doi: 10.7690/bgzdh.2013.07.027

基于消息中间件的数据同步更新方法

袁震
（西安科技大学计算机学院，西安 710054）

摘要：针对分布式环境下同类型数据库数据发生变化后中心数据库同步更新的问题，提出一种基于消息中间件的数据同步更新方法。该方法采用中间件方法，综合使用 Socket 通信机制、多线程等技术，在中心数据库的服务器端使用队列来存储上传来的数据，并给出部分代码及处理流程图。仿真实验结果证明：该方法性能稳定、效率较高，且确保了传输数据的完整性。

关键词：消息中间件；数据同步更新；Socket；多线程；队列
中图分类号：TP311.5　　**文献标志码**：A

A Method of Data Synchronous Updating Based on Message Oriented Middleware

Yuan Zhen
(*College of Computer*, *Xi'an University of Science & Technology*, *Xi'an* 710054, *China*)

Abstract: Aimed at the problem of the central database synchronization update after the data in the distributed database changing, a method of data synchronous updating based on message oriented middleware is presented. By the oriented middleware, both the communication mechanism of Socket and the technology of multi thread are comprehensively used in this method. The data to be received by the centre database server is temporarily saved in the queue as a data structure. The processing flow chart and the part of the code are provided. The simulation experiment results prove that the method has a stable performance and high efficiency while guaranteeing the integrity of the data transmission.

Key words: message oriented middleware; data synchronous updating; Socket; multi thread; queue

0 引言

通常，大型企业都拥有一个中心数据库，用于存储该企业当前的所有数据。为了提高系统的访问效率，各部门的相关数据存储于部门数据库中。当部门数据库的数据更新时，如何在短时间内将更新的结果上传到中心数据库，从而使数据一致性得到保证十分重要。尽管各大数据库供应商都提供了自己的数据复制机制，可以在分布式环境下满足同类型数据库间的数据同步，但配置比较麻烦，而且不能满足一些特定需求，如实时性、数据可靠性等。针对上述问题，笔者提出了一种基于消息中间件的数据同步更新方法，解决了部门数据库发生变化后，中心数据库同步更新的问题。

1 数据同步常用的方法及策略

数据同步分为完全同步和增量同步。完全同步要求部门数据库的所有数据全部传输到中心数据库，由中心数据库根据传输的数据和同步逻辑进行数据更新，此种方法逻辑简单，但如果数据量较大，会带来巨大的网络传输量。而增量同步仅仅将变化的数据由源数据库端传输到目的数据库，从而大大节约了数据传输量，增强了同步效率[1]。增量同步的关键是要捕捉同步对象的变化，即如何获取增量数据的问题[2]。目前，常用的获取增量数据的方法有：中间件法、触发器法、时间戳法、扫描表法[3]。时间戳法原理最为简单，实现起来也较容易，但浪费了系统资源，而且不能达到真正意义上实时更新的目标；触发器法同样易于实现，需要数据库有触发器的支持，对同步表的操作性能会有所影响；扫描表法对于记录不是很多的小表是可行的，但对于记录较大的表，则效率比较低。

笔者采用中间件方法，即在中心数据库的服务器端引入面向消息的中间件。消息中间件是利用高效可靠的消息传递机制进行平台无关的数据交流，并基于数据通信来进行分布式系统的集成，通过提供消息传递和消息排队模型，它可在分布环境下扩展进程间的通信，并支持多通讯协议、语言、应用程序、硬件和软件平台[4]。

收稿日期：2013-02-06；**修回日期**：2013-03-06
作者简介：袁震(1988—)，男，陕西人，在读硕士，从事系统集成及数据库技术研究。

2 基于消息中间件的数据同步更新方法

基于消息中间件的数据同步更新方法是由生成报文、报文入队、报文出队、反馈应答等步骤组成。

2.1 数据表的设计

为了更好完成数据同步，需要对中心数据库和部门数据库的每个基表添加属性，如表 1 所示。

表 1 基表添加属性

数据库类别	属性名称	说明
中心数据库	SysNo	系统编号，自增变量，主键。
	DataSource	数据来源，存储来源部门数据库的标志
部门数据库	IsReceived	标记该条记录是否同步到中心数据库。0 表示未同步更新；1 表示已经同步更新；

2.2 传送报文结构的设计

客户端发送报文的内容格式为：发送时间(年月日时分秒)、操作类型(I 或 D)、操作的数据表名称、数据内容信息、生成效验码。中间用\n 分隔。其中，操作类型中 I 表示添加；D 表示删除。对于修改数据，分解为先进行删除，再进行添加。比如编号为 0001 的用户在 2012 年 1 月 12 日 9 点 14 分 15 秒借用了编号为 00000012 工器具，那么该条记录产生的报文的内容是：borrowopt\nxust\nI\n00000012\n0001\n 借出\n2012-01-12 09:14:15\n 效验码\n。

2.3 队列设计

基于消息中间件的数据同步更新方法是，在中心数据库的服务器端使用队列来存储上传来的数据。采用队列这种存储方式是为了确保每条报文在正确解析之后，都能上传到中心数据库中，确保传输数据的完整性。

基于消息中间件的数据同步更新方法数据结构的结构体定义如下：

```
typedef struct Queue
{
    Data *data;
    int front;//队首数组下标
    int rear;//队尾数据下标
    int len;//队列当前长度
    int maxSize;//队列最大容量
    int sysno;//中心数据库中基表可用最小编号
};
typedef struct Data{
    string table;//操作表的名字
    string source;//数据来源计算机名称
    string operation;//操作类型(D 代表数据删除；I 代表数据添加)
    string sqlstr;//生成的 sql 语句
    struct tm *ptime;//上报时间
};
#define SIZE 30//队列数据的最大长度
```

2.4 多线程

多线程是这样一种机制，它允许在程序中并发执行多个指令流，每个指令流都称为一个线程，彼此间互相独立[5]。应用多线程机制，可以很好地解决程序运行过程中多任务的同时性的问题。本方案由于报文入队和报文出队 2 个任务相对独立，为了减少数据进入中心数据库的时间，提高运行效率，故引入了多线程机制，即主线程为报文入队线程，副线程为报文出队线程。实现代码如下：

```
void main(void){
    _beginthread(&thread_InsertDB, 0, &A);//加载报文出队进程
        while(1){…;//报文入队进程代码
        }
}
void thread_InsertDB(void *param){
    while(1){…;//报文出队线程代码
    }
}
```

2.5 Socket 通信机制

Windows Sockets 规范是一套基于 Windows 的网络编程接口，Windows Sockets 利用下层的网络通信协议功能和操作系统调用实现具体的通信工作，轻松实现远程机器通信[6]。笔者使用 Socket 通信机制来完成报文数据在网络中的传输。实验中需要在部门数据库的客户端和中心数据库的服务器端分别引入 Socket 机制，其处理流图如图 1、图 2 所示。

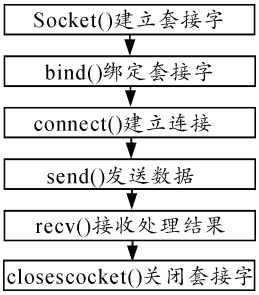

图 1 客户端 Socket 处理流程

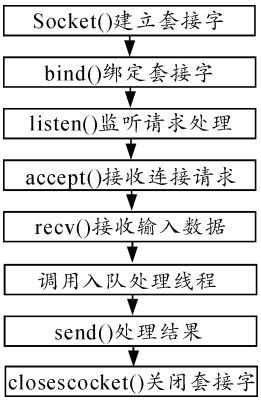

图 2　服务器端 socket 处理流程

2.6　数据同步更新的过程

采用 2 个线程的并发执行来协助完成数据同步更新的工作。一个线程是报文入队线程；另外一个是数据入库线程。具体步骤如下：

Step1：生成报文。当部门数据库发生更新的操作后，按照上述报文格式生成报文后，为了保证报文的正确性，报文生成后在部门数据库的服务器端进行一次校验，并加入效验码，再发送至中心数据库上。为了使报文能够正确地被解析，在服务器端进行适当的解析。如果报文数据项中出现“\n”，则将其变成“_\n_”。

Step2：报文入队。报文入队线程在中心数据库端时刻监视部门数据库端发送报文情况，具体如图 3 所示。当中心数据库端收到部门数据库端发送的报文之后，首先对报文进行正确性校验，如果与部门数据库端效验码一致，则再进行相应的解析报文的内容，并根据报文内容信息，进入合适的队列。如果发生异常，向部门数据库端发出指令，重新打包发送该报文；否则向客户端发出指令，重新打包发送该报文。

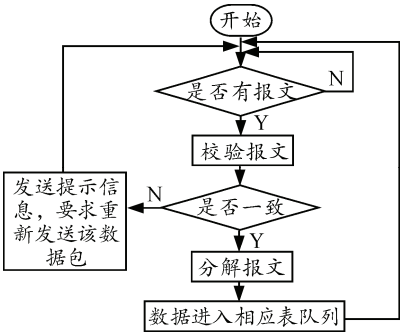

图 3　报文入队线程流程

Step3：报文出队。报文出队线程的流程如图 4 所示。

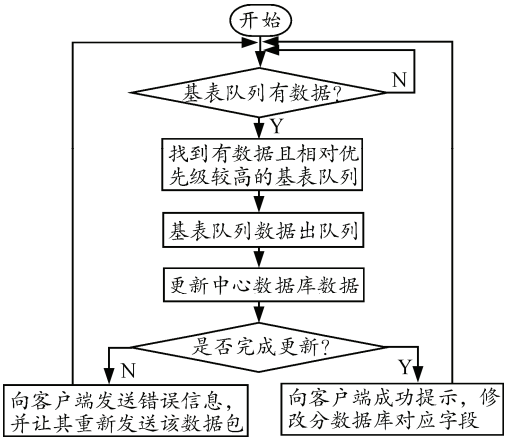

图 4　报文出队线程流程

Step4：反馈应答。数据上报完成后，如果成功，则向客户端发出上报成功指令，修改部门数据库的信息。

3　实验与分析

3.1　实验环境与数据来源

1) 实验环境。

本实验是在内存 2.5 G，双核 CPU 2 GHz，Window XP 系统，SQL server 2005 数据库管理系统的计算机上完成的。

2) 数据来源。

测试数据是由一个测试程序随机生成，总数据量可以根据测试的需要由用户自己在测试时设定。为了模拟真实的数据传输过程，测试程序每次产生一定数量的数据，然后暂停一定的时间，再产生同等数量的数据，直至达到设定的最大数据量为止。

产生的测试数据最终被插入到 3 个不同的数据表中，表的结构说明如下：

emptable(sysno,empno,empname,empage,source);

coursetable(sysno,cno,cname,source);

ecptable(sysno,eno,cno,point,source);

3) 测试数据的组织及相关参数的说明。

采用 2 种不同的队列组织测试数据：整表队列和分表队列。整表队列是指在整个测试程序中，使用一个队列去接收所有的数据，并将数据插入到中心数据库中；分表队列是指为每个基表使用各自的队列接收各自的数据，并将数据插入到中心数据库中。分表队列中 3 个基表的优先级顺序排列为：emptable>coursetable>ecptable。

通过采用上述 2 种队列组织测试数据，选取程序运行时间和数据接收率 2 个指标，并设置合理的

测试用例来说明笔者所提出方法在数据传输过程中的可靠性及效率。以下是测试时相关参数的说明：

TDV：测试程序产生的总数据量；

PDV：测试每次产生的数据量；

STM：测试程序 2 次产生数据之间的间隔时间；

QCZ：测试程序运行时消息队列最大使用长度；

RTM：程序运行时间；

DRR：数据接收率，即存入数据库的数据条数/发送数据条数。

3.2 测试与结果分析

在 VC++6.0 平台下实现该方法，然后通过测试得到实验结果。

3.2.1 测试数据及实验结果

为了验证本方法的正确性及有效性，尤其是在少量及大量数据上报时的处理效率，笔者设置了 3 组不同的测试数据进行测试，见表 2。

表 2 测试用例

序号		TDV	PDV	STM/ms
A		500	100	200
B		5 000	200	200
C	C_1	2 500	200	200
	C_2	2 500	200	200

在表 2 中，A 测试用例是指每隔 200 ms 生成 100 条记录，共产生 500 条记录；B 测试用例是指每隔 200 ms 生成 200 条记录，共产生 5 000 条记录；C 测试用例是指 2 个测试客户端同时每隔 200 ms，生成 200 条记录，各生成 2 500 条记录。

为了更客观地反映实验结果，将由系统产生的误差降至最小，对上述 3 个测试用例，笔者分别使用整表队列和分表队列各测试 5 次，然后对各自的程序运行时间 RTM 取平均值；并对消息队列最大使用长度取最大值；在采用分表队列策略时，QCZ 分别对应 emptable, coursetable, ecptable 最大使用长度。实验结果如表 3～5 所示。

表 3 测试用例 A 实验结果

测试策略	RTM/s	DRR/%	QCZ/条
整表队列	7.509	100	1
分表队列	7.862	100	1 1 1

表 4 测试用例 B 实验结果

测试策略	RTM/s	DRR/%	QCZ/条
整表队列	73.213	100	11
分表队列	66.006	100	7 7 9

表 5 测试用例 C 实验结果

测试策略	RTM/s	DRR/%	QCZ/条
整表队列	60.943	100	19
分表队列	61.351	100	6 5 8

3.2.2 结果分析

通过上述实验得出如下结论：

1) 从表 3 和表 4 可以看出，无论是上报 500 条数据还是 5 000 条数据，所用策略的 DRR 均为 100%；上报 500 条数据时，整表队列和分表队列的 RTM 相差不大，QCZ 均为 1；而上报 5 000 条数据时，分表队列的 RTM 少于整表队列，且分表队列的 QCZ 的最大值小于整表队列的 QCZ；这说明，随着上报总数据量的增加，消息队列中存入的上报数据也越来越多，分表队列的优势越来越明显，上报数据悉数被存入数据库中。笔者使用的方法能够保证数据的完整性，不会导致任何数据的丢失；同时，当大量数据上报时，消息队列最大使用长度均大于 5，这说明消息队列使用情况良好；因此，本方法是可靠的。

2) 对比表 4 和表 5 可以看出，在上报数据总量相同的情况下，无论是整表队列还是分表队列，测试用例 C 的 RTM 均比测试用例 B 的短。这说明，笔者方法可被广泛应用于若干客户端同时向一个中心数据库上报数据的情况，不仅性能稳定，而且效率也能得到保证。

4 结束语

笔者提出了一种基于消息中间件的数据同步更新方法，应用 Socket 通信机制传输报文，使用队列的数据结构接收各个部门数据库上报数据，并嵌入多线程技术，使得报文入队和报文出队 2 个进程同时运行，互不影响。通过实验验证了该方法是可行的，能够保证数据的完整性，执行效率较高，已应用于供电企业安全工器具监控系统中，取得了预期的效果。但在消息队列最大使用长度的设定方面，还存在人为设定的问题，还需进一步的研究并解决。

参考文献：

[1] 林源, 陈志泊. 分布式异构数据库同步系统的研究与应用[J]. 计算机工程与设计, 2010, 31(24): 5278-5281.

[2] 王军. 基于 XML 的分布式异构数据库同步研究与设计[D]. 南昌大学, 2007.

[3] 宫涛. 异构数据库准实时同步技术研究[D]. 华中科技大学, 2010.

[4] 徐晶, 许炜. 消息中间件综述[J]. 计算机工程, 2005(16): 73-76.

[5] 张骏, 刘松鹤. 多核、多线程处理器的低功耗设计技术研究[J]. 计算机科学, 2007(10): 301-305.

[6] 李霞, 陈松, 张国琰. 基于 Socket 的 VC++与 Flash 通信[J]. 重庆交通大学学报, 2011(2): 344-348.