MySQL 数据库运维初级

一、Mysql 数据库同步的分类和原理

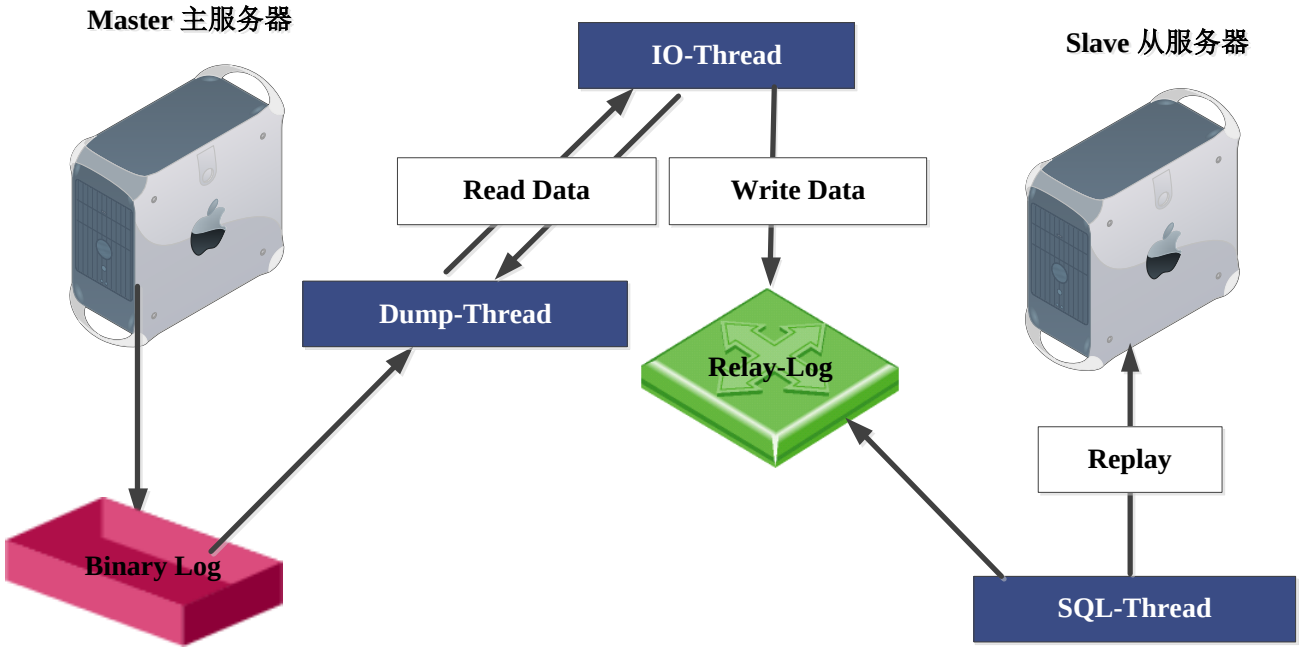
1、mysql 数据同步的方式：

(1)Synchronous Replication **同步复制：**指的是客户端连接到 MySQL 主服务器写入一段数据, MySQL 主服务器同步给 MySQL 从服务器需要等待从服务器发出同步完成的响应才返回客户端 OK，这其中等待同步的过程是阻塞的，如果有 N 台从服务器，效率极低

(2)Asynchronous Replication **异步复制：**指的是客户端连接到 MySQL 主服务器写入一段数据, MySQL 主服务器将写入的数据发送给 MySQL 从服务器，然后直接返回客户端 OK，马上或稍后一段时间进行写操作，可能从服务器的数据会和主服务器不一致

(3)Semisynchronous Replication **半同步复制：**指的是客户端连接到 MySQL 主服务器写入一段数据, MySQL 主服务器只将数据同步复制给其中一台从服务器，半同步复制给其他的从服务器，来达到其中一台从服务器完全同步的效果，其他从服务器之后写操作来提高效率。

MySQL主从复制微观图解



建立主从关系之后，要实现复制首先在 master 上开启 bin-log 日志功能，整个过程需要开

启 3 个线程：master 开启 IO 线程,slave 开启 IO 线程和 SQL 线程。主服务器若有数据修改之后，Binary-Log 会更新，从服务器通过 IO-Thread 读取主服务器的 Binary-Log 到本地并建立 Relay-Log，再通过 SQL-Thread 对其进行重放 Replay，从而同步到本地从服务器上。

Mysql 复制的机制：

(1)Slave 端线程：IO_Thread 向主服务器请求二进制日志中的事件，当读取完毕之后，IO 线程进行睡眠，当主服务器有新数据时则主服务器唤醒从服务器的 IO 线程 SQL_Thread 从中继日志读取事件并在本地执行，如果二进制日志开启同样也会记录日志，但为了节约可不用开启，中继日志通常会位于 OS 的缓存中，所以中继日志的开销很小。注：从服务器不能执行写操作，负责和主服务器的数据无法达到同步；*复制过程有一个很重要的限制——复制在 slave 上是串行化的，也就是说 master 上的并行更新操作不能在 slave 上并行操作。*

(2)Master 端线程：Binlog dump 将 IO_Thread 请求的事件发送到从服务器上，默认的为异步工作方式，主服务器写操作完成，与从服务的 dump 操作完成无关。

(4)在 slave 服务器执行 start slave,服务器的 IO 线程的请求后,master 服务器的 IO 线程根据 slave 服务器发送指定的 bin-log 日志之后的内容,然后返回给 slave 端的 IO 线程;(返回的信息中除了 bin-log 日志内容外,还有本地返回日志内容后在 master 服务器的新的 binlog 文件及在 binlog 中的下一个指定更新位置;

(5)slave 的 IO 线程接收到信息后,将接受到的日志一次添加到 slave 端的 relay-log（中继日志）文件的最末端,并将读取到的 master 端的 bin-log 的文件名和位置记录到 master-info 文件中,以便在下一次读取的时候能够清楚的告诉 master, "我需要从某个 bin-log 的哪个位置开始往后的日志内容,请发给我;"

(6)slave 的 SQL 线程检测到 relay-log 中新增加了内容后,会马上解析 relay-log 的内容成为 master 端真实执行时间的那些可执行的内容,并且重新回放.

2、常见的 Mysql 复制架构：

(1) Master/Slave **一主 N 从模式：**这种模式可以有效的分担读请求，但是写请求并不能完成负载分担、与从节点上的数据可能不一致。

一主一从：并且读写分离；

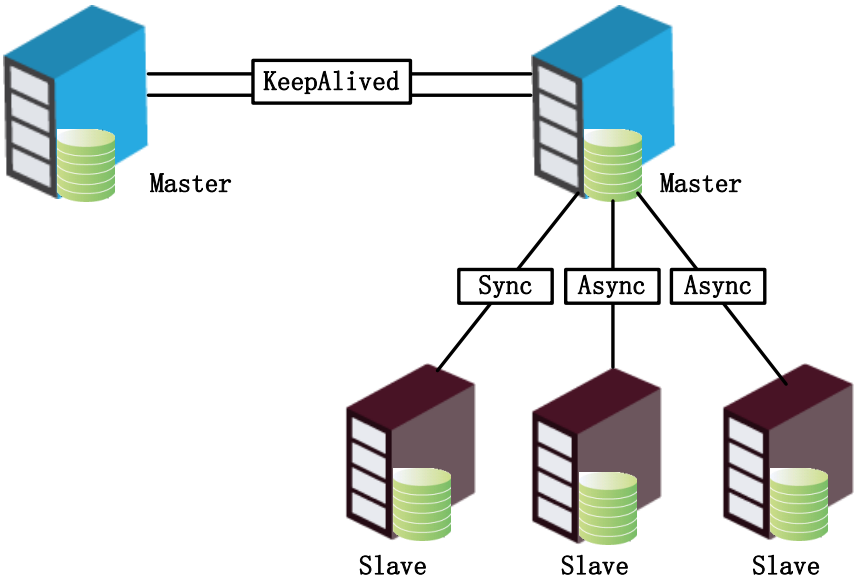
一主多从：主节点负责读，从节点负责写操作；主从模型下，前端分发器可识别读/写操作，并按需调度至目标主机，memocached/mysql-proxy/amoeba 可实现读写分离，读服务器进行负载均衡。使用一致性哈希算法。

一主多从外加一个冷备服务器：只用于备份，每隔一段时间关掉进行冷备份

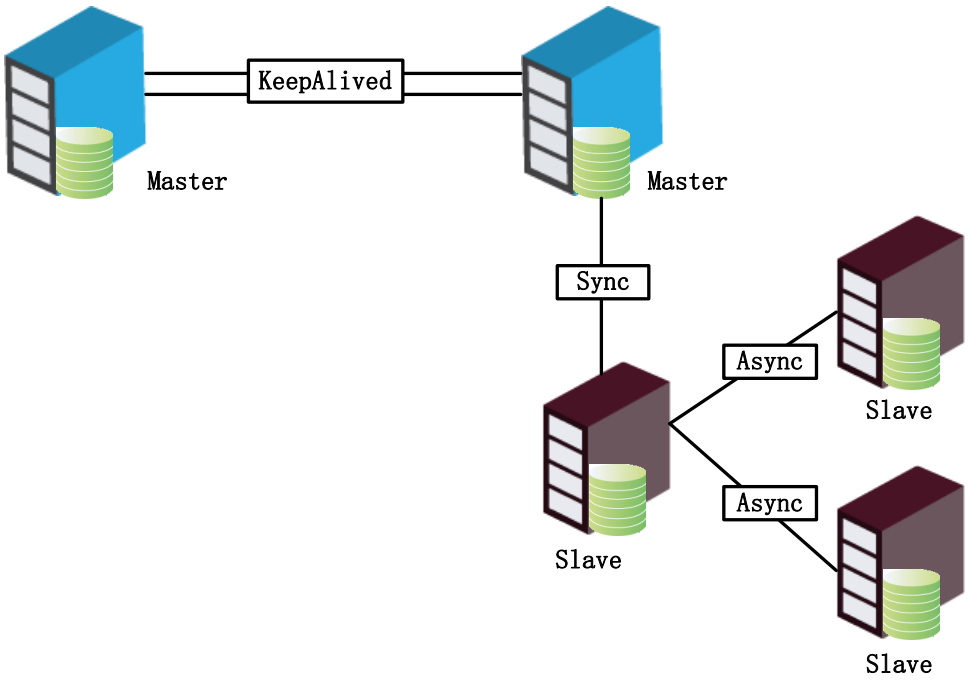
异地同步：主要避免自然灾害。

(2) Master/Master ***多主模式：***多个客户端同时写入数据时由于复制的延迟性可能导致数据冲突等严重问题，设置 log_slave_updates，Slave 可以是其它 Slave 的 Master，从而扩散 master 的更新。

基于以上两种模式的弊端，因此采用如下的策略：利用 keepalived 实现 Master 的高可用，并且使用半同步模式实现数据的完全同步，结构图如下：



注：主节点高可用，一个从节点同步，其他半同步实现



注：主节点实现高可用，一个从节点主从同步，其他从节点和该从节点实现半同步，该模式较上一个更加节省宽带，速度更快

3、Mysql 主从同步复制实践：

(1) 在 mysql 主节点上的配置文件的设置以及账户的建立：

```
[root@node03 ~]# grep  -Ev "^$|^#" /etc/my.cnf
[mysqld]
socket=/tmp/mysql.sock                          #mysql 的锁文件目录
log_bin=ON                                      #开启二进制日志记录
server_id=1                                     #选择一个唯一的 serverID(0-2^32)
datadir=/home/mydata/data           #mysql 的数据目录
max_allowed_packet          =       2M
read_buffer_size =          2M
read_rnd_buffer_size        =       8M
myisam_sort_buffer_size     =       128M
innodb_file_per_table       =       on        #InnoDB 为独立表空间模式，每一个表都会生成一个数据空间.
skip_name_resolve           =       on        #关闭 dns 反解
skip-external-locking
key_buffer_size  =          512M
binlog_format    =          mixed
sync_binlog      =          1         #及时写入二进制日志
replicate-do-db  =          all       #同步全部的数据库，如果只同步某一个数据库，改成数据库名称就可以
sql_mode=NO_ENGINE_SUBSTITUTION,STRICT_TRANS_TABLES
[root@node03 ~]# mysql
mysql>GRANT REPLICATION SLAVE,REPLICATION CLIENT ON *.* TO 'repluser'@'192.168.188.22' IDENT
IFIED BY 'repluser';        #添加从节点上的复制账户信息
mysql>FLUSH PRIVILEGES;
mysql>\q
```

(2) 在 mysql 从节点上作如下操作：

```
[root@node02 mysql]# grep -Ev "^$|^#" /etc/my.cnf
[mysqld]
socket=/tmp/mysql.sock
log_bin=ON
server_id=2                 #从节点必须指定唯一的 server_ID
datadir=/home/mydata/data
skip-external-locking
max_allowed_packet          =       2M
read_buffer_size =          2M
```

```
sort_buffer_size =        2M
read_rnd_buffer_size      =        8M
table_open_cache =        256
myisam_sort_buffer_size   =        128M
query_cache_size =        32M
innodb_file_per_table     =        on
skip_name_resolve         =        on
key_buffer_size  =        512M
binlog_format    =        mixed
sync_binlog      =        1
relay-log        =        relay-log                #开启中继日志，名称为 relay-log
relay-log-index  =        relay-log.index          #定义中继日志的位置和名称
sql_mode=NO_ENGINE_SUBSTITUTION,STRICT_TRANS_TABLES
[root@node02 mysql]# mysql
mysql>CHANGE MASTER TO MASTER_HOST='192.168.188.33',MASTER_USER='repluser',MASTER_PASSWORD=
repluser',MASTER_LOG_FILE='ON.000001',MASTER_LOG_POS=250';
mysql>START SLAVE;
或者如下方式启动：
mysql>START SLAVE IO_Thread;      #开启 IO 线程
mysql>START SLAVE SQL_Thread;     #开启 SQL 线程
mysql>SHOW SLAVE STATUS\G         #查看从节点复制的状态信息
mysql>STOP SLAVE                  #关闭主从复制
```

注：在开启主从复制的过程中可能会报错，检查防火墙以及 SElinux 是否关闭或者开启相应的端口，当出现其他问题时，及时查看日志信息，除非设置，默认在其数据目录之下。在连接至主节点时候，MASTER_LOG_FILE 和 MASTER_LOG_POS 不是必须指定的选项。当 Slave_IO_Running 和 Slave_SQL_Running 显示为 Yes 时则表示已正常开始复制。至此一个简单的 MySQL 主从复制架构完备，以下是测试阶段

(3) 测试阶段：

```
在 master 主节点上创建数据库创建数据表，添加数据，并查看在从节点上是否有数据复制：
[root@node03 ~]# cat test.sh
#/bin/bash
MYSQL="/usr/local/mysql/bin/mysql -uroot -p123 -e"
$MYSQL  "CREATE DATABASE mydb;"
$MYSQL  "CREATE  TABE mydb.tb1(ID INT UNSIGNED AUTO_INCREMENT PRIMARY KEY,NAME VARCHAR(30));"
for LAG in {1..100}
do
   $MYSQL "INSERT INTO mydb.tb1(NAME) VALUES('stu${LAG}');"
```

```
Done
echo "Insert Over!"
[root@node03 ~]# sh test.sh
[root@node02 ~]# mysql
mysql>SHOW DATABASES;
mysql>SELECT * FROM mydb.tb1;
mysql>SHOW SLAVE STATUS;
```

4、Mysql 半同步复制的要点：

(1)配置 master：

```
半同步的配置需要安装插件，插件位置在/usr/local/mysql/lib/plugin 目录下，其中，
        主节点 master 用的插件是：/usr/local/mysql/lib/plugin/semisync_master.so;
        从节点 slave 用的插件是：/usr/local/mysql/lib/plugin/semisync_slave.so
Mysql 安装插件的步骤如下：
[root@node03 ~]# mysql
mysql>INSTALL PLUGIN rpl_semi_sync_master SONAME 'semisync_master.so'; #主节点安装插件
mysql> SHOW GLOBAL VARIABLES LIKE '%semi%';          #查看变量名称，检测是否加载成功
mysql>SET GLOBAL rpl_semi_sync_master_enabled=1;    #设置全局变量启用插件
mysql>SET GLOBAL rpl_semi_sync_master_timeout=2000;#设置连接超时时间，单位是毫秒
```

(2)配置 slave：

```
[root@node02 ~]# mysql
mysql>INSTALL PLUGIN rpl_semi_sync_slave SONAME 'semisync_slave.so';          #从节点安装插件
mysql> SHOW GLOBAL VARIABLES LIKE '%semi%';          #查看变量名称，检测是否加载成功
mysql>SET GLOBAL rpl-semi_sync_enabled=1;            #设置全局变量启用插件
```

5、主从架构配置过程中要点：

(1)版本：主从版本须一致或者主节点的版本高于从节点的版本

(2)主从复制点：一般从起始位置开始复制，如果主服务器已经存在不小的数据集，则主服务器先备份，然后再从服务节点上恢复，从主服务器上备份所处的位置开始复制，从节点上恢复是可关闭二进制日志记录。

(3)服务 ID（server-id）号：切记所有的主从节点机的 server_ID 都不相同切都是唯一的，server_ID 范围是：1~2^32，在配置文件中：

```
[root@node03 ~]# echo "server-id=1" >> /etc/my.cnf           #设置 server-id 号
[root@node03 ~]# service mysqld  restart                     #修改配置文件需要重启服务
mysql> SHOW GLOBAL VARIABLES LIKE 'server_id';               #在 mysql 中查看节点机的服务 ID
+---------------+-------+
| Variable_name | Value |
```

```
+---------------+-------+
| server_id     | 1       |
```

(4)启用二进制日志：

```
[root@node03 ~]# echo "log_bin=/mydata/data/binlog/mysql-bin" >> /etc/my.cnf #设置开启二进制日志记录功能，并设定日志所处的目录和名称
[root@node03 ~]#service mysqld restart
mysql>SHOW GLOBAL VARIABLES LIKE 'log_bin';
```

(5)主节点特有配置过程，创建复制权限的账号：

```
[root@node03 ~]#mysql
mysql>GRANT REPLICATION SLAVE.REPLICATION ON *.* TO 'repluser'@'192.168.188.11' IDENTIFIED BY 'repluser';
mysql>FLUSH PRIVILEGES;
```

(6)从节点特有配置过程，启用中继日志，关闭二进制日志(可选)，写入配置文件中，连接主服务器并启动复制线程：

```
[root@node03 ~]#echo "relay_log=/mydata/data/relay_log" >> /etc/my.cnf
[root@node03 ~]#service mysqld restart
[root@node03 ~]#mysql
mysql>CHANGE MASTER TO \
      MASTER_HOST='192.168.188.33' \        #主服务器的 IP 地址
      MASTER_USER='repluser'         \       #连接主服务器上的 mysql 用户
      MASTER_PASSWORD='repluser'     \       #连接主服务器 mysql 用户的密码
      MASTER_LOG_FILE='ON.00002'     \       #用来复制特定的哪一个二进制文件
      MASTER_LOG_POS=566;           #从上面的二进制文件中的哪个位置开始复制。最终的是前三个参数
mysql>START SLAVE;                  #启动复制线程
mysql>SHOW SLAVE STATUS\G;          #查看复制状态
mysql>SHOW PROCESSLIST\G; #查看从节点的复制过程
```

(7)主从复制实验中的一些总结：

1)主服务器二进制日志如果在数据库初始化的时候开启了，则从服务器需要指定初始化结束后的日志进行复制。否则会报错。可能因为无法再次创建系统库所致

2)主服务器最好在系统库初始化完成后再开启二进制日志。这样从服务器就可以直接从第一个二进制日志开始复制。

3)如果错误日志中出错通常是 SQL 线程，I/O 产生的。 如果 I/O 出问题通常与权限和链接有关，SQL 出问题通常与二进制日志在从服务器执行有关。

(8)主从架构中和复制相关的文件介绍：

1)master.info：用于保存从服务器连接到主服务器需要的账户等信息；

2)relay-log.info：二进制日志和中继日志的位置，日志坐标等信息；

3)为安装起见需在从服务器上开启以下参数：

```
[root@node03 ~]#cat /etc/my.cnf
sync_master_info =        1
sync_relay_log            =        1
sync_relay_log_info       =        1
```

4)从服务器意外崩溃是，建议使用 pt-slave-start 命令启动从服务器

(9)Mysql 复制的三种方式：配置文件(/etc/my.cnf)的设置方式：——>binlog_format=mixed

1)基于语句(列)的复制：在主服务器上执行的 SQL 语句，在从服务器上执行同样的语句，一旦发现没法精确复制时，会自动选着基于行的复制。优点：数据量小，容易查看和识别，适应性强；缺点：特殊语句无法精确复制；

2)基于行的复制：把改变的内容复制过去，而不是把命令在从服务器上执行一遍.优点：都能完成精确复制，包括触发器存储过程，较少占用 CPU 资源，较少锁的使用；缺点：通过日志无法判断执行了哪些语句

3)混合模式：默认采用基于语句的复制，一旦发现基于语句的无法精确的复制时，就会采用基于行的复制。mysql 复制的默认模式。

(10)从服务器落后于主服务器检测方式：

```
[root@node03 ~]#mysql
mysql>SHOW SLAVE STATUS\G
        ..................................
Seconds_Behind_Master: 0  #数值显示落后于主服务器多少秒，若主服务器繁忙则该值通常为正值，若波动不大则都可以接受。
        ..................................
```

(11)主从复制中的复制过滤问题，从节点仅复制主节点上的部分数据库或者数据表，优点是节约了主服务器的 I/O 带宽，缺点是其他数据库的数据安全无法保证，无法完成时间点还原，配置方式：

在主节点上的参数：其作用是过滤二进制日志中的事件

```
        binlog_do_db                    =
        bin_ignore_db                   =
```

在从服务器上的参数：其作用是过滤中继日志中的事件

```
基于数据库：
        replicate_do_db                 =
        replicate_ignore_db             =
```

```
基于数据表：
        replicate_do_table                =       db_name.tb_name
        replicate_ignore_table            =
        replicate_rewrite_db              =
基于表使用的通配符：
        replicate_wild_do_table   =
        replicate_sild_ignore_table       =
```

(12)添加新 slave 服务器

假如 master 已经运行很久了，想对新安装的 slave 进行数据同步，甚至它没有 master 的数据。此时，有几种方法可以使 slave 从另一个服务开始，例如，从 master 拷贝数据，从另一个 slave 克隆，从最近的备份开始一个 slave。Slave 与 master 同步时，需要以下三个数据文件

1)master 的某个时刻的数据快照(其中涉及到逻辑卷 LVM 的建立以及扩容缩容快照知识点)；

2)master 当前的日志文件、以及生成快照时的字节偏移。这两个值可以叫做日志文件坐标(log file coordinate)，因为它们确定了一个二进制日志的位置，

```
[root@node03 ~]#mysql
mysql>SHOW MASTER STATUS                        #查看二进制日志的位置
```

3)master 的二进制日志文件。

可以通过以下几中方法来克隆一个 slave：

1) 冷拷贝(cold copy):停止 master，将 master 的文件拷贝到 slave；重启 master。缺点很明显。

2)热拷贝(warm copy):如果仅用 MyISAM 表，可以用 mysqlhotcopy 拷贝，即使服务器正在运行。

3)使用 mysqldump 来得到一个数据快照可分为以下三部：

<1>锁表：如果还没有锁表，应该对表加锁，防止其它连接修改数据库，否则，得到的数据是不一致的。如下：

```
[root@node03 ~]#mysql
mysql>FLUSH TABLES WITH READ LOCK; #对表施加读锁
```

<2>在另一个连接用 mysqldump 创建一个你想进行复制的数据库的转储：

```
[root@node03 ~]#mysqldump -all-databases -lock-all-tables > dbdump.db   #采用 mysql 自带的备份工具 mysqldump 工具进行冷备份
```

<3>对表释放锁：

```
[root@node03 ~]#mysql
mysql>UNLOCK TABLES;
```

(13)复制的体系结构有以下一些基本原则（重申）：

1)每个 slave 只能有一个 master；

2)每个 slave 只能有一个唯一的服务器 ID；server_id 必须唯一

3)每个 master 可以有很多 slave；

4)如果设置 log_slave_updates，slave 可以是其它 slave 的 master，从而扩散 master 的更新（主主模式需配置的参数）

6、基于 ssl 的 mysql 主从复制

(1)在 Master 主节点上生成根 CA、自签证书以及创建所需要的辅助文件：

```
[root@node03 ~]#cd /etc/pki/CA
[root@node03 ~]#(umask 077;openssl genrsa -des3 -out ./private/cakey.pem 2048)
[root@node03 ~]#openssl req -new -x509 -key ./private/cakey.pem -out cacert.pem -days 3650
[root@node03 ~]#touch index.txt
[root@node03 ~]#echo 01 > serial
[root@node03 ~]#ls
cacert.pem       certs    crl      index.txt        newcerts private  serial
```

(2)为 Master Server 创建证书以及跟 CA 签发 master 的证书申请，将 Master 的自签证书拷贝到目录/usr/local/mysql/ssl/下，并修改各证书文件的权限，

```
[root@node03 ~]#mkdir /usr/local/mysql/ssl
[root@node03 ~]#cd /usr/local/mysql/ssl
[root@node03 ~]#(umask 077;openssl genrsa -des3 -out mysql.master.key 2048)
[root@node03 ~]#openssl req -new -key mysql.master.key -out mysql.master.cer
[root@node03 ~]#openssl ca -in mysql.master.csr -out mysql.master.crt -days 3650
[root@node03 ~]#cp /etc/pki/CA/cacert.pem /usr/local/mysql/ssl/
[root@node03 ~]#chown mysql:mysql /usr/local/mysql/ssl/
```

(3)在 Slave Server 上 SSl 的配置，并将 Master 上的自签证书拷贝到各个服务器上

```
[root@node02 ~]# mkdir /usr/local/mysql/ssl
[root@node02 ~]#cd /usr/local/mysql/ssl
[root@node02 ~]#(umask 077;openssl genrsa -des3 -out mysql.slave.key 2048)
[root@node02 ~]#openssl req -new -key mysql.slave.key -out mysql.slave.csr
[root@node02 ~]#scp mysql.slave.csr node03:/root/
[root@node03 ~]#openssl ca -in mysql.slave.csr -out mysql.slave.crt -days 3650
[root@node03 ~]#scp mysql.slave.crt node02:/usr/local/mysql/ssl/
[root@node03 ~]#scp /etc/pki/CA/cacert.pem node02:/usr/local/mysql/ssl/
[root@node02 ~]# chown mysql:mysql /usr/local/mysql/ssl/
```

(4)编辑 Master 的/etc/my.cnf 配置文件，启用 mysql 的 ssl 功能：

```
[root@node03 ~]#cat /etc/my.cnf
.........................
```

```
ssl
ssl-ca   =        /usr/local/mysql/ssl/cacert.pem
ssl-key  =        /usr/local/mysql/ssl/mysql.master.key
ssl-cert=         /usr/local/mysql/ssl/mysql.master.crt
.....................................................
[root@node03 ~]#service mysqld restart
[root@node03 ~]#mysql
mysql>grant replication slave,replication client on *.* to 'repluser'to'192.168.188.22' iden
tified by 'repluser' require ssl;
mysql>flush privileges;
```

(5)编辑 Slave 的/etc/my.cnf 配置文件，启用 mysql 的 ssl 功能：

```
[root@node02 ~]#cat /etc/my.cnf
.............................
ssl
ssl-ca   =        /usr/local/mysql/ssl/cacert.pem
ssl-key  =        /usr/local/mysql/ssl/mysql.slave.key
ssl-cert=         /usr/local/mysql/ssl/mysql.slave.crt
.................................................
[root@node02 ~]#service mysqld restart
[root@node02 ~]#mysql
mysql>change master to
    -> master_host='192.168.188.33',
    -> master_user='repluser',
    -> master_password='replpasswd',
    -> master_log_file='ON.000003',
    -> master_log_pos='761',
    -> master_ssl=1,
    -> master_ssl_ca='/usr/local/mysql/ssl/cacert.pem ',
    -> master_ssl_cert='/usr/local/mysql/ssl/mysql.slave.crt ',
    -> master_ssl_key='/usr/local/mysql/ssl/mysql.slave.key ';
mysql>start slave;
mysql>show slave status\G;
mysql>show processlist;
```

注意：在建立基于 ssl 的 mysql 连接时，必须要保住 master 和 slave 的时间同步！至此基于 ssl 连接的 mysql 主从复制建立完成，测试部分和之前的类似。

二、Mysql 备份详解(常见的有四种备份方式)

0、二进制日志介绍

(1) 内容：引起 mysql 服务器改变的任何操作都会记录二进制日志；

作用：mysql 的备份复制都依赖于此；Salve 通过复制主服务器的二进制日志完成主从复制，slave 在执行之前将保存于 Slave 的中继日志中;从服务器通常关闭二进制日志记录功能(log_bin=OFF)来提升性能。

存放位置：默认在 mysql 的数据目录下，默认的文件名为：ON.000001，另还有 ON.index 用来记录被 mysql 管理的二进制文件列表；

(2)二进制日志有关的操作：

```
mysql>SHOW MASTER STATUS; ##查看正在使用的二进制日志：
mysql>FLUSH LOGS;          ##手动滚动二进制日志；
mysql>SHOW BINARY LOGS;    ##显示使用过的二进制日志列表
##以表的形式查看二进制日志文件：
##     Usage：SHOW BINLOG EVENTS [IN 'log_name'] [FROM start_pos] [LIMIT [offset,] row_count]
##示例：  mysql>SHOW BINLOG EVENTS IN 'ON.000002'\G;
##二进制日志的读取工具——mysqlbinlog
##       Usage：mysqlbinlog [--start-datetime --stop-datetiem  --start-position  --stop-posi
tion] LOGFILE_NAME
##参数释义：截取 mysql 操作或从时间起始角度或者从标志起始角度；
##示例：  mysql>mysqlbinlog --start-position 2052 --stop-position 30546 ON.000002
##删除二进制日志：
##       Usage：PURGE  {BINARY|MASTER} LOGS {TO 'binlog_name'|BEFROE datetime_expr}
##参数释义：删除某一个日志文件或者删除某一个时间点之前的所有日志文件
##示例：  mysql>PUGRE BINARY LOGS TO 'ON.000001';
         mysql>PURGE BINARY LOGS BEFORE '2016-12-2_17-10-12 22:46:23';
```

(3)二进制日志的记录格式：由 bin_log_format={statement|row|mixed}定义：

statement 基于生成数据的语句记录，缺点是如果当时插入信息为函数生成的，则有可能不同时间点执行结果不同(示例：插入时间点的函数，则在不同时间执行所得的值必定不同：INSERT INTO table_name VALUSE(CURRENT_TIME());)；

row 基于行数据的记录，缺点是频繁操作 mysql 时，会记录大量数据；

mixed 混合模式，由 mysql 最优化选择在何种情况下使用上述哪一种记录方式；

(4)二进制日志相关参数总结：

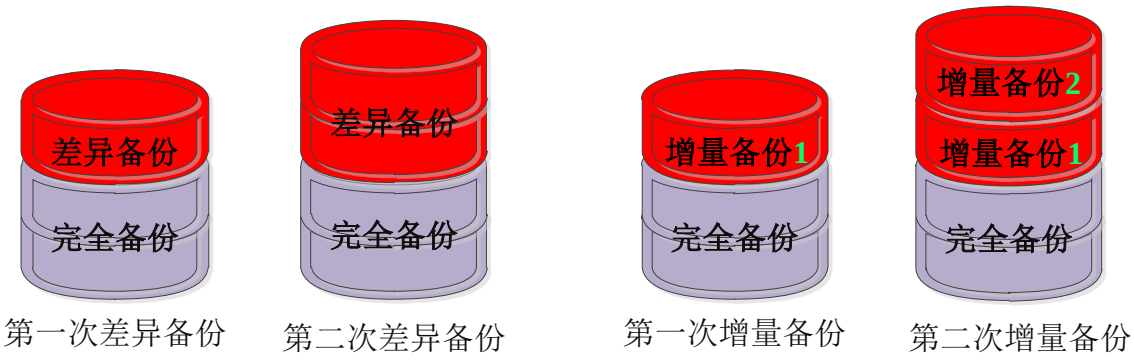
```
[root@node02 ~]#cat /etc/my.cnf              #mysql 的配置文件，在 Mysql 的家目录(MYSQl_HOME)以及 r
oot 的家目录(/root/.my.cnf)均可生效
log_bin  =        {ON|OFF}          #还可以是一个路径名，用于控制全局 binlog 的存放位置和是否开启
```

```
binlog_format     = {statement|row|mixed}    #二进制日志的记录类型，一般选择 mixed 类型
max_binlog_size   =         10M              #二进制日志文件的上限，超过后自动滚动
sql_log_bin       =         {ON|OFF} #会话级别是否关闭 binlog,若关闭则会话内的操作将不会记录，一般在恢复和还原 mysql 时关闭
sync_binlog       =         {0|1}            #是否马上同步事物类操作到二进制日志中
max_binlog_cache_size                        #自定义二进制日志的缓冲空间大小，仅用于缓冲事物类的语句
max_binlog_smt_cache_size                    #自定义语句类缓冲空间大小，非事务类和事务类公用的空间大小
```

1、数据库备份的类型：

完全备份：将整个数据库备份；

部分备份：又分为增量备份：只备份上一次的增量或完全备份以来变化的数据，特点是：节约空间，还原比较麻烦；差异备份：只备份上一次完全备份以来变化的数据，特点是：浪费空间，但是相比增量备份还原方便。



第一次差异备份　第二次差异备份　第一次增量备份　第二次增量备份

2、Mysql 备份数据的方式：

(1)热备份：数据库的读写操作均不受影响；

(2)温备份：数据库的读操作不受影响，但是不能执行写操作；

(3)冷备份：数据库不能进行正常的读写操作，即就是数据库需要下线。

Mysql 常用的两个存储引擎(MyISAM 和 InnoDB)中，MyISAM 仅不支持热备份。数据库中数据的备份方式有：

(1)物理备份：直接通过 tar、cp 命令打包复制数据文件；

(2)逻辑备份：通过特定工具从数据库中导出数据并另存为备份(逻辑备份会丢失数据精度)。

3、Mysql 备份的前提条件：

3.1、需要备份的数据：数据库表、二进制日志，InnoDB 事务日志、代码(存储过程、存储函

数、触发器、时间调度器)、服务器配置文件；

3.2、备份工具简介：

(1)mysqldump：逻辑备份工具，适用于所有存储引擎，支持温备、完全备份、部分备份，对于 InnoDB 支持热备份；

(2)cp、tar 等归档复制工具：物理备份工具，适用于所有的存储引擎；

(3)lvm2 snapshot：几乎热备，借助文件系统（逻辑卷）管理工具进行备份；

(4)xtrabackup：强大的 InnoDB/XtraDB 热备工具，支持完全备份、增量备份，需重点掌握!!!

3.3、设计合适的备份策略：

(1)直接 cp/tar 复制数据库文件；适用于数据量较小的场景中；

(2)mysqldump+复制二进制日志文件(log_bin)；适用于小型数据库的场景中，先使用 mysqldump 进行完全备份，而后定期备份二进制日志；

(3)逻辑卷快照(snapshot)+ 复制二进制日志文件(log_bin)；适用于中型数据库的场景中，先建立 lvm2 快照完全备份，而后定期备份二进制日志；

(4)利用 xtarbackup 工具备份，适用于大型数据库，使用 xtarback 进行完全备份后，定期使用 xtrabackup 进行增量或者差异备份。

4、实战演练——数据库常见备份方式实例

(1)使用 cp/tar 进行备份：

```
[root@node01 ~]#mysql
mysql>FLUSH TABLES WITH READ LOCK;                    #向所有的数据表施加读锁；
[root@node01 ~]#mkdir /backup                         #创建存放备份数据库的文件；
[root@node01 ~]#cp -a /home/mydata/data/* /backup     #保留数据的原始权限进行复制；
或者采用 tar 命令进行数据文件的归档备份：
[root@node01 ~]#tar  zcavf backup.tar.gz  /home/mydata/data/*
[root@node01 ~]#rm -rf /home/mydata/data                    #模拟数据文件丢失
[root@node01 ~]#service mysqld restart                      #重新初始化数据库，源码安装的采用：
[root@node01 ~]#mysqld -initialize -user=mysql -datadir=/home/mydata/data
[root@node01 ~]#cp -a /backup/*    /home/mydata/data     #将备份的数据文件拷贝至 mysql 的数据目录下
至此最简单的物理备份方式和复原进行结束。
```

(2)使用 mysqldump+复制二进制日志(binlog)备份(前提是开启二进制日志记录功能”log_bin=ON”)

```
mysqldump 命令的基本语法：
[root@node01 ~]#mysqldump [options] db_name [tb1_name ...]   #恢复需要手动建立相应的数据库
[root@node01 ~]#mysqldump [options] --databases db_name ... #恢复时自动建立数据库
```

基于不同的存储引擎，mysqldump 使用不同的参数：

```
##MyISAM 存储引擎：支持温备，备份时需要锁定表：
##options：        -x,--lock-all-tables      #给所有数据库的所有表施加读锁；
                   -l,--lock-tables          #给指定的数据库的所有表施加读锁；
##InnoDB 存储引擎：支持温备、热备：
##options:--single-transaction     #创建一个事务，基于此快照执行备份，与事务隔离级别有关
          -R，--routines            #存储过程和存储函数的记录
          --triggers                #备份表的触发器，--skip-triggers
          -E,--events               #备份事物调度器；
          --flush-logs              #锁定表完成后执行日志滚动
          --master-date[=1/2]：1->CHANGE MASTER TO 语句不被注释；2->注释掉 CHANGE MASTER TO 语句
```

示例：

```
[root@node01 ~]#mysql -e "SHOW MASTER STATUS;"                  #查看当前二进制日志文件的状态和 posi
tion
[root@node01 ~]#mysqldump --all-databases --lock-all-tables --flush-logs > backup.sql #将数
据库备份到 backup.sql 文件中
[root@node01 ~]#cp /home/mydata/data/ON.000004      /root     #由于在上述备份过程中会产生新的数据
文件，故产生二进制日志文件，所有 copy 这一段时间内的二级制日志至备份目录下
[root@node01 ~]#rm -rf /home/mydata/data                        #模拟数据文件丢失
[root@node01 ~]#service mysqld restart                          #重新初始化数据库，源码安装的采用：
[root@node01 ~]#mysqld -initialize -user=mysql -datadir=/home/mydata/data
[root@node01 ~]#mysql
mysql>SET sql_log_bin=OFF;          #恢复数据时可暂且关闭二进制日志的记录功能，节省磁盘空间和 I/O
mysql>SOURCE backup.sql             #复原数据
[root@node01 ~]#mysqlbinlog --start-position=106 --stop-position=1923 ON.000004 #通过二进制日
志增量恢复数据文件，指定起始和结束的复原位置
至此 mysqldunmp 的备份和复原进行结束。
```

(3)lvm2 快照备份数据+复制二进制日志(binlog)备份(前提是开启二进制日志记录功能"log_bin=ON")

lvm 对 lv 提供了快照"snapshot"备份功能，也只对 lvm 有效。snapshot 有多种实现方法，用于数据库的多位写时复制 COW(Copy-On-Write)，当一个快照建立的时候，仅拷贝原始数据里的原始数据。当原始文件里有数据写入时，备份卷开始记录原始卷哪些数据发生改变，然后再

原始卷中新数据覆盖旧数据时，将旧数据拷贝到 snapshot 的预留空间中起到备份数据的作用，保证了所有数据和创建备份之前的数据的一致性。

```
##根据上述介绍，snapshot 只针对逻辑卷(lvm)有效，因此需要建立逻辑卷(逻辑卷卷标是 8e)，以下是创建逻辑卷(假设新添加了一块 scsi 盘在/dev/sdb 目录下)：
[root@node01 ~]#cat sdb1
n p 1 1 10G t 8e w                                  #非交互式添加逻辑卷的方式，间隔是换行的方式！
[root@node01 ~]#fdisk /dev/sdb < sdb                #执行上述分区方式
[root@node01 ~]#partprobe /dev/sdb                  #保存并同步分区
##创建逻辑卷
[root@node01 ~]#pvcreate /dev/sdb1
[root@node01 ~]#vgcreate myvg /dev/sdb1
[root@node01 ~]#lvcreate -n mydata -L 5G myvg
[root@node01 ~]#mkfs.ext4 /dev/myvg/mydata
[root@node01 ~]#mkdir /lvm_data
[root@node01 ~]#mount /dev/myvg/mydata /lvm_data
##修改配置文件，将 mysql 的数据目录指向逻辑卷所挂载目录下：
[root@node01 ~]#sed -i  's@datadir=.*@datadir=/lvm_data@g' /etc/my.cnf
[root@node01 ~]#service mysqld restart
##创建快照卷并备份：
[root@node01 ~]#mysql -e "FLUSH TABLES WITH READ LOCK;"      #对所有的数据表施加读锁
[root@node01 ~]#lvcreate -L 1G -n mydata-snap -p r -s /dev/myvg/mydata          #创建快照卷
[root@node01 ~]#mysql -e "UNLOCK TABLES;"           #释放刚才对表所加的读锁
[root@node01 ~]#mkdir /lvm_snap                                 #创建快照卷挂载目录
[root@node01 ~]#mount /dev/myvg/mydata-snap  /lvm_snap       #挂载快照卷，切记不可格式化快照卷
[root@node01 ~]#tar zcvf /backup.tar.gz /lvm_snap/* #将数据文件打包备份
[root@node01 ~]#umount /lvm_snap                                #卸载快照卷
[root@node01 ~]#lvremove myvg mydata-snap                       #删除快照卷
##模拟破坏数据、恢复数据：
[root@node01 ~]#rm -rf /lvm_data
[root@node01 ~]#service mysqld restart                          #重新初始化数据库，源码安装的采用：
[root@node01 ~]#mysqld -initialize -user=mysql -datadir=/home/mydata/data
[root@node01 ~]#tar zxvf /backup.tar.gz -C /lvm_data    #将备份的数据文件解压到 mysql 的属具目録下
至此关于利用逻辑卷的快照功能实现 mysql 的数据备份完成。
```

(4)使用 Xtrabackup 进行完全备份：

```
##下载安装 xtrabackup：
[root@node01 ~]#wget https://dl.fedoraproject.org/pub/epel/6/x86_64/epel-release-6-8.noarch.rpm
```

```
[root@node01 ~]#rpm -ivh epel-release-6.8.norach.rpm          #安装 epel 源
[root@node01 ~]#wget https://www.percona.com/downloads/XtraBackup/Percona-XtraBackup-2.3.4
/binary/redhat/6/x86_64/percona-xtrabackup-2.3.4-1.el6.x86_64.rpm
[root@node01 ~]#yum localinstall --nogpgcheck percona-xtrabackup-2.3.4-1.el6.x86_64.rpm
[root@node01 ~]#mysql
mysql>GRANT RELOAD,LOCK TABLES,REPLICATION CLIENT on *.* TO 'bkuser'@'192.168.188.11' IDENTI
FIED BY 'bkuser';#添加数据库备份的用户(可选)
mysql>FLUSH PRIVILEGES;
##使用 xtarbackup 的前端配置工具 innobackupex 实现对数据库的完全备份(注：innobackupex 备份会调用 xtraba
ckup 备份所有的 innoDB 表，复制所有关于表结构定义文件(.frm)、以及 MyISAM、MEGRE、CSV 和 ARCHIVE 表的相关
文件，同时会备份触发器和数据库配置文件，并且保存为一个以时间命名的目录)
[root@node01 ~]#mkdir /extrabackup
[root@node01 ~]#innobackupex --user=bkuser --password=bkuser /extrabackup/    #备份数据
########################提示 commplete 代表备份成功#########################
##一般情况下，根据上述步骤备份完成后，不能用于恢复数据，因为备份的数据中可能包含仍未提交的事务或者是尚未同步
至数据文件中心的事务。因此，此时的数据文件仍然=不一致，还需准备还原阶段：
[root@node01 ~]#innobackupex --apply-log  /extrabackup/2016-12-2_17-06-35   #指定备份文件目录
############################提示 commplete OK！代表备份成功###################
##模拟破坏数据、恢复数据
[root@node01 ~]#rm -rf /home/mydata/data/*     #模拟破坏数据，可不用重启或初始化数据库也可还原！
[root@node01 ~]#innobackupex --copy-back /extrabackup/2016-12-2_17-06-35      #恢复数据
[root@node01 ~]#killall -9 mysqld                    #关闭 mysqld 进程
[root@node01 ~]#chowd -R mysql:mysql /home/mydata/data/*     #切记修改还原后的数据的属主属组
[root@node01 ~]#service mysqld restart
##重启 mysql 服务，至此采用 xtrabackup 对 mysql 进行完全备份进行完毕
```

使用 Xtrabackup 进行增量备份、恢复数据：

```
##进行如下两次增量备份：
[root@node01 ~]#innobackupex --user=bkuser --password=bkuser --incremental /extrabackup/ --
incremental-basedir=/extrabackup/2016-12-2_17-06-35
[root@node01 ~]#innobackupex --user=bkuser --password=bkuser --incremental /extrabackup/ --
incremental-basedir=/extrabackup/2016-12-2_17-10-12
##innobackupex 参数解释：--incremental
#会在/extrabackup 目录下自动创建一个新的以时间命名的目录，以存放所有的增量备份的数据
                        --incremental-basedir
#指向上一次增量或者完全备份所在的目录，增量备份对于 MyISAM 无法增量只能是完全备份。


##整理增量备份：
[root@node01 ~]#innobackupex --apply-log --redo-only /extrabackup/2016-12-2_17-06-35
```

```
[root@node01 ~]#innobackupex --apply-log --redo-only /extrabackup/2016-12-2_17-06-35  --incremental-dir=/extrabackup/2016-12-2_17-10-12
 [root@node01 ~]#innobackupex --apply-log --redo-only /extrabackup/2016-12-2_17-06-35 --incremental-dir=/extrabackup/2016-12-2_17-06-35
#整理增量备份的文件，注意指定完全备份的文件!!!

##模拟破坏数据、恢复数据：
[root@node01 ~]#service mysqld stop
[root@node01 ~]#rm -rf /home/mydata/data/*
##与完全备份类似，直接 copy-back 完全备份的那个目录：
[root@node01 ~]#innobackupex --copy-back /extrabackup/2016-12-02_17-06-35     #整理完增量备份后可以直接通过全量备份还原数据文件
[root@node01 ~]#chown -R mysql:mysql /home/mydata/data/*     #切记修改还原后的数据文件属主属组
```

xtrabackup 的”数据流”和压缩功能：

```
Usage：innobackupex  --stream=tar  /backup | gzip > /backup/`date +%F_%H-%M-%S`.tar.gz
实例:innobackupex --user=bkuser --password=bkuser --stream=tar /backup/ | gzip > /backup/`date +%T_%H-%M-%S`.tar.gz

##"导出"表，导出表是在备份的 prepare 阶段进行的，因此，一旦完全备份完成，就可以在 prepare 过程中通过-export 选项将某表导出了：
Usage：innobackupex --apply-log --export /path/to/backup
实例：     innobackupex --user=bkuser --passowrd=bkuser /backup/
          innobackupex --apply-log --export /backip/2016-12-02_17-06-35
注：参数--export 会为每个 innodb 表的表空间创建一个以.exp 结尾的文件，这些以.exp 结尾的文件则可以用于导入至其它服务器。
```

三、Mysql 小计

1、MyISAM 存储引擎转化为 InnoDB 的方法：

```
[root@node01 ~]#mysqldump -d mydb > mydb_table.sql  #仅仅含有表结构的文件
[root@node01 ~]#mysqldump -t mydb > mydb_data.sql   #仅仅含有表数据的文件
##mysqldump 参数介绍：[-d,--no-data  -t,--no-create-info]
#[-d,--no-data]表示只导出表的结构，不导出数据，
#[-t,--no-create-info]表示只导出数据，不导出表的结构
##在表结构的文件中将存储引擎更换为 InnoDB
[root@node01 ~]#sed -i  's/MyISAM/innoDB/g'  mydb_table.sql
##新建数据库 mydb_new,并导入表结构以及表数据：
```

```
[root@node01 ~]#mysql -e "CREATE DATABASE mydb_new;"
[root@node01 ~]#mysql mydb_new < mydb_table.sql              #导入表结构
[root@node01 ~]#mysql mydb_new < mydb_data.sql               #导入表数据
[root@node01 ~]#mysql
mysql>SHOW CREATE TABLE test.tb1\G #查看表 test.tb1 的创建信息
mysql>SHOW TABLE STATUS LIKE 'tb1'\G        #查看表同步 tb1 的结构信息
```

2、备份和恢复的经验之谈：

（1）备份：将数据和备份放在不同的磁盘设备上；异机或异地备份存储较为理想；备份的数据应该周期性地进行还原测试；每次灾难恢复后都应该立即做一次完全备份；针对不同规模或级别的数据量，要定制好备份策略；二进制日志应该跟数据文件在不同磁盘上，并周期性地备份好二进制日志文件；

```
备份对象：数据文件
          代码：存储过程，存储函数，触发器等
          OS 相关的配置文件，如 crontab 配置计划及相关脚本
          跟复制相关的配置信息
备份方案：①mysqldump+binlog:
          ②lvm2 快照+binlog: ##lvcreate -s -L 1G -p r -n mydata-snap /dev/myvg/mydata ##创建快照
          ③xtrabackup:对 InnoDB 支持热备、完全、增量备份，对 MyISAM 支持温备、完全备份。
二进制日志文件
1 数据恢复或还原的速度
2 数据丢失的允许程度
3 数据的大小与成本的平衡
制定备份计划 备份周期 备份容量 恢复速度评估

对于小型等级业务量的系统来说,直接每天全量备份,定期删除就可以了 50G<
>50G 主从
对于中等级别业务量的系统来说，备份策略可以这么定：第一次全量备份，每天一次增量备份，每周再做一次全量备份，如此一直重复。而对于重要的且繁忙的系统来说，则可能需要每天一次全量备份，每小时一次增量备份，甚至更频繁。为了不影响线上业务，实现在线备份，并且能增量备份，最好的办法就是采用主从复制机制(replication)，在 slave 机器上做备份

出现操作系统崩溃或电源故障时，InnoDB 自己可以完成所有数据恢复工作。应遵从下面的指导：
· 一定用--log-bin 或甚至--log-bin=log_name 选项运行 MySQL 服务器，其中日志文件名位于某个安全媒介上，不同于数据目录所在驱动器。如果你有这样的安全媒介，最好进行硬盘负载均衡(这样能够提高性能)。
· 定期进行完全备份，使用 mysqldump 命令进行在线非块备份。
· 用 FLUSH LOGS 或 mysqladmin flush-logs 清空日志进行定期增量备份。
```

（2）恢复：停止 MySQL 服务器；记录服务器的配置和文件权限；将数据从备份移到 MySQL 数据目录；其执行方式依赖于工具；改变配置和文件权限；以限制访问模式重启服务器；mysqld 的－skip-networking 选项可跳过网络功能；方法：编辑 my.cnf 配置文件，添加选项：skip-networking，socket=/tmp/mysql-recovery.sock；载入逻辑备份（如果有）；而后检查和重放二进制日志；检查已经还原的数据；重新以完全访问模式重启服务器；注释前面在 my.cnf 中添加的选项，并重启；

（3）性能监控工具: dstat(多类型资源统计)、atop(htop/top)、nmon(类 Unix 系统性能监控)、slabtop(内核 slab 缓存信息)、sar(性能监控和瓶颈检查)、sysdig(系统进程高级视图)、tcpdump(网络抓包)、iftop(类似 top 的网络连接工具)、iperf(网络性能工具)、smem)(高级内存报表工具)、collectl(性能监控工具)

参考链接：http://www.178linux.com/999