第十四讲 Delphi多线程程序设计

吕松茂
lsmtech@163.com

内容提要

- 线程的基本概念
- 定义线程对象
- 管理多线程
- 多线程示例

内容提要

- 线程的基本概念
- 定义线程对象
- 管理多线程
- 多线程示例

一、线程的基本概念

1.1进程与线程

在Windows操作系统中，可以同时运行多个程序，甚至可以启动一个程序的多个实例，此即所谓的多任务。可见，静态的程序与在计算机中运行的程序是有区别的。为明确概念，特将载入内存准备执行的应用程序称为进程。

进程是应用程序的执行实例，每个进程是由私有的虚拟地址空间、代码、数据和其他各种系统资源组成的。进程在运行过程中创建的资源随着进程的终止而被销毁，所使用的系统资源在进程终止时被释放或关闭。

一、线程的基本概念

进程由下列部分组成：

- 一个私有的、4GB大小的虚拟地址空间；
- 程序相关的代码和数据；
- 操作系统分配给进程的系统资源，如文件、单独的消息队列、同步对象等；
- 至少包含一个线程，这个线程称为主线程，通过它可以创建或控制其他线程。

一、线程的基本概念

线程是应用程序中的一条基本的执行路径，它也是win32进程中的最小执行单元，线程由一个堆栈、cpu寄存器的状态和系统调度列表中的一个入口组成，每个线程都可以访问进程中的所有资源。

一个进程由一个或多个线程、代码、数据和应用程序在内存中的其他资源组成。低优先级的线程一般要等待高优先级线程。一般每个线程相互独立运行，各线程间应共享资源，然而必须通过信号或其他进程内通信的方法来协调线程之间的工作。

1.2线程的特点

1. 为什么要使用线程？

- 由于CPU的处理速度比较快，可以使用户在做一件事情的时候还可以做另外一件事。比如在有些杀毒软件杀毒的时候，还可以通过菜单来浏览病毒清单。
- 在多个CPU的情况下，可以充分利用硬件的优势：将一个大任务分成几个小任务由不同的CPU来完成。
- 可以为每个线程设置优先级，调整工作的进度。

一、线程的基本概念

2. 线程的缺点

- 滥用线程容易使程序变得支离破碎，增加程序编写的复杂度。
- 在多个线程对数据进行读和写操作的时候，数据的安全有效性可能会遭到破坏。
- 有时如果频繁地在线程间切换会耗费大量的CPU时间，使得整个工作的处理时间延长了。

内容提要

- 线程的基本概念
- 定义线程对象
- 管理多线程
- 多线程示例

二、定义线程对象

2.1 TThread类

TThread类封装了Windows API和System单元中有关线程运用的函数和例程。与VCL中一般的类不同的是：TThread类是一个抽象类，其所带方法是虚拟抽象的，因而不能直接创建TThread的对象实例，而必须先声明一个由TThread继承来的线程类，再利用这个派生类创建线程对象实例和操纵线程具体类的属性和方法。

二、定义线程对象

1. TThread类的属性

Suspended属性、Terminated属性 、Priority属性 、FreeOnTerminate属性 、ReturnValue属性 。

2. TThread类的方法

Create方法、Execute方法 、Suspend方法、Resume方法 、Terminate方法 、DoTerminate方法 、Synchronize方法 、WaitFor方法

3. TThread类的事件

TThread仅定义了一个OnTerminate事件，当线程对象运行终止时触发该事件。编写该事件代码，可用于通知应用程序的主线程该线程已结束运行。

二、定义线程对象

构造函数Create

- constructor Create(CreateSuspended: Boolean);
- 参数CreateSuspended为一个布尔类型的变量。如果设置为False，则线程对象创建后立即调用TThread类的另一个过程Execute，也就是立即开始执行线程的操作；如果设置为True，则线程对象创建后，要调用过程Resume后线程的操作才开始。
- 可以在TThread类的派生类中重新定义Create构造函数，用来对派生类中的一些属性进行初始化。

析构函数Destroy

- destructor TThread.Destroy;
- 执行流程：首先要检查线程是否还在执行中，如果线程还在执行中，则调用Terminate过程结束线程，否则释放线程。Terminate过程只是简单地设置线程类的Terminated标志，所以线程仍然必须继续执行到正常结束后才行，而不是立即终止线程，这一点要注意。

二、定义线程对象

线程执行Execute

- 可以自己定义TThread类的派生类中的Execute过程，过程Execute中的代码就是线程要做的工作。如果Execute过程执行完毕，则该线程就结束了，如果属性FreeOnTerminate 为true，则释放线程。

二、定义线程对象

OnTerminate事件

- OnTerminate事件里的代码是线程执行完Execute方法，结束线程前自动调用的，可以在这里进行资源回收等工作。

- 提示：OnTerminate事件是在主线程的环境中发生的。这意味着，在处理这个事件的处理过程中，你可以不需要借助于Synchronize()而自由地访问VCL。

二、定义线程对象

等待线程结束的WaitFor

- 当一个线程应该等待另一个线程结束时，可以调用Waitfor方法。
- 这个方法属于等待线程对象，Waitfor方法的原型如下：

```
Function Waitfor(Const Astring:string):string;
```

- 注意：在主线程中要慎用waitfor，否则可能导致主线程死锁。

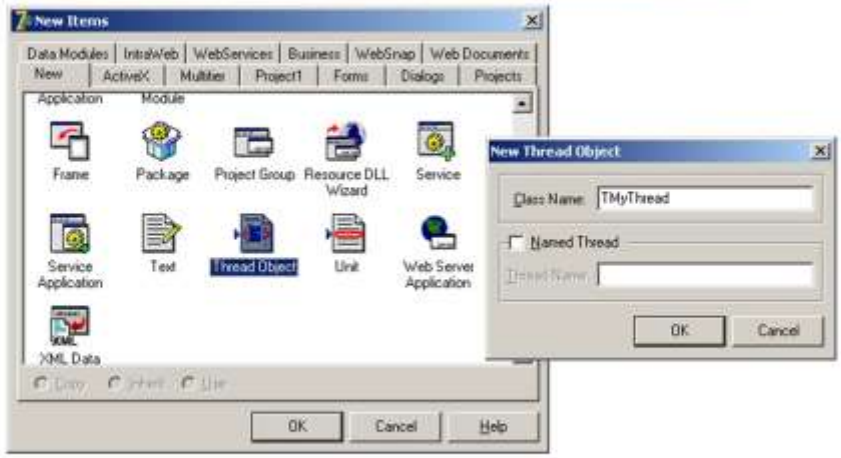
2.2 创建线程对象

要创建一个新的TThread派生类，可以使用如下步骤：

(1)通过Delphi主菜单的【File】|【New】|【Other】在弹出的【New Items】对话框中，选择TThread Objec图标，单击【OK】按钮，系统将自动创建一个Tthread Object。

(2)系统弹出News TThread Object对话框，在其中输入一个新的类名和线程名。输入类名和线程名之后，Delphi将为用户创建一个用于实现线程的新单元文件。

二、定义线程对象


New Items
Data Modules
IntraWeb
WebServices
Business
WebSnap
Web Documents
New
ActiveX
Multitier
Project1
Forms
Dialogs
Projects
Application
Module
Frame
Package
Project Group
Resource DLL Wizard
Service
Service Application
Text
Thread Object
Unit
Web Server Application
XML Data
OK
Cancel
Help
New Thread Object
Class Name:
TMyThread
Named Thread
Thread Name:
OK
Cancel

2.3 初始化线程对象

1. 为线程指定一个优先级

不能无休止的提高大量占用CPU的线程的优先级，否则可能会导致其他线程不能运行。应该只为那些花费大量时间等待一个外部事件的线程指定高优先级。

2. 指定是否释放线程

最简单的方法是让线程自己释放。这种情况下，可以将FreeOnTerminate属性值设为True。 然而，有时用户线程对象可能会代表一个应用程序要反复执行的一个任务 。

2.4 编写线程函数

当用户使用VCL对象库中的对象时，他们的属性和方法不能保证线程是安全的，也就是说，访问属性或执行方法可能会执行一些使用了未受保护的内存的操作。

如果所有对象在一个独立线程中访问它们的属性和执行方法，用户就不必担心对象之间彼此干扰，这时要使用主VCL线程，创建一个执行必要操作的独立过程，然后在用户编程的Synchronize方法中调用这个过程 。

二、定义线程对象

在下述几种情况下，用户不需要使用Synchronize方法：

(1)数据访问组件是线程安全的。

(2)图形对象是线程安全的。

(3)当使用一个线程安全的TthreadList版本时。

二、定义线程对象

1、使用线程局部变量

线程函数及其调用的任何过程都有自己的局部变量。这些过程也可以访问全局变量。

2. 检查是否被其他线程终止

用户线程对象在Execute方法调用时开始运行，并且在Execute方法结束时终止。然而有时应用程序需要一个线程持续执行，直到某个外部条件得到满足。这时，用户可以让其他的线程通过Terminated属性来通知用户线程终止。当其他线程想终止用户线程时，可以调用Terminate方法，该方法将用户线程对象的Terminated属性值设为True。

2.5 编写线程的清除代码

OnTerminate事件处理过程不作为用户线程的一部分运行，它在主VCL线程中运行，因此必须注意以下两点：

(1)用户在OnTerminate事件处理过程中不能使用任何线程局部变量；

(2)用户在OnTerminate事件处理过程中可以安全的访问任何组件以及VCL对象而不必担心与其他线程发生冲突。

内容提要

- 线程的基本概念
- 定义线程对象
- 管理多线程
- 多线程示例

三、管理多线程

3.1 为什么要使用同步？

假设有进行这样操作的两个线程A和B：

- 1、从内存中读出数据
- 2、数据加一
- 3、存入内存

三、管理多线程

现在假设用Inc进行加一操作可能出现的一种情况：

1、线程A从内存中读出数据（假设为3）

2、线程B从内存中读出数据（也是3）

3、线程A对数据加一（现在是4）

4、线程B对数据加一（现在也是4）

5、线程A将数据存入内存（现在内存中的数据是4）

6、线程B也将数据存入内存（现在内存中的数据还是4，但两个线程都对它加了一，应该是5才对，所以这里出现了错误的结果）

3.2临界区

使用临界区进行同步后：

1. 线程A进入临界区（假设数据为3）
2. 线程B进入临界区，因为A已经在临界区中，所以B被挂起
3. 线程A对数据加一（现在是4）
4. 线程A离开临界区，唤醒线程B（现在内存中的数据是4）
5. 线程B被唤醒，对数据加一（现在就是5了）
6. 线程B离开临界区，现在的数据就是正确的了。

三、管理多线程

- 临界区（CriticalSection）是一项共享数据访问保护的技术。它其实也是相当于一个全局的布尔变量。但对它的操作有所不同，它只有两个操作：Enter和Leave，同样可以把它的两个状态当作True和False，分别表示现在是否处于临界区中。
- 一般这样使用临界区：

```
EnterCriticalSection
  Try
  // 操作临界区数据
  Finally
LeaveCriticalSection
End;
```

三、管理多线程

临界区的使用

- 初始化临界区：initializeCriticalSection(sec: TRTLCriticalSection);
- 清除临界区：DeleteCriticalSection(sec: TRTLCriticalSection);
- 进入临界区：EnterCriticalSection (sec: TRTLCriticalSection);
- 退出临界区： leaveCriticalSection (sec: TRTLCriticalSection

三、管理多线程

3.3 线程的优先级

在程序一开始运行的时候，系统会自动创建一个进程和一个主线程。其中进程的优先级被称为基本优先级，线程的优先级则默认为与进程的优先级相同。

数　值	含　义
tpIdle	最低的优先级，只有系统处于空闲状态时才执行
tpLowest	比普通优先级低两级
tpLower	比普通优先级低一级
tpNormal	普通的优先级
tpHigher	比普通优先级高一级
tpHighest	比普通优先级高两级
tpTimeCritical	最高的优先级

三、管理多线程

每个线程的优先级由下面的标准决定：

(1)其他进程的优先级类（高、普通或空闲）。

(2)其他进程优先级类中线程的优先级（最低、普通下、普通、普通上、最高）。

(3)动态优先级增高，如果有的话，系统将在线程的基础优先级上增加。

三、管理多线程

在创建线程时，用户并没有用数字为它们指定优先级，系统将用两个步骤来确定线程的优先级，第一步是给进程分配一个优先级类，进程的优先级类将告诉系统进程与系统中的其他进程相对的优先级。第二步是为该进程所拥有的线程分配相对优先级。

3.4线程的同步

为了避免线程之间的冲突，有必要对访问共享资源的线程进行同步控制设计，同步还可以使线程之间相互依赖的代码能够正确运行。

Win32的API提供了如下一组可以使其句柄用作同步的对象。

三、管理多线程

(1)同步对象：互斥对象（Mutext）、信号灯和事件（Event）句柄
(2)文件句柄
(3)命令管道句柄
(4)控制台输入缓冲区句柄
(5)通信设备句柄
(6)进程句柄
(7)线程句柄

3.5 线程的局部存储（TLS）

线程的局部变量对运行此函数的各个线程是局部的，但是当线程调用另一个函数时，该函数使用的静态或全局变量对所有线程来说将是同样的值。使用线程局部存储方法，可通过对进程中用于存储和获取各个线程不同值的索引来完成对一个线程的存储分配。

3.6执行线程对象

1. 重载优先级

当在线程中指定它所能得到的CPU时间时，应在构造函数中指定线程的优先级。然而，如果线程的优先级依赖于线程何时执行，就应该创建可以进入挂起状态的线程，设置线程的优先级，然后开始执行程序。

2. 启动和停止线程

一个线程在运行前可以被启动和中止很多次。要临时中止一个线程的执行，可以使用线程的Suspend方法。用户可以调用Terminate方法要求一个线程提前停止，该方法将线程的Terminated属性设置为True。

三、管理多线程

3．暂存线程

要暂存线程，用户必须维护一个已经创建的线程的列表，这个列表可以由使用线程的一个对象维护；另一个办法是用户可以使用一个全局变量来暂存线程。

内容提要

- 线程的基本概念
- 定义线程对象
- 管理多线程
- 多线程示例

总结

- 线程的基本概念
- 定义线程对象
- 管理多线程
- 多线程示例

谢谢大家