MySQL读写分离：如何解决写完读不到问题

ImportNew 3月21日

以下文章来源于程序员历小冰 ，作者历小冰

程序员历小冰

历小冰的技术博客，专注于探讨后端生态的点点滴滴，内容包括微服务、分布式、数据库、...

（给ImportNew加星标，提高Java技能）

> 转自：历小冰 / 程序员历小冰

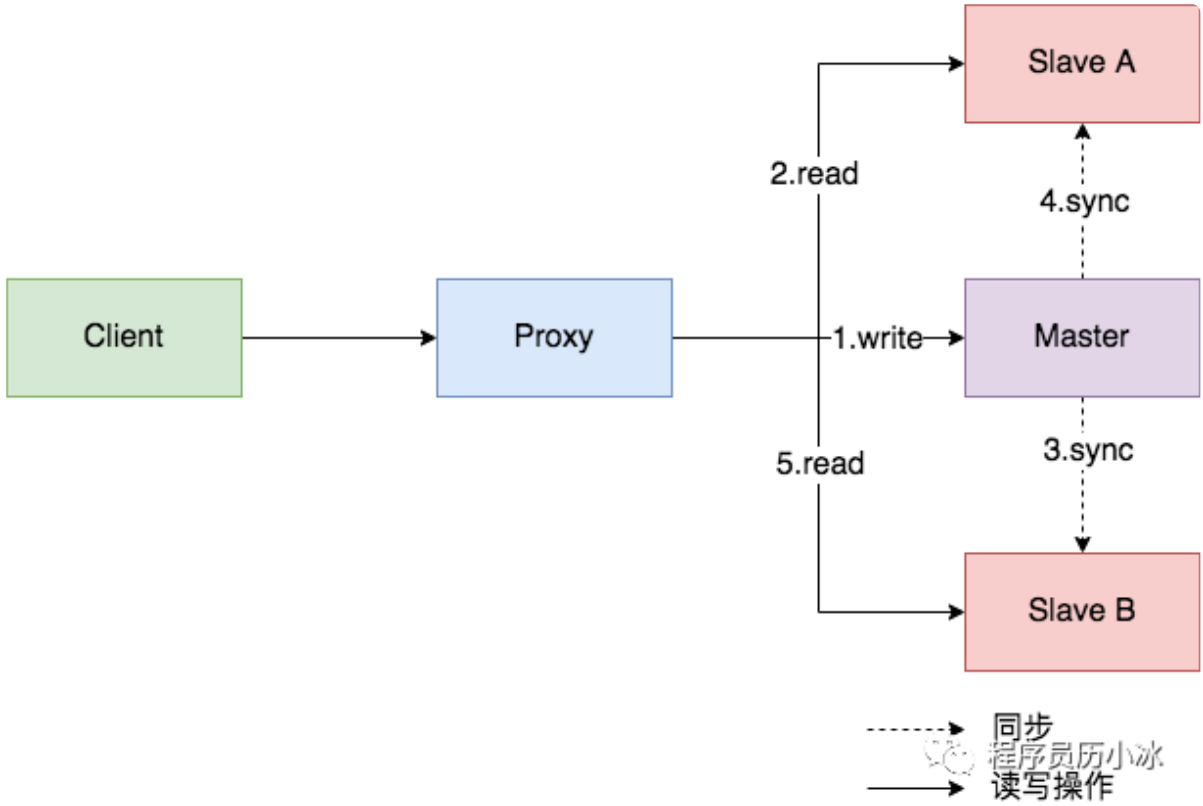
大家好，我是历小冰。

今天我们来详细了解一下主从同步延迟时读写分离发生写后读不到的问题，依次讲解问题出现的原因，解决策略以及 Sharding-jdbc、MyCat 和 MaxScale 等开源数据库中间件具体的实现方案。

写后读不到问题

MySQL 经典的**一主两从三节点架构**是大多数创业公司初期使用的主流数据存储方案之一。主节点处理写操作，两个从节点处理读操作，分摊了主库的压力。

但是，有时候可能会遇到执行完写操作后，立刻去读发现读不到或者读到旧状态的尴尬场景。这是由于主从同步可能存在延迟，在主节点执行完写操作，再去从节点执行读操作，读取了之前旧的状态。



上图展示了此类问题出现的操作顺序示意图：

- 客户端首先通过代理向主节点 Master 进行了写入操作；
- 紧接着第二步去从节点 Slave A 执行读操作，此时 Master 和 Slave A 之间的同步还未完成，所以第二步的读操作读取到了旧状态；
- 当第五步再次进行读操作时，此时同步已经完成，所以可以从 Slave B 中读取到正确的状态。

下面，我们就来看一下为什么会出现此类问题。

MySQL 主从同步

理解问题背后发生的原因，才能更好的解决问题。MySQL 主从复制的过程大致如下图所示，本篇文章只讲解同步过程中的流程，建立同步连接和失联重传不是重点，暂不讲解，感兴趣的同学可以自行了解。

MySQL 主从复制，涉及主从两个节点，一共四个线程参与其中：

- 主节点的 Client Thread，处理客户端请求的线程，执行如图所示的1~5步骤，2、3、4步是为了保证数据的一致性和尽量减少丢失，第三步骤时会通知 Dump Thread；
- 主节点的 Dump Thread 接收到 Client Thread 通知后，负责读取本地的 binlog 的数据，将 binlog 数据、binlog 文件名以及当前发送 binlog 的位置信息发送给从节点；
- 从节点的 IO Thread 负责接收 Dump Thread 发送的 binlog 数据和相关位置信息，将其追加到本地的 relay log 等文件中；
- 从节点的 SQL Thread 检测到 relay log 追加了新数据，则解析其内容（其实就是解析 binlog 文件的内容）为可以执行的 SQL 语句，然后在本地数据执行，并记录下当前执行的 relay log 位置。

上述是默认的**异步同步模式**，我们发现，从主节点提交成功到从节点同步完成，中间间隔了6、7、8、9、10多个步骤，涉及到一次网络传输，多次文件读取和写入的磁盘 IO 操作，以及最后的 SQL 执行的 CPU 操作。

所以，当主从节点间网络传输出现问题，或者从节点性能较低时，主从节点间的同步就会出现延迟，导致文章一开始提及的写后读不到的问题。在高并发场景，从节点一般要过几十毫秒，甚至几百毫秒才能读到最新的状态。

常见的解决策略

一般来讲，大致有如下方案解决写后读不出问题：

- 强制走主库
- 判断主备无延迟
- 等主库位点或 GTID 方案

强制走主库

强制走主库方案最容易理解和实现，它也是最常用的方案。顾名思义，它就是强制让部分必须要读到最新状态的读操作去主节点执行，这样就不会出现写后读不出问题。这种方案问题在于将一部分读压力给了主节点，部分破化了读写分离的目的，**降低了整个系统的扩展性。**

一般主流的数据库中间件都提供了强制走主库的机制，比如在 sharding-jdbc 中，可以使用 **Hint** 来强制路由主库。

它的原理就是在 SQL 语句前添加 Hint，然后数据库中间件会识别出 Hint，将其路由到主节点。

下面，我们就来看一下如果要去从库查询，并且要避免过期读的方案，并分析各个方案的优缺点。

判断主备无延迟

第二种方案是使用 show slave status 语句结果中的部分值来判断主从同步的延迟时间：

```
> show slave status
*************************** 1. row ***************************
Master_Log_File: mysql-bin.001822
Read_Master_Log_Pos: 290072815
Seconds_Behind_Master: 2923
```

```
Relay_Master_Log_File: mysql-bin.001821
Exec_Master_Log_Pos: 256529431
Auto_Position: 0
Retrieved_Gtid_Set:
Executed_Gtid_Set:
.....
```

- seconds_behind_master，表示落后主节点秒数，如果此值为0，则表示主从无延迟；
- Master_Log_File 和 Read_Master_Log_Pos，表示的是读到的主库的最新位点，Relay_Master_Log_File 和 Exec_Master_Log_Pos，表示的是备库执行的最新位点。如果这两组值相等，则表示主从无延迟；
- Auto_Position=1 ，表示使用了 GTID 协议，并且备库收到的所有日志的 GTID 集合 Retrieved_Gtid_Set 和 执行完成的 GTID 集合 Executed_Gtid_Set 相等，则表示主从无延迟。

在进行读操作前，先根据上述方式来判断主从是否有延迟。如果有延迟，则一直等待到无延迟后执行。但是这类方案在判断是否有延迟时存在着假阳和假阴的问题：

- 判断无延迟，其他延迟了。因为上述判断是基于从节点的状态，当主节点的 Dump Thread 尚未将最新状态发送给从节点的 IO SQL 时，从节点可能会错误的判断自己和主节点无延迟。
- 判断有延迟，但是读操作读取的最新状态已经同步。因为 MySQL 主从复制是一直在进行的，写后直接读的同时可能还有其他无关写操作，虽然主从有延迟，但是对于第一次写操作的同步已经完成，所以读操作已经可以读到最新的状态。

对于第一个问题，需要使用主从复制的 **semi-sync 模式**，上文中讲解介绍的是默认的异步模式，semi-sync 模式的流程如下图所示：

- 当主节点事务提交的时候，Dump Thread 把 binlog 发给从节点；
- 从节点的 IO Thread 收到 binlog 以后，发回给主节点一个 ack，表示收到了；
- 主节点的 Dump Thread **收到这个 ack 以后，再通知 Client Thread** ，此时才能给客户端返回执行成功的响应。

这样，写操作执行后就确保从节点已经读取到主节点发送的 binglog 数据，即 Master_Log_File、Read_Master_Log_Pos 或 Retrieved_Gtid_Set 是最新的。这样才能与执行的相关数据进行对比，判断是否有延迟。

可惜的是，上述 semi-sync 模式只需要等待**一个从节点**的ACK，所以一主多从的模式该方案将会无效。

虽然该方案有种种问题，但是对于一致性要求不那么高的场景也能适用，比如 MyCat 就是用 seconds_behind_master 是否落后主节点过多，如果超过一定阈值，就将其从有效从节点列表中删除，不再将读请求路由到它身上。

在 MyCAT 的用于监听从节点状态，发送心跳的 MySQLDetector 类中，它会读取从节点的 seconds_behind_master，如果其值大于配置的 slaveThreshold，则将打印日志，并将延迟时间设置到心跳信息中。

下面，我们就介绍能够解决第二个问题的方案，即判断有延迟，但是读操作读取的特定最新状态已经同步。

等 GTID 方案

首先介绍一下 GTID，也就是**全局事务 ID**，是一个事务在提交的时候生成的，是这个事务的唯一标识。它由MySQL 实例的 uuid 和一个整数组成，该整数由该实例维护，初始值是 1，每次该实例提交事务后都会加一。

MySQL 提供了一条基于 GTID 的命令，用于在从节点上执行，等待从库同步到了对应的 GTID（binlog文件中会包含 GTID）或者超时返回。

MySQL 在执行完事务后，会将该事务的 GTID 会给客户端，然后客户端可以使用该命令去要执行读操作的从库中执行，**等待该 GTID，等待成功后，再执行读操作。**如果等待超时，则去主库执行读操作，或者再换一个从库执行上述流程。

MariaDB 的 MaxScale 就是使用该方案、MaxScale 是 MariaDB 开发的一个数据库智能代理服务（也支持 MySQL），允许根据数据库 SQL 语句将请求转向目标一个到多个服务器，可设定各种复杂程度的转向规则。

MaxScale 在其 readwritesplit.hh 头文件和 rwsplit_causal_reads.cc 文件中的 add_prefix_wait_gtid 函数中使用了上述方案。

举个例子，原来要执行读操作的 SQL 和添加了前缀的 SQL 如下所示：

当 WAIT_FOR_EXECUTED_GTID_SET 执行失败后，原 SQL 就不会再执行，而是将该 SQL 去主节点执行。

后记

感觉大家一直读到文末，后续小冰会继续为大家奉上高质量的文章，也希望大家继续关注。

推荐阅读 — 点击标题可跳转

MySQL中，21个写SQL的好习惯

并发环境下，先操作数据库还是先操作缓存？

MySQL 优化上来就分库分表？面试官：先去补补底层原理吧

看完本文有收获？请转发分享给更多人

关注「ImportNew」，提升Java技能

好文章，我在看♡

喜欢此内容的人还喜欢

为什么数据库字段要使用NOT NULL?

ImportNew

MySQL 8.0的预研清单和计划

杨建荣的学习笔记

故障分析 | MySQL 优化案例 - select count(*)

yangyidba