基于 OCI9 的数据库访问接口的改进与实现①

徐 薇[1] 胡 术[2] 杨 诚[1]

(1.四川大学 计算机学院 四川 成都 610065; 2.四川大学 图形图像研究所 四川 成都 610064)

摘 要： 设计并实现了一种基于 OCI9 的数据库访问接口。接口采用双条件变量机制，来解决数据库异常时函数长时间阻塞问题。同时，针对在网线断开时重连数据库会出现的内存增加的问题，提出了二次连接检测的解决方案。最后通过实验验证该数据库访问接口的应用效果。

关键词： 空管系统；ORACLE 调用接口；阻塞/非阻塞；双条件变量；内存增加；二次连接检测

Improvement and Implementation of Database Access Interface Based on OCI9

XU Wei[1], HU Shu[2], YANG Cheng[1]

(1.Department of Computer, Sichuan University, Chengdu 610065, China;

2.Institute of Image and Graphics, Sichuan University, Chengdu 610064, China)

Abstract: A database access interface based on OCI9 is designed and implemented. Double conditional variable mechanism is presented in the interface to resolve the long time blocking problem caused by function when some anomalies occur in the database. Secondly, when the net wire is pulled out, the database reconnection will have a problem of the increase in memory. To avoid it, a solution of double detections of connection is put forward. Finally, experiments are made to test the application effect of the database access interface.

Keywords: ATC system; OCI blocking/non-blocking; double conditional variable; increase in memory; double detections of connection

1 引言

空中交通管制系统(ATC system)需要实时地采集并存储和维护飞机飞行时的各种海量数据，如设备信息、地图信息、雷达信息、飞行情报等数据。这就要求空管系统中配备具有高效处理能力的数据库服务器来处理并保存这些海量数据，空管系统常选用 Oracle 作为数据库服务器。OCI 是 Oracle 公司提供的编程接口，其函数有阻塞和非阻塞两种工作模式，然而，在 ATC 系统的开发过程中发现 OCI 函数的非阻塞实际上是在服务器、网络工作均正常下表现为非阻塞，而在故障发生时(如网线断掉)表现为阻塞，这时进程将长时间阻塞而无法继续运行。鉴于此，本文基于OCI9 技术设计并实现了一种 Oracle 数据库访问接口，较好地解决了上述问题。

2 OCI简介

ORACLE 调用接口(OCI: Oracle Call Interface)是 ORACLE 公司开发的应用程序开发工具，它提供了一组可对 ORACL E 数据库进行存取的接口子例程(函数)，通过在第三代程序设计语言(如 C 语言) 中进行调用可达到存取 ORACLE 数据库的目的[1]。作为一种数据库访问方法，它比 ODBC 更灵活，访问数据库服务器速度更快[2]，而在 UNIX 平台这几乎是唯一的访问方法。

① 基金项目:国家高技术研究发展计划(863)(2006AA12A104)
收稿时间:2009-08-15;收到修改稿时间:2009-09-03

一个 OCI 应用程序的基本结构包括[3]：

1) 初始化环境和线程。

2) 分配必要的句柄与数据结构。

3) 建立与数据库的连接以及创建用户会话。

4) 通过与服务器交换数据，而后再做数据处理。

5) 结束用户会话与断开与数据库的连接。

6) 释放在程序中所分配的句柄。

3 接口的设计与实现

ATC 系统通常要在各种不同的平台上长时间稳定运行，要具有高度兼容性和跨平台性，本文设计的接口对 OCI 底层的函数调用进行了封装。其实现的主要功能有：

1) 提供了友好的接口供程序调用。

2) 封装了复杂的 OCI 函数调用，只提供初始化、连接、执行、查询、断开等直观的数据库访问方法[4]。

3) 接口可在多平台上使用，具有跨平台性。

4) 在网络断开时能返回错误并能及时重连，且抑制了内存增加问题的出现。

5) 采用基于双条件变量的方法，解决了长时间阻塞的问题。

3.1 接口的实现

调用接口采用 C++语言开发，以动/静态库的形式和相应的头文件一起提供给接口的使用者，接口的实现主要使用了以下几个重要的类：(1)类 CInterface 设计为纯虚类，作为提供给用户的接口，隐藏了接口的内部实现。(2)类 CRealInterface 继承了类 CInterface，接口的具体实现由它来完成。(3)两个重要的底层类：CMutex 类提供了同步互斥锁；CEvent 类提供了条件变量。(4)线程运行类 CThreadManager 和 CTaskTread，它们调用了前面提到的两个底层类来实现双条件变量机制。在类 CEvent 里封装了三个重要的函数 Signal，Wait 和 WaitTime，它们是线程的同步实现的关键。Signal 函数是对条件变量发出信号，唤醒在此条件变量上阻塞的线程；Wait 和 WaitTime 则是实现线程在条件变量上阻塞等待，前者是使用函数 pthread_cond_wait，后者则是使用函数 pthread_cond_timedwait。

3.2 双条件变量机制解决长时间阻塞问题

一个长期不间断运行的应用系统不可避免会出现数据库服务器宕机或网线插拔的动作，如果这时进程正在使用 OCI 接口进行数据库操作，进程将阻塞于正在调用的 OCI 函数上，如果故障不能及时恢复，OCI 函数将长时间阻塞十几分钟，对应用程序来讲大多数情况下这是不允许的。条件变量是线程可用的一种同步机制。条件变量给多个线程提供了一个会合的场所。条件变量与互斥量一起使用时，允许线程以无竞争的方式等待特定的条件发生。在条件变量上阻塞等待的函数有两个：pthread_cond_wait 和 pthread_cond_timedwait。前者是在条件上永久等待，后者指定了等待的超时时间。

因此，本文结合条件变量设计了一种双条件变量的机制来解决这种长时间阻塞的问题，并实现多线程并发处理，即用一个管理者来管理一个任务线程。其主要实现方法是：主线程创建一个用于管理的类 CThreadManager 来充当管理者，这个管理类创建任务线程 T1，任务线程 T1 运行后立即在条件变量 cond1 上等待管理者的触发。管理者把要执行 SQL 语句的函数指针和参数作为一个任务分配给任务线程 T1，管理者接着对 cond1 发出信号，解除 T1 的阻塞，T1 唤醒后开始执行赋给的任务；管理者立即在另一个条件变量 cond2 上阻塞等待，T1 完成任务后对 cond2 发出信号，而 T1 重新在 cond1 上阻塞等待下一个任务。

双条件变量机制的原理图如图 1 所示[5]：

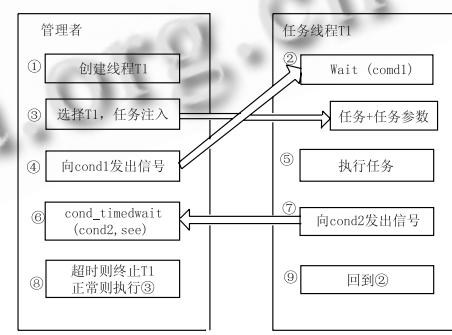

图 1　双条件变量机制原理图

上图中数字为表示因果和时序关系的顺序编号，其中条件变量 cond1 的阻塞等待使用 pthread_cond_wait，条件变量 cond2 的阻塞等待使用 pthread_cond_timedwait 的超时阻塞，当 T1 不能正常完成任务时，管理者会在 cond2 上超时解除阻塞，管理者强行终止 T1，在下一个 SQL 请求来时，管理者需要重新创建任务线程。在数据库和网络工作

正常时，T1 中的任务可以在指定的时间里完成，所以在大部分时间里，任务只是不断的注入到任务线程 T1 来完成，线程只是在阻塞和执行任务间不断循环，只有在异常出现时会出现任务线程的终止。这样不但解决了长时间阻塞问题，而且还节省了线程创建、销毁的开销，又保证了对任务的并发处理，提高了系统的响应速度。下面给出的代码，是任务线程的实现，通过这段代码可以更好地理解该机制：

T1 创建后在此函数(RunReal)里运行，在阻塞和执行任务间不断循环，变量 done 和 runnig 用来控制 T1 的循环何时结束。

```
void CTaskThread::RunReal()
{
while (!done && owner->running)
{
suspend_lock->Wrlock();    //Wrlock() 封装在类 CMutex 里，用于加互斥锁
while (wait && (!done) && owner->running)
{
suspend_cond->Wait(suspend_lock);
}  //即上图的②，Wait 封装在 CEvent 里，对条件变量 suspend_cond 的阻塞等待
suspend_lock->Wrunlock();  //Wrunlock()封装在类 CMutex 里，用于互斥锁解锁
if (done || !owner->running) {break;}
if (action) {(*action)(action_prm);}  //T1 执行任务 action，即上图的⑤
timeout_cond->Signal();  //即上图的⑦，Signal() 封装在 CEvent 里，向 timeout_cond 发送信号，唤醒在此条件上阻塞的线程
...}
```

3.3 内存增加的解决

断开网线后，OCIServerAttach 一定会阻塞，而 OCITransCommit 也是一定阻塞的，因此，函数 Logon()(封 装 了 OCIServerAttach) 和 Execute-SQL()(封装了 OCITransCommit)都是放在任务线程里执行的。由于引入了双条件变量机制，任务线程会在超时后被终止，所以，断网后执行这两个函数的任务线程会被管理线程终止。实验中，当网线一直处于断开状态时，使用 Logon()的任务线程做连接动作时，用“ps －A －o pid,comm,pmem,rss,osz | grep test.solaris”命令看到该程序(test.solaris)使用的内存在增加。以 Solaris 平台为例，每连接失败一次增加 60KB，所以，应该在保证在网线已成功连接上时才执行数据库的重连，这样才不会发生内存增加的问题，这里 OCI 函数已经不能实现该功能，需要使用基于 TCP 的网络编程对 ORACLE 服务器的侦听端口 1521 进行直接网线通断的连接测试。

当在一个非阻塞的 TCP 套接口上调用 connect 时，connect 将立即返回 EINPROGRESS 错误，此时已经发起的 TCP 三路握手继续进行；当连接成功建立时，描述字变为可写，当连接建立遇到错误时，描述字变为既可读又可写[6]。鉴于此，可以设计一种基于非阻塞 connect 的二次连接检测机制，来对网络连接状态进行检测。

具体实现是：(1)第一次连接检测，创建一个 sockfd(套接字)，用此 sockfd 向服务器主机发起非阻塞 connect，若非阻塞 connect 返回 0，那么连接已成功(当服务器处于客户所在主机时这种情况可能发生)，返回检测成功，连接存在；若 connect 的返回为 EINPROGRESS，表示连接已经启动但是尚未完成，需要进行第二次连接检测；其他错误则返回检测失败，连接不存在。(2)在首次连接数秒后开始第二次连接检测，使用 select 来检测 sockfd 上的这个连接或成功或失败的已建立条件，select 的超时设置为一个极小的值，如 10μs。打开读集(rset)和写集(wret)中对应的 sockfd 位，用 select 函数检测 sockfd 是否准备好，若 sockfd 变为可读或可写，就使用 getsockopt 函数获取其 SO_ERROR 选项的状态，若连接成功建立，则值为 0；若连接建立发生错误，则该值就是对应连接错误的 errno 值。对值进行判断后，返回第二次检测的结果。

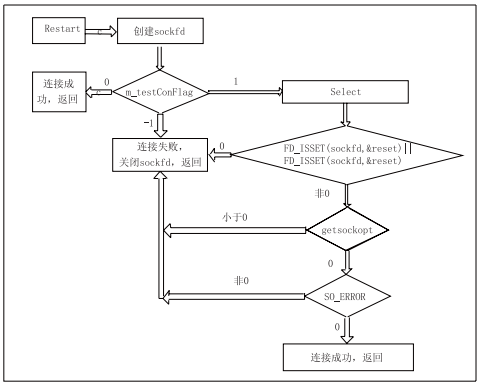

图 2 二次连接检测机制流程图

图 2 给出了二次连接检测机制的实现流程图(其中加入了一些实现可移植性的处理)。

图 2 中 m_testConFlag 为是否进行第二次连接检测的标志。在程序中，将 select 的超时时间设置为一个非常小的值，如 10μs，实现非阻塞检测。针对在 Solaris 下 getsockopt 的返回值的特例[7]有所不同也进行了处理，实现了可移植性。函数 Restart(重连函数)里，在重连数据库前执行上述的二次连接检测，则可保证在网络连接正常的情况下才重连数据库，避免了内存增加。

4　实验结果与分析

4.1 实验设计

实验采用 HP Tru64 操作系统为客户端，Oracle9i 数据库安装在 LINUX Red Hat 2.6.9 EL 系统主机上，测试用 SQL 语句采用：“select * from test_table ”和“ insert into test_table (id, num, sum, new_code, old_code, Time) values(‘1’, ‘1234567890’, ‘333’, ‘4444’, ’55555’,to_date(‘2008-11-1118:11:11’, ’yyyy-mm-dd hh24:mi:ss’))”。

超时时间设置为 5 秒，程序正常运行后，拔掉网线，分别在超时时间内和超时时间后插上网线，查看程序运行的结果。保持网线断开的状态，查看内存的占用率。

4.2 实验结果分析

(1) 程序正常运行一段时间后，拔掉网线，任务线程在 ExecuteSQL()函数里阻塞，5 秒内重新插上网线，ExecuteSQL()成功返回，SQL 语句正常执行。

(2) 程序正常执行一段时间后，拔掉网线，保持网线断开状态，5 秒后，任务线程不再阻塞，报错后该线程被终止，然后一直尝试重连数据库。15 秒时重新插上网线，重连数据库成功，SQL 语句重新成功执行。

(3) 程序正常执行一段时间后，拔掉网线，保持网线断开状态，则程序一直尝试重连数据库，此时，不使用二次连接检测机制时，出现内存增加；使用二次连接检测时，内存增加问题得到解决。

由上述实验结果可看出：使用双条件变量机制，网线断开 5 秒后，由于超时，管理线程从 Wait 函数返回，终止阻塞的任务线程，从而打破了任务线程的阻塞，解决了长时间阻塞的问题。同时，在重连机制中使用了二次连接检测机制成功的解决了内存增加的问题。

5　结论

本文设计并实现了一种基于 OCI9 的数据库调用接口，接口封装了复杂的 OCI 函数调用，具有良好的兼容性，已在多种平台上成功运行。为 ATC 程序访问数据库提供了很大便利。成功的解决了网线断掉或系统宕机时程序长时间阻塞的问题，实现了多线程的并发处理，有效地提高系统响应速度和整体性能。同时，避免了断网后任务线程重新连接数据库时引起的内存增加。

参考文献

1 汪林林,马锐.用 OCI 开发 Oracle 数据库的方法.计算机应用, 2003,23(S2):46－47.
2 郑军,陈正阳.利用 OCI 与 OLE 开发 Oracle 空间数据库.计算机工程与设计, 2004,25(6):1008－1009.
3 庞维翰,陈有青.用 OCI 封装类进行数据库间应用系统的移植.计算机工程与应用, 2005,(29):178－179.
4 杨诚,杨红雨,胡术,程锦.基于 OCI9 的 Oracle 数据库调用接口的实现.微计算机应用, 2009,30(4):36－37.
5 莫晶,胡术,李娜娜,肖启战,吴昊.基于 OCI 的数据库访问接口的改进与实现.福建电脑, 2009,(1):2－3.
6 胡术,刘振超.空中交通管制系统开发中一种消息中间件的设计与实现.四川大学学报(自然科学版), 2009, 46(3):600－601
7 Stevens WR,等. UNIX 网络编程(第 3 版).北京:清华大学出版社, 2006.