# 复制基础

复制是指将主数据库的DDL和DML操作通过二进制日志传到复制服务器（也叫从库）上，然后在从库上对这些日志重新执行（也叫重做），从而使得从库和主库的数据保持同步。

MySQL支持一台主库同时向若干台从库进行复制，从库同时也可以作为其他服务器的主库，实现链式的复制。

MySQL复制的优点：

1. 如果主库出现问题，可以快速切换到从库提供服务；
2. 可以在从库上执行查询操作，降低主库的访问压力；
3. 可以在从库上执行备份，以避免备份期间影响主库的服务。

**注意：**

由于MySQL实现的是**异步的复制**，所以主从库之间存在一定的差距，在从库上进行的查询操作需要考虑到这些数据的差异，一般只有更新不频繁的数据或者对实时性要求不高的数据可以通过从库查询，实时性要求高的数据仍然需要从主数据库获得。

## 原理

MySQL复制原理大致如下：

1. 首先，MySQL主库在事务提交时会把数据变更作为事件Events记录在二进制日志文件binlog中；MySQL主库上的sync\_binlog参数控制binlog日志刷新到磁盘；
2. **主库推送二进制日志文件binlog中的事件到从库的中继日志Relay Log**，之后从库根据中继日志Relay Log重做数据变更操作，通过逻辑复制以此来达到主库和从库的数据一致。

MySQL通过3个线程来完成主从库间的数据复制：其中Binlog Dump线程跑在主库上，I/O线程和SQL线程跑在从库上。当在从库上启动复制（START SLAVE）时，首先创建I/O线程连接主库，主库随后创建Binlog Dump线程读取数据库事件并发送给I/O线程，I/O线程获取到事件数据后更新到从库的中继日志Relay Log中去，之后从库上的SQL线程读取中继日志Relay Log中更新的数据库事件并应用。

**从库有两个线程IO线程和SQL线程**

1. 从库的IO线程向主库的主进程发送请求，主库验证从库，交给主库IO线程负责数据传输；
2. 主库IO线程对比从库发送过来的master.info里的信息，将binlog文件信息，偏移量和binlog文件名等发送给从库；
3. 从库接收到信息后，将binlog信息保存到relay-bin中，同时更新master.info的偏移量和binlog文件名；
4. 从库的SQL线程不断的读取relay-bin的信息，同时将读到的偏移量和文件名写道relay-log.info文件，binlog信息写进自己的数据库，一次同步操作完成；
5. 完成上次同步后，从库IO线程不断的向主库IO线程要binlog信息；
6. 从库如果也要做主库，也要打开log\_bin 和log-slave-update参数。

### 配置文件

配置读写mysql主从复制的步骤：

1. 在主库与从库都安装mysql数据库
2. 在主库的配置文件(/etc/my.cnf)中配置server-id 和log-bin
3. 在登陆主库后创建认证用户并做授权。
4. 在从库的配置文件(/etc/my.cnf)中配置server-id
5. 登陆从库后，指定master并开启同步开关。

需要注意的是server-id主从库的配置是不一样的。

**server-id存在作用：**

mysql同步的数据中是包含server-id的，而server-id用于标识该语句最初是从哪个server写入的。因此server-id一定要有的。

Server-id不能相同的原因：每一个同步中的slave在master上都对应一个master线程，该线程就是通过slave的server-id来标识的；每个slave在master端最多有一个master线程，如果两个slave的server-id 相同，则后一个连接成功时，slave主动连接master之后，如果slave上面执行了slave stop；则连接断开，但是master上对应的线程并没有退出；当slave start之后，master不能再创建一个线程而保留原来的线程，那样同步就可能有问题；

在mysql做主主同步时，多个主需要构成一个环状，但是同步的时候有要保证一条数据不会陷入死循环，这里就是靠server-id来实现的。

### 日志文件

#### 二进制文件Binlog

二进制文件（Binlog）会把MySQL中的所有数据修改操作以二进制的形式记录到日志文件中，包括Create、Drop、Insert、Update、Delete等操作，但二进制日志文件（binlog）不会记录select操作，因为select操作并不修改数据。

可以通过show variables查看binlog格式，binlog支持statement、row、mixed三种格式，也对应了MySQL的三种复制技术。

二进制日志文件binlog的格式有以下3种：

Statement：基于SQL语句级别的binlog，每条修改数据的SQL都会保存在binlog中，存储日志量是最小的。

Row：基于行级别，记录每一行数据的变化，也就是将每行数据的变化都记录到binlog中，记录得非常仔细，但并不记录原始SQL，存储event数据，存储日志量大，但是不能很直接的进行读取；在复制的时候，并不会因为存储过程或触发器造成主从库数据不一致的问题，但是记录的日志量较statement格式要大得多。

Mixed：混合statement和row模式，默认情况下采用statement模式记录，某些情况下会切换到row模式。如果每天数据操作量很大，产生的日志比较多，可以考虑选择使用mixed格式。

注：Row格式比Statement格式更能保证从库数据的一致性（复制的是记录，而不是单纯操作SQL）。当然，Row格式下的Binlog的日志量很可能会增大非常多，在设置时需要考虑磁盘空间问题。

同时对应MySQL复制的3种技术：

binlog\_format=Statement：基于SQL语句的复制，也叫Statement-Based Replication（SBR），MySQL5.1.4之前仅提供基于SQL语句的复制。

binlog\_format=Row：基于行的复制，也叫Row-Based Replication（RBR）。

binlog\_format=Mixed：混合复制模式，混合了基于SQL语句的复制和基于行的复制。

#### 中继日志文件Relay Log

中继日志文件Relay Log的文件格式、内容和二进制日志文件binlog一样，唯一的区别在于从库上的SQL线程在执行完当前中继日志文件Relay Log中的事件之后，SQL线程会自动删除当前中继日志Relay Log，避免从库上的中继日志文件占用过多的磁盘空间。

为了保证从库Crash重启之后，从库的I/O线程和SQL线程仍然能够知道从哪里开始复制，从库上默认还会创建两个日志文件master.info和relay-log.info用来保存复制的进度。这两个文件在磁盘上以文件形式分别记录了从库的I/O线程当前读取二进制日志binlog的进度和SQL线程应用中继日志Relay Log的进度。

### 复制方式

#### 异步复制

MySQL的复制是异步复制。



#### 整体/部分复制

MySQL复制可是对整个实例进行复制，也可以对实例中的某个库或者是某个表进行复制。



#### 半同步复制

MySQL支持半同步复制。



在MySQL5.5之前，MySQL的复制是异步操作，主库和从库的数据之间存在一定的延迟，这样存在一个隐患：当在主库上写入一个事务并提交成功，而从库尚未得到主库推送的Binlog日志时，主库宕机了，例如主库磁盘损坏、内存故障灯造成主库上该事务binlog丢失，此时从库可能损失这个事务，从而造成数据不一致。

为了解决这个问题，MySQL5.5引入了半同步复制机制。在MySQL5.5之前的异步复制时，主库执行完commit提交操作后，在主库写入binlog日志后即可成功返回客户端，无需等待Binlog日志传送给从库。而半同步复制时，为了保证主库上的每一个binlog事务都能够被可靠的复制到从库上，主库在每次事务成功提交时，并不及时反馈给前端应用用户，而是等待其中一个从库页接收到binlog事务并成功写入到中继日志后，主库才返回commit操作成功给客户端。半同步复制保证了事务成功提交后，至少有两份日志记录，一份在主库的binlog日志上，一份在至少一个从库的中继日志Relay Log上，从而更进一步保证了数据的完整性。

半同步复制很大程度上取决于主从库之间的网络情况，往返时延RTT（Round-Trip Time）越小决定了从库的实时性越好。通俗地说，主从库之间网络越快，从库越实时。

从半同步复制的流程会发现，半同步复制的“半”就体现在：虽然主库和从库的Binlog日志是同步的，但是主库并不等待从库应用这部分日志就返回提交结果，这部分操作是异步的，从库的数据并不是和主库实时同步的，所以只能称为半同步，而不是完全的实时同步。

## 类型

存在两种复制类型：

### 基于binlog二进制日志的复制

### 使用GTID完成基于事务的复制

主从切换后，在传统方式里，需要找到binlog和POS点，然后执行change master to指向新的主库。对于不是很有经验的运维人员来说，往往会找错，造成主从同步复制报错，在MySQL5.6版本里，无须再找binlog和POS点，只需要知道master的IP、端口、账号和密码即可，因为同步复制是自动的，MySQL会通过内部机制GTID（Global Transaction ID）自动找点同步。

**GTID的工作原理：**

1、master更新数据时，会在事务前产生GTID，一同记录到binlog日志中。

2、slave端的i/o线程将变更的binlog，写入到本地的relay log中。

3、sql线程从relay log中获取GTID，然后对比slave端的binlog是否有记录。

4、如果有记录，说明该GTID的事务已经执行，slave会忽略。

5、如果没有记录，slave就会从relay log中执行该GTID的事务，并记录到binlog。

6、在解析过程中会判断是否有主键，如果没有就用二级索引，如果没有就用全部扫描。

**优点：**

1、一个事务对应一个唯一ID，一个GTID在一个服务器上只会执行一次

2、GTID是用来代替传统复制的方法，GTID复制与普通复制模式的最大不同就是不需要指定二进制文件名和位置

3、减少手工干预和降低服务故障时间，当主机挂了之后通过软件从众多的备机中提升一台备机为主机

**限制：**

1、不支持非事务引擎

2、不支持create table ... select 语句复制(主库直接报错)

原理：会生成两个sql，一个是DDL创建表SQL，一个是insert into插入数据的sql。由于DDL会导致自动提交，所以这个sql至少需要两个GTID，但是GTID模式下，只能给这个sql生成一个GTID。

3、不允许一个SQL同时更新一个事务引擎表和非事务引擎表

4、在一个复制组中，必须要求统一开启GTID或者是关闭GTID

5、开启GTID需要重启（5.7除外）

6、开启GTID后，就不再使用原来的传统复制方式

7、对于create temporary table 和 drop temporary table语句不支持

8、不支持sql\_slave\_skip\_counter

## 复制拓扑

复制存在多种拓扑结构：一主库多备库，主-主复制，环形复制，树或金字塔形等，3种常见的架构包括：一主多从复制架构、多级复制架构和双主复制/Dual Master架构。

### 一主多从复制架构

在主库读取请求压力非常大的场景下，可以通过配置一主多从复制架构实现读写分离，把大量对实时性要求不是特别高的读请求通过负载均衡分布到多个从库上，降低主库的读取压力。

在主库出现异常宕机的情况下，可以把一个从库切换为主库继续提供服务。

### 多级复制架构

一主多从的架构能够解决大部分读请求压力特别大的场景的需求，考虑到MySQL的复制是主库“推送”Binlog日志到从库，主库的I/O压力和网络压力会随着从库的增加而增长（每个从库都会在主库上有一个独立的Binlog Dump线程来发送事件），而多级复制架构解决了一主多从场景下，主库额外的I/O和网络压力。

对比一主多从的架构，多级复制架构仅仅是在主库master1复制到从库slave1、slave2、slave3的中间增加一个二级主句master2，主库只需要给一个从库master2“推送”binlog日志即可，减轻主库master1的压力。二级主库master2再“推送”binlog日志给从库slave1、slave2、slave3。

多级复制架构解决了一主多从场景下，主库的I/O负载和网络压力，其缺点：MySQL的复制是异步复制，多级复制场景下主库的数据是经历两次复制才到达从库的，期间的延时比一主多从复制场景下只经历一次复制的要大。

### 双主复制/Dual Master复制

双主/Dual Master架构特别适用于DBA做维护等需要主从切换的场景，通过双主/Dual Master架构避免了重复搭建从库的麻烦。

## 容量规划

## 复制管理

## 复制启动选项

## 日常运维

# 基于日志点的复制

在Master端建立复制用户

备份Master端的数据，并在Slave端恢复

使用Change master命名配置复制

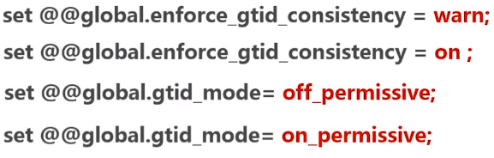
# 在线变更复制类型

在线将基于日志的复制变更为基于事务的复制

先决条件：集群中所有服务器的版本均高于5.7.6

集群中所有服务器gtid\_mode都设为off

处理步骤：





在线将基于事务的复制变更为基于日志的复制

先决条件：集群中所有服务器的版本均高于5.7.6

集群中所有服务器gtid\_mode都设为on

处理步骤：





# 多源复制

多源复制也就是多Master复制，允许一个Slave对应多个Master

如何使用多源复制：



# 多线程复制

