27 主库出问题了，从库怎么办？

**一主多从切换正确性**

图1 一个基本的一主多从结构

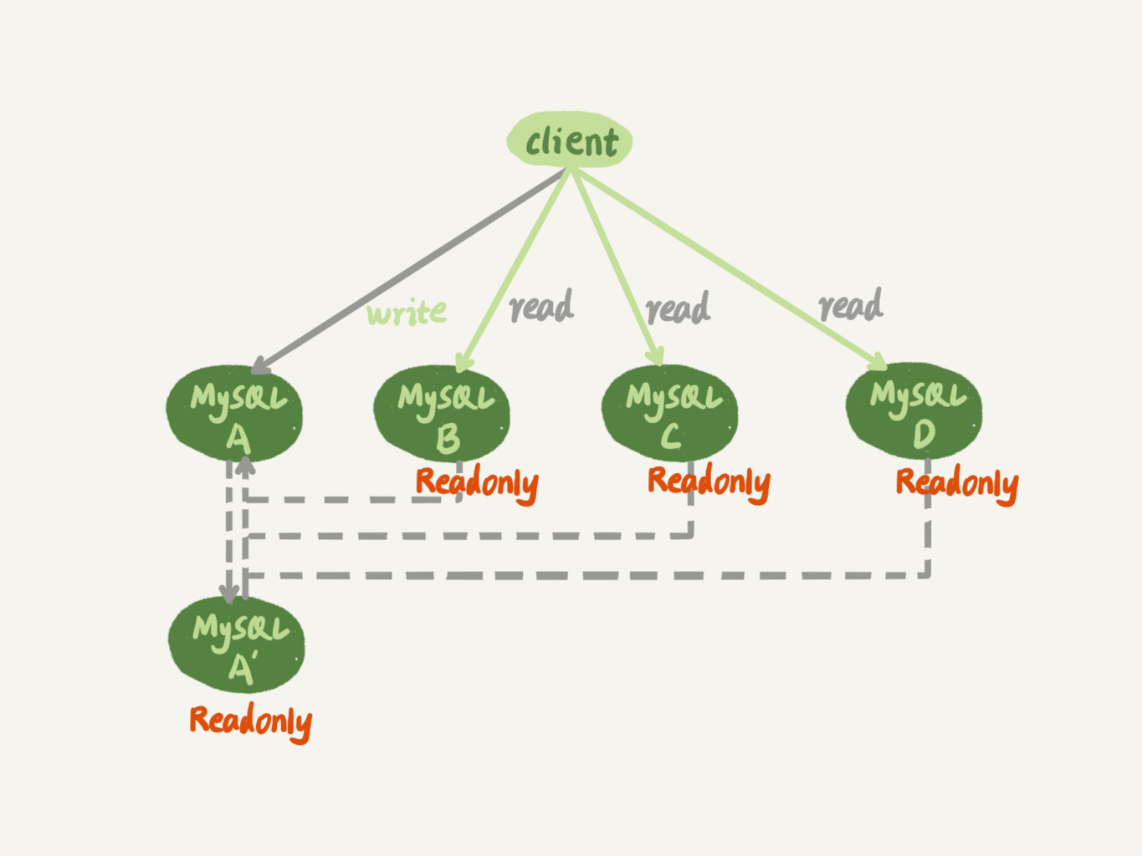


图 1 一主多从基本结构

图中，虚线箭头表示主备关系，也就是A和A`互为主备，从库B,C,D指向的是主库A,一主多从的设置，一般用于读写分离，主库负责所以的写和一部分读，而其他的请求则由从库分担。

图2 主库发生故障，主备切换的结果

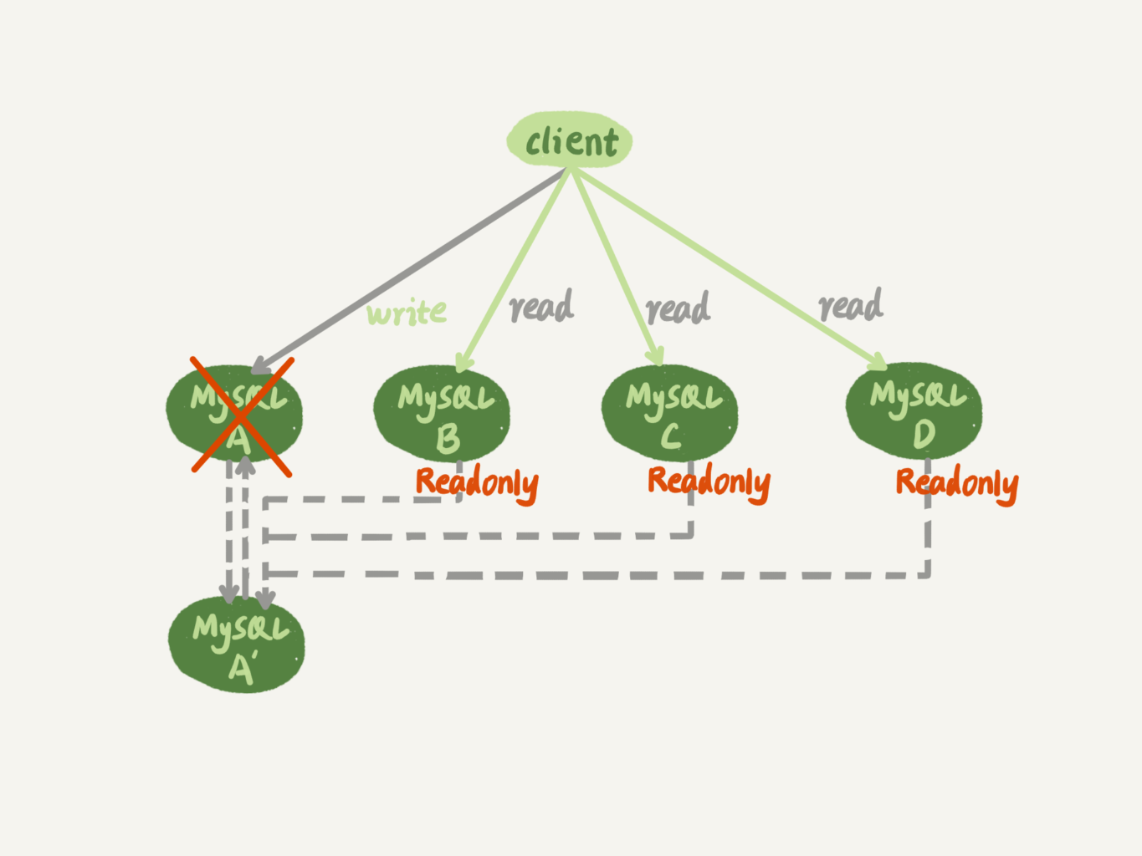


图 2 一主多从基本结构-- 主备切换

想比于一主一备的切换流程，一主多从结构的切换完成后，A`会成为新的主库，从库B,C,D也要改接的A`,正是由于多了从库B,C,D重新指向的这个问题，所以主备切换的复杂性也相应的增加了

**基于位点的主备切换**

把节点B设置为节点A`的从库的时候，需要执行一条change master命令

CHANGE MASTER TO

MASTER\_HOST=$host\_name

MASTER\_PORT=$port

MASTER\_USER=$user\_name

MASTER\_PASSWORD=$password

MASTER\_LOG\_FILE=$master\_log\_name

MASTER\_LOG\_POS=$master\_log\_pos

这条命令有6个参数

·MASTER\_HOST,MASTER\_PORT,MASTER\_USER和MASTER\_PASSWORD四个参数，分别代表了主库A`的IP，端口，用户名和密码

·最后两个参数MASTER\_LOG\_FILE和MASTER\_LOG\_POS表示，要从主库的master\_log\_name文件的master\_log\_pos这个位置的日志继续同步。而这个位置就是所说的同步位点，也就是主对应的文件名和日志偏移量

问题--节点B要设置成A`的从库，就要执行change master命令，就不可避免的要设置位点的这两个参数，但是这个两个参数到底怎么设置？

原来节点B是A的从库，本地记录的也是A的位点，但是相同的日志，A的位点 和A`的位点是不同的，因此，从库B要切换的时候，就需要先经过“找同步位点”这个逻辑。

这个位点很难精确，只能去一个大概位置，为什么这么说？

切换过程中不能丢失数据，所以找位点的时候，总要找一个“稍微往前”的，然后在通过判断跳过那些在从库B上已经执行过的事务。

一种取同步位点的方法是这样的

1. 等待新主库A`把中转日志（relay log） 全部完成
2. 在A`上执行show master status命令，得到当前A`上最新的File和Position
3. 取原主库A故障的时刻T
4. 用mysqlbinlog工具解析A`的Flie,得到T时刻的位置

mysqlbinlog File --stop-datetime=T --start-datetime=T

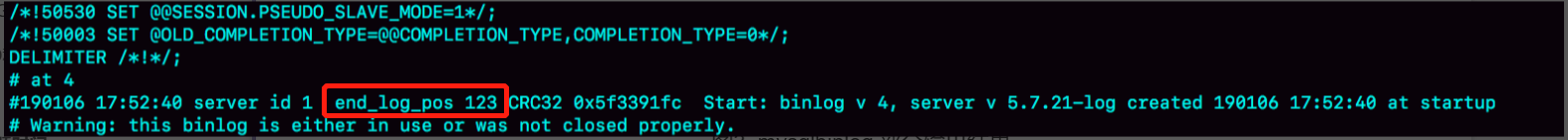


图 3 mysqlbinlog部分输出结果

图中，end\_log\_pos后面的值123,表示就是A`这个实例，在T时刻写入新的binlog位置，然后，可以把123这个值作为$master\_log\_pos，用在节点B的change master命令里。

这个值不精确，为什么？

假设在T时刻，主库A已经执行完一个insert语句插入了一行数据R,并且已经将binlog传给了A`和B,然后在传完的瞬间主库A的主机掉电。

那么，这个时候系统的状态是这样的

1. 在从库B上，由于同步了binlog,R这一行已经存在
2. 在新主库A`上，R这一行已经存在，日志是写在123这个位置之后的
3. 在从库上执行change master命令，指向A`的File文件的123位置，就会把R这一行数据的binlog又同步到从库B去执行。

此时，从库B的同步线程就会报告“Duplicate entry ‘id\_of \_R’ for key ‘PRIMARY’”错误，提示出现了主键冲突，然后停止同步

**通常情况下，切换任务的时候，要先主动跳过这些错误，有两种常用的方法，**

**一种做法，**主动跳过一个事务，命令写法

set global sql\_slave\_skip\_counter=1;

start slave;

切换过程中，可能会不止重复执行一个事务，需要在从库B刚开始接到新主库A`时，持续观察，每次碰到这些错误就停下来，执行一次跳过命令，直到不再出现停下来的情况，以此来跳过可能设计的所有事务。

**另一种方式是，**通过设置slave\_skip\_errors参数，直接设置跳过指定的错误

在执行主备切换时，有这么两类错误，是经常遇到的

1.1062错误是插入数据时唯一键冲突

2.1032错误是删除数据时找不到行

可以把slave\_skip\_errors设置为“1032，1062”，这样碰到这两种错误时就直接跳过

需要注意的是，这种直接跳过指定错误的方法，针对的是主备切换时，由于找不到精确的同步位点，所以只能采用这种方法来创建从库和新主库的主备关系。

前提是。清楚在主备切换过程中，直接跳过1032和1062这两类错误是无损的，所以才可以这么设置slave\_skip\_errors参数，等到主备间的同步关系建立完成，并稳定执行一段时间之后，再把这个参数设置为空，以免之后真的出现主从数据不一致，也跳过了。

**GTID**

通过sql\_slave\_skip\_counter跳过事务和通过slave\_skip\_errors忽略错误的方法，虽然都最终可建立从库B和新主库A`的主备关系，但这两中操作都很复杂，而且容易出错，所以MySQL5.6引入GTID,彻底结局了困难。

GTID是Global Transaction Identifier，也就是全局事务ID,是一个事务在提交的时候发生成的，是这个事务的唯一标识，由两部分组成

GTID=server\_uuid:gno

其中

Server\_uuid是一个实例第一次启动自动生成的，是一个全局唯一的值

gno是一个整数，初始值是1，每次提交事务的时候分配给这个事务，并加1

管方文档GTID定义

GTID=source\_id:transaction\_id

需要注意，source\_id就是server\_uuid,而后面的transaction\_id容易造成误导，改成了gno,

因为transaction\_id就是指事务id,事务id是在事务执行过程中分配的，如果这个事务回滚了，事务id也会递增，而gno是在事务提交的时候才分配的

从效果上看，GTID往往是连续的，因此用gno表示更容易理解

GTID模式的启动也很简单，只需要在启动一个MySQL实例的时候，加上参数

Gtid\_mode=on和enforce\_gtid\_consistency=on就可以了

在GTID模式下，每隔事务都会跟一个GTID一一对应，这个GTID有两种生成方式，而是用那种取决于session变量gtid\_next的值。

1.如果gtid\_next=automatic，代表使用默认值，这时，MySQL就会把server\_uuid:gno分配给这个事务

a.记录binlog的时候，先记录一行SET @@SESSION.GTID\_NEXT=’server\_uuid:gno’

b.把这个GTID加入本实例的GTID集合

2.如果gitd\_next是一个指定的GTID的值，比如通过set gtid\_next=’current\_gtid’指定为current\_id，有两种可能

a.如果current\_gtid已经存在于实例的GTID集合中，接下来执行的这个事务会直接被系统忽略

b.如果current\_gtid没有存在于实例的GTID集合中，就将这个current\_id分配给接下来要执行的事务，也就是说系统不需要给这个事务生成新的GTID,因此gno也不用加1

注意，一个current\_gtid只能给一个事务使用，这个事务提交后，如果执行下一个事务，就要执行set命令，把gtid\_next设置成另外一个gtid或者automatic

这样，每个MySQL实例都维护了一个GTID集合，用来对应“这个实例执行过的所有事务”

在实例X中创建一个表

CREATE TABLE `t` (

`id` int(11) NOT NULL,

`c` int(11) DEFAULT NULL,

PRIMARY KEY (`id`)

) ENGINE=InnoDB;

insert into t values(1,1);

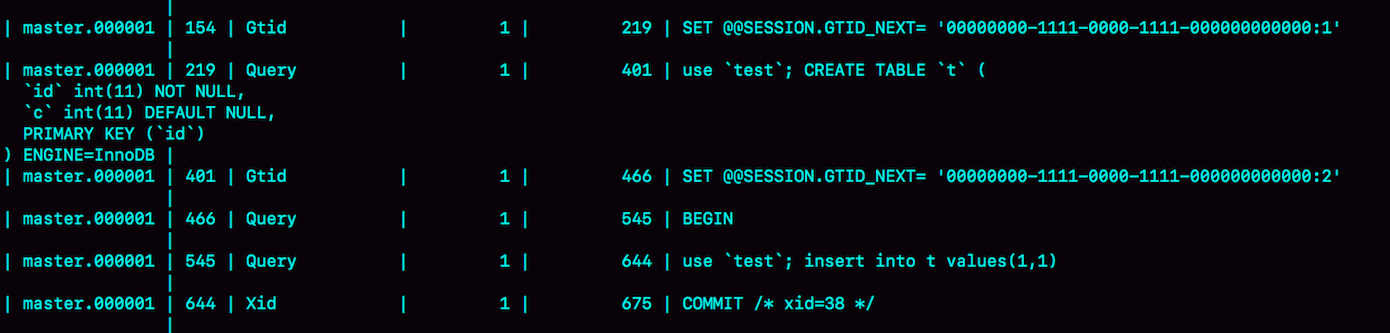


图 4 初始化数据binlog

事务DEGIN之前有同一条SET@@SESSION.GTID\_NEXT命令。这时，如果实例X有从库，那么将CREATE TABLE和insert语句的binlog同步过去执行的话，执行事务之前就会先执行这两个SET命令，这样被加入从库的GTID集合的，就是图中的两个GTID

假设，现在这个实例X是另外一个实例Y的从库，并且此时在实例Y上执行了下面插入语句

insert into t values(1,1);

并且，这条语句在实例Y上的GTID是“aaaaaaa-cccc-dddd-eeee-fffffffff:10”

那么实例X作为Y的从库，就要同步这是事务过来执行，显然会出现主键冲突，导致实例X的同步线程停止，此时该怎么处理？

set gtid\_next='aaaaaaaa-cccc-dddd-eeee-ffffffffffff:10';

begin;

commit;

set gtid\_next=automatic;

start slave;

其中，前三条语句的作用，是通过提交一个空事务，把这个GTID加到实例X的GTID集合中。

如图5所示，就是执行完这个空事务之后的show master status的结果

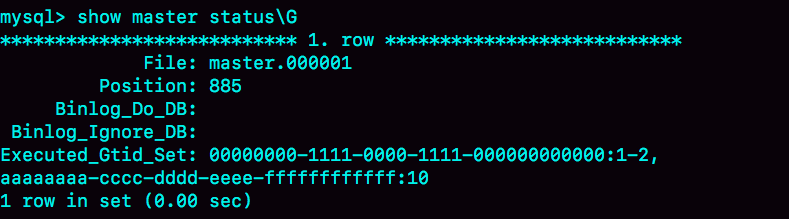


图 5 show master status的结果

实例X的Executed\_Gtid\_set里面，已经加入了这个GTID

在执行start slave命令让同步线程执行起来的时候，虽然实例X上还是会继续执行实例Y传过来的事务，但是由于“aaaaaaa-cccc-dddd-eeee-fffffffff:10”,已经存在于实例X的GTID集合中，所以实例X就会直接跳过这个事务，也就不会再出现主键冲突的错误

在上面的这个语句序列中，start slave命令之前还有一句set gtid\_next=automatic。这句话的作用是“恢复GTID的默认分配行为”，也就是说如果之后有新的事务再执行，就是还是按照原来的分配方式，继续分配gno=3.

**基于GTID的主备切换**

基于GTID的主备复制用法

在GTID模式下，备库B要设置为新主库A`的从库的语法如下

CHANGE MASTER TO

MASTER\_HOST=$host\_name

MASTER\_PORT=$port

MASTER\_USER=$user\_name

MASTER\_PASSWORD=$password

master\_auto\_position=1

Master\_auto\_position=1即表示这个主备关系使用的是GTID协议，MASTER\_LOG\_FILE和MASTER\_LOG\_POS参数，不需要指定了

实例A`的GTID集合记为set\_a,实例B的GTID集合记为set\_b，看下主备切换逻辑

在实例B上执行start slave命令

1. 实例B指定主库A`,基于主备协议建立连接
2. 实例B把set\_b发给主库A`
3. 实例A`算出set\_a和set\_b的差集，也就是所有存在于set\_a,但是不存在与set\_b的GTID的集合，判断A`本地是否包含了这个差集需要的所有binlog事务
4. 如果包含，表示A`已经把实例B需要的binlog给删除了，直接返回错误
5. 如果确认全部包含，A`从自己的binlog文件里面，找出第一个不在set\_b的事务，发给B
6. 之后就从这个事务开始，往后读文件，按顺序去binlog发给B去执行

其实，基于这个逻辑里面包含了一个设计思想，再基于GDIT的主备关系，系统认为只要建立主备关系，就必须保证主库发给备库的日志是完整的，因此，如果实例B需要的日志已经不存在A`就拒绝把日志发给B.

跟基于位点的主备协议不同，基于位点协议，是有备库决定的，备库指定那个位点，主库就发那个位点，不做日志的完整性判断。

引入GTID后，一主多从的切换场景下，主备切换是如何实现的

不需要找位点，从库B,C,D只需要分别执行change master命令指向实例A`即可

主备切换是不需要找位点的，而是找位点的工作，在实例A`内部就已经自动完成了，但是由于这个工作是自动的，所以对HA系统开发来说，非常友好

之后这个系统就有新主库A`写入，主库A`的自己生成binlog中的GDIT集合格式是

Server\_uuid\_of\_A`:1-M

如果之前从库B的GTID集合个格式是server\_uuid\_of\_a:1-N,那么切换后GTID集合的格式就变成了server\_uuid\_of\_A:1-N,server\_uuid\_of\_A`:1-M

主库A`之前也是A的备库，因此主库A`和从库B的GTID集合是一样的，达到了预期

GTID和在线DDL

22篇索引缺失引起的性能问题，可以通过在线加索引的来解决，考虑要避免新增索引对主库性能造成影响，可以现在备库加索引，然后切换

在双M结构下，备库执行DDL语句也会传给主库，为了避免传回后对主库造成影响，要通过set sql\_log\_bin=off关掉binlog

问题：这样操，数据路里面加了索引，但是binlog并有记录下这一个更新，是不是会导致数据和日志不一致？

假设，这两个互为主备关系的库还是实例X和实例Y,且当前主库是X,并且都打开了GTID模式，这时主备切换流程可以变成下面这样

·在实例X上执行stop slave

·在实例Y上执行DDL语句，注意，这里并不需要关闭binlog

·执行完成后，查出这个DDL语句对应的GTID,并记为server\_uuid\_of\_Y:gno

·到实例X上执行一下语句序列

set GTID\_NEXT="server\_uuid\_of\_Y:gno";

begin;

commit;

set gtid\_next=automatic;

start slave;

这样做目的在于，既可以让实例Y的更新有binlog记录，同时也可以确保不会在实例X上执行这条更新。

接下来，执行完主备切换，然后照着上述流程执行一遍即可

**思考题**

在GTID模式下设置主从关系的时候，从库执行start slave命令后，主库发现需要的binlog已经被删除，导致主备创建不成功，这种情况下，可以怎么处理？