.8 的前面板及框图程序

4. Insert Into Array

在数组中指定的位置插入元素，节点的图标及其端口定义如图 5.3.22 所示。



图 5.3.22 Insert Into Array 节点的图标及其端口定义

例 5.3.9 在一个一维数组中插入一个元素, 在一个二维数组中插入一行元素。
VI 的前面板及框图程序如图 5.3.23 和图 5.3.24 所示。



图 5.3.23 在一个一维数组中插入一个元素

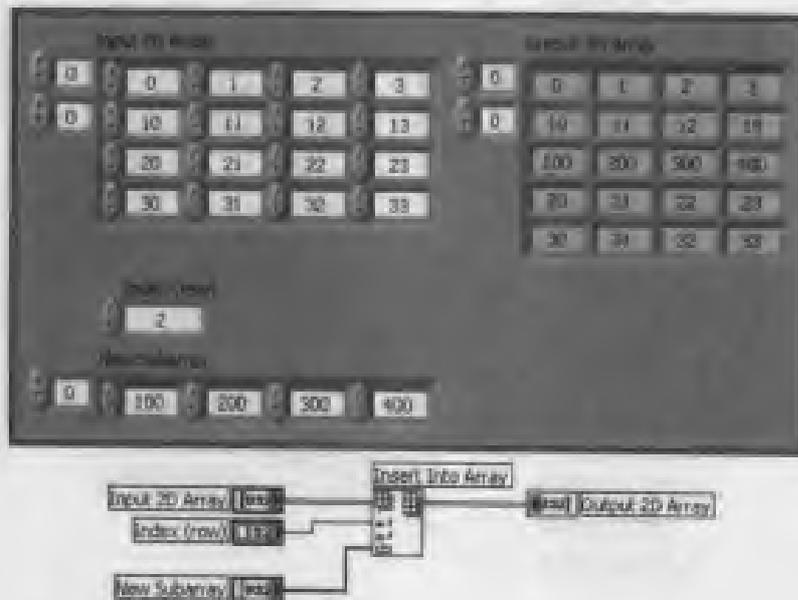


图 5.3.24 在一个二维数组中插入一行元素

从图 5.3.24 可以看出, 只为 Insert Into Array 节点的第一个 Index 端口赋值, 并没有为其第二个 Index 端口赋值, 这样可以在二维数组中的指定行插入一行元素; 若只为 Insert Into Array 节点的第二个 Index 端口赋值, 而没有为其第一个 Index 端口赋值, 这样可以在二维数组中的指定列插入一列元素。

5. Delete From Array

从数组中删除指定数目的元素, 节点的图标及其端口定义如图 5.3.25 所示。Index 端口用于指定所删除元素的起始元素的索引号, Length 端口用于指定删除元素的数目, array

w/ subset deleted 端口返回的是删除元素后的新数组，deleted portion 端口返回的是所删除的元素。

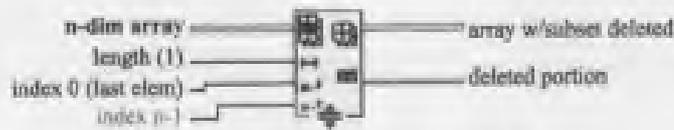


图 5.3.25 Delete From Array 节点的图标及其端口定义

例 5.3.10 在一个一维数组中删除指定数目的元素，在一个二维数组中删除指定行数的元素。

VI 的前面板及框图程序如图 5.3.26 和 5.3.27 图所示。

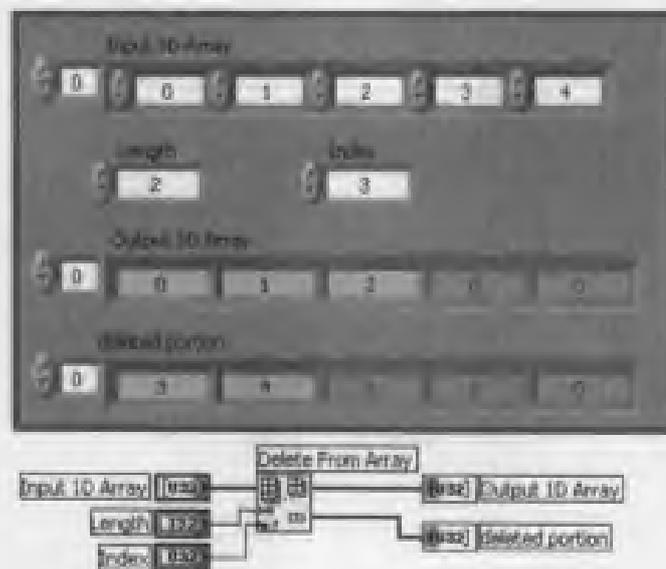


图 5.3.26 在一个一维数组中删除指定数目的元素

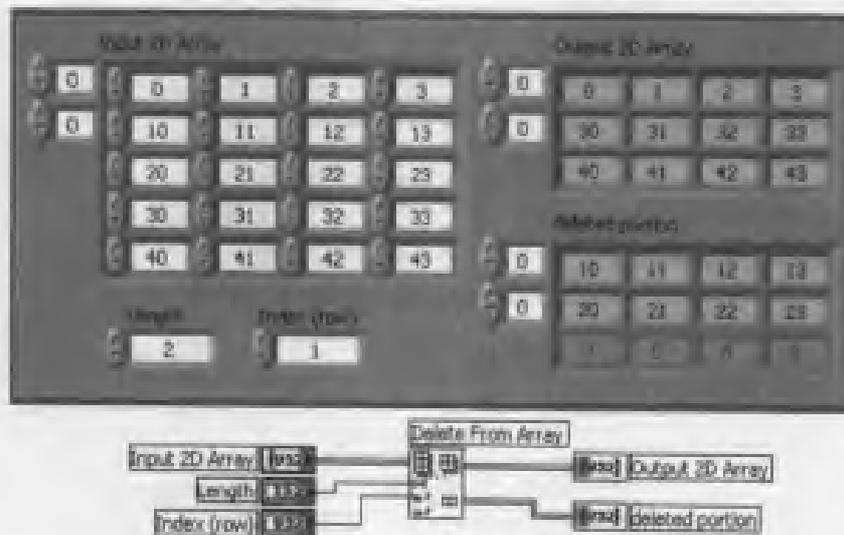


图 5.3.27 在一个二维数组中删除指定行数的元素

6. Initialize Array

在初始化数组时, 节点的图标及其端口定义如图 5.3.28 所示。数组维数由节点左侧 dimension size 端口的个数决定。数组中所有的元素都相同, 均等于输入到 element 端口中的值。

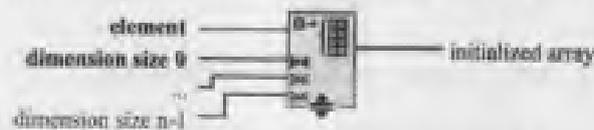


图 5.3.28 Initialize Array 节点的图标及其端口定义

用鼠标(对象操作工具状态)在节点图标下边缘的尺寸控制点上向下拖动, 或在 dimension size 端口的右键弹出菜单中选择 Add Dimension, 可添加 dimension size 端口, 如图 5.3.29 所示。

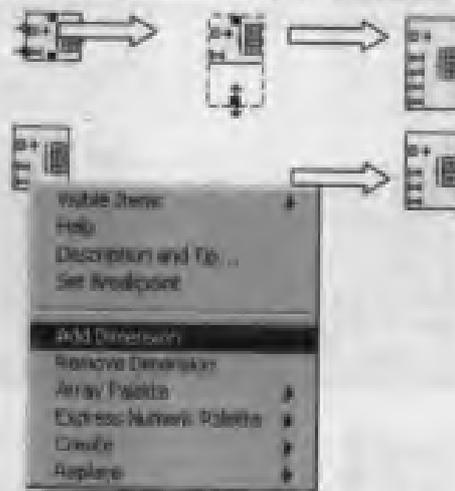


图 5.3.29 添加 dimension size 端口

例 5.3.11 初始化一个一维数组和一个二维数组。

VI 的前面板及框图程序如图 5.3.30 所示。

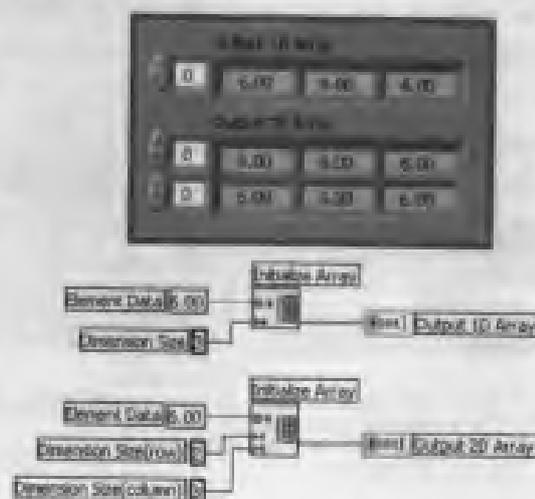


图 5.3.30 例 5.3.11 的前面板及框图程序

另外，在 LabVIEW 中初始化数组还有其他几种方法。

- 如果数组中的元素都是相同的，用一个带有常数的 For 循环即可初始化，这种方法的缺点是创建数组时要占用一定的时间。

- 如果元素值可以由一些直接的方法计算出来，把公式放到第一种方法中的 For 循环中取代其常数即可。例如用这种方法可以产生一个特殊的波形。

- 在框图程序中创建一个数组常量，手动输入各个元素的数值，而后将其连接到需要初始化的数组上。这种方法的缺点是繁琐，并且在 VI 存盘时会占用一定的磁盘空间。

- 在前面板上创建一个数组控制，手动键入各元素的数值，而后从该控制的右键弹出选单中选择 Data Operations → Make Current Value Default，可将键入的数值变为该数组的默认值。如果不再做改动，这些值将一直保持不变。然后，在框图程序窗口中该控制端口的右键弹出选单中选择 Hide Control 将其隐藏，以防止他人修改数据。这种方法的缺点是当 VI 存盘时要占用特别的空间。

- 如果初始化数组所用的数据量很大，可先将其放到一个文件中，在程序开始运行时再装载。

还需补充一点，在初始化数组时有一种特殊情况，那就是空数组 (Empty Array)，空数组不是一个元素值为 0、false、空字符串或其他类似数值的数组，而是一个包含零个元素的数组。这就相当于在 C 或 Pascal 语言中创建一个指向数组的指针。经常用到空数组的例子是初始化一个连有数组的循环移位寄存器 (将在第 6 章介绍)。创建空数组的几种常用方法如下。

- 用一个 dimension size 输入端口未连接数值或其输入值为 0 的 Initialize Array 节点来创建一个空数组。

- 创建一个计数终端 N 为 0 的 For 循环，在 For 循环中放入所需数据类型的常量。For 循环将执行零次，但在其框架通道上将产生一个相应类型的空数组。

- 在前面板上创建一个数组控制，在弹出选单的 Data Operations 中选择 Empty Array，再选择 Make Current Value Default 将其变为默认值。这样也能定义一个空数组。

- 在框图程序中用一个数组常量，并且在其弹出选单中的 Data Operations 中选择 Empty Array。

注意，不能用 Build Array 节点来创建空数组，因为它的输出至少包含一个元素。

7. Build Array

建立一个新数组。节点的图标及其输入输出端口定义如图 5.3.31 所示。节点将从左侧端口输入的元素或数组按从上到下的顺序组成一个新数组。

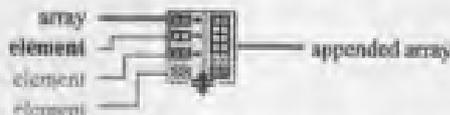


图 5.3.31 Build Array 节点的图标及其输入输出端口

节点在创建之初只有一个输入端口，用鼠标 (对象操作工具状态) 在节点图标下边缘 (或上边缘) 的尺寸控制点上拖动，或在节点左侧端口的右键弹出选单中选择 Add Input，可添加输入端口，如图 5.3.32 所示。

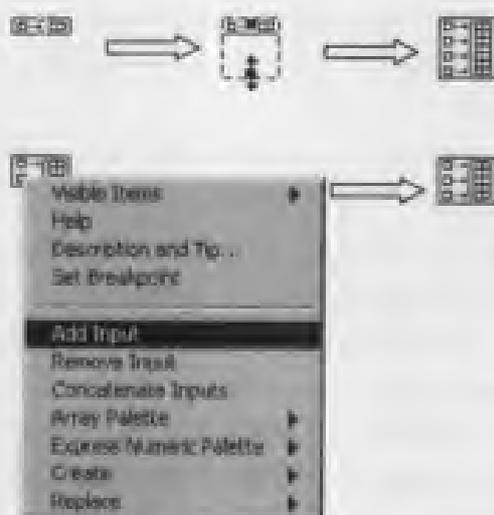


图 5.3.32 添加输入端口

例 5.3.12 利用 Build Array 节点组建几个数组。

VI 的前面板及框图程序如图 5.3.33 所示。

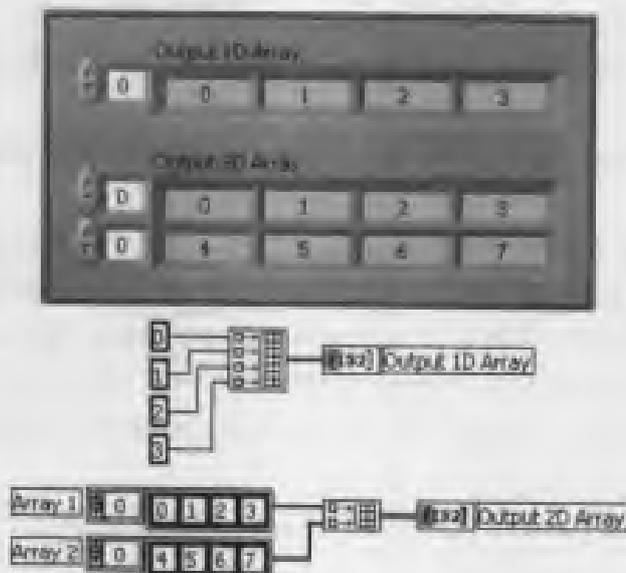


图 5.3.33 例 5.3.10 的前面板及框图程序

例 5.3.12 所实现的功能是几个数（标量）组合成一个一维数组，将两个一维数组组合成一个二维数组，即输出比输入高了一维。但很多情况下，用户需要将两个一维数组连接起来，形成一个更长的一维数组，而不是将两个一维数组组合成一个二维数组；或者是将两个二维数组连接起来形成一个新的二维数组，而不是将两个二维数组组合成一个三维数组。这种情况下，需要利用 Build Array 节点的连接输入（Concatenate Inputs）功能。如图 5.3.34 所示，在 Build Array 节点的右键弹出选单中选择 Concatenate Inputs，即可将 Build Array 节点的功能转换为连接输入功能。注意，当 Build Array 节点的功能不同时，Build Array 节点图标左侧的输入端口上的图案也不同。

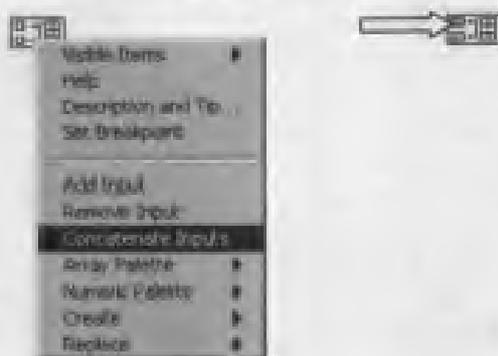


图 5.3.34 将 Build Array 节点的功能转换为连接输入功能

例 5.3.13 利用 Build Array 节点连接几个数组。

本例实现了两个功能，第一个功能是将两个数和一个一维数组连接成一个新的的一维数组，第二个功能是将两个一维数组连接成一个新的的一维数组，VI 的前面板及框图程序如图 5.3.35 所示。

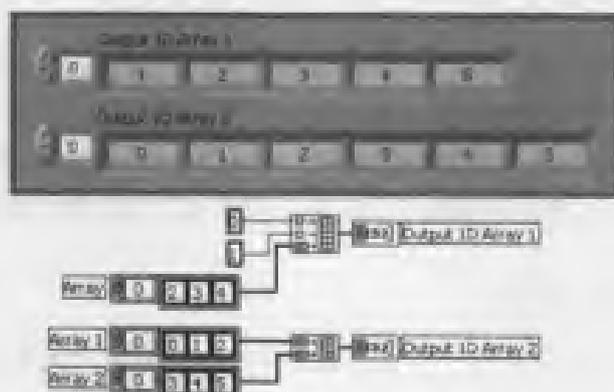


图 5.3.35 例 5.3.13 的前面板及框图程序

8. Array Subset

从输入数组中取出指定的元素，节点的图标及其端口定义如图 5.3.36 所示。输入数组可为 n 维，节点的输出是输入数组中从 Index 处开始，长度为 Length 的那部分元素。注意，当输入数组为 n 维时，节点的输出数组也为 n 维。

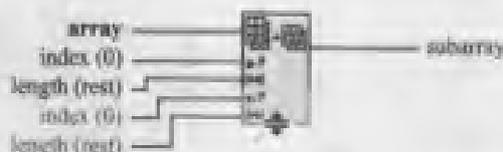


图 5.3.36 Array Subset 节点的图标及其端口定义

例 5.3.14 从一个二维数组中取出一部分元素。

VI 的前面板及框图程序如图 5.3.37 所示。

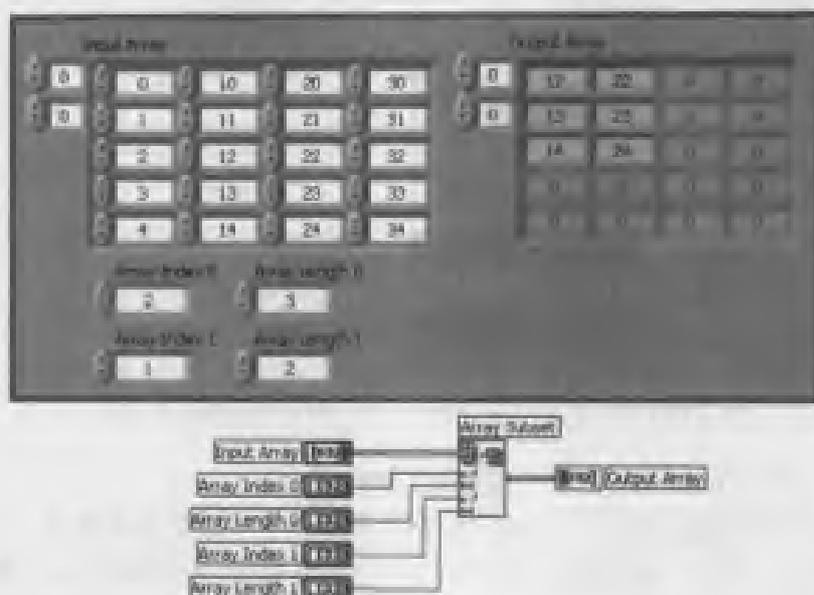


图 5.3.37 例 5.3.14 的前面板及框图程序

Array Subset 节点和 Index Array 节点的功能有些类似，但又有所不同。Index Array 节点只能从数组中取出一个元素或一行（列）元素，不能同时取出数组中的多个元素或多行（列）元素，而 Array Subset 节点则可以实现，但 Array Subset 节点的使用稍微复杂一些，需要更多的输入参数。用户可以根据自己的需求和编程习惯选择合适的节点。

9. Rotate 1D Array

将一个一维数组的最后 n 个元素移至数组的最前面，节点的图标及其端口定义如图 5.3.38 所示。



图 5.3.38 Rotate 1D Array 节点的图标及其端口定义

例 5.3.15 利用 Rotate 1D Array 节点将一个数组的最后两个元素移动至数组的最前面。VI 的前面板及程序框图如图 5.3.39 所示。

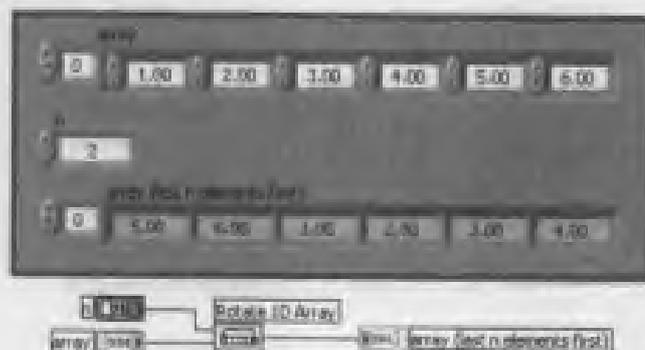


图 5.3.39 例 5.3.15 的前面板及框图程序

10. Reverse 1D Array

将输入的一维数组前后颠倒，输入数组可以是任意类型的数组，节点的图标及其端口定义如图 5.3.40 所示。



图 5.3.40 Reverse 1D Array 节点的图标及其端口定义

例 5.3.16 将一个一维数组前后颠倒。

VI 的前面板及框图程序如图 5.3.41 所示。



图 5.3.41 例 5.3.16 的前面板及框图程序

11. Search 1D Array

搜索指定元素在一维数组中的位置，节点的图标及其端口定义如图 5.3.42 所示。由 start index 端口指定开始搜索的位置，当数组指定位置后的那部分元素中没有该元素时，节点返回-1；若该元素存在，则返回该元素所在的位置。

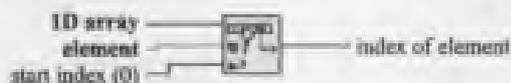


图 5.3.42 Search 1D Array 节点的图标及其端口定义

例 5.3.17 在一个一维数组中搜索一个数字的所在位置。

VI 的前面板及框图程序如图 5.3.43 所示。

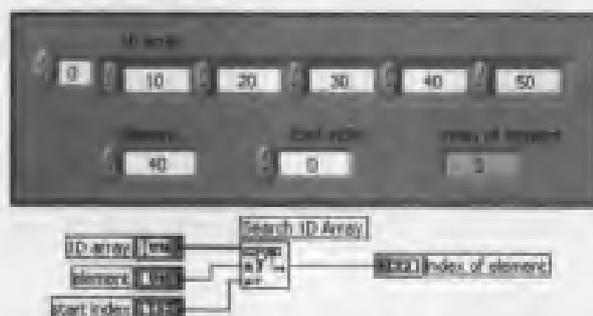


图 5.3.43 例 5.3.17 的前面板及框图程序

12. Split 1D Array

将输入的一维数组在指定的元素处截断，分成 2 个一维数组，节点的图标及其端口定义如图 5.3.44 所示。当输入的 index 值小于等于 0 时，第一个子数组为空；当输入的 index 值大于输入数组的长度时，第二个子数组为空。



图 5.3.44 Split 1D Array 节点的图标及其端口定义

例 5.3.18 将一个一维数组从第 4 个元素开始分成 2 个子数组。VI 的前面板及框图程序如图 5.3.45 所示。

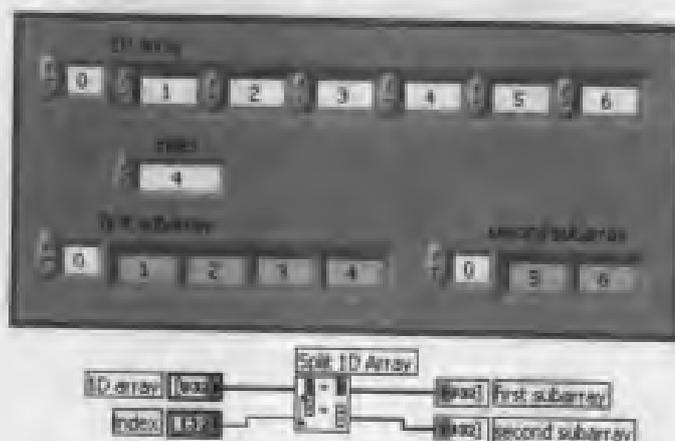


图 5.3.45 例 5.3.18 的前面板及框图程序

13. Sort 1D Array

将输入的一维数组按升序排序，节点的图标及其端口定义如图 5.3.46 所示。本节点与 Sort 1D Array 节点组合使用可实现对数组的降序排序。



图 5.3.46 Sort 1D Array 节点的图标及其端口定义

例 5.3.19 按升序和降序排序 1 个一维数组。

VI 的前面板及框图程序如图 5.3.47 所示。

注意，当输入数组中的元素为簇（有关簇的内容请见第 5.4 节）时，节点将按照簇中的第一个元素进行排序，另外，若使用该节点为布尔数组的排序，则 True 值比 False 值大。

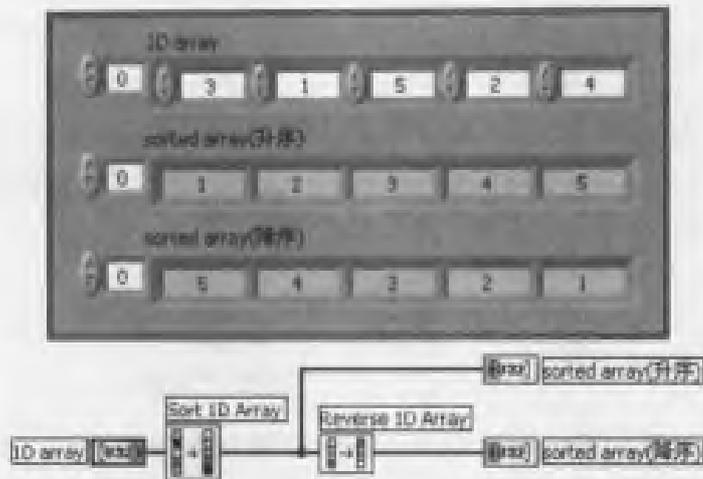


图 5.3.47 例 5.3.19 的前面板及框图程序

例 5.3.20 按升序排序一个簇的数组。

VI 的前面板及框图程序如图 5.3.48 所示。

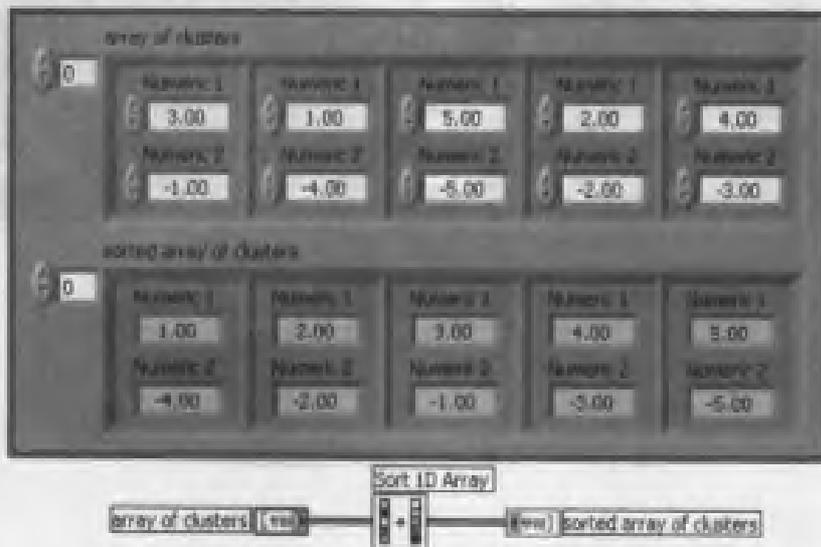


图 5.3.48 例 5.3.20 的前面板及框图程序

例 5.3.21 按升序排序 1 个一维布尔数组。

VI 的前面板及框图程序如图 5.3.49 所示。



图 5.3.49 例 5.3.21 的前面板及框图程序

14. Array Max & Min

返回输入数组中的最大值和最小值, 以及它们在数组中所在的位置, 节点的图标及其端口定义如图 5.3.50 所示。数组可以是任意维的, 当数组中有多个元素同为最大值或最小值时, 节点只返回第一个最大值或最小值所在的位置。

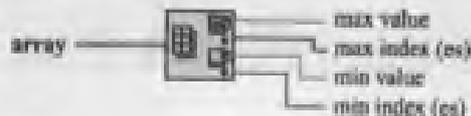


图 5.3.50 Array Max & Min 节点的图标及其端口定义

例 5.3.22 查找一个二维数组中的最大值和最小值以及它们在数组中的位置。VI 的前面板及框图程序如图 5.3.51 所示。

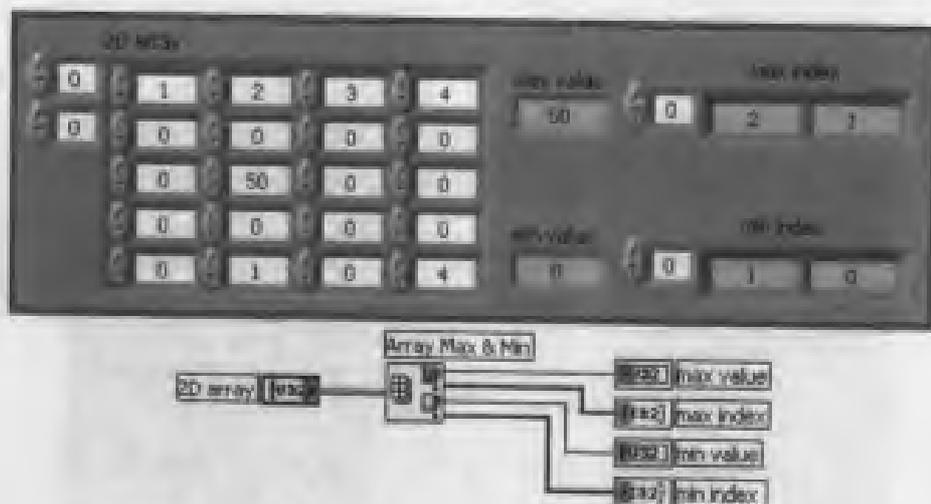


图 5.3.51 例 5.3.22 的前面板及框图程序

15. Transpose 2D Array

转置输入的二维数组, 也即矩阵转置。节点的图标及其端口定义如图 5.3.52 所示。



图 5.3.52 Transpose 2D Array 节点的图标及其端口定义

例 5.3.23 转置一个矩阵。VI 的前面板及框图程序如图 5.3.53 所示。

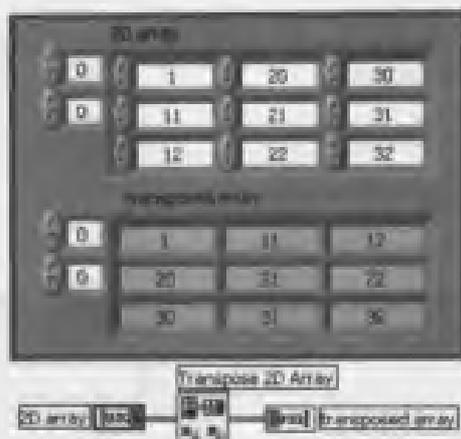


图 5.3.53 例 5.3.23 的前面板及框图程序

16. Interpolate 1D Array

线性插值。节点的图标及其端口定义如图 5.3.54 所示，根据 fractional index or x 端口的输入来确定输入数组中对应的 y 值。



图 5.3.54 Interpolate 1D Array 节点的图标及其端口定义

例 5.3.24 在一个一维数组中进行线性插值。

VI 的前面板及框图程序如图 5.3.55 所示。

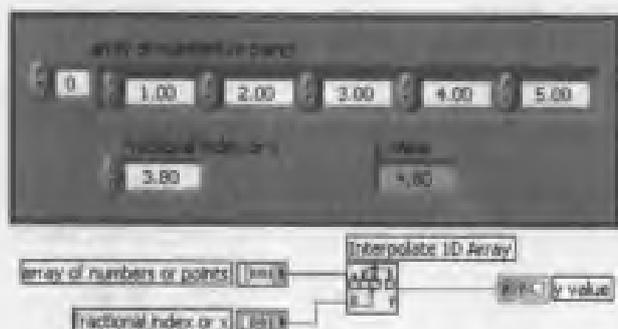


图 5.3.55 例 5.3.24 的前面板及框图程序

17. Threshold 1D Array

一维数组阈值，是线性插值的逆过程。节点的图标及其端口定义如图 5.3.56 所示，根据 threshold y 端口所输入的 y 值，从 start index 端口输入的开始位置在一维数组中确定 y 值的位置。



图 5.3.36 Threshold 1D Array 节点的图标及其端口定义

例 5.3.25 在 1 个一维数组中进行阈值。
VI 的前面板及框图程序如图 5.3.57 所示。

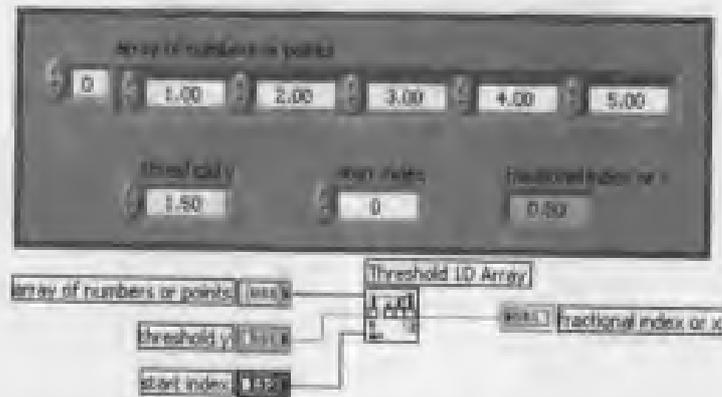


图 5.3.57 例 5.3.25 的前面板及框图程序

18. Interleave 1D Arrays

将从输入端口输入的一维数组插入到输出的一维数组中，节点的图标及其端口定义如图 5.3.58 所示。插入的顺序为：首先按从上到下的顺序取出所有输入数组中的第 0 个元素，放入输出数组中；然后再按从上到下的顺序取出所有输入数组中的第一个元素，放入输出数组中；其他元素的取法依次类推。



图 5.3.58 Interleave 1D Arrays 节点的图标及其端口定义

用鼠标（对象操作工具状态）拖动节点图标下边缘（或上边缘）上的尺寸控制点，或者在输入端口的右键弹出菜单中选择 Add Input，可添加输入端口的个数，如图 5.3.59 所示。

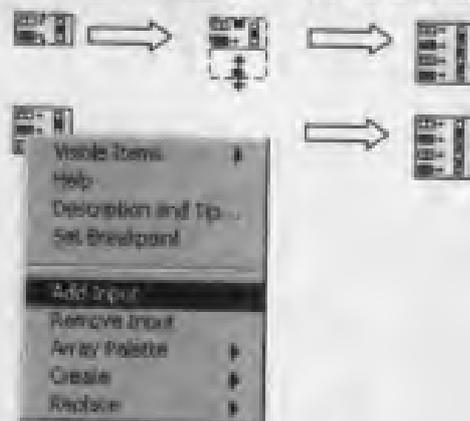


图 5.3.59 添加输入端口

例 5.3.26 Interleave 1D Arrays 节点应用举例。

VI 的前面板及框图程序如图 5.3.60 所示。

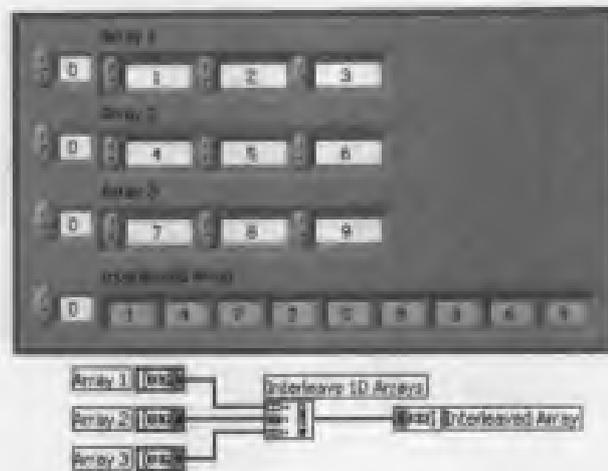


图 5.3.60 例 5.3.26 的前面板及框图程序

19. Decimate 1D Array

将输入的一维数组分成数个一维数组，是 Interleave 1D Arrays 的逆过程。节点的图标及其端口定义如图 5.3.61 所示。



图 5.3.61 Decimate 1D Array 节点的图标及其端口定义

当输出端口数为 n 时，第 1 个输出数组的组成元素为输入数组中的第 $0, n, 2n, \dots$ 个元素；第 2 个输出数组的组成元素为输入数组中的第 $1, n+1, 2n+1, \dots$ 个元素；第 3 个、第 4 个、……第 n 个输出数组依次类推。

用鼠标（对象操作工具状态）拖动节点图标下边缘（或上边缘）上的尺寸控制点，或在输出端口的右键弹出选单中选择 Add Output，可增加输出端口的个数，如图 5.3.62 所示。

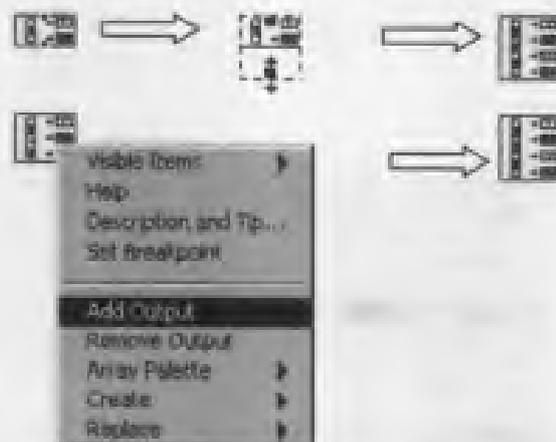


图 5.3.62 添加输出端口

例 5.3.27 Decimate 1D Array 节点应用举例。

VI 的前面板及框图程序如图 5.3.63 所示。

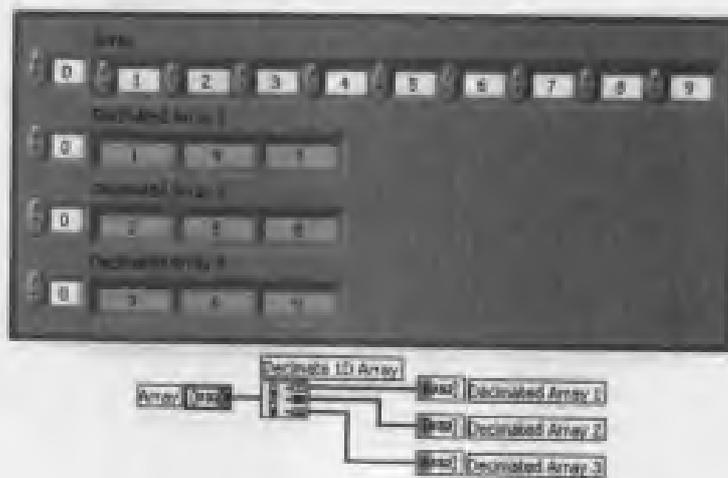


图 5.3.63 例 5.3.27 的前面板及框图程序

20. Reshape Array 

改变输入数组的维数。节点的图标及其端口定义如图 5.3.64 所示。输出数组的维数由节点图标左侧 dimension size 端口的个数决定。

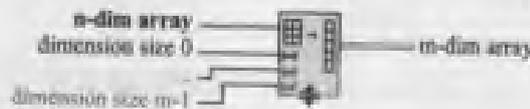


图 5.3.64 Reshape Array 节点的图标及其端口定义

用鼠标（对象操作工具状态）在节点图标下边缘（或上边缘）的尺寸控制点上拖动，或在 dimension size 端口的右键弹出选单中选择 Add Dimention，可添加 dimension size 端口，如图 5.3.65 所示。

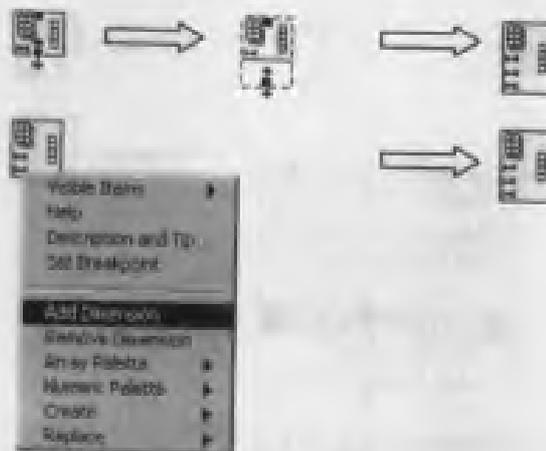


图 5.3.65 添加 dimension size 端口

例 5.3.28 将 1 个一维数组转化为二维数组。
VI 的前面板及框图程序如图 5.3.66 所示。

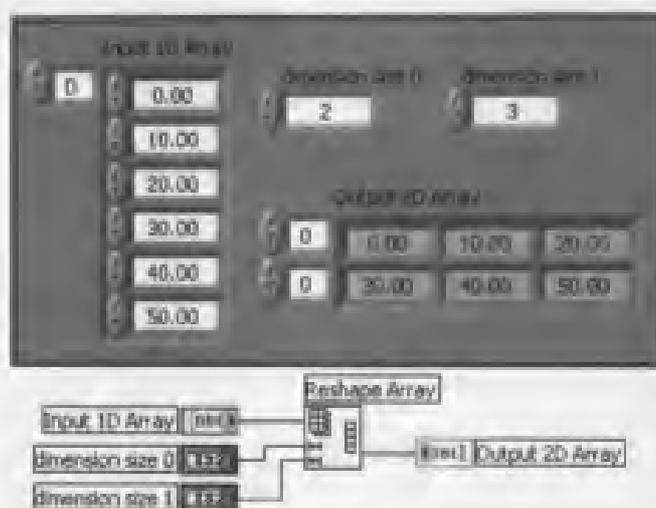


图 5.3.66 例 5.3.28 的前面板及框图程序

另外，在 Array 子模板中还有两个节点 Array To Cluster 和 Cluster To Array，下一节将详细介绍这两个节点的用法。

5.3.3 数组的特点

LabVIEW 中的数组与其他编程语言相比比较灵活，任何一种数据类型的数据（数组本身除外）都可以组成数组。其他的编程语言如 C 语言，在使用一个数组时，必须首先定义该数组的长度，但 LabVIEW 却不必如此，它会自动确定数组的长度。在内存允许的情况下，数组中每一维的元素最多可达 $2^{31}-1$ 个。数组中元素的数据类型必须完全相同，如都是无符号 16 位整数，或都是布尔型等。当数组中有 n 个元素时，元素的索引号从 0 开始，到 $n-1$ 结束。

5.4 簇

簇 (Cluster) 是 LabVIEW 中一个比较特别的数据类型，它可以将几种不同的数据类型集中到一个单元中形成一个整体。簇类似于 Pascal 语言中的 record 或 C 语言中的 struct。簇通常可将框图程序中的多个相关数据元素集中到一起，这样只需一条数据连线即可把多个节点连接到一起，不仅减少了数据连线的数量，还可以减少 SubVI 的连接端口的数量。

5.4.1 簇的组成与创建

前几章介绍了数字型、布尔型和字符串型等不同的数据类型。但仅有这些数据类型是不够的，为便于引用，有时还需要将不同的数据类型组合成一个有机的整体。例如，一个

学生的学号、姓名、性别、年龄、成绩和家庭地址等数据项。这些数据项都与某一个学生相联系。如果将这些数据项分别定义为相互独立的简单变量,是难以反映它们之间的内在联系的,应当把这些数据项组成一个组合项,这样它们就组合成一个有机的整体,LabVIEW 中的簇就是这样一种数据结构。

关于上述学生的数据项在 C 语言中可以用 Structure 描述:



图 5.4.1 簇

```

struct student
{
    int num;
    char name[20];
    char sex;
    int age;
    float score;
    char address[30];
};
    
```

上述数据类型在 LabVIEW 中可以用簇来实现,如图 5.4.1 所示。

簇的创建类似于数组的创建,首先在 Controls 模板 → All Controls 子模板 → Array & Cluster 子模板中创建簇的框架,如图 5.4.2 所示。

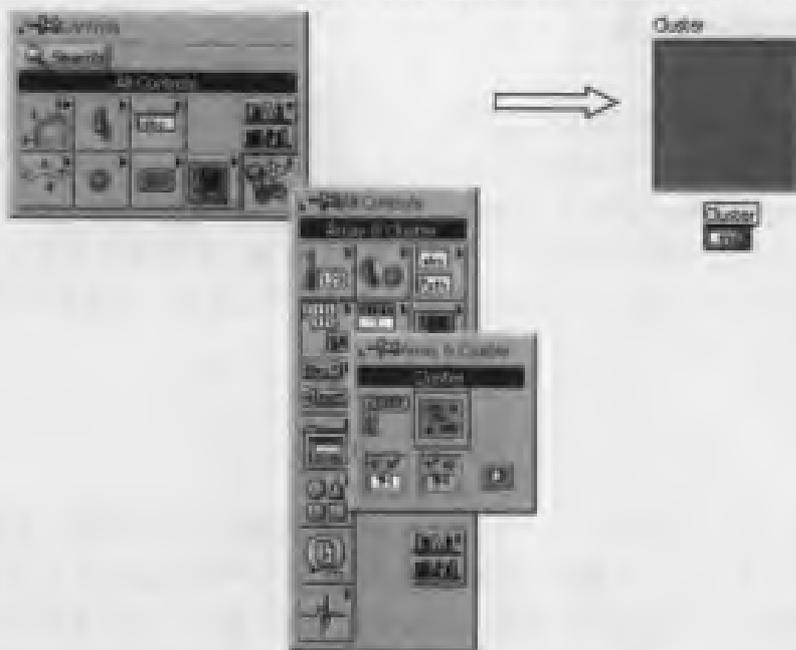


图 5.4.2 创建簇的框架

然后,向框架中添加所需的元素,最后根据编程需要更改簇和簇中各元素的名称,如图 5.4.3 所示。



图 5.4.3 创建簇

在 LabVIEW 中，簇只能包含控制和指示中的一种，不能既包含控制又包含指示。可以用 Decorations 子模板中的装饰将二者集中在一起，但这种集中仅是位置上的集中。若确实需要对一个簇既读又写，那么可用簇的本地变量来实现，但并不推荐用本地变量对簇进行连续地读写。使用簇时应当遵循一个原则：在一个高度交互的面板中，不要把一个簇既用作输入又用作输出元素。

5.4.2 簇的使用

用户在使用一个簇时，主要是访问簇中的各个元素，或者用不同类型但相互关联的数据组成一个簇。在 LabVIEW 中，这些功能由 Functions 模板→All Functions 子模板→Cluster 子模板中的各个节点来实现，如图 5.4.4 所示。本小节将介绍如何使用这些节点。

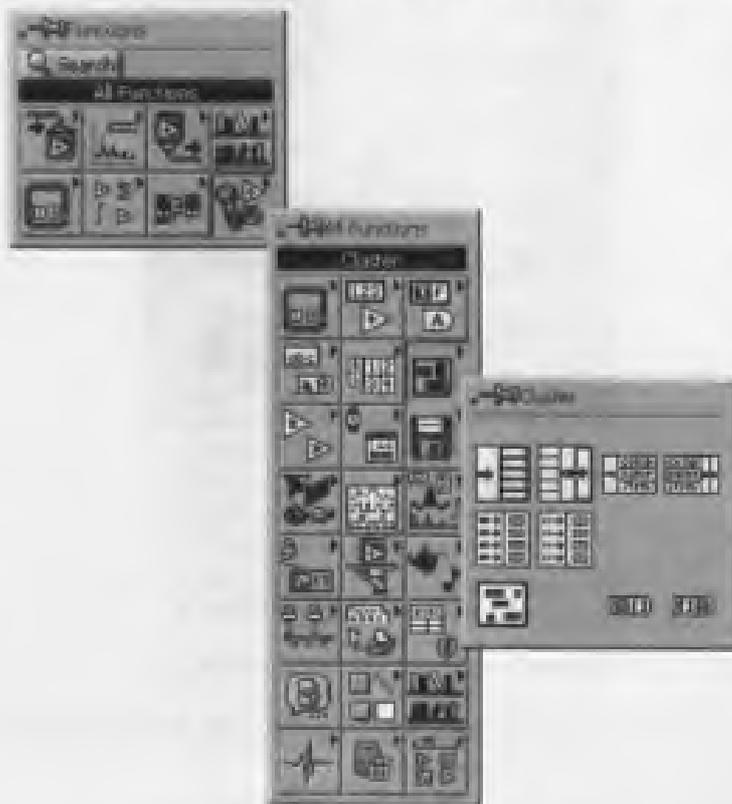


图 5.4.4 Cluster 子模板

1. Unbundle

解包，用该节点可以获得簇中元素的值，节点的图标及其端口定义如图 5.4.5 所示。

一个新的 Unbundle 节点只有两个输出端口，当 Unbundle 节点的输入端口 cluster 中输入一个簇时，Unbundle 节点会检测输入簇的元素的个数，自动生成相应个数的输出端口，如图 5.4.5 所示。

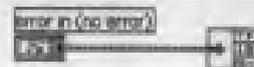
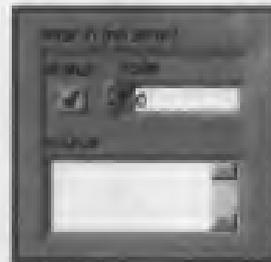
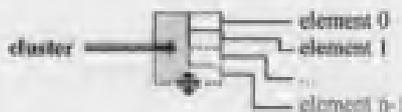


图 5.4.5 Unbundle 节点的图标及其端口定义

图 5.4.6 Unbundle 节点自动生成输出端口

例 5.4.1 将一个簇中的各个元素值分别取出。

VI 的前面板及框图程序如图 5.4.7 所示。



图 5.4.7 例 5.4.1 的前面板及框图程序

注意，节点将按照簇中元素的编号顺序 (Order) 从上到下依次输出簇中各个元素的值。关于簇中元素的编号将在第 5.4.3 小节中详细介绍。

2. Bundle

打包。将相互关联的不同数据类型（当然也可以相同）的数据组成一个簇，或给簇中的某一个元素赋值。节点的图标及其端口如图 5.4.8 所示。

与 Unbundle 节点相同，节点 component 端口的个数必须与簇中元素的个数一致。用鼠标（对象操作工具状态）在节点图标下边缘（或上边缘）的尺寸控制点上拖动，或在图标 component 端口的右键弹出选单中选择 Add Input，可添加 component 端口，如图 5.4.9 所示。

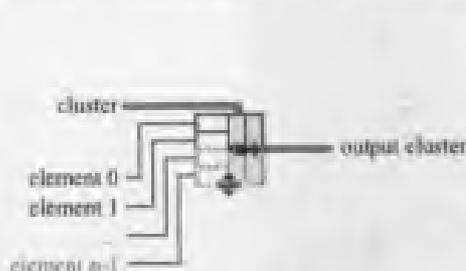


图 5.4.8 Bundle 节点的图标及其端口定义

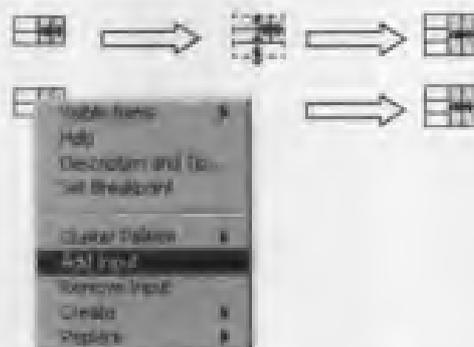


图 5.4.9 添加 component 端口

例 5.4.2 将几个不同的数据类型组成一个簇。

VI 的前面板及框图程序如图 5.4.10 所示。



图 5.4.10 例 5.4.2 的前面板及框图程序

注意, 在例 5.4.2 中, 位于节点中间的输入端口 (Cluster of n Components) 是悬空的, 在这种情况下, 节点会将输入的数据由上到下地组成一个簇。用这种方法给一个簇赋值时, 簇中元素的编号顺序必须与输入数据的顺序一致, 并且输入数据的数据类型必须与簇中元素一一对应。也就是说, 必须给簇中的每一个元素赋值。但如果根据需要, 只需给簇中的某几个元素赋值时, 又该如何处理呢? 例 5.4.3 给出了解决办法。

例 5.4.3 修改一个簇中某些元素的值。

VI 的前面板及框图程序如图 5.4.11 所示。



图 5.4.11 例 5.4.3 的前面板及框图程序

本例为簇 Student 创建了一个本地变量, 并连接到节点中间的输入端口上, 通过这种方法就可以实现只给簇中的某几个元素赋值。此时, 与簇中其他几个元素所对应的那几个 component 端口是悬空的。

3. Unbundle By Name

按名称解包, 节点的图标及其端口定义如图 5.4.12 所示。用该节点可以得到由元素名称指定簇中相应元素的值, 与 Unbundle 节点相比, 该节点的应用较为灵活。



图 5.4.12 Unbundle By Name 节点的图标及其端口定义

例 5.4.4 Unbundle By Name 应用举例。

VI 的前面板及框图程序如图 5.4.13 所示。



图 5.4.13 例 5.4.4 的前面板及框图程序

用数据数据操作工具单击 name 端口，在弹出选单中可选择输入簇中的元素。在 name 端口的右键弹出选单中的 Select Item 子选单中也可以选择元素，如图 5.4.14 所示。

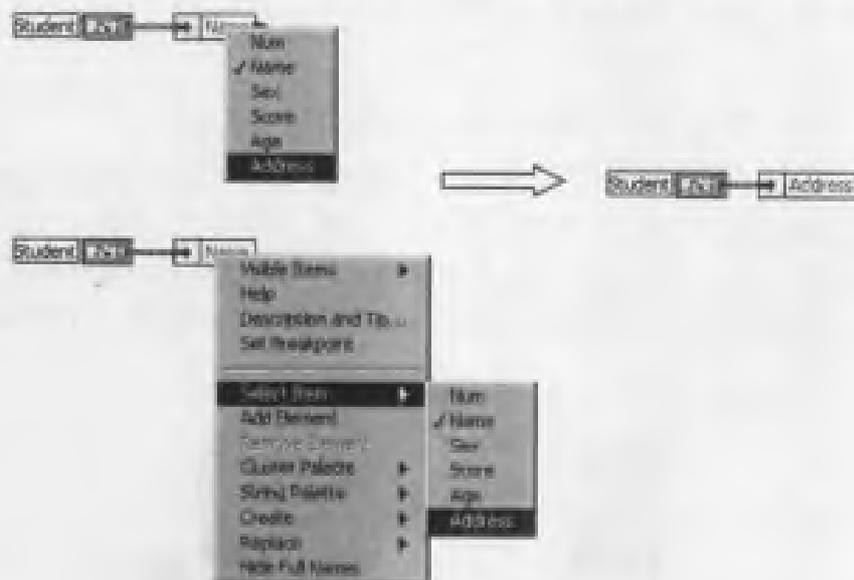


图 5.4.14 选择簇中的元素

注意，该节点 name 端口的个数不限，可根据编程需要添加任意数目的端口。用鼠标（对象操作工具状态）在节点图标下边缘（或上边缘）的尺寸控制点上拖动，或在 name 端口的右键弹出选单中选择 Add Element，可添加 name 端口，如图 5.4.15 所示。

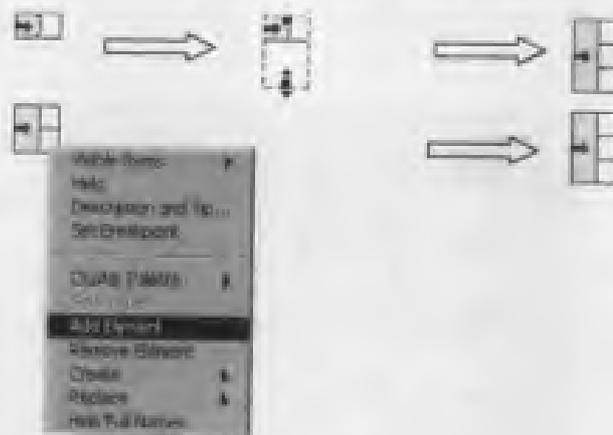


图 5.4.15 添加 name 端口

4. Bundle By Name

按名称打包。它是 UnBundle By Name 的逆过程，可将相互关联的不同数据类型（当然也可以相同）的数据组成一个簇，或给簇中的某一个元素赋值。节点的图标及其端口定义如图 5.4.16 所示。



图 5.4.16 Bundle By Name 节点的图标及其端口定义

与 Bundle 节点不同，在使用本节点构成一个簇时，必须在节点中间的输入端口（Cluster of N named Components）中输入一个簇，确定输出簇的元素组成。并且，由于该节点是按照元素名称进行打包的，所以 name 端口不必像 Bundle 节点那样有明确的顺序，只需按照在 name 端口的弹出选单中所选的元素名称接入相应数据即可，如图 5.4.17 所示。

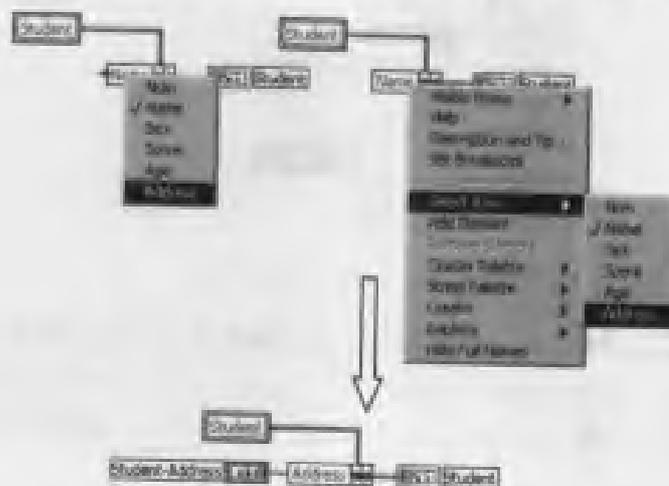


图 5.4.17 在 name 端口接入相应数据

例 5.4.5 用 Bundle By Name 节点修改一族中某一元素的值。
VI 的前面板及框图程序如图 5.4.18 所示。



图 5.4.18 例 5.4.5 的前面板及框图程序

例 5.4.6 用 Bundle By Name 节点创建一个簇。
VI 的前面板及框图程序如图 5.4.19 所示。



图 5.4.19 例 5.4.6 的前面板及框图程序

用鼠标（对象操作工具状态）在节点图标下边缘（或上边缘）的尺寸控制点上拖动，或在 name 端口的右键弹出菜单中选择 Add Element，可添加 name 端口，如图 5.4.20 所示。

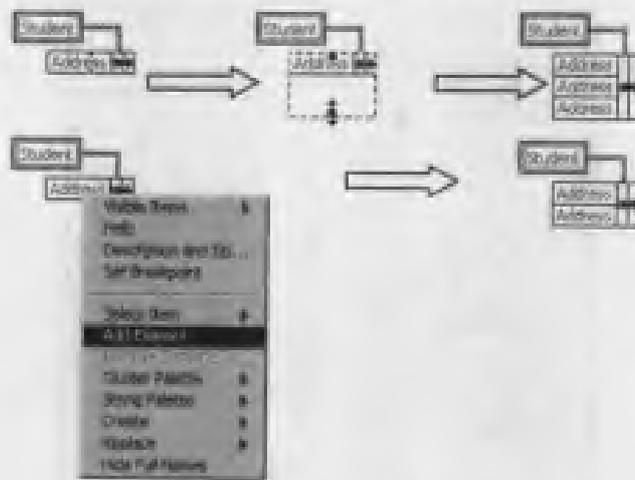


图 5.4.20 添加 name 端口

5. Build Cluster Array

建立簇的数组。用法与 Build Array 节点类似，节点的图标及其端口定义如图 5.4.21 所示。



图 5.4.21 Build Cluster Array 节点的图标及其端口定义

与 Build Array 不同的是，其 component 端口输入的元素可以是簇，节点会首先将输入到其 component 端口上的每一个元素转化为一个簇，然后再将这些簇组成一个簇的数组。

例 5.4.7 Build Cluster Array 节点应用举例 1。

VI 的前面板及框图程序如图 5.4.22 所示。

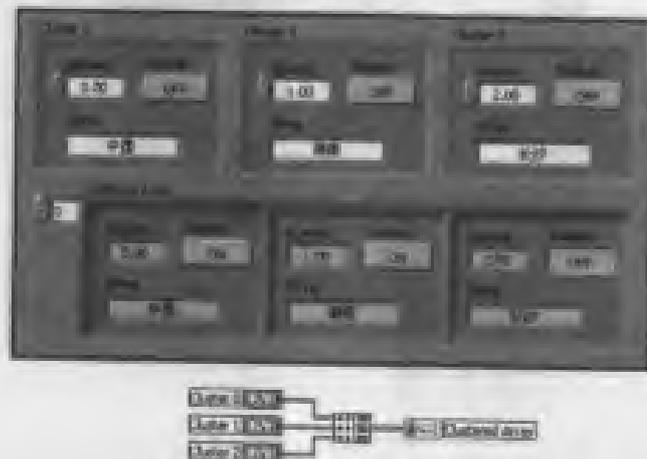


图 5.4.22 例 5.4.7 的前面板及框图程序

注意，所有从 component 端口输入的数据类型必须相同。用鼠标（对象操作工具状态）在节点图标下边缘（或上边缘）的尺寸控制点上拖动，或在输入端口的右键弹出选单中选择 Add Input，可添加输入端口，如图 5.4.23 所示。

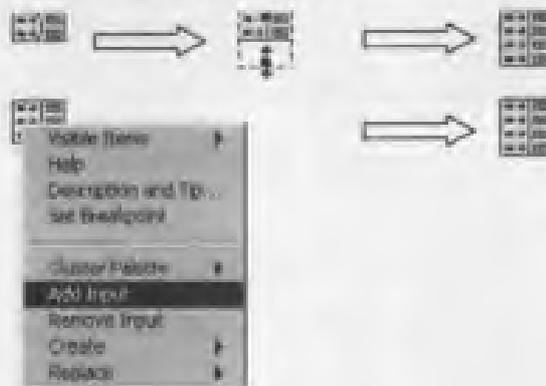


图 5.4.23 添加输入端口

例 5.4.8 Build Cluster Array 节点应用举例 2。

本例将几个数字标量组成簇的数组。VI 的前面板及框图程序如图 5.4.24 所示。

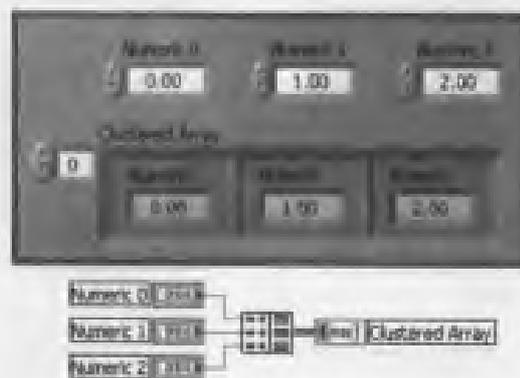


图 5.4.24 例 5.4.8 的前面板及框图程序

6. Index & Bundle Cluster Array

将输入数组中的元素按照索引组成簇，然后将这些簇组成一个数组，节点的图标及其端口定义如图 5.4.25 所示。



图 5.4.25 Index & Bundle Cluster Array 节点的图标及其端口定义

例 5.4.9 Index & Bundle Cluster Array 节点应用举例。

VI 的前面板及框图程序如图 5.4.26 所示。



图 5.4.26 例 5.4.9 的前面板及框图程序

用鼠标 (对象操作工具状态) 在节点图标下边缘 (或上边缘) 的尺寸控制点上拖动, 或在输入端口的右键弹出选单中选择 Add Input, 可添加输入端口, 如图 5.4.27 所示。

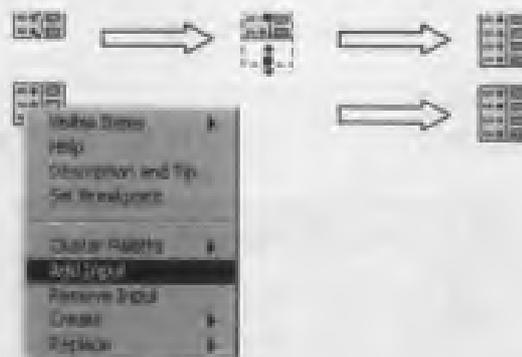


图 5.4.27 添加输入端口

7. Cluster To Array

将簇转化为数组。节点的图标及其端口定义如图 5.4.28 所示。



图 5.4.28 Cluster To Array 节点的图标及其端口定义

输入簇的所有元素的数据类型必须相同。节点将按照簇中元素的编号顺序地将这些元素组成一个一维数组。

例 5.4.10 Cluster To Array 节点应用举例。

VI 的前面板及框图程序如图 5.4.29 所示。

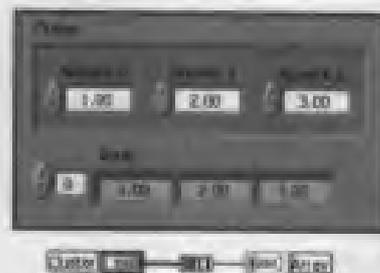


图 5.4.29 例 5.4.10 的前面板及框图程序

8. Array To Cluster

将数组转化为簇,是Cluster To Array的逆过程,节点的图标及其端口定义如图5.4.29所示。



图 5.4.30 Array To Cluster 节点的图标及其端口定义

例 5.4.11 Array To Cluster 节点应用举例。

VI 的前面板及框图程序如图 5.4.31 所示。



图 5.4.31 例 5.4.11 的前面板及框图程序

注意,该节点并不是将数组中所有的元素都转化为簇,而是将数组中前 n 个元素组成一个簇。 n 由编程者指定,默认值为 9。在节点图标的右键弹出选单中选择 Cluster Size...,可在弹出对话框中定义 n ,如图 5.4.32 所示。当 n 大于数组长度时,节点会自动补足簇中元素,元素值为默认值。

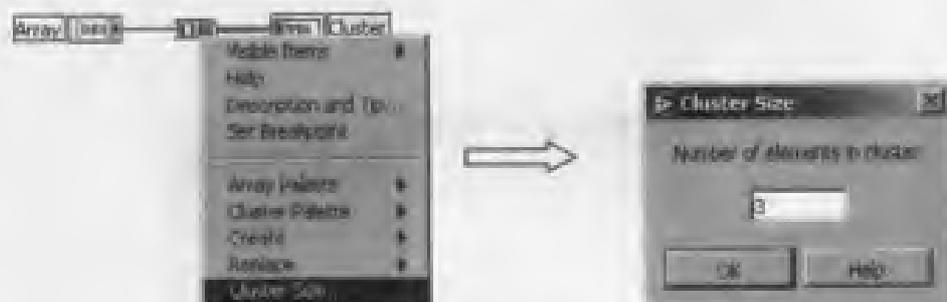


图 5.4.32 设定簇中元素的个数

5.4.3 簇的特点

前面提到某些节点需要按照簇中元素的编号顺序进行某些操作,给元素编号是簇的一大特点。通过这种方式,LabVIEW 可以很容易地管理簇中的元素。

创建一个簇时,LabVIEW 会按照簇中元素创建的先后次序给簇中的元素进行默认编号。编号从 0 开始,依次为 1,2,……当然,也可根据编程需要自己定义元素的编号。在簇框架的右键弹出选单中选择 Reorder Controls In Cluster...,LabVIEW 的前面板会变为元素顺

序编辑器, 在编辑器中用鼠标单击元素的编号, 即可改变元素的编号。编辑完所有编号后, 单击工具条上的 OK 按钮确定, 如图 5.4.33 所示。



图 5.4.33 改变簇中元素的编号

另外, 簇的框架还有自动缩放功能, 能根据簇中的元素自动确定框架的大小。在簇的框架的右键弹出选单中选择 Autosizing → Size to Fit, 簇的框架就自动缩放到合适的大小, 如图 5.4.34 所示。

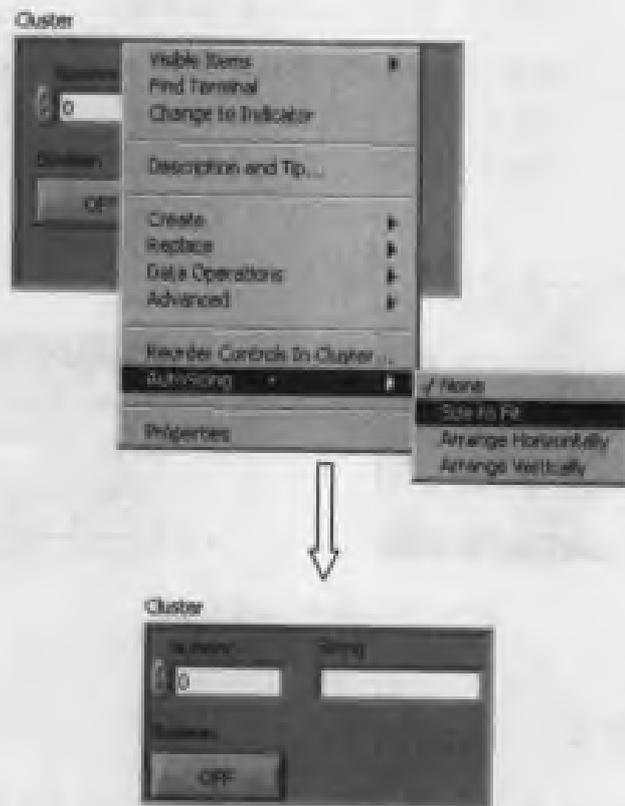


图 5.4.34 设定簇框架的自动缩放功能

另外 Autosizing 子选单中还有两个选项: Arrange Horizontally 和 Arrange Vertically。利用这两个选项, 可以将簇中的元素按照水平方向或垂直方向重新分布。

在 LabVIEW 的 Array & Cluster 子模板中, 还有一种非常特殊但很有用的簇, 称为 Error Cluster, 相应的控制和指示如图 5.4.35 所示。



图 5.4.35 Error Cluster

这种簇主要用于处理 LabVIEW 中的错误, 在 DAQ、VISA、VXI、GPIB、文件 I/O、TCP/IP, 以及 Serial 通信等各种功能节点中均会发生一定的错误, 为了方便地处理这些错误, LabVIEW 引入了 Error Cluster, Error Cluster 中包含三个元素: status, code 及 source。元素 status 是一布尔按钮, 值为 False 时, 表示无错状态; 值为 True 时, 表示出现了错误。元素 code 为 32 位整数, 表示错误代码, 在无错状态时为 0, 错误代码表见附录。元素 source 是一个字符串, 用于解释错误。

在 Functions 模板 → All Functions 子模板 → Time & Dialog 子模板中有 6 个与错误处理相关的节点, 第 10 章将详细介绍其用法。

簇经常被合并成数组, 称为簇的数组 (Array of Cluster)。用这种方法在处理大量采集数据时会非常方便, 如进行多个通道的 I/O 配置, 要将描述一个通道的各种不同数据块放到一个簇中, 再把所有的簇放到数组中。可以用簇来建立一个数组的数组的等效数据结构形式——数组的簇的数组, 注意它与二维数组或矩阵是不同的。通过这种方法可以把不同长度或不同维数的数组集中在一起。

5.5 波形数据

在通常情况下, 一块数据采集卡返回的测试数据通常包含三个部分: 起始时间、两个数据点之间的时间间隔 (即采样频率的倒数), 以及一串离散的波形数据值。在较早的 LabVIEW 版本中, 用户需要用 Bundle 节点将这三部分数据打包后进行相关的处理, 例如, 送到前面板窗口中的 Waveform Graph 中显示波形。这对用户来讲, 需要一定量的编程, 比较麻烦。从 LabVIEW 6 开始, LabVIEW 就把这三部分数据组合成了一个新的数据类型——波形数据 (Waveform Data), 这样用户不需要编程, 直接就可以对这些数据进行了。本节将介绍在 LabVIEW 中如何使用波形数据。

5.5.1 波形数据的组成

LabVIEW 中的波形数据分为两类: 模拟波形数据 (Waveform Data) 和数字波形数据 (Digital Waveform Data)。

1. 模拟波形数据

模拟波形数据用来表示模拟信号的波形。例如, 正弦波、方波或其他形状的模拟信号, 1 个一维模拟波形数据数组可以表示多条模拟波形。模拟波形数据由 4 个元素组成: 起始时间、Delta t、波形数据和属性。

(1) 起始时间: t_0

起始时间 t_0 是第一个数据点的时间。起始时间可以用来同步多个波形, 也可以用来确定两个波形的相对时间。 t_0 的数据类型为时间标识 (Time Stamp)。

(2) Delta t: dt

Delta t (dt) 是一个波形中两个数据点之间的时间间隔。 dt 的数据类型为双精度浮点数。

(3) 数据: Y

模拟波形数据的 Y 是 1 个一维数组, 其默认数据类型为双精度浮点数, 其他数字类型的数字一维数组也可以作为模拟波形数据的 Y , Y 的数据类型为双精度浮点数数组。

(4) 属性: Attributes

属性包含了一些波形数据的信息, 例如, 波形名称、数据采集设备的名称等。利用 Set Waveform Attribute 节点可以设置波形数据的属性, 利用 Get Waveform Attribute 节点可以获得波形数据的属性。Attributes 的数据类型为变体型 (Variant), 模拟波形数据可以有多个属性, 每一个属性由两部分组成: 属性名称和属性值。其中属性名称的数据类型为字符串, 而属性值的数据类型可以是任意的数据类型。注意, 这些属性都包含在一个变体型数据中。

LabVIEW 利用前面板对象 Waveform 来存放模拟波形数据, Waveform 位于 Controls 模板 → ALL Controls 子模板 → I/O 子模板中。Waveform 在默认情况下只显示波形数据中的前三个元素 (t_0 , dt 和 Y), 在 Waveform 的右键弹出选单中选择 Visible Items → Attributes, 可以将第四个元素 Attributes 显示出来, 如图 5.5.1 所示。

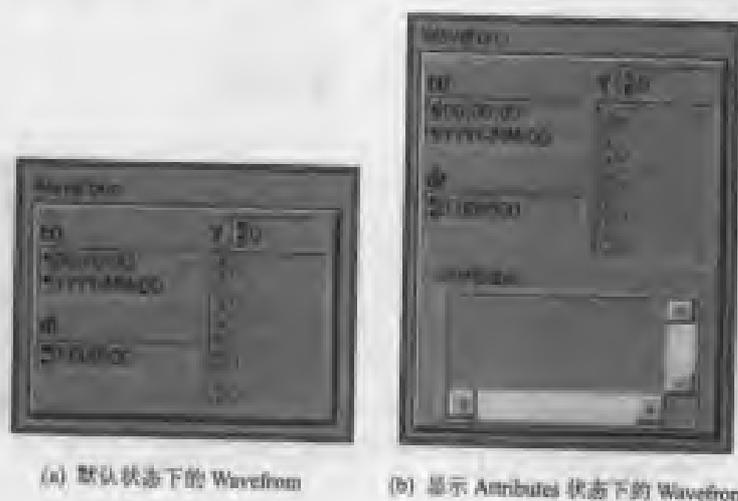


图 5.5.1 前面板对象 Waveform

前面板对象 Waveform Graph 和 Waveform Chart, 可用于显示模拟波形数据的曲线。Waveform Graph 位于 Controls 模板 → ALL Controls 子模板 → Graph 子模板中。

当将一个模拟波形数据连接到前面板对象 Waveform Graph 或 Waveform Chart 上显示

时, 这两个对象会自动根据模拟波形数据中的起始时间、 Δt 和波形数据 Y 将模拟波形曲线显示出来。

当将一个模拟波形数据数组连接到前面板对象 Waveform Graph 或 Waveform Chart 上显示时, 这两个对象会自动显示所有的模拟波形曲线。

另外, LabVIEW 中的许多用于数据采集和数据分析节点大都可以直接返回或处理模拟波形数据。

例 5.5.1 将一个 Waveform Data 数组 (包含一个正弦波和一个方波) 中的模拟波形数据送到一个 Waveform Graph 中显示出来。

本例所要说明的是 Waveform Graph 可以自动将 Waveform Data 数组中的曲线显示出来, VI 的前面板和框图程序如图 5.5.2 所示。

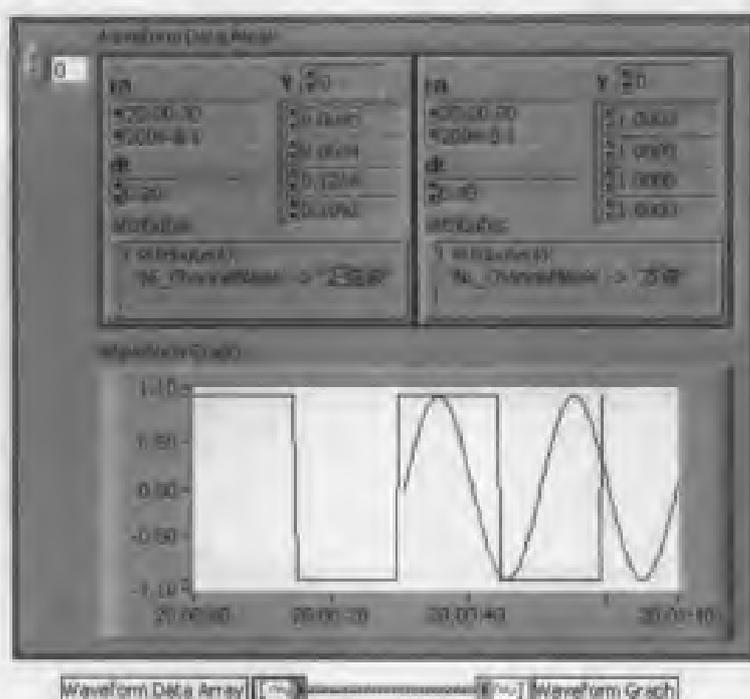


图 5.5.2 例 5.5.1 的前面板和框图程序

2. 数字波形数据

数字波形数据用来表示二进制数据, 例如, 0100110001101100。数字波形数据也是由四个元素组成: 起始时间、 Δt 、波形数据和属性。

(1) 起始时间: t_0

起始时间 t_0 是第一个数据点的时间, t_0 的数据类型是时间标识 (timestamp)。

(2) Δt : dt

Δt (dt) 是一个波形中两个数据点之间的时间间隔, dt 的数据类型为双精度浮点数;

(3) 数据: Y

数字波形数据的 Y 是二进制数字 (Digital Data) 波形, LabVIEW 用一个名为 Digital Data



图 5.5.3 前面板对象 Digital Data

的前面板对象来存放将这些数据, Digital Data 可以存放多条二进制数字波形, 如图 5.5.3 所示。Digital Data 位于 Controls 模板 → ALL Controls 子模板 → I/O 子模板中。Y 的数据类型为二进制数字 (Digital Data)。

(4) 属性: Attributes

数字波形数据的属性与模拟波形数据的属性相同, 在此不再赘述。

LabVIEW 利用前面板对象 Digital Waveform 来存放数字波形数据。Digital Waveform 位于 Controls 模板 → ALL Controls 子模板 → I/O 子模板中。Digital Waveform 在默认情况下只显示波形数据中的前三个元素 (t0, dt 和 Y), 在 Digital Waveform 的右键弹出菜单中选择 Visible Items → Attributes, 可以将第四个元素 Attributes 显示出来, 如图 5.5.4 所示。

中



(a) 默认状态下的 Digital Waveform

(b) 显示 Attributes 状态下的 Digital Waveform

图 5.5.4 前面板对象 Digital Waveform

前面板对象 Digital Waveform Graph 可用于显示数字波形数据的曲线, 如图 5.5.5 所示。Digital Waveform Graph 位于 Controls 模板 → ALL Controls 子模板 → Graph 子模板中。

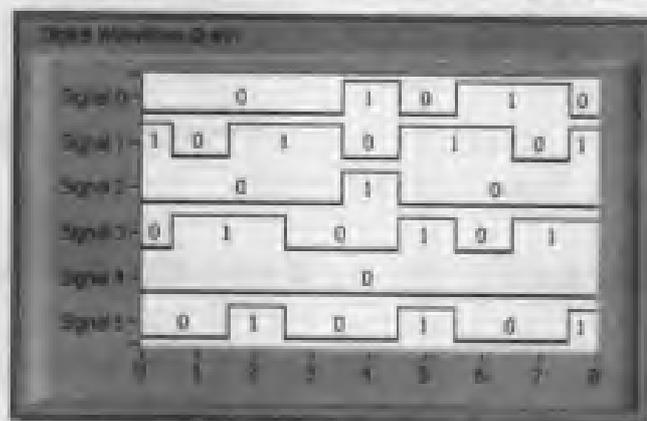


图 5.5.5 前面板对象 Digital Waveform Graph

当将一个数字波形数据连接到 Digital Waveform Graph 上显示时,这个对象会自动根据模拟波形数据中的起始时间、 Δt 和数字波形数据 Y 将数字波形曲线显示出来。

当将一个数字波形数据数组连接到 Digital Waveform Graph 上显示时,这个对象会自动显示所有的数字波形曲线。

5.5.2 波形数据的使用

LabVIEW 提供了大量的波形数据运算节点,利用这些节点可以访问和操作模拟波形数据和数字波形数据。波形数据运算节点位于 Functions 模板→All Functions 子模板→Waveform 子模板中,如图 5.5.6 所示。



图 5.5.6 Waveform 子模板

Waveform 子模板中的波形数据运算节点可分为四个部分:基本波形数据运算节点、模拟波形数据运算节点、数字波形数据运算节点和波形数据的存取节点。

下面将主要介绍基本波形数据运算节点的使用方法,如图 5.5.7 所示。其他波形数据运算节点的用法,请参见 LabVIEW 的帮助文档。注意,如图 5.5.7 所示的 1 个波形数据处理节点既可以用来处理模拟波形数据,也可以用来处理数字波形数据。



图 5.5.7 基本波形数据运算节点

1. Get Waveform Components

将波形数据的 4 个组成元素分离, 节点的图标及其端口定义如图 5.5.8 所示。节点在创建时只有 1 个输出端口, 利用鼠标 (对象操作工具) 拖动节点图标下边缘 (或上边缘) 上的尺寸控制点, 或者在输出端口的右键弹出选单中选择 Add Element, 可以添加输出端口。用鼠标 (数据数据操作工具) 单击输出端口, 或者在端口的右键弹出选单中选择 Select Item, 会出现一个下拉选单, 选单中列出了波形数据的 4 个组成元素的名称, 选择其中一个, 可以将该端口切换为这个元素的输出端口。

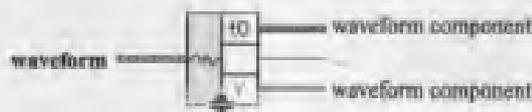


图 5.5.8 Get Waveform Components 节点的图标及其端口定义

例 5.5.2 将一个 Waveform 中的模拟波形数据的 4 个组成部分分离出来。本例的 VI 的前面板和框图程序如图 5.5.9 所示。

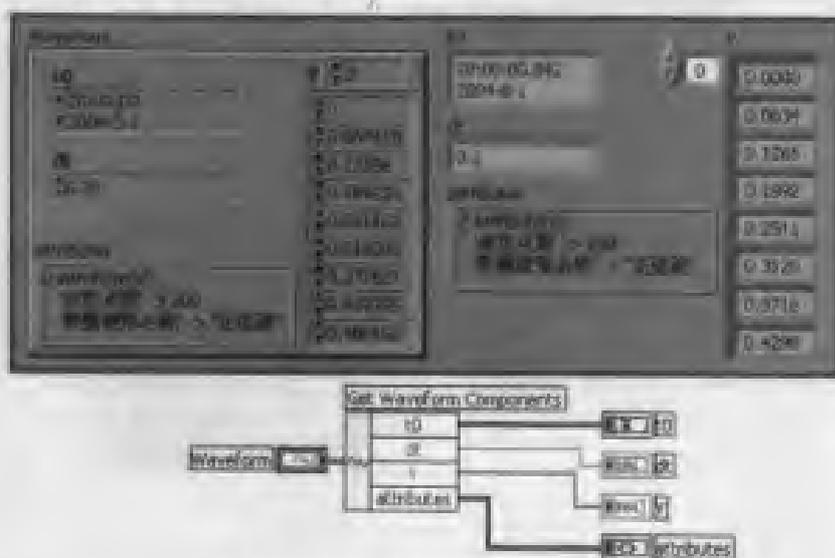


图 5.5.9 例 5.5.2 的前面板和框图程序

注意, Get Waveform Components 节点 attributes 端口返回的是一个变体型数据, 数据中包含了模拟波形数据的属性, 在 LabVIEW 中变体型数据是不能直接使用的, 必须用 Get Variant Attribute 节点和 Variant To Data 节点将其转化为每一个属性值所对应的数据类型的数据之后, 才可以用 LabVIEW 中的相关节点处理。请看下面的例子。

例 5.5.3 利用 Get Waveform Attribute 节点获得波形数据的属性。

本例利用了两个 Variant Attribute 节点和两个 Variant To Data 节点。

首先, 利用第一个 Variant Attribute 节点获得“波形长度”的属性值, 该节点返回的属性值是变体型, 利用第一个 Variant To Data 节点将这个变体型的属性值转换为无符号 32 位整数 200。

然后, 利用第二个 Variant Attribute 节点获得属性“波形数据名称”的属性值, 利用第二个 Variant To Data 节点将这个变体型的属性值转换为字符串“正弦波”。

VI 的前面板和框图程序如图 5.5.10 所示。

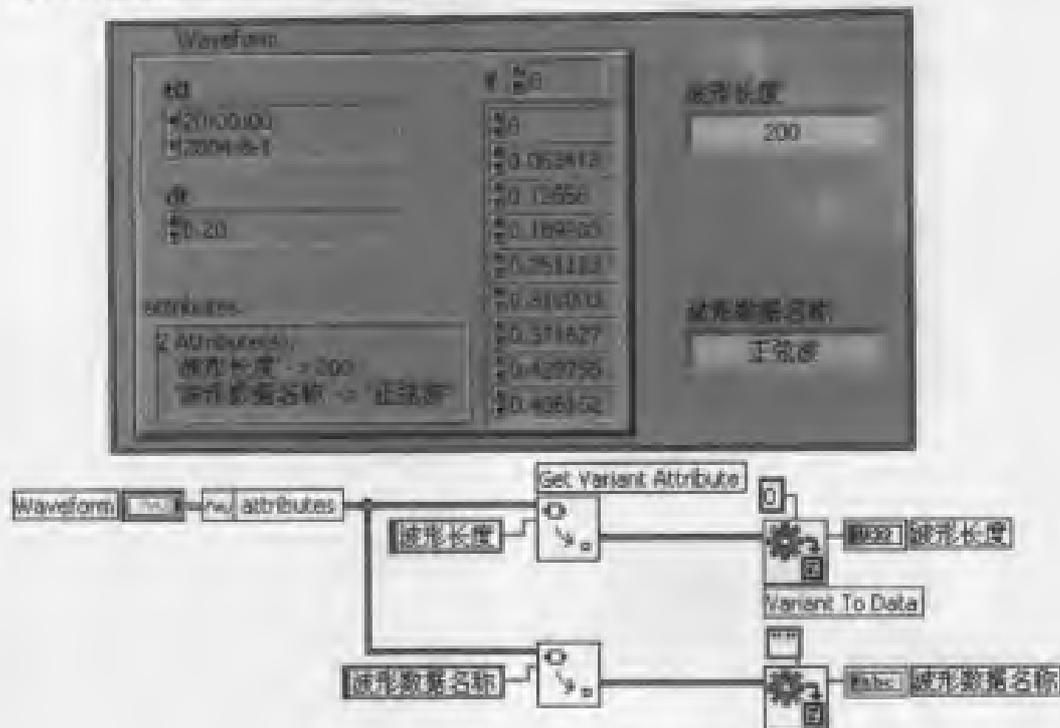


图 5.5.10 例 5.5.3 的前面板和框图程序

另外，利用 Get Waveform Attribute 节点也可以获得模拟波形数据的属性值。LabVIEW 还有一个专门用于获得波形数据属性的节点——Get Waveform Attribute 节点，可以利用该节点实现例 5.5.3 的功能，请见例 5.5.7。

2. Build Waveform

创建一个新的波形数据，或者修改一个波形数据中的某几个元素值，节点的图标及其端口定义如图 5.5.11 所示。当 waveform 端口没有连接数据时，节点创建一个新的波形数据；当 waveform 端口连接了一个波形数据时，节点根据 waveform component 端口的输入，修改这个波形数据中的值，并输出修改后的波形数据。节点在创建之初只有一个输入端口，添加输入端口，以及选择输入端口输入元素名称的方法与 Get Waveform Components 节点相同。

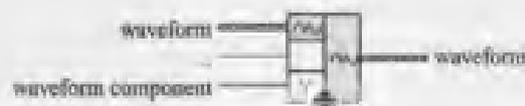


图 5.5.11 Build Waveform 节点的图标及其端口定义

例 5.5.4 组建一个模拟波形数据。

本例的 VI 的前面板和框图程序如图 5.5.12 所示。

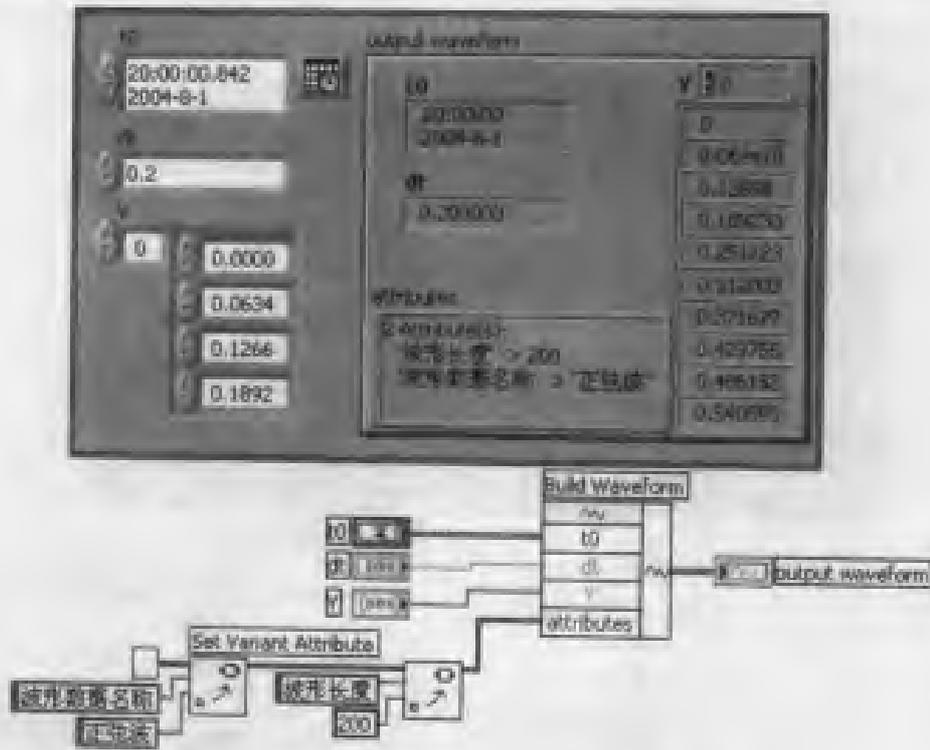


图 5.5.12 例 5.5.4 的前面板和框图程序

在一般情况下,只利用节点的前三个端口(t0, dt 和 Y)组成波形数据就可以了。另外,可以利用该节点和 Set Variant Attribute 节点修改或添加波形数据的属性,请看下面的例子。

例 5.5.5 利用 Build Waveform 节点和 Set Variant Attribute 节点修改并添加模拟波形数据的属性。

首先,利用 Set Variant Attribute 节点创建一个具有两个属性的变体数据,这两个属性见表 5.5.1。

表 5.5.1 Set Variant Attribute 节点创建的两个属性

属性名称	属性值	数据类型
波形数据名称	正弦波	字符串
波形长度	200	无符号 32 位整数

然后,将这个变体数据输入到 Build Waveform 节点的 attributes 端口,由于波形数据 Waveform 1 的属性中已经有一个名为“波形数据名称”的属性,故 Build Waveform 节点将这个属性的值修改为“正弦波”,并在 Waveform 1 的属性中添加一个新的属性“波形长度”。本例的 VI 的前面板和框图程序如图 5.5.13 所示。

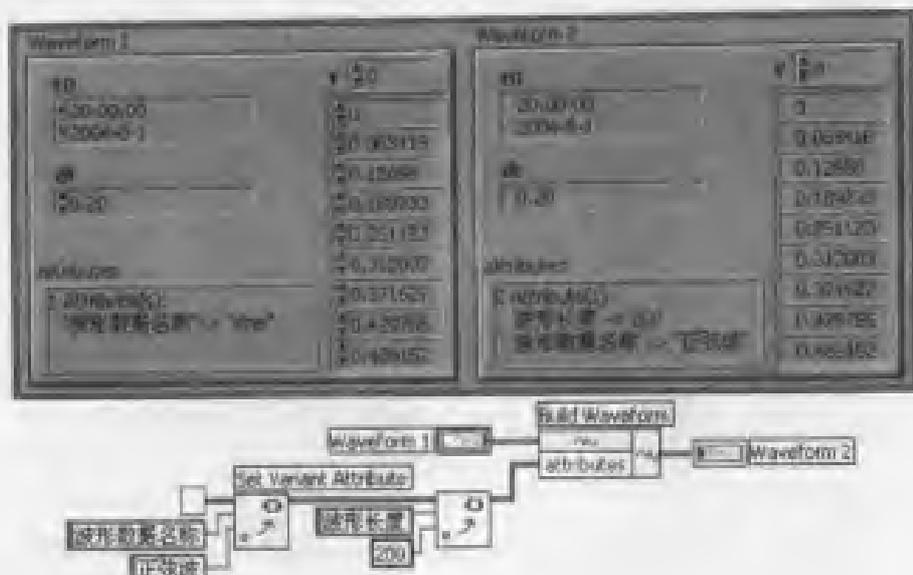


图 5.5.13 例 5.5.5 的前面板和框图程序

另外，LabVIEW 还有一个专门用于修改或添加波形数据属性的节点——Set Waveform Attribute 节点，可以利用该节点实现例 5.5.5 的功能，请见例 5.5.6。

3. Set Waveform Attribute

为波形数据添加或修改属性，节点的图标及其端口定义如图 5.5.14 所示。由于波形数据属性的数据类型为变体型，用户可以使用任何类型的数据作为属性值。

当由 name 端口指定的属性名称已经在波形数据的属性中存在时，节点将根据 value 端口的输入修改这个属性，同时 replaced 端口返回 True。

当由 name 端口指定的属性名称不存在时，节点将根据这个名称以及 value 端口输入的属性值为波形数据添加一个新属性，同时 replaced 端口返回 False。



图 5.5.14 Set Waveform Attribute 节点的图标及其端口定义

例 5.5.6 修改并添加波形数据的属性。

本例采用了两个 Set Waveform Attribute 节点。

首先，利用第一个 Set Waveform Attribute 节点将波形数据中的属性“波形数据名称”由“sine”修改为“正弦波”，其数据类型为字符串。

然后，利用第二个 Set Waveform Attribute 节点添加一个属性“波形长度”，属性值为 200，其数据类型为无符号 32 位整数。

VI 的前面板和框图程序如图 5.5.15 所示。



图 5.5.15 例 5.5.6 的前面板和框图程序

4. Get Waveform Attribute

获得波形数据属性中的属性名称和相对应的属性值，节点的图标及其端口定义如图 5.5.16 所示。节点的输出端口有两种模式：

(1) 模式一

name 端口输入一个属性名称时，节点的输出端口如图 5.5.16 中的 (a) 所示。

若节点从波形数据的属性中找到了 name 端口输入的属性名称，则从 found 端口返回 True，并从 value 端口返回该属性的属性值，返回值的数据类型为变体型，需要利用 Variant To Data 节点将其转化为属性值所对应的数据类型的数据之后，才可以在 LabVIEW 中使用和处理。

若节点从波形数据的属性中没有找到 name 端口输入的属性名称，则从 found 端口返回 False，value 端口返回值为空。

(2) 模式二

name 端口没有任何输入时，节点的输出端口如图 5.5.16 (b) 所示。



图 5.5.16 Get Waveform Attribute 节点的图标及其端口定义

节点从 names 端口返回一个字符串数组，数组中的每一个元素对应一个属性名称；

从 values 端口返回一个变体型数组，数组中的每一个元素对应一个属性值，同样需要利用 Variant To Data 节点将数组中的每一个元素转化为属性值所对应的数据类型的数据之后，才可以在 LabVIEW 中使用和处理。

例 5.5.7 获得波形数据的属性。

本例利用两个 Get Waveform Attribute 节点，分别用于获得波形数据中“波形长度”和“波形数据名称”两个属性的属性值，并利用 Variant To Data 节点将属性值转化为相应的数据类型无符号 32 位整数和字符串。VI 的前面板和框图程序如图 5.5.17 所示。

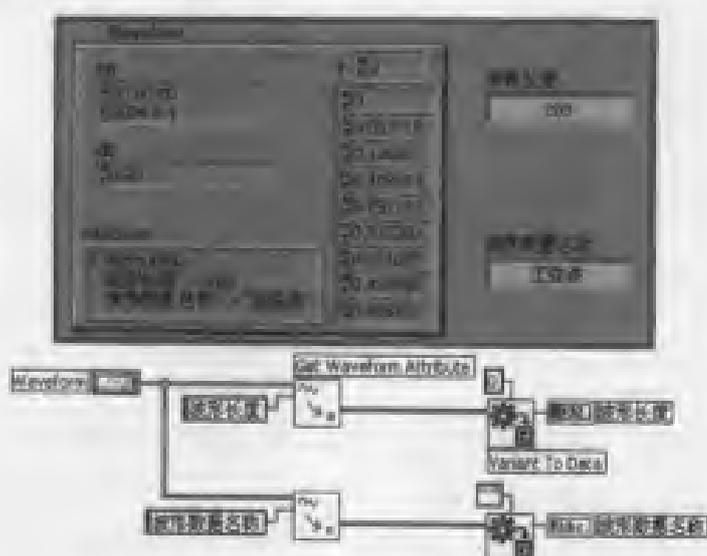


图 5.5.17 例 5.5.7 的前面板和框图程序

5. Analog to Digital Waveform

将模拟波形数据转换为数字波形数据，节点的图标及其端口定义如图 5.5.18 所示。

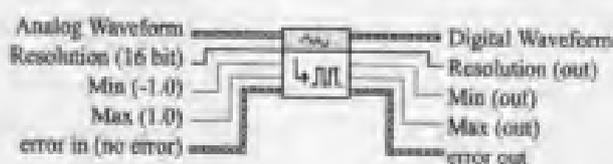


图 5.5.18 Analog to Digital Waveform 节点的图标及其端口定义

数字波形数据中 Y 的每一行对应模拟波形数据中 Y 的每一个元素。Resolution 端口确定了数字波形数据中 Y 的每一行中的二进制数据的位数，节点根据 Resolution, Min 和 Max 将模拟数字转换为二进制数字，转换关系如下。

按照 Resolution 端口指定的二进制数字位数 N ，将 Min 和 Max 之间均分为 N 个台阶，每个台阶的宽度为 $\frac{\text{Max} - \text{Min}}{2^N - 1}$ ，根据模拟数字所处的台阶位置，将其转换为相对应的二进制数。

例如，Resolution = 3，Max = -7，Min = 7，则台阶的宽度为 $\frac{7 - (-7)}{2^3 - 1} = 2$ ，对应关系如表 5.5.2 所示。

表 5.5.2 模拟数字和二进制数字的对应关系表

二进制数字	000	001	010	011	100	101	110	111
对应的台阶	$[-\infty, -5)$	$[-5, -3)$	$[-3, -1)$	$[-1, 1)$	$[1, 3)$	$[3, 5)$	$[5, 7)$	$[7, \infty)$

当模拟数字为 5.5 时, 位于台阶 $[5, 7)$ 之内, 则 5.5 对应的二进制数字为 110; 当模拟数字为 -100 时, 位于台阶 $[-\infty, -5)$ 之内, 则 5.5 对应的二进制数字为 000。

6. Digital to Analog Waveform

将数字波形数据转换为模拟波形数据, 节点的图标及其端口定义如图 5.5.19 所示。

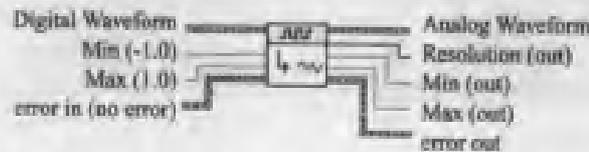


图 5.5.19 Digital to Analog Waveform 节点的图标及其端口定义

二进制数字与模拟数字的转换关系如下。

根据 Digital Waveform 中每一行二进制数字的位数 N , 将 Min 和 Max 之间均分为 N 个台阶, 每个二进制数字对应一个台阶值。图 5.5.20 所示的 Digital Waveform 中的每一行二进制数字的位数 N 为 3。

例如, 二进制数字的位数 $N=3$, $\text{Max}=-7$, $\text{Min}=7$, 则台阶为 2, 对应关系如表 5.5.3 所示。

表 5.5.3 模拟数字和二进制数字的对应关系表

二进制数字	000	001	010	011	100	101	110	111
对应的台阶值	-7	-5	-3	-1	1	3	5	7

图 5.5.20 所示的是按照上述方法将一个 Digital Waveform 转换为相对应的 Analog Waveform。



图 5.5.20 Digital Waveform 转换为 Analog Waveform

7. Index Waveform Array

从一个波形数据数组中取出一个由 index 端口指定的波形数据, 节点的图标及其端口定义如图 5.5.21 所示。

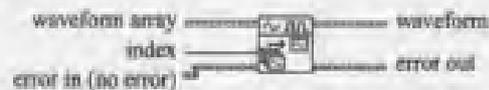


图 5.5.21 Index Waveform Array 节点的图标及其端口定义

Index 端口可以有以下两种输入方式。

① 输入一个数字。

数字表示波形数据数组的索引，当采用这种输入方式时，该节点的功能与 Index Array 节点的功能相同。

② 输入一个字符串。

字符串表示波形数据属性中的通道名称 (NI_ChannelName)，采用这种输入方式时，节点可以按照波形数据属性中的 NI_ChannelName 来搜索波形数据。

第二种方式更加直接。当采用 NI 的 DAQ 设备进行多通道数据采集，并且为每一个通道都指定一个通道名称时，利用这种方法可以直接取出某一个通道所返回的波形数据。请看下面的例子。

例 5.5.8 以通道名称的方式从一个波形数据数组中取出一个指定的波形数据。

本例提供了一个波形数据数组，数组中包含以下 3 个元素：

- ① 第一个元素包含一个正弦波；
- ② 第二个元素包含一个方波；
- ③ 第三个元素包含一个三角波。

这三个波形数据的属性“NI_ChannelName”被分别命名为“正弦波”、“方波”、“三角波”，在 Index Waveform Array 节点的 Index 端口直接输入通道名称“正弦波”，就可以将数组中的第一个元素取出，并放到前面板对象 Waveform Graph 中显示。VI 的前面板和框图程序如图 5.5.22 所示。

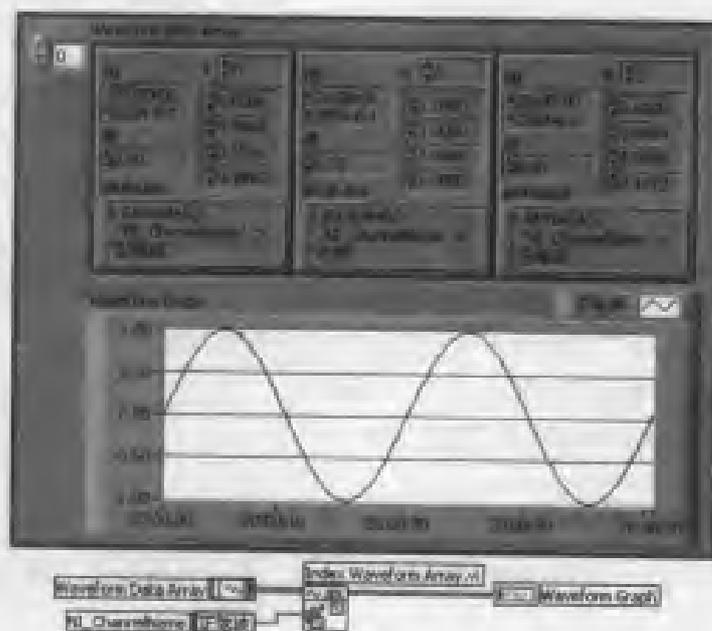


图 5.5.22 例 5.5.8 的前面板和框图程序

8. Copy Waveform dt

用 index 端口指定的波形数据中的 dt 替换波形数据数组中所有元素中的 dt，节点的图标及其端口定义如图 5.5.23 所示。

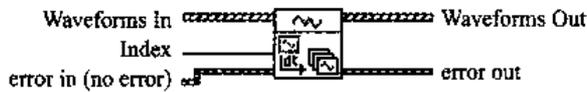


图 5.5.23 Copy Waveform dt 节点的图标及其端口定义

9. Align Waveform Timestamps

用 index 端口指定的波形数据中的 t0 替换波形数据数组中所有元素中的 t0，节点的图标及其端口定义如图 5.5.24 所示。

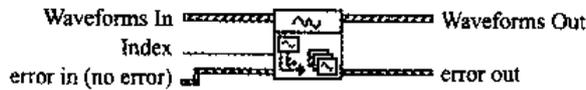


图 5.5.24 Align Waveform Timestamps 节点的图标及其端口定义

10. Get Waveform Subset

从一个波形数据中指定的位置取出指定长度的一段子波形，节点的图标及其端口定义如图 5.5.25 所示。

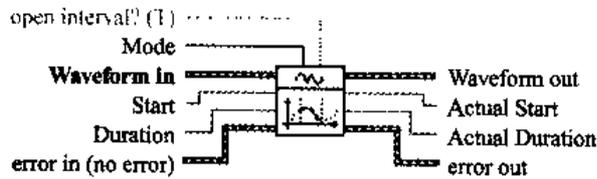


图 5.5.25 Get Waveform Subset 节点的图标及其端口定义

Start 端口用于指定子波形的起始位置；Duration 端口用于指定子波形的长度；Mode 端口用于指定取出子波形时采用的模式；Index 模式表示按照数组 Y 中元素的索引取出数据，Relative Time 模式表示按照波形中数据的相对时间取出数据。

例 5.5.9 从一个波形数据中取出一段子波形。

VI 的前面板和框图程序如图 5.5.26 所示。

11. Get Final Time Value

返回波形数据中最后一个点的所对应的时间 *tf*，节点的图标及其端口定义如图 5.5.27 所示。

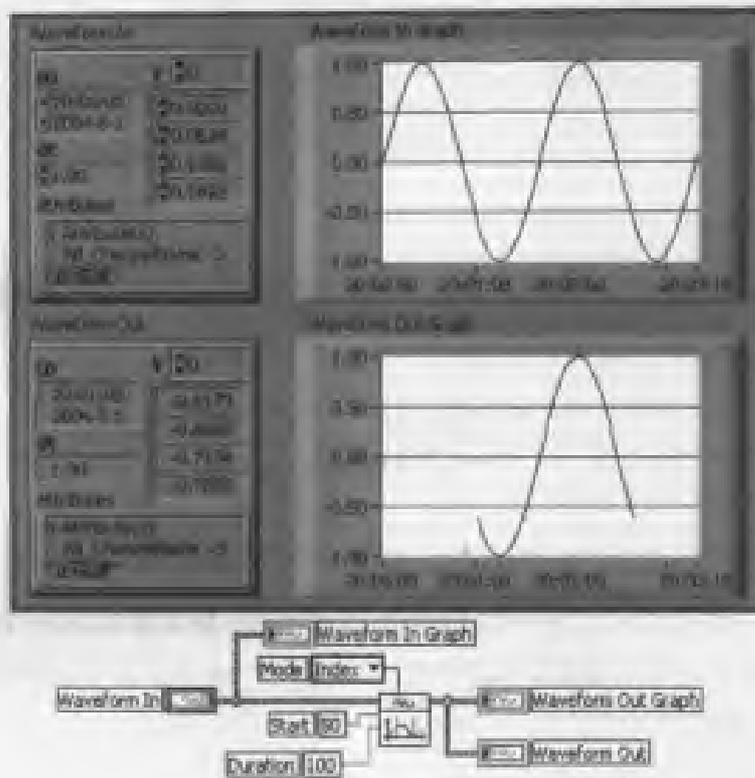


图 5.5.26 例 5.5.9 的前面板和框图程序



图 5.5.27 Get Final Time Value 节点的图标及其端口定义

当 open interval? 端口的输入值为 True 时, $t_f = t_0 + dt \times (N - 1)$; 当 open interval? 端口的输入值为 False 时, $t_f = t_0 + dt \times N$, 式中 N 为波形数据的长度, 即数组 Y 中元素的个数。

12. Waveform Duration

返回波形数据所占的时间段的长度 ΔT , 节点的图标及其端口定义如图 5.5.28 所示。 $\Delta T = dt \times (N - 1)$, 式中 N 为波形数据的长度。



图 5.5.28 Waveform Duration 节点的图标及其端口定义

13. Scale Delta t

将波形数据中的 dt 乘以一个由 scale factor 端口输入的比例因子后, 替换原来的 dt , 节点的图标及其端口定义如图 5.5.29 所示。

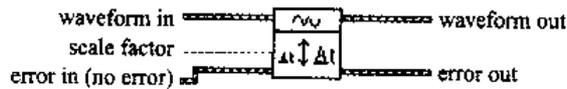


图 5.5.29 Waveform Duration 节点的图标及其端口定义

14. Get Y Value

取出波形数据中数组 Y 中的一个指定的元素值，节点的图标及其端口定义如图 5.5.30 所示。

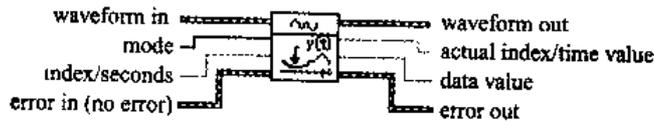


图 5.5.30 Get Y Value 节点的图标及其端口定义

mode 端口用于指定取出 Y 中元素时采用的模式: Index 模式或 Relative Time 模式。Index 模式表示按照数组 Y 中元素的索引取出数据，Relative Time 模式表示按照波形中数据的相对时间取出数据。index/seconds 端口用于指定所取元素的位置。

15. Get Waveform Time Array

将波形数据中每一个数据点所对应的时间值组成一个数组，并通过 X array 端口返回，节点的图标及其端口定义如图 5.5.31 所示。

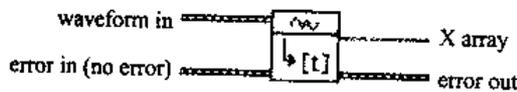


图 5.5.31 Get Waveform Time Array 节点的图标及其端口定义

X array 端口返回的是一个一维数组，其数据类型是双精度浮点数，数组中的元素表示的是时间，单位为秒。例如，时间“2004 年 8 月 1 日 20 点 0 分 0 秒”转化为双精度浮点数就是 3174206400.00s。

5.5.3 波形数据的特点

波形数据是 LabVIEW 特有的一类数据类型，实际上波形数据是一个特殊的簇，但是，用户不能利用 Cluster 子模板中的簇运算节点来处理波形数据。波形数据的引入，可以为测试数据的处理带来极大的方便，若能够有效的使用 LabVIEW 提供的波形数据运算节点，可以为用户节省大量的编程时间，并能够开发出功能强大的测试和数据处理软件。

第6章 结构与属性

LabVIEW 采用结构化数据流图编程，能够处理循环、顺序、条件和事件等程序控制的结构框架，这是 LabVIEW 编程的核心，也是区别于其他图形化编程开发环境的独特与灵活之处。LabVIEW 提供的结构定义简单直观，但应用变换灵活，形式多种多样，要完全掌握并灵活运用并不是一件易事。本章对 LabVIEW 各种结构框架的定义、使用和主要语法规则进行了比较详细的论述，并配以大量应用实例，介绍了各种结构的特殊用法及容易引起混淆的地方，期望能够帮助用户更好地理解 and 掌握这些结构的编程要点。

属性节点也是 LabVIEW 为增强图形化编程功能而设置的前面板对象特征，灵活运用属性节点控制是实现 LabVIEW 程序许多高级人机交互功能的主要技术途径，本章还将对属性节点的创建和前面板对象通用的属性特点进行介绍。

6.1 For 循环

LabVIEW 提供的结构位于功能模板→All Functions 模板→Structures 子模板中，如图 6.1.1 所示。



图 6.1.1 LabVIEW 的 Structures 子模板

For 循环 (For Loop) 是 LabVIEW 最基本的结构之一，它执行指定次数的循环，相当于 C 语言中的 for 循环。

```

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

LabVIEW 中的 For 循环可从 Structures 子模板中创建, 如图 6.1.2 所示。



图 6.1.2 创建 For 循环

6.1.1 For 循环的组成

最基本的 For 循环由循环框架 (Loop Frame)、重复端口 (Loop Iteration) 和计数端口 (Loop Count) 组成, 如图 6.1.3 所示。

For 循环执行的是包含在循环框架内的程序。其重复端口相当于 C 语言 For 循环中的 i , 初始值为 0, 每次循环的递增步长为 1。注意, 重复端口的初始值和步长在 LabVIEW 中是固定不变的, 若要用到不同的初始值或步长, 可对重复端口产生的数据进行一定的数据运算, 也可用移位寄存器来实现。具体用法见例 6.1.2。其计数端口相当于 C 语言 For 循环中的循环次数 N , 在程序运行前必须赋值, 如图 6.1.3 所示。通常情况下, 该值为整型数字, 若将其他数字类型连接到该端口上, For 循环会自动将其转化为整型。

另外, 为实现 For 循环的各种功能, LabVIEW 在 For 循环中引入了移位寄存器 (Shift Register) 和框架通道 (Loop Tunnel) 两个独具特色的新概念, 如图 6.1.4 所示。



图 6.1.3 For 循环的组成

图 6.1.4 移位寄存器和框架通道

移位寄存器的功能是将第 $i-1, i-2, i-3 \dots$ 次循环的计算结果保存在 For 循环的缓冲区内, 并在第 i 次循环时将这些数据从循环框架左侧的移位寄存器中送出, 供循环框架内的节点使用, 其中, $i=0, 1, 2, 3 \dots$ 在循环框架上的右键弹出菜单中选择 Add Shift Register, 可创建一个移位寄存器, 如图 6.1.5 所示。



图 6.1.5 创建移位寄存器

用鼠标（定位工具状态）在左侧移位寄存器的右下角向下拖动，或在左侧移位寄存器的右键弹出菜单中选择 Add Element，可创建多个左侧移位寄存器，如图 6.1.6 所示。



图 6.1.6 创建多个左侧移位寄存器

此时，在第 i 次循环开始时，左侧每一个移位寄存器便会将前几次循环由右侧移位寄存器存储到缓冲区的数据送出来，供循环框架内的各种节点使用。左侧第 1 个移位寄存器送出的是第 $i-1$ 次循环时存储的数据，第 2 个移位寄存器送出的是第 $i-2$ 次循环时存储的数据，第 3 个、第 4 个……移位寄存器送出的数据，依次类推。数据在移位寄存器中的流动过程如图 6.1.7 所示，具体用法见例 6.1.1。



图 6.1.7 数据在移位寄存器中的流动过程

当 For 循环在执行第 0 次循环时，For 循环的数据缓冲区并没有数据存储，所以，在使用移位寄存器时，必须根据编程需要对左侧的移位寄存器进行初始化，如图 6.1.4 所示。否则，左侧的移位寄存器在第 0 次循环时的输出值为默认值，数字的默认值为 0；字符串

的默认值为空字符；布尔数据的默认值为 False。另外，连至右侧移位寄存器的数据类型和用于初始化左侧移位寄存器的数据类型必须一致，例如都是数字型，或都是字符串型、布尔型等。

注意，左侧移位寄存器除了初始化时可以输入数据外，其他情况下只能输出数据；而右侧移位寄存器除了在循环结束时向循环外输出数据，其他情况下只能输入数据。

框架通道是 For 循环与循环外部进行数据交换的数据通道，其功能是在 For 循环开始运行前，将循环外其他节点产生的数据送至循环内，供循环框架内的节点使用，还可在 For 循环运行结束时，将循环框架内节点产生的数据送至循环外，供循环外的其他节点使用。用连线工具将数据连线从循环框架内直接拖至循环框架外，LabVIEW 会自动生成一个框架通道。框架通道共有两种属性：有索引 (Enable Indexing) 和无索引 (Disable Indexing)，分别用于传递数组和标量，其具体用法见例 6.1.3。

6.1.2 For 循环的使用

例 6.1.1 求 $n!$ 。

前面板和程序框图如图 6.1.8 所示。

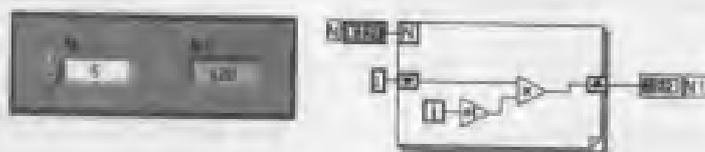


图 6.1.8 例 6.1.1 的前面板和框图程序

该程序相当于 C 语言中的下面一段程序：

```
void main ()
{
    int a, i, n;
    a=1;
    scanf ("%d", &n);
    for(i=0; i<n; i++)
    {
        i=i+1;
        a=a*i;
    }
    printf("n!=%d", a);
}
```

比较这两个程序可以看出，LabVIEW For 循环中的移位寄存器就相当于 C 语言程序中的整型变量 a。

当例 6.1.1 中的 $n=5$ 时，For 循环共循环 5 次，表 6.1.1 列出了每一次循环时从 For 循环的计数端口 i 输出的数据，从左侧移位寄存器输出的数据和从右侧移位寄存器输入的数据。

表 6.1.1 For 循环每一次运行各端口的数据变化

循环次数	计数端口 i 输出的数据	$i+1$	左侧移位寄存器 输出的数据	右侧移位寄存器 输入的数据
第 1 次	0	1	1	1
第 2 次	1	2	1	2
第 3 次	2	3	2	6
第 4 次	3	4	6	24
第 5 次	4	5	24	120

另外，从 LabVIEW 7 Express 版本开始，LabVIEW 新提供了一个反馈节点 (Feedback Node)，可以实现移位寄存器的功能，如图 6.1.9 所示。

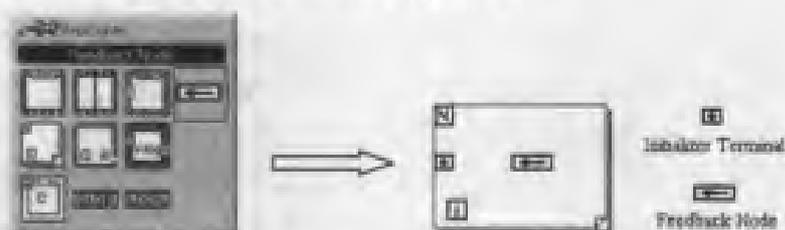


图 6.1.9 反馈节点

创建反馈节点时，必须将该节点放至 For 循环或 While 循环的框架内部，一个新的反馈节点包含两部分，初始化端口和反馈节点本身，初始化端口用于初始化反馈节点的初始值，反馈节点可以替代移位寄存器，例 6.1.1 可以使用反馈节点来实现，其前面板和框图程序如图 6.1.10 所示。

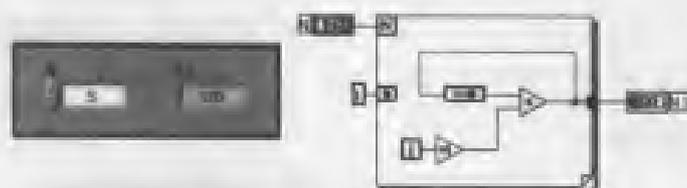


图 6.1.10 使用反馈节点实现例 6.1.1

例 6.1.2 求 0 到 99 之间所有偶数的和。

本例所用 For 循环的递增变量的步长为 2，可用两种方式来完成这个例子，如图 6.1.11 所示。

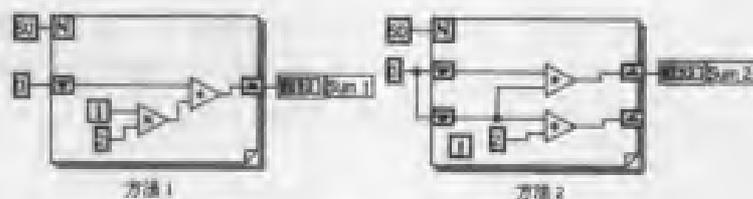


图 6.1.11 例 6.1.2 的框图程序

可以看出，方法 1 直接将重复端口产生的数据乘以 2，即把步长由 1 变为 2，方法 2 采用一个移位寄存器作为递增变量，方法 1 较为简单，方法 2 较为灵活，在实际编程应用

中, 具体采用哪一种方法, 可根据编程者的编程习惯和风格来决定。

例 6.1.3 用 For 循环产生一个长度为 5 的随机数组。

本例主要介绍 For 循环中框架通道的两种属性的不同用法。框图程序和程序运行结果见图 6.1.12。

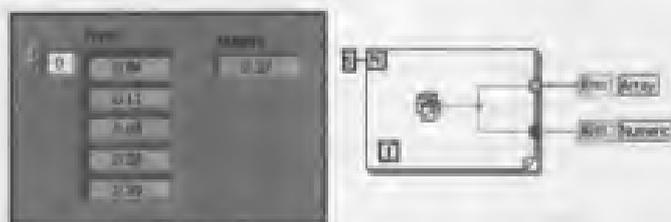


图 6.1.12 例 6.1.3 的前面板和框图程序

从图 6.1.12 可以看出, For 循环产生的随机数经过两个数据通道分别送到了数组指示 (Array) 和一个数字指示 (Numeric) 中, 在数组指示中得到的是正确的结果——一个长度为 5 的随机数组, 在数字指示中得到的仅是一个数 (标量), 并且这个数与随机数组中的最后一个元素相同。显然, 这不是正确结果。这两个框架通道所得结果不同, 原因在于两个框架通道所设置的属性不同, 第一个框架通道的属性为 Enable Indexing, 而第二个框架通道的属性为 Disable Indexing。当框架通道的属性为 Enable Indexing 时, 该框架通道就成为一个数据缓存, 每一次循环时通过数据连线送来的数据在该框架通道中按照先后次序组成一个数组, 循环结束时, 框架通道将这个数组送出; 当框架通道的属性为 Disable Indexing 时, 它只会接受最后一次循环时通过数据连线送来的数据, 并在循环结束时将其送出。

在框架通道的右键选中可设置框架通道的这两种属性, 见图 6.1.13。注意, 框架通道的属性为 Enable Indexing 时, 其外观为一个空心的矩形 □; 框架通道的属性为 Disable Indexing 时, 其外观为一个实心的矩形 ■。



图 6.1.13 设置框架通道的属性

另外, For 循环还有一种很有特色的功能, 称之为自动索引 (Auto Indexing) 功能, 当将一个数组连接到 For 循环上供 For 循环内的节点使用时, For 循环可以自动检测该数组的长度, 执行相应次数的循环, 并按顺序将数组内的元素一一取出。

例 6.1.4 求一个一维数组中所有元素的和。

若用 For 循环来完成本例, 共有两种方法。

第一种方法是常规方法, 首先用 Functions 模板 → All Functions 模板 → Array 子模板中的 Array Size 节点获得数组的长度, 以确定 For 循环的循环次数, 然后再用 Array 子模板中的 Index Array 节点将该数组中的各个元素一一取出, 然后相加, 最后得到结果。框图程

序如图 6.1.14 所示。注意，此时 For 循环框架通道的属性为 Disable Indexing。

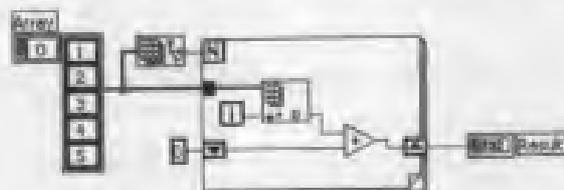


图 6.1.14 例 6.1.4 方法一的框图程序

第二种方法是利用 For 循环的自动索引功能，框图程序如图 6.1.15 所示。与第一种方法不同，这个 For 循环框架通道的属性设为 Enable Indexing，且没有给 For 循环的计数端口赋值。但此时 For 循环具有了自动索引功能，在循环执行时，For 循环自动检测数组的长度，并在每一次循环时将数组中的元素按顺序——取出，然后相加。

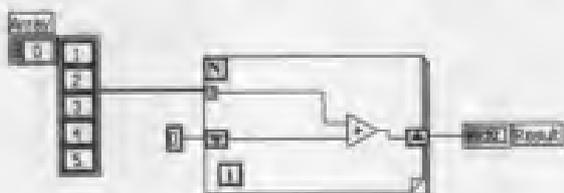


图 6.1.15 例 6.1.4 方法二的框图程序

比较两框图程序可以看出，方法二的框图程序比方法一的框图程序要简单，采用方法二可以大大提高编程效率。

在使用 For 循环的自动索引功能时，有一点值得注意的是，若有两个不同长度的数组同时连接到 For 循环上，并且其框架通道的属性都为 Enable Indexing，给 For 循环的计数端口赋一个与两数组长度不同的值时，For 循环会按最少的循环次数执行。

例 6.1.5 求两个一维数组中所有元素的和。

VI 的前面板和框图程序如图 6.1.16 所示。计数端口的赋值为 3，Array 1 的长度为 4，Array 2 的长度为 5，则此循环只会执行 3 次。最后的计算结果为：Result 1=6；Result 2=18。

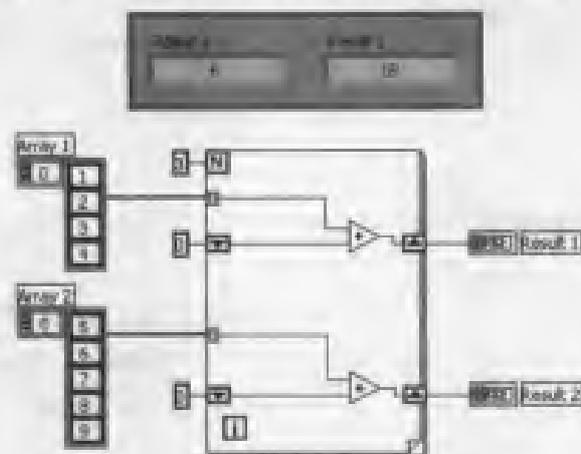


图 6.1.16 例 6.1.5 的的前面板和框图程序

6.1.3 For 循环的特点

与其他编程语言相比, LabVIEW 中的 For 循环除具有一般 For 循环共有的特点之外, 还具有一些一般 For 循环没有的独特之处。

LabVIEW 没有类似于其他编程语言中的 goto 之类的转移语句, 故不能随心所欲地将程序从一个正在执行的 For 循环中跳转出去, 也就是说, 一旦确定了 For 循环执行的次数, 并开始执行后, 就必须在执行完相应次数的循环后, 才能终止其运行。若确实需要根据某种逻辑条件跳出循环, 可用 While 循环来替代 For 循环, While 循环的具体用法详见下一节。

6.2 While 循环

当循环次数不能预先确定时, 就需用到 While 循环 (While Loop)。While 循环也是 LabVIEW 最基本的结构之一, 相当于 C 语言中的 while 循环和 do 循环。

```
while(条件)
{
}

do
{
}while(条件)
```

While 循环可从框图程序中的 Structures 子模板中创建, 创建方法如图 6.2.1 所示。

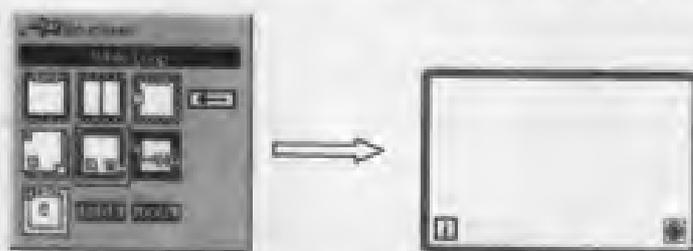


图 6.2.1 创建 While 循环

6.2.1 While 循环的组成



图 6.2.2 While 循环的组成

最基本的 While 循环由循环框架 (Loop Frame), 重复端口 (Loop Iteration), 以及条件端口 (Loop Condition) 组成, 如图 6.2.2 所示。

与 For 循环类似, while 循环执行的是包含在循环框架中的程序, 但执行的循环次数

却不固定，只有当满足给定的条件时，才停止循环的执行。

重复端口的功能与用法与 For 循环的重复端口相同。

条件端口用于控制循环是否继续执行，条件端口有两种使用状态：Stop if True  和 Continue if True 。当每一次循环结束时，条件端口便会检测通过数据连线输入的布尔值，并根据输入的布尔值和其使用状态决定是否继续执行循环。

- 当条件端口的使用状态为 Stop if True 时，若输入值为 True，则停止执行循环；若输入值为 False，则继续执行下一次循环。

- 当条件端口的使用状态为 Continue if True 时，若输入值为 True，则继续执行下一次循环；若输入值为 False，则停止执行循环。

用鼠标（数据操作工具状态）单击条件端口的图标，或者在条件端口的右键弹出选单中选择 Stop if True 或 Continue if True，可以切换条件端口的使用状态。

While 循环也有框架通道和移位寄存器，其用法以及反馈节点在 While 循环中的用法与 For 循环完全相同，在此不再赘述。

6.2.2 While 循环的使用

例 6.2.1 求 $n!$ 。

前面板和框图程序如图 6.2.3 所示。

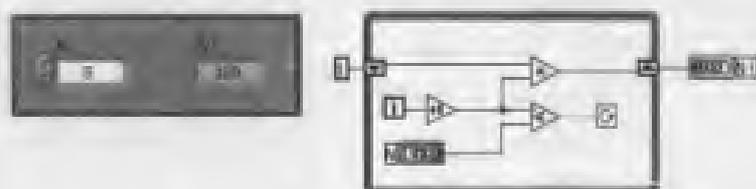


图 6.2.3 例 6.2.1 的前面板和框图程序

该程序相当于 C 语言中的下面一段程序：

```
void main ()
{
    int a, i, n;
    a=1;
    i=0;
    scanf("%d", &n);
    do
    {
        i=i+1;
        a=a*i;
    }while(i<n);
    printf("n!=%d", a);
}
```

例 6.2.2 设计一个简单的平均数滤波器，并对一随机数波形进行滤波。

本例首先在 While 循环框架中产生一个随机数，然后将这个随机数与前三次循环所产生的随机数求平均值，最后将平均值送到前面板上显示。利用移位寄存器可得到前三次循环产生的随机数。前面板和框图程序如图 6.2.4 所示。



图 6.2.4 例 6.2.2 的前面板和框图程序

图 6.2.4 中前面板上名为 Waveform Chart 的实时波形记录图用于实时显示在 While 循环中得到的滤波后的随机波形；名为“stop”的布尔按钮用于控制循环是否继续执行，当按钮在“OFF”状态时（即图 6.2.4 中所示状态），其布尔值为 False，每一次循环结束时，都会产生一个值为 False 的布尔值，这个 False 值将通过数据连线送至 While 循环的条件端口，条件端口（其使用状态为 Stop if True）收到 False 值后，控制循环继续执行。若用鼠标单击“stop”按钮，此时其布尔值为 True，该次循环结束时此值将被送至条件端口，条件端口终止 While 循环的执行，退出程序。

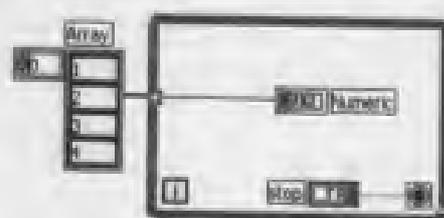


图 6.2.5 While 循环的自动索引功能

While 循环也具有自动索引的功能，如图 6.2.5 所示。当一个数组连接到 While 循环框架上时，将框架通道的属性设为 Enable Indexing，While 循环在运行时就会逐一将数组中的元素按顺序取出。由于 While 循环的循环次数不像 For 循环那样是预先确定的，因而当循环的次数超过数组的长度时，框架通道就取不到数组中的元素，此时框架通道的输出就变为数组元素的默认值。

6.2.3 While 循环的特点

与 For 循环类似，LabVIEW 中的 While 循环与其他编程语言相比也独具特色。由于 While 循环的执行是由条件端口来控制的，所以，编程时如果不注意，就可能会出现死循环。如图 6.2.6 所示，左侧的 While 循环中连接到条件端口（使用状态为 Continue if True）

上的的是一个布尔常量，其值为 True，程序运行时该值是固定不变的，故此 While 循环将永远执行下去；右侧的 While 循环中连接到条件端口的是一个条件判断（ $i \geq 0$ ？）的结果，由于 i 的值为 0, 1, 2...，故这个条件判断的结果永远为 True，While 循环将永远执行下去。要是编程时不注意而出现类似的逻辑错误，就会导致 While 循环出现死循环。



图 6.2.6 处于死循环状态下的 While 循环



图 6.2.7 添加了一个布尔按钮的 While 循环

所以，编程时要尽量避免上述情况的出现。编程时最好在前面板上临时添加一个布尔按钮，与逻辑控制条件相“与（And）”后再连至条件端口，如图 6.2.7 所示。这样，程序运行时一旦出现逻辑错误而导致死循环时，可通过这个布尔按钮强行结束程序的运行。在完成所有程序开发，经检验程序运行无误后，再将这个布尔按钮去掉。当然，当出现死循环时，通过窗口工具条上的停止按钮也可以强行终止程序的运行。

6.3 顺序结构

在传统编程语言中，程序有明确的顺序执行，即程序按照程序代码从上到下的顺序执行，每个时刻只执行一步。这种程序执行方式称为控制流（Control Flow），而 LabVIEW 却是一种数据流（Data Flow）语言，在 LabVIEW 中，只有当某个节点的所有输入均有效时，LabVIEW 才能执行该节点，这一点称为数据从属性（Data Dependency），如图 6.3.1 所示。



图 6.3.1 控制流编程与数据流编程

图 6.3.1 将一般的控制流编程（如 Basic 或 C）与数据流编程进行了比较。在控制流编程（左图）中，程序强制 GET A 在 GET B 之前执行，若数据 B 在数据 A 准备好之前已经准备好，程序就必须浪费时间去等待数据 A。而在数据流编程（右图）中 GET A 和 GET B 就没有前后之分，两个 GET 任务根据需要在时间上相互交叠。LabVIEW 允许在一个框图中并行执行多个不同的节点。也就是说 LabVIEW 环境支持并行执行多任务和多 VIs，从控制流编程发展到数据流编程是编程语言技术的革新，也是 LabVIEW 编程特性的显著标志之一。

虽然数据流编程为用户带来了许多方便，但也在某些方面存在不足。如果 LabVIEW 框图程序中有两个节点同时满足节点执行的条件，那么这两个节点就会同时执行。但如果编程者需要这两个节点按一定的先后顺序执行，那么数据流控制是无法满足要求的，必须引入特殊的结构框架，在此框架内程序要严格按照预先确定的顺序执行，这就是 LabVIEW 顺序结构（Sequence Structure）的由来。

LabVIEW 顺序结构的功能是强制程序按一定的顺序执行。顺序结构可从 Structures 子模板中创建, 如图 6.3.2 所示。

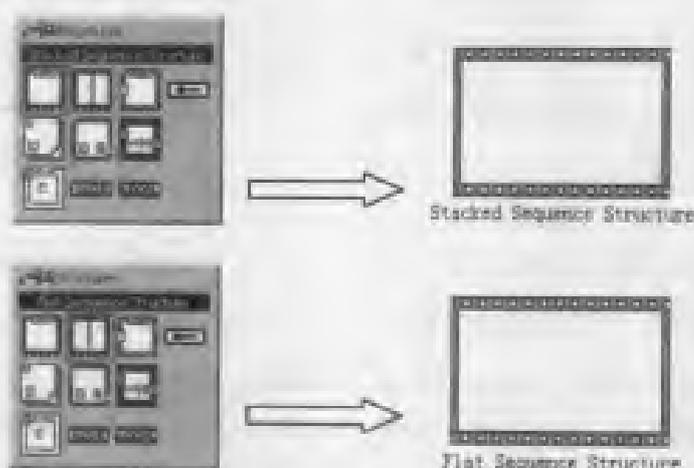


图 6.3.2 创建顺序结构

从图 6.3.2 可以看出, LabVIEW 提供了两种顺序结构: 层叠式顺序结构 (Stacked Sequence Structure) 与平铺式顺序结构 (Flat Sequence Structure)。层叠式顺序结构是 LabVIEW 中经典的顺序结构, 存在于 LabVIEW 的各个版本中, 而平铺式顺序结构是在 LabVIEW 7 Express 版本中新出现的。这两种顺序结构的功能完全相同, 只是其外观和用法稍有不同, 下面将详细介绍两种顺序结构的组成、使用和特点。

6.3.1 顺序结构的组成

LabVIEW 顺序结构看起来很像装在照相机内的胶卷, 是按照顺序一帧接一帧地拍照 (运行) 的。顺序结构共有两种: 单框架顺序结构和多框架顺序结构。

1. 层叠式顺序结构

最基本的层叠式顺序结构由顺序框架 (Sequence Frame)、选择器标签 (Selector Label) 和递增/递减按钮 (Increment/Decrement Buttons) 组成, 如图 6.3.3 所示。

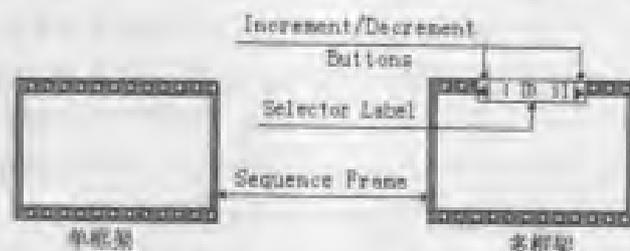


图 6.3.3 层叠式顺序结构的组成

按照上述方法创建的是单框架顺序结构, 只能执行一步操作。但大多数情况下, 用户需要按顺序执行多步操作, 因此需要在单框架的基础上创建多框架顺序结构。在顺序框架的右键弹出菜单中选择 Add Frame After 或 Add Frame Before, 就可添加框架, 如图 6.3.4 所示。

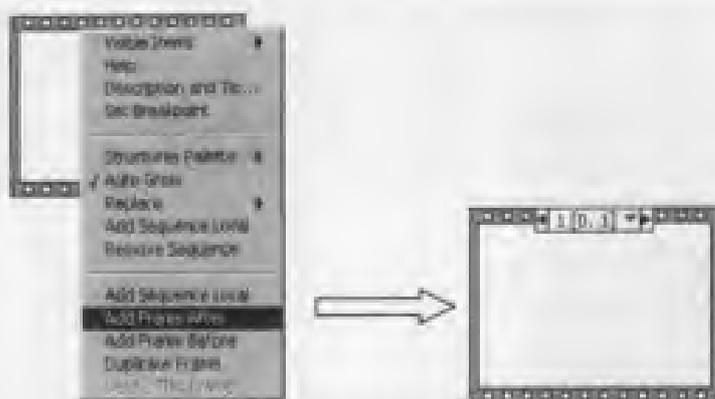


图 6.3.4 创建多框架顺序结构

程序运行时，顺序结构就会按选择器标签 0, 1, 2... 的顺序逐步执行各个框架中的内容。具体用法见例 6.3.1。在程序编辑状态时，用鼠标单击递增/递减按钮可将当前编号的顺序框架切换到前一编号或后一编号的顺序框架；用鼠标（操作工具）单击选择器标签，可从下拉选单中选择切换到任一编号的顺序框架。如图 6.3.5 所示。



图 6.3.5 顺序框架的切换

另外，在编程时，还常常需要将前一个顺序框架中产生的数据传递到后续顺序框架中使用，为此 LabVIEW 在顺序框架中引入了本地结果（Sequence Local）的概念，通过顺序框架本地结果，就可以在顺序框架中向后传递数据。其详细用法见例 6.3.1。为与顺序框架外部的程序节点进行数据交换，顺序结构中也存在框架通道（Frame Tunnel），但这里的框架通道没有 Enable Indexing 和 Disable Indexing 这两种属性。

2. 平铺式顺序结构

多框架层叠式顺序结构由多个框架组成，按照 0, 1, 2... 的顺序编号层叠在一起，并且按照 0, 1, 2... 的顺序执行。多框架平铺式顺序结构的多个框架不是层叠在一起，而是由左自右平铺，并且按照相同的顺序执行，如图 6.3.6 所示。添加框架的方法与层叠式顺序结构相同。



图 6.3.6 平铺式顺序结构的组成

从功能上讲,层叠式顺序结构和平铺式顺序结构完全相同。平铺式顺序结构的所有框架在同一个平面上,较为直观,不需要用户在框架之间切换,并且二者在用法上稍有区别。在顺序结构框架的右键菜单中选择 **Replace with Flat Sequence** 或 **Replace with Stacked Sequence** 可以在层叠式顺序结构和平铺式顺序结构之间相互切换,如图 6.3.7 所示。

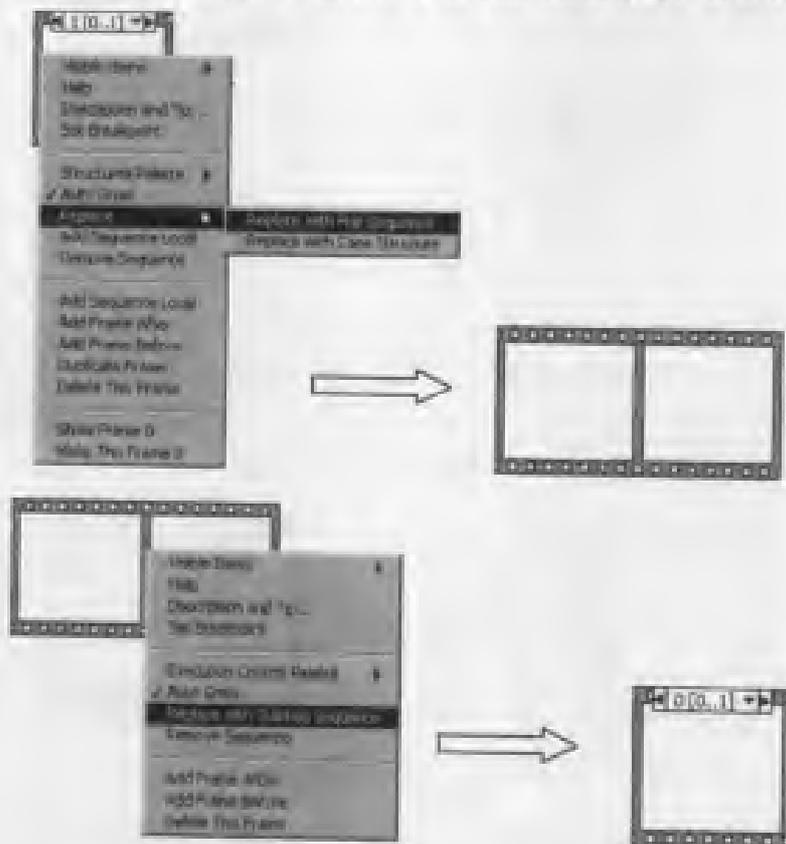


图 6.3.7 层叠式顺序结构和平铺式顺序结构之间相互切换

6.3.2 顺序结构的使用

1. 层叠式顺序结构的使用

例 6.3.1 用 For 循环产生一个长度为 2 000 点的随机波形,并计算所用时间。

本例是一个典型的顺序结构应用。第 1 步确定 For 循环开始前的系统时间;第 2 步运行 For 循环;第 3 步确定 For 循环结束后的系统时间;最后两时间相减即得 For 循环的运行时间。程序框图如图 6.3.8 所示。本例采用层叠式顺序结构实现。

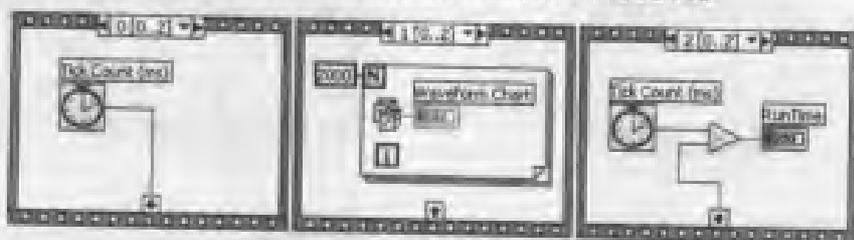


图 6.3.8 例 6.3.1 的框图程序

在顺序框架 0 和顺序框架 2 中，采用了一个名为 Tick Count (ms) 的节点，用于返回当前系统时间，单位为 ms。另外，为把在顺序框架 0 中获得的系统时间传递到顺序框架 2 中求时间差，本例采用了顺序结构本地变量 (Sequence Local)，顺序结构本地变量是层叠式顺序结构中特有的变量，用于向后序的顺序框架中传递数据。在层叠式顺序结构框架的右键弹出选单中选择 Add Sequence Local，可添加顺序结构本地变量，如图 6.3.9 所示。

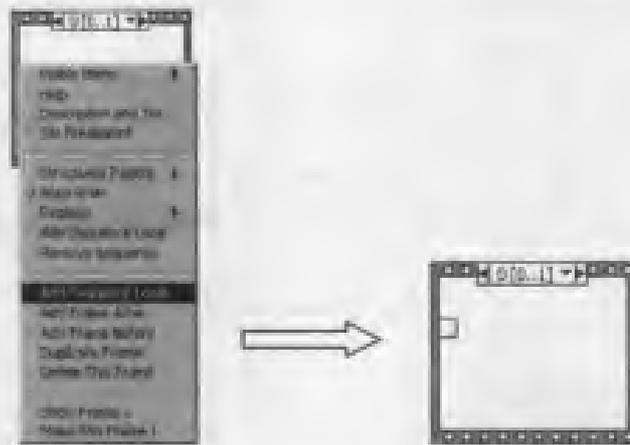


图 6.3.9 在层叠式顺序结构添加顺序结构本地变量

一个新的顺序结构本地变量是一个黄色的矩形，将需要传递的数据连接到顺序结构本地变量后，黄色的矩形会变为一个带箭头的矩形，在顺序框图 0 中，箭头指向顺序框架，在后序的顺序框架 1 中，箭头背向顺序框架，矩形的颜色由该数据测数据类型决定，如图 6.3.10 所示。注意，顺序结构本地变量只能向后续的顺序框架传递数据。



图 6.3.10 连接数据后的顺序结构本地变量

本例的计算结果如图 6.3.11 所示。

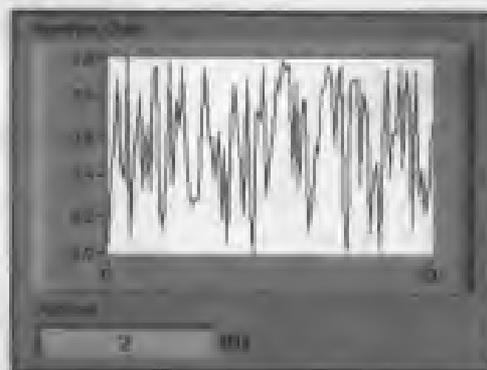


图 6.3.11 例 6.3.1 的前面板

2. 平铺式顺序结构的使用

例 6.3.2 利用平铺式顺序结构实现例 6.3.1 的功能。

利用平铺式顺序结构也可以实现例 6.3.1 的功能,其框图程序如图 6.3.12 所示。与层叠式顺序结构不同的是,平铺式顺序结构没有顺序结构本地变量,需要向后续的顺序框架传递数据时,只需要将数据直接连接到后续的顺序框架中即可。



图 6.3.12 例 6.3.2 的框图程序

6.3.3 顺序结构的特点

LabVIEW 顺序框架的使用比较灵活,在编辑状态时可以很容易地改变层叠式顺序结构各框架的顺序,如图 6.3.13 所示。平铺式顺序结构各框架的顺序不能改变,但可以先将平铺式顺序结构转换为层叠式顺序结构,在层叠式顺序结构中改变各框架的顺序后,将层叠式顺序结构转换为平铺式顺序结构,就可以改变平铺式顺序结构各框架的顺序。



图 6.3.13 改变层叠式顺序结构各框架的顺序

在了解了顺序结构的用法之后,再看下面的例子。

例 6.3.3 打开一个文件,从中读取数据,然后关闭这个文件。

本例共采用了 4 种方法来实现这 3 个操作,4 种方法的框图程序如图 6.3.14 所示。

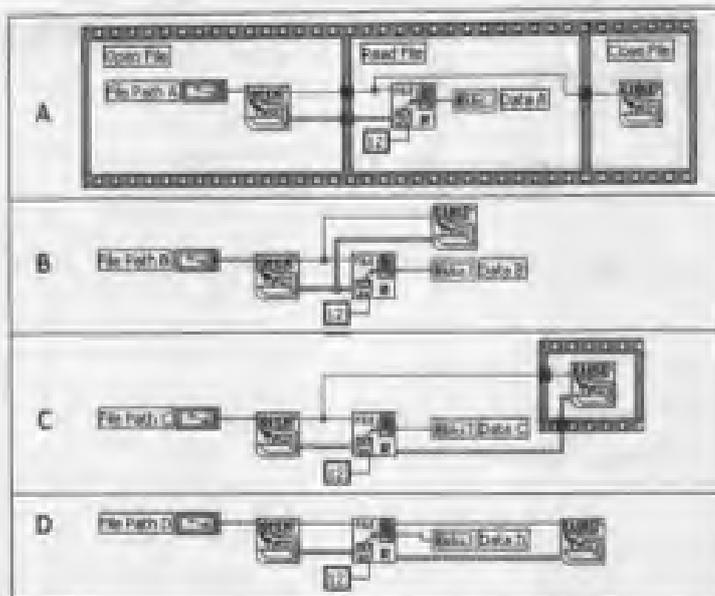


图 6.3.14 例 6.3.3 的框图程序

方法 A 用了一个顺序结构，这种方法没有采用数据流编程，但有效地完成了三步指定的操作。这种方法的主要缺点是用户必须采用一个顺序结构，编程较为烦琐。

方法 B 采用数据流编程方法，用数据连线将所有的功能操作按一定逻辑次序连接起来。方法 B 存在的问题是读取文件是否在关闭文件前执行？哪一种功能操作先执行？注意，当数据从属性不存在时，程序绝对不是按照从上至下，从左到右的顺序执行的。

方法 C 在读取文件和关闭文件之间创建了一个人为的数据从属性，把关闭文件功能放到了一个单框架的顺序结构中，再把读取文件功能的一个输出连接到顺序结构上。这不仅解决了方法 B 中存在的问题，而且程序也一目了然。

方法 D 是方法 C 的简化，是最佳的方法。

例 6.3.3 主要说明了如何根据顺序结构的特点灵活地使用顺序结构，同时也引出了一种新的编程思想——公共连线 (Common Threads)。实际上，方法 D 是利用 LabVIEW 的数据从属性，在框图程序中的节点之间建立了一个流程控制权 (Flow Control Right)。

在 LabVIEW 中，流程控制权的意思是将常用的 SubVIs 集中到一起，加一对公共输入输出端口，这样，不需要顺序结构就可将它们按照逻辑关系定义的前后次序连接到一起。Read File 节点的 File Refnum 端口就是这样一种端口，将其连接到 Close File 节点的输入端口 Refnum 上，就可以保证在完成读取文件的操作后才执行关闭文件的顺序操作。

另外补充一点，错误代码 (Error Code) 也是一种很好的公共连线，因为几乎所有的操作都可能会出现错误。每个 VI 都应该检测输入的错误代码，在发现错误时不执行它本身的功能，然后将错误 (或其本身的错误) 送至其错误输出端口中。LabVIEW 将错误代码、引起错误的节点的名称和用于标志错误状态的布尔值集中到一个叫做 Error Cluster 的簇 (Error Cluster 位于 Controls 模板 → All Controls 子模板 → Array & Cluster 子模板中，如图 6.3.15 所示) 中，作为公共连线。这种技术是

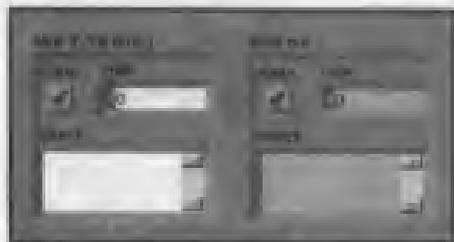


图 6.3.15 Error Cluster

一种通用的标准,称为 ERROR I/O。例 6.3.3 的方法 D 就采用了这种技术。建议用户在使用 LabVIEW 开发应用软件时尽量采纳这种思想。

6.4 选择结构

选择结构(Case Structure)也是 LabVIEW 最基本的结构之一,相当于 C 语言中的 switch 语句:

```
switch (表达式)
{
case 常量表达式 1: 语句 1;
case 常量表达式 2: 语句 2;
  ⋮
case 常量表达式 n: 语句 n;
default      : 语句 n+1;
}
```

在某种意义上,选择结构还相当于 C 语言中的 if 语句:

```
if (条件判断表达式)
{
}
else
{
}
```

C 语言中的 switch 语句选择结构可从 Structures 子模板中创建,创建方法如图 6.4.1 所示。

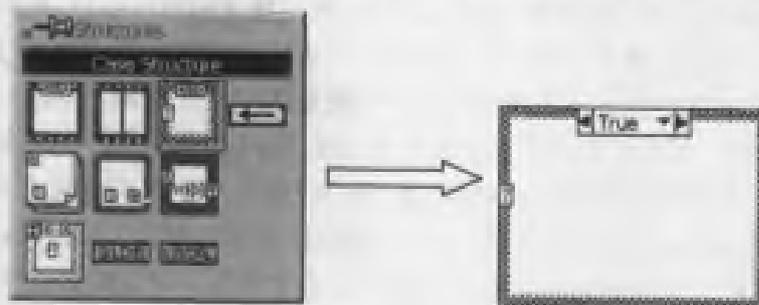


图 6.4.1 创建选择结构

6.4.1 选择结构的组成

最基本的选择结构由选择框架 (Case Frame)、选择端口 (Selection Terminal)、选择器标签 (Selector Label), 以及递增/递减按钮 (Increment/Decrement Buttons) 组成, 如图 6.4.2 所示。

在选择结构中，选择端口相当于上述 C 语言 Switch 语句中的“表达式”，框图表示符相当于“表达式 n”。编程时，将外部控制条件连接至选择端口上，程序运行时选择端口会判断送来的控制条件，引导选择结构执行相应框架中的内容。详细用法见例 6.4.1。

LabVIEW 中的选择结构与 C 语言 Switch 语句相比比较灵活，输入选择端口中的外部控制条件的数据类型有 3 种可选：布尔型、数字整型和字符串型。

当控制条件为布尔型时，选择结构的选择器标签的值为 True 和 False 两种，即有 True 和 False 两种选择框架，如图 6.4.2 所示，这是 LabVIEW 默认的选择框架类型。

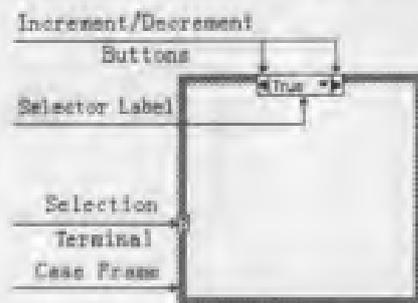


图 6.4.2 选择结构的组成

当控制条件为数字整型时，选择结构的选择器标签的值为整数 0, 1, 2……选择框架的个数可根据实际需要确定，在选择框架的右键弹出菜单中选择 Add Case After 或 Add Case After Before，可以添加选择框架，如图 6.4.3 所示。

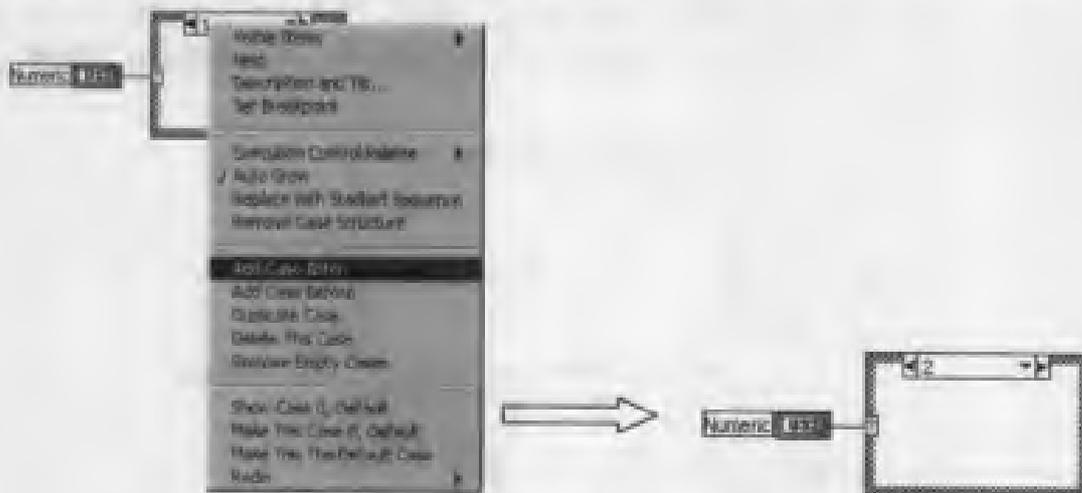


图 6.4.3 添加选择框架

当控制条件为字符串型时，选择结构的选择器标签的值为由双引号括起来的字符串，如“Condition 1”，选择框架的个数也是根据实际需要确定的，如图 6.4.4 所示。

注意，在使用选择结构时，控制条件的数据类型必须与选择器标签中的数据类型一致。二者若不匹配（如图 6.4.5 所示），LabVIEW 会报错，同时，选择器标签中字体的颜色将变为红色。



图 6.4.4 控制条件为字符串型时的选择结构



图 6.4.5 数据类型不匹配状态下的选择结构

在 VI 处于编辑状态时,用鼠标(对象操作工具状态)单击递增/递减按钮可将当前的选择框架切换到前一个或后一个选择框架;用鼠标单击选择器标签,可在下拉选单中选择切换到任一个的选择框架。如图 6.4.6 所示。



图 6.4.6 顺序框架的切换

6.4.2 选择结构的使用

例 6.4.1 求一个数的平方根,当该数大于等于 0 时,输出开方结果;当该数小于 0 时,用弹出式对话框报告错误,同时输出错误代码-99999.00。

本例就是选择结构的一个典型的应用,输入一个数后,首先判断该数是否小于 0,若小于 0,则用弹出式对话框报告错误,输出错误代码-99999.00;否则就输出计算结果。框图程序如图 6.4.7 所示。

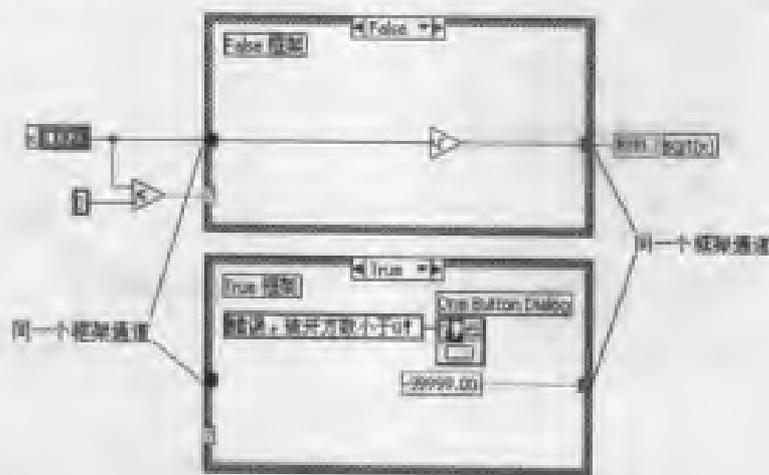


图 6.4.7 例 6.4.1 的框图程序

本例在 True 框架中采用了一个名为 One Button Dialog 的节点来实现弹出对话框,该节点相当于 Visual Basic 中的 MsgBox 函数,或相当于 Visual C++ 中的 MessageBox 函数。

6.4.3 选择结构的特点

LabVIEW 的选择结构与其他语言的选择结构相比,简洁明了,结构简单,不但相当于 Switch 语句,还可以实现 if...else 语句的功能。

为与选择框架外部交换数据,选择结构也有边框通道。选择结构的边框通道与顺序结构的框架通道类似,也没有 Enable Indexing 和 Disable Indexing 这两种属性。不过选择结构的边框通道还是具有其自身特点的。

当外部数据连接到选择框架上供其内部节点使用时，选择结构的每一个子框架都能从该通道中获得输入的外部数据；当选择结构内部的数据需通过框架通道送至外部时，必须在每一个子框架中都连接一个同数据类型的数据到同一个框架通道上，如图 6.4.7 所示。

这主要是因为选择结构执行时是根据外部控制条件从其所有的子框架中选择其一执行的，子框架的选择非此即彼，所以每一个子框架中都必须连接一个数据。对于一个框架通道，一个子框架中如果没有连接数据，那么在根据控制条件执行到这个子框架时，框架通道就没有向外输出数据的来源，程序就会出现错误。基于这种情况，LabVIEW 禁止这种做法，在编制程序时，若有一个子框架没有将数据连接到框架通道上，LabVIEW 便会报错，此时，边框通道的颜色为白色。如图 6.4.8 所示的框图程序就是错误的。

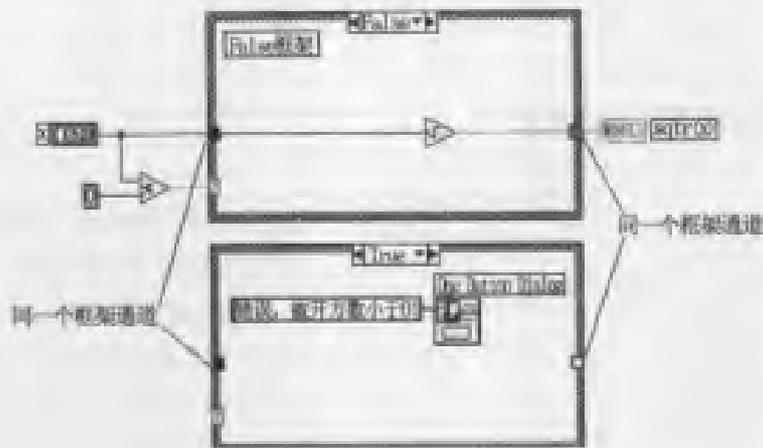


图 6.4.8 处于错误状态下的选择结构

从 LabVIEW 7 Express 开始，允许某些子框架没有将数据连接到框架通道上，但需要在框架通道的右键弹出选单中选择 Use Default If Unwired，如图 6.4.9 所示。此时，框架通道由 □ 状态变为 ■ 状态，当执行到这些没有连接数据的子框架时，框架通道输出默认值。



图 6.4.9 设置框架通道默认值

6.5 事件结构

在 LabVIEW 6.1 之前，老版本的 LabVIEW 没有对事件进行处理等能力，这些事件包括鼠标事件（单击、双击等）、键盘事件、选单事件、窗口事件（如关闭窗口）、对象的数值变化等。这给用户的编程带来了很大的不便。从 LabVIEW 6.1 开始，LabVIEW 提供了一个新的结构，名为事件结构（Event Structure），解决了这个问题，事件结构使 LabVIEW 具有了事件驱动的能力。

事件驱动是 Visual Basic、Visual C++ 等编程语言早已具有的基本功能。使用事件驱动可以让 LabVIEW 应用程序在没有指定事件发生时处于休息状态,直到前面板窗口中有一个事件发生时为止。在这段时间内,可以将 CPU 交给其他的应用程序使用,这大大提高了系统资源的利用率。事件结构可以从 Structures 子模板中创建,创建方法如图 6.5.1 所示。

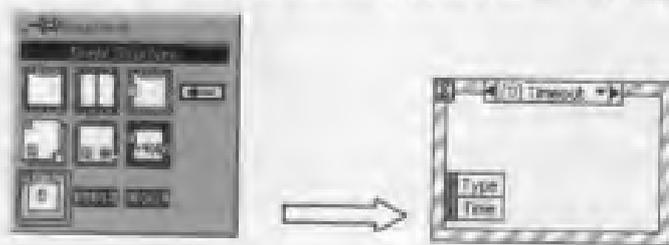


图 6.5.1 创建事件结构

6.5.1 事件结构的组成

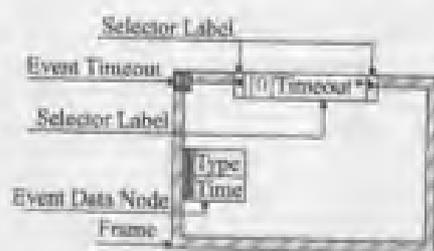


图 6.5.2 事件结构的组成

事件结构由框架 (Frame)、超时端口 (Event Timeout)、事件数据节点 (Event Data Node)、递增/递减按钮 (Increment/Decrement Buttons) 和选择器标签 (Selector Label) 组成,如图 6.5.2 所示。

事件结构与层叠式顺序结构和选择结构类似,是一个多框架的结构,由多个框架组成。事件结构是一种多选择 (Multi Case) 结构,能够同时响应多个事件,例如,单击鼠标左键的同时又

移动鼠标,这是两个事件同时发生,而事件结构会同时响应这两个事件。但是传统的选择结构是没有这个能力的,它只能一次接受并响应一个选择。

递增/递减按钮和选择器标签的功能与层叠式顺序结构和选择结构中的递增/递减按钮和选择器标签类似,用于在事件结构的多个框架之间切换。

超时端口用于设置事件结构在等待指定事件发生时的超时时间,其单位为 ms,默认值为 -1。若输入值为 -1,事件结构处于永远等待状态,直到指定的事件发生为止。若输入值为一个大于 0 的整数,例如 100,事件结构会等待相应的时间,当指定的事件在指定的时间内发生时,事件结构接受并响应该事件,若超过指定的时间,事件没有发生,则事件会停止执行,并返回一个 Time Out 事件。在一般情况下,应当为事件结构指定一个超时时间,否则事件结构将一直处于等待状态,这会影响到其他程序运行。

事件数据节点位于事件结构框架的左内侧,用于输出事件的参数,节点的端口数目和数据类型根据事件的不同而不同,如图 6.5.3 所示。通过该节点可以获得事件的相关信息,例如鼠标的坐标、鼠标按钮的编号、键盘按键、事件发生时间等。用户可以根据这些信息进行事件编程。对于某些事件,事件结构框架的右侧也会有一个节点,这个节点用于处理该事件,例如图 6.5.3 中右侧的事件结构框架中是一个 Panel Close? 事件,这个事件结构框架右侧的节点只有一个名为 Discard? 的端口,该节点用于确定当前面板窗口关闭事件发生时,是否关闭该窗口。

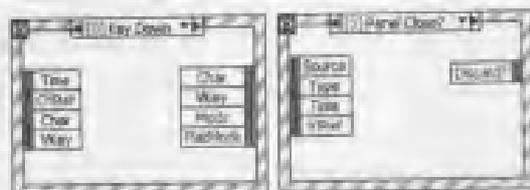


图 6.5.3 事件结构

6.5.2 事件结构的使用

事件结构与框图程序中的其他节点、模块在执行时的流程规则没有什么不同，当没有任何事件发生时，事件结构就会处于睡眠状态，直到有一个或多个预先设定的事件发生时，事件结构才会自动苏醒，并根据发生的事件执行用户预先设定的动作。

事件结构能够响应的事件有两种类型：通告（Notify）事件和过滤（Filter）事件。

通告事件通知 LabVIEW 一个动作已经发生，例如用户改变了一个控件的值。可以编写框图程序，当用户用鼠标单击前面板上的按钮时，通知事件结构。用户用鼠标单击按钮时，按钮值被改变，事件结构即得到该事件发生的通知。事件结构可以同时处理多个通告事件。

过滤事件用来控制用户界面的操作。通过过滤事件，可以在用户界面收到事件数据之前决定是否激活、修改这些数据，或者完全丢弃这些数据而使其对 VI 不产生任何作用。例如，可以通过设置 Menu Selection 事件限制用户不能交互的关闭前面板。可以通过设置 Key Down 事件使得键入某个字符串控件的字符一律大写，或者屏蔽掉某些字符。

请注意，一个事件结构的子框架可以处理多个过滤事件，但前提条件是这些事件返回值是相同的。如果试图在一个子框架中处理两个具有不同返回值的事件，如 Key Down 事件和 Mouse Down 事件，VI 会出错。

在事件结构框架的右键菜单中选择 Edit Events Handled by This Case...，可以弹出 Edit Events 对话框，如图 6.5.4 所示。



图 6.5.4 Edit Events 对话框

通过 Edit Events 对话框, 可以设定某一个事件结构子框架响应的事件。Edit Events 对话框共由 5 栏组成:

(1) Events Handled for Case 栏

选择事件结构的子框架, 选定之后, 可以设定这个子框架中响应的事件。

(2) 选定的事件列表栏

在该栏中列出用户设定的事件, 并列出了该事件属于哪一个控件。按钮  用于添加事件, 按钮  用于删除事件。

(3) 事件信息栏

当用户选定一个事件时, 该栏会提示用户该事件的一些相关信息。

(4) Event Sources 栏

该栏将 VI 中所有具有事件的对象列出

(5) Events 栏

当用户在 Event Sources 栏中选定一个对象时, 该栏将该对象的事件列出。

用户可以在一个事件结构子框架中设定多个事件, 当然, 用户也可以在事件结构中添加多个子框架, 以响应多个事件。添加子框架的方法是, 在事件结构框架的右键菜单中选择 Add Event Case..., 如图 6.5.5 所示。

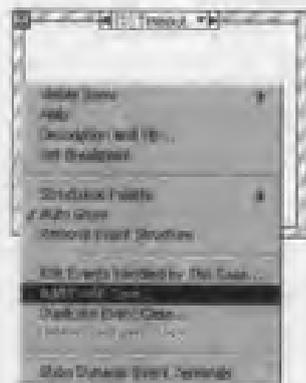


图 6.5.5 添加事件结构子框

例 6.5.1 使用事件结构处理 Panel Close 事件

本例演示当关闭一个 VI 前面板窗口, 即鼠标单击窗口右上角的“关闭”按钮 , 发生 Panel Close 事件时, 对 Panel Close 事件的处理, 这是一个过滤事件处理的实例。框图程序如图 6.5.6 所示。

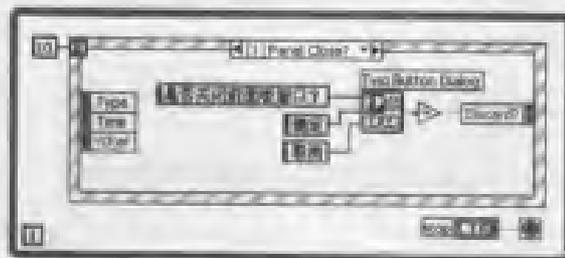


图 6.5.6 Panel Closing 事件处理框图程序



图 6.5.7 询问对话框

在事件结构的 Panel Close?子框架中, 使用了一个 Two Button Dialog 节点, 当 VI 处于运行状态, 用户单击 VI 前面板窗口右上角的关闭按钮时, 就会发生一个 Panel Close 事件, 此时事件结构就会运行 Panel Close?子框架中的程序, 即 Two Button Dialog 节点, 此时, 就会弹出一个对话框, 询问用户是否要关闭前面板窗口, 如图 6.5.7 所示。当单击确定按钮时, VI 前面板窗口就会关闭; 当

单击取消按钮时, VI 前面板窗口会维持原状。

6.5.3 事件结构的特点

事件结构是 LabVIEW 的一个特殊的节点，有一些与其他结构不同的特点，正确的利用这些特点来编程，可以大大减少程序的复杂程度，并且可以实现很多方便用户的功能，但使用不当时，可能会引起一些不必要的错误，因此，有必要提醒读者注意这些特点和相关的问题。

1. 使用 Mechanical Action 属性为 Latch 的布尔控件来触发事件

当使用 Mechanical Action 为 Latch 的布尔控件来触发事件时，在框图程序读出布尔控件的值以前，布尔控件的值不会恢复到默认状态。为了使布尔控件能够进行正常的机械动作，必须在事件结构子框架内读取该控件的值。请注意，当在事件结构中设置一个 Mechanical Action 属性为 Latch 的布尔控件的 Value Changed 事件时，Edit Events 对话框中会出现一个 Note，提示用户当单击该控件改变它的值时，LabVIEW 不会自动触发它的机械动作，用户必须从该控件的端口中读出数据以后，该控件才会正确的重置。

2. 在一个事件结构中使用多个子框架处理相同事件

使用处于同一个事件结构中的多个子框架不能处理相同的事件，这是因为当事件发生时，LabVIEW 只能使用一个事件结构子框架处理该事件。如果令多个事件结构子框架处理同一个事件，VI 会出错。

3. 使用一个事件结构子框架处理多个通告事件

在事件结构中，一个子框架可以处理多个通告事件。但是，如果事件类型不同，只有那些对所有事件通用的事件数据才会出现在子框架的左侧的事件数据节点中。如果在同一个子框架中处理多个不同前面板对象的（数据类型不同）的 Value Changed 事件，OldVal 端口和 NewVal 端口会返回不确定的数据类型。例如，在同一个子框架中处理一个 Numeric 控件 (I32) 和一个 Boolean 控件的 Value Changed 事件时，当 Numeric 控件的数据发生变化，OldVal 端口和 NewVal 端口返回的是有符号 32 位整数，当 Boolean 控件的数据发生变化，OldVal 端口和 NewVal 端口返回的是 Boolean 类型的数据 True 或 False，这样，连接到这两个端口的其他节点或模块就没有办法对这两种不同的数据类型进行处理。

4. 使用一个事件结构子框架处理多个过滤事件。

一个事件结构子框架不能处理多个过滤事件，除非事件拥有相同的事件数据（如 Key Down, Key Repeat 等）。通过过滤事件，可以在用户界面收到事件数据之前决定是否激活、修改这些数据，或者完全丢弃这些数据，使其对 VI 不产生任何作用。因为用户可以些改事件数据，所以过滤事件不同于通告事件，事件结构不能对过滤事件数据进行合并操作。

5. 使用事件结构处理用于循环控制的布尔控件

图 6.5.8 所示的框图程序（例 6.5.2）演示了处理用于循环控制的布尔控件的方法，

LabVIEW 推荐使用这种方法, 下面分析框图程序的运行过程。

事件结构位于 While 循环中, 使用一个事件结构子框架处理 Stop 按钮 (Mechanical Action 为 Latch) 的 Value Changed 事件。当 Stop 按钮的值变为 True 时, 事件结构将处理该 Value Changed 事件, 读出 Stop 按钮的值, 同时重置 Stop 按钮。Stop 按钮与 While 循环的 Conditional 端口相连, 此时 While 循环将终止运行。

但用户在编程时经常不会采用这种方法, 而采用一种错误的方法 (例 6.5.3), 即不通过事件结构来处理 Stop 按钮。错误的框图程序如图 6.5.9 所示。



图 6.5.8 处理用于循环控制的布尔控件的框图程序



图 6.5.9 错误的框图程序

现在对如图 6.5.9 所示的框图程序的运行过程进行分析: 当 While 循环第一次运行时, While 循环读出 Stop 按钮的值, 并检查 Conditional 端口的值, 此时为 False。事件结构运行, 等待事件的发生。当用户单击 Stop 按钮, 产生 Value Changed 事件, 并触发处理 Value Changed 事件的事件结构子框架运行。While 循环再次检查 Conditional 端口的值, 现在为 True, 于是将停止循环 (但本次循环还未结束), 事件结构正在等待下一个事件的发生。当用户再一次单击 Stop 按钮, 触发事件结构运行, 这一次将运行正确的事件结构子框架。循环结束, VI 停止运行。

在这种错误的编程方式下, 用户需要单击两次 Stop 按钮才能够结束程序的运行。

6. 使用事件与使用 Wait for Front Panel Activity 节点

在某种应用中, 需要在 VI 运行期间, 当用户没有对前面板进行操作时, 暂停 VI 的运行, 当用户在前面板进行了某项操作后, 再继续运行 VI, 可以用 Wait for Front Panel Activity 节点 (位于 Functions 模板 → All Functions 模板 → Time & Dialog 子模板中) 来实现这项功能。使用该节点可以节省系统资源, 但是该节点没有任何事件反馈 (如鼠标移动、某个键被按下)。然而, 使用事件结构可以对用户的多种操作 (对控件、VI 或应用程序的) 作出反应。

7. 事件结构只响应用户交互操作引发的事件

LabVIEW 中的事件结构只会处理用户直接施加于前面板的操作, 如果通过在框图程序中使用 VI Server、全局变量、本地变量来改变 VI 或者前面板控件, 则不会触发事件。

8. 避免在同一个循环中使用两个事件结构

事件结构按事件发生的顺序来处理事件。当事件结构处理完毕一个事件后, 循环继续运行。如果在一个循环中使用两个事件结构, 则只有当两个事件结构都处理完毕各自设定的事件后, 循环才会继续运行。如果两个事件结构没有按正确的顺序处理事件, 用户界面

可能会挂起。

例如，如果在一个单循环中放置了两个事件结构，第一个事件结构处理 Mouse Down 事件，第二个事件结构处理 Key Down 事件。第一个事件结构先收到了一个 Mouse Down 事件，接着第二个事件结构收到了一个 Key Down 事件。两个事件结构处理各自的事件，循环得以继续运行。如果事件不断交替，两个事件结构就能够正确处理各自的事件，循环就可以持续下去。

但是，如果两个 Mouse Down 事件连续出现了两次，则会出现用户界面会挂起的情况。这是因为用来处理 Mouse Down 事件的事件结构在处理第一个 Mouse Down 事件后，处理 Key Down 的事件结构在等待 Key Down 事件的发生，但是第二个 Mouse Down 事件接着发生，LabVIEW 在处理第二个 Mouse Down 事件前不能处理 Key Down 事件，而再次处理 Mouse Down 事件需要新一轮的循环，因此，循环无法继续，用户界面将挂起。

要避免出现这种情况，可以使用一个事件结构处理所有事件，或在每个循环中只使用一个事件结构。

6.6 基本公式节点

LabVIEW 是一种图形化编程语言，主要编程元素和结构节点是系统预先定义的，用户只需要调用相应节点构成框图程序即可，这种方式虽然方便直接，但是灵活性受到限制，尤其对于复杂的数学处理，变化形式多种多样，LabVIEW 就不可能把所存的数学运算和组合方式都形成图标。为了解决这一问题，LabVIEW 另辟蹊径，提供了一种专用于处理数学公式编程的特殊结构形式，称为基本公式节点 (Formula Node)。在基本公式节点框架内，LabVIEW 允许用户像书写数学公式或方程式一样，直接编写数学处理节点，形式与标准 C 语言类似。本节将主要介绍基本公式节点的详细用法。

6.6.1 基本公式节点的创建

基本公式节点创建的过程比较复杂，通常按以下步骤进行：

第一步，在 Structures 子模板中选择 Formula Node，然后用鼠标在框图程序中拖动，画出基本公式节点的框架，如图 6.6.1 所示。

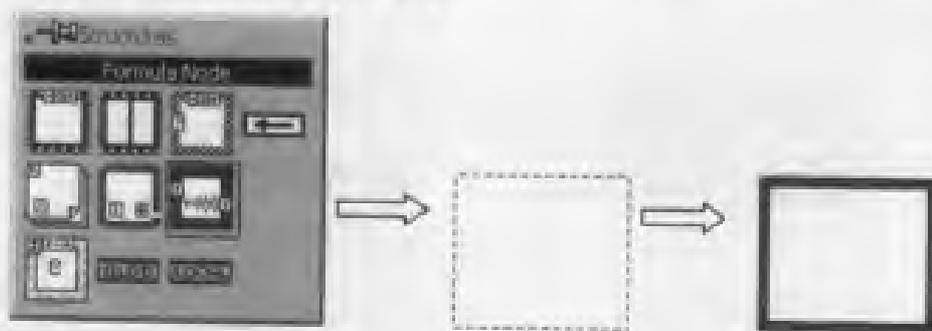


图 6.6.1 创建基本公式节点

第二步, 添加输入输出端口。

在基本公式节点框架的右键弹出菜单中选择 Add Input, 然后在出现的端口图标中填入该端口的名称, 就完成了输入端口的创建, 输出端口的创建与此类似。注意输入变量的端口都在基本公式节点框架的左边, 而输出变量则分布在框架的右边。如图 6.6.2 所示。

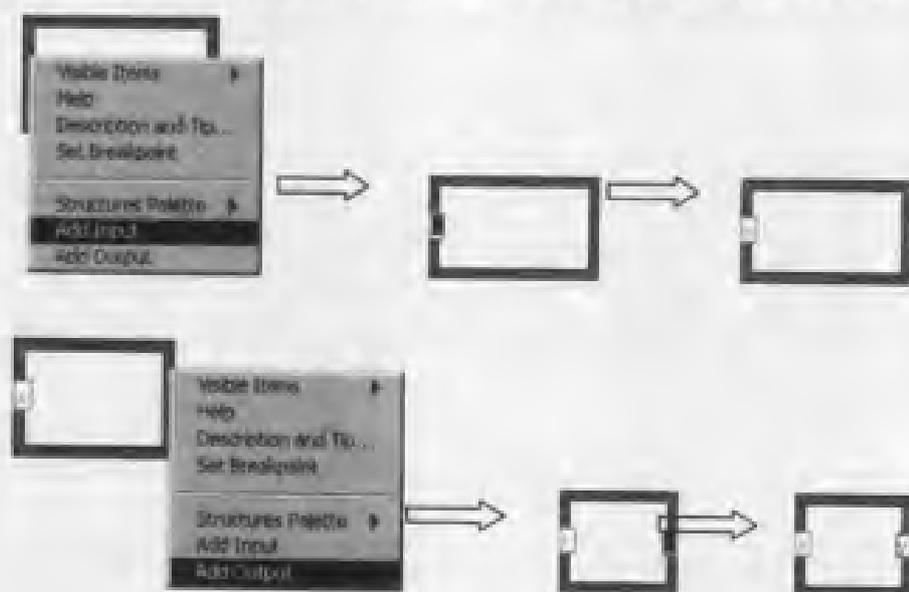


图 6.6.2 添加输入输出端口



图 6.6.3 输入程序代码

第三步, 按照 C 语言的语法规则在基本公式节点的框架中加入程序代码, 特别要注意的是基本公式节点框架内每个公式后都必须有分号“;”标示, 如图 6.6.3 所示。

至此, 就完成了完整的基本公式节点的创建。

6.6.2 基本公式节点的使用

完成一个基本公式节点的创建后, 基本公式节点就可以像其他节点一样使用了。

例 6.6.1 使用基本公式节点。

本例使用了 6.6.1 节中创建的基本公式节点来完成相应的计算。VI 程序的计算结果及框图程序如图 6.6.4 所示。



图 6.6.4 例 6.6.1 的前面板及框图程序

基本公式节点中代码的语法与 C 语言相同, 可以进行各种数学运算。这种兼容性使 LabVIEW 的功能更加强大, 也更容易使用。在基本公式节点中可以使用的数学函数见表 6.6.1。

表 6.6.1 基本公式节点可用数学函数见表

函数名	说明
abs(x)	绝对值函数
acos(x)	反余弦函数, x 的单位是弧度
acosh(x)	反双曲余弦函数, x 的单位是弧度
asin(x)	反正弦函数, x 的单位是弧度
asinh(x)	反双曲正弦函数, x 的单位是弧度
atan(x)	反正切函数, x 的单位是弧度
atanh(x)	反双曲正切函数, x 的单位是弧度
ceil(x)	返回大于 x 的最小整数
cos(x)	余弦函数, x 的单位是弧度
cosh(x)	双曲余弦函数, x 的单位是弧度
cot(x)	余切函数, x 的单位是弧度
csc(x)	余割函数, x 的单位是弧度
exp(x)	指数函数
expm1(x)	返回 $\exp(x) - 1$
floor(x)	返回小于 x 的最大整数
getexp(x)	将 x 表示为, $x = \text{mantissa} * 2^{\text{exponent}}$, 返回指数 exponent
getman(x)	将 x 表示为, $x = \text{mantissa} * 2^{\text{exponent}}$, 返回尾数 mantissa
int(x)	返回距 x 最近的整数
intz(x)	返回 x 与 0 之间距 x 最近的整数
ln(x)	自然对数函数
lnp1(x)	返回 $\ln(x)+1$
log(x)	对数函数, 以 10 为底
log2(x)	对数函数, 以 2 为底
max(x,y)	返回 x, y 中值大者
min(x,y)	返回 x, y 中值小者
mod(x,y)	求模运算, 返回 x/y 商的整数值
pow(x,y)	返回 x 的 y 次方
rand()	产生(0,1)区间上的随机数
rem(x,y)	返回 x/y 的余数
sec(x)	正割函数

续表

函数名	说明
sign(x)	符号函数, 如果 $x > 0$, 返回 1; 如果 $x = 0$, 返回 0; 如果 $x < 0$, 返回 -1
sin(x)	正弦函数, x 的单位是弧度
sinc(x)	$\text{sinc}(x) = \sin(x)/x$, x 的单位是弧度
sinh(x)	双曲正弦函数
sizeOfDim(ary, di)	返回数组 ary 的第 di 维的长度
sqr(x)	求解 x 的平方根
tan(x)	正切函数
tanh(x)	双曲正切函数
Pi	π

在基本公式节点中可以使用的操作符见表 6.6.2。

表 6.6.2 基本公式节点可用操作符

操作符	功能
**	求幂
+ - ! ~ ++ --	正号、负号、逻辑非、位补码、算前/算后增量、算前/算后减量 ++和--在 Express 节点中无效
*/%	乘、除、求模
+-	加、减
>><<	算术右移位、算术左移位
><>=<=	大于、小于、大于等于、小于等于
!= ==	不等于、等于
&	位与
^	位异或
	位或
&&	逻辑与
	逻辑或
?:	条件选择
=op=	赋值、快捷操作和赋值 op 可以是 +、-、*、/、>>、<<、&、^、 、% 或 ** =op= 在 Express 节点中无效

注意, 在基本公式节点中不能使用循环结构和复杂的选择结构, 但可以使用简单的选择结构:

<逻辑表达式> ? <表达式 1> : <表达式 2>

基本公式节点支持的数据类型见表 6.6.3，表 6.6.3 同时也给出了 C 语言中相对应的数据类型。

表 6.6.3 基本公式节点支持的数据类型

Formula node 数据类型	C 语言数据类型
无符号 8 位整型	Unsigned short
无符号 16 位整型	Unsigned Int
无符号 32 位整型	Unsigned Long
有符号 8 位整型	Short
有符号 16 位整型	Int
有符号 32 位整型	Long
单精度浮点型	Float
双精度浮点型	Double

6.6.3 基本公式节点的特点

基本公式节点的引入，使得 LabVIEW 的编程更加灵活，对于一些稍微复杂的计算公式，用图形化编程可能会显得有些烦琐，此时若采用基本公式节点来实现这些计算公式，可能会减少编程的工作量。在进行 LabVIEW 编程时，可根据图形化编程和基本公式节点各自的特点，灵活使用不同的编程方法，这样可以大大提高编程的效率。

使用基本公式节点时，有一点应当注意：在基本公式节点框架中出现的所有变量，必须有一个相对应的输入端口或输出端口，否则 LabVIEW 会报错。

6.7 属性节点

LabVIEW 提供了各种样式的前面板对象，应用这些前面板对象，可以设计定制出仪表化的人机交互界面。但是，仅提供丰富的前面板对象还是不够的，在实际运用中，还经常需要实时地改变前面板对象的颜色、大小和是否可见等属性，达到最佳的人机交互功能。比如对一个实时监控画面，当出现参数潮差和其他异常情况，需要提醒用户注意时，常常是通过改变对象的颜色来完成的，这一属性变化是在程序运行过程中由某一逻辑条件触发而非预先定义的。于是，LabVIEW 引入了属性节点 (Property Node) 这一概念，通过改变前面板对象属性节点中的属性值，可以在程序运行中动态地改变前面板对象的属性。本节主要介绍属性节点的创建与使用方法。

6.7.1 属性节点的创建

属性节点的创建比基本公式节点的创建简单。在前面板对象或其端口的右键弹出选单中选择 Create→Property Node，就可创建一个属性节点（位于框图程序窗口）。如图 6.7.1 所示。

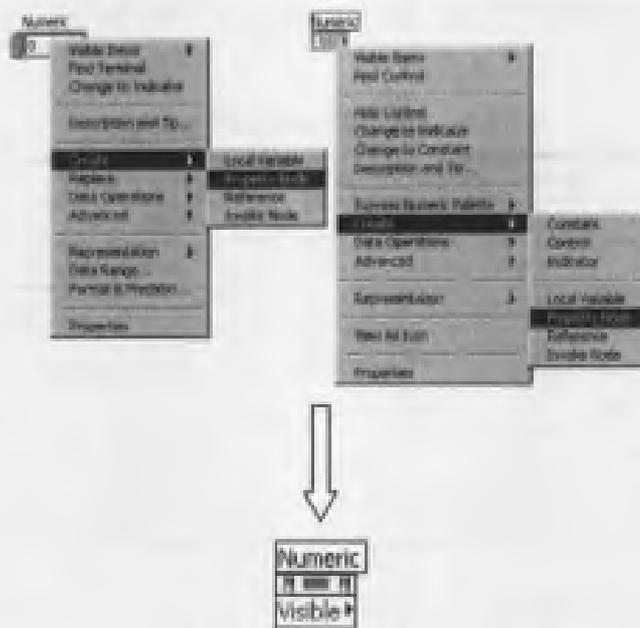


图 6.7.1 创建属性节点

用操作工具直接单击属性节点的图标,或在图标的右键弹出选单中选择 Properties,会出现一个下拉选单,选单列出了前面板对象的所有属性,可根据需要选择相应的属性。如图 6.7.2 所示。



图 6.7.2 属性选择

若需要同时改变前面板对象的多个属性,一种方法是创建多个属性节点,另外一种更加简捷的方法是在一个属性节点的图标上添加多个端口。添加的方法是用鼠标(对象操作工具状态)拖动属性节点图标下边缘(或上边缘)的尺寸控制点,或在属性节点图标的右键弹出选单中选择 Add Element,如图 6.7.3 所示。

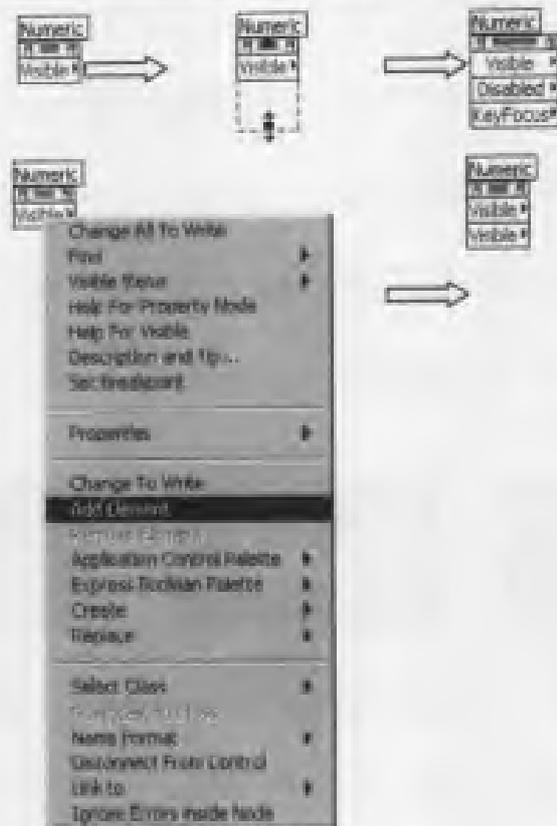


图 6.7.3 添加端口

6.7.2 属性节点的使用

有效地使用属性节点可以使用户设计的图形化人机交互界面更加友好、美观，操作更加方便。由于不同类型前面板对象的属性种类繁多，各不相同，在有限的篇幅内是很难一一介绍的，因此本小节将主要介绍一些前面板对象共有且常用属性的用法。掌握了这些基本属性及用法之后，其他一些特殊属性的用法可以依此类推。

下面以数字量控制（Numeric Control）为例来介绍属性节点的用法。

1. Visible

该属性用来控制前面板对象在前面板窗口中是否可视，其数据类型为布尔型。

- 当 Visible 值为 True 时，前面板对象在前面板上处于可视状态，如图 6.7.4 所示。
- 当 Visible 值为 False 时，前面板对象在前面板上处于隐藏状态，如图 6.7.5 所示。



图 6.7.4 Visible 属性

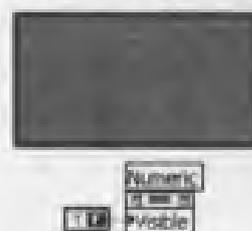


图 6.7.5 Visible 属性

2. Disabled

通过这个属性,当VI处于运行状态时就可以控制用户是否可以访问一个前面板对象,其数据类型为整型。

- 当输入值为 0 时,前面板对象处于正常状态,用户就可以访问该前面板对象,如图 6.7.6 所示。
- 当输入值为 1 时,前面板对象的外观处于正常状态,但用户不能访问该前面板对象,如图 6.7.7 所示。
- 当输入值为 2 时,前面板对象处于 Disable 状态,此时,用户不可访问这个前面板对象。如图 6.7.8 所示。



图 6.7.6 Disabled 属性



图 6.7.7 Disabled 属性



图 6.7.8 Disabled 属性

3. Key Focus

该属性用于控制前面板对象是否处于键盘焦点状态,其数据类型为布尔型。

当输入值为 True 时,前面板对象处于键盘焦点状态,如图 6.7.9 所示。

当输入值为 False 时,前面板对象处于失去键盘焦点状态,如图 6.7.10 所示。



图 6.7.9 Key Focus 属性

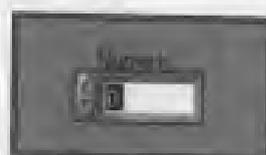


图 6.7.10 Key Focus 属性



图 6.7.11 Blinking 属性

4. Blinking

该属性用于控制前面板对象是否闪烁,其数据类型为布尔型。

- 当输入值为 True 时,前面板对象处于闪烁状态。
- 当输入值为 False 时,前面板对象处于正常状态。

处于闪烁状态的前面板对象如图 6.7.11 所示。

前面板对象闪烁的速度和颜色是可以设置的,不过这两个

属性不能由属性节点来设置，并且一旦设定了闪烁的速度和颜色，在 VI 处于运行状态时，这两种属性值就不能再改变。

在 LabVIEW 主选单 Tools 中选择 Options..., 弹出一个名为 Options 的对话框，在对话框中可以设置闪烁的速度和颜色。

在对话框上部的下拉列表框中选择 Front Panel，对话框中会出现如图 6.7.12 所示的属性设定选项，可以在选项 Blink Speed 中设置闪烁的速度。

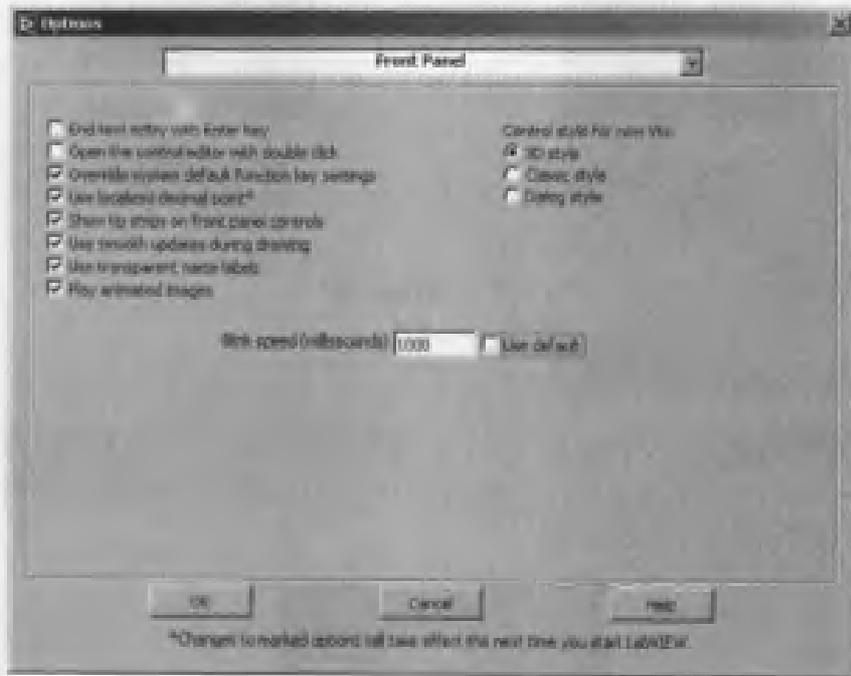


图 6.7.12 设置闪烁速度

在对话框上部的下拉列表框中选择 Colors，对话框中出现图 6.7.13 所示的属性设定选项，选项 Blink Foreground 和 Blink Background 中可以分别设置闪烁的前景色和背景色。

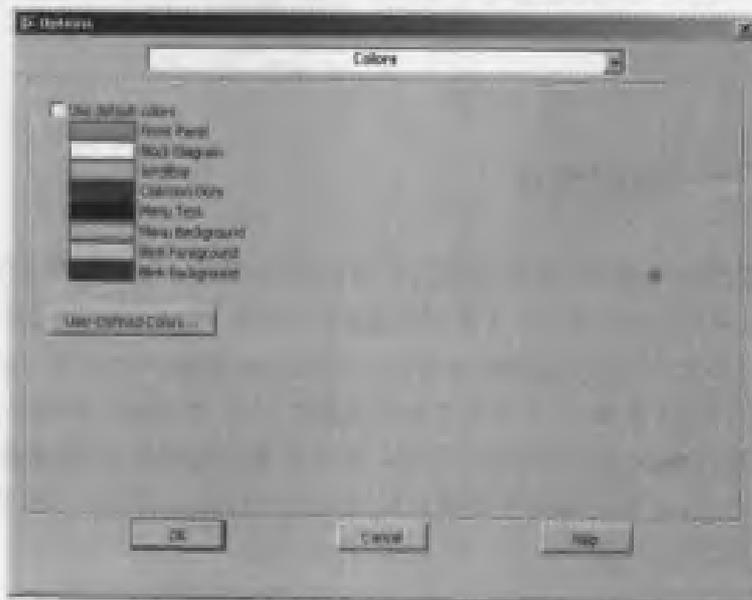


图 6.7.13 设置闪烁的前景色和背景色

5. Position

该属性用于设置前面板对象在前面板窗口中的位置，其数据类型为簇，簇中包含两个元素，均为整型，一个是前面板对象图标左边缘的 y 坐标，另一个是前面板对象图标上边缘的 x 坐标。窗口的左上角为坐标原点，水平向右为 x 轴，垂直向下为 y 轴。具体用法如图 6.7.14 所示。

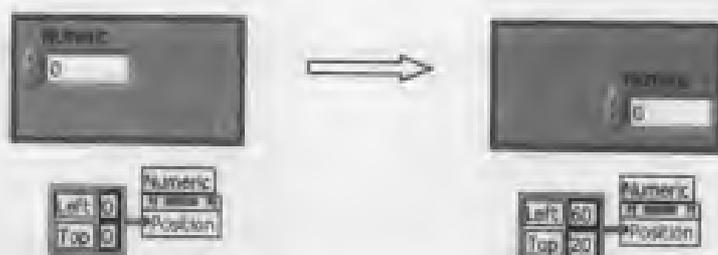


图 6.7.14 Position 属性

6. Bounds (Read Only)

该属性用于获得前面板对象图标的大小，包括高度和宽度。其数据类型为簇，包含两个整型元素，一个为前面板对象图标的宽度，另一个为高度。该属性端口的属性为只读，不能赋值。具体用法如图 6.7.15 所示。

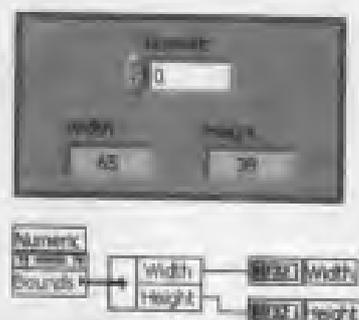


图 6.7.15 Bounds 属性

6.7.3 属性节点的特点

属性节点与本地变量类似，也有 Read 和 Write 两种属性，在属性节点图标某一端口的右键弹出选单中选择 Change to Read 或 Change to Write 可改变该端口的读写属性，选择 Change All to Read 或 Change All to Write 可改变属性节点图标中所有端口的读写属性。

在某种意义上，LabVIEW 中前面板对象的属性节点 (Property Node) 就相当于 Visual Basic 中控件的属性 (Properties)。在编程应用中会经常用到属性节点，特别是对于 Waveform Graph 和 Waveform Chart 两种前面板对象，应用属性节点可以添加不少实用的功能。

第7章 波形显示

测量数据的图形化实时动态显示是高级仪器必备的功能，像常见的数字示波器、频谱分析仪和逻辑分析仪等仪器都具有能够显示测量信号波形和仪器工作状态的 CRT 荧光屏。LabVIEW 为模拟真实仪器的操作面板，提供了强大的交互式界面设计功能。本章介绍的各种波形显示控件就是在 LabVIEW 程序设计中常用的前面板对象之一，也是 LabVIEW 使用比较灵活，功能比较完善，特色比较突出的模板。

按照处理测量数据的方式和显示过程的不同，LabVIEW 波形显示控件主要分成两大类，一类为事后记录图 (Graph)，或事后记录波形控件；另一类称为实时趋势图 (Chart)，或实时趋势波形控件。这两类控件都是用来对波形或图形进行显示的，它们的区别在于两者数据组织方式及波形的刷新方式不同。对于事后记录图 Graph 方式来说，它的基本数据类型为数组，也就是说 Graph 显示是将构成数组的全部测量数据一次显示完成的；而实时趋势图 Chart 方式则是实时显示一个或几个测量数据，而且新接收数据点要接在原有波形的后面连续显示。它的基本数据类型是数据标量，也可以是数组。即使是数组，Chart 方式也是连续不断地一个数组接着一个数组显示，而不是像 Graph 方式一次显示完成。Graph 和 Chart 两种显示方式的区别和灵活使用是本章的重点，本章将常用的波形显示和图形控件予以介绍。

7.1 事后记录波形图

在 LabVIEW 的 Controls 模板 → All Controls 子模板 → Graph 子模板中，可以找到所有的波形显示控件，如图 7.1.1 所示。Graph 模板中还包含了一些三维图形和极坐标图等控件。



图 7.1.1 Graph 子模板

Waveform Graph，即事后记录波形图，其典型前面板结构如图 7.1.2 所示。Waveform Graph 既可显示单个信号波形，也可以同时显示多个信号波形，并且还提供实时任意缩放

和图上测量等高级显示工具。它的坐标、网格和标注等设置都可以根据需要灵活定制。Waveform Graph 显示波形是以成批数据一次刷新方式进行的, 数据输入基本形式是数据数组、簇或波形数据 (Waveform Data Type)。



图 7.1.2 事后记录波形图 (Waveform Graph)

7.1.1 事后记录波形图的组成

一个 Waveform Graph 除了包含标签、波形显示区外还提供了一些控制工具。在默认设置下, 这些工具是隐藏的, 如要显示这些控制工具, 可以在 Waveform Graph 的右键弹出菜单中的 Visible Items 中选择。下面对 Waveform Graph 的组成分别予以介绍。

1. 标签 (Lable)



图 7.1.3 Waveform Graph 的标签

Waveform Graph 左上角的标签 (如图 7.1.3 所示) 和其他前面板对象的标签一样, 可以通过文本编辑工具  来定义, 给 Waveform Graph 命名, 如“温度变化显示”等。

2. 坐标设置工具 (Scale Legend)

将横轴定义为 X 轴, 纵轴定义为 Y 轴。X 轴代表数组中数据的序号, 而 Y 轴表示需要显示测量数据点的数值大小。在默认条件下, X 轴初值为 0, 步长为 1, 最大刻度范围根据数组长度进行自动调整, 而 Y 轴刻度则根据数组中最大与最小值范围自动设定, 也可以



图 7.1.4 坐标工具

根据显示需求自己设定 X、Y 轴的标度, 这将在后续的内容中进行介绍。通过坐标工具, 如图 7.1.4 所示, 可以设置 X、Y 轴的名称、自动量程选择 (Auto Scale)、数据格式、精度、网格线的颜色、坐标类型 (线性或对数) 等属性。

3. 波形显示区域

为了使判读准确方便, Waveform Graph 允许在波形显示区域设置网格线, 网格间距大小与 X、Y 轴刻度相关, 可以由用户自己定义。

4. 波形设置工具 (Plot Legend)

波形设置工具如图 7.1.5 所示。用鼠标单击它可以弹出快捷选单, 通过快捷选单, 用户可以设定波形曲线的各种属性, 包括波形的名称、线型和颜色等。当然, 波形曲线的属性也可以在框图程序中通过对属性节点编程来实现。在多条波形曲线同时显示的情况下, 通过设置波形曲线的不同属性, 可以直观地区分不同信号的波形。



图 7.1.5 Plot Legend

5. 图形控制工具 (Graph Palette)

图形控制工具如图 7.1.6 所示, 由 3 部分组成: 分别是光标选择工具、图形缩放工具和图形移动工具。通过 Waveform Graph 自带的图形控制工具, 可以在程序运行中放大、缩小或移动所显示的波形, 还可改变 X、Y 轴的刻度值, 从而有针对性地对波形中感兴趣的部分进行仔细的观察。



图 7.1.6 图形控制工具

6. 光标控制工具 (Cursor Legend)

光标控制工具如图 7.1.7 所示, 可以移动光标, 设置光标名称、颜色、形状、线形等属性, 还可以显示光标所在位置的坐标, 这在测量中是非常有用的。图 7.1.7 中左边的是光标属性控制工具, 右边的是光标移动工具。在实际应用中还可以直接用鼠标拖动光标。



图 7.1.7 光标控制工具

7.1.2 使用事后记录波形图

Waveform Graph 的基本数据类型有三种: 一维或二维的数据数组、簇或波形数据。当 Waveform Graph 同时显示多条波形曲线或对波形的显示方式有所规定时, 它的数据类型是一个包含有波形数据数组的簇或是波形数据。本节将通过一些实际例程, 详尽说明 Waveform Graph 在不同应用时其数据组织方法的异同。

例 7.1.1 用 Waveform Graph 显示一次拥有 30 个采样点的温度测量结果。

首先, 在程序前面板中放置一个 Waveform Graph, 用文本编辑工具将其标签的默认值“Waveform Graph”改成“例 7.1.1 显示”; 用文字编辑工具把右上角 Plot Legend 中的波形名称“Plot0”改成“温度值”; 然后在框图窗口中设计程序, 显示结果及程序框图如图 7.1.8 所示。

本例中, 首先用一个在 For 循环里的随机数节点 Random Number (0-1) 模拟测量结果, 模拟的测量数据在 For 循环的边框通道上形成一个数组, 然后送到事后记录波形图中进行显示。

本例是 Waveform Graph 最基本、最常用的用法。Waveform Graph 是一次性完成显示图形刷新的, 所以其输入数据必须是完成一次显示所需的数据数组, 而不能把测量结果一次一个地输入。在本例中, 不可以把 Random Number (0-1) 节点输出端口与 Waveform Graph 的端口直接相连, 否则, Waveform Graph 只会显示一个点。

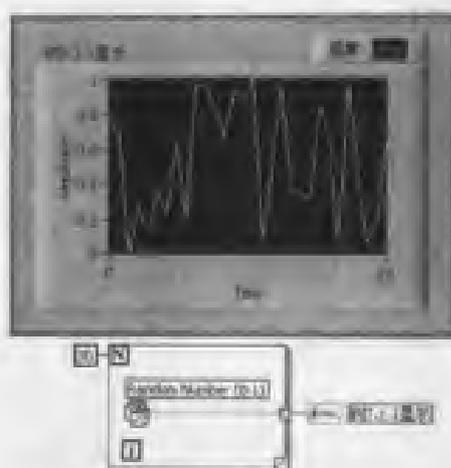


图 7.1.8 例 7.1.1 的前面板及框图程序

例 7.1.2 设计一个程序显示一个电压测量结果。电压采样从 10ms 后开始，每隔 5ms 采一个点，共采集 30 个点；电压在采样前还经过一个信号处理电路的 10 倍衰减。要求程序的显示能够反映出实际的采样时间及电压值。

程序面板的设置与上例相同，显示结果及其框图程序如图 7.1.9 所示。

程序前面板的设计过程与例 7.1.1 一样，本例也是用一个 30 点的 For 循环里一个 Random Number (0-1) 节点来模拟测量结果。为了补偿信号处理电路的衰减，乘了一个放大因子 10；而且依要求加入一个 5 ms 的测量延时。值得注意的是，这个程序数据的组织方法与例 7.1.1 不同。为了使 X 轴的刻度值与实际的测量时间对应，加入了一个起始位置 X0 及步长 DeltaX，并把这两个数与所测量数据数组打包 (Bundle)，然后送入 Waveform Graph。

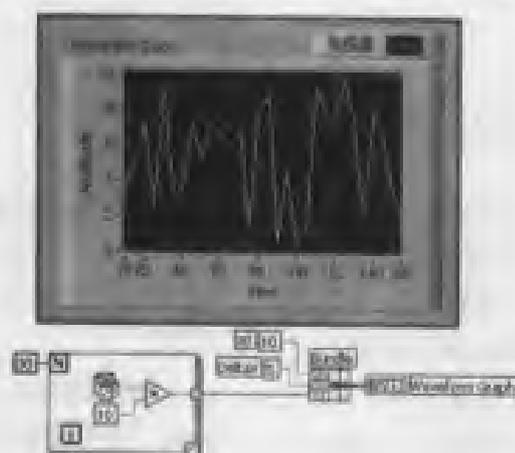


图 7.1.9 例 7.1.2 的前面板及框图程序

在默认情况下，X 轴刻度是从初始位置 $X_0=0$ 开始的，但是在很多实际的测量应用中，有效的测量点或时间并不是从 0 开始。为了使显示的 X 轴刻度符合实际的情况，可以设定 X 轴的初始刻度 X_0 ，在本例中，根据要求设定 $X_0=10$ ，所以在显示中可以看到第一个数据对应的 X 刻度值为 10。在默认情况下，X 轴刻度值按照步长 1 递增，此时，X 轴是对应数据点的序号，即 X 轴上刻度 n 对应数据数组中序号 (Index) 为 n 的数据。但实际应用中，对应显示某个物理参数的 X 轴，其步长可能不是 1，这种情况下，除了 X 初始值外，还必

须设定 X 轴的另一个参数, 即步长 ΔX 。此时, X 轴的刻度值为 $X=X_0+n*\Delta X$, 其中 n 为数据在数组中的序号 (Index)。

在本例中, 根据要求设定 $X_0=10$ 、 $\Delta X=5$, 然后把它们与数据数组一起打包, 送入 Waveform Graph。当 Waveform Graph 收到这个包后, 会发现这个包里的第 1、第 2 个元素是一个数字, 第 3 个元素是一个数组, 在这种情况下, 它会按 X_0 、 ΔX 、数据数组的方法来处理这个数据包。从显示中可以看到每个数据的刻度与实际的测量时间是一一对应的, 每一个数据的时间刻度为 10 ms, 最后一个数据的时间刻度为 $X=10+29*5=155$ ms。

在使用中有几个值得注意的问题:

- 一是虽然用标度尺可以改变 X 轴的刻度, 但实际上并不会增加或减少实际显示的数据个数;

- 二是在默认的情况下, X 轴的标度总是根据 X_0 、 ΔX 及数据数组的长度自动适应的, 所以 X 轴原点处的刻度不一定是 X_0 , 如图 7.1.4 所示, 当然, 也可以根据特定的需要手动调整 X 轴的刻度设置;

- 三是注意数据打包的顺序不能错, 必须以 X_0 、 ΔX 、数据数组的顺序进行。

在 Y 轴上, 如果配置为默认配置, Y 轴将根据所有显示数据的最大最小值之间的范围进行自动标度。在本例中, 未对 Y 轴的标度进行调整。一般来说, 默认设置可满足大多数的应用, Y 轴的手动设定将在后续内容中介绍。

例 7.1.3 设计一个程序, 测量一个信号的电压值并进行滤波处理 (以前 3 点的平均值作为滤波方法), 要求共测量 30 个点, 不仅要显示出实际的信号波形, 还要同时显示滤波后的信号波形。

在本例中, 要同屏显示两条波形曲线, 此时, 如果没有别的要求, 则只需把两组数据组成一个二维的数组, 再把这个二维数组送入波形显示控件即可, 这是多波形同屏显示最简单的方法。显示结果及程序设计如图 7.1.10 所示。

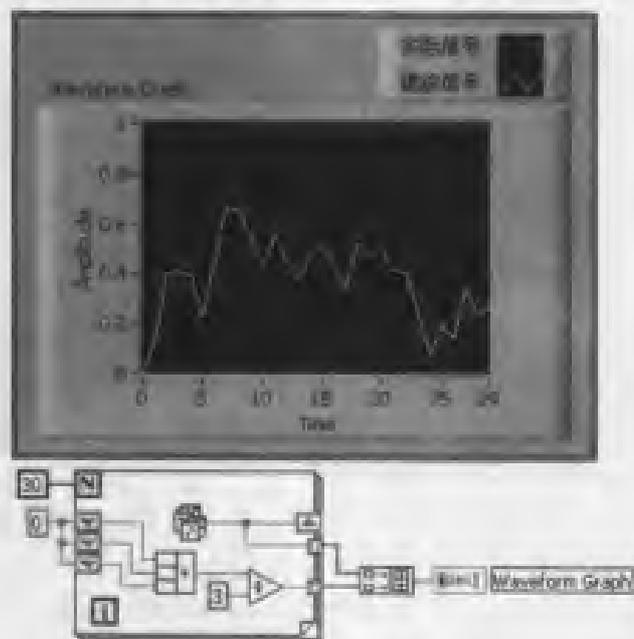


图 7.1.10 例 7.1.3 的前面板及框图程序

Waveform Graph 显示的每条波形, 其数据都必须是一个一维数组, 这是波形显示控件的特点, 所以要显示 N 条波形就必须有 N 组数据。至于这些数据数组如何组织, 可根据不同需要确定。对本例来说, 两组波形数据具有相同的长度, 初始位置 X_0 及标尺 Δx 均采用默认设置, 所以只需把这两组数据组成一个二维数组即可达到要求。

在程序设计中, 除了用一个 Random Number (0-1) 节点模拟测量结果外, 还用到了 For 循环的移位寄存器及加法器, 请参阅本书相关章节的介绍。

例 7.1.4 在上例的基础上, 要求显示每个点的采样时间, 开始测量时间定为 0 ms, 采样间隔为 5 ms。

显示结果及程序设计如图 7.1.11 所示。

本例与上例的不同之处在于, 需要把 X 轴的初始刻度 $X_0=0$ 及标尺 $\Delta x=5$ 组织进数据中。由于两曲线的这两个参数相同, 所以, 只需把 X_0 , Δx 及两条曲线的二维数组打包, 然后送去波形显示控件显示即可。

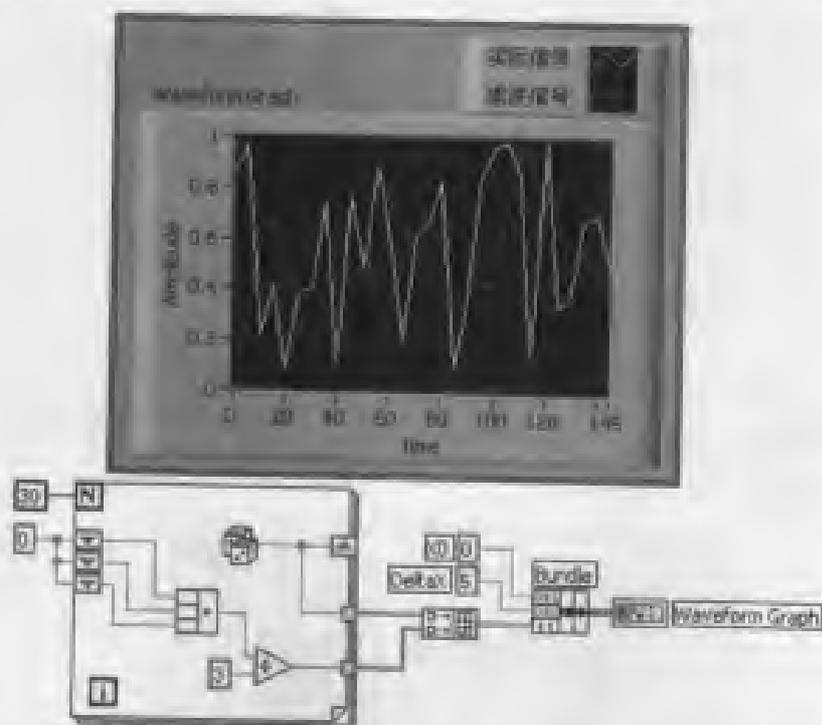


图 7.1.11 例 7.1.4 的前面板及框图程序

例 7.1.5 在一次实验中, 进行了 2 个量的电压采集, 但在相同的时间内, 一个采集了 20 点的数据, 另一个采集了 40 点的数据。用波形显示控件显示实验测量结果。

本例和例 7.1.3 的不同之处在于两组测量数据长度的不一致。能否沿用例 7.1.3 的方法来组织数据, 即把两个长度不一致的一维数据数组组成一个二维数组送去显示呢? LabVIEW 在构成一个二维的数组时, 如果两组数据长度不一致, 整个数组的存储长度将以较长的那组数据的长度为准, 而数据较少的那组在所有的数据存储完后, 余下的空间将被 0 填充。所以, 如果本例仍采用例 7.1.3 的方法来组织数据, 虽然在程序语法上并无错误, 但长度较短的那组在正常的数据显示完后会出现值为 0 的直线拖尾, 如图 7.1.12 所示, 这在观测中将很容易产生歧意。

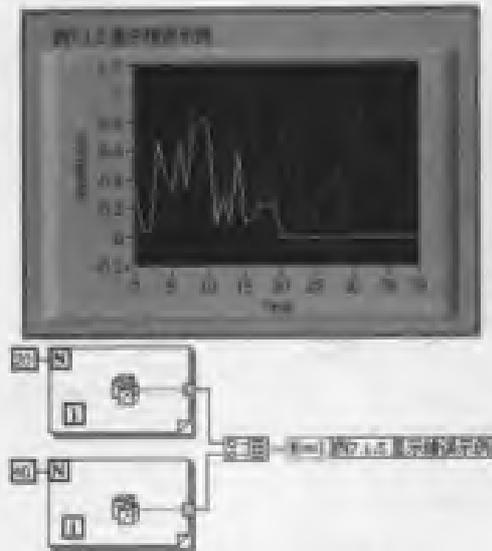


图 7.1.12 错误的组织数据方法的结果显示与框图程序

为了解决这个问题，可以先把两组数据数组打包，然后再组成显示时所需的一个二维数组，前面板和框图程序如图 7.1.13 所示。

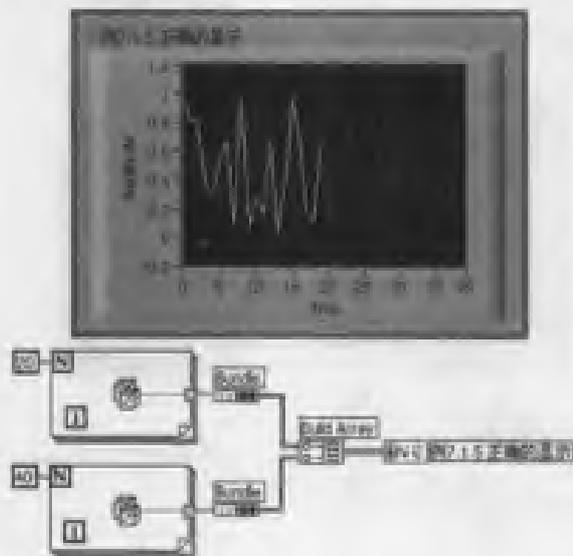


图 7.1.13 正确的组织数据方法的结果显示与框图程序

这种方法容易理解，因为这个二维数组中的元素不是两个一维数组而是两个包，所以程序不会对数组直接进行处理，而是单独处理每个包。在处理包元素时，包里是一个一维的数组，所以 LabVIEW 会将其处理为一条单独的波形。

例 7.1.6 在上例中，假设两个测量信号都有相同的起始测量时间 X_0 及相同的测量间隔 Δt ，要求 X 轴刻度能显示出实际的开始测量时间 X_0 及相应的时间标尺 Δt 。

有了前几例的经验，很自然地会想到，把 X_0 及 Δt 与在上例形成的二维数组进行打包，然后送入波形显示控件，得到如图 7.1.14 所示的框图程序（假设 $X_0=0$ ， $\Delta t=10$ ）。

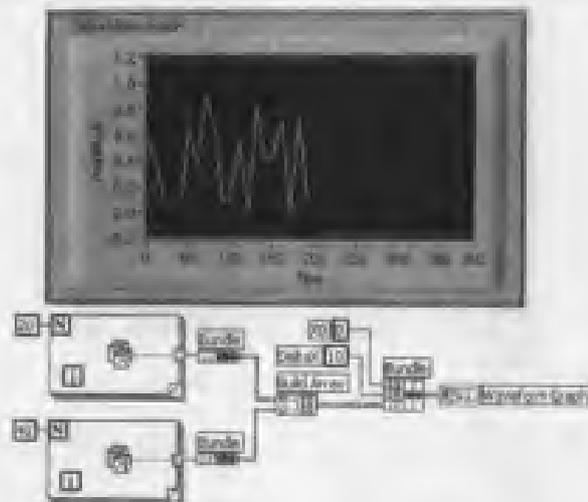


图 7.1.14 例 7.1.6 的前面板和框图程序

例 7.1.7 用一个波形显示控件来显示两次测量的结果。在这两次测量中, 所得的数据个数, 开始测量的时间 X_0 及时间标度 Δt 均不相同。

例 7.1.2 介绍了在单条曲线中如何组织 X_0 、 Δt 及数据数组。在本例中, 把两条不同的曲线分别按例 7.1.2 的方式组织数据, 然后再把结果组成一个二维的数组送去显示, 即能满足题目要求。程序最后设计结果如图 7.1.15 所示。

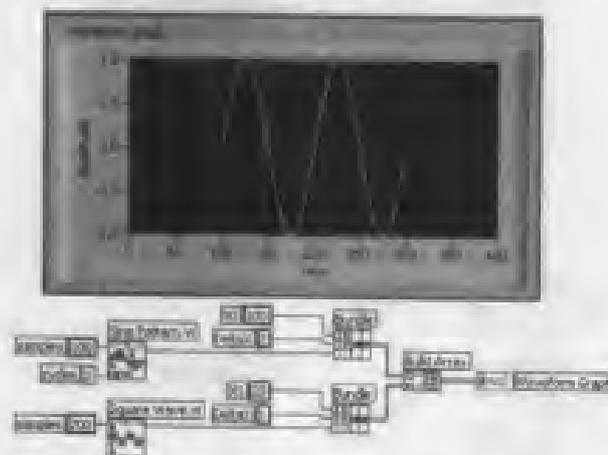


图 7.1.15 例 7.1.7 的前面板和框图程序

应当注意的是, 如果不同曲线间的数据量或数据的大小相差太大, 并不适合用一个波形显示控件进行显示。因为 **Waveform Graph** 总是要在一屏内把一个数组的数据完全显示出来, 如果一组数据与另一组数据的数据量相差太大, 长度短的波形将被压缩, 影响显示效果。

上述几个例子介绍了波形显示控件的应用, 从单条波形显示到多波形的显示, 从 X_0 、 Δt 采用默认值到根据需要自己定义, 都进行了详尽的介绍。但在实际需求中, 这些方法也许不能完全满足要求, 这时可以用以上示例的不同组合来编写在程序。例如, 如果有 4 个通道的测量数据分成 2 组, 组与组之间的数据量、起始时间 X_0 、间隔 Δt 均不相同, 而组内的各参数均相同, 那么就可以先用例 7.1.6 再用例 7.1.7 的方法来组织数据。总之, 以上几个例程是熟练应用 **Waveform Graph** 必须要掌握的。

除了数组和簇, 事后记录波形图还可以显示波形数据。波形数据是 LabVIEW 的一种

数据类型，其本质上还是簇。波形数据包含了信号的起始时间，采样间隔和信号数据。波形数据可以直接送到波形显示控件，并且 LabVIEW 还提供了众多操作波形数据的 VI。有关波形数据的详细介绍请参考本书 5.5 节。

7.1.3 定制事后记录波形图的外观

虚拟仪器与传统仪器相比，最大的一个优点就在于用户可以自定义 VI 功能及外观。波形显示控件的外观可以由用户根据需要自由定制。其显示属性的设置有三种方法：一是通过选择其弹出式选单里的命令来实现；二是可以运用该控件自带的控制工具面板；三是可以在程序中设置其属性节点值来实现。

本节将介绍如何运用这些方法来改变控件的外观。

1. 通过 Waveform Graph 弹出式选单设置

在波形显示控件上单击鼠标右键，弹出波形显示控件的设置选单，如图 7.1.16 所示。在此选单中，可进行下列几项设置。

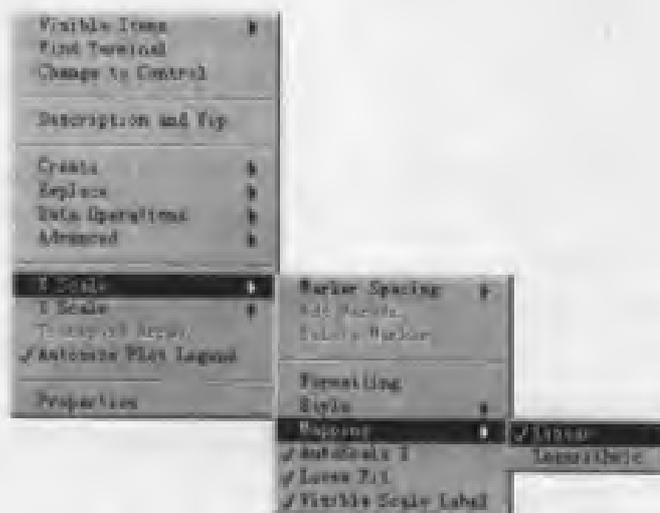


图 7.1.16 Waveform Graph 的右键弹出选单

(1) X 坐标选项 (X Scale)

Marker Spacing: 选择坐标刻度类型。在默认情况下，Waveform Graph 中 X 轴的刻度是根据数组中的数据长度自动标注的，显示区内网格线的位置也是固定的。如果要详细了解所显示波形中某些点的具体变化情况，则可以通过这个选项来任意地标注 X 轴刻度，使网格线恰好落在这些点上。具体操作步骤如下：

① 弹出选单中选择 X Scale → Marker Spacing → Arbitrary，设置刻度类型为任意刻度。此时，Waveform Graph 面板上的 X 轴除了第一个点与最后一个点有刻度外，其余点的刻度均为空白。如图 7.1.17 所示。



图 7.1.17 Waveform Graph 面板上的 X 轴



图 7.1.18 Waveform Graph 面板上的 X 轴

② 在弹出选单中选择 X Scale→Add Marker 添加一个刻度。选中这项后,波形显示控件会在 X 轴的光标停留位置上增加一个刻度及相应的垂直网格线,如图 7.1.18 所示。

③ 调整刻度的位置。有两种方法可以调整该刻度及垂直网格线的位置。一种方法是用 LabVIEW 的文本编辑工具直接改变其刻度值;另一种方法是用文本编辑工具停留在要调整刻度的附近,待光标变成双箭头后,按住鼠标左键并拖拉到任意位置即可。



图 7.1.19 在弹出选单中选择 Delete Marker

④ 如果要删除一个刻度,则用文本编辑工具指向该刻度,在鼠标右键的弹出的选单中选择 Delete Marker 即可,如图 7.1.19 所示。

Formatting: 数据格式设置。选择 Formatting 项,会弹出 Waveform Graph Properties 设置对话框,属性对话框的使用放在稍后的 Properties 设置中介绍。

Style 用于改变 X 轴刻度的标注风格。如是否在刻度旁标注数字,是否在整刻度格内再添加细小刻度等,可根据自己的需要选择。

Mapping...: 用于选择刻度值递增的方法,一种是线性递增 (Linear),这是默认设置;另一种是对数递增 (Log),当输入信号的单位是分贝时选择这种递增方法更符合习惯,如声音的大小或电信号的功率等。

Mapping...: 用于选择刻度值递增的方法,一种是线性递增 (Linear),这是默认设置;另一种是对数递增 (Log),当输入信号的单位是分贝时选择这种递增方法更符合习惯,如声音的大小或电信号的功率等。

Loose Fit 用于取整。在前面所举的各个例子中,Waveform Graph 中 X 轴的刻度并不精确地等于数组中的数据长度。这是因为该控件在 Loose Fit 选项有效 (默认设置) 时,是等宽度地划分刻度的,所以如果数据量不是单位长度的整数倍时,必然使 X 轴长度与数据长度不等。当 Loose Fit 无效时,则要求该控件的 X 轴刻度精确地与波形数据长度一致。

(2) Y 坐标选项 (Y Scale)

Y 坐标选项设置除了对纵轴有效外,其设置方法与 X Scale 完全一致,这里不再赘述。

(3) 清空波形显示区域 (Data Operation→Clear Graph)

一个用波形显示控件输出波形的程序停止运行后,波形显示控件最后一次显示的波形将停留在显示区内。在开始一次新的测量显示之前,如果需要清除上次的显示,这时可以选择该命令来清除原波形,如图 7.1.20 所示,也可以在程序中向 Waveform Graph 发送一个空的数据数组来清除。

(4) 属性设置 (Properties)

在弹出选单中选择 Properties,将弹出 Waveform Graph Properties 设置对话框。如图 7.1.15 所示。与前几个版本不同,LabVIEW 7 Express 将几种属性设置放在同一对话框中,使用起来更加方便。属性对话框共分五页,它们是:外观 (Appearance)、数据格式和精度 (Format and Precision)、线型 (Plots)、刻度 (Scales)、光标 (Cursor)。下面将分别进行介绍。

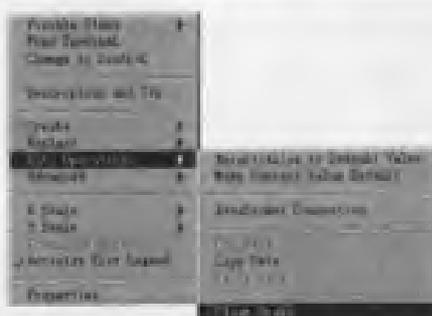


图 7.1.20 在弹出选单中选择 Clear Graph

外观设置 (Appearance)。在外观设置页面中,如图 7.1.21 所示,左上角的 Label 栏,用来设置标签的显示,以及标签的内容。Caption 一栏用来设定标题的显示,以及标题的内容。Enabled State 一栏设定 Waveform Graph 的状态。左下角的几个选项用来设定是否显示特定的工具面板。

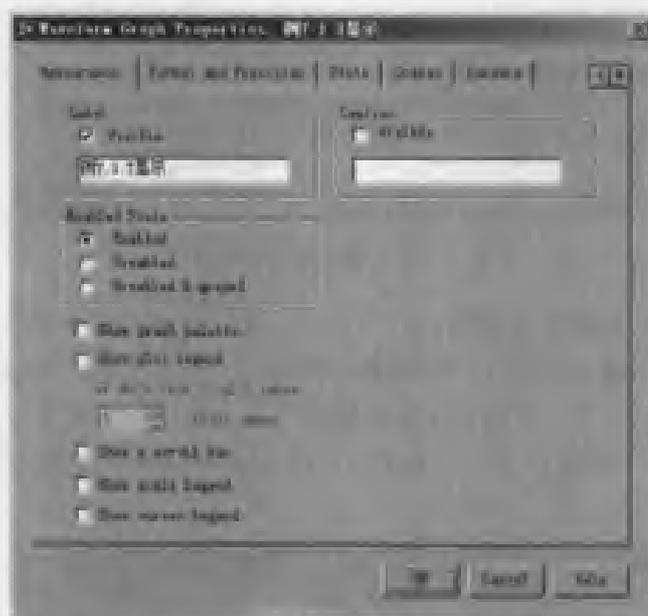


图 7.1.21 外观设置页面

数据格式与精度 (Format and Precision)。在 Formatting and Precision 页面中可以设置两个坐标轴的坐标类型 (数字标度或时间标度)、数据格式、精度、域宽、是否显示无效零等属性,该页如图 7.1.22 所示。

首先在左上角的下拉列表框中选择坐标轴。Time (X-Axis) 为 X 轴, Amplitude (Y-Axis) 为 Y 轴。

下拉列表框的下面是数据格式设定栏,其中:

- Floating Point、Scientific、Automatic formatting、SI notation 为十进制表示;
- Hexadecimal、Octal、Binary 分别为十六进制、八进制和二进制表示;
- Absolute time 为绝对时间,绝对时间的时间零点等于 LabVIEW 7 Express 的系统时间零点,LabVIEW 7 Express 规定系统时间零点为 1904 年 4 月 1 日零时整;
- Relative time 为相对时间。

如果选择数字标度, 数据格式栏的右侧将出现几个选项, 可以进一步设置数据精度、是否显示无效零、指数格式、域宽等属性。如果选择时间标度, 则可以选择标度时间的方法, 比如是以 24 小时制还是以 12 小时制显示, 显示时间精确到秒后几位 (Digits), 年、月、日的排列顺序 (M/D/Y, D/M/Y, Y/M/D) 等。

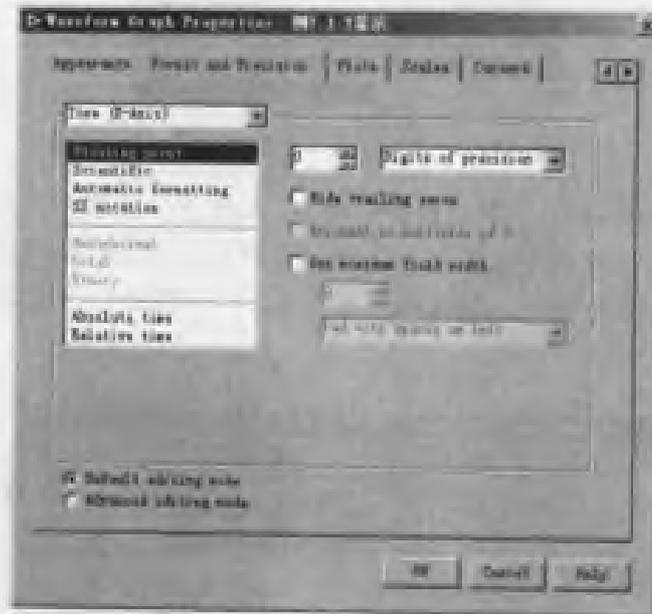


图 7.1.22 数据格式与精度

线条设置 (Plot)。Plot 页面如图 7.1.23 所示, 用于设置与图形线条相关的属性。如果图形中有多条曲线, 从最上面的下拉列表框中选择要设定的曲线。Name 一栏设定曲线名称。在名称下方有四栏选项, 分别用来设定线条类型 (实线, 虚线等)、线宽、点型 (数据点在图中的表示形式)、连线方式。Colors 一栏用于设定线条和数据点的颜色, Fill to 选项设定填充方式。

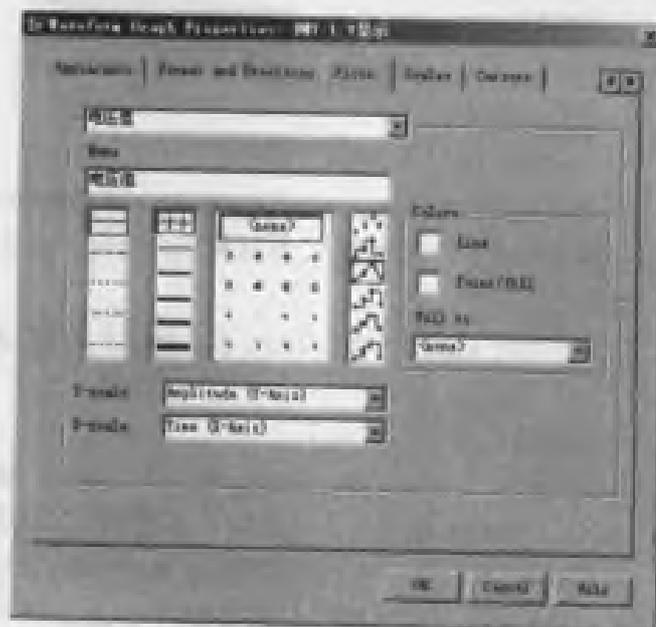


图 7.1.23 线条设置

标尺属性设置 (Scales)。标尺属性设置页面如图 7.1.24 所示。最上方的下拉列表框用于选择当前设定的坐标轴 (X 轴还是 Y 轴)。Name 一栏设定坐标轴的名称。Show scale label 用来设定是否显示坐标轴名称。Show scale 用来设定是否显示标尺。Log 用来设定是否采用对数坐标。Inverted 用来设定是否反转坐标轴方向。Autoscale 用来设定是否自动选择标尺量程。如果为 True, LabVIEW 将自动调整标尺量程以显示所有数据。Minimum 表示标尺最小刻度值。Maximum 表示标尺最大刻度值。Scaling Factors 用于设置默认的显示起始位置 X0 及标尺 DeltaX。在显示单条波形时, 或是显示这两个参数都相同的多条波形时, 在这里设置这两个参数可以简化程序设计。

Scale Style and Colors 一栏用来设定标尺颜色。Grid Style and Colors 一栏用以设定网格线的样式和颜色。单击按钮  在弹出的选单中选择网格线的样式。网格线有 3 种样式, 第一种是不显示网格; 第二种是仅在刻度点上显示网格 (这也是默认设置); 第三种是把相邻刻度之间的空间再均分为 5 份。Major grid color 用来设定刻度处网格线的颜色。Minor grid color 用来设定刻度间网格线的颜色。



图 7.1.24 标尺属性设置

光标设置 (Cursors)。光标设置页面如图 7.1.25 所示。最上方的下拉列表框用于选择当前设置的光标。Name 用来设定光标名称。名称栏下面是 4 项属性选项, 分别设定光标的线型、线宽、光标十字线交点的形状和光标形状。Cursor 用来设定光标颜色。Show name 用来设定是否显示光标名, 若为 True 则在光标交点上方显示光标名。Show cursor 用来设定是否显示光标。Allow dragging 一栏设定光标的拖曳属性。拖曳属性有 3 种模式: Free dragging 模式, 可以自由拖曳; Lock to point 模式, 光标交点只能位于各个数据点; Lock to plot 模式, 光标交点只能位于曲线上。单击 Add 按钮可以添加新的光标, 单击 Delete 按钮可以删除当前光标。



图 7.1.25 光标设置

2. 波形显示控件中工具的使用

与图形显示相关的工具有两个：一个是坐标设置工具 (Scale Legend)，另一个是图形控制工具 (Graph Legend)。可以通过操作波形显示控件的快捷选单 Visible Items 中的相应选项来显示它们，如图 7.1.26 所示，图右侧上方为 Scale Legend，下方为 Graph Legend。波形显示控件的控制模板不但可以快捷地调整控件外观，还可以在程序运行过程中实现波形的动态调整。下面逐项介绍其用法。

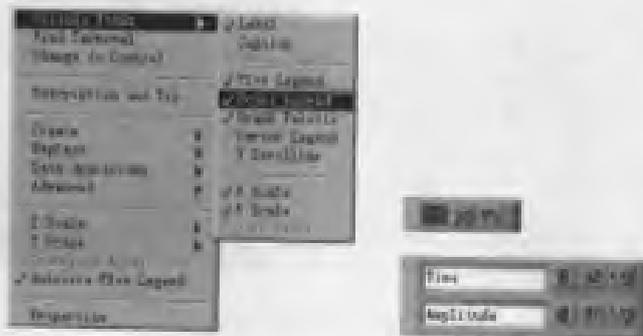


图 7.1.26 显示 Waveform Graph 的工具

(1) 自动缩放

按钮的功能是 AutoScale X，按钮的功能是 AutoScale Y。按钮所示的锁定状态为默认设置，此时 AutoScale X 与 AutoScale Y 有效。单击该按钮可以将其解锁为，即 AutoScale X 及 AutoScale Y 无效，此时可以手动改变 X、Y 轴的刻度设置。

(2) 数字标度设置

按钮和按钮用来设置 X 和 Y 轴刻度数字标度方式。单击后将弹出一个如图 7.1.27 所示的设置选单，其选项内容与 X Scale→Formatting→Format & Precision 一样，这里不再赘述。



图 7.1.27 在弹出选单中设置 X、Y 轴刻度值数字标度方式

(3) 图形拖动

使用工具可以在波形显示区域内随意拖动波形。用数据操作工具单击它，把鼠标移到显示区内，鼠标将变成手状，按住鼠标左键不放，可以把波形“粘”住并在显示区内自由移动。

(4) 光标移动

按钮是光标移动工具，当处于选中状态时，可以通过鼠标拖曳光标。

(5) 图形缩放

按钮是波形缩放工具。当用数据操作工具单击它时，可弹出波形缩放方式的选择项，如图 7.1.28 所示。



图 7.1.28 图形缩放方式

各选项功能如表 7.1.1 所示。

表 7.1.1 图形缩放方式

图 标	功 能
	矩形缩放。选择该项后，在显示区上，按住鼠标左键拉出一个方框，方框内的波形将被放大。
	水平缩放。波形只在水平方向上被放大，垂直方向上保持不变。
	垂直缩放。波形只在垂直方向上被放大，水平方向上保持不变。
	自动缩放。自动调整图形使其充满显示区。
	连续放大。选中该项后，在显示区内按住鼠标左键，波形将以鼠标指针停留位置为中心连续地放大。
	连续缩小。

3. 波形设置

波形曲线的线型、颜色和填充模式等属性可以通过单击控件右上角的 Plot Legend 进行设置。选择弹出选单 Visible Items → Plot Legend，使 Plot Legend 可见。在默认条件下，Plot Legend 显示一条波形曲线。如果图形显示有多条曲线，那么可以将 Plot Legend 拉大，显示更多的曲线，如图 7.1.29 所示。



图 7.1.29 显示 3 条波形的 Plot Legend

在显示多条波形曲线情况下,不同信号波形的区分是必要的。比较好的方法是每条波形都用不同的颜色或用不同的线型来表示,再根据信号代表的实际物理量名称给每条波形赋一个一目了然的名字。在默认条件下将按照 Plot 0, Plot 1...给每条曲线命名。

用鼠标单击 Plot Legend 中的波形线条图标,将弹出控件的图例设置项,如图 7.1.30 所示。



图 7.1.30 图例设置选单

(1) 图形表示方式 (Common Plots)

用于确定波形曲线的式样。系统提供了数据点光滑曲线拟合、直方图、数据点直接连接和填充模式等 6 种常见的图形供用户选择,每种选择的线型、线宽都是预先设置的。默认方式是光滑曲线拟合。

(2) 颜色 (Color)

选取波形曲线的颜色。

(3) 线型 (Line Style)

该选项提供数据连线线型,如实线、虚线、点划线等。

(4) 线宽 (Line Width)

用于选择连线线宽。注意第一个选项是“极细线”(Hairline),如果选择这种线宽,在屏幕显示上不会有什么变化(一个点宽),如果打印机支持 Hairline 的打印,则可以以极细的线型输出波形。

(5) 反走样 (Anti-Aliased)

选中此项将使波形曲线更加光滑,但会降低绘图的速度。

(6) 直方图 (Bar Plots)

可选择多种直方图的绘制方式,包括水平直方图和垂直直方图等。

(7) 填充模式 (Fill Baseline)

可选择填充基线,填充时是从波形开始向某条水平基线垂直填充的。基线有 3 个选择: $Y=0$, $Y=+\infty$ (Infinity), $Y=-\infty$ (-Infinity)。

(8) 连线方式 (Interpolation)

选择数据点之间的连线方式,可以选择不连线,只显示散布的数据点 (Scatter point),也可以选择仅以简单的直线或折线相连。

(9) 数据点风格 (Point Style)

该选项提供了数据点形状选择,如实心、空心、圆点、方点等。

(10) X 轴 (X Scale)

设置与 X 轴相连的变量。只有一个选项时间 Time。

(11) Y轴 (Y Scale)

设置与Y轴相连的变量。只有一个选项幅值 Amplitude。

7.2 实时趋势图

本节介绍 LabVIEW 的另一种波形显示控件，即实时趋势图控件 (Waveform Chart)。与前面介绍的波形显示控件一样，它也可显示一个或多个的波形，其前面板及端口如图 7.2.1 所示。



图 7.2.1 Waveform Chart

从图 7.2.1 中可以看出，实时趋势图控件与波形显示控件面板不同，实时趋势图控件的输入是一个双精度浮点数，而波形显示控件的输入是一个双精度浮点数组。这主要是由于两者的波形刷新方式和数据组织方法是不一样的。

波形显示控件通常把显示的数据先收集到一个数组中，然后再把这组数据一次性送入波形显示前面板对象中进行显示；而实时趋势图控件是把新的数据连续添加到已有数据的后面，波形是连续向前推进显示的，这种显示方法可以很清楚地观察到数据的变化过程。

实时趋势图控件一次可以接收一个点的数据，也可以接收一组数据。不过，这组数据与波形显示控件的数据数组在概念上是不同的。实时趋势图控件的数据只不过是代表一条波形上的几个点，而波形显示控件的数据代表的则是整条波形。

实时趋势图控件内置了一个显示缓冲器，用来保存一部分历史数据，并接收新数据。这个缓冲区的数据存储按照先进先出 (FIFO) 的规则管理，它决定了该控件的最大显示数据长度。在默认情况下，这个缓冲大小为 1KB，即最大的数据显示长度为 1024 个。

实时趋势图控件适合用于实时测量中的参数监控，而波形显示控件适合用在事后数据显示与分析。

7.2.1 使用实时趋势图

现在通过几个例程来详细介绍实时趋势图控件的使用方法。

例 7.2.1 用实时趋势图控件实时监测一个温度测量输出。

程序及显示结果如图 7.2.2 所示。温度信号为一随机数。



图 7.2.2 例 7.2.1 的前面板及框图程序 1

例 7.2.1 把 Random Number (0-1) 节点的输出直接送给实时趋势图控件, 当其接收到数据后, 从第 0 个数据开始显示。在本例中, 屏幕宽度与缓冲器的大小都使用了默认值, 分别为 100 个点和 1 024 个点。显示宽度可用文本编辑工具改变, 但不能超过缓冲区大小。当显示的数据量超过 100 后, 在接收到下一个数据时, 波形将自动左移一位, 而 X 轴上起始数据序号与最后位置的数据序号都要加 1, 即当第 101 位数据到来时, X 轴的开始刻度为 1, 最后的刻度为 101, 坐标宽度保持 100 不变。

如果需要实时趋势图控件一次接收一部分数据, 要先把 10 个测量数据形成一个数组, 然后再送到实时趋势图控件中显示。程序将如图 7.2.3 所示。当把一个数组连到 For 循环框架时, For 循环的 Auto Indexing 功能会自动将浮点数据组成一个浮点数组。此时, 当数据量超过 100 后, 在接收到下一个数据组时, 波形每次将以 10 个点向左移动, 可以看出, 这是实时趋势图控件与波形显示控件的又一个不同之处。

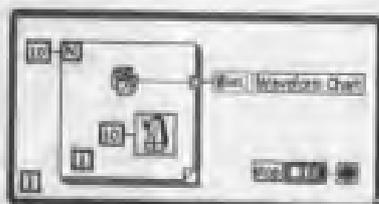


图 7.2.3 例 7.2.1 框图程序 2

例 7.2.2 用实时趋势图控件来显示两个测量结果的波形输出。

用实时趋势图控件同样可以同屏显示多个信号波形, 其数据组织方法主要有以下几种:

① 把每种测量的一个点打包在一起, 然后把该数据包送到实时趋势图控件中显示。这是最简单, 也是最常用的方法。利用这种方法, 波形是通过单个点的平移刷新的。框图程序如图 7.2.4 所示。

② 先对单个点进行打包, 但不是直接送去显示, 然后将这些数据包再组成一个数组, 最后送到实时趋势图控件中显示, 如图 7.2.5 所示。在此例中, 每 10 个点显示一次。

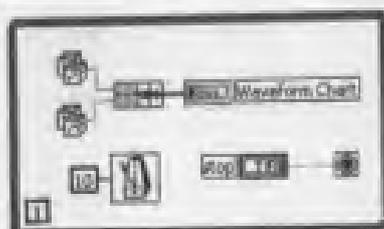


图 7.2.4 例 7.2.2 框图程序 1

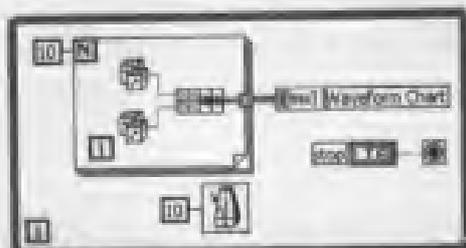


图 7.2.5 例 7.2.2 框图程序 2

7.2.2 定制实时趋势图的外观定制

在控件面板上右击鼠标，弹出定时趋势图控件的设置选单，如图 7.2.6 所示。该选单中大部分设置项与波形显示控件的选单设置项在意义及用法上都是一致的，Graph Palette、Plot Legend 也与波形显示控件一致。所以，这里只介绍定时趋势图控件的一些特殊设置项，其他项的设置请参阅 7.1 节有关内容。

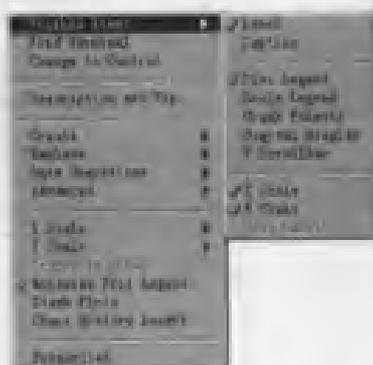


图 7.2.6 实时趋势图控件的设置选单

1. 数字显示

定时趋势图控件是以一次一个点或几个点的方式来接收数据的。在 Waveform Chart 的右键弹出选单中选择 Visible Items → Digital Display 后，Waveform Chart 将在前面板外附加一个数字指示器 (Digital Indicator)，这个指示器直观地显示了最新显示的一个数据的大小，如图 7.2.7 所示。如果有多条波形，则每条波形都可以有一个对应的数字指示器。



图 7.2.7 数字指示器及滚动条

2. 滚动条

定时趋势图控件有一个数据缓冲区。若要显示滚动条,在定时趋势图控件的右键弹出菜单中选择 Visible Items → X Scrollbar。当这个选项有效时,定时趋势图控件可以用一个滚动条来查看缓冲区内前后任何位置的一段数据波形,如图 7.2.7 所示。

3. 波形刷新模式

在定时趋势图控件的右键弹出菜单中选择 Advanced → Update Mode。这个选项提供了 3 种刷新方式,如图 7.2.8 所示。



图 7.2.8 三种波形刷新方式

这三种模式的说明如表 7.2.1 所示。

表 7.2.1 波形刷新方式

名称	图标	说明
Strip Chart Mode		默认模式。在这种情况下,波形从左向右开始绘制,当最新的一个点超出显示区右边界时,整个波形顺序左移
Scope Chart Mode		这种模式下,波形同样由左边界开始绘制,当最新一点超出显示区右边界时,整个波形将被清除刷新,波形显示从左边界重新开始
Sweep Chart Mode		当选用这种模式时,波形由左开始绘制,当最新的一点超出右边界时,新的点从左边界重新开始绘制,原有的波形由一条垂直扫描线从左至右逐渐清除,为新的点腾出显示位置

以上 3 种波形的刷新在模式上虽不一样,但它的缓冲更新是不变的,都遵循先进先出规则 (FIFO)。

4. 多层图

在默认条件下,定时趋势图控件将在相同的纵坐标下显示多条波形曲线。如果这些测量信号的大小范围相差比较大或是显示量纲不同,那么在相同纵坐标下,就可能出现信号显示不匹配的情况。针对这种情况,定时趋势图控件专门提供了多层图选项,允许不同信号在不同的纵坐标设置下进行显示,显示效果如图 7.2.9 所示。设置方法是在定时趋势图控件的右键弹出菜单中选择 Stack Plots。

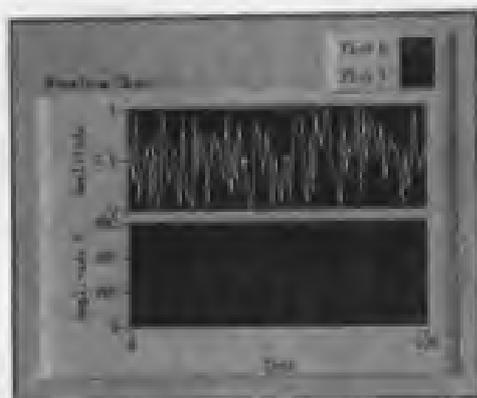


图 7.2.9 多层图

如果选择 Stack Plots 有效, 每个波形的 Y 轴值就可以单独设置, 但 X 轴的设置是共用的。

5. 历史记录长度

该选项用于设置缓冲区的大小, 默认值为 1024 点的浮点数。缓冲区越大, 保留的历史数据就越多, 但也要注意实际系统的物理内存大小, 否则将引起系统性能的下降。设置方法是在定时趋势图控件的右键弹出菜单中选择 Chart History Length。

7.3 XY 波形图

7.1 节介绍了波形显示控件的特点与应用。波形显示控件的 Y 值对应实际的测量数据, X 值对应测量点的序号, 适合显示等间隔数据序列的变化。比如按照一定采样时间采集的数据的变化。但是它不适合描述 Y 值随 X 值变化的曲线, 对于这种曲线, LabVIEW 专门设计了 XY 波形记录控件 (即 XY Graph), XY Graph 及其端口如图 7.3.1 所示。

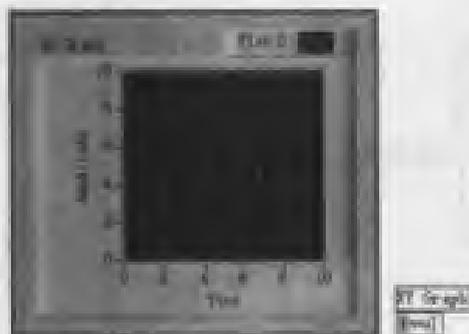


图 7.3.1 XY Graph 及其端口

与 Waveform Graph 相同, XY Graph 也是一次性完成波形显示刷新, 不同的是, XY 波形记录控件的输入数据类型是由两组数据打包 (bundle) 构成的簇, 簇的每一对数据都对应一个显示数据点的 X, Y 坐标。下面通过几个例程来详细介绍 XY 波形记录控件的使用方法。

例 7.3.1 用输入作 XY Graph 的 X 轴, 输出作 XY Graph 的 Y 轴, 观察输入输出之间的关系。

XY Graph 的 X 轴和 Y 轴都是受控的, 所以要求有两组输入数据, 问题是如何来组织这两组数据。LabVIEW 提供了两种方法, 分别如图 7.3.2 所示的两个框图程序。

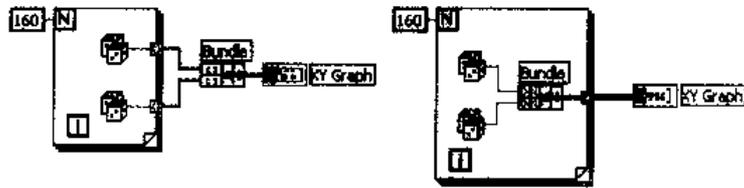


图 7.3.2 例 7.3.1 框图程序

第一个程序把两组数据数组打包后送给了 XY Graph, 此时, 两个数据数组里具有相同序号的两个数据组成一个点, 而且必定是包里的第一个数组对应 X 轴, 第二个数组对应 Y 轴。使用这种方法来组织数据, 要确保两组数据长度相同, 如果两组数据的长度不一样, XY Graph 将以长度较短的那组为参考, 而长度较长的那组多出来的数据将被抛弃。

第二个程序先把每一对坐标点 (X, Y) 打包, 然后用这些点坐标形成的包形成一个数组, 再送到 XY Graph 中显示, 这种方法可确保两组数据的长度一致。在实际显示效果上, 这两种方法是一样的。

上面介绍了 XY Graph 用于显示单一波形的数据组织方法。同样, 它也可用来显示多条波形。下面通过举例来介绍显示多条波形的数据组织方法。

例 7.3.2 在一次数据测量中, 有两个信号, 请设计一个 LabVIEW 程序, 用 XY Graph 显示出这两个信号的输入输出之间的关系。

在显示多条曲线的情况下, 基本上只有两种数据组织方法, 对应的框图程序如图 7.3.3 所示。

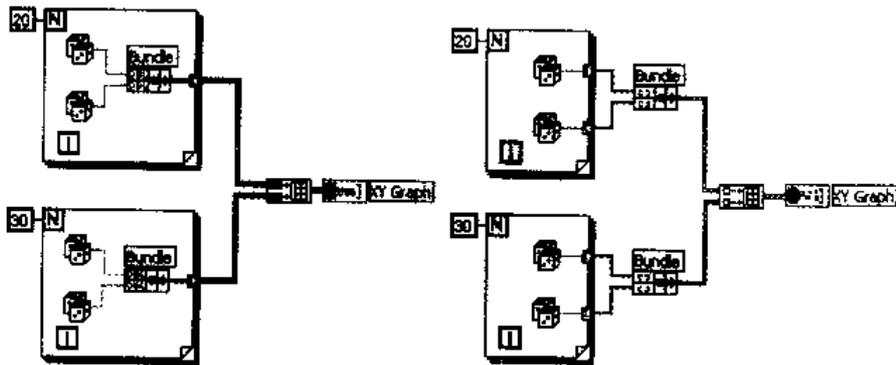


图 7.3.3 例 7.3.2 框图程序

第一个程序先把两组测量结果的各个数据打包, 然后分别在两个 For 循环的边框通道上形成两个一维的数组, 再把这两个一维数组组成一个二维数组送到 XY Graph 中显示。

第二个程序先让两组的输入输出在 For 循环的边框通道上形成数组, 然后分别打包, 用一个二维数组送到 XY Graph 中显示, 这种方法较为直观。

XY Graph 的选单设置、Scale Legend、Graph Palette、Cursor Legend 以及 Plot Legend 的使用与 Waveform Graph 完全一致, 请参阅 7.1 节的相关内容。

7.4 密度图形显示控件 (Intensity Graph)

密度图形显示控件, 是 LabVIEW 提供的另一种波形显示控件。它用一个二维密度图表达一个三维的数据类型。一个典型的 Intensity Graph 及其端口如图 7.4.1 所示。

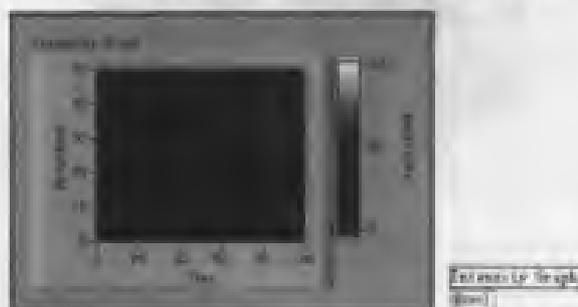


图 7.4.1 Intensity Graph 及其端口

7.4.1 使用密度图

从密度显示控件的前面板可以知道, 它接收的是一个大小为 $N \times M$ 的二维双精度浮点数据数组, 而密度显示控件的波形显示区域划分成 $M \times N$ 个小方格, 显示区的每一列方格对应着数据数组中的一行数据, 每个方格用不同的颜色来填充, 以表示所对应单元数据值的大小。波形显示区域方格位置与输入数组单元的具体对应关系如图 7.4.2 所示。



图 7.4.2 显示区方格位置与输入数组单元对应关系

从图 7.4.2 中可以看出, 密度图形显示控件的 Y 轴对应于数据数组中的行, X 轴对应于数据数组中的列。把数组中的数据也当做一维, 则 Intensity Graph 的显示区可称为 Z 轴, 数组中的数据即为 Z 轴的输入。

例 7.4.1 Intensity Graph 应用举例。

这是一个典型的程序设计, 从中可以看出如何使用 Intensity Graph。在这个程序中, 利用了 2 个 For 循环构造了一个 5×10 的二维数组, 所以在显示屏上共有 5 列, 每列的高度为 10。程序及显示结果如图 7.4.3 所示。

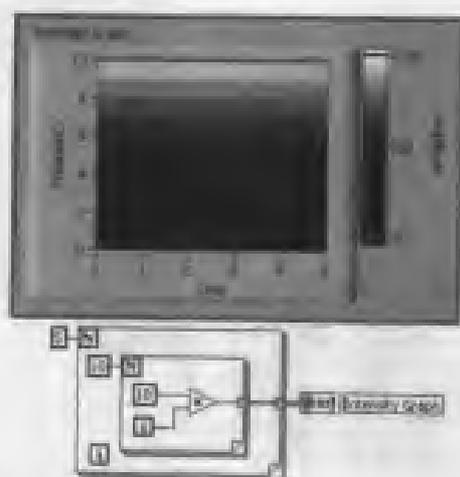


图 7.4.3 例 7.4.1 的前面板及框图程序

7.4.2 定义密度图的颜色

在默认设置下, Intensity Graph 波形显示区域的右侧有一个颜色映射条 (Color Mapping Scale), 因为它标示出了输入数据大小与波形显示区域显示颜色之间的对应关系, 所以称为 Z 轴的“数值-颜色” (Value-Color) 对应关系。这个颜色条其实是一个颜色控件 (Color Ramp Control), Z 轴的“数值-颜色”对应关系可以根据需要自行定义, 这里介绍 2 种方法。

方法 1: 通过其弹出式选单进行设置。如增加一个刻度, 选择一个刻度对应的颜色 (Marker Color), 是否用插值 (Interpolate) 来平滑颜色过渡等操作。如图 7.4.4 所示。



图 7.4.4 在选单中设置 Z 轴的“数值-颜色”对应关系

方法 2: 通过 Intensity Graph 的 Color Table 属性节点来改变数值颜色的对应关系。这个节点的输入为一个大小为 256 的整数数组, 这个数组其实是一张颜色列表, 它与 Z 轴的刻度一起决定了颜色映射条上的“数值-颜色”对应关系。在颜色条上可以定义上溢出和下溢出的数值大小, 颜色表数组中序号为 0 的单元里的数据对应为下溢出时的颜色, 序号为 255 的单元里的数据对应为上溢出时的颜色, 而序号为 1~254 中的数据从颜色条中最大最小值之间按插值的方法进行对应。

例 7.4.2 密度显示控件应用举例。

分析这个例程, 具体地来看在程序中如何使用 Color Table 来改变“数值-颜色”对应关系。请先看图 7.4.5 所示的程序前面板。



图 7.4.5 例 7.4.2 的前面板

在前面板中有一个颜色框 Base Color，它有时用来指定基本色的，在本例中，颜色表是在基本色的基础上修改而得到的。框图程序如图 7.4.6 所示。

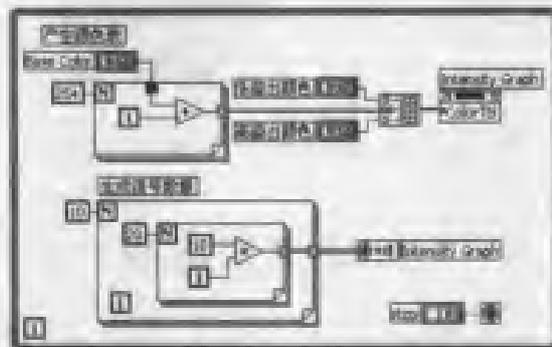


图 7.4.6 例 7.4.2 的框图程序

程序中用了一个 For 循环来定义一张颜色表：For 循环产生大小为 1~254 的 254 个颜色值，这些值与下溢出颜色和上溢出颜色共同构成一个容量为 256 的数组送到 Color Table 属性节点中，这个表中的第一个及最后一个颜色值，分别对应 Z 轴上溢出及下溢出时的颜色值。当 Color Table 属性节点有赋值操作时，颜色表被激活，同时 Ignore Array 也被自动设为 ON。此时，Z 轴的“数值-颜色”对应关系由颜色表来决定。

7.4.3 设置密度图的外观

Intensity Graph 的外观设置与 Waveform Graph 有很多相似的地方，如光标的设置、坐标轴的设置等。由于 Intensity Graph 是以颜色块来表示数据的大小的，因而它没有属性设置，但它增加了一个 Z 轴的设置，如图 7.4.7 所示。

Z 轴的设置与 X、Y 轴的设置项目有些不同，因为它是以颜色来表示数据的，同时它还是一个坐标轴，所以，它除了有颜色设置项目外，还有通用坐标轴的设置项目和前述一些颜色条的设置项目，这些设置的用法都已详细介绍过，这里不再赘述。

Intensity Graph 的这种显示三维数据的方法使用很简单，不同的颜色表示了不同数值的大小，它如果用来表示一个平面里的某种量的密度变化是最合适不过的了（如一个平面的

温度场、电磁场)。但是其局限性也是很明显的:首先,因为 X、Y 这两维的数据是固定的整数(为数组的行列序号),因而不具有三维数据的代表性;其次,它显示的结果只能看到 Z 轴的数据变化情况,并不具有三维的立体感。

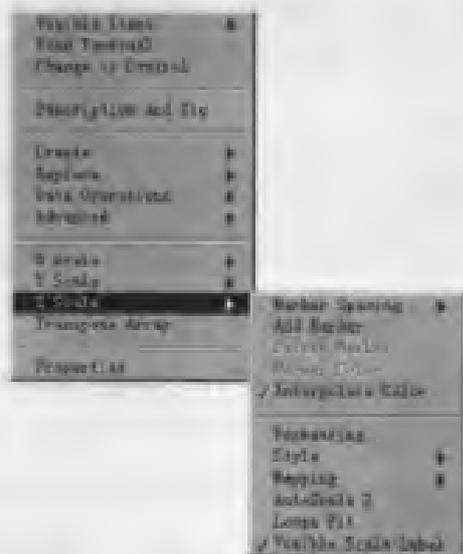


图 7.4.7 Intensity Graph 的右键弹出选单

7.5 密度趋势图

与 Intensity Graph 一样,密度趋势图也是用一个二维的显示结构来表达一个三维的数据类型,它们之间的主要区别在于图像刷新方式的不同,也就是 Chart 与 Graph 的区别。

第 7.4 节介绍了 Intensity Graph 的数据格式为一个二维的数组,它可以一次性把这些数据显示出来。虽然 Intensity Chart 也是接收和显示一个二维的数据数组,但它显示的方式不一样,它可以一次性显示一列或几列图像,它在屏幕及缓冲区保存一部分旧的图像和数据,每次接收到新的数据时,新的图像紧接着在原有图像的后面显示。当下一列图像将超出显示区域时,将有一列或几列旧图像移出屏幕。数据缓冲同 Waveform Chart 一样,也是先进先出(FIFO)的,大小可以自己定义,但结构与 Waveform Chart 不一样,是一个二维的缓冲结构,而 Waveform Chart 缓冲区结构是一维的。

例 7.5.1 一个典型的 Intensity Chart 程序设计。

前面板及其框图程序如图 7.5.1 所示。

在这个程序中,先让正弦函数在循环的边框通道上形成一个一维数组,然后再形成一个列数为 1 的二维数组送到控件中去显示,因为二维数组是 Intensity Chart 所必需的数据类型,所以,即使只有一行数据,这步工作也是必要的。此例中的 Intensity Chart 一次只显示一列图像。其显示结果见图 7.5.1 中的前面板(图中显示的是从 33 列到 133 列的图像)。

Intensity Chart 的 Z 轴“数值-颜色”对应关系的设置与 Intensity Graph 的设置完全一致;关于其外观的各种设定,有些在 Waveform Chart 中介绍过,有些在 Intensity Graph 中介绍过,这里不再赘述。

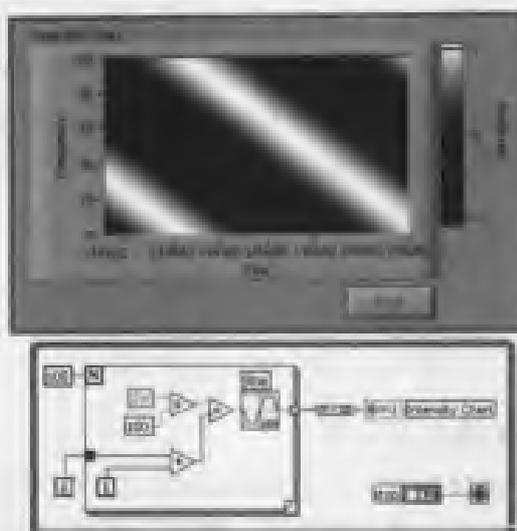


图 7.5.1 例 7.5.1 的前面板及框图程序

7.6 三维曲面图

三维曲面图, 即 3D Surface Graph 用于显示三维空间的一个曲面。3D Surface Graph 是一个 ActiveX 控件, 提供了用于作图的属性和方法。在前面板放置一个三维曲面控件时, 框图中将出现两个图标, 如图 7.6.1 所示, 一个是 ActiveX 控件的图标, 另一个是 3D Surface.vi, 该 VI 负责三维作图。

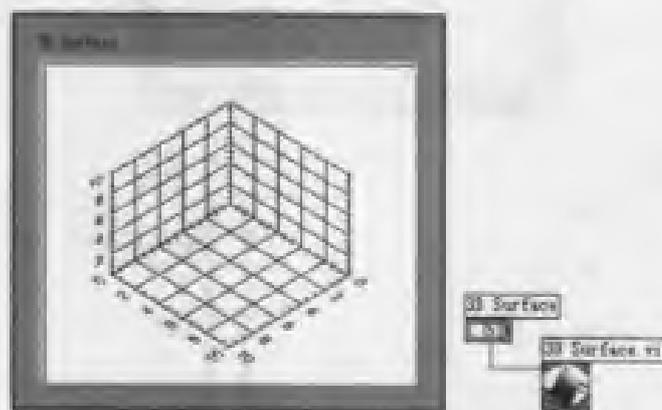


图 7.6.1 3D Surface Graph

7.6.1 使用三维曲面图

Activex 控件 3D Surface Graph 只负责图形显示, 作图则由 3D Surface.vi 负责。3D Surface.vi 的输入端口如图所示, 3D Graph 端口是 ActiveX 控件输入端, 该端口的下面是两个一维数组输入端, 用以输入 X, Y 坐标值。端口 z matrix 的数据类型为二维数组, 用以输入 Z 坐标。3D Surface.vi 在作图时采用的是描点法, 即根据输入的 X, Y, Z 坐标在三维空间确定一系列数据点, 然后通过插值得到曲面。在作图时, 3D Surface.vi 根据 X, Y 坐标数组

在 XY 平面上确定一个矩形网格, 每个网格结点都对应着三维曲线上的一个点在 XY 坐标平面的投影。z matrix 数组给出了每个网格结点所对应的曲面点的 Z 坐标, 3D Surface.vi 根据这些信息就能够完成作图。3D Surface Graph 不能显示三维空间的封闭图形, 如要显示封闭图形可以使用三维参数曲面控件, 三维参数曲面控件将在第 7.7 节中介绍。



图 7.6.2 3D Surface.vi 的图标

例 7.6.1 3D Surface Graph 使用举例。

该例中要显示曲面 $z = \sin(x)\cos(y)$, $x, y \in [0, 2\pi]$, X, Y 坐标的步长为 $\pi/50$ 。本例的前面板和框图程序如图 7.6.3 所示, 使用两个 For 循环, 计算曲面在每个网格结点的 Z 坐标, 然后通过 For 循环边框的 Indexing 功能将 Z 坐标组成一个二维数组, 送到 3D Surface.vi 显示。在本例中, X 坐标数组和 Y 坐标数组是相同的。

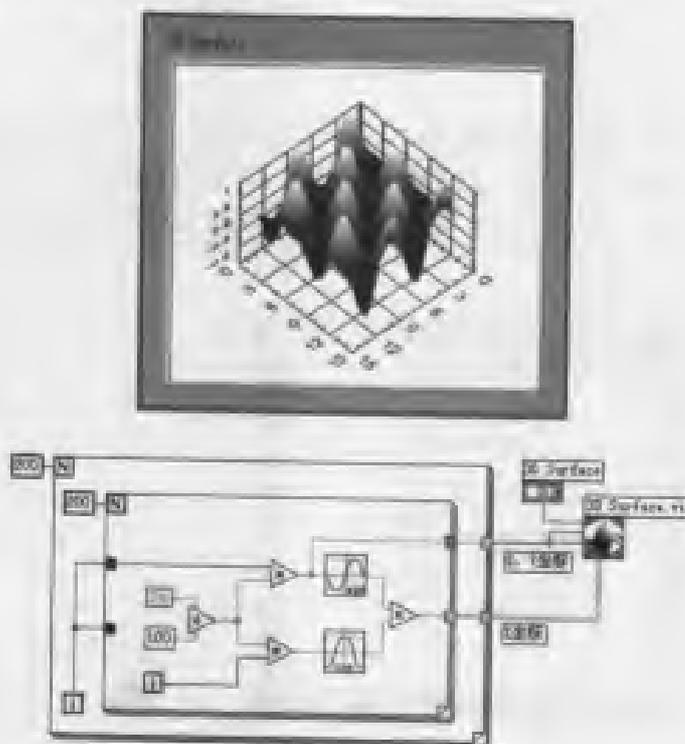


图 7.6.3 例 7.6.1 的前面板和框图程序

将鼠标放在前面板 3D Surface Graph 上, 按下鼠标左键并移动鼠标可以改变视点位置, 如图 7.6.4 所示。

如果鼠标带有滚轮, 在 3D Surface Graph 上单击鼠标左键, 然后转动滚轮可以对图形进行缩放, 如图 7.6.5 所示。

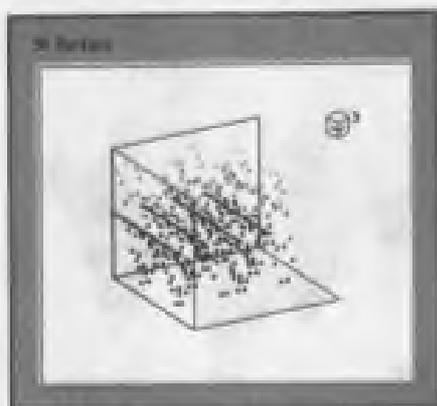


图 7.6.4 3D Surface Graph 的旋转操作



图 7.6.5 3D Surface Graph 的缩放操作

3D Surface Graph 还可以显示光标。光标可用于测量曲面上点的坐标。首先要添加光标，方法是在 3D Surface Graph 上单击鼠标右键，从弹出选单中选择 CWGraph3D→特性 (P)...，在弹出的属性对话框中选择光标设置页 Cursors，如图 7.6.6 所示，同时还将显示一个光标预览窗口。单击左边 Cursor 列表框下的 Add 按钮添加光标。Add 按钮下面的文字框可以编辑光标的名称。在对话框的右边还可以设置光标的其他属性。设置完毕后单击“确定”按钮，光标就添加到图形中了，如图 7.6.7 所示。在使用时，用鼠标指向光标原点，按下鼠标左键，可以拖动光标。



图 7.6.6 光标属性设置对话框



图 7.6.7 在 3D Surface Graph 中使用光标

7.6.2 设置三维曲面图的外观

三维曲面图有两种属性设置的方法：一是通过弹出选单设置，二是通过属性结点在程序运行时设置。本节介绍第一种方法。在三维曲线控件面板上单击鼠标右键将弹出快捷选单，如图 7.6.8 所示。

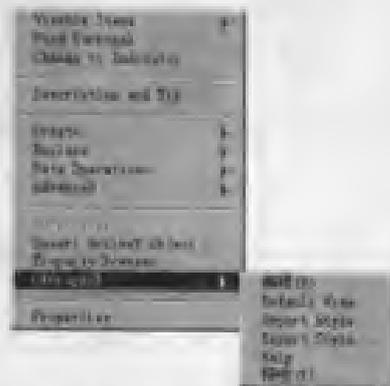


图 7.6.8 3D Surface Graph 属性设置快捷选单

1. 属性浏览器

在前面提到过 3D Surface Graph 是 ActiveX 控件，就是在 ActiveX 容器中插入了 CWGraph3D 控件。设置 3D Surface Graph 的外观主要是设置 CWGraph3D 控件的外观。通过 Property Browser 可以浏览和修改 CWGraph3D 控件的属性。在 3D Surface Graph 上单击鼠标有右键，在弹出的快捷选单中选择 Property Browser...，打开属性浏览对话框，如图 7.6.9 所示。对话框分两栏，第一栏是属性名，第二栏是相应的属性值。可以在对话框中直接修改属性值，有些属性在 Property Browser 对话框中是只读的。

例如，要设置 CWGraph3D 控件的标题为“三维曲面”，并将其颜色设为绿色。可以这样操作：在 Caption 属性值一栏中键入“三维曲面”，在 CaptionColor 属性值一栏中单击鼠标左键，属性栏右侧将出现一个下拉箭头，单击该箭头，在弹出的颜色对话框中选择绿色，更改后的效果如图 7.6.10 所示。

通过 Property Browser 对话框设置的方法还不够直观，更常用的方法是通过

CWGraph3D 的属性对话框进行设置，下面就介绍这种方法。

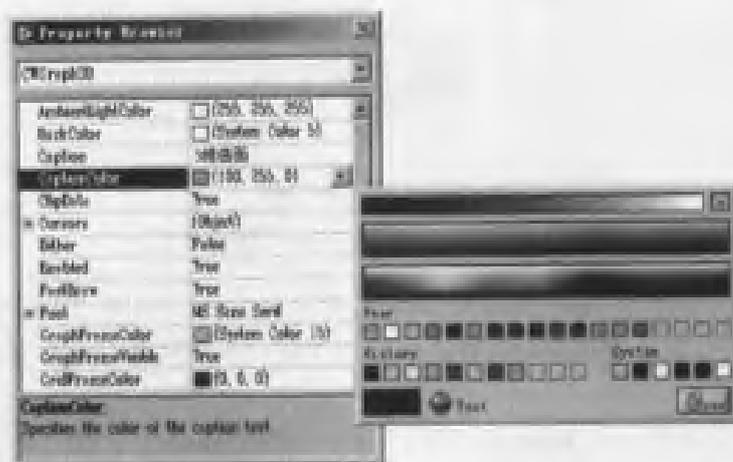


图 7.6.9 属性浏览对话框

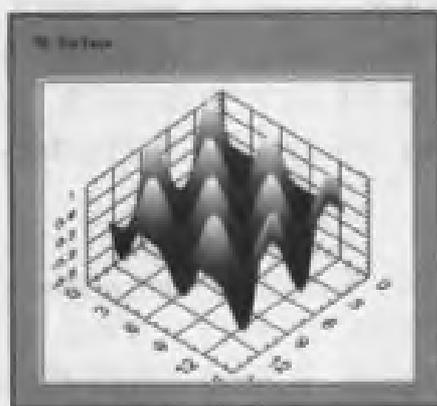


图 7.6.10 属性设置效果

2. 设置 CWGraph3D 属性

在 3D Surface Graph 上单击鼠标右键，从弹出选单中选择 CWGraph3D→特性(P)...，将弹出 CWGraph3D 控件的属性设置对话框，如图 7.6.11 所示，同时会出现一个小的 CWGraph3D 控件面板，如图 7.6.12 所示，所有的属性设置都会在该面板实时显示出来。

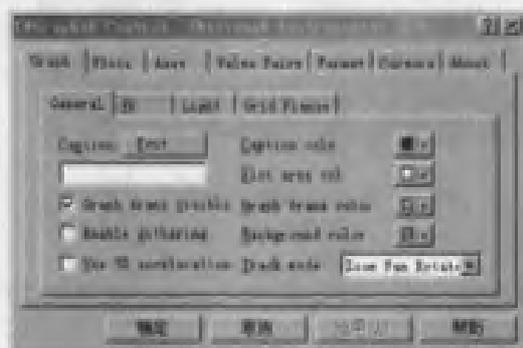


图 7.6.11 CWGraph3D 控件的属性设置对话框

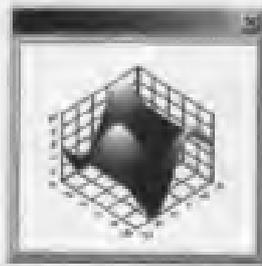


图 7.6.12 CWGraph3D 控件的预览面板

属性设置对话框共有 7 页, 分别是 Graph, Plots, Axes, Value Pairs, Format, Cursors, About。各项属性的含义非常明显, 设置方法也很类似, 下面只以 Graph 页为例介绍属性设置的方法。

Graph 页中包含 4 部分: General、3D、Light、Grid Planes, 如图 7.6.11 所示。

(1) 常规属性设置 (General)

Caption 用来设置 CWGraph3D 控件的标题, Font 用来设置标题的字体, Graph frame Visible 用来设置图像边框的可见性, Enable dithering 用来设置是否开启抖动, 开启抖动可以使颜色过渡更为平滑, Use 3D acceleration 用来设置是否使用 3D 加速, Caption color 用来设置标题颜色, Plot area color 用来设置作图区的背景色, Graph frame color 用来设置控件边框的颜色, Background color 用来设置标题的背景色, Track mode 用来设置跟踪的时间种类, 共有 4 种类型。

(2) 三维显示设置 (3D)

Projection 用来设置投影类型, 有正交投影 (Orthographic) 和透视 (Perspective) 两种, Fast Draw for Pan/Zoom/Rotate 用来设置是否开启快速画法, 此项开启时, 在进行移动、缩放、旋转时只将用数据点来代替曲面, 以提高作图速度, 默认值为 True。Clip Data to Axes Ranges 用来设置是否剪切数据, 当此项为 True 时只显示坐标轴范围内的数据, 默认值为 True。View Direction 用来设置视角, User Defined View Direction: 设定用户视角, 共有 3 个参数为: 纬度、精度和视点距离, 如图 7.6.13 所示。

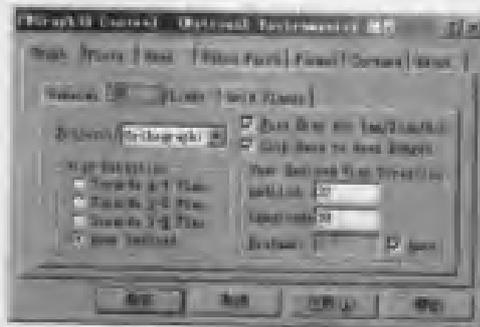


图 7.6.13 三维显示设置对话框

(3) 灯光设置 (Light)

除了默认的光照, CWGraph3D 控件还提供了 4 盏用户可以控制的灯。Enable Lighting 用来设置是否开启辅助灯照明, Ambient 用来设置环境光的颜色, Enable Light 用来设置具体设置每一盏灯的属性, 包括纬度 (Latitude)、精度 (Longitude)、距离 (Distance)、衰减 (Attenuation)、颜色, 如图 7.6.14 所示。



图 7.6.14 灯光设置对话框

(4) 网格平面设置 (Grid Planes)

Show Grid Plane 用来设定显示网格的平面。Smooth grid line 用来选中该项以平滑网格线。Grid frame color 用来设置网格边框的颜色, 如图 7.6.15 所示。

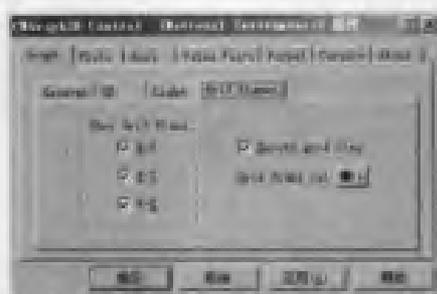


图 7.6.15 网格平面设置对话框

7.7 三维参数曲面图

本章第 7.6 节介绍了 3D Surface Graph 的使用方法。3D Surface Graph 可以显示三维空间的一个曲面, 但在显示三维空间的封闭图形时就无能为力了, 这种情况下可以使用三维参数曲面图, 即 3D Parametric Surface。3D Parametric Surface 及其图标如图 7.7.1 所示。与 3D Surface Graph 类似, 当在前面板放置一个 3D Parametric Surface 时, 框图中将出现两个图标, 一个是 ActiveX 控件 3D Parametric Surface 的图标, 另一个是 3D Parametric Surface.vi, 该 VI 负责三维作图。

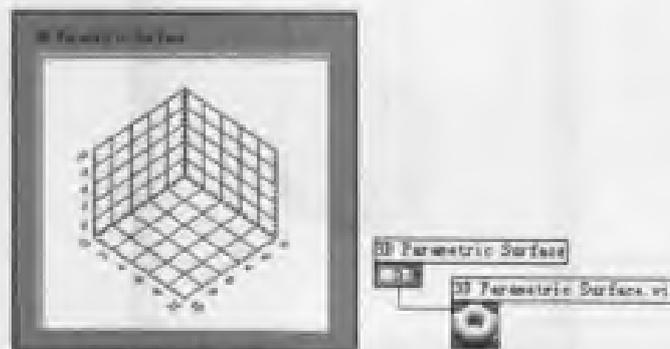


图 7.7.1 3D Parametric Surface 及其端口

7.7.1 使用三维参数曲面图

3D Parametric Surface 的使用较为复杂,但借助参数方程的形式则可以容易的理解。若要显示三维图形,需要3个方程:

$$x = f_x(i, j)$$

$$y = f_y(i, j)$$

$$z = f_z(i, j)$$

其中, x, y, z 是图形中点的三维坐标, i, j 是两个参数。

图 7.7.2 显示了 3D Parametric Surface.vi 的端口。各端口的含义为: 3D graph 表示 ActiveX 控件 3D Parametric Surface 输入端; x matrix 表示参数变化时 x 坐标所形成的二维数组; y matrix 表示参数变化时 y 坐标所形成的二维数组; z matrix 表示参数变化时 z 坐标所形成的二维数组。

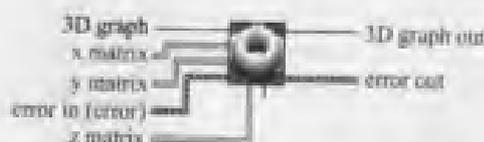


图 7.7.2 3D Parametric Surface.vi 的端口

例 7.7.1 绘制单位球面。

本例将使用 3D Parametric Surface 绘制一个单位球。球面的参数方程可以表示为

$$x = \cos \theta \cos \varphi$$

$$y = \cos \theta \sin \varphi$$

$$z = \sin \varphi$$

其中 θ 是球到球面任意一点的矢径与 Z 轴的夹角, φ 是该矢径在 XY 平面的投影与 X 轴的夹角。令 θ 从 0 变化到 π , 步长为 $\frac{\pi}{24}$, φ 从 0 变化到 2π , 步长为 $\frac{\pi}{12}$, 上述方程将确定一个球面。程序的前面板和框图如图 7.7.3、图 7.7.4 所示。

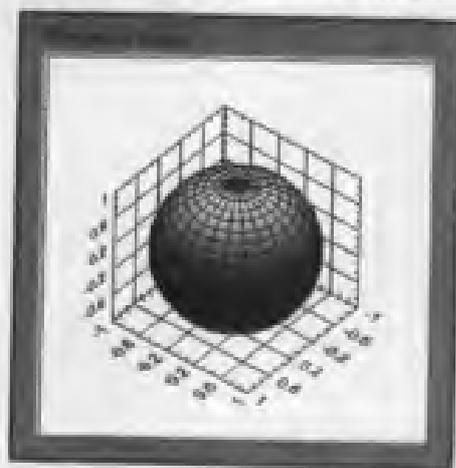


图 7.7.3 使用 3D Parametric Surface 绘制单位球面前面板

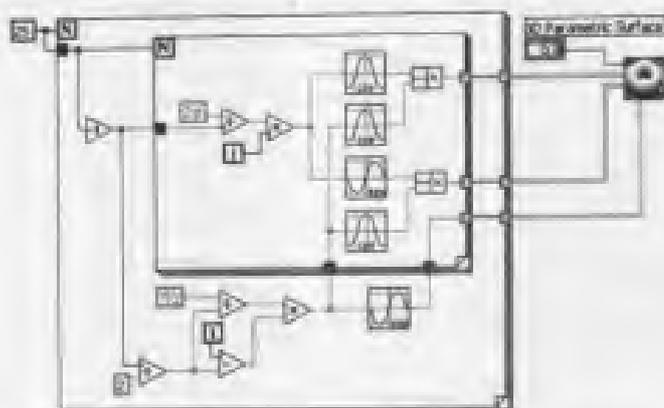


图 7.7.4 使用 3D Parametric Surface 绘制单位球面框图

7.7.2 设置三维参数曲面图的外观

3D Parametric Surface 的外观设置方法与 3D Surface Graph 类似, 请参考第 7.6.2 小节, 在此不再赘述。

7.8 三维曲线图

三维曲线图即 3D Curve Graph, 用于显示三维空间中的一条曲线。它的前面板和图标如图 7.8.1 所示。当在前面板放置一个 3D Curve Graph 时, 框图中将出现两个图标, 一个是 ActiveX 控件 3D Curve 的图标, 另一个是 3D Curve.vi 的图标, 该 VI 负责三维曲线作图。

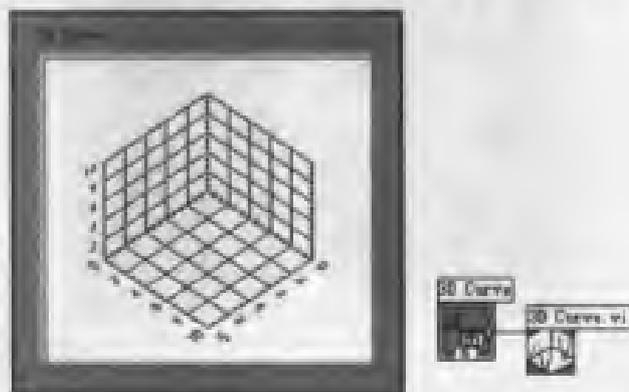


图 7.8.1 3D Curve Graph 及其端口

7.8.1 使用三维曲线图

3D Curve Graph 的使用较为简单, 只要给出曲线的点的坐标即可。图 7.8.2 显示了 3D Curve.vi 的图标。各输入端口的含义为: 3D graph 表示 3D Curve Graph 的输入端; x vector 表示 x 坐标, 以数组形式给出; y vector 表示 y 坐标; z vector 表示 z 坐标。点的坐标可以直接指定, 但通常使用公式计算出点的坐标, 然后在组成数组。



图 7.8.2 3D Curve.vi 的图标

例 7.8.1 绘制螺旋线。

本例将绘制一条螺旋线。螺旋线的坐标由下面的公式给出。

$$x = \cos \theta$$

$$y = \sin \theta$$

$$z = \theta$$

其中 $\theta \in [0, 2\pi]$ ，步长为 $\frac{\pi}{12}$ 。程序的前面板和框图程序如图 7.8.3 所示。为了看起来更加直观，在 3D Curve Graph 的右键弹出选单中选择 CWGraph3D → 特性(P)...，通过对话框将 3D Curve Graph 的背景色 (General → Plot area color) 设为黑色，将网格边框色 (General → Grid planes → Grid frame color) 设为浅绿色，将网格色 (Axes → Grid → Major Grid → Color) 设为深绿色。将 Color map style (Plot → Style → Color map style) 设为 None。

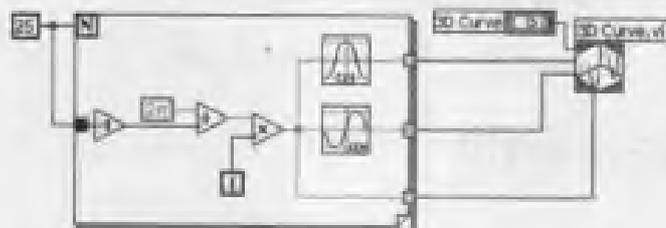


图 7.8.3 在三维控件绘制螺旋线

7.8.2 设置三维曲线图的外观

3D Curve Graph 的外观设置方法与 3D Surface Graph 类似, 在此不再赘述, 请参考 7.6.2 节。

7.9 极坐标图

极坐标图即 Polar Plot, 如图 7.9.1 所示。Polar Plot 实际上是一个 Picture 控件, 并绑定了 Polar Plot.vi, 可直接用来绘制极坐标图。



图 7.9.1 Polar Plot 及其图标

7.9.1 使用极坐标图

在使用 Polar Plot 时, 需要提供以“极径—极角”方式表示的数据点的坐标。Polar Plot.vi 的图标和端口如图 7.9.2 所示。data array[mag,phase]端口连接点列的坐标数组, dimension(w, h)端口设置极坐标图的尺寸。在默认设置下, 该尺寸等于 Picture 控件的尺寸, 即通过 Picture 控件的属性结点获取 DrawAreaSize 属性, 然后将该属性值输入 dimension(w,h)端口。polar attributes 端口设置 Polar Plot 的图形颜色、网格颜色、显示象限等属性。

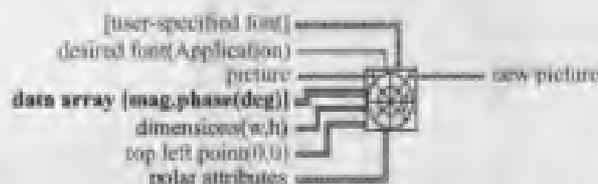


图 7.9.2 Plot.vi 的图标和端口

例 7.9.1 Polar Plot 的使用。

本例将绘制一条三叶的玫瑰线, 使用的方程为:

$$\rho = \sin 3\alpha$$

极角从 0° 变化到 360° 。前面板和框图程序如图 7.9.3 所示。在本例中, 将 Polar Plot 的背景色设为了黑色, 网格线的颜色 (grid color) 设为了绿色, 线条色 (plot color) 设为了黄色。

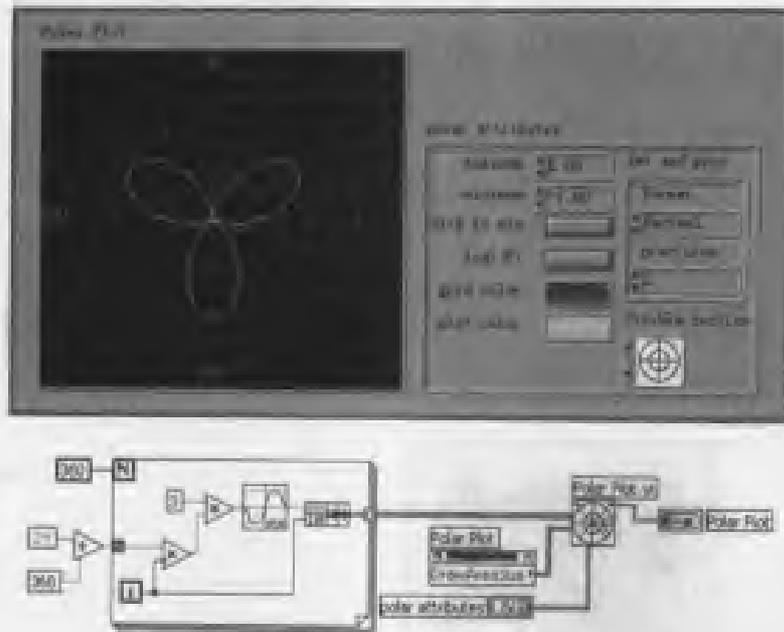


图 7.9.3 使用极坐标图绘制三叶玫瑰线

7.9.2 设置极坐标图的外观

因为 Polar Plot 实际上是一个 Picture 控件，因此 Polar Plot 的外观通过 Polar Plot.vi 设置。polar attributes 端口提供了一些基本显示属性，这些属性如图 7.9.4 左边所示。各项属性设置的含义为：maximum 表示极径的最大值；minimum 表示极径的最小值，允许为负，该属性与 maximum 属性共同决定了显示区的大小；clip to min 表示是否按极径小值剪裁，如果为真则不显示极径小于 minimum 的图形部分；log?(F) 表示是否以对数方式显示坐标；grid color 表示网格的颜色；plot color 表示图形颜色；format 表示数据格式；precision 表示精度；visible section 表示显示的象限，共有 9 中形式，如图 7.9.4 右侧所示。

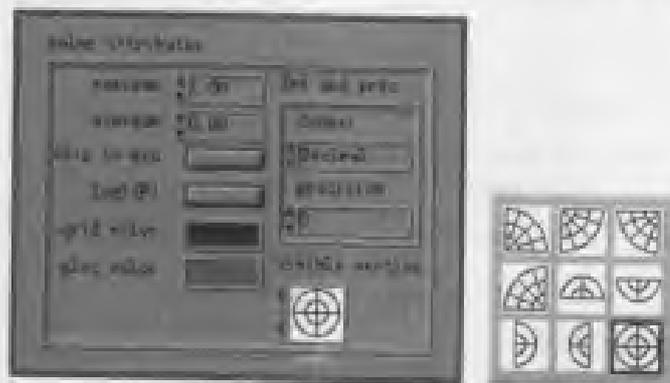


图 7.9.4 Polar Plot 外观设置

第 8 章 文件操作与管理

文件操作与管理是测试系统软件开发的重要组成部分，数据存储、参数输入、系统管理都离不开文件的建立、操作和维护。LabVIEW 为文件的操作与管理提供了一组高效的 VI 集。本章首先概要介绍了 LabVIEW 文件操作与管理的基本概念与术语，包括文件路径与标识、文件 I/O 操作流程与出错处理等，在此基础上通过实例对 LabVIEW 有关文件 I/O VIs 的功能和用法进行了论述，最后简要介绍了 LabVIEW 系统中数据文件类型和相关的数据存储与读取问题。值得指出的是，文件 I/O 的一些底层操作和高级功能相对来说比较难于理解，因此，本章也是学习 LabVIEW 的一个难点。

8.1 基本概念及术语

在介绍 LabVIEW 文件 I/O 操作之前，首先来熟悉一下 LabVIEW 文件操作中的一些基本概念和术语。

8.1.1 路径

任何一个文件的操作（如文件的打开、创建、读写、删除、拷贝等），都需要确定文件在磁盘中的位置。LabVIEW 与 C 语言一样，也是通过文件路径（Path）来定位文件的。不同的操作系统对路径格式有不同的规定，但大多数的操作系统都支持所谓的树状目录结构，即有一个根目录（Root），在根目录下，可以存在文件和子目录（Sub Directory），子目录下又可以包含有各级子目录及文件。

在 Windows 系统下，一个有效的路径格式如下：

```
driver:\<dir...>\<file or dir>
```

其中，<driver:>是文件所在的逻辑驱动器盘符，<dir...>是文件或目录所在的各级子目录，<file or dir>是所要操作的文件或目录名。LabVIEW 的路径输入必须满足这种格式要求。

在由 Windows 操作系统构造的网络环境下，LabVIEW 的文件操作节点支持 UNC 文件定位方式，可直接用 UNC 路径来对网络中的共享文件进行定位。可在路径控制中直接输入一个网络路径，在路径指示中返回一个网络路径，或者直接在文件对话框中选择一个共享的网络文件（文件对话框参见本小节后续内容）。只要权限允许，对用户来说网络共享文件的操作与本地文件操作并无区别。

一个有效的 UNC 文件名格式为：

```
\\<machine>\<share name>\<dir>\...<file or dir>
```

其中，<machine> 是网络中的机器名，<share name>是该机器中的共享驱动器名，

<dir>... 为文件所在的目录, <file>即为所选择的文件。

LabVIEW 用路径控制 (Path Control) 输入一个路径, 用路径指示 (Path Indicator) 显示一个路径。路径及其端口如图 8.1.1 所示。



图 8.1.1 路径输入指示的面板及端口

路径名的输入操作与字符串的输入完全相同, 路径名实际就是一种符合一定格式的字符串。路径值可以是一个有效的路径名, 一个空值或“非路径” <Not A Path>。单击路径控件上的标志 , 可以从其下拉式选单中选择 <Not A Path> 值, 此时, 控件上的“路径”标志  将变成“非路径”标志 , 并且 <Not A Path> 将出现在 Path Control 的显示区。如图 8.1.2 所示。



图 8.1.2 设置路径控件属性

在一些文件 I/O 节点中, 如果节点要求有一个路径输入, 而这个路径的值如果是空 (Empty) 或非路径, 则在运行时, 它将通过一个标准的 Windows 文件对话框来选择所要操作的文件。

一个文件节点如果有一个路径输出, 且这个输出通过 Path Indicator 显示, 如果该节点操作失败, 则 Path Indicator 将显示“非路径”值, 且其“路径”标志  将变成“非路径” (Not A Path) 标志 .

在后续的介绍中, 会用到 3 个易混淆的概念, 在这里需明确一下。文件名仅指文件的名称。在 Windows 系统下, LabVIEW 支持汉字及长文件名。目录仅指文件存在的目录位置, 不包括文件名。路径包含文件所在的目录及文件名。

8.1.2 标识号

标识号 (RefNum) 的概念在 LabVIEW 的各种 I/O 操作中非常重要, 使用也非常广泛。正确理解标识号的概念是掌握文件操作及其他对象 I/O 操作的基础。LabVIEW 在对一个对象进行 I/O 操作前, 通常先要打开这个对象的一个标识号, 这个标识号实际上包含了这个

特定对象的很多信息。对一个文件来说，它包含了这个文件的位置、大小、读写权限等所有在文件操作中所必须的信息；而对一个 VXI 或其他接口的仪器的标识号来说，它包含了仪器的接口形式、地址及其他一些信息；对一个网络连接来说，它包含了连接的地址、协议等。在建立了一个 I/O 对象的标识号后，对这个 I/O 的后续操作都是依据这个标识号来进行。一次 I/O 完成后，应该释放这个标识号，以便释放它所占用的资源。LabVIEW 中使用的标识号如图 8.1.3 所示。



图 8.1.3 LabVIEW 中的标识号

在 Controls 模板 → All Controls 子模板 → Refnum 子模板中，可以找到所有的 Refnum Control。每一个 Refnum Control 都有一个相应的 Refnum Indicator，在右键弹出菜单中选择 Change to Indicator 或 Change to Control 可以在 Refnum Control 与 Refnum Indicator 之间相互切换。

本节介绍文件标识号 (File Refnum) 的用法，其他标识号的用法会在相关内容中进行介绍。文件标识号根据文件类型的不同，分成流文件标识号 (Byte Stream File Refnum) 和块记录文件标识号 (Data Log File Refnum)。当打开一个文件时，LabVIEW 将给这个打开的文件赋予一个与其相联系的标识号，其后，所有针对此文件的操作都可通过这个标识号来进行。当文件被关闭时，此标识号被释放。标识号的分配是随机的，同一文件每次被打开时，它被赋予的标识号是不同的。

例 8.1.1 利用文件的标识号进行文件的读写操作。

前面板及框图程序如图 8.1.4 所示。

本例进行了 3 步最基本的文件操作：打开一个文件，读文件，关闭文件（所使用的文件操作节点的具体用法见 8.1.2 节）。首先，在控制面板上建立一个流文件标识号控件（假设这个文件是一个流文件）。在框图程序中，用 Open/Create/Replace File.vi 打开一个文件，然后把节点操作产生的文件标识号赋给控制面板中创建的标识号控件。把标识号控件的一个局部变量连接到读文件 (Read File) 及关闭文件 (Close File) 节点的 RefNum 输入端口端，这两个节点就能通过这个标识号来读并关闭该文件。这个例子只是为了说明文件标识在文件 I/O 中的作用，在本例中，完全可以把 Open/Create/Replace File.vi 的 RefNum 输出直接连到读文件及关闭文件节点的 RefNum 的输入端。在实际应用中，如果需要不同的 VI

中同时对同一文件进行 I/O 操作, 则可以通过创建文件 RefNum 的全局变量来实现。



图 8.1.4 例 8.1.1 的前面板及框图程序

8.1.3 文件 I/O 的出错管理

在 LabVIEW 中, 大多数的文件 I/O 节点有一个 error in 和 error out 连线端口, 它们的数据类型为一个簇。用这两个连线端口可以把几个文件 I/O 节点连接在一起, 如果其中有一个节点因操作出错, 则该节点的 error out 将返回一个错误信息, 并把错误信息传递到下一个节点的 error in 端口, 此时, 接收到这个错误信息的节点将不再执行文件 I/O 操作, 而是直接把这个错误信息通过它的 error out 端口再传递到下一个节点直至最后一个节点, 这样就可以在最后一个节点的 error out 端口连接错误处理 VI。通过这种方法, 除了可以避免文件操作错误被扩散外, 还可以避免无用的操作 (如打开一个文件失败后, 用这种方法可以避免试图对这个文件进行读写操作)。error out 簇中包含了具体的错误代码。

8.1.4 文件 I/O 操作流程控制

在本节的后续内容中将介绍, 很多文件 I/O 节点都有所谓的直接传递 (flow-through) 参数, 这个参数只是简单地把一个输入端口的值原样输出, 并不进行实际的操作。可以把一个节点的 flow-through 参数输出连到下一个节点的 flow-through 参数输入端, 此时, 只有当前一个节点执行完后, 第二个节点才能执行。通过这种方法, 可以控制程序的执行顺序, 否则, 必须多次使用顺序结构。error in 与 error out 就是一对 flow-through 参数, 它可以用来控制程序的执行顺序。

8.2 文件操作

LabVIEW 提供了一组文件操作节点, 利用这些节点, 可以执行创建新文件, 读、写文件, 删除、移动及拷贝文件, 查看文件及目录列表等一系列操作。

一个基本的文件操作主要包含 3 个步骤:

第一步, 在对文件进行读写之前必须要创建或打开这个文件;

第二步，对文件进行读写操作；

第三步，在读写操作完成后，必须关闭该文件，否则将造成不可预知的结果，如可能引起文件的数据丢失。

在 LabVIEW 提供的文件操作节点里，有些节点一次只能完成单一的一个操作，如仅仅打开一个文件，这种只具有单一功能的节点称为低级（Low level）文件操作节点，有些节点调用一次可以完成多步甚至是一次完整的文件 I/O 操作，这种节点称为高级（High level）文件操作节点。文件的高级操作节点是通过调用一个或几个低级节点来实现的，使用较为简单，但不具备低级节点操作的灵活性。

LabVIEW 的文件操作节点位于 Functions 模板→All Functions 子模板→File I/O 子模板中，如图 8.2.1 所示。下面对主要节点的用法进行介绍。



图 8.2.1 File I/O 子模板

8.2.1 文件定位与文件对话框

1. 文件定位节点（Advance File Function 子模板→Seek）

在 C 语言中，有一个文件指针的概念（Pointer），用来确定在文件的何处进行操作。在 LabVIEW 中，相应有一个 Offset 的概念，它的含义及用法与 C 语言的文件指针完全相同，所以也称 Offset 为文件指针。文件的读写总是从文件指针所在位置开始的。

文件定位节点，即 Seek 节点，位于 Advance File Function 子模板中，Seek 节点可以用来移动文件指针，Seek 节点的图标及其端口定义如图 8.2.2 所示。

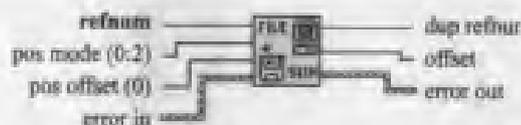


图 8.2.2 Seek 节点的图标及其端口定义

指针移动位置由 pos mode (0:2) 与 pos offset (0) 端口的值共同决定。端口 pos mode (0:2) 设置起始位置，端口 pos offset (0) 设置指针的偏移量。

- 当 pos mode (0:2) = 0 时，则新指针位置为文件起始位置与 pos offset (0) 的值

相加的结果:

- 当 $\text{pos mode}(0:2) = 1$ 时, 则新指针位置为文件尾位置 (即文件长度) 与 $\text{pos offset}(0)$ 的值相加的结果;
- 当 $\text{pos mode} = 2$ 时, 则新指针位置为文件当前位置与 $\text{pos offset}(0)$ 的值相加的结果, 这也是这个参数的默认值。

无论是流文件还是块记录文件, $\text{pos offset}(0)$ 的单位都是 Byte, 所以为了保证数据读取的正确性, 要根据数据单元的长度来确定这个值的大小, 使其位于文件中某个数据单元的开始位置。值得注意的是, $\text{pos offset}(0)$ 可以为正值, 也可以为负值, 所以指针可以在文件中前后移动, 但有效的文件指针却不能小于 0, 也不能大于文件长度, 否则将导致非法操作。

如果 $\text{pos offset}(0)$ 端口有数据连线, 则 $\text{pos mode}(0:2)$ 的默认值为 0, 否则 $\text{pos mode}(0:2)$ 的默认值为 2。

如果这两个输入端口计算出来的文件指针数是非法的, 则文件指针保持在当前的位置, 并返回一个错误代码。

2. 文件截断操作节点 (Advance File Function 子模板 \rightarrow EOF)

文件截断操作节点, 即 EOF 节点, 位于 Advance File Functions 子模板中, EOF 节点的图标及其端口定义如图 8.2.3 所示。



图 8.2.3 EOF 节点的图标及其端口定义

在标识号代表的流文件的指定位置设置一个文件结束符 EOF, 截断位置由端口 $\text{pos mode}(0:1)$ 与端口 $\text{pos offset}(0)$ 的值决定, 此位置就是文件指针的位置 (参见 Seek 节点)。当操作完成后,

文件指针位于文件尾, offset 返回其值, 这个值也就是文件当前的长度。但不能用这个节点来截断一个块记录文件。

3. 文件对话框节点 (Advance File Function 子模板 \rightarrow File Dialog)

在后述的文件操作中, 常要借助于一个 Windows 标准文件对话框来输入、选择所要操作的文件或目录, 标准文件对话框如图 8.2.4 所示。



图 8.2.4 Windows 标准文件对话框

可以通过调用 LabVIEW 的文件对话框，即 File Dialog 节点，来打开这个对话框，并利用这个节点的输入端口来设定这个对话框的各种显示模式，利用其输出口来返回所选的文件或目录。File Dialog 节点位于 Advance File Function 子模板中，节点的图标及其端口定义如图 8.2.5 所示。

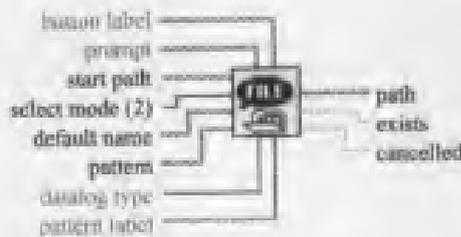


图 8.2.5 File Dialog 节点的图标及其端口定义

File Dialog 节点的输入端口如下。

- **Prompt** 端口：字符串类型，用来定义对话框的标题栏，通常为在操作中作为给用户的提示信息。在如图 8.2.4 所示的对话框中，其值为“对话框示例”。
- **start path** 端口：路径类型，定义文件对话框的初始显示路径。如果这个输入的值为 <Not A Path>，则其初始位置为对话框最近一次使用的路径。
- **default name** 端口：字符串类型，定义对话框中的初始文件名。在图 8.2.4 的对话框中，其值为 Power.vi。这个值在让用户选择一个特定的文件时非常有用。
- **Pattern**：该输入通过使用通配符来过滤对话框中文件显示的类型，只有文件名与这个参数匹配的文件才会显示在对话框中。文件名匹配的方式与 Windows 系统中使用的通配符方式是一致的，即用“?”来通配单个字符，用“*”号来通配一系列字符。如果有多个匹配条件，则每个条件之间用分号“;”来隔开。如匹配条件为“*.html;*.doc”，则文件对话框将显示所有以 html 和 doc 为后缀的文件。在如图 8.2.4 所示的对话框中，pattern 参数的输入为*.vi。
- **select mode** 端口：该参数用来限制用户选择文件或目录的类型。借助于这个参数，可以有效地防止用户的误操作。其端口参数含义见表 8.2.1。
- **catalog type** 端口：可以是任何数据类型。当这个输入有数据连线时，则对话框只能显示具有这种数据类型的记录文件。

表 8.2.1 select mode 端口参数表

参 数 值	含 义
0	选择一个已经存在的文件。在这个参数值下，只能选择一个已经存在的文件，如果输入了一个不存在的文件名，对话框将弹出一个警告窗口，要求重新选择，直到选择了正确的文件路径或按“取消”按钮取消本次文件选择操作为止
1	定义一个新文件。在这个参数值下，只要在“文件名(N)”输入框中输入一个适当的文件路径，后续的程序操作就可以用这个节点的返回结果及创建文件节点来创建一个新的文件。对话框本身并不创建文件，它只用来定义新文件的路径及文件名
2	选择一个已经存在的文件或定义一个新文件。这个值是该参数的默认值
3	当使用该参数值时，用户只能选择一个文件目录而不能选择一个具体的文件。如果在程序中使用一个目录来存取一次测量中的多个数据文件或要定义一个默认文件操作目录时，这个参数值是最好的选择
4	这个参数值用来定义一个新的目录，可以用创建目录节点来创建这个新的目录

续表

参 数 值	含 义
5	选择一个存在的目录或定义一个新的目录
6	选择一个 LabVIEW 库 (LLB) 中存在的文件
7	在一个 LabVIEW 库 (LLB) 中定义一个新的文件
8	在一个 LabVIEW 库 (LLB) 中选择一个存在的文件或定义一个新的文件

注: *LLB 是 LabVIEW 的库文件, 它是若干 LabVIEW 文件的集合, 用其他的应用程序并不能对库内的文件进行列表, 当该参数值为 6, 7, 8 时, 可以对 LLB 中的文件进行操作。

File Dialog 节点的输出端口如下。

- path 端口: 返回已选择或已输入的文件路径或目录。如果因为某种原因使对话框不能返回正确的路径, 则其值为 <Not A Path>。

- exists 端口: 如果其返回值为 True, 则说明 path 中返回的路径或目录是已经存在的, 否则说明 Path 中的路径或目录是新文件或目录。后续程序操作可以依此来判断下一步的操作是打开文件还是创建文件。



图 8.2.6 警告对话框

- Cancelled 端口: 如果在对话框中选择了“取消”, 其值为 True。

- 在文件对话框使用过程中, 如果用户的操作有误, 会有警告对话框弹出, 如图 8.2.6 所示。

8.2.2 文件操作

本节将介绍如何打开和关闭文件, 以及文件的读写操作。

1. 打开、关闭文件



图 8.2.7 Open File 节点的图标及其端口定义

(1) 打开文件 (Advance File Function → Open File)

Open File 节点用于打开一个文件。节点图标及其端口定义如图 8.2.7 所示。

节点的端口说明如下。

- datalog type 端口: 当该端口有数据连线时, 表明打开的是一个块记录文件, 反之默认为比特流文件 (有关 LabVIEW 数据文件类型的内容, 请参考第 8.4.1 小节)。

- open mode (0) 端口: 定义了打开文件的方式。当 open mode=0 时, 可以对打开的文件进行读写操作; 当 open mode=1 时, 只能对打开的文件进行读操作而不能进行修改。如果文件不存在, 则返回一个错误代码。

- deny mode (2) 端口: 用于设定其他用户同时操作文件的权限。如果为 0, 则禁止其他用户与当前用户同时读或写文件; 如果为 1, 则禁止其他用户在同一时间对此文件

执行写操作；如果为 2，则允许其他用户与当前用户同时读写文件。

Open File 节点是打开文件操作最基本的一个节点，既可用于打开流文件也可以打开块记录文件。一个文件打开后，后续的程序可以用这个节点返回的标识号对这个文件进行各种操作。

(2) 创建一个新文件 (Advance File Function → New File.vi)

New File.vi 节点用于创建一个新文件，并使之处于打开状态以备读写。节点图标及其端口定义如图 8.2.8 所示。datalog type 可以连接任何数据类型。但是，当这个输入端口有数据连线时，说明创建的文件是块记录文件。deny mode (2) 端口的设置方法同 Open File 节点。如果输入的文件已经存在，且参数 overwrite 为 True 时，则覆盖该输入文件；如果 overwrite 为 False，则返回一个错误代码。

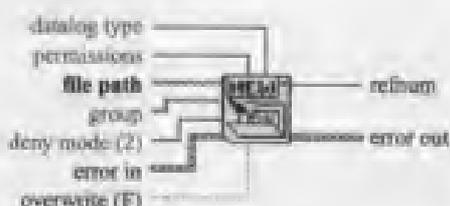


图 8.2.8 New File.vi 的图标及其端口定义

(3) 打开/创建/覆盖文件 (Open/Create/Replace File.vi)

Open/Create/Replace File.vi 节点用于打开、覆盖一个已经存在的文件或创建一个新文件。节点图标及其端口定义如图 8.2.9 所示。文件路径的输入既可以直接在路径控制里输入，也可用文件对话框来输入。这个节点其实是通过调用 File Dialog、Open File 和 New File 三个节点来实现其功能的。

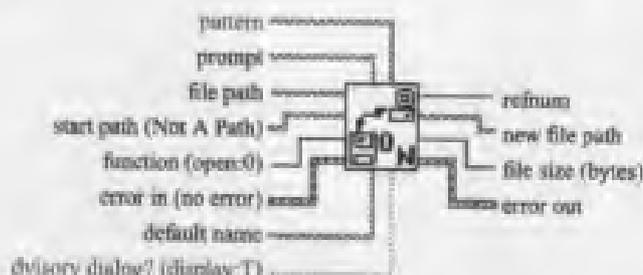


图 8.2.9 Open/Create/Replace File.vi 的图标及其端口定义

节点的具体功能由 function 端口的值来确定，其值所代表的意义见表 8.2.2。

表 8.2.2 function 端口参数表

参 数 值	含 义
0	打开一个已经存在的文件，如果文件不存在，则返回一个错误代码
1	如果一个文件存在，则打开这个文件；如果不存在，则创建一个新文件
2	创建一个新文件或覆盖掉一个已经存在文件
3	创建一个新文件，如果同名文件已经存在，则返回错误代码

advisory dialog 为 true 时, 每当有错误操作或在覆盖文件前, 将会出现警示窗口要求确定。new file path 表示返回打开或创建的新文件的路径。file size (Bytes) 表示返回已打开文件的大小, 单位为字节 (Byte)。

其他端口使用方法参阅 File Dialog、Open File 和 New File 节点的相应说明。

(4) 关闭文件 (Close File)

Close File 节点用于关闭一个指定的文件。关闭标识号 RefNum 所代表的文件。节点图标及其端口定义如图 8.2.10 所示。关闭一个文件要进行下列操作:

第一步, 把文件写在缓冲区里的数据写入物理存储介质中;

第二步, 更新文件列表的信息, 如大小、最后更新日期等;

第三步, 释放标识号 RefNum。

值得注意的是, 这个节点不管 error in 是否有错误信息输入, 都要执行关闭文件的操作。所以, 必须从 error out 中判断关闭文件操作是否成功。

(5) 清空文件缓冲区 (Advance File Functions → Flush File)

与 C 语言的文件操作一样, 当向文件写数据时, 数据是先存放在一个缓冲区里而不是直接写入物理存储介质的, 只有当缓冲区满或文件关闭时才执行真正的物理写操作, 这样可以减少对磁盘的操作频率并提高文件读写速度。Flush File 节点强迫缓冲区的数据写入到物理存储器中, 但它并不关闭文件。节点图标及其端口定义如图 8.2.11 所示。



图 8.2.10 Close File.vi 的图标及其端口定义



图 8.2.11 Flush File 节点的图标及其端口定义

2. 读文件

(1) 读文件 (Read File.vi)

Read File.vi 节点的图标及其端口定义如图 8.2.12 所示。

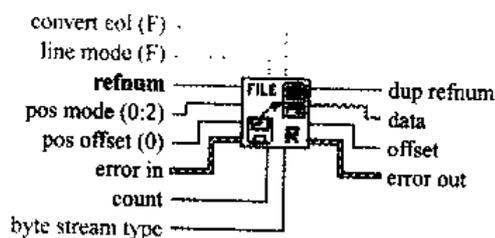


图 8.2.12 Read File.vi 的图标及其端口定义

如果 refnum 端口输入的文件标识号代表的是一个打开的比特流文件, 则该节点从一个比特流文件中读取数据。此时, byte stream type 定义其数据结构。该数据结构应该严格与写数据时的数据结构一致 (即使写入的是 SGL 而读数据时设置为 DBL, 也将引起读数据错误)。如果数据类型长度是可变的 (如字符串、数值与字符串构成的簇等), 则该节点总是假定数据文件中的每一个完整的数据单元都具有相同的结构 (长度可以不相同)。要读取的数据量由 count 输入, 其单位不是 Byte 而是数据单元的个数。每读一个数据单元, 文件

指针向后移动该数据单元的长度。如果该节点在读取数据时,发现文件中的数据类型与 `byte stream type` 不匹配,则由 `data` 返回默认值,并通过 `error out` 返回一个错误代码。

读取的数据量由 `line mode` 端口与 `count` 端口共同决定:

- 如果 `line mode` 为 `True`, `count` 没有数据连线或其值为 0, 则节点读数据直到遇到一个行结束符或者文件结束符;
- 如果 `line mode` 为 `True`, `count` 的值大于 0, 则节点读取数据直到已读取数据单元的个数等于 `count` 的值或直到遇到行结束符或文件结束符;
- 如果 `line mode` 为 `False`, `count` 的值大于 0, 则节点读取数据直到已读取的数据单元的个数等于 `count` 的值或遇到行结束符。

如果 `RefNum` 代表的是一个已经打开的块记录文件,则该节点的 `line mode` 和 `data stream type` 均无效,而 `count` 输入的是要求读取的数据记录的个数。如果此时返回数据 `data` 连接的数据结构与文件中的记录结构不一致,将导致读数据失败,并通过 `error out` 返回一个错误代码。

值得注意的是,该节点的 `offset` 不论在流文件中还是在记录文件中总是有效的,而且单位总是 `Byte`。如果想设定读取数据时的开始位置,则必须把它放在一个数据单元的开始位置。如果文件是一个流文件而且数据单元的长度是可变的,使用这个参数就很危险,很可能导致数据的读取错误;而在记录文件中,则必须保证 `offset` 是一个记录的开始位置。

(2) 读电子表格文件 (Read From Spreadsheet File.vi)

`Read From Spreadsheet File.vi` 节点用于从一个电子表格文件中读取一定数量的数据,并把这些数值字符串转换成单精度浮点数从一个二维数组或一维数组中返回。节点图标和连接端口如图 8.2.13 所示。值得注意的是,必须保证这个电子表格文件的所有字符串全部由有效的数值字符组成。此节点在读取数据前打开这个文件,操作完成后关闭该文件。

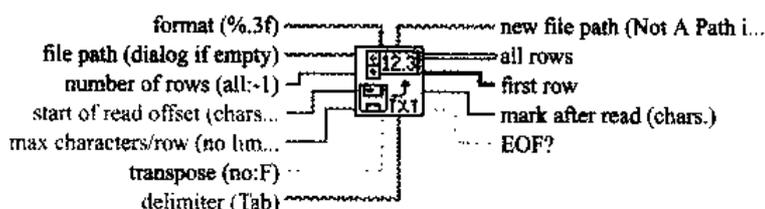


图 8.2.13 `Read From Spreadsheet File.vi` 的图标及其端口定义

下面介绍它的输入输出端口。

输入端口如下。

- `format (%.3f)` 端口: 确定字符串到浮点数的转换格式。
- `file path (dialog if empty)` 端口: 输入文件路径及文件名。如果其值为空或为 `<Not A Path>`, 则将通过文件对话框来选择文件,如果在对话框中选择“取消”键,则由 `error` 返回一个错误代码,错误代码为 43。
- `number of rows (all:-1)` 端口: 确定该节点读取的最多行数。如果该值小于 0, 则该节点将读取整个文件的数据。注: 电子表格的一行,是以一个回车符 (carriage return)、一个换行符 (line feed) 或一个文件结束符 (EOF) 来结束的。
- `start of read Offset (chars...)` 端口: 确定读取数据的初始位置。单位是 `Byte`。

输出端口如下。

- new file path (Not A Path if...) 端口：返回文件的路径。如果操作有错误，则返回 <Not A Path>。
- all rows 端口：读取的所有浮点数都通过这个二维数组返回。
- first row 端口：返回所读取数据的每一行。如果想把数据的一行读入一个一维的数组中则，要使用这个输出端。
- mark after read (chars.) 端口：返回当前文件指针的位置。

(3) 字符读取节点 (Read Characters From File.vi)

从一个文本文件中读取字符，节点图标及其端口定义如图 8.2.14 所示。读取字符的数量由 number of characters 端口输入，如果输入-1，则读取整个文件。输出端口 character string 返回所读取的字符。操作完成后，节点将关闭该文件。

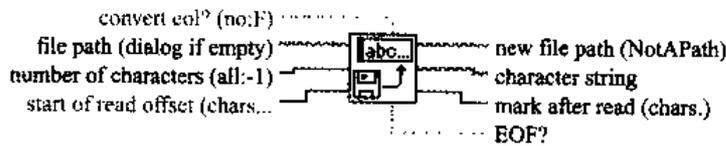


图 8.2.14 Read Characters From File.vi 的图标及其端口定义

其他输入输出端口说明见节点 Read From Spreadsheet File。

(4) 字符串读取节点 (Read lines from file.vi)

Read lines from file.vi 节点用于从一个比特流文件中读取一定行数的字符串。读取操作完成后，节点关闭文件。节点图标及其端口定义如图 8.2.15 所示。

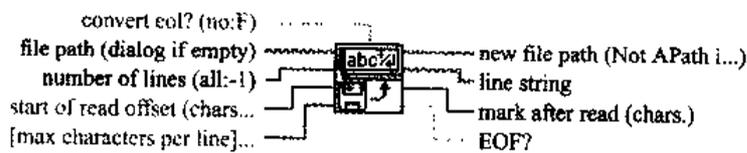


图 8.2.15 Read lines from file.vi 的图标及其端口定义

在输入端口中， number of lines 端口确定读取行数的最大数量，如果输入小于 0，则读取整个文件的数据。输出端口 line string 返回所读取的字符串。其他的输入输出端口参见 Read From Spreadsheet File 节点。

(5) 读整数节点 (Read From I16 File.vi)

Read From I16 File.vi 节点用于从一个比特流文件中读取一批 16 位的整数数据 (I16)，并把它们通过一个二维的或一维的数据数组返回。在读取数据前打开文件，读取完毕后，关闭文件。节点图标及其端口定义如图 8.2.16 所示。

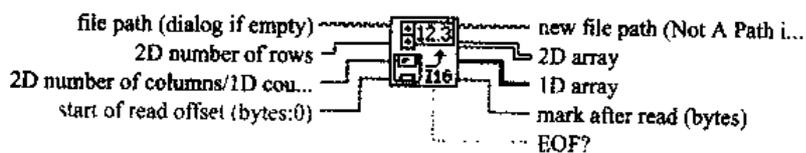


图 8.2.16 Read From I16 File.vi 的图标及其端口定义

输入端口如下。

- 2D number of rows 端口：决定输出的二维数组的行数。
- 2D number of columns/1D count 端口：决定输出二维数组的列数或输出一维数组的长度。与 2D number of rows 一起决定读取数据的数量及数据返回的形式。具体设置如下：把文件的数据读入一个一维的数组中，设置 2D number of columns/1D count<0（默认值），2D number of rows 不用连线。读 N 个数据到一维数组中，设置 2D number of columns/1D count=N，2D number of rows=0。把整个文件的数据读入到一个二维数组中，2D number of columns/1D count=N>0，N 是这个二维数组的列数，其行数 2D number of rows=Size/N，Size 为文件的长度，单位为 16 位的字。

读取 N*M 个数据到一个二维数组中去，2D number of columns/1D count=N，2D number of rows = M。

输出端口如下：1D array，2D array 端口，返回所读取的数据。最后由哪个数组返回数据要由上述两个输入端口的组合来决定。其他参数的意义参见 Read From Spreadsheet File 节点。

(6) 读浮点数节点 (Read From SGL File.vi)

从一个比特流文件中读取指定数量的单精度浮点数 (SGL)，并由一个二维的或一维的数据数组返回。在读取数据前打开文件，读取完毕后，关闭文件。节点图标及其端口定义如图 8.2.17 所示，输入输出端口见 Read From I16 File 节点。

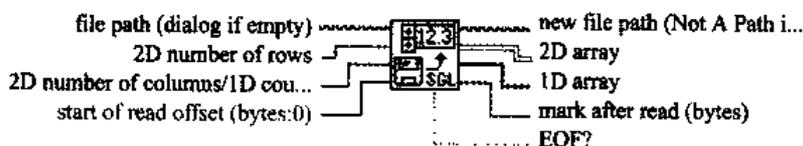


图 8.2.17 Read From SGL File.vi 的图标及其端口定义

3. 写文件

(1) 写文件 (Write File)

节点图标及其端口定义如图 8.2.18 所示。

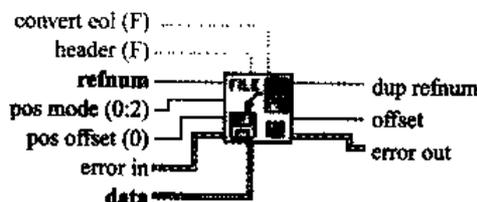


图 8.2.18 Write File 节点的图标及其端口定义

当 refnum 端口输入的文件标识号代表的是一个比特流文件时，该节点写数据到文件中的位置由参数 pos mode (0, 2) 和 pos offset (0) 决定（这两个参数的含义见 8.2.1 节“文件定位节点”）。

LabVIEW 语言把字符串一行的结束符定义为一个换行符 (LF, \0A)，而 Windows 操

作系统把字符串一行的结束符定义为一个回车符 (CR, \OD), 再接一个换行符。所以, 如果不进行转换, 则 LabVIEW 创建的文本文件由 Windows 的文本编辑器读出, 读出内容会不一样, 反之亦然。Convert eol 确定是否把 LabVIEW 输入字符串中的换行符转换成操作系统换行符。这个参数只有在输入数据为字符串时才有效。

Data 端口输入要写入文件的数据, 可以是任何的数据类型。当数据是可变长度类型 (如数组、字符串), 输入端口 header (F) 为 True 时, 该节点写数据时会在数据前加上一个数据头部 (header), 这个头部用来说明本次写入数据的长度。这样, 读数据时如果使用相同的数据类型, 可以使得读取数据的长度与写入时的一致, 但是, 对于字符串来说, 因为每行都有一个结束符可以用来区别一组的数据, 所以 header (F) 此时一般设置成 False。

如果 Refnum 代表的是一个块记录文件, 则该节点把数据当成一个记录块写入该文件。写操作总是在文件尾开始, 即记录文件只能追加在文件尾。此时, convert eol, header, pos mode 及 pos offset 等输入端口都是无效的。

本节点是写文件最基本的节点, 既可以写流文件, 也可以写记录文件。后续介绍的写文件节点, 都是通过调用这个节点来实现的。

(2) 写电子表格节点 (Write to Spreadsheet File.vi)

Write to Spreadsheet File.vi 节点把一个二维的或一维单精度浮点数据数组写到一个电子表格文件中。如果文件是已经存在的, 则既可以把数据追加到该文件, 也可以覆盖原有的数据; 如果文件不存在, 则创建新文件。节点图标及其端口定义如图 8.2.19 所示该节点要调用 Array to Spreadsheet String 节点把数据数组中转换成字符串。该节点在写数据之前把文件打开, 数据写完之后将自动关闭该文件。

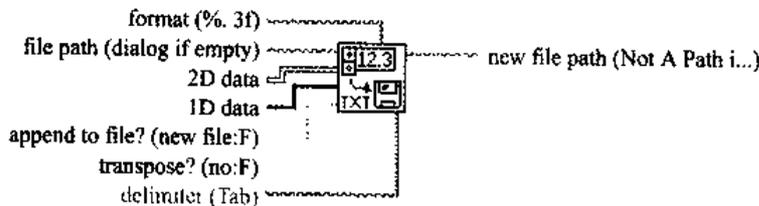


图 8.2.19 Write To Spreadsheet File.vi 节点的图标及其端口定义

输入端口如下。

- format (%.3f) 端口: 决定单精度浮点数到字符串的转换格式。
- file path (dialog if empty) 端口: 文件路径输入。可以直接在 file path 中输入一个文件路径和文件名, 如果文件是已经存在的, 则打开这个文件, 如果输入的文件不存在, 则创建这个新文件。如果 file path 的值为空或是非法的路径, 则调用 file dialog, 通过文件对话框来选择或输入文件。
- 2D data、1D data 端口: 输入二维或一维的写入数据。
- append to file? (new file:F) 端口: 布尔输入。如果设为 True, 且写数据前该文件已经存在, 则新的数据被追加到文件尾。当然, 对于一个新创建的文件来说, 该输入为 True 或 False 含义是一样的; 如果该输入的值 of False (默认值), 则对于已经存在的文件, 新的数据将覆盖掉原有的数据, 在使用时必须要注意。

输出端口如下。

- new file path (Not A Path if...) 端口: 路径输出。输出已写入数据的文件路径名。

如上所述,可以使用一个文件对话框来选择或输入要操作的文件路径。但是,如果此时选择了对话框的 Cancel 键,则该输出将返回一个 <Not A Path>值。

(3) 写十进制整数文件 (Binary File VIS 子模板 → Write To I16 File.vi)

Write To I16 File.vi 节点把一个一维或二维整数数组中的数据写入到文件。新的数据可以追加到已有文件尾,也可以覆盖原有的数据;如果输入的文件不存在,则创建一个新文件。节点图标及其端口定义如图 8.2.20 所示。

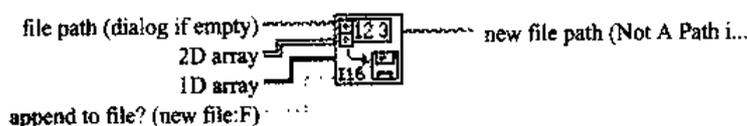


图 8.2.20 Write To I16 File.vi 节点的图标及其端口定义

输入端口 2D array 要求输入二维 16 位整数数组。如果另一个输入端 1D array 的内容为空或没有数据连线,则节点把这个二维数组中的数据写入文件中。其他参数详见 Write to Spreadsheet File 节点。

(4) 写浮点数文件 (Binary File VIS 子模板 → Write to SGL File.vi)

Write to SGL File.vi 节点把一组单精度的浮点数写入到一个流文件中。这个文件如果是已经存在的,则可以在文件尾追加数据或覆盖掉原有的数据;如果输入的文件不存在,则创建一个新文件。写操作完成后,关闭该文件。节点图标及其端口定义如图 8.2.21 所示具体的输入输出关系参见 Write to I16 File。

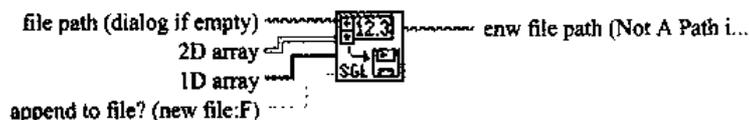


图 8.2.21 Write to SGL File.vi 节点的图标及其端口定义

(5) 写字符串文件 (Write Characters to File.vi)

Write Characters to File.vi 节点把 character string 输入的字符串写到一个新文件中或追加到一个已经存在的文件尾。在写字符串前,打开或创建文件,写操作完成后,关闭文件。节点图标及其端口定义如图 8.2.22 所示

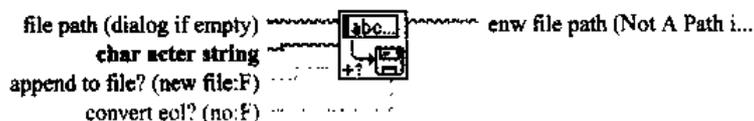


图 8.2.22 Write Characters To File.vi 节点的图标及其端口定义

8.3 文件管理

本节介绍文件的管理操作。包括文件的删除、移动和复制;获取文件、目录的信息;目录、路径操作。

8.3.1 文件的删除、移动和复制

1. 删除文件 (Advance File I/O 子模板 → Delete.vi)

Delete.vi 节点用于删除由 path 输入的文件或目录。节点图标及其端口定义如图 8.3.1 所示。如果一个目录为空或用户没有写文件的权限, 则删除操作无效, 且 error out 将返回一个错误代码。



图 8.3.1 Delete.vi 节点的图标及其端口定义

2. 移动文件 (Advance File I/O 子模板 → Move.vi)

Move.vi 节点用于把一个文件从源位置 (source path) 移到目标位置 (target path)。操作完成后, 原文件被删除。节点图标及其端口定义如图 8.3.2 所示。

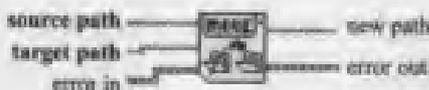


图 8.3.2 Move 节点的图标及其端口定义

3. 复制文件 (Advance File I/O 子模板 → Copy.vi)

Copy.vi 节点用于把文件从源位置复制一份到目标位置, 即进行一个文件拷贝。操作完成后, 原文件仍存在。节点图标及其端口定义如图 8.3.3 所示。

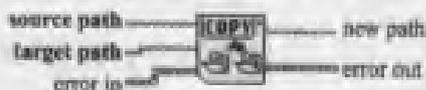


图 8.3.3 Copy 节点的图标及其端口定义

8.3.2 获取文件、目录的信息

1. 返回文件/目录的信息 (File/Directory Info)

File/Directory Info 节点用于返回由路径 path 输入的文件或目录的属性, 比如大小 (size)、文件最后的修改时间 (last mod)。如果 path 输入的仅是目录名, 则 directory 返回 True。节点图标和连接端口如图 8.3.4 所示。

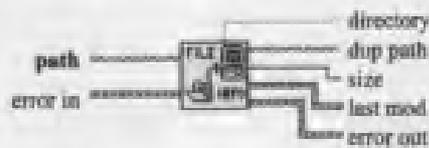


图 8.3.4 File/Directory Info 节点的图标及其端口定义

2. 返回驱动器信息 (Volume Info)

节点图标及其端口定义如图 8.3.5 所示。

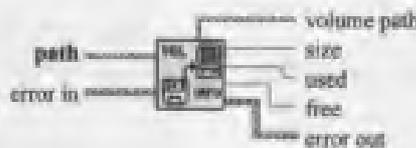


图 8.3.5 Volume Info 节点的图标及其端口定义

Volume Info 节点用于返回由 path 输入的文件或目录所在驱动器的信息,包括存储总容量 (size)、已用容量 (used)、剩余容量 (free)。

3. 文件列表 (List)

List 节点用于用于显示 directory path 输入目录中的文件列表。节点图标及其端口定义如图 8.3.6 所示。

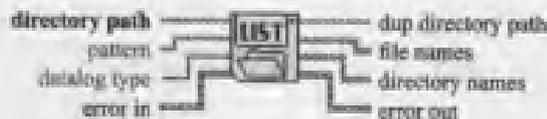


图 8.3.6 List 节点的图标及其端口定义

节点返回两个字符串数组: file names 和 directory names。directory names 列出了这个输入目录下的所有子目录; file names 列出了这个目录下的与 pattern 相匹配的文件 (pattern 参数的设置参见第 8.2.1 小节中 File Dialog 节点)。

8.3.3 路径、目录操作

1. 创建一个路径 (Advance File I/O 子模板 → Build a Path)

Build a Path 节点用于在一个已经存在的路径 (base path) 后添加一个字符串输入,构成一个新的路径名。节点图标及其端口定义如图 8.3.7 所示。



图 8.3.7 Build a Path 节点的图标及其端口定义

在实际的应用中,可以把 base path 设置成工作目录,每次存取文件时就不用再在 Path Control 里输入很长的一个目录名,而只需输入相对路径或文件名。

2. 分离一个路径 (Advance File I/O 子模板 → Strip a Path)

Strip a Path 节点用于把输入路径从最后一个反斜线的位置分成两部分,分别从路径 stripped path 与字符串 name 中输出。因为一个路径的后面常常是一个文件名,所以这个节点可以用来把文件名从路径中分离出来。如要写一个文件重命名的 VI,就可以使用这个节点。节点图标及其端口定义如图 8.3.8 所示。



图 8.3.8 Strip a Path 节点的图标及其端口定义

3. 创建一个目录 (Advance File I/O 子模板 → New Directory)

New Directory 节点用于创建由 directory path 输入的目录。如果同名的目录已经存在,则返回一个错误代码。节点图标及其端口定义如图 8.3.9 所示。



图 8.3.9 New Directory 节点的图标及其端口定义

4. 字符串与文件路径 (path) 的相互转换

Path 与 String 在 LabVIEW 中是两个不同的数据类型,所以它们的节点不能相互处理对方的数据。但在实际应用中却常常相互转换。在 String 子模板 → String/Array/Path Conversion 子模板中,提供了 4 个路径与字符串相互转换的节点。下面逐一介绍这些节点的应法。

(1) Path to Array of Strings.vi

该节点把一个合法的路径拆分成一个字符串数组,数组的每个单元都包含了下一级的目录名,直至文件名,如图 8.3.10 所示。



图 8.3.10 Path to Array of Strings 节点使用举例

(2) Array of Strings to Path.vi

该节点把一个字符串数组转换成一个路径名，如图 8.3.11 所示。

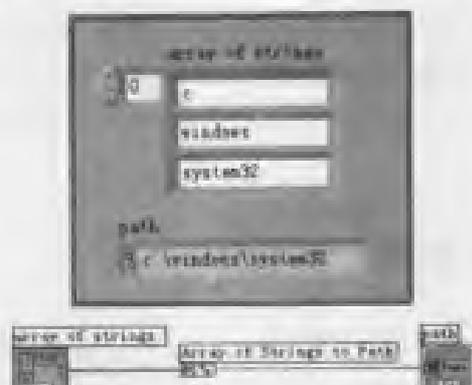


图 8.3.11 Array of Strings to Path 节点使用举例

(3) Path to String.vi 与 String to Path.vi

这两个节点简单地完成两个数据类型之间的转换，如图 8.3.12 所示。



图 8.3.12 Path to String 与 String to Path 节点使用举例

5. 返回 LabVIEW 的工作路径设置

在 LabVIEW 主菜单中选择 Tools→Options，在弹出的 Options 对话框中可以设置 LabVIEW 工作时的各项参数，其中一项就是可以设置各种工作目录，File I/O→File Constant 提供了一些节点可以返回这些设定值。具体功能如下：

	path	返回用户工作时使用的 Temp 目录
	path	返回当前 VI 库的目录
	path	返回当前工作 VI 的路径
	path	返回默认路径
	App Default Data Directory	返回默认数据目录所在的路径

可以用这些返回值来判断用户的工作目录是否设置正确，排除一些应用程序的错误，还可根据这些路径的设置来调整程序文件的读写路径。

8.4 数据存储与读取

在实际的测试应用中, 通常需要将采集到信号数据保存起来, 以便日后进行分析、存档。这就涉及到了数据的存储和读取问题, 作为面向测试应用的软件, LabVIEW 提供了完善的数据存储、读取功能。本节就如何在 LabVIEW 中存储和读取测试数据进行简要的介绍。

8.4.1 LabVIEW 数据文件类型

LabVIEW 支持的数据文件包括三类: 流文件 (Byte Stream File)、块记录文件 (Datalog File)、测试数据文件 (LabVIEW Measurement Data File)。

1. 流文件

顾名思义, 流文件的基本数据单元为 Byte。流文件的数据输入可以是一种单一的数据类型, 也可以是任意数据类型的组合: 字符、数值、布尔值、数值或字符串、字符串数组或簇等。

最简单的流文件可能是一个纯 ASCII 码的文本文件。电子表格文件 (Spreadsheet Text File) 也是一个流文件, 它是一个多行的文本文件, 每一行含有多个数据, 同一行的数据与数据之间由制表符隔开, 行与行之间由回车符区别开。LabVIEW 产生的这两种文件可以用任何一种文本编辑器打开。还有一种比特流文件是纯数值的二进制文件, 它的数据单元可以是整数数组或浮点数数组。

总之, 一个流的数据可以是任何的数据类型或它们的组合。LabVIEW 的所有文件操作节点都适用于流文件的操作。

流文件有两种类型: ASCII 码文件 (ASCII Byte Stream) 和二进制文件 (Binary Byte Stream)。

ASCII 码文件 (即文本文件)。这种格式的文件可以被任何的其他文本编辑器打开, 具有很好的直观性和兼容性。但是, 如果想用这种文件存储数值, 如十进制的整数或浮点数, 则在写文件前必须进行数值到字符串的转换。数据读出后, 还必须进行字符串到数值的转换。有关数值与字符串转换的内容请参考 4.4 节“字符串节点介绍”。

电子表格文件也是 ASCII 码文件, 在读写时也必须进行数据转换, 只不过转换过程由电子表格读写节点自动完成。

二进制文件具有更快的处理速度及更紧凑的文件结构。但是, 这种文件因为在存储时, 不管写文件时的数据类型是什么, 在文件的结构上都是一样的, 都是一个一个的字节, 所以, 在读数据时, 如果希望准确地还原出写入的数据类型, 则必须知道写数据时的数据结构, 而文件本身及文件操作节点并不提供这种数据结构的任何信息。

2. 块记录文件

块记录文件是 LabVIEW 独有的一种文件形式, 它的基本数据单元为组成结构的记录块 (Record)。这些记录块可以是 LabVIEW 的任何数据类型, 也可以是它们的组合构成,

例如簇。同一个文件的数据块必须有相同的结构。

一个流文件可以在文件尾追加一个新的数据，也可以在文件的任何位置覆盖掉一些数据；但一个块记录文件只能在文件尾增加或删除一个记录，它不能在任意位置覆盖一个已经存在的记录。

在读取一个流文件时，定位方法要指明定位是从这个文件的第几个字节开始的，以字节为单位。而一个块记录文件的定位方法则要指明定位是从第几个记录开始的，以及要读取多少个记录。

当一个 VI 要连续采集数据，或者在处理已存盘的数据时，不仅要按顺序对数据进行处理，而且有时也要随机地读取数据（或者这些数据有可能要被其他程序应用）。最好使用流文件，因为块文件除了 G 语言外，用其他的应用程序很难打开。

如果要采集很多组数据，每组数据里同时又包含多个测量通道的数据，而且每组数据都需要单独处理时，使用 LabVIEW 的块记录文件格式存储是最合适的，因为每个记录块都有一定的独立性，这有助于数据组之间的区分。

3. 测试数据文件

测试数据文件是 LabVIEW 7 Express 新增加的一种文件类型。可用于存储多种类型的数据，包括各种类型的实数（整数、双精度数据、扩展精度数据）、实数数组、波形数据（Waveform）、波形数组、布尔量。LabVIEW 提供了两个 Express VI 负责测试数据的读和写，使用起来非常方便。测试数据文件实质上是 ASCII 码文件，可以使用任何文本编辑工具查看，除了数据本身，测试数据文件还包含一些说明信息，如文件创建的时间、日期、创建人等。

8.4.2 数据文件存储与读取

无论哪种类型的文件，其存储与读取的基本流程都是相同的，即打开文件、读文件或写文件、关闭文件，如图 8.4.1 所示。文件操作节点的功能在第 8.2 节已经介绍，本节通过几个实例进一步说明数据文件存储与读取的具体方法。

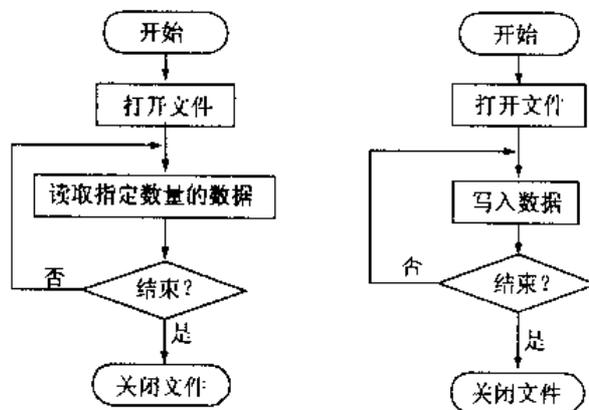


图 8.4.1 文件读写流程

1. ASCII 码文件的读写

例 8.4.1 写入 ASCII 文件。

本例的操作步骤如下。

- ① 使用 Open/Create/Replace File.vi 打开一个文件，它的 function(open:0) 端口设为 2，即创建新文件或替换已有文件。文件名的后缀并不重要，但习惯上常取“txt”或“dat”。
- ② 利用 While 循环将多块数据写入文件。
- ③ 使用 Close File 节点关闭数据文件。

本例中的信号源是一个随机噪声，VI 的前面板和框图程序如图 8.4.2 所示。

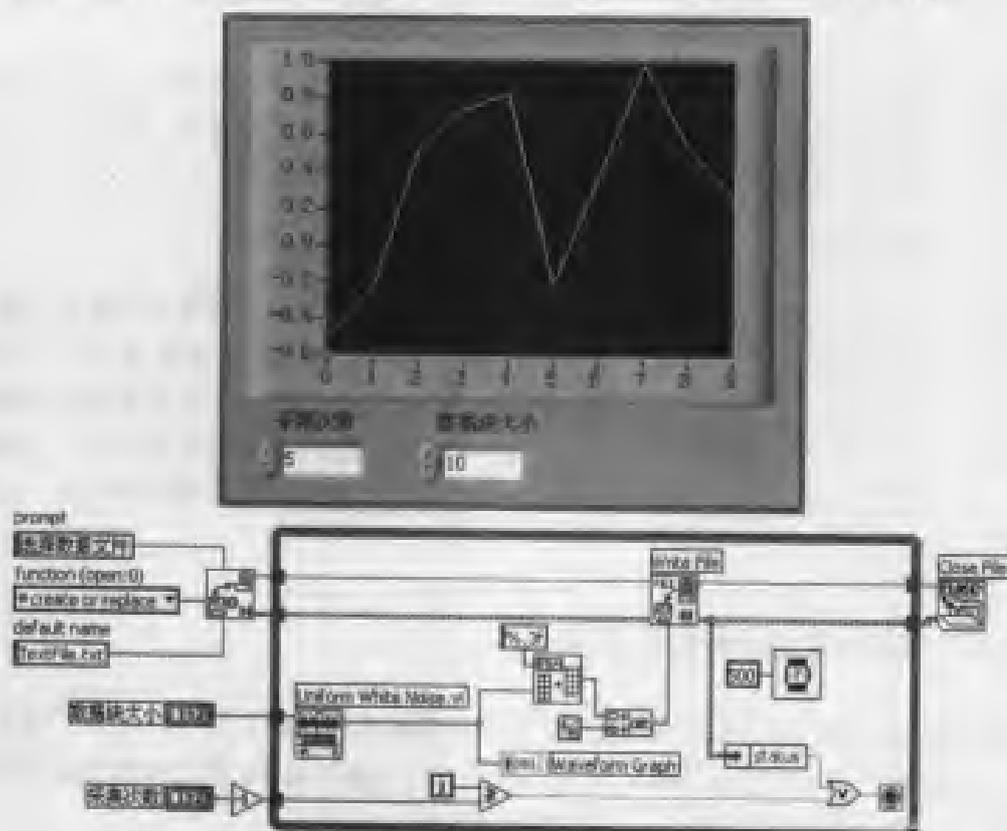


图 8.4.2 例 8.4.1 的前面板和框图程序



图 8.4.3 Array To Spreadsheet String.vi 节点

在将数据写入文件之前，使用 Array To Spreadsheet String.vi 节点把信号数据转换成字符串。Array To Spreadsheet String.vi 节点的图标如图 8.4.3 所示，delimiter 端口设置数据之间的分隔符，默认设置为 tab。format string 端口用来设置转换格式。

在本例中，转换时使用的格式为“%.3f”，即三位精度的浮点数。转换后的数据之间使用 tab 作为分隔符。为了区分每次采样的数据，在每个转换后的数据块的结尾输入一个回车符，使数据文件中的每一行对应着一次采样。本例设定采样 5 次，每次采集 10 个数据点。数据采集完毕后，关闭数据文件。

可以使用 Windows 操作系统的文本编辑工具查看文件中的数据。图 8.4.4 所示的是用

记事本打开的本例所存储的数据文件。从图 8.4.4 可以看出，数据共有 5 行 10 列，每一行对应着一次采集，每次采集的数据块中包含 10 个数据。

文件(F)	编辑(E)	格式(O)	查看(V)	帮助(H)					
-0.802	-0.848	0.339	0.221	0.915	-0.484	0.798	-0.757	-0.638	0.673
0.786	-0.532	-0.879	0.548	0.982	0.622	-0.742	0.881	-0.511	0.542
-0.885	0.839	0.474	0.252	0.188	0.488	0.398	-0.634	-0.119	0.747
0.465	0.884	0.616	0.124	-0.359	-0.146	-0.998	0.197	-0.891	-0.988
-0.547	0.499	-0.261	-0.717	0.512	-0.478	0.378	-0.182	0.548	-0.127

图 8.4.4 ASCII 码数据文件的内容

例 8.4.2 读 ASCII 文件。

本例的操作步骤如下。

① 使用 Open/Create/Replace File.vi 打开文件（也可使用 Open File.vi），其图标如图 8.4.5 所示。Open/Create/Replace File.vi 的输出端口有一个是 file size，这个端口输出以字节为单位的文件大小。

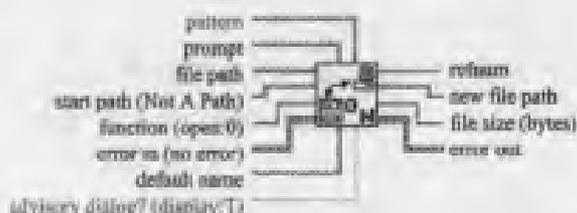


图 8.4.5 Open/Create/Replace File.vi 的图标

② 使用 Read File.vi 将所有数据读出。读出数据后，使用 Spreadsheet String To Array.vi 将字符串转换成双精度数据。要注意的是，Spreadsheet String To Array.vi 的设置 delimiter 和 format string 必须和写数据文件 Array To Spreadsheet String.vi 的设置相同，这样才能保证正确的恢复数据。读取出来的数据使用 Graph 控件显示出来。

③ 使用 Close File 节点关闭数据文件。

VI 的前面板和框图程序如图 8.4.6 所示。

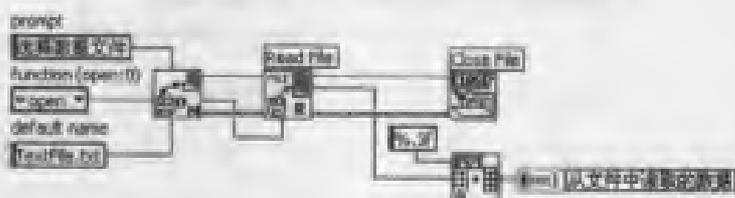
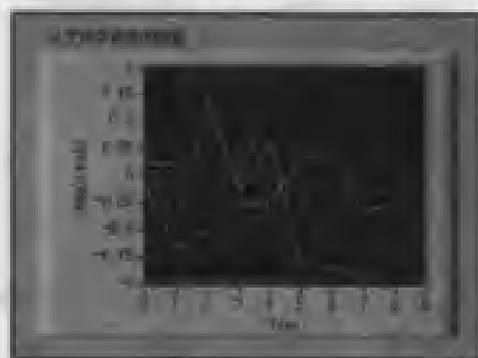


图 8.4.6 例 8.4.2 的前面板和框图程序

由以上两个例子可以看出 ASCII 文件的特点是:

- 无论读还是写都需要进行数据转换。数据转换是需要时间的,特别是当数据块比较大时。所以在数据采集速率较高的场合,不宜使用 ASCII 码文件存储数据。如果数采速率很高,写文件不及时,则会产生数据丢失现象,即数据文件只记录了部分数据。

- 体积大。在 ASCII 码文件中,一个字符要占用一个字节的空,这是非常浪费的,例如一个拥有 10 个数字的整数,在 ASCII 码文件中要占用 10 个字节,而在内存中表示这个整数只需要 2 个字节而已。ASCII 码文件的可读性是以牺牲磁盘空间为代价的。

在数据采集速率较高的情况下,宜使用二进制数据文件。

2. 二进制文件的读写

二进制数据文件体积小,在存储时无须数据转换,尤其适合于数据量巨大,数采速率高的场合。

例 8.4.3 写二进制文件。

本例的操作步骤如下。

- ① 使用 File Dialog.vi 打开文件对话框,选择文件路径。
- ② 使用 New File.vi 创建一个新的文件。通过 While 循环采集数据并将数据写入文件。
- ③ 使用 Close File 节点关闭数据文件。

VI 的前面板和框图程序如图 8.4.7 所示。

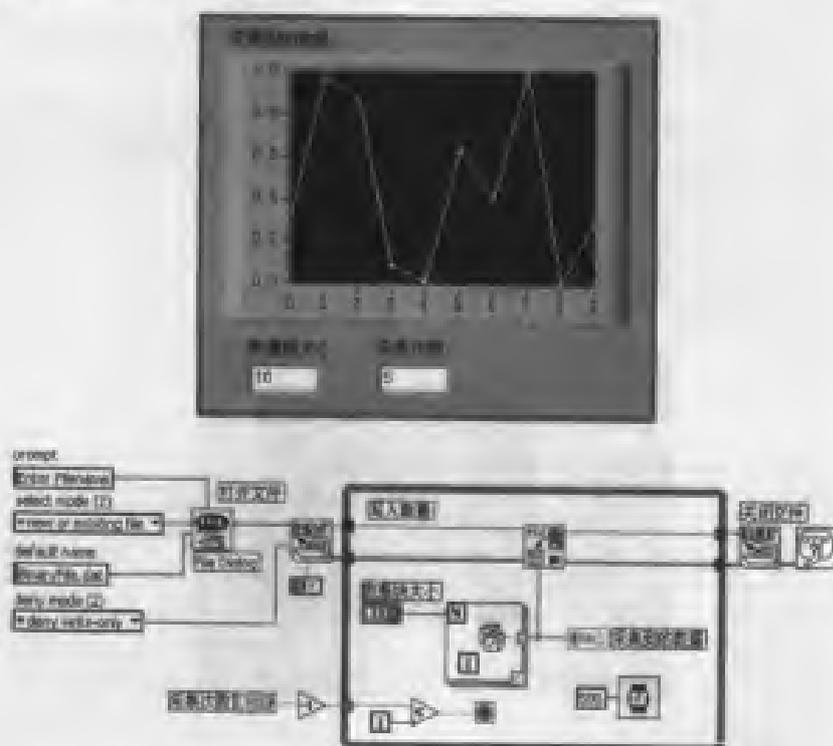


图 8.4.7 例 8.4.3 的前面板和框图程序

在本例中,信号源是一个随机数产生器,通过 For 循环将随机数组成数组。可以看到,在存储数据时,是将双精度数组数据直接写入文件的,而没有经过数据转换,因此写二进制文件的速度是很快的。

需要注意的是，应该把文件的打开和关闭操作放在 While 循环的外面。如果单独将打开文件的操作放在循环里面，将会重复打开文件；如果单独将关闭文件的操作放在循环内部，在循环第一次运行结束后，文件 Refnum 将被关闭，在循环第二次运行时，Close File.vi 将试图关闭一个不存在的文件 Refnum，程序会报错；如果同时将文件打开和关闭操作放在循环内，虽然程序能够运行，但数据文件中只能记录最近一次采集的数据。

例 8.4.4 读二进制文件。

读二进制数据文件时需要注意两点：一是计算数据量；二是必须知道存储文件时使用的数据类型。

本例的操作步骤如下。

- ① 使用 Open File.vi 打开文件。
- ② 利用 EOF.vi 计算文件长度，并根据所使用的数据类型的长度计算出数据量，本例中的数据类型为双精度数据，每个双精度数据占用 8 个字节，所以数据量等于文件长度除以 8。使用 Read File.vi 读取数据时，必须指定数据类型，方法是所需类型的数据连接到 Read File.vi 的 datalog type 端口。
- ③ 读取完毕，使用 Close File 节点关闭数据文件。

VI 的前面板和框图程序如图 8.4.8 所示。

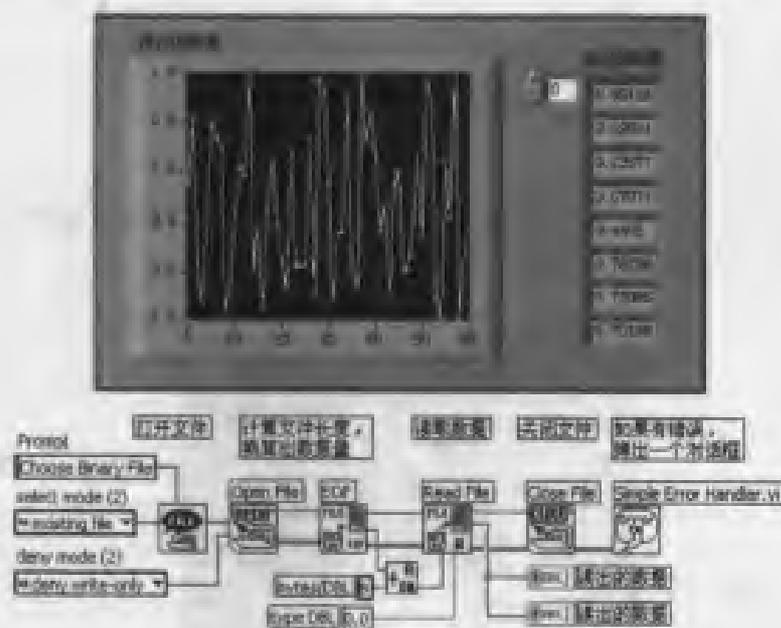


图 8.4.8 例 8.4.4 的前面板和框图程序

3. 块记录文件的读写

例 8.4.5 写块记录文件。

VI 的前面板和框图程序如图 8.4.9 所示。块记录文件的读写方法与二进制文件的读写方法类似，只是在读写时要指定数据类型。

在使用 New File.vi 创建文件时，必须指定数据类型，方法是所需类型的数据连接到 New File.vi 的 datalog type 端口。指定的数据类型必须和要存储的数据的类型相同。

文件对话框节点 File Dialog.vi 也有一个 datalog type 端口，如果连接了此端口，在文件

对话框中将只显示指定数据类型的文件。

在本例中, 每个信号数据由两部分组成: 一个以字符串表示的时间; 一个双精度数组。这两种数据打包后组成一个簇, 然后写入文件。

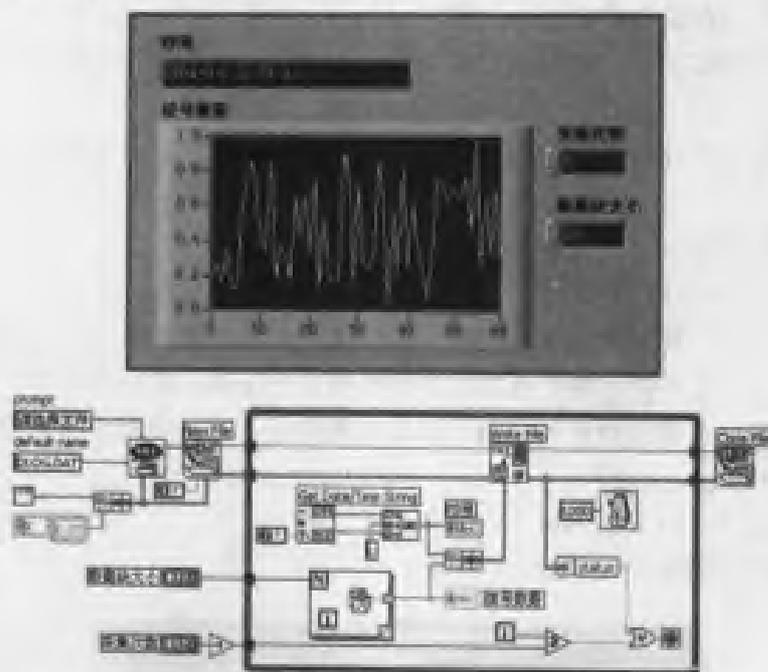


图 8.4.9 例 8.4.5 的前面板和框图程序

例 8.4.6 读块记录文件。

VI 的前面板和框图程序如图 8.4.10 所示。在打开块记录文件时也要指定数据类型。在将数据读出后使用 Unbundle.vi 将数据解包, 并且分别显示。

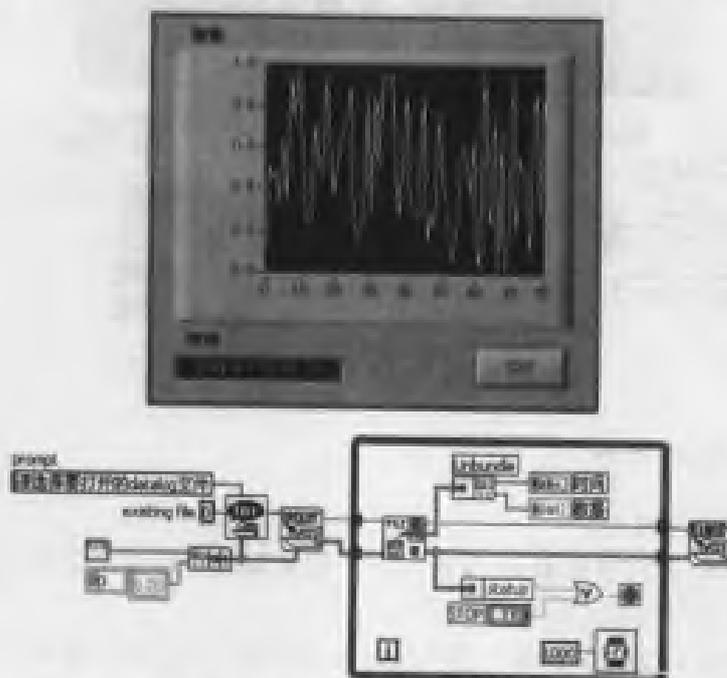


图 8.4.10 例 8.4.6 的前面板和框图程序

在使用 Read File.vi 读取数据时，并没有指定读取的数据个数，Read File.vi 默认为读取一个数据。在遇到文件尾之后，Read File.vi 将报错，While 循环捕获到这一信息，停止运行。在数据读取过程中，如果用户按下了 Stop 按钮，程序也将中止运行。

4. 测试数据文件的读写

测试数据文件的读写相对容易，LabVIEW 提供了两个 Express VI 用来完成对测试数据的读写，它们是 Write LabVIEW Measurement File 和 Read LabVIEW Measurement File，使用时只要进行简单的设置即可。这两个 Express VI 实际上是在内部集成了多个文件读写 VI。测试数据文件本质上属于 ASCII 码文件，它的内容是可阅读的，可以使用任何文本编辑工具查看。

例 8.4.7 写测试数据文件。

本例首先产生一个正弦波，然后使用 Write LabVIEW Measurement File 节点将波形数据写入测试数据文件。Write LabVIEW Measurement File 节点位于 Functions 模板→Output 子模板→Write LabVIEW Measurement File。VI 的前面板和框图程序如图 8.4.11 所示。

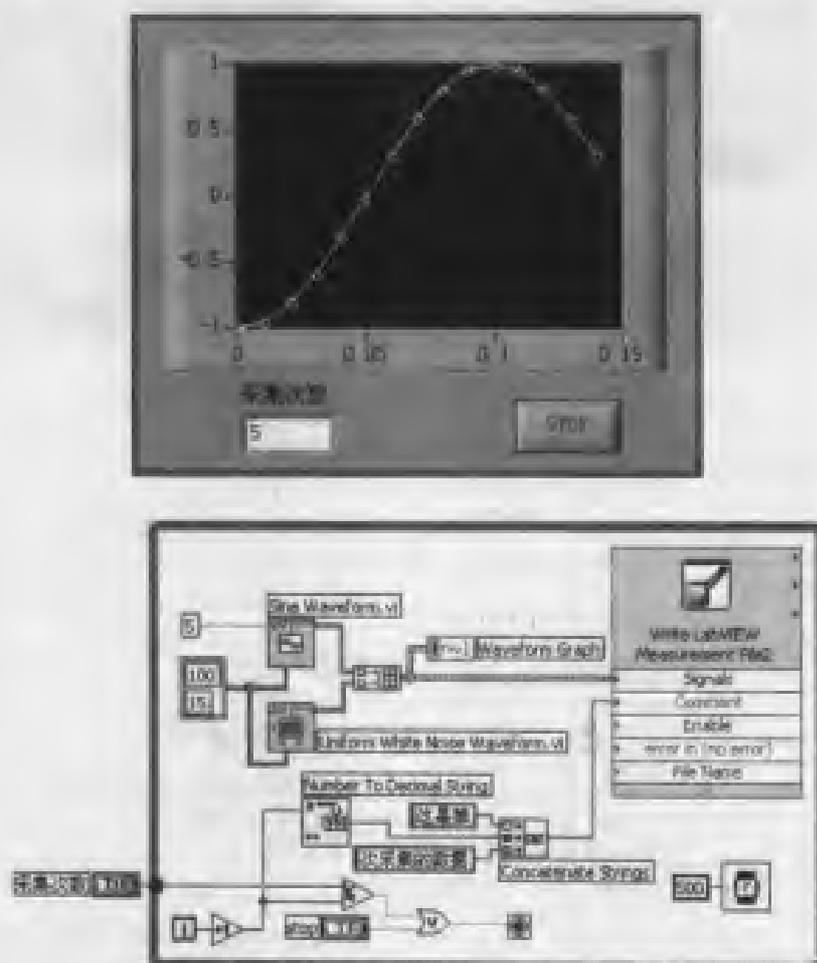


图 8.4.11 例 8.4.7 的前面板和框图程序

在使用 Write LabVIEW Measurement File 时，必须对它进行设置。在默认设置时，将 Express VI 放到框图中，LabVIEW 将打开 Express VI 属性面板。在程序设计中，也可以双

击 Express VI 的图标打开它的属性面板, Write LabVIEW Measurement File 的属性对话框如图 8.4.12 所示。



图 8.4.12 Write LabVIEW Measurement File 的属性对话框

各项设置说明如表 8.4.1 所示。

表 8.4.1 Write LabVIEW Measurement File 的属性设置说明

选项	说明
File name	指定测试数据文件的存储路径
Action	设定存储时的相关操作如下 Save to one file: 将所有数据保存到一个文件中 Ask user to choose file: 运行时弹出对话框提示用户选择文件 Ask only once: 只提示用户一次 Ask each iteration: 在 Express VI 每次运行时都提示用户选择文件 Save to series of files (multiple files): 将数据存入一系列文件中 Settings: 设定系列文件的命名规则和创建规则
If a file already exists	用户指定的文件已存在时执行的操作 Rename existing file: 重命名已存在的文件 Use next available name: 使用下一个可用的文件名, 此项设定针对系列文件, 例如, 如果 test001.lvm 已存在, 将根据命名规则使用下一个可用的文件名 test002.lvm Append to file: 将数据粘贴到已存在的文件的尾部 Overwrite file: 覆盖已有文件
Segment Headers	设置每个数据段的标题 One header per segment: 每个数据段都拥有自己的标题 One header only: 所有数据使用一个标题 No headers: 不使用标题

续表

选 项	说 明
X Value Columns	设置 X 轴数据的存储的属性 One column per channel : 存储时, 每个通道的数据组成一列, 存储数据时可以同时存储数据的时间信息, 即 X 轴数据, 如果选择 One column per channel, 则每个通道的数据都有自己的 X 轴数据 One column only : 所有通道使用同一个 X 轴数据列 Empty time column : 不存储时间信息
Delimiter	存储数据时使用的分隔符。 Tab : 使用 Tab 符号作为分隔符 Comma : 使用逗号作为分隔符
File Description	文件的描述信息, 该信息将被添加到数据文件的标题中

在本例中, 所有的数据均存入同一个文件中。如果要将每次采集的数据存入不同的文件, 可以在属性对话框 Action 一栏中选择 Save to series of files (multiple files), 将数据存入一系列按一定规则命名的文件中。Write LabVIEW Measurement File 节点的 Comment 端口用以输入每个数据段的说明。

例 8.4.8 读测试文件。

VI 的前面板和框图程序如图 8.4.13 所示, 使用 Read LabVIEW Measurement File 节点将数据和说明信息读出并显示, Read LabVIEW Measurement File 节点位于 Functions—模板—Input 子模板中。



图 8.4.13 例 8.4.8 的前面板和框图程序

打开 Read LabVIEW Measurement File 的属性对话框, 进行相应设置, 如图 8.4.14 所示。



图 8.4.14 Read LabVIEW Measurement File 的属性对话框

各项设置说明如表 8.4.2 所示。

表 8.4.2 Read LabVIEW Measurement File 的属性设置

项 目	说 明
File name	指定测试数据文件的存储路径
Action	单点运行时执行的操作 Ask user to choose file: 弹出对话框提示用户选择测试数据文件
Segment Size	设置读取数据段的大小 Retrieve segments of original size: 按原始大小读出数据段 Retrieve segments of specified size: 指定数据段的大小 "Samples": 数据段的大小, 以数据段计
Time Stamps	设置时间格式 Relative to start of measurement: 使用相对时间, 测试开始时间为零点 Absolute (date and time): 显示绝对时间

续表

选 项	说 明
Generic Text File	从通用文本文件中读取数据 Read generic text files: 从通用文本文件中读取数据 Start row of numeric data : 指明数据开始的列 First row is channel names : 告诉程序文本文件中的第一列为通道名 First column is time channel : 告诉程序文本文件中的第一列为时间数据 Read File Now : 立即读取数据, 并显示在 Sample data 表格中
Delimiter	指明数据的分隔符 Tab : 使用 Tab 作为分隔符 Comma : 使用逗号作为分隔符
Decimal point	指明十进制数据的分位符

第9章 数学分析与信号处理

测试的目的在于获取被测对象的性能、状态或特征，所以信号采集只是测试工作的第一步。信号的分析 and 数据处理是构成测试系统的重要组成部分之一，常用的分析方法可以分为数学分析和数字信号处理两大类，其中大部分方法已经形成了标准的算法程序。LabVIEW 提供了内容丰富、功能强大的分析节点，配合出色的数据显示工具，完全可以胜任复杂的信号分析与数据处理工作。另外，LabVIEW 还针对各种特定应用提供了多种专门的分析工具包（需单独购买），这些工具包有以下几种。

- Signal Processing Toolkit: 联合时频分析、小波分析及滤波器设计工具包。
- Sound and Vibration Toolkit: 声音和震动信号分析工具包。
- Order Analysis Toolkit: 阶次分析工具包。
- Modulation Toolkit: 调制工具包。
- Vision and Image Processing Toolkit: 高级图像分析工具。

这些工具包既可以在 LabVIEW 开发环境中使用，也可以脱离 LabVIEW 独立运行。

本章将介绍 LabVIEW 的数学分析和信号处理功能。有关数学分析和信号处理的节点位于 Functions 模板 → All Functions 子模板 → Analyze 子模板中，如图 9.0.1 所示。

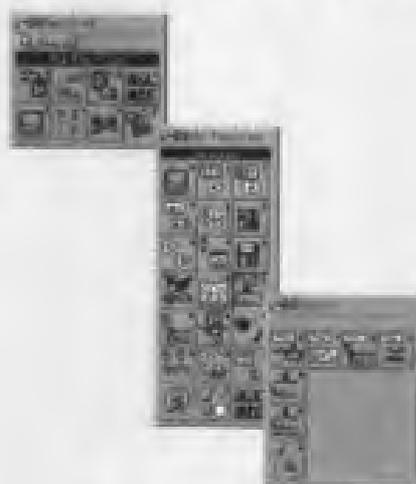


图 9.0.1 Analyze 子模板

Analyze 子模板共包含 7 个子模板，各子模板的功能如表 9.0.1 所示。

表 9.0.1 Analyze 子模板中的各功能模板

子模板名称	图标	说明
Waveform Measurements		波形测量模板，功能包括直流—交流成份分析、振幅测量、脉冲测量、傅里叶变换、功率谱计算、谐波畸变分析、过波分析、频率响应、信号提取等

续表

子模板名称	图标	说 明
Waveform Conditioning		波形调理子模板, 提供 FIR 滤波器、IIR 滤波器、归一化窗函数
Waveform Monitoring		波形监测模板, 功能包括边界测量、尖峰捕获、触发检测
Waveform Generation		波形产生模板, 可以产生正弦波、方波、三角波、锯齿波、白噪声、高斯噪声、周期随机噪声, 还可以利用公式产生函数波形
Signal Processing		数字信号处理模板, 包括信号产生, 大量时域和频域分析函数, 各种滤波器和窗函数
Point By Point		逐点信号分析库, 这是自 LabVIEW6.1 起, NI 新推出的库函数, 与以往基于数组和缓冲的分析库不同, 逐点信号分析库的分析可以针对每个数据, 无须缓冲, 因此更适用于实时系统
Mathematics		数学分析库, 高级分析库的一部分由数学库组成, 因为没有数学库的帮助, 分析起来是不方便的

Analyze 子模板提供了丰富的数学分析和信号处理节点, 共有 400 多个。由于节点数量众多, 故不能一一详细介绍。本章在介绍各节点时, 将首先说明模板的总体功能, 然后给出模板中所含节点的功能列表, 最后利用具有代表性的节点给出应用实例。

9.1 数学分析

LabVIEW 自身集成了丰富而功能强大的数学工具, 这些工具涵盖了线性代数、概率统计、最优化、曲线拟合、微积分等方面的应用, 为用户的编程提供了极大的方便, 同时也是对数字信号处理节点的有力支持。数学分析节点位于 Functions 模板→All Functions 子模板→Analyze 子模板→Mathematics 子模板中, 如图 9.1.1 所示。下面分别介绍 Mathematics 子模板中的各个节点的功能。

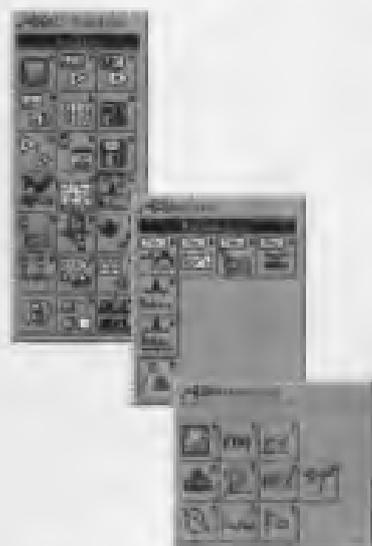


图 9.1.1 Mathematics 子模板

9.1.1 公式运算节点

公式运算节点主要提供了将外部公式或数学描述直接连入到 LabVIEW 中的功能, 对于不太复杂的公式和运算过程, 使用公式节点更加灵活方便。同时, LabVIEW 提供了与 MATLAB 的接口, 可以通过使用 MATLAB 语言节点 (MATLAB Script), 在 LabVIEW 环境中编辑、运行 MATLAB 程序。

1. 公式运算节点

公式运算节点位于 Functions 模板 → All Function 子模板 → Analyze 子模板 → Mathematics 子模板 → Formula 子模板中, 如图 9.1.2 所示。



图 9.1.2 Formula 子模板

公式运算模板中的节点如表 9.1.1 所示

表 9.1.1 公式运算模板中的节点

节点名称	图标及端口	功能
Formula Node		基本公式节点, 使用传统的描述语言完成计算功能, 语法类似 C 语言
MATLAB Script		MATLAB 语言节点, 编辑并执行 MATLAB 程序, 系统需要安装 MATLAB5.0 以上版本
Eval Formula Node.vi		类似 Formula Node, 但是可以通过前面板输入变量
Advanced Formula Parsing		高级公式节点子模板, 含有众多高级公式翻译、计算节点

续表

节点名称	图标及端口	功 能
Formula		Express VI 类型的公式节点

2. 公式运算节点应用实例

公式节点实际上是 LabVIEW 与传统的文本语言的接口，可以在公式节点中书写小的程序。对于那些由简单的函数计算组成的运算过程，使用公式节点比使用 LabVIEW 的数学运算节点更为方便。

有关基本公式节点的使用方法，在本书 6.6 节中已有讲述，在此不再重复。

例 9.1.1 扩展的公式节点 Eval Formula Node.vi 的使用。

Eval Formula Node 节点和基本公式节点差不多，但是它更加灵活。基本公式节点的数学表达式只能直接书写在框图程序的基本公式节点中，这意味着在程序运行过程中，基本公式节点的内容是不能修改的。Eval Formula Node 节点允许从前面板输入公式和参数，使用起来灵活性更大。Eval Formula Node 节点的图标及其端口定义如图 9.1.3 所示。

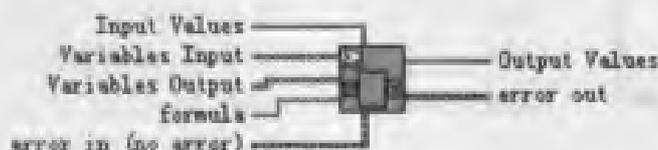


图 9.1.3 Eval Formula Node 节点的图标及其端口定义

节点的输入输出端口见表 9.1.2。

表 9.1.2 Eval Formula Node 节点的端口说明

端口名称	说 明
Input Values	输入变量值
Variables Input	输入变量名
Variables Output	输出变量值
Formula	公 式
Error in	错误信息输入
Output Value	计算结果
Error out	错误信息输出

其中，Input Values 与 Variables Input 一一对应，Variables Output 与 Output Value 也一一对应。

本例将从面板输入各项参数和公式并计算结果。VI 的前面板和框图程序如图 9.1.4 所示。

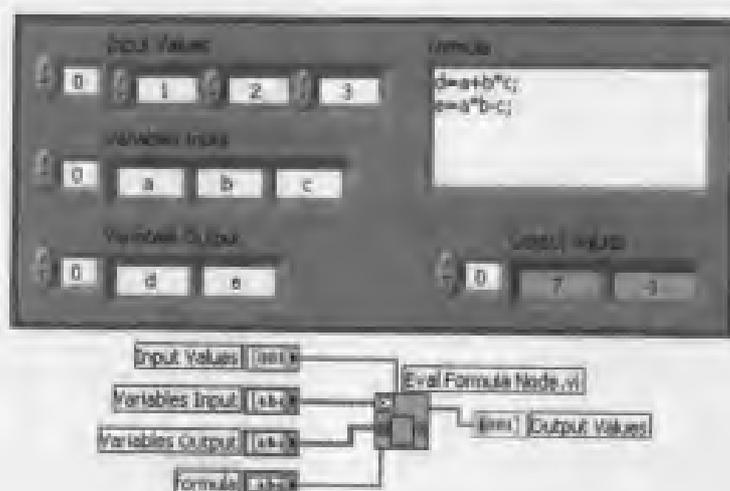


图 9.1.4 例 9.1.1 的前面板及框图程序

例 9.1.2 MATLAB 语言节点 (MATLAB Script).



图 9.1.5 MATLAB 语言节点

MATLAB 是一个常用的数学运算工具, 将它和 LabVIEW 有机地结合起来, 在大型的系统测试与仿真过程中是非常有利的。有 MATLAB 语言节点就可以在 LabVIEW 中直接运行 MATLAB 程序了。在使用 MATLAB 语言节点时, 系统中要安装 MATLAB 5.0 以上的版本。节点的形式如图 9.1.5 所示, 与公式节点类似, MATLAB 语言节点同样可以在边框上添加输入输出接口。

可以在 MATLAB 语言节点中直接书写 MATLAB 程序, 也可以将写好的 MATLAB 程序导入 MATLAB 语言节点。在导入 MATLAB 程序时, 按照以下 3 个步骤进行:

- 在节点上单击右键;
- 在弹出式菜单中选择“Import”;
- 在弹出的对话框中选择要导入的文件并确认。

完成这 3 步操作后, 要导入的文件内容就出现在节点中。

本例利用 MATLAB Script 节点调用一个简单的 MATLAB 程序, 生成一个三维曲线图, 运算结果及框图程序如图 9.1.6 所示。程序运行后, LabVIEW 将自动启动 MATLAB。本例是使用 MATLAB 绘制并显示函数图形的, 也可以使用 LabVIEW 自带的三维波形图显示, 有关三维波形图的使用方法请参考本书第 7 章的相关内容。

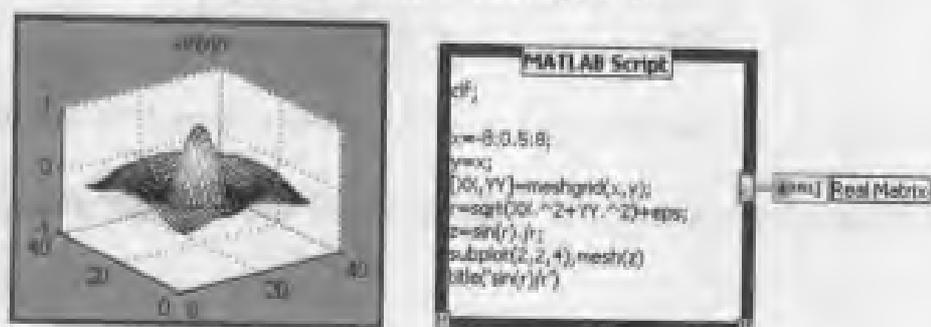


图 9.1.6 例 9.1.2 的运算结果和框图程序

9.1.2 函数计算与微积分

1. 函数计算

1D、2D Evaluation 模板提供一元、二元函数的计算功能。函数可以是因变量—自变量的形式，也可以用参数方程给出。坐标系可以采用笛卡儿坐标系或极坐标系。

(1) 函数计算节点

函数计算节点位于 Functions 模板 → All Functions 子模板 → Analyze 子模板 → Mathematics 子模板 → 1D and 2D Evaluation 子模板中，如图 9.1.7 所示。



图 9.1.7 1D and 2D Evaluation 子模板

1D and 2D Evaluation 子模板中的节点如表 9.1.3 所示。

表 9.1.3 1D、2D Evaluation 模板中的节点

节点名称	图标	功能
Eval $y=f(x)$.vi		以给定步长计算一元函数的值。步长 $h=(start-end)/number\ of\ points$
Eval $y=f(x)$ Optimal Step.vi		计算一元函数的值。函数更复杂，计算精度更高

续表

节点名称	图标	功能
Eval $y=f(x)$.vi		类似于 Eval $y=f(x)$, 但是可以输入若干参数
Eval X-Y(t).vi		计算用参数方程表示的函数 $y=f(t)$, $x=g(t)$ 的值
Eval X-Y(t) Optimal Step.vi		类似 Eval X-Y(t).vi, 计算精度更高
Eval X-Y(a,t).vi		类似 Eval X-Y(t).vi, 但是可以输入若干参数
Eval Polar to Rect.vi		计算极坐标函数的值, 并转换到直角坐标系, 输出为函数的 x,y 坐标
Eval Polar to Rect Optimal Step.vi		类似 Eval Polar to Rect.vi, 计算精度更高
Eval $y=f(x1,x2)$.vi		计算二元函数的值, 函数定义在一个矩形区间上
Eval $y=f(a,x1,x2)$.vi		计算二元函数的值, 函数定义在一个矩形区间上, 允许输入若干参数
Eval X-Y-Z(t1,t2).vi		计算三维空间中的一个曲面
Eval X-Y-Z(a,t1,t2).vi		类似于 Eval X-Y-Z(t1,t2), 允许输入若干参数

(2) 函数运算节点应用实例

例 9.1.3 计算函数 $\sin(x)/x$ 的值。

本例将使用 Eval $y=f(x)$.vi 计算 $\sin(x)/x$ 的值。Eval $y=f(x)$.vi 的功能是以给定步长计算一元函数的值，其图标如图 9.1.8 所示。端口 formula 连接函数表达式。端口 start 设置自变量的起点，端口 end 设置自变量的终点。端口 number of points 设置参与计算的点数，它的值等于区间[start, end]被分割成的段数。

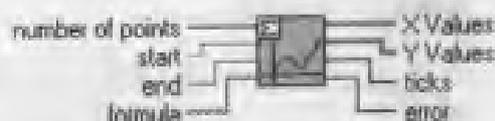


图 9.1.8 Eval $y=f(x)$.vi 的图标

计算结果打包后送到 XY Graph 显示出来，本例的前面板与框图程序如图 9.1.9 所示。

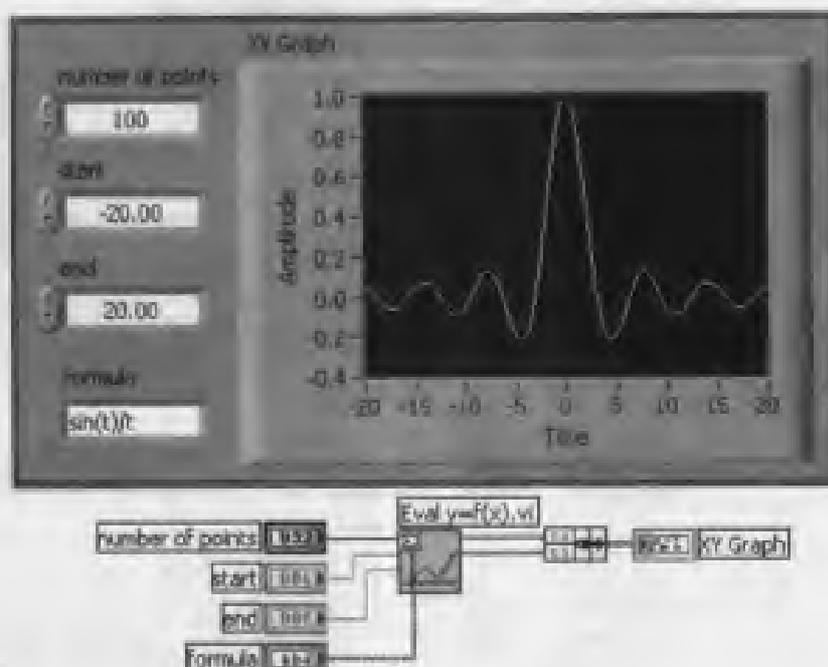


图 9.1.9 例 9.1.3 的前面板和框图程序

2. 微积分运算

Calculus 模板提供了有关微积分运算和零、极点求解的常用函数。包括数值积分，数值微分，函数曲线长度计算，一元函数的零、极点求解，二元函数的极点求解，偏微分方程、常微分方程的求解等。

(1) 微积分运算节点

微积分运算节点位于 Functions 模板 → All Functions 子模板 → Analyze 子模板 → Mathematics 子模板 → Calculus 子模板中，如图 9.1.10 所示。



图 9.1.10 Calculus 模板

Calculus 子模板中的节点如表 9.1.4 所示。

表 9.1.4 Calculus 模板中的节点

节点名称	图标	功能描述
Numeric Integration.vi		对指定的数组进行数值积分
Integration.vi		在指定的区间上对给出的函数进行数值积分
Differentiate.vi		在指定的区间上, 计算函数在各点的微分
Limit.vi		计算函数在某点的左极限和右极限
Curve Length.vi		计算函数曲线的长度
Partial Derivatives of f(x1,x2).vi		计算二元函数的偏微分

续表

节点名称	图标	功能描述
Extrema of $f(x_1, x_2)$.vi		求解二元函数的极值点, 包括极大值点和极小值点
Zeros and Extrema of $f(x)$.vi		求解一元函数的零点和极值点, 并解出极值
ODE Runge Kutta 4th Order.vi		4阶 Runge Kutta 法, 求解微分方程的经典方法
ODE Cash Karp 5th Order.vi		用5阶 Cash Karp 法求解微分方程
ODE Euler Method.vi		用 Euler 法求解微分方程
ODE Linear nth Order Numeric.vi		求解 n 阶齐次常微分方程, 返回数值解
ODE Linear nth Order Symbolic.vi		求解 n 阶齐次常微分方程
ODE Linear System Numeric.vi		求解 n 维齐次线性系统的状态方程, 返回数值解
ODE Linear System Symbolic.vi		求解 n 维齐次线性系统的状态方程

(2) 微积分运算应用实例

例 9.1.4 一元函数积分。

本例将计算 $Y=X^3$ 在区间 $[0, 10]$ 上的积分。

计算所使用的节点是 Integration.vi。Integration.vi 根据给定的函数式, 在起点和终点间进行曲线积分, 节点图标及其连接端口如图 9.1.11 所示。

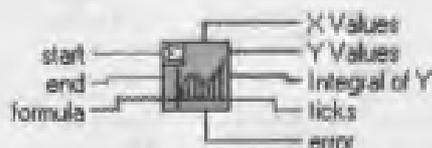


图 9.1.11 Integration.vi 的图标及其连接端口

节点的输入输出端口说明见表 9.1.5。

表 9.1.5 Integration.vi 节点的输入输出端口说明

端口名称	说 明
Start	积分起点
End	积分终点
Formula	被积函数
X values	计算点的自变量值
Y values	计算点的函数值
Integral of Y	计算点的函数积分值
Ticks	用时统计
Error	错误代码

在计算中, 程序自动将积分区间分成 200 份, 所以计算点共有 201 个, 输出的 3 个数组长均为 201。

前面板和框图程序如图 9.1.12 所示。

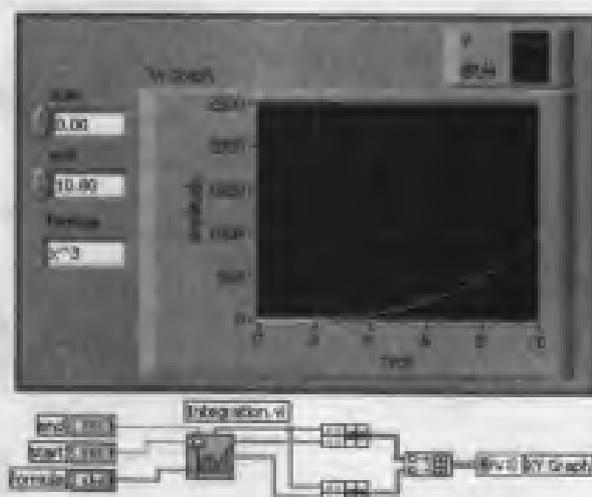


图 9.1.12 例 9.1.4 的前面板及框图程序

9.1.3 概率统计与曲线拟合

1. 概率统计

概率统计的理论与方法在科学技术领域的应用十分广泛。LabVIEW 提供了相应的概率统计节点，这些节点主要包含了 3 部分内容，基本概率统计函数、分布函数和方差分析。

(1) 概率统计节点

与概率统计相关的节点位于 Functions 模板→All Functions 子模板→Analyze 子模板 Mathematics 子模板→Probability and Statistics 子模板中，如图 9.1.13 所示。



图 9.1.13 概率统计模板

概率统计模板中的节点如表 9.1.6 所示

表 9.1.6 概率统计 VIs

名称	图标及端口	功能
General Histogram.vi		生成输入数组的直方图
Histogram.vi		生成输入数组的直方图
Mean.vi		计算输入数组的平均值

续表

名 称	图标及端口	功 能
Median.vi		计算输入数组的中值
Mode.vi		查找数组中出现次数最多的数据
Moment About Mean.vi		计算输入数组的 m 阶矩
MSE.vi		计算两个输入数组的均方误差
RMS.vi		计算输入数组的均方根 (root mean square)
Standard Deviation and Variance.vi		计算输入数组的均值、方差和标准方差
Statistics		Express VI 形式的统计节点, 可进行多种统计运算
Create Histogram		Express VI 形式的直方图节点
Probability Vis		概率字模板, 包含多种概率函数计算节点, 如正态分布、F 分布、t 分布等, 此处不再一一列出
Analysis of Variance Vis		方差分析子模板, 提供用于进行单因素、双因素和 3 因素方差分析的 VI, 此处不再一一列出

(2) 概率统计应用实例

例 9.1.5 绘出随机数的直方图。

直方图是考察数据分布规律的常用方法, 直方图表示了输入数据落在某区间内的个数。本例首先使用 Gaussian White Noise.vi 产生 10 000 个随机数, 然后使用 Histogram.vi 绘出随机数的直方图。

Gaussian White Noise.vi 的功能是产生高斯分布的随机数, Gaussian White Noise.vi 位于 Functions 模板 → All Functions 子模板 → Analyze 子模板 → Signal Processing 子模板 → Signal Generation 子模板中。

Histogram.vi 的图标如图 9.1.14 所示, 端口 intervals 设定划分的统计区间的个数, 区间长度由下式决定

$$\text{delta}_x = (\text{max} - \text{min}) / \text{intervals}.$$



图 9.1.14 Histogram.vi 的图标

VI 的前面板和框图程序如图 9.1.15 所示。

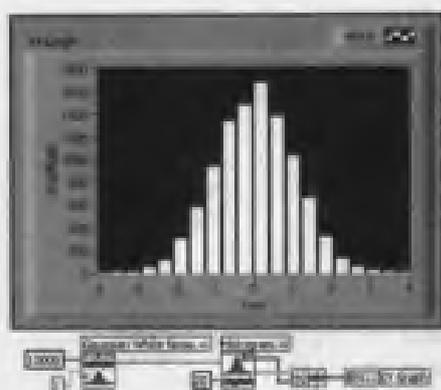


图 9.1.15 例 9.1.5 的前面板和框图程序

2. 曲线拟合

曲线拟合常用于考察对象之间隐藏的函数关系。LabVIEW 中曲线拟合节点的功能包括曲线拟合与插值。提供的曲线拟合方法有线性拟合、指数曲线拟合、多项式拟合、最小二乘拟合。提供的插值方法有多项式插值和样条插值。

(1) 曲线拟合与插值节点

曲线拟合节点位于 Functions 模板 → All Functions 子模板 → Analyze 子模板 → Mathematics 子模板 → Curving Fitting 子模板中，如图 9.1.16 所示。



图 9.1.16 Curve Fitting 模板

Curving Fitting 子模板中的节点如表 9.1.7 所示。

表 9.1.7 Curve Fitting VIs 说明

名称	图标与端口	功能
Linear Fit.vi		线性拟合
Linear Fit Coefficients.vi		线性拟合
Exponential Fit.vi		对数拟合
Exponential Fit Coefficients.vi		对数拟合
General Polynomial Fit.vi		多项式拟合
General LS Linear Fit.vi		最小二乘拟合
Nonlinear Lev-Mar Fit.vi		使用 Levenberg-Marquardt 算法进行非线性最小二乘拟合
Levenberg Marquardt.vi		使用 Levenberg-Marquardt 算法进行最小二乘拟合
Polynomial Interpolation.vi		多项式插值
Rational Interpolation.vi		分式插值
Spline Interpolant.vi		计算样条插值所需要的各插值节点的二阶导数, 供 Spline Interpolation.vi 使用

续表

名称	图标与端口	功能
Spline Interpolation.vi		样条插值, 与 Spline Interpolant.vi 联用
Curve Fitting		Express VI 形式的曲线拟合节点, 提供多种拟合算法

(2) 曲线拟合应用实例

例 9.1.6 多项式曲线拟合。

拟合所用的数据由 Eval $y=f(x)$.vi 产生, 所用公式为 $y=x^2-3x$, 并在其上叠加了一个随机数。数据拟合利用 General Polynomial Fit.vi 实现, General Polynomial Fit.vi 的图标和端口如图 9.1.17 所示。端口 polynomial order 设置拟合多项式的次数; 端口 Best Polynomial Fit 输出拟合的函数值。



图 9.1.17 General Polynomial Fit.vi 的图标和端口

程序的前面板和框图如图 9.1.18 所示。原始数据和拟合后的数据分别打包, 然后组成数组送入 XY Graph 显示。可以看出, 拟合后的数据削弱了噪声的影响, 能够更加清晰地反映函数关系。

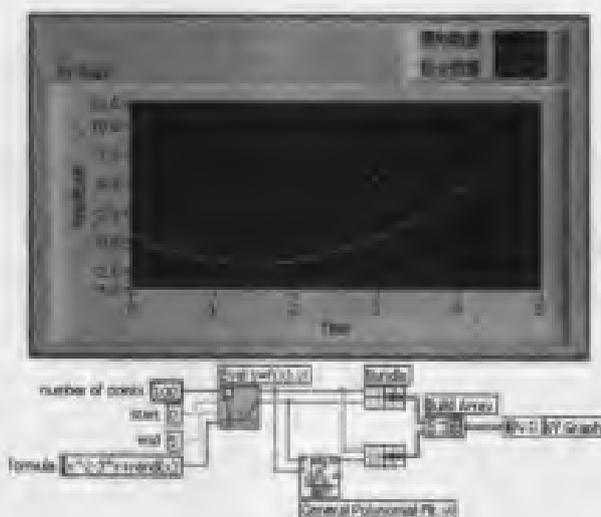


图 9.1.18 多项式拟合的前面板和框图程序

9.1.4 矩阵与数组运算

1. 矩阵运算

矩阵运算是工程数学的主要组成部分, 其运算量非常大, 而 LabVIEW 提供了矩阵运算的基本和高级运算节点, 为这类运算提供了便捷。

(1) 矩阵运算节点

矩阵运算节点位于 Functions 模板 → All Functions 子模板 → Analyze 子模板 → Mathematics 子模板 → Linear Algebra 子模板中, 如图 9.1.19 所示。



图 9.1.19 Linear Algebra 子模板

Linear Algebra 子模板中的节点如表 9.1.8 所示。

表 9.1.8 Linear Algebra 模板中的节点 s 列表

名称	图标	功能
Solve Linear Equation.vi		求解实线性方程 $AX=Y$
Inverse Matrix.vi		求解实矩阵的转置
Determinant.vi		求解实方阵的行列式
EigenValues and Vectors.vi		求解实方阵的特征值和特征向量
A x B.vi		求解两个实矩阵的积
A x Vector		求解实矩阵与实向量之积

续表

名称	图标	功能
Dot Product.vi		求解两个实向量的点积
Out Product.vi		求解两个实向量的外积
Advanced Linear Algebra		高级线性代数功能节点
LU Factorization.vi		求解实方阵的LU分解
QR Factorization.vi		求解实矩阵的QR分解
SVD Factorization.vi		求解实矩阵的奇异值分解 (singular value decomposition)
Cholesky Factorization.vi		求解实矩阵的Cholesky分解
Trace.vi		求解实矩阵的迹
Matrix Rank.vi		求解实方阵的秩
Matrix Norm.vi		求解实矩阵的范数
Matrix Condition Number.vi		求解实矩阵的条件数
Pseudoinverse Matrix.vi		求解实矩阵的伪逆矩阵
Create Special Matrix.vi		生成特殊矩阵
Test Positive Definite.vi		判断方阵是否正定
复数线性代数		提供与所有实数线性代数节点等同的功能, 只是操作数是复数, 故在此不再列出

(2) 矩阵运算实例

例 9.1.7 两个矩阵相乘。

矩阵相乘节点 A x B.vi 的图标及其连接端口如图 9.1.20 所示。



图 9.1.20 A x B.vi 的图标

A x B.vi 的端口说明见表 9.1.9。

表 9.1.9 A x B.vi 节点的端口

端口名称	说 明
A	输入矩阵
B	输入矩阵
A x B	输出矩阵
error	正误判断

例中的矩阵运算式为

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix} \times \begin{bmatrix} 2 & 3 & 4 \\ 5 & 6 & 7 \end{bmatrix} = \begin{bmatrix} 12 & 15 & 18 \\ 26 & 33 & 44 \\ 40 & 51 & 62 \end{bmatrix}$$

VI 的前面板和框图程序如图 9.1.21 所示。

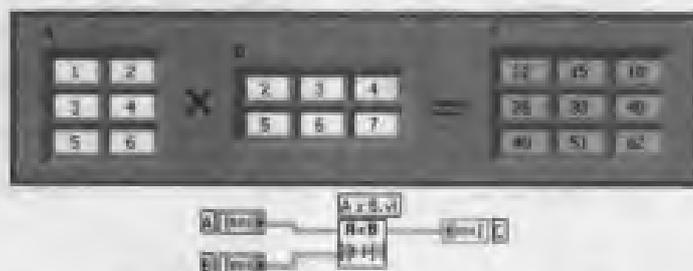


图 9.1.21 例 9.1.3 的前面板及框图程序

LabVIEW 中矩阵的表示实际上是用二维数组表示的,在进行乘法运算时一定要注意行列数目的匹配。

2. 数组运算

数组运算模板提供多种针对数组和矩阵的运算,包括数组的平移、单位化,数组和矩阵的标准化等。该模板中各 VI 的使用方法较为简单,故在此只给出说明,不再举例。

数组运算节点位于 Functions 模板 → All Functions 子模板 → Analyze 子模板 → Mathematics 子模板 → Array Operations 子模板,如图 9.1.22 所示。



图 9.1.22 Array Operations 子模板

Array Operations 模板中的节点如表 9.1.10 所示。

表 9.1.10 Array Operations 模板中的节点 a 列表

节点名	图标	功能说明
1D Linear Evaluation		数组的线性运算
1D Polynomial Evaluation		数组的多项式运算
2D Linear Evaluation		矩阵的线性运算
2D Polynomial Evaluation		矩阵的多项式运算
Quick Scale 1D		将数组在区间[-1,1]上归一化
Quick Scale 2D		将矩阵在区间[-1,1]上归一化
Scale 1D		对数组进行归一化运算
Scale 2D		对矩阵进行归一化运算

续表

节点名	图标	功能说明
Unit Vector		将数组单位化
Normalize Vector		将数组标准化
Normalize Matrix		将矩阵标准化
1D Polar To Rectangular		将极坐标转换为直角坐标
1D Rectangular To Polar		将直角坐标转换为极坐标

9.1.5 最优化与零点求解

1. 最优化

最优化是一门重要的管理学科，在工业生产和管理实践中有着广泛的应用，并衍生出了组合优化、线性规划、非线性规划、动态控制、最优控制等分支。

(1) 最优化节点

LabVIEW 中与最优化相关的节点位于 All Functions 模板 → Analyze 子模板 → Mathematics 子模板 → Optimization 子模板中，如图 9.1.23 所示。



图 9.1.23 Optimization 子模板

Optimization 子模板中的节点如表 9.1.11 所示。

表 9.1.11 Optimization 模板中的节点

名 称	图标及端口	功 能
Brent with Derivatives 1D.vi	 accuracy a (start) b (start) c (start) formula minimum f(minimum) ticks error	在给定区间上计算一元函数的局部极小值点
Chebyshev Approximation.vi	 number of points start end order formula C X Y error	用 Chebyshev 多项式逼近给定函数
Conjugate Gradient nD.vi	 accuracy gradient method line minimization Start X f(x) Minimum f(Minimum) ticks error	用共轭梯度法计算 n 元函数的局部极小值点
Downhill Simplex nD.vi	 accuracy Start X f(x) Minimum f(Minimum) ticks error	用 Downhill Simplex 计算 n 元函数的局部极小值点
Find All Minima 1D.vi	 accuracy step type algorithm start end formula Minima f(Minima) ticks error	计算一元函数在给定区间内的所有极小值点
Find All Minima nD.vi	 accuracy algorithm gradient method line minimum number of trials Start End X F(x) X Values F Values ticks error	计算 n 元函数在给定区间内的所有极值点
Fitting on a Sphere.vi	 X Y Z x0, y0, z0 r error	将所有的点拟合到 3 维空间的一个球面上
Golden Section 1D.vi	 accuracy a (start) b (start) c (start) formula minimum f(minimum) ticks error	用黄金分割法计算一元函数的极小值点
Linear Programming Simplex Method.vi	 C M B maximum X ticks error	解算线性规划方程
Pade Approximation.vi	 m n C[0, m+n] A[0, m] B[0, n] error	用有理分式逼近给定函数, 该有理分式与给定函数有 n+m 阶导数相同

(2) 最优化应用实例

例 9.1.8 生产中的最优化问题。

下面是一个线性最优化的简单例子，实际中的问题则可能复杂得多。

一个工厂生产两种产品 P 和 Q。每售出一件 P 可得利润 50 元，每售出一件可得利润 140 元。由于仓库容量有限，P 与 Q 的总数量不能超过 120 件。P 的成本为 15 元/件，Q 的成本为 30 元/件。工厂所能用来购买原料的资金最多为 1 500 元。每生产一件产品需要付给工人的报酬为，生产 P 报酬 15 元，生产 Q 报酬 45 元。需支付的工人报酬总和应小于 1 800 元。

为了解决这个问题，设 x_1 表示 P 产品的数量， x_2 表示 Q 产品的数量。根据题目给出的条件，可以列出以下约束方程

$$\begin{aligned} x_1 + x_2 &\leq 120 \\ 15x_1 + 30x_2 &\leq 1\,500 \\ 15x_1 + 45x_2 &\leq 1\,800 \\ x_1, x_2 &\geq 0 \end{aligned}$$

目标函数为

$$50x_1 + 140x_2 = \max!$$

这里使用了 Linear Programming Simplex Method.vi 节点，其端口如图 9.1.24 所示。Linear Programming Simplex Method.vi 节点要求约束方程写为“表达式 \geq 常数”的形式，如果约束方程是“表达式 \leq 常数”的形式，可以在不等式两端同乘 -1。

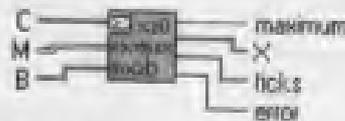


图 9.1.24 Linear Programming Simplex Method.vi 节点的端口

在输入端口中，向量 C 是目标函数的系数，矩阵 M 是约束方程的系数，向量 B 是约束方程右端的值。各参数的设定如图 9.1.25 所示。程序框图如图 9.1.26 所示。程序运行结果表明，当 P 产品的产量为 60 件，Q 产品的产量为 20 件时，可以获得最大利润 5 800 元。



图 9.1.25 例 9.1.8 的前面板



图 9.1.26 例 9.1.8 的框图程序

2. 零点求解

Zeros 模板提供的功能有一元函数、多项式函数的零点求解，非线性系统方程求解。

(1) 零点求解节点

Zeros 模板位于 Functions 模板 → Analyze 子模板 → Mathematics 子模板 → Zeroes 子模板，如图 9.1.27 所示。



图 9.1.27 Zeros 模板

Zeroes 子模板中的节点如表 9.1.12 所示。

表 9.1.12 Zeroes 子模板中的节点列表

节点名称	图标	功能
Find All Zeros of f(x).vi		求解已知函数在指定区间内的所有零点
Newton Raphson Zero Finder.vi		使用牛顿切线法求解已知函数(一元)在某个区间内的零点

续表

节点名称	图标	功能
Ridders Zero Finder.vi		使用 Ridders 法求解已知一元函数在某个区间内的零点
Nonlinear System Single Solution.vi		求解非线性系统方程组
Nonlinear System Solver.vi		求解非线性系统方程组
Complex Polynomial Roots.vi		在复数域求解多项式的根
Polynomial Real Zero Counter.vi		计算已知多项式在指定区间内根的个数

(2) 零点求解应用实例

例 9.1.9 计算 π 值。

函数 $\sin(x)$ 在区间 $[1,4]$ 上有一个零点, 即 $x = \pi$ 。本例使用 Newton Raphson Zero Finder.vi 计算函数的零点, 其图标如图 9.1.28 所示。

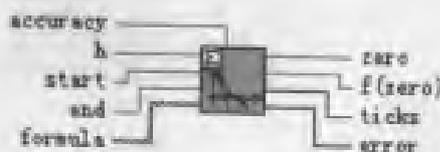


图 9.1.28 Newton Raphson Zero Finder.vi 的图标

VI 的前面板和框图程序如图 9.1.29 所示。

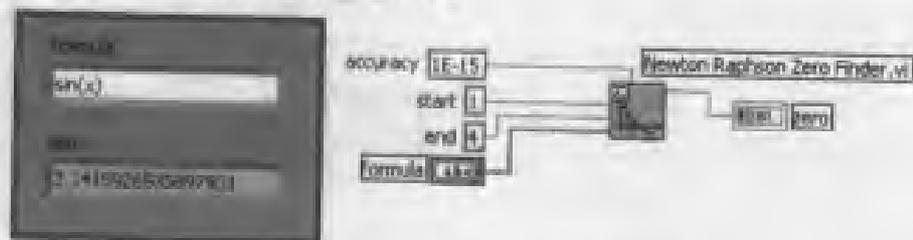


图 9.1.29 例 9.1.9 的前面板和框图程序

9.1.6 数值函数

数值函数函数模板提供了多种常用的数值函数, 如 Bessel 函数、Gamma 函数、Beta 函数等。这个模板中各 VI 的使用方法较为简单, 故只给出功能说明。

常用数值函数节点位于 Functions 模板 \rightarrow All Functions 子模板 \rightarrow Mathematics 子模板 \rightarrow Numeric Functions 子模板中, 如图 9.1.30 所示。



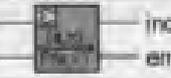
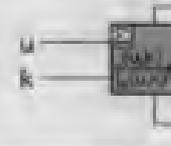
图 9.1.30 数值函数函数模板

Numeric Functions 子模板中的节点如表 9.1.13 所示。

表 9.1.13 Numeric Functions 子模板中的节点 a 列表

节点名称	图 标	功 能
Function $J_n(x)$.vi		对于实数 x , 计算 x 的 n 阶第一类 Bessel 函数的值
Bessel Function $Y_n(x)$.vi		对于实数 x , 计算 x 的 n 阶第二类 Bessel 函数的值
Bessel Polynomial.vi		对于实数 x , 计算 x 的 n 阶 Bessel 多项式的值
Beta Function.vi		计算 Beta 函数的值
Incomplete Beta Function.vi		对于给定的正实数 a, b 及 $x(0 < x < 1)$, 计算不完全 Beta 函数的值

续表

节点名称	图标	功能
Gamma Function.vi	 gamma(x)	对于给定的实数 x , 计算 Gamma 函数的值
Incomplete Gamma Function.vi	 incomplete gamma(a,x) error	对于给定的正实数 a 及实数 x , 计算不完全 Gamma 函数的值
Binomial Coefficient.vi	 binomial coefficient(n,k) error	给定非负整数 n , $k(k \leq n)$ 计算 Binomial 系数
Continued Fraction.vi	 result error	对于给定的数组 A 和 B , 计算由其确定的连分数的值
Jacobian Elliptic Function.vi	 sn(u,k) cn(u,k) sc(u,k) error dn(u,k)	给定实数 u 和 $k(0 \leq k \leq 1)$, 计算 Jacobian Elliptic 函数的值 (包括 sn, cn, sc, 和 dn)
Spike Function.vi	 spike(x)	给定实数 x , 计算 Spike 函数的值
Square Function.vi	 square(x)	给定实数 x , 计算 Square 函数的值
Step Function.vi	 step(x)	给定实数 x , 计算 Step 函数的值
Sine Integral.vi	 si(x)	给定实数 x , 计算其 sine 积分
Cosine Integral.vi	 ci(x) error	给定非负实数 x , 计算其 cosine 积分
Legendre Elliptic Integral 1st Kind.vi	 F(phi,k) error	给定实数 ϕ 和 $k(0 \leq k < 1)$, 计算 Legendre 椭圆积分

9.2 数字信号处理

LabVIEW 的数字信号处理模板的功能包括五部分内容:

- 信号产生;
- 时域分析;
- 频域分析;
- 滤波器;
- 窗函数。

数字信号处理模板位于 Functions 模板 → All Functions 子模板 → Analyze 子模板 →

Signal Processing 子模板, 如图 9.2.1 所示。



图 9.2.1 数字信号处理模板

1. 信号发生

信号发生模板提供了常用的信号发生器。信号发生模板位于 Functions 模板 → All Functions 子模板 → Analyze 子模板 → Signal Processing 子模板 → Signal Generation 子模板, 如图 9.2.2 所示。



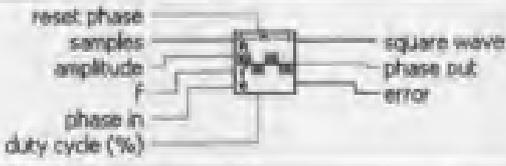
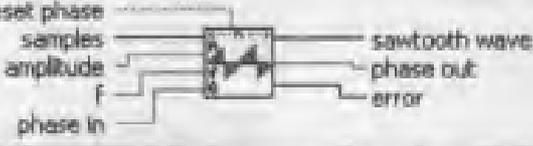
图 9.2.2 信号发生模板

时域分析模板中的节点如表 9.2.1 所示。

表 9.2.1 时域分析模板中的节点

名称	图标及端口	功 能
Signal Generator by Duration.vi		信号发生器，可产生 6 种基本波形
Tones and Noise.vi		产生的信号包含多个频率的正弦波、噪声和直流偏移
Sinc Pattern.vi		产生一个 Sine Pattern 信号
Impulse Pattern.vi		产生一个脉冲信号
Ramp Pattern.vi		产生一个斜坡信号
Sinc Pattern.vi		产生一个 Sinc 信号
Pulse Pattern.vi		产生一个冲击信号
Chirp Pattern.vi		产生一个线性调频信号
Sine Wave.vi		产生一个 Sine Wave 信号。此信号与 Sine Pattern 信号均为正弦信号，只是定义方式不同
Triangle Wave.vi		产生一个三角波信号

续表

名称	图标及端口	功能
Square Wave.vi		产生一个方波信号
Sawtooth Wave.vi		产生一个锯齿波信号
Uniform White Noise.vi		产生一个白噪声信号
Gaussian Noise.vi		产生一个高斯噪声信号
Periodic Random Noise.vi		产生一个周期随机噪声
Binary M.S.vi		产生一个二进制的最大序列
Gamma Noise.vi		Gamma 噪声发生器
Poisson Noise.vi		Poisson 噪声发生器
Binomial Noise.vi		二项式噪声发生器
Bernoulli Noise.vi		贝努利噪声发生器

例 9.2.1 产生 Sinc 信号。

Sinc 信号的生成公式为 $\frac{\sin(x)}{x}$ ，Sinc Pattern.vi 的图标如图 9.2.3 所示，端口说明见表 9.2.2。

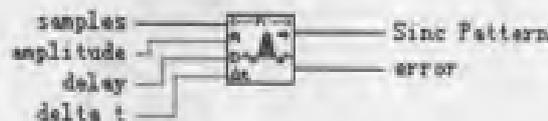


图 9.2.3 Sinc Pattern.vi 的图标

表 9.2.2 Sinc Pattern. VI 端口

端口名称	说明
samples	信号点数
amplitude	振幅
delay	延迟
delta t	相邻两点的事件间隔
Sinc Pattern	生成的信号数据
error	错误代码

程序的前面板和程序框图如图 9.2.4 所示。注意横轴的坐标实际上是信号点的索引，而不是时间。

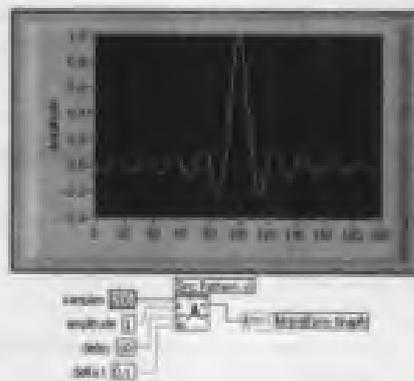


图 9.2.4 例 9.2.1 的前面板和框图程序

2. 时域分析

时域分析模板提供了卷积、相关计算、移位运算、积分、微分、脉冲测量等功能。时域分析模板位于 Functions 模板 → All Functions 子模板 → Analyze 子模板 → Signal Processing 子模板 → Time Domain 子模板，如图 9.2.5 所示。



图 9.2.5 时域分析模板

Time Domain 子模板中的节点如表 9.2.3 所示。

表 9.2.3 时域分析模板中的节点

名称	图标及端口	功能
AC & DC Estimator.vi	 <p>Signal [V] — AC estimate (Vrms) — DC estimate [V]</p>	测量信号的直流成分和交流成分
AutoCorrelation.vi	 <p>X — R_{xx} — error</p>	求信号的自相关函数
Convolution.vi	 <p>X — X * Y Y — error</p>	求两个信号的卷积
CrossCorrelation.vi	 <p>X — R_{xy} Y — error</p>	求两个信号的互相关函数
Decimate.vi	 <p>X — Decimated Array decimating factor — error averaging</p>	抽取信号数据
Deconvolution.vi	 <p>X * Y — X Y — error</p>	逆卷积运算
Derivative s(t).vi	 <p>X — ds/dt initial condition — error final condition dt</p>	数值微分
Integral s(t).vi	 <p>X — Integral X initial condition — error final condition dt</p>	数值积分
Peak Detector.vi	 <p>X — # found Locations Amplitudes 2nd Derivatives error threshold width peaks/valleys initialize [T] end of data [T]</p>	查找波峰或波谷的位置、幅度和二阶微分
Pulse Parameters.vi	 <p>X — slow rate overshoot rsetime lop amplitude base undershoot error falltime duration delay</p>	脉冲参数测量
Threshold Peak Detector.vi	 <p>X — Indices threshold — count width — error</p>	计算超过门限值的尖峰个数
Unwrap Phase.vi	 <p>Phase — Unwrapped Phase — error</p>	减小相位的不连续性

续表

名称	图标及端口	功能
$Y[i]=\text{Clip}(X[i], n)$	 Input Array upper limit lower limit Clipped Array error	截取信号
$Y[i]=X[i-n]$	 Input Array shifts: n Shifted Array error	移位运算
Zero Padder.vi	 Input Array Zero Padded Array	补零运算, 在数组末尾补零, 使数组长度等于 n , n 为整数

例 9.2.2 尖峰捕获。

时域分析模板中的尖峰捕获功能可以捕捉到信号中振幅超过某一门限值的尖峰, 将此功能进行修改, 可以实现信号的毛刺检测。Peak Detector.vi 的图标如图 9.2.6 所示, 端口说明见表 9.2.4。

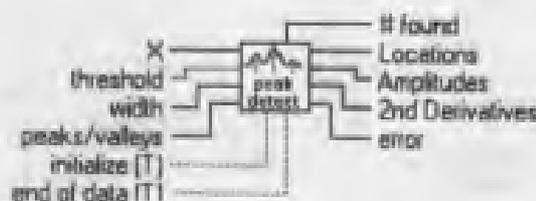


图 9.2.6 Peak Detector.vi 的图标

表 9.2.4 Peak Detector.vi 的端口说明

端口名	说明
X	输入信号
threshold	门限
peaks/valleys	设置检测尖峰还是低谷
initialize (T)	初始化
end of data (T)	如果要分析多块数据, 设置此端口为 False
#found	检测到的尖峰(低谷)数
Locations	尖峰(低谷)的位置
Amplitudes	尖峰(低谷)位置的振幅
2nd Derivatives	尖峰(低谷)处的二阶导数
error	错误代码

VI 的前面板和程序框图如图 9.2.7 所示。

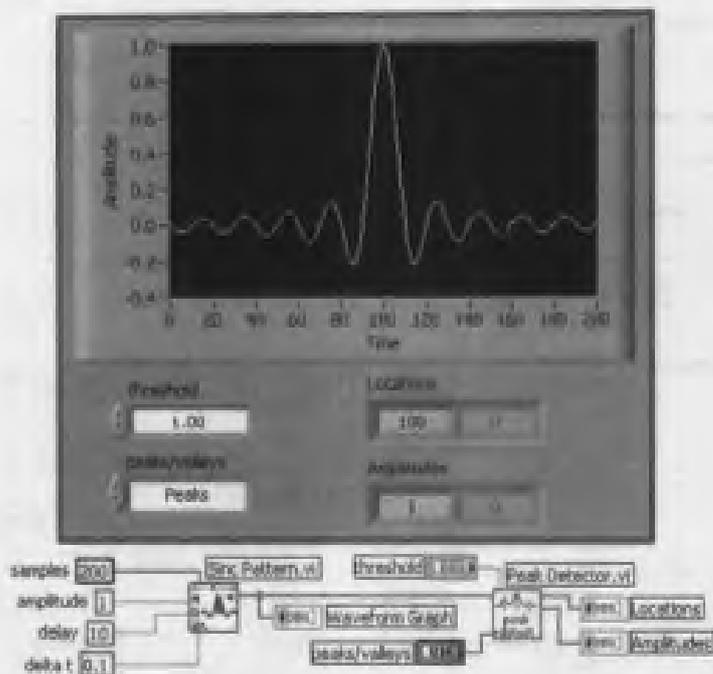


图 9.2.7 例 9.2.2 的前面板和程序框图

3. 频域分析

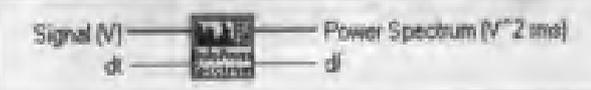
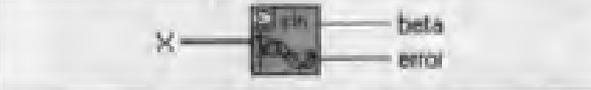
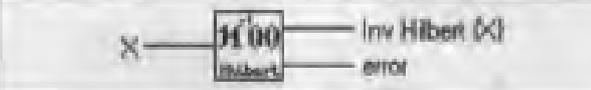
频域分析模板提供了丰富的信号频域分析函数，包括傅里叶变换、Hilbert 变换、小波变换、Hartley 变换、功率谱分析、联合时频分析、谐波分析、系统辨识等。频域分析模板位于 Functions 模板→All Functions 子模板→Analyze 子模板→Signal Processing 子模板→Frequency Domain 子模板，如图 9.2.8 所示。



图 9.2.8 频域分析模板

Frequency Domain 子模板中的节点如表 9.2.5 所示。

表 9.2.5 频域分析模板中的节点

名 称	图标及端口	功 能
Amplitude and Phase Spectrum.vi		计算时域信号的单边幅度谱和相位谱
Auto Power Spectrum.vi		计算时域信号的单边自功率谱
Burman Frequency Estimator.vi		估计未知长度 sine 信号的频率
Complex FFT.vi		复数快速傅里叶变换
Cross Power.vi		计算输入信号的互功率谱
Cross Power Spectrum.vi		计算输入信号的单边互功率谱
Fast Hilbert Transform.vi		快速 Hilbert 变换
FHT.vi		快速 Hartley 变换
Harmonic Analyzer.vi		谐波分析
Inverse Complex FFT.vi		复数快速傅里叶反变换
Inverse Fast Hilbert Transform.vi		快速 Hilbert 反变换
Inverse FHT.vi		快速 Hartley 反变换
Inverse Real FFT.vi		实数快速傅里叶反变换
Laplace Transform Real.vi		实数 laplace 变换
Network Functions (avg).vi		计算一个系统的单边平均频率响应、冲击响应、系统输入输出的单边平均互功率谱、相关函数

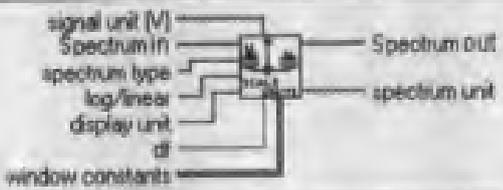
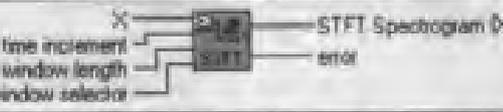
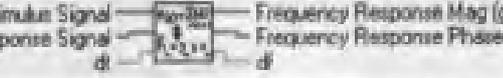
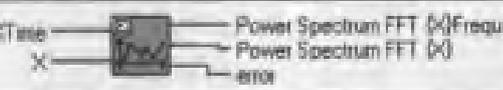
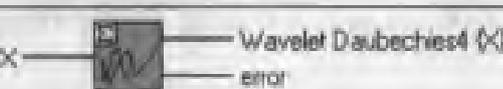
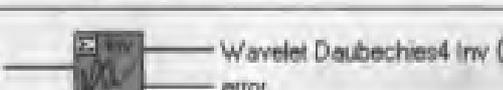
续表

名称	图标及端口	功能
Power & Frequency Estimate.vi	 <p>est frequency peak est power peak</p> <p>Power Spectrum (V^2 ms) peak frequency [max] window constants d span</p>	求功率谱的峰值和对应的频率
Power Spectrum.vi	 <p>Power Spectrum error</p> <p>X</p>	计算输入序列的双边功率谱
Real FFT.vi	 <p>FFT (X) error</p> <p>X</p>	实数快速傅里叶变换
Spectrum Unit Conversion.vi	 <p>Spectrum Out spectrum unit</p> <p>signal unit (V) Spectrum In spectrum type log/linear display unit d window constants</p>	谱单位转换
STFT Spectrogram.vi	 <p>STFT Spectrogram (X) error</p> <p>X time increment window length window selector</p>	使用短时傅里叶变换计算信号能量的联合时频分布
Transfer Function.vi	 <p>Frequency Response Mag (gain) Frequency Response Phase (radians)</p> <p>Stimulus Signal Response Signal d</p>	计算一个系统的频率响应
Unevenly Sampled Signal Spectrum.vi	 <p>Power Spectrum FFT (X) (Frequency) Power Spectrum FFT (X) error</p> <p>X XTime</p>	计算非等距采样信号的功率谱
Walsh Hadamard.vi	 <p>Walsh Hadamard (X) error</p> <p>X</p>	实数 Walsh Hadamard 变换
Walsh Hadamard Inverse.vi	 <p>Walsh Hadamard Inverse (X) error</p> <p>X</p>	实数 Walsh Hadamard 反变换
Wavelet Transform Daubechies4.vi	 <p>Wavelet Daubechies4 (X) error</p> <p>X</p>	小波变换
Wavelet Transform Daubechies4 Inverse.vi	 <p>Wavelet Daubechies4 Inv (X) error</p> <p>X</p>	逆小波变换
WVD Spectrogram.vi	 <p>WVD Spectrogram (X) error</p> <p>X line increment</p>	使用 Wigner-Ville 算法计算信号能量的联合时频分布
Amplitude and Phase Spectrum.vi	 <p>Amp Spectrum Mag (Vrms) Amp Spectrum Phase (radians)</p> <p>Signal (V) unwrap phase (T) d</p>	计算时域信号的单边幅度谱和相位谱
Auto Power Spectrum.vi	 <p>Power Spectrum (V^2 ms) d</p> <p>Signal (V) d</p>	计算时域信号的单边自功率谱
Burman Frequency Estimator.vi	 <p>beta error</p> <p>X</p>	估计未知长度 $\sin x$ 信号的频率

续表

名称	图标及端口	功能
Complex FFT.vi		复数快速傅里叶变换
Cross Power.vi		计算输入信号的互功率谱
Cross Power Spectrum.vi		计算输入信号的单边互功率谱
Fast Hilbert Transform.vi		快速 Hilbert 变换
FHT.vi		快速 Hartley 变换
Harmonic Analyzer.vi		谐波分析
Inverse Complex FFT.vi		复数快速傅里叶反变换
Inverse Fast Hilbert Transform.vi		快速 Hilbert 反变换
Inverse FHT.vi		快速 Hartley 反变换
Inverse Real FFT.vi		实数快速傅里叶反变换
Laplace Transform Real.vi		实数 Laplace 变换
Network Functions (avg).vi		计算一个系统的单边平均频率响应, 并输出响应, 系统输入输出的单边平均互功率谱, 相关函数
Power & Frequency Estimate.vi		求功率谱的峰值功率和对应的频率
Power Spectrum.vi		计算输入序列的双边功率谱
Real FFT.vi		实数快速傅里叶变换

续表

名称	图标及端口	功能
Spectrum Unit Conversion.vi		谱单位转换
STFT Spectrogram.vi		使用短时傅里叶变换计算信号能量的联合时频分布
Transfer Function.vi		计算一个系统的频率响应
Unevenly Sampled Signal Spectrum.vi		计算非等距采样信号的功率谱
Walsh Hadamard.vi		实数 Walsh Hadamard 变换
Walsh Hadamard Inverse.vi		实数 Walsh Hadamard 反变换
Wavelet Transform Daubechies4.vi		小波变换
Wavelet Transform Daubechies4 Inverse.vi		逆小波变换
WVD Spectrogram.vi		使用 Wigner-Ville 算法计算信号能量的联合时频分布

例 9.2.3 计算信号的快速傅里叶变换。

傅里叶变换是数字信号处理中最重要的一个变换之一，傅里叶变换的意义在于人们从此能够在频域中观察一个信号的特征了。

用于分析的信号为 $\sin 100x$ ，所用 Real FFT.vi 的图标如图 9.2.9 所示。注意 Real FFT.vi 的输入为实数数组，而输出结果为复数数组。



图 9.2.9 Real FFT.vi 的图标

VI 的前面板和框图程序如图 9.2.10 所示。从前面板可以看出，在频率 100 Hz 处有一亮线，表明信号拥有一个 100 Hz 的频率分量。

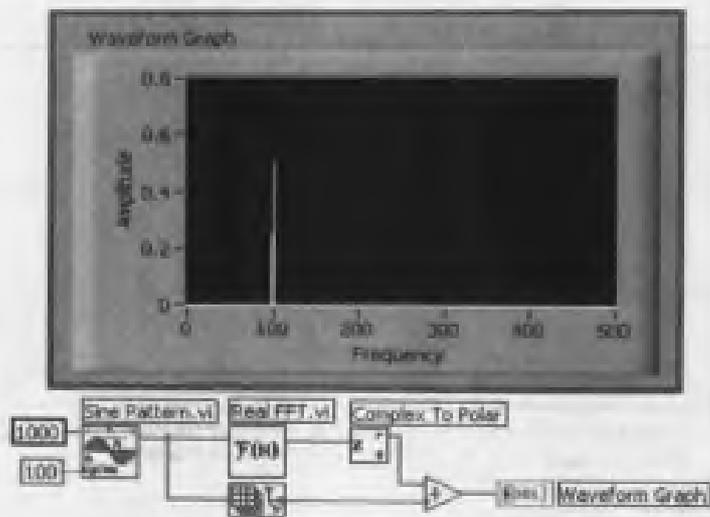


图 9.2.10 例 9.2.3 的前面板和框图程序

4. 数字滤波器

数字滤波器模板提供了多种常用的滤波器，并且提供了设计 FIR 和 IIR 滤波器的 VI。使用起来非常方便，只需输入相应指标参数即可。滤波器模板位于 Function 模板 → All Functions 子模板 → Analyze 子模板 → Signal Processing 子模板 → filters 子模板，如图 9.2.11 所示。



图 9.2.11 数字滤波器模板

filters 子模板中的节点如表 9.2.6 所示。

表 9.2.6 数字滤波器模板中的节点

名称	图标	功能
Butterworth Filter.vi		巴特沃斯滤波器
Chebyshev Filter.vi		契比雪夫滤波器
Inverse Chebyshev Filter.vi		契比雪夫 II 型滤波器
Elliptic Filter.vi		椭圆滤波器
Bessel Filter.vi		贝塞尔滤波器
Equi-Ripple LowPass.vi		等纹波低通滤波器
Equi-Ripple HighPass.vi		等纹波高通滤波器

续表

名称	图标	功能
Equi-Ripple BandPass.vi		等纹带通滤波器
Equi-Ripple BandStop.vi		等纹带阻滤波器
FIR Windowed Filter.vi		加窗 FIR 滤波器
Median Filter.vi		中值滤波器
高级 IIR 滤波器子模板		
Butterworth Coefficients.vi		计算用于设计基于 Butterworth 滤波器模型的 IIR 滤波器的系数
Cascade->Direct Coefficients.vi		将 IIR 滤波器的系数由级联形式转换为直接方式
Chebyshev Coefficients.vi		计算用于设计基于 Chebyshev 滤波器模型的 IIR 滤波器的系数
Bessel Coefficients.vi		计算用于设计基于 Bessel 滤波器模型的 IIR 滤波器的系数
Elliptic Coefficients.vi		计算用于设计基于 Elliptic 滤波器模型的 IIR 滤波器的系数

续表

名称	图标	功能
IIR Cascade Filter.vi		级联 IIR 滤波器
IIR Cascade Filter with I.C..vi		可设定初始状态的级联 IIR 滤波器
IIR Filter.vi		直接型 IIR 滤波器
IIR Filter with I.C..vi		可设定初始状态的直接型 IIR 滤波器
Inv Chebyshev Coefficients.vi		计算用于设计基于 Chebyshev II 型滤波器模型的 IIR 滤波器的系数
高级 FIR 滤波器子模板		
Convolution.vi		卷积
FIR Narrowband Coefficients.vi		计算用于设计内差窄带 FIR 滤波器的系数
FIR Narrowband Filter.vi		窄带 FIR 滤波器
FIR Windowed Coefficients.vi		计算用于设计加窗 FIR 滤波器的系数
Parks-McClellan.vi		计算具有线性相频响应的数字 FIR 滤波器的系数

例 9.2.4 低通滤波。

滤波是测试中常用的信号调理手段,高级的信号采集设备通常都集成了信号调理工具,通过滤波能够有效的提高信号的信噪比。

VI 的前面板和框图程序如图 9.2.12 所示。原始信号是一个叠加了高频噪声的正弦波。

产生高频噪声的方法是将白噪声通过一个巴特沃斯高通滤波器，截止频率为 100Hz。使用低通滤波器对原始信号滤波，滤掉高频噪声。

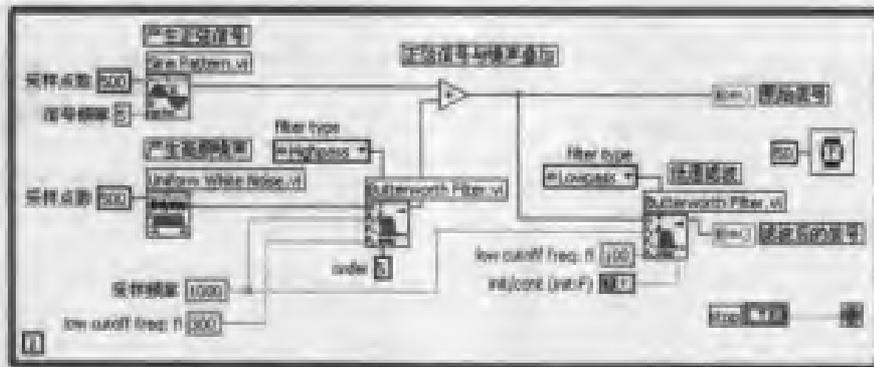
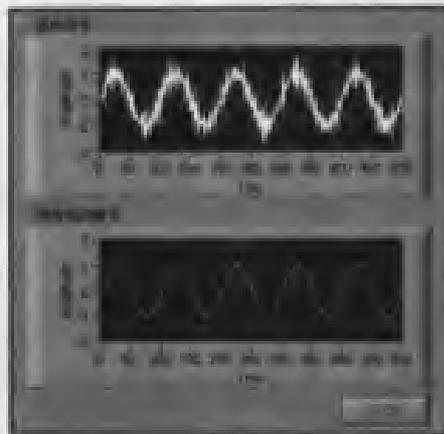


图 9.2.12 例 9.2.4 的前面板和框图程序

5. 窗函数

窗函数具有：截短信号、减少谱泄漏、分离频率相近的大幅值信号与小幅值信号的作用。

当使用 DFT 或 FFT 分析信号的频率成分时，算法将假设信号为周期信号，第一个周期即采样信号，整个信号则是由采样信号进行周期延拓而得，如图 9.2.13 所示。

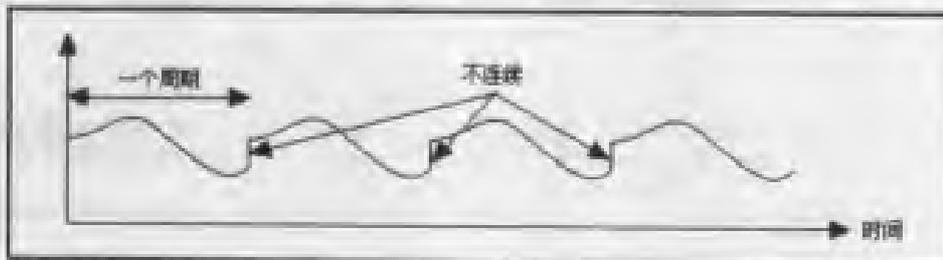


图 9.2.13 采样信号的周期延拓

可以看出周期与周期之间信号是不连续的，造成这种不连续的原因是实际中很难做到整周期采样。这将不可避免的引起谱泄漏，造成计算所得的频谱与实际信号的频谱不一

致。减小谱泄漏的一个简单方法是使用平滑窗。对采样信号加窗，可以减小截断信号的转折沿，从而减小谱泄漏。

窗函数模板提供了多种常用的窗函数，对一个数据序列加窗时，LabVIEW 认为此序列即是信号截断后的序列，因此窗函数的宽度等于数据列的长度。窗函数模板位于 Functions 模板 → All Functions 子模板 → Analyze 子模板 → Signal Processing 子模板 → Windows 子模板，如图 9.2.14 所示。



图 9.2.14 窗函数模板

Windows 子模板中的节点如表 9.2.7 所示。

表 9.2.7 窗函数模板中的节点

名 称	图 标	功 能
Scaled Time Domain Window.vi		归一化窗函数
Hanning Window.vi		汉宁窗
Hamming Window.vi		海明窗
Triangle Window.vi		三角窗
Blackman Window.vi		Blackman 窗
Blackman-Harris Window.vi		Blackman-Harris 窗

续表

名 称	图 标	功 能
Flat Top Window.vi		平顶窗
Kaiser-Bessel Window.vi		Kaiser-Bessel 窗
General Cosine Window.vi		General Cosine 窗
Cosine Tapered Window.vi		Cosine Tapered 窗
Force Window.vi		Force 窗
Exponential Window.vi		指数窗

9.3 波形测量

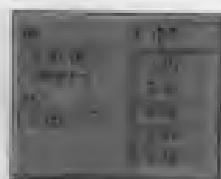


图 9.3.1 波形数据

波形数据是 LabVIEW 的一种数据类型，波形数据最大的特点是数据带有时间信息，每个数据点都唯一的对应一个时间戳。波形数据以簇的形式给出，如图 9.3.1 所示，包括起始时间 t_0 、采样时间间隔 dt 和一个由采样数据构成的数组。

由 DAQ 板卡采集的数据可以很容易地转换成波形数据，LabVIEW 提供的丰富的波形数据处理函数将大大减少编制数据分析程序的工作量。对于一般的数组，可以通过“Build Waveform.vi”将其转化为波形数据，Build Waveform.vi 的图标如图 9.3.2 所示。

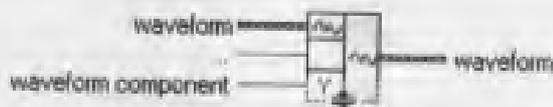


图 9.3.2. Build Waveform.vi

有关波形数据的详细内容请参考本书第 5.5 节。

9.3.1 波形测量节点

波形测量模板提供的功能包括直流交流成分分析、振幅测量、脉冲测量、傅里叶变换、功率谱计算、谐波畸变分析、过波分析、频率响应等。波形测量节点位于 Functions 模板 → All Functions 子模板 → Analyze 子模板 → Waveform Measurements 子模板中，如图 9.3.3 所示。



图 9.3.3 波形测量模板

Waveform Measurements 子模板中的节点如表 9.3.1 所示。

表 9.3.1 波形测量模板中的节点

名 称	图标及端口	功 能
Amplitude and Levels.vi	<p>signal in error in state settings amplitude high state level low state level error out</p>	返回波形的振幅、最大值、最小值
Averaged DC-RMS.vi	<p>reset signal in averaging type averaging time error in averaging control data ready DC value waveform RMS value waveform error out measurement info</p>	计算信号的直流分量大小和信号的均方根 (RMS)
Basic Averaged DC-RMS.vi	<p>reset signal in averaging type window error in DC value RMS value error out measurement info</p>	计算信号的直流分量大小和信号的均方根
Cross Spectrum (Mag-Phase).vi	<p>restart averaging (F) time signal X time signal Y window view error in averaging parameters averaging done magnitude phase averages completed error out</p>	计算信号的单边互功率谱密度
Cross Spectrum (Real-Im).vi	<p>restart averaging (F) time signal X time signal Y window error in averaging parameters averaging done real part imaginary part averages completed error out</p>	计算信号的单边互功率谱密度

续表

<p>Cycle Average and RMS.vi</p>		<p>计算信号的周期平均和周期均方根</p>
<p>Extract Single Tone Information.vi</p>		<p>提取振幅最大的谐波信号</p>
<p>FFT Power Spectral Density.vi</p>		<p>计算信号的功率谱密度</p>
<p>FFT Power Spectrum.vi</p>		<p>计算信号的功率谱密度</p>
<p>FFT Spectrum (Mag-Phase).vi</p>		<p>求傅里叶变换频谱, 计算结果表示为幅度和相位</p>
<p>FFT Spectrum (Real-Im).vi</p>		<p>求傅里叶变换频谱, 计算结果表示为实部和虚部</p>
<p>Frequency Response Function (Mag-Phase).vi</p>		<p>计算系统的频率响应, 计算结果表示为幅度和相位</p>
<p>Frequency Response Function (Real-Im).vi</p>		<p>计算系统的频率响应, 计算结果表示为实部和虚部</p>
<p>Harmonic Distortion Analyzer.vi</p>		<p>谐波畸变分析</p>

续表

Pulse Measurements.vi		脉冲测量
SINAD Analyzer.vi		有噪声信号的谐波畸变分析
Transition Measurements.vi		过度过程测量
Spectral Measurements		Express VI 形式的谱分析节点
Distortion Measurements		Express VI 形式的谐波畸变分析节点
Tone Measurements		搜索单频分量, Express VI
Timing and Transition Measurements		时间与过渡测量, Express VI
Amplitude and Level Measurements		阈值测量, Express VI

9.3.2 波形测量应用实例

例 9.3.1 直流分量与有效值测量。

本例将测量正弦信号的直流分量和有效值。使用的 VI 为 Basic Averaged DC-RMS.vi, 其图标如图 9.3.4 所示。输出端口 DC value 为测量所得的信号直流分量, 端口 RMS value 为信号的有效值。

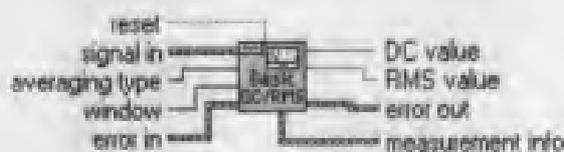


图 9.3.4 Basic Averaged DC-RMS.vi 的图标

VI 的前面板和程序框图如图 9.3.5 所示。信号源为振幅为 1 的正弦波。我们知道, 振

幅为 1 的正弦波的直流分量为零, 有效值为 $1/\sqrt{2} = 0.7071068$ 。从前面板可以看出, 测量所得的 DC 值为 $4.53E-16$, 这个值几乎为零, 有效值为 0.7071068 。

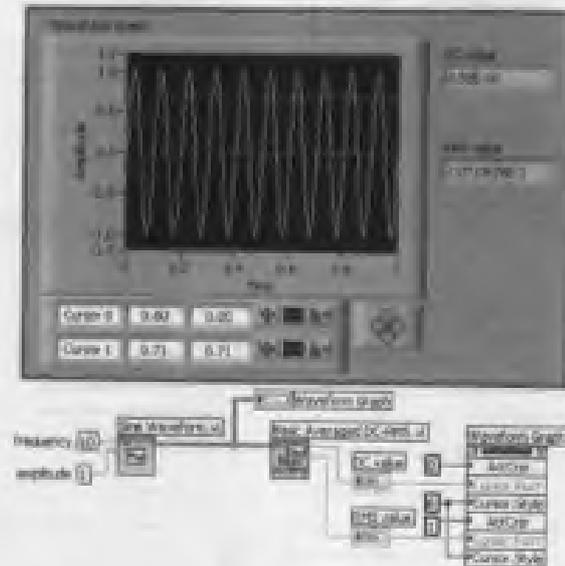


图 9.3.5 例 9.3.1 的前面板和框图程序

9.4 信号调理

信号调理是在信号分析前所做的必要处理, 其目的在于尽量减小干扰信号的影响, 提高信号的信噪比, 信号调理的好坏直接影响到分析效果。常用的信号调理方法有滤波、放大、加窗等。

9.4.1 信号调理节点

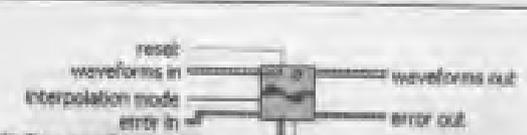
波形调理模板提供了 FIR 滤波器、IIR 滤波器和归一化窗函数。波形调理模板位于 Functions 模板 → All Functions 子模板 → Analyze 子模板 → Waveform Conditioning 子模板。



图 9.4.1 波形调理模板

Waveform Conditioning 子模板中的节点如表 9.4.1 所示。

表 9.4.1 波形调理模板中的节点

名称	图标和端口	功能
Digital FIR Filter.vi		数字 FIR 滤波器
Digital IIR Filter.vi		数字 IIR 滤波器
Scaled Window.vi		归一化窗函数, 加窗信号将被归一化, 所以在计算功率谱或幅度谱时所有的窗将提供
Align Waveforms (continuous).vi		波形对齐(连续方式), 输出信号为所有输入信号在公共时间段内的信号部分, 如果各输入信号在时间上没有重叠, 则输出信号为空
Align Waveforms(single shot).vi		波形对齐(单次方式)
Resample Waveforms (continuous).vi		再次采样(连续方式)
Filter		滤波器 Express VI
Align and Resample		波形对齐和再次采样, Express VI

9.4.2 信号调理应用实例

例 9.4.1 FIR 滤波器使用。

LabVIEW 提供的数字滤波器使用起来非常方便, 只需根据需要设定各参数即可。本例中使用 FIR 滤波器。在对相位信息有要求时, 通常使用 FIR 滤波器, 因为 FIR 滤波器的相频响应总是线性的, 可以防止时域数据发生畸变。Digital FIR Filter.vi 的图标和端口如图

9.4.2 所示, 该节点在进行滤波的同时, 还可以输出滤波器的幅频和相频响应。

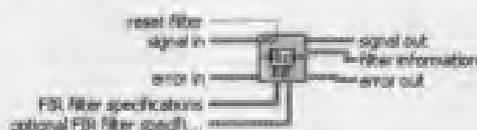


图 9.4.2 Digital FIR Filter.vi

VI 的前面板和框图程序如图 9.4.3 所示。子程序 WaveGenerator.vi 用以产生带有高频噪声的信号, 其框图程序如图 9.4.4 所示。FIR 滤波器的设置如表 9.4.2 所示。对于低通滤波器, Upper PB 和 Upper SB 参数不起作用。

表 9.4.2 FIR 滤波器参数设置

参 数	参数值	说 明
Topology	Equi-ripple FIR	设计滤波器的方法
Type	Low Pass	滤波器类型
Lower PB	10Hz	通带最高频率
Lower SB	20Hz	阻带最低频率

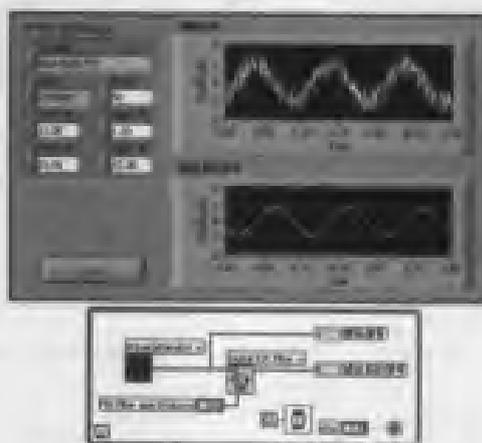


图 9.4.3 例 9.4.1 的前面板和框图程序



图 9.4.4 WaveGenerator.vi 的框图程序

9.5 波形监测

9.5.1 波形监测节点

波形监测模板提供的功能有波形边界检测、触发检测、尖峰捕获。波形监测节点位于 Functions 模板→All Functions 子模板→Analyze 子模板→Waveform Monitoring 子模板中，如图 9.5.1 所示。



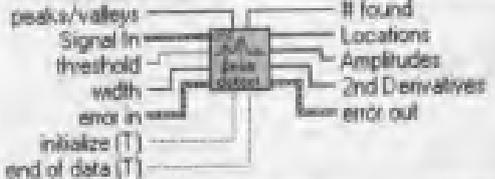
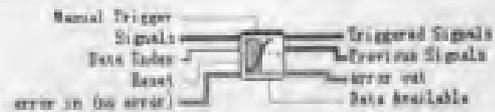
图 9.5.1 波形监测模板

Waveform Monitoring 子模板中的节点如表 9.5.1 所示。

表 9.5.1 波形监测模板中的节点

名称	图标和端口	功能
Basic Level Trigger Detection.vi		检测波形的触发点
Limit Specification.vi		创建检测边界，输出数据为 Limit Testing.vi 所用，两个 VI 共同完成边界测量
Limit Specification By Formula.vi		创建检测边界，输出数据为 Limit Testing.vi 所用，共同完成边界测量
Limit Testing.vi		边界测量，检测输入信号是否完全位于所定边界之内，与 Limit Specification By Formula.vi 或 Limit Specification.vi 联合使用

续表

<p>Waveform Peak Detection VI</p>		<p>峰值捕获, 可以检测波峰和波谷</p>
<p>Mask and Limit Testing</p>		<p>边界检测, Express VI</p>
<p>Trigger and Gate</p>		<p>触发测量, Express VI</p>

9.5.2 波形监测应用实例

例 9.5.1 边界检测。

边界检测用于检查信号是否位于某个范围之内。本例使用 Express VI “Mask and Limit Testing” 来完成边界检测。前面板和框图程序如图 9.5.2 所示。



图 9.5.2 边界检测的前面板和框图程序

双击 Mask and Limit Testing 节点, 打开属性设置对话框, 如图 9.5.3 所示, 单击 Upper Limit 一栏中的 Define... 按钮, 弹出边界定义对话框, 如图 9.5.4 所示。在 Data Point 一栏中输入 (0, 1) 和 (1, 1) 两个点, 即定义上边界为 $y=1$ 的直线, 使用相同的方法定义下边界为 $y=-1$ 的直线。

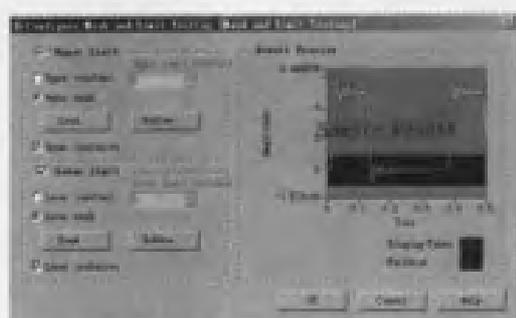


图 9.5.3 Mask and Limit Testing 节点的属性对话框

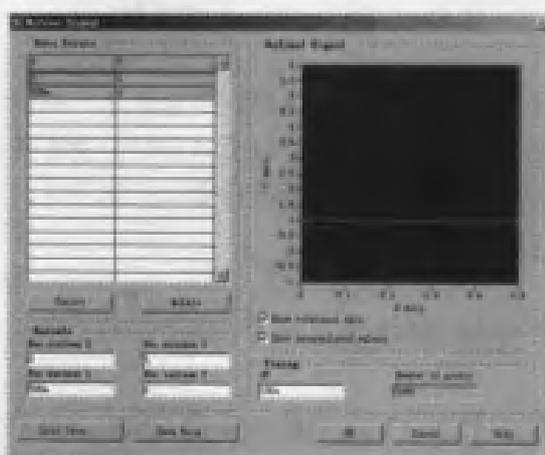


图 9.5.4 边界定义对话框

另一个 Express VI “Simulate Signal” 用以产生信号，双击它打开属性设置对话框，进行如图 9.5.5 所示的设置。

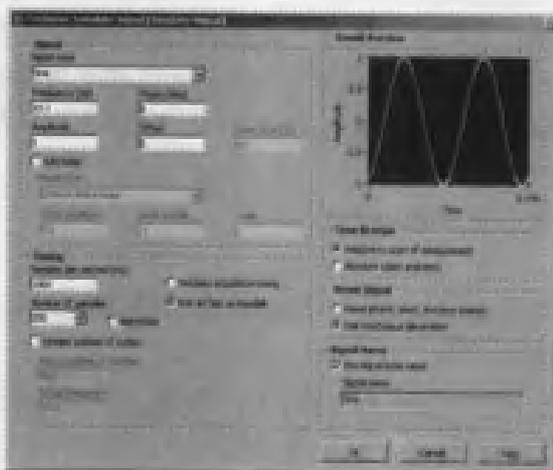


图 9.5.5 Simulate Signal 节点的属性对话框

运行程序、调整信号幅值，当信号完全位于两个边界围成的区域内时，Passed 指示灯为绿色，如图 9.5.2 前面板所示。当信号超出了这个界限后，Passed 指示灯变成红色。位于边界外的信号的采样点将以红色的方框标出，如图 9.5.6 所示。



图 9.5.6 信号超出边界时的图像

9.6 逐点信号分析

从 LabVIEW 6.1 开始, LabVIEW 的分析模板中出现了一类新的节点——逐点信号分析 (Point By Point Analysis) 节点。逐点信号分析是信号分析方法的一大革新。有了逐点信号分析, 信号分析终于可以逐点进行, 而无需设置数据缓冲区, 因此更适合编写实时的分析程序。

9.6.1 逐点信号分析的特点

传统的基于缓冲和数组的数据分析过程是: 缓冲区准备、数据分析、数据输出, 分析是按数据块进行的。由于构建数据块需要时间, 因此使用这种方法难以构建高速实时的系统。在逐点信号分析中, 数据分析是针对每个数据点的, 一个数据点接一个数据点连续进行的, 数据可以实现实时处理。使用逐点信号分析库能够跟踪和处理实时事件, 分析可以与信号同步, 直接与数据相连, 数据丢失的可能性更小, 编程更加容易, 而且因为无须构建数组, 所以对采样速率要求更低。

逐点信号分析具有非常广泛的应用前景。实时的数据采集和分析需要高效稳定的应用程序, 逐点信号分析是高效和稳定的, 因为它与数据采集和分析是紧密相连的, 因此它更适用于控制 FPGA (field programmable gate array) 芯片, DSP 芯片、内嵌控制器、专用 CPU 和 ASIC (特定用途集成电路)。

在使用逐点信号分析库时要注意以下两点:

(1) 初始化

逐点信号分析的程序必须进行初始化, 以防止前后设置发生冲突。

(2) 重入 (Re-Entrant)

逐点信号分析中 VI 必须被设置成为可重入的。可重入 VI 在每次被调用时将产生一个副本, 每个副本会使用不同的存储区, 所以使用相同 VI 的程序间不会发生冲突。

9.6.2 逐点信号分析节点

逐点信号分析节点位于 Functions 模板→All Functions 子模板→Analyze 子模板→Point By Point 子模板中, 如图 9.6.1 所示。逐点信号分析节点的功能与相应的标准节点相同, 只是工作方式有所差异, 故在此不再一一列出。



图 9.6.1 逐点信号分析模板的位置

9.6.3 逐点信号分析应用实例

例 9.6.1 基于逐点信号分析的滤波。

在本例中，信号源为正弦信号，随机信号作为噪声叠加在正弦信号上。使用两种方法进行滤波。在逐点信号分析中，VI 读取一个数据并分析它，然后输出一个结果，同时读入下一个数据，并重复以上过程，一点接一点连续、实时地进行分析。在基于数组的分析中，VI 必须等待数据缓冲准备好，然后读取一组数据，分析全部数据，产生全部数据的分析结果，因此分析是间断的、非实时的。VI 的前面板和框图程序如图 9.6.2 和图 9.6.3 所示。

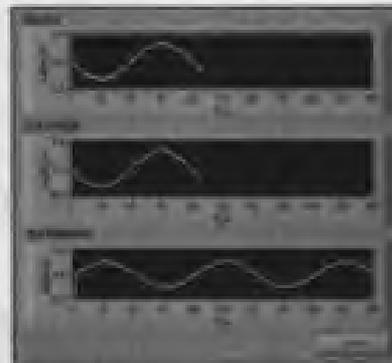


图 9.6.2 逐点信号分析和基于数组分析的滤波前面板窗口



图 9.6.3 逐点信号分析和基于数组分析的滤波框图程序

第 10 章 LabVIEW 程序设计

本章介绍关于创建与维护程序的一些信息及方法，其中包括人机交互界面的设计与修饰、文件的管理、定时与常用对话框的使用、应用程序的建立等方面的问题，最后介绍一些编程的经验和心得，希望能够帮助读者培养良好的编程风格。

10.1 人机交互界面

人机交互过程主要是对程序的控制和对程序执行结果的观察，因此人机交互界面的定制就包含了两个方面的内容：其一是程序运行时界面的显示模式，其二是程序对操作的响应方式。可控制的对象包括按钮、键盘输入前面板对象和选择框等。LabVIEW 前面板的最大特色是所见即所得，即在编程中排布的界面就是软件运行时的界面。设计人机交互界面时，主要从审美和方便实用的角度出发。

10.1.1 定制前面板对象

LabVIEW 提供了多种前面板对象，基本能够满足显示与控制的要求，但在特定的情况下，前面板对象可能需要进行一些修改，如将一个标尺的刻度进行换位，或者用一些合适的图片或文本取代前面板对象本身的图片或文本等，这一工作可以采用前面板对象编辑来完成。

基本方法是从 Controls 模板中选取一个与目标前面板对象相近的前面板对象，置于 VI 的前面板上，在前面板对象的右键弹出选单中选择 Advanced → Customize...，就进入了前面板对象编辑界面。如图 10.1.1 所示。



图 10.1.1 前面板对象编辑窗口

前面板对象编辑器的界面与前面板很相似，只是没有框图，并且每次只能编辑一个前面板对象。每一个前面板对象都是由很多部件构成的，如一个标尺，是由 Scale, Housing, Label, Digital Display, Slider, Unit Label, Increment 及 decrements 等 8 个部件组成，定制前面板对象就是按照用户的要求对这些部件进行修改、替换等操作。

在前面板对象编辑器中选取 Window→Show Parts Window 选单项，就会弹出一个浮动窗口，显示的是被编辑前面板对象的各个部件，如图 10.1.2 所示。在新窗口中，可以按照弹出式选单中的各项操作进行编辑。在编辑结束后，可以将这个前面板对象利用 Save 选单存为以“.ctl”为扩展名的外部文件。在以后的编辑过程中，若需要调用这一前面板对象，则可以从 Controls 模板中的“Select a VI...”中选择。



图 10.1.2 编辑前面板对象

10.1.2 选单的编辑与响应

人机交互界面中有一个重要的内容是用户选单，它是操作人员与程序对话的途径，在前面板和框图中都可以对它进行定义。

从主选单中选择 Edit→Run-time Menu...，就可以进入选单编辑对话框，用户选单只有 3 个类型：Default（默认）、Minimal（简单选单）和 Custom（自定义选单），默认选单是按照 LabVIEW 本身定义的选单给出的，Minimal 是对默认选单的简化，用户定制选单则是用户根据实际情况定义选单的过程。在选单模式的选择中，选取“Custom”项就可以编辑选单了，选单文件的保存是以.rtm 为扩展名的，最好将它与程序放在同一个路径下。

选单编辑对话框如图 10.1.3 所示，其中有系统选单、工具按钮、选单类型选项、选单项类型、选单项名称、选单项标签和快捷键定义框、选单编辑区和预览区。

下面简单介绍对话框中各项内容的含义。

- 工具按钮：此按钮共有 6 个，从左到右的功能分别是，在当前位置插入或添加一个选单项、删除当前选单项、使选中的选单项变成上一级选单、使选中的选单项成为下一级的选单、使选中的选单项和上一级交换位置、使选中的选单项和下一级交换位置。
- 选单类型：其中包括默认类型、简化类型和用户定制 3 种。前两种由 LabVIEW 内建，在创建自己的程序时，一般需定制程序的选单。
- 编辑区：主要是用来输入所需选单项，它提供的选单的树型视图。

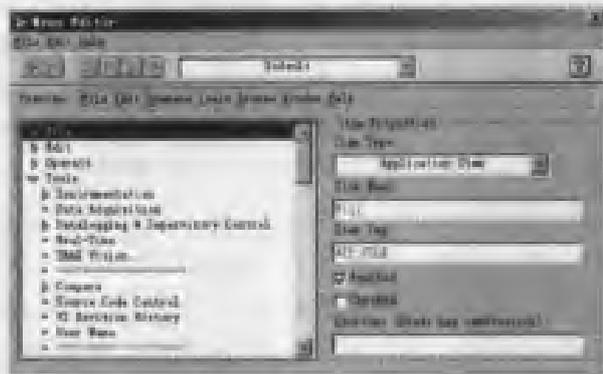


图 10.1.3 选单编辑对话框

- 预览区 (Preview): 用来实时显示生成选单的风格。
- 选单类型 (Item Type): 此选单共有 3 项。其中, User Item 是用户输入的名称, Separator 是用来分隔选单中的几个项的工具, Application Icon 是系统内建的一些选单, 它的响应方式由系统提供。
- 选单名称 (Item Name): 就是选单中的字符串。
- 选单标签 (Item Tag): 选单的唯一的标识符, 由用户自己定义。
- 快捷键: 输入一个字符后, 由它确认为 Ctrl 与之组合的快捷方式。

例 10.1.1 选单的响应。

选单的响应是程序的一个重要交互方式, LabVIEW 中的选单响应节点位于 Functions 模板→All Functions 子模板→Application Control 子模板→Menu 子模板中。

选单响应主要由两个节点配合一个 While 循环和 Case 框完成, 一个节点是 Current VI's Menubar 节点, 它指示出当前程序的选单, 产生相当于句柄的标识符; 另一个 Get Menu Selection 节点接受这一标识符, 接受用户对选单的操作, 并输出选单项的标识符 (Item Tag) 到 While 循环中的 Case 框, Case 框中的选择项要与选单项对应起来, 并且还必须要有一个空的默认项, 以保证在没有选择选单项时循环能够继续进行。

VI 的前面板和框图程序如图 10.1.4 所示。运行时每选中一项, 就显示一个单按钮对话框, 显示出刚才所选择的内容, 在每一个 Case 中, 必须有一个布尔量保证循环的继续进行。

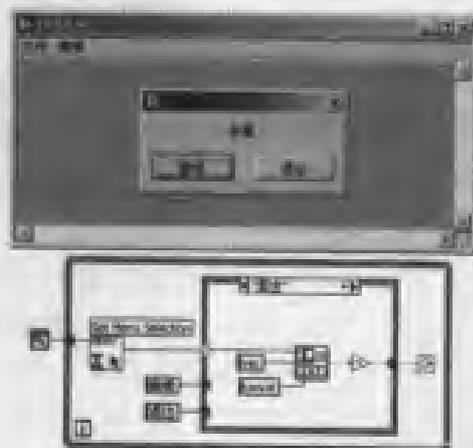


图 10.1.4 例 10.1.1 的前面板和框图程序

10.1.3 子面板的使用

子面板即 SubPanel，它是 LabVIEW 7 Express 新增的一项功能，通过 SubPanel，主 VI 在调用子 VI 时，可以将子 VI 的面板显示在主 VI 的面板中。这是一项非常有用的功能。例如，如果要设计一台多功能的信号分析仪，该信号分析仪可能包含了时域、频域、联合视频分析、小波分析等领域众多的分析功能，仪器使用统一的信号采集模块，而各功能的显示界面是不同的。有了子面板功能，我们可以将各分析模块做成插件，由主程序动态调用，分别设计分析模块的界面，在调用各分析模块的同时，显示各模块的界面。

SubPanel 位于 Controls 模板 → All Controls 子模板 → Containers 子模板中，如图 10.1.5 所示。



图 10.1.5 SubPanel 的位置

SubPanel 子模板如图 10.1.6 所示，在没有调用子 VI 前，SubPanel 是空白的。SubPanel 没有终端图标，当用户在前面板放置一个 SubPanel 时，框图程序中将同时出现 SubPanel 的 Insert VI 方法节点，如图 10.1.6 所示。

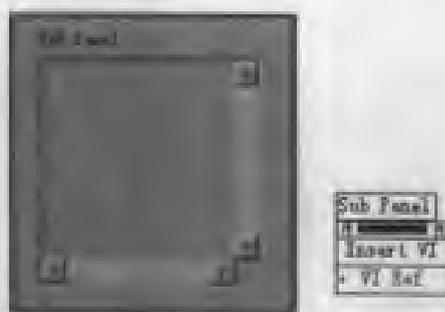


图 10.1.6 SubPanel 与 Insert VI 方法节点

SubPanel 的使用方法类似于 VI 的动态调用。按照动态调用 VI 的方法调用子 VI，将得到的 VI Ref 输入 SubPanel 的 Insert VI 方法节点。需要说明的是，子 VI 的面板只有在子 VI 被调用时才会显示在主 VI 的 SubPanel 中，子 VI 调用结束后，子 VI 的面板也会消失。并且，在使用 SubPanel 调用子 VI 时，子 VI 必须处于关闭状态，如果子 VI 已经打开，主程序会报错。

例 10.1.2 SubPanel 的使用。

在本例中，主 VI 将调用一个函数发生器 VI，并用 SubPanel 将其面板显示出来。主

VI 的前面板和框图程序如图 10.1.7 所示。在主 VI 调用子 VI 前, SubPanel 是空白的。主 VI 调用子 VI 后, 子 VI 的面板将显示在 SubPanel 中, 如图 10.1.8 所示。函数发生器 VI 的前面板和框图程序如图 10.1.9 所示。

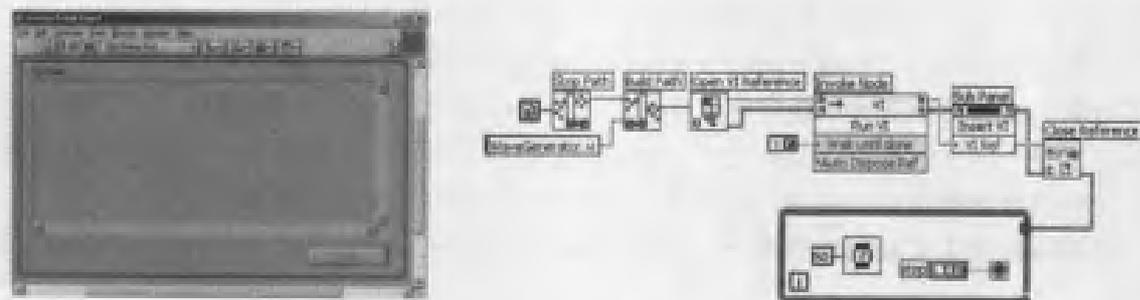


图 10.1.7 主 VI 的前面板和框图程序

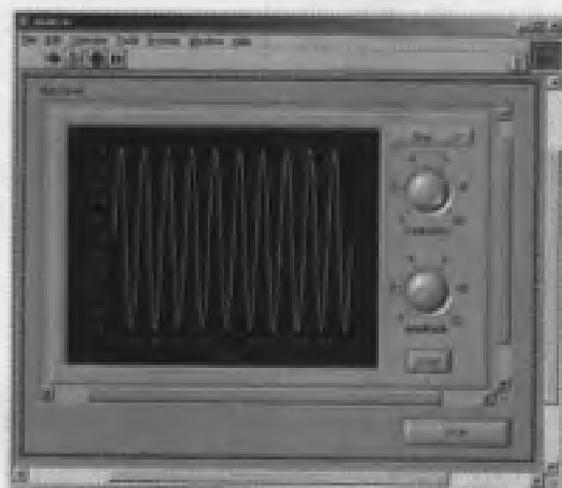


图 10.1.8 主 VI 运行时的界面

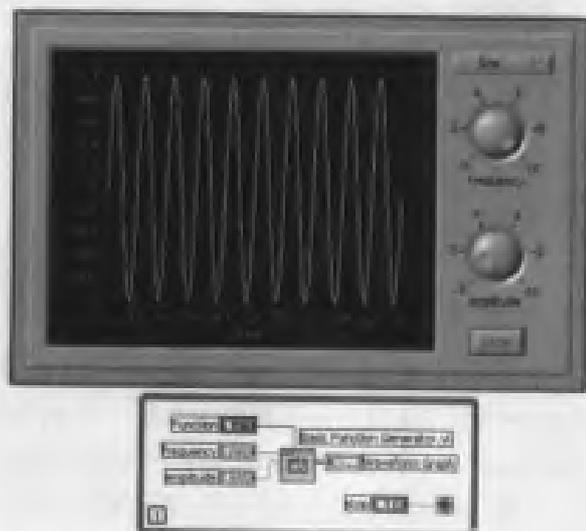


图 10.1.9 函数发生器 VI 的前面板和框图程序

10.1.4 界面装饰

界面装饰是编程之外的工作，主要是使程序变得整洁美观。完成这些功能的装饰没有程序代码，只在前面板出现，位于 Controls 模板 → All Controls 子模板 → Decorations 子模板中，如图 10.1.10 所示。

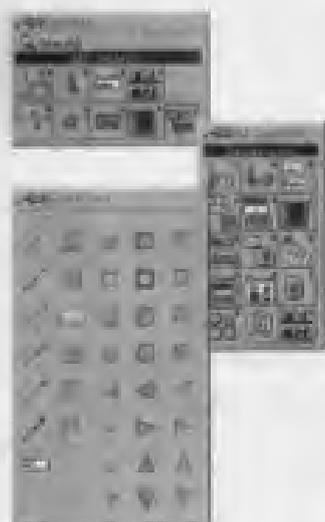


图 10.1.10 Decorations 子模板

装饰有四方形、圆形和三角形 3 种基本形状，以及多种箭头和直线，可以拖动它们的边缘适当地改变其形状。这些装饰有些具有覆盖功能，可以覆盖界面上不需要的前面板对象，有些则没有这一功能，只是提供一个边框。另外，在进行界面装饰时，还要结合工具条中提供的对齐、均匀分布及层次关系 3 种工具才能达到满意的效果。

对于装饰的使用没有定法，主要是编程者的主观感受。如图 10.1.11 所示，一个简单的文本在不同的装饰下，就会显示出截然不同的视觉效果。



图 10.1.11 不同的装饰下文本

除了 Controls 模板中提供的装饰，LabVIEW 还允许将外部图片文件插入前面板和框图程序，这一功能使得用户可以更灵活的装饰 VI。例如，可以将真实仪器的照片贴到前面板，并使用同样的方法修改前面板对象的外观，从而使 VI 的界面和真实仪器一模一样。

将图片插入 VI 前面板的一种方法是，从选单中选择 Edit → Import Picture From File...，然后从弹出的文件对话框中选择要导入的图片文件，单击确定，这时文件已经被复制到剪贴板中了。在前面板窗口中合适的位置单击鼠标，然后按“Ctrl+V”，图片就被贴在前面板中了，如图 10.1.12 所示。



图 10.1.12 在前面板插入图片

这种方法将把整个图片作为一个整体插入面板。还有另一种插入图片方法，即使用任何一种图片编辑工具打开图片，使用选择功能选中要插入的图片部分，将其复制到剪贴板，然后在 LabVIEW 中使用粘贴功能将图片粘到面板中。

LabVIEW 7 Express 支持常用的图形文件格式，如 JPEG、位图、GIF 等。JPEG 和位图文件可以在 LabVIEW 面板中进行缩放，但不是所有的文件格式都可以进行缩放。

10.2 定时与对话框

LabVIEW 提供了许多定时的节点，它们的作用方式都不尽相同，同时还提供了一些显示提示信息的对话框。这些节点并不处理任何信息，只是输出用户对信息的选择。这些功能节点位于 Functions 模板 → All Functions 子模板 → Time & Dialog 子模板中，如图 10.2.1 所示。该模板中还包括了处理错误信息的多个节点及用于检测用户前面板动作的节点。



图 10.2.1 Time & Dialog 子模板

10.2.1 定时器

定时器完成关于时间计算、换算和定时的功能，有 8 个节点，具体介绍见表 10.2.1。

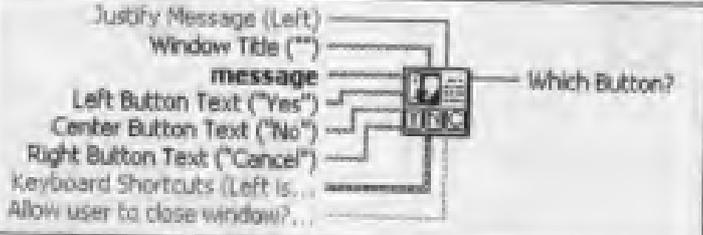
表 10.2.1 定时器节点表

节点名称	图 标	功 能
tick Count	 millisecond timer value	返回毫秒计数的值
Wait (ms)	milliseconds to wait 	使程序在此等待指定的毫秒数时间后再继续，并且返回这一毫秒值
Wait Until Next ms Multiple	millisecond multiple 	程序暂停指定的毫秒数，并且返回这一毫秒数，在循环中常用来减小循环速度
Format Date/Time String Function	time format string (in)  data/time string time stamp	将日期和时间用特定的格式表示出来
Get Date/Time String	data format (D)  data string time stamp  time string wait seconds? (F)	将指定的秒数转换成从 1904 年中午 12 时起的日期和时间串
Get Date /Time in seconds	 current time	返回从 1904 年 1 月 1 日中午 12 时到当前时间的绝对秒数
Date/Time to Seconds	date time rec  time	把输入的日期和时间转换成从 1904 年 1 月 1 日中午 12 时起算至现在时间的绝对值
Seconds to Date/Time	time stamp  data time rec	将从 1904 年 1 月 1 日中午 12 时到指定的秒数转化成日期和时间
Time Delay	Delay Time (s)  error out error in (in error)	产生一定时间的延迟 (Express VI)
Elapsed Time	 Get Start Time (s) Time Target (s) Reset Auto Reset error in (in error) Present (s) Present Test Elapsed Time (s) Time has Elapsed Elapsed Time Text error out Start Time Text Get Start Time (s)	计算已消耗的时间 (Express VI)

10.2.2 对话框

对话框是用来显示信息和提示有关操作的常用手段，LabVIEW 的对话框包括提示对话框和输入对话框两种。提示对话框用于向用户显示特定信息，提示对话框包含若干按钮，通常是要用户针对某种条件进行选择。输入对话框用于提示用户输入某类信息，如姓名、数字等。对话框节点如表 10.2.2 所示。

表 10.2.2 对话框节点

节点名称	图标	功能
One Button Dialog		显示带有一个按钮的对话框
Two Button Dialog		显示带有两个按钮的对话框
Three Button Dialog		显示带有三个按钮的对话框
Prompt User for Input		输入对话框, 用于提示用户输入信息
Display Message to User		向用户显示一条信息

1. 单按钮对话框

单按钮对话框 (One Button Dialog) 多用于确认提示。单按钮对话框的图标如图 10.2.2 所示。端口 message 设置对话框中显示的信息; button name (“OK”) 设置按钮的名称, 默认的按钮名为 “OK”。单按钮对话框的返回值为布尔量 True。



图 10.2.2 单按钮对话框

例 10.2.1 单按钮对话框的使用。

下面通过一个简单的例子说明单按钮对话框的使用。程序的前面板和程序框图如图 10.2.3 所示。程序运行后弹出一个对话框, 然后等待用户确认, 单击“确定”按钮后, 程序退出运行。

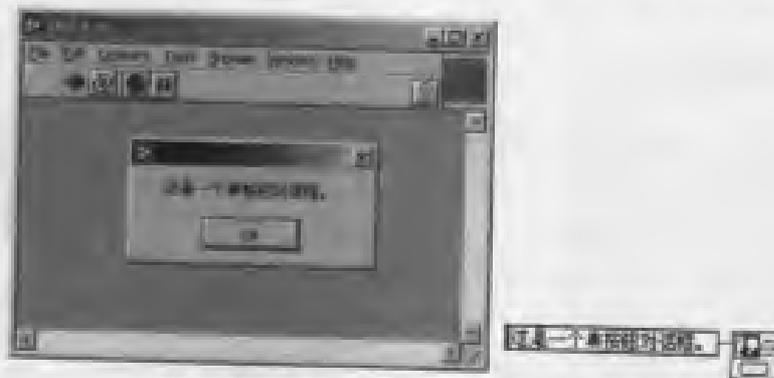


图 10.2.3 例 10.2.1 的前面板和框图程序

2. 双按钮对话框

双按钮对话框 (Two Button Dialog) 如图 10.2.4 所示, 其返回值有两个: True 和 False, 具体视用户所按按钮情况而定。按钮的名称默认值分别为“OK”和“Cancel”, 它的操作方法基本与单按钮对话框一样。



图 10.2.4 Two Button Dialog 节点的图标及其端口定义

3. 三按钮对话框

三按钮对话框 (Three Button Dialog) 的图标如图 10.2.5 所示。三按钮对话框拥有更多的设置选项。端口 Justify Message (Left) 设置所显示消息的对齐方式; 端口 Keyboard Shortcuts 设置各个按钮的快捷键; 端口 Allow user to close window? 设置用户是否可以通过对话框窗口右上角的关闭按钮关掉对话框。

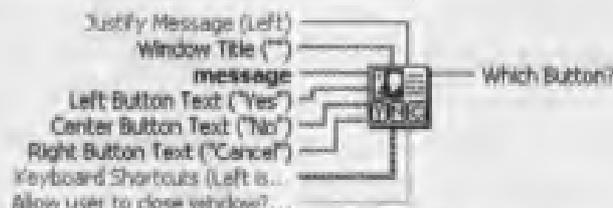


图 10.2.5 Three Button Dialog 节点的图标及其端口定义

4. 输入对话框

输入对话框 (Prompt User for Input) 是一个 Express VI, 它的图标和端口如图 10.2.6 所示。端口 Enable 用于禁止或允许该 Express VI, 输入对话框用来提示用户输入某类信息, 目前可输入的信息有三种: 字符串、数字、布尔量。输入对话框提供了一种简洁的与系统交流信息的方式, 例如可以使用输入对话框输入姓名、年龄等信息。



图 10.2.6 Prompt User for Input 节点的图标及其端口定义

例 10.2.2 输入对话框的使用。

下面的例子提示用户输入个人信息, 并将用户输入的信息在前面板显示出来。程序前面板和程序框图如图 10.2.7 所示。



图 10.2.7 例 10.2.2 的前面板和框图程序

输入对话框节点是 Express VI，它的图标会因属性设置的不同而有所差异。在使用输入对话框之前首先要设定其属性，在节点图标上单击鼠标右键，从快捷菜单中选择 Properties，弹出属性设置对话框 (Configure Prompt User for Display)，如图 10.2.8 所示。属性设置对话框包含四部分。

- 提示信息设置区 (Message to Display)：设置显示在对话框中的提示信息，程序运行时，提示信息将显示在对话框的上部。
- 输入设置区 (Inputs)：设置输入量的名称和数据类型，输入量最多可有 10 个，数据类型有 Number (数字)、Checkbox (布尔量)、Text Entry (字符串) 三种。
- 按钮设置区 (Buttons to Display)：设置显示在对话框中的按钮个数和按钮名称。
- 窗口标题区 (Window Title)：设置对话框的标题。



图 10.2.8 输入对话框的属性设置

设置输入对话框的属性。步骤如下：在提示信息设置区的文本框中输入提示信息“请填写个人信息”；输入设置如表 10.2.3 所示；设置第一个按钮的名称为确定，使 Display second button 复选框处于未被选择状态；窗口标题设为个人信息登记。设置完成后的属性

对话框如图 10.2.8 所示。

表 10.2.3 输入设置

Input Name	Input Data Type
姓名	Text Data Entry
工作单位	Text Data Entry
籍贯	Text Data Entry
年龄	Number
婚否	Checkbox

在运行程序时，首先弹出一个输入对话框，在对话框中填入各栏信息，如图 10.2.9 所示，单击确定按钮，退出对话框，这时从对话框读取的信息已经显示在前面板中了，如图 10.2.10 所示。



图 10.2.9 输入对话框的界面



图 10.2.10 输入的信息

10.2.3 错误处理节点

错误处理节点在程序正常的运行过程中是不起任何作用的，但是它对调试程序却非常有用。它能配合错误代码表准确地指出程序中的错误所在。所以在编程时，要尽可能地根据需要合理使用这些节点。

1. Find First Error 节点

Find First Error 节点用于检测具有错误代码输出的函数是否出错，如果出错，它将在“错误输出”簇中置状态位，并且可以通过 General Error Handler 节点或 Simple Error Handler 节点去判断和描述这一错误。节点图标及其端口定义如图 10.2.11 所示。

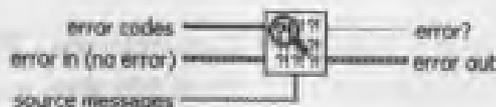


图 10.2.11 Find First Error.vi 节点的图标及其端口定义

2. General Error Handler 节点

General Error Handler 节点用于判断程序是否有错误，如果有，就会对错误进行描述，并且显示错误对话框。节点图标及其端口定义如图 10.2.12 所示。

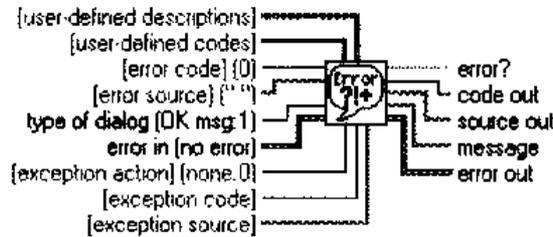


图 10.2.12 General Error Handler.vi 节点的图标及其端口定义

3. Simple Error Handler 节点

Simple Error Handler 节点与 General Error Handle 节点基本相似，只是少了一些选项。节点图标及其端口定义如图 10.2.13 所示。

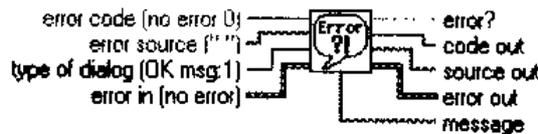


图 10.2.13 Simple Error Handler.vi 节点的图标及其端口定义

4. Clear Errors 节点

Clear Errors 节点用于清除错误状态。节点的图标及其端口定义如图 10.2.14 所示。



图 10.2.14 Clear Errors 节点的图标及其端口定义

10.3 LabVIEW 环境参数设置

在很多情况下，用户无须变更 LabVIEW 环境设置参数就能够很好的完成设计工作。尽管如此，LabVIEW 还是提供了环境设置工具。用户可以根据自身需要调整环境设置，以方便编程和更好地完成开发工作。

如要设置环境参数，可以从主菜单中选择 Tools→Options，弹出 Options 对话框，如图 10.3.1 所示。从最上方的下拉列表框中可以选择环境参数类别，LabVIEW 7 Express 的环境参数共有一下几类。

- New and Changed in 7.0: LabVIEW 7 Express 的新特性。
- Paths: 设置 LabVIEW 临时文件和库文件的路径，以及搜索 VI 时使用的路径次序。

- Performance and Disk: 设置资源分配以及 VI 的运行方式。
- Front Panel: 设置前面板对象的选项。
- Block Diagram: 设置框图程序对象的选项。
- Alignment Grid: 设置所有 VI 的网格选项。
- Controls/Functions Palettes: 设置 Controls 模板和 Functions 模板的选项。
- Debugging: 设置程序框图的调试选项。
- Colors: 设置 LabVIEW 使用的颜色。
- Fonts: 设置应用字体、对话框字体和系统字体。
- Printing: 设置打印信息。
- Revision History: 设置 History 窗口的行为, History 窗口通常通过选单 Tools → VI

Revision History 访问。

- Miscellaneous: 设置前面板和框图对象的其他选项。

VI Server 配置

- Configuration: 设置 the VI Server。
- TCP/IP Access: 设置允许访问本地 VI Server 的 TCP/IP 地址。
- Exported VIs: 设置允许被远程访问的 VI。

Web Server 配置

- Configuration: 启动并配置 Web Server。
- Browser Access: 设置访问者的权限, 设置权限有三种, 分别是浏览和控制、浏览、

禁止访问。

- Visible VIs: 指定通过网络进行发布的 VI。

下面对各项设置分别进行简要说明。

10.3.1 新特性

从 Options 对话框的下拉列表框选择 New and Changed in 7.0, 弹出新特性设置页面, 如图 10.3.1 所示。这一页面用于设置 LabVIEW 7 Express 新增加的特性。

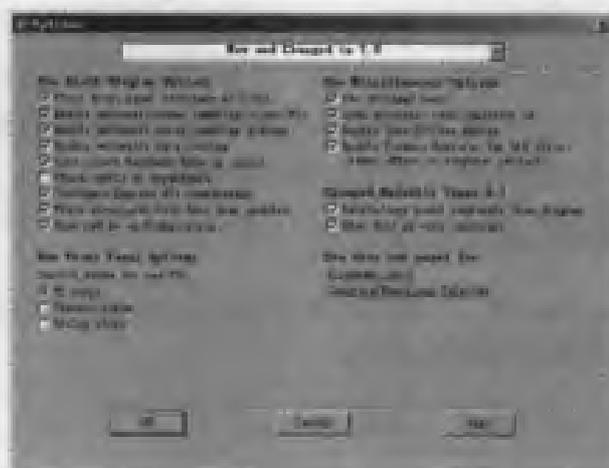


图 10.3.1 LabVIEW 7 Express 新特性设置页面

1. LabVIEW 7 Express 框图程序新特性

- ① Place front panel terminals as icons: 设置以图标形式还是数据类型形式显示前面板对象。数据类型形式是 LabVIEW 的传统显示方式, 图标形式是 LabVIEW 7 Express 新增的显示方式。这两种显示方式如图 10.3.2 所示。此外, 还可以通过在图标上单击鼠标右键, 通过在快捷选中单中选择 View As Icon 来选择显示方式。



图 10.3.2 前面板前面板对象图标的两种形式

- ② Enable automatic error handling in new VIs: 新建空白 VI 时, LabVIEW 将自动进行错误处理 (停止运行、高亮显示有错误的子 VI、弹出错误说明对话框)。取消该选择框将禁止自动错误处理功能。

- ③ Enable automatic error handling dialogs: 取消该选择框, 即使已经开启了自动错误处理功能, 在有错误发生时也不弹出错误对话框。

- ④ Enable automatic wire routing: 自动布线。如果选择了该项, 连线时 LabVIEW 将自动选择数据线的所经路线。

- ⑤ Auto insert Feedback Node in cycles: 自动添加反馈节点。如果选择了该项, 当用户将一个 VI 的输出端与输入端相连时, LabVIEW 将自动在数据线中放置反馈节点。

- ⑥ Place subVIs as expandable: 将插入框图程序的子 VI 以扩展形式显示。扩展形式与普通形式显示的图标如图 10.3.3 所示。该选项不影响 Express VI。



图 10.3.3 VI 的扩展形式和标准形式图标

- ⑦ Configure Express VIs immediately: 如果选中该选项, 在放置 Express VI 到框图程序中时, 将弹出一个 Express VI 设置对话框。

- ⑧ Place structures with Auto Grow enabled: 设置结构体的自动放大属性。如果选中了该项, 当结构体中的对象过于靠近结构体的边界时, LabVIEW 将自动放大结构体。

- ⑨ Show red Xs on broken wires: 在断的连线上放置“X”标志。

2. LabVIEW 7 Express 前面板新特性

Control style for new VIs: 通过右键快捷选中单 Create→Control 或 Create→Indicator 创建前面板对象时, 设置前面板对象的外观风格。

3. LabVIEW 7 Express 的其他新特性

- ① Use abridged menus: 在默认情况下, 只显示最常用的选单项。取消此选项则显示所有选单项。

② Lock automatic tool selection on: 如果选择了该项, 当用户按下 <Tab> 或 <Shift-Tab> 键时, 将工具自动选择功能开启并锁定。

③ Enable Just-In-Time Advice: 允许 Just-In-Time Advice 提示, 当用户设置了特定动作时将显示 Just-In-Time Advice 提示窗口。

④ Enable Windows Explorer for LLB files: 允许通过 Windows 浏览器查看 LabVIEW 库文件。用户可以通过浏览器查看、重命名、移动、复制、删除库文件中的 VI。

4. 在 LabVIEW 7 Express 中改变了默认值的选项

① Delete/copy panel terminals from diagram: 允许通过 Delete 键在程序框图中删除前面板对象。同时允许通过按下 Ctrl 键并拖动前面板对象的图标以复制前面板对象。

② Show dots at wire junctions: 在连线的结合处显示一个圆点。

10.3.2 路径与性能

1. 路径设置

从 Options 对话框的下拉列表框选择 Path, 切换到路径设置页面, 如图 10.3.4 所示。该页面用来设置 LabVIEW 临时文件和库文件的路径, 以及搜索 VI 时使用的路径次序。

① 路径类型: 为一下拉列表框, 位于 Use default 选择框的左侧, 用来选择要编辑的路径。

- Library Directory: LabVIEW 系统库文件路径。
- Temporary Directory: 临时文件的路径。
- Default Directory: 默认路径。
- Default Data Directory: 默认数据路径。
- Menus Directory: 默认选单路径, 该路径下存放有 Controls 和 Functions 模板的设置选单, 重新启动 LabVIEW 后生效。
- VI Search Path: LabVIEW 搜索 VI 的路径。



图 10.3.4 路径设置页面

- ② Use default: 取默认设置。
- ③ Path list: 位于 Path Chooser 下拉列表框下面, 显示当前编辑的路径种类的路径值。
- ④ Browse: 打开路径选择对话框。
- ⑤ Insert Before: 在当前路径前增加一个路径。
- ⑥ Insert After: 在当前路径后增加一个路径。
- ⑦ Replace: 替换所选路径。
- ⑧ Remove: 删除所选路径。

2. 性能与资源

从 Options 对话框的下拉列表框中选择 Performance and Disk, 切换到性能与资源设置页面, 如图 10.3.5 所示。该页面用来设置磁盘、内存资源分配及 VI 的运行方式。

① Check available disk space during launch: 在 LabVIEW 启动时报告临时目录下可用的磁盘空间。该选项在下次启动 LabVIEW 时生效。

- Minimum KB for Abort: 设置运行 LabVIEW 所需的最小磁盘空间, 如果小于这个空间, LabVIEW 退出运行。

- Minimum KB for Warning: 如果小于这个运行所需的最小磁盘空间, LabVIEW 将给出警告信息, 但用户可以继续运行。

② Run with multiple threads: 以多线程运行 VI。

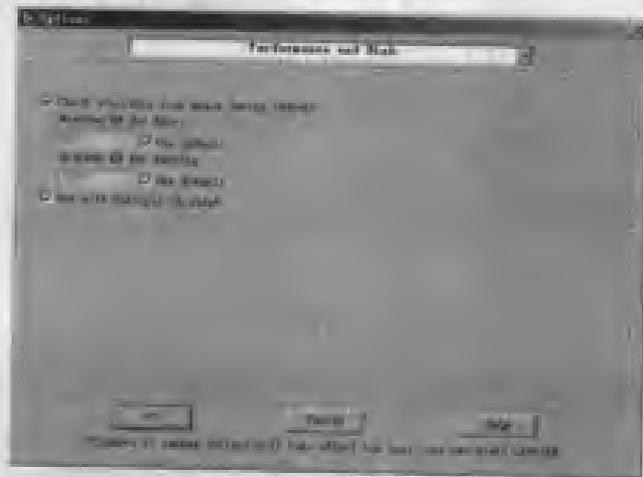


图 10.3.5 性能与资源设置页面

10.3.3 编程界面

1. 前面板

从 Options 对话框的下拉列表框中选择 Front Panel, 切换到性能与前面板设置页面, 如图 10.3.6 所示。该页面将设置前面板的各种显示属性。

- End text entry with Enter key: 允许在文本框中使用 Enter 键结束一行, 然后另起一行进行输入。

- Open the control editor with double click: 允许通过双击前面板前面板对象来打开前面板对象的编辑窗口。

- **Override system default function key settings:** 重载功能键, LabVIEW 将根据操作系统的不同而重新设置默认的功能键。
- **Use localized decimal point:** LabVIEW 默认的十进制数字的分位符是逗号, 选中此项则使用操作系统默认的分位符。
- **Show tip strips on front panel controls:** 当鼠标悬停在前面板对象上时显示对象的提示信息。
- **Use smooth updates during drawing:** 选中此项, 则界面更新时显得更加平滑, 但会降低程序的性能。
- **Use transparent name labels:** 使用透明的名字标签。
- **Play animated images:** 播放动画, 选中此项则播放任何格式的动画文件, 如 GIF 文件。
- **Blink speed:** 前面板对象闪烁的速度。

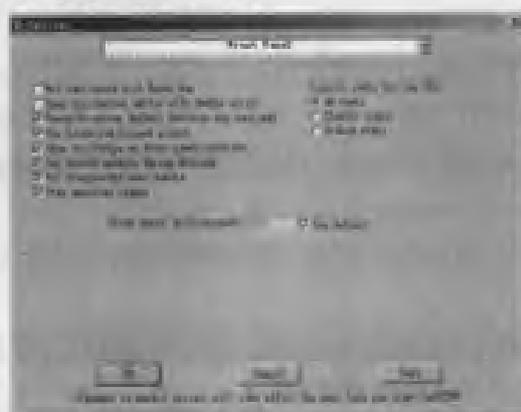


图 10.3.6 前面板属性设置页面

2. 框图

从 Options 对话框的下拉列表框选择 Block Diagram, 切换到性能与框图设置页面, 如图 10.3.7 所示。该页面设置程序框图的各种特性。

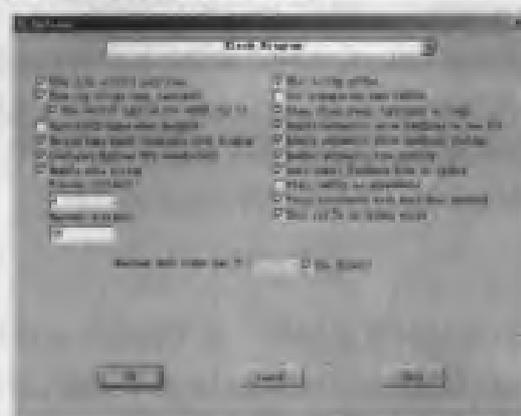


图 10.3.7 框图属性设置页面

- **Show dots at wire junctions:** 在连线的结合处显示一个圆点。
- **Show tip strips over terminals:** 当鼠标悬停在端口图标上时显示对象的提示信息。
- **Use control caption for subVI tip strips:** 显示子 VI 的标题而不是标签。

- Show subVI names when dropped: 当放置 VI 到框图中时, 显示 VI 的标签。
- Delete/copy panel terminals from diagram: 允许在框图程序中删除和复制前面板对象。
- Configure Express VIs immediately: 立即配置 Express VI。当向框图程序中放置 Express VI 时, 弹出对话框让用户设置 Express VI 的属性。
- Enable auto wiring: 允许自动连线, 当两个 VI 靠近时, LabVIEW 自动连接两个 VI 相互靠近的端口。

Maximum distance: 可以自动连线的最小距离。

Minimum distance: 可以自动布线的最小距离。

edges number (1)



图 10.3.8 连线提示

- Show wiring guides: 显示连线提示。若选中该项, 则当将连线工具悬停在 VI 上时, 显示 VI 的连线端口。如图 10.3.8 所示。
- Use transparent name labels: 使用透明的名称标签。
- Place front panel terminals as icons: 设置以图标形式还是数据类型形式显示前面板前面板对象。数据类型形式是 LabVIEW 的传统显示方式, 图标形式是 LabVIEW 7 Express 新增的显示方式。这两种显示方式如图 10.3.2 所示。此外, 还可以通过在图标上单击鼠标右键, 通过选择快捷选单 View As Icon 来选择显示方式。

● Enable automatic error handling in new VIs: 新建空白 VI 时, LabVIEW 将自动进行错误处理 (停止运行、高亮显示有错误的子 VI、弹出错误说明对话框), 取消该选择框将禁止自动错误处理功能。

● Enable automatic error handling dialogs: 取消该选择框, 即使已经开启了自动错误处理功能, 在有错误发生时也不弹出错误对话框。

● Enable automatic wire routing: 自动布线, 如果选择了该项, 连线时 LabVIEW 将自动选择数据线的所经路线。

● Auto insert Feedback Node in cycles: 自动添加反馈节点, 如果选择了该项, 当用户将一个 VI 的输出端与输入端相连时, LabVIEW 将自动在数据线中放置反馈节点。

● Place subVIs as expandable: 将插入程序框图的子 VI 以扩展形式显示。扩展形式与普通形式显示的图标如图 10.3.3 所示。该选项不影响 Express VI。

● Place structures with Auto Grow enabled: 设置结构体的自动放大属性。如果选中了该项, 当结构体中的对象过于靠近结构体的边界时, LabVIEW 将自动放大结构体。

● Show red Xs on broken wires: 在断的连线上放置“X”标识。

● Maximum undo steps per VI: 设置撤销操作的最大步数。

3. 网格线

网格线通常用于对齐前面板对象或图标。从 Options 对话框的下拉列表框中选择 Alignment Grid, 切换到网格线设置页面, 如图 10.3.9 所示。该页面设置网格线的各种属性。

① Show front panel grid: 显示前面板网格线。

● Enable panel grid alignment: 启用网格对齐功能, 前面板对象的边界将自动与网格对齐。

● Default panel grid size in pixels: 默认的网络尺寸, 单位为像素。

● Contrast with panel background: 网格线与前面板背景的对比度。

② Show block diagram grid: 在所有打开的 VI 中显示框图网格线。

● Enable diagram grid alignment: 启用框图的网格对齐功能。

- Default diagram grid size in pixels: 默认的网络尺寸, 单位为像素。
- Contrast with diagram background: 网格线与框图面板背景的对比度
- ③ Resize new objects to grid sizes: 调整新前面板对象的尺寸, 以使其贴紧网格线。
- ④ Alignment grid draw style: 网格线的线形。

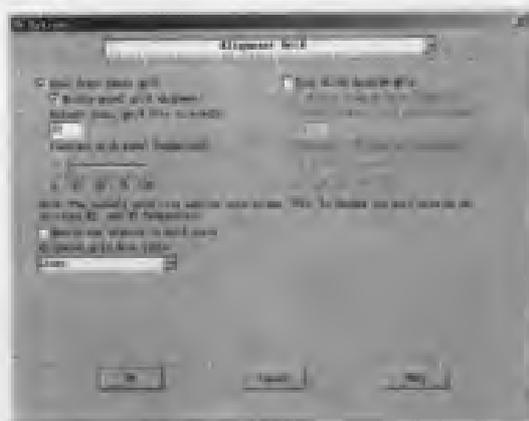


图 10.3.9 网格线设置页面

10.3.4 模板与调试

1. 模板

从 Options 对话框的下拉列表框中选择 Controls/Functions Palettes, 切换到网格线设置页面, 如图 10.3.10 所示。该页面设置 Controls 模板和 Functions 模板的各种属性, 如视图、格式、导航方式、模板载入方式等。

① Palette View: 设置 Controls 模板和 Functions 模板的视图类型。默认为 Express 型。用户可以定义自己的视图类型。

② Format: 模板的显示格式。

- Standard: 将所有的模板都显示为独立的模板, 包括主模板。

- Standard (Icons or Text): 将主模板中的条目显示为图标, 将子模板中的条目显示为独立的模板。

- All Icons: 将模板中所有的项目都显示为图标。

- All Text: 显示模板条目的文字列表。

- Icons and Text: 将模板条目显示为带有文字说明的图标。

③ Navigation Buttons: 设定如何显示模板上条目的标签。

- Label Selected Icons: 在 Search 按钮旁显示鼠标所指条目的标签。

- Icon Only: 不显示文字说明。

- Label All Icons: 显示所有条目的文字说明。

④ Palette Loading: 设置 LabVIEW 如何载入模板信息。

- Load palettes in background: 将模板信息载入后台, 这将提高模板搜索的速度, 但会影响 LabVIEW 的运行速度。

- **Load palettes when needed:** 在操作模板的时候载入模板信息。当用户操作 LabVIEW 模板的时候, LabVIEW 会显得慢一些。
- **Load palettes during launch:** LabVIEW 启动的时候载入所有模板信息。
- ⑤ **Use Window Titles in Functions palettes:** 在 Functions 模板中显示 VI 窗口名而不是 VI 的名称。
- ⑥ **Allow search in temporary palette:** 单击右键以显示临时模板, 模板上部有一个 Search 按钮, 取消此选项则不在临时模板中显示 Search 按钮。

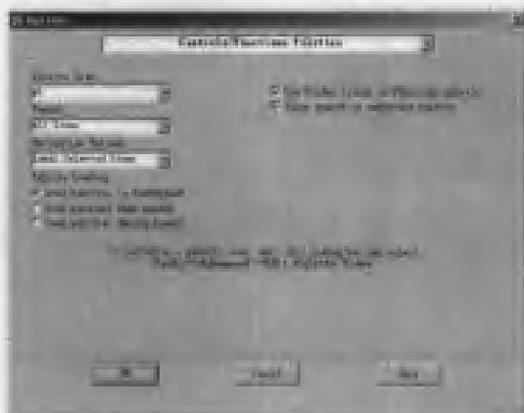


图 10.3.10 前面板对象和 Functions 模板属性设置页面

2. 调试

从 Options 对话框的下拉列表框选择 Debugging, 切换到调试设置页面, 如图 10.3.11 所示。

- **Show data bubbles during execution highlighting:** 调试时显示数据流动画, 数据将显示为沿着连线运动的圆点。
- **Auto probe during execution highlighting:** 自动显示单点数据。
- **Prompt to investigate internal errors on startup:** 若选中此项, LabVIEW 在启动时将显示前一次运行时产生的内部错误。
- **Show warnings in error box by default:** 在错误列表窗口中显示警告信息。



图 10.3.11 调试设置页面

10.3.5 属性设置

1. 颜色

从 Options 对话框的下拉列表框中选择 Colors，切换到颜色设置页面，如图 10.3.12 所示。在这里可以设置系统使用的各种颜色，如前面板颜色、框图颜色、滚动条颜色等。

- Use default colors: 使用默认颜色设置。
- Front Panel: 前面板颜色，影响新创建的 VI。
- Block Diagram: 框图的颜色，影响新创建的 VI。
- Scrollbar: 滚动条的颜色。
- Coercion Dots: 数据舍入指示符的颜色。
- Menu Text: 选单文字的颜色。
- Menu Background: 选单背景的颜色。
- Blink Foreground: 前面板对象闪烁时的前景色。
- Blink Background: 前面板对象闪烁时的背景色。
- User-Defined Colors: 弹出颜色设置对话框。



图 10.3.12 颜色设置页面

2. 字体

从 Options 对话框的下拉列表框中选择 Fonts，切换到字体设置页面，如图 10.3.13 所示。



图 10.3.13 字体设置页面

① 字体选择列表框: 从该下拉列表框中选择要设置的字体。

- **Application Font:** 用于设置 Controls 模板、Functions 模板、新创建的前面板和框图对象的文字字体。

- **Dialog Font:** 用于设置对话框的字体。

- **System Font:** 用于设置选单的字体。

② Use default font: 使用默认的字体设置。

③ Custom font: 使用用户自定义的字体。

④ Font Style: 弹出字体定义对话框。

3. 打印设置

从 Options 对话框的下拉列表框中选择 Printing, 切换到打印设置页面, 如图 10.3.14 所示。可以在这里设置打印方式、页边距等。

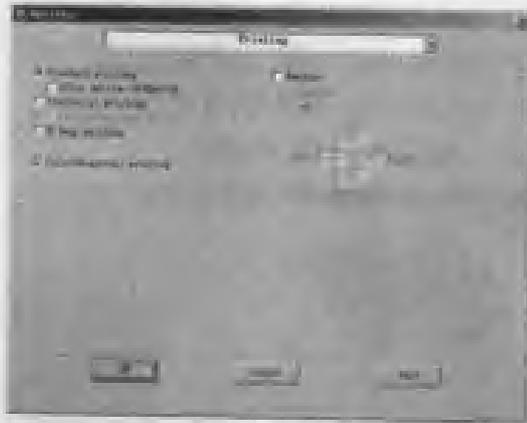


图 10.3.14 打印设置页面

① **Standard printing:** 标准打印方式, 如果打印机不支持 PostScript 方式, 则必须选择此项。

- **Allow printer dithering:** 允许打印抖动, 以产生灰度效果。

② **PostScript printing:** 将 VI 数据转换为 PostScript 格式, 然后送往打印机打印。

- **PostScript level 2:** 如果打印机支持 PostScript level 2 代码, 发送 PostScript level 2 代码到打印机。

③ **Bitmap printing:** 打印的文字效果较好, 图像的分辨率较低。

④ **Color/Grayscale printing:** 彩色 (需要彩色打印机) 或灰度打印。

⑤ **Margins:** 页边距设置。

- **Inches:** 使用英寸为单位。

- **Mm:** 使用厘米为单位。

- **Top:** 顶部空白。

- **Bottom:** 底部空白。

- **Left:** 左边距。

- **Right:** 右边距。

4. 版本历史

从 Options 对话框的下拉列表框中选择 Revision History，切换到版本历史设置页面，如图 10.3.15 所示。在这里可以设置版本信息的修改方式和用户登录方式等。



图 10.3.15 版本历史页面

- **Add an entry every time the VI is saved:** 保存 VI 时添加一项记录。如果用户没有在弹出对话框的 Comment 一栏中填写任何信息，只在版本信息中增加一个标题。
- **Prompt for comment when the VI is closed:** 如果 VI 有改动，在关闭 VI 时弹出一个对话框提醒用户写入版本信息。
- **Prompt for a comment when the VI is saved:** 如果 VI 有改动，在保存 VI 时弹出一个对话框提醒用户写入版本信息。
- **Record comments generated by LabVIEW:** 如果 LabVIEW 自身对 VI 进行了修改（如由于 LabVIEW 升级而重新编译 VI），则自动在 History 窗口中产生一条记录。
- **Show revision number in titlebar:** 在 History 窗口的标题中显示版本信息。
- **Login automatically with the LabVIEW registration name:** 自动使用已注册用户名登录。
- **Login automatically with the system user name:** 自动使用系统用户名登录。

Show the login prompt at LabVIEW startup time: 在 LabVIEW 启动时显示登录对话框，提示用户输入用户名。设置用户名可以通过在选单中选择 Edit→User Name。

5. 其他属性

从 Options 对话框的下拉列表框中选择 Miscellaneous，切换到杂项设置页面，如图 10.3.16 所示。该页面用于设置无法明确归类的其他属性。

用户的口令，以便下一次访问时无需再次输入口令。选择此项 LabVIEW 将清除口令缓存，下次登录时需重新输入口令。

10.3.6 VI Server 与 Web Server

1. VI Server

LabVIEW VIs 的程序控制是通过 VI Server 技术实现的。VI Server 是 LabVIEW 独有的一项技术，利用 VI Server 技术，用户可以通过编程来动态控制 LabVIEW VIs，这些 VIs 可以位于本地计算机中，也可以位于网络上的远程计算机中。利用 VI Server 可实现下列功能：

- 本地动态调用 VIs，动态地将 VIs 加载到内存中；
- 将一台计算机中的 LabVIEW 配置成为一个可以导出 VIs 的 VI Server，在另一台计算机上通过网络远程动态调用这些 VIs；
- 动态更改 VI 的属性和 LabVIEW 的属性；
- 获得 LabVIEW 的相关信息，例如版本和版本编号；
- 通过为应用程序创建一个内插结构来为其增加功能。

默认的 VI Server 设置可以满足大部分程序的需求，对于特殊的要求，则应手动设置 VI Server。下面介绍 VI Server 的设置方法。

(1) 协议与服务配置

从 Options 对话框的下拉列表框中选择 VI Server: Configuration，切换到协议与服务配置页面，如图 10.3.17 所示。

- Protocols：设置 VI Server 使用的协议。协议有两种：TCP/IP 和 ActiveX。
- Server Resources：设置服务器提供的服务。

Allow VI calls：允许用户远程调用本地 VI。

Allow VI methods and properties：允许远程用户读取和设置本地 VI 的属性。

Allow application methods and properties：允许远程用户读取和设置 VI Server 的属性。

Allow control methods and properties：允许远程用户读取和设置本地 VI 的前面板前面板对象的属性。



图 10.3.17 协议与服务配置页面

(2) TCP/IP 协议访问控制 (TCP/IP Access)

从 Options 对话框的下拉列表框中选择 VI Server: TCP/IP Access, 切换到 TCP/IP 协议访问控制页面, 如图 10.3.18 所示。



图 10.3.18 TCP/IP 协议访问控制页面

- TCP/IP Access List: 进行访问控制的地址列表。地址前面有“√”表示允许访问; 地址前面有“×”的表示禁止访问。
- Allow Access: 允许访问。
- Deny Access: 禁止访问。
- Add: 增加新地址。
- Remove: 删除一个地址。
- Strict Checking: 设置 TCP/IP 服务器是否检查域名。

(3) 设置导出 VI

从 Options 对话框的下拉列表框选择 VI Server: Exported VIs, 切换到导出 VI 页面, 如图 10.3.19 所示。导出的 VI 就是可以被访问的 VI。一个 VI 只有在添加到 VI 列表且被设置为“Allow Access”, 才能够被访问。



图 10.3.19 导出 VI 页面

- Exported VIs: 导出 VI 的列表。
- Allow Access: 允许 VI 被访问。
- Deny Access: 禁止 VI 被访问。
- Add: 单击此按钮以增加一个 VI。
- Remove: 从导出列表中移除一个 VI。

2. Web Server

Remote Panels 技术允许用户可以用极为简单的方式直接在本地 (Client 端) 计算机上打开并操作位于远程 (Web Server 端) 计算机上的 VI 前面板, 甚至可以将 LabVIEW VIs 的前面板窗口嵌入到一个网页中并在网页中直接操作它。必须首先在 Server 计算机上运行 LabVIEW, 并配置 Web Server。Web Server 需要下面三个方面的配置:

- 文件路径和网络设置。
- 客户机访问权限设置。
- VIs 访问权限设置。

(1) 文件路径和网络设置

从 Options 对话框的下拉列表框中选择 Web Server: Configuration, 切换到文件路径和网络设置配置页面, 如图 10.3.20 所示。

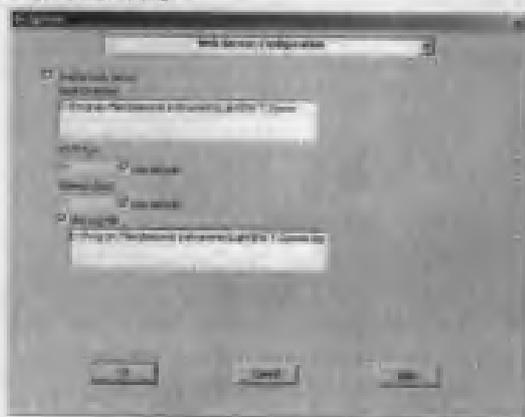


图 10.3.20 文件路径和网络设置配置页面

选中 Enable Web Server, 可以启动 LabVIEW Web Server。LabVIEW Web Server 默认的 HTTP 端口号为 80, 在通常情况下, 端口号 49152~65535 是推荐给用户自定义 TCP/IP 应用程序使用的网络端口。配置页中的其他选项包括 Server HTML 的根目录、Timeout 时间设定、Log 文件路径设置等。

(2) 客户机访问权限设置

从 Options 对话框的下拉列表框中选择 Web Server: Browser Access, 切换到客户机访问权限设置页面, 如图 10.3.21 所示。

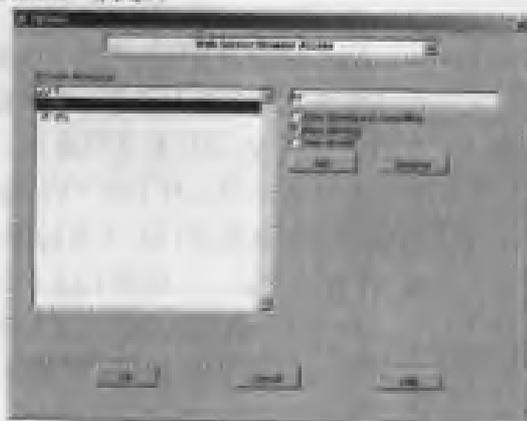


图 10.3.21 客户机访问权限设置页面

在这个页面中可以设置允许或禁止访问的客户机, 以及其访问权限。下面是该页面中几种符号的含义:

- 符号 “*” 表示任意客户机;
- 符号 “√√” 表示允许客户机观看并控制 Remote Panels;
- 符号 “√” 表示仅允许客户机观看 Remote Panels;
- 符号 “×” 表示禁止该计算机访问。

(3) VIs 访问权限设置

从 Options 对话框的下拉列表框中选择 Web Server: Visible VIs, 切换到 VIs 访问权限设置页面, 设置允许客户访问的 VIs。如图 10.3.22 所示。



图 10.3.22 VIs 访问权限设置页面

下面是该页面中符号和选项的含义:

- 符号 “*” 表示允许访问所有的 VIs;
- 符号 “√” 表示允许客户机访问该 VI;
- 符号 “×” 表示禁止客户机访问该 VI;
- Control Time Limit 选项用于设定客户机控制该 VI 的时间限制。

上述所有的配置参数可以利用 VI Server 在程序中进行动态的配置。

10.4 VI 属性设置



图 10.4.1 VI 图标的右键弹出选单

VI 在运行时的表现除了在编程过程中要进行适当的规定外, 更多的是在 VI 属性设置中完成的。VI 的属性设置通过 VI 属性对话框完成。打开 VI 属性对话框的方法是从选单中选择 File → VI Properties... 或者用鼠标右键单击前面板窗口右上角的图标, 弹出快捷选单, 选择 VI Property... 如图 10.4.1 所示。

VI 属性对话框如图 10.4.2 所示。在 Category 下拉列表框中选择需要设定的属性类别, 目前的属性类别有以下几种。

- General: 提供图标编辑器, 显示 VI 路径和版本

信息。

- Memory Usage: 显示 VI 所占用的磁盘空间和系统信息。
- Documentation: 设置 VI 的说明信息, 建立 VI 和帮助文件之间的链接。
- Revision History: 设置当前 VI 的版本信息。
- Editor Options: 设置使用右键快捷选单命令 Create→Control 或 Creat→Indicator 创建前面板对象时前面板对象的外观, 如前面板对象字体、显示风格等。
- Security: 设定 VI 的访问口令。
- Window Appearance: 设定 VI 的窗口外观, 如标题名、窗口内容等。
- Window Size: 设定窗口尺寸。
- Execution: 设定 VI 的运行属性, 包括优先级、运行子系统等。
- Print Options: 设置 VI、模板的打印属性, 如页边距设置、是否打印页头、前面板图形是否加边框等。

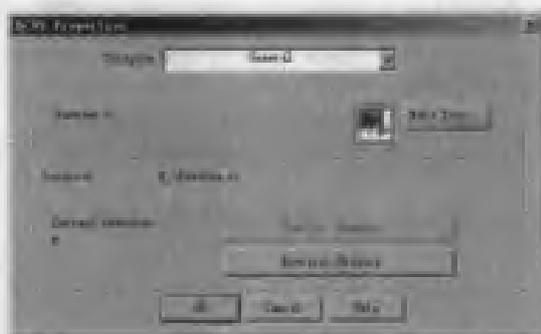


图 10.4.2 General 页面

10.4.1 一般设置

General 页面提供图标编辑器, 显示 VI 路径和版本信息, 如图 10.4.2 所示。

1. 修改图标

单击 Edit Icon 按钮将弹出图标编辑界面。图标的编辑对整个程序没有影响, 只是为了得到一个比较容易识别的标记, 编辑界面如图 10.4.3 所示。



图 10.4.3 图标编辑器

左边的工具和 Windows 中的画图工具一样, 只是功能稍微简单一些。利用它们可以做出程序的图标, 方法与画图工具基本一样。一旦确认后, 在程序的右上角就会显示出这一图标, 被别的程序调用时, 该程序将以这一图标出现在框图中。

2. VI 的变更信息

通过查看变更信息, 可以了解当前 VI 相对上一版本所进行的修改, 以及可能会对 VI 造成的影响。要查看 VI 的变更信息, 只需单击 Current Changes... 按钮, 切换到的变更解释对话框, 如图 10.4.4 所示。

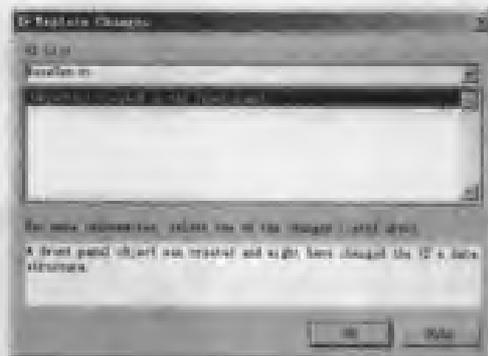


图 10.4.4 VI 变更信息查看对话框

3. 版本信息

一个 VI 在设计过程中可能会进行多次修改, 从而形成多个版本。可以为每个版本添加一定的说明信息, 以备日后参考。将每次 VI 修改后的变更信息以及对 VI 的影响加入版本信息中是一个很好的习惯。版本信息设置对话框如图 10.4.5 所示。

单击 Reset 按钮将消除原有的版本信息。如果要增加一条版本信息, 可以在 Comment 文字框中编写版本信息, 然后单击 Add 按钮。

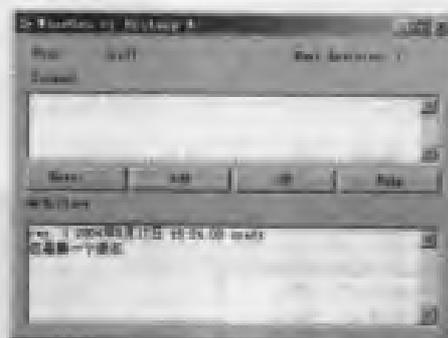


图 10.4.5 版本信息编辑对话框

10.4.2 存储空间

在有些应用场合需要限制 VI 的体积, 因此需要对 VI 占用的磁盘空间有所了解。LabVIEW 提供了相应的手段, 可以查看 VI 所占用的内存和硬盘前面板对象。在 VI 属性

对话框的 Category 一栏中选择 Memory Usage, 切换到存储空间使用页面, 该页面显示了 VI 所占用的内存大小以及所占用的硬盘空间大小, 如图 10.4.6 所示。

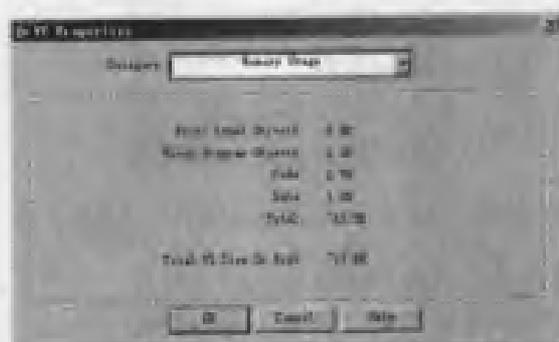


图 10.4.6 存储空间查看页面

- Front Panel Objects: 前面板前面板对象所占用的内存。
- Block Diagram Objects: 框图中的对象所占用的内存。
- Code: VI 编译时产生的代码所占用的内存。
- Data: VI 的数据空间所占用的内存。
- Total: VI 占用的内存总量。
- Total VI Size On Disk: VI 占用的硬盘空间。

10.4.3 帮助与编辑

1. 帮助信息

可以为 VI 添加说明, 并将帮助文档链接到 VI, 以增进用户对 VI 的理解。从 VI 属性对话框的 Category 下拉列表框中选择 Documentation, 切换到文档页面, 如图 10.4.7 所示。

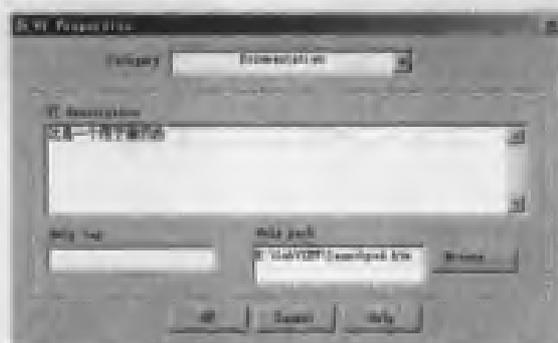


图 10.4.7 帮助文档设置页面

① VI description: VI 的说明信息, 该信息将显示在 Context Help 窗口中。可以使用 **** 和 **** 标记加粗文本, 位于 **** 和 **** 标记标记之间的文字将以粗体显示, 如图 10.4.8 所示。



图 10.4.8 显示在 Context Help 窗口中的帮助信息

② Help tag: 当帮助文件是.chm 或.hlp 格式的文件时, 这些格式的帮助文件是由多个 HTML 文件组成的, Help tag 指定帮助文件中特定 HTML 文件的文件名。

③ Help path: 帮助文件的路径, 帮助文件可以是 HTML 文件, 也可以是.chm 或.hlp 文件。在应用中, 单击 Context Help 窗口中的 Click here for more help. 链接将自动打开帮助文件。

④ Browse: 弹出一个对话框, 可通过此对话框选择帮助文件。

2. 编辑

编辑器属性设置(Editor Options)页面用来设置前面板和框图网格的大小, 以及使用右键快捷选单创建前面板对象时前面板对象的外观风格。编辑器属性设置页面的界面如图 10.4.9 所示。

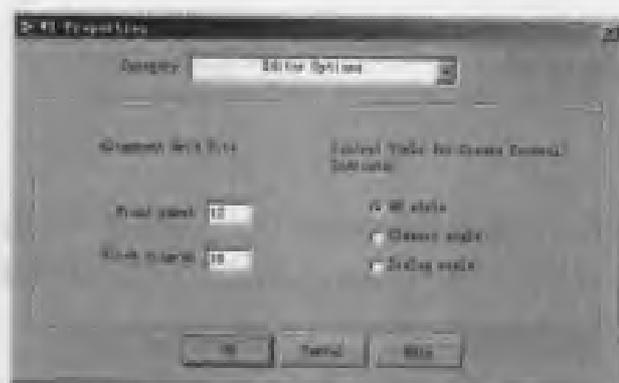


图 10.4.9 编辑器属性设置页面

10.4.4 版本历史与安全

1. 版本历史

版本历史记录(Revision History)页面用以设置 VI 的版本信息记录方式。版本历史记录页面的界面如图 10.4.10 所示。

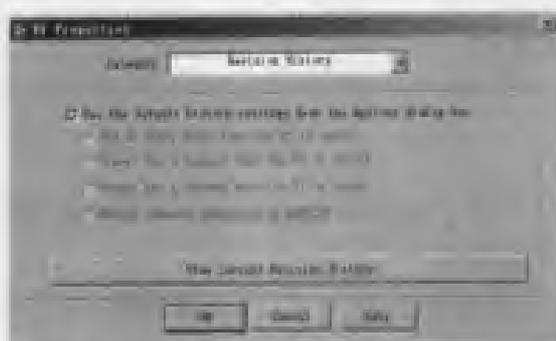


图 10.4.10 版本历史记录页面

① Use the default history settings from the Options dialog box: 使用 Options 对话框中的设置。

② Add an entry every time the VI is saved: 保存 VI 时添加一项记录, 如果用户没有在弹出对话框的 Comment 一栏中填写任何信息, 只在版本信息中增加一个标题。

③ Prompt for comment when the VI is closed: 如果 VI 有改动, 在关闭 VI 时弹出一个对话框, 提醒用户写入版本信息。

④ Prompt for a comment when the VI is saved: 如果 VI 有改动, 在保存 VI 时弹出一个对话框提醒用户写入版本信息。

⑤ Record comments generated by LabVIEW: 如果 LabVIEW 自身对 VI 进行了修改(如由于 LabVIEW 升级而重新编译 VI), 则自动在 History 窗口中产生一条记录。

⑥ View Current Revision History: 显示 VI 当前的版本历史记录。

2. 安全设置

如果 VI 设置了口令, 用安全设置 (Security) 页面设置 VI 的访问属性, 如图 10.4.11 所示。

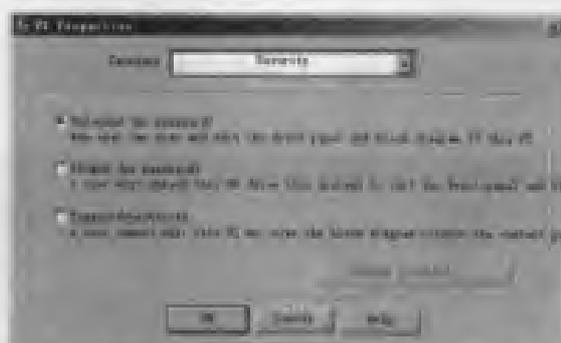


图 10.4.11 安全设置页面

① Unlocked (no password): 允许任何用户浏览、编辑前面板和框图程序。

② Locked (no password): 锁定 VI, 用户必须到此页面将 VI 解锁才能够编辑前面板和框图程序。

③ Password-protected: 用户如要编辑 VI 必须输入正确的口令。

④ Change Password: 修改 VI 的访问口令。

10.4.5 窗口与运行

1. 窗口外观

设置 VI 运行时的窗口外观 (Window Appearance)，设置界面如图 10.4.12 所示。

① Window title: 窗口标题。

② Same as VI name: 如果选中 Same as VI Name 选择框，窗口标题将使用 VI 的名称。

③ 窗口类型共有四种。

- Top-level application window: 显示标题和选单栏，隐藏滑动条和工具条，不允许改变窗口大小，如果作为子 VI，在被其他 VI 调用时打开前面板。

- Dialog: 对话框类型，VI 打开时用户不能操作其他 LabVIEW 窗口。

- Default: 使用 LabVIEW 开发环境的窗口设置。

- Custom: 用户自定义。

④ Customize: 显示用户自定义设置对话框。

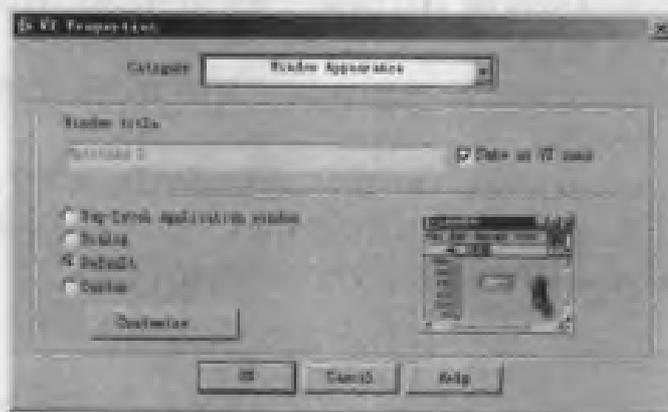


图 10.4.12 窗口外观设置页面

2. 窗口尺寸

从 Category 一栏中选择 Window Size，打开窗口尺寸属性设置 (Window Size) 页面，如图 10.4.13 所示。

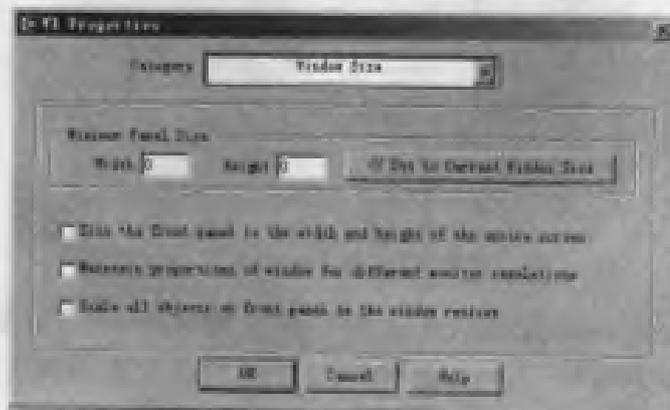


图 10.4.13 窗口尺寸设置页面

① **Minimum Panel Size:** 设置窗口的最小尺寸。当允许用户缩放窗口时,窗口的尺寸将不能小于在此设定的尺寸。

- **Width:** 前面板的最小宽度,以像素为单位。
- **Height:** 前面板的最小高度,以像素为单位。
- **Set to Current Window Size:** 使用当前窗口的宽度和高度设置。

② **Size the front panel to the width and height of the entire screen:** 当运行 VI 时,自动调整窗口至整个屏幕大小。

③ **Maintain proportions of window for different monitor resolutions:** 若选中此选项,则 VI 自动调整前面板的大小,以便当屏幕的分辨率不同时前面板在显示器屏幕中的尺寸大致相等。

④ **Scale all objects on front panel as the window resizes:** 调整窗口大小时,前面板对象的尺寸将同时成比例缩放。

3. 运行

运行 (Execution) 页面用于设定 VI 运行时的一些属性,如图 10.4.14 所示。各选项的含义如下所述。

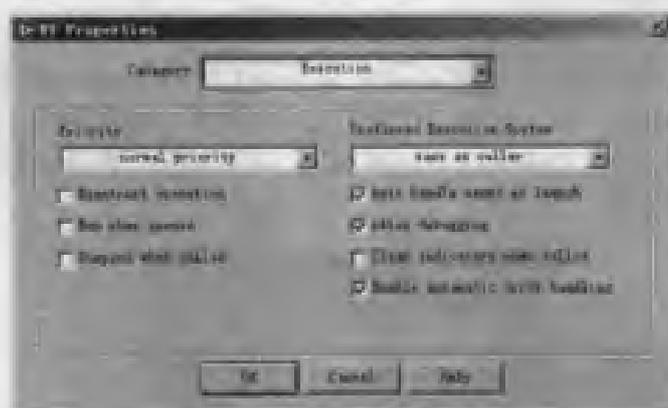


图 10.4.14 运行页面

● **Priority:** 设置 VI 的优先级。关于优先级和运行子系统的相关内容请参考清华大学出版社出版的《LabVIEW 程序设计高级教程》一书。

● **Preferred Execution System:** 设置运行子系统。

● **Run when Opened:** 当程序被打开时,自动运行,一般用于主程序的设置。

● **Suspend when Called:** 被调用时程序挂起,主要用于调试程序。

● **Reentrant Execution:** 可同时被多个程序调用。在一般情况下,不同的程序不能同时调用同一个程序,只能采用等待的方式,待第一次调用结束后,再进行第二次调用。选中此项后,就可以同时被调用了。

● **Auto handle menus at launch:** 设定在打开 VI 时,LabVIEW 是否自动处理选单。如果取消该选项,运行时选单 (run-time menu) 将被禁止。

● **Allow debugging:** 选中该选项将允许调试 VI,如单步运行、设置断点。

● **Clear indicators when called:** 如果选中该项,VI 运行时首先清除前面板中的指示。

● **Enable automatic error handling:** 开启自动错误处理。

10.5 文件管理

LabVIEW 的文件管理方式有两种:一种是将各个文件分散独立地保存;另一种是将相关文件集中起来保存于一个库文件中。这两种方式各有利弊,利用分散独立保存方式保存,文件的管理和使用比较方便,可以进行常规的文件操作,如复制、更名等,也能够在一个文件夹中建立新的文件夹,对整个项目的所有文件进行分层次的管理。同时,独立保存的文件在加载和保存时要比用库的形式进行得快,需要的资源也少。利用库的形式保存的文件就没有这些特点,而且 VI 库还与一些专业的开发工具不兼容,不过库文件的保存形式是一种压缩方式,所以节省磁盘空间,并且可以方便地移植到其他平台上去,安全性也比较好。

分散独立保存的方法与系统中其他文件类型类似,这里主要介绍如何用库的形式管理文件。库文件的扩展名是 LLB,它的实质是将所有选中的文件压缩在一个库文件中,这些文件保持着自己的独立性。

1. 将文件保存到库文件中

这种方式有两种情形:一种是创建一个新的库文件;另一种是将文件保存于现有的库文件中。

创建库文件的基本步骤如下:首先在 File 选单中选择 Save as,弹出一个文件对话框,然后按下 New VI Library 按钮,出现一个新的提示对话框,如图 10.5.1 所示,用户就可以输入库文件名称,建议库文件的名称不使用汉字,否则可能会在发布程序时出错。这样,就建立了一个新的库文件。



图 10.5.1 New VI Library 对话框

将文件保存在已有库文件中的基本步骤如下:在 File 选单中选择任意一个保存项后,在保存文件的对话框中选择一个已有的库文件,弹出一个文件对话框,如图 10.5.2 所示,只要将文件名称输入到 Name the VI 栏中,确认后就完成了工作,将 MovePointGroup.vi 保存到了 fungen.llb 中。

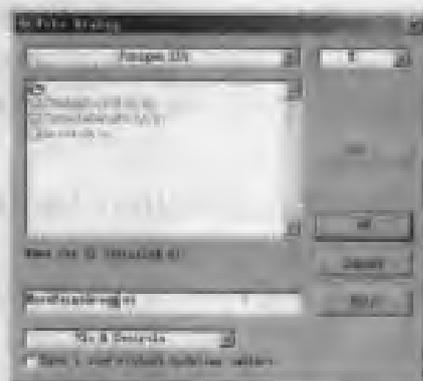


图 10.5.2 VI 文件保存对话框

2. 库文件中的内容

对库文件的整理,实际上是对其中的内容进行调整的过程,在选择 Tools 选单中的 Edit VI Library 选项后,出现一个编辑 VI 库的对话框,如图 10.5.3 所示。选择好某一文件后,可以按删除按钮将它从库中清除掉。对话框中有一个“Top Level”选项,这一项的作用有两个:一个是当运行应用程序时,处于顶层的程序都自动打开;另一个是在双击库文件名时,LabVIEW 将自动打开这些程序。

整理库文件的内容还可以在 Tools 选单中进行,选择 VI Library Manager 项,弹出一个文件管理对话框,如图 10.5.4 所示。这个对话框表示要把左边 LabVIEW 文件夹中的文件 LABVIEW.ini 复制到右边的库文件 fungen.lib 中去。另外,这一对话框还提供了文件夹与库文件相互转换的功能。

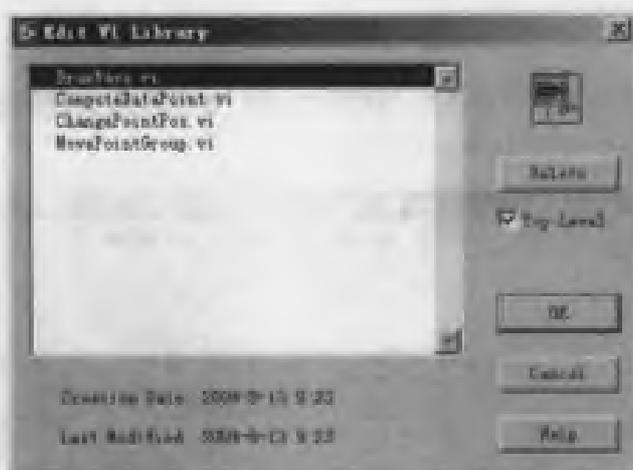


图 10.5.3 Edit VI Library 对话框



图 10.5.4 File Manager 对话框

10.6 创建应用程序

LabVIEW 作为一个编程环境,其编译过程是实时完成的,但是要完成更多的功能必须要有相应工具包的支持。LabVIEW 7 Express 提供了多种实用的工具包,这里介绍 Application Build 工具的使用方法。

LabVIEW 环境本身不能产生可执行文件,所以在程序完成后,不能脱离 LabVIEW 环境运行,这给发布应用程序带来极大的不便。Application Build 工具包弥补了这一不足,它将程序转换成可执行文件,使之能和普通的应用程序一样运行在操作系统中。

从系统菜单中选择 Tools → Build Application or Shared Library (DLL), 将弹出 Build Application 对话框,其中包含 5 个页面: Target, Source Files, VI Settings, Application Settings 和 Installer。

1. Target 页面

Target 页面主要用于设定目标文件的相关信息,包括目标文件的名称、路径、支持文件以及创建方式等内容,如图 10.6.1 所示。

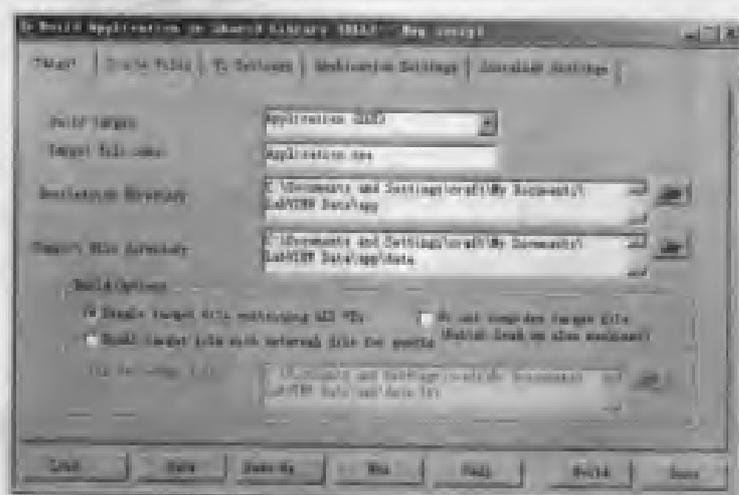


图 10.6.1 Build Application 对话框的 Target 页面

第一项是创建目标文件的名称,默认的名称为 Application.exe。

第二项中的内容是该目标文件的路径,默认路径为 C:\WINDOWS\TEMP\app,单击右边的立体按钮,可以选择其他的路径,也可以直接输入路径名。

第三项中的内容是相关支持文件的路径,同样可以改变它的设置。

Build 一栏中有两个选项,是创建可执行程序的具体形态。其一是将所有的子程序都封装在一起,形成一个大的应用程序;其二是将子程序所在的库文件共享,让它们挂在外部,在发布程序时将它们一起发布,这些库文件可以被其他程序调用。

2. Source Files 页面

Source Files 页面规定了应用程序源文件的相关信息和相互关系,创建应用程序所用到的文件都称为源文件,这些源文件按照它们在程序中的作用,分为顶层子程序、动态子程序和支持文件 3 类,如图 10.6.2 所示。

顶层子程序是在应用程序运行时用以自动加载的程序, 类似于一般编程语言的主程序, 通常顶层子程序要尽量地少。

动态子程序是应用程序在运行过程中, 需要动态调用的子程序, 它们可完成项目中的一个或者若干个功能, 通常以程序库的形式保存在适当的路径下。



图 10.6.2 Build Application 对话框的 Source Files 页面

支持文件是一些与应用程序相关的文本文件或非 LabVIEW 环境下的文件, 它们中有些是必需的, 也有一些是用以标识程序的。

3. VI Settings 页面

VI Settings 页面可对应用程序中用到的所有程序的运行特点和保存方式进行设置, 如图 10.6.3 所示。VI 有 9 个选项设置, 可以根据每个程序在项目中的作用和地位, 对前面板、打开方式、窗口类型等进行设置, 这在应用程序的占用空间和运行效率方面起着重要的作用。在 VI 列表中选中 VI, 单击 Edit Build Settings..., 将弹出设置对话框, 在这里可以更改 VI 编译设置。



图 10.6.3 Build Application 对话框的 VI Setting 页面

4. App Settings 页面

App Settings 页面用于设置应用程序的基本特性, 共有 3 项内容: 定制程序图标、设置命令行参数传递方式和激活 ActiveX 服务器, 如图 10.6.4 所示。



图 10.6.4 Build Application 对话框的 App Setting 页面

任何图标文件都可以成为应用程序的图标, 如果不指定文件, 系统将提供图中默认的 4 个图标。一旦程序被定制为 ActiveX 服务器使能, 该应用程序就可以作为 ActiveX 服务器被其他程序使用。服务器的名称可以指定, 也可以使用系统提供的默认名称。

5. Installer 页面

Installer 页面是关于建立安装程序的有关内容, 如图 10.6.5 所示。在发布程序时, 一般要创建一个安装程序, 为完成这一工作, LabVIEW 提供了 6 个项目的内容、1 个安装程序属性 (Properties...) 设置和 1 项高级特性 (Advanced...) 的设置。

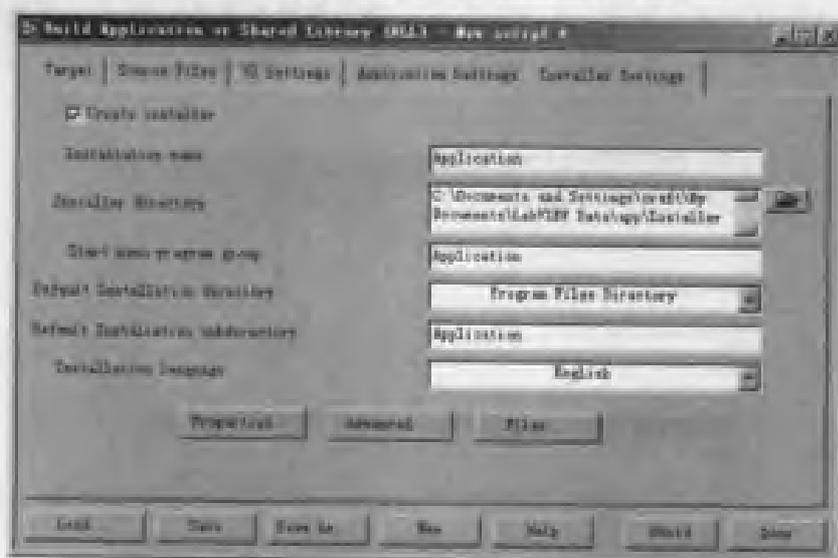


图 10.6.5 Build Application 对话框的 Installer 页面

在确定了要创建安装程序后，页面上的所有项目都由灰色变为可操作状态，这些项目分别规定了安装程序的名称、在开始选单中的程序组名称、默认的安装路径、安装环境的语言、发布程序所用载体的容量以及首张安装盘上的额外空间等内容。

安装环境的语言由系统提供了包括英语、法语、德语和日语在内的4种语言，但没有中文，这并不影响汉字在 LabVIEW 中的一般应用。发布程序所用载体分为软盘和光盘，可以根据实际情况选择。

安装程序属性设置的内容有安装文件的名称、数据压缩文件的名称、发行商、作者、软件版本等。

高级选项中有6个内容，都非常实用，包括安装完毕后立即运行程序、安装 Run-Time Engin、安装远程面板许可证、串口通信支持、端口 I/O 支持、硬件设置。

10.7 培养良好的编程风格

编程风格是一种主观性很强的个人意识的表达，没有绝对的正确与错误的区分，完全是根据项目的要求和个人的习惯而定的。但是编写程序时，一个好的开发习惯会使工作方便许多。对于 LabVIEW 也是如此，并且由于 LabVIEW 是一种与其他传统语言不太一样的图形化编程语言，所以在培养良好的编程风格时，更应该注意一些细节问题。这里搜集了一些开发者在编程中的心得，描述得并不是十分完全，但可能会对读者有一点帮助。

1. 对待程序总体设计的基本行为

任何一个项目，不管大小、复杂程度如何，形成过程都是基本一致的。

(1) 先总体设计，再考虑代码。实际上是要产生一个用户需求文档，了解清楚软件最终的界面和功能。有了基本的框架后，就可以考虑数据结构、程序流程、事件、时间与经费预算等因素了。

(2) 在使用一小段代码前先对它进行验证（包括现成的 VI）。验证的结果必须保证正确，这样在以后的程序调试中，就可以不考虑它的负面影响了，无疑对编程效率会有很大的提高。

(3) 保存各个子程序，以供将来使用。每一个子程序都是思维的阶段性总结，除了能够完成特定的软件功能外，还可以为解决难题提供参考方法。

(4) 保证代码的整洁与清晰。一目了然的代码，能帮助用户查找错误和创建子程序。

(5) 增加注释。写出一部分代码后，及时地写出响应的注释，注释需要简洁和全面，能够清晰地描述代码的功能和构思过程。

(6) 对程序命名时，尽量用描述性的语言（注意不同平台的文件命名方法）。争取从名称上就能看出程序的作用。例如 Stxt.vi，就可以简单地描述这个程序的作用是保存文本文件，甚至可以用汉字命名，如“读数据库.VI”，就很清楚了。

(7) 在完成一个程序后，尽量在 File→VI Property 对话框中的 Documentation 页简要地解释程序的功能，这些内容可以在 Help→Show Context Help 形式中显示出来。

2. 面板与用户界面的区别

首先判断一个 VI 是一个节点程序还是具有交互界面的程序, 对待它们的方式是不同的。对于一般的节点, 它的界面是不出现的, 只需要考虑其输入输出参数, 可以采用较常见的 LabVIEW 前面板对象, 而不必考虑修饰。

对于用户界面, 需要考虑的因素就多一些了, 既要考虑界面的简洁美观大方, 还要考虑操作方便。

(1) 把相应前面板对象进行编组、分类。如果界面上的一些前面板对象是相互关联的, 就可以把它们归入一个单独的集体中, 或封装成一个 Cluster, 还可以把它们摆放在一起, 在框图中也要如此安排。

(2) 充分考虑屏幕的大小。要保证用户界面既不能太小, 也不能超出屏幕的显示范围。当然, 屏幕的分辨率也是一个重要的因素。

(3) 前面板对象间要留有足够的间隔以便操作。这一点对于触摸屏尤其重要, 误操作的后果在自动化测试中是不可估量的。弹出对话框时, 其按钮不要与下一层的按钮重合, 否则, 一个双击很有可能变成对两个按钮的单击。

(4) 在默认选项上使用焦点, 可以提高程序运行中的操作效率, 给各个前面板对象使用含义明确的名称, 也会在操作中起到意想不到的作用。

(5) 在适当的地方标出默认值, 各个前面板对象一般都有默认的值或者状态。为了提示程序用户, 尽量把这些状态值显示出来, 可以避免操作过程中的混乱。

3. 图标和连线端

对它的合理定义会带来很多的方便。

(1) 在定义 VI 的连线端时, 最好要定义其连接属性, 包括必选、推荐使用和可选 3 种属性。按照程序的输入输出参数特性, 将对应端点的属性定义好, 对于以后的编程中如何使用这一程序有指导作用。

(2) 如果需要简洁明了, 直接用文字作图标也是一个好的方法。例如在编辑图标时, 不用图形的方式, 而直接写入“读端口 1”, 其含义就十分明确了, 这种文字表示就好像注释一样。

4. 框图程序

提高框图程序的效率主要有两个方面: 一是代码清晰, 操作方便; 二是资源占用少, 运行效率高。

(1) 为了方便, 可以定制自己的功能模板。可以将当前项目中常用的节点、函数等置于一个由自己定制的功能模板中, 操作时就会方便一些。

(2) 在数组操作和字符串操作时, 要高效率地使用内存。数组和字符串是两类常用的数据集合方式, 每一次应用都会占用一些内存, 所以要尽可能地不重复调用它们。主要是考虑程序的运行方式, 如在创建数组和字符串时, 不要放在循环中进行, 保证内存中只有一个操作对象。

(3) 限制局部变量和全局变量的使用。这两种变量的作用是在程序内部的数据通信以及程序间的信息传递中, 作为中间载体。全局变量的生命周期一直保持到整个程序结束,

它的开销是固定的，局部变量的生命周期与它所在的子程序相同，它们共同的特点是在一段时间内都要占用系统资源，如果能够用函数的形式实现信息传递，就尽量地不使用它们，尤其在资源不十分充足的情况下，更应如此。

(4) 参数将所有代码都放入一个屏幕内，尽量不用滚动条，这也是一目了然的表现形式。

5. 数据连线

数据连线是一个最容易被忽视的过程，对它的使用，基本上是利用一些技巧，但会直接影响到编程效率。

(1) 不要滥用 **Remove Broken Wires** 功能，尽管这是一个很好的操作，但是对它的使用容易删除一些并不需要删除的连线，而且还可能造成移位寄存器中数据的丢失，而要恢复它们可能得花上一两个小时。

(2) 如果要查询一段连线，可以用鼠标单击它，这时选中的就是连线的一个区段，双击就可以选中包括一个分支在内的连线，三击选中的是整个连线，这样可以避免在修改连线时的一些偶然错误。

(3) 尽可能地使用前面板对象或常量。在使用一个外部子程序时，对它的输入输出往往不是很清楚，这时可以在连线的端点处创建一个前面板对象、常量或者显示前面板对象，通过对它们进行简单地分析，就可以了解该子程序的功能了。

第11章 数据采集

随着计算机和总线技术的发展,基于 PC 的数据采集(Data Acquisition,以下简称 DAQ)板卡产品得到广泛应用。一般而言,DAQ 板卡产品可以分为内插式(plug-in)板卡和外挂式板卡两类。内插式 DAQ 板卡包括基于 ISA, PCI, PXI/ Compact PCI, PCMCIA 等各种计算机内总线的板卡,外挂式 DAQ 板卡则包括 USB、IEEE1394、RS232/RS485 和并口板卡。内插式 DAQ 板卡速度快,但插拔不方便;外挂式 DAQ 板卡连接使用方便,但速度相对较慢。NI 公司最初以研制开发各种先进的 DAQ 产品成名,因此,丰富的 DAQ 产品支持和强大的 DAQ 编程功能一直是 LabVIEW 系统的显著特色之一,并且许多厂商也将 LabVIEW 驱动程序作为其 DAQ 产品的标准配置。另外,NI 公司还为没有 LabVIEW 驱动程序的 DAQ 产品提供了专门的驱动程序开发工具——LabWindows/CVI。

本章首先介绍了数据采集的一些基础知识,概要论述了 LabVIEW 中各种 DAQ VIs 的功能、组织结构与参数设置,通过 PCI-1200 数据采集卡实例,介绍了 LabVIEW 环境下 DAQ 硬件的安装与配置,以模拟量采集(Analog Input)为例,论述了简易 DAQ 编程、扩展 DAQ 编程和高级 DAQ 编程的基本原理和方法。

11.1 数据采集基础

在一般情况下,DAQ 硬件设备的基本功能包括模拟量输入(A/D)、模拟量输出(D/A)、数字 I/O (Digital I/O)和定时(Timer)/计数(Counter)。因此,LabVIEW 环境下的 DAQ 模板设计也是围绕着这 4 大功能来组织的,下面将简要介绍一些与此有关的 DAQ 基本概念。

11.1.1 DAQ 功能

1. A/D

A/D 转换器是把输入模拟量转换为输出数字量的器件,也是 DAQ 硬件的核心。就工作原理而言,A/D 转换有 3 种方法:逐次逼近法 A/D、双积分法 A/D 和并行比较法 A/D。在 DAQ 产品中应用较多的方法是逐次逼近法 A/D。双积分法 A/D 主要应用于速度要求不高,但可靠性和抗干扰型要求较高的场合,如数字万用表等。并行比较法 A/D 主要应用于高速采样,比如数字示波器、数字采样器等应用场合。衡量 A/D 转换器性能好坏主要有两个指标:一是采样分辨率,即 A/D 转换器位数;二是 A/D 转换速度。这二者都与 A/D 转换器的工作原理有关。

2. D/A

DAQ 系统经常需要为被测对象提供激励信号,也就是输出模拟量信号。D/A 转换器就

是将数字量信号转换为模拟量输出的器件。D/A 转换器的主要性能参数是分辨率和线性误差分辨率，分辨率取决于 D/A 转换器的位数，线性误差则刻画了 D/A 转换器的精度。

3. 数字 I/O

在 DAQ 应用中经常需要采集外部设备的工作状态，建立与外部设备的通信，此时就需要用到 DAQ 设备的数字 I/O 功能。一般的数字 I/O 板卡均采用 TTL 电平。需要强调的一点是，对于大功率外部设备的驱动需设计专门的信号处理装置。

4. 定时/计数器

在 DAQ 应用中还经常用到定时/计数功能，比如脉冲周期信号测量、精确时间控制和脉冲信号产生等。定时/计数器的两个主要性能指标是分辨率和时钟频率，分辨率越大，计数器位数越大，计数值也越高。

11.1.2 DAQ 节点的组织与结构

LabVIEW 是通过 DAQ 节点来控制 DAQ 设备完成数据采集的，所有的 DAQ 节点都包含在 Functions 模板 → All Functions 子模板 → NI Measurements 子模板 → Data Acquisition 子模板中，如图 11.1.1 所示。



图 11.1.1 Data Acquisition 子模板

Data Acquisition 子模板共包含 6 个子模板和一个常量，每个子模板分别完成不同的数据采集任务，如图 11.1.2 所示。

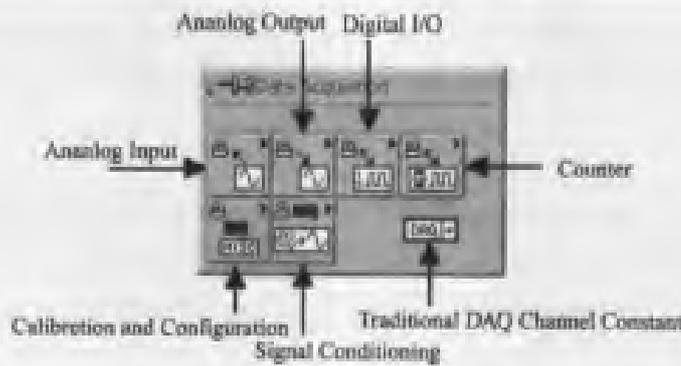


图 11.1.2 Data Acquisition 子模板

各子模板的主要功能如下:

- Analog Input 子模板 完成模拟信号的数据采集, 将外部模拟信号通过 DAQ 设备的 A/D 功能转化为数字信号, 并采集到计算机中。
- Analog Output 子模板 完成模拟信号的输出, 将计算机所产生的数字信号通过 DAQ 设备的 D/A 功能转化为模拟信号, 然后输出。
- Digital I/O 子模板 用于控制 DAQ 设备的数字 I/O 功能。
- Counter 子模板 用于控制 DAQ 设备的计数器功能。
- Calibration and Configuration 子模板 用于校准和配置 DAQ 设备, 并且能够返回 DAQ 设备的配置信息。由于 NI 公司在 DAQ 设备出厂前已经进行了校准, 一般情况下, 用户将很少用到 Calibration and Configuration 子模板中的 DAQ VIs。
- Signal Conditioning 子模板 将从温度传感器、应变片或热电偶中采集到的模拟电压信号转化为相应的应力单位或温度单位。在这些 VIs 中, 可以根据特定的精度需求编辑转换公式。需要注意的一点是, 如果用户编辑了自己的公式, 则需将这些 VIs 另外保存到用户自己的文件夹中。

11.1.3 DAQ VIs 的组织结构

在上述前 4 个 DAQ 子模板中, 其中的 VI 根据功能的不同共分为 4 层, 如表 11.1.1 所示。

表 11.1.1 DAQ VIs 子模板层次表

项 层	相应 DAQ 节点
第三层	中间层 DAQ 节点
第二层	工具 DAQ 节点
底 层	高级 DAQ 节点

Analog Input 子模板就是一个很好的例子, 如图 11.1.3 所示。下面介绍每一层 VIs 的特点。

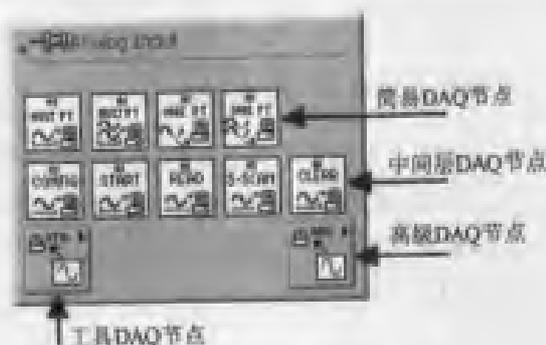


图 11.1.3 Analog Input 子模板

1. 简易 DAQ 节点

简易 DAQ 节点 (Easy VIs) 位于 DAQ 子模板的第一行, 可以执行简单的 DAQ 操作, 例如, 采集一路信号或输出一路信号等。可以将简易 DAQ 节点作为一个单独的 VI 来运行, 也可以将其当作一个 SubVI 来使用。如果需要执行一个基本的 DAQ 操作, 仅运行一个相应的 VI 就可以了, 并不需要其他 DAQ 节点的参与。出现错误时, 简易 DAQ 节点会自动弹出一个对话框, 询问用户是停止 VI 的运行还是忽略该错误继续运行。

实际上, 简易 DAQ 节点是中间层 DAQ 节点的逻辑组合。简易 DAQ 节点提供了一个用于控制简单 DAQ 输入输出的最基本并且很方便的用户接口。对于复杂的 DAQ 应用, 应当使用中间层 DAQ 节点或者高级 DAQ 节点, 以便获得更多的功能和更好的 DAQ 执行方式。

2. 中间层 DAQ 节点

与简易 DAQ 节点相比, 中间层 DAQ 节点 (Intermediate VIs) 具有更多的硬件功能, 用户可以更加有效, 更加灵活地开发应用程序。中间层 DAQ 节点为用户提供了更多的控制手段和错误处理方法, 用户可以检查错误, 甚至可以将错误以错误簇 (Error Cluster) 的形式传递到其他 VIs 中。

3. 工具 DAQ 节点

工具 DAQ 节点 (Utility VIs) 同中间层 DAQ 节点类似, 具有着更多的硬件功能, 可以帮助用户更加有效地开发应用程序。

4. 高级 DAQ 节点

高级 DAQ 节点 (Advanced VIs) 是 DAQ 驱动器最底层的用户接口。用户在开发应用程序时, 很少会用到高级 DAQ 节点, 除非由于简易 DAQ 节点或中间层 DAQ 节点缺少某些参数, 而不能利用它们来实现一些特殊的 DAQ 功能时, 才会用到高级 DAQ 节点。高级 DAQ 节点可以为用户提供更多的 DAQ 驱动器的状态信息。

11.1.4 DAQ 节点常用参数简介

在详细介绍 DAQ 节点的功能及其用法之前, 为使用户更加方便地学习和使用 DAQ 节

点,有必要先介绍一些 LabVIEW 通用的 DAQ 参数的定义。

1. 设备号和任务号 (Device ID 和 Task ID)

在 Analog Input 子模板、Analog Output 子模板、Digital I/O 子模板以及 Counter 子模板中,输入端口 device 是指在 DAQ 配置软件中分配给所用 DAQ 设备的编号,每一个 DAQ 设备都有一个惟一的编号与之对应。在使用工具 DAQ 节点配置 DAQ 设备时,这个编号可以由用户指定。输出参数 Task ID 是系统给特定的 I/O 操作分配的一个惟一的标识号,贯穿于以后的 DAQ 操作的始终。

2. 通道 (Channels)

Analog Input 子模板和 Analog Output 子模板中的 DAQ 节点有一个输入端口叫通道 (Channels),用户通过该端口指定用于 DAQ 读写操作的通道。在 Digital I/O VIs 中也有一个类似的参数,叫做数字通道列表 (digital channel list),而在 Counter VIs 中,这个参数叫做计数器列表 (counter list)。

Channels 中所有指定的通道会形成一个通道组 (Group),VIs 会按照 Channels 中所列出的通道顺序进行采集或输出数据的 DAQ 操作。

3. 通道命名 (Channel Name Addressing)

要在 LabVIEW 中应用 DAQ 设备,必须事先对 DAQ 硬件进行配置。为了让 DAQ 设备的 I/O 通道的功能和意义更加直观地为用户所理解,用每个通道所对应的实际物理参数意义或名称来命名通道是一个理想的办法。在 LabVIEW 中配置 DAQ 设备的 I/O 通道时,可以在 Channels 中输入一定物理意义的名称来确定通道的地址。在简易 DAQ 节点中,Channels 参数就是一个字符串数组或者是一个字符串,如图 11.1.4 所示。用户可以在数组的每一个元素中填写一个通道的名称,也可以将数个通道的名称填写到一个元素中,但通道名称之间要有一个逗号隔开。在填写通道名称时特别要注意名称的拼写,要与在预先定义的通道名称一致。

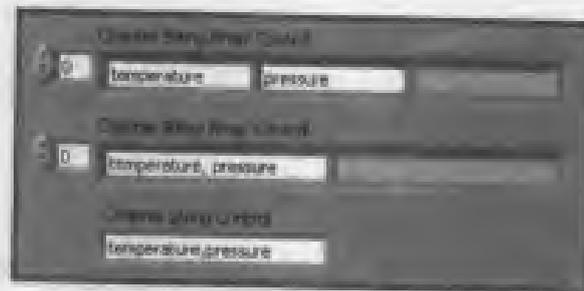


图 11.1.4 通道命名

用户在使用通道名称控制 DAQ 设备时,就不需要再连接 device, input limits 以及 input config 这些输入参数了, LabVIEW 会按照在 DAQ Channel Wizard 中的通道配置自动来配置这些参数。

4. 通道编号命名 (Channel Number Addressing)

如果用户不使用通道名称来确定通道的地址，那么还可以在 channels 中使用通道编号来确定通道的地址，如图 11.1.5 所示。

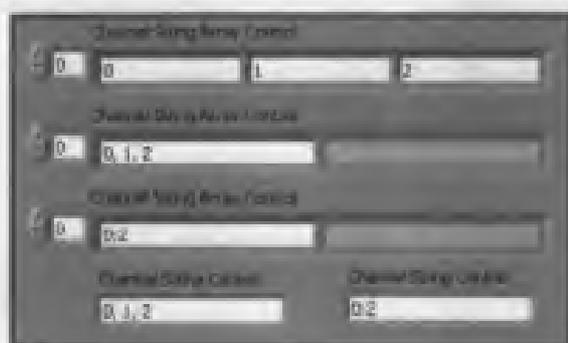


图 11.1.5 通道编号命名

可以将每一个通道编号都作为一个数组中的元素，也可以将数个通道编号填入一个数组元素中，编号之间用逗号隔开；可以在一个数组元素中指定通道的范围，例如 0:2，表示通道 0, 1, 2。

5. I/O 范围设置 (Limit Settings)

Limit Settings 是指 DAQ 卡所采集或输出的模拟信号的最大/最小值。请注意，在使用模拟输入功能时，用户设定的最大最小值必须在 DAQ 设备允许的范围之内。一对最大最小值组成一个簇，多个这样的簇形成一个簇的数组，每一个通道对应一个簇，这样，用户就可以为每一个模拟输入或模拟输出通道单独指定最大最小值了，如图 11.1.6 所示。



图 11.1.6 I/O 范围设置

按照图 11.1.6 中的通道及 limit 设定，通道 0, 1, 2, 3 的范围是-10~10，通道 4 的范围是-5~5，通道 5, 6, 7 的范围是 0~1。

但是，在图 11.1.7 所示的设定中，仅为通道 0, 1, 2, 3 指定了范围-10~10，此时，LabVIEW 将自动为其余的通道 4, 5, 6, 7 设定同样的范围-10~10。



图 11.1.7 I/O 范围设置

在模拟信号的数据采集应用中, 用户不但需要设定信号的范围, 还要设定 DAQ 设备的极性和范围。一个单极性的范围只包含正值或只包含负值, 而双极性范围可以同时包含正值和负值。用户需根据自己的需要来设定 DAQ 设备的极性。

6. 采集数据的构成

当用户在多个通道进行多次数据采集时, 采集到的数据以 2D 数组的形式返回。在 LabVIEW 中, 用户可以用两种方式来组织 2D 数组中的数据。

第一种方式是用数组中的行 (row) 来组织数据。假如数组中包含了来自模拟输入通道中的数据, 那么, 数组中的一行就代表一个通道中的数据。这种方式通常称为行顺方式 (row major order)。当用户用一组嵌套 For 循环来产生一组数据时, 内层的 For 循环每循环一次就产生 2D 数组中的一行数据。用这种方式构成的 2D 数组如图 11.1.8 所示。



图 11.1.8 行顺方式组织数据

第二种方式是通过 2D 数组中的列 (column) 来组织数据。LabVIEW 中的 Analog Input VIs 就是采用这种方式来组织数据的。节点把从一个通道采集来的数据放到 2D 数组的一列中, 这种组织数据的方式通常称之为列顺方式 (column major order)。此时 2D 数组的构成如图 11.1.9 所示。



图 11.1.9 列顺方式组织数据

注意, 在图 11.1.9 中出现了一个术语 scan, 称为扫描。一次扫描是指从用户指定的一

组通道按顺序进行一次数据采集。

假如需要从这个 2D 数组中取出其中某一个通道的数据,按照 5.3 节介绍的方法将数组中相对应的一列数据取出即可,如图 11.1.10 所示。

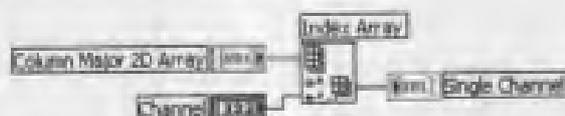


图 11.1.10 从 2D 数组中取出其中某一个通道的数据

7. 扫描次数 (Number of Scans to Acquire)

扫描次数是指在用户指定的一组通道进行数据采集的次数。

8. 采样点数 (Number of Samples)

采样点数是指一个通道采样点的个数。

9. 扫描速率 (Scan Rate)

扫描速率是指每秒完成一组指定通道数据采集的次数,它决定了在所有的通道中在一定时间内所进行数据采集次数的总和。

11.2 DAQ 设备的安装与配置

在了解了一些 DAQ 的基本概念、常用参数的定义及有关术语之后,用户在使用 LabVIEW 进行 DAQ 编程应用之前,需要首先安装 DAQ 设备,并进行一些必要的配置。

本节将以 PCI-1200 数据采集卡为例,介绍如何在系统中安装和配置 DAQ 设备,通常情况下,LabVIEW 安装和配置 DAQ 板卡的主要步骤如图 11.2.1 所示。



图 11.2.1 安装和配置 DAQ 板卡的主要步骤

11.2.1 安装 PCI-1200 数据采集卡

PCI-1200 数据采集卡是一块基于 32 位 PCI 总线的多功能数据采集控制卡, 支持 DMA 方式和双缓冲区模式, 保证了实时信号的不间断采集与存储。它支持单极和双极性模拟信号输入, 信号输入范围分别为 0~10V 和 -5~5V。提供 16 路单端 / 8 路差动模拟输入通道、2 路独立的 D/A 输出通道、24 线的 TTL 型数字 I/O、3 个 16 位的定时计数器等多种功能。这些功能使得用户不仅可以用该卡设计虚拟示波器, 还可以设计虚拟函数发生器或虚拟计数器, 做到一卡多用。

将 PCI-1200 数据采集卡插到计算机主板上的一个空闲的 PCI 插槽中, 接好各种附件, 其驱动程序就是 NI-DAQ。附件包括一条 50 芯的数据线, 一个型号为 CB-50LP 的转接板, 转接板直接与外部信号连接。

CB-50LP 转接板的引脚定义如图 11.2.2 所示。

在 LabVIEW 中, DAQ 设备与 DAQ 节点以及用户开发的 VI 的层次关系如图 11.2.3 所示。

ACH0	1	2	ACH1
ACH2	3	4	ACH3
ACH4	5	6	ACH5
ACH6	7	8	ACH7
AISENSE/AGND	9	10	DAC0OUT
AGND	11	12	DAC1OUT
DGND	13	14	PA0
PA1	15	16	PA2
PA3	17	18	PA4
PA5	19	20	PA6
PA7	21	22	PB0
PB1	23	24	PB2
PB3	25	26	PB4
PB5	27	28	PB6
PB7	29	30	PC0
PC1	31	32	PC2
PC3	33	34	PC4
PC5	35	36	PC6
PC7	37	38	EXTTRIG
EXTUPDATE*	39	40	EXTCONV*
OUTB0	41	42	GATB0
OUTB1	43	44	GATB1
CLKB1	45	46	OUTB2
GATB2	47	48	CLKB2
+5V	49	50	DGND

图 11.2.2 CB-50LP 转接板的引脚定义图

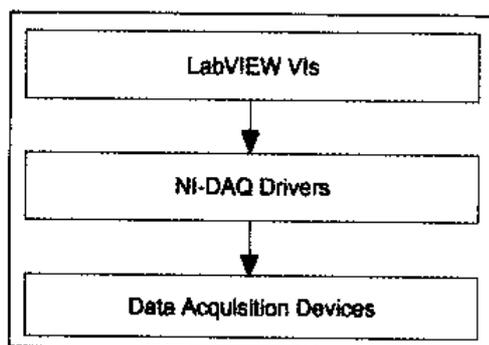


图 11.2.3 DAQ 设备与 DAQ 节点以及 VI 的层次关系图

11.2.2 配置 PCI-1200 数据采集卡

在安装 NI-DAQ 软件时, 系统会自动安装一个名为 Measurement & Automation Explorer 的软件, 简称 MAX, 该软件用于管理和配置硬件设备。下面介绍如何用 MAX 配置 PCI-1200 数据采集卡。

配置 PCI-1200 数据采集卡的步骤如下。

第一步，运行 MAX，在 MAX 窗口左侧的设备管理树的 Devices and Interfaces 选项中，选择 PCI-1200 数据采集卡，如图 11.2.4 所示。

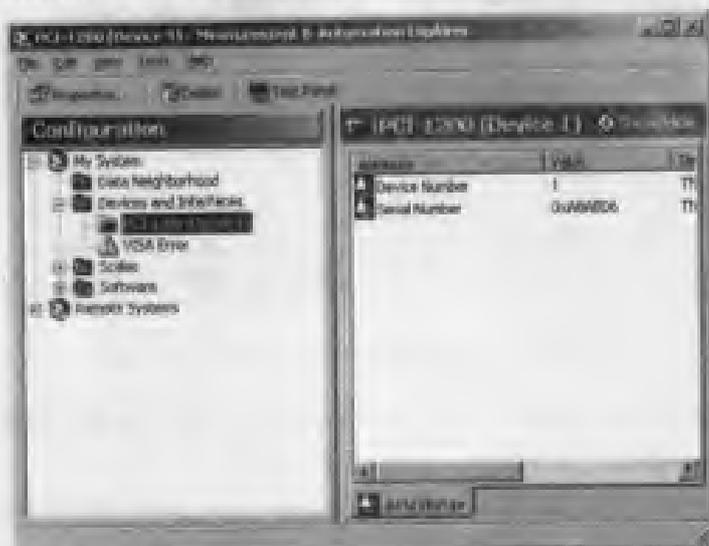


图 11.2.4 Measurement & Automation Explorer 主窗口

第二步，在 PCI-1200 数据采集的右键弹出选单中选择 Properties...，弹出 PCI-1200 数据采集卡的配置对话框，对话框共分 5 部分：System, AI, AO, Accessory 和 OPC。在这个对话框中可以完成对 DAQ 设备的配置，如图 11.2.5 所示。

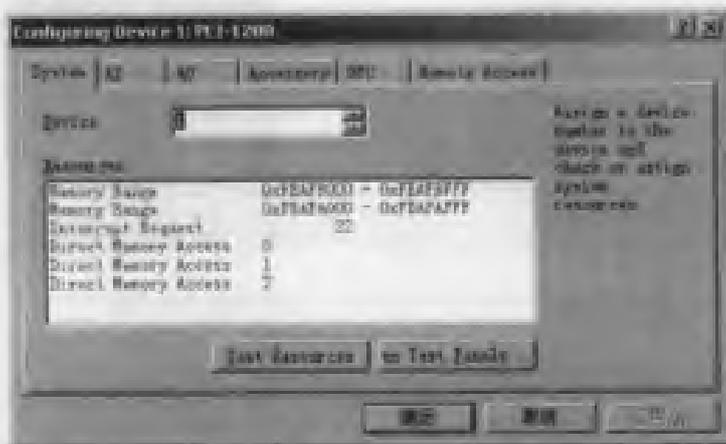


图 11.2.5 Configuring Device 对话框

第三步，设置 DAQ 设备在系统中的设备 (Device) 编号。在 Configuring Device 对话框的 System 页面中将 Device 属性值设为 1，如图 11.2.5 所示。另外，该页面会将 DAQ 设备在 Windows 中所占用的资源列出，例如中断号、内存范围等。

第四步，设置模拟输入 (AI) 属性。在 Configuring Device 对话框的 AI 页面中，将 Polarity 属性值设为 -5.0V ~ +5.0V，将 Mode 属性值设为 Differential (差分输入)，如图 11.2.6 所示。

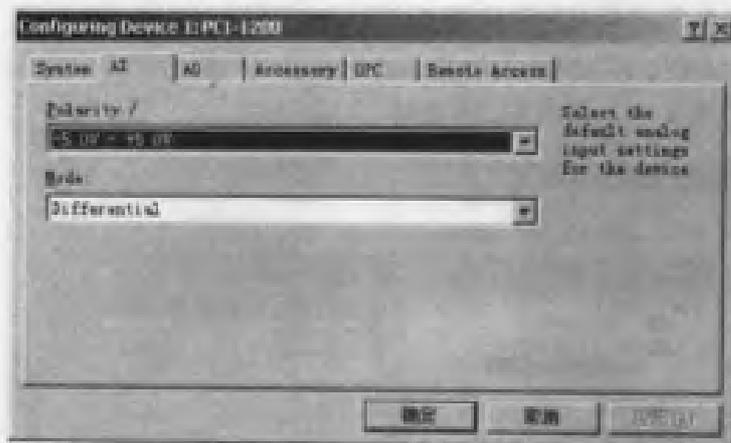


图 11.2.6 Configuring Device 对话框的 AI 页面

第五步，设置模拟输出（AO）属性。在 Configuring Device 对话框的 AO 页面中，将 Polarity 属性值设为 Bipolar（双极性），如图 11.2.7 所示。

第六步，设置附件（Accessory）。在 Configuring Device 对话框的 Accessory 页面中，将 Accessory 属性值设为 CB-50LP，如图 11.2.8 所示。

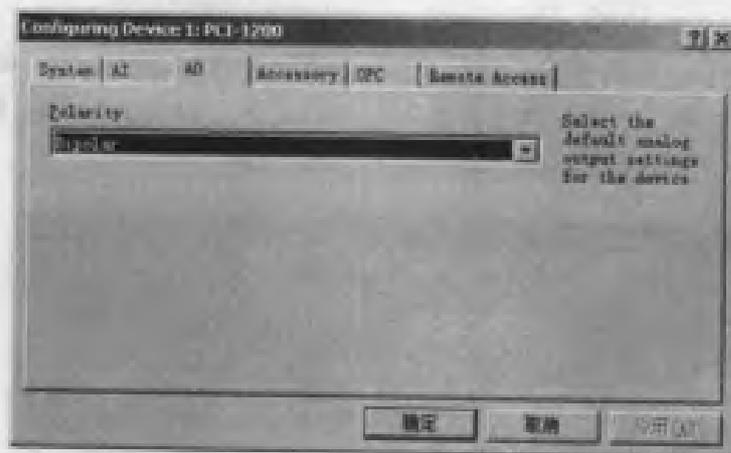


图 11.2.7 Configuring Device 对话框的 AO 页面

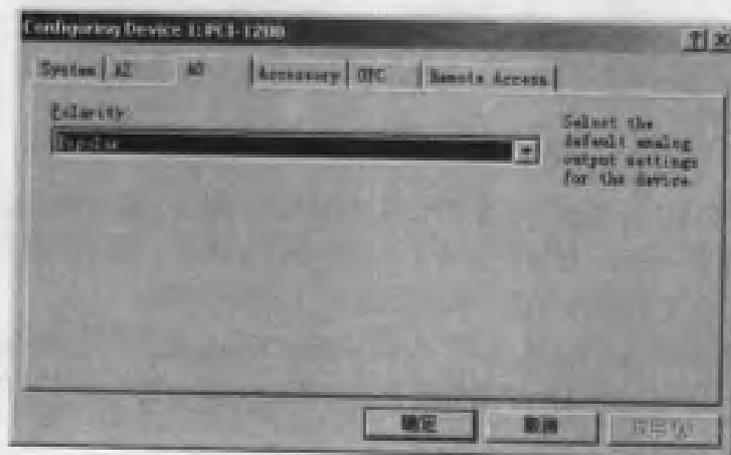


图 11.2.8 Configuring Device 对话框的 Accessory 页面

第七步, 设置过程控制 OLE (OPC)。在 Configuring Device 对话框的 OPC 页面中, 将 OPC 属性值设为 Disabled, 如图 11.2.9 所示。

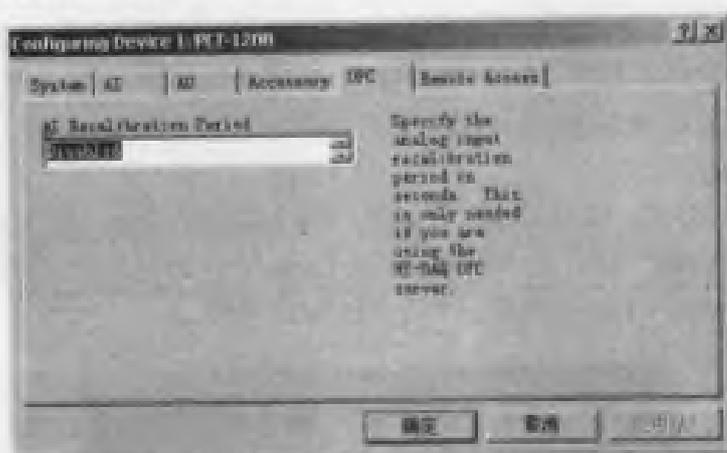


图 11.2.9 Configuring Device 对话框的 OPC 页面

在完成了上述属性的设定之后, 单击“确定”按钮。

若属性配制成功, PCI-1200 数据采集卡正常工作, 单击在 Configuring Device 对话框的 System 页面中的“Test Resources”按钮, 系统就会弹出一个对话框, 告知用户 DAQ 设备通过了测试, 如图 11.2.10 所示。



图 11.2.10 测试结果对话框

至此, 就完成了 PCI-1200 数据采集卡的配置。在 PCI-1200 数据采集的右键弹出选单中选择 Test, 弹出 Test Panel 窗口, 在该窗口中可以通过真实信号测试测试 PCI-1200 数据采集卡的全部功能, 如图 11.2.11 所示。

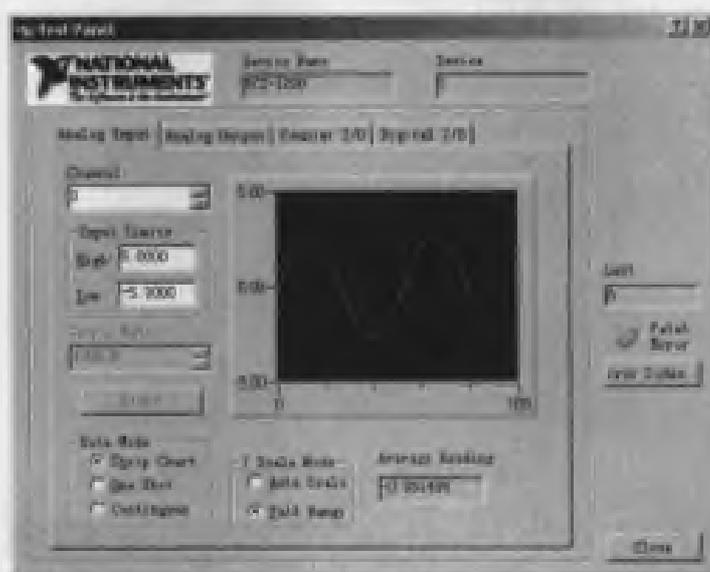


图 11.2.11 Test Panel 窗口

11.2.3 配置数据采集虚拟通道

在使用 DAQ 设备的模拟输入或数字 I/O 功能时,可以配置设备的虚拟通道,在 LabVIEW 中进行 DAQ 编程时,直接指定这些虚拟通道的名称,就可以控制这些通道完成数据采集。下面以配置 PCI-1200 卡的模拟输入通道为例,介绍在 MAX 中如何进行虚拟通道配置。操作步骤如下。

第一步,运行 MAX,在 MAX 窗口左侧的设备管理树的 Data Neighborhood 选项的右键弹出选单中选择 Creat New..., 弹出一个名为 Creat New 的对话框,如图 11.2.12 所示。选中对话框中的 Virtual Channel 选项,单击“Finish”按钮。



图 11.2.12 Creat New 对话框

第二步,在随后弹出的 Create New Channel 对话框中将通道的类型设置为 Analog Input,单击“下一步”按钮,如图 11.2.13 所示。



图 11.2.13 Create New Channel 对话框

第三步,在随后弹出的 Enter Channel Name and Description 对话框中,将通道名称设置为 Channel01,并填写适当的通道描述,单击“下一步”按钮,如图 11.2.14 所示。

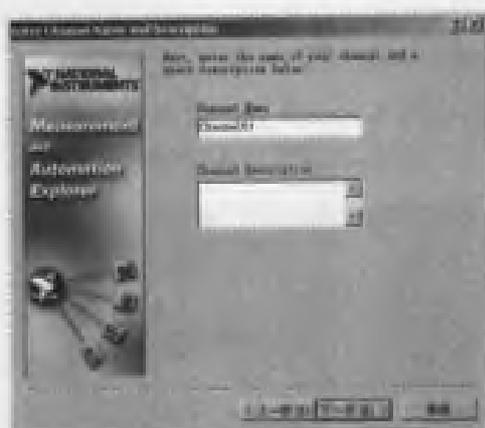


图 11.2.14 Enter Channel Name and Description 对话框

通道名称由用户任意指定，在以后的 DAQ 编程应用中，就可以用这个名称直接控制这个通道。通道描述在 DAQ 编程中并无用处，只是为用户查看通道设置提供方便的信息。

第四步，在随后弹出的 Channel Wizard 对话框中，将输入信号的类型设为 Voltage，单击“下一步”按钮，如图 11.2.15 所示。

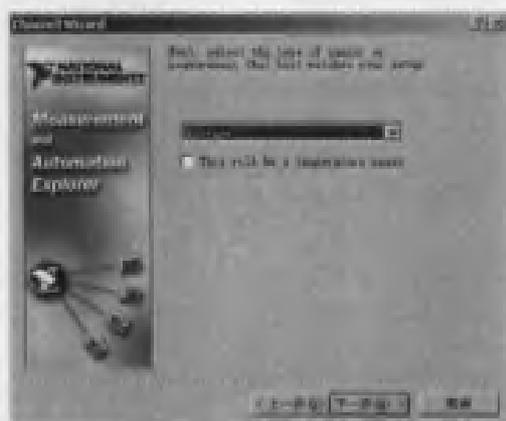


图 11.2.15 Channel Wizard 对话框

第五步，在随后弹出的第二个 Channel Wizard 对话框中，将单位 (Unit) 设为 V，输入范围 (Range) 设为 -5V~5V，单击“下一步”按钮，如图 11.2.16 所示。

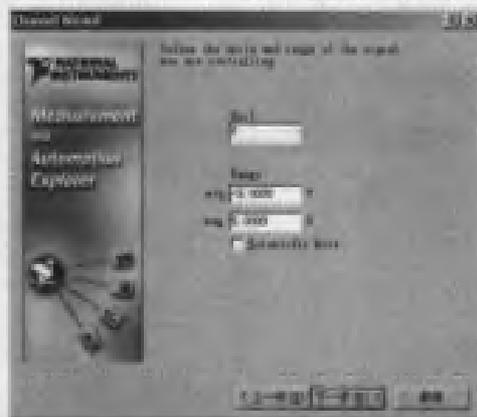


图 11.2.16 第二个 Channel Wizard 对话框

注意,在这个对话框中的输入范围设定不能超过在 11.2.2 节中配置 PCI-1200 数据采集卡的第四步所设定的输入范围。

第六步,在随后弹出的第三个 Channel Wizard 对话框中,将信号的缩放比例因子设置为 No Scaling,单击“下一步”按钮,如图 11.2.17 所示。



图 11.2.17 第三个 Channel Wizard 对话框

第七步,在随后弹出的第四个 Channel Wizard 对话框中,将 DAQ 硬件指定为 Dev1:PCI-1200,将通道编号设定为 0,单击“完成”按钮,如图 11.2.18 所示。

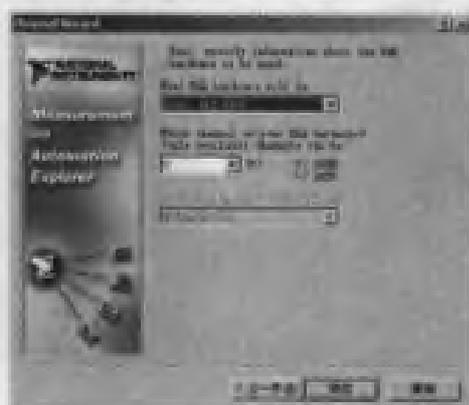


图 11.2.18 第四个 Channel Wizard 对话框

至此,就完成了 PCI-1200 数据采集卡的一个模拟输入通道的设置,该通道出现在 MAX 窗口中,如图 11.2.19 所示。

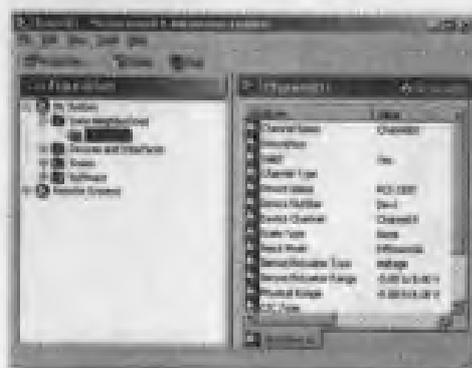


图 11.2.19 MAX 窗口

在MAX窗口的左侧的Channel01选项的右键弹出选单中选择Test,弹出Virtual Channel Test Panels窗口,可用于测试该通道是否正常,如图11.2.20所示。



图 11.2.20 Virtual Channel Test Panels 窗口

另外,若需要修改一个虚拟通道的配置,可以在该通道的右键弹出选单中选择Properties...,弹出虚拟通道的配置对话框,如图11.2.21所示,在该对话框中可以修改虚拟通道的各种配置参数。



图 11.2.21 虚拟通道配置对话框

至此,就完成了对系统中DAQ设备的安装和配置,下面就可以在此基础上编制DAQ应用软件了。

11.3 DAQ 编程

在了解了一些DAQ参数、术语以及DAQ软硬件的安装及配置之后,就可以进行DAQ编程应用了。本节主要介绍Analog Input子模板中的DAQ节点,并以利用这些节点控制PCI-1200DAQ卡进行模拟信号采集为例,介绍如何进行DAQ编程。

11.3.1 简易DAQ编程

简易DAQ编程采用简易Analog Input节点,简易Analog Input节点共有4个:AI Acquire

Waveform.vi、AI Acquire Waveforms.vi、AI Sample Channel.vi 和 AI Sample Channel.vi，位于 Functions 模板→All Functions 子模板→NI Measurements 子模板→Data Acquisition 子模板→Analog Input 子模板中，简易 Analog Input 节点可以实现简单的模拟输入功能。

下面介绍这 4 个节点的用法及相关实例。

1. AI Acquire Waveform.vi

按照指定的扫描频率 (sample rate 端口)，控制一个指定的通道 (channel 端口) 采集指定数目 (number of samples 端口) 的数据，节点的图标及其端口定义如图 11.3.1 所示。

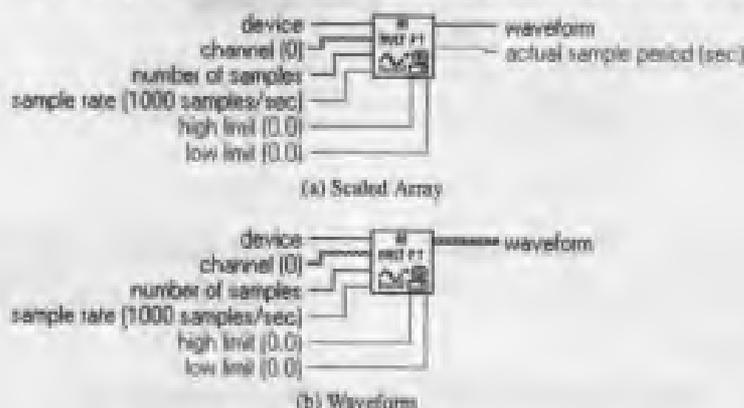


图 11.3.1 AI Acquire Waveform.vi 节点的图标及其端口定义

从图 11.3.1 可以看出，节点的输出端口 waveform 是一个多态性的端口，其数据类型可以是一维数组，也可以是波形数据，在节点图标的右键弹出菜单中选择 Visible Items→Polymorphic VI Selector，可以将节点的 Polymorphic VI Selector 显示出来，如图 11.3.2 所示。

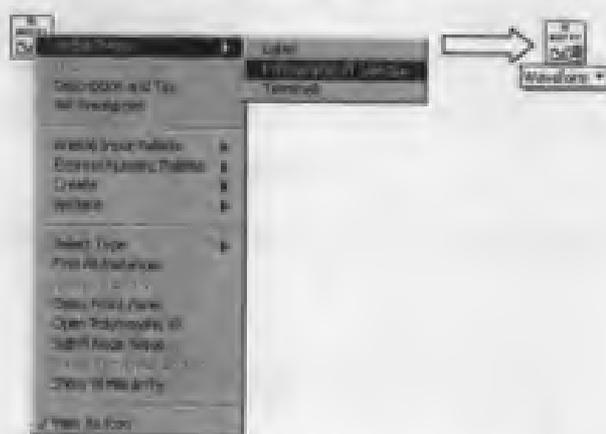


图 11.3.2 显示节点的 Polymorphic VI Selector

用鼠标单击节点的 Polymorphic VI Selector，会出现一个下拉选单，在选单中可以选择节点端口的数据类型，如图 11.3.3 所示。若选择 Automatic，则节点端口会根据连接到该端口上的数据的数据类型自动选择该端口上的数据类型，其他的 DAQ 节点的端口也有多态性，其数



图 11.3.3 选择节点端口的数据类型

据类型选择方法与此相同。

例 11.3.1 利用 AI Acquire Waveform.vi 节点实现数据采集。

VI 的前面板和框图程序如图 11.3.4 所示。

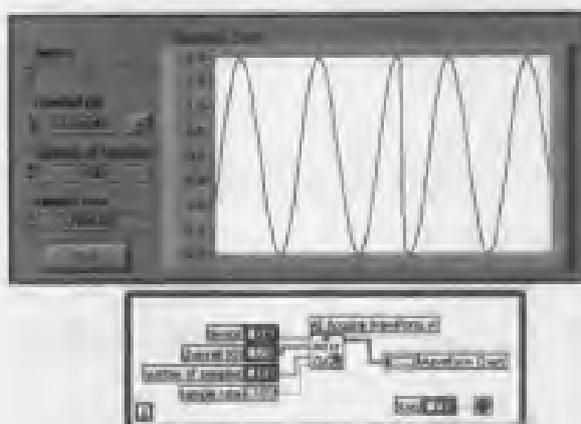


图 11.3.4 例 11.3.1 的前面板及其框图程序

从图 11.3.4 中可以看出, AI Acquire Waveform.vi 节点位于一个 While 循环中, 每循环一次, 采集一次数据, 两次循环所返回的数据并不连续, 即两次数据采集之间丢失了数据。这是由于该节点在每一次循环都执行了四步操作: 配置 DAQ 设备、启动数据采集、读出数据、终止 DAQ 操作。这样在每一次的终止数据采集与启动数据采集之间有一个时间差, 这就是造成数据丢失的原因。

所以, 简易 Analog Input 节点只能用于对数据采集要求不高的数据监控场合, 不能用于需要对信号进行连续采集与数据记录的应用场合, 利用中间层 Analog Input 节点可以实现对信号进行连续采集与数据记录。

2. AI Acquire Waveforms.vi

按照指定的扫描频率, 控制指定的通道 (多个) 采集指定数目的数据, 节点的图标及其端口定义如图 11.3.5 所示。

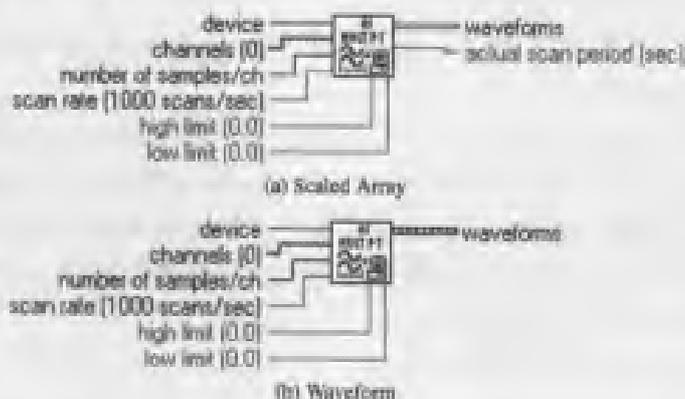


图 11.3.5 AI Acquire Waveforms.vi 节点的图标及其端口定义

3. AI Sample Channel.vi

控制一个指定的通道采集一个点的数据，节点的图标及其端口定义如图 11.3.6 所示。

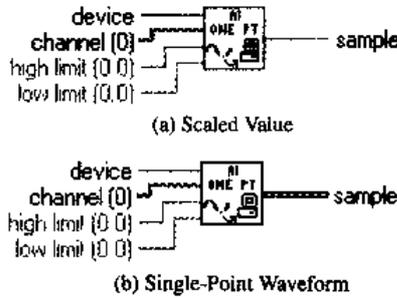


图 11.3.6 AI Sample Channel.vi 节点的图标及其端口定义

4. AI Sample Channels.vi

控制每一个指定的通道（多个）采集一个点的数据，节点的图标及其端口定义如图 11.3.7 所示。

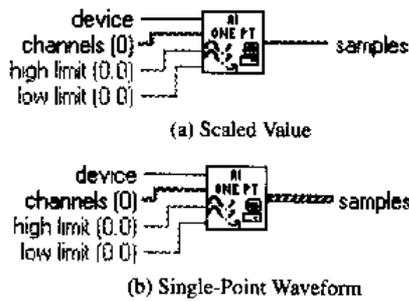


图 11.3.7 AI Sample Channels.vi 节点的图标及其端口定义

11.3.2 扩展 DAQ 编程

扩展 DAQ 编程采用中间层 Analog Input 节点和工具 Analog Input 节点，下面介绍这两类节点的使用方法和编程实例。

中间层 Analog Input 节点共有 5 个：AI Config.vi、AI Start.vi、AI Read.vi、AI Single Scan.vi 和 AI Clear.vi，位于 Functions 模板→All Functions 子模板→NI Measurements 子模板→Data Acquisition 子模板→Analog Input 子模板中，中间层 Analog Input 节点建立在高级 Analog Input 节点的基础上，可以对 DAQ 设备进行一些基本的配置，并控制其完成数据采集，可以使用这些节点进行灵活的编程。

下面介绍这 5 个节点的用法及相关实例。

1. AI Config.vi

设置用于数据采集的通道和 DAQ 设备的内部缓存的大小，节点的图标及其端口定义如图 11.3.8 所示。

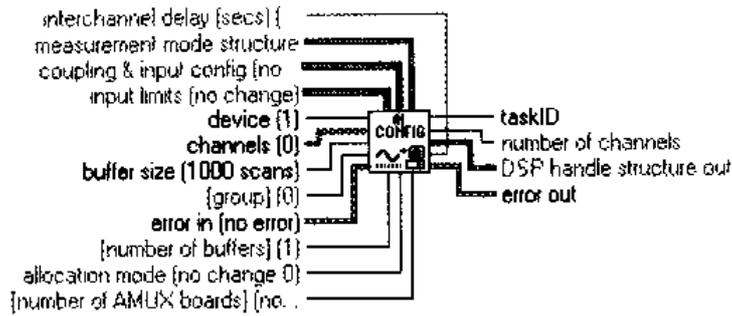


图 11.3.8 AI Config.vi 节点的图标及其端口定义

2. AI Start.vi

设置采样点数和扫描频率，并启动 DAQ 设备的带有缓存的数据采集操作，节点的图标及其端口定义如图 11.3.9 所示。

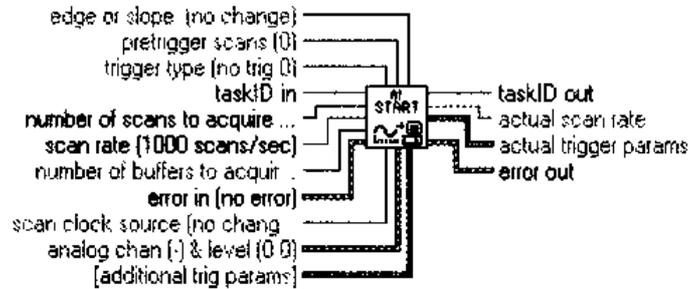


图 11.3.9 AI Start.vi 节点的图标及其端口定义

3. AI Read.vi

从 DAQ 设备的内部缓存中读出指定数目的数据，并返回，节点的图标及其端口定义如图 11.3.10 所示。节点返回的数据格式可以设置为：Binary Array、Scaled and Binary Arrays、Scaled Array 或 Waveform。

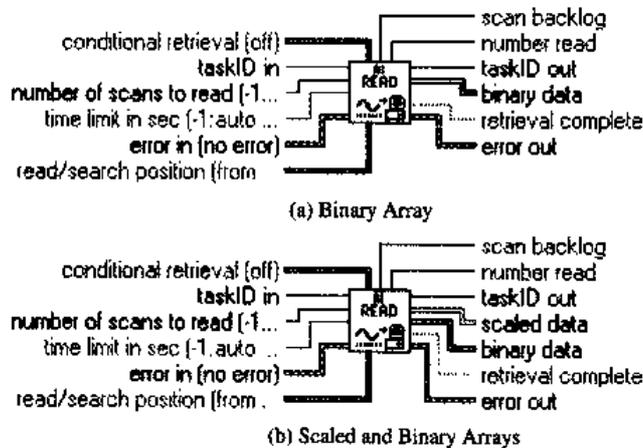


图 11.3.10 AI Read.vi 节点的图标及其端口定义

4. AI Single Scan.vi

从 DAQ 设备的模拟输入通道中直接返回一次扫描的数据, 节点的图标及其端口定义如图 11.3.11 所示。节点返回的数据格式可以设置为: Binary Array、Scaled and Binary Arrays、Scaled Array 或 Single-Point Waveform。

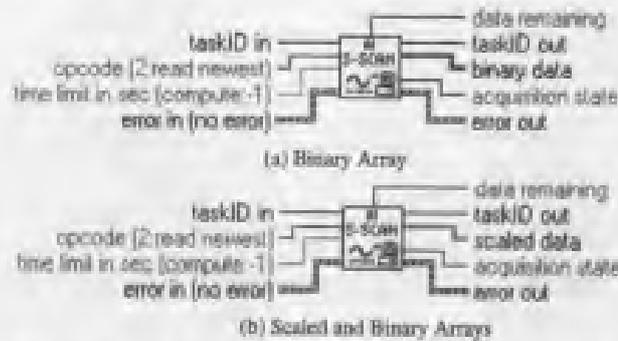


图 11.3.11 AI Single Scan.vi 节点的图标及其端口定义

5. AI Clear.vi

终止 DAQ 设备的模拟输入操作, 并删除相关配置, 释放 DAQ 设备的相关资源, 节点的图标及其端口定义如图 11.3.12 所示。



图 11.3.12 AI Clear.vi 节点的图标及其端口定义

利用上述中间层 Analog Input 节点可以实现对信号进行连续采集与数据记录。请看下面的实例。

例 11.3.2 利用中间层 Analog Input 节点对信号进行连续采集。

VI 的前面板和框图程序如图 11.3.13 所示。

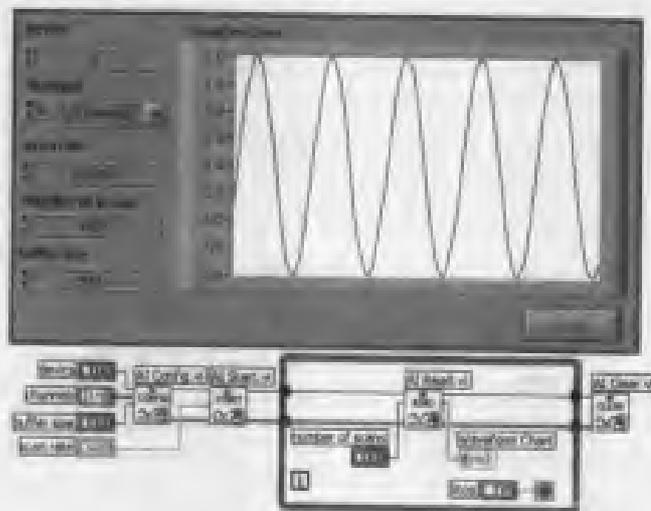


图 11.3.13 例 11.3.2 的前面板及其框图程序

从图 11.3.13 可以看出, 连续的数据采集操作共分为四步:

① 利用 AI Config.vi 节点设置采集通道、缓存大小, 这一步很重要, 只有有效地利用缓存, 才能够实现连续的数据采集。

② 利用 AI Start.vi 节点设置数据扫描频率, 并启动 DAQ 设备进行数据采集, 此时 DAQ 设备就进行连续不停的数据采集操作, 并把采集到的数据实时的写入 DAQ 设备的内部缓存中。

③ 在一个 While 循环中利用 AI Read.vi 节点将缓存中的数据及时读出。

④ 利用 AI Clear.vi 节点终止 DAQ 设备的数据采集操作。

例 11.3.2 使用缓存技术实现了对信号的连续采集, 解决了利用简易 Analog Input 节点进行数据采集时存在的问题。另外, 在很多情况下, 用户还需要将采集到的数据进行实时的记录, 利用缓存技术和高速磁盘流技术可以实现对信号的连续实时记录。请看下面的实例

例 11.3.3 利用中间层 Analog Input 节点对信号进行连续实时记录。

VI 的框图程序如图 11.3.14 所示。

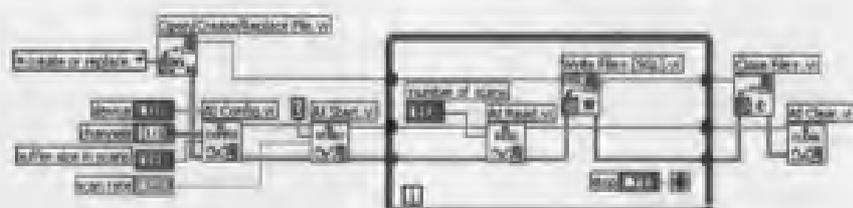


图 11.3.14 例 11.3.3 的框图程序

从图 11.3.14 可以看出, 连续的数据记录操作共分为七步:

① 利用 Open/Create/Replace File.vi 节点打开或创建一个字节流数据文件。

② 利用 AI Config.vi 节点设置采集通道、缓存大小。

③ 利用 AI Start.vi 节点设置数据扫描频率, 并启动 DAQ 设备进行数据采集。

④ 在一个 While 循环中利用 AI Read.vi 节点将采集到的数据不停地从 DAQ 设备的缓存中读出。

⑤ 利用 Write File+[SGL].vi 节点将读出的数据写入数据文件中。

⑥ 利用 Close File+.vi 关闭数据文件。

⑦ 利用 AI Clear.vi 节点模块终止 DAQ 设备的数据采集操作。

从上面的实例可以看出, 使用中间层 Analog Input 节点编程较为复杂, 而利用工具 Analog Input 节点可以大大降低编程的复杂度, 工具 Analog Input 节点建立在中间层 Analog Input 节点的基础上, 对中间层 Analog Input 节点进行了一定的封装, 只需一个节点就可以完成对 DAQ 设备进行一些基本的配置, 并控制其完成数据采集, 使用较为简单。

工具 Analog Input 节点共有 3 个: AI Read One Scan.vi、AI Waveform Scan.vi 和 AI Continuous Scan.vi, 位于 Functions 模板 → All Functions 子模板 → NI Measurements 子模板 → Data Acquisition 子模板 → Analog Input 子模板 → Analog Input Utilities 子模板中。

下面介绍这 3 个节点的用法及相关实例。

6. AI Read One Scan.vi

控制指定的通道完成数据采集并返回采集的数据，节点的图标及其端口定义如图 11.3.15 所示。节点返回的数据格式可以设置为：Binary Array、Scaled and Binary Arrays、Scaled Array 或 Single-Point Waveform。

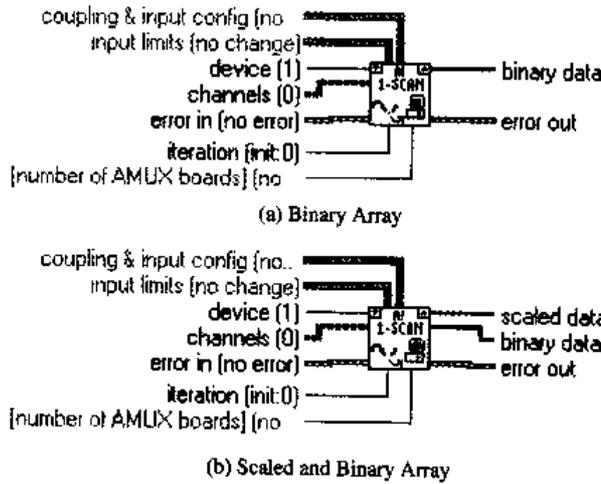


图 11.3.15 AI Read One Scan.vi 节点的图标及其端口定义

7. AI Waveform Scan.vi

设置采样点数、扫描频率并控制指定的通道完成数据采集，然后返回所有的采集数据，节点的图标及其端口定义如图 11.3.16 所示。节点返回的数据格式可以设置为 Scaled Array 或 Waveform。

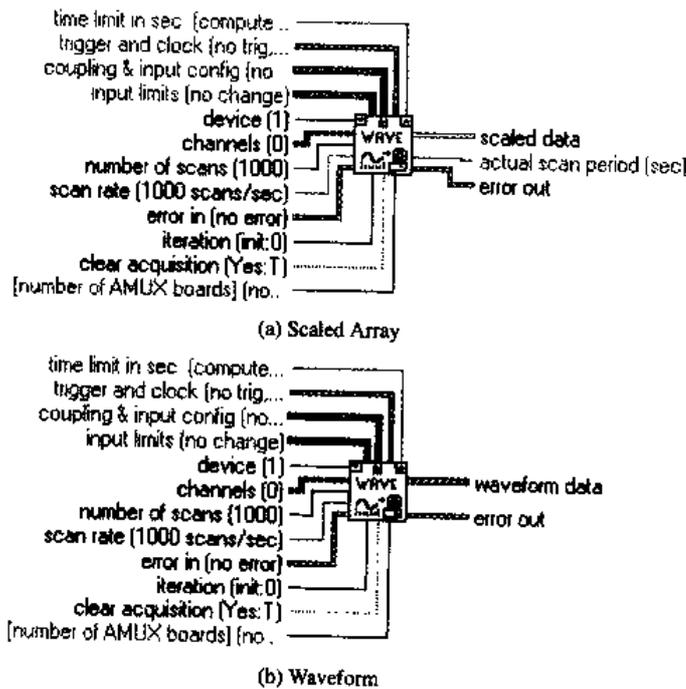


图 11.3.16 AI Waveform Scan.vi 节点的图标及其端口定义

8. AI Continuous Scan.vi

设置采样点数、扫描频率并控制一组指定的通道进行连续的数据采集，并将采集到的数据实时存放到 DAQ 设备的内部缓存中，节点的图标及其端口定义如图 11.3.17 所示。节点返回的数据格式可以设置为 Scaled Array 或 Waveform。

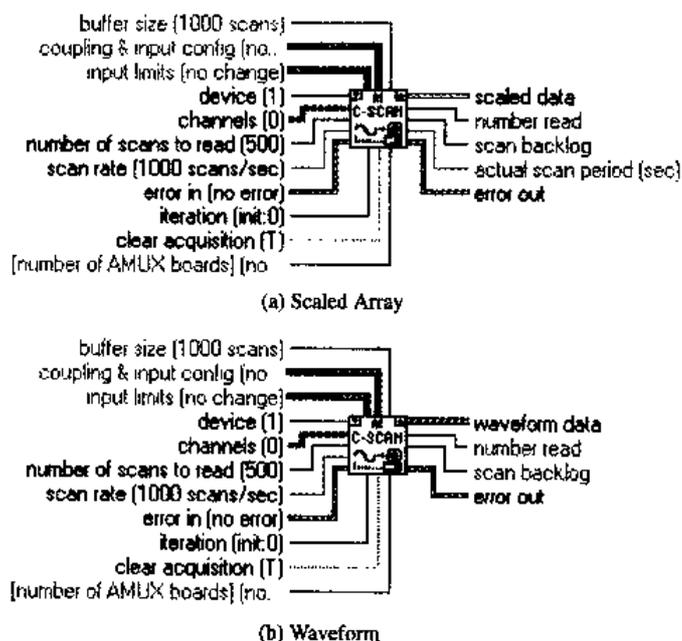


图 11.3.17 AI Continuous Scan.vi 节点的图标及其端口定义

从图 11.3.17 可以看出，AI Read One Scan.vi 节点比 AI Acquire Waveform.vi 节点多了一个名为 iteration 的端口和一个名为 clear acquisition 的端口，这两个端口很重要，可以用于控制节点实现下列操作：

- 当 iteration 端口的输入值为 0 时，调用 AI Config.vi 节点配置 DAQ 设备，调用 AI Start.vi 节点启动数据采集；
- 当 iteration 端口的输入值大于 0 时，调用 AI Read.vi 节点读出数据操作；
- 当 clear acquisition 端口的输入值为 True 时，调用 AI Clear.vi 节点终止 DAQ 设备的数据采集操作。

这样，将 AI Read One Scan.vi 节点放到一个 While 循环中，并将 While 循环的重复端口  连接到 iteration 端口上，就可以实现对信号的连续采集。请看下面的实例。

例 11.3.4 利用 AI Continuous Scan.vi 节点进行连续数据采集。

VI 的前面板和框图程序如图 11.3.18 所示。

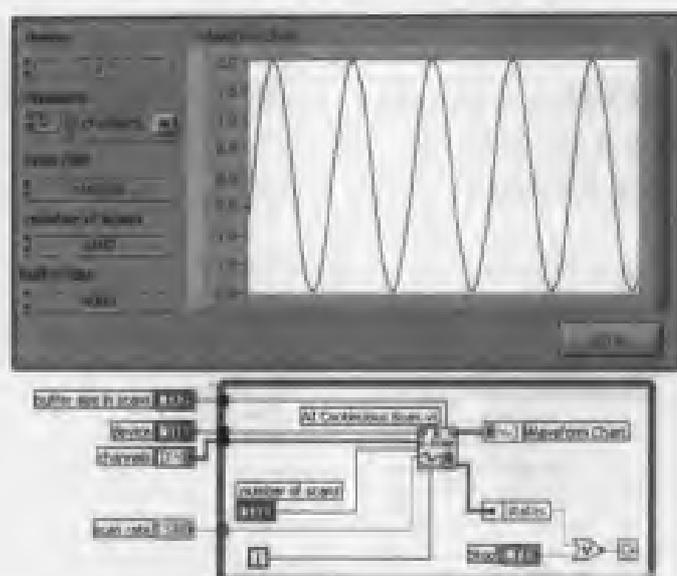


图 11.3.18 例 11.3.4 的前面板及其框图程序

从图 11.3.18 可以看出, 例 11.3.4 只利用了一个节点就实现了例 11.3.2 的功能, 大大简化了程序的复杂程度和减少了用户编程的时间。

11.3.3 高级 DAQ 编程

高级 DAQ 编程采用高级 Analog Input 节点, 高级 Analog Input 节点共有 9 个: AI Group Config.vi、AI Hardware Config.vi、AI Clock Config.vi、AI Trigger Config.vi、AI Buffer Config.vi、AI Control.vi、AI Buffer Read.vi、AI Parameter.vi 和 AI SingleScan.vi, 位于 Functions 模板 → All Functions 子模板 → NI Measurements 子模板 → Data Acquisition 子模板 → Analog Input 子模板 → Advanced Analog Input 子模板中, 高级 Analog Input 节点是建立其他节点的基础, 可以从底层配置并控制 DAQ 设备, 但是需要进行复杂的编程, 一般情况下不需要使用这些节点进行 DAQ 编程, 只需要利用中间层 Analog Input 节点或工具 Analog Input 节点就可以了。下面介绍这 9 个节点的法。

1. AI Group Config.vi

定义通道组, 并为其分配通道, 节点的图标及其端口定义如图 11.3.19 所示。



图 11.3.19 AI Group Config.vi 节点的图标及其端口定义

2. AI Hardware Config.vi

设置 DAQ 设备的输入范围的上下限, 信号的极性, 增益输入模式以及信号耦合方式, 节点的图标及其端口定义如图 11.3.20 所示。

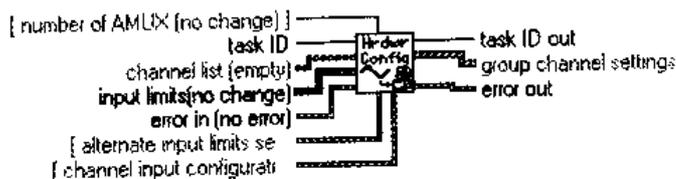


图 11.3.20 AI Hardware Config.vi 节点的图标及其端口定义

3. AI Clock Config.vi

设置 DAQ 设备的扫描时钟频率，节点的图标及其端口定义如图 11.3.21 所示。

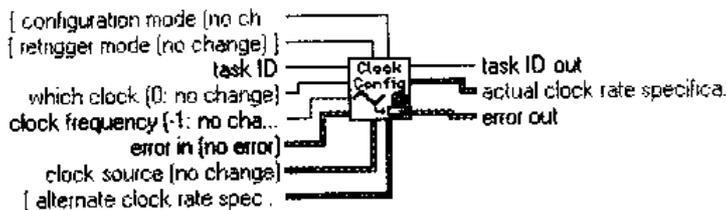


图 11.3.21 AI Clock Config.vi 节点的图标及其端口定义

4. AI Trigger Config.vi

设置 DAQ 设备的触发方式，节点的图标及其端口定义如图 11.3.22 所示。

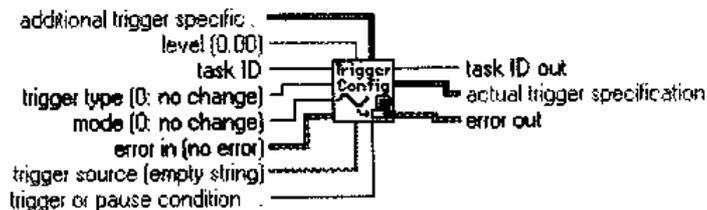


图 11.3.22 AI Trigger Config.vi 节点的图标及其端口定义

5. AI Buffer Config.vi

分配 DAQ 设备的内部缓存，节点的图标及其端口定义如图 11.3.23 所示。

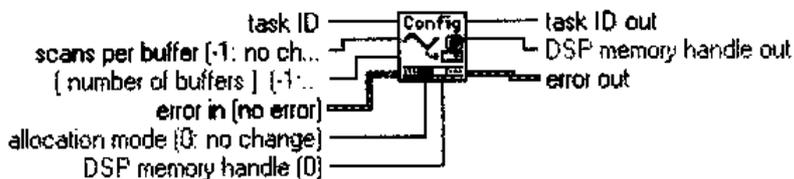


图 11.3.23 AI Buffer Config.vi 节点的图标及其端口定义

6. AI Control.vi

控制 DAQ 设备的模拟输入操作，并指定采集数据的数目，节点的图标及其端口定义如图 11.3.24 所示。

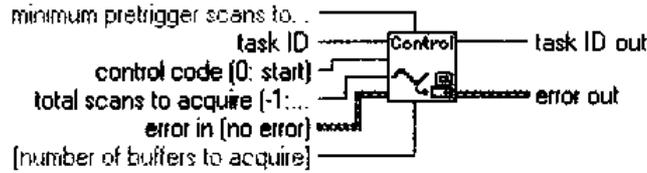
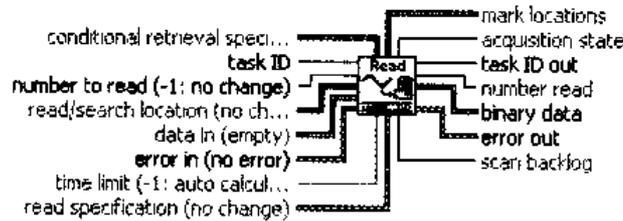


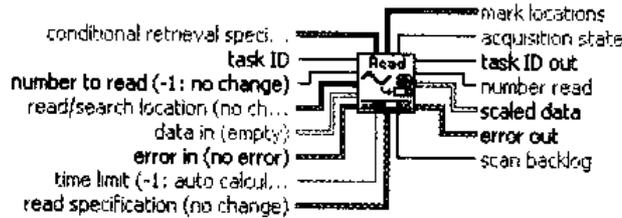
图 11.3.24 AI Control.vi 节点的图标及其端口定义

7. AI Buffer Read.vi

返回位于 DAQ 设备内部缓存中的模拟输入数据，节点的图标及其端口定义如图 11.3.25 所示。



(a) Binary Array



(b) Scaled Array

图 11.3.25 AI Buffer Read.vi 节点的图标及其端口定义

8. AI Parameter.vi

设置并返回其他 DAQ 节点没有设置的数据采集参数，节点的图标及其端口定义如图 11.3.26 所示。

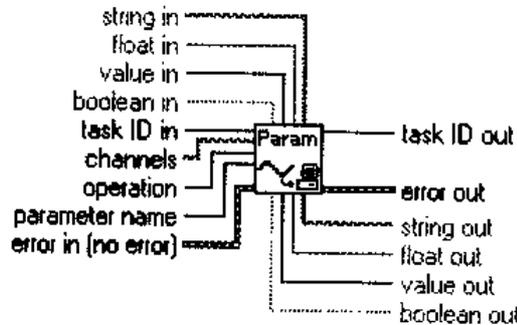


图 11.3.26 AI Parameter.vi 节点的图标及其端口定义

9. AI SingleScan.vi

从一个模拟输入通道中直接返回一次扫描的数据，节点的图标及其端口定义如图 11.3.27 所示。

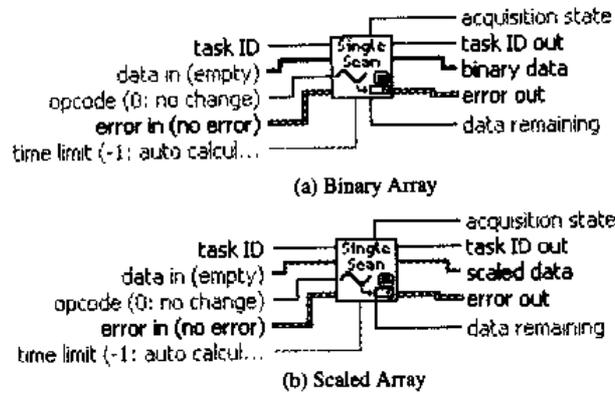


图 11.3.27 AI SingleScan.vi 节点的图标及其端口定义

本章介绍的主要内容都是本地数据采集，LabVIEW 还提供了远程数据采集（RDA）的功能。

第12章 仪器控制

强大、灵活的仪器控制功能是 LabVIEW 区别于其他编程语言的主要特点。LabVIEW 不仅提供了数百种不同接口测试仪器的驱动程序，而且支持 VISA、SCPI 和 IVI 等最新的程控软件标准，为用户设计开发先进的测试系统提供了软件支持。本章首先概要介绍仪器驱动器、VPP 仪器驱动器、IVI 仪器驱动器和 VISA 库函数等基本概念，然后对 LabVIEW 提供的 VISA 编程节点的功能和用法进行了论述，最后结合一个典型的 VXI 仪器模块——VT1432A 在 LabVIEW 环境下的编程控制，介绍了如何将一般的 VPP 驱动程序转换为可以直接在 LabVIEW 环境下调用的 VPP 节点，并且通过应用实例分析了 VPP 节点的使用方法。

12.1 仪器驱动器

仪器驱动器 (Instrument Driver) 也称仪器驱动程序，是完成对某一特定仪器控制与通信的软件程序集，也可认为是仪器的软件描述，它是应用程序实现仪器控制的桥梁。每个仪器模块都有自己的仪器驱动器，仪器厂商将仪器驱动器以源代码的形式提供给用户。

12.1.1 VPP 仪器驱动器

由于虚拟仪器需要提供模拟实际仪器操作面板的虚拟面板，因此虚拟仪器驱动器不仅是实施仪器控制的程控代码，还是仪器程控代码、高级软件编程与先进人机交互技术三者相结合的产物，是一个包含实际仪器使用和操作信息的软件模块。虚拟仪器驱动器上层是一系列按功能分组的主/副软面板，每个软面板又由一些按键、旋钮、表头等控件组合而成，每个控件都对应不同的功能，即其程控代码相异。它的底层部分则基于一组 I/O 函数和测试接口，在实时模式下，测试人员对软面板上控件的操作将直接反映到真实仪器上。仪器驱动器和用户直接打交道的部分是操作接口，即虚拟软面板和面板上的控件。

应用软件建立在仪器驱动程序之上，直接面对操作用户，通过提供直观友好的测控操作界面、丰富的数据分析与处理功能，来完成自动测试任务。仪器驱动程序模块负责处理与某一专门仪器通信和控制的具体过程，通过封装复杂的仪器编程细节，为用户使用仪器提供了简单的函数接口，用户不必对各种诸如 GPIB、VXI、数据采集板等仪器硬件有专门的了解，就可以通过仪器驱动程序来使用这些仪器硬件。仪器驱动程序一般由仪器厂商以动态链接库 (DLL) 的形式提供给用户。在仪器驱动程序的开发方面已形成了一系列的标准，这使得各个厂商能遵循统一的标准来开发驱动程序。当测试要求改变，需要更换新的仪器硬件时，只需更新相应的驱动程序，并保证它对上层的接口保持不变，那么新的仪器硬件就能在原来的系统中正常运行。

进入 20 世纪 90 年代，随着 VXI 总线标准的建立和 VXI 仪器的发展，程控仪器驱动软

件与编程环境的标准化成为测试与仪器领域关注的问题。由世界上几十家最有实力的仪器厂商(其中包括 HP, Tek 和 Racal 等大公司)联合成立了 VXI 即插即用系统联盟(VXIplug&play Systems Alliance)。该联盟的主要目标是提高 VXI 技术的易用性。作为软件兼容性工业标准发展的第一步,该联盟开发了新一代程控仪器 I/O 软件规范——虚拟仪器软件规范 VISA (Virtual Instrumentation Software Architecture)。在此基础上, VXI 即插即用系统联盟还制定了包括操作系统、编程语言和 I/O 软件在内的测试系统软件框架。最常用、最流行的系统框架是 WIN95/98 和 WINNT 框架。操作系统是 Windows95/98 和 WindowsNT; 编程语言包括 LabWindows/CVI, ANSI C, Visual Basic 和 LabVIEW, I/O 软件就是 VISA。

VISA 标准的制定,为高级仪器驱动程序和低级 I/O 驱动程序之间提供了一个层,使得高级仪器驱动程序和硬件无关,更是大大提高了各种接口仪器(包括 VXI 仪器、GPIB 仪器、RS232 仪器等)的互换性。

随后, VXI 即插即用系统联盟颁布执行了 VXI 总线即插即用(VXI Plug&Play, 简称 VPP)标准,规定今后仪器驱动器的设计必须符合两个 VPP 规范,即《仪器驱动程序结构和模型》(VPP3.1)和《仪器驱动程序设计规范》(VPP3.2)。为此,测试界提出了两个基本的概念模型:

第一个模型是仪器驱动程序外部接口模型,表示仪器驱动程序如何与外部软件系统接口;

第二个模型是仪器驱动程序内部设计模型,所有 VPP 仪器驱动程序都是根据该模型构成的。

一般而言,满足 VXI Plug&Play 标准的 VPP 仪器驱动器是一个由 C 函数库文件、可执行程序、文本文件和 Windows 帮助文件组成的软件集。C 函数库文件包括 ANSI C 源代码文件(.c, .h), 动态链接库文件(.dll)和函数面板文件(.fp)。这个库使用 VISA 进行 I/O 操作。软面板可执行程序是一个独立的可执行程序,它提供交互的、有鼠标驱动的图形接口,用于控制仪器和显示测量结果。知识库文件(Knowledge Base File, .kb)是一个 ASCII 文件,它描述仪器的规范与配置。帮助文件(.hlp)提供了有关 C 函数库的编程实例,仪器概述和对软面板的帮助文件。VPP 仪器驱动器是随仪器由厂家提供的,也可以从互联网上免费下载。对于自制模块或一些厂家早期生产的不符合 VXI 即插即用规范的模块,则需自行开发相应的仪器驱动器。

VPP 仪器驱动器可以与绝大多数流行的测试软件开发环境共同工作: NI 公司的 LabVIEW、LabWindows/CVI, Microsoft 公司的 Visual Basic (VB)、Visual C/C++ (VC/VC++) 和 HP 公司的 HP VEE for Windows 等。这些应用软件开发环境大致可分为面向对象的编程语言和图形化编程语言两大类。对需要自己编写 VPP 仪器驱动器的用户, LabWindows/CVI 是最好的开发工具,它提供了交互式编程接口,只需选择若干参数和选项,基本上就可以由系统自动完成仪器驱动器的开发。

驱动程序的开发有三个要求:

- 驱动程序要给出源代码,以使用户能根据应用场合需要,修改优化这个驱动程序;
- 驱动程序必须模块化、层次化,使用户调用起来方便;
- 驱动程序结构要符合 VPP 规范,使得用户熟悉了一个仪器的驱动程序后,能就很快地使用其他仪器驱动程序。

在仪器驱动器开发过程中会遇到对 VISA 函数库的调用,可以说 VISA 函数库是理解和掌握仪器驱动器最重要的基础。

LabVIEW 提供了一个仪器驱动器库，库中包括了各种程控仪器（GPIB 仪器、VXI 仪器和串行仪器等）的仪器驱动器，例如 HP 34401A 数字万用表的仪器驱动器。仪器驱动器在 Functions 模板 → All Functions 子模板 → Instrument Drivers 子模板中，如图 12.1.1 所示。



图 12.1.1 Instrument Driver 子模板

利用这些仪器驱动器，用户可以很容易地控制各种仪器，并将主要精力放在仪器功能的实现上，而不必关心具体的编程细节。这一点是 LabVIEW 强大功能的体现之一。LabVIEW 中的仪器驱动器模型如图 12.1.2 所示。

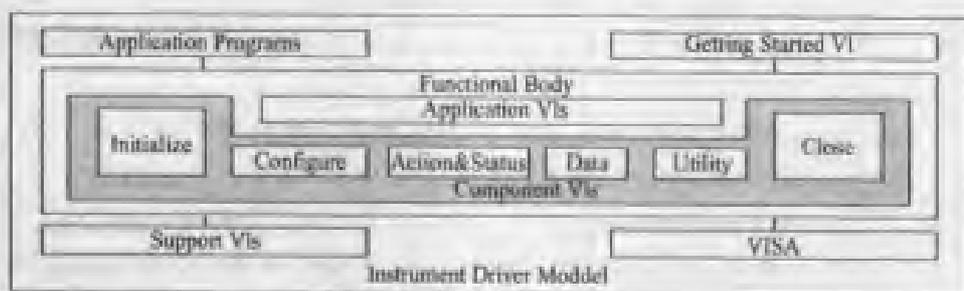


图 12.1.2 仪器驱动器模型

LabVIEW 中的仪器驱动器主要用 VISA 编写，用户可以在 LabVIEW 的安装光盘中找到更多的仪器驱动器。

根据 VPP 软件标准，仪器驱动器的相关技术问题，如命名约定、文件格式、开发框架和软面板设计格式等已经建立统一标准，使得由不同厂家提供的仪器驱动器具有互操作性，为测试人员快速开发模块化的测试程序提供了各种直观易用的高层函数。VPP 驱动器标准建立的初衷是提供仪器驱动器的高层编程接口，代替低层的 GPIB 和 VXI 仪器命令，所以，

按照这个标准设计的仪器驱动器没有考虑仪器控制器的执行效率。许多仪器驱动器将仪器设置和操作等多个动作组合为一个函数，使得冗余的 I/O 命令重复配置和处理操作占用了许多时间。为了解决这个问题，1998 年美国 NI 公司最先提出了一种新的基于状态管理的仪器驱动器体系结构，即可互换虚拟仪器驱动器（Interchangeable Virtual Instruments，简称 IVI）模型和规范。IVI 仪器驱动器使建立在仪器驱动器基础上的测试程序独立于仪器硬件，很快成为新的仪器驱动器标准。

12.1.2 IVI 仪器驱动器

IVI 基金会努力从基本的互操作性（Interoperability）到可互换性（Interchangeability），为仪器驱动程序提升了标准化水平。通过为仪器类制订一个统一的规范，使测试工程师获得更大的硬件独立性，减少了软件维护和支持费用，缩短了仪器编程时间，提高了运行性能。运用 IVI 技术可以使许多部门获益。例如使用 IVI 技术的事务处理系统可以把不同的仪器用在其系统中，当仪器陈旧或者有升级的、高性能或低造价的仪器时，可以任意更换，而不需要改变测试程序的源代码。目前，IVI 基金会共制订了五类仪器的规范：

- 示波器/数字化仪（IVIScope）；
- 数字万用表（IVIDmm）；
- 任意波形发生器/函数发生器（IVIFGen）；
- 开关/多路复用器/矩阵（IVISwitch）；
- 电源（IVIPower）。

NI 开发的 IVI 驱动程序库包括 IVI 基金会定义的这五类仪器的标准驱动器、仿真驱动程序和软面板，为仪器的交换提供了一个标准接口，通过定义一个可互换性虚拟仪器的驱动模型来实现仪器的互换性，NI 设计的 IVI 体系结构如图 12.1.3 所示。

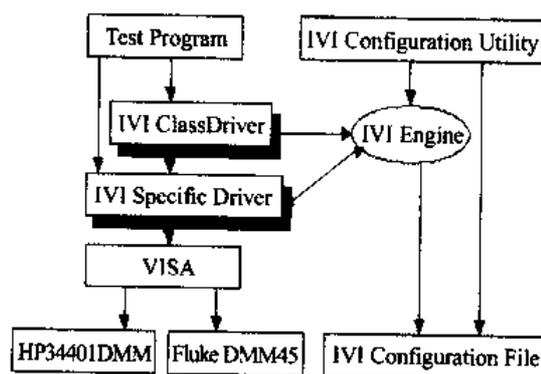


图 12.1.3 IVI 体系结构

从图 12.1.3 可以看出，IVI 驱动程序比 VXI Plug&Play（简称 VPP）联盟制订的 VISA 规范更高一层。它扩展了 VPP 仪器驱动程序的标准并加上了仪器的可互换性、仿真和状态缓存等特点，使得仪器厂商可以继续用他们的仪器特征和新增功能。因此 IVI 基金会是对 VPP 系统联盟的一个很好的补充。

对用户而言，IVI 仪器驱动器的使用方式与传统的 VPP 仪器驱动器一样，但两者的内部结构是完全不同的。IVI 仪器驱动器的内部结构包括源代码本身和状态管理库两部分，IVI

仪器驱动器的核心是仪器属性和状态缓存两个模型。

与 VPP 仪器驱动器相比,新的 IVI 仪器驱动器具有以下特点。

① 在不牺牲易用性的基础上改善了仪器驱动器的运行效率。在编写测试程序时, VPP 仪器驱动器包含的高层仪器控制函数是非常直观和容易使用的,任何改进仪器驱动器性能的方法都不能牺牲基本的编程模型。IVI 仪器驱动器与 VPP 仪器驱动器保持相同的高层编程接口,也就是对用户而言, VPP 仪器驱动器和 IVI 仪器驱动器提供的功能函数的调用格式是完全相同的。

② 提高了采用仪器驱动器编程的灵活性。由于 IVI 仪器驱动器可以跟踪和管理所有的仪器状态或设置,因此,用户可以进入仪器所有底层设置;另一方面,通过仪器状态管理,IVI 仪器驱动器可以工作在“测试开发”和“正常运行”两种模式下。在“测试开发”工作模式,IVI 仪器驱动器可以自动完成一系列状态检查,帮助用户发现编程错误。当程序调试进行正常使用后,可以切换到“正常运行”模式,使驱动器软件高速运行。

③ 提供多线程安全运行和仪器仿真功能。IVI 仪器驱动器可以多线程安全运行,因此可以应用在先进的多线程并行测试系统中,此外,它还提供仿真功能,用户可以在不连接实际仪器的情况下开发测试程序。

④ IVI 仪器驱动器与接口总线无关。IVI 仪器驱动器只与仪器的测试功能相关,与仪器采用什么接口总线方式无关。例如对数字示波器,无论是采用 GPIB 总线还是采用 VXI 总线,其 IVI 仪器驱动器的全部功能函数都是相同的,只是通过一个初始化函数 InitWithOptions 来区分仪器接口总线和地址的异同。

IVI 仪器驱动器在 VPP 仪器驱动器的基础上,提高了仪器驱动器功能函数的执行效率,实现了不同接口总线仪器驱动器的互换性,并且为测试程序调试的方便性与运行效率的兼容提供了编程手段,这些都为实现智能虚拟仪器概念提供了软件支持。可以预计,今后越来越多的仪器厂商将提供符合 IVI 标准的仪器驱动器。

IVI 驱动器具有与传统的仪器驱动器不同的功能和特点。

- 通用标准配置 (Standardized Configuration Utility)。
- 标准化 (Standardization)。
- 状态缓存 (State Caching)。
- 范围检查 (Range Checking)。
- 仿真 (Simulation)。
- 状态检查 (Status Checking)。
- 强制记录 (Coercion Recording)。

正是这些新的功能和特点的出现,使得 IVI 成为一代新的仪器驱动器标准。对于用户来讲,仪器驱动器是一个必不可少的工具,它可以帮助用户快速开发测试应用程序。IVI 驱动器的新技术继承了传统仪器驱动器的优点,同时增加了提高应用程序开发速度和应用程序性能的新特性。除了 IVI 所具有的新的优点之外,用户使用 IVI 驱动器开发应用程序时采用的方法与使用传统的 LabVIEW 驱动器开发应用程序时采用的方法基本相同。

12.2 VISA 标准

VISA 是用于仪器编程的标准 I/O 函数库及其相关规范的总称,一般称这个 I/O 函数库

为 VISA 库。VISA 库驻留于计算机系统中，是计算机与仪器之间的软件层连接，用以实现对仪器的程控。VISA 对于测试软件开发者来说是一个可调用的操作函数集，本身不提供仪器编程能力，它只是一个高层 API（应用程序接口），通过调用低层的驱动程序来控制仪器。NI-VISA 的层次如图 12.2.1 所示。

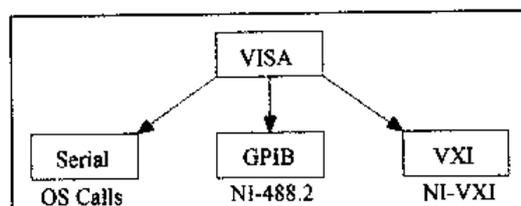


图 12.2.1 NI-VISA 层次图

仪器 I/O 控制软件是处理在计算机与仪器之间通过物理连接进行通信的问题，也就是处理如何在计算机与仪器之间传送命令与数据的问题。在 VISA 标准出现之前，最著名的仪器 I/O 软件是 HP 公司推出的标准仪器控制库（Standard Instruments Control Library，简称 SICL）。包括 SICL 在内的 I/O 控制软件的主要不足是不同的接口仪器采用不同的 I/O 库，没有解决仪器互操作问题。随着仪器类型的增加和测试系统复杂化的提高，使用通用、统一的 I/O 库来实现对各类程控仪器的编程变得十分重要。因此 I/O 接口无关性对于 I/O 控制软件来说是至关重要的。当用户编程控制一组仪器时，总是希望该应用程序在各种硬件接口上都能工作，尤其是对于使用 VXI 仪器的用户。比如说，用户在控制 VXI 仪器时，希望无论计算机内置于 VXI 主机箱，还是通过 MXI 总线与 VXI 仪器相连，或是通过 GPIB-to-VXI 适配器与 VXI 仪器连接，它的应用程序都能工作。如果 I/O 软件接口一致，用户在切换控制方法时就不必修改应用程序代码。VISA 的出现使用户的这种希望成为可能，通过调用相同的 VISA 库函数并配置不同的设备参数，就可以编写控制各种 I/O 接口仪器的通用程序。

VISA 的内部结构是一个先进的、面向对象的结构，这一结构使得 VISA 和在它之前的 I/O 控制软件相比，在接口无关性、可扩展性和功能上都有很大提高。VISA 的可扩展性远远超出了 I/O 控制软件的范畴，而且由于 VISA 内部结构的灵活性，使得 VISA 在功能和灵活性上超过了其他的 I/O 控制库。不过，尽管 VISA 具有许多的优越性，但它的 API 函数却比其他具有类似功能的 I/O 库少得多，因此，VISA 很容易被初学者掌握。另外，VISA 高度的可访问性和可配置性又使得熟练的用户可以利用 VISA 的许多独有特性，这些独有的新特性使得 VISA 的应用范围大大超过了传统的 I/O 软件。VISA 不仅为将来的仪器编程提供了许多新特性，而且还兼容了过去已有的仪器软件。

VISA 函数模型建立在设备资源无关性模型之上，因此，资源和资源管理器是 VISA 函数模型的两个最基本概念。

1. 资源（Resource）与资源类

VISA 中的资源是指一个计算机可访问或可与之通信的实体，例如各种仪器、内存访问资源、仪器的读写端口等。一个 VISA 资源包括三部分：一组资源属性，一组可以在资源上异步执行或由资源产生的事件，一组控制资源的操作。一个资源类通常指设备某种功

能的定义 (比如读、写、触发等功能)。

2. 默认资源管理器 (Default Resource Manager)

VISA 资源管理器是控制设备资源, 可完成 VISA 系统初始化的最高层次 VISA 操作。用 `viOpenDefaultRM` 函数打开默认资源管理器是 VISA 编程不可缺少的第一步 (在 LabVIEW 中, 这一步操作是由 LabVIEW 自动完成的, 不需要任何编程), 然后用户可以通过默认资源管理器打开指定的设备资源。

表 12.2.1 描述了 VISA 结构的基本模型, 模型共有 5 层。该模型以设备资源无关性模型为基础。

表 12.2.1 VISA 分层结构模型

顶层	应用程序
第四层	虚拟仪器
第三层	VISA 仪器控制组织器
第二层	VISA 仪器控制资源
底层	VISA 资源管理器

① VISA 资源管理器。VISA 资源管理器是 VISA 模型的最低层, 用于管理、控制和分配 VISA 资源。它将 VISA 应用与 VISA 资源的分配和管理的具体细节隔离开来。

② 三层资源。VISA 规定了三种层次的资源: I/O 层资源、仪器层资源、用户定义资源组, 这三层表示了一种高层资源使用低层资源的能力。一般来说, VISA 允许任意资源使用系统中的其他资源和资源组。

- I/O 资源。I/O 资源是第一层资源, 换言之是 VISA 仪器控制资源, 它用于控制所有 GPIB、VXI 和串行设备的功能, 还可以被扩展用于控制其他任意类型的接口。

- 仪器层资源。仪器层资源是第二层资源, VISA 仪器控制组织器就属于这类资源, 它用于通过一个资源的同一通道控制任意多个仪器控制资源, 还可用于控制具体物理设备的所有资源。

- 用户定义资源组。用户定义资源组是第三层资源, 又称为虚拟仪器, 它包括前两层资源再加上用户在前两层资源的基础上创建的任何资源。

③ 用户层应用程序。用户层应用程序是 VISA 模型的最高层, 它利用若干个 VISA 资源来完成特定的任务, 应用程序本身不是 VISA 资源。应用程序利用 VISA 资源管理器创建与 VISA 系统资源相连的通道。VISA 资源管理器为系统中的所有 VISA 资源提供了一个通用接口。

VISA 可以控制 VXI 仪器、GPIB 仪器或串行仪器, 是仪器驱动程序发展的一个工业标准。无论接口如何, VISA 都可以使用相同的操作与仪器通信。在理解掌握了 VISA 的基本概念后, 就可以利用 VISA 编程对仪器进行程控。

12.3 VISA 编程

作为仪器 I/O 函数库, VISA 编程与传统的 I/O 软件编程基本相同, 主要通过设备 I/O 端口的读写操作和属性控制, 实现与仪器的命令与数据交换。当然, 作为一个完整的仪器

I/O 软件标准, VISA 库函数还必须考虑到编程的灵活性和操作的完整性。因此, VISA 函数按照功能基本上可分为基本 I/O、格式化 I/O、内存 I/O、资源管理、共享内存管理、事件处理和属性控制等几大类。用户可以在 VISA 中利用 SCPI 命令来控制消息基的 GPIB 和 VXI 仪器, 以及带有 SCPI 命令翻译节点的寄存器基的 GPIB 和 VXI 仪器。在 LabVIEW 中 VISA 函数库中的函数是以 VISA 节点的形式出现的。

12.3.1 VISA 节点

LabVIEW 中所有的 VISA 节点均在 Functions 模板→All Functions 子模板→Instrument I/O 子模板→VISA 子模板中, 如图 12.3.1 所示。



图 12.3.1 VISA 子模板

VISA 子模板又分为基本 VISA 节点、指定接口 (Interface Specific) 子模板、事件处理 (Event Handling) 子模板、高级寄存器读写 (High Level Register Access) 子模板、低级寄存器读写 (Low Level Register Access) 子模板等几个部分。

用户可以利用 SCPI 命令通过基本 VISA 节点来控制消息基的 GPIB 和 VXI 仪器, 以及带有 SCPI 命令翻译节点的寄存器基的 GPIB 和 VXI 仪器。本节仅介绍一些常用基本 VISA 节点的法, 限于篇幅, 其余一些复杂的或用于仪器底层控制的 VISA 节点, 这里不再详述。

1. VISA Find Resource 节点

VISA Find Resource 节点的功能是查找当前系统中存在的 VISA 资源, 节点的图标及其端口定义如图 12.3.2 所示。

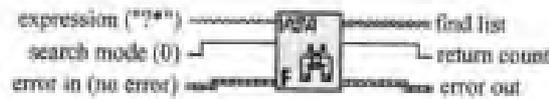


图 12.3.3 VISA Find Resource 节点的图标及其端口定义

下面详细介绍节点的几个端口的功能。

(1) expression 端口

expression 端口用于设定节点的查找范围，如查找所有的 VISA 资源，或仅查找 GPIB 仪器资源，或仅查找 VXI 仪器资源等。expression 端口中可以使用的表达式见表 12.3.1。

表 12.3.1 expression 端口中可以使用的表达式

表达式	所匹配仪器资源实例
GPIB?*INSTR	匹 配: GPIB0::2::INSTR, GPIB1::1::INSTR, GPIB-VXI::8::INSTR
GPIB[0-9]*?*INSTR	匹 配: GPIB0::2::INSTR, GPIB1::1::INSTR 不匹配: GPIB-VXI::8::INSTR
GPIB[0]:?*INSTR	匹 配: GPIB1::1::INSTR 不匹配: GPIB0::2::INSTR, GPIB12::8::INSTR
VXI?*INSTR	匹 配: VXI0::1::INSTR 不匹配: GPIB-VXI0::1::INSTR
GPIB-VXI?*INSTR	匹 配: GPIB-VXI0::1::INSTR 不匹配: VXI0::1::INSTR
?*VXI[0-9]*?*INSTR	匹 配: VXI0::1::INSTR, GPIB-VXI0::1::INSTR
ASRL[0-9]*?*INSTR	匹 配: ASRL1::INSTR 不匹配: VXI0::5::INSTR
ASRL1+::INSTR	匹 配: ASRL1::INSTR, ASRL11::INSTR 不匹配: ASRL2::INSTR
(GPIB VXI)?*INSTR	匹 配: GPIB1::5::INSTR, VXI0::3::INSTR 不匹配: ASRL2::INSTR
(GPIB0 VXI0)::1::INSTR	匹 配: GPIB0::1::INSTR, VXI0::1::INSTR
?*INSTR	匹配所有的仪器资源
?*VXI[0-9]*?*MEMACC	匹 配: VXI0::MEMACC, GPIB-VXI1::MEMACC
VXI0:?*	匹 配: VXI0:1::INSTR, VXI0:2::INSTR, VXI0::MEMACC
?*	默认值, 匹配所有的 VISA 资源

(2) search mode 端口

search mode 端口用于确定搜索模式和 find list 端口返回的 VISA 资源仪器描述符的格式。端口输入值表示的功能见表 12.3.2。

表 12.3.2 search mode 端口输入值表示的功能

输入值	功能
0	默认值, 搜索所有的总线, find list 端口返回搜索到的 VISA 资源的仪器描述符, 仪器描述符的格式为 VISA 标准规定的正规格式
1	不搜索任何总线, find list 端口返回搜索到的 VISA 资源的仪器描述符, 仪器描述符的格式由用户定义的别名。这种搜索模式下, 只有表达式“?”有效
2	搜索所有的总线, 如果有些 VISA 资源用户定义了别名, 则返回的 VISA 资源仪器描述符采用别名的格式, 其他的 VISA 资源仪器描述符采用 VISA 标准规定的正规格式
4	搜索所有的总线, find list 端口返回搜索到的 VISA 资源和用户定义了别名但没有搜索到的 VISA 资源的仪器描述符。在所返回的 VISA 资源仪器描述符中, 如果用户定义了别名, 则仪器描述符采用别名格式, 其他采用 VISA 标准规定的正规格式

(3) return count 端口

输出端口 return count 返回查找到的 VISA 资源的数目。

(4) Find List 端口

Find List 端口返回搜索到的 VISA 资源的仪器描述符 (Instrument Descriptor) 的数组, 其数据类型是字符串。仪器描述符代表了一个 VISA 资源的名称和位置, 其格式如下:

Interface Type (board index) ::Address::VISA Class

Interface Type 是指 VISA 资源的类型, 如 GPIB, VXI, ASRL1 等。Board Index 是指当系统存在两个以上的同类 VISA 资源时, 每个 VISA 资源的系统编号, 例如系统中存在两块 GPIB 卡, 则第一块卡的 Board Index 为 0, 第二块卡的 Board Index 为 1。Address 是指仪器的地址。对于 VXI 仪器而言, 是 VXI 仪器的逻辑地址, 范围是 0~255; 对于 GPIB 仪器而言, 是指 GPIB 仪器在 GPIB 总线上的地址, 范围是 0~15; 该参数对串行仪器无效, 如接到 COM1 上的串行仪器的仪器描述符为 ASRL1::INSTR。请注意, 在 Address 和 Interface Type (board index) 两参数之间用“::”隔开。VISA Class 是指 VISA 类, 将 VISA 部分或全部的操作封装在一起。在 LabVIEW 中可用的 VISA 类详见表 12.3.3。最常用的 VISA 类是 Instr, 它包含了一个仪器中所有的 VISA 操作。当参数 VISA Class 为空时, 其默认值为 Instr。建议用户在使用仪器描述符时, 最好使用该参数。

表 12.3.3 可用 VISA 类表

VISA 类	VISA 类
Instr	Generic Instr
GPIB Instr	Service Request Event
VXI/GPIB-VXI MDB Instr	Trigger Event
VXI/GPIB-VXI RDB Instr	VXI Signal Instr
Serial Instr	Resource Manager

例 12.3.1 利用 VISA Find Resource 节点搜索当前系统中存在的所有的 VISA 资源。

若要搜索当前系统中存在的所有 VISA 资源, 则 expression 端口输入的表达式为 “?*”, VI 的前面板和程序框图如图 12.3.3 所示



图 12.3.3 例 12.3.1 的前面板及框图程序

2. VISA Open 节点

当完成 VISA 资源的搜索后, 对于搜索到的 VISA 资源, 可以通过 VISA Open 节点打开, 建立计算机与这些 VISA 资源之间的通信管道。节点图标及其端口定义如图 12.3.4 所示。

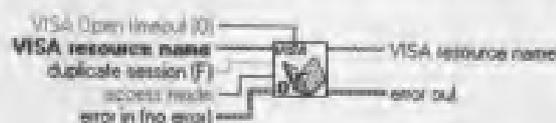


图 12.3.4 VISA Open 节点的图标及其端口定义

VISA resource Name 端口用于指定需要打开的 VISA 资源的名称, 实际上就是前面所说的 VISA 资源仪器描述符 (Instrument Descriptor)。例如, 需要打开一台地址为 10 的示波器 (GPIB 总线) 的资源, 应输入字符串 “GPIB0::10::INSTR”。

当 VISA Open 节点操作正常完成后, 实际上已打开了示波器的资源, 系统与示波器之间就建立了一种特殊的连接关系, 此时, 可以利用 VISA 的读写节点与仪器进行通信了。

实际上, VISA Open 资源在打开一个仪器资源时, 进行了两步操作:

第一步, 打开 VISA 默认资源管理器 (Default Resource Manager)。

为了能够正确有效地应用 VISA 节点控制仪器, 用户必须理解默认资源管理器和仪器描述符之间的关系。可以打个比方, 默认资源管理器是一名电话接线员, 用户打开一个默认资源管理器 (LabVIEW 会自动完成) 时, 就好像拿起电话听筒, 命令接线员建立 VISA 资源与系统之间一条通信线路。然后, 接线员就开始拨打电话号码来建立 VISA 资源与系统之间的通信线路。默认资源管理器所用的电话号码就是仪器描述符。

另外, 利用 VISA Find Resource 节点可以为默认资源管理器搜索出系统中所有可用的电话号码。

第二步, 按照 VISA Resource Name 端口指定的仪器地址打开仪器资源。

3. VISA Close 节点

与 VISA Open 节点相反, VISA Close 节点用于将打开的 VISA 资源关闭。节点图标及其端口定义如图 12.3.5 所示。

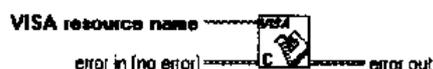


图 12.3.5 VISA Close 节点的图标及其端口定义

当 VISA Close 节点运行时, 也进行了两步操作: 第一步, 关闭仪器资源; 第二步, 关闭 VISA 默认资源管理器。

4. VISA Write 节点

VISA Write 节点的功能是将 write buffer 端口输入的 SCPI 命令 (例如 *RST, *CLS, *IDN? 等) 发送到仪器中。节点图标及其端口定义如图 12.3.6 所示。



图 12.3.6 VISA Write 节点的图标及其端口定义

5. VISA Read 节点

VISA Read 节点的功能是从仪器缓存中读出数据, 节点图标及其端口定义如图 12.3.7 所示。

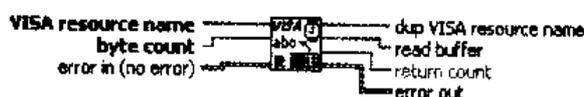


图 12.3.7 VISA Read 节点的图标及其端口定义

节点的 byte count 端口用于指定读出数据的字节数, 其默认值为 0。该参数应当根据仪器输出缓存中的实际情况来确定, 对于某些仪器, 若该参数设置不当, 就不能正确读出数据。节点的输出端口 read buffer 将读出的数据以字符串的形式送出。

12.3.2 VISA 编程实例

利用 VISA 控制仪器的最简单的流程如图 12.3.8 所示。



图 12.3.8 VISA 控制仪器的最简单流程

下面以一个 LabVIEW 程序为例, 进一步说明 LabVIEW 中的 VISA 编程的基本过程和步骤。

例 12.3.2 询问一台地址为 10 的 GPIB 示波器的识别码。

本例按照图 12.3.12 所示的流程进行编程, VI 前面板及框图程序如图 12.3.9 所示。

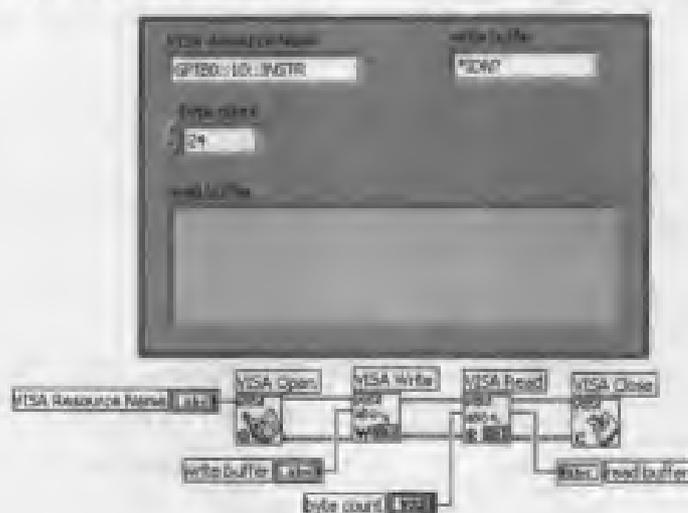


图 12.3.9 例 12.3.2 的前面板及框图程序

下面分析这个例子的框图程序。

第一步, 利用 VISA Open 节点打开仪器资源。在这一步中, 只需要为 VISA Open 节点指定被打开的仪器资源的仪器描述符。

第二步, 利用 VISA Write 节点写命令。将 write buffer 端口输入的 SCPI 命令发送给仪器。

第三步, 利用 VISA Read 节点从仪器缓存中读出数据。按照 byte count 端口指定的字节数, 从仪器缓存中读出数据。

第四步, 关闭 VISA Close 节点仪器资源。在这一步中, 只需要为 VISA Close 节点指定需要关闭的仪器资源的仪器描述符即可。

上面介绍的 VISA 编程只适用于消息基仪器, 以及带有 SCPI 命令翻译节点的寄存器基仪器, 对于大多数的寄存器基仪器来讲, 并不接受 SCPI 命令, 若要使用 VISA 函数库控制仪器, 须直接访问仪器的底层寄存器, 这需要用到 VISA 的寄存器读写节点, 这一过程较为复杂, 不适合一般的用户使用, 利用仪器的 VPP 驱动程序可以解决这个问题。

12.4 VPP 驱动程序转换与编程

利用 VISA 函数库可以开发仪器的驱动程序, 较为灵活, 但是需要用户熟悉 VISA 函数库、SCPI 命令, 特别是对于不支持 SCPI 命令的寄存器基仪器来讲, 用户更要熟悉仪器中大量寄存器的复杂定义。对于大多数的用户来讲, 由于并不是专职的开发仪器驱动器的测试工程师, 因此, 要做到熟练的利用 VISA 开发满足自己需要的仪器驱动器较为困难。并且, 即使用户对上述内容非常熟悉, 利用 VISA 函数库开发寄存器基仪器驱动器也需要一些时间, 这都是用户所不希望的。于是, 各大仪器厂商在推出测试仪器时, 同时也提供一个基于 VISA

函数库的 VPP 驱动程序, VPP 驱动程序将该测试仪器的各种操作按照类别封装到一个个驱动器功能函数当中, 用户直接调用这些驱动器功能函数, 就可以驱动测试仪器, 使用十分简单。使用 VPP 驱动程序, 只需要用户了解仪器的功能即可, 因此, 大多数情况下, 对测试仪器的驱动方式主要采用调用 VPP 驱动程序的方式。本节将以 VXI Technology 公司的 VT1432A 数字化仪为对象, 介绍如何在 LabVIEW 中使用 VPP 驱动函数。

12.4.1 VT1432A 数字化仪简介

VT1432A 数字化仪是一个 C 尺寸寄存器基 VXI 仪器模块, 如图 12.4.1 所示。



图 12.4.1 VT1432A 数字化仪

VT1432A 数字化仪性能参数如下。

- 16 通道同步采样, 每个通道使用独立的 ADC。
- 单通道最大采样率 51.2KSample/s。
- 精度: 16 位。
- 输入范围: $\pm 10V$ 。
- 内置 DSP。
- 可编程抗混滤波器。
- 高速数据传输, 传输速率可达 7.5M Samples/s。
- 支持 Windows 和 HP-UX 中的 VXIplug&play Drivers。

VT1432A 数字化仪的安装十分简单, 安装步骤如下:

第一步, 设置仪器逻辑地址。通过位于 VT1432A 数字化仪板卡一侧的跳线开关, 将仪器的逻辑地址设为合适的值。注意, 其逻辑地址不能与其他 VXI 仪器模块的逻辑地址发生冲突。

第二步, 将 VT1432A 数字化仪插在 VXI 机箱中任意一个槽中, 拧紧面板上的固定螺丝。

第三步, 安装 VPP 驱动程序。VT1432A 数字化仪的驱动程序可从 VT 公司的官方网站上下载, 是一个大约为 5M 的 EXE 文件, 安装过程十分简单, 按照其提示进行操作即可。

VT1432A 数字化仪是一块经典的 VXI 仪器模块, 由于其采样频率较高, 且通道间同步采样, 可用于测试振动、动态的压力变化等信号, 使用较为广泛, 具有一定的代表性。

12.4.2 VPP 驱动程序转换

VT1432A 的 VPP 驱动程序没有提供 LabVIEW 可直接使用的驱动函数, 并且大多数的 VXI 仪器模块的 VPP 驱动程序也没有提供 LabVIEW 可直接使用的驱动函数。因此, 如何在 LabVIEW 中调用 VXI 仪器模块的 VPP 驱动程序是很多 VXI 和 LabVIEW 用户所关心的

问题, 并且这是一个具有普遍性的问题。

LabVIEW 提供了一个仪器驱动器转换工具, 可以将 VPP 驱动程序中的函数转换为 LabVIEW 可以直接使用的 VPP 节点。在通常情况下, VPP 驱动程序提供的函数是一些 API 函数, 这些函数存放在一个动态链接库 (DLL) 中, 并且提供一个 .fp 文件, 用于记录这些函数。利用这个 .fp 文件, 就可以将动态链接库 (DLL) 中的 VPP 驱动函数转换为 LabVIEW 可以使用的 VPP 节点。下面以转换 VT1432A 数字化仪的 VPP 驱动程序为例, 介绍如何在 LabVIEW 转换 VPP 驱动程序。转换步骤如下。

第一步, 在 LabVIEW 窗口的主菜单中选择 Tools → Instrumentation → Import CVI Instrument Driver..., 弹出一个文件选择对话框, 如图 12.4.2 所示。

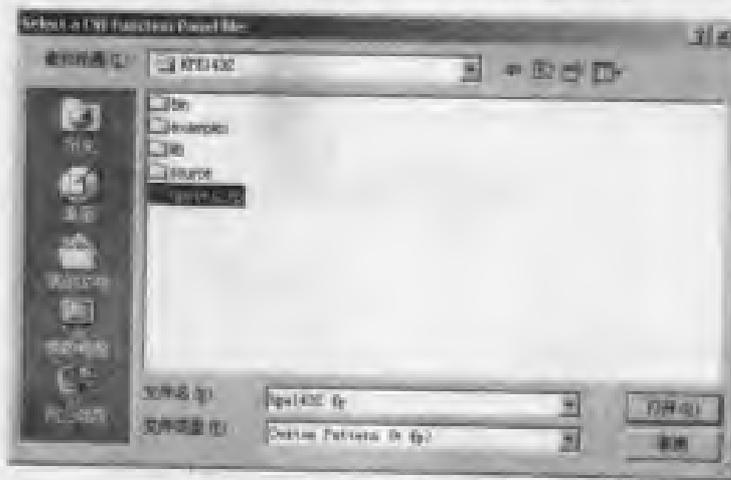


图 12.4.2 文件选择对话框

第二步, 在文件选择对话框中选择 VT1432A 数字化仪 VPP 驱动函数的 .fp 文件。在通常情况下, VXI 仪器模块的 VT1432A 数字化仪 VPP 驱动程序的安装路径为 C:\VXIppp, 因此, .fp 文件的路径通常为 C:\VXIppp\WinNT\HPE1432\hpe1432.fp。选中该文件后, 会弹出一个名为 CVI Function Panel Converter 的对话框, 如图 12.4.3 所示。

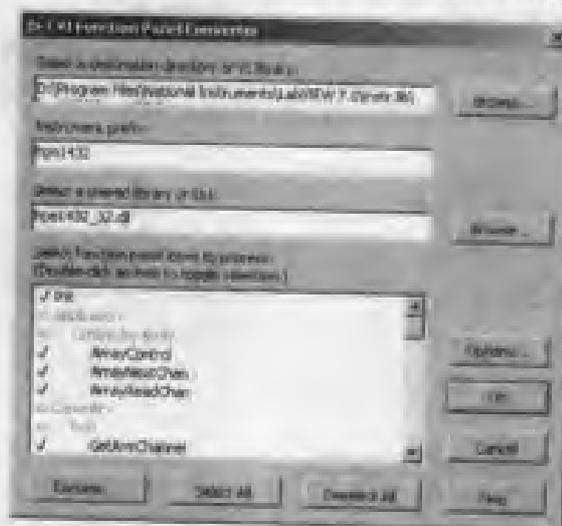


图 12.4.3 CVI Function Panel Converter 的对话框

第三步, 在 CVI Function Panel Converter 的对话框中设置相关的转换参数。对话框共

分为4栏。

Select a destination directory or VI library 用于设定转换后 VPP 节点的存放路径，默认路径为：

..\National Instruments\LabVIEW 7.0\instr.lib\hpe1432\hpe1432.lib

Instrument prefix 用于设置转换后 VPP 节点名称的前缀，其默认值为 hpe1432。

Select a shared library or DLL 用于设定 VPP 驱动函数所在的动态链接库，其默认值为 hpe1432_32.dll。

Select function panel items to process 用于选择需要转换的 VPP 驱动函数，当在第3栏中指定了一个动态链接库后，该栏会将动态链接库中的所有 VPP 驱动函数列出，供用户选择。单击该栏右侧的 Options 按钮，会弹出一个名为 FP Conversion Options 的对话框，如图 12.4.4 所示。在 FP Conversion Options 对话框中可以设定各种转换参数，在通常情况下，用其默认值即可。

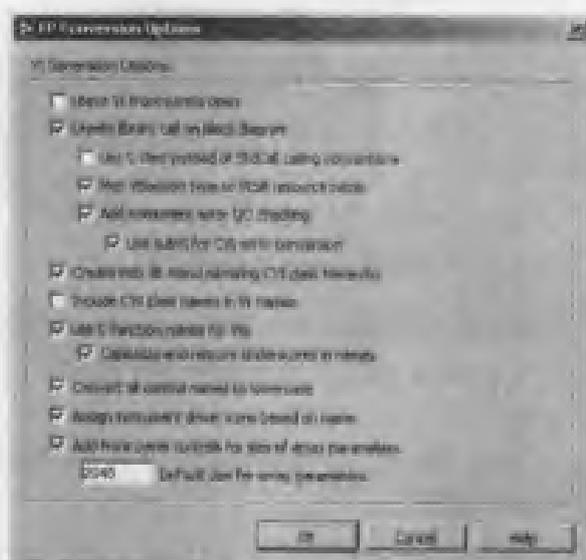


图 12.4.4 FP Conversion Options 的对话框

第四步，在 CVI Function Panel Converter 的对话框中设置好转换参数后，单击对话框中的 OK 按钮，LabVIEW 开始转换 VPP 驱动函数，转换过程中会出现一个名为 CVI Conversion Status 的对话框，对话框实时显示 VPP 驱动函数转换的状态，如图 12.4.5 所示。



图 12.4.5 CVI Conversion Status 的对话框

第五步，等待 VPP 驱动程序转换完成之后，重新运行 LabVIEW，在 LabVIEW 的 Functions 模板 → All Functions 子模板 → Instrument I/O 子模板 → Instrument Drivers 子模板中

会出现一个名为 hpe1432 的子模板, 如图 12.4.6 所示。



图 12.4.6 hpe1432 的子模板

所有的 VT1432A 数字化仪的 VPP 驱动节点都在 hpe1432 子模板中。此时, 用户就可以在 LabVIEW 中直接使用 VT1432A 数字化仪的 VPP 驱动程序了。

hpe1432 子模板中的 VPP 节点分为 4 个部分:

- 仪器初始化与关闭节点。仪器初始化与关闭节点用于初始化 VT1432A 数字化仪和关闭与 VT1432A 数字化仪的通信。
- Application 节点。Application 节点用于对 VT1432A 数字化仪数据采集通道数组的控制。
- Capability 节点。Capability 节点用于设置 VT1432A 数字化仪的各种数据采集参数, 并完成数据采集。
- Utility 节点。Utility 节点用于对 VT1432A 数字化仪中寄存器的访问、底层参数的设置和错误处理等。

12.4.3 VPP 驱动程序编程实例

在 LabVIEW 中使用 VPP 节点的方法和使用其他节点的方法一样, 但是需要用户了解这些 VPP 节点的功能和参数。下面以控制 VT1432A 数字化仪进行数据采集为例, 介绍在 LabVIEW 中如何使用 VPP 节点以及编程流程。

一个最基本的 VT1432A 数字化仪编程流程如图 12.4.7 所示。

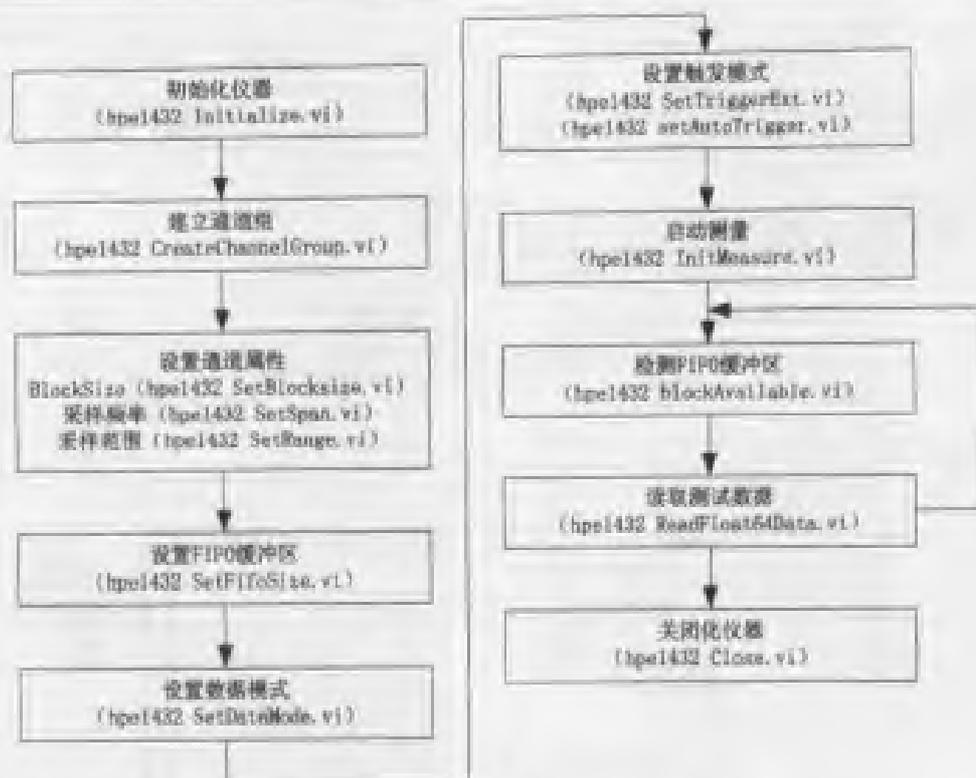


图 12.4.7 VT1432A 数字化编程流程

例 12.4.1 利用 VPP 节点控制 VT1432A 数字化仪进行数据采集，并实时将指定通道所采集的数据曲线利用 Waveform Graph 显示出来。

本例按照图 12.4.7 所示的流程进行编程，VI 的前面板和框图程序如图 12.4.8 和图 12.4.9 所示。

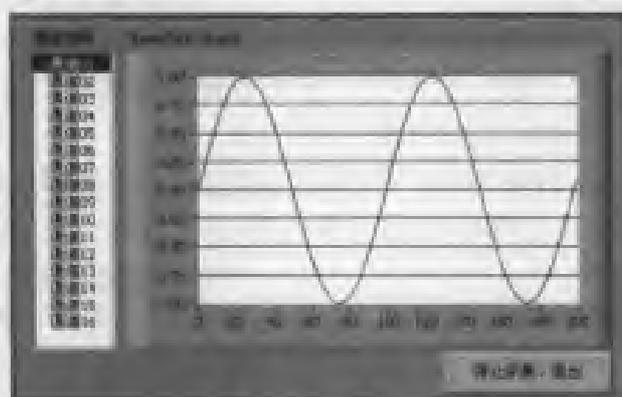


图 12.4.8 例 12.4.1 的前面板

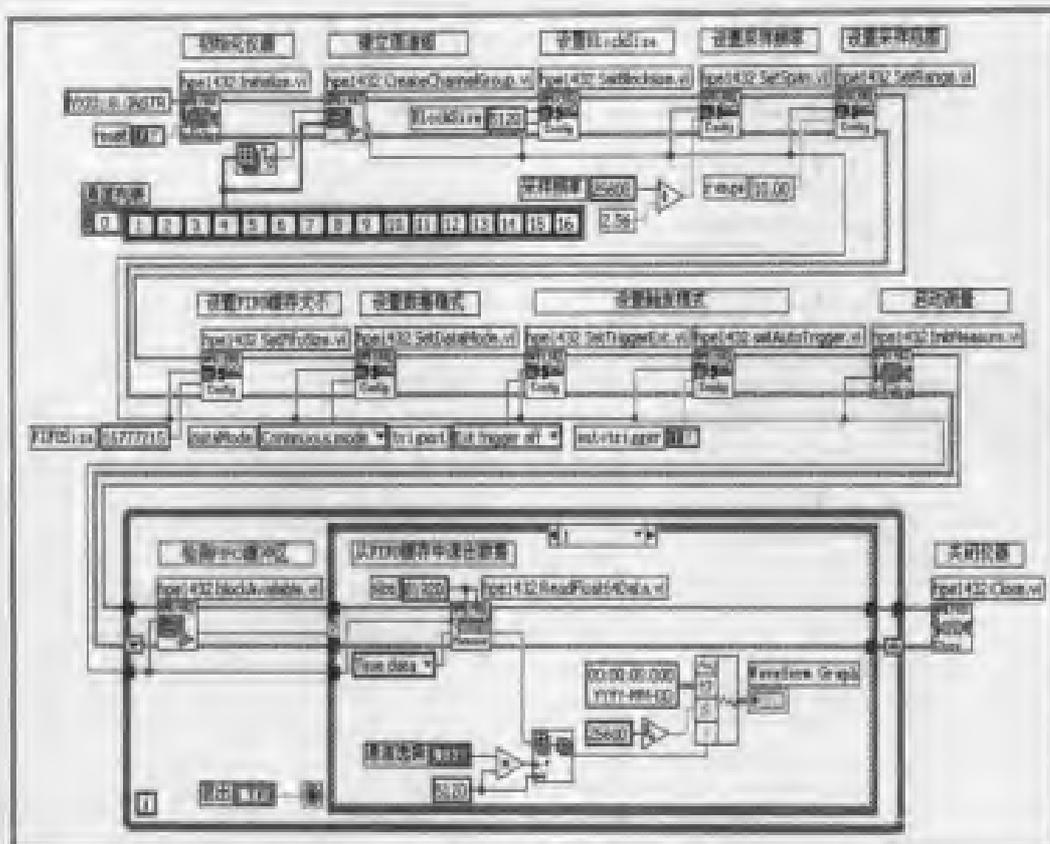


图 12.4.9 例 12.4.1 的框图程序

从图 12.4.9 可以看出，虽然采用的 VPP 节点较多，但是程序结构却很简单。在启动测量之后，采用一个 While 循环将采集到的数据读出，并送到前面板窗口中的 Waveform 中显示。在 While 循环中首先利用 hpe1432 blockAvailable.vi 节点检测 VT1432A 数字化仪的 FIFO 缓冲区的状态，若 FIFO 缓冲区中达到了指定的数据长度，则利用 hpe1432 ReadFloat64Data.vi 节点将数据从 FIFO 缓冲区中读出。

从例 5.4.1 可以看出，VPP 驱动程序所封装的函数意义非常明确，完全是根据函数所要实现的功能确定的，用户在使用时只需要了解函数中参数的含义就可以了，利用 VPP 驱动程序进行编程可以为用户节省大量的时间。例 5.4.1 所讲述的使用 VPP 驱动程序的方法具有一定的代表性，在编写其他 VXI 仪器模块或者其他类型的支持 VPP 的测试仪器的程序时，都可以使用该方法。

另外，上述方法用于寄存器基仪器编程可以节约大量的时间，对于消息基仪器或者 SCPI 命令翻译节点的寄存器基仪器来讲，可以利用 12.3 节所介绍的 VISA 编程的方法，直接使用 SCPI 命令来控制仪器，这种方法也很节省时间，但需要用户熟悉 SCPI 命令。

第 13 章 通 信

串行通信是工业现场仪器或设备常用的通信方式，网络通信则是构建智能化分布式自动测试系统的基础。本章首先对串行通信和网络通信的各种接口协议和基本参数进行了简要介绍，然后结合 LabVIEW 提供的串行通信和网络通信节点编程实例，介绍了 LabVIEW 串行和网络通信编程的基本特点与步骤。DataSocket 和 Remote Panels 通信是 LabVIEW 为适应互联网时代发展而开发的便捷通信方式，DataSocket 可用于测试数据高速实时发布，而 Remote Panels 支持直接操作位于远程计算机上的 VI 面板。本章最后对 LabVIEW 支持的 DataSocket 和 Remote Panels 编程方式与应用进行了论述。

13.1 串 行 通 信

串行通信的通信方式是将一条信息的各位数据按顺序逐位传送。串行通信是一种古老但目前仍较为常用的通信方式，早期的仪器、单片机等均使用串口与计算机进行通信。当然，目前也有不少仪器或芯片仍然使用串口与计算机进行通信，如 PLC、Modem、GPS OEM 电路板等。本节将详细介绍如何在 LabVIEW 中进行串行通信。

13.1.1 串口简介

计算机串行接口（简称串口）采用 RS232 协议，RS232（RS 是 Recommend Standard 的缩写）协议是历史较为悠久的一种通信协议，于 1969 年被国际组织认可。RS232 协议定义了串口的电气特性（如电压值）、机械特性（如接头形状）及功能特性（如脚位信号）等。协议允许一个发送设备连接到一个接收设备以传送数据，最大传输速度为 115200b/s。计算机串行口采用 Intel 8250 异步串行通信组件构成，COM1、COM2、COM3、COM4 的基地址分别为 3F8、2F8、3E8 和 2E8（十六进制）。

RS232 协议规定：逻辑 1 的电平为 $-3\sim-15\text{V}$ ，逻辑 0 的电平为 $+3\sim+15\text{V}$ ，常用的信号有 8 个：RXD、TXD、RTS、DTR、CD、DSR、CTS、BELL，其中 RXD、TXD 为收、发数据，可与 RS232 串行口设备直接进行通信，RTS、DTR、CD、DSR、CTS、BELL 为控制与检测 MODEM 的信号，在通信过程中起联络与控制作用。

RS-232C 标准是 RS232 协议中常用的标准，其接口有 25 条线，4 条数据线、11 条控制线、3 条定时线、7 条备用和未定义线，常用的只有 9 根，分别介绍如下。

1. 联络控制信号线

联络控制信号线包含以下几种。

(1) DSR 与 DTR

数据装置准备好 (Data set ready-DSR) ——有效 (ON) 状态, 表明通信装置处于可以使用的状态。

数据终端准备好 (Data set ready-DTR) ——有效 (ON) 状态, 表明数据终端可以使用。

这两个信号有时连到电源上, 一上电就立即有效。这两个设备状态信号有效, 只表示设备本身可用, 并不说明通信链路可以开始进行通信了, 能否开始进行通信要由下面的控制信号决定。

(2) RTS 与 CTS

请求发送 (Request to send-RTS) ——用来表示 DTE 请求 DCE 发送数据, 即当发送设备要发送数据时, 使该信号有效 (ON 状态), 向接收设备请求发送。它用来控制发送设备是否要进入发送状态。

允许发送 (Clear to send-CTS) ——用来表示 DCE 准备好接收 DTE 发来的数据, 是对请求发送信号 RTS 的响应信号。当接收设备已准备好接收发送设备传来的数据, 并向前发送时, 使该信号有效, 通知发送设备开始沿发送数据线 TxD 发送数据。

这对 RTS/CTS 请求应答联络信号是用于半双工系统中发送方式和接收方式之间的切换。在全双工系统中作发送方式和接收方式之间的切换。在全双工系统中, 因配置双向通道, 故不需要 RTS/CTS 联络信号, 使其变高。

(3) RLSD

接收线信号检出 (Received Line detection-RLSD) ——用来表示 DCE 已接通通信链路, 告知 DTE 准备接收数据。

(4) RI

振铃指示 (Ringin-RI) ——当接收设备收到发送设备送来的振铃呼叫信号时, 使该信号有效 (ON 状态), 通知发送设备, 已被呼叫。

2. 数据发送与接收线

(1) TxD

发送数据 (Transmitted data-TxD) ——通过 TxD 终端将串行数据发送到接收设备。

(2) RxD

接收数据 (Received data-RxD) ——通过 RxD 线终端接收从发送设备发来的串行数据。

3. 地线

有两根线 SG、PG——信号地和保护地信号线, 这两根线无方向。

目前计算机上常用的串口有 9 个引脚, 这是从 RS-232C 规标准简化而来的, 这些引脚定义如图 13.1.1 和表 13.1.1 所示。

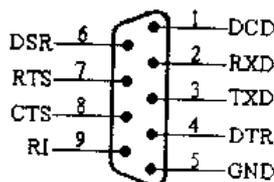


图 13.1.1 RS232 接口各引脚定义

表 13.1.1 RS232 接口各引脚定义

引脚号	缩写	作用	方向
1	DCD	数据载波检测	输入
2	RXD	接收数据	输入
3	TXD	发送数据	输出
4	DTR	数据终端准备就绪	输出
5	GND	信号地	
6	DSR	数据设备准备就绪	输入
7	RTS	请求发送	输出
8	CTS	清除发送	输入
9	RI	振铃指示	输入

RS232 协议可以说是一种最为简单的通信标准, 若不使用流控制 (Flow Control, 又称为握手协议), 最少需利用 3 根信号线, 如图 13.1.2 所示, 便可做到全双工的传输作业。但是, RS232 协议定义的电气特性属于非平衡传输方式, 抗干扰能力较弱, 故传输距离较短, 最大传输距离为 15m 左右。在 RS-232 协议基础上, 为了提高驱动能力, 又陆续推出了 RS-422 / RS-485 串行通信接口协议。虽然在接口电气特性这三者各有差别, 但驱动控制软件却是完全兼容的。

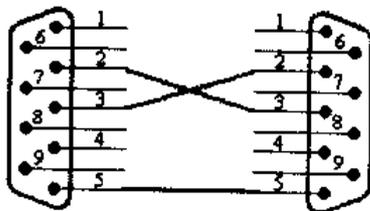


图 13.1.2 最简单的串口连接方式

利用上述串口连接方式, 在串行通信中存在的一个的问题是, 如何在数据传送过程中使接收者和发送者保持一致。串口可以将接收到的数据放入输入缓存, 但输入缓存的容量是有限的, 当输入缓存满时, 设备就会忽略送来新的数据, 直至从输入缓存中读出数据为新数据腾出足够的空间为止。这样就造成了数据的丢失。

使用流控制可以帮助避免缓存溢出。通过流控制, 发送者和接收者可以在缓存将满时相互通报, 发送者暂缓发送新的数据, 直至接收者做好接收新数据的准备。流控制可分为软件流控制和硬件流控制。

(1) 硬件流控制

硬件流控制常用的有 RTS/CTS (请求发送/清除发送) 流控制和 DTR/DSR (数据终端就绪/数据设置就绪) 流控制。

(2) 软件流控制

一般通过 XON/XOFF 来实现软件流控制。当输入缓存快满时, 接收者就会发送 XOFF

(Ctrl-S, decimal 19) 信号, 通知发送者停止发送数据; 当输入缓存完全清空时, 接收者发送 XON (Ctrl-Q, decimal 17) 信号, 通知发送者继续发送数据。

若采用流控制, 则串口之间应当按照如图 13.1.3 所示的方式连接。

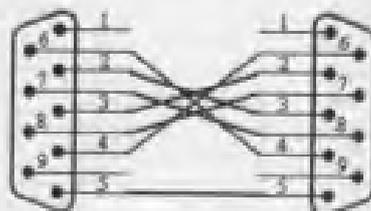


图 13.1.3 最完整的串口连接方式

下面介绍如何在 LabVIEW 中实现串行通信。

13.1.2 串行通信节点

LabVIEW 中用于串行通信的节点实际上是 VISA 节点, 为了方便用户使用, LabVIEW 将这些 VISA 节点单独组成一个子模板, 共包括 6 个节点, 分别实现初始化串口、串口写、串口读、检测串口缓存、中断以及关闭串口等功能, 这些节点位于 Functions 模板 → All Functions 子模板 → Instrument I/O 子模板 → Serial 子模板中, 如图 13.1.4 所示。

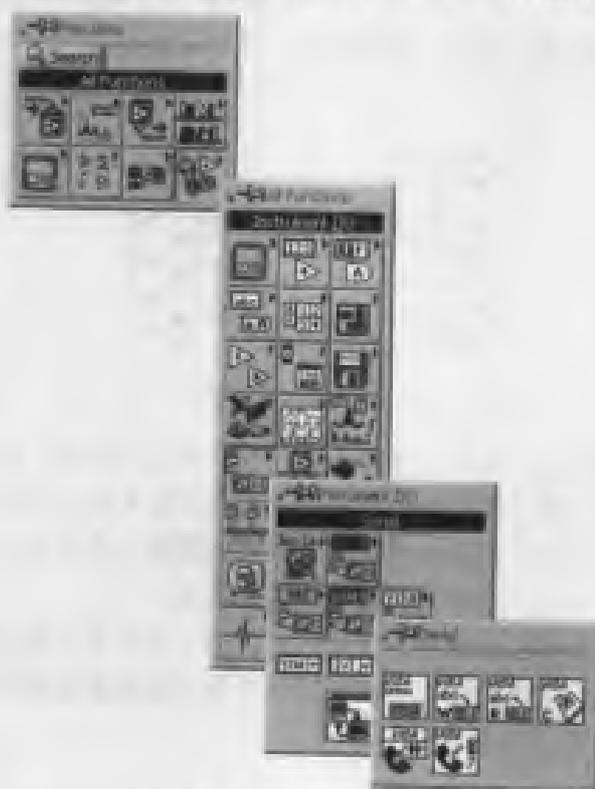
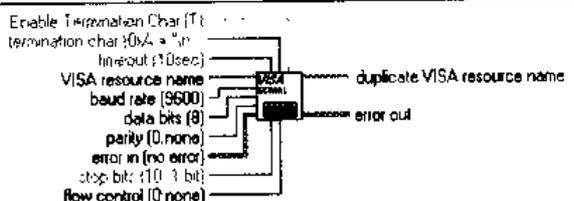
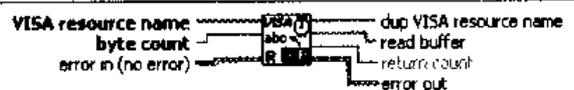


图 13.1.4 Serial 子模板

串行通信节点的使用方法比较简单, 且易于理解, 表 13.1.2 详细列出了各节点的参数定义、用法及功能。

表 13.1.2 串行通信节点功能表

节点名称	图标及端口	功 能
VISA Configure Serial Port.vi		初始化、配置串口。用该节点可以设置串口的波特率、数据位、停止位、奇偶校验、缓存大小以及流量控制等参数。各端口参数设置详表 13.1.2
VISA Write		将把 write buffer 端口输入的数据写入由 VISA resource name 端口指定的设备中。可用于将字符串写入串口的输出缓存
VISA Read		从由 VISA resource name 端口指定的设备中由 byte count 端口指定长度的数据。可用于从串口缓存中读出指定长度的数据
VISA Close		关闭由 VISA resource name 端口指定的设备连接。可用于关闭一个已经打开的串口，从而释放 LabVIEW 对这个串口资源的占用
VISA Bytes at Serial Port		返回串口的输入缓存中存在的数据的字节数。在使用 VISA Read 节点读串口前，可以先用 VISA Bytes at Serial Port 节点检测当前串口输入缓存中存在的字节数，然后由此指定 VISA Read 节点从串口输入缓存中读出的字节数，可以保证一次将串口输入缓存中的数据全部读出
VISA Serial Break		将制定的输出端口中断一段时间(至少 250 ms)，该时间由 delay 端口指定，单位为毫秒

VISA Configure Serial Port.vi 节点用于初始化串口，在利用计算机控制串口仪器设备时，会经常用到这个节点。在进行串行通信前，首先要配置好串口，也即先初始化串口，使计算机串口的各种参数设置与仪器设备的串口保持一致，这样才能够正确地通信。VISA Configure Serial Port 节点的各个端口参数见表 13.1.3。

表 13.1.3 VISA Configure Serial Port.vi 节点端口参数表

端口名称	参数设置
串口号 (VISA resource name)	可选的串口号为： ASRL1::INSTR: COM1 (默认值) ASRL2::INSTR: COM2 注：其他节点的 port number 端口的设置与此相同
波特率 (baud rate)	可选波特率参数为： 110 300 1200 4800 9600 (默认值) 19200 38400 57600 115200

续表

端口名称	参数设置
数据位 (data bits)	可选数据位参数为: 5 6 7 8 (默认值)
停止位 (stop bits)	可选停止位参数为: 10: 1 bit (默认值) 15: 1.5 bits 20: 2 bits
奇偶校验 (parity)	可选的奇偶校验参数为: 0: no parity (无校验, 默认值) 1: odd parity (奇校验) 2: even parity (偶校验) 3: mark parity (标记) 4: space parity (空)
流控制 (flow control)	0 默认值, 不使用 flow control, 此时串口的输入缓存被假设为有足够的空间, 可以接受所有的数据。实际上串口的输入缓存是有限的, 若不即使读出收到的数据, 可能会造成数据丢失
	1 XON/XOFF——使用字符 XON 和 XOFF 作为 flow control。当接收设备的串口输入缓存快溢出时, 该设备发送字符 XOFF; 当发送设备收到字符 XOFF 时, 暂缓发送数据, 直到收到接收设备发送来的 XON 时为止
	2 RTS/CTS——使用输出信号 RTS 和输入信号 CTS 作为 flow control。当接收设备的串口输入缓存快溢出时, 该设备发送 RTS 信号; 当发送设备收到 CTS 信号时, 暂缓发送数据
	3 XON/XOFF 和 RTS/CTS——使用字符 XON 和 XOFF 以及输出信号 RTS 和输入信号 CTS 作为 flow control。当接收设备的串口输入缓存快溢出时, 该设备发送字符 XOFF 和 RTS 信号; 当发送设备收到字符 XOFF 和 CTS 信号时, 暂缓发送数据
	4 DTR/DSR——使用输出信号 DTR 和输入信号 DSR 作为 flow control。当接收设备的串口输入缓存快溢出时, 该设备发送 DTR 信号; 当发送设备收到 DSR 信号时, 暂缓发送数据
	5 XON/XOFF 和 DTR/DSR——使用字符 XON 和 XOFF 以及输出信号 DTR 和输入信号 DSR 作为 flow control。当接收设备的串口输入缓存快溢出时, 该设备发送字符 XOFF 和 DTR 信号; 当发送设备收到字符 XOFF 和 DSR 信号时, 暂缓发送数据
激活终止符 (Enable Termination Char)	设定终止符 termination char 是否有效。当该端口输入 True (默认值) 时, 终止符有效; 当该端口输入 False 时, 终止符无效
终止符 (termination char)	该端口用于确定读操作的终止位置, 当从串口缓存中读出 termination char 端口指定的终止符时, 读操作终止。0xA 是十六进制的换行符 (\n), 0xD 是十六进制的回车符 (\r), 0xA 和 0xD 都可作为终止符
超时 (timeout)	设置读/写操作所等待的超时时间
复制的串口号 (duplicate VISA resource name)	该端口输出的是 VISA resource name 的一个拷贝, 以方便后续的串行通信节点使用

13.1.3 串行通信编程举例

例 13.1.1 双机串行通信。

本例使用两台计算机进行通信，一台计算机作为服务器，通过串口向外发送数据；另一台计算机作为客户机，接收由服务器发送来的数据。两台计算机之间利用一条串口数据线将连接起来，串口数据线两端的串口引脚的接线顺序如图 13.1.2 所示。

两台计算机之间的串行通信流程如图 13.1.5 所示。



图 13.1.5 串行通信流程图

服务器的前面板及框图程序如图 13.1.6 所示。

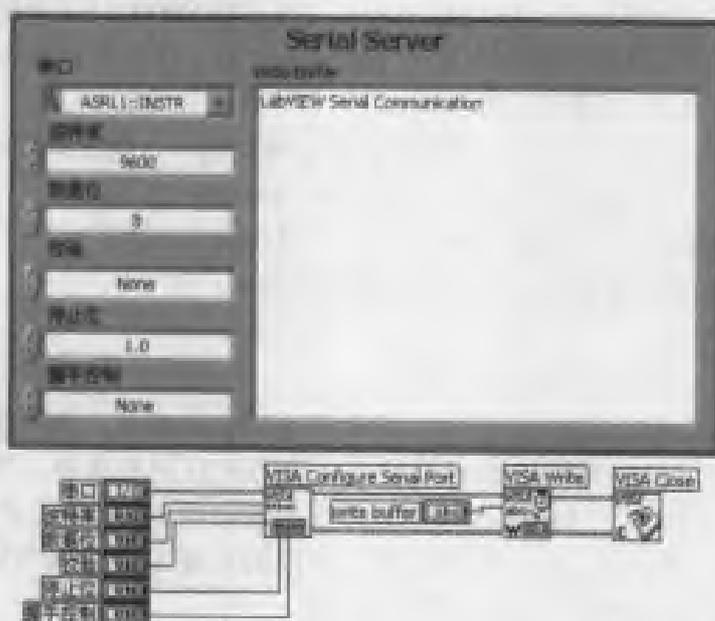


图 13.1.6 服务器程序的前面板及程序框图

客户机的前面板及程序框图如图 13.1.7 所示。



图 13.1.7 客户机程序的前面板及程序框图

例 13.1.2 与 PLC 进行串行通信。

PLC 的全称是 Programmable Logic Controller (可编程控制器), 是一种成熟的工业控制技术, 在工业控制领域得到了广泛地应用。PLC 利用串口与计算机进行通信, 本例以松下 FP0-C32 小型 PLC 进行串行通信为例, 介绍在 LabVIEW 中如何使用串行通信功能实现与 PLC 的通信。PLC 与计算机之间通过一条串口数据线相连接。

本例向 PLC 发送一条命令, 将 PLC 中的 0 号寄存器 R0000 中的数据位置 1, 并接受 PLC 返回的信息。发送的命令是“%01#WCSR0000123r”, PLC 收到该命令后, 返回响应字符串“%01\$WC14r”。其通信过程如下。

第一步, 初始化串口, 设置串口的通信参数与 PLC 的串行通信参数一致。

第二步, 向 PLC 中发送命令字符串“%01#WCSR0000123r”。

第三步, 延时 50ms, 等待 PLC 执行命令, 并返回相应字符串。

第四步, 从串口输入缓存中读出 PLC 的响应字符串。

第五步, 关闭串口。

本例的程序框图及结果如图 13.1.8 所示。

值得一提的是, PLC 在工业控制中具有举足轻重的地位, 具有其他控制技术无法比拟的优势, 而 LabVIEW 在测控软件方面也有其独到的优势, 因此, 利用 PLC 作为控制系统的硬件核心, 利用 LabVIEW 开发控制系统软件, 将二者有机结合起来, 发挥各自的优势, 可以开发出一套动能强大的控制系统。建议该领域的用户在开发工业控制系统时, 采用 PLC+LabVIEW 的方案。

注意, 当在 LabVIEW 中利用 VISA Configure Serial Port.vi 节点初始化了一个串口之后, 若在串行通信结束后没有利用 VISA Close 节点将该串口关闭, 那么, 只要没有退出 LabVIEW, LabVIEW 会一直占用该串口资源, 其他外部的程序在此时是不能访问该串口的。

另外, 还有一点值得注意的是, 串口只要初始化一次即可, 要尽量避免重复初始化串口 (除非改变其参数), 否则有可能降低系统的运行效率。

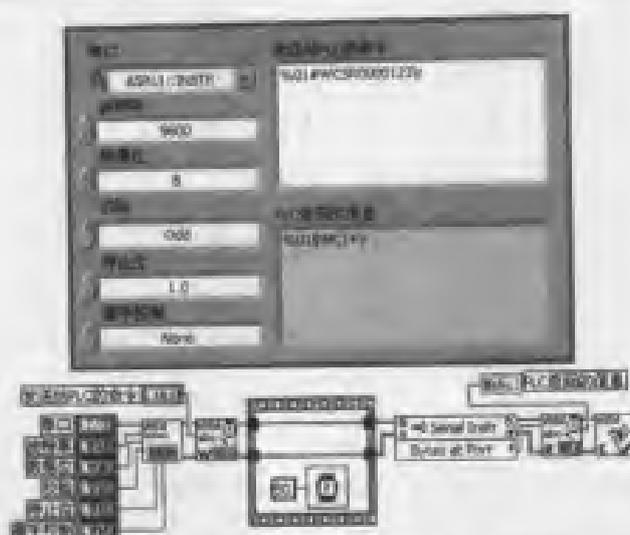


图 13.1.8 例 13.1.2 的前面板及程序框图

13.2 网络通信

LabVIEW 提供了强大的网络通信功能, 包括 TCP、UDP、.NET、SMTP-Email、IrDA、DataSocket、ActiveX 及远程面板等。其中基于 TCP 协议的通信方式是最为基本的网络通信方式, 本节将详细介绍如何在 LabVIEW 中实现基于 TCP 协议的网络通信。

13.2.1 TCP 协议简介

TCP 协议是 TCP/IP 协议中的一个子协议。TCP/IP 是 Transmission Control Protocol/Internet Protocol 的简写, 中文译名为传输控制协议/互联网络协议。TCP/IP 协议是 Internet 最基本的协议, TCP/IP 协议是 20 世纪 70 年代中期美国国防部为其 ARPANET 广域网开发的网络体系结构和协议标准, 以它为基础组建的 Internet 是目前国际上规模最大的计算机网络, Internet 的广泛使用, 使得 TCP/IP 成了事实上的标准。TCP/IP 实际上是一个由不同层次上的多个协议组合而成的协议族, 共分为四层: 链路层、网络层、传输层和应用层, 如图 13.2.1 所示。从图 13.2.1 可以看出 TCP 协议是 TCP/IP 传输层中的协议, 使用 IP 作为网络层协议。

TCP (Transmission Control Protocol, 传输控制协议) 协议使用不可靠的 IP 服务, 提供一种面向连接的、可靠的传输层服务, 面向连接是指在数据传输前就建立好了点到点的连接。大部分基于网络的软件都采用 TCP 协议。TCP 采用比特流 (即数据被作为无结构的字节流) 通信分段传送数据, 主机交换数据必须建立一个会话。通过每个 TCP 传输的字段指定顺序号, 以获得可靠性。如果一个分段被分解成几个小段, 接收主机会知道是否所有小段都已收到。通过发送应答, 用以确认别的主机收到了数据。对于发送的每一个小段, 接收主机必须在一个指定的时间返回一个确认。如果发送者未收到确认, 发送者会重新发送数据; 如果收到的数据包损坏, 接收主机会将其舍弃, 因为确认未被发送, 发送者会重新发送分段。

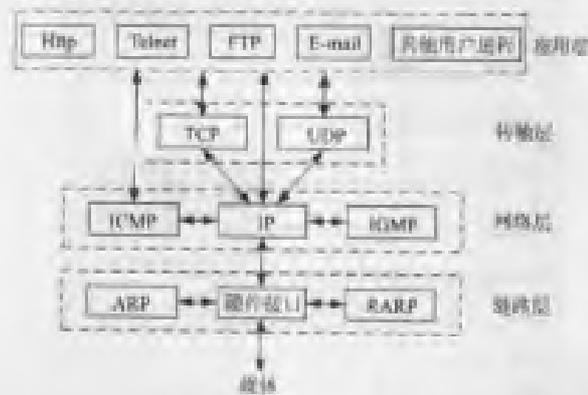


图 13.2.1 TCP/IP 协议栈层次图

TCP 对话通过三次握手来初始化，目的是使数据段的发送和接收同步，告诉其他主机其一次可接收的数据量，并建立虚连接。三次握手的过程如下。

第一步，初始化主机通过一个具有同步标志置位的数据段发出会话请求。

第二步，接收主机通过发回具有以下项目的数据段表示回复：同步标志置位、即将发送的数据段的起始字节的顺序号、应答并带有将收到的下一个数据段的字节顺序号。

第三步，请求主机再回送一个数据段，并带有确认顺序号和确认号。

在 LabVIEW 中可以利用 TCP 协议进行网络通信，并且，LabVIEW 对 TCP 协议的编程进行了高度集成，用户通过简单编程更就可以在 LabVIEW 中实现网络通信。

13.2.2 TCP 节点

在 LabVIEW 中，可以采用 TCP 节点来实现局域网通信，TCP 节点在 Functions 模板 → All Functions 子模板 → Communication 子模板 → TCP 子模板中，如图 13.2.2 所示。

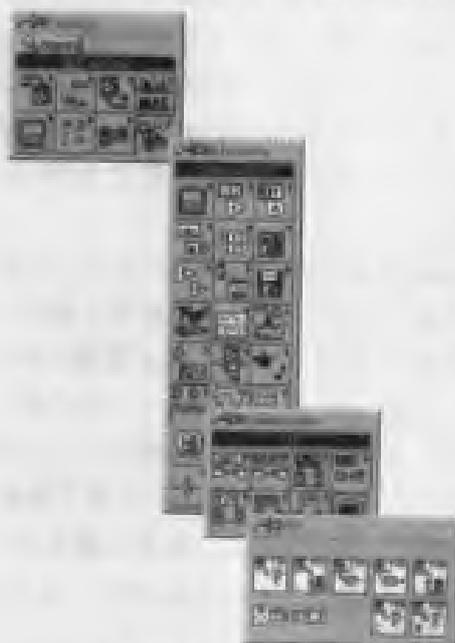
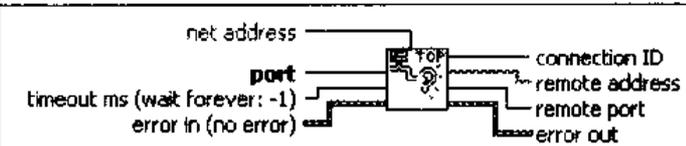
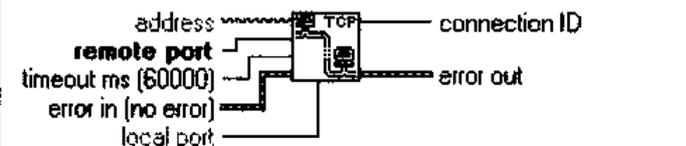
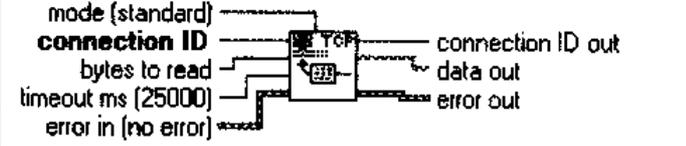
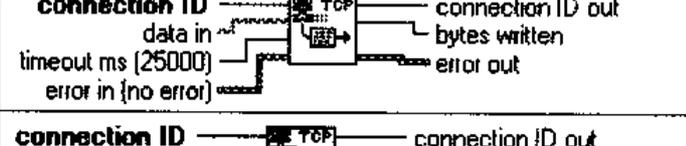
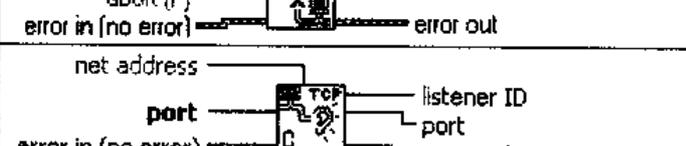
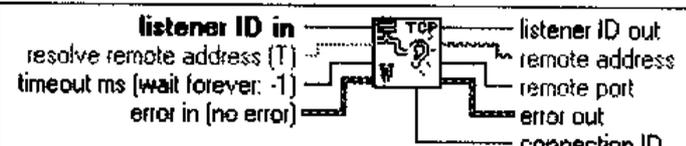


图 13.2.2 TCP 子模板

TCP 节点的使用方法比较简单，并且易于理解，表 13.2.1 详细列出了各节点的参数定义、用法及功能。

表 13.2.1 TCP 节点功能表

节点名称	图标及端口	功 能	备 注
TCP Listen .vi		创建一个听者，并在指定的端口上等待 TCP 连接请求	该节点只能作为服务器的计算机上使用
TCP Open Connection		用指定的计算机名称和远程端口来打开一个 TCP 连接	该节点只能作为客户机的计算机上使用
TCP Read		从指定的 TCP 连接中读出数据	数据的长度由 bytes to read 端口指定
TCP Write		通过 data in 端口将数据写入指定的 TCP 连接中	
TCP Close Connection		关闭指定的 TCP 连接	
TCP Create Listener		创建一个 TCP 连接的听者 (Listener)	
TCP Wait On Listener		在指定的端口上等待 TCP 连接请求	TCP Listen .vi 节点就是 TCP Create Listener 节点与本节点的综合使用
IP to String		将 IP 地址转化为计算机名称	可用 dot notation 端口来设置 name 端口的输出格式是否为 dot notation 格式
String to IP		将计算机名称转化为 IP 地址	如不指定计算机名称，则节点输出当前计算机的 IP 地址

13.2.3 TCP 通信编程实例

例 13.2.1 利用 TCP 协议进行双机通信。

采用服务器 / 客户机模式进行双机通信，是在 LabVIEW 中进行网络通信的最基本的结构模式。本例由服务器产生一组随机波形，通过局域网送至客户机进行显示。双机通信的流程如图 13.2.3 所示。



图 13.2.3 双机通信流程图

服务器的前面板及框图程序如图 13.2.4 所示。

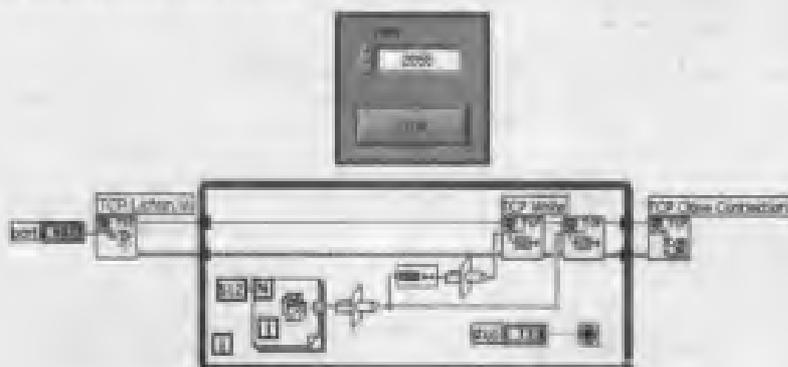


图 13.2.4 服务器程序的前面板及框图程序

在服务器的框图程序中，首先指定网络端口 (Port)，并用 TCP Listen 节点建立 TCP 听者，等待客户机的连接请求，这是初始化的过程。

框图程序采用了两个 TCP Write 节点来发送数据：第一个为 TCP Write 节点发送的数据是随机波形的长度；第二个为 TCP Write 节点发送随机波形数据。

这种发送方式有利于客户机接收数据。客户机的前面板及程序框图如图 13.2.5 所示。

与服务器框图程序相对应，客户机框图程序也采用了两个 TCP Read 节点读出由服务器送来的随机波形的数据。第一个节点读出随机波形的长度，然后第二个节点根据这个长度将随机波形的数据全部读出。这种方法是 TCP / IP 通信中常用的方法，可以有效地发送、接收数据，并保证数据不丢失。建议用户在使用 TCP 节点进行双机通信时采用这种方法。

注意，在用 TCP 节点进行通信时，需要在服务器框图程序中指定网络通信端口 (Port)，客户机也要指定相同的端口，才能与服务器之间进行正确的通信，如上例中的端口值为 2055。端口值由用户任意指定，只要服务器与客户机的端口保持一致即可。在一次通信连接建立后，就不能更改端口的值了。如的确需要改变端口值，则必须首先断开连接，才能重新设置端口值。

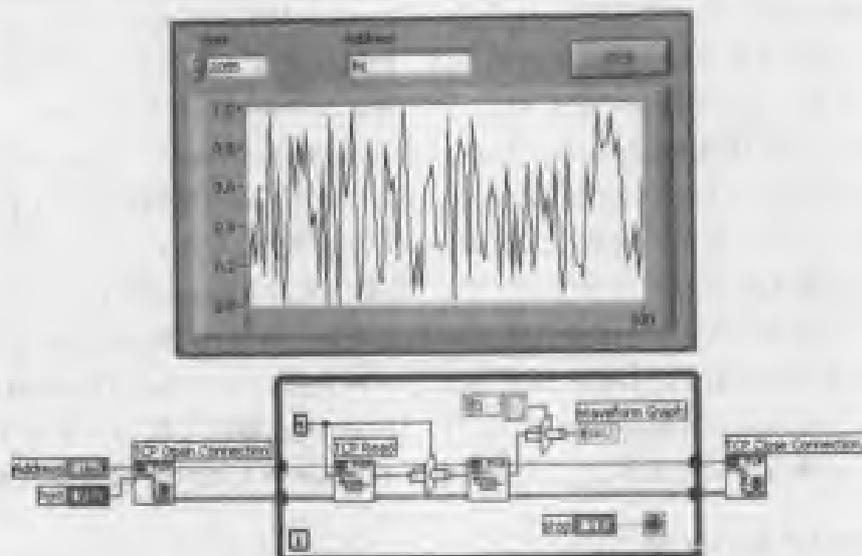


图 13.2.5 客户机程序的前面板及框图程序

还有一点值得注意的是，在客户机框图程序中首先要指定服务器的名称才能与服务器之间建立连接。服务器的名称是指服务器的计算机名。若服务器和客户机程序在同一台计算机上同时运行，客户机框图程序中输入的服务器的名称可以是 localhost，也可以是这台计算机的计算机名，甚至可以是一个空字符串。

13.3 DataSocket 通信

13.3.1 DataSocket 基本概念

DataSocket 是 NI 公司推出的一项基于 TCP/IP 协议的新技术，DataSocket 面向测量和网上实时高速数据交换，可用于一个计算机内或者网络中多个应用程序之间的数据交换。虽然目前已经有 TCP/IP、DDE 等多种用于两个应用程序之间共享数据的技术，但是这些技术都不是用于实时数据（Live Data）传输的。只有 DataSocket 是一项在测量和自动化应用中用于共享和发布实时数据的技术，如图 13.3.1 所示。

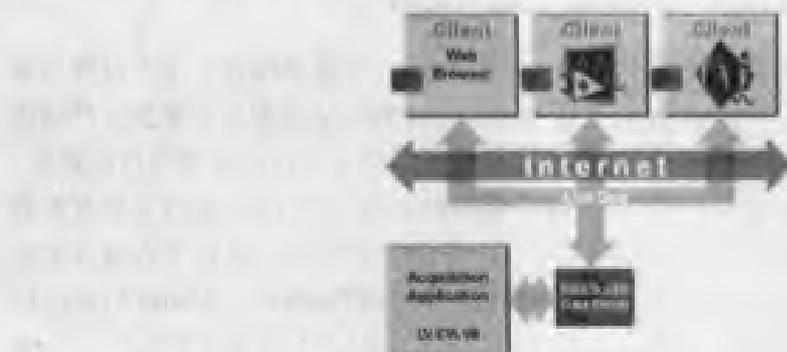


图 13.3.1 DataSocket 技术示意图

DataSocket 实际上是一个基于 URL 的单一的、一元化的末端用户 API, 是一个独立于协议、独立于语言以及独立于操作系统的 API。DataSocket API 被制作成 ActiveX 控件、LabWindows 库和一系列的 LabVIEW VIs, 用户可以在任何编程环境中使用。

DataSocket 包括 DataSocket Server Manager、DataSocketServer 和 DataSocket 函数库等三大部分, 以及 Dstp (DataSocket Transfer Protocol) 协议、通用资源定位符 URL (Uniform Resource Locator) 和文件格式等规程。DataSocket 遵循 TCP/IP 协议, 并对底层进行高度封装, 所提供的参数简单友好, 只需要设置 URL 就可用来在 Internet 进行及时分送所需传输的数据。用户可以像使用 LabVIEW 中的其他数据类型一样使用 DataSocket 读写字符串、整形数、布尔量及数组数据。DataSocket 提供了三种数据目标: file、DataSocket Server、OPC Server, 因而可以支持多进程并发。这样, DataSocket 摒除了较为复杂的 TCP/IP 底层编程, 克服了传输速率较慢的缺点, 大大简化了 Internet 网上测控数据交换的编程。

1. DataSocket Server Manager

DataSocket Server Manager 是一个独立运行的程序, 它的主要功能是设置 DataSocket Server 可连接的客户端程序的最大数目和可创建的数据项的最大数目, 创建用户组和用户, 设置用户创建数据项 (Data Item) 和读写数据项的权限。数据项实际上是 DataSocket Server 中的数据文件, 未经授权的用户不能在 DataSocket Server 上创建或读写数据项。DataSocket Server Manager 如图 13.3.2 所示。

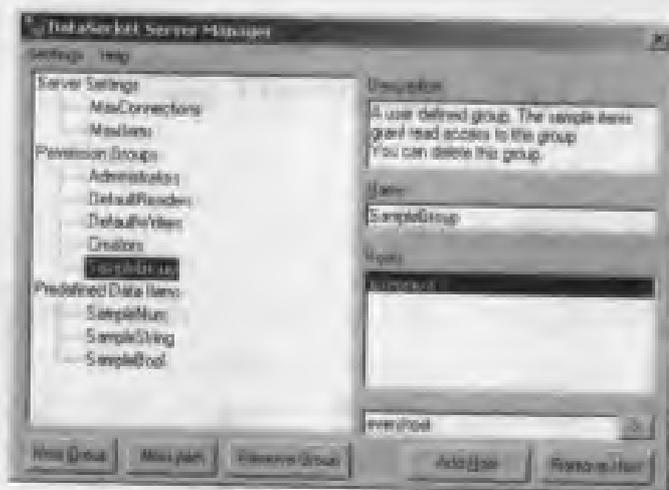


图 13.3.2 DataSocket Server Manager

DataSocket Server Manager 窗口左栏中的 Server Settings (服务器配置) 用于设置与服务器性能有关的参数: 参数 MaxConnections 是指 DataSocket Server 最多允许多少客户端连接到服务器, 其默认值为 50; 参数 MaxItem 用于设置服务器最大允许的数据项目的数量。

DataSocket Server Manager 窗口左栏中的 Permission Groups (许可组) 是与安全有关的部分设置, Groups (组) 是指用一个组名来代表一组 IP 地址的集合, 这对于以组为单位进行设置比较方便。DataSocket Server 共有 3 个内建组: DefaultReaders、DefaultWriters 和 Creators, 这 3 个组分别代表了能读、写以及创建数据项目的默认主机设置。可以利用 New Group 按钮来添加新的组。

DataSocket Server Manager 窗口左栏中的 Predefined Data Items (预定义的数据项目) 中预先定义了一些用户可以直接使用的数据项目, 并且可以设置每个数据项目的数据类型、默认值以及访问权限等属性。默认的数据项目共有 3 个: SampleNum、SampleString 和 SampleBool, 用户可以利用 New Item 按钮添加新的数据项目。

2. DataSocket Server

DataSocket Server 也是一个独立运行的程序, 它能为用户解决大部分网络通信方面的问题。它负责监管 DataSocket Server Manager 中所设定的各种权限和客户程序之间的数据交换。DataSocket Server 与测控应用程序可安装在同一台计算机上, 也可以分装在不同计算机上。后一种方法可增加整个系统的安全性, 因为两台计算机之间可用防火墙加以隔离。而且, DataSocket Server 程序不会占用测控计算机 CPU 的工作时间, 测控应用程序可以运行的更快。DataSocket Server 运行后的窗口如图 13.3.3 所示。

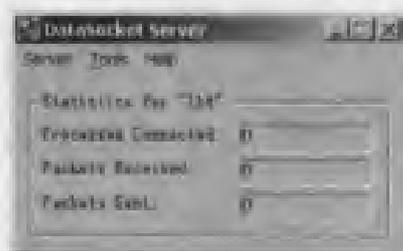


图 13.3.3 DataSocket Server 窗口

在安装了 LabVIEW 之后, 可以在 Windows 的开始菜单中运行 DataSocket Server, 运行路径如下

「开始」选单\程序\National Instruments\DataSocket\DataSocket Server

在 LabVIEW 中进行 DataSocket 通信之前, 必须首先运行 DataSocket Server。

3. DataSocket 函数库

DataSocket 函数库用于实现 DataSocket 通信。利用 DataSocket 发布数据需要三个要素: Publisher(发布器)、DataSocket Server 和 Subscriber(订阅器)。Publisher 利用 DataSocket API 将数据写到 DataSocket Server 中, 而 Subscriber 利用 DataSocket API 从 DataSocket Server 中读出数据, 如图 13.3.4 所示。Publisher 和 Subscriber 都是 DataSocket Server 的客户程序。这三个要素可以驻留在同一台计算机中。



图 13.3.4 DataSocket 通信过程

13.3.2 DataSocket 节点

在 LabVIEW 中, 利用 DataSocket 节点就可以完成 DataSocket 通信。DataSocket 节点位于 Functions 模板→All Functions 模板→Communication 子模板→DataSocket 子模板中, 如图 13.3.5 所示。

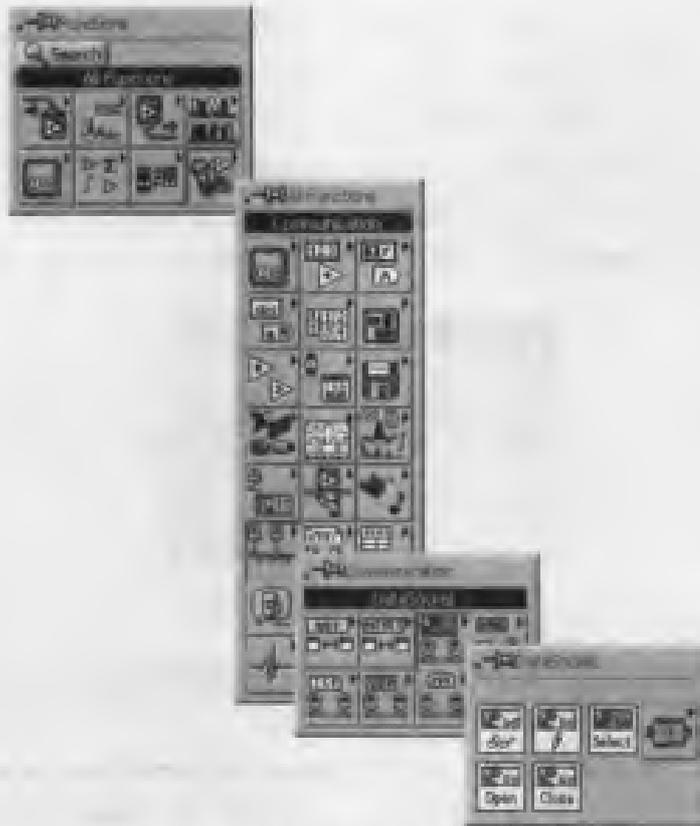


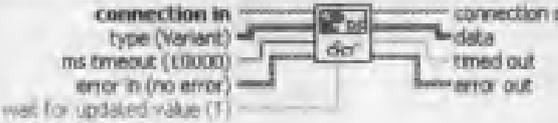
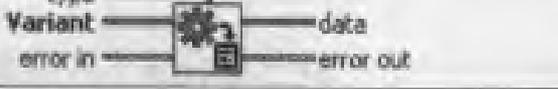
图 13.3.5 DataSocket 子模板

LabVIEW 将 DataSocket 函数库的功能高度集成到了 DataSocket 节点中, 与 TCP/IP 节点相比, DataSocket 节点的使用方法更为简单和易于理解。DataSocket 节点分为 DataSocket 通信节点和 DataSocket 变量转换节点两大类, DataSocket 通信节点用于完成 DataSocket 通信, DataSocket 变量转换节点用于完成 DataSocket 节点所使用的 Variant 变量和其他所有类型的变量之间的转换。表 13.3.1 详细列出了 DataSocket 节点的参数定义、用法及功能。

表 13.3.1 DataSocket 节点功能表

节点名称	图标及端口	功能
DataSocket 通信节点		
DataSocket Open		打开一个用户指定 URL 的 DataSocket 连接

续表

节点名称	图标及端口	功能
DataSocket Write		将数据写到由 connection in 端口指定的 URL 连接中。数据可以是单个或数组形式的字符串、逻辑(布尔)量和数值量等多种类型。connection in 端口可以是一个 DataSocket URL 字符串,也可以是一个 aSocket connection refnum (即 DataSocket Open 节点返回的 connection id)。
DataSocket Read		从由 connection in 端口指定的 URL 连接中读出数据。connection in 端口可以是一个 DataSocket URL 字符串,也可以是一个 aSocket connection refnum (即 DataSocket Open 节点返回的 connection id)。
DataSocket Close		关闭一个 DataSocket 连接。
DataSocket Select URL		弹出一个 Select URL 对话框,用户在这个对话框中可以搜索网上存在的 URL 数据库,如图 13.3.6 所示。节点最后返回一个 URL 地址。
DataSocket 变量转换节点		
to Variant		将任意的数据类型转换为 Variant 类型。
Variant to Data		将 Variant 类型的数据转换为 LabVIEW 可以显示和处理的数据类型。
Variant Flattened String		将 Variant 类型的数据转换为 Flattened String 类型数据和一个表示数据类型的数据数组。
Flattened String to Variant		将 Flattened String 类型数据转换为 Variant 类型的数据。
Get Variant Attribute		获得与 Variant 类型的数据相关联的属性和数值。
Set Variant Attribute		为 Variant 类型的数据改变或创建一个属性和数值。
Delete Variant Attribute		删除与 Variant 类型的数据相关联的属性和数值。

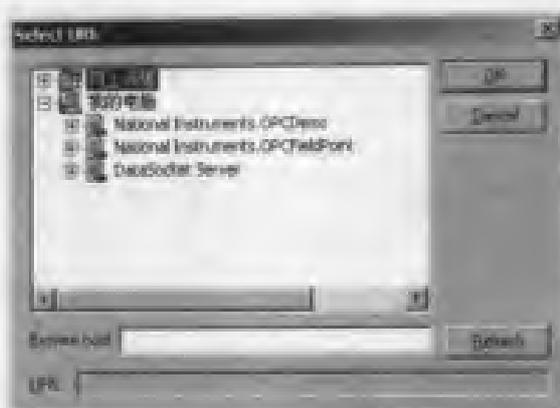


图 13.3.6 Select URL 对话框

与 TCP/IP 通信一样，利用 DataSocket 进行通信时也需要首先指定 URL，DataSocket 可用的 URL 共有下列四种：Dstp、Opc、logos 和 file 传输协议。

(1) Dstp

DataSocket 传输协议 Dstp 是 DataSocket 的固有协议，当使用这个协议时，VI 与 DataSocket Server 连接，用户必须为数据提供一个附加到 URL 的具有名称的 Tag。DataSocket 连接利用这个 Tag 在 DataSocket server 上为一个特殊的数据项目指定地址。利用这个协议，用户可以运行一个 DataSocket Server。

(2) Opc

Opc 是 Windows OLE for Process Control，特别为实时产生数据而设计，例如工业自动化操作而产生的数据。要使用该协议，必须首先运行一个 Opc Server。

(3) Logos

Logos 是一个 NI 公司的内部技术，用于在网络和本地计算机之间传输数据。

(4) File 传输协议

File 传输协议可用于提供一个到包含数据的本地文件或网络文件的连接。

表 13.3.2 列举了上述四种协议下的 URL 实例。

表 13.3.2 URL 的应用举例

协议	举 例
Dstp	dstp://servername.com/numericdata (numericdata 是具有名称的 tag)
Opc	opc://National Instruments.OPCTest/item1
	opc://machine/National Instruments.OPCModbus/Modbus Demo Box.4:0
	opc://machine/National Instruments.OPCModbus/Modbus Demo Box.4:0?updateRate=100&deadband=0.7
Logos	logos://computer_name/process/data_item_name
File	file://ping.wav
	filec://mydata/ping.wav
	file://machine/mydata/ping.wav

由于 `dstp`、`opc` 和 `logos` 协议能够更新远程和本地的控制量和指示量，所以可以利用这些协议下的 URL 共享实时数据；由于 `ftp` 和 `file` 协议不能更新远程和本地的控制量和指示量，所以可以利用 `ftp` 和 `file` URLs 从文件中读取数据。

注意，利用 `DataSocket` 节点进行通信的过程与利用 `TCP` 节点进行通信的过程相同，操作步骤如下。

第一步，利用 `DataSocket Open` 节点打开一个 `DataSocket` 连接。

第二步，利用 `DataSocket Write` 节点和 `DataSocket Read` 节点完成通信。

第三步，利用 `DataSocket Close` 节点关闭这个 `DataSocket` 连接。

另外，由于 `DataSocket` 功能的高度集成性，用户在进行 `DataSocket` 通信时，可以省略第一步和第三步，只利用 `DataSocket Write` 节点和 `DataSocket Read` 节点就可以完成通信了。

13.3.3 DataSocket 编程举例

例 13.3.1 DataSocket 应用实例之一。

本例包括一个服务器 VI (`DataSocket Writer`) 和一个客户机 VI (`DataSocket Reader`)，用以说明 `DataSocket` 节点的使用方法。

服务器 VI 产生一个由 40 个随机数组成的波形，并利用 `DataSocket Write` 节点将数据发布到 URL “`dstp://lht/wave`” 指定的位置中。服务器 VI 的前面板和程序框图如图 13.3.7 所示。

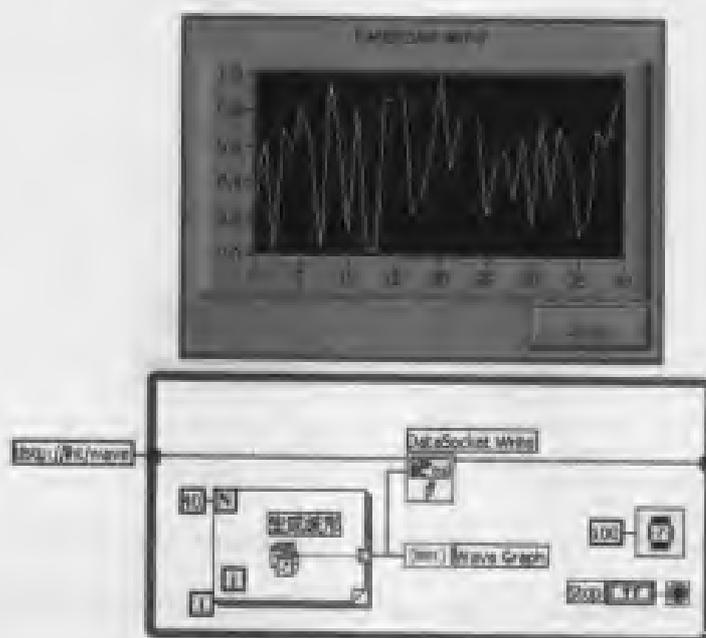


图 13.3.7 服务器 VI 的前面板和程序框图

客户机 VI 利用 `DataSocket Read` 节点将数据从 URL “`dstp://lht/wave`” 指定的位置读出，并还原为原来的数据类型送到前面板窗口中的 `Wave Graph` 指示中显示。客户机 VI 的前面板和程序框图如图 13.3.8 所示。

注意，在利用上述两个 VI 进行 `DataSocket` 通信之前，必须首先运行 `DataSocket Server`。

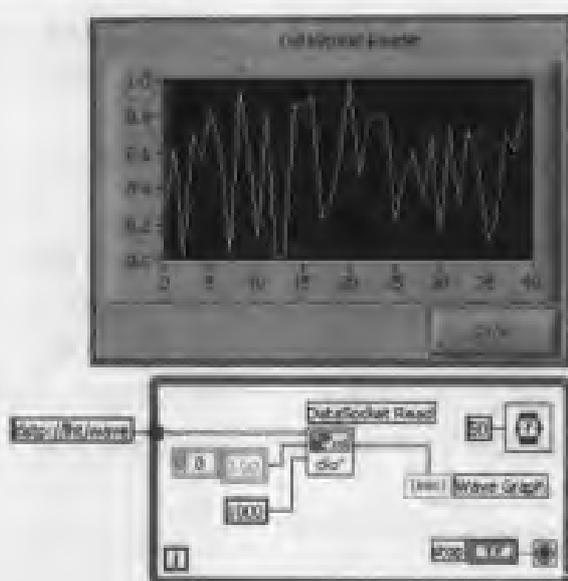


图 13.3.8 客户机 VI 的前面板和框图程序

上面的例子利用 LabVIEW 提供的 DataSocket 节点完成 DataSocket 通信, 这需要进行一些简单的编程。但是, LabVIEW 6i 以后的版本却提供了另外一种更加简单的方法来完成 DataSocket 通信。

LabVIEW 6i 以后的版本中所有的前面板对象都增加了一个叫做 DataSocket Connection 的特殊属性, 如图 13.3.9 所示。

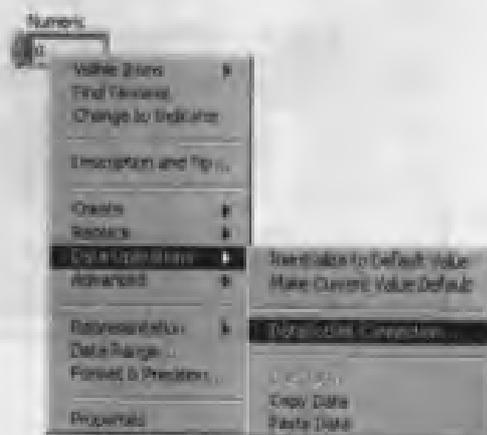


图 13.3.9 LabVIEW 前面板对象的 DataSocket Connection 属性

在控件的右键选单中选择这个属性, 就会弹出 DataSocket Connection 配置对话框, 如图 13.3.10 所示。

利用 DataSocket Connection 配置对话框可以完成对前面板对象 DataSocket Connection 属性的配置。这样, 不需要编程, 这个前面板对象就可以直接进行 DataSocket 通信了。注意, 如果为一个 LabVIEW 前面板对象设置了 DataSocket Connection 属性, 这个前面板对象的右上角就会出现一个小方框, 用于指示该对象的 DataSocket 连接状态。当小方框为灰色时, 表示该对象没有连接到 DataSocket Server 上, 当小方框为绿色时, 表示该对象已经连接到 DataSocket Server 上。

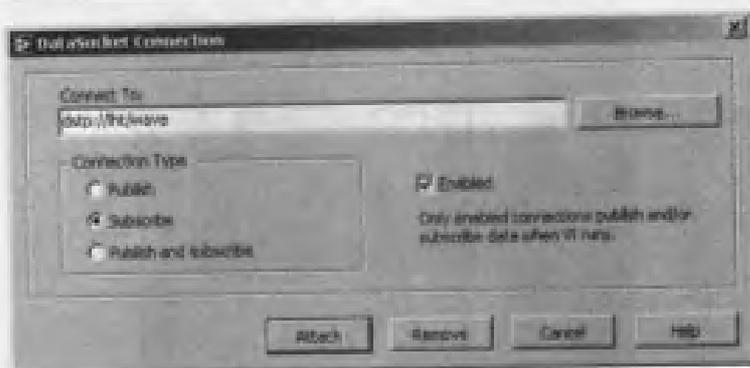


图 13.3.10 DataSocket Connection 属性的配置对话框

例 13.4.2 DataSocket 应用实例之二。

按照上述方法改进的 DataSocket 通信的例子的前面板和程序框图如图 13.3.11 和图 13.3.13 所示。两个 VI 中 Waveform 对象的 DataSocket Connection 属性的配置如图 13.3.12 和图 13.3.14 所示。

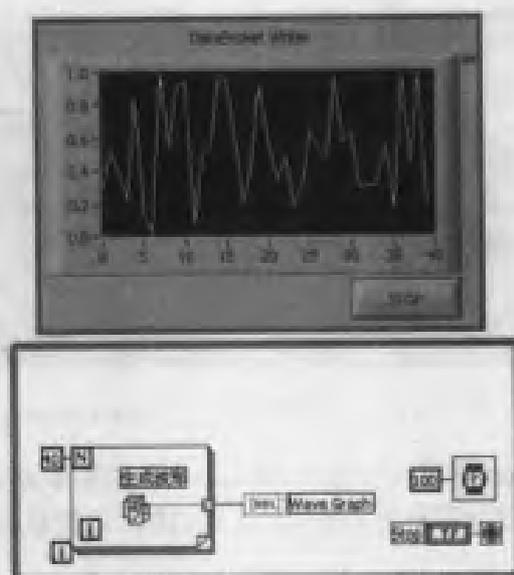


图 13.3.11 改进的服务器 VI 的前面板和程序框图

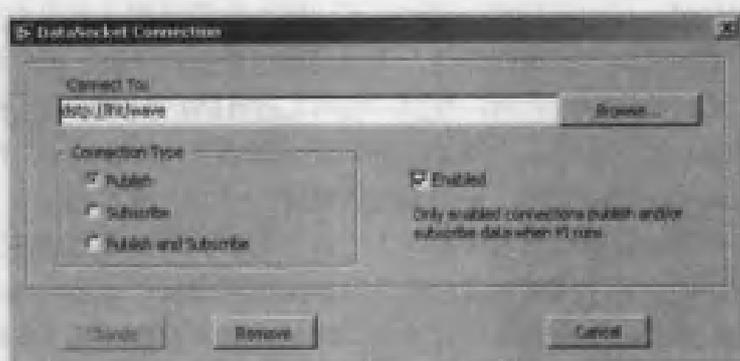


图 13.3.12 服务器 VI 中 Wave Graph 控件的 DataSocket Connection 属性的配置



图 13.3.13 改进的客户机 VI 的前面板和程序框图



图 13.3.14 客户机 VI 中 Wave Graph 控件的 DataSocket Connection 属性的配置

注意, 在使用 DataSocket 进行通信时, DataSocket 传输的数据文件一般不宜超过 1MB。使用 DataSocket 技术, 可以编写出更加强大的 LabVIEW 远程数据采集应用程序。

13.4 远程面板

在进行远程测控时, 用户往往希望能够在家中或办公室里的计算机中通过网络直接控制位于测控现场的测控系统完成测控任务, 不但要随时操作位于位于测控现场的主控计算机上的测控软件, 而且要实时观察测控数据, 利用 LabVIEW 提供的网络通信节点, 例如, TCP/IP、Remote Data Acquisition (RDA)、Internet Toolkit、VI Server、Front Panel Web Publishing、DataSockets...等, 加上一些高级编程技术和技巧, 也可以实现上述功能。这需要用户具有高深的网络知识并付出艰苦的努力, 而大多数的用户希望的是通过简单快捷的方式来实现上述功能。并且, 目前很多已经开发完成的测控软件并没有网络通信功能, 若需要实现上述功能, 必须对测控软件进行改造, 这一点需要花费一些工作时间, 并且有些情况下, 不允许用户自行修改测控软件。利用 LabVIEW 的远程面板 (Remote Panel) 技术, 不需要任何编程, 只需在 LabVIEW 中设置几个参数, 就可以轻松解决这个问题。

Remote Panel 技术, 允许用户直接在本地 (Client 端) 计算机上打开并操作位于远程

(Web Server 端) 计算机上的 VI 的前面板。

从 LabVIEW 6.1 开始, LabVIEW 集成了 Remote Panel 技术, 用户可以用极为简单的方式直接在本地 (Client 端) 计算机上打开并操作位于远程 (Web Server 端) 计算机上的 VI 的前面板, 甚至可以将 LabVIEW VIs 的前面板窗口嵌入到一个网页中并在网页中直接操作它。另外, 也有一些第三方公司也提供了一些 Remote Panel 发布的工具, 例如 LabVNC。图 13.4.1 所示的是 Remote Panel 发布示意图。

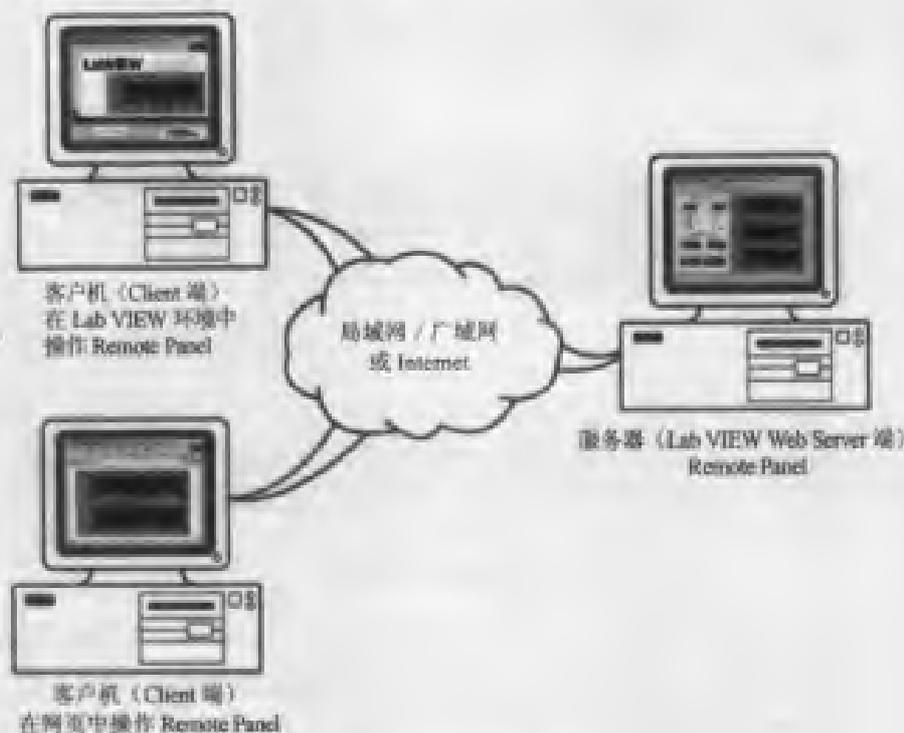


图 13.4.1 Remote Panels 发布示意图

在 LabVIEW 中设定并使用 Remote Panel 仅需两个步骤:

- 第一步, 在 LabVIEW Web Server 端的计算机上开启 LabVIEW Web Server 服务。
- 第二步, 在 Client 端计算机上连接并运行 Remote Panel。

目前, 有两种方式可以实现在 Client 端计算机进行 Remote Panel 操作:

- 在 LabVIEW 环境中直接操作 Remote Panel。
- 利用网页浏览器在网页中直接操作 Remote Panel。

13.4.1 配置 LabVIEW Web Server

在 Client 端使用 Remote Panel 之前, 必须首先在 Server 计算机上运行 LabVIEW, 并配置 Web Server, Web Server 需要下面三个方面的配置:

- 文件路径和网络设置。
- 客户机访问权限设置。
- VIs 访问权限设置。

具体的设置方法详见第 10.3.6 节。

为了提供网页浏览器访问，必须在配置服务器时增加一步，利用选单栏中的 Tools→Web Publish Tool 将网页发布出去，如图 13.4.2 所示。

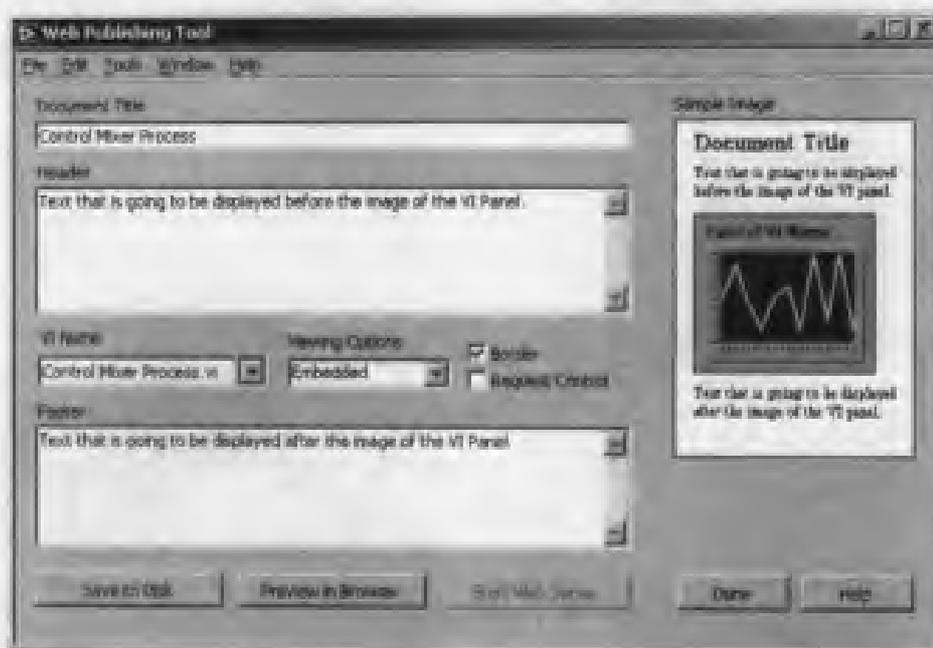


图 13.4.2 Web Publish Tool

Web Publish Tool 允许用户输入一个 VI 的名称，并自动生成一个 HTML 文件，当然，要将这个 HTML 文件保存在 Web Server Configuration 所指定的根目录中（请注意，这个根目录最好与 Windows IIS 的 Internet 信息服务中的默认 Web 站点的主目录相一致）。

如果用户想要发布的 VI 中包含了数个 SubVI，这些 SubVI 的前面板窗口在需要时也可以打开，那么用户只需要创建一个发布最上层 VI（top-level VIs）的网页，而其他所有 subVI 的前面板的属性设定为 Open During Execution 即可。这样，就可以在客户端打开这些 SubVI 的前面板。

13.4.2 在 LabVIEW 环境中操作 Remote Panel

完成上述的配置之后，就可以在 LabVIEW 环境中运行一个 Remote Panel 了。操作步骤如下：

第一步，在 Web Server 端计算机中打开一个 VI 的前面板窗口（必须要打开，否则客户端在连接这个 VI 时会出错），例如 Control Mixer Process.vi（LabVIEW 自带的的应用实例），如图 13.4.3 所示。

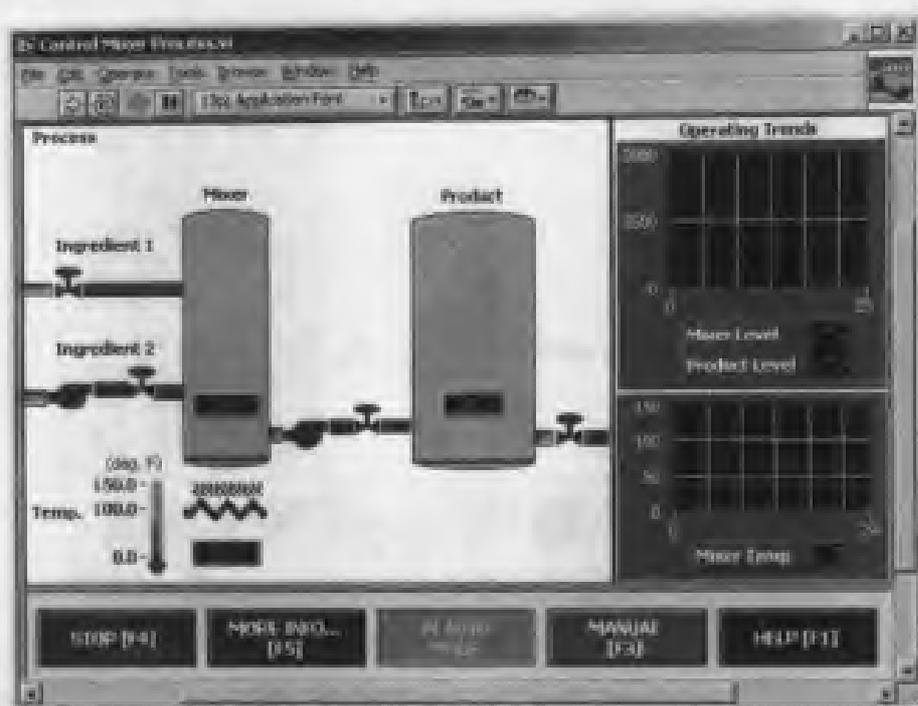


图 13.4.3 Control Mixer Process.vi 的前面板窗口

第二步,在 Client 端的 LabVIEW 的菜单栏中选择 Operate→Connect to Remote Panel..., 弹出 Connect to Remote Panel 对话框,如图 13.4.4 所示。

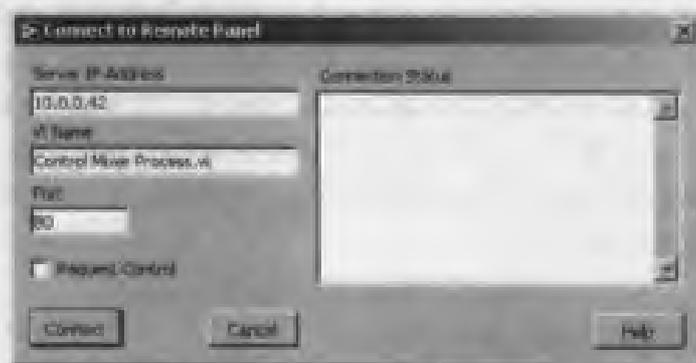


图 13.4.4 Connect to Remote Panel 对话框

第三步,在 Connect to Remote Panel 对话框的 Server IP Address 栏中,输入 Server 端计算机的 IP 地址、域名或计算机名,如 10.0.0.42、www.ni.com、lht 等;在 VI Name 栏中输入想要控制的远程 VI 的名称,如 Control Mixer Process.vi;在 Port 栏中输入 Web Server configuration 中所设定的 HTTP Port (默认值为 80);如果想要立即得到 Remote Panel 的控制权,请选中 Request Control 选项(当然,也可以在 Remote Panel 出现时单击鼠标右键来获得控制权)。

第四步,单击 Connect 按钮,Remote Panel 就会出现在屏幕上了,如图 13.4.5 所示。请注意图 13.4.3 和图 13.4.5 所示的两个前面板窗口是有所不同的。

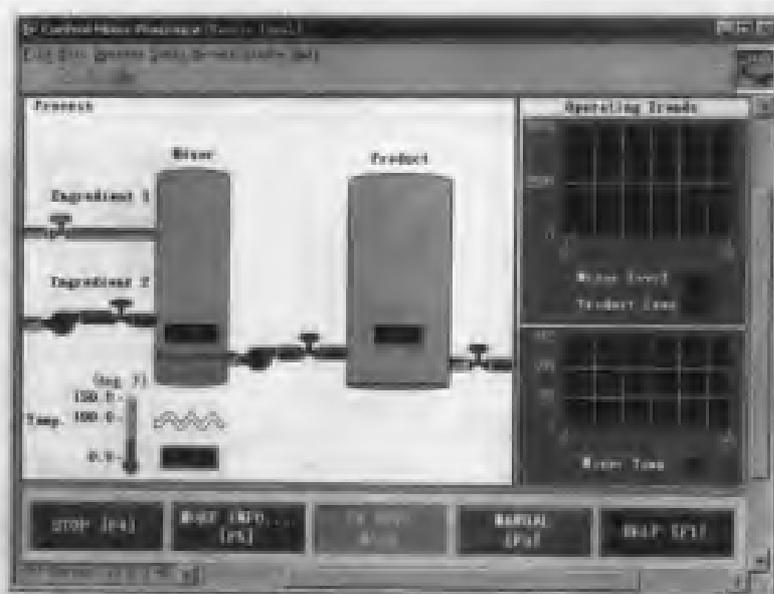


图 13.4.5 Remote Panel

如果操作失败，Connect to Remote Panel 对话框中的 Connect Status 一栏中会出现相关的错误信息。

13.4.3 通过网页浏览器在网页中操作 Remote Panel

Remote Panels 最令人兴奋的一个特征就是通过网页浏览器也可以控制远端 VI 的前面板。要实现这个功能，首先必须在 Client 端计算机上安装 LabVIEW Run-Time Engine 或 LabVIEW 7 Express。在 NI 公司的官方网站 www.ni.com 中可以下载到 LabVIEW Run-Time Engine。若计算机中已经安装了 LabVIEW，就不需要安装 LabVIEW Run-Time Engine 了。

接下来的工作就是在网页浏览器的地址栏中输入想要操作的 VI 的网址。

在局域网内，其地址格式是：

`http://PcName:Port/ViName.htm`

在 Internet 互联网上，其地址的格式为：

`http://IpAddr:Port/ViName.htm`

当 Remote Panel 出现在浏览器上时，可以单击鼠标的右键，这时会出现一个下拉菜单，在菜单中可以获得 Remote Panel 的控制权。如图 13.4.6 所示。

注意，LabVIEW 6.1 或 LabVIEW 7 Express 允许可同时连接到 Web Server 上的 Client 端的默认数目为一个，用户可以向 NI 公司购买授权，这样就可以使更多的 Client 端同时连接到 Web Server 端。

当有多个客户机同时使用 Remote Panels 时，需要在服务器上对 Remote Panel 进行管理，LabVIEW 提供了一个名为 Remote Panel Connection Manager 的工具，可以用来管理 Remote Panel。在 LabVIEW 选单栏中选择 Tools→Remote Panel Connection Manager，可以打开该工具，如图 13.4.7 所示。

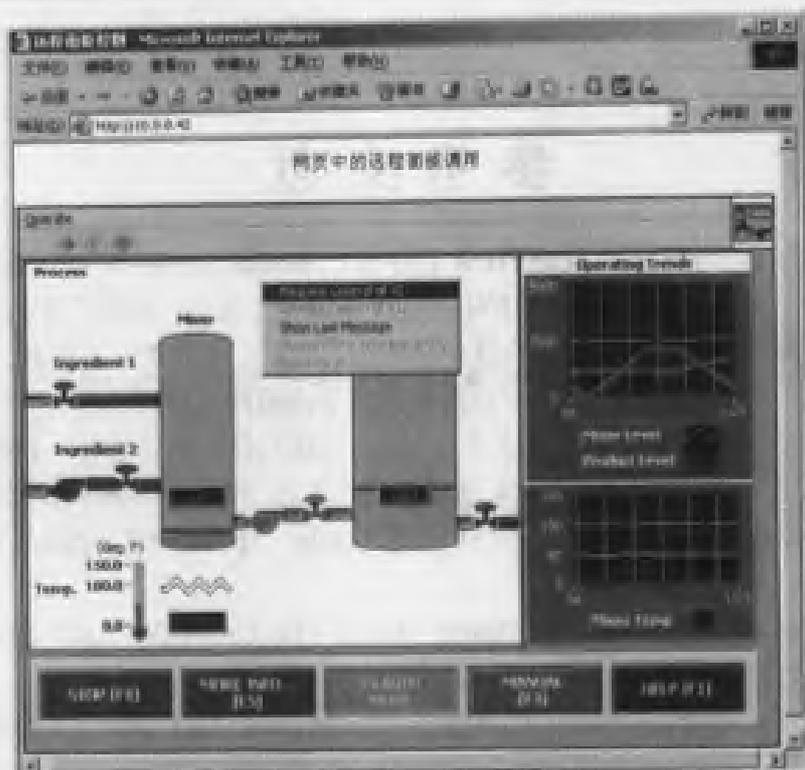


图 13.4.6 网页中的 Remote panel

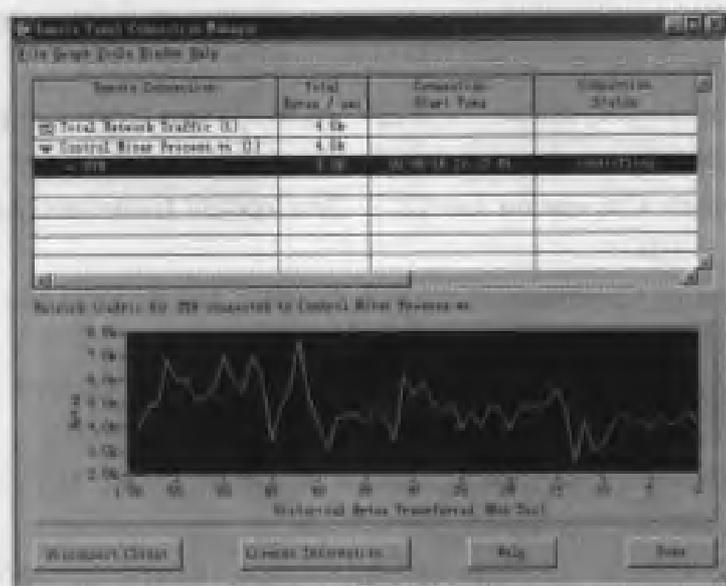


图 13.4.7 Remote Panel Connection Manager 工具

LabVIEW 的 Remote Panel 不仅可以观看，而且可以在 LabVIEW 的环境中或浏览器上加以控制。这个强大的功能让许多开发人员可以轻松地创建远程应用程序，使用户在周末的时候坐在家中的计算机前轻松地监控办公室、实验室甚至生产线上的各种情况。

参 考 文 献

- 1 杨乐平, 李海涛, 肖相生等. LabVIEW 程序设计与应用. 北京: 电子工业出版社, 2001
- 2 杨乐平, 李海涛, 赵勇等. LabVIEW 高级程序设计. 北京: 清华大学出版社, 2003
- 3 杨乐平, 李海涛, 肖凯等. 虚拟仪器技术概论. 北京: 电子工业出版社, 2003
- 4 杨乐平. 飞行器测试技术. 长沙: 国防科技大学出版社, 1997
- 5 杨乐平. 自动化测试与虚拟仪器技术. 长沙: 国防科技大学出版社, 1998
- 6 计算机虚拟仪器图形编程 LabVIEW 实验教材. 北京: 中科泛华测控技术有限公司
- 7 National Instruments Corporation. LabVIEW Help. April 2003 Edition, Part Number 370117C-01
- 8 National Instruments Corporation. Getting Started with LabVIEW. April 2003 Edition. Part Number 323427A-01
- 9 National Instruments Corporation. LabVIEW User Manual. April 2003 Edition Part. Number 320999E-01
- 10 National Instruments Corporation. LabVIEW User Manual. July 2000 Edition. Part Number 320999C-01
- 11 National Instruments Corporation. LabVIEW User Manual. January 1998 Edition. Part Number 320999B-01
- 12 National Instruments Corporation. LabVIEW Measurements Manual. April 2003 Edition. Part Number 322661B-01
- 13 National Instruments Corporation. LabVIEW Development Guidelines. April 2003 Edition. Part Number 321393D-01
- 14 National Instruments Corporation. LabVIEW Function and VI Reference Manual. January 1998 Edition. Part Number 321526B-01
- 15 National Instruments Corporation. BridgeVIEW™ and LabVIEW™ G Programming Reference Manual. January 1998 Edition. Part Number 321296B-01
- 16 National Instruments Corporation. LabVIEW Communications VI Reference Manual. November 1995 Edition. Part Number 320587C-01
- 17 TECHNIQUES Rick Bitter Taqi Mohiuddin Matt Nawrocki. LabVIEW Advanced Programming. CRC Press Boca Raton New York London Tokyo, 2001
- 18 Gary W. Johnson. McGraw-Hill. LabVIEW Power Programming. McGraw-Hill Companies, 1998
- 19 National Instruments Corporation. Integrating the Internet into Your Measurement System-DataSocket Technical Overview. www.ni.com, 1998
- 20 National Instruments Corporation. DAQ PCI-1200 User Manual. July 1998 Edition. Part Number 320942C-01
- 21 National Instruments Corporation. Signal Processing Toolset User Manual, 2001

- 22 Noël Adorno. Developing a LabVIEW™ Instrument Driver. National Instruments Application Note 006. www.ni.com
- 23 Noël Adorno. LabVIEW™ Instrument Driver Standards. National Instruments . Application Note 111. www.ni.com
- 24 Noël Adorno. Developing a LabVIEW™ Instrument Driver. Application Note 006. www.ni.com
- 25 Norma Dorst. Using LabVIEW to Create Multithreaded VIs for Maximum Performance and Reliability. Application Note 114. www.ni.com
- 26 John Pasquarette. Using IVI Drivers to Build Hardware-Independent Test Systems with LabVIEW™ and LabWindows™/CVI. National Instruments Application Note 121. www.ni.com
- 27 John Pasquarette. Improving Test Performance through Instrument Driver State Management. National Instruments Application Note 122. www.ni.com
- 28 Kamran Shah. Using IVI Drivers in LabVIEW. National Instruments Application Note 140. www.ni.com
- 29 National Instruments Corporation. Using LabVIEW with TCP/IP and UDP. National Instruments Application Note 160. www.ni.com
- 30 National Instruments Corporation. The Measurement and Automation Catalog 2002, 2002
- 31 汪浩, 孙兴, 刘森石. 高等数学. 长沙: 国防科技大学出版社, 1988
- 32 程佩清. 数字信号处理教程. 北京: 清华大学出版社, 1995
- 33 L.R.拉宾纳, B.戈尔德. 数字信号处理的原理和应用. 北京: 国防工业出版社, 1982
- 34 贾惠芹, 高天德, 郭恩全. IVI 技术研究. 国外电子测量技术, 2000 (3)
- 35 毛顿, 郭庆平. LabVIEW 中用 DataSocket 技术实现网络化应用. 现代电子技术, 2002 (3)