复制

复制解决的基本问题：让一台服务器的数据与其他服务器保持同步。

一台主库数据可以同步到多台备库上，备库本身也可以配置成另外一台服务器的主库。

通过复制，可以将读操作指向备库来获得更好的读扩展。但是对于写操作，并不适合通过复制来实现扩展。

同时，一主库多备库的设计，可能会造成一些资源的浪费（复制大量重复的数据）

MySQL 的复制是异步、串行的。

复制方式：

1. 基于行的复制（5.1后才有）
2. 基于语句的复制

都是通过在主库上记录二进制日志、在备库中重放日志的方式来实现异步的数据复制。意味着在同一时间点上备库上的数据可能与主库存在不一致。

复制通常不会增加主库的开销，主要的开销是启用二进制日志带来的。

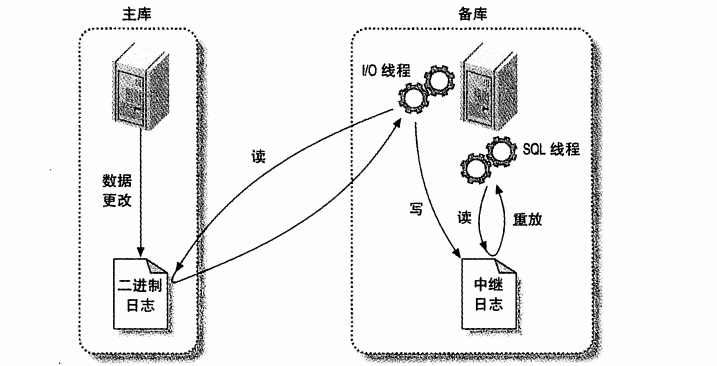
此外，每个备库向主库请求读取二进制日志时，会增加一些主库的负载，造成IO开销。

复制常见的用途：

1. 数据分布备份：可以将数据复制到不同的位置来分布数据备份。
2. 负载均衡：将多个读操作分布到多个服务器上，实现基本的负载均衡
3. 高可用性和故障切换：帮助避免单点故障，减少一个服务器发生故障时，服务器的切换时间。

复制三个步骤：

1. 在主库上把数据更改记录到二进制日志（Binary Log，二进制日志事件）中：在每次准备提交事务完成数据更新前，主库将数据更新的事件记录到二进制日志中。MySQL 按事务提交的顺序，而非每条语句执行顺序来记录。记录之后，主库通知存储引擎可以提交事务了。
2. 备库将主库上的日志复制到字节的中继日志（Relay Log）中：首先，备库启动一个工作现场（I/O线程），IO线程和主库建立一个普通的客户端连接，然后在主库上启动一个特殊的二进制转储线程，这个线程会读取主库中二进制日志中的事件。如果这个线程追上了主库，它会进入睡眠状态，直到有新的事件产生时被唤醒。备库的IO线程会接收事件，并记录到中继日志中。
3. 备库读取中继日志中的事件，将它们重放到备库数据中：备库的SQL线程从中继日志中读取事件，并在备库中执行，实现数据的更新。如果SQL线程赶上IO线程，中继日志通常已经在系统缓存中，所以中继日志的开销很低。



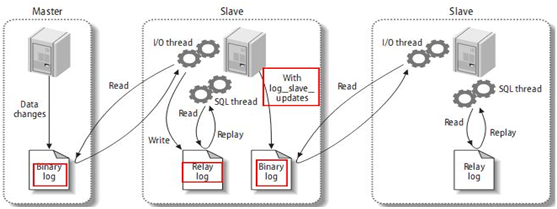
备库有两个运行的线程，主库有一个运行的线程。由备库发起连接。

IO线程独立于SQL线程，允许两个过程异步执行。

限制：主库中并发执行的SQL查询，在备库中重放的时候只能串行执行，因为只有一个SQL线程来重放中继日志中的事件。

**主从多级复制**

读操作很多可以采用单一maste和多slave，但增大到一定slave后连到master的slaveIO线程太多会造成master压力增大，从而造成数据复制延时。多级复制就是为了解决这个问题。如果想实现主-从（主）-从多级复制，需要设置log-slave-updates参数。同时二进制日志也必须启用。



当然，增加复制的级联层次，同一个变更传到最底层的Slave所需要经过的MySQL也会更多，同样可能造成延时较长的风险。如果条件允许，倾向于通过拆分成多个Replication集群来解决。

基于语句的复制

主库记录更新数据的事件，备库重放这些事件时，只是把主库上执行过的SQL再执行一遍。

优点：

1. 实现简单，容易理解；
2. 二进制日志中的事件紧凑，在复制时不会使用很多的带宽。

缺点：

1. 其他因素：比如元数据信息，当前时间戳，无法被正确复制；使用触发器或者存储过程也会有很多问题。

基于行的复制

将实际变化的数据记录在二进制日志中。

优点：

1. 可以正确地复制每一行；可以知道哪些数据发生了什么更改。

缺点：

1. 很难进行时间点的恢复；除了变化的行以外，不知道执行了哪些SQL。

表级锁

[**MySQL**](http://lib.csdn.net/base/mysql)表级锁分为读锁和写锁。

读锁

用法：LOCK TABLE table\_name [ AS alias\_name ] READ

UNLOCK tables。

成功申请读锁的前提是当前没有线程对该表使用写锁，否则该语句会被阻塞。

申请读锁成功后，其他线程也可以对该表进行读操作，但不允许有线程对其进行写操作，就算是当前线程也不允许。当锁住了A表之后，就只能对A表进行读操作，对其他表进行读操作会出现错误。

写锁

用法： LOCK TABLE table\_name [AS alias\_name] [ LOW\_PRIORITY ] WRITE

申请写锁之后，其他线程不允许读。

与读锁不同的是，写锁中可以指定锁的优先级。LOW\_PRIORITY是一种比读锁更低优先级的锁，当多个线程同时申请多种锁（LOW\_PRIORITY，READ，WRITE）时，LOW\_PRIORITY的优先级最低。

行级锁

仅对指定的记录进行加锁。其他线程还可以对同一张表中的其他记录进行操作。

行级锁分为共享锁和排他锁。

共享锁(S LOCK)

用法：SELECT ...LOCK IN SHARE MODE;

Mysql会对查询结果中的每行都加共享锁，当没有其他线程对查询结果集中的任何一行使用排他锁时，可以成功申请共享锁，否则会被阻塞。其他线程也可以读取使用了共享锁的表，而且这些线程读取的是同一个版本的数据。

排他锁(X LOCK)

用法：SELECT ...LOCK FOR UPDATE;

Mysql会对查询结果中的每行都加排他锁，当没有其他线程对查询结果集中的任何一行使用排他锁时，可以成功申请排他锁，否则会被阻塞。

行级锁都是基于索引的，如果一条SQL语句用不到索引是不会使用行级锁的，会使用表级锁。

表级锁在MyISAM和innoDB中都有用到。

* 创建锁的开销小，加锁快；
* 不会出现死锁；
* 由于锁定的是整张表，发生锁冲突的概率最高，所以并发度低。

当需要频繁对大部分数据做 GROUP BY 操作或者需要频繁扫描整个表时，推荐使用表级锁。

行级锁在innoDB用到。

* 创建锁的开销大，加锁慢；
* 会出现死锁；
* 发送锁冲突的概率最低，并发度最高。

InnoDB行锁是通过给索引上的索引项加锁来实现的，意味着：只有通过索引条件检索数据，InnoDB才使用行级锁，否则，InnoDB将使用表锁！

如果一条语句操作了主键索引，MySQL就会锁定这条主键索引；

如果一条语句操作了非主键索引，MySQL会先锁定该非主键索引，再锁定相关的主键索引。

因此，当两个事务同时执行，一个锁住了主键索引，在等待其他相关索引；另一个锁定了非主键索引，在等待主键索引。这样就会发生死锁。

发生死锁后，InnoDB一般都可以检测到，并使一个事务释放锁回退，另一个获取锁完成事务。

应用场景：

表级锁：更于以查询为主，只有少量按索引条件更新数据的应用，如WEB应用；

行级锁：更适合于有大量按索引条件更新少量不同数据，同时又有并发查询的应用，如一些在线事务处理系统。

**MyISAM与InnoDB的区别**

**1、 存储空间**

MyISAM：索引和数据分开，并且索引可被压缩，存储空间较小。  
InnoDB：索引和数据绑定在一起，而且没有压缩。

**2、 可移植性、备份及恢复**

MyISAM：数据是以文件的形式存储，所以在跨平台的数据转移中会很方便。在备份和恢复时可单独针对某个表进行操作。而且因为可以压缩，备份和恢复很快。  
InnoDB：拷贝数据文件、备份 binlog，或者用 mysqldump，在数据量比较大的时候很慢。

**3、 事务支持**

MyISAM：强调的是性能，每次查询具有原子性，其执行速度更快，不提供事务支持。  
InnoDB：提供事务支持事务、外部键等高级数据库功能。

**4、 表锁差异**

MyISAM：只支持表级锁，用户在操作myisam表时，select，update，delete，insert语句都会给表自动加锁，如果加锁以后的表满足insert并发的情况下，可以在表的尾部插入新的数据。  
InnoDB：支持表级锁和行级锁。行锁大幅度提高了多用户并发操作性能。

**5、 表的具体行数**

MyISAM：保存表的总行数，如果select count(\*) from table;会直接取出该值。  
InnoDB：没有保存表的总行数，如果使用select count(\*) from table；就会遍历整个表，消耗相当大，但是在加了wehre条件后，myisam和innodb处理的方式都一样。

应用场景：

Innodb：需要支持事务或者外键；大量 update 操作，可以提供并发操作的性能。

MyISAM：大量COUNT() 操作；insert 很少，select 很多；不要求事务

备份方案

**数据备份部分**

**1 逻辑备份**

* 应用场景

       数据量较小；数据库出现数据故障，对于恢复时间要求不高。

备份时间及地点

        每天固定时间进行备份，备份文件存放在本地，更安全的方式是备份到远程服务器（scp）。

* 备份方式

        采用mysqldump进行全库备份，通过定时任务，定时执行shell备份脚本。

**2 物理备份**

* 应用场景

      恢复时间要求较高；数据量比较大；

* 备份时间及地点

       每周在主库上备份。备份文件存放远程服务器目录下。

* 备份方式

       采用percona的社区工具innobackupex，该工具可以在线热备，不影响线上的业务。

以上两种方式的备份只能恢复某段时间的数据，对于按照时间点的恢复是无能为力的。

实时同步binlog日志到远程服务器上，这样理论上是可以恢复到任意时间点的。

**3 binlog备份**

* 应用场景

       对于一些由于错误操作等造成数据丢失错误的，需要按照时间点进行还原。

* 备份时间及地点

       备份服务器实时将主库上binlog同步到远程服务器上。

* 备份方式

       mysqlbinlog工具进行日志拉取，shell脚本如下：

       mysqlbinlog  --read-from-remote-server --host=1.1.1.1 --port=3306 --user="backup" --password="backup" --raw --stop-never mysql-bin.000840  --result-file=/data/backup/binlog/

经过以上三种结合的备份方式，基本上可以满足在数据异常丢失情况下，恢复到正常状态。

**备份需要考虑的问题**

可以容忍丢失多长时间的数据；  
恢复数据要在多长时间内完；   
恢复的时候是否需要持续提供服务；  
恢复的对象，是整个库，多个表，还是单个库，单个表。

总体策略：全备+二进制日志。

写个定时执行任务，定时在凌晨（三点）自动全备份(考虑数据库服务器在运行中不能停机)。同时删除一个星期以前的备份文件。

写另外一个定时任务，每隔一段时间（）做二进制日志的增量备份。同时删除以前的备份文件。

用crontab定时执行备份脚本代码：crontab -e

1. # /bin/bash
2. DB\_NAME="\*\*\*\*"
3. DB\_USER="\*\*\*\*"
4. DB\_PASS="\*\*\*\*"
5. BIN\_DIR="/usr/bin"
6. BACK\_DIR="/data/backdata"
7. DATE="mysql-`date +'%Y%m%d-%H:%M:%S'`"
8. LogFile="$BACK\_DIR"/dbbakup.log #日志记录保存的目录
9. BackNewFile=$DATE.sql
10. $BIN\_DIR/mysqldump --opt --force -u$DB\_USER  -p$DB\_PASS $DB\_NAME **>** $BACK\_DIR/$DATE.sql
11. echo ----------"$(date +"%y-%m-%d %H:%M:%S")"------------ **>>** $LogFile
12. echo  createFile:"$BackNewFile" **>>** $LogFile
13. #find "/data/backdata/" -cmin +1 -type f -name "\*.sql" -print **>** deleted.txt
14. find "/data/backdata/" -ctime +7 -type f -name "\*.sql" -print **>** deleted.txt
15. echo -e "delete files:\n" **>>** $LogFile
16. #循环删除匹配到的文件
17. cat deleted.txt | while read LINE
18. do
19. rm -rf $LINE
20. echo $LINE**>>** $LogFile
21. done
22. echo "---------------------------------------------------------------" **>>** $LogFile

#!/bin/bash

# description: MySQL buckup shell script

# author: Daniel

# web site: http://home.ustc.edu.cn/~danewang/blog/

st=$(date +%s)

USER="root"

PASSWORD="\*\*\*\*\*"#用户名

DATABASE="myblogdb" #数据库用户密码

MAIL="abcd@gmail.com"#mail

BACKUP\_DIR=/home/daniel/data\_backup/ #备份文件存储路径

LOGFILE=/home/daniel/data\_backup/data\_backup.log #日志文件路径

DATE=`date +%Y%m%d-%H%M`#用日期格式作为文件名

DUMPFILE=$DATE.sql

ARCHIVE=$DATE.sql.tar.gz

OPTIONS="-u$USER -p$PASSWORD $DATABASE"

#判断备份文件存储目录是否存在，否则创建该目录

if [ ! -d $BACKUP\_DIR ]

then

mkdir -p "$BACKUP\_DIR"

fi

#开始备份之前，将备份信息头写入日记文件

echo " ">> $LOGFILE

echo "--------------------" >> $LOGFILE

echo "BACKUP DATE:" $(date +"%y-%m-%d %H:%M:%S") >> $LOGFILE

echo "-------------------" >> $LOGFILE

#切换至备份目录

cd $BACKUP\_DIR

mysqldump $OPTIONS > $DUMPFILE

#判断数据库备份是否成功

if [[ $? == 0 ]]

then

tar czvf $ARCHIVE $DUMPFILE >> $LOGFILE 2>&1

echo "[$ARCHIVE] Backup Successful!" >> $LOGFILE

rm -f $DUMPFILE #删除原始备份文件,只需保留备份压缩包

&nbsp; &nbsp; # 把压缩包文件备份到其他机器上。

&nbsp; &nbsp; scp -P 1110 $BACKUP\_DIR$ARCHIVE ubuntu@\*.\*.\*.\*:/home/user/data\_backup/ >> $LOGFILE &nbsp;2>&1

&nbsp;else

echo "Database Backup Fail!" >> $LOGFILE

#备份失败后向管理者发送邮件提醒

mail -s "database:$DATABASE Daily Backup Fail!" $MAIL

fi

echo "Backup Process Done"

#删除3天以上的备份文件

#Cleaning

find $BACKUP\_DIR -type f -mtime +2 -name "\*.tar.gz" -exec rm -f {} \;