第 35 卷 第 8 期 计 算 机 工 程 2009 年 4 月
Vol.35 No.8 Computer Engineering April 2009
·软件技术与数据库· 文章编号：1000—3428(2009)08—0079—03 文献标识码：A 中图分类号：TP311.13

异构数据库复制方法的改进与实现

李晓航[1]，胡晓鹏[1]，李 岗[2]
(1. 西南交通大学信息科学与技术学院，成都 610031；2. 西南交通大学电气工程学院，成都 610031)

摘 要：在基于数据库的大型分布式应用系统中，以较小的开销实现异构数据库间的复制是一个难题。分析原有异构数据库复制方法存在的不足，提出在数据访问接口中捕获数据变更 SQL 语句的改进方法，在不显著增加额外开销的情况下能完全捕获数据变更，同时保持变更的事务特性。实际应用验证了该方法的可行性。

关键词：异构数据库复制；SQL 重现；数据访问接口

Improvement and Implementation of Heterogeneous Database Replication Approach

LI Xiao-hang[1], HU Xiao-peng[1], LI Gang[2]
(1. School of Information Science and Technology, Southwest Jiaotong University, Chengdu 610031;
2. School of Electrical Engineering, Southwest Jiaotong University, Chengdu 610031)

【Abstract】It is difficult to deal with heterogeneous database replication with little overhead in large distributed applications. After analyzing original approach, this paper intends to capture database changes SQL in data access interface of applications. All database changes can be easily captured with new method and transaction property of database is guaranteed. Practical application verifies the feasibility of the approach .

【Key words】heterogeneous database replication; SQL reproducing; data access interface

1 背景介绍

基于数据库的大型分布式应用系统常采用分布式数据库存储应用数据，在各数据库之间保证数据的同步更新是关键。如果分布的各个数据库是同一种数据库，则可使用数据库管理系统(DBMS)提供的数据库复制(database replication)功能实现数据同步，但在异构数据库之间则行不通。

例如，在图 1 所示的一个 2 级应用系统中，公司总部数据库用来存放整个公司的数据，各个分公司数据库存放各个分公司的数据。

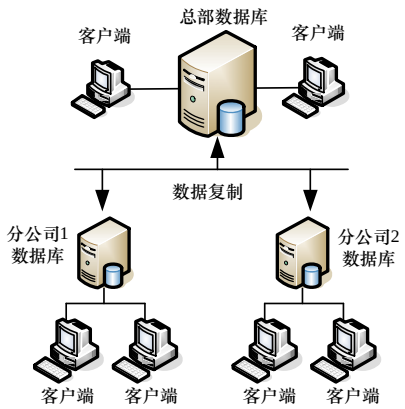

图 1 数据复制技术在 2 级分布式系统中的应用

日常业务主要发生在分公司中，生成的业务数据先保存在分公司的数据库中，然后更新到总部数据库中。在公司总部会对一些全局性的数据进行修改，例如数据字典，同时要更新各个分公司的数据库，以保持全局性数据的一致。考虑到分公司数据的存储需求、DBMS 的购买成本和今后的维护成本，公司总部使用 Oracle 9i，分公司使用 Microsoft Access 2003。在异构的 Oracle 和 Access 之间，无法使用 DBMS 提供的数据库复制功能。

数据库的复制问题是一个研究热点[1-5]。文献[1-2]讨论了 DBMS 中的数据库复制问题。文献[3-5]讨论了数据库复制在应用级的实现问题，其中文献[5]介绍的异构数据库复制方法较为新颖，在一定程度上能解决异构数据库的复制问题，但其提出的用触发器(trigger)来记录数据变更的方法存在不足，本文对文献[5]提出的方法进行了改进，改进后的方法克服了原有方法的不足且通用性更好。

2 SQL 重现方法分析

文献[5]提出了基于 SQL 重现的方法来实现数据库复制。SQL 重现方法的基本思想是：在需要复制的数据库表上设置触发器，其作用是将对表的更新操作还原为相应的 SQL 语句(即 INSERT, UPDATE, DELETE 语句，本文将这 3 种操作称为数据变更操作)序列，并记录在变更轨迹表中。事后将这些 SQL 语句序列传输到另外一台需要复制数据的数据库服务器上，在此服务器上按顺序执行这些 SQL 语句序列，以此实现数据的复制。以图 1 所示的场景为例，假设总部数据库中的数据因为业务需要发生了变化，要将这些变化同步到各分公司数据库中，就可以按图 2 的流程来实现数据变化的复制。

基金项目：铁道部科技研究开发计划基金资助项目“牵引供电自动化系统方案及技术规范研究”(2007J022)
作者简介：李晓航(1973－)，男，讲师、硕士，主研方向：软件工程，密码学；胡晓鹏，副教授、硕士；李 岗，讲师、博士研究生
收稿日期：2008-09-07 **E-mail：**xhli_scce@home.swjtu.edu.cn

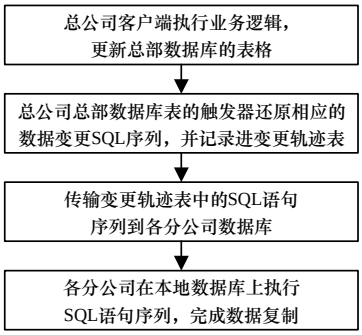

图 2　基于 SQL 重现方法的数据复制实现流程

文献[5]的方法存在以下不足：

(1)假如对数据库中每一个需要复制的表，要记录 3 种不同的变更，就需要分别编写 3 个触发器。如果一个应用系统中需复制很多表，编写和维护大量的触发器代码就非常繁琐。

(2)大量使用触发器对数据库性能的影响很大。在某些表上建立了触发器后，如果根据业务需要同时有多个并发用户对这些表进行查询和更新，会使数据库性能明显下降。

(3)用触发器记录数据变更无法保证变更的事务特性。为保证数据库中数据的一致性，应用程序对数据的更新必须采用事务机制。用触发器可以记录单个表的更新，但如果某个业务涉及多个表的更新操作，用触发器无法获知这些更新应该属于一个事务。无法保证事务机制，就无法保证复制后数据的一致性。

(4)一些桌面数据库不支持触发器。用触发器记录数据变更显然不适用于 Access, Foxbase, DBase 等不支持触发器的桌面数据库，使用范围受限。

3　对 SQL 重现方法的改进

3.1　数据变更的捕获

本文提出的方法仍基于 SQL 重现的思想，但在数据变更捕获方法上进行了改进。改进后的方法不在数据库表上设置触发器捕获数据变更 SQL 语句，而由应用程序的数据访问层捕获数据变更 SQL 语句。

对大型的分布式应用系统而言，良好的系统结构非常重要。多层(multi-tier)客户-服务器体系结构如图 3 所示。在该结构中，专门提供了一个数据访问层。数据访问层除了提供基本的数据库连接(connect)、断开(disconnect)等功能外，主要提供了执行数据变更 SQL 语句和返回数据集 SQL 语句(即 SELECT 语句)的功能。由于执行 SELECT 语句返回数据集的操作不会对数据库进行更新，因此无须考虑复制。在该结构中，应用程序业务逻辑对数据库中数据的更新一定会调用数据访问层的方法，因此，可以在这些方法中捕获数据变更 SQL 语句。

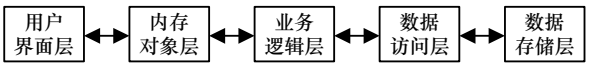

图 3　多层客户-服务器体系结构

假设数据访问层采用 ADO 技术访问数据库。在数据访问层为了能执行数据变更 SQL，通常须提供 2 个方法：ExecSQL 和 ExecSQLs。ExecSQL 方法用于执行一条数据变更 SQL 语句，ExecSQLs 方法用于在一个事务中执行多条数据变更 SQL 语句。它们的声明如下(采用 Delphi 语法)：

```
procedure ExecSQL(const aSQL, ReplDest: string);
procedure ExecSQLs(aSQLs: TStrings; const ReplDest: string);
```

其中，参数 aSQL 表示要被执行的一条 SQL 语句；参数 aSQLs 表示要在一个事务中被执行的多条 SQL 语句；参数 ReplDest 给出要被复制的目标数据库类型(即某种商用 DBMS 之一)，加入这个参数的主要目的是为了处理不同 DBMS 数据变更 SQL 语句存在的差异，在记录 SQL 语句之前即完成 SQL 语句的差异处理。以 ExecSQLs 方法为例(ExecSQL 与 ExecSQLs 方法类似)来介绍数据变更 SQL 语句的执行以及对这些 SQL 语句的捕获过程，其实现代码如下：

```
procedure TDataAccessLayer.ExecSQLs (const aSQLs: TStrings;
const ReplDest: string);
var    aTransID: string;
       i : Integer;
begin
       acnDatabase.BeginTrans;     //事务处理开始
       try
         for i := 0 to aSQLs.Count-1 do     //逐条执行 SQL
         begin
           acmEngine.CommandText := aSqls[i];
           acmEngine.Execute;
         end;
         aTransID  :=  CreateClassID;     //为要记录的多条 SQL
//创建唯一事务标识(GUID)
         for i := 0 to aSQLs.Count-1 do
//逐条记录(保存)SQL 语句，以便重现
          begin
         acmEngine.CommandText := CreateSQLForRepl(aTransID,
aSQLs[i], ReplDest);
         acmEngine.Execute;
         end;
         acnDatabase.CommitTrans;   //所有 SQL 语句执行成功，
//提交
       except
         acnDatabase.RollbackTrans; //SQL 语句执行有异常，回滚
         raise;   //执行错误，抛出异常
       end;
end;
```

其中，acnDatabase 是 ADOConnection 控件；acmEngine 是 ADOCommand 控件。当有多条 SQL 语句要执行时，通过一个循环来逐条执行。执行过的数据变更 SQL 语句必须保存起来，以便以后重现。本文将这些 SQL 语句保存到数据库的一个表中，该表的名字为 tb_updatelog，包括以下字段：

(1)ID：序号(整型)，是自动增长的数据类型，唯一标识一条记录；

(2)TransID：事务标识(字符型)，同一个事务中执行的 SQL 使用相同的事务标识；

(3)SQLText：执行过的 SQL 语句(字符型)；

(4)UpdTime：时间戳(日期时间类型)，记录 SQL 语句的执行时间。

在本方法中，所有 SQL 语句的执行及保存在同一事务中完成。如果在执行(不管是执行 SQL 语句还是执行保存 SQL 语句的操作)过程中发生任何错误，都会回滚，只有所有操作全部执行成功，才会提交到数据库中。

在同一事务中执行的(多条)SQL 语句，以后在其他数据库上重现执行的时候，也要放在同一个事务中执行，即保持事务特性。为达到这一目的，对每一个事务进行了标识，标识记录在表 tb_updatelog 的 TransID 字段中。在保存属于同一

事务的 SQL 语句之前，生成了一个全局唯一的 GUID[6](调用 CreateClassID 实现)，赋给 TransID 字段，GUID 的生成算法保证了 TransID 的唯一性。

在 ExecSQLs 方法中，调用 CreateSQLForRepl(aTransID, aSQLs[i], ReplDest)方法，生成保存 SQL 语句的 SQL 语句。在 CreateSQLForRepl 方法中不但生成了 SQL 语句，同时还根据参数 ReplDest 完成了 SQL 语句差异的处理。表 1 给出了保存在 tb_updatelog 表中的部分 SQL 语句。ID 为 1001 和 1002 的 SQL 语句在一个事务中，ID 为 1003 的 SQL 在一个事务中，ID 为 1004 和 1005 的 SQL 语句在一个事务中。

表 1　表 tb_updatelog 中保存的部分 SQL 语句

ID	TransID	SQLText	UpdTime
1001	{C9A7054A-9D26-4AF9-BC0C-AF1E3B1B72FC}	INSERT INTO tb_employee (EmpID, Name, Gender, Birthdate, Salary) VALUES (‘A001’, ’Tom’, ’Male’, ’1972-01-02’,500)	2005-05-23 9:55:23
1002	{C9A7054A-9D26-4AF9-BC0C-AF1E3B1B72FC}	INSERT INTO tb_resume (ID, EmpID, Detail) VALUES (421, ‘A001’,’Graduate from Harvard Univ and …’)	2005-05-23 9:55:23
1003	{5FB3884F-40E4-4BB9-95BA-A7548CB285D9}	UPDATE tb_employee SET Salary=800 WHERE EmpID = ‘B324’	2005-06-02 14:02:30
1004	{CDDFDE64-B21C-4AA2-B979-BC633F297C43}	DELETE FROM tb_resume WHERE EmpID = ‘C501’	2005-06-02 14:22:54
1005	{CDDFDE64-B21C-4AA2-B979-BC633F297C43}	DELETE FROM tb_employee WHERE EmpID = ‘C501’	2005-06-02 14:22:54

具体的应用系统并不一定要用多层架构。对要进行数据库复制的应用系统而言，只要有一个统一的数据访问接口，就可以在这个接口中捕获数据变更 SQL 语句。分析 ExecSQLs 方法的实现代码可以发现，在数据访问接口中增加数据库变更 SQL 的捕获功能只需要很小的额外开销，因为在数据访问接口中 SQL 语句本来是现成的，只须进行适当的差异处理后保存起来即可。

3.2　数据变更的传输和同步

在一台数据库服务器上保存了数据变更 SQL 语句，为了在其他数据库上能重现这些 SQL 语句，必须先将这些 SQL 语句(即表 tb_updatelog 的内容)传输到其他的数据库服务器上。如何传输这些 SQL 语句，用自动(定时)方式还是人工干预方式，取决于具体的应用。在传输 SQL 语句时，同样要考虑事务特性。不能把一个事务中 SQL 语句分开传输，否则可能导致潜在的事务特性无法保证。为减少 SQL 语句传输占用的带宽，可以将多个事务的 SQL 语句打包在一起，并将打包后的数据压缩传输。SQL 语句传输到某台数据库服务器上以后，可以将它们保存在一个结构类似于 tb_updatelog 的表中，也可以把它们保存在内存缓冲区中。本文假设将 SQL 语句保存在 tb_pendinglog 表中，其结构和 tb_updatelog 类似。

数据变更的同步(即重现 SQL 语句的执行)可以有不同的策略。可以创建一个应用进程持续地监视 tb_pendinglog 表的变化，如果 tb_pendinglog 表有记录，就按事务执行其中的 SQL 语句。同步后的 SQL 语句序列已无作用了，可以从 tb_pendinglog 表删除。

4　要点问题分析

4.1　SQL 语句差异的处理

对于异构数据库的复制，必须解决异构数据库之间 SQL 语句的差异问题。例如，在 SQL Server 中，日期类型的字段值可以用一个形如“2007-01-10”的简单字符串来提供；而在 Oracle 中要对某个日期类型字段进行更新时，在 SQL 语句中该日期类型字段值必须用 PL/SQL 提供的 TO_DATE 函数进行转换。假设表 tb_test 中有一个日期型字段 birthdate，如果是 ORACLE，则对应的更新操作 SQL 为 UPDATE tb_test SET birthdate = TO_DATE (‘1990-01-10’, ’YYYY-MM-DD’) WHERE ID= ‘0005’。即将 birthdate 字段值更新为 1990 年 1 月 10 日，’YYYY-MM-DD’表示日期格式。对于 SQL Server，对应的更新操作 SQL 为 UPDATE tb_test SET birthdate = ‘1990-01-10’ WHERE ID= ‘0005’。

在捕获数据变更 SQL 语句时，应知道以后要重现的数据库的类型，从而在保存时按照数据库类型完成 SQL 语句的转换。在具体的基于数据库的应用中，如果使用了多种异构数据库，则保存 SQL 语句时应将 SQL 转换成差异最大(最复杂)的数据库所支持的 SQL 语句。例如一个应用系统使用了 Oracle, SQL Server 和 Access 数据库，并且它们之间要复制数据，则应将 SQL 语句保存为 Oracle 格式的 SQL 语句。以后在 SQL Server 或 Access 数据库上执行这些 SQL 语句时，将 Oracle 格式的 SQL 语句转换为 SQL Server 或 Access 格式的 SQL 语句。由复杂格式的 SQL 语句转换为相对简单格式的 SQL 语句，用程序实现并不难。

4.2　一对多的复制

在一些具体的应用系统中，一个数据库的变化往往需要反映在多个数据库中。如图 1 所示，总部数据库的变化希望及时反映在各分公司数据库中，意味着在每个分公司数据库上都要重现 1 次总部数据库上保存的数据变更 SQL 语句序列，而且要避免重复复制。要实现这样的功能，总部数据库上保存的数据变更 SQL 语句序列传输到一个分公司后，不能被删除。各个分公司的数据库要避免重复复制，应记录哪些 SQL 语句已经重现过了。为了尽可能增加复制的透明性，在总部数据库上记录哪些分公司数据库已经重现了 SQL、哪些还没有重现并不是最佳选择，这一记录工作可以由分公司数据库自身完成。在 tb_updatelog 中有一个 ID 序号字段，一个分公司数据库完成了一次复制后，应记录下本次复制的最大 ID 序号值，下次复制时应从上次结束的位置(ID 序号加 1)处开始复制，就避免了重复问题。

如果存在双向的复制需求，那么在每个要进行双向复制的数据库上都要建立表 tb_updatelog 和 tb_pendinglog，实现的思路和单向复制一样，因为双向复制可以看作是 2 个单向复制。另外图 1 给出的是一个 2 级应用系统的数据库复制场景，如果是 2 级以上的应用系统需要进行数据库复制，实现的方法类似。

4.3　推(push)和拉(pull)

在数据库复制范型中有推和拉 2 个重要概念。推和拉实际上是要明确复制双方哪一方是同步的发起方。以图 1 的场景为例，如果是总部数据库到分公司数据库的复制，则发起方最好是分公司数据库，这是拉；如果是分公司数据库到总部数据库的复制，则发起方最好是分公司数据库，这是推。推和拉实际上是相对的，究竟用推还是拉完全取决于具体的应用场景。

5　结束语

本文提出的异构数据库复制方法在多层架构应用程序的数据访问层进行数据变更捕获，因此，对应用程序的业务逻辑层是不透明的，业务逻辑确定它产生的数据更新是否应该复制到其他服务器上，而对于内存对象层和用户界面层，则完全感觉不到数据复制的存在。笔者开发的铁路牵引供电管

(下转第 84 页)