


# MySQL面试题解析：主从复制专场

来自： 小6互联网求职面试

 马听

2024年03月19日 16:25



扫码加入  
查看更多优质内容

## 1 主从复制的原理

当主库有写操作时（比如insert、update、delete），会记录到主库的Binlog中  
从库通过IO线程读取主库的Binlog里面的内容，传给从库的Relay Log（中继日志）  
从库的sql线程负责读取它的relay log里的信息并应用到数据库中

## 2 主从复制常遇到的错误

### 2.1 server\_id重复

解决办法：主从设置不同的server\_id

### 2.2 Slave\_IO\_Running的值是Connecting

解决办法，打通主从机器的网络，并保证从库能访问主库的3306端口

### 2.3 从库延迟一直增加，执行stop slave的时候，会出现等待

原因，可能从库磁盘空间满了，这个时候，删除一些不要的文件或者文件夹，释放出空间就行

### 2.4 主库要新增的内容，从库有了

解决办法：从库跳过这条记录

### 2.5 主库要更新的记录从库没有

关闭同步，找到原记录，在从库补数据，在开启同步

### 2.6 找不到主库Binlog位点

如果有其他正常的从库，可以把报错的从库接到这个从库，复制缺失的事务  
如果没有其他从库，那建议是重建复制。

## 3 多线程复制的演进是怎样的？

### 3.1 在 MySQL 5.6 之前

MySQL 只支持单线程复制

### 3.2 从 MySQL 5.6 版本开始，支持库级别的并行复制策略

如果表都集中在一个 DB 里，或者热点表集中在一个库中，那就没有什么效果了。

### 3.3 MySQL 5.7的并行复制

由参数：slave-parallel-type 控制并行复制策略。

配置为 DATABASE，表示使用 MySQL 5.6 版本的按库并行策略；

配置为 LOGICAL\_CLOCK，同时处于 prepare 状态的事务，在备库执行时是可以并行的；处于 prepare 状态的事务，与处于 commit 状态的事务之间，在备库执行时也是可以并行的。

### 3.4 MySQL 5.7.22的并行复制

MySQL 5.7.22 版本里，MySQL 增加了一个新的并行复制策略，基于 WRITESSET 的并行复制。

相应地，新增了一个参数 binlog-transaction-dependency-tracking，用来控制是否启用这个新策略。这个参数的可选值有以下三种。

COMMIT\_ORDER，表示的就是前面介绍的，根据同时进入 prepare 和 commit 来判断是否可以并行的策略。

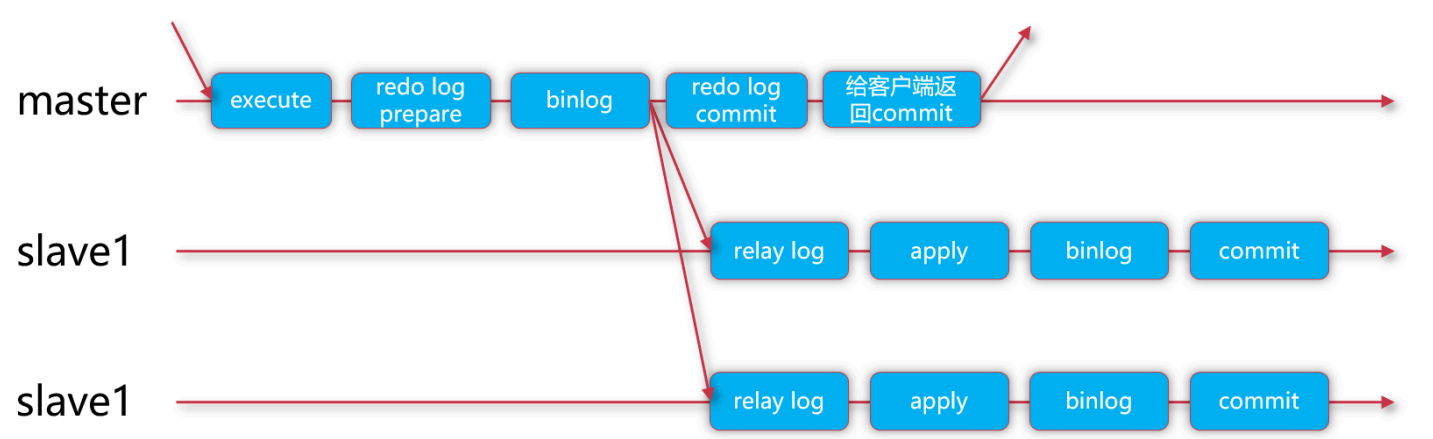
WRITESSET，表示的是对于事务涉及更新的每一行，计算出这一行的 hash 值，组成集合 writeset。如果两个事务没有操作相同的行，也就是说它们的 writeset 没有交集，就可以并行。

WRITESSET\_SESSION，是在 WRITESSET 的基础上多了一个约束，即在主库上同一个线程先后执行的两个事务，在备库执行的时候，要保证相同的先后顺序。

## 4 半同步复制的演进历程

### 4.1 异步复制

# 异步复制



异步复制的大致过程如下：

SQL解析：解析客户端发来的SQL；

存储引擎处理：存储引擎执行 SQL 操作；

写Redo Log（Prepare）阶段：事务的更改被准备好（prepare），但还没有提交到数据库；

写binlog：记录操作的Binlog；

传到从的中继日志：把 binlog 里面的内容传给从库的中继日志（relay log）中；

Redo Log提交：事务在InnoDB存储引擎中被提交。这时，更改正式应用到数据库文件中；

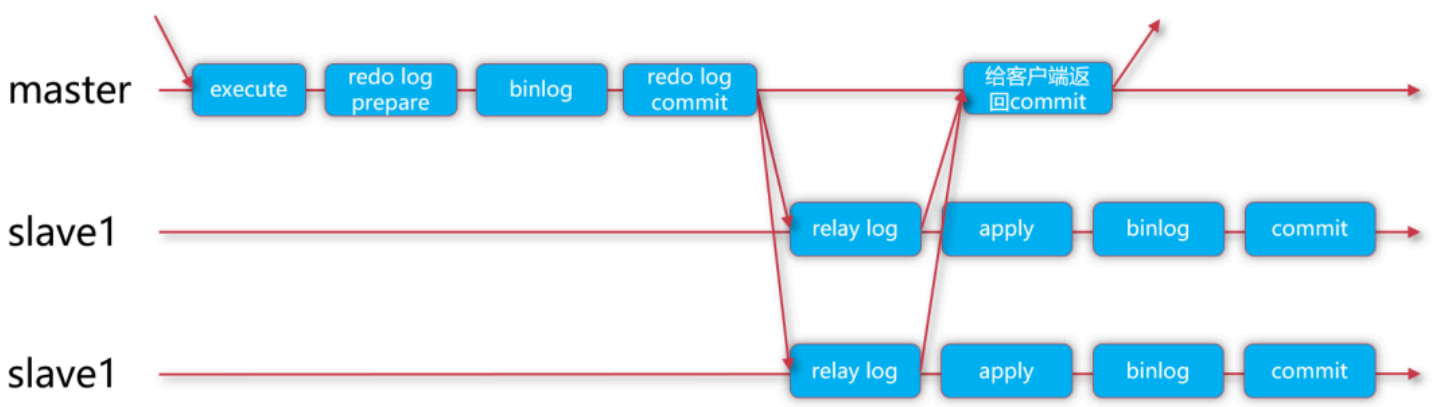
主库确认提交：主库在不等待从库确认的情况下向客户端返回commit成功；

从库应用更改：从库的 SQL 线程负责读取它的 relay log 里的信息并应用到从库数据库中。

在异步复制下，假如配置了自动切换的前提下，主库突然宕机，然后从提升为主时，原来主库上可能有一部分已经完成提交的数据还没来得及发送到从库，就可能产生数据丢失。为了解决这个问题，在 MySQL 5.5 版本中引入了半同步复制。下面来看下半同步复制的原理。

## 4.2 传统半同步复制after\_commit

# 半同步复制： after\_commit



半同步复制的大致过程如下：

SQL解析

存储引擎处理：执行SQL操作

写Redo Log（Prepare）阶段：事务的更改被准备好（prepare），但还没有提交到数据库；

写binlog：记录操作的Binlog；

把Binlog里的内容传给从库的中继日志；

Redo Log提交：事务在InnoDB存储引擎中被提交。这时，更改正式应用到数据库文件中；

从库发送 ACK：从库收到 binlog 后，发送给主库一个 ACK，表示收到了

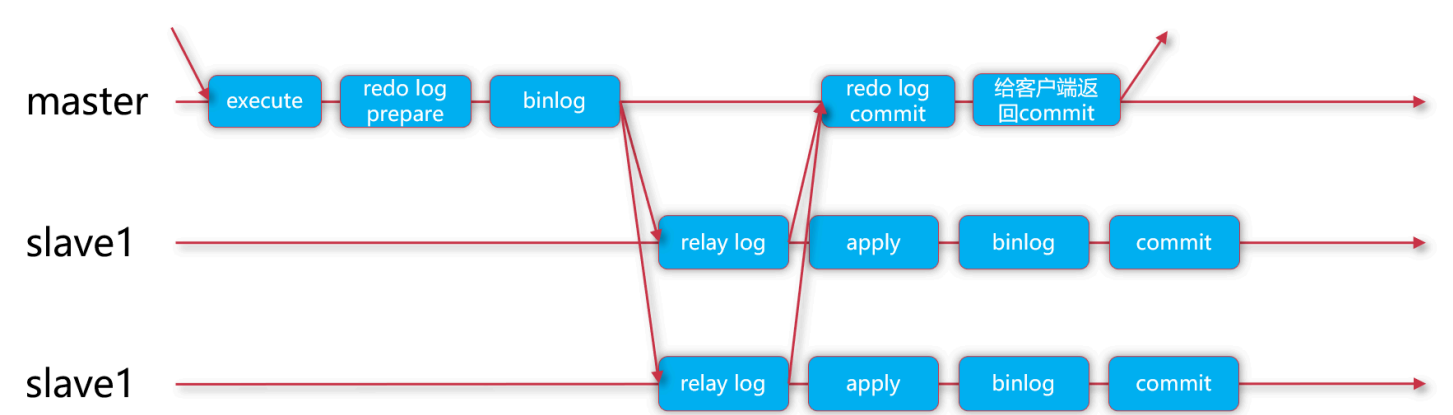
主库确认提交：主库收到这个 ACK 以后，才向客户端返回 commit 成功；

跟传统异步复制相比，半同步复制保证了所有客户端发送过确认提交的事务，从库都已经收到这个日志了。

但是这种模式下，实际上主库已经将该事务 commit 到事务引擎层，只是在等待返回而已，而此时其他 session 已经可以看到数据发生了变化，如果此时主库宕机，有可能从库还没写 Relay log，就会发生其他 session 切换前后查询的数据不一致的情况。

4.3 增强半同步复制after\_sync

增强半同步复制： after\_sync



增强半同步复制after\_sync的大致流程：

SQL解析

存储引擎处理：执行SQL操作，但不提交事务

写Redo Log（Prepare）阶段：事务的更改被准备好（prepare），但还没有提交到数据库；

写Binlog

把Binlog里的内容传给从库的中继日志

等待从库确认：等待从库成功接收binlog的返回ACK信号

Redo Log提交：在收到从库的 ACK 信号后，事务在InnoDB存储引擎中被提交。这时，更改正式应用到数据库文件中；

反馈至客户端：确认提交后向客户端返回成功信息

4.4 组复制

MySQL 5.7 推出了组复制（MySQL Group Replication，简称：MGR），是基于内置的主从复制的架构实现的，主要在事务提交的过程中，嵌入单独的 binlog 封装逻辑，并通过专门的复制通道进行数据传输，Group Replication 复制插件使用 Paxos 协议的原子广播特性，来保证在集群内的大多数节点都能接收到数据包，当数据节点接收到 write set 之后，每个节点上按照相同的规则对事务进行排序，并进行冲突检测。对于 master，当冲突检测通过之后，数据变更写入自身的 binlog 中，然后进行存储引擎层的提交（如果发现事务冲突，则进行事务回滚）；

对于 slave，当冲突检测通过之后，就把主库发来的 binlog 写入自身的 relay log 中，然后 sql 线程读取 relay log 进行重放，并把重放的 binlog 日志写入自身的 binlog 中，然后存储引擎内部进行提交（如果发现事务冲突，则丢弃主库发送过来的 binlog 日志）。

## 5 哪些情况可能会导致复制延迟？

主库增删改并发大

大表在做DDL

从库备份导致延迟

大事务

从库配置差

## 6 怎样判断复制是否有延迟？

### 6.1 Seconds\_Behind\_Master

一种常规的方法就是 show slave status 查看 Seconds\_Behind\_Master，这个参数表示从库延迟的秒数。

如果是0，表示可能没有延迟。这里为什么是可能呢？

当从库正在主动处理更新时，此字段显示从库上的当前时间戳与从库上当前正在处理的事件的主库上记录的原始时间戳之间的差异。

当副本上当前没有处理任何事件时，该值为 0

在某些情况下，Seconds\_Behind\_Master 并不一定准确。比如网络中断时，Seconds\_Behind\_Master = 0，并不能代表主从无延迟。

因此，有比这个更准确的一种方法：对比位点或 GTID。

### 6.2 对比位点

如果是基于位点的复制，则判断 Master\_Log\_File 跟 Relay\_Master\_Log\_File 是否相等，如果

Relay\_Master\_Log\_File 落后 Master\_Log\_File，则表示主从存在延迟。

其中

Master\_Log\_File 表示 IO 线程正在读取的主库 binlog 文件名

Relay\_Master\_Log\_File 表示SQL 线程最近执行的事务对应的主库 binlog 文件名

或者判断 Read\_Master\_Log\_Pos 跟 Exec\_Master\_Log\_Pos 是否相等，如果后者落后前者很多，则表示延迟比较高。

其中

Read\_Master\_Log\_Pos 表示IO 线程正在读取的主库 binlog 文件中的位点

Exec\_Master\_Log\_Pos 表示 SQL 线程最近读取和执行的事务对应的主库 binlog 文件中的位点

### 6.3 对比GTID

如果开启了 GTID 复制，则可以对比 Retrieved\_Gtid\_Set 和 Executed\_Gtid\_Set 是否相等，如果

Executed\_Gtid\_Set 落后很多，则表示存在延迟。

其中

Retrieved\_Gtid\_Set：从库收到的所有日志的 GTID 集合；

Executed\_Gtid\_Set：从库已经执行完的 GTID 集合。

## 7 MySQL从库延迟高，怎么确定是哪条SQL导致的？

### 7.1 从库show processlist

因为已经发现从库延迟了，假如是某个大事务导致的，那说明在主库已经提交了，正在从库执行，这个时候就可以找到具体的SQL语句；

### 7.2 主库慢查询日志

这种导致主从延迟的，很可能在主库就已经是慢查询了，当然有时候，可能因为一些DDL操作导致的延迟，要慢查询日志能打印DDL，需要开启log\_slow\_admin\_statements，可以记录管理语句（比如alter table语句、create index语

句等)

### 7.3 解析Binlog

如果前面两种方式还没找到，就解析最近的Binlog，看是否是因为主库增删改并发大

## 8 现在你发现从库延迟比较高了，有哪些解决方案？

降低延迟的处理办法：

### 8.1 开启多线程

开启多线程复制对比不开启多线程复制，延迟是底很多的。所以最推荐的解决延迟的方法就是开启多线程复制。

### 8.2 调整一些参数

innodb\_flush\_log\_at\_trx\_commit和sync\_binlog，都临时调整成不为1的值，从库延迟也能得到缓解。

### 8.3 调整从库机器配置

当然，有些公司为了节约成本，从库配置很差，就可以考虑提高配置。

### 8.4 避免大事务

比如一条delete删除上千万数据，那就拆分成小事务，每次删1000行，业务低峰操作2、如果是主库并发大，那就在从库开启多线程复制

### 8.5 使用PT工具执行耗时长的DDL

pt-online-schema-change，可以实现在线修改表结构，不锁表，比如mysql 5.6，DDL，就强烈建议使用，工具的具体用法，我们会在后面PT工具一章详细介绍。

### 8.6 调整架构

比如某张表比较大，延迟经常是这张表导致的，从库上读取数据时又用不上这张表，就可以考虑把大表单独创建一个从库进行复制。然后在原来的从库忽略这张表的复制，业务查询原来的从库就基本没延迟了。

## 9 使用GTID复制，限制有哪些？

不能混合使用事务和非事务引擎

8.0.21之前的版本，create table ... select ...这类语句不能执行，8.0.21开始是可以使用该语句的(create table...select)，因为在该版本中row格式的binlog也会记录为一个事务

同一个拓扑的所有MySQL，都需要开启GTID