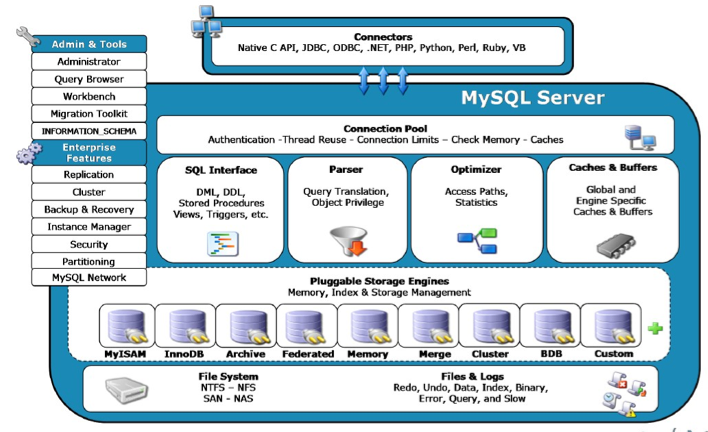
# MySQL背景

MySQL是一个小型关系型数据库管理系统，开发者为瑞典MySQL AB公司。在2008年1月16号被Sun公司收购。而2009年，SUN又被Oracle收购。



# 基本操作

## 常用命令

启动：

Mysqld\_safe&

关闭：

Mysqladmin shutdown –u –p –h

客户端：

mysql –u{用户名} –p{密码} –h {主机IP}

## show常用命令：

数据库清单：show databases;

表清单：show tables;

Show table status;  查看表的引擎

字段清单：show columns from mysql.servers; 等同于desc mysql.servers;

Show full columns from mysql.servers;

索引清单：show index from mysql.servers;

建表SQL:show create table mysql.servers;

数据库状态：Show status;

数据库变量：Show variables

数据库连接：Show processlist

数据库所支持的引擎：Show engines;

存储过程：         show procedure status;

show create procedure {procedure name}

改密码：

update mysql.user set password=PASSWORD('wq1982') where user='root';

flush privileges;

mysqladmin -uroot –p oldpwd –password ‘newpassword’

./mysqladmin -h203.88.193.228 -uroot -p20130505 password 20130505

忘记密码：

# vi /etc/my.cnf   
在[mysqld]的段中加上一句：skip-grant-tables

## 建表SQL：

Create table user(

Name varchar(20) COMMENT '姓名',

Gendor varchar(20) binary COMMENT '性别',

Fee int unsigned COMMENT '费用';

Ctime datetime COMMENT '建立时间'

) engine=myisam default charset=gbk comment '用户表';

InnoDB

**大小写：**

数据库和表名在Linux下面是区分大小写，但字段字却又不同，字段名和字段值都是不区分大小写的．

这样是成立的：create table user(); create table User;

这样就不对了：alter table user add Name; alter table user add name;

Insert into user(name) values(‘a’); Insert into user(name) values(‘A’);

select \* from user where name = ‘a’; # 出来两条记录

如何区分大小写：

Alter table user modify user varbinary(20);

或者: alter table user modify name varchar(20) binary;

改表名：

Alter table user rename user\_t1;

改字段名：

Alter table change name name1 varbiary(20);

**备注COMMENT：**

加备注：

Alter table user modify name varchar(20) comment ‘姓名’;

查询备注：

Show full columns from user;

Show Create table user;

类型：

整型：tinyint, smallint, mediumint, int, bigint [unsigned] 分别对应的字节数：1,2,3,4,8

int(M) 在 integer 数据类型中，M 表示最大显示宽度。  
在 int(M) 中，M 的值跟 int(M) 所占多少存储空间并无任何关系。 int(3)、int(4)、int(8) 在磁盘上都是占用 4 btyes 的存储空间。说白了，除了显示给用户的方式有点不同外，int(M) 跟 int 数据类型是相同的。  
  
如果int的值为10，alter table tbltt modify uid int(6) unsigned zerofill;  
int（10）显示结果为0000000010  
int（3）显示结果为010  
就是显示的长度不一样而已 都是占用四个字节的空间

AUTO\_INCREMENT：到达最大值后，会重复之前的int的最大数值，同时报错。

日期：date,datetime,time,timestamp

**NULL**

**避免使用NULL字段**

不要以为 NULL 不需要空间，其需要额外的空间，并且，在你进行比较的时候，你的程序会更复杂。

对于某些字段类型，NULL值被特殊地处理。如果你将NULL插入表的第一个TIMESTAMP列，则插入当前的日期和时间。如果你将NULL插入一个AUTO\_INCREMENT列，则插入顺序中的下一个数字。

Int类型not null,如果不填值,MySQL会warning,还是会成功给字段一下0的默认值.Varchar 类型not null则会给一个””的默认值.时间会给一个’0000-00-00 00:00:00’默认值.

mysql> insert into t values (1),(2),(NULL),(NULL);

select count(\*) from t; **4条记录**

select count(c1) from t; **2条记录**

select \* from t where c1 =1; **1条记录**

select \* from t where c1 !=1; **还是1条记录**

select \* from t where c1 is null; **原来还有两条NULL记录没有统计,同时该SQL不会使用到索引**

## 索引：

create index idx\_mem\_account on tblmember(account);

alter table user add index idx\_username(username);

drop index accout on tblmember;

alter table user drop index idx\_username;

show indexes from tblmember;

show keys from tblmember;

repair table tblmember;[维护索引]

myisamchk -c -r ../data/dedecmsv4/dede\_archives.MYI [索引文件修复]

explain select name from user where ctime>’2012-02-10’;

以innodb为例，每个page(注意这里是innodb的page,不是linux page)是16K. B+Tree的话那么每层都会存放key. 假设key为16个字节的话，包括overhead 16个字节，那么一个page里面就能够存放512个节点。如果记录界级别在billion级别的话，那么深度在3-4层左右。

估算出层级数目是非常有好处的，可以对query做envelope calculation. 假设不考虑page cache的话，那么查询一条记录通常需要读取3-4次。假设存在cache命中50%的话，那么读取次数在1.5-2次。如果使用MySQL没有缓存层并且都是简单查询的话，要求查询性能在2w/s. 那么要求disk IOPS必须在3w-4w/s上。

## 常见函数

时间函数

date\_format(now(), ‘%Y-%m-%d’)

date\_add(now(), interval 3 day);

# 常见配置

请使用ext4，而不是ext3

**/etc/my.cnf**

[client] [mysqld] [mysqldump] [mysql] [myisamchk] [mysqlhotcopy]

## 客户端配置

[client]

user = root

password = wq1982

host = db.uu1758.com

port = 3306

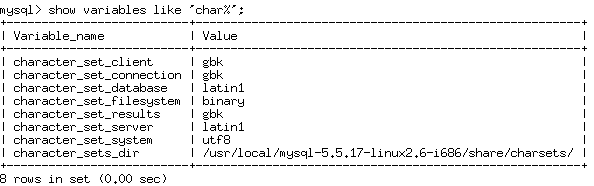
socket = /tmp/mysql.sock

#default-character-set=gbk

[root@host70-151 ~]# mysql

ERROR 1045 (28000): Access denied for user 'root'@'localhost' (using password: NO)

## 多层级字符集管理



建议用UTF-8的字符集.

Linux环境设置默认语言为utf-8即可正常显示.

## 连接

# Try number of CPU's\*2 for thread\_concurrency

thread\_concurrency = 8

max\_connections=20000 /\*最大连接数，需要通过ulimit –n 20000修改系统设置\*/

max\_connect\_errors=50 /\*某一客户端尝试连接此MySQL服务器，但是失败（如密码错误等等）100次，则MySQL会无条件强制阻止此客户端连接。 防止mysql攻击\*/

max\_connections：最大连接数，这个参数实际起作用的最大值（实际最大可连接数）为16384，即该参数最大值不能超过16384，即使超过也以16384为准； 11 1111 1111 = 16384

但实际我看了下代码是



备注：可以在session中设置set GLOBAL max\_connections=1000，但重启后参数将不在生效，需要重新设置．

#空闲连接维持时间

interactive\_timeout = 2678400 #3600\*24\*31 1 month

**unix域或者TCP/IP**

远程登陆：% [任意通配符]，本地：localhost

update mysql.user set host='%' where host='localhost' and user='root';

update mysql.user set host='localhost' where host='%' and user='root';

flush privileges;

**指定IP登录访问**

grant all privileges on \*.\* to 'root'@'192.168.10.%' identified by 'wq1982' with grant option ;

grant all on \*.\* to 'root'@'192.168.10.0/255.255.255.0' identified by 'wq1982' with grant option ;

flush privileges;

【grant option】授权的权限。

grant all privileges on \*.\* to 'root'@'%' identified by 'wq1982' with grant option;

**绑定指定服务器IP**

[mysqld]

bind-addr = 192.168.10.218

**skip\_name\_resolve**

一般情况这个参数都要设置为OFF，在MySQL 5.6版本中，默认值就是OFF。如果这个参数设置为ON，当客户端链接MySQL服务器时，服务器将客户端IP地址反向解析为域名，然后匹配授权配置表，以判断这个连接是否是授权的。DNS反向地址解析一般来说，都是很慢的，所以这会导致客户端连接缓慢，以至连接数积累超限。

关闭这个参数，所有在授权配置表中以域名形式的授权（例如：root@'zeuux.com'）均会失效。

## 内存：

mysql used mem = key\_buffer\_size + query\_cache\_size +

tmp\_table\_size+ InnoDB\_buffer\_pool\_size +

InnoDB\_additional\_mem\_pool\_size+ InnoDB\_log\_buffer\_size+

max\_connections \*

(read\_buffer\_size + read\_rnd\_buffer\_size+ sort\_buffer\_size+ join\_buffer\_size+ binlog\_cache\_size + thread\_stack)

数据库级别：

table\_open\_cache = 128 /\*可缓存的最大表数据\*/

sort\_buffer\_size = 512K /\*排序缓冲区\*/

read\_buffer\_size = 256K /\*读入缓冲区\*/

net\_buffer\_length = 8K /\*TCP/IP缓冲区\*/

max\_allowed\_packet = 1M /\*TCP/IP最大数据包\*/

join\_buffer\_size

MyISAM:

/\*MyISAM只会缓存索引，它不会缓存其他的字段数据\*/

key\_buffer\_size = 256M

show status like ‘%key%’

| key\_buffer\_size | 268435456 |

| key\_cache\_block\_size | 1024 |

show variables like ‘%key%’;

| Key\_blocks\_unused | 214247 |

| Key\_blocks\_used | 85164 |

| Key\_read\_requests | 17447327 |

| Key\_reads | 118439 |

| Key\_write\_requests | 2903676 |

| Key\_writes | 2899438 |

InnoDB

# of RAM but beware of setting memory usage too high

#缓存索引和数据，写入缓存等

#InnoDB\_buffer\_pool\_size =512M

#InnoDB用来存储数据字典和其他内部数据结构的内存池大小

#InnoDB\_additional\_mem\_pool\_size = 32M

#日志缓存

#InnoDB\_log\_buffer\_size = 32M

如果直接修改my.cnf中的innodb\_log\_file\_size配置，然后重启，innodb会加载失败。安全的方法是停止mysql服务，然后删除ib\_logfile0和ib\_logfile1，再重启mysql，ib\_logfile[01]文件便会重建

**innodb\_flush\_logs\_at\_trx\_commit=2**

这个参数用于控制在用户发出提交(Commit命令)时, MySQL如何处理日志缓冲中的内容. 在默认情况下, MySQL会每秒种将日志文件强行刷新到磁盘, 由于日志文件还受操作系统文件缓冲的控制, 因此在写时就可以分为两步了, 第一步是写入日志文件, 写到文件缓冲就算成功, 第二步将日志文件刷到磁盘, 真正写入磁盘. 这个值有三个值可以选:

0 : 用户提交时不做动作, 由MySQL每秒种写入日志文件并和磁盘同步.

1 : 提交时写日志文件, 并将日志文件和磁盘同步, Oracle的默认值.

2 : 提交时写日志文件, 但和磁盘的同步交给MySQL来每秒种进行.

**innodb\_flush\_method=O\_DIRECT**

**innodb\_flush\_method究竟应不应该使用O\_DIRECT？**

所有MySQL调优的建议都说，如果硬件没有预读功能，那么使用O\_DIRECT将极大降低InnoDB的性能，因为O\_DIRECT跳过了操作系统的文件系统Disk Cache，让MySQL直接读写磁盘了。

但是在我的实践中来看，如果不使用O\_DIRECT，操作系统被迫开辟大量的Disk Cache用于innodb的读写缓存，不但没有提高读写性能，反而造成读写性能急剧下降。而且buffer pool的数据缓存和操作系统Disk Cache缓存造成了Double buffer的浪费，显然从我这个实践来看，浪费得非常厉害。

说O\_DIRECT造成MySQL直接读写磁盘造成得性能下降问题，我觉得完全是杞人忧天。因为从JavaEye的数据库监测来看，Innodb的buffer pool命中率非常高，有98%以上，真正的磁盘操作是微乎其微的。为了1%的磁盘操作能够得到Disk Cache，而浪费了98%的double buffer内存空间，无论从性能上看，还是从内存资源的消耗来看，都是非常不明智的。

innodb\_flush\_method有三个值，分别是fdatasync，O\_DSYNC和O\_DIRECT，其中fdatasync是默认值。它们控制了InnoDB刷新日志和数据的模式。

查询状态：

show engine innodb status;

参考：

MySQL InnoDB性能调整的一点实践

<http://robbin.iteye.com/blog/461382>

## 日志文件

日志文件的作用：

1. 恢复数据用
2. 双机热备用，主从数据库之间同步数据，当数据量很大时，mysqldump的性能会跟不上，我们要采用主从ＤＢ方式，同时也可以减轻生产机的查询负担，同时也增加了ＤＢ的健壮性，大致流程：主ＤＢ发送日志文件到从ＤＢ，从ＤＢ将日志写中继日志文件，从ＤＢ再通过中继日志文件更新从ＤＢ。

# Replication Master Server (default)

# binary logging is required for replication

log-bin=mysql-bin

如果需要关闭日志直接将该选项屏蔽，则mysql不会再生成日志文件。

同时也可以进入系统将现有的日志文件进行清除：

mysq > reset master;

查看相关信息：

show master status;

mysqlbinlog [日志名]

与redo日志的区别：

重做日志（redo）记录的是事务的信息（InnoDB引擎），而二进制日志文件记录的是数据库的所有变更信息。在InnoDB下，先写redo，再写数据文件，主要用于InnoDB的事务处理。bin\_log对执行完成的事务记录数据变化，供复制及恢复用，二进制是通用性与引擎无关，它支持所有MySQL引擎。

redo日志：ib\_logfile0

二进制日志：mysql-bin.000001

## 慢SQL查询

long\_query\_time = 0.1 #100毫秒 系统默认为10秒

slow\_query\_log = /usr/local/mysql/var/slowquery.log #日志目录

也可以记录没有用到索引的查询：

log-queries-not-using-indexes

查询是否有效．Show variables like ‘%quer%’

慢查询分析：

Mysqldumpslow {logfile} –s t –t 2

-s, 是表示按照何种方式排序，c、t、l、r分别是按照记录次数、时间、查询时间、返回的记录数来排序，ac、at、al、ar，表示相应的倒叙；

-t, 是top n的意思，即为返回前面多少条的数据；

-g, 后边可以写一个正则匹配模式，大小写不敏感的，查询具体某个表的情况；

或者第三方msyqlsla {logfile}

http://blog.csdn.net/wyzxg/article/details/7268969

show processlist也可以发现一些非常明显的慢SQL。

## 查询优化

默认方式下该功能是关闭的：

mysql> select @@profiling;

+-------------+

| @@profiling |

+-------------+

| 0 |

+-------------+

1 row in set (0.00 sec)

打开profile功能

mysql> set profiling =1;

Query OK, 0 rows affected (0.00 sec)

mysql> select \* from tblgamesync where account='2632568';

Empty set (0.00 sec)

mysql> show profiles;

+----------+------------+---------------------------------------------------+

| Query\_ID | Duration | Query |

+----------+------------+---------------------------------------------------+

| 1 | 0.00071000 | select \* from tblgamesync where account='2632568' |

+----------+------------+---------------------------------------------------+

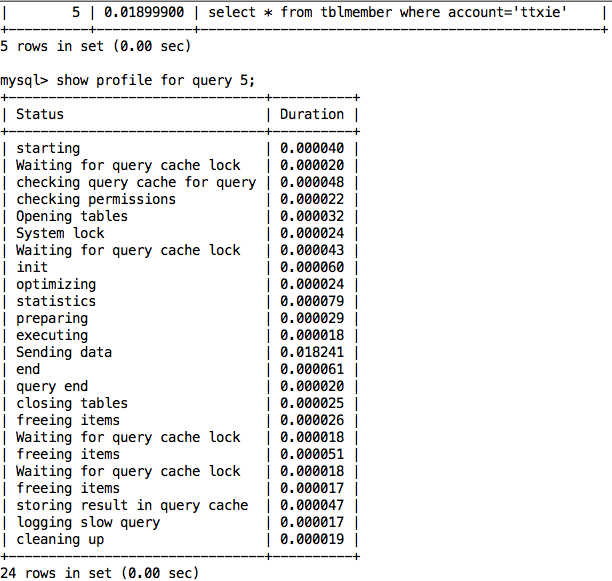
1 row in set (0.00 sec)

通过指定的Query\_ID 来查询指定的sql语句的执行信息：

mysql> **show profile for query 1;**

+--------------------------------+----------+  
| Status                         | Duration |  
+--------------------------------+----------+  
| starting                       | 0.000028 |  
| checking query cache for query | 0.000008 |  
| checking privileges on cached  | 0.000009 |  
| sending cached result to clien | 0.000023 |  
| logging slow query             | 0.000004 |  
| cleaning up                    | 0.000003 |  
+--------------------------------+----------+  
6 rows in set (0.00 sec)

mysql> show profile cpu,block io for query 1;



索引使用情况：

explain select \* from tblgamesync where account=’xxx’;

# 备份

## Mysqlhotcopy(物理备份)

该备份方案只适用于MyISAM的数据库表，对于InnoDB的表是不能用该方法去操作，同时该操作必须在数据库所在的服务器上面才可以完成该操作．

操作流程：对全表做只读锁定――＞做文件复制――＞解锁

.frm：数据定义文件

.MYD：数据文件

.MYI：索引数据文件

mysqlhotcopy -u root -p wq1982 -h db.uu1758.com gm\_bbs .  
 -u:用户名

-p:密码

-h:主机名

{db\_name}：具体的数据库名

{path}:备份路径．

在恢复过程中，需要注意：

chmod 0660 \*修改权限

chown mysql:mysql \* mysql权限．

## Mysqldump(逻辑备份)

数据库全备份

mysqldump -u root -pwq1982 -h 220.231.142.242 gm\_boss tblmember tblmember\_server tblgame tblbosslogin tblserver tblsource tblghserver> bk.`date +%Y%m%d`.sql

指定备份某个数据库

mysqldump –uroot –psss –h11.1.1.11 {db\_name} > bk.sql

指定备份某个数据库，只导出表结构

mysqldump -d –uroot –pwq1982 –h220.231.142.242 gm\_boss tblmemberlogin > bk.sql

-d只是结构不用数据

只导出存储结构/function：

mysqldump -uroot -p20130505 -h192.168.1.24 -P3306 -n -d -t -R condor\_data > p\_condor\_data.sql

压缩：

mysqldump –uroot –psss –h11.1.1.11 {db\_name}|gzip > bk.sql.gz

gzip –d bk.sql.gz

数据恢复：

Mysql 之后再source执行就行

或者 mysql {db\_name} < bk.sql

## xtrabackup

使用mysqldump来进行备份与恢复，在备份的时候锁住表，然后全部备份，在数据少的时候没问题，但如果数据很多，不允许锁表，同时需要恢复数据块的情况，mysqldump就不适合了，我在恢复一个4G数据文件的数据库的时候，恢复的数据是使用mysqldump的数据，恢复了3个小时还没有反应，造成的影响很严重，所以我开始寻找其他的别发软件来满足以上的需求，幸好找到了，就是使用xtrabackup来进行备份与恢复，恢复4G数据文件的数据库，仅需要14秒，同时在备份的时候不会锁表，而且支持增量备份.

xtrabackup可以在不加锁的情况下备份innodb数据表，不过此工具不能操作myisam。也可以备份xtraDB

innobackupex-1.5.1是一个脚本封装，能同时处理innodb和myisam，但在处理myisam时需要加一个读锁。

innobackupex [--defaults-file=/etc/my.cnf] –user=root [--host=192.168.10.198] [--password=xxx] [--port=3306] /www/backup/ 2>/www/backup/backup.log

安装文档：

<http://www.percona.com/doc/percona-xtrabackup/2.1/installation/compiling_xtrabackup.html>

## 记录集导入导出

导出：

mysql> select \* into outfile “/home/mysql/bk.out” from user;

写出文件要求，权限足够，导出文件各字段值会以TAB分隔．

SELECT a,b,a+b INTO OUTFILE '/tmp/result.txt'

FIELDS TERMINATED BY ',' OPTIONALLY ENCLOSED BY '"'

LINES TERMINATED BY '\n'

FROM test\_table;

导入：

mysql> load data infile '/home/mysql/user.bk' into table user

fields terminated by '\t';

通过自定义分隔符来分隔几字段，如此可以方便的把其他数据导入数据库中．

mysql -h log1.kdb temp -uipoxx -pxxx -e "select \* from tt" -N > tt.sql

#mysql --socket=/data/infobright/mysql-ib.sock -pipxxx fb -e "load data local infile 'tt.sql' into table tt"

mysql -h 183.63.52.218 99\_boss -uroot -pwq1982 -e "select \* from tblpayrecord where ctime>'2014-10-17'" -N > payrcord.sql

mysql

load data local infile 'payrcord.sql' into table tblpayrecord;

# 主从复制

http://hatemysql.com/tag/sync\_binlog/

### binlog & relaylog

Binlog分类：STATEMENT，ROW，MIXED，对应的复制模式：

– 基于SQL语句的复制(statement-based replication, SBR)，

– 基于行的复制(row-based replication, RBR)，

* 混合模式复制(mixed-based replication, MBR)。

Statement:文件简短，记录更新时的SQL语句，但是复制时uuid()唯一码，auto\_increment自动增长，sysdate(),user()的生成，就会出现不一致。

ROW：直接复制文件内容，文件很大，复制时网络负担大,建议启用压缩。

MIX:以Statement为主要记录方式，出现statement可能出现误差的地方采用ROW方式记录。

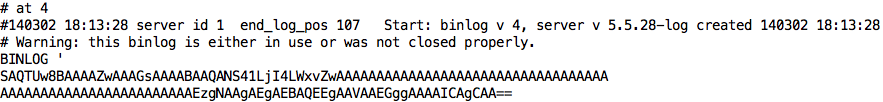
日志内容查看：

show binlog events in 'mysql-bin.000004' limit 10;

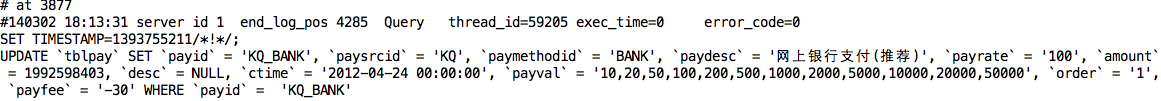
mysqlbinlog [binlogfile]

mysqlbinlog --start-datetime='2014-03-02 00:00:00' mysql-bin.000084|more

RBR:



SBR:



        整体上来说，复制有3个步骤：

       (1)    master将改变记录到二进制日志(binary log)中（这些记录叫做二进制日志事件，binary log events）；  
       (2)    slave将master的binary log events拷贝到它的中继日志(relay log)；  
        (3)    slave重做中继日志中的事件，将改变反映它自己的数据。

下图描述了复制的过程：



mysql实现上是所有操作都会写到binlog里面，然后slave有一个专门的io线程不断地从master binlog里面取出增量数据，写到本地的relay-log.同时slave本地有一个执行线程，将这些realy-log执行修改自己的数据库，达到同步的目的。relay-log里面的内容和master binlog内容每条记录都是完全相同的，最后进入slave binlog记录和master binlog对应记录也是一样的。执行id是master id,执行时间是master binlog记录的时间，本地slave是不会进行任何修改的。

MySQL主从复制通常采用单主星型结构，如A-->B、A-->C；当然，也可以采用链式结构，如A-->B-->C。

从服务器可以设置为只读，以避免误操作造成的不一致性，也因此可以将数据库查询、更新分离（A可读可写，B/C只读）

存在滞后问题。

操作流程：

<http://blog.chinaunix.net/uid-20164485-id-3290661.html>

http://wangwei007.blog.51cto.com/68019/965575

http://faq.comsenz.com/library/system/serviceext/serviceext\_slave.htm

修改主库mysql配置配置文件，在[mysqld]段添加以下内容：

server-id = 1

log\_bin=/home/mysql/logs/binlog/bin-log

max\_binlog\_size = 500M

binlog\_cache\_size = 128K

binlog-do-db = adb

binlog-ignore-db = mysql

log-slave-updates

expire-logs-days = 90 //只保留7天的二进制日志，以防磁盘被日志占满

binlog\_format="ROW" [MIXED执行SQL会修改到其他的数据，不适合M-M结构]

**#max master host:10**

**auto\_increment\_increment = 10**

**auto\_increment\_offset = 1**

sync\_binlog=1  #binlog更新的时候写入磁盘

GRANT REPLICATION SLAVE ON \*.\* TO '帐号'@'从服务器IP' IDENTIFIED BY '密码';

GRANT REPLICATION SLAVE ON \*.\* TO 'rep'@' 113.107.205.23' IDENTIFIED BY 'rep1234';

在主MySQL服务器上执行命令，把数据库设置成只读状态：

FLUSH TABLES WITH READ LOCK;

执行命令，并且记下file及position的值：

show master status;

备份需要做主从备份的数据库，用导出成SQL或者直接复制数据库文件方式都可以

回到MYSQL命令行窗口，解封数据库只读状态，执行：

UNLOCK TABLES;

1.修改从库mysql配置配置文件，在[mysqld]段添加以下内容：

server-id=2 //服务器ID不能重复

master-port=3306 //主库的端口

master-connect-retry=30

replicate-do-db=dzx2 //需要做复制的数据库名

replicate-ignore-db = mysql

replicate-ignore-table=dzx2.pre\_common\_session //自动跳过的表，session表没必要做复制

slave-skip-errors = 1032,1062,126,1114,1146,1048,1396 //自动跳过的错误代码，以防复制出错被中断 #其中1062为主键重复错误。

master-info-file = /usr/local/mysql/data/master.info

relay-log = /home/mysql/logs/relay-bin

relay-log-index = /home/mysql/logs/relay-bin.index

relay-log-info-file = /home/mysql/logs/relay-log.info

2、将主库上备份的数据库恢复到从库

3、重启从库MYSQL

4、登录从库的MySQL命令行，执行：

change master to master\_host='192.168.10.218',master\_port=3306, master\_user='rep', master\_password='rep1234', master\_log\_file='mysql-bin.000002', master\_log\_pos=120;

//设置连接信息，file及position的值是之前记录下来，position的值没有单引号，其他的值要单引号

5、执行：

start slave; //启动从库连接

6、查看从库状态：

show slave status\G; //查看连接情况，是不是两个YES，两个YES为成功

server\_uuid：服务器身份ID。在第一次启动Mysql时，会自动生成一个server\_uuid并写入到数据目录下auto.cnf文件里，官方不建议修改。

相关参数：

**为了保证事务InnoDB复制设置的最大可能的耐受性和一致性，应在主服务器的my.cnf文件中使用innodb\_flush\_log\_at\_trx\_commit=1和sync-binlog=1。**

--log-slave-updates=1 #//记录从库更新，以便允许链式复制

--read-only #//将从库设为只读，仅允许从服务器线程或具有SUPER权限的用户执行

--slave\_compressed\_protocol=1 #//复制过程启用压缩，若启用此项，主、从服务器都应该添加。跨集群配置复制的时候,可启用–slave\_compressed\_protocol=1压缩传输数据.

--skip-slave-start #//服务器启动时跳过复制，需要手动 SLAVE START

--master-connect-retry=60 #//当主服务器连接丢失时，再次重试之前等待的时间（默认为60秒）

**双master的数据一致性风险：**

如果master之间网络断开，双方处于stop slave状态。期间双方相同记录发生修改，数据将出现不一致问题，双方都会执行对方的变动，而不会按照双方的综合时序执行。

如，server-id=1修改字段account为’ttxie1’

server-id=2修改字段account为’ttxie2’

双方恢复slave后,执行relog之后变成：

server-id=1字段account为’ttxie2’

server-id=2字段account为’ttxie1’

双方分别执行了对方的变动SQL。

在执行Master-Master方案时，不能发生交叉，需要做好垂直分割，如按照区域进行分割，华东区域，华南，华北……或者按平台进行分割，99yx分配一个数据库，uu1758.com/其他平台分配一个数据库，再或者执行读写分离。

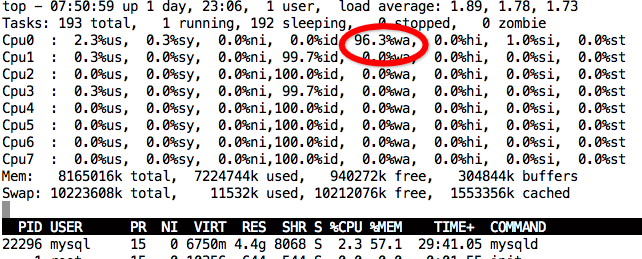
从库要保护好，无法设置read\_only，设置后依然可以修改。

set global slave\_net\_timeout=30; //超时连接时间，系统自动重新连接

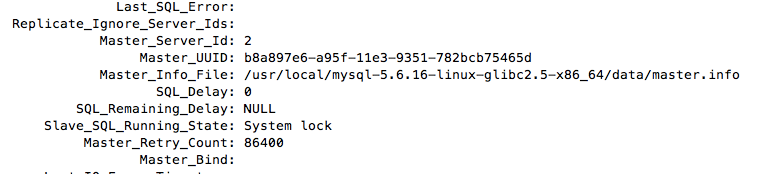
必须重启后才生效。

性能问题：

slave 单线程重写非常慢，4天数据18小时都没有执行。



**System Lock案例**



系统锁，表阻塞

* 处理方法一：

start slave;

　　回到主库

　　unlock tables; 解锁

　　回到从库 查看

　　show slave status \G;

* 方法二：

stop slave;

　　set global sql\_slave\_skip\_counter=1; （1是指跳过一个错误）

　　start slave;

　　再show slave status \G;查看

* 方法三(慎重)：

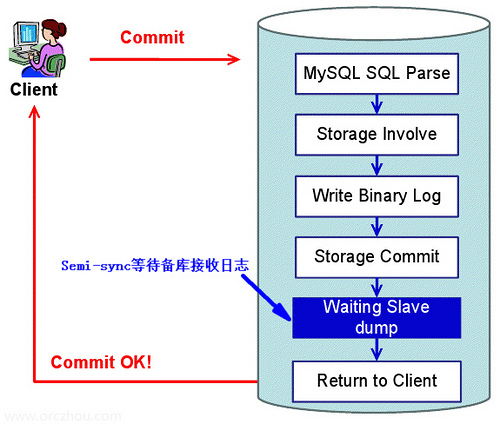
跳过当前文件：

change master to master\_host='183.63.52.218',master\_port=3306, master\_user='rep', master\_password='rep1234', master\_log\_file='mysql-bin.000035', master\_log\_pos=286402025;

### 半同步(semi-sync)

* <http://www.db110.com/?p=3364>
* <http://code.google.com/p/google-mysql-tools/wiki/SemiSyncReplication>
* <http://code.google.com/p/google-mysql-tools/wiki/SemiSyncReplicationDesign>
* <http://dev.mysql.com/doc/refman/5.5/en/replication-semisync.html>
* http://www.orczhou.com/index.php/2011/07/why-and-how-mysql-5-5-semi-sync-replication/

很早之前做了一个mysql集群主从切换模块，里面就涉及到了半同步。按照半同步的定义（全同步的语义应该就是等待所有的slave都同步完成，强一致性），半同步还是会存在丢数据的可能，半同步的语义仅仅是认为一个slave同步到数据之后的话同步就完成。但是如果master挂掉同时slave也挂掉（或者是没有等其他slave补齐数据的话），那么就会存在数据丢失的可能（仅仅是提供最终一致性可能）。

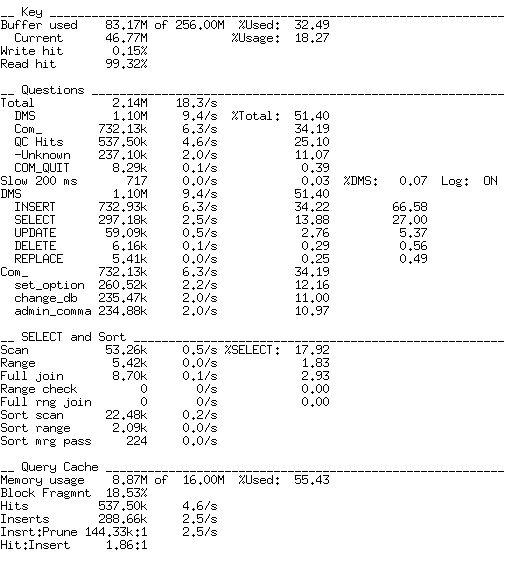


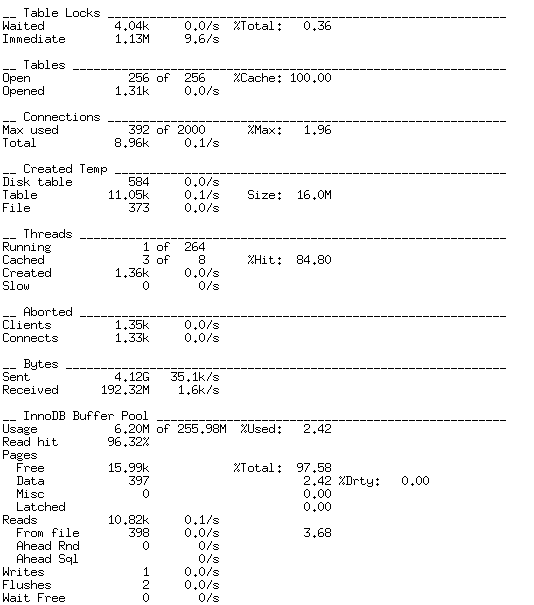
备库Crash时，主库会在某次**等待超时**后，关闭Semi-sync的特性，降级为普通的异步复制，这种情况比较简单。可以做多个备库，任何一个备库接收完成日志后，主库就可以返回给客户端了。

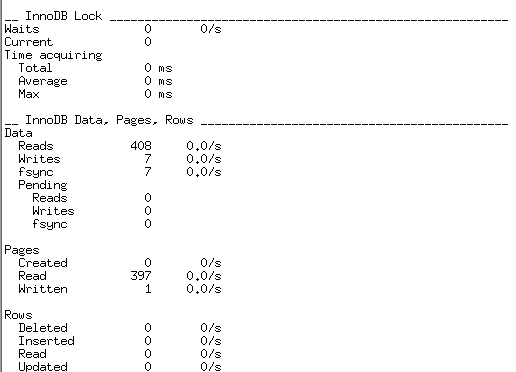
半同步在MySQL 5.5才开始提供，并且可能引起并发和效率的一系列问题，比如只有一个备库，备库挂掉了，那么主库在事务提交等待10秒(rpl\_semi\_sync\_master\_timeout控制)后，才会继续，之后变成传统的异步方式。**所以目前在生产环境下使用半同步的比较少。**

# MySQLReport

http://jinyan2049.blog.51cto.com/881440/928828







# MyISAM与InnoDB:

MySQL的数据库引擎有很多，如：MyISAM，InnoDB，CVS，HEAP，MEMORY等．．．

最大的区别：事务处理，日志处理，数据完整性，表锁

MyISAM：不支持事务，容错性差．

MyISAM比InnoDB快，是有条件的，像我们的服务器都是十几G甚至几十G内存的情况下，InnoDB远比MyISAM快，内存可以Cache大部分数据，大部分查询都在内存中完成。MyISAM只能缓存索引，并不能有效利用大内存。

myisam数据量上来之后可能经常得repair，修复时间也相对较长．InnoDB的修复时间就相对快不少．

一个数据库 一般以查询和插入为主 有比较少的更新和删除操作的推荐使用MyISAM.如果对事务要求高,有并发性要求高的，有大量更新和删除的，就推荐使用InnoDB

修改数据库表引擎：

alter table user type=innodb;

alter table tblmember engine innodb;

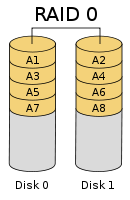
innodb数据文件：ibdata1 .frm

修改了innodb\_log\_file\_size参后，启动下面报错，然来是改了这个参数后需要关闭MySQL并删除ib\_logfile0, ib\_logfile1这些文件，再启动MYSQL．

INNODB在做SELECT的时候，要维护的东西比MYISAM引擎多很多；  
1）数据块，INNODB要缓存，MYISAM只缓存索引块， 这中间还有换进换出的减少；   
2）**innodb**寻址要映射到块，再到行，MYISAM 记录的直接是文件的OFFSET，定位比INNODB要快  
3）INNODB还需要维护MVCC一致；虽然你的场景没有，但他还是需要去检查和维护

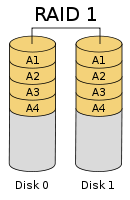
# RAID选择:

Raid 0:速度最快,将相邻的数据,并行分别写入到不同的磁盘上,性能最佳,但没有冗余,安全性差.

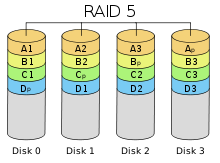


不用Raid卡，SATA硬盘+华硕P5Q主板自带Raid。  
Q9400CPU/8G内存/希捷双500G HDD  
HD Tune测试，不用Raid0的时候120/130M左右，用Raid的时候是200M左右。

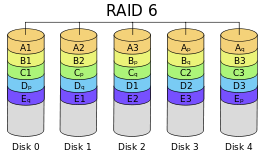
Raid 1:安全性最佳,直接对数据进行镜像,1对1 的数据安全备份.



Raid 5:N+1的方案,在rai0raid1之间一个折中的一个办法,读取速度非常快,数据和raid0 类似,相邻数据写入到不同磁盘。 RAID 5的磁盘空间利用率要比RAID 1高，存储成本相对较便宜；但写入的时候要做一下奇偶校验位的运算,相对就慢一点

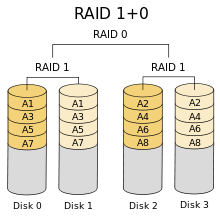


Raid6:raid5升级版,N+2的方案.数据更安全.



**RAID1+0**

OLTP最好选择Raid1+0.读写数据的性能有保证,同时安全性也是最高.只是利用率稍差.





参考：

MySQL的InnoDB和MyISAM对比及优化

<http://www.51testing.com/?uid-174101-action-viewspace-itemid-218184>