13 | MySQL主从数据库同步是如何实现的?

李玥 · 后端存储实战课

你好，我是李玥。

回顾我们之前讲 MySQL 相关的几节课程，你会发现主从同步有多重要。解决数据可靠性的问题需要用到主从同步；解决 MySQL 服务高可用要用到主从同步；应对高并发的时候，还是要用到主从同步。

我们在运维 MySQL 集群时，遇到的很多常见的问题，比如说，为什么从节点故障会影响到主节点？为什么主从切换之后丢数据了？为什么明明没有更新数据，客户端读到的数据还是变来变去的？这些都和主从同步的配置有密切的关系。

你不但要理解 MySQL 主从同步的原理，还要掌握一些相关配置的含义，才能正确地配置你的集群，知道集群在什么情况下会有什么样的行为，可能会出现什么样的问题，并且知道该如何解决。

今天这节课我们就来详细讲一下，MySQL 的主从同步是怎么实现的，以及如何来正确地配置主从同步。

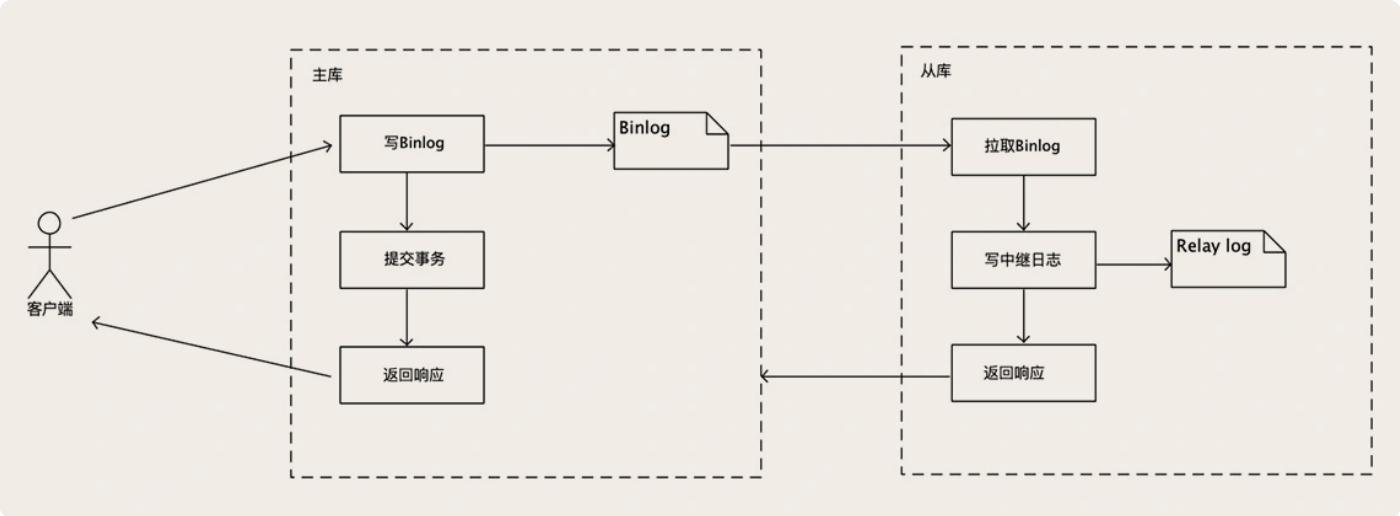
如何配置 MySQL 的主从同步?

当客户端提交一个事务到 MySQL 的集群，直到客户端收到集群返回成功响应，在这个过程中，MySQL 集群需要执行很多操作：主库需要提交事务、更新存储引擎中的数据、把 Binlog 写到磁盘上、给客户端返回响应、把 Binlog 复制到所有从库上、每个从库需要把复制过来的 Binlog 写到暂存日志中、回放这个 Binlog、更新存储引擎中的数据、给主库返回复制成功的响应。

这些操作的时序非常重要，这里面的“时序”，说的就是这些操作的先后顺序。同样的操作，因为时序不同，对应用程序来说，有很大的差异。比如说，如果先复制 Binlog，等 Binlog 复制到从节点上之后，主节点再去提交事务，这种情况下，从节点的 Binlog 一直和主节点是同步的，任何情况下主节点宕机也不会丢数据。但如果把这个时序倒过来，先提交事务再复制 Binlog，性能就会非常好，但是存在丢数据的风险。

MySQL 提供了几个参数来配置这个时序，我们先看一下默认情况下的时序是什么样的。

默认情况下，MySQL 采用异步复制的方式，执行事务操作的线程不会等复制 Binlog 的线程。具体的时序你可以看下面这个图：



MySQL 主库在收到客户端提交事务的请求之后，会先写入 Binlog，然后再提交事务，更新存储引擎中的数据，事务提交完成后，给客户端返回操作成功的响应。同时，从库会有一个专门的复制线程，从主库接收 Binlog，然后把 Binlog 写到一个中继日志里面，再给主库返回复制成功的响应。

从库还有另外一个回放 Binlog 的线程，去读中继日志，然后回放 Binlog 更新存储引擎中的数据，这个过程和我们今天讨论的主从复制关系不大，所以我并没有在图中画出来。**提交事务和复制这两个流程在不同的线程中执行，互相不会等待，这是异步复制。**

掌握了异步复制的时序之后，我们就很容易理解之前几节课中讲到的一些问题的原因了。比如说，在异步复制的情况下，为什么主库宕机存在丢数据的风险？为什么读写分离存在读到脏数据的问题？产生这些问题，都是因为**异步复制它没有办法保证数据能第一时间复制到从库上。**

与异步复制相对的就是同步复制。同步复制的时序和异步复制基本是一样的，唯一的区别是，什么时候给客户端返回响应。异步复制时，主库提交事务之后，就会给客户端返回响应；而同步复制时，主库在提交事务的时候，会等待数据复制到所有从库之后，再给客户端返回响应。

同步复制这种方式在实际项目中，基本上没法用，原因有两个：一是性能很差，因为要复制到所有节点才返回响应；二是可用性也很差，主库和所有从库任何一个数据库出问题，都会影响业务。

为了解决这个问题，MySQL 从 5.7 版本开始，增加一种半同步复制（Semisynchronous Replication）的方式。异步复制是，事务线程完全不等复制响应；同步复制是，事务线程要等待所有的复制响应；半同步复制介于二者之间，事务线程不用等着所有的复制成功响应，只要一部分复制响应回来之后，就可以给客户端返回了。

比如说，一主二从的集群，配置成半同步复制，只要数据成功复制到任意一个从库上，主库的事务线程就直接返回了。这种半同步复制的方式，它兼顾了异步复制和同步复制的优点。如果主库宕机，至少还有一个从库有最新的数据，不存在丢数据的风险。并且，半同步复制的性能也还凑合，也能提供高可用保证，从库宕机也不会影响主库提供服务。所以，半同步复制这种折中的复制方式，也是一种不错的选择。

接下来我跟你说一下，在实际应用过程中，选择半同步复制需要特别注意的几个问题。

配置半同步复制的时候，有一个重要的参数“rpl_semi_sync_master_wait_slave_count”，含义是：“至少等待数据复制到几个从节点再返回”。这个数量配置的越大，丢数据的风险越小，但是集群的性能和可用性就越差。最大可以配置成和从节点的数量一样，这样就变成了同步复制。

一般情况下，配成默认值 1 也就够了，这样性能损失最小，可用性也很高，只要还有一个从库活着，就不影响主库读写。丢数据的风险也不大，只有在恰好主库和那个有最新数据的从库一起坏掉的情况下，才有可能丢数据。

另外一个重要的参数是“rpl_semi_sync_master_wait_point”，这个参数控制主库执行事务的线程，是在提交事务之前（AFTER_SYNC）等待复制，还是在提交事务之后（AFTER_COMMIT）等待复制。默认是 AFTER_SYNC，也就是先等待复制，再提交事务，这样完全不会丢数据。AFTER_COMMIT 具有更好的性能，不会长时间锁表，但还是存在宕机丢数据的风险。

另外，虽然我们配置了同步或者半同步复制，并且要等待复制成功后再提交事务，还是有一种特别容易被忽略、可能存在丢数据风险的情况。

如果说，主库提交事务的线程等待复制的时间超时了，这种情况下事务仍然会被正常提交。并且，MySQL 会自动降级为异步复制模式，直到有足够多（rpl_semi_sync_master_wait_no_slave）的从库追上主库，才能恢复成半同步复制。如果这个期间主库宕机，仍然存在丢数据的风险。

复制状态机：所有分布式存储都是这么复制数据的

在 MySQL 中，无论是复制还是备份恢复，依赖的都是全量备份和 Binlog，全量备份相当于备份那一时刻的一个数据快照，Binlog 则记录了每次数据更新的变化，也就是操作日志。我们这节课讲主从同步，也就是数据复制，虽然讲的都是 MySQL，但是你要知道，这种基于“快照 + 操作日志”的方法，不是 MySQL 特有的。

比如说，Redis Cluster 中，它的全量备份称为 Snapshot，操作日志叫 backlog，它的主从复制方式几乎和 MySQL 是一模一样的。

我再给你举个例子，之前我们讲过的 Elasticsearch，它是一个内存数据库，读写都在内存中，那它是怎么保证数据可靠性的呢？对，它用的是 translog，它备份和恢复数据的原理和实现方式也是完全一样的。这些什么什么 log，都是不同的马甲儿而已，**几乎所有的存储系统和数据库，都是用这一套方法来解决备份恢复和数据复制问题的**。

既然这些存储系统他们实现数据复制的方法是完全一样的，那这几节课我们讲的 MySQL 主从复制时，讲到的那些问题、丢数据的风险，对于像 Redis Cluster、ES 或者其他分布式存储也都是一样存在的。那我们讲的，如何应对的方法、注意事项、最佳实践，这些也都是可以照搬的。

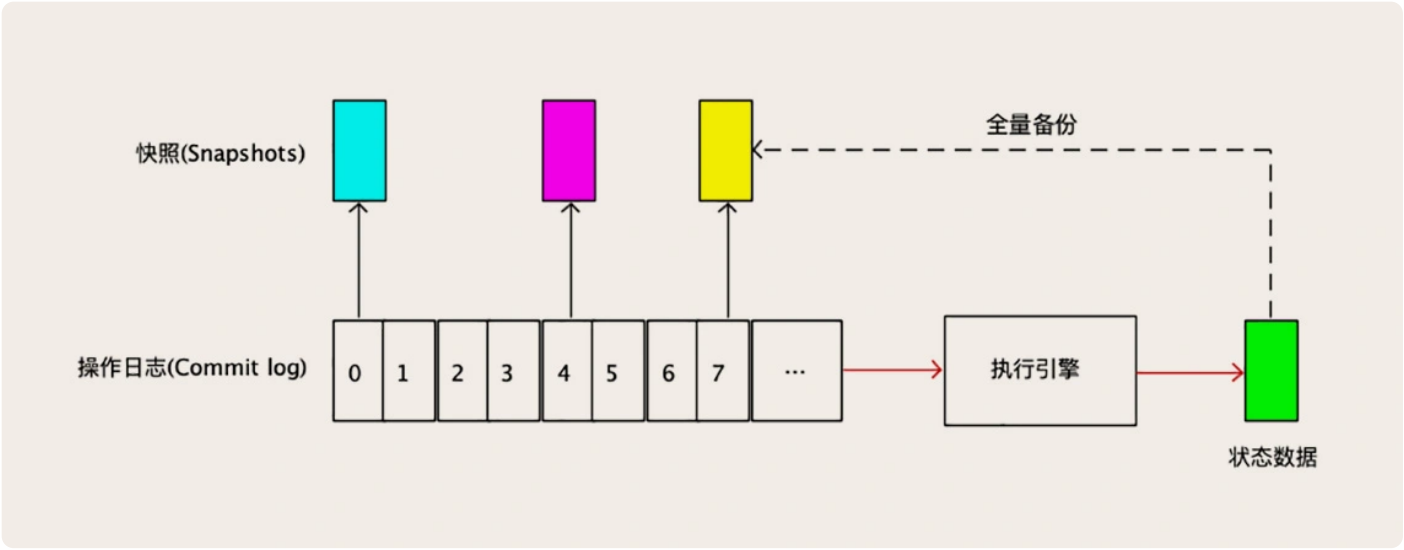
这一套方法其实是有理论基础的，叫做复制状态机 (Replication State Machine)，我能查到的最早的出处是 1978 年 Lamport 的一篇论文《The Implementation of Reliable Distributed Multiprocess Systems》。

1978 年啊，同学，那时候我们都还没出生呢！这么老的技术到今天仍然在被广泛地应用！无论应用技术发展的多快，实际上解决问题的方法，或者说是理论基础，一直是没什么变化的。所以，你在不断学习新的应用技术的同时，还需要多思考、总结和沉淀，这样会让你学习新技术的时候更快更轻松。

小结

最后，那为了便于你理解复制状态机，我们把这套方法再抽象总结一下。任何一个存储系统，无论它存储的是什么数据，用什么样的数据结构，都可以抽象成一个状态机。

存储系统中的数据称为状态（也就是 MySQL 中的数据），状态的全量备份称为快照（Snapshot），就像给数据拍个照片一样。我们按照顺序记录更新存储系统的每条操作命令，就是操作日志（Commit Log，也就是 MySQL 中的 Binlog）。你可以对照下面这张图来理解上面这些抽象的概念。



复制数据的时候，只要基于一个快照，按照顺序执行快照之后的所有操作日志，就可以得到一个完全一样的状态。在从节点持续地从主节点上复制操作日志并执行，就可以让从节点上的状态数据和主节点保持同步。

主从同步做数据复制时，一般可以采用几种复制策略。性能最好的方法是异步复制，主节点上先记录操作日志，再更新状态数据，然后异步把操作日志复制到所有从节点上，并在从节点执行操作日志，得到和主节点相同的状态数据。

异步复制的劣势是，可能存在主从延迟，如果主节点宕机，可能会丢数据。另外一种常用的策略是半同步复制，主节点等待操作日志最少成功复制到 N 个从节点上之后，再更新状态，这种方式在性能、高可用和数据可靠性几个方面都比较平衡，很多分布式存储系统默认采用的都是这种方式。

思考题

课后请你想一下，复制状态机除了用于数据库的备份和复制以外，在计算机技术领域，还有哪些地方也用到了复制状态机？欢迎你在留言区与我讨论。

感谢你的阅读，如果你觉得今天的内容对你有帮助，也欢迎把它分享给你的朋友。

精选留言 (30)

李玥 置顶
2020-03-26

Hi，我是李玥。

这里回顾一下上节课的思考题：

课后请你对照你现在负责开发或者维护的系统来分享一下，你的系统实施读写分离的具体方案是什么样的？比如，如何分离读写数据库请求？如何解决主从延迟带来的数据一致性问题？欢迎你在留言区与我讨论。

课后很多小伙伴在留言区对这个问题做了回复，分离读写请求大多数采用的是代理或者Sharding-JDBC这两种方案。那解决主从延迟，没有完全避免延迟的方法，但至少要做到能够监控主从延迟，当延迟太大的时候，采用一些降级方案。比如说，把重要业务的读请求切回主库，暂停一些不重要的业务，甚至限流等等。

共 1 条评论>

32

夏目
2020-04-09

redis在重启时现在也可以采用这种策略，RDB快照加上aof日志恢复数据

24

leslie
2020-03-26

最典型的应当是VMware:其实复制技术不仅仅在linux系统的中间件存储 mysql、redis、ES上使用，Windows的sql server同样适用了此技术。

17

一步
2020-03-26

MySQL 启用半同步复制，需要登录MySQL安装插件(或者修改配置文件)
Linux： install plugin rpl_semi_sync_master soname 'semisync_master_so' (window 是 dll)
从库使用的是 semisync_slave.so 文件

这时候MySQL 会自动去 安装目录的 lib/plugin 文件夹下去找

10

Wiggins
2020-03-27

读完mysql实战45讲再读老师的文章感觉又复习了一遍 老师的文章很清晰 更加偏向实战中的配置了

共 2 条评论>

9

渔夫
2020-03-31

目前主流前端技术的状态管理也是这个状态机机制：state + action = next state

8

渐渐迷失
2020-03-26

老师你好
之前学mq的时候您课讲完了，我才进入学习的，有好些问题还不会，追过来问问可以吗

> 作者回复: 建议你还是在MQ的课程中去提问，这样也便于其它同学查看。虽然课程已经更新完结，但我还是会关注并回答留言区的问题。

7

苗
2020-05-03

基于事件朔源的应用程序，就是用snapshot + 事件来快速恢复对象的状态。

3

image
2020-03-26

fork and write，典型有docker镜像，linux fork都和复制状态机有关系，这也是一种基本的构建模式

3

建强
2021-12-18

思考题：
复制状态机的思想也用于数据同步，当把一批频繁变化的数据从一个地方同步到另一个地方时，先对数据做一个初始化的全量同步，相当于给数据拍一个快照，然后给数据加一个时间戳字段，记录下拍快照时的时间戳，每当数据发生增、删、改时，时间戳也发生改变，根据时间戳的变化，只需要把时间戳大于快照时间戳的数据进行增量同步即可，这种思想算不是复制状态机，请老师指正。



1

Ling
2021-05-27

– rpl_semi_sync_master_timeout
为了防止半同步复制在没有收到确认的情况下发生堵塞，如果主库在`rpl_semi_sync_master_timeout`毫秒超时之前没有收到确认，将恢复到异步复制
以毫秒为单位，默认值是10000(10秒)。

– rpl_semi_sync_master_wait_for_slave_count
在继续处理事务之前，必须接收的副本确认的数量；
越小，需要等待确认的从节点越少，性能越好；
MySQL默认值是`1`；阿里云的一主一备高可用版RDS，配置该值为`1`

– rpl_semi_sync_master_wait_no_slave
当前存活从节点数小于`rpl_semi_sync_master_wait_for_slave_count`中配置的值时，是否还需要等待`rpl_semi_sync_master_timeout`中配置的超时时间；
环境：如果配置的`rpl_semi_sync_master_wait_for_slave_count`是2，但是当前只有一个从节点
如果配置为`ON`：主库还是会等待`rpl_semi_sync_master_timeout`秒后超时，然后切换为`异步复制`
如果配置为`OFF`：立刻变为`异步复制`
MySQL默认该值为`ON`，阿里云的一主一备高可用版RDS，配置该值为`OFF`



2

夜辉
2021-04-10

redis持久化存储方式：rdb 和 aof
后面还出了一种二者混合的方式

1

疯码
2020-12-12

游戏同步。从同一基础状态开始，各客户端施加同样的操作 得到同样的结果。

1

Sam Fu
2023-05-13 来自北京

对于半同步场景，如果主库写入 binlog 成功，但是主库宕机导致发送 binlog 到从库失败。这个时候把从库切到主库，从库会认为这条数据没有写入成功。
但是主库重启后，因为主库的 binlog，redo log 都写入成功。这个时候主库会不会认为这条数据写入成功呢？这种 case 会不会造成主从不一致呀

ifelse
2022-12-09 来自浙江

任何一个存储系统，无论它存储的是什么数据，用什么样的数据结构，都可以抽象成一个状态机。--记下来

DonaldTrumpppppppppp
2022-03-04

老师好，现在有个问题急需您指点。一主一备时，半同步模式下，主节点退化为异步同步模式后，从节点恢复后但未和主同步一致的情况下，主挂了，备升级为主。这时出现不一致的情况该如何处理?

ALEX
2021-12-23

git commit log 加上本地副本

凯文小猪
2021-04-13

感谢老师深入浅出的分享 看过这篇后 对于redis mongodb kafka mysql这些存储介质豁然开朗。原来一天学完存储介质真的不是梦


flyCoder

2021-02-10

半同步复制，主节点挂了，怎么样保证数据的完整性呢？ 从节点之间会相互复制吗?

共 1 条评论>

innocent

2021-02-02

Raft协议也用到了复制状态机