LabVIEW Real-Time模块用户手册

2025-03-26

目录

LabVIEW Real-Time模块用户手册

LabVIEW Real-Time模块用户手册详细介绍了产品功能和使用步骤。

还想了解其他信息?

对于产品用户手册中未包含的信息（如规格或API参考），请浏览***相关信息***。

相关信息：

- LabVIEW Real-Time模块入门指南
- 下载LabVIEW Real-Time模块
- 许可证设置和激活
- LabVIEW Real-time模块发行说明
- LabVIEW用户手册
- NI培训中心

什么是Real-Time模块?

Real-Time模块是LabVIEW的附加软件，可用于创建在嵌入式硬件设备上执行的可靠独立应用程序。

LabVIEW Real-Time (RT)模块扩展了LabVIEW的功能，可满足实时性能的要求。LabVIEW Real-Time模块将LabVIEW的图形化编程与实时操作系统结合起来，可用于创建确定性应用程序。在LabVIEW中开发VI，然后在RT终端上执行VI。RT终端可在具有实时性能的平台上运行VI。

关于创建LabVIEW Real-Time应用程序的入门信息，请浏览相关主题。

相关概念：

- Real-Time模块最佳实践
- 教程：创建Real-Time应用程序

安装LabVIEW Real-Time模块

您可以使用NI Package Manager或LabVIEW开发平台安装盘安装LabVIEW Real-Time模块。

应用软件支持

LabVIEW 2024 Q1Real-Time模块支持LabVIEW 2024 Q1。关于LabVIEW Real-Time模块早期版本中应用软件支持的信息，请参见这些版本的自述文件。

使用Package Manager安装LabVIEW Real-Time模块

关于使用Package Manager安装、移除、升级NI软件的信息，见***Package Manager***。

使用LabVIEW开发平台安装盘安装LabVIEW Real-Time模块

如果使用LabVIEW开发平台安装盘进行安装，请插入LabVIEW开发平台安装盘，并按照屏幕上的说明安装LabVIEW、模块、工具包及驱动程序。出现提示时，登录您的NI用户账号激活您的NI产品。如果使用批量许可证服务器管理您的许可证，您将通过电子邮件接收批量许可证。

注： 如启用Windows更新，安装过程可能会由于Windows更新和安装Microsoft Visual C++ 2015 Run-Time引擎的冲突而挂起。关于解决该问题的详细信息和步骤，参见***安装Visual C++ 2015 Run-Time引擎时NI安装程序挂起***支持文章。

注： 如已通过NI软件套件或NI产品包购买产品，请使用随附的安装盘安装产品。

注： 使用软件附带的序列号激活Advanced Signal Processing工具包。关于Real-Time模块激活的更多信息，见***许可证管理器***。

在RT终端上安装和使用日语和简体中文语言包

在开发计算机上安装Real-Time模块后，可按照下列步骤在RT终端上安装和使用日语和简体中文语言包：

1. 在NI Measurement & Automation Explorer (NI MAX)中使用LabVIEW Real-Time软件向导，在RT终端上安装Language Support for Japanese或Language Support for Simplified Chinese软件组件。关于LabVIEW Real-Time软件向导的信息，见Measurement & Automation Explorer帮助。
2. 打开“系统设置”选项卡，“语言环境”选择“日语”或“简体中文”。

注： RT终端上不能使用日语或中文字符作为目录名或文件名。

注： Real-Time模块（64位）仅有英文版。

配置RT终端

NI MAX用于配置RT终端及在终端上安装软件和驱动程序，可通过NI Package Manager或LabVIEW开发平台光盘安装NI MAX。

- 网络RT终端 – 关于配置网络RT终端的信息，请在MAX中选择“帮助»MAX帮助”，查看Measurement & Automation Explorer帮助中的远程系统章节。

相关信息：

- Package Manager
- 安装Visual C++ 2015 Run-Time时NI安装程序挂起
- 许可证管理器
- 从Web浏览器监控和配置远程设备

新增功能和改动

了解LabVIEW Real-Time模块各个版本的新增功能和行为变化等更新。

LabVIEW Real-Time模块2023 Q1改动

- LabVIEW 2023 Q1 Real-Time模块（64位）版本不支持Modbus IO Server。作为变通，可使用Modbus LabVIEW API的直接通信选项。
- LabVIEW 2023 Q1 Real-Time模块（32位和64位）版本不支持Real-Time Trace Viewer。可选择使用KernelShark替代，这是一种类似的开源工具。

LabVIEW Real-Time模块2022 Q3改动

- 支持RT系列设备。使用LabVIEW中的**管理软件**对话框安装和管理软件，而不是使用NI MAX。

LabVIEW Real-Time模块2020改动

- 支持RT系列设备。使用**RT板LED**节点控制所有RT系列设备上的LED。
- 支持CompactRIO控制器。

Real-Time模块最佳实践

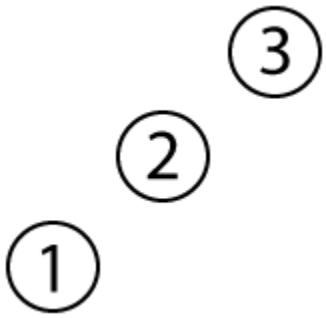
开发复杂的LabVIEW Real-Time应用程序需理解LabVIEW映射实时计算模型的方式。某些适用于开发通用操作系统的LabVIEW编程策略可能不适用于无用户界面、优先级调度的实时操作系统(RTOS)执行模型。

本文档将介绍使用LabVIEW Real-Time模块设计、开发和部署应用程序的最佳实践。按照下列最佳实践操作，部署系统的成功率更高。将***Real-Time模块最佳实践***用作图形化导航窗口，找到最佳实践文档。

相关概念：

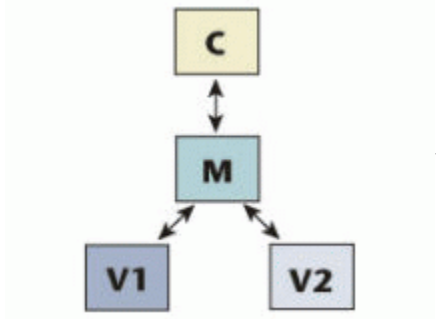
- 什么是Real-Time模块?
- Real-Time模块最佳实践

Real-Time模块最佳实践

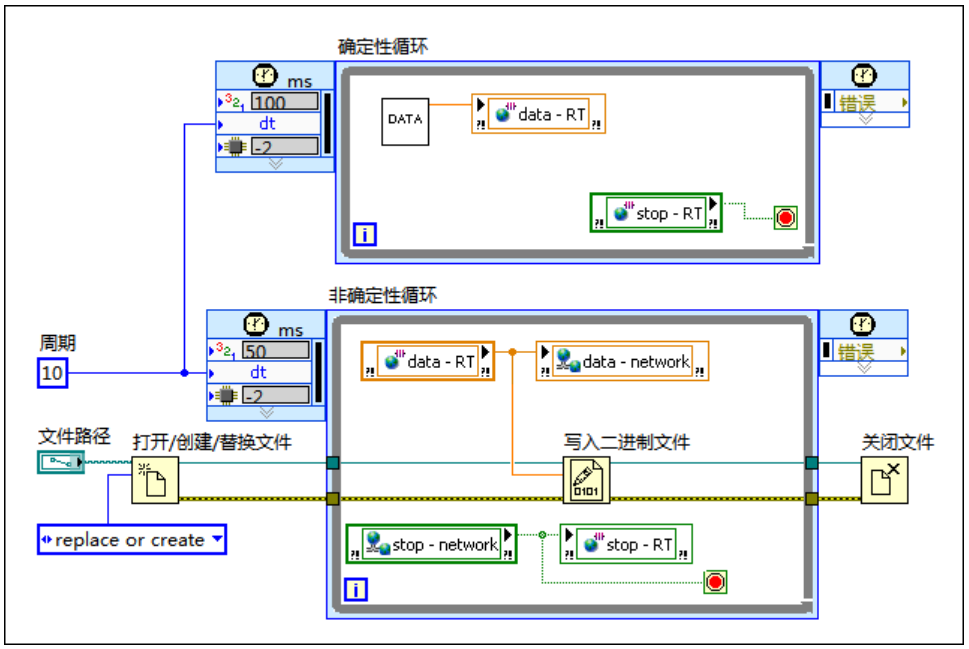


了解基于优先级的调度模式

避免抖动



使用设计模式

	用户界面互连	合
		
分离任务	高效使用共享变量	充分

相关概念：

- 基于优先级的调度模式
- 避免抖动
- 切换为部署设置
- 使用设计模式
- 用户界面互连
- 合理进行基准测试
- 分离任务
- 高效使用共享变量
- 减少CPU占用

基于优先级的调度模式

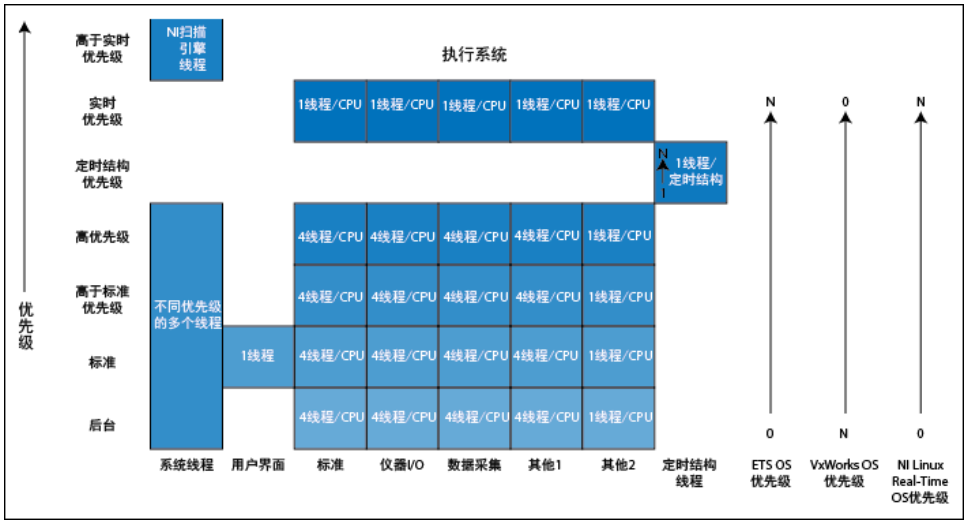
本章节讲解LabVIEW Real-Time模块创建和调度线程时如何处理优先级和执行系统。理解基于优先级的调度模型之后，可利用LabVIEW Real-Time模块创建确定性应用程序。

线程、优先级和执行系统

LabVIEW将计算机的各种复杂操作简化为直观的图形化编程界面，是一种结构化的数据流语言。下列几个关键概念定义有助于理解LabVIEW的并行任务优先级调度。

- **线程** – LabVIEW Real-Time模块的并行基本单位。线程中的任务按顺序执行，单个线程独立运行。
- **调度器** – 操作系统用于调度线程的算法。调度器用于决定特定时间各个CPU上运行的线程。
- **运行队列** – 准备调度的线程队列。
- **抢占式调度** – 调度器中断低优先级线程，高优先级线程准备运行的过程。
- **轮叫调度** – 调度器在同等优先级的几个线程之间定期切换，促进处理器时间平等共享。
- **执行系统** – 相关线程的池。使用“VI属性”对话框的“执行”页配置VI的执行系统。
- **VI优先级** – 在“VI属性”对话框的“执行”页上分配VI的优先级。
- **定时循环优先级** – 使用**优先级**输入或“配置定时循环”对话框为定时循环分配的优先级。
- **操作系统优先级** – 实时操作系统分配给线程的优先级。LabVIEW根据VI或定时循环的优先级分配给各个线程的优先级。
- **实时优先级** – LabVIEW VI可用的最高优先级。实时优先级线程的优先级高于调度器本身。因此，实时优先级线程总是运行至结束或阻隔。
- **优先级继承** – 线程暂时承担调用线程优先级的过程。优先级继承有助于避免优先级倒置。
- **抖动** – 程序的执行事件没有满足确定性预期。

下图是LabVIEW中优先级和执行系统的概览。



从上面的图示中，可以得出：

- 执行系统独立于优先级。只有在线程轮询，且不按照执行系统排列线程的优先级时，LabVIEW才使用执行系统。对于每个执行系统，LabVIEW在每个优先级上创建四个线程。该规则适用于多核系统，例如，在双核系统上，LabVIEW在每个执行系统的优先级上可创建8个线程。因为每个执行系统的线程数量是有限的，所以需平衡执行系统的分配。如在一个执行系统中分配太多VI，VI就不得不共享线程，从而影响了VI并行机制。
- 定时结构的优先级位于高优先级和实时优先级VI之间。
- 将并行代码放在一个定时结构中，LabVIEW将依顺序执行代码，因为每个定时结构只包含一个且仅有一个线程。
- LabVIEW Real-Time模块系统包括许多与组成LabVIEW VI的线程并行运行的系统线程。这些系统线程按不同的优先级执行。例如，若存在NI扫描引擎线程，默认运行优先级高于实时线程。

优先级继承

LabVIEW使用优先级继承来避免优先级倒置。如子VI的优先级低于调用方VI的优先级，两个VI都在一个执行系统中运行，子VI将继承调用方VI的优先级。如子VI的优先级高于调用方VI的优先级，子VI保持原有优先级。

注： 如子VI和调用方VI在不同的执行系统中运行，则不会发生优先级继

承。

实时操作系统(RTOS)也有优先级继承。如低优先级线程占用了高优先级线程所需的资源，实时操作系统会暂时提高占有共享资源所需的优先级。这样，高优先级线程就可占有所需的资源。

实时优先级

有两种类型的实时优先级线程：

- **设置为实时优先级的VI** – 设置为实时优先级的VI，运行优先级高于调度器本身。因此，实时优先级的VI一直运行到结束或中断，例如，定时函数或外部事件发生。
- **定时结构** – 定时结构可被另一个较高优先级的定时结构或设置为实时优先级的VI中断。

注： 建议在每个CPU上创建不超过一个实时任务。

定时结构优先级

LabVIEW使用两种优先级机制，VI优先级和定时结构优先级。这两种优先级机制既相互独立，又相互关联。定时结构优先级是数值，数值越大，表示相对于终端上的其他定时结构优先级越高。但是，所有定时结构优先级都在高和实时VI优先级上。

注： (Real-Time Linux)请勿在VI中使用超过32个定时结构优先级。否则，LabVIEW将返回错误。

使用优先级机制时避免错误

NI建议仅在应用程序中使用一个优先级机制，使用范例如下：

- 如应用程序使用定时结构，将所有VI设为普通优先级。

- 如应用程序使用实时优先级的VI，请勿将定时结构置于实时优先级的VI中。

在应用程序中使用一种优先级策略，应用程序更易于理解，并且不容易出错。

调度

调度就是判定指定时间运行哪个任务的进程。调度是所有操作系统的核心任务，尤其是实时操作系统(RTOS)。除VI引发的线程之外，实时终端运行多个OS并以不同的优先级驱动线程。为确保任务根据定时要求执行，需了解LabVIEW Real-Time线程调度的规则。

基本的RT调用规则

当一个新的线程进入运行队列时，LabVIEW Real-Time调度器进行下列操作：

1. 在不同优先级的线程之间进行抢占式调度。
2. 在同等优先级的线程之间进行轮叫调度。
3. 实时优先线程通常保持运行直至结束。

LabVIEW Real-Time模块调度工作原理

在任何给定时间，系统中的线程为“运行”或“阻止”两种状态中的一种：

- **运行** – 线程位于运行队列中，即线程执行就绪或正在执行。
- **阻止** – 线程必须等到某些事件（例如，I/O操作完成或定时函数）发生才能执行。

线程开始执行后，除非发生下列条件，线程才停止运行：

- 线程可被While循环中的等待VI、定时循环的内置定时机制，或读取变量（超时方式）函数等阻止函数阻止。
- 线程被更高线程中断。
- 线程完成执行。

线程取消阻止后，重新进入运行队列。运行队列按优先级排序，当线程进入队列

时，线程排在低于其优先级的所有线程之前。如进入运行队列的线程比当前运行的线程优先级更高，高优先级线程要么在另一个CPU核上运行，要么中断当前运行的线程，叫做抢占式调度。

抢占式调度

抢占式调度是根据线程的相对优先级对线程的执行进行排序。如某线程的优先级比当前运行线程的优先级高，该线程进入运行队列时，调度器会中断当前线程，较高优先级的线程可立即执行。中断的线程返回运行队列。

轮询调度

轮叫调度是指在不同的线程之间切换，每个线程被分配的CPU时间大致相等。在轮叫调度中，调度器按固定时间间隔中断和切换线程，记录每个线程分配的CPU时间。

LabVIEW Real-Time模块在同等优先级的线程之间执行轮叫调度。但是，LabVIEW Real-Time模块不会在实时优先级的VI之间执行轮叫调度。

根据定时要求设置循环优先级

根据循环的重要性设置各个循环的优先级，该方法虽然直观，但是会导致抖动。不要根据任务在应用程序中的重要性，而要根据该任务满足特定时间要求的重要性来设置循环的优先级。例如，一个数据记录应用程序，包含数据采集循环和数据记录循环，用户可能会把数据记录循环设置为较高的优先级。但是，数据采集必须按照固定间隔进行，否则，容易丢失数据点。只要最终将各个数据点记录到磁盘，何时记录某个数据点并不重要。在该情况下，尽管应用程序的目的在于记录数据，仍可将较高的优先级分配给数据采集循环。

使用设计模式

设计模式是用户可用于LabVIEW Real-Time模块应用程序，来解决软件工程的常见问题的构件。在应用程序中使用设计模式，可充分利用软件工程中的已知累积经

验。设计模式的优点包括：

- 更快的开发时间
- 更好的稳定性
- 更好的代码复用
- 更简单的调试
- 更简便的维护

建议对实时硬件平台上的LabVIEW Real-Time模块应用程序使用下列设计模式。如已安装NI-DAQ或NI-RIO驱动程序，可使用“创建项目”对话框打开专门用于CompactDAQ、PXI DAQ和CompactRIO平台的范例项目，查看LabVIEW应用程序中上述设计模式的范例。

使用场景	设计模式	说明
在编程状态间转移	通过“创建项目”对话框打开简单状态机。	应用程序设计由一组有限个状态、状态间的转移及与状态关联的动作组成。
在并行循环间进行通信	通过“创建项目”对话框打开队列消息处理器。	以不同的速率产生和消费数据，在两个进程之间传输数据。
创建实时数据服务器	客户端服务器	客户端向服务提供端（服务器）请求动作或服务的应用程序设计。

关于特定于CompactRIO的设计模式，见ni.com上的基于NI LabVIEW的CompactRIO开发者指南。

注： 建议从生产者/消费者模型开始学习如何在实时应用程序任务内进行通信。但与确定性循环通信时，则使用启用RT FIFO的单进程共享变量，或者RT FIFO函数。

分离任务

任务分离是指将任务放在不同的并行循环之中，创建多速率应用程序。任务分离对于大多数实时应用程序来说非常重要，原因在于：

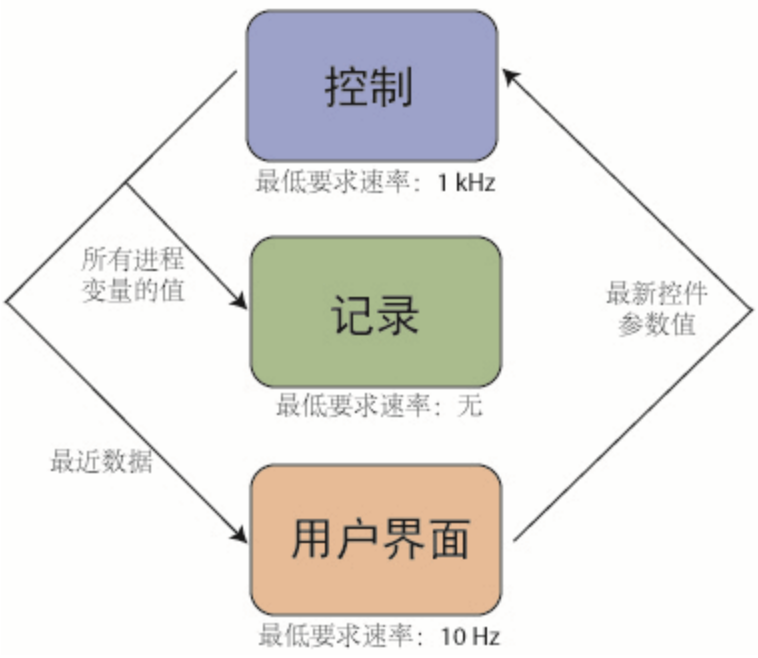
- **确定性** – 要尽量减小确定性循环中的抖动，需保证循环中不能包含潜在的非确定性代码。因为大多数大型应用程序均包含非确定性任务，识别及分离这些任务对于RT应用程序来说至关重要。
- **CPU效率** – RT终端上的CPU周期通常非常重要，所以CPU效率是设计RT应用程序的一个常见制约因素。在独立的循环内以不同的速率运行任务，仅在需要时执行每个任务能够最大化CPU的效率。

定义任务

开始编写代码之前，要定义应用程序必须进行的任务、各个任务所需的速率（如适用）和任务之间的数据传输关系，仔细考虑应用程序的顶层设计。例如，实时应用程序可能包含下列任务：

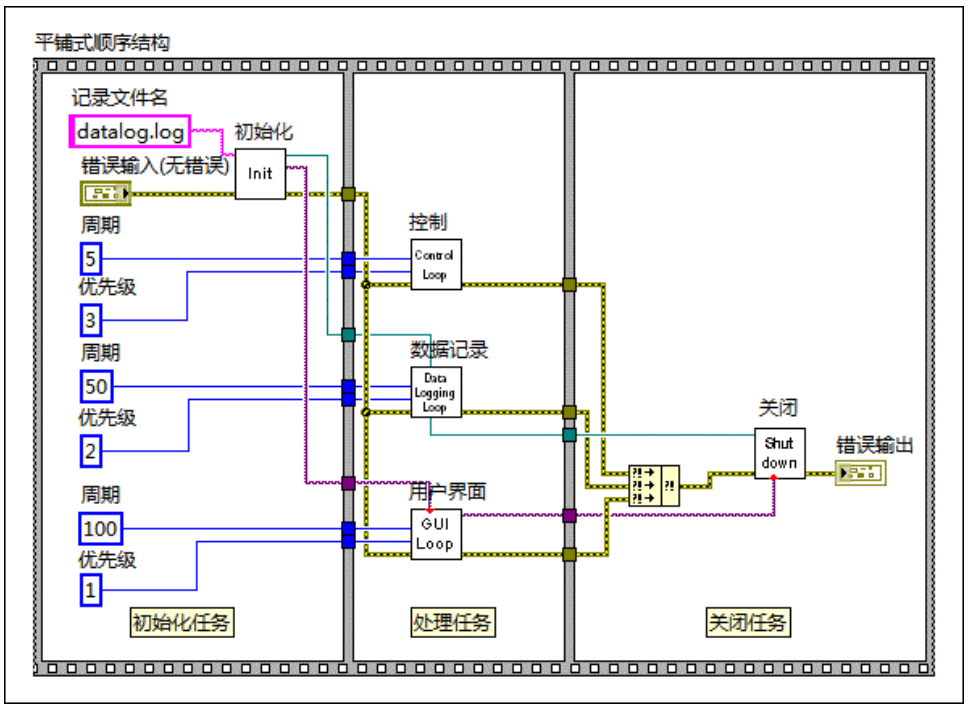
- **控制**－控制液罐中液体的温度和高度。该任务的最低速率是1 kHz。
- **记录**－记录采集到的温度和电平的历史数据。该任务的要求是记录控制任务采集到的每个进程变量值。该任务没有最低速率限制。
- **网络**－通过网络传输用户界面数据，接收来自用户的命令以及显示连续的数据图供用户监控。该任务的最低速率是10 Hz。

定义组成应用程序的顶层任务后，需定义这些任务之间的数据传输关系。例如，下列示意图显示了上面列表中三个范例任务之间的数据传输关系。



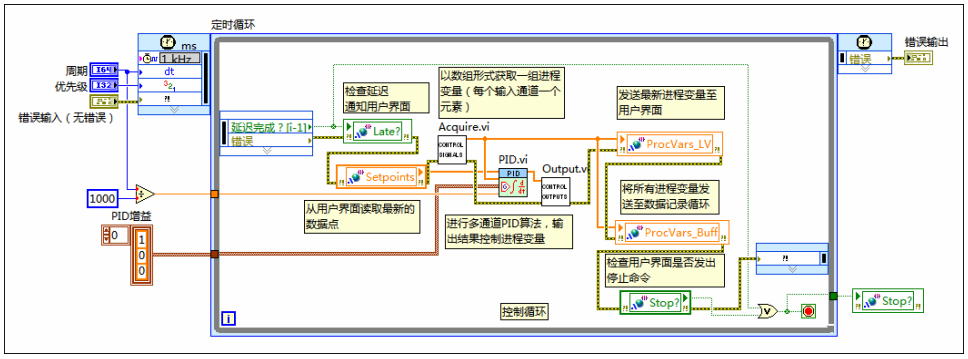
将设计转换为LabVIEW程序框图

定义应用程序的顶层设计之后，即可开始将设计转换为LabVIEW代码。例如，下列程序框图显示了包含上面章节中定义的正在运行的任务的顶层VI：

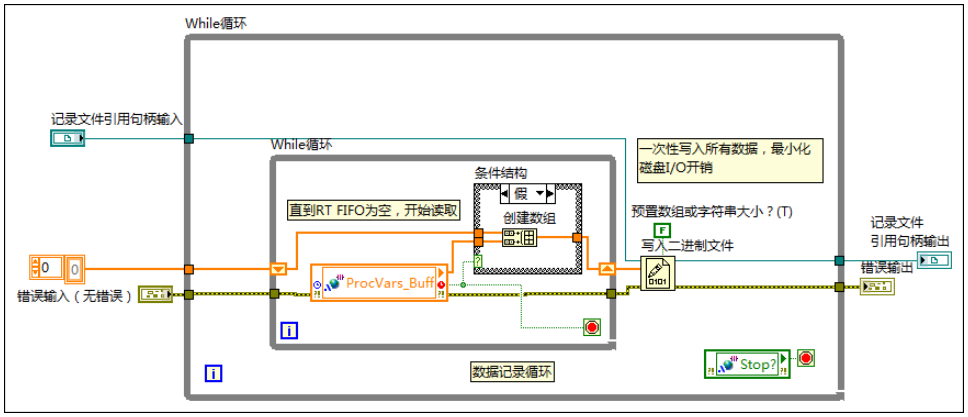


注： 除了应用程序的主体任务，建议另创建一个初始化任务和关闭任务，如上面程序框图所示。

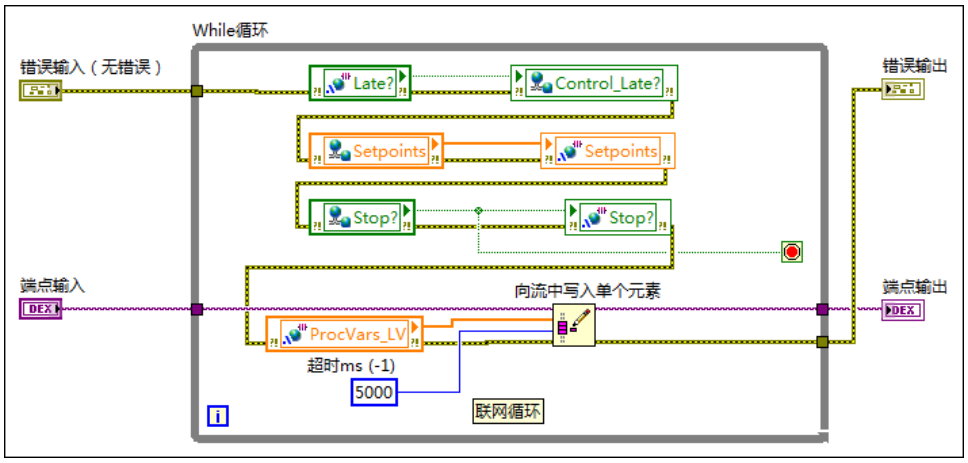
以下程序框图显示了子VI `Control.vi`中的控制循环：



以下程序框图显示了子VI `Log.vi`中的数据记录循环：



以下程序框图显示了子VI `User_Interface.vi`中的网络循环：

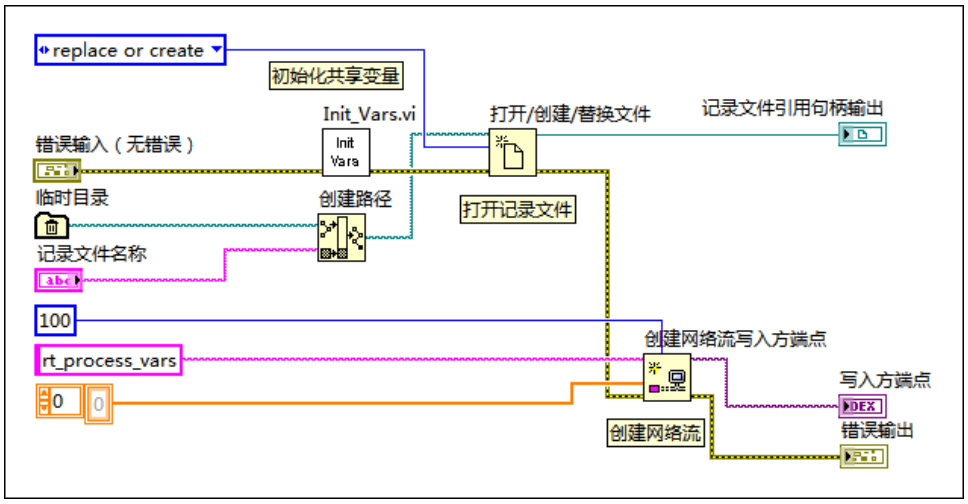


创建初始化程序

创建初始化程序，在任务开始执行之前处理一些准备任务。根据应用程序的性质，初始化程序包括下列操作：

- 初始化共享变量
- 打开文件引用
- 预分配数组

以下程序框图显示了子VI `Initialize.vi`的初始化程序：



创建关闭程序

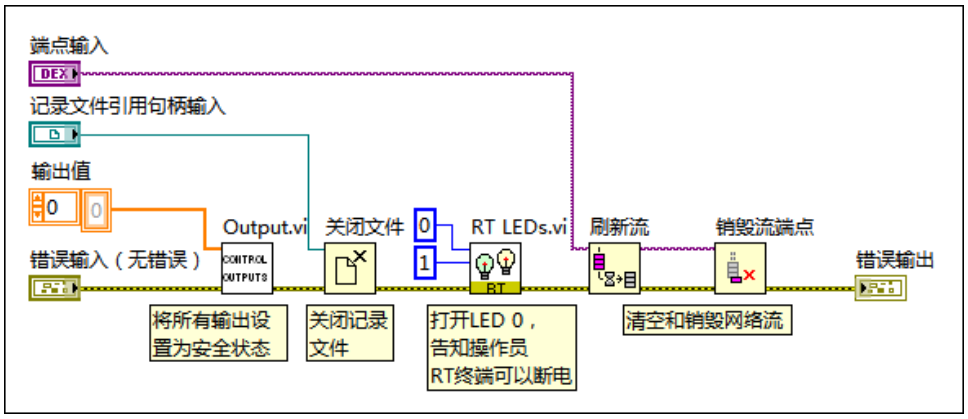
建议在关闭实时终端之前先将应用程序切换至关闭状态。创建一个关闭程序，在当

前应用程序任务停止执行后运行，以确保RT终端处于关闭状态。根据应用程序的性质，关闭程序包括下列操作：

- 将输出设置为已知值
- 关闭文件引用
- 通过网络发布的共享变量或RT LED VI告知操作员何时可以关闭RT终端

注： 即使实时终端使用的是Reliance文件系统，文件打开时关闭终端仍可能损坏数据。Reliance只保证文件系统本身的完整性，但是不保证关闭实时终端时仍处于打开状态的单个文件的完整性。

以下程序框图显示了子VI `Shutdown.vi`的关闭程序。



相关概念：

- 高效使用共享变量
- 分离任务

避免抖动

精确的定时是实时应用程序的关键性因素。抖动是指程序的执行事件没有满足确定性预期。本章节介绍如何在实时应用程序中减小抖动。

避免引起抖动的常见源

不要在实时循环中加入非确定性的内容，以减少程序执行时间的抖动。常见的抖动源包括：

- **内存分配** – 先预分配数组，然后在确定性循环中使用数组，避免确定性循环中可变大小的数据结构。使用替换数组子集函数对确定性循环中的预分配数组进行操作。使用Real-Time Trace Viewer识别**等待内存**系统事件标识，该标识表示内存管理操作。
- **打开和关闭引用** – 在初始化阶段（进入确定性循环之前）打开引用，在关闭阶段（确定性循环终止之后）关闭引用。打开和关闭引用会影响循环的确定性。
- **通信** – 与确定性循环通信时，使用RT FIFO函数或启用了Real-Time FIFO的共享变量。RT FIFO将数据存放在缓冲区中，确定性循环可访问数据，不受低优先级循环的影响。
- **共享资源** – 避免访问共享资源，如硬件资源、I/O通道或确定性循环和其他循环中的非重入VI。同时访问某共享资源是引起抖动的常见原因。如确定性循环使用其他循环同时也在使用的硬件资源或非重入VI，确定性循环可能需要等到低优先级的循环释放后才能使用这些资源。
- **文件I/O** – 在确定性循环中避免文件I/O操作。访问硬盘会引起确定性循环的执行时间抖动。
- **异步I/O** – 在定时循环或实时VI中避免异步I/O操作。LabVIEW通过轮询操作来确定操作是否已完成。轮询会引起抖动，还会阻止低优先级任务运行。建议使用同步操作。同步任务会一直运行到结束，保证读/写操作在完成之前不被轮询。使用GPIB读取、GPIB写入、VISA读取、VISA写入、VISA等待事件时，同步和异步操作的区别较大。如要在定时循环或实时VI中使用这些函数，右键单击函数，从快捷菜单中选择**同步I/O模式»同步**，将函数切换为同步I/O模式。其他函数可能也需该配置。
- **网络互连** – 不要在确定性循环中进行网络互连。网络互连会引入确定性循环的无约束抖动。
- **无约束的算法** – 某些算法的性质是非确定性的。不要在确定性循环中使用无约束的算法。如不确定VI或函数的抖动特性，建议先进行基准测试。

创建定时策略

对于不间断的重复的循环，需创建一致的定时策略以保证精确定时。LabVIEW Real-Time模块的定时策略包括循环结构和定时源。LabVIEW Real-Time模块包含了大量可用于控制循环定时的定时方法和定时源。

选择循环

除标准的While循环之外，LabVIEW还具有内置定时功能的定时循环。使用定时循环的输入和输出端，确认应用程序的定时特性。下表总结了两种循环结构的特性：

循环结构	特性	优点	缺点
While循环	基本的循环结构，多线程	CPU开销较小	无内置定时功能
定时循环	单线程循环结构，用于多速率应用程序	内置时间控制、定时数据、CPU选择、优先级控制；易于确定性调度的单线程	CPU开销大于While循环

选择定时源

无论使用的是定时循环或While循环，选择一个合适的定时源驱动循环的各个循环定时。

While循环定时源

在While循环中放置定时VI或函数，可控制While的定时设置。下表总结了LabVIEW Real-Time模块可用的内置定时VI和函数：

使用场景	定时方法	内置定时源
通过硬件定时I/O同步循环	硬件定时I/O方法，例如，DAQmx读取VI或FPGA I/O方法节点上的等待方法	外部（取决于硬件平台）
x毫秒后延迟循环执行	等待	CPU衍生的毫秒或

使用场景	定时方法	内置定时源
		微秒计时器
控制循环执行速率和同步循环	等待下一个整数倍	CPU衍生的毫秒或微秒计时器
NI扫描引擎每次扫描后，触发循环运行	同步至扫描引擎	NI扫描引擎

定时循环的定时源

关于选择定时循环定时源的信息，见选择定时结构的定时源主题。

创建时间预算

创建应用程序的时间预算，有助于避免非预期的时间分配。时间预算涉及确定执行应用程序中各个循环的时间，以及设置相应的循环速率。创建和遵循时间预算有助于应用程序按照预期运行。

测量循环执行时间

将应用程序分为若干个循环后，可使用RT基准测试方法确定应用程序中各个循环所需的时间。基准测试运行上千个循环计数，然后将其中最长的执行时间用作循环的预期持续时间。

注： 分析执行时间时，不要使用**高亮显示执行过程**工具。高亮显示执行过程会显著降低执行速度。

创建时间预算表

测量循环的执行时间后，记录每个循环的持续时间和周期，创建应用程序的时间预算表，如下列范例所示：

任务/循环	持续时间(µS)	周期(µS)
控制	400	1000
监视	3000	10000
记录	16000	30000

降低CPU使用率

CPU的使用率在100%以下时，可降低抖动并确保应用程序无需抢占CPU时间。创建时间预算表后，可通过下列公式确定理论CPU使用率。

CPU使用率(%) = 100 * Sum[Loop 1, Loop 2,.., Loop n]（持续时间/周期）

将表中的数据代入公式，得到该例中CPU的使用率是：100 * (400/1,000 + 3,000/10,000 + 16,000/30,000) = 123%。因为该范例中的监控循环的优先级高于记录循环，实时操作系统在控制循环完成后立即调用监控任务。实时操作系统只有在监控循环完成一次循环后才开始记录，留出的时间不足以满足要求的周期。

在该情况下，可延长一个循环或多个循环的周期，保证循环按指定速率运行。例如，将监控循环的周期延长为25,000 µS，将记录循环的周期延长为80,000 µS，理论CPU使用率将变为100 * (400/1,000 + 3,000/25,000 + 16,000/80,000) = 72%。

注： 如无法延长循环的周期，可使用其他方法来降低RT终端的CPU使用率。

相关概念：

- 合理进行基准测试
- 减少CPU占用

用户界面互连

有些RT终端包含一个嵌入式用户界面，可以用来查看和控制RT终端VI的前面板。然

而，LabVIEW Real-Time模块支持的其他RT终端则没有用户界面设计，只有基本的输出显示。对于不支持嵌入式用户界面的RT终端，为LabVIEW Real-Time模块应用程序创建图形化用户界面(GUI)需要使用分布式计算方法。如实时应用程序需要用户界面且无法使用嵌入式UI，可在通用操作系统上显示图形化用户界面，并使用远程通信协议在RT终端和GUI设备间传输数据。

避免前面板对象、方法和事件

开发RT VI用于不支持嵌入式UI的终端时，可通过前面板输入控件和显示控件调试。但是在最终生成的独立RT应用程序中不能使用RT VI的前面板作为用户界面。将VI部署为独立的实时应用程序时，LabVIEW将移除VI的前面板，包括全部对象、属性、方法和关联事件。即使事件驱动的编程对于多数LabVIEW VI来说是强有力的用户界面设计模式，但是不能在RT终端上运行的VI中使用典型LabVIEW事件驱动UI设计模式。但可使用事件驱动设计模式创建运行在主机上的用户界面VI，主控计算机使用网络协议与RT终端VI通信。

使用正确的网络设备

用户通常倾向于使用现成的或较廉价的网络设备。但是，各种商用网络设备大相径庭，需根据应用程序选择相应的网络设备。对于一些应用程序，一般的网络设备即够用。对于某些依赖于网络吞吐率、延时或可靠性的应用，需要使用较高级的网络设备。

使用正确的网络协议

下表总结了在RT终端和GUI设备间传输数据的通信协议的优劣比较和适用范围：

使用场景	协议	优点	缺点
监控最新值	网络发布共享变量	易于编程，一次发布至多台计算机，启用实时FIFO后可在确定性循环中使用，集成工业协议	CPU开销大，专有，需共享变量引擎
在RT终端和主机之间以	网络	高吞吐量，无损耗，自动连接管	读取方和写入方一一对

使用场景	协议	优点	缺点
数据流方式传输数据，发送命令至RT终端	流	理，LabVIEW数据类型支持	应，单向数据传输，非确定性，专有
STM等自定义协议的起点，见ni.com	TCP或UDP	高效的CPU和内存使用，便于互操作的协议，灵活，标准	较难编程，仅限于字符串数组，无内置方法识别数据
RT VI的动态控件	VI服务器	VI和其他LabVIEW构造体的本地和远程控制	低速，专有
基于Web的用户界面，与标准Web技术兼容	Web服务	Web标准，客户端连接无需LabVIEW	开销大，编程难，需要用其他编程语言创建一个轻量的客户端

下表显示了最常见的RT GUI使用场景的推荐网络协议：

使用场景	范例	推荐协议
命令（一对一）	将用户界面值改动从主机发送到RT终端	网络流函数
命令（一对多）	将用户界面值改动从主机发送到多个RT终端	网络发布共享变量
命令（多对一）	将用户界面值改动从多个Web客户端发送到一个RT终端	Web服务VI或网络发布的共享变量
监视器（一对一）	使用主机监视RT终端发布的最新值	网络发布共享变量或STM
监视器（一对多）	使用多个网络客户端监视RT终端发布的最新值	Web服务VI或网络发布的共享变量
监视器（多对一）	使用主机监视多个RT终端发布的最新值	网络发布共享变量
数据流（一对一）	将连续的数据流从RT终端发送到记录数据的主机	网络流函数
数据流（一对多）	Web广播服务器	TCP或UDP

使用场景	范例	推荐协议
数据流（多对一）	Web广播客户端	TCP或UDP

注： 关于使用网络发布的共享变量发送和接收命令的范例，见ni.com上的基于NI LabVIEW的CompactRIO开发指南。

创建专用界面

对于需要单个专用用户界面接线端的应用程序，可使用网络流函数连续流式传输数据和发送命令。使用网络发布共享变量或ni.com上的Simple Messaging Reference Library (STM)，监测每个变量的最新值。使用STM库发送命令到RT终端

创建SCADA界面

监视控制和数据采集用户界面通常包括与多个嵌入式控制器交互的一个或多个用户界面。对于SCADA应用程序，使用网络发布共享变量发送命令和监视最近值。要尽可能提高大型应用程序的可扩展性，可以编程方式查找、读取和写入共享变量。

关于使用共享变量发布用户界面数据的范例，见labview\examples\Real-Time Module\RT Communication\RT FIFO Variables目录下的RT FIFO Variables - networked.lvproj。

注： LabVIEW DSC模块包含创建高级SCADA应用程序的各项功能。

使用正确的载荷

载荷大小就是通过网络一次性发送数据的大小。载荷越大，程序的吞吐量和CPU效率就越高。等待数据积累到某一特定的值，然后发送数据，网络发送数据的时间可能会稍有增加。通过数据流传输数据时，总是希望速度越快越好。相比之下，在发送命令时，延迟越少越好。

连续数据流

下列建议有助于提高网络传输的速度。

- 使用高效的网络协议（例如，UDP、TCP）或网络流。
- 通过网络发送数据前，先累积数据。以1 KB（1024字节）或若干字节为单位发送数据，速度快且CPU系统开销少，对于大多数应用程序能保证较小的延迟。但是，使用数据流传输数据时，如要减小延迟，建议使用较小的载荷。
- 考虑切换为以太网数据包检测的轮询模式。RT终端的**数据包检测**设置位于NI Measurement & Automation Explorer**更多设置**下拉菜单中的**网络设置**选项卡下。

> **注：** 轮询模式的抖动小于中断模式。但是，轮询模式的CPU占用量大，会影响应用程序的整体速度。并非所有实时终端都支持轮询模式。

发送命令

下列建议有助于最小化延时：

- 使用高效的网络协议（例如，UDP、TCP）或网络流。
- 尽快地发送各条命令。例如，可使用刷新流函数，编写命令后通过网络立即发送命令。

高效使用共享变量

本章节介绍在LabVIEW Real-Time应用程序有效使用共享变量的规范。

适量使用共享变量

不要使用太多网络发布的共享变量，以尽量减少CPU占用。对于大通道数的应用程序，可将通道组合为一个数组，使用一个数组型网络发布共享变量传输数组的值。

通过启用了RT FIFO的共享变量进行确定性通信

LabVIEW提供多种在不同循环中运行的任务之间传输数据的选项。但是，大多数数据传输的方法都会影响到确定性。因此，LabVIEW Real-Time模块提供启用了RT FIFO的共享变量，用于在确定性循环和非确定性循环之间进行通信。

关于启用了RT FIFO的共享变量的使用范例，见`labview\examples\Real-Time Module\RT Communication\RT FIFO Variables`目录下的`RT FIFO Variables - local.lvproj`。

避免不必要的缓冲

缓冲会消耗CPU和内存资源。需要读取写入网络发布共享变量的每一个值时，可在“共享变量属性”对话框的“网络”页上，勾选**使用缓冲**复选框。缓冲可避免网络暂时拥堵造成的数据丢失，但是不能确保数据传输的过程中没有损耗。缓存区被填满时，LabVIEW会覆盖旧数据并生成警告。

在高通道数应用中使用程序访问

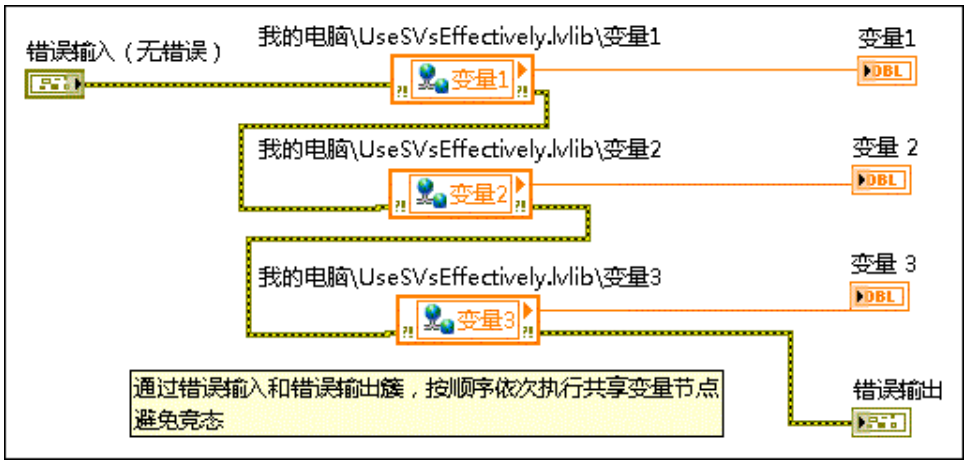
使用共享变量选板上的程序访问函数，在高通道数应用程序中创建整洁、可扩展的程序框图。

顺序执行共享变量

按数许执行共享变量，可提高程序运行速度，避免出现竞态。使用共享变量节点的**错误输入**和**错误输出**接线端，或共享变量程序访问函数来实现顺序化执行。顺序化执行后，可最小化调度的开销并提高速度。

例如，下列程序框图并行读取三个共享变量：

以下程序框图使用共享变量节点的**错误输入**和**错误输出**接线端来实现顺序化执行：



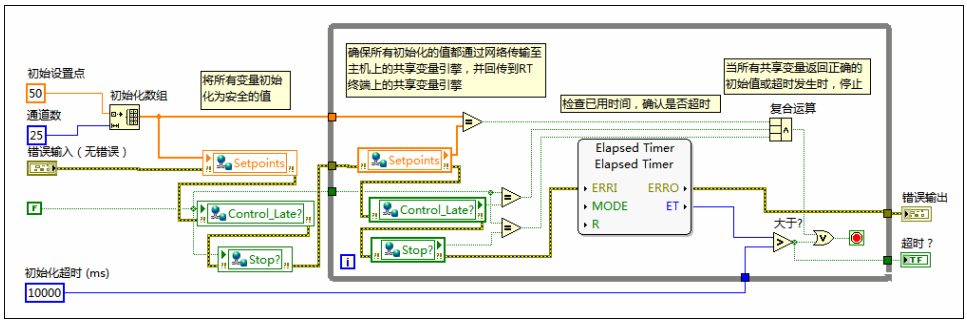
避免读取失效的共享变量数据

使用共享变量节点的**毫秒超时**输入端或读取变量（超时方式）函数，避免反复读取循环中的同一个值。如需为共享变量节点添加**毫秒超时**输入，可右键单击共享变量节点，从快捷菜单选择**显示超时**。仅可对读取数据的共享变量启用超时。无法对访问本地I/O变量的节点节点超时周期。

为了避免读取应用程序上次运行时的过期数据，应先取消部署，然后重新部署，清除共享变量的值。使用应用程序属性对话框的共享变量部署页，配置独立的实时应用程序在运行之前先部署共享变量，并且在应用程序退出之前，先取消部署变量。

初始化共享变量

运行应用程序的任务循环之前，初始化所有共享变量，有助于保证应用程序不受过期共享变量数据的影响。要初始化其他计算机上不带缓冲的网络发布共享变量，先写入默认值至共享变量，然后等待主机上的共享变量引擎通过网络将初始化值传递给RT终端。下列程序框图显示了不带缓冲的网络发布共享变量的初始化范例：



注： 要初始化带缓冲的网络发布共享变量，取消部署然后重新部署共享变量库即可。

选择合适的设备作为共享变量的主机

在多数情况下，RT终端可作为共享变量的主机。有些情况下，台式机更为合适。完成应用程序之前，确保共享变量位于最合适的设备上。下表总结了RT终端作为共享变量主机的优势和劣势：

优点	缺点
多台计算机可访问一台RT终端上的共享变量	增加RT终端的CPU开销
RT终端具有较高的稳定性，可保证较长的正常运行时间	增加RT终端的内存开销

相关概念：

- 减少CPU占用

切换为部署设置

默认情况下，RT终端和VI的设置最适于开发和调试的要求。但是，这些设置会影响RT应用程序的速度和确定性。本章节讲述了优化部署的应用程序的性能的步骤。

选择最优的部署设置

对RT应用程序进行基准测试或部署之前，考虑下列设置优化程序的性能和确定性：

设置名称	位置	最优性能设置	详细信息
超线程	RT终端BIOS	禁用	超线程会引起严重的抖动，NI建议不要使用超线程。NI在RT终端上将超线程设置为默认情况下禁用。如使用台式机作为RT终端，默认情况下超线程可能为启用状态。使用RT终端BIOS查看超线程是否被禁用。如未禁用，在BIOS中禁用超线程。
包检测	NI Measurement & Automation Explorer中**更多设置**下拉菜单下的**网络设置**选项卡	轮询	轮询模式的抖动小于中断模式。轮询模式的CPU占用较多。并非所有实时终端都支持轮询模式。
启用CPU负载监控	其他属性页	禁用	CPU负载检测会增加CPU和内存开销。
允许	执行页	禁用	调试支持会增加CPU和内存开销。

设置名称	位置	最优性能设置	详细信息
调试			

其他部署操作

除上述设置之外，可进行下列操作优化部署的RT应用程序的性能：

- 只安装必须的软件。
- 控制VI的大小，优化缓冲区效果。
- 以独立应用程序生成和运行RT项目

合理进行基准测试

部署实时应用程序之前，需要知道应用程序能在较长的执行时间内始终满足定时要求。本章节介绍实时应用程序确定性基准测试中的最佳实践。

> **注：** 分析执行时间时，请不要高亮显示执行过程。高亮显示执行过程会显著降低执行速度。

使用部署设置进行基准测试

对RT应用程序进行基准测试之前，切换为部署设置可确保基准测试的结果准确反映最终应用程序的性能。

使用实时基准测试的范例

部署应用程序之前，建议对整个实时循环的内容进行基准测试。对要包含在时间敏

感循环中的代码进行基准测试时，可使用RT基准测试范例项目作为起始点。RT基准测试范例可提供上千个循环中精确的平均耗时和最严重的抖动时间。请参考`labview\examples\Real-Time Module\RT Benchmarking`目录下的`Benchmark Project.lvproj`。

实时基准测试范例可用作基准测试的工具，确定代码是否具有确定性。也可在非实时代码中使用该范例进行基准测试。开始基准测试之前，要将基准测试VI中的优先级改为代码在应用程序中运行的优先级。使用“VI属性”对话框中“执行”页的**优先级**下拉菜单可设置VI的优先级。

注： 使用RT获取时间标识VI或时间计数器VI，对不在循环内的代码进行基准测试。使用定时结构内置的定时接线端，对定时结构进行基准测试。

使用Real-Time Trace Viewer

Real-Time Trace Viewer VI用于获取RT终端上VI的定时和执行数据，以及应用程序的线程事件。Real-Time Trace Viewer可显示主机上的定时和事件数据，或跟踪会话。在LabVIEW中，选择**工具»Real-Time模块»Trace Viewer**，显示Real-Time Trace Viewer。

关于跨不同操作系统的线程名称的映射，见知识库。

使用NI分布式系统管理器

NI分布式系统管理器用于监测RT终端上的CPU和内存使用、警告、VI状态和共享变量。选择**工具»分布式系统管理器**，可打开NI分布式系统管理器。

相关概念：

- 切换为部署设置

减少CPU占用

本文档介绍如何减少RT终端上CPU的占用率。CPU的使用率在100%以下时，可降低抖动并确保应用程序无需抢占CPU时间。

请参考下列改进LabVIEW性能的建议：

- VI执行速度
- 大量数据的内存管理

合理的循环运行速度

在应用程序中以最快速度运行循环不可取，因为这样会导致不可预期的时间消耗、增加时间抖动，甚至引起系统死锁。例如，如用户界面数据循环的发布速度如超过了操作员可以处理的速度，循环就造成了CPU和RT终端资源的无意义消耗。在多数情况下，通过网络发布用户界面数据的频率为2 Hz至15 Hz。

提示 创建时间预算帮助确定应用程序中各个循环合适的运行速率。

避免过量使用并行算法

LabVIEW程序框图是可视化的流程图，易于并行化VI代码，有助于提高程序在多核系统上的性能。但是，大量使用并行算法会使LabVIEW创建和管理更多线程。额外的线程带来的系统开销会影响程序的性能。一般而言，并行算法仅能提高多核系统上程序的性能。

1. 包含多个CPU内核的RT终端。
2. 按顺序执行代码所需的总时间超出了执行最长并行分支加线程管理和切换系统开销的和值时间。

要确定某个VI是否能受益于并行算法，可能需要对VI的串行模式和并行模式进行基准测试。关于在多核RT终端上使用并行算法的信息，见在多CPU系统上优化RT应用程序。

理解网络发布共享变量的性能基准

网络发布的共享变量呈现了CPU使用、数据传输频率和应用程序中变量数量之间的线性关系。关于网络发布共享变量CPU性能的信息，见ni.com上的性能基准。网络发布的共享变量应主要用于低频率、最近值数据传输的应用程序。如要从一台计算设备发送连续数据流至另一台计算设备，可使用网络流函数。一般来说，如使用低数据传输频率，使用大量网络发布的共享变量不会过度使用CPU资源。如需通过减少网络发布的共享变量数量优化CPU使用，可考虑将各个数据项组合为数组或簇，通过一个网络发布的共享变量来传输该数组或簇。

必要时取消加载任务

在台式机或FPGA终端上取消加载特定任务，可减少RT终端上CPU的使用率。例如，在主机上，而不是RT终端上使用网络发布共享变量。

注： 如要从多台计算机访问一个RT共享变量，将共享变量置于RT终端上。

使用下列规范，为任务挑选最合适的设备：

任务	适合的设备
数据采集	RT或FPGA
控制循环	RT或FPGA
用于记录或监控目的的数据分析（离线分析）	台式计算机
数据记录	RT设备或台式计算机
主控网络发布共享变量	台式计算机或RT设备

注： 可以使用Real-Time Trace Viewer确定应用程序中占用了最多CPU时间的VI和线程。

相关概念：

- 避免抖动
- 合理进行基准测试

遵循安全性最佳实践

可将实时系统部署至多种不同的危险环境。为帮助减少环境中产生的安全威胁和影响，National Instruments公司为实时终端和应用提供了多个安全选项。通过了解系统适用的安全选项，可评估在实时部署中包含这些选项的重要性。访问ni.com/info并输入RTSecurity，了解安全问题。

Real-Time模块概念

该部分介绍LabVIEW Real-Time模块和实时编程的一些技巧，有助于您构建确定性的应用程序：

LabVIEW Real-Time模块简介

大多数LabVIEW应用程序都在通用操作系统(OS)上运行，如Windows、Linux或Mac OS X。有些应用程序需要实时性能，而通用操作系统无法确保提供这种性能。LabVIEW Real-Time模块扩展了LabVIEW的功能，可满足实时应用的要求。

LabVIEW Real-Time模块将LabVIEW的图形化编程与实时操作系统结合起来，可用于创建确定性应用程序。在LabVIEW中开发VI，然后在RT终端上执行VI。RT终端可在具有实时性能的平台上运行VI。

相关概念：

- 使用Real-Time模块创建确定性应用程序

LabVIEW Real-Time模块平台

LabVIEW Real-Time模块在NI Linux Real-Time操作系统上执行VI。

NI Linux Real-Time操作系统在NI RT系列硬件上运行，以实现确定性动作和扩展可靠性。

LabVIEW Real-Time模块不支持某些LabVIEW功能。关于NI Linux Real-Time操作系统不支持的LabVIEW功能的详细信息，请参见***NI Linux Real-Time操作系统***。

相关概念：

- NI Linux Real-Time操作系统

实时终端

Real-Time终端运行NI Linux Real-Time操作系统。RT终端上运行的实时操作系统决定了终端所支持的功能。

终端配置和属性

可通过MAX配置网络设置、安装软件、配置RT PXI终端的多以太网设备。关于在终端上安装软件的详细信息，见MAX帮助。右键单击RT终端，从快捷菜单中选择**属性**，打开RT终端的属性对话框。在对话框中设置RT终端的名称、IP地址、启用VI和Web服务器、权限等等。

> **注：** 在RT终端的属性对话框中改变RT终端的设置，设置部署至终端后生效。

部署RT终端设置

在RT终端的属性对话框中修改设置后，只有将设置部署到终端后，设置的改动才生效。如需部署终端，请右键单击项目浏览器窗口中的RT终端，从快捷菜单中选择**部署**。

连接和断开连接终端

可在项目中打开与RT终端的前面板连接。如要连接终端，右键单击RT终端，从快捷菜单中选择**连接**。如要断开连接，右键单击RT终端，选择**断开连接**。

项目浏览器窗口中RT终端的状态参见下表中的图标。

表 1. RT终端状态

	RT终端上有一个打开的前面板连接。
	RT终端上没有打开的前面板连接。

相关概念：

- NI Linux Real-Time操作系统

Real-Time系统的组成部分

实时系统由软件和硬件两部分组成。软件部分包括LabVIEW、RT引擎，以及LabVIEW项目和VI。硬件部分包括主机和RT终端。实时系统各个部分的介绍如下。

主机

主机是装有LabVIEW和LabVIEW Real-Time模块的计算机，在主机上开发实时系统的VI。开发实时系统VI后，可将VI下载到RT终端上。主机上可运行与RT终端交互的VI，作为用户界面。

LabVIEW

用户在主机上通过LabVIEW开发VI。Real-Time模块扩展了LabVIEW的功能，提供了创建、调试和部署确定性VI的额外工具。

LabVIEW项目

使用LabVIEW项目将LabVIEW文件和非LabVIEW文件组合，创建独立的实时应用程序，然后将VI和其他文件下载或部署到RT终端上。保存项目时，LabVIEW会创建一个项目文件(`.lvproj`)，其中包括对项目中文件的引用、配置信息、生成信息以及部署信息等。

RT引擎

RT引擎是在RT终端上运行的LabVIEW。RT引擎运行下载至RT终端的VI。RT引擎具有实时性，因为：

- RT引擎在实时操作系统(RTOS)上运行，这保证了LabVIEW执行系统和其他服务均为实时操作。
- RT引擎在RT系列硬件上运行。RT终端仅用于运行RT应用程序所需的VI和设备驱动，避免其他应用程序干扰RT VI的运行。
- RT终端不使用虚拟内存，因为虚拟内存可能会造成不可预期的结果。

RT终端

RT终端指运行RT引擎和VI的RT系列硬件。联网RT系列设备是具有嵌入式处理器和实时操作系统的硬件平台，实时操作系统用于用于RT引擎和VI。可使用单独的主机通过以太网与联网RT设备的控制VI进行通信。联网RT系列设备的范例如下：

- **NI RT系列PXI控制器**－安装在NI PXI机箱中的联网设备，与机箱中的NI PXI模块通信。可编写VI，使用PXI机箱中所有PXI模块、SCXI模块和其他信号调理设备的I/O功能。RT引擎也支持RT系列PXI控制器的功能。关于RT引擎支持的特定网络设备的功能，请参考NI网站上的LabVIEW Real-Time支持页。
- **NI CompactRIO系列**－可重配置的控制和采集系统，高速可靠。
- **NI RT系列FP和cFP**－运行RTOS的网络设备。
- **NI CVS-1457RT Compact Vision System**－易用的分布式实时图像系统，通过GigE Vision摄像头采集、处理和显示图像。
- **台式机RT终端**－配置RT引擎的台式机。关于将台式计算机配置为终端的详细信息，请参考将台式计算机作为RT终端与Real-Time模块配合使用。

注： LabVIEW帮助中不包含硬件相关的信息。关于设备的详细信息，请参考特定的设备文档。

USB存储设备

Real-Time模块支持在RT终端上使用USB存储设备，例如，闪存盘、USB硬盘。将外部USB存储设备连接至RT终端的USB端口，然后通过RT终端运行的VI访问这些设备。

把闪存盘插入RT系统时，闪存盘会被自动分配盘符`U:`。每个新增的驱动器都会被分配一个盘符，盘符为下一个可用的字母。例如，`V:`、`W:`、`X:`，等等。

警告 在操作过程中断开连接USB驱动器可能会导致数据损坏。为了保持数据一致性，断开连接USB驱动器之前先关闭驱动器上的所有文件。

相关概念：

- 将LabVIEW项目和Real-Time模块配合使用
- 使用Real-Time模块创建确定性应用程序

组织和管理LabVIEW Real-Time模块项目

设计项目之前，建议先完成LabVIEW Real-Time模块入门指南中的练习，熟悉项目和Real-Time模块的其他新功能。

改进FTP传输的技巧

无

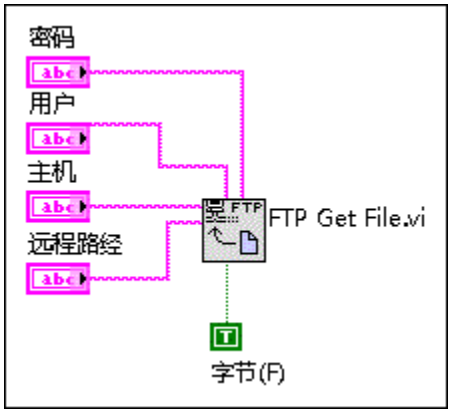
确保文件完整性

如要确保FTP传输文件的完整性，需选择适合的文件传输模式。下列传输模式可用。

- **ASCII**-文本文件主要使用的传输模式
- **二进制**-非文本文件主要使用的传输模式

如未选择传输模式，FTP VI将检查文件的扩展名，以检测要使用的传输模式。如传输的文件具有不常见的扩展名或不具有文件扩展名，该进程可能选择不正确的传输模式并损坏文件。

为了避免使用FTP VI时发生文件损坏，连线TRUE至**二进制**输入端以使用二进制模式传输文件。如下列程序框图所示：

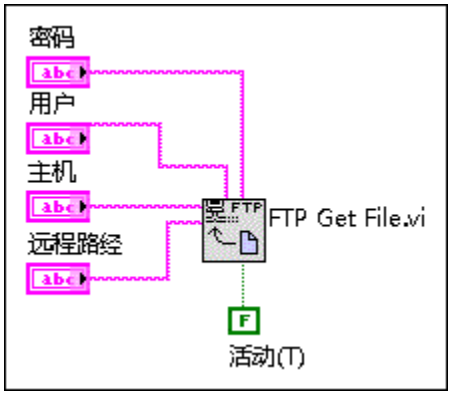


当使用第三方FTP程序时，请参阅FTP终端帮助文件获取维持文件完整性的信息。

确保连接的稳定性

为避免主机防火墙拦截连接，NI建议使用被动式FTP传输。通过主机启动与RT终端的连接，被动传输减少了与防火墙相关的不稳定性。

使用FTP VI时，如要设置传输类型为被动，可连线FALSE至**主动**接线端。如下列程序框图所示：



注：

- 被动FTP传输相比主动FTP传输将占用更多的临时连接端口。设计应用程序时，请考虑应用程序使用的连接端口数量。
- 使用FTP VI从RT终端发送文件至另一台计算机上的第三方FTP程序时，可能产生额外的问题。详细信息见FTP终端的帮助文件。

被动传输的另一种替代方法是配置用户防火墙，以允许FTP传输。允许通过防火墙的程序包括LabVIEW或LabVIEW创建的执行程序及第三方FTP程序。

将LabVIEW项目和Real-Time模块配合使用

LabVIEW项目用于组织LabVIEW文件和非LabVIEW特有的文件，创建程序生成规范，部署或下载VI至RT终端。保存项目时，LabVIEW会创建一个项目文件(`.lvproj`)，

其中包括对项目中文件的引用、配置信息、生成信息以及部署信息等。必须通过项目使用RT终端，以及生成独立的应用程序。

Real-Time项目向导

选择**工具»Real-Time模块»项目向导**，使用Real-Time项目向导创建一个新项目。

项目浏览器窗口

也可通过项目浏览器窗口手动创建一个新项目。通过项目浏览器窗口创建和编辑LabVIEW项目。选择**文件»创建项目**打开**项目浏览器**窗口。

相关概念：

- Real-Time系统的组成部分
- 通过项目浏览器窗口创建和编辑LabVIEW项目

通过Real-Time项目向导创建LabVIEW项目

使用Real-Time项目向导创建一个新项目，项目文件定义RT终端，并包括VI和其他应用程序文件。可通过Real-Time项目向导创建下列三种架构的项目——连续通信、状态机，以及自定义。

连续通信架构

连续通信架构用于创建连续记录数据至磁盘，并与主机通信提供用户界面的数据采集应用程序。

在连续通信架构下，可选择使用一个或两个定时循环的RT终端VI作为应用程序配置。

- 一个循环－RT终端VI使用一个定时循环控制应用程序任务的定时和执行。
- 两个循环－RT终端VI使用两个优先级不同的定时循环控制应用程序任务的定时和执行。高优先级的定时循环控制确定性任务。低优先级的定时循环控制用户界面通信和文件I/O任务。

可通过在主机上运行的VI或通过LabVIEW远程前面板连接至RT终端VI，提供实时应用程序的用户界面。

- 主控VI－使用LabVIEW共享变量从RT终端VI发送用户界面数据。在主机上运行的VI访问用户界面数据，然后将数据显示在前面板上。
- 远程前面板－使用LabVIEW远程前面板查看RT终端VI的前面板控件。如应用程序要求较高的确定性，则不能在只含有一个定时循环的VI上使用远程前面板功能。Real-Time项目向导创建一个HTML文件，可将该文件通过Web服务器发布到RT终端上。可使用该HTML文件从主机访问RT终端VI的前面板。

状态机架构

使用状态机架构实现复杂的决策算法，决策算法可由状态图和流程图表示。状态机架构可实现摩尔机描述的任何算法。对于状态图中的任意一种状态，摩尔机都会进行一次操作，操作持续的时间长度有限。要避免数据丢失，执行每个操作的时间必须足够短，使终端上所有数据都在内存中不丢失。RT终端还必须能够将所有数据在一个数据包中发送至主机。状态机架构将通信任务和非确定性任务与确定性任务区分，并在确定性任务开始前后执行。

Real-Time项目向导会生成一个使用条件结构的RT终端VI，并定义状态图的状态。Real-Time项目向导还会生成一个在主机上运行并提供用户界面的VI，可用于选择在RT终端上运行的状态。主控VI使用共享变量触发RT终端上特定状态的执行。RT终端VI执行用户选择的子程序框图，并将结果返回至主控VI。

注： 可使用LabVIEW Statechart模块，在实时终端上设计和实施基于状态的应用程序。关于Statechart模块的详细信息，请参考NI网站。

自定义项目

使用自定义项目结构，添加一个空白VI或导入现有VI至主机或RT终端。

通过项目浏览器窗口创建和编辑LabVIEW项目

通过项目浏览器窗口创建和编辑LabVIEW项目。选择**文件»创建项目**打开**项目浏览器**窗口。或选择**项目»创建项目**或在**新建**对话框中选择**项目**，打开**项目浏览器**窗口。

用户可在**项目浏览器**窗口统一管理终端、VI以及其他项目支持文件。**项目浏览器**窗口还用于连接至RT终端、设置终端属性以及部署VI至终端。添加RT终端时，**项目浏览器**窗口包括下列部分：

- **项目根目录**－包含当前项目的主机和RT终端。项目根目录的标签包括该项目的文件名。右键单击项目的根目录，从快捷菜单中选择**新建»终端和设备**，添加RT终端至项目。
  - **我的电脑**－表示项目中用作终端的本地计算机（即主机）。
  - **RT终端**－添加至项目的RT终端。默认情况下，RT终端的名称是在NI Measurement & Automation Explorer (MAX)中指定的终端名称。添加至RT终端的VI和库显示在**项目浏览器**窗口的终端下。右键单击RT终端，从快捷菜单中选择**属性**，可配置终端设置。
    - **依赖关系**－用于查看某个终端下VI所需的项。
    - **程序生成规范**－生成源文件发布、独立实时应用程序以及zip文件的规范。如已安装LabVIEW专业版开发系统或应用程序生成器，右键单击**程序生成规范**，从快捷菜单中选择**新建»实时应用程序**，可创建定义生成独立实时应用程序的程序生成规范。

相关概念：

- 将LabVIEW项目和Real-Time模块配合使用

添加RT终端到LabVIEW项目

使用添加终端和设备对话框向LabVIEW项目添加终端或设备。右键单击项目的根目录，从快捷菜单中选择**新建»终端和设备**，打开**添加终端和设备**对话框。在项目中添加RT终端时，LabVIEW会在项目浏览器窗口中创建一个新项，表示RT终端。如添

加的RT终端支持其他终端，可右键单击RT终端，从快捷菜单中选择**新建»终端和设备**，在现有RT终端下添加新的终端。例如，RT系列PXI控制器上装有FPGA终端，可将FPGA终端添加至RT PXI控制器。

在多个RT终端上使用选板

LabVIEW可为每个RT终端单独加载选板。如果在一个项目的多个终端上都有打开的VI，那么内存中将存在多个选板。在RT终端间切换时，LabVIEW将尽量保持各个锁定的选板与活动RT终端中对应选板的映射关系。如果锁定的选板没有对应的选板，LabVIEW将显示该活动RT终端的顶层选板。

部署和运行RT终端上的VI

部署是指将VI和关联文件下载至RT终端。在终端下运行VI时，LabVIEW将VI、VI的依赖关系项，以及终端设置部署至RT终端。

从项目浏览器窗口部署VI

可在项目浏览器窗口向RT终端添加VI和关联文件。右键单击**项目浏览器**窗口RT终端下的项，从快捷菜单中选择**部署**，即可将该项部署至终端。

RT终端上的部署位置

下表列出了RT终端上下载至磁盘和内存的项。

项	下载位置
属性、设置及生成的应用程序	磁盘
VI、库及共享库	存储器

部署使用LabVIEW类或调用成员VI的VI

如部署使用了LabVIEW类或调用了成员VI的VI，只有应用程序中引用的VI和类才会被部署到终端。如已经部署的VI引用其他动态分配的成员VI，重写动态成员VI的VI也

将被部署。

禁用自动变量部署

默认情况下，运行VI时LabVIEW会自动部署所有VI引用的共享变量，包括I/O变量。但是，在某些情况下，需要禁用自动变量部署。下表列出了启用和禁用自动变量部署的范例。

自动变量部署	使用场景	优点
启用	开发VI。	按下**运行**按钮即可运行VI，无需单独部署或重新部署VI引用的变量。
禁用	运行引用变量的主控VI，该变量在运行启动VI的RT终端上。	允许运行主控VI，不会发生部署冲突。

在**项目浏览器**窗口右键单击主机或RT终端，从快捷菜单中选择**禁用自动部署变量**，可禁用主机或RT终端上的自动变量部署。

编译目标缓存

LabVIEW将用于部署VI的已编译代码存放在编译目标缓存中。编辑或重新部署VI时，LabVIEW会更新编译目标缓存。

> **注：** 如删除之前部署至RT终端的大量VI，可清空已编译目标缓存，释放RT终端的磁盘空间。LabVIEW必须重新编译未删除的VI，下次部署时LabVIEW可能需要较长时间。清空目标缓存之前需仔细考虑对性能的影响。

访问RT终端VI的前面板

用户可连接至RT终端并访问终端内存中VI的前面板。

连接—右键单击**项目浏览器**窗口的RT终端，从快捷菜单中选择**连接**，打开与终端

的前面板连接。LabVIEW将打开VI的前面板，表示这些VI在RT终端的内存中。关闭前面板将从终端的内存中移除这些VI。

断开连接－右键单击**项目浏览器**窗口的RT终端，从快捷菜单中选择**断开连接**，断开与终端的前面板连接，VI仍在终端的内存中，处于运行状态。

在CompactRIO终端上部署设置

将配置部署到连接I/O机箱的CompactRIO终端，需要考虑下列三个部分：

- **项目**－LabVIEW项目中配置的终端设置。
- **终端**－终端上当前部署的设置。
- **机箱**－连接至机箱的物理I/O模块。

保持项目、终端和机箱配置同步

添加、移除或移动模块时，可能会使项目、终端和机箱配置失去同步性。

- **添加I/O模块**－向空机箱插槽添加模块时，必须将新添加的模块添加至项目并部署项目项至终端，以保持模块的同步性。在**项目浏览器**窗口右键单击机箱，选择**新建»C系列模块**，打开添加终端和设备对话框。使用该对话框找到新模块，并将模块添加至项目。添加新模块至项目后，右键单击模块，选择**部署**，将新配置同步至终端。如添加多个新模块至项目，右键单击机箱，选择**部署**，一次性将所有模块部署至终端。
- **删除I/O模块**－从机箱插槽中移除模块时，必须在**项目浏览器**窗口右键单击模块并选择**取消部署**，从终端上移除模块配置以保证同步性。然后，可从项目中删除模块。
- **修改I/O模块类型**-如指定机箱中的模块位置与实际CompactRIO机箱模块的位置不匹配，在**项目浏览器**窗口右键单击原有的模块，然后从快捷菜单中选择**取消部署**。然后在**项目浏览器**中右键单击机箱，选择**新建»C系列模块**，打开**添加终端和设备**对话框。使用**添加终端和设备**对话框查看新添加的模块并将模块添加至项目。

NI扫描引擎部署的错误排查步骤

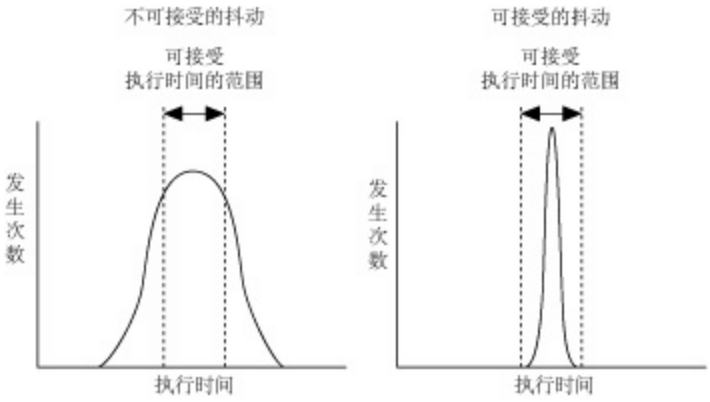
如发生与NI扫描引擎相关的部署问题，可尝试下列错误排查步骤：

1. **安装和初始化NI扫描引擎**－必须在RT终端上安装NI扫描引擎支持才能部署使用NI扫描引擎的项和设置。如已安装NI扫描引擎的终端重新上电或重启，必须等待NI扫描引擎初始化后才能开始部署终端。
2. **强制部署**－为了减少与终端的通信，LabVIEW只部署与上次部署配置不同的I/O模块和I/O变量。因此，在某些情况下，即使没有任何信息发送至终端，终端和主机没有进行同步操作，也会显示部署成功或`Nothing to deploy`消息。如同步终端、项目和机箱的问题反复出现，可先取消部署I/O项，然后重新部署，以实现强制部署。
3. **清除NI扫描引擎配置**－如强制部署未解决问题，则可能需要从终端上清除已部署的NI扫描引擎设置。通过清除NI扫描引擎的问题，创建至主机的FTP连接，移除主机上`ni-rt/config/`目录下的所有文件。(NI Linux Real-Time)建立至终端的SSH连接，并删除`/var/local/natinst/deployfwk/config/`目录下的全部文件。清空相应目录后，重新启动终端。在**项目浏览器**窗口右键单击终端，从快捷菜单中选择**连接**，重新部署终端设置。
4. **重新格式化终端**－如其他方法都失败，可考虑重新格式化终端和重新安装软件。在终端上重新安装软件后，可重新部署文件至终端。

注： 连接至某个终端，LabVIEW检测出当前项目缺少部署项时，LabVIEW会在冲突解决对话框中报告一个错误，并提供自动取消部署这些项的选项。可使用项目和系统比较对话框查看和解决项目和部署项设置之间的差别。

创建确定性应用程序

确定性是系统的特性之一，它用于描述系统响应外部事件的一致性，即系统要在给定事件范围内完成操作。抖动是指程序的执行事件没有满足确定性预期。多数实时应用程序要求执行上所需时间的一致性，可接受的抖动量通常很小。如下图所示：



如要创建一个执行确定性任务的确定性应用程序，使用该部分建议的编程技巧减少RT终端上可能出现的抖动。

注： RT循环通常需要1-2个预处理循环，而后才开始确定性执行。检查应用程序是否满足定时要求前，应使各个实时循环先执行若干预处理循环。

相关概念：

- Real-Time Trace Viewer

使用Real-Time模块创建确定性应用程序

确定性应用程序受益于实时操作系统(RTOS)的多线程和基于优先级的调度模式。创建确定性应用程序时，应将应用程序任务分开以创建多线程的应用程序，根据RTOS设置的优先级接收专有的处理器资源。

多线程

只有一个CPU的计算机通过交替运行应用程序实现多任务处理。只要处理器分配给各个应用程序的时间足够小，计算机就仿佛在同步运行多个应用程序。具有多个CPU的计算机可实现真正的并行机制。

多线程是将多任务的概念应用到应用程序中，将一个应用程序分解为多个交替执行的较小任务，每个任务在不同的线程内执行。线程是在执行系统的应用程序中独立的执行流。多线程应用程序将处理器效率最大化，有其他线程运行就绪时，处理器

不会处于闲时状态。读取写入文件，包含I/O操作或轮询用户界面的应用程序都能受益于多线程的优点，因为在上述任务间隔的空隙，应用程序可利用处理器运行其他任务。

基于优先级的调度模式

RT终端上的RTOS使用循环和抢占式的排序方法对执行线程进行排序。

轮询调度－为所有线程使用相同的优先级。相同优先级的线程分配的处理器时间相同。例如，每个普通优先级的线程都有10 ms运行时间。处理器在10 ms内执行任务，未完成的任务等待在下一个时间周期中完成。

抢占式调度－较高优先级线程立即执行，较低优先级线程暂停执行。实时优先级是最高级别的优先级。

定时结构的优先级

LabVIEW定时结构的优先级低于实时优先级高于高优先级。用户可设置定时循环的优先级，指定某个定时循环相对于VI中其他定时循环的相对优先级。使用配置定时循环对话框配置定时源、周期、优先级和其他定时循环执行的高级选项。

划分任务创建确定性多线程应用程序

应用程序的确定性取决于各个任务都在指定时间范围内完成。因此，确定性任务需要专有的处理器资源以保证按时完成。划分任务保证了每个任务都能获得所需的处理器资源，并在按时完成任务的执行。

将确定性任务与其他任务分离可保证确定性任务有足够的处理器资源。例如，应用程序以固定时间间隔采集数据，并将数据保存在磁盘上，数据采集的定时和控制必须是确定性的。但是，将数据保存在磁盘上不是一个确定性的任务，因为文件I/O操作具有不可预期的响应时间。文件I/O的时间取决于硬件和可用的硬件资源。可不同优先级的定时循环或VI控制确定性任务的执行和定时属性。

注： 在确定性任务中，确保每个操作都有专有的处理器资源，避免不必

> 要的并行机制。在多CPU计算机上，创建的操作数不要大于可用的CPU的数量。因为无法确定并行操作的执行顺序，不必要的并行处理会影响程序的确定性。

相关概念：

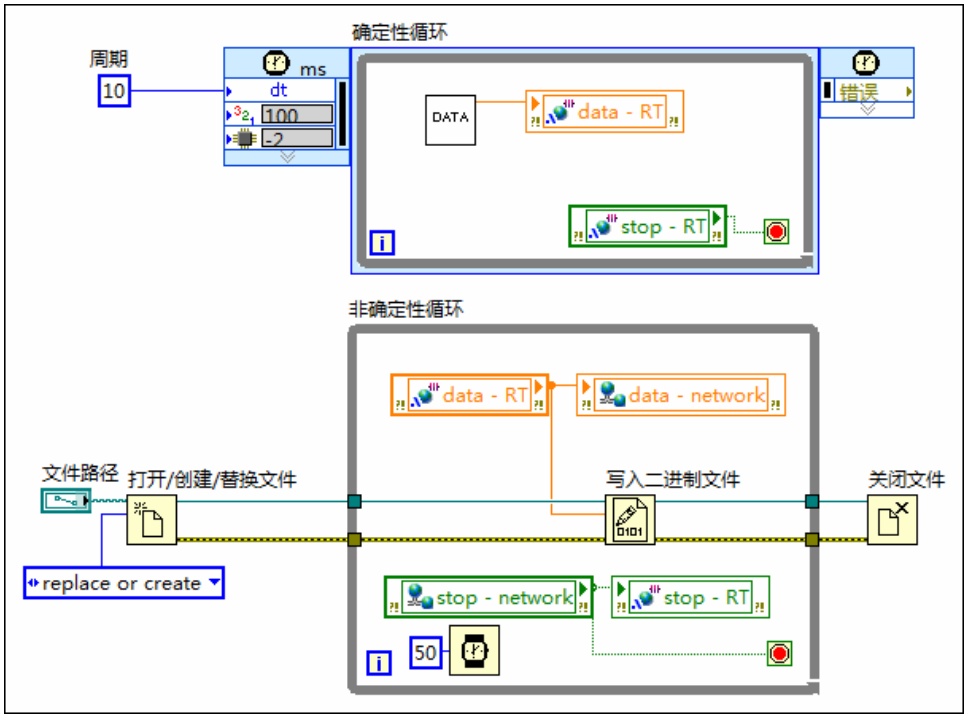
- 使用定时循环创建确定性应用程序
- 使用优先级不同的VI创建确定性应用程序
- 优化多CPU系统上的实时应用程序
- LabVIEW Real-Time模块简介
- Real-Time系统的组成部分
- 确定性应用程序的时间控制

使用定时循环创建确定性应用程序

将确定性任务与非确定性任务分开，并将确定性任务放在RT终端VI的不同循环中，保证确定性任务有足够的处理器资源。定时循环根据指定周期和优先级在每次循环时执行一个子程序框图。定时循环的优先级越高，它相对于程序框图中其他定时结构的优先级便越高。可使用确定性通信方法共享来自确定性定时循环的本地数据，或共享网络上的远程数据。

> **注：** 在实时应用程序中使用定时循环时，要注意避免抖动。

下列程序框图显示了一个RT终端VI，包含一个定时循环和一个While循环，用于控制数据采集和记录应用程序：



确定性循环的优先级为100，其中的子VI在每次循环时采集数据。确定性循环通过共享变量**data**与非确定性循环共享采集到的数据。启用了实时FIFO的共享变量可确定性地共享RT终端VI的数据，但是不影响VI的确定性。非确定性循环读取共享变量**data**，将采集到的数据记录到RT终端的磁盘上。

定时循环执行的协同因素

定时循环具有独占性，可占用所有处理器资源。一个定时循环可占用所有处理器资源，不允许程序框图上的其他较低优先级任务执行。

确定性定时循环的周期的长度必须配置为足够完成确定性任务，以及包含充足的空闲时间允许低优先级任务执行。通过对定时循环设定周期，可减少确定性任务的执行时间，将处理器资源让渡给程序框图上优先级较低的任务。

注： 如RT应用程序在多CPU系统上运行，可考虑为实时定时循环分配一个专有CPU。

相关概念：

- 在RT终端上共享本地数据
- 通过网络远程共享数据
- 通过共享变量共享数据
- 确定性应用程序的时间控制
- 优化多CPU系统上的实时应用程序
- 使用Real-Time模块创建确定性应用程序

使用优先级不同的VI创建确定性应用程序

将确定性任务与非确定性任务分开，并把确定性任务放在不同优先级的VI中，确保足够的处理器资源。可对VI设置不同的优先级，然后将其归入一个可用的执行系统，控制分配给各个VI的处理器资源。

根据指定的VI优先级和执行系统，LabVIEW为每个VI分配一个执行系统线程。线程在处理器上执行。

分配VI优先级

在项目浏览器窗口右键单击VI，从快捷菜单中选择**属性**，打开VI属性对话框，可修改VI的优先级。在**VI属性**对话框中，从**类别**下拉列表中选择**执行**，打开执行页设置VI的优先级。可从下列VI属性中选择，优先级从低到高排列：

- 后台优先级（最低）
- 标准优先级（默认）
- 高于标准优先级
- 高优先级
- 实时优先级（最高）

注意 LabVIEW使用两种优先级机制，VI优先级和定时结构优先级。这两种优先级机制既相互独立，又相互关联。定时结构优先级是数值，数值越大，表示相对于终端上的其他定时结构优先级越高。但是，所有定时结构优先级都在高和实时VI优先级上。建议在一个应用程序中只使用一种优先级机制，避免非预期结果。如应用程序使用定时结构，将所有VI设为普通

优先级。

标准优先级是VI的默认优先级。子VI继承调用方VI的优先级。例如，确定性VI调用的子VI为实时优先级。VI的实时优先级高于所有其他优先级。实时优先级的VI直到完成所有任务后才释放处理器资源。确定性VI可显式地让渡处理器资源，确保不独占处理器资源。

注： 因为实时优先级VI之间无法抢占资源，所以每个CPU上只能有一个实时VI。这样才能保证VI执行的确定性。

除了上述五个优先级之外，还可将VI设置为子程序优先级。设置成子程序级别的VI不与其他VI共享执行时间。当VI处于子程序优先级时，它有效地控制正在执行的线程，并与其调用者运行在同样的线程上。在子程序VI完成运行之前，其他VI不能在这个线程上运行，即使这些VI也是子程序级的。

为VI分配执行系统

在**项目浏览器**窗口右键单击VI，从快捷菜单中选择**属性**，打开VI属性对话框，可修改VI的执行系统。在**VI属性**对话框中，从**类别**下拉列表中选择**执行**，打开执行页设置VI的执行系统。LabVIEW有六个执行系统：

- 用户界面
- 标准
- 仪器I/O
- 数据采集
- 其他1
- 其他2

默认情况下，所有VI都在标准执行系统中运行。用户可根据实际需要，将VI分配至其他执行系统。除了上述六个执行系统之外，也可将子VI的执行系统设置为与调用方相同。设置为与调用方相同表示子VI的执行系统与调用该子VI的顶层VI相同。

除了用户界面之外，每个执行系统都有一个按指定优先级执行VI的线程队列。例如，将三个VI分配至某执行系统，无论何时，该执行系统中都是一个VI运行，其他两个等待。假设所有VI有同等的优先级，线程先运行一段时间。然后，线程移至队列末尾，下一个线程开始运行。线程结束后，执行系统将线程从队列中移出。

用户界面执行系统处理所有用户界面任务。其他执行系统不负责用户界面。如VI需更新用户界面，执行系统将任务传递给用户界面执行系统，请求更新用户界面。

确定性VI执行

确定性VI具有抢占性，可独占处理器资源。确定性VI可能会占用所有处理器资源，不允许其他低优先级任务（例如，低优先级VI、RT终端的FTP服务器）执行。

确定性VI可在不影响代码确定性的前提下允许其他低优先级任务执行。通过设置定时VI的时间，可保证VI在合适的时候释放处理器资源。

注： 在多CPU系统上运行的应用程序中，可手动为确定性定时循环分配专有的处理器。如通过定时循环为VI分配专有CPU，VI可专享处理器资源，而不对其他任务造成负面影响。

相关概念：

- 避免子VI开销
- 确定性应用程序的时间控制
- 优化多CPU系统上的实时应用程序
- 使用Real-Time模块创建确定性应用程序
- 在RT终端上共享本地数据

确定性应用程序中的数据共享

在应用程序中分离确定性任务和非确定性任务后，必须使用确定性通信方法共享数据。使用确定性通信方法可在VI中不能使用连线的两个位置之间、在RT终端上的VI之间、或联网的不同终端的VI之间共享数据。

相关概念：

- 通过流水线优化多CPU实时应用程序

在RT终端上共享本地数据

将应用程序中的任务分为独立的定时循环或不同优先级的VI后，程序框图的循环之间或RT终端上不同的VI之间可能需要进行通信。使用共享变量和Real-Time FIFO VI在RT终端上的VI之间和VI内部共享数据。

> **注：** 也可使用全局变量和功能全局变量在本地共享数据。但是，这些通信方法会对确定性产生不利影响。

单进程共享变量

使用单进程共享变量在程序框图的两个位置之间，或RT终端上的两个VI之间传递数据。在项目浏览器窗口右键单击RT终端，从快捷菜单中选择**新建»变量**，打开共享变量属性对话框，从中创建单进程共享变量。

Real-Time模块向共享变量添加了实时FIFO（先进先出缓存）功能。通过启用共享变量的RT FIFO功能，可共享数据而不影响RT终端上VI的确定性。在**共享变量属性**对话框的“Real-Time FIFO”页，勾选**启用Real-Time FIFO**复选框，即可启用共享变量的实时FIFO。

启用RT FIFO之后，单进程共享变量是易用且确定性高的通信方法。

> **注：** 如使用启用Real-Time FIFO的共享变量传输波形数据，共享变量不传输波形中的变体元素，因为变体的大小不确定，与Real-Time FIFO不兼容。

Real-Time FIFO函数

使用RT FIFO函数在RT终端上运行的VI之间共享数据。RT FIFO类似一个固定大小的

队列，写入FIFO的第一个值是能从FIFO读取的第一个值。RT FIFO限制共享数据的大小，并为数据预分配内存，以保证程序执行的确定性。必须定义RT FIFO元素的数量和大小，确保不读取和写入不同大小的数据。

使用RT FIFO创建函数创建一个新的FIFO，或创建对之前已存在FIFO的引用。使用RT FIFO读取和RT FIFO写入函数，读取和写入数据至FIFO。使用RT FIFO删除函数删除RT FIFO的引用，释放RT终端上分配的内存。

注： 如使用Real-Time FIFO传输波形数据，则无法传输波形的变体元素，因为变体的大小由变量确定，与Real-Time FIFO不兼容。

定义Real-Time FIFO的读取和写入模式

RT FIFO会等待直到有空槽可写入或有值可读取时，才执行读取或写入操作。用户可指定RT FIFO的写入和读取模式。写入和读取模式定义了从空的FIFO读取或写入已满的FIFO时的处理方式。可指定下列读取或写入模式：

轮询－该模式能最大化读取和写入操作的数量。轮询模式连续轮询FIFO，检测是否有新数据或空槽。轮询模式对新数据的新出现空槽的响应比阻隔模式更快，但是需要更多的CPU开销。使用RT FIFO读取或RT FIFO写入函数的**超时(ms)**输入端指定写入操作轮询空槽或读取操作轮询新进数据的时间。也可使用RT FIFO写入VI的**超时覆盖**输入端指定当**超时毫秒**输入端的值过期时，是否覆盖RT FIFO中最早写入的值。

阻隔－该模式下，读取和写入操作的CPU利用率最好。阻隔模式使VI的线程等待时进入休眠状态，允许系统中其他任务执行。使用RT FIFO读取或RT FIFO写入函数的**超时(ms)**接线端指定读取操作等待新值的时间，或写入操作等待空槽的时间。也可使用RT FIFO写入VI的**超时覆盖**输入端指定当**超时毫秒**输入端的值过期时，是否覆盖RT FIFO中最早写入的值。

注： 如使用“RT FIFO创建”函数返回现有RT FIFO的引用，引用使用现有FIFO的读取和写入模式，忽略“读取/写入模式”接线端指定的模式。

使用RT FIFO函数创建大型应用程序的可扩展程序框图

如要为大型应用程序创建一个可扩展的任务间通信架构，使用“RT FIFO创建”函数，通过编程创建RT FIFO。例如，可在For循环中使用“RT FIFO创建”函数，创建所需数量的RT FIFO。然后在For循环中使用RT FIFO读取和RT FIFO写入函数，连续从RT FIFO中写入和读取数据。通过该方法，可方便地扩展应用程序的RT FIFO，同时保持程序框图整洁易读。

注： 使用RT FIFO创建函数的轮询模式，优化读取和写入操作的速度。使用阻隔模式，减少CPU开销。轮询模式连续轮询FIFO，检测是否有新数据或空槽。轮询模式对新数据的新出现空槽的响应比阻隔模式更快，但是需要更多的CPU开销。在多CPU系统中，考虑使用轮询模式并为轮询RT FIFO函数分配专有的CPU。

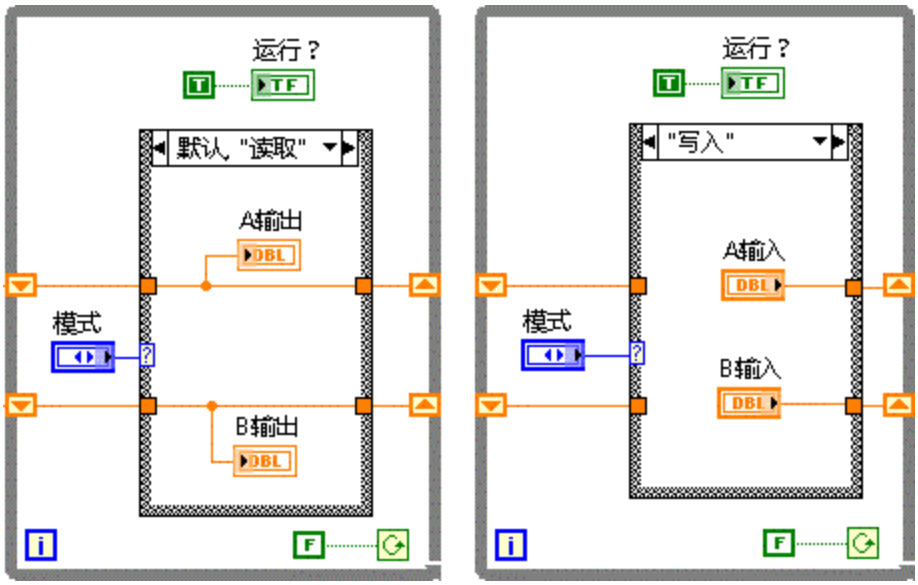
全局变量

使用全局变量在两个VI之间访问和传递少量数据，例如，实时VI和低优先级VI。全局变量可在两个VI之间确定性地共享小于32位的数据，例如，换算数据。全局变量的数据类型较大，是共享资源，必须在实时VI中谨慎使用。如使用数据类型大于32位的全局变量将输出从实时VI中传出，必须保证实时VI再次写入全局变量之前，由较低优先级的VI读取全局变量的数据。

全局变量不是一种严格的通信方式，全局变量的值如未被及时读取，就有可能会被覆盖。较低优先级VI中的VI可能没有足够的时间在其他VI的任务覆盖数据之前先读取数据。

功能全局变量

功能全局变量用于在VI之间传输数据。功能全局变量是设置为子程序优先级的子VI。子VI包含一个While循环，While循环中又有一个进行读取和写入操作的条件结构。下列程序框图显示了功能全局变量中条件结构的读/写分支：



While循环包含未初始化的移位寄存器，用于存储数据。功能全局变量的**模式**输入参数接收一个指定VI进行任务的操作输入，如上述程序框图所示。从这步之后对功能全局变量的调用都可访问最新的数据。功能全局变量类似于队列，可添加更多的移位寄存器来保存更长历史记录。也可添加一组以上移位寄存器，传递更多数据。

与共享变量不同，功能全局变量不是共享资源。右键单击子VI，设置为子程序优先级并从快捷菜单中选择**遇忙时跳过子程序调用**。如子程序被调用时正在其他线程中运行，则跳过子VI。实时VI不等待在其他线程中运行的子VI，跳过子VI有助于保证实时VI的确定性。跳过子VI执行程序，子VI返回的值可能有误。如要检测功能全局变量的执行，将TRUE常量连接至功能全局变量的布尔输出，保证显示控件的默认值为FALSE。如布尔输出返回TRUE，功能全局变量执行。如布尔输出返回默认值FALSE，功能全局变量不执行。在实时VI中跳过功能全局变量，但在低优先级VI中不跳过，可以等待接收非默认值。

VI读取移位寄存器的值之前，移位寄存器的值可能被覆盖，所以，功能全局变量不是一种严格的通信方式。

相关概念：

- 使用定时循环创建确定性应用程序
- 使用优先级不同的VI创建确定性应用程序
- 通过共享变量共享数据

- 通过网络远程共享数据

通过网络远程共享数据

创建在RT终端上运行的VI后，可能需要与主机或其他终端上的VI共享数据。使用远程通信方法，在RT终端VI和其他终端VI之间共享数据。

使用远程通信方法，RT终端上的VI可与其他终端上的VI通过网络传输数据。在下列情况下需远程通信：

- 要在主机和RT终端之间通过用户界面控制交换的数据。用户可自定义通信VI指定从主机查看的内容。
- 要控制数据传输的时间和顺序。
- 要在其他终端上进行额外的数据处理或记录。

例如，使用共享变量在RT终端的两个VI之间本地共享数据。然后使用远程通信方法与主机或其他终端上的VI共享数据。

相关概念：

- 使用定时循环创建确定性应用程序
- 远程通信方法
- 在RT终端上共享本地数据

远程通信方法

可使用高层的软件协议在RT终端的两个VI之间，以及与其他终端的VI通信。每种协议都有其优点和劣势。下面列出了各种通信方法：

- 网络通信－用于以太网通信。
  - 网络流
  - 网络发布的共享变量
  - TCP
  - UDP

    - VI服务器
    - Logos
    - SMTP（仅限于发送）
- 总线通信－用于不同总线端口之间的通信。
    - 串口
    - CAN

网络通信

下面介绍与LabVIEW Real-Time模块配合使用的常见通信协议，帮助您找到最适合应用程序的网络协议。

网络流

可使用网络流在RT终端和主机之间流数据或发送命令。网络流不损耗数据，是单向的一对一通信通道，由写入方和读取方端点构成。

网络发布的共享变量

可使用网络发布的共享变量在不同终端运行的VI之间传输数据。启用共享变量实时FIFO后，在网络间共享数据不会影响VI的确定性。但是，通过网络传输数据是非确定性的。因为网络延迟，机器上运行的VI可能无法通过网络读取最新写入的数据。在该情况下，VI试图从网络发布的共享变量读取数据，结果却返回上一次读取的值。对于数据记录应用程序，可使用时间标识通过程序确保每次记录一个值。

TCP

TCP是网络通信的业界标准。主机上的VI可通过TCP VI和函数与RT终端VI通信。但是，TCP具有非确定性。在确定性VI中使用TCP通信会影响VI的确定性。

Real-Time模块拓展了现有TCP函数的功能，使TCP函数能用于联网RT系列设备。

UDP

UDP是在网络上两个位置之间传输数据的网络传输协议。UDP不是基于连接的协

议，发送方和接受方不必建立网络连接。因此，通过UDP传输数据，系统开销较小。但是，UDP具有非确定性。在确定性VI中使用UDP通信会影响VI的确定性。

使用UDP VI和函数发送数据时，发送方发送数据之前，接受方计算机必须有一个读取端口。使用UDP打开函数，打开写入端口，指定接收方计算机的IP地址和端口。数据通过字节流传输，字节流的长度称为数据报。数据报到达侦听端口，接受方计算机缓存数据，然后读取数据。

可使用UDB双向传输数据。在双向数据传输下，两台计算机都分别指定读取和写入端口，然后使用相应的端口发送或接收数据。可使用UDP发送数据至RT终端的VI或接收RT终端VI的数据。

UDP可进行高速数据传输。但是，UDP无法保证所有数据报都能到达接收方计算机。因为UDP不要求网络连接，所以无法验证数据报是否到达。必须保证网络繁忙不会影响数据报的传输。另外，读取接收方计算机数据缓冲区的数据必须足够快，否则容易发生数据溢出。

VI服务器

VI服务器可用于监控RT终端上的VI。VI可通过VI服务器远程调用RT终端VI。VI可将参数值传输至RT终端VI，或传输来自RT终端VI的值，从而实现分布式应用。

主机VI只能调用已保存在RT终端上的VI。主机VI不能通过VI服务器动态下载VI至RT终端。

VI服务器可在LabVIEW的框架内利用TCP，这是VI服务器的优点。但是，VI服务器不具有确定性，使用VI服务器通信可能会影响确定性VI的确定性。

SMTP Email

使用SMTP Email VI发送RT终端VI的数据至另一台计算机上的VI。SMTP VI可通过简单邮件传输协议(SMTP)发送包含数据和文件附件的电子邮件。使用SMTP选板上的VI不能用来接收信息。

SMTP具有非确定性。在确定性VI中使用SMTP通信会影响VI的确定性。

Logos（仅限于FieldPoint）

Logos用于在主界面和传统FP或cFP以太网模块之间传输数据。以太网FP模块有一个嵌入的Logos服务器，可从远程服务器FP通道组上读写物理I/O模块。

总线通信

串口

串口通信是指通过串口两点之间的通信。串口VI和函数提供LabVIEW中的串口通信支持，在具有串口设备的RT终端和计算机之间进行通信。数据率低且传输路程较远时，串口通信是理想的选择。必须在NI Measurement & Automation Explorer (MAX)中将NI-Serial RT软件添加至RT终端。关于在RT终端上安装软件的详细信息，见MAX帮助。

串口通信具有非确定性。在确定性VI中使用串口通信会影响VI的确定性。

CAN

控制器区域网(CAN)是确定性的多点通信总线协议，为ISO 11898国际标准。通过CAN，每帧可传输8个数据字节，传输速度达1 Mbit每秒。可通过NI-CAN接口的板卡和NI-CAN驱动连接多个RT系统。必须在MAX中将NI-CAN RT软件添加至RT终端。关于在RT终端上安装软件的详细信息，见MAX帮助。

访问`ni.com/info`并输入`CANManual`，查看NI-CAN硬件和软件手册，其中包含将NI-CAN硬件和软件与LabVIEW配合使用的信息。

相关概念：

- 通过共享变量共享数据
- 通过网络远程共享数据

通过共享变量共享数据

共享变量可用于在项目的VI之间或联网VI之间读写数据。在**项目浏览器**窗口右键单

击RT终端，从快捷菜单中选择**新建»变量**，打开共享变量属性对话框，从中创建、使用和配置共享变量。共享变量是可在下列情况下传递数据：程序框图两个无法连线的位置之间、RT终端上运行的两个VI之间、不同RT终端上联网的VI之间。单进程共享变量用于在一个RT终端上本地共享数据，网络发布共享变量用于在不同计算机的数据记录VI之间共享数据。

Real-Time模块在共享变量中新增了FIFO功能。启用共享变量实时FIFO后，共享数据不会影响VI的确定性。RT FIFO不支持大小变化的数据类型（例如，簇、字符串和变体）。

注： 关于共享变量的性能比较和高级编程理论，请访问NI网站。

创建共享变量

使用共享变量属性对话框在RT终端上创建共享变量。要访问该对话框，在**项目浏览器**窗口右键单击终端或库，从快捷菜单中选择**新建»变量**。在终端上选择**新建»变量**，可在终端下新建一个库，并将变量添加至库。在现有库中选择**新建»变量**，可在库中新建一个变量。右键单击库，从快捷菜单中选择**部署**，将共享变量部署至终端。

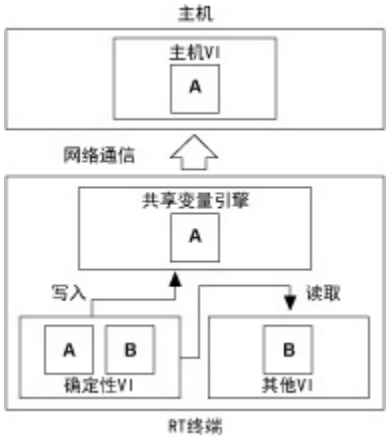
注： 调用变量的VI运行时，LabVIEW自动部署变量，除非用户禁用RT终端的自动部署选项。

共享变量引擎

在终端上创建网络部署共享变量时，终端上的共享变量引擎即开始成为共享变量的服务方。共享变量引擎将共享的数据通过网络发送至其他终端。在**项目浏览器**窗口选择特定的终端创建共享变量，可选择共享变量所服务的终端。

当某个RT终端使用网络发布共享变量时，该终端控制数据，其他终端不影响共享变量的可用性。如可用性不是考虑因素，可在主机上使用共享变量。这样可释放终端上的处理器资源，使不支持共享变量引擎的终端也可访问共享变量，并使用户可使用LabVIEW DSC模块的安全功能。

下图显示了Real-Time模块的应用，这些应用通过单进程或网络发布共享变量在本地VI之间或联网VI之间共享数据：



RT终端上的确定性VI与终端上的其他VI通过单进程共享变量B共享数据。确定性VI还通过网络发布的共享变量A与主机上的VI共享数据。A启用了RT FIFO。RT终端上的网络发布共享变量将数据发送到终端上的共享变量引擎，然后与主机共享数据。

注： 必须在RT终端上安装Network Variable Engine和Variable Client Support，才能在终端上使用网络发布共享变量。RT终端需要至少32 MB内存才能运行共享变量引擎。但是，终端上内存少于32 MB时，可在终端上安装Variable Client Support，然后使终端引用其他主机上的网络发布共享变量。

创建共享变量后，可使用共享变量属性对话框修改变量的配置。在**项目浏览器**窗口右键单击共享变量，从快捷菜单中选择**属性**，可打开该对话框。修改共享变量的属性时，对变量的改动在部署后生效。

启用Real-Time FIFO

在共享变量属性对话框的实时FIFO页上，可启用共享变量的实时FIFO功能。勾选**启用实时FIFO**复选框，使用单元素或多元素FIFO确定性共享数据。

单元素FIFO

单元素FIFO共享最新的数据值。共享变量接收到新值时会覆盖原来的数据值。只在需要最新值时使用该选项。如要使用数组或波形数据类型，为FIFO缓冲区配置数组元素的大小和波形的大小。

多元素FIFO

多元素FIFO缓存共享变量共享的值。用户可将FIFO缓冲区的大小和元素配置为与网络页中**使用缓冲**部分相同，或者自定义FIFO的大小和元素。

注： 对于单元素FIFO和多元素FIFO，如变量包含数组或波形数据，则必须将FIFO的大小配置为要共享的数据相同的大小。如共享变量启用了网络缓冲和RT FIFO，网络缓冲区必须与一个FIFO元素同等大小。共享与指定大小不同的数据，会导致内存分配问题，影响确定性。

注： 波形中包含可变大小的元素，与实时FIFO不兼容。因此，在包含波形数据的共享变量上启用实时FIFO，波形数据的可变元素不会被传输。

相关概念：

- 使用定时循环创建确定性应用程序
- 远程通信方法
- 在RT终端上共享本地数据

确定性读取和写入共享变量值

创建共享变量后，可在VI中使用共享变量节点访问共享变量数据。共享变量节点是一个程序框图对象，用于引用**项目浏览器**窗口中相应的共享变量。共享变量节点用于从共享变量读取数据，以及将数据写入共享变量。将**项目浏览器**窗口中的共享变量拖放至相同项目中VI的程序框图，可创建一个共享变量节点。

也可在数据通信选板上选择共享变量节点，放置在程序框图上。要将共享代码节点

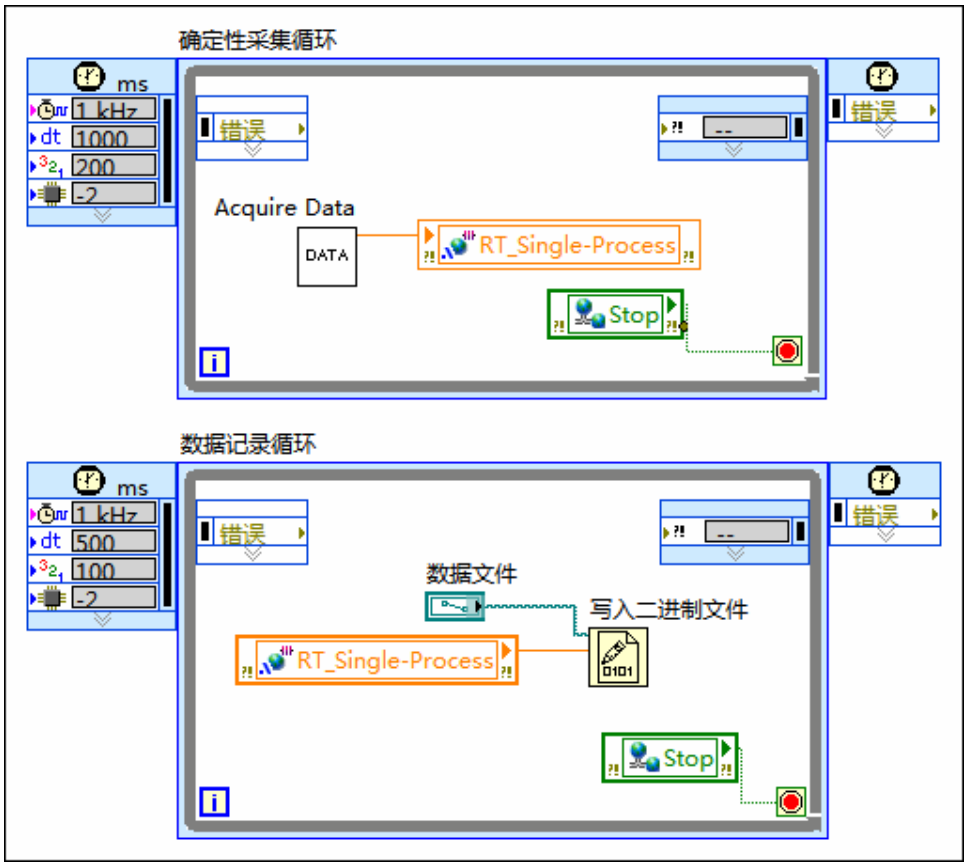
与项目中的共享变量绑定，右键单击共享变量节点，从快捷菜单中选择**选择变量»查找**，打开查找共享变量对话框。从树形目录中选择一个共享变量，然后单击**确定**按钮。

默认情况下，共享变量节点被设置为读取。要将程序框图中的共享变量节点的设置转换为写入，只要右键单击这个共享变量节点，从快捷菜单中选择**转换为写入**。

在RT终端上共享本地数据

使用启用了FIFO的单进程共享变量，在程序框图的两个无法连线的位置之间，或RT终端上的两个VI之间共享数据。启用共享变量的实时FIFO确保了程序框图或RT终端上运行VI的确定性部分的共享数据不会影响程序框图或VI部分的确定性。

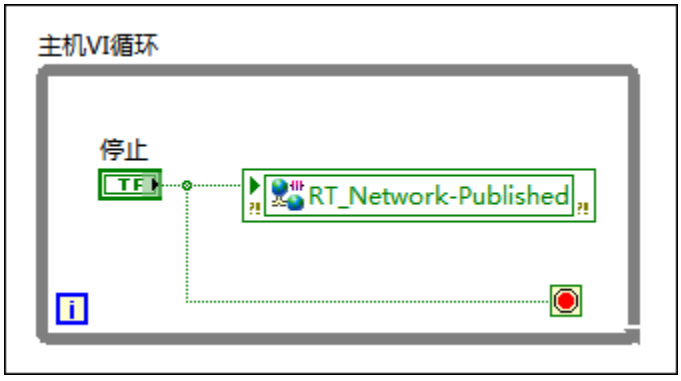
下列程序框图显示了一个RT终端VI，VI使用两个定时循环采集波形数据，然后将数据记录到文件。VI使用启用了实时FIFO的单进程共享变量**RT_Single-Process**在确定性定时循环（标签为“确定性采集循环”）和较低优先级的定时循环（标签为“数据采集循环”）之间共享数据。启用共享变量实时FIFO后，确定性采集循环可与数据记录循环共享数据，但是不会影响数据采集循环的确定性。



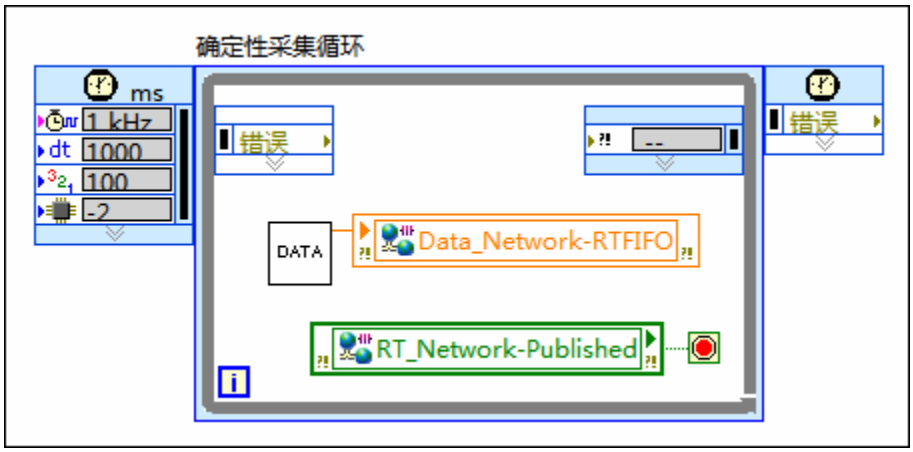
与远程终端共享数据

使用网络发布的共享变量，在启用实时FIFO后，可在VI和不同终端之间共享数据。启用共享变量实时FIFO后，可确保在网络上共享来自确定性VI的数据时不影响VI的确定性。

下列程序框图显示了一个主机VI，并发布布尔值停止RT终端VI。主机VI写入布尔值至启用了实时FIFO的网络发布共享的共享变量**RT_Network-Published**。



通过启用共享变量的实时FIFO，下列程序框图中的“确定性采集循环”可通过网络接收数据，不影响数据采集的确定性。也可通过启用了FIFO的网络发布共享变量共享来自确定性采集循环的数据。



确定性应用程序的时间控制

因为RT终端上的实时操作系统(RTOS)具有独占性，确定性任务可独占终端上的处理器资源。确定性任务可能会占用所有处理器资源，不允许应用程序中非确定性任务的执行。区分确定性任务和非确定性任务之后，可使用定时方法保证确定性任务确定执行，非确定性任务在有需要时执行。

可使用Real-Time定时VI，定时结构和外部定时源从时间上控制循环执行的速率。

注： 有些RT终端自动禁用毫秒定时引擎。详细信息，请参考RT终端的设备说明文档。

注： Real-Time模块8.0之前的版本中的毫秒定时引擎有0.07%的误差。如应用程序中包含0.07%误差的纠错代码，必须将代码移除或启用8.0之前版本的毫秒定时引擎。关于该错误的详细信息，见知识库。

使用Real-Time定时VI控制循环的定时

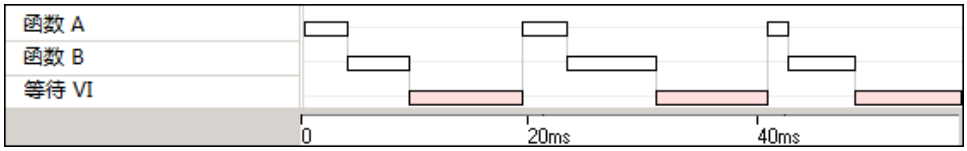
使用Real-Time定时VI、等待和等待下一个整数倍，控制RT终端上循环的定时。通

过Real-Time定时VI，可使用RT终端上RTOS毫秒或微秒计时器来控制循环的精度。

等待VI

等待VI使VI在指定时间长度内为休眠状态。例如，在运行于RT终端的VI中使用“等待”VI和设置为毫秒的**计数器单位**，提供1 kHz的循环速率。假设，等待VI执行时RTOS的毫秒计时器值为112 ms，**计数**输入为10，VI进入休眠状态，直到毫秒计时器的值等于122 ms。

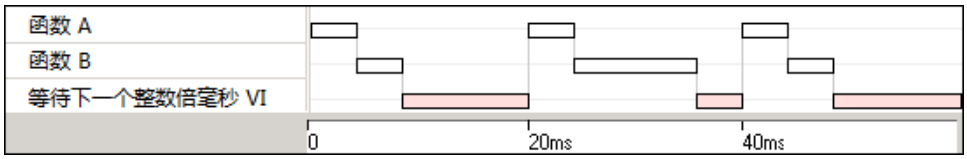
下图显示了使用定时VI的实时应用程序定时机制。应用程序执行函数A、函数B，在函数B执行完后，使用等待VI等待10毫秒。



等待下一个整数倍VI

等待下一个整数倍VI使一个线程进入休眠状态，直到RTOS的毫秒或微秒计时器到达**计数**输入端指定的整数倍值。例如，如等待下一个整数倍VI的**计数**输入端为10毫秒，RTOS毫秒计时器的值为112毫秒，VI将进入休眠状态直到毫秒计时器的值等于120毫秒。因为，120毫秒是在等待下一个整数倍VI执行后到达的第一个10毫秒的整数倍。

下图显示了使用等待下一个整数倍VI的实时应用程序定时机制。应用程序先执行函数A，再执行函数B，然后进入休眠状态，直到RTOS毫秒计时器的值等于20毫秒的整数倍。计时长度的值由等待下一个整数倍VI的**计时**输入端指定。

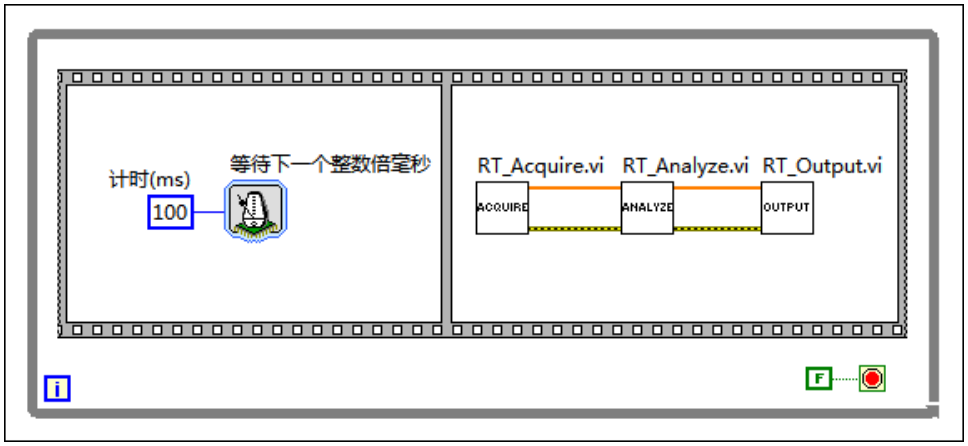


在上图中，等待下一个整数倍VI的执行时间由函数A和函数B的执行情况决定。当函数A和函数B执行完成后，等待下一个整数倍VI开始执行，直到操作系统计时器到达第一个20 ms的整数倍。

定义Real-Time定时VI的执行顺序

避免与其他LabVIEW代码并行使用等待VI或等待下一个整数倍VI，并行使用可能会导致预期之外的定时结果。在定时结构或实时优先级的VI中，只可使用一个线程。即使代码是并行的，定时VI和其他LabVIEW代码按先后顺序执行。在该情况下，LabVIEW编译VI时将并行代码顺序化，定时VI的执行顺序在各次编译中可能有所不同。如代码不在定时结构或实时VI中，LabVIEW将定时VI和其他LabVIEW代码并行执行，会改变定时VI的效果。

可使用顺序结构强制规定特定的执行顺序，以保证明确的时间性。在以下程序框图中，等待下一个整数倍VI使VI进入休眠模式，直到毫秒计时器达到100毫秒的整数倍。当VI从休眠模式中恢复，VI开始执行顺序结构的下一帧。



使用定时结构控制RT终端VI的定时

定时结构用于按指定的优先级执行在固定的时间内执行结构中的代码。定时结构的优先级设置是指定一个定时结构相对于终端上的其他定时结构的优先级。

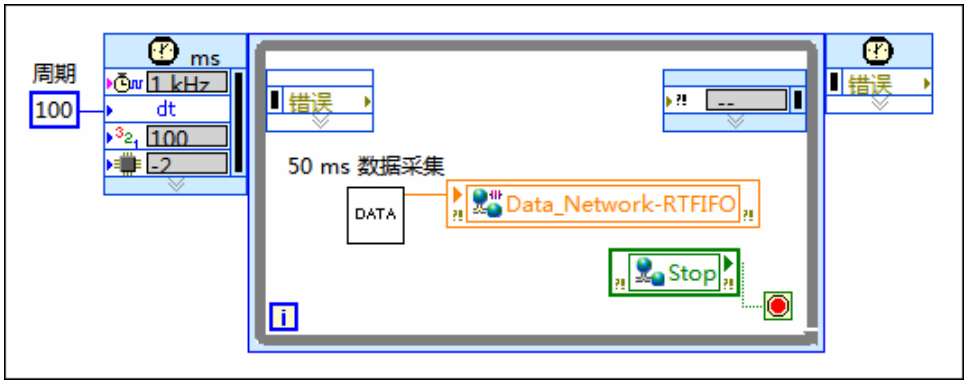
警告 LabVIEW使用了两种既相对独立又相互关联的优先级机制，VI优先级和结构优先级。定时结构优先级是数值，数值越大，表示相对于终端上的其他定时结构优先级越高。但是，所有定时结构优先级都在高和实时VI优先级上。建议在一个应用程序中只使用一种优先级机制，避免非预期结果。如应用程序使用定时结构，将所有VI设为普通优先级。

定时循环

定时循环根据指定的循环周期执行一个子程序框图或帧。在以下情况中可以使用定时循环结构：例如，开发支持多速率定时功能的VI、循环执行时返回值、动态改变定时功能或者多种执行优先级。

使用配置定时循环对话框配置定时源、周期、优先级和其他定时循环执行的高级选项。

定时循环具有独占性，可占用所有处理器资源。定时循环可能会使用全部处理器资源，不允许程序框图上的其他任务执行。必须将最高优先级的定时循环的周期配置为足够大，每次循环，进行确定性任务后有足够的空闲时间保证其他优先级较低的任务执行。下列定时循环包含的子VI每次数据采集耗时50毫秒。



定时循环的周期为100毫秒，每次循环均留出50毫秒的空闲时间。定时循环空闲时，LabVIEW将执行程序框图上较低优先级的任务。

定时顺序

定时顺序结构由一个或多个任务子程序框图或帧组成，是根据外部或内部信号时间源定时后顺序执行的结构。定时顺序适于开发精确定时、执行反馈、定时特征等动态改变或有多层执行优先级的VI。

使用配置定时顺序对话框配置定时源、优先级和其他定时顺序结构执行的高级选项。

多帧定时循环

可向定时循环添加帧，按顺序执行各个帧。多帧定时循环相当一个带有嵌入顺序结构的定时循环。

使用外部定时源设置确定性应用程序的定时

运行在RT终端上的NI驱动程序支持使当前LabVIEW线程进入休眠状态的VI和函数，驱动检测到特定事件时，线程从休眠状态中返回。例如，可使用NI-DAQmx和NI数据采集硬件对实时应用程序进行定时。关于休眠和等待驱动事件的VI和函数的详细信息，请参考特定的NI驱动程序文档。

使用NI-DAQmx对实时应用程序进行定时

NI数据采集硬件与NI-DAQmx配合使用可将循环速率设置为与硬件时钟速率相同。通过NI-DAQmx，可使用下列方法对实时应用程序进行定时：

- **硬件定时单点**－NI-DAQmx支持硬件定时单点采样模式。在该模式下，使用硬件定时采集和生成样本，没有缓冲机制。可使用硬件定时单点模式来控制确定性时间内输入输出的应用程序。关于使用硬件定时单点操作对确定性应用程序进行定时的信息，请参考NI-DAQmx帮助下的NI-DAQmx单点实时应用程序。
- **计数器定时器**－NI-DAQmx支持使用硬件定时计数器输入来驱动一个控制循环。使用“等待下一个采样时钟”VI同步计数器和计数器上的采样时钟。关于使用计数器输入进行确定性应用程序定时的详细信息，请参考NI-DAQmx帮助的硬件定时计数器任务。
- **用于定时结构的DAQmx定时源**－定时结构可通过硬件定时，是多速率应用的理想选择。默认情况下，在Windows操作系统上或安装了实时操作系统的RT终端上，定时循环使用1 kHz时钟。借助NI-DAQmx，也可使用DAQ设备的外部信号作为定时结构的定时源。使用“DAQmx创建定时源”VI，创建一个定时源，同步定时结构和硬件时钟。关于使用DAQ设备上的外部信号控制定时源的详细信息，请参考NI-DAQmx帮助中的使用定时循环的硬件定时同步更新I/O。

关于使用NI数据采集硬件和NI-DAQmx的定时控制循环的详细信息，请参考NI-

DAQmx帮助。选择**开始»所有程序»National Instruments»NI-DAQ»NI-DAQmx帮助**，可打开NI-DAQmx帮助。

相关概念：

- 使用定时循环创建确定性应用程序
- 使用优先级不同的VI创建确定性应用程序
- 使用Real-Time模块创建确定性应用程序

创建独立的实时应用程序

LabVIEW应用程序生成器用于生成独立的实时应用程序，应用程序可在RT终端上电后自动运行。根据应用程序的需要，可从“项目浏览器”窗口或LabVIEW开发环境之外部署独立实时应用程序。

注： 如未安装应用程序生成器，无法创建独立应用程序，也可将某个VI设置为RT终端的启动VI。关于将VI设置为RT终端启动VI的信息，请参考NI网站。

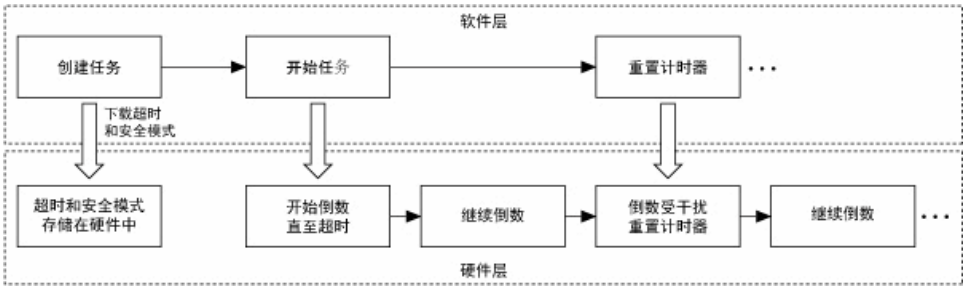
使用看门狗硬件从嵌入式软件故障中恢复

大多数嵌入式实时系统必须是独立的。嵌入式系统发生软件故障时，等待操作员重启系统不符合实际情况。因此，嵌入式系统必须能检测到故障状态，并执行相应的恢复程序。要检测软件故障并从软件故障中恢复，大多数嵌入式系统都包含一个嵌入式的硬件定时器，也称看门狗定时器或失知制动装置。看门狗定时器用于保证嵌入式软件进程正常执行，以及软件无响应时启动一个恢复程序。

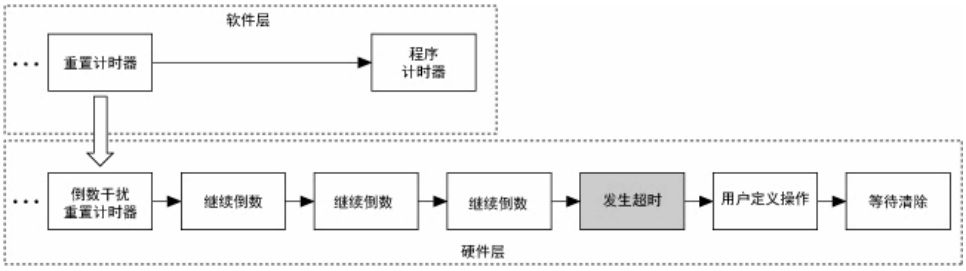
硬软件接口

看门狗定时器是一个计数器，与嵌入式应用程序交互，检测是否发生软件故障，以及从故障中恢复。在正常操作时，应用程序将初始化看门狗定时器，从一个特定的数以固定间隔倒数计数，应用程序还定义定时器到达零时的操作。应用程序启动看门狗定时器后，将周期性地重置定时器，以确保定时器不会到达零值，如下列示意

图所示：



如发生软件故障，应用程序无法重置定时器，将会发生超时。因为看门狗定时器独立于应用程序，会一直倒数直到零。看门狗定时器过期后，硬件触发恢复程序，如下列示意图所示：



RT系列PXI、CompactRIO和CompactRIO FieldPoint控制器中都包含内置的看门狗定时器硬件，可通过RT看门狗VI访问。使用看门狗配置VI，设置看门狗定时器的超时时间，并指定超时发生时的操作。使用Watchdog Whack VI，在超时发生前周期性地重置定时器。

注： Real-Time看门狗VI依赖于RT系列PXI、CompactRIO和Compact FieldPoint控制器的硬件特性。不建议将Real-Time看门狗VI与第三方看门狗硬件配合使用。

选择合适的超时设置

超时时间的合理范围取决于应用程序具体的执行速度特性以及时间要求。超时的时间必须大于可接受的系统抖动产生的时间差。超时的时间又必须小于应用程序启动故障恢复程序的等待时间。

使用高级看门狗VI

高层的看门狗VI－Watchdog Configure、Watchdog Whack和Watchdog Clear是基于低层的高级看门狗VI创建的。如使用高层看门狗VI无法满足应用程序的要求，可使用高级看门狗VI进行自定义设置。

创建多看门狗对象

可使用看门狗配置VI或高级看门狗VI，创建具有不同属性的多看门狗对象。多看门狗对象用于具有不同时间属性的独立状态的应用程序。例如，设计一个具有状态A和状态B的状态机，状态A的超时时间为5秒，状态B的超时时间为10秒。该情况下，可使用多看门狗对象。

但是，RT终端通常只有一个硬件看门狗定时器，所以一次只能使用一个看门狗对象。使用看门狗对象后，必须使用Watchdog Clear VI或高级看门狗VI，关闭当前看门狗对象后才能使用其他看门狗对象。

调试确定性应用程序

调试确定性应用程序需使用LabVIEW调试的一般技巧。对于确定性应用程序，调试包含两个方面的内容，应用程序的执行情况以及所用时间。

相关概念：

- 验证应用程序运行正常
- 验证时间确定性

验证应用程序运行正常

要验证一个RT应用程序，可先使用LabVIEW调试工具检测程序是否有错误，单步调试找到错误所在位置。然后，使用记录性能和内存信息窗口或Real-Time Trace Viewer检测应用程序的执行时间和内存使用。

LabVIEW调试工具

主机连接至RT终端时，使用LabVIEW调试工具（高亮显示执行过程、单步调试）步过调试LabVIEW代码。

注： 不要使用LabVIEW调试工具调试程序的执行时间，调试工具会影响应用程序的执行时间。

Real-Time模块唯一不支持的功能是**调用链**。单步调试时，该选项会出现在子VI程序框图的工具栏上。

注： 必须在VI属性对话框的**执行**页勾选**允许调试**复选框，才能使用LabVIEW调试工具调试VI。

性能和内存信息窗口

“性能和内存信息”窗口是对应用程序执行时间和内存使用进行静态分析的有力工具。**性能和内存信息**窗口可显示内存中所有VI和子VI的性能信息。这些信息有助于找到潜在的问题，从而优化VI的性能。选择**工具»性能分析»性能和内存**，显示**记录性能和内存信息**窗口。

记录性能和内存信息之前，必须先勾选**记录内存使用**复选框。收集内存使用信息将极大地增加VI执行时系统的开销，会影响记录过程中用时统计的准确性。因此，分别执行内存检测与时间检测可返回更为精确的信息。

在记录信息的过程中，可记录某一个时刻的数据，然后将数据保存至ASCII数字表格文件。每次运行VI时，时间测量均累积。

注： **性能和内存信息**窗口的许多选项在开始记录后才可用。

NI分布式系统管理器

NI分布式系统管理器显示RT终端上运行的VI的详细信息，并动态显示终端内存和

CPU资源。可通过NI分布式系统管理器停止VI和启用VI。在LabVIEW中，选择**工具**»**分布式系统管理器**，可打开NI分布式系统管理器。

调试应用程序或共享库对话框

使用调试应用程序或共享库对话框调试在RT终端上运行的独立的实时应用程序。

> **注：** 生成用于调试的独立实时应用程序之前，必须在Real-Time应用程序属性的高级页勾选**启用调试**复选框。

相关概念：

- 调试确定性应用程序

验证时间确定性

确定性应用程序的一个重要特征是执行时间的确定性。使用下列方法验证应用程序的时间确定性。

Real-Time比较VI

使用“RT获取时间标识”VI和“RT时间标识分析”VI比较RT终端上运行VI以及VI中部分代码的执行速度。可使用比较信息来优化RT终端VI的设计。

使用“RT获取时间标识”VI可返回高精度定时源的64位时间标识值。使用“RT时间标识分析”VI分析“RT获取时间标识”VI返回的时间标识值。

Real-Time Trace Viewer

Real-Time Trace Viewer是实时事件和执行跟踪工具，用于捕捉和显示VI时间和事件数据以及LabVIEW Real-Time模块应用程序的线程事件。Real-Time Trace Viewer包括Real-Time Trace Viewer VI。Real-Time Trace Viewer VI可用于获取RT终端上VI的定时和执行数据，以及应用程序的线程事件。Real-Time Trace Viewer VI用于显示主机上的时间和事件数据，供分析使用。

监测实时终端资源

可使用NI分布式系统管理器，监测RT终端的资源使用和RT终端VI的细节信息。在一些情况下，RT终端上内存或CPU资源不足都有可能导致应用程序时间故障。

> **注：** 如RT终端连接至显示器，可使用CPU Load Measurement工具监测终端上CPU的使用情况。

相关概念：

- 调试确定性应用程序

使用和定义错误代码

使用错误处理机制调试和管理VI中出现的错误。LabVIEW错误处理器VI能在VI发生错误时返回错误代码。错误代码表示VI中错误的特定类型。配置RT终端时，LabVIEW会自动将错误处理器VI的错误代码文件复制到终端上。

也可在RT终端的错误处理VI中使用自定义错误代码。使用错误代码编辑器对话框创建错误文件。选择**工具»高级»编辑错误代码**，打开**错误代码文件编辑器**。如在RT终端上使用自定义错误代码，必须将文件扩展名重命名为`*.err`，然后将错误文件放在终端的`c:\ni-rt\system\user.lib\errors`目录或`c:\ni-rt\system\errors`目录下。(NI Linux Real-Time)将错误文件放置在`/usr/local/natinst/labview/errors`目录下。

优化确定性应用程序

本章的编程技巧有助于减少运行在RT终端上的应用程序执行时的抖动。

- 避免资源共享
- 避免连续内存冲突
- 避免子VI开销
- 设置VI属性

- 批量编译VI
- 最小化RT终端Web服务器使用的内存
- 优化多CPU系统上的实时应用程序

相关概念：

- 避免资源共享
- 避免连续内存冲突
- 避免子VI开销
- 设置VI属性
- 批量编译VI
- 最小化RT终端Web服务器使用的内存
- 优化多CPU系统上的实时应用程序

避免资源共享

在LabVIEW中，部分资源可能由两个或两个以上VI共享。资源共享可能会造成程序执行时间的抖动，并导致应用程序无法利用多CPU的优点。常见的共享资源包括：

- 全局变量
- 非重入子VI
- LabVIEW内存管理器
- 队列操作函数
- 信号量VI
- 单线程DLL

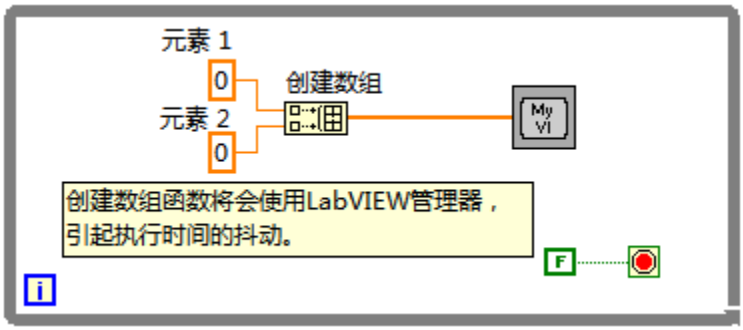
如VI使用共享资源，VI将使用资源的操作系统互斥量，防止其他VI访问资源。互斥量独占性锁定某个资源，防止其他VI访问该资源。如实时优先级VI试图抢占了一个较低优先级VI的锁定资源，因为资源被锁定，没有可用的共享资源，实时VI必须等待。在该情况下，低优先级VI必须完成操作，释放共享资源，实时VI才能继续执行，使得低优先级VI比实时VI似乎更为重要。这种情况被称为优先级倒置，会影响VI执行的确定性。

内存分配和预分配数组

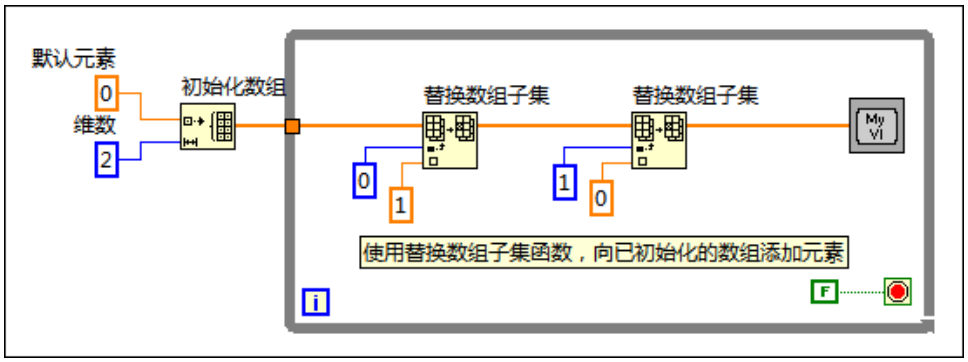
VI分配内存时，VI将访问LabVIEW内存管理器。LabVIEW内存管理器分配用于数据存储的内存。LabVIEW内存管理器是共享资源，可由互斥量锁定长达若干毫秒。在确定性VI中分配内存会影响VI的确定性。

如在确定性VI的控制循环中使用数组，进入循环之前预分配数组有助于减少抖动。

下列程序框图在循环中创建数组。每次循环时，循环中的“创建数组”函数都使用LabVIEW内存管理器为数组分配内存，会影响循环的确定性。



以下程序框图在循环外使用初始化数组函数，在循环中使用替换数组子集函数，从而创建数组。因为在循环外预分配数组的内存，循环不必在每次循环中访问LabVIEW内存管理器。



转换合适的数据类型

将RT终端VI中的数据转换为合适的数据类型。LabVIEW进行数据转换时会在内存的缓冲区中创建一个数据副本，在数据转换后保存新的数据类型。LabVIEW内存管理器必须为副本分配内存，这会影响确定性VI的确定性。创建数据缓存副本会占用RT

终端上的内存资源。

转换数据类型时，应尽可能使用最小的数据类型。如要转换数组的数据类型，在创建数组之前转换。另外，只有输入和输出的数据类型一致时，函数输出才会使用输入的缓存。数组的结构和元素数量必须相等，函数输出才会复用输入缓冲。

减少使用全局变量

LabVIEW会为VI中使用的每个全局变量创建副本。减少使用全局变量，提高VI的效率和性能。创建全局变量的副本会占用RT终端上的内存资源。

相关概念：

- 确定何时使用多CPU
- 优化确定性应用程序
- 优化多CPU系统上的实时应用程序
- 在多CPU实时应用程序中使用并行操作

避免连续内存冲突

LabVIEW处理许多内存的细节操作，方式类似于传统的文本语言。例如，向数组或字符串添加新信息时，LabVIEW将分配新的内存空间来容纳新的数据或字符串。不再需要数据时，LabVIEW取消分配相关的内存。使用LabVIEW时需考虑下列内存因素：

- 分配内存会造成抖动。
- RT终端内存可能不足。
- RT终端连续内存可能不足。连续内存是一组分配给进程的连续内存地址。(NI Linux Real-Time) NI Linux Real-Time终端含有内存管理单元，因而较少发生连续内存不足的情况。

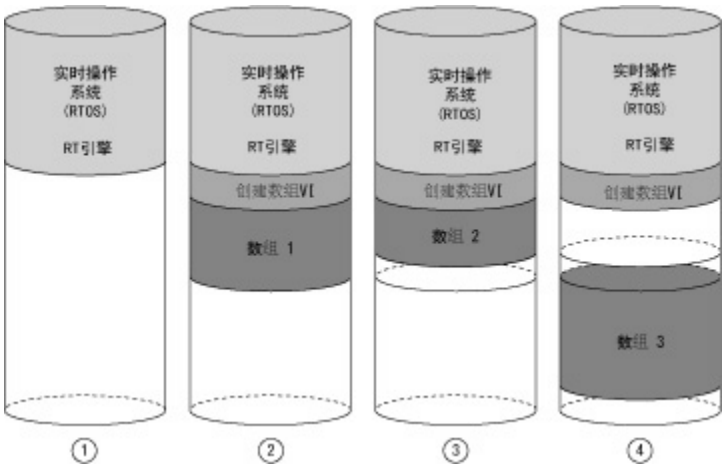
设计考虑内存的VI

NI为设计考虑内存的VI用于RT终端时提供下列建议：

- 始终预分配数组空间，数组的大小为可能使用的最大容量。
- 定义用于RT终端的LabVIEW类时，不要定义需要大量内存的默认值，例如，较大的数组或字符串。在RT终端上使用LabVIEW类时，“请求释放内存”函数在某些情况下实际上是分配内存。如类的默认值包含较大的大小可变的数据结构，例如，数组、字符串等，应用程序将设置类实例为较小的数组或字符串，“请求释放内存”函数分配内存，因为函数将把数据设置为数据类型的默认值。

连续内存冲突范例

下图演示了应用程序的最大数组未预先分配足够空间时将如何引起连续内存冲突。



①

重启或重置RT终端时，RTOS和RT引擎加载至内存。RT引擎使用可用的内存运行RT终端VI和存储数据。

②

`ArrayMaker.vi`创建数组1。数组1中的所有元素在内存中是连续的。RTOS预留的连续内存空间相当于数组1使用的内存。

③

如停止VI后再次运行，数组与上次运行的数组大小相同或较小时，实时操作系统使用相同的内存地址。在图3中，`ArrayMaker.vi`创建数组2。RTOS在数组1预留的

内存空间中创建数组2。分配给数组1的预留内存空间足够容纳数组2，因此不会导致连续内存冲突。数组1的多余连续内存仍位于预留的内存空间内。

④

如ArrayMaker.vi第三次运行时使用较大的数组，或其他VI生成较大的数组，RT引擎必须找到一个足够大的连续内存空间。在图4中，ArrayMaker.vi必须在可用的内存中创建比之前数组都大的数组3。即使ArrayMaker.vi停止运行，RT引擎仍继续运行，之前预留的内存不可用。如ArrayMaker.vi第四次运行并尝试创建比数组3更大的数组时，操作将失败。没有足够的连续内存空间来创建数组。

为避免在图4中发生连续内存冲突，可预分配最大可能用到的数组存储空间。

相关概念：

- 优化确定性应用程序

避免子VI开销

从RT终端VI调用一个子VI会对整个应用程序增加小量的开销。虽然单次调用的开销较小，但是，在循环中多次调用子VI仍会显著增加系统开销。可将循环嵌入子VI，减少系统开销。

可修改VI优先级，将子VI转换为子程序优先级。LabVIEW调用子程序时开销最小。子程序是需重复执行的小段代码，无需用户交互。子程序无法显示前面板数据，不与其他VI多任务执行。

相关概念：

- 使用优先级不同的VI创建确定性应用程序
- 优化确定性应用程序

设置VI属性

要降低内存要求，提高VI的性能，可在“VI属性”对话框中禁用非必要选项。在“项目

浏览器”窗口右键单击VI，从快捷菜单中选择**属性**，可打开“VI属性”对话框。在**类别**下拉菜单中选择**执行**，取消勾选**允许调试**和**运行时自动处理菜单**复选框。禁用上述选项后，VI占用的内存更少，编译速度更快，引发的抖动也更小。

相关概念：

- 优化确定性应用程序

批量编译VI

批量编译VI是提高执行速度的方法之一。批量编译VI就是编译顶层VI以及重新编译和链接程序框图上的所有子VI和函数。选择**工具»高级»批量编译**，选择要批量编译的VI。

相关概念：

- 优化确定性应用程序

最小化RT终端Web服务器使用的内存

如要尽量减少启用RT终端上Web服务器时的内存占用，在“RT终端属性”对话框的**Web服务器:可见VI**页指定只包括需远程访问的VI。在“项目浏览器”窗口右键单击RT终端，并从快捷菜单中选择**属性**，打开**RT终端属性**对话框。默认情况下，**可见VI**列表框中显示*，表示所有VI都可见。移除*，仅添加在Web服务器上需可见的VI，可节省内存。

相关概念：

- 优化确定性应用程序

优化多CPU系统上的实时应用程序

借助LabVIEW Real-Time模块，用户可充分利用多CPU系统并行处理的优势。多CPU系统又称多核、多处理器系统（SMP系统）。

Real-Time模块包含CPU排序器，用于向可用的CPU分配线程。可使用CPU排序器实现多CPU系统上的并行执行。许多RT应用无需重新编码代码，即可使用多CPU带来的各种好处。但是，如根据多CPU系统重新修改RT应用程序的代码，程序的运行速度会有明显改善。要实现多CPU RT系统的所有优点，必须在RT终端VI中使用并行或流水线设计结构。

注： 对于某些应用程序，在未安装NI RT Extensions for SMP的单CPU系统上，时间延迟更少。

编写一个在多CPU系统运行的VI时，需避免共享资源冲突。多CPU系统的优点在于单独的线程可同时在不同的CPU上执行。同时请求使用共享资源将加快并行执行，削弱多CPU系统的性能优势。

Real-Time模块在所有可用的CPU之间平衡分配线程。但是，有些应用程序可能通过使用定时结构手动分配CPU和CPU池，实现最优性能。

注：

- 打开一个在LabVIEW 8.5或更早版本中创建的VI，所有定时循环自动在默认CPU上运行。在该情况下，可通过手动CPU分配和CPU池，发挥多CPU的优点。
- 关于优化多CPU实时系统性能的详细信息，请参考NI网站。

相关概念：

- 使用定时循环创建确定性应用程序
- 使用优先级不同的VI创建确定性应用程序
- 使用Real-Time模块创建确定性应用程序
- 优化确定性应用程序
- 确定何时使用多CPU
- 在多CPU实时应用程序中使用并行操作
- 避免资源共享

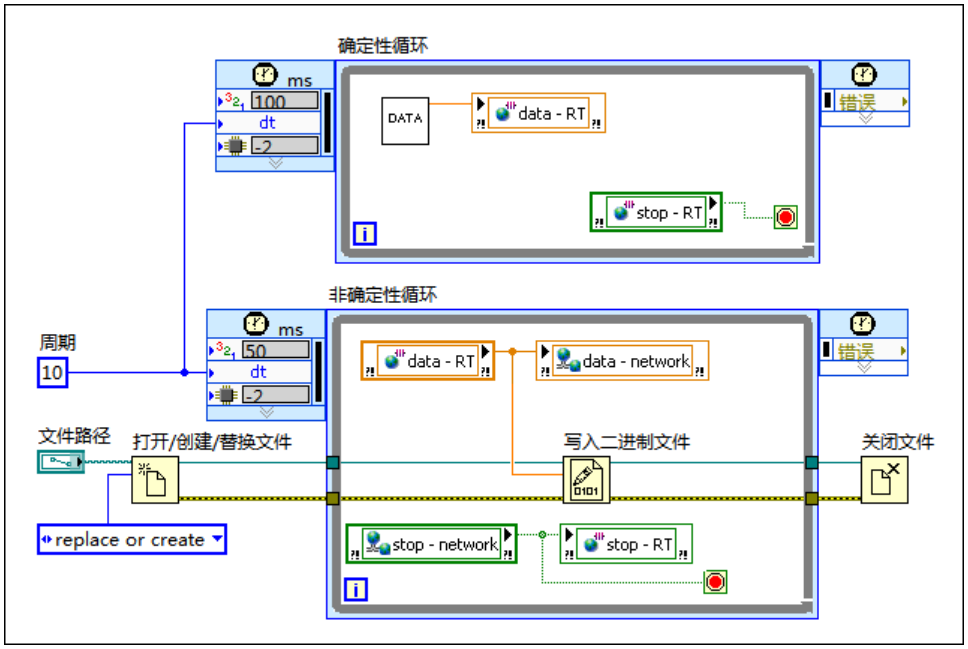
确定何时使用多CPU

安装NI RT Extensions for SMP后，Real-Time模块在可用的CPU之间平衡分配线程，这个过程叫做自动负载平衡。自动负载平衡使具有并行机制的RT终端VI能充分使用多CPU系统的各种优点。多CPU系统也叫多核、多处理器或SMP系统。但是，在CPU之间传输数据的速度比在一个CPU上前后两个操作之间传输数据慢。另外，CPU排序器的开销会对RT系统的延迟稍有影响，减慢单点I/O的速度，单点I/O不受益于并行机制。总而言之，只有当并行处理机制节省的时间超过在CPU之间传输数据消耗在分配系统资源上的时间时，应用程序才能从多CPU系统中受益。

注： 如不确定是否使用NI RT Extensions for SMP运行应用程序，可在两种情况下分别测试应用程序的运行。使用NI Measurement & Automation Explorer (MAX)中的LabVIEW Real-Time软件向导安装和卸载用于SMP的NI RT Extensions。关于使用LabVIEW Real-Time软件向导的信息，请参考MAX帮助。(NI Linux Real-Time)在NI Linux Real-Time终端上，不能卸载NI RT Extensions for SMP。

受益于多CPU系统的应用程序

实时应用程序通常包括与实时任务并行运行的数据记录或通信任务。例如，下列程序框图显示的RT终端VI包含两个并行循环。因为并行结构，在多CPU RT系统上运行该应用程序可提高运行速度，无需重写VI。



要求多个实时进行的应用程序也可受益于多CPU系统。因为实时任务具有独占性，单CPU系统上一般只包含一个实时循环。但是，在多CPU系统上，可在每个CPU上包含一个实时循环。

不受益于多CPU系统的应用程序

虽然大多数实时应用程序转移到多CPU系统上后，运行速度会有明显提高，但是，还是有部分VI必须重写才能受益于多CPU系统的优点，有些VI甚至不受益于多CPU系统。多CPU系统对单个顺序循环的实时应用的优点不明显。如循环中包含密集处理的部分，使用流水线结构，可显著提高运行速度。多CPU系统对主要由单点I/O组成的应用程序的执行速度改进不大，因为CPU排序器会小幅增加时间延迟。

包含共享资源的应用程序必须重写代码以避免资源冲突，才能受益于多CPU系统的优势。同时请求使用共享资源将加快并行执行，削弱多CPU系统的性能优势。

注： 关于何时使用NI RT Extensions for SMP的详细信息，请参考NI网站。

相关概念：

- 避免资源共享

- 优化多CPU系统上的实时应用程序

在多CPU实时应用程序中使用并行操作

LabVIEW的多线程图形化编程环境能利用多CPU系统并行执行的各种优势。多CPU系统也叫多核、多处理器或SMP系统。要在LabVIEW中实现并行结构，在程序框图上放置两个或两个以上对象，但是不连接这些对象即可。如VI在一个包含N个CPU的系统中执行，该系统可并行执行N条线程。可将代码分别放在各个并行循环中，或利用并行数据处理的优势，将多个输入通道分配给各自的CPU处理。

注： 过度使用CPU会在CPU之间频繁传输数据，延长时滞。设计并行结构时还必须避免因为共享资源引起的延时。同时请求使用共享资源将加快并行执行，削弱多CPU系统的性能优势。可使用Real-Time Trace Viewer确定并行操作是否确实为并行执行。

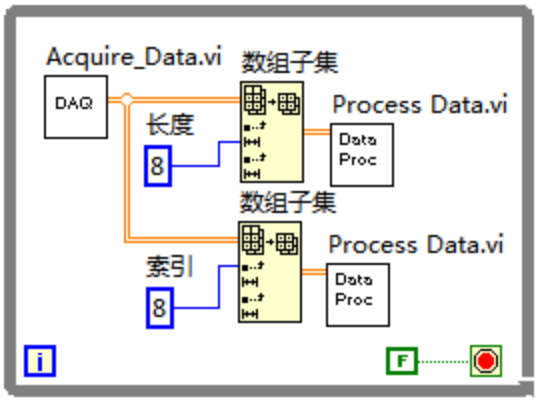
并行循环

如在多CPU系统上运行的RT应用程序中包括并行循环，CPU排序器会进行自动负载平衡，在CPU之间合理分配执行线程。也可使用定时循环的**已分配CPU**输入端，将定时循环手动分配至指定的CPU。

注： 打开一个在LabVIEW 8.5或更早版本中创建的VI，所有定时循环自动在默认CPU上运行。在该情况下，可通过手动CPU分配和CPU池，发挥多CPU的优点。

并行数据处理

可将多个数据通道分为若干组，然后在单独的CPU上并行处理这些通道组。例如，如要在双核系统上处理16路输入通道，可将通道分为两组，每组8条通道，如下列程序框图所示：



在该范例中，Process Data.vi被配置为重入子VI，因此子VI的每个实例接收到一个专用的内存空间，且两个实例无需在同一CPU上执行。Process Data.vi的两个实例之间没有数据流依赖关系，CPU排序器在可用的CPU之间自动平衡两个实例。

注： LabVIEW将定时循环作为一个单独的线程。要利用定时循环并行处理的优点，可将并行处理的代码放在单独的定时循环中。关于在多CPU实时系统中使用并行机制的详细信息，请参考NI网站。

相关概念：

- 优化多CPU系统上的实时应用程序
- 避免资源共享

通过流水线优化多CPU实时应用程序

在多CPU系统上，流水线处理可用于提高顺序执行的实时应用程序的效率。多CPU系统又称多核、多处理器或SMP系统。

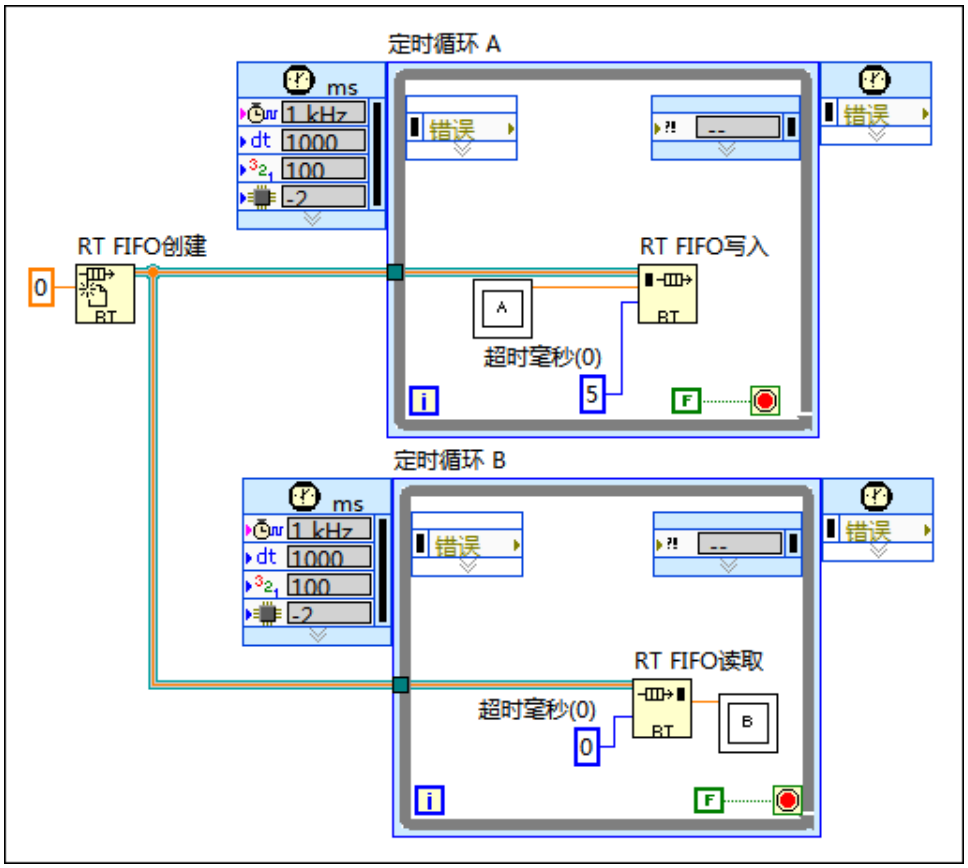
使用While循环的流水线

实现流水线最简便的方法是使用While循环，通过移位寄存器传递数据。

使用定时循环的流水线

如要使用优先级或手动CPU分配等功能，需使用定时循环。LabVIEW将每个定时循

环映射到一个线程，故不可在定时循环中使用移位寄存器来实施流水线。如要通过定时循环实现一个流水线，必须将每个流水线步骤放置在一个并行的定时循环中，并通过队列、局部变量、全局变量或RT FIFO在定时循环中传递数据。下列程序框图显示使用RT FIFO实现一个流水线。



选择数据传输方法

根据数据传输方式的不同，流水线数据流的工作方式也有所差异。根据应用程序是否需要流水线输入与流水线输出一一对应，来选择数据传输的方式。

- **RT FIFO**－为保证流水线输入与流水线输出一一对应，必须等待各个数据元素到位后才执行下一个步骤。上例创建了一个RT FIFO，使用RT FIFO读取函数的**超时毫秒**输入端保证子VI B在接收到子VI A的数据后才开始执行。
- **局部/全局变量**－如应用程序不要求流水线输入与流水线输出一一对应，可使用局部/全局变量、超时为0的单元素RT FIFO来传输数据。

关于各种数据传输方法优劣的详细信息，见“在确定性应用中共享数据”手册中的主题。

自动加载平衡

使用定时循环实现流水线时，CPU排序器进行自动负载平衡，在多个CPU之间合理分配线程。也可使用定时循环的**已分配CPU**输入端，将定时循环手动分配至指定的CPU，从而提高执行控制。

提示 可使用Real-Time Trace Viewer诊断利用率低的CPU。

优化多CPU RT系统的其他资源

关于优化多CPU RT系统性能的详细信息，请访问`ni.com/info`并输入`RTSMP`，查看多核编程资源。

相关概念：

- 确定性应用程序中的数据共享

指定用于自动负载平衡的CPU集

在多CPU的RT终端上安装用于对称多处理(SMP)的NI RT Extensions，可增加在CPU中平均分配应用程序线程的自动加载平衡特性。使用MAX中的LabVIEW Real-Time软件向导安装用于SMP的NI RT Extensions。关于使用LabVIEW Real-Time软件向导的信息，请参考MAX帮助。

注： (NI Linux Real-Time)在NI Linux Real-Time终端上，不能卸载NI RT Extensions for SMP。

使用RT SMP CPU工具VI，指定自动加载平衡时可用的CPU组，以及为定时结构预留专用CPU。

CPU池

安装NI RT Extensions for SMP后，RTOS将会为自动加载平衡功能创建两个池：定时结构池和系统池。定时结构池中的CPU可用于定时循环和定时顺序结构自动处理器分配时的自动加载平衡。系统池中的CPU可用于所有其他进程，包括底层的操作系统任务。

自动加载平衡

自动加载平衡是将线程合理分配至CPU，使各个CPU负载相对平衡。启动时，LabVIEW Real-Time模块将系统池和定时结构池中的所有CPU，然后平衡各个CPU的负载。用户可使用SMP CPU工具VI制定系统和定时结构池。

除手动分配的定时结构线程之外，LabVIEW Real-Time对所有线程进行自动加载平衡。LabVIEW将每个定时结构的代码执行与专有线程一一对应，但是只在配置为自动CPU分配的定时结构线程上进行自动加载平衡。将CPU索引连接至定时结构的**处理器**输入端，手动分配一个定时结构至某一个CPU。**处理器**的默认值是-2，通过自动加载平衡，自动分配定时结构至处理器。也可使用配置定时循环对话框或配置定时顺序的**处理器分配**部分，为定时结构手动分配处理器。

一般情况下，自动加载平衡进程避免了在两个CPU之间的特定线程交换，从而减少了CPU内的数据传输开销。如各个CPU上同等优先级或优先级较高线程的负载变化较大，SMP排序器可能会反复将某线程从一个CPU转移到另一个CPU。可使用Real-Time Trace Viewer来确定定时结构线程是否从一个CPU转移至另一个CPU。

为了保证SMP排序器不意外地将定时结构分配至一个特定的CPU，可使用SMP CPU工具VI将CPU从定时结构池中移出。为了保证定时结构能独占地访问手动分配的CPU，还必须将CPU从系统池中移出。这样，即可最优化一个确定性循环的性能，如隔离定时结构所述。

池的配置

在启用了SMP的RT终端上，CPU有四种状态。CPU可能在：

- 定时结构池。在该池中，SMP排序器可将CPU配置为执行自动处理器分配的定时结构。
- 系统池。在该池中，SMP排序器可将CPU配置为执行不响应定时结构的线程。
- 同时位于两个池。SMP排序器可将CPU用于上述两种情况。
- 不属于任何池（预留）。在该情况下，CPU只用于手动分配给该CPU的定时结构。

启用了SMP的RT终端启动后，默认情况下，LabVIEW将所有CPU分配给两个池。用户可使用SMP CPU工具VI将CPU分配为任意一个上述四种状态。

注： CPU可以由两个池共享，但是每个池必须包含至少一个CPU。

隔离确定性定时结构

要在确定性定时循环中最优化性能和最小化抖动，可在特定CPU上隔离该定时结构。要隔离一个定时结构，必须将定时结构分配给既不属于系统池，也不属于定时结构池的CPU。如CPU未分配至池，则CPU仅用于运行通过手动分配指定的定时结构。在一个CPU上隔离定时结构，可独占CPU的处理资源，从而实现更高的频率和输出率。例如，可使用隔离在数据采集的定时循环中实现更高的轮询率。

要最小化高速确定性定时循环的延迟，可考虑隔离索引较高的CPU上的定时循环。例如，如系统包含四个CPU核，索引为0-3，考虑在CPU3上隔离定时循环。对定时结构的执行顺序进行排序时，实时操作系统从索引最高的CPU开始，依次向低索引CPU执行。因此，如果同等优先级的多个定时结构同时被唤醒，在低索引CPU上执行的定时结构的唤醒延迟比高索引CPU上定时结构的延迟多若干毫秒。在非定时循环线程中，情况相反。因为这些线程从CPU 0开始依次向高索引CPU执行。为了减少延迟，建议将低索引CPU分配至系统池，将高优先级CPU用于确定性定时结构。

避免线程空闲

要避免系统线程空闲，可考虑为系统线程预留至少一个CPU。例如，如分配CPU 0至系统池，但不分配至定时结构池，则不要将任何定时结构分配至CPU 0。系统线程始终可使用CPU 0。

CPU使用率最大化

根据定时结构线程和系统线程的负载对比，确定每个池中的CPU数量，可最大化CPU的使用率。可使用RT Get CPU Loads VI估计定时结构线程和系统线程的总处理时间，相应地分配定时结构池和系统池。如RT终端外接显示器，也可使用屏幕上CPU Load Measurement工具估计负载的分布情况。

避免局部池重叠

如用户定义的系统池和定时结构池局部重叠，自动负载平衡过程可能无法做出最优化的决策，如下例所示：

范例1：

假设有一个包含三个定时循环的应用程序。将定时循环的周期配置为大多数时间三个定时循环只有一个在运行。但是，有时这些定时循环的周期可能会对齐，该情况发生时，需确保三个定时循环可并行执行。用户或许会将四核计算机上的三个CPU(CPU 0-2)分配至系统池，再将三个CPU(CPU 1-3)分配至定时结构池，认为CPU 3可始终运行三个定时结构中的一个，CPU 1和CPU 2大多数情况下运行系统线程，定时循环的周期对齐时各运行一个定时循环。但是，SMP排序器在索引最高的CPU上开始定时结构，然后在同一个CPU上运行各个线程，前两个定时循环可能都在CPU 3上运行，第三个在CPU 2上运行。三个循环的周期第一次对齐时，一个定时周期会转移到CPU 1，造成时间上的抖动。在该情况下，较好的解决方案是手动分配各个定时循环到不同的CPU，然后将所有四个CPU都分配至系统池。

范例2：

假设有一个应用程序，包含两个连续轮询的定时循环和两个间歇性运行的定时循环。用户可能会将四核系统的四个CPU都分配至定时结构池，将CPU 2和CPU 3分配至系统池。一般认为两个轮询定时循环在CPU 0和CPU 1上运行，间歇性运行的定时循环共享CPU 2和CPU 3的处理时间。但是，SMP排序器可能将轮询的定时循环分配给CPU 2和CPU 3，使其他系统线程处于空闲状态，CPU 0和CPU 1利用率较低。在该情况下，最高的解决方案是隔离两个CPU上的轮询定时循环，将剩余的两个CPU分

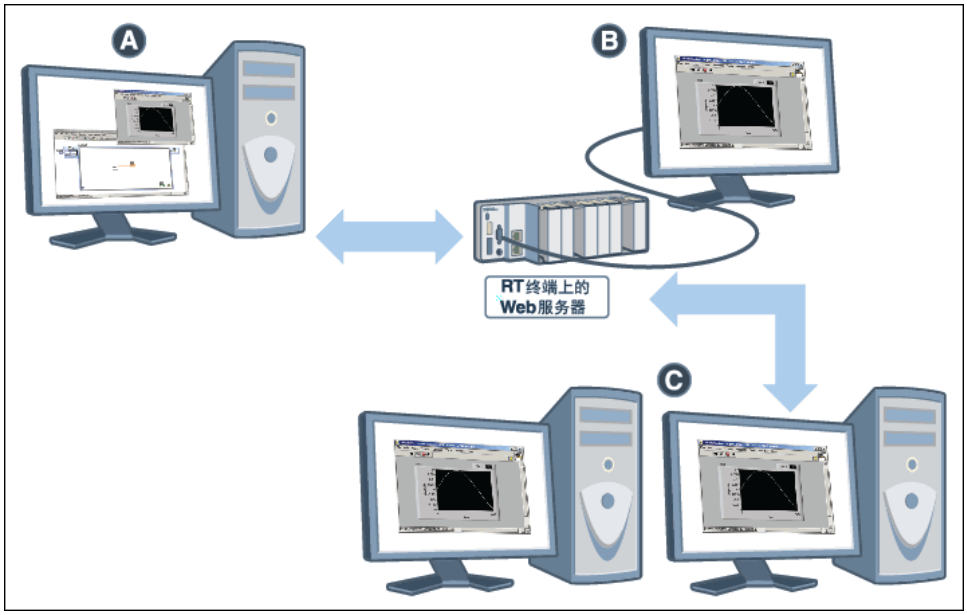
配给两个池。

使用SMP CPU工具VI

使用SMP CPU工具VI定义系统池和定时结构池。SMP CPU工具选板上有三个VI，提供了三种定义CPU池的不同方法。用户可根据编程风格和程序要求选择最合适的VI。RT设置CPU池大小 VI基于 RT设置CPU池分配 VI，后者基于RT设置CPU池 VI创建。RT设置CPU池大小VI对于大多数应用来说，是较好的选择。可使用RT Set CPU Pool Sizes VI创建默认池配置（两个池包含所有CPU），或创建连续的相邻的池。对于自定义池配置，可选择使用RT设置CPU池指派VI。只要各个池包含至少一个CPU，可使用RT设置CPU池指派VI创建池配置。如偏好使用比特掩码，可使用RT设置CPU池VI。

与RT终端的VI前面板交互

可通过主机、RT终端上的嵌入式UI，或远程前面板与RT终端VI的前面板交互。



A-主机

主机是用于开发VI和执行初始测试的主系统。通过主机可查看和控制运行在RT终端上的VI前面板和程序框图。虽然主机连接至RT终端，但仍然保留对前面板的控制，

使用户可通过连接至主机的输入设备与前面板上的UI元素交互。如前面板要求更新输入控件和显示控件的值，则主机将根据RT终端的的值进行相应更新。

B-嵌入式UI

许多RT终端没有用户界面，只有基本的控制台输出显示，但某些RT终端支持提供高级显示功能的嵌入式UI。嵌入式UI允许用户通过直接连接至RT终端的设备与RT终端VI的前面板交互，无须在开发完成后通过主机查看前面板。例如，在支持嵌入式UI的RT终端上启用嵌入式UI后，可在连接至RT终端的显示设备上查看VI的前面板。此外，如将主机从RT终端断开，还可通过连接至RT终端的输入设备控制VI前面板。除上述优点外，嵌入式UI还提供了一个图形化工作环境，包括文件浏览器和文本编辑器，有助于在RT终端上浏览和编辑文件。

C-远程前面板

远程前面板连接为通过Web服务器连接至RT终端提供了与前面板交互的方法。连接远程前面板后，可从远程计算机查看和控制运行在RT终端上的VI前面板。一个时间点上，VI的前面板只能由一个用户控制，但可被多个不同位置的用户同时查看。

显示RT终端VI的设计考虑要素

下表列出了开发通过嵌入式UI或远程前面板连接显示RT终端VI的设计考虑要素。一般来说，UI更新应置于实时循环之外，避免对确定性产生负面影响。同时注意启用嵌入式UI也会影响确定性。其他抖动对大多数应用程序都是可接受的。但如需保证系统最佳确定性，请禁用嵌入式UI。

在实时循环外放置前面板控件	在实时循环中更新前面板控件对确定性有负面影响。
在实时循环外放置属性节点	在实时循环中更新属性节点对确定性有负面影响。

请勿启用同步显示属性	启用控件的同步显示属性对确定性有负面影响。
为UI更新预留时间	更新前面板显示的代码运行优先级低于实时结构中的代码。实时代码占用大部分RT终端资源时，前面板更新可能有延迟。为避免更新延迟，应为定时结构的周期指定足够的持续时间，以便实时代码不会在循环中占用所有分配的资源。

显示器校验

由于操作系统的差异，前面板元素（如字体和控件）在嵌入式UI或远程前面板连接中的显示可能与主机上的显示有所不同。访问`ni.com/info`并输入`DisplayParity`，了解不同操作系统显示器校验的更多信息。

嵌入式UI的语言支持

嵌入式UI不支持本地化版的Real-Time模块。因此，使用嵌入式UI前需确保主机和RT终端使用英文版Real-Time模块。

Xfce与嵌入式UI

NI使用Xfce创建用于RT终端的嵌入式UI。Xfce组件集为类似UNIX操作系统提供轻量级桌面环境。关于使用和配置Xfce的信息，见Xfce网站上的Xfce帮助。

嵌入式UI模板

访问`ni.com/info`并输入`RTUITemplate`，可查看演示带嵌入式UI功能的生成者/消费者设计模式的应用。

嵌入式UI不支持的功能

除每个RT终端不支持的功能外，嵌入式UI还不支持下列LabVIEW功能。大多数情况下，启用嵌入式UI后将无法查看适用选板上的下列功能。如现有VI包含这些功能，则启用嵌入式UI后该VI将不会运行。

- 子面板

- XControl
- 三维图形
- 三维图片控件VI

Real-Time模块详解

本节的主题提供了一些具体流程，可帮助配置实时项目设置、实时应用程序内数据共享及生成确定性应用程序。

组织和管理LabVIEW Real-Time模块项目

项目用于组合LabVIEW文件和非LabVIEW文件、创建程序生成规范、部署设置以及部署VI至RT终端。可通过Real-Time项目向导或“项目浏览器”窗口创建项目。可在“项目浏览器”窗口添加RT终端至项目并配置RT终端的设置。

相关概念：

- 通过实时项目向导创建LabVIEW项目
- 在项目浏览器窗口创建LabVIEW项目
- 添加RT终端到LabVIEW项目
- 在LabVIEW项目中配置RT终端属性

添加RT终端到LabVIEW项目

必须使用LabVIEW项目来处理RT终端。按照下列步骤，添加RT终端至项目：

1. 使用Real-Time项目向导或“项目浏览器”窗口创建项目。
2. 右键单击项目的根目录，从快捷菜单中选择**新建»终端和设备**，显示“添加终端和设备”对话框。如果该项目中的RT终端支持其他终端，也可右键单击该终端并从快捷菜单中选择**新建»终端和设备**，在现有终端下添加终端。例如，如在NI PXI终端下安装了NI FPGA终端，可将设备安装在RT终端下。
3. 从**添加终端和设备**对话框的**终端和设备**部分，选择要添加的RT终端类型。选择下列选项指定RT终端：
   - **现有终端或设备** – 允许显示本地子网或指定地址上的所有终端和设备。
     - **搜索现有终端或设备** – 显示本地子网上的全部终端和设备。
     - **按IP地址指定终端或设备** – 显示支持的设备类型。从列表中选择一个设

备，然后输入设备的IP地址以将设备添加到项目中。

- **新终端或设备** – 显示在没有物理终端或设备的情况下，可创建的终端和设备。必须在RT终端“属性”对话框的“常规属性”页中修改终端的属性，以指定名称和IP地址，且在尝试连接或在终端上运行VI前，必须将其绑定至现有RT系列设备。

4. 选择终端并单击**确定**按钮。**项目浏览器**窗口会出现一个表示RT终端的项。

在RT终端下可添加和删除项。

相关概念：

- 通过实时项目向导创建LabVIEW项目
- 在项目浏览器窗口创建LabVIEW项目
- 在LabVIEW项目中配置RT终端属性

在LabVIEW项目中配置RT终端属性

按照下列步骤，配置RT终端的项目设置：

1. 在“项目浏览器”窗口右键单击RT终端，并从快捷菜单中选择**属性**，打开RT终端“属性”对话框。
2. 从**类别**列表中选择配置页。
3. 在配置页上，配置RT终端的各项设置。
4. 单击**确定**按钮，接受设置并关闭RT终端**属性**对话框。
5. 在**项目浏览器**窗口选择**项目**»**保存项目**，保存RT终端**属性**对话框中为RT终端选择的设置，从而保存项目。必须先部署RT终端，设置才会生效。部署RT终端就是将项目的设置应用至终端。

> **注：** 部署RT终端就是用RT终端**属性**对话框中指定的最新设置覆盖当前的终端设置。

6. 右键单击RT终端，从快捷菜单中选择**部署**，将设置部署至终端。

通过实时项目向导创建LabVIEW项目

使用实时项目向导新建项目、配置项目设置、定义RT终端以及包含VI和其他应用程序文件。

创建新项目

按照下列步骤使用实时项目向导创建一个项目：

1. 选择**工具**»**Real-Time模块**»**项目向导**，打开实时项目向导。也可在“启动”窗口的**新建**部分单击**Real-Time项目**，打开实时项目向导。
2. 在**选择项目类型、名称、文件夹**页的**项目类型**下拉菜单中选择要创建的项目类型。

   提示 可通过Real-Time项目向导创建下列三种架构的项目——连续通信、状态机，以及自定义。如选择**连续通信架构**，则在需要实时任务的确定性性能时，启用**应用程序包含确定性组件**。

3. 在**项目名称**文本框中输入新项目的名称，在**项目文件夹**文本框中输入项目的保存目录。
4. 单击**下一步**按钮，接受设置并继续。

配置项目设置

实时项目向导根据不同的项目架构提供不同的设置。按照所选的项目架构完成下列操作。

连续通信自定义架构选项

1. 在**终端配置**部分选择所需的定时循环数量。

   注： 如要创建一个既包含确定性行为，又包含RT终端文件I/O的应用程序，必须选择**两个循环**。

2. 如要创建一个记录数据至RT终端磁盘的应用程序，需启用**包含文件I/O**。
3. 启用**包含用户界面**，然后从**主机配置**部分选择用户界面的类型，创建在主机上提供用户界面的应用程序。
4. 单击**下一步**按钮，接受设置并继续。

状态机 – 状态机架构不需要额外的配置。单击**下一步**按钮继续。

自定义 添加顶层VI

1. 选择添加空VI或导入现有VI至项目。

- 选择**添加空VI至终端**，添加一个运行在RT终端上的空VI。跳到步骤。
- 选择**导入现有VI至主机和终端**，导入运行在终端或主机上的现有VI。

1. 单击**导入以便在主机上运行的VI**列表框下的**添加VI**按钮，添加在主机上运行的VI。
2. 单击**导入以便在终端上运行的VI**列表框下的**添加VI**按钮，添加在RT终端上运行的VI。
3. 单击**下一步**按钮，接受设置并继续。

选择项目的RT终端

按照下列步骤，添加RT终端至项目：

1. 单击**浏览终端**页上的**浏览**按钮，选择要添加到项目中并在其上运行终端VI的终端。Real-Time项目向导会在**选定终端**文本框中显示所选终端的名称。
2. 单击**下一步**按钮，接受设置并继续。

生成项目

按照下列步骤生成项目：

1. 在**预览项目**页查看新项目配置。
2. 如不想在Real-Time项目向导创建完毕后立即打开VI，需禁用**完成后打开VI**。
3. 单击**完成**按钮，接受设置并生成新项目。

在项目浏览器窗口创建LabVIEW项目

必须通过项目才能使用RT终端。项目可用于组合LabVIEW文件和非LabVIEW特有的文件，创建程序生成规范以及部署或下载文件至RT终端。

按照下列步骤，手动创建项目：

1. 选择**文件**»**创建项目**打开**项目浏览器**窗口。也可以选择**项目**»**创建项目**或在新建对话框中选择**空项目**，显示**项目浏览器**窗口。
2. 添加RT终端至项目。
3. 添加要在项目中包含的项至RT终端。
4. 选择**文件**»**保存项目**，保存项目。

也可以使用RT项目向导交互地创建项目。

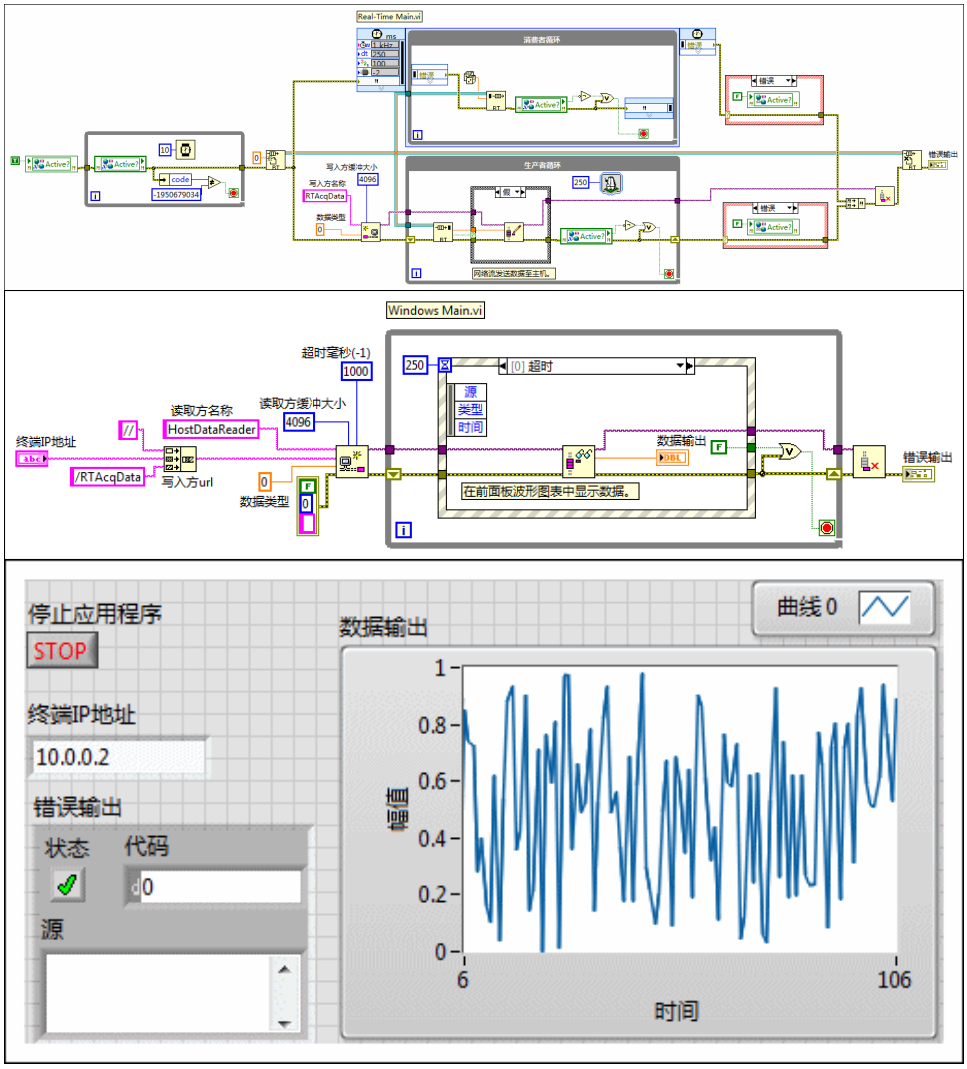
相关概念：

- 添加RT终端到LabVIEW项目
- 通过实时项目向导创建LabVIEW项目

教程：创建Real-Time应用程序

本教程将教授如何使用Real-Time模块创建一个基本的实时应用程序。用户所了解的实时应用程序由2个VI组成：一个在RT终端上运行，一个在主机上运行。RT终端上的VI从RT终端生成数据，然后将数据流式传输到主机上的VI。主机上的VI读取数据并将其显示在波形图中。

本教程从空白VI开始讲起，提供了全面的学习体验。如果想通过修改一个正在运行的应用程序来进行学习，也有多个模板和范例项目可供参考。要获取这些模板，请从LabVIEW**启动**窗口中选择**创建项目**，显示**创建项目**对话框。



准备

确认可以在网络上找到RT终端，并且已安装软件。关于查找RT终端和安装软件的详细信息，见Measurement & Automation Explorer帮助中的远程系统主题。要打开Measurement & Automation Explorer帮助，请从NI MAX选择**帮助»MAX帮助**。

教程内容

第一部分：在RT终端上生成数据 了解如何创建生产者循环，在RT终端上生成数据。

第二部分：在RT终端上处理数据 了解如何创建消费者循环，在RT终端上处理数据，以及如何将数据从生产者循环以确定性方式传输到消费者循环。

第三部分：同时停止多个循环 了解如何使用网络发布的共享变量创建同时停止生产者循环和消费者循环的方法。

第四部分：生成用户界面 了解如何在主机上创建基本的用户界面，以及如何使用创建的UI停止RT终端上运行的循环。

第五部分：发送RT数据至主机 了解如何使用网络流从RT终端向主机发送数据。

第六部分：在用户界面上显示RT数据 了解如何使用网络流接收主机上的RT终端数据，以及如何在用户界面上显示数据。

相关概念：

第一部分：在RT终端上生成数据

可使用RT终端上的实时操作系统指定进程以确定性方式运行。确定性进程可以优先访问硬件资源，这意味着它可以在限定的时间范围内执行，而不会有被较低优先级进程中断的风险。在本教程中，我们将使用定时循环来指定一个可在RT终端上生成数据的确定性进程。在生产者/消费者设计中，我们将此循环称为生产者循环，因为它会生成数据供另一个循环处理。我们将在本教程的下一部分介绍消费者循环。访问`ni.com/info`并输入`PCdesign`，了解生产者/消费者设计。

设置LabVIEW项目

在开始设计应用程序之前，您需要将RT终端添加到LabVIEW项目中，然后添加一个

VI以在RT终端上运行。添加VI至LabVIEW项目层次结构中的硬件上，它将在此运行。如果将VI添加至**我的电脑**，它将在主机上运行。如果将VI添加到RT终端项目项中，它将在RT终端上运行。

按照下列步骤，创建LabVIEW项目：

1. 在LabVIEW**启动**窗口中，选择**文件**»**创建项目**，显示**创建项目**对话框。
2. 选择**空白项目**，然后单击**完成**。
3. 选择**文件**»**保存**，保存项目。
4. 将项目命名为`Getting Started`。
5. 单击**确定**。

按照以下步骤，找到RT终端：

1. 在**项目浏览器**窗口中，右键单击项目根目录，然后从快捷菜单中选择**新建**»**终端和设备**，显示**添加终端和设备**对话框。
2. 选择**现有终端或设备**。
3. 选择**搜索现有终端或设备**。
4. 在**终端和设备类型**列表框中指定RT终端的类型。例如，如果RT终端为cRIO-9033，请展开**Real-Time CompactRIO**文件夹并选择**cRIO-9033**。
5. 单击**确定**，显示**选择编程模式**对话框。
6. （FPGA终端）选择**扫描接口**。

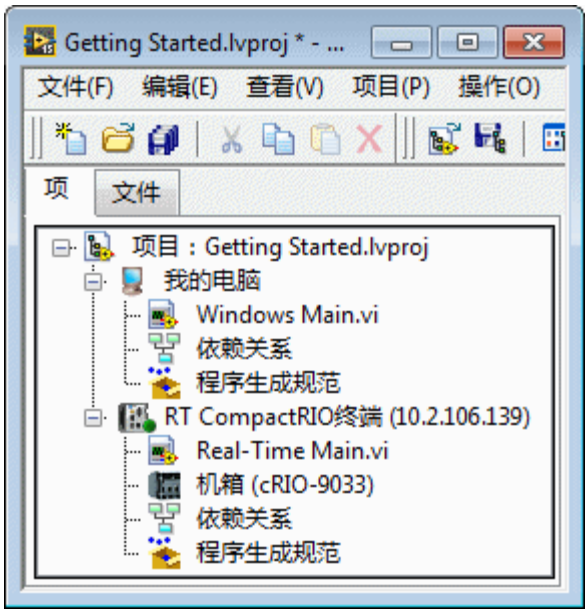
> **注：** 可以选择**LabVIEW FPGA接口**而不是**扫描接口**来对FPGA进行编程，以获得更高的性能和灵活性。

7. 单击**继续**。

按照下列步骤，添加新VI至项目：

1. 右键单击**项目浏览器**窗口的RT终端，选择**新建**»**VI**。
2. 在前面板中，选择**文件**»**保存**，保存VI。
3. 将VI命名为`Real-Time Main`。

4. 单击**确定(OK)**。



创建定时循环

实时操作系统会为LabVIEW中的每个进程分配优先级。后台任务的优先级最低。LabVIEW进程分为低优先级、正常优先级和高优先级。设置为实时优先级的VI将获得最高优先级。定时循环是一种确定性结构，执行优先级在高优先级之上，实时优先级之下。向VI添加定时循环时，必须为循环指定周期和优先级。周期是循环一次迭代所分配的执行时间。当VI中有多个循环时，优先级将告诉LabVIEW先执行哪个循环。

注： 为避免破坏定时循环的确定性，请将所有其他循环设置为正常优先级，并在应用程序中使用多个定时循环时谨慎行事。

按照以下步骤，在RT终端VI中配置定时循环：

1. 使用**编程»结构»定时结构**下的**函数**选板在“RT主程序”的程序框图中添加定时循环。
2. 双击定时循环的输入节点，显示**配置定时循环**对话框。
3. 从**源类型**列表框中选择**1 kHz时钟**。
4. 在**周期**文本框中输入250，使循环每秒运行4次。

> **注：** 设置的周期应能为定时循环内的代码提供足够的执行时间。此外，周期还应为低优先级进程留出额外的执行时间，防止CPU空闲。

5. 在**优先级(Priority)**文本框中输入`100`。
6. 单击**确定(OK)**。
7. 右键单击定时循环的循环条件接线端，从快捷菜单中选择**创建输入控件**。LabVIEW将在前面板上添加一个“停止”按钮。

添加代码以生成数据

在本节中，将使用随机数函数在RT终端上生成数据。随机数函数是特定于应用程序代码的占位符。对于实际应用程序，请使用特定代码替换随机数函数，该代码可生成要以确定性方式处理的数据。

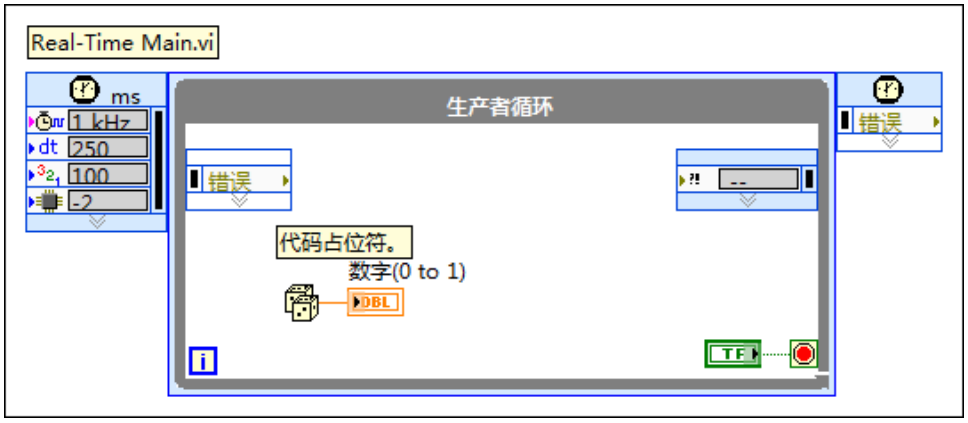
按照以下步骤，生成和显示数据：

1. 在定时循环中添加一个随机数函数。
2. 右键单击随机数函数，从快捷菜单中选择**创建»显示控件**。

> **注：** 根据最佳实践，不应在实时代码中包含显示控件。以确定性方式生成数据非常重要，但不需要以确定性方式显示数据。本步骤中的显示控件用于测试RT终端是否生成数据。本教程后续部分中会将其删除。

3. 保存该VI。

程序框图应与下图一致。



在RT终端上运行VI

在开始运行代码之前，必须先部署VI，使其能够在RT终端上运行。部署VI时，LabVIEW会将VI和所有相关文件保存到RT终端内存中。

按照以下步骤，将VI部署到RT终端：

1. 在**项目浏览器**窗口右键单击RT终端，从快捷菜单中选择**属性**，显示**RT终端属性**对话框。
2. 在**IP地址/DNS名称**文本框中指定IP地址。

   > **注：** 可以使用Measurement & Automation Explorer (MAX)确定RT终端的IP地址。

3. 单击**确定**。
4. 在**项目浏览器**窗口，右键单击VI并从快捷菜单中选择**部署**。

结果

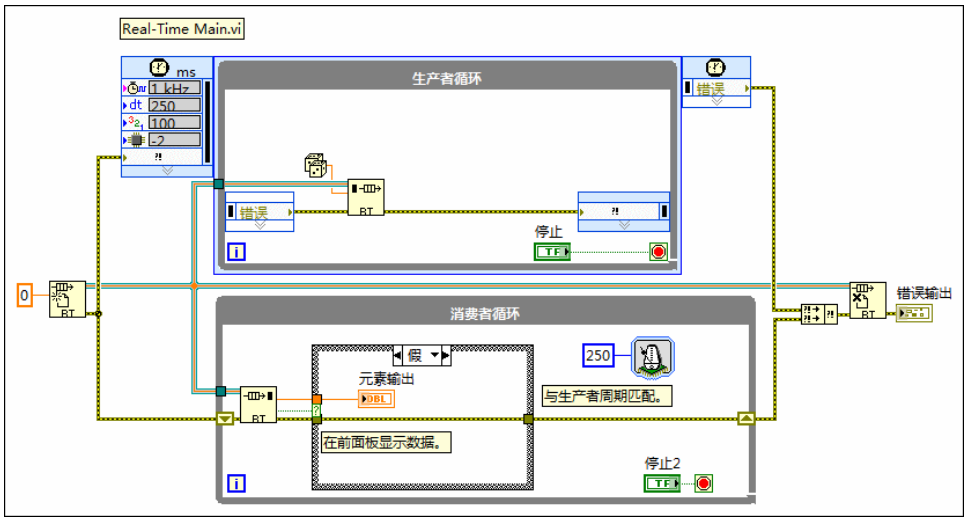
运行“RT主程序”时，RT终端会以确定性方式生成随机数，显示控件会在前面板上显示这些数字。

相关概念：

- 教程：创建Real-Time应用程序
- 第二部分：在RT终端上处理数据

第二部分：在RT终端上处理数据

本教程的这一部分将创建消费者循环来处理生产者循环生成的数据。消费者循环使用While循环而不是定时循环，因为处理数据任务比生成数据的优先级要低。也就是说，如果硬件资源有限，在限定时间内生成数据比处理数据更重要。



创建消费者循环

典型的做法是使用等待下一个整数倍毫秒函数同步While循环和定时循环。以这种方式同步循环并非必要，但可为While循环提供定时循环执行结束后周期内剩余的执行时间。

按照以下步骤，创建一个与生产者循环周期相同的消费者循环：

1. 在“RT主程序”的程序框图中，在定时循环下方放置一个While循环。
2. 在While循环中放置一个等待下一个整数倍毫秒函数。
3. 右键单击等待下一个整数倍毫秒函数的**毫秒倍数**输入端，从快捷菜单中选择**创建»常量**。
4. 输入250作为常量，以匹配定时循环的周期。

将数据从生产者循环以确定性方式传输到消费者循环

LabVIEW提供了许多传输数据的方法，其中有些是确定性的，有些则不是。RT FIFO就是确定性数据通信方法的一个范例。为实现确定性，RT FIFO在运行前会为数据

传输预先分配内存，从而防止向内存写入数据时出现意外延迟。RT FIFO以先进先出的顺序缓冲和传输数据。可以在函数选板的**Real-Time»RT FIFO**下找到RT FIFO函数。如需了解其他可用的数据通信方法，见LabVIEW中的数据通信方法帮助主题。

按照以下步骤，使用RT FIFO将数据从生产者循环传输到消费者循环：

1. 将RT FIFO创建函数放置在循环的外部左侧。
2. 用RT FIFO写入函数替代定时循环中的显示控件。

> **注：** 确保将随机数函数的输出端连线至RT FIFO写入函数的**元素**输入端。

3. 在While循环内放置RT FIFO读取函数。
4. 将RT FIFO删除函数放置在循环的外部右侧。
5. 将DBL数值常量连线至RT FIFO创建函数的**类型**输入端。
6. 将RT FIFO创建函数的**rt fifo**输出端连线至其他3个RT FIFO函数的**rt fifo**输入端。
7. 右键单击While循环的循环条件图标，从快捷菜单中选择**创建输入控件**，创建“停止”按钮。
8. 在While循环中放置一个条件结构。
9. 在条件结构选择器标签中选择**False**。
10. 将RT FIFO读取函数的**空?**输出端连线至条件结构的条件选择器。
11. 右键单击RT FIFO读取函数的**元素输出**输出端，从快捷菜单中选择**创建»显示控件**。
12. 将显示控件放置在条件结构中。
13. 将显示控件连线至RT FIFO读取函数的**元素输出**输出端。

添加错误处理

最佳实践是连线函数的错误接线端。这样就可以通过VI传递错误信息，并在不暂停VI执行的情况下恰当地处理错误。

按照以下步骤，在“RT主程序”中连线错误：

1. 调整定时循环输入节点的大小，使其包含5个输入端。

2. 右键单击定时循环输入节点的底部输入端，从快捷菜单中选择**选择输入»错误**。
3. 将RT FIFO创建函数的**错误输出**输出端连线至定时循环输入节点的**错误**输入端。
4. 将定时循环左数据节点的**错误**输出端连线至RT FIFO写入函数的**错误输入**输入端。
5. 将RT FIFO写入函数的**错误输出**输出端连线至定时循环右数据节点的**错误**输入端。
6. 在循环的外部右侧放置合并错误函数。
7. 将定时循环输出节点的**错误**输出端连线至合并错误函数的顶部输入端。
8. 将RT FIFO创建函数的**错误输出**输出端连线至RT FIFO读取函数的**错误输入**输入端。
9. 将RT FIFO读取函数的**错误输出**输出端通过条件结构连线至合并错误函数的底部**错误输入**输入端。
10. 将合并错误函数的**错误输出**输出端连线至RT FIFO删除函数的**错误输入**输入端。
11. 右键单击RT FIFO删除函数的**错误输出**输出端，选择**创建»显示控件**。
12. 右键单击While循环错误隧道之一，从快捷菜单中选择**替换为移位寄存器**。
13. 单击另一个While循环错误隧道，将其替换为移位寄存器。
14. 在条件结构选择器标签中选择`True`。
15. 将错误连线直接穿过条件分支。
16. 保存该VI。
17. 在**项目浏览器**窗口，右键单击VI并从快捷菜单中选择**部署**。

结果

运行“RT主程序”时，生产者循环具有最高优先级，可将RT终端生成的值写入RT FIFO缓冲区。在每个周期内只要CPU可用，消费循环就会从RT FIFO缓冲区读取这些值。

注： 单击前面板上的两个“停止”按钮，停止VI。

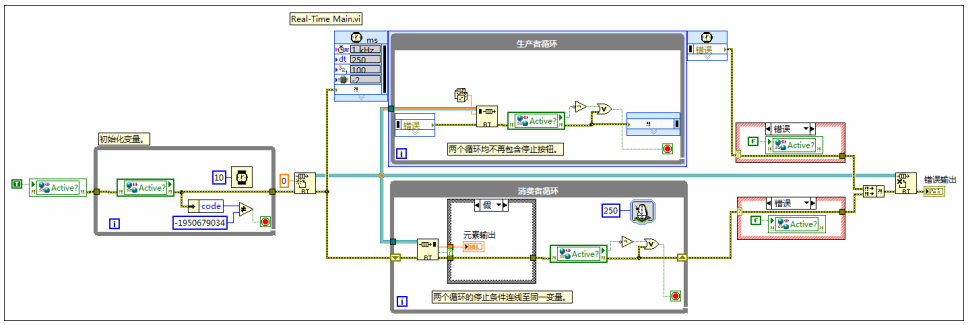
相关概念：

- 第一部分：在RT终端上生成数据

- 教程：创建Real-Time应用程序
- 第三部分：同时停止多个循环

第三部分：同时停止多个循环

创建一个安全的关闭进程是实时应用程序的基本组成部分。例如，应用程序一个部分控制设备的某个部分，应用程序另一部分监视预警状态，则可能希望发生预警状态时停止设备。此时，可使用网络发布的共享变量将多个循环的停止和错误函数合并为一个函数。



创建共享变量

网络发布的共享变量十分适用于广播关闭事件，因为网络发布的共享变量可与多个VI通信且其读写操作不会互相干扰。

按照下列步骤，配置网络发布的共享变量：

1. 在**项目浏览器**窗口右键单击RT终端，从快捷菜单中选择**新建»变量**，显示**共享变量属性**对话框。
2. 按照下列步骤配置变量：
   a. 在**共享变量属性**对话框**变量**页的**名称**文本框中，输入`Active?`。
   b. 从**变量类型**下拉菜单中选择**网络发布**。
   c. 从**数据类型**下拉菜单中选择**布尔**。
   d. 在**共享变量属性**对话框的**RT FIFO**页中，勾选**启用RT FIFO**复选框。
   e. 单击**确定**。
3. 在**项目浏览器**窗口中，右键单击包含`Active?`变量的库，并从快捷菜单中选择**保存»保存**。

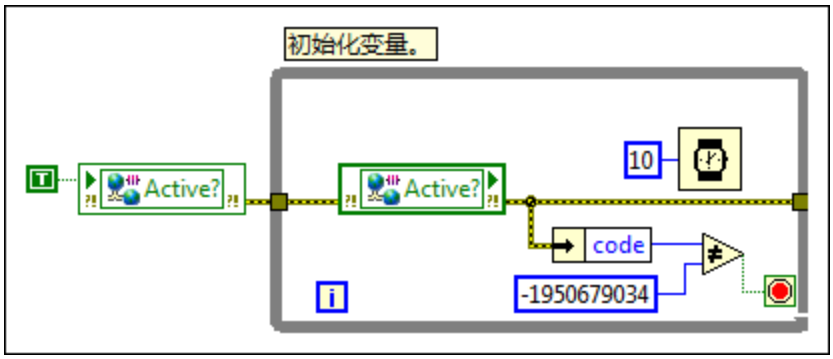
4. 将库命名为Variables。
5. 单击**确定**。
6. 单击Active?变量并将其从**项目浏览器**窗口拖曳到“RT主程序”的程序框图上，并将其放置在RT FIFO创建函数的左侧。
7. 右键单击Active?变量，从快捷菜单中选择**访问模式**»**写入**。
8. 将真常量连线至Active?变量的**活动?** 输入端。

初始化变量

启动应用程序时，代码的许多片段会并行加载。这意味着应用程序的某些部分可能会在其他部分准备就绪之前加载。为防止应用程序因网络发布的共享变量初始化时间不足而停止，最佳实践是使用While循环检查初始化是否正确。

按照以下步骤初始化Active?变量：

1. 在Active?变量和RT FIFO创建函数之间添加一个While循环。
2. 创建Active?变量的副本并将其放置于While循环中。
3. 右键单击While循环中的Active?变量，从快捷菜单中选择**访问模式**»**读取**。
4. 在While循环中添加按名称解除捆绑函数。
5. 将按名称解除捆绑函数的输入端连线至Active?变量的**错误输出**输出端。
6. 右键单击按名称解除捆绑函数，从快捷菜单中选择**选择项**»**代码**。
7. 在While循环中添加一个不等于？函数。
8. 将按名称解除捆绑函数的**代码**输出端连线至不等于？函数的**x**输入端。
9. 右键单击不等于？函数的**y**输入端，从快捷菜单中选择**创建**»**常量**。
10. 输入**-1950679034**作为常量。这是共享变量无值时LabVIEW返回的错误代码。
11. 将不等于？函数的**x!=y?**输出端连线至While循环的停止条件。
12. 在While循环中添加一个等待(ms)函数。
13. 右键单击等待(ms)函数的**等待时间（毫秒）** 输入端，然后从快捷菜单中选择**创建**»**常量**。
14. 输入**10**作为常量，以检查变量是否每10毫秒初始化一次。



创建关闭条件

关闭条件可确保在应用程序的某个部分出现严重错误或用户停止应用程序时，每个进程都会停止。

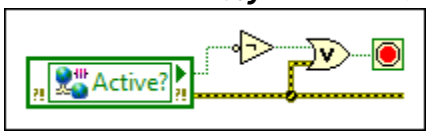
按照以下步骤，使用共享变量创建安全的关闭条件：

1. 将Active?变量的**错误输出**输出端连线至While循环中Active?变量的**错误输入**输入端。
2. 将While循环中Active?变量的**错误输出**输出端连线至RT FIFO创建函数的**错误输入**输入端。
3. 删除RT FIFO写入函数与定时循环右数据节点之间的错误连线。
4. 删除定时循环输出节点与合并错误函数之间的错误连线。
5. 复制Active?变量，并将副本粘贴到定时循环内、RT FIFO写入函数的右侧。
6. 将RT FIFO写入函数的**错误输出**输出端连线至定时循环中Active?变量的**错误输入**输入端。
7. 右键单击定时循环中的Active?变量，从快捷菜单中选择**访问模式»读取**。
8. 在定时循环中的Active?变量右侧放置一个非函数。
9. 将Active?变量的**活动?** 输出端连线至非函数的**x**输入端。
10. 在非函数的右侧放置一个或函数。
11. 将非函数的**非x?** 输出端连线至或函数的**x**输入端。
12. 将定时循环中Active?变量的**错误输出**输出端连线至或函数的**y**输入端。
13. 在定时循环的外部右侧放置一个条件结构。
14. 将定时循环中Active?变量的错误连线连接到定时循环右数据节点的**错误**输入端。
15. 将定时循环输出节点的**错误**输出端连线至条件结构的条件分支选择器。

16. 选中选择器标签中的Error，在条件结构中放置Active?变量的副本。
17. 右键单击Active?变量，从快捷菜单中选择**访问模式»写入**。
18. 将假常量连线至Active?变量的**活动?** 输入端。
19. 穿过条件结构连接错误连线，但不要将其连线至Active?变量。

> **注：** 确保选中选择器标签中的No Error，然后再次完成此步骤。

20. 将离开条件结构的错误连线连接到合并错误函数的**错误输入**输入端。
21. 删除定时循环循环条件中的“停止”按钮。
22. 将或函数的**x或y?** 输出端连线至定时循环的循环条件。



23. 重复步骤3至22设置消费者循环，以与图像一致。

> **注：** 确保从RT FIFO读取函数到Active?变量的错误连线穿过条件结构。

24. 保存该VI。

终止VI

在教程的这一部分，VI将持续运行，因为无法通过前面板将Active?变量更改为False。要停止VI，可以使用NI分布式系统管理器更改Active?变量的值。NI分布式系统管理器是一款非常有用的工具，可以在代码执行时监控网络发布的共享变量。

按照以下步骤关闭VI，无需使用“中止执行”按钮。

1. 打开NI分布式系统管理器。
2. 展开**名称**列中的**网络项**文件夹。
3. 找到并展开RT终端根目录。
4. 展开**变量**文件夹。
5. 选择Active?变量。
6. 在**自动视图**窗口中，将**新值**设置为**False**。
7. 单击**设置**。

8. 关闭分布式系统管理器。

结果

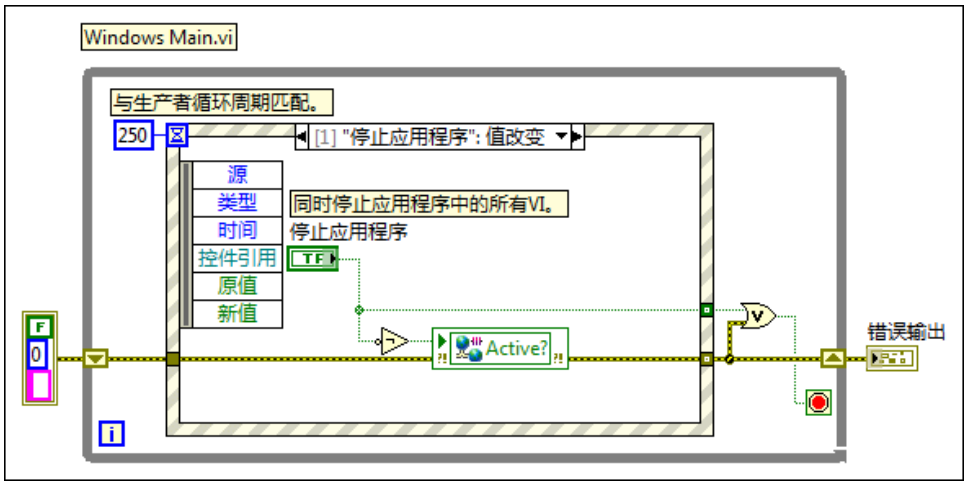
运行VI时，如果`Active?`变量为False或LabVIEW检测到错误，循环将同步停止。

相关概念：

- 第二部分：在RT终端上处理数据
- 教程：创建Real-Time应用程序
- 第四部分：生成用户界面

第四部分：生成用户界面

虽然许多RT终端没有用户界面，但显示RT终端的信息或向实时应用程序发送命令往往很有用。如要完成这些任务，可以创建一个在主机或RT终端（适用于支持嵌入式UI远程前面板的RT终端）上运行的用户界面。在本部分教程中，我们将创建一个UI，使用户能够通过主机停止实时应用程序。



添加VI至主机

与RT终端VI不同，运行前无需部署主机VI。由于是在主机上进行开发，LabVIEW会自动将VI保存到计算机上。

按照以下步骤，将新VI添加至主机：

1. 在**项目浏览器**窗口中，右键单击**我的电脑**，然后从快捷菜单中选择**新建»VI**。
2. 在前面板中，选择**文件»保存**，保存VI。
3. 将VI命名为`Windows Main`。
4. 单击**确定**。

创建停止应用程序的事件条件分支

在教程的UI中，事件结构会等待用户按下“停止”按钮。可以设置事件结构的周期，以匹配定时循环，使事件结构与RT终端上运行的循环同步。

按照以下步骤，创建等待用户按下“停止”按钮的事件结构：

1. 添加停止按钮至“Windows主程序”的前面板。
2. 在“停止”按钮标签中输入`Stop Application`。
3. 添加While循环至VI程序框图。
4. 在While循环中添加一个事件结构。
5. 右键单击事件结构的选择器标签，从快捷菜单中选择**添加事件分支**，显示**编辑事件**对话框。
6. 在**事件源**列表框中选择**停止应用程序**。
7. 在**事件**列表框中选择**值改变**。
8. 单击**确定**。
9. 在事件结构中放置**停止应用程序**输入控件。
10. 右键单击事件结构的事件超时接线端，然后从快捷菜单中选择**创建常量**。
11. 输入`250`作为常量，与“RT主程序”中定时循环的周期相符。

同步两个VI的停止函数

本节将介绍如何使用“RT主程序”中所用的相同共享变量同步应用程序中所有循环的停止函数。将整个应用程序链接到一个停止按钮，可以安全、同步地停止应用程序中的所有进程。

按照以下步骤，同步应用程序的停止函数：

1. 在“Windows主程序”的事件结构中放置一个`Active?`变量的副本。

2. 右键单击Active?变量，从快捷菜单中选择**访问模式»写入**。
3. 在事件结构中添加一个非函数。
4. 在事件结构外、While循环内添加一个或函数。
5. 将**停止应用程序**输入控件的输出端连线至非函数的**x**输入端。
6. 将**停止应用程序**输入控件的输出端连线至或函数的**x**输入端。
7. 将非函数的**非x?**输出端连线至Active?变量的**活动?**输入端。
8. 将或函数的**x或y?**输出端与While循环的条件接线端连线。

添加错误处理

与教程的其他部分一样，建议连线函数的错误接线端。

按照以下步骤，在“Windows主程序”中连接错误连线：

1. 右键单击Active?变量的**错误输出**输出端，从快捷菜单中选择**创建»显示控件**。
2. 右键单击Active?变量的**错误输入**输入端，从快捷菜单中选择**创建»常量**。
3. 将**错误输入**常量放置在While循环的外部左侧。
4. 将**错误输入**常量穿过While循环和事件结构连线，并连接到Active?变量的**错误输入**输入端。
5. 将Active?变量的**错误输出**输出端连线至或函数的**y**输入端。
6. 将**错误输出**显示控件放置在While循环的外部右侧。
7. 将Active?变量的**错误输出**输出端连线至**错误输出**显示控件的输入端。
8. 选择事件结构选择器标签中的**超时**。
9. 将错误连线直接穿过事件结构。
10. 右键单击While循环错误隧道之一，从快捷菜单中选择**替换为移位寄存器**。
11. 单击另一个While循环错误隧道，将其替换为移位寄存器。
12. 保存该VI。

结果

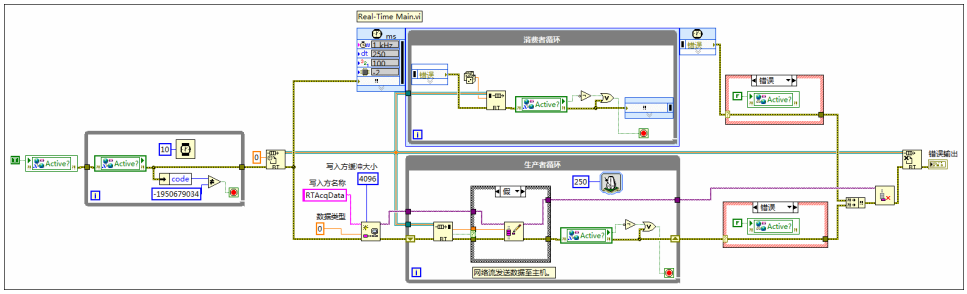
运行“RT主程序”和“Windows主程序”并单击Windows主程序中的**停止应用程序**按钮时，LabVIEW将停止整个应用程序，包括在“RT主程序”中运行的循环。

相关概念：

- 第三部分：同时停止多个循环
- 教程：创建Real-Time应用程序
- 第五部分：发送RT数据至主机

第五部分：发送RT数据至主机

在本部分教程中，将使用网络流函数在网络上发送流数据。关于其他数据传输方法的信息，见LabVIEW帮助中LabVIEW中的数据通信方法主题。



配置网络流写入方

要配置用于发送数据的网络流，必须指定写入方名称、数据类型和写入方缓冲区大小。这些属性指定了发送数据的端点、发送的数据类型以及发送的数据量。

按照以下步骤配置创建网络流写入方端点函数：

1. 在“RT主程序”的程序框图中添加创建网络流写入方端点函数。
2. 删除RT FIFO创建函数和RT FIFO读取函数之间的错误连线。
3. 依次连接RT FIFO创建函数、创建网络流写入方端点函数和RT FIFO读取函数之间的错误连线。
4. 右键单击创建网络流写入方端点函数的**写入方名称**输入端，从快捷菜单中选择**创建»常量**。
5. 输入`RTAcqData`作为常量来命名写入方端点。
6. 将DBL数值常量连线至创建网络流写入方端点函数的**数据类型**输入端。
7. 右键单击创建网络流写入方端点函数的**写入方缓冲区大小**输入端，从快捷菜单中选择**创建»常量**。

> **注：** 设置的缓冲区大小应足以容纳需要流式传输的数据。所用缓冲区大小不足会导致错误。

8. 设置常量为`4096`。

从RT FIFO函数向网络流传输数据

与RT FIFO类似，网络流可传输缓冲数据。不过，由于网络流是通过网络而不是在RT终端内传输数据，不能保证确定性通信。所有网络通信本质上都是非确定性的，因为网络上的执行时间和传输速率因循环迭代而异。

按照以下步骤，设置网络流写入方，将数据流式传输到网络：

1. 从条件结构选择器标签中选择**False**。
2. 在条件结构中添加写入单个元素至流函数。
3. 删除贯穿条件结构的错误连线。
4. 按顺序连接RT FIFO读取函数、写入单个元素至流函数和`Active?`变量之间的错误连线。
5. 将创建网络流写入方端点函数的**写入方端点**输出端连线至写入单个元素至流函数的**端点输入**输入端。
6. 删除**错误输出**显示控件。
7. 将RT FIFO读取函数的**元素输出**输出端连线至写入单个元素至流函数的**数据输入**输入端。
8. 在循环的外部右侧添加一个销毁流端点函数。该函数可安全关闭网络流。
9. 删除合并错误函数和RT FIFO删除函数之间的错误连线。
10. 按顺序连接合并错误函数、销毁流端点函数和RT FIFO删除函数之间的错误连线。
11. 将写入单个元素至流函数的**端点输出**输出端连线至销毁流端点函数的**端点输入**输入端。
12. 从条件结构选择器标签中选择**True**。
13. 将错误连线直接穿过条件分支。
14. 穿过条件结构连接端点连线。
15. 保存该VI。

结果

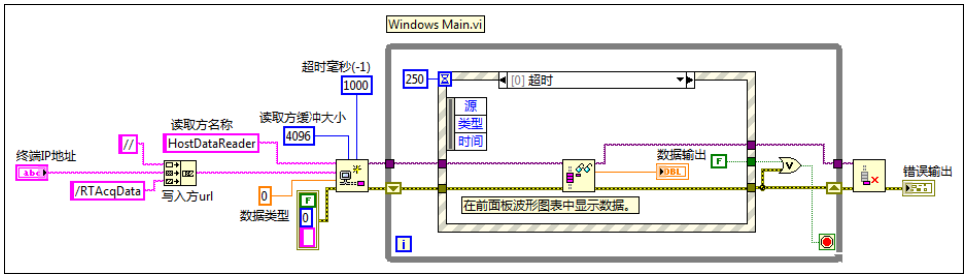
此时请不要运行VI。网络流可以使用写入方端点发送数据，但在添加读取方端点之前无法接收数据。可以在本教程的下一部分添加读取方端点后测试VI。

相关概念：

- 第四部分：生成用户界面
- 教程：创建Real-Time应用程序
- 第六部分：在用户界面上显示RT数据

第六部分：在用户界面上显示RT数据

在本部分教程中，使用网络流通过网络接收RT终端数据，并在用户界面上以图形方式显示数据。访问ni.com/info并输入EmbUIOptions，了解关于创建显示RT终端数据的界面的详细信息。



配置网络流读取方

要配置用于接收数据的网络流，必须指定读取方名称、数据类型和读取方缓冲区大小。这些属性指定了读取数据的端点、读取的数据类型以及读取的数据量。还可以指定一个时间限制，规定读取方端点等待数据的时间。

按照以下步骤配置创建网络流读取方端点函数：

1. 在“Windows主程序”的程序框图中添加创建网络流读取方端点函数。
2. 删除While循环的**错误输入**常量和左侧错误隧道之间的错误连线。
3. 在事件结构选择器标签内选中Stop Application，依次连接**错误输入**常量、创建网络流读取方端点函数和Active?变量之间的错误连线。

4. 右键单击创建网络流读取方端点函数的**读取方名称**输入端，从快捷菜单中选择**创建»常量**。
5. 输入`HostDataReader`作为常量来命名读取方端点。
6. 将DBL数值常量连线至创建网络流读取方端点函数的**数据类型**输入端。
7. 右键单击创建网络流读取方端点函数的**超时毫秒**输入端，从快捷菜单中选择**创建»常量**。
8. 输入`1000`作为常量，以指定读取方在等待新数据1秒后才将执行传递到VI的下一部分。
9. 右键单击创建网络流读取方端点函数的**读取方缓冲区大小**输入端，从快捷菜单中选择**创建»常量**。
10. 输入`4096`作为常量。

创建输入控件指定RT终端

网络流需要发送数据的RT终端的IP地址或主机名才能读取数据。

按照以下步骤，在前面板上创建一个输入控件，以指定RT终端的IP地址：

1. 在创建网络流读取方端点函数的左侧添加连接字符串函数。
2. 扩展连接字符串函数，使其包含3个输入端。
3. 右键单击连接字符串函数顶部的输入端，从快捷菜单中选择**创建»常量**。
4. 输入`//`作为常量。
5. 右键单击连接字符串函数底部的输入端，从快捷菜单中选择**创建»常量**。
6. 输入`/RTAcqData`作为常量，指定写入方端点。
7. 在前面板上添加一个字符串控件。
8. 将字符串输入控件命名为`Target IP Address`。
9. 在**终端IP地址**文本框中，输入RT终端的IP地址。

> **注：** 每次重新打开应用程序时，都需确保在前面板中输入的IP地址与RT终端的IP地址一致。RT终端的IP地址可能会发生变化，具体取决于网络设置。

10. 在程序框图中，将**终端IP地址**输入控件连线至连接字符串函数的中间输入端。

11. 将连接字符串函数的**连接字符串**输出端连线至创建网络流读取方端点函数的**写入方URL**输入端。

从RT终端接收流数据

与网络流写入方端点类似，读取方端点也需要从流中读取单个元素函数和销毁流端点函数来接收数据。

按照以下步骤，在“Windows主程序”中完成网络流读取方端点：

1. 选中选择器标签中的`Timeout`，在事件结构中放置一个从流中读取单个元素函数。
2. 在While循环外部右侧添加一个销毁流端点函数。
3. 删除While循环和**错误输出**显示控件之间的错误连线。
4. 删除事件结构中的错误连线。
5. 依次连接创建网络流读取方端点函数、从流中读取单个元素函数、销毁流端点函数和**错误输出**显示控件之间的错误连线。
6. 将创建网络流读取方端点函数的**读取方端点**输出端连线至从流中读取单个元素函数的**端点输入**输入端。
7. 将从流中读取单个元素函数的**端点输出**输出端连线至销毁流端点函数的**端点输入**输入端。

在前面板中显示数据

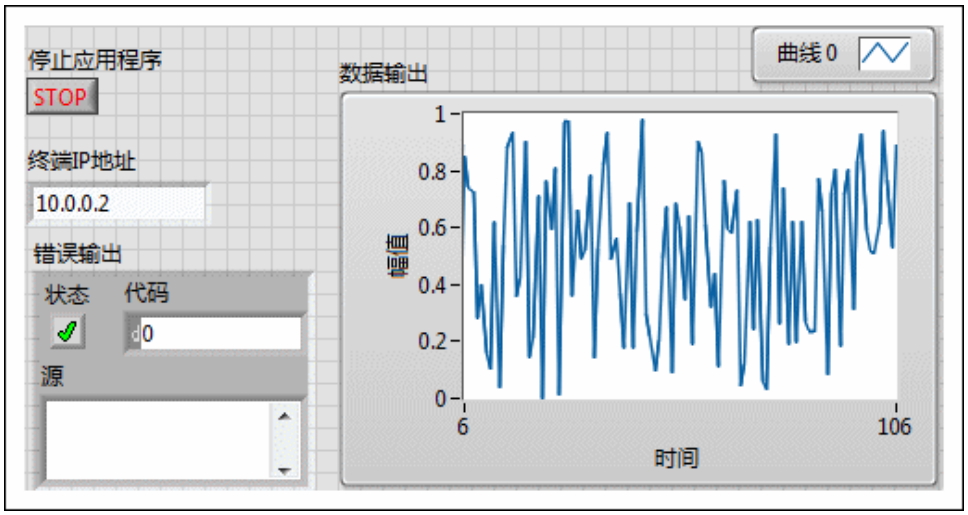
要在UI上显示RT终端的数据，可以使用波形图。

按照以下步骤，在“Windows主程序”前面板上显示RT终端的数据：

1. 右键单击从流中读取单个元素函数的**数据输出**输出端，然后从快捷菜单中选择**创建»显示控件**。
2. 在前面板中，右键单击**数据输出**显示控件，然后从快捷菜单中选择**替换»银色»图形»波形图**。
3. 在程序框图中，右键单击事件结构中的布尔隧道，从快捷菜单中选择**创建»常量**。

4. 指定一个False常量。
5. 保存该VI。
6. 运行“RT主程序”。
7. 运行“Windows主程序”。

前面板应与下图一致。



相关概念：

- 第五部分：发送RT数据至主机
- 教程：创建Real-Time应用程序
- 相关文档

相关文档

以下资源进一步提供了本教程未详细介绍的实时概念信息。

RT终端的类型

访问ni.com/info并输入信息代码RTSystems，了解使用NI硬件和软件生成实时系统的信息。

确定性和预处理循环

关于确定性以及实时循环预计何时开始以确定性方式执行的信息，见***创建确定性应***

用程序。

预分配数组以尽量减少抖动

如果使用数组管理大量数据段，请预分配数组以尽量减少抖动。访问`ni.com/info`并输入`PreAllocateArrays`，了解为确定性循环预分配数组的信息。

相关概念：

- 第六部分：在用户界面上显示RT数据
- 教程：创建Real-Time应用程序
- 创建确定性应用程序

确定性应用程序中的数据共享

共享变量可用于在项目的VI之间或联网VI之间读写数据。共享变量是可在下列情况下传递数据：程序框图两个无法连线的位置之间、RT终端上运行的两个VI之间、不同RT终端上联网的VI之间。Real-Time模块在共享变量中新增了FIFO功能。通过启用共享变量的实时FIFO功能，共享数据不会影响RT终端上VI的确定性。

相关概念：

- 创建带实时FIFO的共享变量

创建带实时FIFO的共享变量

可以创建一个共享变量，然后启用实时FIFO，在RT终端上确定性地本地共享数据，或通过网络与其他终端上的VI共享数据。RT FIFO不支持大小变化的数据类型（例如，簇、字符串和变体）。

按照下列步骤，创建一个共享变量并启用实时FIFO：

1. 右键单击要创建共享变量的RT终端，从快捷菜单中选择**新建»变量**，打开“共享变量属性”对话框。

2. 在**名称**文本框中输入变量的名称。
3. 从**数据类型**下拉菜单中选择变量的数据类型。
4. 从**变量类型**下拉菜单中选择**单进程**或**网络发布**。
5. 在**类别**列表中单击**Real-Time FIFO**，打开“Real-Time FIFO”页。
6. 勾选**启用Real-Time FIFO**复选框。
7. 选择**单元素**或**多元素**，创建单元素FIFO缓冲区或多元素FIFO缓冲区。
8. 如选择**单元素**，选择数组或波形数据类型时，配置数组元素的大小或用于FIFO元素的波形大小。如选择**多元素**，将FIFO缓冲区的大小和元素配置为与“网络”页**使用缓冲**部分的设置一致，或者也可以配置用于FIFO和FIFO元素的自定义大小。
9. 单击**OK**按钮。

生成、部署和调试确定性应用程序

LabVIEW应用程序生成器用于生成独立的实时应用程序，然后从“项目浏览器”窗口或LabVIEW开发环境之外将程序部署至RT终端。使用下表选择应用程序适合的部署方法：

注： 部署独立的实时应用程序之后，可从远程主机上调试应用程序。

使用场景	方法
开发独立的实时应用程序，部署到少量终端上。不打算更新应用程序或其依赖项。	从**项目浏览器**窗口部署。
通过程序将独立的实时应用程序部署至一台或多台终端。	从LabVIEW开发环境之外部署。
开发独立的应用程序，由其他用户来部署应用程序。	
在LabVIEW开发环境之外部署独立实时应用程序，例如，在工厂现场部署。	
预期后续更新独立实时应用程序，以使用更新的依赖项内容。例如，应用程序的	

使用场景	方法
当前版本使用NI-DAQmx，预期升级应用程序以使用NI-DAQmx最新版本的功能，且要确保应用程序总是使用正确的NI-DAQmx版本执行部署。	

相关概念：

- 生成和部署独立的实时应用程序
- 调试独立的实时应用程序
- Real-Time Trace Viewer

生成和部署独立的实时应用程序

按照下列步骤生成独立的实时应用程序，并将其部署至RT终端：

注：

- 部署应用程序之前，确定应用程序最适合哪种部署方法。
- 生成应用程序之后，如要更新**启动VI**或**始终包括**文件，应重新生成应用程序，而不是试图将文件直接复制到RT终端。

1. 打开要生成独立实时应用程序的项目。
2. 右键单击RT终端下的**程序生成规范**，从快捷菜单中选择**新建**»**实时应用程序**，显示“实时应用程序属性”对话框。
3. 填写“实时应用程序属性”对话框“信息”页的下列项：
   a. 在**程序生成规范名称**文本框中输入一个程序生成规范的名称。**程序生成规范**中将显示该名称。同一个RT终端下，程序生成规范的名称必须是唯一的。
   b. 在**目标文件名**文本框中输入独立实时应用程序的名称。独立实时应用程序扩展名必须为 `.rtexe`。
   c. 浏览“应用程序信息”页的其他项。
4. 填写“源文件”页的下列项：

a. 在**项目文件**树中，选择一个VI作为独立实时应用程序的启动VI，即顶层VI。RT终端上电后，启动VI即打开并运行。每个生成的实时应用程序都必须至少有一个启动VI。
b. 单击**启动VI**列表框旁的右箭头按钮，将选中的VI移动至**启动VI**列表框中。
c. 在**项目文件**树中，选择要定义为动态调用VI或支持文件的项，如要通过VI服务器调用的VI或从VI读取的文本文件。
d. 单击**始终包括**列表框旁的右箭头按钮，将选中项移至**始终包括**列表框。

注： **启动VI**或**始终包括**列表框中不能包含多态VI。应使用多态VI的某个实例。

1. 在“目标”页，配置目标设置并添加独立实时应用程序的目标目录。
2. 在“源文件设置”页，编辑独立实时应用程序中单个文件或文件夹的目标和属性。

注： 如果要发布一个使用了共享变量的独立可执行应用程序，那么在LLB或可执行程序中不要包含`.lvlib`文件。设置`.lvlib`文件的**目标**为可执行文件或LLB以外的地址。

3. 在“高级”页，配置独立实时应用程序的高级设置。
4. 在“附加排除项”页，配置设置以减少独立实时应用程序的大小及载入生成的程序时，优化加载时间并降低内存使用量。
5. 在“生成前/后操作”页上，指定生成之前或之后LabVIEW要执行的VI。
6. 在“预览”页，单击**生成预览**按钮以查看生成的独立实时应用程序文件。

注： 为确保预览的准确性，创建或编辑程序生成规范前，应将对VI所作的改动保存在内存中。

7. 从“项目浏览器”窗口或LabVIEW开发环境之外部署应用程序。

注： 如果修改了代码并想重新部署应用程序，请右键单击程序生成规范，从快捷菜单中选择**取消部署**，以删除先前部署的文件。此操作

> 会删除先前部署到终端的所有文件，而不仅仅是单个程序生成规范部署的文件。为避免冲突，在使用不同的系统传输文件（如通过SystemLink进行安装）之前，请使用此选项从终端删除已部署的文件。

从项目浏览器窗口部署

按照下列步骤，生成一个独立应用程序并从“项目浏览器”窗口将该应用程序部署至RT终端：

1. 使用“实时应用程序属性”对话框指定应用程序的设置。
2. 单击**生成**按钮。程序生成规范的名称出现在“项目浏览器”窗口RT终端下的**程序生成规范**树中，LabVIEW会将应用程序放置在“实时应用程序属性”对话框“目标”页上指定的目录中。
3. 右键单击程序生成规范，从快捷菜单中选择**设置为启动项**。

> **注：** 可为RT终端下的独立实时应用程序的不同配置，生成多个程序生成规范。但只能将一个独立实时应用程序设置为RT终端的启动程序。将程序生成规范设置为RT终端的启动应用程序后，LabVIEW将在**项目浏览器**窗口的程序生成规范图标周围显示绿色外框。

4. 右键单击程序生成规范，从快捷菜单中选择**部署**，将应用程序部署至终端。
5. 右键单击RT终端，选择**工具»重启**，重启RT终端，运行独立的实时应用程序。

> **注：** 必须将应用程序设置为启动并重启RT终端，才能运行独立实时应用程序。

从LabVIEW应用环境之外部署

按照下列步骤，从LabVIEW开发环境之外部署独立的实时应用程序：

1. 使用“实时应用程序属性”对话框指定应用程序的设置。

> **注：** 必须在“源文件”页指定至少一个启动VI。

2. 在“组件定义”页上，勾选**创建组件定义文件(.cdf)并指定依赖关系**复选框。
3. （可选）如要为应用程序指定一个版本号，取消勾选**版本号**面板的**自动递增**复选框，为应用程序指定一个版本号。
4. (PXI)在**必需的软件组件**列表中，勾选需要随应用程序一并安装的每个软件组件旁的复选框。在**软件版本**下拉菜单中，从每个软件组件的可用版本中进行选择。(CompactRIO)在**必需的软件组**列表中，选择需要随应用程序一并安装的软件组。在**附加软件**列表中，勾选要安装的附加软件旁的复选框。（可选）取消勾选**按组显示软件**复选框，查看可用软件组件的列表。

> **注：** LabVIEW会始终指定作为应用程序依赖项运行的LabVIEW Real-Time模块的版本。

5. 单击**生成**按钮。LabVIEW生成应用程序和组件定义文件(`.cdf`)，该文件定义了应用程序及其依赖项。LabVIEW将`.cdf`置于`National Instruments\RT Images\User Components`目录下。
6. 生成完成后，在**生成状态**对话框中检查生成好的应用程序位置，单击**完成**按钮。
7. 如要将应用程序及其依赖项安装至少量终端，可使用NI Measurement & Automation Explorer (MAX)；如要安装至大量终端，可使用安装VI的安装启动实例。关于使用MAX安装独立应用程序的步骤信息，见Measurement & Automation Explorer帮助中的安装启动应用程序。

关于从LabVIEW开发环境之外部署独立实时应用程序的详细信息，见ni.com上的“使用应用程序组件部署LabVIEW Real-Time应用程序”。

调试独立的实时应用程序

可在远程主机上调试通过LabVIEW应用程序生成器创建的独立实时应用程序。

按照下列步骤调试一个独立的实时应用程序：

1. 在Real-Time应用程序“属性”对话框的**高级**页，勾选**启用调试**复选框，在独立实时应用程序的程序生成规范中启用调试。
2. 右键单击独立实时应用程序的程序生成规范，从快捷菜单中选择**生成**，生成应用程序。
3. 重启RT终端，运行独立的实时应用程序。
4. 在**项目浏览器**窗口中，选择**操作»调试应用程序或共享库**，显示“调试应用程序或共享库”对话框。
5. 在**机器名或IP地址**文本框中输入RT终端的IP地址。单击**刷新**按钮，查看RT终端上可调试独立实时应用程序的列表。
6. 选择要调试的实时应用程序。
7. 单击**连接**按钮，打开待调试启动VI的前面板窗口。通过启动VI的程序框图，使用探头、断点和其他调试技术调试应用程序。
8. 调试关闭后，关闭启动VI，同时也关闭了RT终端的实时应用程序。然后重启RT终端，重新启动独立实时应用程序。如要不关闭启动VI并断开连接，右键单击启动VI的前面板窗口，从快捷菜单中选择**远程调试»退出调试服务**。

相关概念：

- 生成和部署独立的实时应用程序

与RT终端的VI前面板交互

可通过下列方法与RT终端VI的前面板交互：

- 启用RT终端上的嵌入式UI – 允许直接连接显示设备至RT终端并与图形化工作环境交互，包括运行在RT终端上的VI的前面板。
- 通过远程前面板访问RT终端VI – 允许从远程计算机查看和控制运行在RT终端上的VI的前面板。一个时间点上，VI的前面板只能由一个用户控制，但可被多个不同位置的用户同时查看。

相关概念：

- 启用RT终端上的嵌入式UI

- 通过远程前面板访问RT终端VI

通过嵌入式UI访问RT终端VI

关于通过嵌入式UI访问RT终端VI的详细信息，请参考下列主题：

- 启用RT终端上的嵌入式UI
- 通过嵌入式UI查看和控制RT终端VI的前面板
- 自定义前面板显示嵌入式UI的方式

请参考与RT终端VI的前面板交互，了解可用于显示RT终端VI前面板的不同方法。

相关概念：

- 启用RT终端上的嵌入式UI
- 通过嵌入式UI查看和控制RT终端VI的前面板
- 自定义前面板显示嵌入式UI的方式

启用RT终端上的嵌入式UI

按照下列步骤，启用RT终端的嵌入式UI。如启用了嵌入式UI，用户可在RT终端上访问桌面环境，包括浏览文件、访问接线端和设置以及与运行在RT终端上的VI前面板交互。

> **注：** 某些RT终端不支持嵌入式UI。请参考具体的RT终端硬件文档，了解该终端是否支持嵌入式UI。

1. 连接显示设备至RT终端。也可连接其他UI设备，如鼠标和键盘。
2. 将RT终端连接至和主机相同的子网。关于连接RT终端的信息，见Measurement & Automation Explorer帮助中的连接至LabVIEW RT终端主题。在NI Measurement & Automation Explorer (MAX)中单击**帮助»MAX帮助**，可打开Measurement & Automation Explorer帮助。
3. 如果RT终端上安装了推荐的软件组（例如，NI CompactRIO 版本 - 版本日期），

请继续执行步骤4。否则，请按照以下步骤安装推荐的软件组。

a. 在MAX中，展开配置目录树中的**远程系统**，然后展开RT终端。
b. 在配置目录树中选择**软件**。
c. 单击工具栏上的**添加/删除软件**图标，打开LabVIEW Real-Time软件向导。
d. 选择向导中显示的推荐软件组并单击**下一步**。

MAX将显示安装进程，然后重启终端。

4. 在MAX中启用嵌入式UI复选框:
   a. 展开配置目录树中的**远程系统**，然后选择RT终端。
   b. 在**系统设置**选项卡的**启动设置**中勾选**启用嵌入式UI**复选框。
   c. 单击**保存**，然后在提示重启终端时单击**是**。

> **注：** 也可使用NI基于Web的配置和监控启用嵌入式UI复选框。

终端重启并显示嵌入式UI桌面。

> **注：** NI使用Xfce创建用于RT终端的嵌入式UI。关于使用和配置Xfce的信息，见Xfce网站上的Xfce帮助。

相关概念：

- 通过嵌入式UI查看和控制RT终端VI的前面板
- 自定义前面板显示嵌入式UI的方式

通过嵌入式UI查看和控制RT终端VI的前面板

通过连接至RT终端的显示设备可查看运行在RT终端上的VI前面板。此外，还可传输前面板的控制至RT终端，以便通过直接连接至RT终端的输入设备与前面板对象交互。

查看前面板

根据下列步骤，通过嵌入式UI查看前面板：

1. 连接显示设备至RT终端。也可连接其他UI设备，如鼠标和键盘。
2. 启用RT终端上的嵌入式UI。
3. 添加RT终端至LabVIEW项目。
4. 在**项目浏览器**窗口，添加VI至RT终端下。
5. 在**项目浏览器**窗口右键单击RT终端下的VI，选择**部署**。
6. 在主机上运行VI。

VI前面板显示在连接至RT终端的显示设备上。虽然现已可查看前面板，但仍然无法使用连接至RT终端的输入设备与前面板对象交互。请按照下节中的步骤，将对前面板的控制从主机传递至RT终端。

控制前面板

部署VI至RT终端后，请根据下列步骤，将对前面板的控制传递至嵌入式UI：

1. 根据需要，从主机运行VI以便在嵌入式UI上显示前面板。
2. 在**项目浏览器**窗口右键单击RT终端，选择**断开**。

现在可通过嵌入式UI与前面板输入控件、显示控件以及菜单项交互，同时还可停止和运行VI。

提示 如将RT终端VI生成到独立的应用程序并部署至RT终端，也可使用嵌入式UI查看和控制RT终端VI的前面板。

相关概念：

- 启用RT终端上的嵌入式UI
- 添加RT终端到LabVIEW项目
- 生成和部署独立的实时应用程序
- 自定义前面板显示嵌入式UI的方式

自定义前面板显示嵌入式UI的方式

对于某些应用，自定义前面板显示嵌入式UI的方式后可能更为方便。这些自定义对

于创建仅专注于前面板及其功能的终端用户体验十分有用。例如，可能需要全屏显示前面板以隐藏桌面、从屏幕删除不需要的LabVIEW对象（如标题栏和菜单），或者阻止终端用户从前面板切换出来。

设置全屏显示前面板

根据下列步骤，在嵌入式UI上全屏显示前面板，使其与连接至RT终端的显示设配分辨率相匹配。

1. 在LabVIEW中选择**文件»VI属性**，打开**VI属性**对话框。
2. 在**类别**下拉菜单中选择**窗口大小**。
3. 指定**宽度**和**高度**，使其与连接至RT终端的显示设备的屏幕分辨率匹配。
4. 在**类别**下拉菜单中选择**窗口运行时位置**。
5. 从**位置**下拉菜单中选择**居中**。
6. 禁用**使用当前前面板大小**复选框，指定**宽度**和**高度**，使其与连接至RT终端的显示设备的屏幕分辨率相匹配。
7. 单击**确定**以保存改动。

在RT终端上运行和查看VI时，前面板将与显示设备的分辨率相匹配。

从前面板删除LabVIEW对象

按照下列步骤，从VI前面板删除不需要的LabVIEW对象（如标题栏和菜单）：

1. 在LabVIEW中选择**文件»VI属性**，打开**VI属性**对话框。
2. 从**类别**下拉菜单中选择**窗口外观**。
3. 启用**自定义**按钮并单击**自定义**，打开**自定义窗口外观**对话框。
4. 根据应用程序需求指定要从前面板删除的选项。
5. 单击**确定**，关闭**自定义窗口外观**对话框。
6. 单击**确定**以保存改动。

在RT终端上运行和查看VI时，前面板将显示一组自定义的对象。

从显示中删除嵌入式UI元素

按照下列步骤，从显示中删除不需要的嵌入式UI元素。嵌入式UI元素包含打开Xfce文件管理器、终端仿真程序和显示设置的桌面图标。

1. 在RT终端VI程序框图的实时循环外放置一个“执行系统命令”VI。
2. 指定执行系统命令VI运行以下命令：`hidepanel`

在RT终端上运行和查看VI时，Xfce元素将不会显示在屏幕上。

提示 如需恢复嵌入式UI元素，可将执行系统命令VI的**等待直到结束？**参数设为FALSE，并指定以下命令：`showpanel`

相关概念：

- 启用RT终端上的嵌入式UI
- 通过嵌入式UI查看和控制RT终端VI的前面板

通过远程前面板访问RT终端VI

通过LabVIEW，使用远程前面板查看和控制RT终端上所运行VI的前面板。一个时间点上，VI的前面板只能由一个用户控制，可被多个用户同时查看。

注： 必须启用RT终端上的Web服务器，并添加远程计算机至终端的访问列表，才能从远程计算机查看或控制VI的前面板。

关于通过远程前面板访问RT终端VI的详细信息，请参考下列主题：

- 启用RT终端VI的远程前面板连接
- 向RT终端浏览器访问列表添加项
- 远程查看和控制RT终端VI的前面板

请参考与RT终端VI的前面板交互，了解可用于显示RT终端VI前面板的不同方法。

相关概念：

- 远程查看和控制RT终端VI的前面板（Real-Time模块）
- 启用RT终端VI的远程前面板连接
- 向RT终端浏览器访问列表添加项

启用RT终端VI的远程前面板连接

可通过计算机使用LabVIEW查看RT终端上所运行VI的前面板。必须在RT终端上启用Web服务器，将VI添加至**Web服务器可见VI**列表，将计算机添加至终端的访问列表，允许用户远程访问VI前面板。

按照下列步骤，启用RT终端的Web服务器，并将VI添加至**Web服务器可见VI**列表：

1. 在“项目浏览器”窗口右键单击RT终端，并从快捷菜单中选择**属性**，显示“RT终端属性”对话框。
2. 从**类别**列表中选择**Web服务器**。
3. 勾选**启用远程前面板服务器**复选框，启用Web服务器。
4. 在**根目录**文本框中输入要用作Web服务器根目录的目录。Web服务器根目录是Web服务器文件系统的顶层目录。
5. 在**可见VI**部分，输入要添加至**可见VI**列表的RT终端存储器中VI的文件名，然后单击**添加**按钮。

   > **注：** 如在启用Web服务器前部署VI到RT终端，必须重新部署VI以访问前面板。

6. 单击**确定**按钮，关闭RT终端的**属性**对话框。

相关概念：

- 向RT终端浏览器访问列表添加项
- 远程查看和控制RT终端VI的前面板（Real-Time模块）

向RT终端浏览器访问列表添加项

必须在RT终端上启用Web服务器，将计算机添加至终端的访问列表之后，才能通过LabVIEW查看和控制终端上所运行VI的前面板。

按照下列步骤，将计算机添加到RT终端的访问列表：

1. 在“项目浏览器”窗口右键单击RT终端，并从快捷菜单中选择**属性**，显示“RT终端属性”对话框。
2. 从**类别**列表中选择**Web服务器**。**Web服务器**页在右窗格显示。
3. 滚动页面查看**浏览器访问**部分。
4. 单击**添加**按钮，在**浏览器访问列表**中添加新的项。
5. 在**浏览器地址**文本框中输入允许访问的计算机IP地址。可输入域名（如`mydomain.com`），或在IP地址中使用通配符（如`*.164.123.123`），允许访问整个域。

   > **注：** 如Web服务器所运行的RT终端不能访问DNS服务器，不要在**浏览器访问列表**中使用域名。

6. 选择**允许查看和控制**、**允许查看**或**拒绝访问**，设置IP地址的访问权限。如允许查看和控制前面板，项的左边将出现两个绿色的勾选标志；如只允许查看前面板，将出现一个绿色勾选标志；如拒绝访问，将出现一个红色的X。

   > **注：** 如项的名称旁没有绿色的勾选标志或红色的X，则该项中存在语法错误。编辑项或单击**删除**按钮将项删除。

7. 单击**确定**按钮，关闭RT终端**属性**对话框并保存更改。

相关概念：

- 启用RT终端VI的远程前面板连接

远程查看和控制RT终端VI的前面板（Real-Time模块）

用户可通过LabVIEW，查看和控制RT终端上所运行VI的前面板。如超过一个客户端请求控制VI，第一个连接的客户端将获得控制权。RT终端根据请求顺序，在第一个客户端释放前面板控制权之后，将控制权赋予下一个客户端。

注： 必须启用RT终端上的Web服务器，将VI添加至可见VI列表，并将远程计算机添加至浏览器访问列表，才能从远程计算机上查看和控制VI的前面板。

按照下列步骤，从远程计算机的LabVIEW查看和控制RT终端VI的前面板：

注： 必须将主机添加至浏览器访问列表，使用远程前面板访问RT终端上的VI。

1. 在“项目浏览器”窗口选择**操作»连接远程前面板**，打开“连接远程前面板”对话框。
2. 在**服务器地址**文本框中输入要连接的RT终端的IP地址或名称。
3. 在**VI名称**文本框中输入要查看的VI前面板的VI名称。
4. 在**端口**文本框中输入RT终端Web服务器的HTTP端口。默认值为80。
5. 如需立刻请求前面板的控制权，勾选**请求控制**复选框。
6. 单击**连接**按钮。主机上会显示VI的前面板，允许用户查看和控制。
7. 如未勾选**请求控制**复选框，右键单击前面板，从快捷菜单中选择**请求VI控制权**，以请求VI控制权。

关闭LabVIEW窗口可完全关闭RT终端上Web服务器的连接。但是，如要释放前面板控制权，却仍保持连接，可以查看前面板，右键单击前面板，从快捷菜单中选择**远程前面板客户端»释放VI的控制权**。

相关概念：

- 远程查看和控制RT终端VI的前面板（Real-Time模块）

- 启用RT终端VI的远程前面板连接
- 向RT终端浏览器访问列表添加项

NI Linux Real-Time操作系统

LabVIEW Real-Time模块在NI Linux Real-Time操作系统上执行VI。但使用上述实时操作系统时必须考虑特殊的限制条件。

特殊注意事项

使用NI Linux Real-Time终端时应牢记下列事项。也可访问`ni.com/info`并输入`Linux`，了解NI Linux Real-Time操作系统的详细信息。

常规

- 默认情况下，FTP服务器已禁用，且不能在安全模式使用FTP服务器。建议使用WebDAV作为文件传输机制，以增强安全性。
- 支持虚拟内存，即表明：
  - 如进程崩溃，可重启进程而无需重启终端。
  - 监视最大连续内存块不能为NI Linux Real-Time终端提供有用/必需的信息。
- 支持执行系统命令VI。
- 时间区更改无需终端重启。

文件系统和目录结构

- 文件系统需区分大小写。
- 目录路径与Linux格式一致。例如，路径使用正斜杠并包含驱动器字符。
- NI Linux Real-Time强制文件和文件夹访问权限。关于这些权限的信息，见知识库文章***Real-Time终端上的文件路径使用***。
- RTOS在终端重启时删除临时文件夹`/tmp`中的内容。临时文件夹从RAM分配最大64 MB的空间。
- NI Linux Real-Time终端上不存在`c:\ni-rt`目录。关于NI Linux Real-Time终端上目录映射的信息，见知识库文章***Real-Time终端上的文件路径使用***。
- 强烈不建议直接修改`ni-rt.ini`配置文件。直接修改该文件可能会导致系统不正常工作。

> **注：** 有些错误诊断流程可能会要求手动修改该文件。修改该文件时，请谨慎小心。

Web服务

- 默认情况下，已启用WebDAV。
- 默认情况下，已启用SSL。
- NI基于Web的配置和监控在安全模式下可用。
- 仅可通过IP地址（DHCP、 Link-local或静态）配置终端，mDNS用于终端检测。
- 无法在NI Linux Real-Time终端上通过NI基于Web的配置和监控使用Console Out。可使用串口连接NI Linux Real-Time终端至计算机，查看Console Out控制台输出。关于如何使用Console Out的详细信息，见知识库文章***cRIO、sbRIO和cFP控制器的Console Out***。

授权

- 通过NI基于Web的配置和监控、SSH或串口连接登陆终端。关于在RT终端上启用SSH的详细信息，见相关硬件文档。

> **注：** 建议您通过SSH或串口连接登陆时设置密码。

- 如要重置忘记的管理员密码，必须物理访问终端并将其重置为出厂默认值。请联系National Instruments获得关于该过程的帮助。
- 终端创建一个`admin`用户，其相当于Linux系统中的`root`用户。

> **警告** 请勿在“NI基于Web的配置和监控”中创建与系统账户同名的用户。例如，请勿创建`root`或`ssh`用户名。执行此操作将覆盖系统帐户。

- SSH支持将公共密钥作为授权的替代方式。

安全

- 访问ni.com/info并输入RTSecurity，及时获取NI产品相关的安全性信息。

不支持的功能

LabVIEW Real-Time模块不支持运行在RT终端上VI的特定LabVIEW功能。例如，如尝试在RT终端上部署和运行具有不支持功能的VI，VI可能仍能执行。但是，不支持的部分不执行，并返回标准LabVIEW错误代码。LabVIEW Real-Time模块不支持运行NI Linux Real-Time RTOS的RT终端的下列功能：

- ActiveX VI
- NI TestStand VI（基于ActiveX）
- .NET VI
- 报表生成VI
- Windows注册表访问VI
- 报表Express VI（基于不支持的报表生成VI）
- 图形与声音VI
- Database Connectivity VI
- 菜单函数
- 光标VI
- IrDA函数
- 文件系统Web服务LabVIEW API
- 调用库函数节点，访问除NI Linux Real-Time之外的其他操作系统API
- 交互式打开文件对话框的打开/创建/替换文件函数
- 执行属性页的**调用时清空显示控件**选项
- 创建直方图Express VI的**自动配置**选项
- 特定高级TDMS函数
- **监测缓冲区分配**窗口
- 最大可用内存块属性节点－该属性节点不会提供NI Linux Real-Time终端的有用信息，且一旦使用将出现错误。请使用“释放物理内存属性”节点。
- **保持触发直到释放**和**保持转换直到释放**布尔机械动作 – 关于这些布尔机械动作

在RT目标上未正常运行的原因，见知识库文章***Real-Time终端不支持布尔“直到释放”机械动作***。

修改RT终端VI的前面板对象

VI或独立应用程序运行在无用户界面的RT终端，或禁用嵌入式UI时，无法执行修改前面板的VI。例如，不能通过属性节点改变或读取前面板对象的属性，因为RT终端上运行的VI没有前面板。VI仍在RT终端上运行，但是前面板对象不受影响，返回一个错误。在一些情况下，可创建与RT终端的前面板连接，使用RT终端上不支持的LabVIEW功能。右键单击项目浏览器窗口的RT终端，从快捷菜单中选择**连接**，打开与终端的前面板连接。

只有在前面板连接时，RT终端上的下列功能才有效：

- 前面板属性节点和控件引用。

  注： RT终端异步更新属性节点的值和控件引用，表示属性节点或控件引用不一定包含控件最新的值。

- 对话框VI和函数
- VI服务器前面板函数

注： 部分使用RT终端不支持的VI和函数的LabVIEW范例VI。

Real-Time模块和Express VI的注意事项

LabVIEW Express VI使用交互式的对话框改进了VI的易用性，并减少了测量应用所需的编程时间。Express VI在执行过程中会占用额外的系统开销，所以，不要在确定性应用程序或大处理量的应用程序中使用Express VI。建议使用标准VI开发实时应用程序。

调试重入VI

在RT终端上执行的重入VI的副本中，无法使用LabVIEW调试工具。LabVIEW通过前面板区分重入VI的各个副本，RT终端上的VI没有前面板，所以LabVIEW无法打开重入VI副本进行调试。但是，将应用程序部署到RT终端之前，可在Windows操作系统上运行和部署应用程序。

相关概念：

- LabVIEW Real-Time模块平台
- 实时终端

相关信息：

- Real-Time终端上的文件路径使用
- cRIO、sbRIO和cFP控制器的Console Out
- Real-Time终端不支持布尔“直到释放”机械动作

Real-Time Trace Viewer

Use the Real-Time Trace Viewer and the Real-Time Trace Viewer VIs to capture the timing and execution data of VI and thread events for applications running on an RT target. The Real-Time Trace Viewer displays the timing and event data, or trace session, on the host computer. In LabVIEW, select **Tools»Real-Time Module»Trace Viewer** to display the Real-Time Trace Viewer. For information about developing deterministic applications using LabVIEW, refer to ***Creating Deterministic Applications*** and ***Building, Deploying, and Debugging Deterministic Applications***.

注： You can use the Real-Time Trace Viewer 2.0 to load trace files created using version 7.1 or higher of the LabVIEW Real-Time Module.

相关概念：

- 创建确定性应用程序
- 生成、部署和调试确定性应用程序

Real-Time Trace Viewer 2.0 Features and Changes

Refer to the `readme_ExecTrace` file in the `Real-Time Execution Trace Toolkit 2.0\readme` directory for information about known issues with the Real-Time Trace Viewer 2.0.

The Real-Time Trace Viewer 2.0 includes the following new features to help you more quickly and easily view and analyze trace sessions.

注： You can use the Real-Time Trace Viewer 2.0 to load trace files created using version 7.1 or higher of the LabVIEW Real-Time Module.

- **Multiple-CPU Trace Analysis**—The Real-Time Trace Viewer 2.0 supports trace sessions involving multiple CPUs, and includes new **Highlight CPU Mode** options

that you can use to highlight all thread activity that executed on a particular CPU.

- **Faster Trace Rendering**—The Real-Time Trace Viewer 2.0 features a new drawing engine that renders traces quickly, so there is less delay when you change the zoom level or move to a different section of the trace.
- **Improved Interface**—The Real-Time Trace Viewer 2.0 includes the following new interface components.
  - **Flag Configuration Dialog Box**—This dialog box replaces the functionality of the **Create Custom Event Flag** dialog box and the **Customize Event Flags** dialog box. Use the Flag Configuration dialog box to configure display settings for event flags in a trace session. The options you configure carry over to future sessions, so you do not lose your configuration settings.
  - **Frame Resizing**—You now can horizontally resize the frames containing VI and thread names. You also can vertically resize the frames containing the VI and thread trace views.
  - **Multi-Item Selection**—You now can select multiple VIs or threads at once.
  - **Trace Sorting**—You now can right-click the list of thread or VI names and select to sort by name, sequence, or amount of activity. You also can drag and drop list items to create a logical order. The list automatically scrolls as you drag.
  - **Pan Tool**—Use this new tool to pan horizontally across the active trace session.
  - **Improved Region Selection**—You now can zoom in and out while selecting a measurement region. You also can define a measurement or zoom region that spans from a point defined in the VI view to a point defined in the Thread view, and vice-versa.
  - **Shortcuts**—The Real-Time Trace Viewer 2.0 includes the following new keyboard shortcuts.
    - **<Ctrl-D>**—Opens the **Flag Configuration** dialog box.
    - **<spacebar>**—Cycles through mouse tools.
    - **<Ctrl-+>**—Zooms in toward the center of the current view.
    - **<Ctrl-->**—Zooms out from the center of the current view.
    - **<Ctrl-E>**—Jumps to a view of the entire trace.
    - **<left arrow>**—Pan left in the active trace session.
    - **<right arrow>**—Pan right in the active trace session.

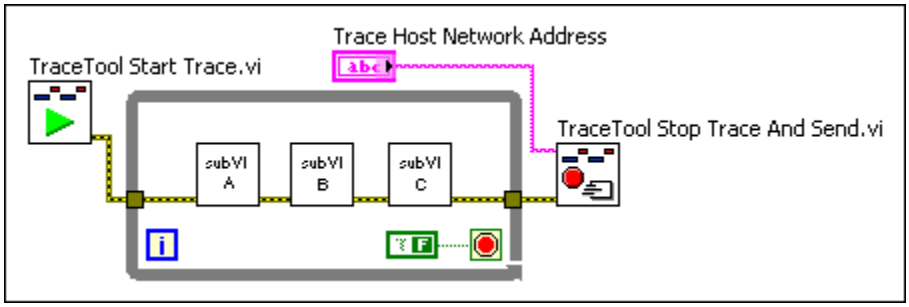
Capturing Trace Sessions

The Real-Time Trace Viewer displays trace sessions you capture using the Real-Time Trace Viewer VIs. You must add Real-Time Trace Viewer VIs to the block diagram of an

application running on the RT target to start and stop the logging of timing and event data from VIs and operating system threads.

Starting a Trace Session

The TraceTool Start Trace VI starts logging event data on the RT target. The following block diagram shows a typical use of the Real-Time Trace Viewer VIs.



The **Buffer Size** input of the TraceTool Start Trace VI sets the size of the memory buffer that stores event data for the application on the RT target. Set the size of the memory buffer large enough to store all the event data you need to collect. If you reach the memory buffer limit when logging event data, the TraceTool Start Trace VI overwrites the oldest data in the buffer.

注： You cannot change the size of the memory buffer on the RT target after you run the TraceTool Start Trace VI. You must reboot the RT target to resize the memory buffer.

The amount of time you can capture in a trace session depends on the size of the memory buffer and the type of events that you choose to capture. You can disable the logging of VI, thread, and detailed events. The **Thread Tracing?** and **VI Tracing?** inputs of the TraceTool Start Trace VI specify whether to log thread and VI events. The **Detailed Tracing?** input of the TraceTool Start Trace VI specifies whether to log detailed events, which the Real-Time Trace Viewer uses to display system events.

Stopping a Trace Session

The TraceTool Stop Trace And Send VI stops logging event data on the RT target and then sends the trace session to the Real-Time Trace Viewer running on the host

computer. Specify the IP address of the host computer running the Real-Time Trace Viewer using the **Trace Host Network Address** input of the TraceTool Stop Trace and Send VI. The preceding block diagram shows a typical use of the TraceTool Stop Trace and Send VI.

> **注：** The Real-Time Trace Viewer must be running on the host computer to receive the trace session.

The preceding block diagram uses error clusters to define the dataflow and force the TraceTool Start Trace VI to execute before VIs in the application. The TraceTool Stop Trace and Send VI executes after the application completes. You can start and stop the trace session in a subVI of an application to target a specific section of code and to conserve space in the memory buffer.

You also can use the TraceTool Stop Trace and Save VI to stop logging event data and save the trace session to a file on the RT target. You can transfer the trace session file to the host computer using the TraceTool Load Trace and Send VI. You then can work with the trace session on the host computer.

Starting and Stopping a Trace Session at a Specific Location

You can start and stop a trace session at the beginning and end of an application. However, it might be useful to start and stop a trace session in a specific location of the application depending on the situation.

- Capturing a single subVI—You can start and stop the trace session around a subVI if you want to log only the execution of a single VI. The trace session captures events from other VIs running on the target, but the trace session begins before the subVI executes and ends immediately after.

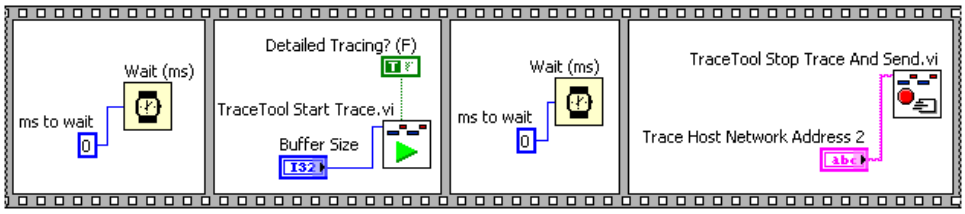
> **注：** The TraceTool Start Trace VI captures the VI events of other VIs that execute in parallel if they start executing after the TraceTool Start Trace VI executes.

- Capturing a defined event—You can start logging at the beginning of an application and then stop logging when an event occurs. In the following example, the Trace

Tool Stop Trace and Send VI executes when the Case structure receives a TRUE value. The TraceTool Start Trace VI might overwrite old event data in the memory buffer several times during execution, but the trace session always retains the most recent event data.

- Capturing a specific time span—You can start and stop the trace session in an independent VI to log a specific time slice of an application. Add the Wait (ms) function with the appropriate time value to the VI and then start the trace session. Add another Wait (ms) function with the appropriate time and then stop the trace session. For example, the following VI starts and stops a trace session to capture all events on the RT target that occur during a 10000 ms time span that begins 5000 ms after you start the VI. You can start the VI when you start the application you want to analyze.

注： This method might not capture the VI events of VIs that begin to execute before the TraceTool Start Trace VI executes.



相关概念：

- Viewing and Analyzing VI Events
- Viewing System Events

Creating an Appropriate Memory Buffer

The **Buffer Size** input of the TraceTool Start Trace VI sets the size of the memory buffer

that stores VI and thread events. The TraceTool Start Trace VI allocates a memory space for the buffer the first time you run the VI. The TraceTool Start Trace VI reuses the same memory space for the memory buffer in subsequent iterations. The **Actual Buffer Size** output of the TraceTool Start Trace VI returns the actual size of the memory buffer if the real-time operating system cannot allocate the requested **Buffer Size**.

> **注：** You cannot change the size of the memory buffer on the RT target after you run the TraceTool Start Trace VI. You must reboot the RT target to resize the memory buffer.

You must create an appropriately sized memory buffer to capture all event data. Buffer size might be the reason why some VIs listed in the VI view do not appear in the trace session. When the buffer fills, the TraceTool Start Trace VI overwrites the least recent event data with the most recent event data.

If you reach the memory buffer limit, the TraceTool Start Trace VI overwrites the oldest data in the buffer. Therefore, the trace session always contains the latest event data. You can use the following techniques to improve the efficiency of the memory buffer.

- Increase the size of the memory buffer—The default size of the memory buffer is 250 KB. You can use the **Buffer Size** input of the TraceTool Start Trace VI to set the size of the memory buffer equal to the largest contiguous block of memory available on the RT target.
- Disable unused events—Use the **Detailed Tracing?**, **Thread Tracing?**, and **VI Tracing?** inputs on the TraceTool Start Trace VI to disable unnecessary events and preserve memory in the buffer.

To ensure that data was not overwritten, use the **Zoom Extents** tool from the Real-Time Trace Viewer toolbar to display the entire trace session. The TraceTool Start Trace VI is always the first VI event logged to the memory buffer. If the memory buffer reaches the buffer size limit, the buffer overwrites the oldest events. If the TraceTool Start Trace VI is the first event that appears in the trace session, the memory buffer did not overwrite any events.

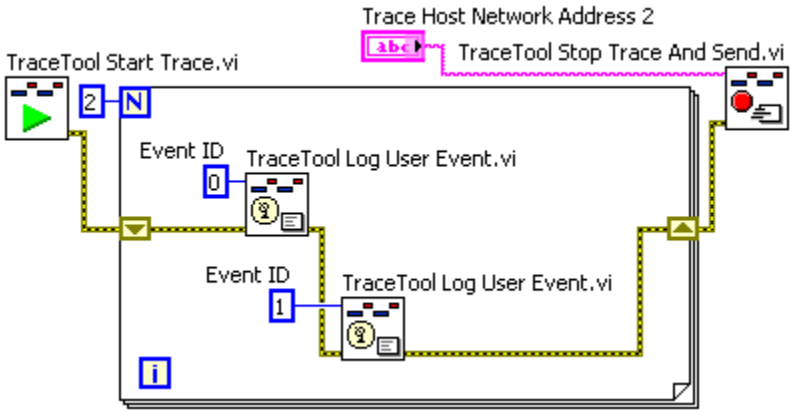
Logging User Events

You can log user events in a trace session to show the execution of specific sections of an application.

> **注：** You must enable thread tracing using the TraceTool Start Trace VI when you start the trace session to display user events.

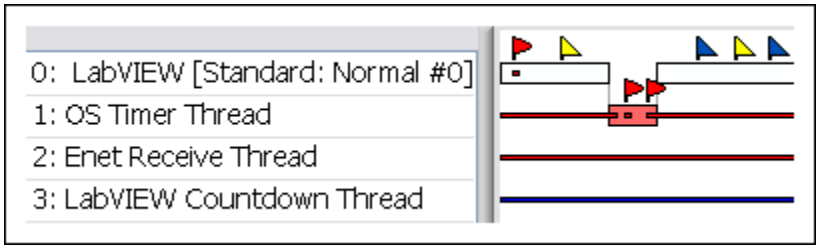
Use the TraceTool Log User Event VI to log a user event in the trace session. You then can use the Flag Configuration dialog box to configure the display settings of the user flag, which appears in the Thread view. The **Event ID** input of the TraceTool Log User Event VI corresponds to the **User Event** number (0 - 254) in the **Flag Configuration** dialog box.

The following block diagram shows a typical use of the TraceTool Log User Event VI. The **Event ID** input of the first TraceTool Log User Event VI receives the event code 0 and the second receives the event code 1. The trace session logs any occurrence of the user events and the Real-Time Trace Viewer marks the occurrences with user event flags. You can configure the style and color of the user event flags in the Flag Configuration dialog box to make the flags easy to recognize in the Thread view.



The following illustration shows the user events logged in the VI shown above. The VI executed in a LabVIEW thread running at normal priority and assigned to the Standard execution system. The user event with Event ID 0 is configured to display a yellow flag and the user event with Event ID 1 is configured to display a blue flag. The user event flags in the Thread view show the exact time when the TraceTool Log User Event VI

executes in the application.



Viewing Trace Sessions

The Real-Time Trace Viewer displays a trace session graphically using the VI and Thread views. When the Real-Time Trace Viewer loads a trace session, it displays the entire trace session in the VI and Thread views. To display only the Thread Events view or only the VI Events view in the Real-Time Trace Viewer, use the **View** menu to deselect the view that you want to hide.

The Real-Time Trace Viewer displays VI and thread events in different colors to distinguish the execution priority of each event. The following table lists the color associated with each LabVIEW priority.

Priority	Color
Time Critical	Red
High	Dark Pink
Above Normal	Light Pink
Normal	White

Background	Blue
Subroutine	Black

You can zoom in and out of a region of the trace session or measure the timing of a specific range in the trace session using the following tools available on the Real-Time Trace Viewer toolbar:

	**Measure**—Displays the length of a region you select. Click once in the VI or Thread view to select a starting point. You then can press the <Esc> key to redefine the starting point or click again to define the end of the region. The length of the region appears in a tip strip.
	**Zoom Region**—Zooms into a region you select. Click once in the VI or Thread view to select a starting point. You then can press the <Esc> key to redefine the starting point or click again to define the end of the region.
	**Zoom In**—Zooms into the VI and Thread views. Click on the VI or Thread view with the Zoom In tool to zoom into the currently displayed range. You also can zoom in by pressing the <Ctrl-Z> keys.
	**Zoom Out**—Zooms out of the VI and Thread views. Click on the VI or Thread view with the Zoom Out tool to zoom out of the currently displayed range. You also can zoom out by pressing the <Ctrl-Shift-Z> keys.
	**Pan**—Scrolls the trace view left or right as you drag the mouse. You also can pan by pressing the <left arrow> and <right arrow> keys.
	**Zoom Extents**—Displays the entire trace session in the VI and Thread views. You also can display the entire trace session by pressing the <Ctrl-E> keys.

注： You can scroll through the above tools by pressing <spacebar>.

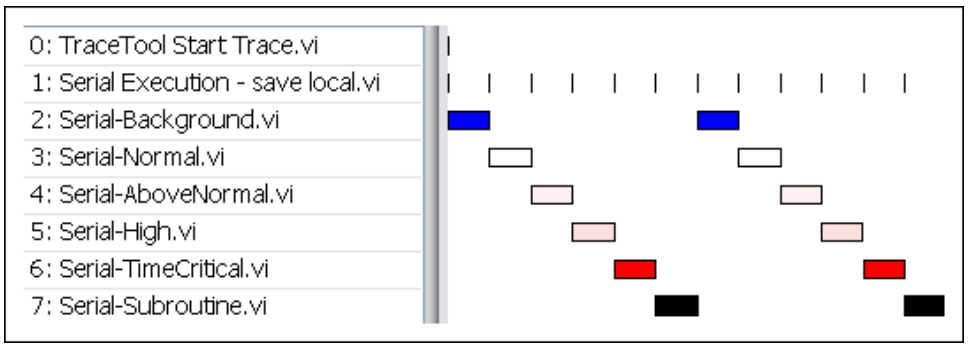
相关概念：

- Working with Trace Sessions

Viewing and Analyzing VI Events

The Real-Time Trace Viewer displays VI event data in the VI view. The VI view shows all VIs in memory on the RT target when you captured the trace session and when the VIs executed on the RT target with respect to time from left to right.

The Real-Time Trace Viewer displays the time range for the current view below the VI view. The following illustration shows the VI view of the trace session for the example VI shown in the Capturing Trace Sessions topic.



The left pane of the VI view lists the names of all VIs in memory on the RT target when you captured the trace session. The number that precedes the VI name is the dataspace ID assigned to the VI by the real-time operating system. The dataspace ID identifies the memory dataspace assigned to a VI. The Real-Time Trace Viewer lists reentrant subVIs more than once with a different dataspace ID for each copy of the VI in memory. Use the Thread and VI views to identify the individual VIs by analyzing the VI execution sequence and thread priority levels. You can rearrange the VI list in the VI view to create a logical order.

The Real-Time Trace Viewer draws the events for each VI in sequential order directly to the right of the VI name. When you reorder the names in the VI view, the Real-Time Trace Viewer redraws the events in the VI view to show the new order. You also can right-click the list of thread names and select **Arrange By** from the shortcut menu to arrange the threads in one of the following ways:

- **Name**—Arranges threads in alphabetical order.
- **Activity**—Arranges threads according to the amount of activity in the thread.
- **Sequence**—Arranges threads in sequential order by start time.

The Real-Time Trace Viewer might list VIs in the VI view that do not run while you logged the trace data. For example, you might have a VI in a case of a Case structure that never runs, which means that VIs within that case never run. Or you might have a VI that executes before you start the trace session or after you stop the trace session.

If you create an application that executes a shared VI from both a time-critical and a lower priority VI, the shared VI executes at the priority of the highest caller. If the time-critical VI sleeps to allow the lower priority VI to execute, the shared VI called by the lower priority VI attempts to execute at time-critical priority. However, the shared VI cannot execute because the time-critical thread is sleeping. The lower priority VI stops executing and waits for the time-critical thread to resume. When the time-critical thread resumes, it does not allow the lower priority VI to execute. You must execute the lower priority VI in a different LabVIEW execution system to allow the shared VI to execute in a separate time-critical thread. Refer to the Creating Deterministic Applications with the Real-Time Module topic for more information about creating multithreaded applications.

Only one VI can execute on a given processor at a given time. The TraceTool Start Trace VI logs a change event when a VI starts, stops, or calls a subVI. However, if a thread swap occurs due to operating system scheduling or a higher priority thread preempting a lower priority thread, the VI does not continue to execute and waits to receive processor resources. If two VIs appear to run simultaneously in the VI view, use the Thread Events view to determine which VI actually executed at the given time.

相关概念：

- Viewing and Analyzing Thread Events
- Capturing Trace Sessions

Viewing and Analyzing Thread Events

The Real-Time Trace Viewer displays thread event data in the Thread view. The Thread view shows the execution of threads in the real-time operating system of the RT target. The left pane of the Thread view lists the following thread names preceded by a dataspace ID.

Thread Name	Description
ETS Enet Xmit Thread	A thread that transmits data to the Ethernet driver.
ETS Null Thread	An idle execution thread.
ETS TCP ether1 DHCP Thread	A thread that handles communication with the DHCP server and establishes and maintains DHCP leases.
ETS TCP/IP Timer Thread	A thread that handles TCP communication.
ETS Timer Thread	A thread that handles timer tick interrupts and callbacks.
Exception Handler Thread	A thread that handles critical exceptions and handles the cleanup, stack trace print, and reboot of uncaught exceptions.
LabVIEW Thread [execution system #N]	A thread that runs one or more VIs in a LabVIEW execution system. execution system is the LabVIEW execution system where the thread executes and N is the thread number within the execution system.
Main Application Thread	A thread that handles front panel communication.
Unnamed Thread	A thread that handles operating system tasks.

The Real-Time Trace Viewer shows the name of the execution system where a thread runs in brackets for all LabVIEW threads. Unnamed threads are operating system threads that take very little time and run at priorities lower than the LabVIEW time-critical threads. Therefore, the Unnamed threads do not interrupt a VI set to time-critical priority.

You can right-click the list of thread names and select **Arrange By** from the shortcut menu to arrange the threads in one of the following ways:

- **Name**—Arranges threads in alphabetical order.
- **Activity**—Arranges threads according to the amount of activity in the thread.
- **Sequence**—Arranges threads in chronological order.

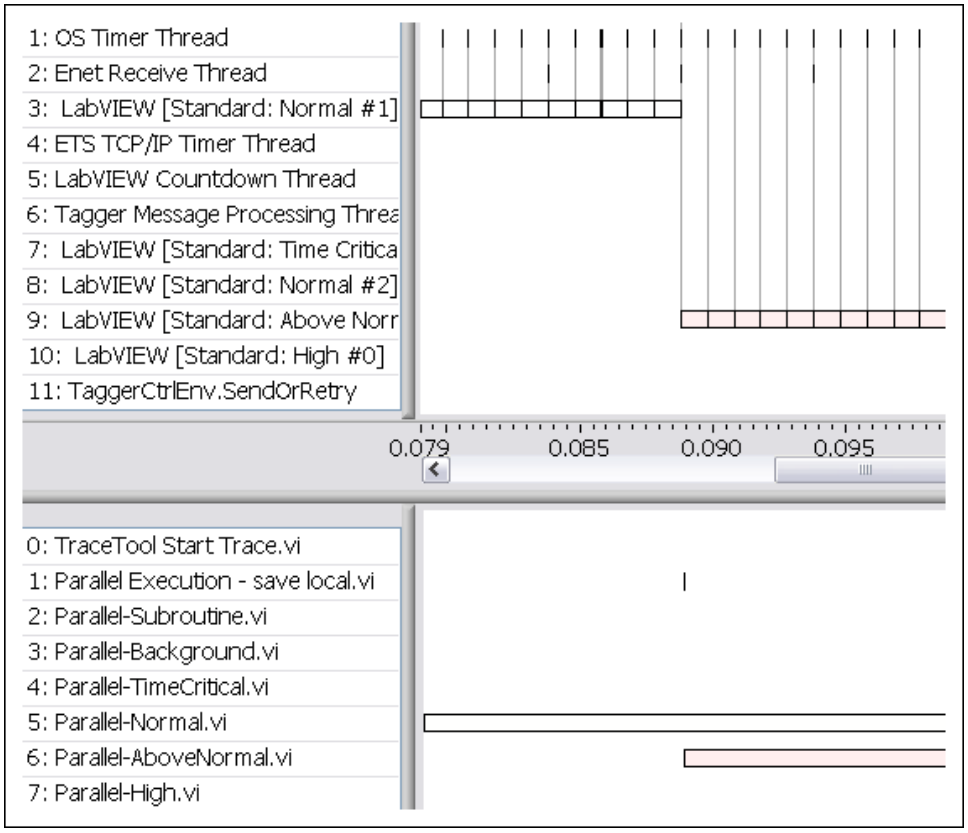
注： You also can drag and drop thread names to create a logical order.

Identifying Threads

Threads in the Thread view often share names. To identify threads, use one or more of the following methods:

- You can identify a thread by the unique combination of its execution system, thread number, and priority level.
- If you wrap code in a subVI, you can identify its threads by matching the threads in Thread view to the corresponding VI name in VI view.
- If you place code in a Timed Loop, the corresponding thread name contains the name of the Timed Loop. You also can use the TraceTool Log User Event VI in LabVIEW to generate a user flag that you then can use to identify a thread in the Real-Time Trace Viewer.
- If you captured the trace session on a multiple-CPU RT system, you can hover the mouse over a thread to highlight the other threads that executed on the same CPU.

In the following example, the Parallel-Normal VI and the Parallel-AboveNormal VI appear to run at the same time in the VI view. The Thread view shows two threads running at the two different priorities during the time span. The VIs run only during the time spans indicated in the Thread view.



When a thread is forced to wait for another thread to finish using a shared resource, the Trace Viewer displays the Wait for Object flag. Use the thread view to analyze the effects of resource contention on timing behavior.

Analyzing Multiple-CPU Trace Sessions

Multi-threaded applications that implement a parallel or pipelined architecture on a multiple-CPU system can be challenging to debug. The graphical interface of the Real-Time Trace Viewer provides an intuitive debugging environment for complex multiple-CPU applications. The Real-Time Trace Viewer includes new **Highlight CPU Mode** options that you can use to highlight all thread activity that executed on a particular CPU.

When you open a trace session that includes execution data from multiple CPUs, the **Highlight CPU Mode** ring control appears on the Real-Time Trace Viewer toolbar. The **Highlight CPU Mode** options also appear in the **View** menu. Select the **All** option to highlight all threads. Select the **Active** option to highlight only the thread activity that executed on the same CPU as the thread over which you hover the mouse. Select a particular CPU from the **Highlight CPU Mode** options to highlight all thread activity

that executed on that CPU.

You can use the **Highlight CPU Mode** options to debug parallel or pipelined architectures on multiple-CPU systems. By highlighting all the thread activity that executed on a particular CPU, you can easily trace the execution path of each CPU in the system to determine whether the threads execute as you intend.

相关概念：

- Viewing and Analyzing VI Events

Viewing System Events

The Real-Time Trace Viewer can show the occurrence of specific system events, such as sleep spans, memory manager calls, and resource mutexes, in the trace session of an application.

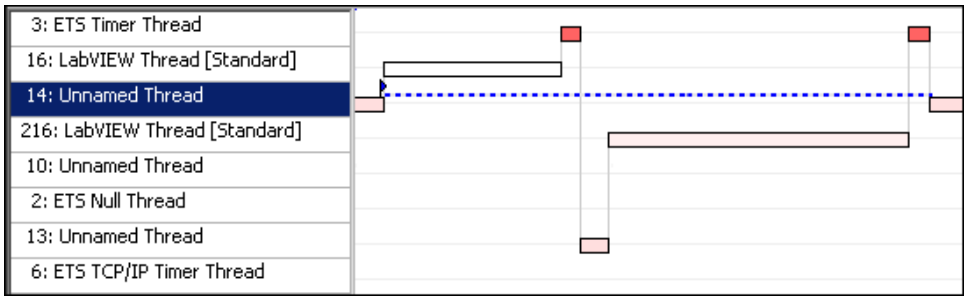
注： To log system events, you must enable detailed tracing and thread tracing using the TraceTool Start Trace VI.

Select the system events you want to view from the **Event** list of the Flag Configuration dialog box. The Real-Time Trace Viewer displays a flag followed by a dashed line in the Thread view to indicate the time range for the occurrence of the system event. You can show the following system events:

- Sleep—Occurs when a thread sleeps to allow lower priority threads to execute. By default, the Real-Time Trace Viewer displays Sleep events using a blue flag.
- Wait For Object—Occurs when a thread waits for a low-level resource, such as an event or mutex, before executing. The span is the amount of time between when the thread begins waiting and when the resource is ready for the thread to use. By default, the Real-Time Trace Viewer displays Wait For Object events using a red flag.
- Wait For Memory—Occurs when a thread conducts any memory management operation. The span is the amount of time it takes to complete the memory management operation. By default, the Real-Time Trace Viewer displays Wait For Memory events using a green flag.

- Priority Inheritance—Occurs when a thread inherits the priority of a higher-priority thread because the lower-priority thread is holding a shared resource required by the higher-priority thread. Once the resource is released, the priority returns to the previous level. By default, the Real-Time Trace Viewer displays Priority Inheritance events using an orange flag.

The following block diagram shows a Sleep event that begins in an unnamed thread running at normal priority. The unnamed thread sleeps and other threads execute during the Sleep event. The unnamed thread continues to execute when the Sleep event ends.



相关概念：

- Capturing Trace Sessions

Configuring Event Flags

You can configure event flags to make them easier to view and analyze in the trace session of an application.

Complete the following steps to configure an event flag.

1. Select **View»Configure Flags** from the Real-Time Trace Viewer to display the Flag Configuration dialog box.
2. Select the type of event you want to edit from the **Event** list.
3. Click the **Style** box to open the flag picker and select a flag style for the event.
4. Click the **Color** box to open the color picker and select a color for the event.
5. Click the **OK** button.
6. Repeat steps 2-5 to configure additional events and then click the **OK** button to close the **Flag Configuration** dialog box.

Working with Trace Sessions

You can print the current trace session from the Real-Time Trace Viewer. You also can save a trace session and then load the session in the Real-Time Trace Viewer at a later time.

Printing Trace Sessions

You can print the current view of a trace session by selecting **File»Print Window** from the Real-Time Trace Viewer. You can print the entire trace session, or you can print a specific range of the trace session by using the zoom tools to display the range before printing.

Saving Trace Sessions

After the Real-Time Trace Viewer receives a trace session from the RT target, you can save the trace session on the host computer. Select **File»Save Session** to save the trace session to file. You can reload the trace session into the Real-Time Trace Viewer at a later time.

Select **File»Close Session** to close a trace session. You must save the trace session before closing if you want to load the session into the Real-Time Trace Viewer at a later time.

Loading Trace Sessions

You can load saved trace sessions into the Real-Time Trace Viewer by selecting **File»Open Session**. You can load multiple trace sessions at once. If you have more than one trace session open, you can use the list at the left of the Real-Time Trace Viewer window to select the trace session you want to view.

相关概念：

- Viewing Trace Sessions