文章编号:1001 - 9081(2003)04 - 0105 - 04

基于 COM/COM + 的多进程数据共享接口

林荣德
(华侨大学 数学系,福建 泉州 362011)

摘 要:在 COM/COM + 组件技术的应用中,由多个组件接口对象共享一个数据空间带来了对共享数据的并发控制及提高其性能的问题。本文通过对 COM/COM + 的同步机制的分析,指出在不同情况下数据共享池设置及其访问接口设计应注意的问题,并借助阅读者/写入者问题为例,提出数据共享池接口组件的设计方案。

关键词:多线程同步;数据共享接口;COM /COM + ;中间件
中图分类号: TP316 **文献标识码**:A

Multi-threaded Data-shared Interface Based on COM/COM +

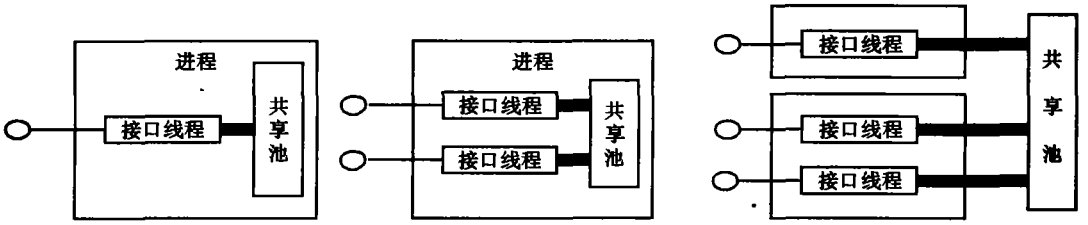
LIN Rong-de
(Department of Maths, HuaQiao University, Quanzhou Fujian 362011, China)

Abstract: This article discusses the concurrency issue and its performance, when the multi-component interface objects share a data-section. In this paper, the Synchronic mechanism of COM/COM + has been analyzed. And the problems that should be noted in the design of the data-shared pool and its interface, has been pointed out. The designing scheme of data-shared interface component about the readers/writers problem, has also been described,

Key words: multi-threaded synchronization; data-shared interface; COM /COM + ; middleware

1 背景

中间件是处于应用软件和系统软件(操作系统,网络协议,数据库)之间的软件层。它屏蔽了环境底层的复杂性,提供给应用开发者统一的功能强大的 API。使应用开发者只专注于业务逻辑,快速地开发可靠、高效的企业分布式应用。COM/COM + 技术的采用则使中间件的应用开发拥有了坚实的平台。在中间件的应用中,多个组件对象共享数据池的方法得到了普遍的采用。数据共享池是一个逻辑的数据存储区域,它可以是一块共享内存区域也可以是若干个磁盘文件等等,可供工作在多个进程(线程)内的各个组件对象访问。在组件应用中,数据共享池及接口的设计一般有如图 1 所示三种解决方案。即:单服务器进程单线程接口、单服务器进程多线程接口和多服务器进程多线程接口等三种模式。



(a) 单服务器进程单线程接口 (b) 单服务器进程多线程接口 (c) 多服务器进程多线程接口

图 1

文献[1]对上述方案进行了一定的工作。但因对 COM/COM + 的同步机制等问题上的认识存在一定的偏差,对此的解决方法也存在一些问题。因此文章通过对 COM/COM + 的同步机制的分析,对上述三种有代表性的数据共享池及其接口问题提出了一个解决方案,以供参考。

2 数据共享池的访问及其并发访问控制问题

在 Microsoft 的 COM/COM + 规范中,套间(apartment)是进程中的一个逻辑包容器。它包含那些可以被共享访问的 COM 对象和对其进行访问的线程,每个 COM 对象能够并且只能够生存在一个套间中,每个套间可以有多个 COM 对象并且线程对 COM 对象的访问操作必须在套间中进行。COM 的规范中,只有二种套间类型:单线程套间(Single Threaded Apartment)和多线程套间(Multi - Threaded Apartment),前者只能由某一线程单独进入并占有,直至该线程主动放弃为止;后者则可以同时由多个线程进入,并行访问该套间内的 COM 对象。

COM 对象有不同的套间模型对组件间的数据共享池的

收稿日期:2003 - 01 - 02(修改稿)
作者简介:林荣德(1967 -),男,福建龙岩人,讲师,主要研究方向:数据库技术、分布式处理系统.

影响主要是列集(marshaling)处理和同步处理。

对生存于单线程套间中的 COM 对象的调用必须通过列集,列集使某个线程对 COM 对象的调用必须通过代理/存根机制,由接收线程(被调方的代理/存根)到达对象所驻留、运行的单线程套间中。所以在某一时刻至多只能有一个对该对象的调用在进行。这样,虽然在组件设计中可以不必对该 COM 对象访问进行同步处理,但因列集涉及到线程切换,并且如果接收线程很忙的话,就会造成性能瓶颈。

如果被调用的 COM 对象生存于多线程套间,则线程可直接进入套间得到一个指向该 COM 对象的直接联系。但是因为对该 COM 对象的调用可以在任何时候、任何线程中来,所以在组件设计中必须编写必要的同步代码来保证在并发访问下仍然可以使 COM 对象正确执行。

由上述可知,若 COM 对象生存于单线程套间中会存在着访问性能瓶颈问题,而访问多线程套间中的 COM 对象则需要额外编写同步代码。为此,COM + 引入了一种新的套间类型,称为"标准套间"(Neutral Apartment,简称 NA)。与前述二种套间类型不同的是"标准套间"不能包含线程,它只有 COM 对象,在 NA 中的对象要求的同步方案是由 COM + 提供的称为"基于行为的同步"(activity - based synchronization)来解决的。一个行为(Activity)是一组代表单个客户线程执行的对象集,每个行为都包含一个进程范围内的锁。当 COM + 客户线程调用某一个行为的对象时,必须获取该行为锁的所有权。若锁空闲,或者该对象是属于同一行为时,调用继续;否则调用必须等待。一旦调用者得到锁,则调用继续;当调用返回时释放锁。入调用按照严格的 FIFO 的顺序进行。可以看出,NA 的引入在访问性能和编程方便中找到了一个平衡点。

但是,不论是在 COM,还是在 COM + 中,对于跨进程边界的调用,因线程与被调 COM/COM + 对象生存于彼此独立的跨进程地址空间中,所以不管被调 COM/COM + 对象存在于那种类型的线程套间中,都必须通过列集进行处理。

通过以上对 COM/COM + 的同步机制的分析中可以对前述三种数据共享池方案进行规划并作比较:

方案 1(如图 1(a)):数据共享池被唯一的接口组件对象拥有,甚至它也可以存在于接口组件对象的私有空间中,将共享池和对其的操作封装在该 COM 对象内,并由其接口完成对数据访问。因此,这个生存于单线程套间的数据共享池接口对象必须被所有的客户线程来访问。这样,不论是该服务器进程的其他客户线程,还是其他进程的客户线程,对该数据共享池接口对象的访问都应当通过列集处理。结果是,接口组件对象可不必做任何同步处理,但因系统对同一 COM 对象的列集处理是串行的,当有大量的数据访问时,此数据共享池接口对象往往成为性能瓶颈。故此方案只能应用于较小数据访问量的共享池接口组件设计中。

方案 2(如图 1(b)):数据共享池存在于一个服务器进程的地址空间中,共享池作为进程内的全局数据区独立于该进程内的所有线程。在这个方案中,该服务器进程内可以有多个工作于单线程套间的接口组件对象并发地对共享池进行访问。因此,各接口对象间必须设计相应的同步机制来保证并发访问的顺利进行。但在支持 COM + 的系统中,更简单的方法是将共享池和对它的操作封装在 COM 对象中(共享池接口对象),该共享池接口对象必须工作于"标准套间"中,这样,进程内的各接口对象间就不必考虑同步问题了,而将该任务交给操作系统按 COM + 的"基于行为的同步"方案来处理。

方案 3(如图 1(c)):数据共享池独立存在于任意一个进程之外,工作于各进程的各接口对象可以并发访问数据共享池,但是各接口对象间必须设计相应的同步机制来保证对数据共享池的并发访问能正确进行。

总之,根据方案 1 设计的数据共享池及其接口组件,比较简单,容易实现。但是对共享池的访问受唯一的单线程服务器的限制。而根据方案 2 设计的数据共享池及其接口组件,对共享池的访问是通过一个多线程服务器来实现的,性能较好;而且在支持 COM + 的操作系统下也容易实现。但是,它们都由唯一的服务器来进行的,这对于大量的,频繁的共享数据访问,往往形成瓶颈。方案 3 可以实现多个服务器进程直接访问数据共享池,甚至在某种情况下客户进程可以直接访问数据共享池(象服务器一样)。在需要提供大量的,频繁的共享数据访问的情况下,有明显的优势。

文献[1]对方案 1 的实现有较详细的叙述,但应注意到该文献中描述的同步操作是不必要的。方案 2 的实现问题也可以参照文献[1]得到解决。笔者将对方案 3 实现共享池的访问和并发问题作进一步叙述。

3 跨进程数据共享池及其接口的组件设计

3.1 数据共享池及其接口组件的体系结构和基本的接口

基于 COM/COM + 技术的数据共享池及其接口组件,一般是建立于客户/服务器模式下,可采用二层、三层或多层结构形式。但实际上三层或多层结构形式中的任意层之间也是建立在二层结构基础之上的,故本文为了描述的方便仅仅讨论二层结构形式。

图 2

如图 2 描述了采用了客户/服务器二层结构形式的数据共享池及其接口组件的体系结构。工作于单线程套间中的数据提供者对象,可以直接和共享数据区建立数据联系的通道,同时也作为共享数据区唯一的外部接口供基客户调用。基客户:可以是应用程序或扩展的接口组件对象等,当它们需要访问共享数据区时,通过此数据提供者的接口来获取数据服务。所以该数据提供者对象必须承担共享数据区的并发访问控制任务。

接口组件中数据提供者对象的主要工作:

(1) 获取共享内存区域的物理地址,必要时对内存区域进行初试化操作;

(2) 创建或获取全局同步信号量;

(3) 遵循同步规则读、写共享池数据,来实现接口所要求的内部操作。

接口组件主要 COM 接口:

(1) 读数据的一些方法或属性。如:Getdata 等;

(2) 数据的一些方法或属性。如:Setdata 等;

(3) 和关闭与共享池联系的方法。如:Open,Close 等。

基客户典型的操作步骤如下,以 VB ActiveX 脚本为例:

(1) 作系统处获取数据提供者对象一个实例

```
Set ProviderObj = CreateOleObject( "Provider")
```

(2) 打开数据连接

```
ResFlags = ProviderObj. Open( 参数)
```

(3) 通过数据提供者对象进行数据操作

```
Buf = ProviderObj. Getdata( 参数)                    //读数据
ProviderObj. Setdata( 参数)                          //写数据
```

(4) 关闭数据

```
ProviderObj. Close( 参数)
```

3.2 数据共享池存储空间的获取

因为数据共享池必须能够被所有进程访问,所以能够提供跨进程共享空间只能是磁盘文件或共享内存区域,本文只考虑后者。在 Windows 系统中,跨进程之间的共享内存可以通过发送窗口消息、动态数据交换、消息管道、套接字或内存映像文件等机制来实现。内存映像文件是一种相对简单有效的实现内存共享的方法。同时,实际上它也是在 Windows 系统中实现其他共享内存方法的基础。

内存映像文件是操作系统页式内存的一部分区域。一个文件映射对象可以通过句柄方式,映射到各个进程的虚拟地址空间中。一个进程创建一个内存文件映射对象时,可以给该对象分配一个名字,这个名字可以被其它进程用来获取该对象的句柄,并将其映射到自己虚拟地址空间中,达到共享内存区域的目的。Windows 系统相关的 API 函数主要有两类:

CreateFileMapping:该函数用来创建或打开一个内存文件对象,该函数有两个重要参数:指定内存文件对象最大存储空间和指定内存文件对象的名字。使用这个名字其它进程调用该函数时,若该内存文件对象未建立,则创建它并返回它的引用句柄;否则,仅返回它的引用句柄。也可以用此名字用 OpenFileMapping 函数获取该对象的引用句柄。用 CloseHandle 函数关闭该句柄的引用。

MapViewOfFile:该函数利用内存文件句柄,将内存文件对象映射到本进程的地址空间中,返回它物理地址指针。用 UnmapViewOfFile 函数关闭该句柄的物理地址指针映射。

3.3 阅读者/写入者问题的解决方法

3.3.1 共享数据池的分区和它们的同步操作方法

因为数据共享池是跨进程的,所以数据共享池的基本参数信息也应存储在数据共享池中的固定地址中。同时,工作在各个进程的数据共享池接口对象被创建后,并不知道数据共享池是否已经可用。所以必须设置一个数据共享池是否已经可用标志。共享数据池可分为基本参数区和数据区两部分。

共享数据池的每一部分都必须按临界段的访问规则进行。关于获取或创建全局同步信号量的方法经以及 P、V 操作的具体实现方法,可参照文献[1]或其它相关文档。

3.3.2 阅读者/写入者模式问题的主要接口实现

1) 共享数据池基本参数区的数据主要有:

InitFlag:数据共享池是否已经可用的标志(如 0x55AA 为可用),也是数据共享池是否可以初始化的标志;

Readcount:阅读者个数,初值应为:0;

Writecount:写入者个数,初值应为:0;

2) 接口对象的需创建或获取的全局同步信号量有:

Mutex_initflag:对 InitFlag 标志操作的互斥量;

Mutex_Readcount:对 Readcount 操作的互斥量;

Mutex_Writecount:对 Writecount 操作的互斥量;

Mutex_Read:进入阅读者队列的互斥量;

Mutex_Write:进入写入者队列的互斥量;

Mutex_Readers:进入阅读申请者队列的互斥量。

3) Open 方法接口的实现:

输入参数:共享数据池的名字。返回值:成功或失败信息。主要工作有:

(1) 根据输入参数,打开或创建内存映像文件对象,获取共享内存区域的物理地址,必要时对内存区域进行初试化操作;

(2) 创建或获取全局同步信号量。

可如下 C + + 伪代码所示:

a) 调用 CreateFileMapping API 函数获取共享存储区内存映射文件句柄。

b) 调用 MapViewOfFile API 函数获取共享存储区起始地址。

c) 调用 CreateMutex API 函数创建全局同步互斥信号量。

d) 通过 Mutex_initflag 互斥量访问共享数据池基本参数 InitFlag,判断共享数据池是否可用,若不可用,则要完成初始化:设置 Readcount、Writecount 初值为 0,清空数据区,最后设置共享数据池可用标志。

该接口输入参数的作用在于,根据共享数据池的名字,该组件可以适应操作不同的数据共享池。当然在这种情况下,该组件的所有内部操作都必须根据共享数据池的名字进行判断后,选择不同路径来运行。本文在此从略描述。

4) GetData 方法接口(阅读者)

可如下 C + + 伪代码所示:

```
P( Mutex_Readers);                          //不允许进入申请者队列
P( Mutex_Read);                             //不允许进入阅读者队列
P( Mutex_Readcount); //不允许其他阅读者对 Readcount 的操作
Readcount + +;
if( Readcount = 1) P( Mutex_Writecount);
                                            //第一个阅读者设置不可写
V( Mutex_Readcount);    //允许其他阅读者对 Readcount 的操作
V( Mutex_Read);                             //允许进入阅读者队列
V( Mutex_Readers);                          //允许进入申请者队列
阅读者在此进行读数据操作;
P( Mutex_Readcount); //不允许其他阅读者对 Readcount 的操作
Readcount - -;
if( Readcount = 0) V( Mutex_Writecount);
                                            //最后一个阅读者设置可写
V( Mutex_Readcount);    //允许其他阅读者对 Readcount 的操作
```

5) SetData 方法接口(写入者)

可如下 C + + 伪代码所示:

```
P( Mutex_Writecount); //不允许其他写入者对 Writecount 的操作
```

```
Writecount + +;
if( Writecount = 1) P( Mutex_Read);
                    //第一个写入者设置不允许进入阅读者队列
V( Mutex_Writecount);   //允许其他写入者对 Writecount 的操作
P( Mutex_Write);                    //不允许其他写入者进入
写入者在此进行读、写数据操作;
V( Mutex_Write);                    //允许其他写入者进入
P( Mutex_Writecount); //不允许其他写入者对 Writecount 的操作
Writecount - -;
if( Writecount = 0) V( Mutex_Read);
                    //最后一个写入者设置允许进入阅读者队列
V( Mutex_Writecount);   //允许其他写入者对 Writecount 的操作
```

6) Close 方法接口:

调用 CloseHandle API 函数释放所有互斥量。

调用 UnmapViewOfFile API 函数关闭共享数据池对象句柄的物理地址指针映射。

调用 CloseHandle API 函数关闭共享共享数据池对象句柄的引用。

需要说明的是,以上有关阅读者/写入者问题的解决方法有多种方案,文中仅选择其中的一种(参见文献[3])来说明数据共享池接口的设计。有关此问题的其它解决方案中的组件接口设计,可根据实际情况来处理。有关生产者/消费者等问题的解决方法也可按本文的思路来处理。

4 结语

在采用 COM/COM + 组件技术的应用中,由多个组件接口对象共享一个数据空间带来了对共享数据的并发控制及提高其性能的问题。通过对 COM/COM + 的同步机制的分析,指出在不同情况下数据共享池设置及其访问接口设计应注意的问题,并借助阅读者/写入者问题为例,提出数据共享池接口组件的设计方案,该方案在某机构开发的基于 Intranet/Internet 框架下的信息服务系统中,为系统的数据文件和 ASP 脚本文件提供数据共享池的接口服务,取得了良好的效果。

基于 COM/COM + 技术的三层或多层结构形式组件中,任意层之间也是建立在 C/S 二层结构基础之上的。因此,本文所述的数据共享池解决方案,对在多层结构的数据共享接口组件的设计具有一定的参考价值。

参考文献

[1] 高升,等. 基于 COM 的生产者——消费者问题的解法[J]. 微型计算机与应用,2001,(4):6-8.

[2] 蒋雄伟,马范援. 中间件与分布式计算[J]. 计算机应用,2002,22(4):6-8.

[3] Stallings W. Operating System[M]. 北京:清华大学出版社,1998.

[4] 潘爱民. 深入理解 COM +[M]. 北京:清华大学出版社,2000.

(上接第 104 页)

保数据分为三类,即空间数据(各种比例尺的地形图数据和多种分辨率的影像数据),办公业务数据和环境监测数据。利用编程工具实现地理空间元数据管理系统,使得该系统能够对各种信息资源进行统一管理,并为用户提供友好的用户界面及方便快捷的手段。参考 FGDC 地理空间元数据标准,同时进行必要的补充,将元数据信息进行分类和规划,确定各元数据项类型,形成用于对环境综合考评的元数据指标体系,采用 Arc/Info 提供的 MapObjects 进行信息自动提取,建立相应的地理元数据库。

该系统由服务器管理,元数据管理,数据服务器管理和应用管理四部分,系统直接支持 TCP/IP 的网络连接,采用客户端/服务器型操作方式。元数据管理系统通过 ODBC 进行元数据信息管理,它所操纵的对象是地理空间元数据信息表(图 3)。

图 3 元数据管理系统的逻辑图

整个系统围绕元数据管理展开,当信息管理人员制作好一幅地形图或图像数据后,启动管理系统完成该图的元数据注册并传送给数据集服务器。其它科室的终端工作人员通过管理系统的元数据查询搜索找到该图的元数据内容,直接从空间数据集服务器中获取该图,系统根据数据类型启动 Arc/Info 对图进行分析,修改等操作,完成后将该图重新注册、提交和打印输出。通过元数据管理系统的成功应用,工作人员可以实现对图库数据的快捷检索、获取和修改更新等工作。

5 结语

数据仓库技术以关系数据库、并行处理和分布式技术的发展为基础的,其目标是解决在信息技术发展中存在的拥有大量数据资源,但有用信息贫乏的问题。元数据实现是数据仓库技术实现的核心,其组织管理、功能的实现与数据仓库的成功应用有直接的关系。元数据技术必将成为分布式信息计算的核心技术之一。

参考文献

[1] 杜明义,郭达志. 空间数据仓库技术与模型研究[J]. 计算机工程与应用,1999,(12):16-18.

[2] 沈体雁,程承旗,袁文. 基于空间元数据的分布式地理数据管理模型及应用研究[J]. 测绘通报,1999,(7):34-37.

[3] 汪小林,罗英伟,丛升日,等. 空间元数据研究及应用[J]. 计算机研究及进展,2001,38(1):321-327.

[4] 承继承,林珲,周成虎,等. 数字地球导论[M]. 北京:科学出版社,2000.

[5] http://www.esri.com/devsupport/devconn/mapobjects/index.html,2001-05.

[6] http://www.fgdc.gov/clearinghouse/mitre/minimal set.html,2001-05.

[7] http://geology.usgs.gov/tools/metadata/tools/metada tools suvey.html,2001-05.

[8] Kirland M. 基于组件的应用程序设计[M]. 北京:北京大学出版社,1999.

[9] 张立,龚健雅. 基于 ASP 技术的空间数据交换中心安全性问题研究[J]. 武汉测绘科技大学学报,2000,25(1):66-69.

[10] 沈体雁,程承旗. 地理元数据技术系统的设计与实现[J]. 武汉测绘科技大学学报,1999,24(4):326-330.

[11] 赵永平,过静军,陈爱军. Metadata 共享体系的实现模型[J]. 清华大学学报(自然科学版),1999,39(12):90-93.

[12] 张健挺,万庆. 地理信息系统集成平台框架结构研究[J]. 遥感学报,1999,3(1):77-83.