联动北方
LANDING

企业级 IT 运维宝典系列

联动北方 出品

目　录

第 3 章 OLTP 基本知识

当今的数据处理大致可以分成两大类：OLTP(Online Transaction Processing)和 OLAP(Online Analytical Processing)。

联机事务处理系统(OLTP),也称为面向交易的处理系统，其基本特征是顾客的原始数据可以立即传送到计算中心进行处理，并在很短的时间内给出处理结果。这样做的最大优点是可以即时地处理输入的数据，及时地回答，也称为实时系统(Real time System)。

衡量联机事务处理系统的一个重要指标是系统性能，具体体现为实时响应时间(Response Time)，即用户在终端上送入数据之后，到计算机对这个请求给出答复所需要的时间。

而 OLAP 则是数据仓库系统的主要应用，支持复杂的分析操作，侧重决策支持，并且提供直观易懂的查询结果，对数据的及时性没有 OLTP 要求的那么高。

3.1 三层或多层 C/S 架构

在软件体系架构中，有 C/S 和 B/S 两大架构。

C/S(Client/Server)结构，即客户机和服务器结构，通过它可以充分利用两端硬件环境的优势，将任务合理分配到 Client 端和 Server 端来实现，降低了系统的通讯开销。

B/S(Browser/Server)结构即浏览器和服务器结构。它是随着 Internet 技术的兴起，对 C/S 结构的一种变化或者改进的结构。在这种结构下，用户工作界面是通过浏览器来实现，极少部分事务逻辑在前端(Browser)实现，但是主要事务逻辑在服务器端(Server)实现，形成所谓三层（3-tier）结构。这样就大大简化了客户端电脑载荷，减轻了系统维护与升级的成本和工作量，降低了用户的总体成本。

传统的 C/S 结构为二层结构，它的局限性体现为：

它是单一服务器且以局域网为中心的,所以难以扩展至大型企业广域网或 Internet：

- 受限于供应商；
- 软、硬件的组合及集成能力有限；
- 难以管理大量的客户机。

因此,三层 C/S 结构应运而生。三层 C/S 结构是将应用功能分成表示层、功能层和数据层三部分。其解决方案是:对这三层进行明确分割,并在逻辑上使其独立。原来的数据层作为 DBMS 已经独立出来,所以关键是要将表示层和功能层分离成各自独立的程序,并且还要使这两层间的接口简洁明了。

表示层是应用的用户接口部分,它担负着用户与应用间的对话功能。它用于检查用户从键盘等输入的数据,显示应用输出的数据。为使用户能直观地进行操作,一般使用图形用户接口(GUI),操作简单、易学易用。在变更用户接口时,只需改写显示控制和数据检查程序,而不影响其他两层。检查的内容也只限于数据的形式和值的范围,不包括有关业务本身的处理逻辑。

图形界面的结构是不固定的,这便于以后能灵活地进行变更。例如,在一个窗口中不是放入几个功能,而是按功能分割窗口,以便使每个窗口的功能简洁单纯。在这层的程序开发中主要是使用可视化编程工具。

功能层相当于应用的本体,它是将具体的业务处理逻辑编入程序中。例如,在制作订购合同的时候要计算合同金额,按照定好的格式配置数据、打印订购合同,而处理所需的数据则要从表示层或数据层取得。表示层和功能层之间的数据交换要尽可能简洁。例如,用户检索数据时,要设法将有关检索要求的信息一次传送给功能层,而由功能层处理过的检索结果数据也一次传送给表示层。在应用设计中,一定要避免"进行一次业务处理,在表示层和功能层间进行多次数据交换"的笨拙设计。

通常,在功能层中包含有:确认用户对应用和数据库存取权限的功能以及记录系统处理日志的功能。这层的程序多半是用可视化编程工具开发的,也有使用 COBOL 和 C 语言的。

数据层就是 DBMS,负责管理对数据库数据的读写。DBMS 必须能迅速执行大量数据的更新和检索。现在的主流是关系型数据库管理系统(RDBMS)。因此,一般从功能层传送到数据层的请求大都使用 SQL 语言。

3.2 事务的概念

在 OLTP 处理中，最核心的一个概念是事务，这是联机处理的一个立足点和基本目标，下面我们对其进行细节上的一些展开。

3.2.1 什么是事务

事务(Transaction)是一组逻辑上相关联的操作，这些操作要么全都成功执行，要么全都不执行。事务所含的操作可以分布在不同的程序甚至不同的机器上。

事务应该具有 4 个属性：原子性、一致性、隔离性、持久性。这四个属性通常称为 ACID 特性。

- 原子性(atomicity)：一个事务是一个不可分割的工作单位，事务中包括的诸操作要么都做，要么都不做。
- 一致性(consistency)：事务必须是使数据库从一个一致性状态变到另一个一致性状态。一致性与原子性是密切相关的。
- 隔离性(isolation)：一个事务的执行不能被其他事务干扰。即一个事务内部的操作及使用的数据对并发的其他事务是隔离的，并发执行的各个事务之间不能互相干扰。
- 持久性(durability)：也称永久性(permanence)，指一个事务一旦提交，它对数据库中数据的改变就应该是永久性的，除非有新的事务改变它。接下来的其他操作或故障不应该对其有任何影响。

3.2.2 什么是全局事务

所谓全局事务，是指分布式事务处理环境中，多个数据库可能需要共同完成一个工作，这个工作即是一个全局事务，例如，一个事务中可能更新几个不同的数据库。对数据库的操作发生在系统的各处，但必须全部被提交或回滚。此时一个数据库对自己内部所做操作的提交不仅依赖本身操作是否成功，还要依赖与全局事务相关的其它数据库的操作是

否成功，如果任一数据库的任一操作失败，则参与此事务的所有数据库所做的所有操作都必须回滚。

一般情况下，某一数据库无法知道其它数据库在做什么，因此，在一个 DTP（分布式事务处理:distributed transaction processing） 环境中，交易中间件是必需的，由它通知和协调相关数据库的提交或回滚。而一个数据库只将其自己所做的操作（可恢复）影射到全局事务中。

3.2.3 XA 规范

X/Open 组织（即现在的 Open Group ）定义了分布式事务处理模型。 X/Open DTP 模型(1994)包括应用程序(AP)、事务管理器(TM)、资源管理器(RM)、通信资源管理器(CRM)四部分。一般，常见的事务管理器(TM)是交易中间件，常见的资源管理器(RM)是数据库，常见的通信资源管理器(CRM)是消息中间件。

XA 就是 X/Open 为 DTP 定义的交易中间件与资源管理器（如数据库）之间的接口规范（即接口函数），交易中间件用它来通知数据库事务的开始、结束以及提交、回滚等。XA 接口函数由数据库厂商提供。

通常情况下，交易中间件与数据库通过 XA 接口规范，使用两阶段提交来完成一个全局事务， XA 规范的基础是两阶段提交协议。

在第一阶段，交易中间件请求所有相关数据库准备提交（预提交）各自的事务分支，以确认是否所有相关数据库都可以提交各自的事务分支。当某一数据库收到预提交后，如果可以提交属于自己的事务分支，则将自己在该事务分支中所做的操作固定记录下来，并给交易中间件一个同意提交的应答，此时将不能再在该事务分支中加入任何操作，但此时数据库并没有真正提交该事务，数据库对共享资源的操作还未释放（处于上锁状态）。如果由于某种原因数据库无法提交属于自己的事务分支，它将回滚自己的所有操作，释放对共享资源上的锁，并返回给交易中间件失败应答。

在第二阶段，交易中间件审查所有数据库返回的预提交结果，如果所有数据库都可以提交，交易中间件将要求所有数据库做正式提交，这样该全局事务被提交。而如果有任一数据库预提交返回失败，交易中间件将要求所有其它数据库回滚其操作，这样该全局事务被回滚。

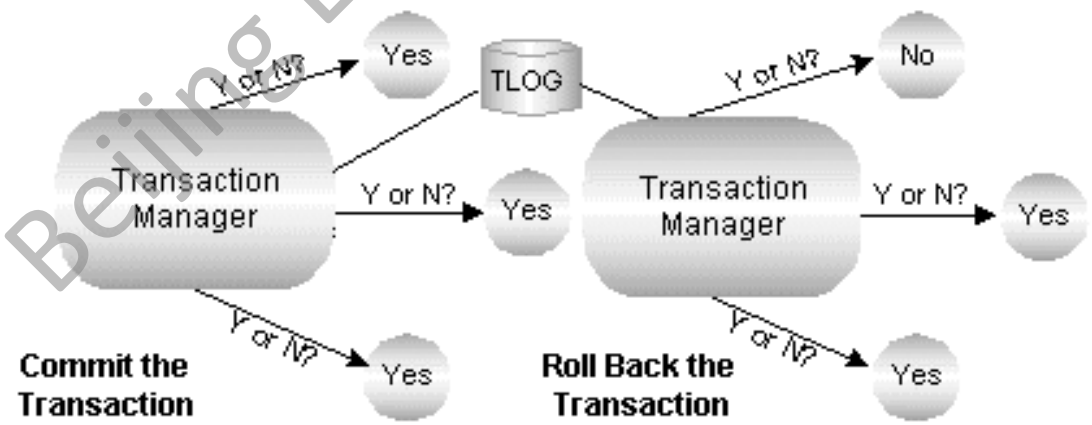

图 3-1

XA 规范对应用来说，最大好处在于事务的完整性由交易中间件和数据库通过 XA 接口控制， 应用程序只需要关注操作数据库的应用逻辑的处理，而无需过多关心事务的完整性，应用设计开发会简化很多。

另外，一般的 OLTP 实现，为了保证其高效灵便，多大量的利用了宿主机本身的核心机制，特别是 IPC 相关的资源使用。

3.3 IPC 机制简介

IPC 是 Inter-Process Communication，进程间通信。常见的除了管道之外，也是 Tuxedo 常用的 IPC 有三种：信号量、共享内存、消息队列。

3.3.1 命名管道

管道分为两种：管道和命名管道。

管道是 UNIX 系统 IPC 的最古老形式，并且所有的 UNIX 系统都提供这种通信机制。可以在有亲缘关系(父子进程或者是兄弟进程之间)进行通信，管道的数据只能单向流动，如果想双向流动的话，必须创建两个管道。

管道应用的一个重大缺陷就是没有名字，因此只能用于亲缘进程之间的通信。后来从管道为基础提出命名管道(named pipe，FIFO)的概念，该限制得到了克服。FIFO 不同于管道之处在于它提供一个路径名与之关联，以 FIFO 的文件形式存在于文件系统中。这样，即使与 FIFO 的创建进程不存在亲缘关系的进程，只要可以访问该路径，就能够彼此通过 FIFO 相互通信(能够访问该路径的进程以及 FIFO 的创建进程之间)，因此，通过 FIFO 不相关的进程也能交换数据。值得注意的是，FIFO 严格遵循先进先出(first in first out)，对管道及 FIFO 的读总是从开始处返回数据，对它们的写则把数据添加到末尾。它们不支持诸如 lseek()等文件定位操作。

3.3.2 消息队列

消息队列(messeage queue)是一个结构化的排序内存段表，该段表可以被多个进程共享，用来存放或者检索数据。每个消息队列都有一个队列头，用来描述消息队列的大量信息(队列键值、用户 ID、消息数目和最近读写消息队列的进程 ID)，消息可以看作一个记录，具有特定的格式以及特定的优先级。对消息队列有写权限的进程可以按照一定的规则添加消息，对消息队列有读权限的进程可以从消息队列中读取消息。消息队列是随内核持续的，只有内核重启或者显示的删除一个消息队列时，该消息队列才会真正被删除。

Tuxedo 在客户机和服务器通信中大量使用 UNIX 系统的消息队列，消息包括客户请求、服务响应、会话消息、通知消息、管理消息、事物控制消息等。默认情况下每个服务器都有一个请求队列来接受客户端的请求，这就是 SSSQ(single server single Queue)模型，每个客户机都有一个响应队列来接受服务器的响应消息，如果服务器太多，也可以配置多个服务器共享同一个请求队列，这就是 MSSQ(Multiple servers single Queue)模式。如果一个服务器调用了其它服务，还需要为它创建一个响应队列。

3.3.3 信号量

信号量(semaphore)是为那些访问相同资源的进程以及同一进程不同线程之间提供的一个同步机制。它不是用于传输数据，而只是简单地协调对共享资源的访问。

信号量包含一个计数器，表示某个资源正在被访问和访问的次数，用来控制多进程对共享数据的访问。一旦成功拥有了一个信号量，对它所能做的操作只有两种：请求和释放。当执行释放操作时，系统将该信号值减 1（如果小于 0，则设置为 0）；当执行请求操

作时，系统将该信号值加 1，如果加 1 后的值大于设定的最大值，那么系统将会挂起处理进程，直到信号值小于最大值为止。

Tuxedo 用信号量来确保在某一时刻只有一个进程对某一块共享内存进程访问。信号量配置太低会导致 Tuxedo 系统应用程序无法启动。

3.3.4 共享内存

共享内存(shared memory)是一片指定的物理内存区域，这个区域通常是在存放正常程序数据区域的外面，它允许两个或多个进程共享一给定的存储区，是针对其他通信机制运行效率较低而设计的。因为数据不需要来回复制，更不需要在客户机和服务器之间复制，进程可以直接读写内存，所以是最快的一种进程间通信机制。为了实现更安全通信，它往往与其它通信机制，如信号量结合使用，来达到进程间的同步及互斥。

在 Tuxedo 中，就是利用这个特性，使用共享内存存储公告牌，用来公告进程状态信息和需要在进程间共享或传递的数据，大大提高了对这些信息访问的效率。

3.3.5 IPC 资源相关的操作系统内核参数

由于 Tuxedo 大量采用 IPC 技术，实现“无连接”通信，提高系统性能，在部署 Tuxedo 应用时需要调整系统内核 IPC 参数，以达到良好运转。

更多关于 IPC 的详细内容，请参看后续第 17 章。