计算机系统应用 ISSN 1003-3254, CODEN CSAOBN
Computer Systems & Applications,2017,26(7):278−282 [doi: 10.15888/j.cnki.csa.005926]

E-mail: csa@iscas.ac.cn
http://www.c-s-a.org.cn
Tel: +86-10-62661041

支持MongoDB的事务管理方案研究①

汪添生[1,2], 尉双梅[3], 崔　蔚[1], 刘　迪[1,4], 何金陵[5], 孙　琦[5]

[1](国网信息通信产业集团有限公司, 北京 100761)
[2](厦门亿力吉奥信息科技有限公司, 厦门 361000)
[3](安徽继远软件有限公司, 合肥 230088)
[4](北京中电普华信息技术有限公司, 北京 100085)
[5](国网江苏省电力公司通信分公司, 南京 210008)

摘　要: MongoDB作为一个基于分布式文件存储的数据库, 强大的单表查询语言, 以及可扩展的高性能数据存储受很用户喜爱, 但其没有对事务的完全支持, 使得用户对MongoDB的使用处于被动状态. 为改善MongoDB对事务管理方面的兼容性, 提出一种支持MongoDB事务管理, 完善MongoDB功能的方案. 利用MQ与守护进程间的消息通信, 使守护进程对事务提交或者回滚后的脏数据进行清理, 保证了MongoDB在事务管理方面的可用性与安全性.

关键词: 分布式数据库; mongoDB; 事务管理; 守护进程; MQ; 消息通信

引用格式: 汪添生,尉双梅,崔蔚,刘迪,何金陵,孙琦.支持MongoDB的事务管理方案研究.计算机系统应用,2017,26(7):278–282. http://www.c-s-a.org.cn/1003-3254/5926.html

Analysis of Scheme Supporting MongoDB Transaction Management

WANG Tian-Sheng[1,2], YU Shuang-Mei[3], CUI Wei[1], LIU Di[1,4], HE Jin-Ling[5], SUN Qi[5]

[1](State Grid Information &Telecommunication Co. Ltd., Beijing 100761, China)
[2](Xiamen Great Power Geo Information Technology Co. Ltd., Xiamen 361000, China)
[3](Anhui Jiyuan Software Co.Ltd., Hefei 230088, China)
[4](Beijing China-Power Information Technology Co. Ltd., Beijing 100085, China)
[5](State Grid Jiangsu Electric Power Company &Telecommunication Branch, Nanjing 210008, China)

Abstract: As a database based on distributed file storage, with powerful single table query language, and extensible high performance data storage, MongoDB is very popular among users. But is does not fully support the transaction, leaving the uses of the MongoDB passive. In order to improve the mongo compatibility of transaction management, a solution of supportting MongoDB transaction management and perfecting its function is proposed. The communication between daemon processes and MQ are utilized to make daemon processes clean up the dirty data after a transaction commits or rolls back, ensuring the availability and security of MongoDB in terms of transaction management.

Key words: distributed database; mongoDB; transaction management; daemon processes; MQ; message communication

随着网络快速普及发展, 信息资源的丰富和获取方式的高效便捷, 使得人们对web信息资源的需求越来越大, 在需求量与日俱增的情况下[1,2], 一个安全稳定高效的存储总会受到人们的关注. 传统的关系型数据库, 在面对高并发读写请求, 海量数据的高效存储以及高扩展性和可用性等需求时, 关系型数据库的瓶颈就会逐渐暴露出来[1], 此时, 停机维护或者数据库迁移可能普遍成为解决该问题的通用方法. 非关系型数据库提出的另一种概念, 灵活多变的数据字段, 结构没有固定, 以键值或者面向文档的形式存储数据[2], 脱离关系型数据库的一成不变的格式, 无需经过SQL层的解析, 大大减少了部分时间和空间的开销, 支持横向扩展, 从而支

① 收稿时间: 2016-11-09; 收到修改稿时间: 2017-01-04

持海量数据的存储等.

然而关系型数据库支持对多表复杂查询的功能以及对事务操作的高安全性的特点在非关系型数据库领域却没有完全得到体现. MongoDB作为一个基于分布式文件存储的数据库代表, 拥有支持使用高效的二进制数据存储(如视频等), 支持完全索引, 面向集合, 模式自由由, 支持复制和故障恢复等等优点, 但也明确说明了MongoDB并不适用高度事务性的系统, 由于这个原因, 使得用户在使用MongoDB这样的非关系型数据库的时候不得不慎重考虑. 本文旨在通过数据访问层对需要进行事务控制数据更新至MongoDB数据库的方式进行针对性的处理, 同时开启守护进程, 不管MongoDB数据库是否发生异常, 事务最后的结果或者是成功提交, 又或者是数据回滚, 采用MQ与守护进程间通信, 将事务处理的结果及相关信息传递给守护进程, 让守护进程对事务操作后的脏数据进行处理[3,4], 以达到保证事务在整个过程中保持原有的ACID特性[5].

1 MongoDB数据访问层

MongoDB是开源、面向文档、模式自由、可扩展的分布式数据库[2]. 利用MongoDB作为后台数据存储的服务, 本节旨在解决MongoDB在事务控制方面的不足, 需要结合MongoDB对数据存储的形式, 设计一个针对进行事务操作时增删改的接口.

1.1 具有事务标识的PO类

PO(persistant object)持久对象, 是一个与数据库中的表相映射的java对象. 这里对MongoDB存储的业务数据需要进行一个和PO概念一样的映射对象, 尽管MongoDB是模式自由的数据库, 为了更好完成事务控制的功能, 对于存储的数据应该规定满足业务所需具体的字段是完全合理的, 但有个不同的是, PO中除了必要的业务字段, 还需要新增一个标识字段, 如: transactionId, 用来作为事务控制后, 清理数据的唯一标识, 此字段完全不会影响到对MongoDB数据的读写, 也不会影响到对前端界面的展示, 因为它只是一个标识字段, 展现时无需获取该字段的值.

1.2 更新方法的处理

在进行MongoDB事务增删改的时候, 需要说明的是, 对于修改和删除的特殊处理. 这里修改的操作实际上在MongoDB数据访问层中处理方式是新增一条新数据, 但是此数据是在原来旧数据与新改数据之间的合并而成的, 它将作为一条新的数据重新保存进MongoDB数据库中, 同时该条数据的transactionId字段将被赋予一个唯一值; 而删除的操作则是给准备删除的数据添加transactionId字段的值, 此值和修改数据时分配的值是一样, 因此对于删除操作并非真正删除, 而是修改它, 给它分配一个transactionId值; 而完全新增的数据则是和修改一样的操作, 就是在新增数据的transactionId字段赋予一个唯一值. 如果事务的操作序列中包含了新增和修改, 则新增和修改的数据的transactionId值是一样的.

在事务开始的前期, 开发人员需要获取即将操作的记录的主键, 这里的操作指的是针对修改操作, 然后将主键保存进BasicDBList这个集合中, 对于删除操作, 上述说过要删除的数据的主键值会保存到一个集合中, 此集合即是BasicDBList. 此目的是在事务成功执行提交时, 可以作为清理无用数据时的主键, 如果事务由于MongoDB数据库的异常发生回滚, 则上述中的transactionId则将作为清理数据的唯一值, 相当于主键. 更新方法与MongoDB数据访问层的交互过程如图1所示.

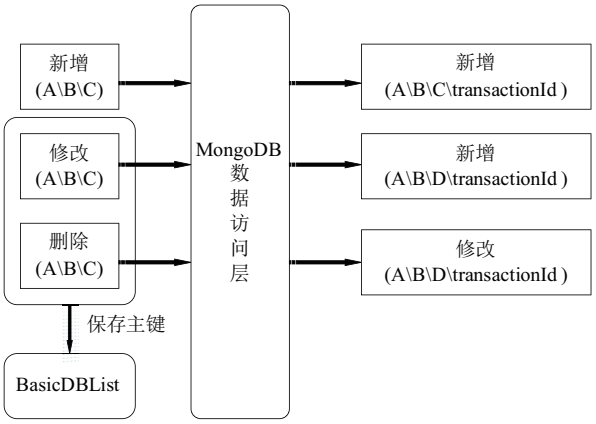

图1 增删改方法与MongoDB数据访问层交互图

1.3 事务控制

事务是由一组操作组成的可靠的独立的工作单元[5]. 一个事务具有ACID4个基本的属性, 原子性(Atomicity): 是指事务是一个完整的工作单位, 事务中的操作要么都执行, 要么都不执行; 一致性(Consistency): 事务使数据库从一个一致性状态变换到另外一个一致性状态; 隔离性(Isolation): 隔离性是指一个事务的执行不能被其他并发事务所干扰, 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): 指一个事务一旦被提交, 它对数据库中数据的改变就是永久性的, 后面的其他操作或者数据库故障不会对其有任何影响.

为保证尽量做到事务在关系型数据库中原来的特性, 本方案中, 定义一个事务管理器, 该管理器的主要作用是对事务进行提交和回滚时的操作, 在提交和回滚时将会利用消息通信机制与守护进程进行通信. 在事务开始前期, 由要更新的数据主键值和transactionId都可以确定下来, 因此会在前期将这些信息预告发送至MQ队列中, 而事务提交时将会发送一个成功提交的标识给守护进程, 告诉守护进程可以进行事务成功后的脏数据清理工作; 在事务回滚时将会发送给一个事务失败的标识给守护进程, 告诉守护进程可以进行事务失败后的脏数据清理工作. 而在执行提交或者回滚操作时, 为保证一个事务的工作不被并发事务的干扰, 对提交和回滚的操作需要加上同步锁, 保证本事务在队列中的消息被清除后, 释放同步锁, 让其他的事务获得该锁以完成自己的工作.

2 MQ消息通信

Message Queue, 一种应用程序与应用程序间进行消息交换的技术[6,7]. MQ使得消息在应用程序间的传送变得简单、安全、可靠, 它摆脱了程序间直接调用彼此来获取信息的方式, 通过队列管理器来管理队列Queue, 应用程序无需建立专用的链接来与之连接而获得队列中消息, 同时发送者程序只关心将消息传送至MQ中, 处理消息的程序并不依赖于消息的发送者, 也没有时间的限制, 这就将消息的发送者与处理者之间的耦合度降到了最低, 其实这也是MQ的异步处理机制, 所以MQ所扮演的角色是一个消息管理者, 除了消息双方必须实现处理过程的接口外, 只负责接收管理消息并提供多种需要的服务机制, 对于什么时候需要消息、什么方式领取消息的另一方则是它自己的事.

在本方案中, 事务管理器可以获取在事务执行成功或者失败后的需要发送的标识. MQ在规定的队列接收到事务前期发送的主键值和transactionId时, 将不再接收下一个消息, 需要等待此消息被处理后, 再接收下个事务发送的信息. 事务成功提交时, MQ将会得到一个事务执行成功标识; 事务执行失败时, MQ将会得到一个事务执行失败的标识. 在MQ发送者端, 分发模式DeliveryMode设置成PERSISTENT, 以防发生故障, 导致存储在内存中的消息被清除, 所以设置成PERSISTENT模式可以将消息存储在本地文件中, 当队列中的消息被消费后, 此数据才有被清除的资格, 也保证了发送的消息在没有被守护进程读取之前, 能够一直存在直到任务完成.

另外, MQ在创建连接时使用同步方式, 消息在被守护进程接收后需要发回一个确认收到的消息, 而消息发送端在收到接收者发送的确认标识后, 知道事务操作结束, 可以释放对提交和回滚操作的同步锁给下一个等待的事务, 这样才算完成整个事务的过程, MQ的工作也就到此结束. MQ完成的交互如图2所示.

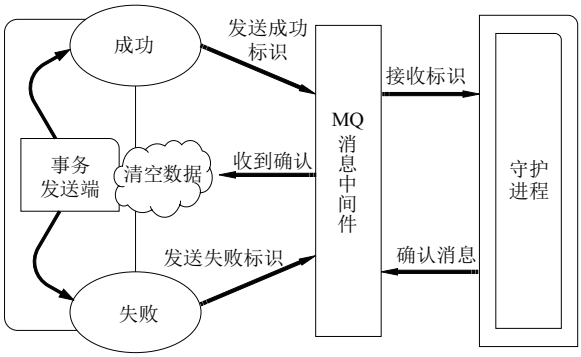

图2 事务标识与MQ的交互图

3 守护进程

守护进程(Daemon)是一种独立于控制终端并周期性地执行某种任务或者等待某些事件的发生而运行于后台的一种特殊进程[8,9]. 而事务在执行成功或者失败后, 留在MongDB数据库中的脏数据, 借助守护进程进行清理[10]则是一个不错的选择.

守护进程在开启后, 定期的进行请求MQ队列中的消息[11,12], 如果接收到事务前期发送的相关数据, 接着在规定的时间段定期请求的事务执行成功或者失败的标识, 如果在规定的时间内没有接收到标识, 则默认为事务执行失败, 将按失败事务的处理方式处理数据; 如果在规定的时间内接收到标识, 则按标识的定义来处理数据. 守护进程在不管事务成功还失败情况下, 只要进入到处理数据, 守护进程需心跳性监测MongoDB数据库的连接性, 在一次连接数据库成功之后, 完成数据的清理. 事务成功提交时, 获取的主键值, 会作为清理的旧数据的标识; 事务执行失败回滚时, 获取的

transactionId, 会作为清理新增的数据的标识. 这样, 不管在事务端执行的对MongoDB写操作有几个, 最后都将转变为一个对MongoDB删除的操作, 而MongoDB单个执行命令是保证原子性的, 执行删除操作后, MongoDB数据库中的数据则保留了事务操作后的一致性状态. 后期, 守护进程断开与MongoDB的连接, 清空队列中的数据并给MQ队列发送对消息接收的确认标识, 接着再进行周期性的请求MQ队列中新的消息, 开始下一个的事务的控制. 图3是守护进程在事务控制中的工作图.

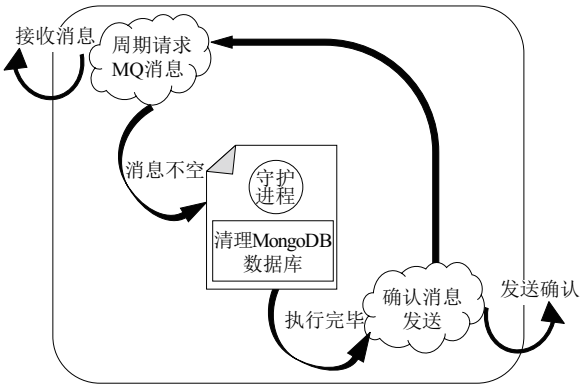

图3 守护进程工作图

4 事务管理过程

现在结合一个转账的例子来讲解本方案的一个大致事务管理的过程. 一个转账的操作涉及到的有账户类, 简单来说, 这个账户包括了账号、姓名、余额; 现在涉及事务的操作共分成两步: 一个账户是扣掉金额, 另一个则是增加金额.

首先账户这个PO类在进行设计时, 除了必要的字段外, 还需增加一个事务标识, 设为transationId, 所以现在这个账户类共有4个字段: ID(账号)、NAME(姓名)、BALANCE(金额)、TRANSACTIONID(事务标识), 此事务标识字段在涉及到事务操作时才会保存在数据库中.

假设现在张三李四两个账户转账之前在MongoDB数据库中的信息如表1所示.

表1 转账前张三李四账户信息

_id	1	name	张三	balance	1000
_id	2	name	李四	balance	1000

现在张三给李四转账200块, 如果事务执行成功, 那么在业务逻辑层处理时, 涉及事务的2个操作是: 1) 张三余额扣200; 2) 李四余额增加200. 这两个修改的操作在经过了MongoDB数据访问层后, 实际的操作由修改两个账户的余额的操作变成新增两条记录的操作, 记录中带有事务标识, 此事务标识完全不影响账户的信息, 只是这个标识让我们知道这两条信息曾经参与过了某个事务的控制过程而已. 因此通过MongoDB数据访问层之后, 现在张三李四两个账户转账后在MongoDB数据库中的信息如表2所示.

表2 转账后张三李四账户信息

…	name	张三	balance	1000			
…	name	李四	balance	1000			
…	name	张三	balance	800	transactionId	transactionId123456	
…	name	李四	balance	1200	transactionId	transactionId123456	

从表2可以看出, 新增两个账户的金额已经修改了, 同时带有相同的事务标识字段, 同时原来张三和李四在转账前的数据记录保持不变. 在事务前期, 两个账户的主键及生成的transactionId需预先发送至MQ队列中, 而在事务成功提交后, 发送给MQ的是个事务执行成功的标识, 守护进程定期监测接收MQ中的事务处理标识, 最后守护进程根据事务执行成功的标识, 通过之前接收到的主键值删除张三李四转账前的记录, 此删除操作是具有原子性的一个命令操作, 进而销毁了转账前的旧数据, 保留了转账后的数据.

如果现在张三余额在执行扣钱操作之后发生了某个数据库异常, 此时事务执行失败, 此时张三李四两个账户在MongoDB数据库中的信息如表3所示.

表3 转账异常后张三李四账户信息

…	name	张三	balance	1000			
…	name	李四	balance	1000			
…	name	张三	balance	800	transactionId	transactionId123456	

从表3看出由于异常发生在了张三给李四转账的过程中, 数据库只增加一条张三的信息, 异常导致事务将进行回滚操作. 而新增的数据的transactionId在事务开始前期已经发送给守护进程, 守护进程将利用它对MongoDB数据库中的脏数据进行清理, 而原来的张三李四的账户信息还保留着, 回到了最初转账前账户的数据信息, 从而做到数据回滚.

5 结语

本文通过对MongoDB数据访问层的设计, 规定了

数据保存到MongoDB时的转变模式, 事务控制类获取更新记录相关数据后通过MQ消息中间件来与守护进程进行事务处理结果的通信, 从而将原本需要在MongoDB进行多次的写操作转变成了在MongoDB进行一次性删除数据的操作.

此方案将MongoDB对单个操作的原子特性用在最后的脏数据处理上, 而不是事务的操作序列上, 即将多个原子操作转变成一个原子操作, 同时配合守护进程与MQ的及时通信, 保证事务原有的ACID特性.

参考文献

1 徐娟娟, 朱成亮. NOSQL在WEB日志分析中的应用. 中国新技术新产品, 2011, (10): 27. [doi: 10.3969/j.issn.1673-9957.2011.10.025]

2 MongoDB Home. http://grokbase.com/t/gg/mongodb-user/12836feerg/locking-for-atomic-transactions.

3 Plantikow S, Reinefeld A, Schintke F. Transactions for distributed wikis on structured overlays. Proc. Distributed Systems: Operations and Management 18th IFIP/IEEE International Conference on Managing Virtualization of Networks and Services. San José, CA, USA. 2007. 256–267.

4 Bernstein PA, Newcomer E. Principles of transaction processing. San Francisco, California: Morgan Kaufmann Publishers, Inc., 1997. 56–59.

5 Transaction Processing Performance Council. TPC BENCHMARK™ C, Standard Specification Revision 5.0, 2001-02-26.

6 Haase K. JavaTM message service API tutorial. Palo Alto, California: Sun Microsystems, Inc., 2002.

7 周城, 葛斌, 蒋林承. 一种基于消息中间件的网页实时处理技术. 电脑知识与技术, 2011, 7(10): 2269–2271. [doi: 10.3969/j.issn.1009-3044.2011.10.021]

8 Tackett J Jr, Gunter D. Linux大全. 3版. 北京: 电子工业出版社, 1998: 25–26.

9 张忠杰, 田质广. 编写Linux守护进程. 山东轻工业学院学报, 2001, 15(4): 5–9.

10 张伟, 居悌. UNIX系统中常驻进程的一个应用实例. 微机发展, 1999, (5): 50–52.

11 彭勇. Delphi 5.0网络与通信开发应用. 北京: 中国水利水电出版社, 2000: 36–38.

12 Zhang YT, Liao JX, Zhang TY, *et al*. A novel method for the short message or multimedia message synchronization. Proc. International Conference on Wireless and Mobile Communications. Bucharest, Romania. 2006. 75.