# 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。

### 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。

## 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

# 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

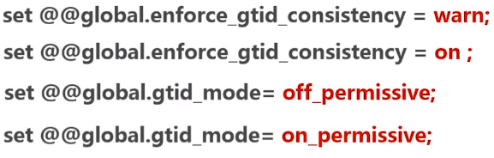
# 在线变更复制类型

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：





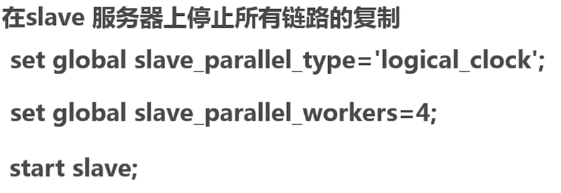
# 多源复制

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



# ----

# 主从复制

## 背景

为什么要做主从复制？

1、在业务复杂的系统中，有这么一个情景，有一句sql语句需要锁表，导致暂时不能使用读的服务，那么就很影响运行中的业务，使用主从复制，让主库负责写，从库负责读，这样，即使主库出现了锁表的情景，通过读从库也可以保证业务的正常运作。

2、做数据的热备

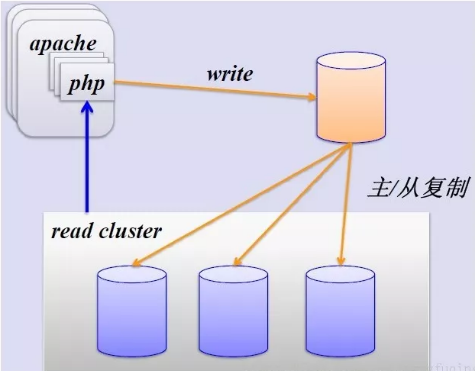
3、架构的扩展。业务量越来越大，I/O访问频率过高，单机无法满足，此时做多库的存储，降低磁盘I/O访问的频率，提高单个机器的I/O性能。

## 原理

mysql主从复制的原理是什么？

binlog: binary log，主库中保存所有更新事件日志的二进制文件。

主从复制过程：





主从复制的基础是主库记录数据库的所有变更记录到binlog。binlog是数据库服务器启动的那一刻起，保存所有修改数据库结构或内容的一个文件。

mysql主从复制是一个异步的复制过程，主库发送更新事件到从库，从库读取更新记录，并执行更新记录，使得从库的内容与主库保持一致。

在主库里，只要有更新事件出现，就会被依次地写入到binlog里面，之后会推到从库中作为从库进行复制的数据源。

**binlog输出线程**。每当有从库连接到主库的时候，主库都会创建一个线程然后发送binlog内容到从库。

对于每一个即将发送给从库的sql事件，binlog输出线程会将其锁住。一旦该事件被线程读取完之后，该锁会被释放，即使在该事件完全发送到从库的时候，该锁也会被释放。

在从库里，当复制开始的时候，从库就会创建两个线程进行处理：

**从库I/O线程**。当START SLAVE语句在从库开始执行之后，从库创建一个I/O线程，该线程连接到主库并请求主库发送binlog里面的更新记录到从库上。

从库I/O线程读取主库的binlog输出线程发送的更新并拷贝这些更新到本地文件，其中包括relay log文件。

**从库的SQL线程**。从库创建一个SQL线程，这个线程读取从库I/O线程写到relay log的更新事件并执行。

可以知道，对于每一个主从复制的连接，都有三个线程。拥有多个从库的主库为每一个连接到主库的从库创建一个binlog输出线程，每一个从库都有它自己的I/O线程和SQL线程。

从库通过创建两个独立的线程，使得在进行复制时，从库的读和写进行了分离。因此，即使负责执行的线程运行较慢，负责读取更新语句的线程并不会因此变得缓慢。比如说，如果从库有一段时间没运行了，当它在此启动的时候，尽管它的SQL线程执行比较慢，它的I/O线程可以快速地从主库里读取所有的binlog内容。这样一来，即使从库在SQL线程执行完所有读取到的语句前停止运行了，I/O线程也至少完全读取了所有的内容，并将其安全地备份在从库本地的relay log，随时准备在从库下一次启动的时候执行语句。

查看主从复制的状态

当主从复制正在进行中时，如果想查看从库两个线程运行状态，可以通过执行在从库里执行”show slave statusG”语句，以下的字段可以给你想要的信息：

Master\_Log\_File — 上一个从主库拷贝过来的binlog文件

Read\_Master\_Log\_Pos — 主库的binlog文件被拷贝到从库的relay log中的位置

Relay\_Master\_Log\_File — SQL线程当前处理中的relay log文件

Exec\_Master\_Log\_Pos — 当前binlog文件正在被执行的语句的位置

## 流程

整个主从复制的流程可以通过以下图示理解：



步骤一：主库db的更新事件(update、insert、delete)被写到binlog

步骤二：从库发起连接，连接到主库

步骤三：此时主库创建一个binlog dump thread，把binlog的内容发送到从库

步骤四：从库启动之后，创建一个I/O线程，读取主库传过来的binlog内容并写入到relay log

步骤五：还会创建一个SQL线程，从relay log里面读取内容，从`Exec\_Master\_Log\_Pos`位置开始执行读取到的更新事件，将更新内容写入到slave的db

注：上面的解释是解释每一步做了什么，整个mysql主从复制是异步的，不是按照上面的步骤执行的。

## 问题

主从复制也带来其他一系列性能瓶颈问题：

1、写入无法扩展

2、写入无法缓存

3、复制延时

4、锁表率上升

5、表变大，缓存率下降

针对上述问题，引入数据库垂直分区和水平分区解决。