申请上海交通大学工程硕士学位论文

**基于MySQL复制改进的多主复制数据库扩展实现**

|  |  |
| --- | --- |
| 学校代码： | 10248 |
| 作者姓名： | 朱振 |
| 学 号： | 1090379200 |
| 第一导师： | 陈昊鹏 |
| 第二导师： |  |
| 学科专业： | 软件工程 |
| 答辩日期： | 年 月 日 |

上海交通大学软件学院

2013年3月

A Dissertation Submitted to Shanghai Jiao Tong University

for Master Degree of Engineering

**IMPLEMENTATION OF MULTI-MASTER REPLICATION DATABASE EXTENSION BASED ON MYSQL REPLICATION**

|  |  |
| --- | --- |
| University Code： | 10248 |
| Author： | Zhu Zhen |
| Student ID: | 1090379200 |
| Mentor 1： | Chen Haopeng |
| Mentor 2: |  |
| Field： | Software Engineering |
| Date of Oral Defense： |  |

School of Software

Shanghai Jiaotong University

March, 2013

**上海交通大学**

**学位论文原创性声明**

本人郑重声明：所呈交的学位论文，是本人在导师的指导下，独立进行研究工作所取得的成果。除文中已经注明引用的内容外，本论文不包含任何其他个人或集体已经发表或撰写过的作品成果。对本文的研究做出重要贡献的个人和集体，均已在文中以明确方式标明。本人完全意识到本声明的法律结果由本人承担。

学位论文作者签名：

日期： 年 月 日

**上海交通大学**

**学位论文版权使用授权书**

本学位论文作者完全了解学校有关保留、使用学位论文的规定，同意学校保留并向国家有关部门或机构送交论文的复印件和电子版，允许论文被查阅和借阅。本人授权上海交通大学可以将本学位论文的全部或部分内容编入有关数据库进行检索，可以采用影印、缩印或扫描等复制手段保存和汇编本学位论文。

**保密□**，在 年解密后适用本授权书。

本学位论文属于

**不保密**。

（请在以上方框内打“**√**”）

学位论文作者签名： 指导教师签名：

日期： 年 月 日 日期： 年 月 日

**基于MySQL复制改进的多主复制数据库扩展实现**

**摘 要**

随着互联网系统的发展，系统的访问量、并发量和数据量都在剧烈增加，性能问题愈发严重。数据库作为互联网系统数据的可靠载体，其性能直接影响着互联网系统的整体性能，因此如何提高数据库的性能变得非常有研究意义。目前业界主要使用基于MySQL主从复制的机制进行数据库架构扩展，从而解决数据库性能问题。由于主从复制机制中只支持单台主服务器，无法从根本上解决单台主服务器写操作的性能问题。

本研究对MySQL主从复制机制进行改进，在主从复制时通过采用串行消息队列解决数据冲突问题，从而支持多台主服务器同时进行数据复制，使数据库的写操作压力得到分散，从根本上解决数据库性能问题。

本研究的主要工作有如下几个方面：

1. 分析国内外多主复制技术的研究现状，确定目前业界没有成熟的MySQL多主复制实现方案，从而提出基于MySQL主从复制改进的多主复制机制的研究目标和内容。此外，通过对Oracle多主复制技术的分析，确定在多主复制机制中借鉴Oracle Streams的消息队列设计思想。
2. 介绍常用的数据扩展方式，分析这些扩展方式的优缺点。同时，对淘宝数据库架构和多主复制数据库架构进行分析和对比，说明多主复制数据库架构的产生意义。
3. 介绍MySQL多主复制数据库扩展的应用场景和总体设计。首先分析MySQL多主复制的应用场景，多主复制架构可以替代垂直切分或者水平切分，也可以作为垂直切分、水平切分以及垂直和水平联合切分的进一步升级方案。其次阐述MySQL多主复制的总体设计，主要包括唯一性主键设计、解析从服务器设计、消息队列设计、SQL处理进程设计以及集合数据库设计。
4. 介绍MySQL多主复制数据库扩展的具体实现。首先介绍MySQL多主复制的需求背景，从基于一主多从复制的7个基本步骤扩展出多主复制的9个重要步骤。其次介绍从主服务器和解析从服务器的实现，将从服务器改进为解析从服务器，修改解析从服务器SQL线程解析中继日志的逻辑。然后介绍消息中间件的实现，借鉴Oracle多主复制的设计理念，提出使用消息队列串行的进行数据复制，在不影响性能的情况下，合理的解决了数据冲突问题。最后说明SQL处理进程和集合数据库的实现，通过词法和语法解析工具Java Compiler Compiler（简称JavaCC）解析SQL语句。

最后对基于MySQL复制改进的多主复制数据库扩展进行并发读、写以及复制实验，通过实验可以证明，对于读和写的操作，基于MySQL复制改进的多主复制数据库扩展可以有效的降低整个数据库系统的读写压力，从根本上解决数据库性能问题。对于复制操作，由于使用串行消息队列的处理方式，一般会有小于5秒的延时，考虑到集合数据库主要进行数据统计和分析，因此小于5秒的复制延迟不会造成较大问题。因此基于MySQL复制改进的多主复制数据库扩展完全可以应用于出现数据库性能问题的互联网系统中。

关键词 主从复制，一主多从，多主一从，多源复制，多主复制

**IMPLEMENTATION OF MULTI-MASTER REPLICATION DATABASE EXTENSION BASED ON MYSQL REPLICATION**

**ABSTRACT**

With the development of the Internet system, the amount of access to the system, the concurrency and the amount of data is dramatically increasing. In this era of “Big Data”, as the carrier of the Internet system, database is often in the spotlight of people’s discussions, mainly for the reason that its performance directly affects the performance of the overall Internet system. Therefore, the performance improvement of database is of great importance. For Internet companies, one of the most effective ways in improving database performance is to extend database framework. The current popular database extension methods are Master-Slave Replication, Write-Read Separation, Vertical Segmentation, and Horizontal Segmentation, etc. These methods can indeed improve the performance of database in the general scene, for example, Vertical Segmentation and Horizontal Segmentation are adopted by Taobao to resolve database performance issues. However, these above-mentioned extension methods are not “universal syrup”, with the possibility of failing to solve all database performance issues. Therefore, with it comes to the scenario when Vertical Segmentation and Horizontal Segmentation are not fully competent, new database extension methods need to be introduced. According to this actual need, this article proposes the Multi-Master Replication database extension method based on MySQL Replication.

This Multi-Master Replication database extension method based on MySQL Replication refers to Oracle Streams Replication’s Advanced Message Queuing technology (Message Queue, i.e. MQ) in theory; meanwhile, this method improves the current MySQL Master-Slave Replication technology by modifying the operation in the Slave server’s SQL thread parse relay log, thereby resolving the data conflict issues caused by Multi-Master Replication.

This article mainly discusses the following aspects:

1. MySQL Master-Slave Replication technology improvement: upgrading Slave server into parsing Slave server, meanwhile modifying the logic of Slave server’s SQL thread parse relay log, by adding in SERVER ID filter to avoid data inconsistencies in the collective database.
2. Considering the data conflict issues caused when replicating data from Multi-Master servers to collective database, learning from the design concept of Oracle Streams Multi-Master Replication, it is proposed to use serial data replication by Message Queue to resolve data conflict issues, without affecting the overall performance; meanwhile, it is a significant improvement that Message Queue can automatically persist messages, in terms of ensuring the stability of data replication.
3. As in the Multi-Master framework, there are multiple data replication initiation points, if the primary key in each initiation point’s database table is auto-incremental, this will inevitably lead to primary key conflict issues in the collective database. Thus, the primary key cannot be auto-incremental in the Multi-Master framework’s database table. This article provides a unique primary key generation tool in the implementation phase to generate globally unique primary key, to avoid primary key conflict issues.
4. SERVER ID filter is introduced in order to avoid data inconsistencies in Multi-Master Replication framework. The addition of SERVER ID filter cannot be realized by simple string manipulation, but by specific MySQL statement lexical or syntax parsing. Therefore, this article proposes to use Java Complier Complier (JavaCC), the lexical and syntax parsing tool, to realize the parsing functionality and SERVER ID addition. In the implementation phase, the challenge of how to add SERVER ID filter into SQL statements has been resolved by upgrading the open source SQUL parsing tool jSQLParser.
5. Because of the introduction of SERVER ID filter, in the collective database in Multi-Master Replication framework, SERVER ID filter needs to be added into each table, without the filter value being NULL. In this way, data consistency in Multi-Master Replication is ensured.

**Keywords** Master-Slave Replication, Master-Slaves, Masters-Slave, Multi-Source Replication, Multi-Master Replication

**目 录**

[1 绪 论 1](#_Toc352204332)

[1.1 研究背景 1](#_Toc352204333)

[1.2 国内外研究现状 1](#_Toc352204334)

[1.2.1 MARIADB多主复制 2](#_Toc352204338)

[1.2.2 ORACLE STREAMS复制 2](#_Toc352204339)

[1.3 论文的主要研究内容 4](#_Toc352204340)

[2 数据库架构分析 5](#_Toc352204341)

[2.1 常用数据库扩展方式介绍 5](#_Toc352204343)

[2.1.1 主从复制和读写分离 5](#_Toc352204347)

[2.1.2 垂直切分 6](#_Toc352204348)

[2.1.3 水平切分 7](#_Toc352204349)

[2.1.4 垂直和水平联合切分 8](#_Toc352204350)

[2.1.5 MYSQL CLUSTER 10](#_Toc352204351)

[2.2 淘宝数据库架构分析 10](#_Toc352204352)

[2.3 多主复制数据库架构分析 12](#_Toc352204353)

[2.4 本章总结 13](#_Toc352204354)

[3 多主复制数据库扩展的设计思路 14](#_Toc352204355)

[3.1 应用场景 14](#_Toc352204358)

[3.2 总体设计 17](#_Toc352204359)

[3.2.1 唯一性主键设计 19](#_Toc352204362)

[3.2.2 解析从服务器设计 20](#_Toc352204363)

[3.2.3 消息队列设计 22](#_Toc352204364)

[3.2.4 SQL处理进程设计 24](#_Toc352204365)

[3.2.5 集合数据库设计 24](#_Toc352204366)

[3.3 本章总结 25](#_Toc352204367)

[4 多主复制数据库扩展的具体实现 26](#_Toc352204368)

[4.1 需求背景 26](#_Toc352204370)

[4.2 主服务器和解析从服务器实现 30](#_Toc352204374)

[4.2.1 日志分析 30](#_Toc352204375)

[4.2.2 具体实现 37](#_Toc352204376)

[4.2.3 实现步骤 43](#_Toc352204377)

[4.3 消息中间件实现 46](#_Toc352204378)

[4.3.1 消息队列 46](#_Toc352204381)

[4.3.2 具体实现 47](#_Toc352204382)

[4.3.3 实现步骤 48](#_Toc352204383)

[4.4 SQL处理进程和集合数据库实现 49](#_Toc352204384)

[4.4.1 语法解析 49](#_Toc352204385)

[4.4.2 具体实现 51](#_Toc352204386)

[5 实验与总结 54](#_Toc352204387)

[5.1 实验环境 54](#_Toc352204391)

[5.2 实验方案 54](#_Toc352204392)

[5.2.1 读操作测试 54](#_Toc352204393)

[5.2.2 写操作测试 55](#_Toc352204394)

[5.2.3 数据复制测试 56](#_Toc352204395)

[5.3 实验分析 57](#_Toc352204396)

[5.4 总结与展望 57](#_Toc352204397)

[参考文献 59](#_Toc352204398)

[致 谢 61](#_Toc352204399)

[攻读学位期间发表的学术论文目录 62](#_Toc352204400)

1. **绪 论**

## 研究背景

随着信息技术的不断发展，以及互联网的日益普及，人们的生活正在发生着巨大的变化，传统生活方式正在面临着新兴生活方式的挑战，而这场变革的发起者就是人们每天都在使用的各种互联网系统，例如，百度地图、淘宝网、12306火车票售票系统、当当网、京东商城、优酷视频、新浪微博等。

现在，外出的时候使用百度地图查询具体的交通路线变成大家经常要做的功课，通过输入起点和终点，就可以得到一份详尽的交通路线。此外，如果手机支持GPRS，甚至可以通过百度地图的手机应用进行线路导航；商场和超市一直是人们购物的首选，随着淘宝网和京东商城的出现，这一习惯被打破了，网上商城不仅商品种类繁多，而且价格还比商场和超市便宜。完全可以足不出户，就买到自己所需要的商品；对于中国人来说，买火车票是一件特别麻烦的事情，遇到过年过节，很多人甚至要在售票窗口外彻夜排队。现在这一情况完全被改变了，通过12306火车票售票系统，坐在电脑面前点击鼠标就可以买到自己想要的票……

随着这些互联网系统的普及和发展，系统访问量、并发量和数据量都是与日俱增，性能问题也就日益严重。例如，2012年双十一期间，淘宝天猫商城在发起降价促销活动时就出现了严重的系统问题，原因就是当天0点左右系统最为繁忙，并发量达到了顶点，造成了系统问题；2012年十月黄金周前，由于网上购票的人太多，通过12306火车票售票系统购买车票时，经常出现系统忙碌以及无法完成交易的情况。产生上述问题的原因很多，例如WEB服务器性能问题、负载均衡问题、系统设计问题等，其中数据库性能问题通常是需要被优先解决的[1]。作为互联网系统中数据的可靠载体，数据库的性能问题一直是互联网系统的瓶颈问题，困扰着众多的DBA和架构师。对于如何通过数据库扩展来提高性能，就变成非常有研究意义的课题。

## 国内外研究现状

目前，解决数据库性能问题的方法很多，但一般情况下都会采用基于MySQL复制机制的方法来实现数据库性能的优化。当前大多数互联网系统采用的都是一主多从的数据库架构，一方面是因为一主多从的数据库架构能够很好的解决数据库性能问题，例如通过MySQL复制实现负载均衡以及读写分离等[2]；另一方面，由于目前MySQL复制机制只能支持一个主服务器向一个或多个从服务器进行数据复制，不能支持多个主服务器向一个或多个从服务器进行数据复制，因此大多数MySQL数据库架构只能采用一主多从的设计。如果要实现多个主服务器向一个或多个从服务器进行数据复制的功能，最需要解决的就是数据库冲突问题。MySQL 5.1开始的基于行复制的方式确实为实现这个功能作出了努力[3]。



### MARIADB多主复制

MySQL的创始人Michael Widenius(Monty)正在开发MariaDB数据库，MariaDB数据库是一个采用Maria存储引擎的MySQL分支版本。通过askmonty网站和Monty的博客了解到MariaDB 10.x将包含多主复制的功能[4][5]。这个功能的主要开发者是来自于淘宝网的DBA彭立勋(P.Linux)。从彭立勋的博客了解到这个功能的代码已经合并到MariaDB的主干上，将在10.x版本中出现[6]。目前的MySQL复制机制只能支持从服务器从一台主服务器上复制数据，而彭立勋提交的多主复制功能可以允许从服务器同一时间复制多个主服务器上的数据。针对每个主服务器，从服务器都有各自独立的一套I/O线程、SQL线程等，确保每个主服务器上的数据都能稳定和高效的被复制。

通过对这个功能的进一步了解后发现，这种方式的多主复制确实可以将多个主服务器上的数据变更复制到一个或多个从服务器上，但是复制时必须保证没有冲突发生，如果有冲突发生，相关的从服务器SQL线程会停止工作，需要DBA手工解决冲突后再运行。考虑到多主复制环境中，出现冲突的情况非常普遍，没有冲突解决能力的数据库根本无法保障系统数据的准确性。当然MariaDB数据库确实也有一些成功使用的案例，特别是随着Sun被Oracle收购后，很多系统都考虑由MySQL向MariaDB的迁移。对于需要7\*24小时稳定运行的互联网系统来说，目前基本上业界都在使用MySQL数据库，在没有太多MariaDB数据库使用和维护经验的情况，贸然将MySQL数据库替换为MariaDB数据库风险太高。

### ORACLE STREAMS复制

关系型数据库领域的霸主Oracle提供了多主复制技术，主要通过Oracle Streams复制技术来实现[7]。Oracle Streams是分布式环境中进行数据复制和信息共享的解决方案。Oracle Streams支持双向复制和点对点复制[8]。双向复制指的是两个数据库间相互复制。点对点复制相当于双向复制的扩展，可以实现两个以上的数据库间相互复制[9]。通过修改点对点复制时相应的配置参数可以实现多主复制的功能，如图1-1所示。Oracle Streams特性利用活动和归档的重做日志，捕捉数据库中出现的改动。Oracle Streams捕捉这些改动并经过适当的格式化后将其存储到队列中。然后，Oracle Streams将这些改动传播到其他数据库，并把改动应用到目标数据库。使用Oracle Streams特性，可以在一个Oracle数据库内、两个Oracle数据库间、多个Oracle数据库间或一个Oracle数据库和一个非Oracle数据库间捕捉、传播和应用信息[10]。Oracle Streams从Oracle 9i Database Release 2开始出现，到现在已经有很多成功的案例。在Oracle Steams中，最小的信息单元是消息。在对数据库进行修改时，消息可以被捕捉到。通常情况下，消息包括数据库中的表的DML操作，例如：新增记录、删除记录、修改记录等。此外消息也可包括DDL操作，例如：创建索引、修改表结构等[11]。对于增加一个数据文件、启动备份或者卸载表空间等，则不包含在消息中。Oracle Streams使用消息队列来交换消息，队列支持插入消息到队列中或者让消息出队等操作。此外，Oracle Streams能够自动检测和处理数据冲突。当冲突无法解决时Oracle Streams必须按照具体的业务需求去处理数据冲突[12]。



图1-1 Oracle Streams

Fig.1-1 Oracle Streams

通过对Oracle Streams复制技术的分析，发现其实时性和稳定性较高，对于数据冲突问题也有比较好的解决方案。不过一般互联网系统都使用MySQL数据库作为数据载体，Oracle数据库由于费用原因，一般都应用在银行或电信行业。互联网系统建设时，软、硬件成本也是需要重点考虑的问题，将MySQL替换为Oracle的经济成本非常高，且后期还有大量的支持和维护费用，例如，淘宝的数据库早已开始从Oracle向MySQL的迁移。不过，Oracle Streams复制中的消息队列技术对于MySQL多主复制却是有很高的借鉴意义。

## 论文的主要研究内容

本文主要研究基于MySQL主从复制机制进行改进的多主复制数据库扩展方式。针对MySQL目前只支持一个主服务器向一个或多个从服务器进行数据复制，通过改进MySQL的复制机制，使其支持多个主服务器向一个或多个从服务器进行数据复制，并且巧妙的通过消息队列来解决多主复制时引起的数据冲突问题。

文章的结构安排如下：

第一章主要介绍了MySQL数据库多主复制的研究背景和国内外的研究现状。

第二章主要对目前主流的数据库扩展方式进行介绍，并对互联网系统的优秀代表淘宝网的数据库架构进行对比分析，凸显MySQL多主复制数据库扩展方式的研究意义。

第三章详细介绍MySQL多主复制数据库扩展方式的设计思路。

第四章详细介绍MySQL多主复制数据库扩展方式的具体实现。

第五章是实验结果以及对研究课题的总结。

1. **数据库架构分析**

## 常用数据库扩展方式介绍

目前，解决数据库性能问题的扩展方式很多，本文选取了几种常见的数据库扩展方式进行简单介绍，通过介绍和比较这些数据库扩展方式来说明多主复制数据库扩展方式的研究意义。



### 主从复制和读写分离

MySQL主从复制机制主要是：主服务器根据操作变更生成二进制日志事件，保存在二进制日志文件中，二进制日志事件通过I/O线程传递到从服务器上，并通过从服务器的SQL线程在从服务器上执行，从而实现数据的复制这么一个过程[13] [14]。

MySQL主从复制实际上是一个异步复制的过程，在整个复制过程中主要三个核心线程来完成复制操作，主服务器上的I/O线程以及从服务器上的I/O线程和SQL线程[15]，MySQL主从复制的主要流程是[16]：

1. 从服务器的I/O线程成功连接主服务器，请求二进制日志文件指定位置之后的日志内容。
2. 主服务器接收到从服务器的I/O线程请求后，主服务器的I/O线程根据请求信息读取指定位置之后的二进制日志内容返回给从服务器的I/O线程，除了日志外，主服务器还将返回日志所属的二进制日志文件名称和在二进制日志文件中的所处位置传递给从服务器。
3. 从服务器的I/O线程收到主服务器发送的日志信息后，将日志信息依次写入从服务器的中继日志文件的最末端，同时将获取的信息所属的二进制日志文件名称和在二进制日志文件中的位置记录到master-info文件中。
4. 从服务器的SQL线程一旦发现中继日志文件有更新，会马上解析中继日志文件中的新内容，解析成主服务器上执行的具体SQL语句，并执行这些SQL语句。

通过MySQL复制机制完成数据复制后，可以通过读写分离机制来合理分配主服务器和从服务器的具体工作任务。所有的增加、删除和修改操作统一放在主服务器上执行，这些DML操作会复制到从服务器，因此只有在主服务器上执行才能保证主从数据的一致性。如果放在从服务器上执行，由于从服务器之间并没有数据复制机制（常规复制架构），势必造成从服务器之间的数据不一致；所有的查询操作统一放在从服务器上执行，由于绝大多数数据库系统的读操作要远远大于写操作，将大量的读操作转移到多个从服务器上执行，系统的性能得到巨大的提升。

目前，在实际应用中，MySQL主从复制机制是使用最为广泛的数据库扩展方式，同时这种扩展方式也是成本最为低廉的方式，大多数数据库应用系统都很难拒绝这种物美价廉的数据库扩展方式。主从复制机制有多种实现架构，例如：常规复制架构(Master-Slaves)、Dual Master复制架构(Master-Master)、级联复制架构(Master-Slaves-Slaves)和Dual Master与级联复制结合架构(Master-Master-Slaves)等，通过名称就可以发现这些架构实际上都是常规复制架构的扩展。特别说明下Dual Master复制架构，搭建Dual Master复制架构并不是为了让两个主服务器都提供写服务，一般情况下只会开启其中一个主服务器，另外一个主服务器只会提供读服务或者完全不提供任何服务，只作为备用服务器而已。本文所研究的多主复制的数据库扩展方式，具体实现也是基于MySQL的主从复制技术。

### 垂直切分

如果数据库的写操作非常频繁，这种情况下，仅仅使用主从复制和读写分离机制进行数据库扩展，可能效果就不那么明显了。当主服务器的写操作压力越来越大时，通过主从复制机制去添加更多的从服务器就没有太大的意义了，因为这种方式释放的只是数据库的读操作压力，对于数据库的写操作压力没有任何的释放作用。如果主服务器大部分时间都在处理写操作请求的情况下，从服务器也在长时间的进行主从数据复制，处理读操作请求的时间自然也变的更少，这时候使用主从复制机制实现数据库扩展的预期回报也变得越来越小。

这种情况下，就应该考虑去释放数据库的写操作压力。解决这个问题，DBA和架构师一般会采取垂直切分数据库的扩展方式。数据库垂直切分的意思是将数据库中的表切分到多个数据库中去。切分时并不是随意的将表切分到不同的数据库中，而是按照一定的原则。一般来说，互联网系统是由多个功能模块组成，理想状态下，各个模块之间应该相对独立。例如，权限管理模块、参数管理模块、客户资料管理模块、订单管理模块等等。每个模块都有其对应的一类数据库关联表，这类关联表之间的耦合度很高，经常需要进行JOIN操作，如果将这些关联表切分到多个数据库中，势必会造成跨数据库的JOIN操作，同时带来数据操作时事务复杂度的提升，因此，拆分时不能选择耦合度高的关联表进行拆分。考虑到各个功能模块之间的耦合度较低，相互影响较小，如果按照模块进行拆分，可以避免跨数据库的JOIN操作以及复杂的分布式事务等问题，因此根据模块进行垂直切分是一个比较好的拆分方式。例如上面的例子，由于客户资料管理和订单管理属于系统中的两个核心模块，用户访问量和数据量都很大，而这两个模块之间的关联性相对较小，因此将这两个模块切分到不同的数据库中；对于权限管理和参数管理这两个模块，由于不是系统的核心业务，因此访问量和数据量都不大，既可以切分到两个数据库中，也可以将这两个模块切分到一个数据库中。

垂直切分的优点是这种数据库扩展方式简单明了，切分规则明确，维护方便，整合容易。缺点是，很难有系统能够做到各个功能模块之间没有任何关联，对应的表之间没有任何的JOIN操作，例如上面的例子，客户资料管理和订单管理这两个模块的表之间经常需要进行JOIN操作。如果切分，必然引起跨数据库的JOIN操作以及事务复杂度的提升。如果不进行切分，合并在一起，模块间的强耦合没有拆解，日后系统无法支撑时，再进行垂直切分，难度就更大。垂直切分经常造成表的关联无法在数据库中完成，需要在系统使用程序实现。一些业务核心表由于没有发生任何变化，访问量和数据量仍然巨大，依然是系统的性能瓶颈。需要使用分布式事务来保障跨数据库的数据操作，处理过程相对复杂。

### 水平切分

由于垂直切分需要特定的切分条件，一般情况下系统进行垂直切分后都不能达到满意的结果，数据库仍然存在性能瓶颈。从实际使用经验来看，大多数数据库应用系统在解决数据库性能问题时都经历了从简单主从复制到垂直切分，再到水平切分的过程。水平切分可以简单理解为将某个访问量或者数据量很大的表按照某个字段的某种规则切分到多个表或者多个分区中，每个表或者分区包含一部分数据。由此可以看出，水平切分对于字段或者规则的选取非常重要。

水平切分一般通过分区或者分表来实现[17]：

1. 分表

分表这种切分方式并不依赖于特定的技术，更多的是一种逻辑层面的规划。例如：系统日志表，考虑这个表一般都是根据时间来查询，所以可以按照时间来进行分表。这里以“天”作为单位，每天创建一张表，如：TB\_LOG20130101, TB\_LOG20130102等，每张表的表名基本一致，只是最后的时间后缀不同而已。创建了分表以后，对系统日志表的操作就需要根据时间来进行选择。实际上每个分表都是独立的物理表，分表与分表之间只有逻辑关系，没有任何物理关系。

1. 分区

这种切分方式依赖于MySQL分区技术，需要注意的是，MySQL分区技术并不是在存储引擎层完成的，因此并不是所有存储引擎都支持分区技术。例如，学生成绩表，可以根据成绩进行分区，60分以下的一个分区；60-70分一个分区；70-80分一个分区；80-90分一个分区；90-100分一个分区，创建5个分区，如图2-1所示。

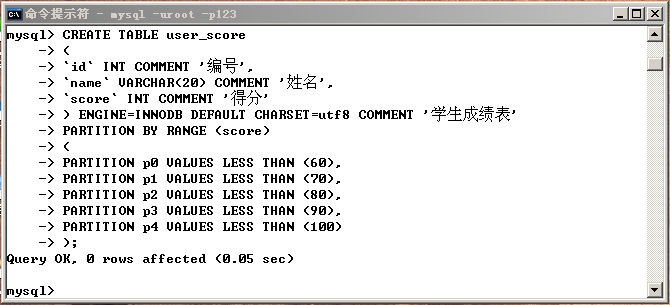


图2-1 分区表

Fig.2-1 Partition Table

当用户对分区表进行数据操作时，不需要像分表一样指定表名。MySQL会自动进行处理，根据建表时的分区规则将相关数据保存到正确的分区或者从正确的分区返回数据。对于用户来说，操作分区表和操作普通表是没有任何区别的。目前MySQL主要支持：RANGE分区、LIST分区、HASH分区和KEY分区。使用哪种分区方式要看具体的应用场景。

水平切分能够很好的解决访问量或者数据量都很大的表给系统带来的性能瓶颈问题。从垂直切分到水平切分，这是一个系统成长的见证。但是水平切分也有其明显的问题，如果采用分表的切分方式，系统中涉及分表操作的代码需要修改，对于已经稳定运行的互联网系统来说，进行系统升级，不仅工作量巨大，而且会影响系统的稳定性；如果采用分区的切分方式，这种方式需要特别的强调，大多数人都认为使用分区，能够提高数据库查询性能，实际上分区有其应用的特定场景。如果数据库是在线分析处理(On-Line Analytical Processing)（简称OLAP）系统，分区的确可以提高查询性能，因为OLAP应用经常需要去查询一张数据量巨大的表，且查询条件相对比较固定。采用分区之后，只需要去查询相应的分区即可，查询性能自然大幅度提高；如果数据库是在线事务处理(On-Line Transaction Processing)（简称OLTP）系统，大多数的查询只需要返回表中的很少的数据，B+树结构可以很好的完成检索，不需要分区的帮助，并且设计不好的分区会带来严重的性能问题，可能会造成查询操作需要搜索所有分区的情况。分区主要用于高可用性，利于数据库的管理，不能寄希望于通过分区解决一切性能问题。

### 垂直和水平联合切分

通过上面两节的介绍，对垂直切分和水平切分都有了基本的了解。在实际应用的场景中，除了一些负载压力不大或者业务逻辑相对简单的系统可以通过只使用垂直切分或者只使用水平切分来完成数据库的扩展，其他一些负载压力大或者业务逻辑很复杂的系统很难做到只使用垂直切分或者只使用水平切分来完成数据库的扩展。

一般情况下，由于垂直切分成本低、代价小，因此大多数DBA或者架构师都会优先选择进行垂直切分，只有这种方式是最物美价廉的。垂直切分之后，系统中的某些核心业务表仍然无法减轻压力，因为垂直切分一般都是对功能模块进行拆解，例如：权限管理模块、参数管理模块、客户资料管理模块、订单管理模块等等。经过垂直切分，权限管理模块和参数管理模块存储在一个数据库中，客户资料管理模块和订单管理模块分别存储在独立的数据库中，由于这两个模块涉及系统核心业务，系统的访问量和数据量都是巨大的。这样切分之后，短时间内，确实能够提高整体性能，但每一个数据库应用系统的负载都不是一夜之间激增上来的，而是一步一步增长起来的。随着系统的发展，功能越来越强，用户数越来越多，数据量越来越大。对于客户资料管理模块和订单管理模块来说，两个模块对应表的数据量增大，访问量也增多，性能问题又开始出现。之前经过垂直切分的数据库又开始不堪重负。如果再次进行垂直切分，首先必须对现有业务继续进行分解，问题是经过第一次切分后，剩余的各个模块耦合度已经很高，否则第一次切分时这些模块就已经被拆解掉了，也不会遗留到现在；其次，每次遇到性能问题就细化模块，进行拆解，很多关联性很强的模块被切分开，造成跨数据库JOIN操作和事务复杂化等问题，经常需要在切分的同时从系统架构上去改进。长此以往，系统的架构会越来越复杂，最终，系统很可能会出现失控的惨剧。

这个时候，可以在垂直切分的基础上对客户资料管理模块和订单管理模块进行水平切分。结合垂直和水平两种切分方式，扬长避短，发挥各自的特点。首先，水平切分和垂直切分之间没有任何的冲突，不需要推倒之前垂直切分的工作成果，反而可以充分的加以利用，在现有的垂直切分的基础上发挥水平切分的优势，解决垂直切分造成的系统架构复杂性问题。举例说明：权限管理模块、参数管理模块、客户资料管理模块、订单管理模块这四个模块。第一次发现数据库性能问题的时候，可以将一个数据库垂直切分成四个数据库，权限管理模块和参数管理模块各自存储在一个数据库上，客户资料管理模块存储在一个数据库上，订单管理模块存储在一个数据库上。垂直切分完，性能问题得到初步解决。随着系统的发展，系统的数据量增大了，越来越多的用户去使用这个系统，造成访问量激增。这个时候，如果继续垂直切分，必然要对核心模块客户资料管理和订单管理进行垂直切分，但问题来了，客户资料管理中的很多表需要进行JOIN查询，而且对于客户资料管理模块，只有客户资料表的访问量最大，因为对其他客户资料附属表的操作都需要通过客户资料表进行关联，这种情况下如果对客户资料表相关表进行拆分，必然会造成跨数据库JOIN问题。如果这个时候采用水平切分方式，对客户资料表进行分表管理，根据客户资料编号对客户资料进行分表，既避免了跨数据库JOIN问题，而且还提高了系统性能。

垂直和水平联合切分是一种非常好的数据库扩展方式，考虑到这种方式是垂直和水平的叠加，不可能解决垂直和水平切分自身的问题。而且从实际应用来说，也会带来数据库系统复杂度和系统架构复杂度的问题。是否采用联合的切分方式，需要考虑业务、架构等背景情况。从经验上来说，目前稳定运行且数据量和访问量比较大的数据库应用系统都已经采用了垂直和水平联合切分的方式，但是系统性能仍然需要提高，正是因为这个原因，才有本文所研究的多主复制数据库扩展方式的产生意义。

### MYSQL CLUSTER

MySQL Cluster是MySQL从5.0版本开始提供的新功能。简单的说，MySQL Cluster就是在没有共享存储设备的情况下实现的一种内存数据库，MySQL Cluster主要通过NDB Cluster存储引擎实现[18]。

一个MySQL Cluster环境主要由以下三部分组成

1. 负责管理各个节点的Manage节点主机。
2. 层的SQL服务器节点，就是通常所说的MySQL Server。
3. Storage层的NDB数据节点，就是通常所说的MySQL Cluster。

MySQL Cluster是一种能对数据进行持久化操作的内存数据库，所有的数据和索引只能依靠装载在内存中才能正常运行。MySQL Cluster的核心在于NDB Cluster存储引擎，不仅能够对数据进行了水平切分，还能够对数据进行了跨节点的冗余。既解决了数据库的扩展问题，同时也很大程度的提高了数据库的整体可用性。目前，MySQL Cluster的应用还不是很广泛，MySQL Cluster这种Share Nothing的Cluster架构可能是未来的趋势，但目前还不是太成熟。目前，在整个行业来看，采用MySQL Cluster架构的应用还不是很多，缺少已经长期稳定运行的使用案例。

## 淘宝数据库架构分析

淘宝网，这个中国最大的在线购物网站，2010年时注册用户数就已经达到了3.7亿，从2009年开始每年以35%的速度持续增长。根据淘宝数据盛典中提供的数据显示[19]，淘宝网和淘宝商城（天猫）每天的包裹数已经超过了800万个，占整个快递行业总包裹数的60%左右。如果把这些包裹堆起来的话，淘宝网一个月就可以建成一座胡夫金字塔。2011年底淘宝网的单日独立访问量最高达到了1.2亿人，比2010同期增长了120%左右，这就意味着当天中国每四个网民中就有一位登录了淘宝网，人数相当于8000个超级卖场的总和。2011年12月12日淘宝网的单日成交额达到了43.8亿元，超过了北京和上海一天的零售业销售额总和。通过上面这一系列的数字，不难看出，淘宝网数据库的访问量和并发量一定是非常之庞大，那么对淘宝网数据库架构的设计和发展就非常值得研究与学习。

2003年，淘宝网创办初期，考虑到快速开发的需要，主要采用MySQL、PC服务器和本地硬盘存储的组合方式，这种组合方式在稳定性和高性能方面都存在问题；因此2004年，逐步开始采用Oracle、PC服务器和独立存储的组合方式。这种组合方式虽然保证了高可用性，但是也带来了高成本；2005年，随着业务的发展，在数据库整体架构不变的情况下，引入高端硬件，减少硬件的故障概率。高端硬件的引入，为未来的高增长奠定了基础。这时开始使用Oracle、IBM小型机和EMC高端存储的组合方式，这种组合方式不仅从容量上还是处理能力上都有了质的飞跃，但成本也从几万上升到几百万；2006年和2007年继续从硬件上进行升级，升级主机CPU、内存等，引入更高端的存储设备，成本继续不断增长。这时的淘宝数据库主要是由三个数据库组成；2008年，鉴于淘宝的业务量从2007年开始自然翻一番的增长，数据库每天有几十亿次的调用，高峰时期事务数超过了3000个，主机和存储都达到了瓶颈，这时通过硬件的提升已经无法从根本上解决问题了，因此开始进行垂直拆分，把三个数据库拆分成更多个数据库，每一个数据库支持一个比较单一的业务。比如用户、商品和交易等，都分成独立的数据库；2009年和2010年，淘宝继续发展，系统压力增长远远超过了2倍/每年的速度，新业务也不断地上线，参考了eBay、亚马逊等国外大公司的方案，开始采用水平拆分。水平拆分后，数据库的数量又会巨大增加，成本问题凸显出来，因此这时提出了去Oracle、去小型机和去高端存储的去O、I、E的方案。这时数据库架构开始从Oracle和小型机向MySQL和PC服务器转移，对数据库进行水平拆分，比如完成交易数据库、商品数据库、用户数据库等拆分。上面就是淘宝网的一个大致的发展过程，排除软件和硬件方面的因素，实际上淘宝网的数据库发展历程就是一个典型的先主从分离，再垂直切分，最后水平切分的过程。淘宝的数据库架构主要有以下几个特点：

1. SQL语句由复杂到简单转变，多表关联JOIN变成单表复杂查询，单表复杂查询又变成根据主键查询[20]。
2. 通过Tair分布式缓存、实时搜索、Hadoop、NoSQL等技术解决大数据量查询的问题[21]。
3. 通过Taobao Distributed Data Laye（简称TDDL）淘宝分布式存储与中间件解决水平拆分后数据库操作复杂性问题[22]。

实际上，淘宝网的数据库架构从整体上来看还有一个一主多从的结构，对于核心模块是垂直拆分为多个库，对于核心表是水平拆分为多个表或者分区，对于垂直拆分和水平拆分后需要解决的数据库操作问题通过TDDL框架进行解决，而且淘宝相应的查询主要根据客户资料ID或者订单ID等进行主键查询，绝大多数查询都只需要访问一个数据库，因此跨库查询的场景不多。另外对于一些大数据量的查询，可能需要扫描所有的表或者分区，淘宝是通过Tair、Hadoop、NoSQL等技术来解决，而不是通过MySQL物理查询所有表来实现，这样既提高了性能，也减少了数据库的压力。

## 多主复制数据库架构分析

通过上面对淘宝网数据库架构的分析，大家不难看出，淘宝网的数据库架构有其独特的应用场景，其他系统在参考其架构时需要特别注意这些问题。

首先就是主键查询的问题，对于淘宝等很多互联网系统来说，实际上这些系统的业务相比较CRM、BOSS等系统的业务来说，是比较简单的，因此通过主键查询可以很好的完成大多数的基本功能；对于一些复杂功能，可以通过Tair、NoSQL等技术来实现。对于CRM系统、BOSS系统来说，只通过主键查询，肯定不能满足大多数的业务需求，这些系统都要提供丰富的查询方式，例如：根据时间查询、根据人员查询、根据分公司查询等。通过主键查询时，只需要定位到一张表或一个分区，而如果要实现丰富的查询方式，就需要扫描多张分表或多个分区，因此分表和分区的水平拆分方式在这些系统中使用就一定要慎重考虑，如果使用不当，不仅不能提高性能而且还会降低性能。

其次就是缓存和NoSQL等技术的应用。淘宝等互联网系统，在实际运行时，经常通过缓存提高性能，例如，淘宝卖家发布一个新商品后，淘宝首页短时间内是查询不到这个新商品的，原因就是这个查询是通过缓存实现的。新发布的商品一般在30分钟后才能在首页查询到。这种方式虽然降低了查询精确度，但是却提高了查询速度，对于互联网系统来说，查询速度是第一位的，降低查询精确度来提升查询速度是完全值得的。但对于CRM、BOSS等系统来说，一般都要求数据是实时准确的。比如销售去跟进一个具体的客户，和客户成功签单后需要立即记录这个结果，否则统计这个销售成单率时就会产生误差等。另外，由于缓存等软件一般都是免费开源的，免费软件基本没有软件支持服务，这对于大型电信或者金融企业影响很大，比如中国移动等企业基本上就很少使用开源软件。

再次就是分布式存储与中间件的使用问题。一般系统的开发都不具有淘宝TDDL这个便利的基础框架，毕竟愿意花如此大精力、人力和财力去开发这个框架的公司不多。另外，虽然现在TDDL已经开源，但是对于很多商业公司，特别是那些还和淘宝有着竞争关系的公司，去使用TDDL框架基本上是不可能的。如果自己去开发这个框架，投入的资源和期望的回报可能不成正比，一般情况下都会让人望而却步。

最后，也是最重要的一个问题，就是CRM、BOSS等系统经常需要去提供强有力的实时报表，也就是说设计OLTP数据库系统的同时，还得提供OLAP的功能，这就需要提供所谓报表数据库的设计，本文所论述的集合数据库主要就是提供统计报表和全集查询功能。

考虑到上面这些淘宝网没有解决或者无法解决的问题，因此实际去扩展数据库架构时需要根据特定的业务背景。本文所提出的基于MySQL复制改进的多主复制的数据库扩展的业务场景主要针对的是实时性要求很高的CRM系统。首先，考虑到本扩展方案使用的场景需要数据实时准确，因此必须通过数据库事务进行保证，无法通过缓存等技术实现。再加上系统的并发访问量非常之高，势必会造成数据库的连接数也非常之高，扩展前的数据库性能已经接近数据库最大负载。系统已经采用一主多从的数据库架构，所有的写操作都集中在一台数据库上，而这台数据库经常出现性能问题。因此需要提供多个写数据库，降低一个写数据库的压力，每个写数据库根据某个辨别字段进行区分，比如分公司编号等。一般情况下，系统提供的查询功能都是根据分公司等关键字段进行区分，这样，用户的查询操作只需要在一个写数据库上执行就可以了，不需要复杂的跨数据库JOIN或者UNION操作；另一方面，对于统计报表和全集查询等功能，就需要提供统一的集合数据库，考虑到写数据库拆分成了多个，每个写数据库的数据变更都需要复制到一个集合数据库中，现有的数据库扩展方式无法实现这个需求，这就需要提供新的数据库扩展方式，这就是本文论述的多主复制数据库扩展方案的产生原因。

## 本章总结

本章首先对常用数据库扩展方式进行介绍，并分析常用数据库扩展方式的优点和缺点；接着对淘宝网的架构设计和发展过程进行介绍；最后通过对比淘宝网的架构设计，阐述多主复制数据库扩展方式的产生意义。

1. **多主复制数据库扩展的设计思路**

## 应用场景

前面介绍了数据库扩展常见的几种方式以及这些方式的优缺点，每种数据库扩展方式都有其使用的特定条件和应用场景，多主复制的数据库扩展方式也有其应用的特定场景，下面将对其进行详细说明。

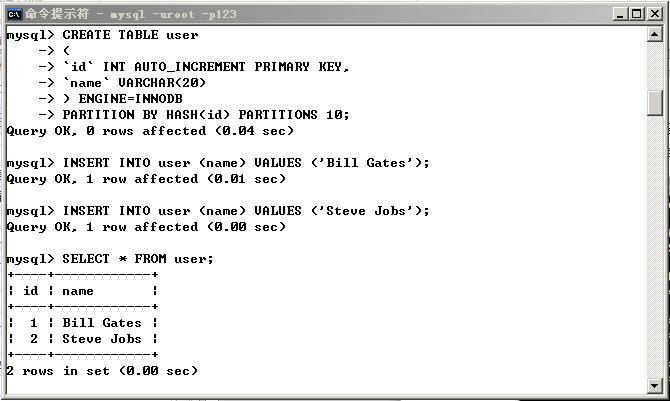


图3-1 创建哈希分区表

Fig. 3-1 Create Hash Partition Table

首先，系统的核心业务主要集中在一张表或一类表，无法通过垂直切分进行扩展。如果核心业务集中在一张表上，系统的访问量和数据量主要集中在这张核心表上，这张核心表就成为系统的瓶颈，而其他表的压力并不大。如果采用垂直切分，考虑到垂直切分的原则是切分的模块间耦合度要尽可能的低，一张单独的表无法进行数据切分，因此系统的性能压力无法通过垂直切分进行缓解；如果核心业务集中在一类表上，系统的访问量和数据量主要集中在这类表上，这类核心表就成为系统的瓶颈，其他表的压力并不大。由于这类表之间耦合度很大，相互之间的关联性又很强，表之间经常需要进行JOIN操作。因此如果将这类核心表进行垂直切分，势必会造成跨数据库或跨实例的JOIN操作和分布式事务复杂化。目前，由于很多数据库应用系统在建设的初期根本没有考虑系统后期可能遇到的访问量和数据量增大的问题，因此很难有系统能够做到系统的各个模块都相对独立，彼此之间没有任何关联关系。这种背景之下，系统很难通过垂直切分来减轻数据库的压力。如果强行切分业务，势必需要在架构上进行改进，造成系统整体架构的复杂化，甚至会带来系统架构需要不断升级的恶心循环，直至出现失控的情况。

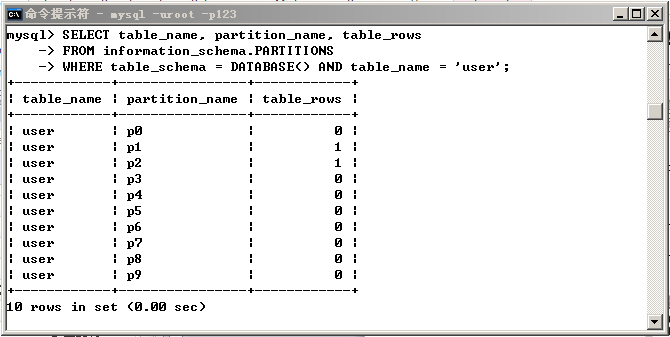
****

图3-2 查看分区表

Fig. 3-2 View Partition Table

其次，数据库是OLTP系统，无法通过水平切分进行扩展。如果数据库是OLTP系统，用户的查询方式可能会各式各样，按照特定的规则制定的分区在一些查询方式下可能会提高性能，但是在另外一些查询方式下不仅不能提高性能，还可能造成性能问题。举例说明，假设创建一个分区表，根据ID进行HASH分区，并插入两条记录，如图3-1所示。然后查看该表的分区情况，从图3-2所示的执行结果可以看出，由于指定了HASH分区的数量为10，因此分区后该表实际上产生了10个分区。新插入的两条记录分别存储在p1和p2两个分区中，接下来通过执行计划查看根据ID查询时分区的使用情况，从图3-3所示的执行结果可以看出通过id=1的条件进行查询时，只需要在p1分区中进行检索；通过id=2的条件进行查询时，只需要在p2分区中进行检索。证明根据ID作为条件进行查询时，确实提高了查询效率。最后通过执行计划查看根据NAME作为条件查询时分区的使用情况，从图3-4所示的执行结果可以看出不管是通过name=’Steve Jobs’还是通过name=’Bill Gates’作为条件进行查询，都会搜索p0、p1、p2、p3、p4、p5、p6、p7、p8和p9这10个分区，相当于进行了全表扫描。通过这个例子可以发现，在OLTP系统中，利用分区进行水平切分并不一定可靠。如果查询时使用非分区字段进行检索，可能会造成全表扫描，不仅无法提高性能，反而会增加数据库压力。当然，也可以通过分表的方式进行水平切分。经过分表之后，原来单一表中巨大的数据被平均分配多个表中，性能大幅度提高，但问题随之而来，需要对代码进行修改，使系统在访问数据库时支持分表操作，由于进行切分的表一般都是系统核心业务表，因此涉及的代码很多，进行系统升级的工作量巨大，而相应的升级工作也会影响系统的稳定性。

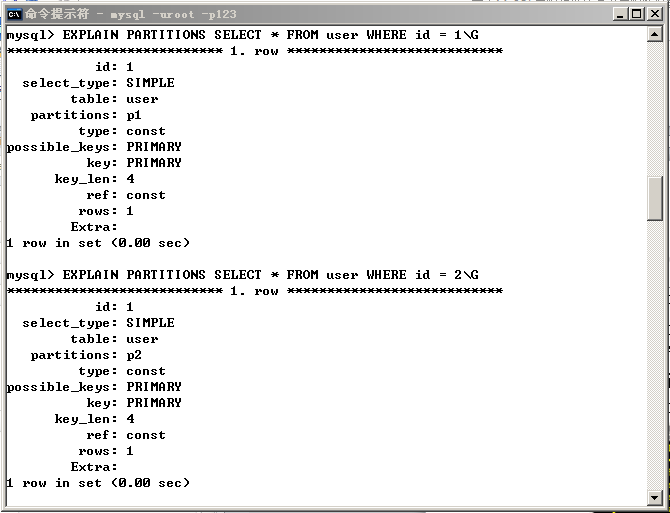


图3-3 查看ID作为查询条件的执行计划

Fig. 3-3 View Execute Plan of ID Query

最后，对于多主复制的数据库扩展方式，由于多个主服务器中所有的表结构都是相同，不同的只是存储的数据，因此和水平切分一样，需要选择一个合理的切分规则，例如根据不同分公司进行存储，上海分公司的数据存储在一个主服务器中，北京分公司的数据存储在一个主服务器中，江苏分公司的数据存储在一个主服务器中等。多主复制的数据库扩展合理的采用了垂直切分和水平切分的思想，并使之升华，同时充分考虑了数据一致性问题，提出了集合数据库的概念，所谓的集合数据库就是将多个主服务器中存储的数据合并到一个数据库中，这个数据库仍然可以继续采用传统的MySQL一主多从的复制方式进行扩展。对于很多OLAP系统来说，集合数据库也可以理解为系统的数据中心，用来进行数据统计和数据分析。需要特别说明的是，由于多主复制的扩展方式实现起来有一定的复杂度，因此一般都是在系统进行垂直切分以及水平切分之后，系统仍然存在性能问题时使用，具体实施前需要进行详细的调研。当然，对于访问量和数据量都很大的系统来说，在合理考虑扩展成本和整体架构的前提下，多主复制的扩展方式回报率还是很大的。接下来本文详细介绍多主复制的设计思路。

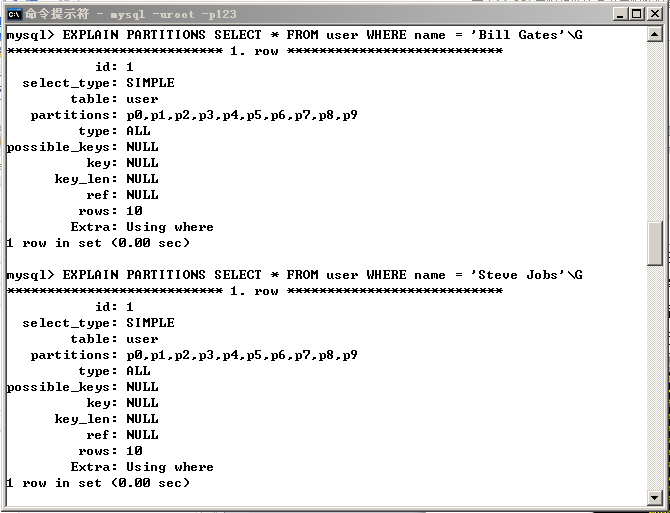


图3-4 查看NAME作为查询的执行计划

Fig. 3-4 View Execute Plan of Name

## 总体设计

目前，MySQL只支持从服务器从一个主服务器进行数据复制，如果需要MySQL支持从服务器从多个主服务器进行数据复制，最需要解决的就是数据冲突问题。实际上，将多主复制引入MySQL的计划由来已久，但由于很难解决数据冲突问题，因此一直就没有什么比较好的实现案例。举个简单的例子说明，有两个主服务器A和B，A和B中除了存储的数据有差别以外，数据库表结构都是一样的，服务器A和B对应同一个从服务器C，这里模拟了一个多主复制的环境，服务器A和B形成了多主结构。数据库用户甲在主服务器A上创建了一个订单表，创建成功后，主服务器A的I/O线程将改变的信息传递给从服务器C，从服务器C的SQL线程还原用户甲在主服务器A上的建表操作，在从服务器C上也创建一个订单表。完成之后，数据库用户乙在主服务器B上也创建了一个订单表，创建成功后，主服务器B的I/O线程将改变的信息传递给从服务器，从服务器的SQL线程在还原用户乙在主服务器B上的建表操作时，发现从服务器上已经存在同名的订单表，这时候建表操作会执行失败，用户乙在主服务器B上的建表操作并没有复制到从服务器上，由于复制操作是在后台进行的，用户乙可能无法获知复制操作已经失败了。实际上，用户甲和用户乙创建的订单表除了表名一致以外，表结构是完全不相同的。在用户乙没有关注MySQL复制日志的情况下，所有对于主服务器B上订单表的数据复制都不会成功，最终当用户乙在从服务器上查询订单表时才会发现，数据库中存在的订单表并不是自己创建的订单表。



图3-5 总体设计

Fig. 3-5 Overall Design

在实现MySQL多主复制数据库扩展方式的时候，最需要解决就是数据冲突问题。简单来说，需要为每个主服务器配置与之一一对应的解析从服务器，这些解析从服务器只负责解析从服务器上的中继日志并将SQL、SERVER ID以及TRANSACTION ID封装为SQL消息传递给消息队列；消息队列保证SQL消息串行的被处理以及SQL消息持久化到数据库；SQL处理进程从消息队列中获取SQL消息首先进行预处理，接着根据SQL消息中的TRANSACTION ID进行验证，验证完成后再通过JavaCC分析SQL语法并在解析后的SQL语句中加入SERVER ID作为辨别条件，然后将生成的SQL语句传递给集合数据库去执行，最后还需要记录成功执行的SQL消息的TRANSACTION ID，避免同样的SQL消息被多次执行，这就是多主复制的主要流程，如图3-5所示。接下来详细介绍MySQL多主复制数据库扩展方式的核心组件设计。



### 唯一性主键设计

首先需要介绍的是唯一性主键。一般情况下，在创建新表的时候，一般会为新表添加一个主键字段，该字段通常情况下都是自增的，例如：MySQL中的AUTO\_INCREMENT。目前，在进行系统研发的时候，大家都会采用Object Relational Mapping框架，例如：Java的Hibernate或者C#的NHibernate框架，这些框架都建议大家在表中提供一个自增字段。对于一主多从的数据库架构，主服务器中表的自增主键值会复制到从服务器中，由于架构中只存在一个主服务器，因此表的自增主键值在整个主从架构中都是唯一的。但面对多主复制架构时，从服务器复制数据时就会遇到主键冲突问题。例如，主服务器A中订单表插入第一条记录，自增主键的值为1；主服务器B中订单表也插入第一条记录，自增主键的值也为1；当主服务器A和主服务器B向从服务器进行数据复制的时候，从服务器订单表就会出现主键重复的情况。因此就需要考虑唯一性主键问题，实现唯一性主键的方案很多，例如：Oracle的Sequence、Twitter的Snowflake、MySQL主键表等。下面对这三种方案进行对比和分析：

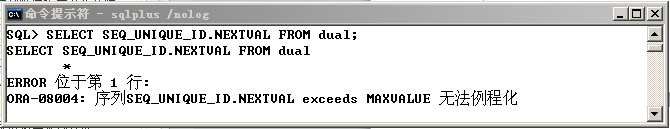


图3-6 序列超过最大值

Fig. 3-6 Sequence Exceeds Maxvalue

1. 模拟Oracle区间序列

MS SQL Server、MySQL等主流数据库都提供自增字段，但是Oracle没有提供这个功能，而是通过序列实现相同的功能。Oracle的序列是通过不断地加锁和解锁保存序列最大值，以保证序列的有序增长。创建序列时可以设定序列起始值、步长和最大值。当系统具有多个主服务器时，可以根据业务要求给每个主服务器设定特定的主键范围，例如主服务器A的主键范围为101-200，主服务器B的主键范围为201-300，然后分别在主服务器A和B上创建序列，主服务器A上序列起始值为101，步长为1，最大值为200；主服务器B上序列起始值为201，步长为1，最大值为300。创建一个名称为SEQ\_UNIQUE\_ID的序列，通过SEQ\_UNIQUE\_ID.NEXTVAL就可以获取到相应的序列值，不同的主服务器通过对应的序列可以获取到各自独立的主键，之间不会有任何主键冲突的情况，当序列超过最大值后会有错误提示，如图3-6所示。

1. Twitter的Snowflake

Twitter早期使用MySQL作为数据库保存数据，随着数据量的增长，MySQL实例无法承受海量的数据，因此Twitter开始使用Cassandra替代MySQL，但是Cassandra不能为每一条数据生成唯一主键，因此Twitter提出了Snowflake。Snowflake是Twitter开源的一款适用于分布式环境的主键生成服务器。Snowflake生成的主键是64 bits，其结构与编码方式和MongoDB生成的ObjectId相似。主键的前41 bits是以微妙为单位的时间戳，中间的10 bits是机器ID，最后的12 bits是累加计数器。

1. MySQL主键表

MySQL提供自增主键的功能。和Oracle相似，MySQL也可以设置自增主键的起始值，但是无法限定自增主键的最大值，因此无法实现Oracle序列的主键区间效果，仍然存在主键冲突问题。这时可以利用MySQL InnoDB引擎行锁的特性来实现唯一性主键，这就是所谓的MySQL主键表。MySQL主键表有两种实现方式，第一种方式是通过建立一张主键表，表中的每条记录都保存着一张表的主键，如图3-7所示。由于InnoDB引擎的锁机制是行锁，因此更新主键时只会锁定一行，对其他行不会有影响。另一种方式是建立若干张主键表，每张主键表只有一条记录，该条记录保存着实际的主键值，这种方式解决了单独一张主键表访问量过大的问题，能够应对更高的并发访问。

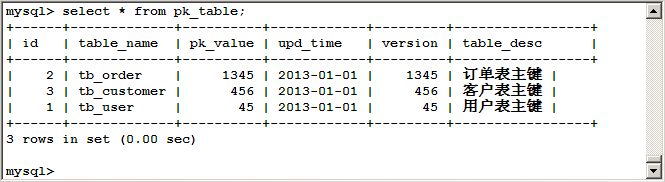


图3-7 主键表

Fig. 3-7 Primary Key Table

由于本文论述的实际应用场景中，某些表的主键具有业务记忆作用，不能使用Snowflake的64 bits主键结构，考虑到多主复制数据库扩展方案实现时要尽可能的简单、可靠、高效，因此本文采用MySQL主键表的方案，就是通过设置一个公共的配置数据库，该数据库只负责生成唯一性主键。对于主服务器上各种需要自动生成主键的表，在配置数据库都会对应一张表，每张表都只有一行记录，记录着当前的主键值，并通过乐观锁机制进行控制。

### 解析从服务器设计

解析从服务器实际上就是传统一主多从复制架构中的从服务器的升级。在介绍解析从服务器的功能之前，需要考虑下面这个问题：多个主服务器中存储着不同的数据，假设主服务器A和主服务器B中都存在结构相同的用户表。用户甲在主服务器A上查询用户表，发现有两条记录，如图3-8所示；用户乙在主服务器B上查询用户表，发现有两条记录，如图3-9所示。主服务器A和主服务器B的用户表数据都复制到集合数据库中，这时集合数据库的用户表中有四条记录，如图3-10所示。用户甲想删除主服务器A中的两条记录，因此输入SQL语句DELETE FROM user WHERE id < 4，由于SQL语句中出现了小于筛选条件，因此该SQL语句复制到集合数据库中执行之后，会将集合数据库的用户表中id=2的记录也删除掉。由于用户甲在主服务器A上并不能看到id=2的记录，所以用户甲原本是想删除id=1和id=3的记录，最后却造成id=2的记录也被错误删除，造成数据不一致问题。

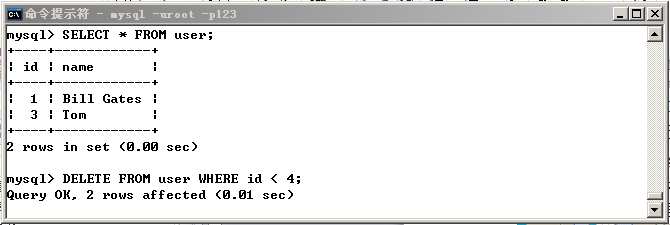


图3-8 主服务器A中数据

Fig. 3-8 Data in Master A

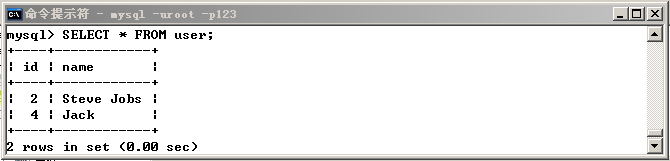


图3-9 主服务器A中数据

Fig. 3-9 Data in Master B

多主复制时必然会遇到这个问题，为了解决这个问题，本文提出SERVER ID的概念。举例说明，用户甲在主服务器A上对用户表进行查询，发现有两条记录后，准备删除这两条记录，因此输入SQL语句DELETE FROM user WHERE id < 4，执行的时候通过在SQL语句中加入SERVER ID辨别条件，既DELETE FROM user WHERE id < 4 AND server\_id = ?的SQL语句，SERVER ID是MySQL复制机制中已经存在的服务器编号，每个服务器在复制机制中SERVER ID是唯一的。同时在集合数据库中为每个表创建SERVER ID字段，该字段中记录着这条记录来自于哪台主服务器。最终执行的语句为DELETE FROM user WHERE id < 4 AND server\_id=11（假设主服务器A的SERVER ID为11）。这样，这条DELETE语句只会删除集合数据库中来源于主服务器A中的数据，而不会删除来源于其他主服务器的数据。

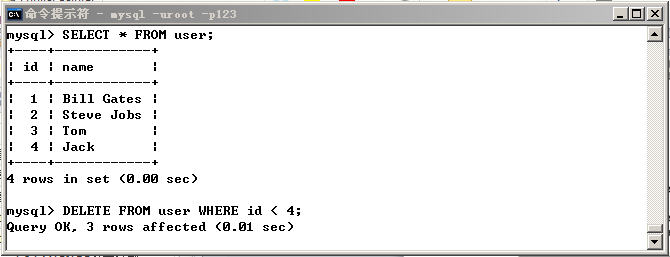


图3-10 集合数据库中数据

Fig. 3-10 Data in Collective DB

通过在SQL语句中加入SERVER ID这个辨别条件来解决多主复制时数据不一致问题，而添加SERVER ID的操作就是解析从服务器来完成的。除此以外，解析从服务器还能够根据EVENT TYPE对主服务器传递的事件进行过滤和特殊处理。多主复制的场景下，用户甲和用户乙分别在两个不同的主服务器中创建同名的表，当这两个DDL操作复制到集合数据库中后，必然会造成其中一个DDL操作失败的情况。本文叙述的多主复制数据库扩展的实际应用场景中，DDL语句的操作相对较少，绝大多数操作都是DML语句，此外考虑到对DDL语句实现多主复制非常复杂，DDL语句涉及的都是对数据库表的重大改动，和DML语句相比，如果出现问题危害性较大，因此暂不提供对DDL语句的支持，需要在解析从服务器中过滤掉DDL事件。当遇到DDL操作时统一由DBA手工操作，这样既减轻了多主复制数据库扩展方案的实现复杂度，也提高了系统的安全性，避免造成重要数据的丢失。

### 消息队列设计

消息队列是多主复制数据库扩展方案的核心组件。实现多主复制数据库扩展，最需要解决的就是数据冲突问题，参考Oracle Streams多主复制的设计思路，最好的解决方案就是使用消息队列串行的处理消息。如果让集合数据库并行的接收主服务器上的变更信息并执行这些变更操作，势必会带来死锁等一系列问题，实际上这种情况和两个Session进行并发操作造成死锁是类似的问题，如表3-1所示。

表3-1 死锁

Table3-1 Dead Lock

|  |  |  |
| --- | --- | --- |
| 时间点 | 主服务器A上的操作 | 主服务器B上的操作 |
| 1 | mysql> SELECT \* FROM user\G  \*\*\*\*\*\*\*\*\*\* 1. row \*\*\*\*\*\*\*\*\*\*  id: 1  name: Bill Gates  \*\*\*\*\*\*\*\*\*\* 2. row \*\*\*\*\*\*\*\*\*\*  id: 2  name: Steve Jobs  2 rows in set (0.00 sec) | mysql> SELECT \* FROM user\G  \*\*\*\*\*\*\*\*\*\* 1. row \*\*\*\*\*\*\*\*\*\*  id: 1  name: Bill Gates  \*\*\*\*\*\*\*\*\*\* 2. row \*\*\*\*\*\*\*\*\*\*  id: 2  name: Steve Jobs  2 rows in set (0.00 sec) |
| 2 | mysql> BEGIN;  Query OK, 0 rows affected | mysql> BEGIN;  Query OK, 0 rows affected |
| 3 | mysql> UPDATE user SET name = 'Tom' WHERE id = 1;  Query OK, 1 row affected  Rows matched: 1 Changed: 1 Warnings: 0 |  |
| 4 |  | mysql> UPDATE user SET name = 'Jack' WHERE id = 2;  Query OK, 1 row affected  Rows matched: 1 Changed: 1 Warnings: 0 |
| 5 | mysql> UPDATE user SET name = 'Jack' WHERE id = 2;  Query OK, 1 row affected  Rows matched: 1 Changed: 1 Warnings: 0 |  |
| 6 |  | mysql> UPDATE user SET name = 'Tom' WHERE id = 1;  1213 - Deadlock found when trying to get lock; try restarting transaction |

通过上面死锁的试验可以看出，如果让集合数据库并行的接收主服务器上的变更信息并在集合数据库上执行这些变更操作，在试验类似的实际场景中肯定会出现死锁问题，上面的实验最后执行UPDATE user SET name = 'Tom' where id = 1时，数据库系统就抛出了1231的出错提示，即发生了死锁。因此，多主复制的数据库扩展方式考虑到并行处理可能带来的异常问题，并借鉴和学习了Oracle Streams技术中相关消息队列的设计，提出使用消息队列串行处理消息的工作方式。对于队列中的消息，消息队列服务器会自动持久化这些消息，保证系统的稳定性以及容错性。

### SQL处理进程设计

和消息队列密不可分的就是SQL处理进程，解析从服务器将SQL消息传递给消息队列后，需要对应的处理程序去消息队列中获取SQL消息并进行处理，SQL处理进程就是做这个工作的。从消息队列中获取的SQL消息并不是实际的SQL语句，而是类似于Map的结构，Map中有三种Key：SQL、SERVER ID和TRANSACTION ID，这个SQL消息不能够直接放到集合数据库上去运行。解析从服务器起着添加SERVER ID的作用，但是解析从服务器并没有将SERVER ID直接加入到SQL语句中，而是将SERVER ID、TRANSACTION ID以及相应的SQL语句包装成Key-Value结构的SQL消息发送给消息队列。因此SQL处理进程就需要获取消息队列中的SQL消息，并进行消息的预处理和TRANSACTION ID验证，然后将获取到的SERVER ID添加到具体SQL语句中。这里有个难点，就是从消息队列中获取的SQL语句，需要分析其SQL语法，然后才能在合适的位置加入SERVER\_ID = ?这个条件，这里使用JavaCC这个工具去进行语法和词法分析，然后在合适的位置加入SERVER\_ID = ?这个条件。完成这些工作之后，SQL处理进程就可以通过JDBC将具体的SQL语句发送给集合数据库去执行，并记录执行完的SQL消息的TRANSACTION ID避免同一消息被重复处理。

### 集合数据库设计

最后需要介绍的就是集合数据库的设计，为了解决数据一致性问题，本文提出添加SERVER ID这个辨别条件，主要就是为了标记数据来源于不同的主服务器，因此对于集合数据库中的数据库表，都需要添加一个SERVER\_ID字段。例如，数据库用户在主服务器A（SERVER ID为11）的用户表中插入一条新记录后，变更信息经过解析从服务，消息队列以及SQL处理进程这一系列环节，最后传递到集合数据库去执行，集合数据库也会新增一条记录，和主服务器A的用户表中的记录不同的是，集合数据库中的记录会多一个字段：SERVER\_ID，值为11；之后，当数据库用户操作主服务器A的用户表时，传递到消息队列的SQL消息中都会加入SERVER ID这个条件。通过引入数据来源辨别机制，从逻辑上合理的分隔了来源于不同主服务器的数据，保证数据的一致性。此外，集合数据库的设计也是一主多从的结构，集合数据库作为复制中心，向多个集合从数据库进行数据复制，确保系统对于集合数据库的访问也是读写分离的。

## 本章总结

本章主要对MySQL多主复制的数据库扩展方式的设计思路进行总体介绍。首先分析了MySQL多主复制的数据库扩展方式的应用场景，说明多主复制架构可以替代垂直切分或者水平切分，也可以作为垂直切分、水平切分以及垂直和水平联合切分的进一步升级方案，同时也指出MySQL多主复制的数据库扩展方式具体实现时有一定的复杂性，建议实际应用时需要因地制宜、充分调研。接着从总体设计上对MySQL多主复制的数据库扩展方式进行介绍，由总体设计图出发，逐个组件的进行介绍，分别对唯一性主键、解析从服务器、消息队列、SQL处理进程以及集合数据库的设计进行了分析。

1. **多主复制数据库扩展的具体实现**

第3章已经从总体设计上对MySQL多主复制的数据库扩展方式进行了简单介绍，本章主要对多主复制的具体实现进行介绍，力求完整的展现多主复制数据库扩展方式的所有实现细节。



## 需求背景



介绍多主复制的数据库扩展方式之前，先对比下MySQL传统的主从复制流程，可以总结为以下7个重要步骤，如图4-1所示：

1. 配置一台主服务器(Master)。
2. 配置一台或多台从服务器(Slaves)。
3. 将从服务器连接到主服务器。
4. 主服务器将数据库中的所有变更记录在二进制日志文件(Binary Log)中。
5. 主服务器通过I/O线程将日志信息传递给从服务器。
6. 从服务器的I/O线程接收日志信息并保存在中继日志文件(Relay Log)中。
7. 从服务器的SQL线程发现中继日志有更新后就读取中继日志并还原主服务器的数据变更。

目前，MySQL复制是不支持多主复制的，只支持一主多从的复制，由于一主多从复制架构的稳定性和扩展性，一般情况下，都会选择Dual Master复制架构(Master-Master)、级联复制架构(Master-Slaves-Slaves)或Dual Master与级联复制结合架构(Master-Master-Slaves)等。归根到底这些高级复制架构都是常规复制架构的扩展，都只具备一个主服务器(Master)，不管有多少个从服务器，或者多少层的复制架构，始终只有一个主服务器生成二进制日志文件，这些二进制日志事件传递到数据库架构中的每一台从服务器上，每一台从服务器去还原主服务器上的数据变更，这个过程对于每一台从服务器都是相同的。每一台从服务器都等于得到了一个主服务器上的二进制日志文件拷贝，甚至级联复制架构中第一层的从服务器和最后一层的从服务器得到的二级制日志事件也是基本相同。

实现多主复制的数据库扩展，需要优先考虑和解决的就是数据冲突问题。一主多从复制方案中，数据变更的发起者只有一个，多个从服务器都是各自独立的还原主服务器的数据变更，从服务器之间没有任何的依赖关系和从属关系。而多主复制数据库扩展中，数据变更的发起者变成了多个，一台从服务器同一时间可能需要还原多个主服务器的数据变更，这种情况下就会出现数据冲突问题，因此避免数据冲突就成为一个大难题。本文通过引入SERVER ID作为辨别条件和使用消息队列串行处理SQL消息来解决数据冲突问题。

图4-1 主从复制

Fig.4-1 Master-Slaves Replication

在一主多从复制流程的基础上，多主复制流程扩展为如下9个步骤，如图4-2所示：

1. 配置多台主服务器(Masters)，需要为每台主服务器设定全局唯一的SERVER ID。
2. 一台主服务器配置一台解析从服务器，采用最经典的一主一从复制架构(Master-Slave)。
3. 将解析从服务器连接到主服务器。
4. 主服务器将数据库中的所有变更记录在二进制日志文件(Binary Log)中。
5. 主服务器通过I/O线程将日志信息传递给解析从服务器。
6. 解析从服务器的I/O线程接收日志信息并保存在中继日志文件(Relay Log)中。
7. 解析从服务器的SQL线程发现中继日志有更新后就读取中继日志，但并不直接还原主服务器的数据变更，而是将SQL语句、SERVER ID和TRANSACTION ID封装为Key-Value的SQL消息传递给消息队列。
8. SQL处理进程从消息队列中获取解析从服务器发送的SQL消息，对SQL消息进行处理和验证后，通过JavaCC对SQL语句进行词法和语法分析，并在其中加入SERVER ID这个辨别条件，最后通过JDBC发送给集合数据库。
9. 集合数据库执行SQL处理进程发送过来的SQL语句，并将这些数据变更复制到集合数据库的多个从库中去。

通过上面的9个步骤，可以看出实现多主复制的数据库扩展，现有的MySQL复制流程中的一些关键步骤需要进行改进：

1. 主服务器和解析从服务器

由于解决数据冲突需要SERVER ID作为辨别条件，因此需要在主服务器上配置SERVER ID参数。其次需要修改从服务器的SQL线程处理逻辑，以便SQL线程发现中继日志更新后将主服务器上的SQL语句、主服务器上配置的SERVER ID以及自动生成的TRANSACTION ID封装为SQL消息传递给消息队列，并跳过还原主服务器数据变更的操作。

1. 消息队列

在传统的MySQL一主多从复制机制中是没有消息队列这个设计的，在设计多主复制数据库扩展方案的时候，通过调研发现Oracle数据库的高级复制技术Oracle Streams使用了消息队列的设计，因此多主复制参考了Oracle Streams实现多主复制的原理，使用消息队列作为解决数据冲突问题的关键组件。此外，消息队列可以自动持久化消息，可以在消息队列发生故障时快速恢复数据。

1. SQL处理进程和集合数据库

由于多主复制数据库扩展中添加了SERVER ID作为辨别条件来解决数据冲突问题，因此就需要提供能够解析SQL语句并添加SERVER ID辨别条件的组件，这就是SQL处理进程的产生原因。前面提到解析从服务器作为生产者将消息发送给消息队列，与之相配合的就是SQL处理进程作为消费者从消息队列中获取消息并进行处理，SQL处理进程产生最终需要在集合数据库中执行的SQL语句。集合数据库可以理解为一主多从复制结构中的从数据库，不提供写操作，只提供读服务，同时集合数据库和集合数据库的多个从库又实现了一套传统意义上的一主多从复制架构。



图4-2 多主复制

Fig.4-2 Masters-Slave Replication

## 主服务器和解析从服务器实现

### 日志分析

本章主要对二进制日志和中继日志进行分析，因为中继日志是SQL线程的主要操作对象，多主复制的数据库扩展方案中一个非常重要的环节就是对SQL线程处理逻辑的升级。由于二进制日志和中继日志的格式是相同，因此理解了二进制日志的内容，也就等于理解了中继日志的内容，在这个基础上才能完成对SQL线程的逻辑升级。

1. 分析二进制日志结构



图4-3 二进制日志

Fig.4-3 Binary Log

二进制日志是一系列二进制日志事件。日志事件存储在二级制日志文件中，此外还有一个二进制日志索引文件，用来定位已有的二进制日志文件[23]。当前服务器正在操作的二进制日志文件称作活动二进制日志文件，如图4-3所示。二进制日志文件主要是用于记录数据库表的变更信息，然后将这些信息进行传递和复制。一般情况下，二进制日志文件只记录数据库的变更，对于不变更数据库、数据库表或者数据库表记录的语句，二进制日志文件是不会记录的[24] [25]。

1. 分析中继日志结构



图4-4 中继日志

Fig.4-4 Relay Log

中继日志是MySQL复制的核心元素。中继日志的结构和二进制日志的结构类似，同时中继日志还维护着两个文件用于追踪复制进度，一个是中继日志信息文件，另一个是主数据库日志信息文件，如图4-4所示。当主服务器上有语句执行之后，主服务器的二进制日志文件中就会增加一个或多个事件日志，这时主服务器上的转储线程(Dump Thread)就会从二进制日志文件中读取新的事件，然后发给解析从服务器的I/O线程，解析从服务器的I/O线程负责接收传递来的事件信息，并将其写入中继日志的末尾。

实际上，MySQL的复制有多种实现级别，不同的复制级别，二进制日志文件中保存的信息是完全不同的，常用的有基于行复制(Row Level)、基于语句复制(Statement Level)和混合复制(Mixed Level) [26]。

1. 基于语句复制：这种方式会在二进制日志文件中记录具体的更新SQL语句。
2. 基于行复制：这种方式会在二进制日志文件中记录行数据被修改的形式。
3. 混合复制：这种方式会在基于语句复制和基于行复制之间进行自动抉择。

通过上面三种级别的对比，最终还是采用基于语句复制的方式，主要考虑到目前的业务场景中不会出现复杂的SQL语句，同时目前的数据库主要作为OLTP系统使用，更新、修改和删除操作比较多，而且经常是根据某个字段进行范围操作，例如根据年龄字段进行操作，更新所有小于20岁的用户数据，如果使用基于行复制的方式，所有小于20岁的用户数据都会保存在日志中，而使用基于语句的复制，则只会记录一条UPDATE语句。因此使用基于行复制的方式不如使用基于语句复制的方式效率高。在本文叙述的多主复制数据库扩展方案中采用基于语句复制的方式。

主服务器上同一时间存在的二进制日志文件并不只一个，因此需要知道目前活动的二进制日志文件名，通过在主服务器上执行SHOW MASTER STATUS命令可以知道目前活动的二进制日志文件名。

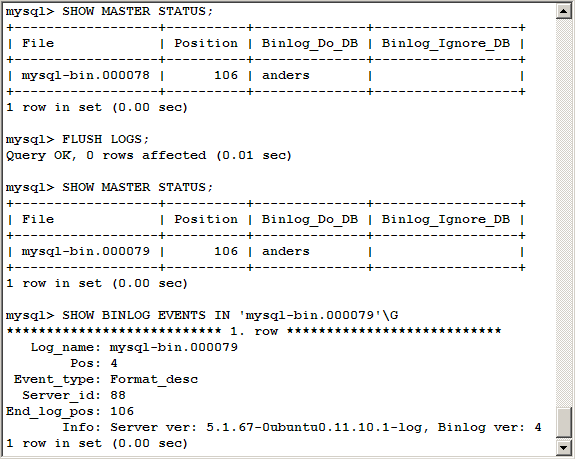


图4-5 强制轮换二进制日志文件

Fig.4-5 Flush Binary Log

如图4-5所示，可以知道目前活动的二进制日志文件名为mysql-bin.0000078，考虑到目前活动的二进制日志文件中已经保存着历史日志信息，不便于查看和分析，因此使用命令FLUSH LOGS强制轮换二进制日志文件，重新生成一个新的二进制日志文件。通过SHOW MASTER STATUS可以查询到新的二进制日志文件名为mysql-bin.000079。接着可以通过命令SHOW BINLOG EVENTS IN ‘二进制日志文件名’查看二进制日志文件的实际内容，MySQL新创建的二进制日志文件mysql-bin.000079中只有一个格式描述事件，格式描述事件是服务器内部用于管理二进制日志的一个系统事件。对二进制日志文件中出现的6个信息字段需要详细说明，修改解析从服务器的SQL线程主要就是围绕这些信息字段进行扩展的：

1. Log\_name：保存事件的二进制日志文件名，一个事件只能保存在一个二进制日文件中，不会存在多个二进制日志文件中。
2. Pos：当前事件在二进制日志文件中的起始位置。
3. Event\_type：当前事件类型，MySQL内部有完整的事件类型表。
4. Server\_id：产生事件的主服务器编号，如11、21、31等。
5. End\_log\_pos：当前事件在二进制日志文件中的结束位置。
6. Info：当前事件的基本信息，可以通过这个字段获得当前事件所包含的SQL语句。

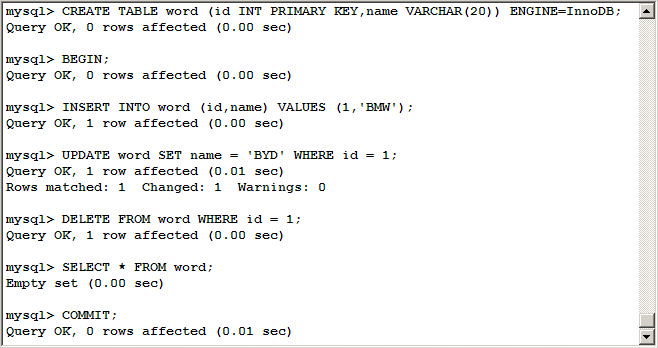


图4-6 建表、增、改、删、查操作

Fig.4-6 Create Table, Insert, Update, Delete, Query Operation

接下来，通过一个试验详细了解二进制日志文件中的事件内容。首先创建一张word表，然后开启事务并在表中插入一条记录，完成后更新这条记录，接着删除这条记录，最后查询word表并提交事务，如图4-6所示。这一系列的操作完成之后，通过命令SHOW BINLOG EVENTS IN ‘二进制日志文件名’查看二进制日志文件后发现，除了查询word表的操作事件没有记录在日志文件中，其他的操作都详细的记录在二进制日志文件中，并且按照操作执行的先后顺序依次记录在日志文件中，如图4-7所示。由此可以证明二进制日志文件中记录的都是对数据库、数据库表和数据库表记录的变更操作日志，查询等非变更操作并不记录在二进制日志文件中。实验中也发现，日志中包括开启事务和提交事务的操作事件，但无法确定回滚事务的操作事件是否记录在二进制日志文件中。

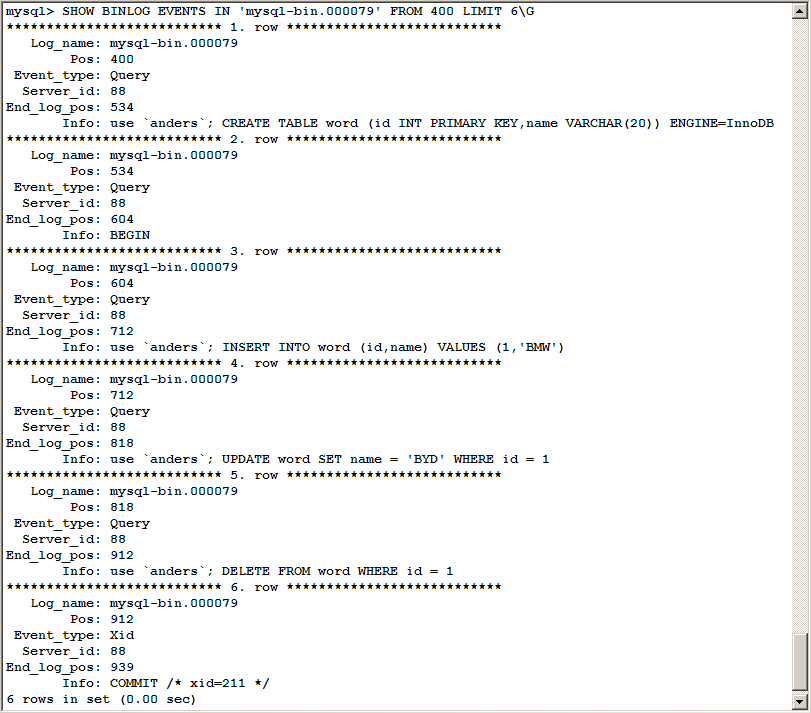


图4-7建表、增、改、删、查操作日志

Fig.4-7 Create Table, Insert, Update, Delete, Query Operation Log

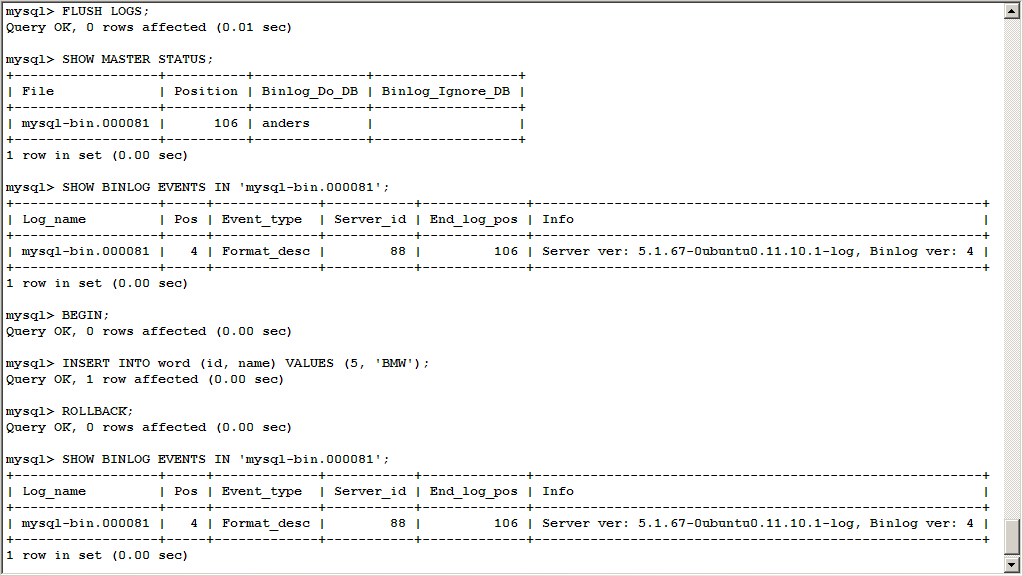


图4-8 回滚事务日志

Fig.4-8 Rollback Transaction Log

表4-1 两个会话同时执行新增操作

Table4-1 Two Session Insert Simultaneously

|  |  |  |
| --- | --- | --- |
| 时间点 | 会话1 | 会话2 |
| 1 | mysql> BEGIN;  Query OK, 0 rows affected (0.00 sec) | mysql> BEGIN;  Query OK, 0 rows affected (0.00 sec) |
| 2 | mysql> INSERT INTO word (id, name) VALUES (6, 'BYD');  Query OK, 1 row affected (0.00 sec) | mysql> INSERT INTO word (id, name) VALUES (5, 'AUDI');  Query OK, 1 row affected (0.00 sec) |
| 3 | mysql> COMMIT;  Query OK, 0 rows affected (0.00 sec) | mysql> COMMIT;  Query OK, 0 rows affected (0.00 sec) |

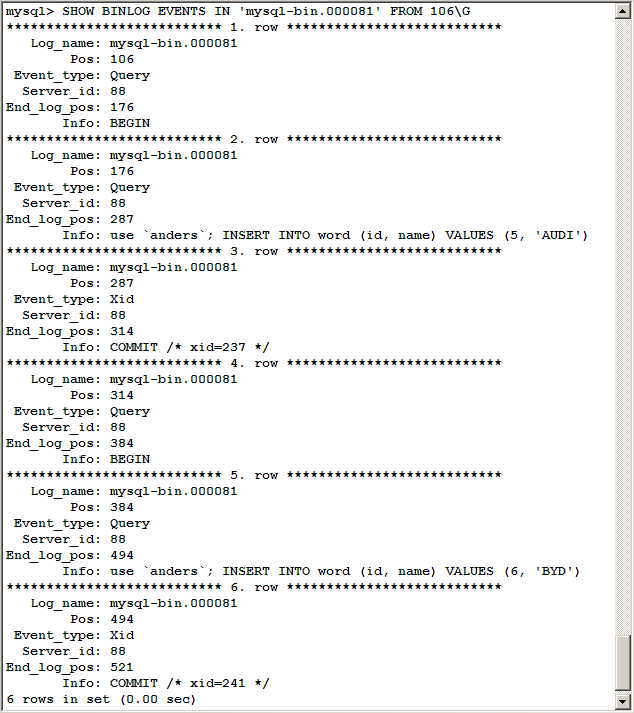


图4-9 两个会话同时执行新增操作日志

Fig.4-9 Log of Two Session Insert Simultaneously

为了验证这个问题，使用命令FLUSH LOGS强制轮换二进制日志文件，重新生成一个二进制日志文件，通过SHOW MASTER STATUS查询到新的二进制日志文件名为mysql-bin.000081。首先开启事务，然后插入一条新记录，完成后回滚事务。查看二进制日志文件后发现日志文件中并没有这次操作的事件日志，如图4-8所示。由此可见，MySQL并不是将数据库的变更操作第一时间记录到二进制日志文件中，MySQL会将所有未提交的操作保存到一个日志缓冲中，当事务提交后直接写入二进制日志文件中，对于回滚的事务则不记录在二进制日志文件中。

此外，还要考虑一种情况，由于MySQL同一时间可能有多个用户在进行操作，这些操作事件都会记录在二进制日志文件中，这些日志会不会混杂在一起。创建两个MySQL会话，在相同的时间点分别开启事务、插入一条数据以及提交事务。查看二进制日志文件后发现，MySQL将两个事务操作事件依次记录在二进制日志文件中，两个事务中的语句并没有混杂在一起，如图4-9所示。

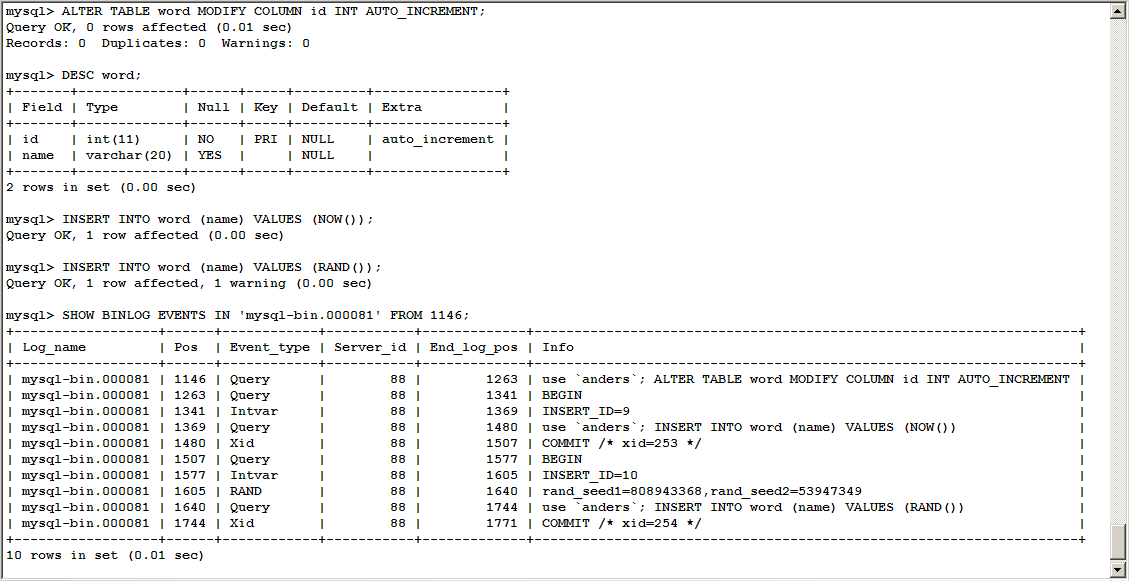


图4-10 自增、时间函数、随机函数日志

Fig.4-10 Auto Increment, Time Function, Rand Function Log

最后需要考虑的是，对于MySQL的随机函数、时间函数以及自增字段，二进制日志文件是如何记录的。将word表id字段修改为自增类型，然后分别插入两条记录，一条使用NOW函数，另一条使用RAND函数。查看二进制日志文件后发现，对应的SQL语句都保存在二进制日志文件中，除此以外，还添加了一些额外的参数设置语句，如图4-10所示。

### 具体实现

正如第三章介绍的，实现MySQL多主复制的数据库扩展时，需要通过增加SERVER ID这个条件解决数据冲突问题。增加SERVER ID的入口很多，第一种方式是主服务器I/O线程写二进制日志文件的时候添加SERVER ID，但是这种方式有一定风险，添加SERVER ID需要分析SQL词法和语法，在SQL语句正确的位置加入SERVER ID，对于稳定运行的主服务器来说，这是一个非常大的风险点，因为MySQL的源代码非常复杂，如果修改不慎，运行时主服务器可能会发生一些无法预知的问题，从而造成数据的丢失或错乱，这是数据库无法允许的错误。

第二种方式就是在从服务器SQL线程解析中继日志的时候添加SERVER ID，由于这种方式不涉及主服务器代码的改动，保证了MySQL主服务器的可靠性。改动从服务器SQL线程解析中继日志的代码，虽然也有可能造成从服务器运行时出现故障，但和主服务器发生故障相比，这个影响级别小很多。由于主服务器中数据并没有丢失，后期可以通过补偿方式将数据复制到从服务器上。

对比上面两种方案的优劣，本文采用在从服务器SQL线程解析中继日志的时候添加SERVER ID这种实现方式。解析从服务器的SQL线程发现中继日志有更新后就读取中继日志，但并不还原主服务器的数据变更，而是将SQL语句、SERVER ID和TRANSACTION ID封装为Key-Value的消息传递给消息队列。

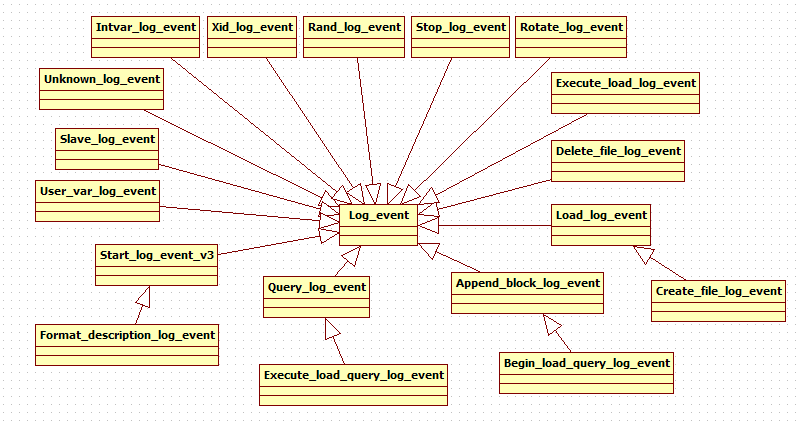


图4-11 日志事件类图

Fig.4-11 Log\_event Class Diagram

表4-2 日志事件类型表

Table4-2 Log Event Type Table

|  |  |  |  |
| --- | --- | --- | --- |
| 日志事件 | 事件值 | 说明 | 对应的Log\_event子类 |
| UNKNOWN\_EVENT | 0 | 未知事件 | Unknown\_log\_event |
| START\_EVENT\_V3 | 1 | 主服务器启动后创建的第一个事件，5.0开始被FORMAT\_DESCRIPTION\_EVENT代替 | Start\_log\_event\_v3 |
| QUERY\_EVENT | 2 | 常见的DML操作事件 | Query\_log\_event |
| STOP\_EVENT | 3 | mysqld服务关闭事件 | Stop\_log\_event |
| ROTATE\_EVENT | 4 | 轮换事件，如执行FLASH LOGS操作或二级制日志文件超过文件大小最大值 | Rotate\_log\_event |
| INTVAR\_EVENT | 5 | 有AUTO\_INCREMENT自增字段或者调用LAST\_INSERT\_ID函数事件 | Intvar\_log\_event |
| LOAD\_EVENT | 6 | LOAD DATA INFILE | Load\_log\_event |
| SLAVE\_EVENT | 7 | 暂不使用 | Slave\_log\_event |
| CREATE\_FILE\_EVENT | 8 | LOAD DATA INFILE | Create\_file\_log\_event |
| APPEND\_BLOCK\_EVENT | 9 | LOAD DATA INFILE | Append\_block\_log\_event |
| EXEC\_LOAD\_EVENT | 10 | LOAD DATA INFILE | Execute\_load\_log\_event |
| DELETE\_FILE\_EVENT | 11 | LOAD DATA INFILE | Delete\_file\_log\_event |
| NEW\_LOAD\_EVENT | 12 | LOAD DATA INFILE | Load\_log\_event |
| RAND\_EVENT | 13 | 调用RAND函数事件 | Rand\_log\_event |
| USER\_VAR\_EVENT | 14 | 使用用户变量事件 | User\_var\_log\_event |
| FORMAT\_DESCRIPTION\_EVENT | 15 | 新二进制日志文件格式描述事件 | Format\_description\_log\_event |
| XID\_EVENT | 16 | 事务开启、提交或回滚事件 | Xid\_log\_event |
| BEGIN\_LOAD\_QUERY\_EVENT | 17 | LOAD DATA INFILE | Begin\_load\_query\_log\_event |
| EXECUTE\_LOAD\_QUERY\_EVENT | 18 | LOAD DATA INFILE | Execute\_load\_query\_log\_event |

MySQL主从复制机制会将上游主服务器发送过来的数据封装为Log\_event对象。Log\_event类在MySQL源代码log\_event.h中定义，这个类又细分为多个子类，如图4-11所示。 log\_event.h中定义了一个枚举Log\_event\_type，用来区分各种EVENT，每种EVENT都对应一个Log\_event的子类。

如表4-2所示，MySQL日志事件中有多个事件是和LOAD DATA INFILE语句相关的，MySQL复制在处理该命令时需要从文件系统中读取文件，如果读取的文件太大，必然造成大量的I/O操作和网络传输，如果在多主复制架构中支持该命令，势必会影响消息队列处理消息的性能，因此多主复制架构中禁止使用LOAD DATA INFILE语句。此外，考虑到多主复制架构中存在多个主服务器，如果支持DDL语句，集合数据库有可能出现DDL冲突问题，因此多主复制架构中暂不支持DDL语句。下面就是需要特殊处理的日志事件：

1. QUERY\_EVENT：MySQL中绝大多数的DDL和DML语句都被封装为这种对象。其中需要特别处理时间函数。
2. INTVAR\_EVENT和LAST\_INSERT\_ID：如果INSERT语句中有自增列，则会产生这种事件。
3. RAND\_EVENT：如果在SQL语句中调用了RAND函数，则会产生这种事件。
4. USER\_VAR\_EVENT：如果SQL语句中引用了用户自定义变量，则会产生这种事件。
5. XID\_EVENT：事务的提交或者回滚会产生这种事件。

当解析从服务器接收到主服务器传递来的变更操作后，解析从服务器需要去还原这些操作。对于MySQL复制机制来说，最核心的代码就是LOG\_EVENT的处理逻辑，因此改进从服务器SQL线程的功能实际上就是去修改LOG\_EVENT的处理逻辑。接下来对各种LOG\_EVENT处理方案进行介绍：

1. Query\_log\_event

针对这种事件，忽略解析从服务器中更新数据库的所有错误。因为多主复制只关心主服务器上变更操作语句的获取，并不关心这些语句是否已经在解析从服务器上执行过或者执行是否成功。一般情况下，MySQL的DDL语句和DML语句都被包装成这种事件，其中DML语句和SERVER ID条件会一起传递给消息队列。对于NOW、CURDATE、CURTIME和UNIX\_TIMESTAMP这些时间函数，MySQL为了正确的处理，日志事件会记录一个时间戳，并将这个时间戳从主服务器传递给从服务器，保证主从数据库的数据一致性，如图4-12所示。对于使用SYSDATE函数的情况就需要特别处理，因为MySQL在处理该函数时并不是和NOW、CURDATE、CURTIME和UNIX\_TIMESTAMP函数一样。如果没有处理SYSDATE函数，主服务器的SQL语句传递到集合数据库中执行可能会产生时间差异，导致主从数据库上的时间不一致，因此多主复制架构中禁止使用SYSDATE函数。

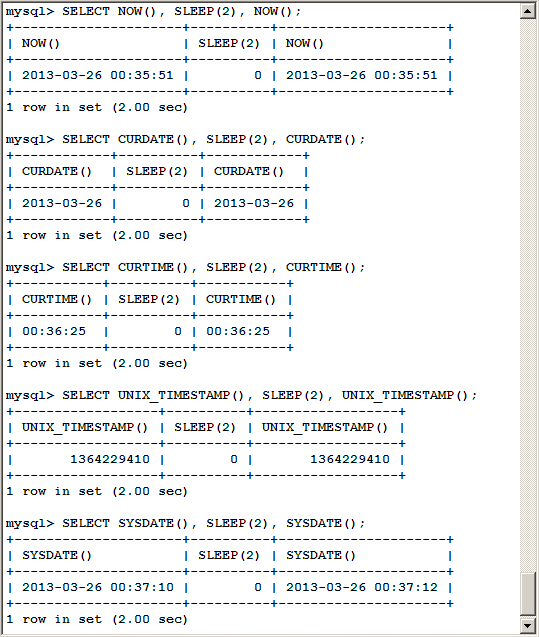


图4-12 时间函数

Fig.4-12 Time Function

1. Intvar\_log\_event

如果在主服务器上执行带有自增列的INSERT操作或者调用LAST\_INSERT\_ID函数，那么在Query\_log\_event事件处理之前会先接收到Intvar\_log\_event事件。由于多主复制架构中为了避免产生主键冲突问题，使用了唯一性主键生成工具的设计方式，因此多主复制架构方案中禁止使用自增列和LAST\_INSERT\_ID函数。

1. Rand\_log\_event

如果在主服务器上执行了RAND函数，那么在Query\_log\_event事件处理之前会先接受到Rand\_log\_event事件。RAND函数是基于伪随机数的函数，看上去生成的数字是随机的，实际上是均匀分布的。RAND并不是真正的随机函数，而是从一个种子数字开始，参照一个伪随机函数从而产生具体的数字序列。如果种子相同，则RAND函数会返回相同的数字。因此当发现主服务器的SQL语句中出现RAND函数后，SQL线程会根据日志事件中的SEED1和SEED2的数字设置变量@@RAND\_SEED1和@@RAND\_SEED2，从而保证主数据库和从数据库的数据一致性。如图4-13所示，连续调用两次RAND函数，随机生成的值完全不同。通过查看二进制日志信息，手工修改种子变量的值，然后再调用RAND函数，发现第三次随机生成的值和第二次随机生成的值完全一致。

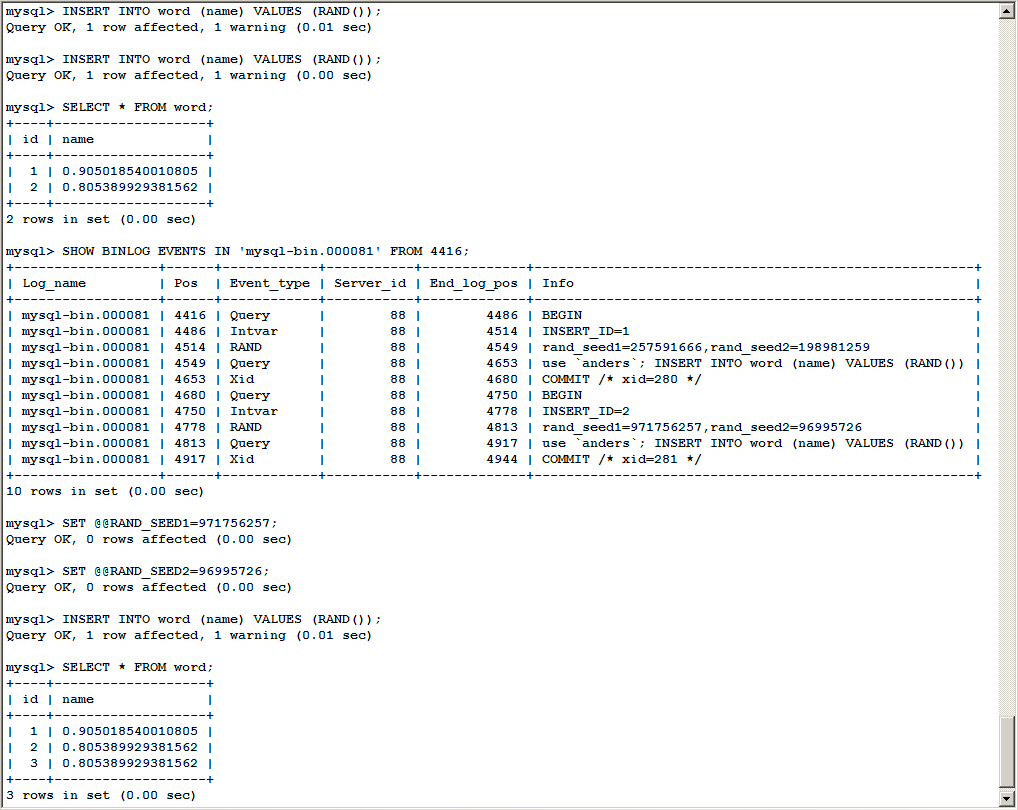


图4-13 随机函数日志

Fig.4-13 Rand Function Log

1. User\_var\_log\_event

这种事件主要针对主服务器SQL语句中使用了用户变量的情况。和Rand\_log\_event处理方式相同，对于用户变量事件的处理，也是通过设置变量@@用户变量名= 用户变量值，并将这条SQL传递到消息队列中，以便能够正确的还原主服务器上的变更操作。

1. Xid\_log\_event

事务的提交或者回滚会产生此种事件，由于多个主数据库的操作事件是依次记录在日志事件中的，因此只需要将BEGIN和END间的语句作为一组SQL封装成SQL消息传递给消息队列就可以了，下游的SQL处理进程能够方便的识别这组SQL。此外封装SQL消息时需要按照日志事件中事件发生顺序给SQL语句进行编号并加入TRANSACTION ID信息。

表4-3 主服务器配置

Table4-3 Master Configuration

|  |  |
| --- | --- |
| 服务器名 | 服务器配置参数 |
| 主服务器1 | [mysqld]  ……  server-id = 11  log-bin = mysql-bin  log-bin-index = mysql-bin.index  binlog\_do\_db = db\_user  …… |
| 主服务器2 | [mysqld]  ……  server-id = 21  log-bin = mysql-bin  log-bin-index = mysql-bin.index  binlog\_do\_db = db\_user  …… |
| 主服务器3 | [mysqld]  ……  server-id = 31  log-bin = mysql-bin  log-bin-index = mysql-bin.index  binlog\_do\_db = db\_user  …… |

### 实现步骤

经过前面的叙述，已经实现了多主复制的主服务和解析从服务器，接下来对具体的配置过程进行介绍。

1. 配置多个主服务器

配置主服务器时，首先必须给主服务器分配一个唯一的服务器编号，其次要确保主服务器有一个活动的二进制日志文件。常规的主从复制架构已经采用了给主服务器分配一个唯一的服务器编号来区分服务器的方式。特别需要注意的是配置的主服务器编号一定要是唯一的，如果两个主服务器配置的编号相同，就会产生服务器编号重复的错误。修改MySQL的服务器配置文件my.cnf，添加二进制日志文件配置和服务器编号配置，如表4-3所示。

主服务器配置参数：

1. log-bin：该参数说明二进制日志文件的通用文件名。
2. log-bin-index：该参数说明二进制日志索引文件的文件名。
3. binlog\_do\_db：该参数指明了MySQL需要复制的数据库名。

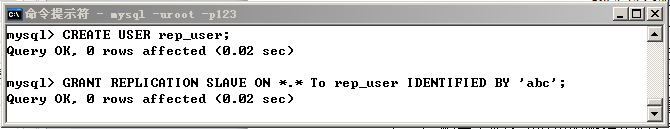


图4-14 创建复制用户

Fig.4-14 Create Rep User

由于解析从服务器连接主服务器时需要一个有权限转储二进制日志文件数据的数据库用户，考虑到这个用户权限的特殊性，一般情况下都有数据库管理员专门创建一个用于复制的用户并赋予复制的权限，如图4-14。不建议将复制权限赋予普通用户。

经过上面的操作，成功配置了三个主服务器，SERVER ID依次为11、21、31，同时也配置三个主服务器需要复制的数据库db\_user，然后重启三个主服务器，使主服务器配置生效，最后创建了专门用于数据复制的用户rep\_user，并赋予该用户复制的权限。这样，就实现了主服务器的基本配置。

1. 配置主服务器对应的解析从服务器

首先需要说明的是目前的MySQL复制机制，从服务器的SQL线程既负责解析中继日志文件又负责执行解析后得到的语句，充分利用SQL线程的特性，本文提出解析从服务器的概念。解析从服务器只负责解析中继日志文件，不执行解析后得到的语句，而是将解析后得到的语句以及主服务器的SERVER ID包装成Key-Value的消息传递给消息队列。由此可以看出解析从服务器实际上就是传统意义上的从服务器，只不过解析从服务器不负责执行主服务器的数据库语句也不用于存储数据，只是单纯的提供解析中继日志文件的作用。

配置解析从服务器时，和配置主服务器一样，仍然需要给解析从服务器分配一个唯一的服务器编号，其次要确保解析从服务器有一个活动的中继日志文件。需要注意的是配置的解析从服务器编号一定是唯一的，如果解析从服务器之间或者解析从服务器和主服务器的编号相同，就会产生服务器编号重复的错误。

和配置主服务器一样，需要修改MySQL的服务器配置文件my.cnf，添加二进制日志文件和服务器编号配置，如表4-4所示。

表4-4 解析从服务器配置

Table4-4 Analysis Slave Configuration

|  |  |
| --- | --- |
| 解析从服务器名 | 解析从服务器配置 |
| 解析从服务器1 | [mysqld]  ……  server-id = 12  relay-bin = slave-relay-bin  relay-bin-index = slave-relay-bin.index  replicate\_do\_db = db\_user  …… |
| 解析从服务器2 | [mysqld]  ……  server-id = 22  relay-bin = slave-relay-bin  relay-bin-index = slave-relay-bin.index  replicate\_do\_db = db\_user  …… |
| 解析从服务器3 | [mysqld]  ……  server-id = 32  relay-bin = slave-relay-bin  relay-bin-index = slave-relay-bin.index  replicate\_do\_db = db\_user  …… |

解析从服务器配置参数：

1. relay-bin：该参数说明中继日志文件的通用文件名。
2. relay-bin-index：该参数说明中继日志索引文件的文件名。
3. replicate\_do\_db：该参数指明了MySQL需要复制的数据库名。

经过上面的操作，成功配置了三个解析从服务器，SERVER ID依次为12（对应主服务器11）、22（对应主服务器12）、32（对应主服务器31），同时也配置三个解析从服务器需要复制的数据库db\_user，最后重启了三个解析从服务器，使修改的配置生效。这样，就实现了解析从服务器的基本配置。

1. 解析从服务器连接到主服务器

通过上面介绍，目前已经配置了三台主服务器和对应的三台解析从服务器，一台主服务器和一台解析从服务器构成一组。现在要做的就是让主服务器和解析从服务器之间连接起来，进行通信。首先要将解析从服务器指向主服务器，让解析从服务器知道从指定的主服务器进行数据复制，因此需要配置主服务器的如下信息：

1. 主机名。
2. 端口号。
3. 主服务器上具有复制权限的用户。

第一步配置主服务器时，已经为三个主服务器创建了专门用于数据复制的用户rep\_user，该用户的密码为abc。三个主服务器的主机名分别为aix01、aix02、aix03；三个主服务器的端口号默认为3306。有了这些信息之后，就可以将主服务器和解析从服务器连接起来，使用CHANGE MASTER TO命令将解析从服务器指向主服务器，如图4-15所示，再通过START SLAVE命令启动复制。

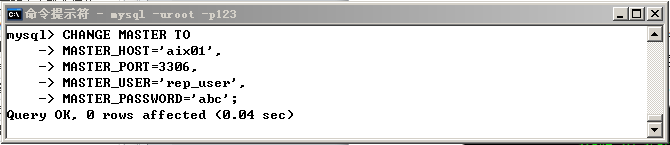


图4-15 解析从服务器连接主服务器

Fig.4-15 Analysis Slave Connected Master

经过上面的操作，已经成功的将主服务器和解析从服务器连接在一起，可以通过在解析从服务器上执行SHOW SLAVE STATUS\G命令查看解析从服务器的运行情况。

执行SHOW SLAVE STATUS\G命令后，重点需要查看的是Slave\_IO\_Running和Slave\_SQL\_Running这两个信息，如图4-16所示，目前解析从服务器的I/O线程和SQL线程都运行良好，如果这两个线程运行有问题，就会出现Slave\_IO\_Running: No或者Slave\_SQL\_Running: No的信息。此外Slave\_IO\_State显示Waiting for master to send event，这就表示解析从服务器已经准备好了，随时等待主服务器发送事件信息过来。如此，主服务器和解析从服务器已经完成连通，可以正常运行起来。

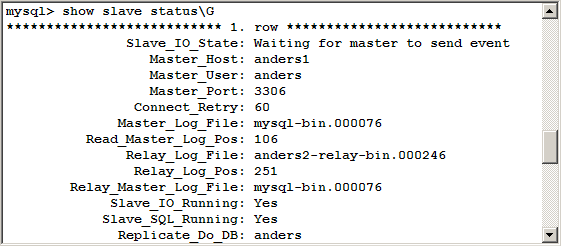


图4-16 解析从服务器状态

Fig.4-16 Analysis Slave Server Status

## 消息中间件实现

MySQL多主复制架构中使用Apache社区提供的JMS开源消息组件ActiveMQ作为消息队列的具体实现。在介绍消息队列之前先介绍下消息队列。



### 消息队列

消息队列的实现方式很多，例如微软消息队列(Microsoft Message Queue)（简称MSMQ）、Java消息服务(Java Message Service)（简称JMS）等。JMS是Sun公司根据JSR-914创建的一种传递企业消息的应用程序编程接口[27]。

JMS主要有两种实现模式

1. 点对点模型：生产者将消息发送到一个特定的消息队列中，而消费者从这个消息队列中获取消息，每条消息都有一个消费者，如果一条消息被某个消息者接收了，那么其他的消费者就不能得到这条消息了。
2. 订阅/发布模型：生产者发布消息，而消费者订阅感兴趣的消息，生产者将消息和一个特定的主题连在一起，消息传递系统根据消费者注册的兴趣，将消息传递给消费者。

多主复制数据库扩展具体实现时需要提供消息队列的功能，本文主要使用JMS点对点的模式实现消息队列的设计。目前支持JMS协议的产品很多，例如：JBossMQ、JORAM、OpenJMS、ActiveMQ等等。考虑到ActiveMQ是一个开放源代码的项目，是Apache下的顶级项目，有着强劲能力的开源消息总线，是完全支持JMS1.1和J2EE1.4规范的JMS Provider实现[28]。从实际应用来说，ActiveMQ和Spring结合的开发模式比较好，对于使用Java作为开发语言的系统来说，使用起来也很方便。最重要的是，MySQL主要使用C语言开发，而ActiveMQ支持通过多种语言和协议编写客户端，便于SQL处理进程后期的升级和扩展。因此本文选用ActiveMQ作为JMS协议的具体实现。

### 具体实现

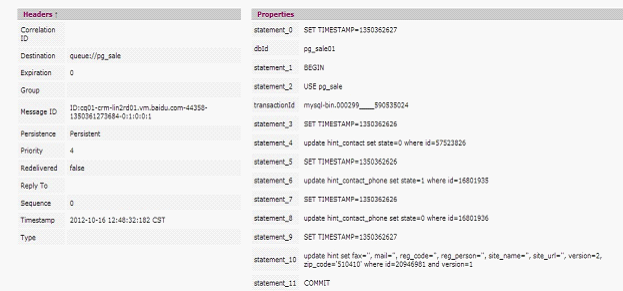


图4-17 消息例子

Fig.4-17 Message Example

这里需要特别说明的是，消息队列是实现多主复制的技术保障。由于多主复制数据库扩展的复杂性，如果提供并行处理SQL消息的执行方式，可能会造成数据冲突等问题。通过对Oracle Streams复制功能的研究后发现，使用消息队列串行的传递多个主服务器的变更到集合数据库可以有效的解决数据冲突等问题。当解析从服务器将主服务器的二进制日志信息拷贝到自己的中继日志文件中后，解析从服务器就开始运行I/O线程，这个I/O线程打开一个连接去主服务器上进行BINLOG DUMP PROCESS。BINLOG DUMP PROCESS从主服务器的二进制日志文件中读取事件，解析从服务器上的I/O线程把这些事件写入中继日志文件中，SQL线程从更新的中继日志文件中获取更新信息，也就是解析从主服务器获取新的事件，但并不执行更新操作，而是直接将SQL消息发送给消息队列。这里需要说明的是，发送消息给消息队列的时候是按照事件组的概念进行传输，每个二进制日志文件中，都会包含着若干个二进制日志事件组，按照Format\_description事件作为文件头，以日志轮换事件作为文件尾。除了这两个事件以外，其他二进制日志事件都被分成若干个组，在事务存储引擎中，每个组差不多对应一个事务的开启和提交。概括来说，二进制日志文件中的事件组，要么是多条语句组成的一个事务，要么就是不属于事务的单条语句。一般来说，同一组操作要么全部执行，要么全部不执行，不能切分开执行。当消息传输给消息队列的时候，也需要按照组的方式进行传输。这样才能保证数据的一致性。

如图4-17所示是多主复制时一个发送到消息队列的消息示例。发送到消息队列的消息是以组为单位来发送的。每条消息中，SQL语句以Key-Value的形式保存。其中SERVER ID为上游主服务器配置的SERVER ID，TRANSACTION ID是该消息对应的主服务器中二进制日志文件名称及所处的文件位置的组合，以“\_\_\_”分隔。其他以statement\_开头的Key为实际需要在集合数据库上执行的SQL语句，执行顺序按照Key中的数字来判断。需要注意的是每条query\_log\_event对应的语句前都带有一个SET TIMESTAMP语句，这是为了保证在主数据库中调用NOW函数与在集合数据库中调用NOW函数返回相同的时间。

当解析从服务器作为客户端发送完消息，不需要等待消息队列的回应，只需要将消息发送给消息队列，也就是ActiveMQ，ActiveMQ负责将消息传递给SQL处理进程，如果SQL处理进程作为接收端出现问题，并不影响客户端的正常工作，确保消息能够正确传递到SQL处理进程是ActiveMQ的工作。通过这个解决方案，成功的将解析从服务器和SQL处理进程分解开。考虑到实际需要的是一个串行消息队列，队列中的消息按照进入队列的时间顺序依次被处理，因此采用点对点的消息传递模型。

鉴于消息队列在整个多主复制流程中的重要性，当解析从服务器将消息传递给消息队列后，消息队列会把消息持久化到消息数据库，这样当消息队列出现问题的时候，可以启动备用消息队列，备用消息队列启动后从消息数据库中重新获取消息传递给SQL处理进程，这样就保障了系统的稳定性。当SQL处理进程处理完消息后，会在消息数据库中记录该条消息已经被处理过。

### 实现步骤

1. 配置ActiveMQ控制台账户密码

默认情况下，当ActiveMQ启动时内置的Jetty服务器也会启动，该内存WEB服务器会提供一个用于监控和测试的管理系统，在浏览器打开http://localhost:8161/admin就可以看到这个管理系统。这个管理系统默认的用户名和密码都是admin，考虑到该管理系统的安全性，因此需要修改这个用户名和密码，修改conf文件夹中的jetty-realm.properties文件，将默认的用户名admin和密码admin改为安全的用户名和密码。

1. 配置ActiveMQ持久化消息

ActiveMQ消息中间件默认使用KahaDB存储引擎持久化消息数据，考虑到KahaDB存储引擎的底层是文件系统，当数据文件过大后，可能会影响消息队列服务的性能，因此需要将其更换为MySQL持久化消息数据，修改conf文件夹中的activemq.xml文件，将默认的persistenceAdapter注释掉，添加新的persistenceAdapter以支持MySQL数据库，并配置MySQL的数据源。需要注意的是，此处的MySQL消息持久化数据库实际是一个一主多从的架构。

1. 配置发送/接收消息队列的账户密码

接下来需要对发送和接收消息的操作进行权限控制，因此需要设置用户名和密码。

1. 配置ActiveMQ主从热备结构

考虑到ActiveMQ在整个多主复制架构中的重要性，因此需要配置多台ActiveMQ消息中间件，目的是当主消息中间件出现问题，其他的从消息中间件可以立刻代替主消息中间件提供服务。配置多台ActiveMQ消息中间件指向相同的数据库，同一时间，先拿到数据库表ACTIVEMQ\_LOCK锁的消息中间件会成为主消息中间件，一旦当前的主消息中间件失效，其他的消息中间件会再次抢占锁并自动升级成主消息中间件。在成为主消息中间件之前，其他的消息中间件均已启动，但不可访问。

## SQL处理进程和集合数据库实现

### 语法解析

SQL处理进程主要从消息队列中获取SQL消息进行处理、验证和保存，并从获取的SQL消息中提取SERVER ID信息，然后SQL处理进程去分析消息中的SQL信息，并在合适的位置加入SERVER\_ID = ?这个辨别条件，最后SQL处理进程将完整的SQL通过JDBC发送给集合数据库去执行。作为多主复制数据库扩展方式的核心技术之一，SQL处理进程如何识别不同的SQL语句，并在合适的位置加入SERVER\_ID = ?这个条件是个难点。

计算机专业有一门非常重要的专业课程：编译原理，这门课程的主要目标就是介绍编译程序构造的一般原理和基本方法。该课程的主要包括：语言和文法、词法分析、语法分析、语法制导翻译、中间代码生成、存储管理、代码优化和目标代码生成等内容。其中词法分析主要是对输入数据，从左到右逐个字符读取，然后根据构词规则识别成具体的单词，语法分析是在词法分析的基础上将识别出来的单词组合成各类的语法短语。举个简单的例子，上海市浦东新区管理委员会，这个短语可以划分成上海市、浦东新区、管理委员会，但计算机会认为这只是一条12个字符的字符串，如果想让计算机识别这个字符串的具体含义，就需要定义识别规则，这时候，可以先定义词法规则和语法规则，然后计算机就可以正确的识别这句短语。

MySQL中有很多类型的SQL语句，例如：

1. SELECT \*\*\* FROM \*\*\* WHERE \*\*\*
2. UPDATE \*\*\* SET \*\*\* WHERE \*\*\*
3. DELETE FROM \*\*\* WHERE \*\*\*
4. INSERT INTO \*\*\* (\*\*\*) VALUES (\*\*\*)

对于这些语句，SQL处理进程需要进行识别，并且在合适的位置加入SERVER\_ID = ?这个条件。最简单的方式就是通过if-else判断，如果是UPDATE或者DELETE语句，就在最后加上SERVER\_ID = ?这个条件；如果是INSERT的语句，就截取SQL语句，并在其中插入SERVER\_ID，伪代码大致如下：

if (SQL语句 = UPDATE || SQL语句 = DELETE) {

SQL语句 += “SERVER\_ID = ?”;

}

else if (SQL语句 = INSERT) {

…...

}

对于简单的SQL语句，这种方法确实可以解决问题，但是SQL处理进程面对的是各种复杂的MySQL语句，无法通过逻辑判断加上字符串截取去分析所有的SQL，就算勉强实现出来，这个判断逻辑肯定是复杂且难以维护和升级。此外，当遇到“(”，“+”等包含多种意义的符号时，就更加难以解析。因此就需要寻找更好的方法来分析SQL语句。

MySQL在遇到用户输入的各种命令时，是通过Yacc进行语法分析的，一般情况Lex和Yacc是结合使用的，Lex负责词法分析，Yacc负责语法分析，不过MySQL只使用了Yacc作为语法分析器。鉴于MySQL是通过词法分析和语法分析来解析SQL，因此在设计SQL处理进程时也可以参考MySQL的解析方式[29]。

考虑到本文介绍的多主复制数据库扩展方式中SQL处理进程使用Java进行开发，因此选用JavaCC分析SQL处理进程从消息队列中获取到的SQL消息。当然如果SQL处理进程使用其他的语言进行开发，比如C/C++，也可以直接使用Lex和Yacc的搭配。

接下来对JavaCC进行简单介绍，JavaCC是一个用Java开发的词法和语法生成工具。这个生成工具可以读取上下文无关且有着特殊意义的语法并将其转换成可以识别且匹配该语法的Java程序。JavaCC 是百分之百的纯Java代码，可以在多种平台上运行。同时JavaCC还提供了JJTree工具来建立语法树，以及JJDoc工具为源文件生成巴科斯·诺尔范式的文档[30]。

开发JavaCC最重要的是编写jj文件，这其中主要是定义JavaCC的具体词法和语法规则，也有一些相关的Java代码。当编写完jj文件之后，使用JavaCC命令行工具解析jj文件，生成7个核心文件：

1. CCJSqlParser.java
2. CCJSqlParserConstants.java
3. CCJSqlParserTokenManager.java
4. ParserException.java
5. SimpleCharStream.java
6. Token.java
7. TokenMgError.java

需要注意的是，CCJSqlParser.java、CCJSqlParserConstants.java以及CCJSqlParserTokenManager.java这三个文件的文件名都依赖jj文件的文件名。这7个文件中，最核心的文件是CCJSqlParser.java文件，该文件中包含主要的处理逻辑和接口函数。将这7个文件引入项目中，首先获取需要解析的内容，对于SQL处理进程就是SQL语句和SERVER ID，然后调用CCJSqlParser中的方法，将SQL语句和SERVER ID作为参数传入相关方法，最后获取JavaCC处理后SQL，这个SQL就是加了SERVER\_ID = ?的语句。

通过上面的介绍可以看出，使用JavaCC作为SQL语句的词法和语法解析器，开发人员只需要把精力集中在词法和语法文件的编写上就可以了。此外JavaCC的解析效率也是非常之高，完全不是通过逻辑判断和字符串截取所能比拟的。jj文件的编写可以参考开源SQL解析工具jSQLParser。由于jSQLParser主要用于分析标准SQL，对于MySQL支持的不够全面，对于子查询等复杂SQL在解析时也经常出现问题。因此需要在其基础上进行修改并添加SERVER ID。

### 具体实现

SQL处理进程从消息队列获取消息后，首先需要对消息进行处理，消息主要包含三种信息：SERVER ID，TRANSACTION ID以及SQL语句，SQL处理进程首先获取SQL语句并调用JavaCC解析SQL语句并加入SERVER ID这个条件，然后将SERVER ID，TRANSACTION ID以及重新组装后的SQL语句封装成MysqlStatement对象。接着SQL处理进程按照责任链的方式将MysqlStatement对象依次传递给如下类，如图4-18所示：

1. ValidateDuplicateHandler：验证消息队列持久化数据库中是否已经存在MysqlStatement对象中的TRANSACTION ID，如果存在，说明当前处理的消息已经被处理过，直接跳过当前处理的消息并获取下一条消息进行处理。
2. Push2MysqlHandler：通过JDBC批量执行MysqlStatement对象中保存的SQL语句。
3. MarkDBStatementHandler：执行完MysqlStatement对象中保存的SQL语句后，将TRANSACTION ID保存到消息队列持久化数据库中，避免重复执行相同的消息。

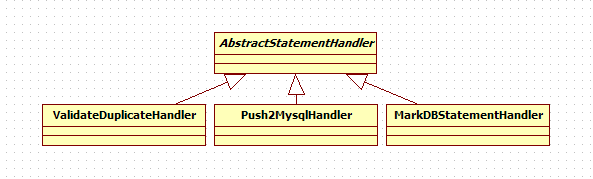


图4-18 消息处理类图

Fig.4-18 Message Handle Class Diagram

SQL处理进程将最终的SQL通过JDBC传递给集合数据库去执行。由于引入了SERVER ID辨别条件到实现方案中，因此集合数据库中的所有表必须添加SERVER ID的字段。需要考虑的是默认情况下，这个新增的SERVER ID字段不允许为空，添加这个约束主要是担心之前多个环节在处理过程出现未知的错误，造成集合数据库上执行的SQL缺失了SERVER ID = ?这个条件，为了保证数据的稳定性，需要加入这个非空约束。

此外，本文只设计了单一层次从服务器的实现，实际上具体应用是，完全可以将单一层次的从服务器进行扩展，实现多层次的MySQL数据库扩展架构，相关设计在第一章已经进行过介绍，在此不再进行叙述。

最后要提到是，由于多主复制不支持DDL语句直接从主服务器复制到从服务器，对于DDL的执行由DBA手工进行统一执行。考虑到具体实现时主服务器可能会有多个，此外还有多个层次的从服务器，综合下来，有多个服务器需要手工执行DDL语句，这种情况下，不仅增加了DBA的工作量，而且由于手工操作需要串行执行，需要耗费大量的时间。对于稳定运行多主复制的数据库来说，这是一个风险点，考虑如下的场景：假设有40台主服务器，DBA在主服务器1上执行DDL语句之后，需要再在主服务器2-40上执行同样的DDL语句，由于服务器中相关表的数据量已经很大，DBA执行完DDL的操作需要大量的时间，当DBA最后在从服务器上执行DDL语句的时候，主服务器的相关变更操作已经传递到从服务器上，由于从服务器还没有完成DDL的变更操作，因此可能会造成数据无法被复制到从服务器上的情况。应对这种操作，比较可行的方式是关闭数据库访问，让数据复制暂时中止。但是对于稳定运行的数据库应用系统来说，关闭数据库访问等同于关闭整个系统，如果单个DDL语句的操作需要10分钟，40个主服务器执行完DDL语句可能需要400分钟，由于从服务器中的数据是40个主服务器数据之和，因此从服务器的DDL操作执行时间肯定是大于10分钟的。综合起来，整个互联网系统需要停机7个小时左右，这肯定是无法接受的。此外，对于数据库应用系统来说，在维护过程中不可能没有DDL操作，由于每次DDL操作都需要停机。假设一个月有十次DDL操作，那就表示，一个月中因为DDL操作就需要停机十次。

因此，对于DDL的操作，需要提供DDL并行执行方案，由于DDL的操作都是平均分配到各个服务器上，而不是在单一服务器上同时进行，因此完全可以通过开启多个操作线程来并行执行DDL语句，每个线程调用JDBC去连接不同的主服务器，然后在每个服务器上并行的执行DDL操作，既减少了DBA的工作量，又保证了数据复制的正常运行。不过需要注意的是，由于使用并行的方式进行DDL操作，需要特别注意不要出现人为失误，因为一旦出现人为失误，回滚操作需要涉及所有的服务器，影响性较大。

1. **实验与总结**

## 实验环境

表5-1 实验环境

Table5-1 Experimental environment

|  |  |  |
| --- | --- | --- |
| 硬件环境 | CPU | Intel(R) Xeon(R) CPU E5620 @ 2.40GHz |
| 内存 | 16G |
| 软件环境 | 操作系统 | Red Hat Enterprise Linux AS release 4 (Nahant Update 3) |
| Kernal | Linux version 2.6.9\_5-9-0-0 |
| MySQL | MySQL 5.5.30 |
| ActiveMQ | 5.8.0 |

本实验的软硬件环境如表5-1所示，数据库架构配置如下：

1. 一主多从：一台主服务器搭配三台从服务器组成一主多从测试环境。
2. 多主复制：三台主服务器，每台主服务器对应一台解析从服务器和两台从服务器，三台解析从服务器再连接至一台ActiveMQ消息中间件，消息中间件对应一台集合数据库服务器，集合数据库服务器搭配三台从服务器。

## 实验方案

多主复制的数据库扩展方案，在读取数据时，如果只是获取一台主服务器上的数据，可以直接读取该台主服务器对应的从服务器上的数据，读操作得以分流；如果需要获取全量数据，则需要读取集合数据库对应的从库上的数据，一般情况下，读取全量数据的情况非常少，主要出现在统计分析功能中。由于多主复制时主节点和从节点都存在独立的主从结构，读操作的压力分散到多个从库上，因此和一主多从的数据库架构相比，优势非常明显。在写入数据时，由于多主复制修改了数据复制的逻辑，采用串行消息队列进行数据复制，和一主多从的数据库架构相比，复制的实时性可能会有所降低，因此本次试验侧重于测试数据写入的情况。

### 读操作测试

首先对读操作进行测试，使用MySQL自带的压力测试工具mysqlslap进行测试。假设同一时间有900个用户需要查询数据库，而每个用户需要做100次查询操作。

1. 一主多从
2. 方案：使用mysqlslap分别在三台从服务器上连续模拟100次300个用户并发查询100次的操作。
3. 命令：mysqlslap -uroot -p123 -a --concurrency=300 --iterations=100 --engine=innodb --auto-generate-sql-load-type=read --number-of-queries=100 --debug-info
4. 结果：如图5-2所示。

表5-2 一主多从读结果

Table5-2 Master-Slaves Read Result

|  |  |  |  |
| --- | --- | --- | --- |
| 服务器 | 最大值 | 最小值 | 平均值 |
| 从服务器1 | 144 ms | 12 ms | 120 ms |
| 从服务器2 | 129 ms | 8 ms | 107 ms |
| 从服务器3 | 158 ms | 11 ms | 131 ms |

1. 多主复制
2. 方案：使用mysqlslap分别在九台从服务器上连续模拟100次100个用户并发查询100次的操作。
3. 命令：mysqlslap -uroot -p123 -a --concurrency=100 --iterations=100 --engine=innodb --auto-generate-sql-load-type=read --number-of-queries=100 --debug-info
4. 结果：如图5-3所示。

表5-3 多主复制读结果

Table5-3 Masters-Slave Read Result

|  |  |  |  |
| --- | --- | --- | --- |
| 服务器 | 最大值 | 最小值 | 平均值 |
| 主服务器1对应从服务器1 | 27 ms | 10 ms | 13 ms |
| 主服务器1对应从服务器2 | 39 ms | 12 ms | 20 ms |
| 主服务器2对应从服务器1 | 21 ms | 18 ms | 19 ms |
| 主服务器2对应从服务器2 | 34 ms | 18 ms | 23 ms |
| 主服务器3对应从服务器1 | 32 ms | 15 ms | 27 ms |
| 主服务器3对应从服务器2 | 39 ms | 11 ms | 19 ms |
| 集合数据库从服务器1 | 44 ms | 20 ms | 31 ms |
| 集合数据库从服务器2 | 50 ms | 29 ms | 33 ms |
| 集合数据库从服务器3 | 47 ms | 24 ms | 37 ms |

### 写操作测试

然后对写操作进行测试，还是使用MySQL自带的压力测试工具mysqlslap进行测试。假设同一时间有300个用户需要更新数据库，而每个用户需要做100次更新操作。

1. 一主多从
2. 方案：使用mysqlslap分别在一台主服务器上连续模拟100次300个用户并发更新100次的操作。
3. 命令：mysqlslap -uroot -p123 -a --concurrency=300 --iterations=100 --engine=innodb --auto-generate-sql-load-type=update,write --number-of-queries=100 --debug-info
4. 结果：如图5-4所示。

表5-4 一主多从写结果

Table5-4 Master-Slaves Write Result

|  |  |  |  |
| --- | --- | --- | --- |
| 服务器 | 最大值 | 最小值 | 平均值 |
| 主服务器 | 445 ms | 172 ms | 321 ms |

1. 多主复制
2. 方案：使用mysqlslap分别在三台主服务器上连续模拟100次100个用户并发更新100次的操作。
3. 命令：mysqlslap -uroot -p123 -a --concurrency=100 --iterations=100 --engine=innodb --auto-generate-sql-load-type=update,write --number-of-queries=100 --debug-info
4. 结果：如图5-5所示。

表5-5 多主复制写结果

Table5-5 Masters-Slave Write Result

|  |  |  |  |
| --- | --- | --- | --- |
| 服务器 | 最大值 | 最小值 | 平均值 |
| 主服务器1 | 359 ms | 139 ms | 277 ms |
| 主服务器2 | 327 ms | 101 ms | 232 ms |
| 主服务器3 | 379 ms | 1. ms | 259ms |

### 数据复制测试

最后就是对数据复制进行测试，假设系统需要插入三次数据，第一次300万，第二次600万，第三次900。

1. 一主多从
2. 方案：在主服务器上新建存储过程分三次循环插入数据，第一次300万，第二次600万，第三次900。
3. 结果：如图5-6所示。

表5-6 一主多从复制结果

Table5-6 Master-Slaves Replicate Result

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 服务器 | 300万 | 复制延迟 | 600万 | 复制延迟 | 900万 | 复制延迟 |
| 主服务器 | 1:16:24 | < 1 ms | 2:48:01 | < 1 ms | 4:08:53 | < 1 ms |

1. 多主复制
2. 方案：在三台主服务器上分别新建存储过程，分三次循环插入数据，第一次100万，第二次200万，第三次300。
3. 结果：如图5-7所示。

表5-7 多主复制复制结果

Table5-7 Masters-Slave Replicate Result

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 服务器 | 100万 | 复制延迟 | 200万 | 复制延迟 | 300万 | 复制延迟 |
| 主服务器1 | 00:27:32 | 983 ms | 00:55:34 | 2437 ms | 01:15:11 | 3099 ms |
| 主服务器2 | 00:26:09 | 1243 ms | 00:51:19 | 1837 ms | 01:20:28 | 2903 ms |
| 主服务器3 | 00:31:34 | 845 ms | 00:49:23 | 2109 ms | 01:18:24 | 3180 ms |

## 实验分析

通过以上的测试可以看出，读操作时由于多主复制方案比一主多从方案提供了更多的从数据库，一主多从方案中的大量并发访问操作被分流到多个从数据库上，因此响应速度明显提升。写操作时，由于多主复制方案中的多个主数据库上均匀的保存着一主多从方案中单个主数据库上的数据，因此写操作的响应时间也显著下降。复制操作时，由于使用消息队列进行串行处理，因此复制操作的时间显著上升。

综上所得，经过测试，多主复制数据库扩展确实可以提高数据库性能，不过复制操作相对比较耗时，考虑到集合数据库主要用于进行数据统计和分析，因此小于5秒的复制延迟不会造成较大问题。

## 总结与展望

考虑到数据库对于当前这个信息化社会的重要性，MySQL多主复制数据库扩展方案的提出，有着巨大的作用。本文主要对基于MySQL复制改进的多主复制数据库扩展进行介绍，鉴于本方案在具体实施时需要有特定的应用场景，如果全面推广或发展，必然需要解决对于DDL复制、复制性能等问题。此外本文所论述的设计思想完全可以应用在其他关系型数据库系统的扩展上。

**参考文献**

[1] 郭欣. 构建高性能Web站点(修订版)[M]. 电子工业出版社, 2012.

[2] 简朝阳. MySQL性能调优与架构设计[M]. 电子工业出版社, 2009.

[3] Schwartz B, Zaitsev P, Tkacbenko V. Hig Performance MySQL[M]. 东南大学出版社, 2012.

[4] Michael Widenius. Multi-source replication for MariaDB is here[EB/OL]. [2013-3-5]. http://monty-says.blogspot.com/2012/10/multi-source-replication-for-mariadb-is.html.

[5] Michael Widenius. Multi-source replication[EB/OL]. [2013-3-5]. https://kb.askmonty.org/en/multi-source-replication/.

[6] 彭立勋. MariaDB 10.x 将包含多主复制功能[EB/OL]. [2013-3-4]. http://www.penglixun.com/tech/database/multi\_source\_replication\_for\_mariadb.html.

[7] Deshpande K. Oracle Streams 11g Data Replication[M]. Oracle Press, 2010.

[8] Tumma M. Oracle Streams High Speed Replication and Data Sharing[M]. Independent Pub Group, 2005.

[9] 管东华. 基于Oracle流复制技术的数据库容灾备份应用研究[D]. 成都理工大学, 2009.

[10] Alapati S R. Expert Oracle Database 11g Administration[M]. Apress, 2008 :670.

[11] 黄 亮, 李曙东. Oracle Streams 双向复制功能的研究[J]. 中国水运, 2010, 10(6):85-87.

[12] 熊曾刚, 张学敏, 陈建新, 等. 基于Oracle9i分布式数据库系统复制机制的研究[J]. 计算机工程与应用, 2004.

[13] 杜凯, 缪嘉嘉, 杨树强, 等. 数据库复制技术研究进展[J]. 计算机工程与科学. 2008(07)

[14] 张伟丽, 江春华, 魏劲超. MySQL复制技术的研究及应用[J]. 计算机科学. 2012(11)

[15] 姜承尧. MySQL技术内幕:InnoDB存储引擎[M]. 机械工业出版社, 2011.

[16] Bell C, Kindahl M, Thalmann L. MySQL High Availability: Tools for Robust Data Centers[M]. 1. O'Reilly Media, 2010.

[17] 谭怀远. 让Oracle跑得更快2—基于海量数据的数据库设计与优化[M]. 电子工业出版社, 2011.

[18] 高鹃. 网络备份系统中管理控制的设计与实现[D]. 华中科技大学 2011.

[19] 淘宝数据盛典[EB/OL]. [2013-3-10]. http://www.taobao.com/go/act/sale/tbsjsd.php?&ad\_id=&am\_id=13010039655808759e9e&cm\_id=&pm\_id=.

[20] 丁原. 淘宝网数据库架构演变[EB/OL]. [2013-3-10]. http://wenku.baidu.com/view/cc19d3fafab069dc50220172.html.

[21] 丹臣, 赵林. 淘宝数据库架构演进历程[EB/OL]. [2013-3-10]. http://wenku.baidu.com/view/e444f0cda1c7aa00b52acb3b.html.

[22] 淘宝网综合业务平台团队. TDDL动态数据源基本说明[EB/OL]. [2013-3-10]. http://rdc.taobao.com/team/jm/archives/1642.

[23] 祝定泽, 张海, 黄建昌. MySQL核心内幕[M]. 清华大学出版社, 2010.

[24] DuBois P. MySQL技术内幕[M]. 人民邮电出版社, 2011.

[25] Pachev S. 深入理解MySQL核心技术[M]. 中国电力出版社, 2009.

[26] 刘腾. MySQL复制技术的研究与改进[D]. 浙江大学, 2011.

[27] Richards M, MonsonHaefel R, Chappell D. Java Message Service[M]. 东南大学出版社, 2010.

[28] Snyder B. ActiveMQ in Action[M]. Manning Publications, 2011.

[29] Levine J. Flex & Bison: Text Processing Tools[M]. 东南大学出版社, 2010.

[30] Reis A. Compiler Construction Using Java, JavaCC, and Yacc[M]. Wiley-IEEE Computer Society Press, 2011.

**致 谢**

首先非常感谢我的研究生导师陈昊鹏老师在论文写作阶段对我的指导，正是陈老师不辞辛劳的讲解和修正，我才能顺利的完成这篇论文。论文写作期间，陈老师治学严谨的态度给我留下了深刻的影响。非常荣幸能够成为陈老师的学生，再次向陈老师表示衷心的感谢。

由于平时工作比较忙，因此都是下班后撰写论文，写作的最后几个月，经常都是熬夜到两三点，付出了辛勤的汗水。经过这段时间的努力，感觉自己受益匪浅，也是一次历练自己的机会。特别感谢我的爱人，没有他的陪伴和鼓励，我也无法完成这篇论文。

最后对所有帮助过我的同学、同事和朋友们报以最真挚的谢意，你们都是这篇论文的幕后英雄。

**攻读学位期间发表的学术论文目录**