1. **备份方式**
2. **备份方式分类**
3. **热备**

online backup，数据正在运行

1. **冷备份**

offline backup，数据库停止，直接复制物理文件

1. **温备份**

数据库正在运行，但是对数据库操作有影响，如加入一个全局读锁以保障备份数据的一致性。

1. **按照备份的内容分类：**
2. **逻辑备份**

mysqldump和select \* into outfile，内容生成sql。优点是可以观察导出文件内容，缺点是replay时间较长；

1. **文件备份**

备份物理文件，既可以用于热备（ibbackup和xtrabackip），又可以用于冷备份，恢复时间较逻辑备份时间短。

**XtraBackup备份原理：直接拷贝数据库文件和redo log，并且记录下当前二进制日志中已经提交的最后一个事务标记。在新的数据库实例上完成 recovery 操作。Binlog日志的replication是point in time的恢复而不是增量备份。而XtraBackup支持innodb存储引擎的增量备份：**

1. 首先完成一个全量备份，记录下此时检查点的LSN；
2. 在进行增量备份，比较表空间中每页的LSN是否大于上次备份的LSN，如果是则备份该页，并记录当前监测点的LSN。
3. **快照备份**

Mysql本身不支持快照功能，我们可以使用linux的 LVM来实现快照，即对磁盘分区的一种管理机制。

1. **按照备份数据库的内容分类：**
2. **完全备份**

拷贝一个数据库进行完整备份。

1. **增量备份**

在上次完全备份的基础上，对更改的数据进行备份。

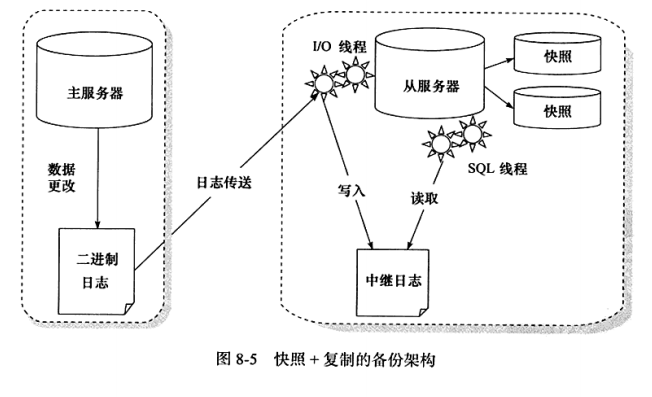
1. **日志备份**

主要是对binlog的备份，通过binlog的重做replay来完成数据库Point-in-time的恢复工作。

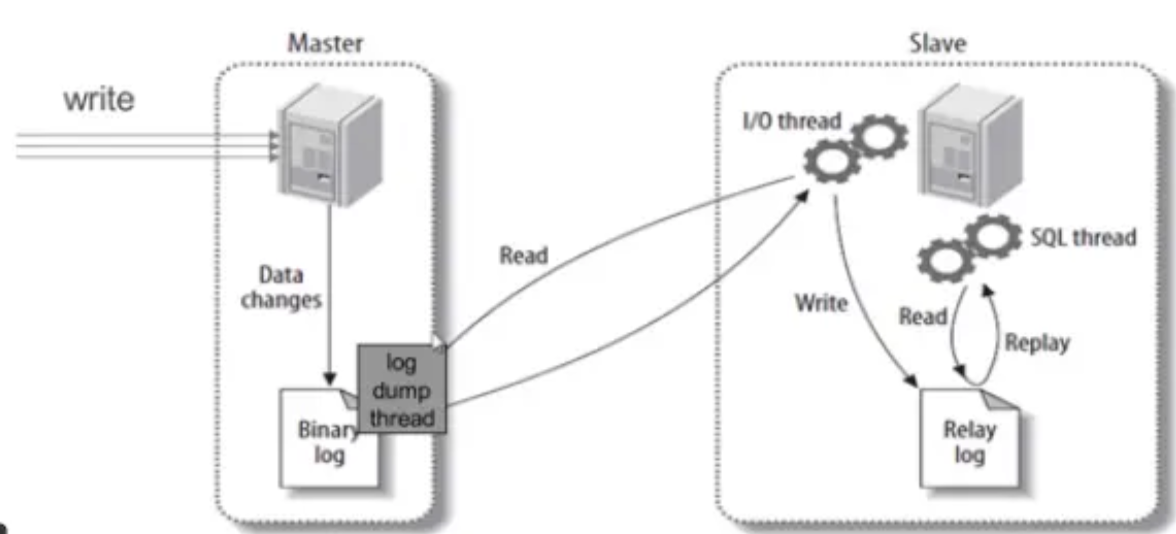
Mysql的主从复制就是从库根据自己的能力向主库请求binlog，主库异步实时响应从库请求将Binlog传输给并从库，从库执行relay日志回放到自己的数据库中。

通过show master status和show slave status可以查看主从同步的延迟。

如果执行了drop table等异常操作，如何恢复数据呢？一个比较好的办法就是在数据库分区做快照，当执行误操作时然后从服务器快照恢复，然后再通过binlog进行Point-in-time的恢复即可。



1. **主-备复制**
2. **master/slave**



首先从库通过手工执行change master to 语句连接主库，提供了连接的用户一切条件（user、password、port、ip），并且让从库知道，二进制日志的起点位置（file名 position号）。然后 start slave。

1. **主从复制过程**

Step1: 从库的IO和主库的dump线程建立连接；

Step2: 从库根据change master to语句提供的file名称和position，其IO线程向主库dump线程发起Binlog请求；

Step3: 主库dump线程根据从库请求，将本地binlog以events方式发给从库Io线程；

从库IO线程接收binlog events，存放在本地relay-log中，传送过来的信息记录到master.info中；

Step4: 从库SQL线程应用relay-log，并把应用后的记录到relay-log.info中。被应用过的relay-log会被purge。

Step5: 到此位置，刚建立主从关系的主从复制就完成了，**之后从库IO线程只需要监听主库dump线程的Binlog event即可，不需要再发送request。**

Step6: 从库IO线程接收relay log后需要给主库dump回复ack，默认情况下主库不会在意从库发不发送ack。只有semi sync时才会关心ack情况。

Step7: 从库IO线程长时间没收到binlog event，则可能考虑重连，从上面Step1开始。

1. **从库落后主库的原因**

从库记录的已经主库已经给我传送的binlog事件的坐标，一般在繁忙的生产环境下会落后于主库。

* 硬件条件有关的，机器磁盘IO性能不足。
* 主要还是网络问题，网络传输的性能。
* 主库存放二进制日志的存储性能太低，建议binlog日志存咋SSD中。
* 主库DUMP线程太繁忙，主要发生在一主多从的环境下。
* Sql线程繁忙；
* 从库IO线程太忙
* 人为控制（delay节点、延时节点 ）

1. **Parallel replication**

之前从库应用relay log的sql线程是单线程处理，处理速度慢。

Mysql 5.7由于实现了三阶段组提交。显然我们可以对于不同schema使用多线程并发应用relaylog。更进一步，由于行锁的限制，只有相互不影响的数据才能执行组提交，显然我们可以对同一个组提交的relay log使用并行应用。

5.7引入了新的变量slave-parallel-type，其可以配置的值有：

* DATABASE：默认值，基于库的并行复制方式
* LOGICAL\_CLOCK：基于组提交的并行复制方式

1. **semi sync**

实际监听binlog不保证主机宕机后从机的日志完备性，这是Mysql主从集群最为被诟病的地方。MySQL 5.7。 MySQL semi-sync是以插件方式引入，会保证在事务提交的时候至少有1台从机接收到了主机的binlog event，这样在宕机时就可以让那个日志最全的从机成为主机。

主要源码如下[3]：

*plugin/semisync/semisync\_master.cc*

*403 /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\**

*404 \**

*405 \* <ReplSemiSyncMaster> class: the basic code layer for sync-replication master.*

*406 \* <ReplSemiSyncSlave> class: the basic code layer for sync-replication slave.*

*407 \**

*408 \* The most important functions during semi-syn replication listed:*

*409 \**

*410 \* Master:*

*//实际由Ack\_receiver线程调用，处理semi-sync复制状态，获取备库最新binlog位点，唤醒对应线程*

*411 \* . reportReplyBinlog(): called by the binlog dump thread when it receives*

*412 \* the slave's status information.*

*//根据semi-sync运行状态设置数据包头semi-sync标记*

*413 \* . updateSyncHeader(): based on transaction waiting information, decide*

*414 \* whether to request the slave to reply.*

*//存储当前binlog 文件名和偏移量，更新当前最大的事务 binlog 位置*

*415 \* . writeTranxInBinlog(): called by the transaction thread when it finishes*

*416 \* writing all transaction events in binlog.*

*//实现客户端同步等待逻辑*

*417 \* . commitTrx(): transaction thread wait for the slave reply.*

*418 \**

*419 \* Slave:*

*//确认网络包头是否有semi-sync标记*

*420 \* . slaveReadSyncHeader(): read the semi-sync header from the master, get the*

*421 \* sync status and get the payload for events.*

*//给Master发送ACK报文*

*422 \* . slaveReply(): reply to the master about the replication progress.*

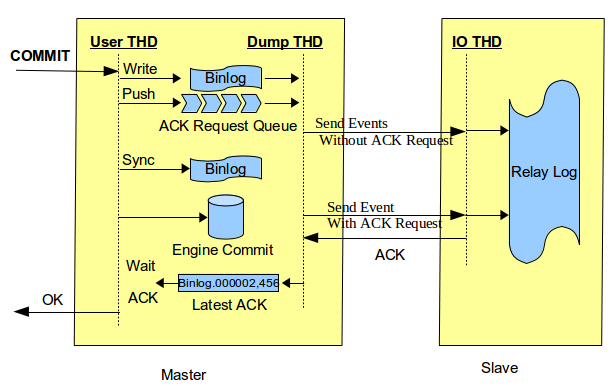
*423 \**

*424 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/*

下面通过rpl\_semi\_sync\_master\_wait\_point、sync\_binlog、sync\_relay\_log的配置来对semi-sync做数据一致性的分析。

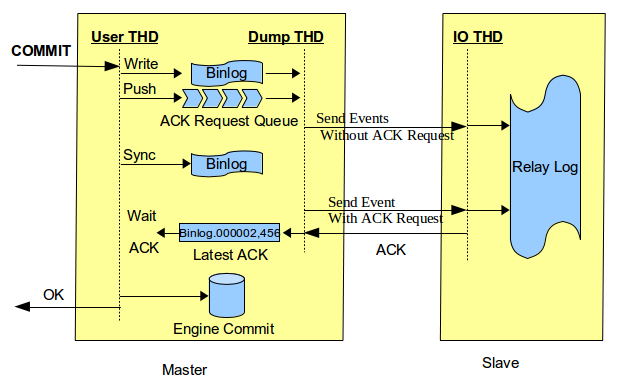
1. **主机先提交再发送等待**

如果半同步主机等待节点rpl\_semi\_sync\_master\_wait\_point是WAIT\_AFTER\_COMMIT，也就是在存储引擎commit后才发生binlog给从机，如果binlog event在发送过程中主机宕机，从机显然没有监听到该binlog日志，而主机已提交事务，显然导致从机丢事务。

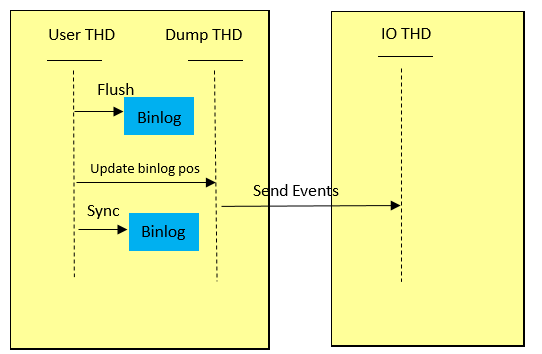


1. **主机先发送等待再提交**

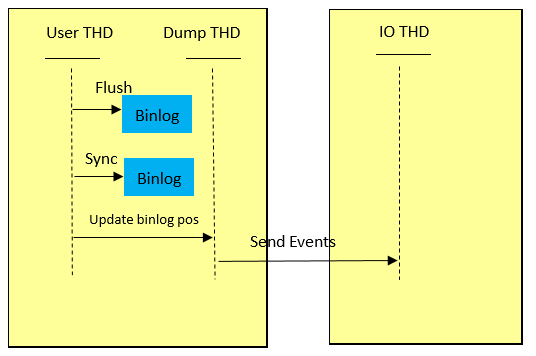
如果半同步主机等待节点rpl\_semi\_sync\_master\_wait\_point是WAIT\_AFTER\_SYNC，此时binlog选择的发送时机在组提交flush之后和commit之前，也就是说当前事务未提交。



如果从机收到binlog，而主机binlog在sync时没有fsync刷盘（sync\_ binlog参数不是1），此时主机事务未提交，从机反而收到了该事务的binlog导致不一致。



如果sync\_binlog设置为1，会强制保证完成binlog落盘，且sync binlog和send events是同一个线程顺序执行，效率降低。但是可以保证主机和从机binlog的写状态一致。在发送events给从机后即使主机宕机事务没有提交，但是由于binlog已经落盘在主机恢复后该事务会从Binlog恢复后提交。



1. **从机sync\_relay\_log**

同样，sync\_relay\_log也会影响主从事务的一致性，它代表从机应用relay\_log的时机。sync\_relay\_log=1时，从机在应用主机的events后才会回复ack。

sync\_relay\_log不是1，从机会立即回复主机ack，之后才会应用relay\_log。显然如果此时从机宕机，主机其实已经接受到从机的ack认为从机应用了日志，但是从机却丢失了events事务。

所以，为了保证较强的一致性，我们需要设置rpl\_semi\_sync\_master\_wait\_point=wait\_after\_syn，sync\_binlog=1.sync\_relay\_log=1。

1. **group replication——paxos**

MySQL Group Replication（简称MGR）是MySQL官方于2016年12月推出的一个全新的高可用与高扩展的解决方案。MySQL组复制提供了高可用、高扩展、高可靠的MySQL集群服务。高一致性，基于原生复制及paxos协议的组复制技术，并以插件的方式提供，提供一致数据安全保证。

**参考**

1. Mysql 5.7文档 <http://rsc.anu.edu.au/~rsccu/manuals/mySQL/refman-5.0-en.html-chapter/replication.html>
2. MySQL 5.7 Parallel replication <https://www.cnblogs.com/xiaotengyi/p/5532191.html>
3. MySQL 半同步复制数据一致性分析<http://mysql.taobao.org/monthly/2017/04/01/>
4. <https://www.jianshu.com/p/328ad87bde5e>
5. Mysql 主从复制 <https://www.cnblogs.com/clsn/p/8150036.html>
6. Mysql group replication <https://www.zhihu.com/question/53617036>