**基本操作**

1. **备份、导入**

Mysqldump不会备份库中的空表，遇到空表就会立即停止备份并发出错误警告。以下是忽略指定表而备份整个库的实例：

mysqldump --ignore-table=ipay130.待审核的个人用户 --ignore-table=ipay130.待审核的企业用户 -h 127.0.0.1 -uroot -pwotltd.2125 ipay130 > ipay130.sql

远程备份数据库，备份脚本如下：

#!/bin/bash

user=backup

password=backup

backdir="/home/data"

db9="10.1.1.9"

date=`date +%y%m%d.%H%M`

yestdate=`date +%y%m%d --date="-2 day"`

/usr/bin/mysqldump -h ${db9} -u${user} -p${password} ipay\_alpha > /home/data/${date}-ipay\_alpha.sql

rm /home/data/${yestdate}\*

**导入库**

如果导入是新库，需要在mysql中创建该库：

CREATE DATABASE 库名;

mysql -u root -p 库 < 库.sql

导出test库的m\_menu表

mysqldump -uroot -pltd@YCK2015\* test m\_menu > test.menu.sql

将上面导出的m\_menu表导入到新库menu

mysql -uroot -pltd@YCK2015\* menu < test.menu.sql

shell查看mebu库的m\_menu表的结构

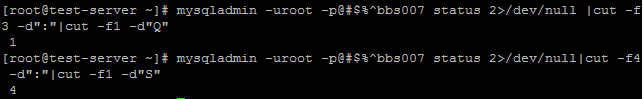
mysql -uroot -pltd@YCK2015\* -e "desc menu.m\_menu"

明文密码操作屏蔽warning信息

Zabbix监控mysql，需要输入明文用户名密码，屏蔽Warning: Using a password on the command line interface can be insecure.用2>/dev/null



mysqladmin -uroot -p@#$%^bbs007 status 2>/dev/null |cut -f3 -d":"|cut -f1 -d"Q"



为了更快的进行导入，在导入时临时禁用外键约束、临时禁用唯一性检测.

1. **用户管理**

创建并授权test@bz\_mysql从任意主机登录，并设置密码

create user test@bz\_mysql;

grant all on \*.\* to 'test@bz\_mysql'@'%';

UPDATE mysql.user SET Password=PASSWORD('test\_BZ135mysql') where User='test@bz\_mysql';

FLUSH PRIVILEGES;

远程登陆mysql，在安装了mysql（客户端）的主机上：

mysql -h 101.251.205.164 -P 3306 -u root -p

指定远程主机要小写，指定端口要大写

修改密码

UPDATE mysql.user SET Password = PASSWORD('wotltd.2125') where Host='10.192.168.16';

FLUSH PRIVILEGES;

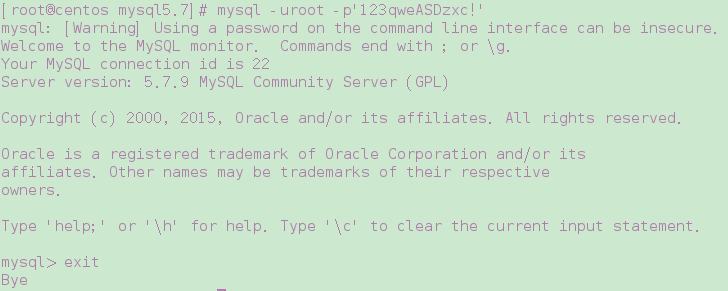
上如果面的方法不行

mysqladmin -u root -p123456 password @#$%^bbs007

小技巧：

含有特别字符的密码，不能直接跟在-p后面，需要加上单引号

mysql -uroot -p'123qweASDzxc!'



mysql是通过User表，Db表，Host表，Tables\_priv 表，Columns\_priv 表这5张表实现用户权限控制，均可以通过直接对这些表的操作以达到对用户的管理

权限回收

revoke delete on test.\* from admin@'localhost';

nagios监控系统使用nagios连接数据库，权限只有select,SUPER, REPLICATION CLIENT

UPDATE mysql.user SET password=password("Za+Na668#U!") WHERE user='Monitor';

delete User from mysql.user where User="Monitor" and Host="%";

grant select,SUPER, PROCESS on \*.\* to Monitor@'10.160.59.68';

flush privileges;

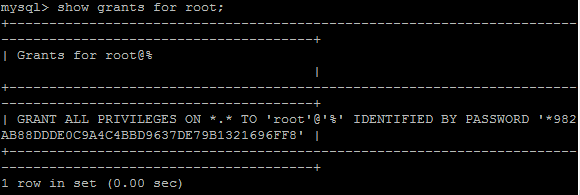
授权zabbix用户执行show engine innodb status\G命令

grant select,SUPER, PROCESS on \*.\* to zabbix@'localhost';

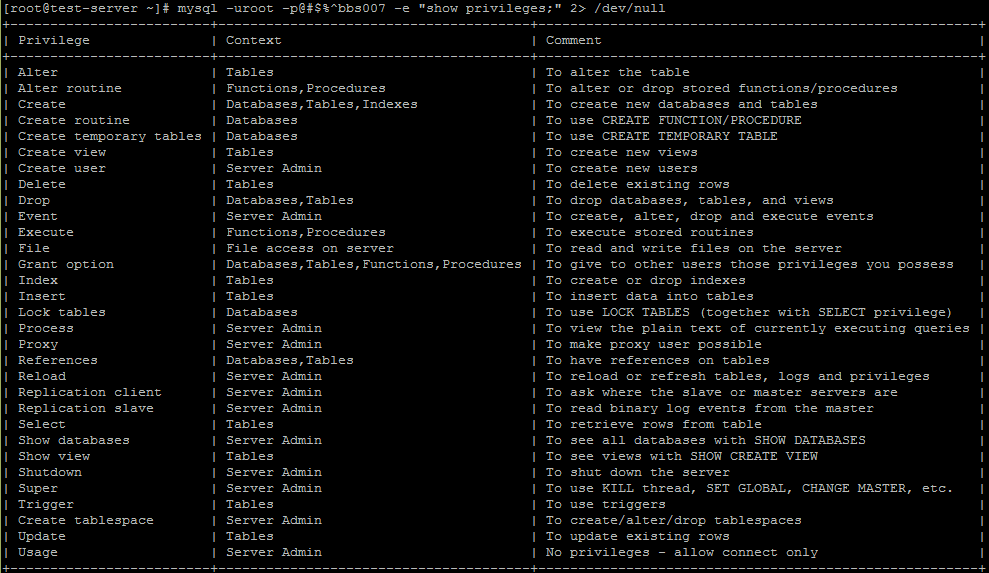
FLUSH PRIVILEGES;

查看用户的权限

show grants for root;

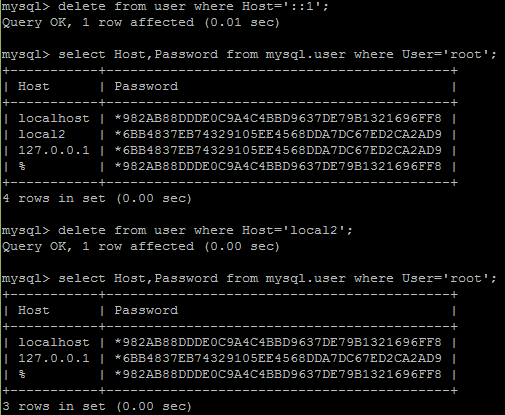


show privileges;显示服务器所支持的不同权限



删除Host为IP v6的记录

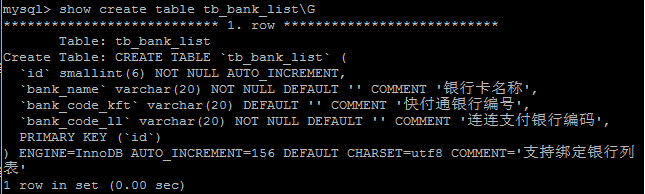
delete from user where Host='::1';



1. 基础SQL语句

show create database database\_name;显示创建指定数据库的SQL语句

show create table table\_name;显示创建指定数据表的SQL语句

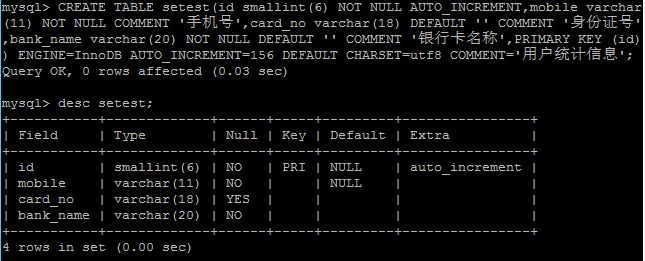


CREATE TABLE

在shell命令行下建表，不能给表和字段加单引号

CREATE TABLE shop (article INT(4) UNSIGNED ZEROFILL DEFAULT '0000' NOT NULL,dealer CHAR(20) DEFAULT '' NOT NULL,price DOUBLE(16,2) DEFAULT '0.00' NOT NULL,PRIMARY KEY(article,dealer));

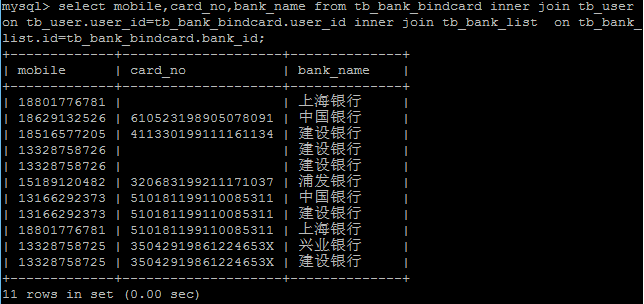
CREATE TABLE setest(id smallint(6) NOT NULL AUTO\_INCREMENT,mobile varchar(11) NOT NULL COMMENT '手机号',card\_no varchar(18) DEFAULT '' COMMENT '身份证号',bank\_name varchar(20) NOT NULL DEFAULT '' COMMENT '银行卡名称',PRIMARY KEY (id)) ENGINE=InnoDB DEFAULT CHARSET=utf8 COMMENT='用户统计信息';



INNER JOIN

三表连查，tb\_bank\_bindcard的桥梁，要取出字段的值在tb\_user和tb\_bank\_list的表里

select mobile,card\_no,bank\_name from tb\_bank\_bindcard inner join tb\_user on tb\_user.user\_id=tb\_bank\_bindcard.user\_id inner join tb\_bank\_list on tb\_bank\_list.id=tb\_bank\_bindcard.bank\_id;



INSERT INTO

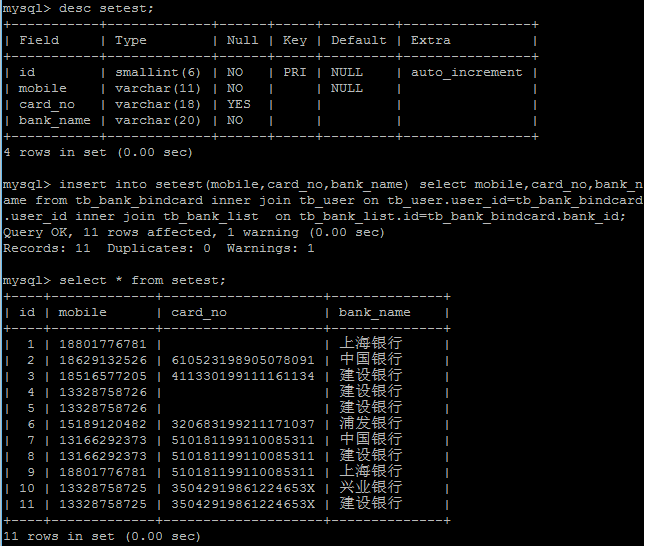
表中插入数据，商品1和3分别有两个供应商：

INSERT INTO shop VALUES(1,'A',3.45),(1,'B',3.99),(2,'A',10.99),(3,'B',1.45),(3,'C',1.69),(3,'D',1.25),(4,'D',19.8);

INSERT INTO SELECT

3表查询的结果，然后插入新表

insert into setest(mobile,card\_no,bank\_name) select mobile,card\_no,bank\_name from tb\_bank\_bindcard inner join tb\_user on tb\_user.user\_id=tb\_bank\_bindcard.user\_id inner join tb\_bank\_list on tb\_bank\_list.id=tb\_bank\_bindcard.bank\_id;

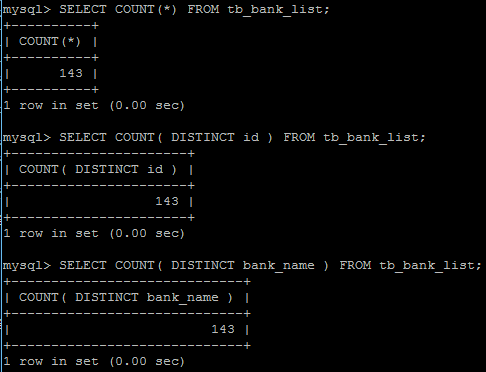


SELECT COUNT DISTINCT

统计数据总行数SELECT COUNT(\*) FROM tb\_bank\_list;

按照索引去统计

SELECT COUNT( DISTINCT id ) FROM tb\_bank\_list;



MAX()

最贵的商品价格是多少：

SELECT MAX(price) FROM shop;

GROUP BY

价格最贵的商品名称是什么：

1. 通过子查询实现：

SELECT article,dealer,price FROM shop WHERE price=(SELECT MAX(price) FROM shop);

1. 通过ORDER BY … DESC降序排序和LIMIT子句只得到第一行：

SELECT article,dealer,price FROM shop ORDER BY price DESC LIMIT 1;

每种商品最高的价格是多少：

SELECT article,MAX(price) AS price FROM shop GROUP BY article;

语句分析： MAX()只能找出一列中的最大值，BROUP BY找出一列中重复的值的第一个：

各种商品中最贵价格的，对应的供应商分别是谁：

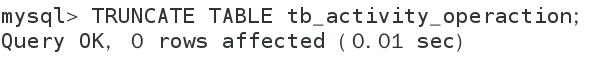
SELECT article,dealer,price FROM shop s1 WHERE price=(SELECT MAX(s2.price) FROM shop s2 WHERE s1.article = s2.article);

找出价格最高或最低的商品：

SELECT \* FROM shop WHERE price=@min\_price OR price=@max\_price;

清空表数据

TRUNCATE TABLE '表名'



根据两个键搜索

寻找两个通过OR组合到一起的关键字：

SELECT field1\_index, field2\_index FROM test\_table WHERE field1\_index = '1' OR field2\_index = '1'

还可以使用UNION将两个单独的SELECT语句的输出合成到一起:

SELECT field1\_index, field2\_index FROM test\_table WHERE field1\_index = '1' UNION SELECT field1\_index, field2\_index FROM test\_table WHERE field2\_index = '1';

查询建议

1.避免在整个表上使用count(\*),它可能锁住整张表.

2. 使查询保持一致以便后续相似的查询可以使用查询缓存.

3. 在适当的情形下使用GROUP BY而不是DISTINCT.

4. 在WHERE, GROUP BY和ORDER BY子句中使用有索引的列.

5. 有时候MySQL会使用错误的索引,对于这种情况使用USE INDEX.

6. 对于记录数小于5的索引字段，在UNION的时候使用LIMIT不是用OR.

7. 为了避免在更新前SELECT，使用INSERT ON DUPLICATE KEY或者INSERT IGNORE ,不要用UPDATE去实现.

8. 不要使用 MAX,使用索引字段和ORDER BY子句.

9. 避免使用ORDER BY RAND().

10. LIMIT M，N实际上可以减缓查询在某些情况下，有节制地使用.

11. 在WHERE子句中使用UNION代替子查询.

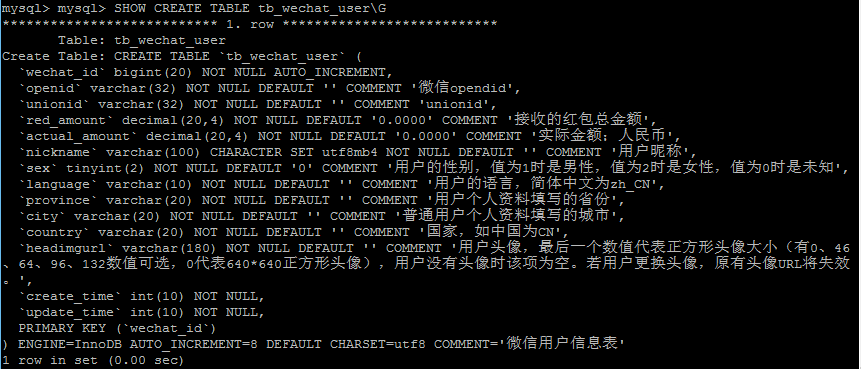
12. 对于UPDATES（更新），使用 SHARE MODE（共享模式），以防止独占锁.

13. 使用DROP TABLE，CREATE TABLE DELETE FROM从表中删除所有数据.

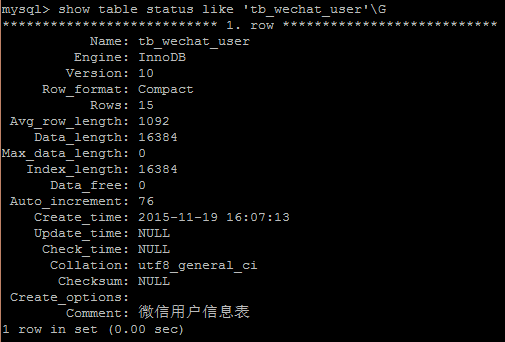
14. 考虑持久连接，而不是多个连接，以减少开销.

表结构

SHOW CREATE TABLE 表名称; 查看表结构



show table status like 'tb\_wechat\_user'\G查看表状态



1. 修改字符集

默认创建的表不支持中文字符，所以需将表编码设置为utf8

ALTER TABLE tb1 CONVERT TO CHARACTER SET UTF8;



1. 索引 index

索引是一种特殊的文件(InnoDB数据表上的索引是表空间的一个组成部分)，它们包含着对数据表里所有记录的引用指针。与MyISAM数据表相比，索引对InnoDB数据的重要性要大得多。在InnoDB数据表上，索引对InnoDB数据表的重要性要在得多。在 InnoDB数据表上，索引不仅会在搜索数据记录时发挥作用，还是数据行级锁定机制的苊、基础。”数据行级锁定”的意思是指在事务操作的执行过程中锁定正 在被处理的个别记录，不让其他用户进行访问。这种锁定将影响到(但不限于)SELECT…LOCK IN SHARE MODE、SELECT…FOR UPDATE命令以及INSERT、UPDATE和DELETE命令。

出于效率方面的考虑，InnoDB数据表的数据行级锁定实际发生在它们的索引上，而不是数据表自身上。显然，数据行级锁定机制只有在有关的数据表有一个合 适的索引可供锁定的时候才能发挥效力。

索引不是万能的！索引可以加快数据检索操作，但会使数据修改操作变慢。每修改数据记录，索引就必须刷新一次。为了在某种程序上弥补这一缺陷，许 多SQL命令都有一个DELAY\_KEY\_WRITE项。这个选项的作用是暂时制止MySQL在该命令每插入一条新记录和每修改一条现有之后立刻对索引进 行刷新，对索引的刷新将等到全部记录插入/修改完毕之后再进行。在需要把许多新记录插入某个数据表的场合，DELAY\_KEY\_WRITE选项的作用将非 常明显。[2]另外，索引还会在硬盘上占用相当大的空间。因此应该只为最经常查询和最经常排序的数据列建立索引。注意，如果某个数据列包含许多重复的内 容，为它建立索引就没有太大的实际效果。

索引的限制

如果WEHERE子句的查询条件里有不等号(WHERE coloum != …)，MySQL将无法使用索引；

如果WHERE子句的查询条件里使用了函数(WHERE DAY(column) = …)，MySQL也将无法使用索引；

在JOIN操作中(需要从多个数据表提取数据时)，MySQL只有在主键和外键的数据类型相同时才能使用索引；

如果WHERE子句的查询条件里使用比较操作符LIKE和REGEXP，MySQL只有在搜索模板的第一个字符不是通配符的情况下才能使用索引(比如说， 如果查询条件是LIKE ‘abc%’，MySQL将使用索引；如果查询条件是LIKE ‘%abc’，MySQL将不使用索引。)；

在ORDER BY操作中，MySQL只有在排序条件不是一个查询条件表达式的情况下才使用索引 (虽然如此，在涉及多个数据表查询里，即使有索引可用，那些索引在加快 ORDER BY方面也没什么作用)；

如果某个数据列里包含许多重复的值，就算为它建立了索引也不会有很好的效果（比如说，如果某个数据列里包含的净是些诸如”0/1″或”Y/N”等值，就没 有必要为它创建一个索引）；

在为CHAR和VARCHAR类型的数据列定义索引时，可以把索引的长度限制为一个给定的字符个数（这个数字必须小于这个字段所允许的最大字符个数）。这么做的好处是可以生成一个尺寸比较小、检索速度却比较快的索引文件。在绝大多数应用里，数据库中的字符串数据大都以各种各样的名字为主，把索引的长度设置为10~15个字符已经足以把搜索范围缩小到很少的几条数据记录了。在为BLOB和TEXT类型的数据列创建索引时，必须对索引的长度做出限制；MySQL所允许的最大索引长度是255个字符；

从理论上讲，完全可以为数据表里的每个字段分别建一个索引，但MySQL把同一个数据表里的索引总数限制为16个。

索引类型

1、普通索引

普通索引（由关键字KEY或INDEX定义的索引）的唯一任务是加快对数据的访问速度。因此，应该只为那些最经常出现在查询条件（WHERE column=）或排序条件（ORDER BY column）中的数据列创建索引。只要有可能，就应该选择一个数据最整齐、最紧凑的数据列（如一个整数类型的数据列）来创建索引。

添加INDEX(普通索引) : ALTER TABLE ‘table\_name’ ADD INDEX index\_name (‘column’);

2、唯一索引

普通索引允许被索引的数据列包含重复的值。比如说，因为人有可能同名，所以同一个姓名在同一个"员工个人资料"数据表里可能出现两次或更多次。

如果能确定某个数据列将只包含彼此各不相同的值，在为这个数据列创建索引的时候就应该用关键字UNIQUE把它定义为一个唯一索引。这么做的好处：一是简化了MySQL对这个索引的管理工作，这个索引也因此而变得更有效率；二是MySQL会在有新记录插入数据表时，自动检查新记录的这个字段的值是否已经在某个记录的这个字段里出现过了；如果是，MySQL将拒绝插入那条新记录。也就是说，唯一索引可以保证数据记录的唯一性。事实上，在许多场合，人们创建唯一索引的目的往往不是为了提高访问速度，而只是为了避免数据出现重复。

添加UNIQUE(唯一索引) : ALTER TABLE ‘table\_name’ ADD UNIQUE (‘column’);

3、主索引

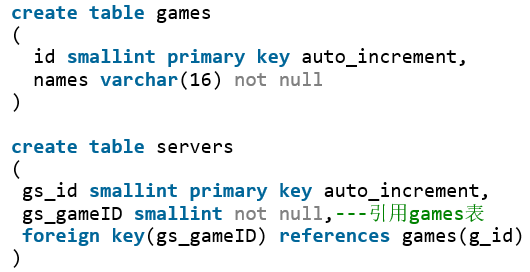
在前面已经反复多次强调过：必须为主键字段创建一个索引，这个索引就是所谓的"主索引".主索引与唯一索引的唯一区别是：前者在定义时使用的关键字是PRIMARY而不是UNIQUE.主键本身对mySQL就是一种唯一索引。

添加PRIMARY KEY（主键索引）：ALTER TABLE ‘table\_name’ ADD PRIMARY KEY ( ‘column’);

4、外键索引

如果为某个外键字段定义了一个外键约束条件，MySQL就会定义一个内部索引来帮助自己以最有效率的方式去管理和使用外键约束条件。

foreign key(gs\_gameID) references games(g\_id)



5、复合索引

索引可以覆盖多个数据列，如像INDEX（columnA,columnB）索引。这种索引的特点是MySQL可以有选择地使用一个这样的索引。如果查询操作只需要用到columnA数据列上的一个索引，就可以使用复合索引INDEX（columnA,columnB）。不过，这种用法仅适用于在复合索引中排列在前的数据列组合。比如说，INDEX（A,B,C）可以当做A或（A,B）的索引来使用，但不能当做B、C或（B,C）的索引来使用。

添加复合（多列）索引 : ALTER TABLE ‘table\_name’ ADD INDEX index\_name (‘column1’, ‘column2’, ‘column3’);

6、全文索引

全文索引文本字段上的普通索引只能加快对出现在字段内容最前面的字符串（也就是字段内容开头的字符）进行检索操作。如果字段里存放的是由几个、甚至是多个单词构成的较大段文字，普通索引就没什么作用了。这种检索往往以的形式出现，这对MySQL来说很复杂，如果需要处理的数据量很大，响应时间就会很长。

这类场合正是全文索引（full-textindex）可以大显身手的地方。在生成这种类型的索引时，MySQL将把在文本中出现的所有单词创建为一份清单，查询操作将根据这份清单去检索有关的数据记录。全文索引即可以随数据表一同创建，也可以等日后有必要时再使用下面这条命令添加：

　　ALTER TABLE tablename ADD FULLTEXT（column1,column2）有了全文索引，就可以用SELECT查询命令去检索那些包含着一个或多个给定单词的数据记录了。下面是这类查询命令的基本语法：

　　SELECT \* FROM tablename WHERE MATCH（column1,column2）AGAINST（'word1','word2','word3'）

上面这条命令将把column1和column2字段里有word1、word2和word3的数据记录全部查询出来。

注解：InnoDB数据表不支持全文索引。

查询和索引的优化

如果在测试数据库里只有几百条数据记录，它们往往在执行完第一条查询命令 之后就被全部加载到内存里，这将使后续的查询命令都执行得非常快–不管有没有使用索引。

在不确定应该在哪些数据列上创建索引的时候，可以用EXPLAIN SELECT命令，给一条普通的SELECT命令加一个EXPLAIN关键字，MySQL将不去执行那条SELECT命令，而是去对它进行分析，以表格的形式把查询的执行过程和用到的索引(如果有的话)等信息列出 来。

在EXPLAIN命令的输出结果里，第1列是从数据库读取的数据表的名字，它们按被读取的先后顺序排列。type列指定了本数据表与其它数据表之间的关联 关系(JOIN)。在各种类型的关联关系当中，效率最高的是system，然后依次是const、eq\_ref、ref、range、index和 All(All的意思是：对应于上一级数据表里的每一条记录，这个数据表里的所有记录都必须被读取一遍–这种情况往往可以用一索引来避免)。

possible\_keys数据列给出了MySQL在搜索数据记录时可选用的各个索引。key数据列是MySQL实际选用的索引，这个索引按字节计算的长 度在key\_len数据列里给出。比如说，对于一个INTEGER数据列的索引，这个字节长度将是4。如果用到了复合索引，在key\_len数据列里还可 以看到MySQL具体使用了它的哪些部分。作为一般规律，key\_len数据列里的值越小越好(意思是更快)。

ref数据列给出了关联关系中另一个数据表里的数据列的名字。row数据列是MySQL在执行这个查询时预计会从这个数据表里读出的数据行的个数。row 数据列里的所有数字的乘积可以让我们大致了解这个查询需要处理多少组合。

最后，extra数据列提供了与JOIN操作有关的更多信息，比如说，如果MySQL在执行这个查询时必须创建一个临时数据表，就会在extra列看到 using temporary字样。

修改和创建索引

show index from mysql.user;

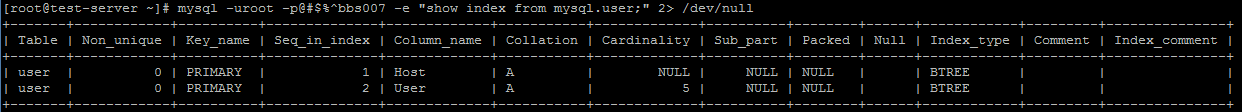


Table 表名

Non\_unique 0，如果索引不能包含重复。

Key\_name 索引名

Seq\_in\_index 索引中的列顺序号, 从 1 开始。

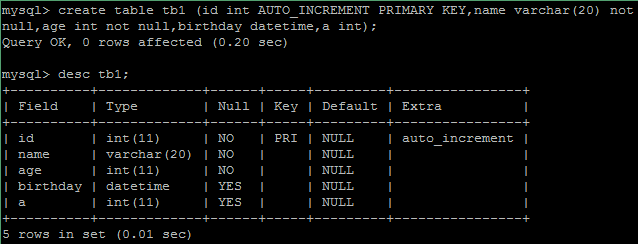
Column\_name 列名。

Collation 列怎样在索引中被排序。在MySQL中，这可以有值A（升序) 或NULL（不排序)。

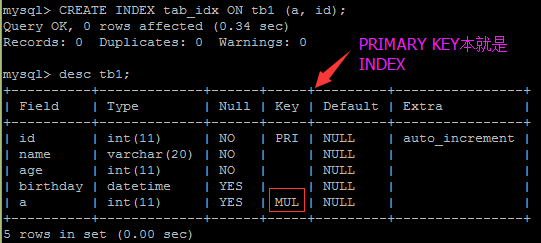
Cardinality 索引中唯一值的数量。这可通过运行isamchk -a更改.

Sub\_part 如果列只是部分被索引，索引字符的数量。NULL，如果整个键被索引。

创建INDEX



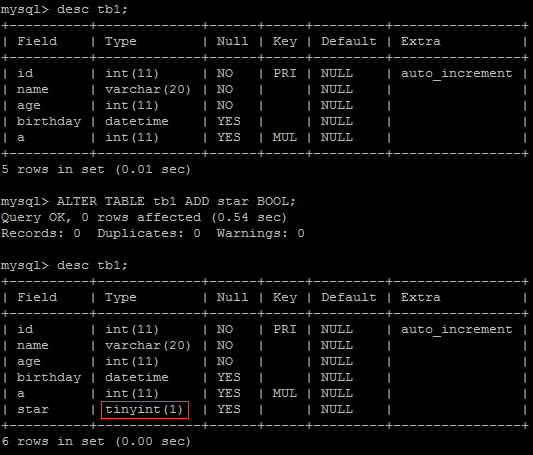
CREATE INDEX tab\_idx ON tbl (a, id);



添加索引

1. 修改字段属性

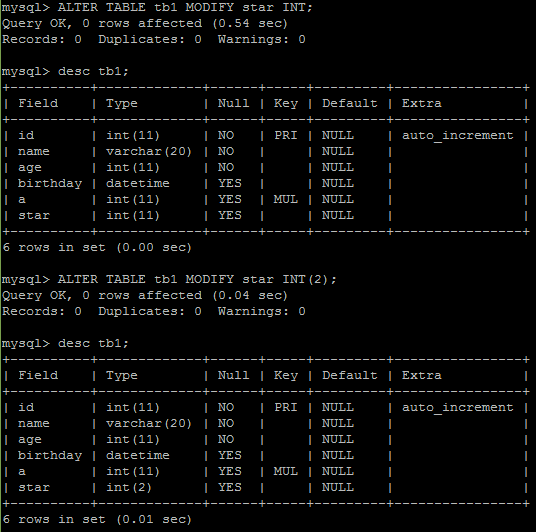
新建字段，默认为tinyint(1)类型(布尔类型会自动转换为tinyint(1)类型)



将字段的tinyint(1)类型转换为int类型(默认为INT(11))

ALTER TABLE tb1 MODIFY star INT;

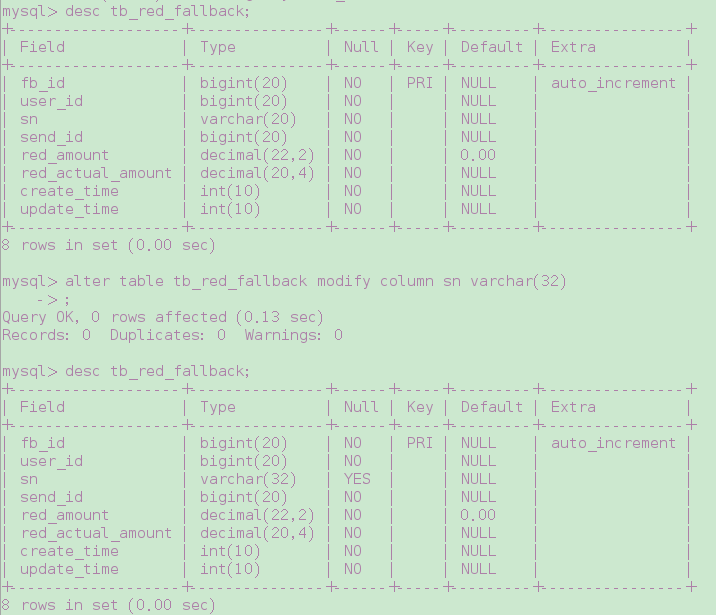
ALTER TABLE tb1 MODIFY star INT(2);



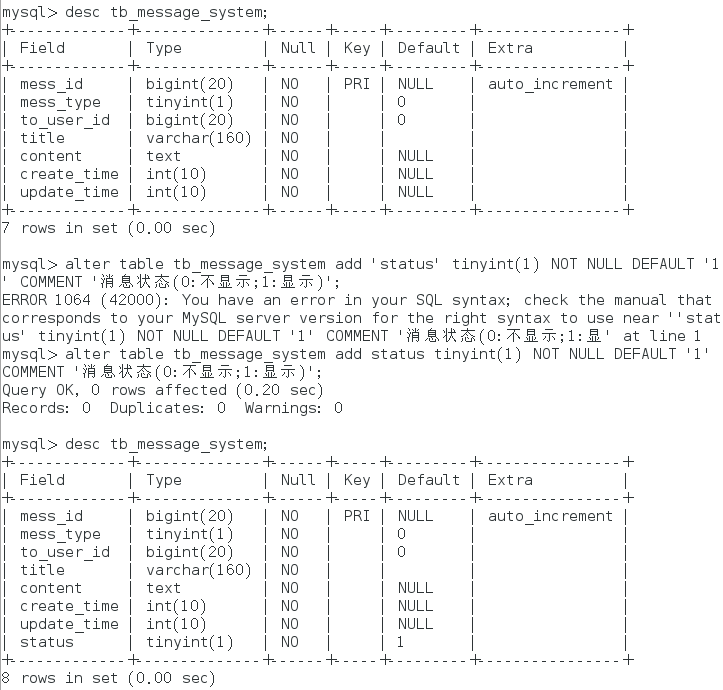
4.修改字段长度

tb\_bank\_bindcard的字段sn的长度是varchar(20)，现改为varchar(32）

alter table tb\_red\_fallback modify column sn varchar(32);



5.增加字段

alter table tb\_message\_system add status tinyint(1) NOT NULL DEFAULT '1' COMMENT '消息状态(0:不显示;1:显示)';

注意：字段名称不能有引号。

QPS TPS

QPS (Query per second) （每秒查询量）

TPS(Transaction per second) （每秒事务量，如果是InnoDB会显示，没有InnoDB就不会显示）

计算方式：

QPS

Questions = SHOW GLOBAL STATUS LIKE 'Questions';

Uptime = SHOW GLOBAL STATUS LIKE 'Uptime';

QPS=Questions/Uptime

TPS

Com\_commit = SHOW GLOBAL STATUS LIKE 'Com\_commit';

Com\_rollback = SHOW GLOBAL STATUS LIKE 'Com\_rollback';

Uptime = SHOW GLOBAL STATUS LIKE 'Uptime';

TPS=(Com\_commit + Com\_rollback)/Uptime

(1)QPS(每秒Query量)

QPS = Questions(or Queries) / seconds

mysql > show global status like 'Question%';

(2)TPS(每秒事务量)

TPS = (Com\_commit + Com\_rollback) / seconds

mysql > show global status like 'Com\_commit';

mysql > show global status like 'Com\_rollback';

主主同步

Master-connect-retry

master-connect-retry=n表示从服务器与主服务器的连接没有成功，则等待n秒（s）后再进行管理方式（默认设置是60s）。如果从服务器存在mater.info文件，它将忽略些选项。

Auto\_increment\_offset和Auto\_increment\_increment

Auto\_increment\_increment和auto\_increment\_offset用于主－主服务器（master-to-master）复制，并可以用来控制AUTO\_INCREMENT列的操作。两个变量均可以设置为全局或局部变量，并且假定每个值都可以为1到65,535之间的整数值。将其中一个变量设置为0会使该变量为1。

这两个变量影响AUTO\_INCREMENT列的方式：auto\_increment\_increment控制列中的值的增量值，auto\_increment\_offset确定AUTO\_INCREMENT列值的起点。

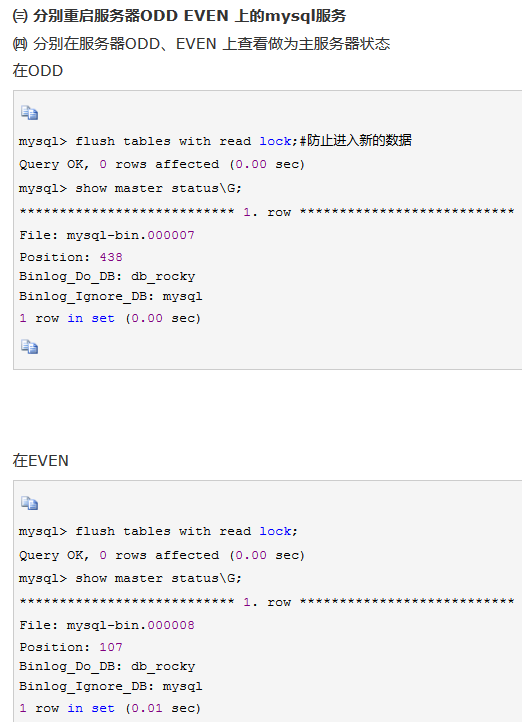
如果auto\_increment\_offset的值大于auto\_increment\_increment的值，则auto\_increment\_offset的值被忽略。例如：表内已有一些数据，就会用现在已有的最大自增值做为初始值。

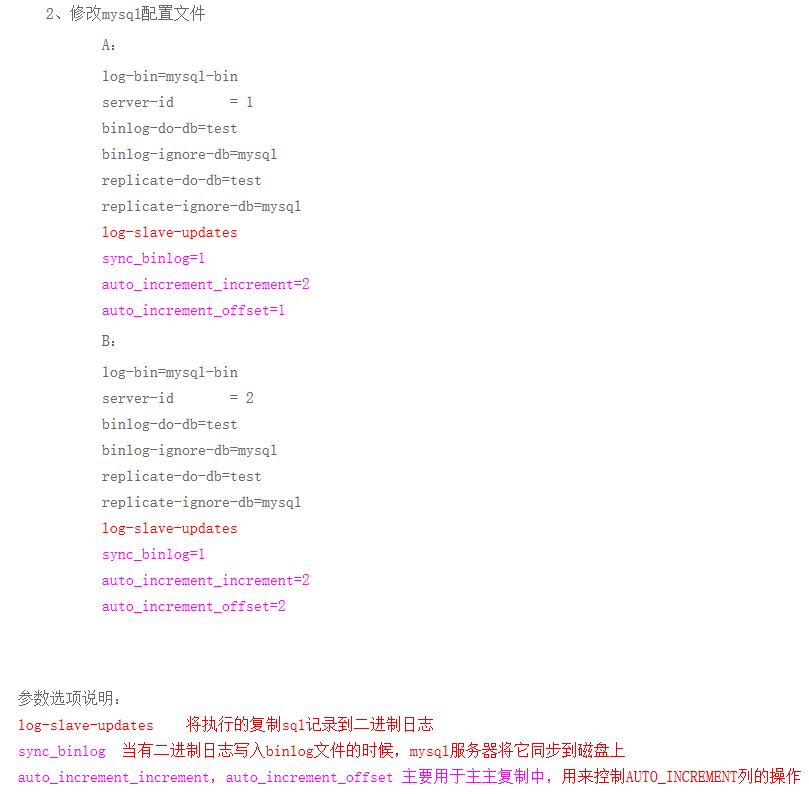
当innodb\_flush\_log\_at\_trx\_commit和sync\_binlog 都为 1 时是最安全的，在mysqld 服务崩溃或者服务器主机crash的情况下，binary log 只有可能丢失最多一个语句或者一个事务。但是鱼与熊掌不可兼得，双11 会导致频繁的io操作，因此该模式也是最慢的一种方式。

当innodb\_flush\_log\_at\_trx\_commit设置为0，mysqld进程的崩溃会导致上一秒钟所有事务数据的丢失。

当innodb\_flush\_log\_at\_trx\_commit设置为2，只有在操作系统崩溃或者系统掉电的情况下，上一秒钟所有事务数据才可能丢失。

双1适合数据安全性要求非常高，而且磁盘IO写能力足够支持业务，比如订单,交易,充值,支付消费系统。双1模式下，当磁盘IO无法满足业务需求时 比如11.11 活动的压力。推荐的做法是 innodb\_flush\_log\_at\_trx\_commit=2 ，sync\_binlog=N (N为500 或1000) 且使用带蓄电池后备电源的缓存cache，防止系统断电异常。







[主主同步主键冲突处理](http://www.db2china.net/home/space.php?uid=158799&do=blog&id=33887)

[主主同步延迟问题的定位及解决](http://01wang.blog.51cto.com/2763708/1635952)

配置文件

首要考虑

no-auto-rehash

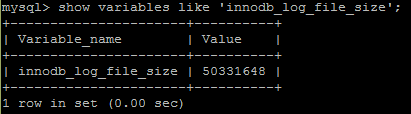
这是在[mysql]项中的配置(不是[mysqld])，这个配置段设置启动MySQL服务的条件，no-auto-rehash确保这个服务启动得比较快。

innodb\_buffer\_pool\_size

默认值为 128M. 这是最主要的优化选项,因为它指定 InnoDB 使用多少内存来加载数据和索引(data+indexes). 针对专用MySQL服务器,建议指定为物理内存的 50-80%这个范围. 如果将该值设置得更大可能会存在风险,比如没有足够的空闲内存留给操作系统和依赖文件系统缓存的某些MySQL子系统(subsystem),包括二进制日志(binary logs),InnoDB事务日志(transaction logs)等.

innodb\_log\_file\_size

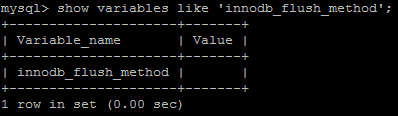
这是redo日志的大小，默认48M。redo日志被用于确保写操作快速而可靠并且在崩溃时恢复。有很高写入吞吐量的系统需要增加该值以允许后台检查点活动在更长的时间周期内平滑写入,得以改进性能. 将此值设置为4G以下是很安全的. 过去的实践表明,日志文件太大的缺点是增加了崩溃时所需的修复时间,但这在5.5和5.6中已得到重大改进。redo日志默认可以有2个log文件，一开始就把innodb\_log\_file\_size设置成512M(这样有1GB的redo日志)会使你有充裕的写操作空间。如果你知道你的应用程序需要频繁的写入数据，可以一开始就把它设置成4G。



所有MySQL调优建议都说，innodb\_log\_file\_size要越大越好，避免无谓的buffer pool的flush操作。但是在我的实践中来看，innodb\_log\_file\_size开得太大，会明显增加innodb的log写入操作，而且会造成操作系统需要更多的Disk Cache开销，innodb\_log\_file\_size也不宜太大。（参考：[MySQL InnoDB性能调整的一点实践](http://www.iteye.com/topic/461382)）

innodb\_flush\_method

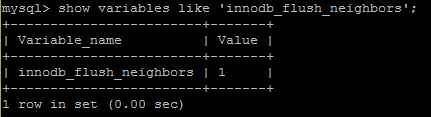
这项配置决定了数据和日志写入硬盘的方式。一般来说，如果你有硬件RAID控制器，并且其独立缓存采用write-back机制，并有着电池断电保护，那么应该设置配置为O\_DIRECT；否则，大多数情况下应将其设为fdatasync（默认值）。sysbench是一个可以帮助你决定这个选项的好工具。



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。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。（参考：[MySQL InnoDB性能调整的一点实践](http://www.iteye.com/topic/461382)）

innodb\_flush\_neighbors

默认值为 1. 在SSD存储上应设置为0(禁用) ,因为使用顺序IO没有任何性能收益. 在使用RAID的某些硬件上也应该禁用此设置,因为逻辑上连续的块在物理磁盘上并不能保证也是连续的.



sync\_relay\_log=1

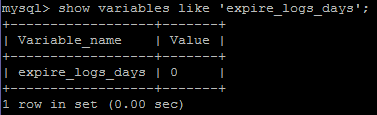
启用二进制日志后， 默认情况下二进制日志不是事故安全的(not crash safe)

relay-log-info-repository=TABLE 和 master-info-repository=TABLE

配合sync\_relay\_log=1做出的优化

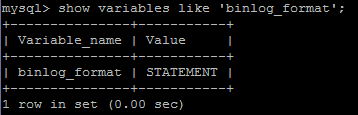
expire-logs-days

默认值是0，旧日志会一直保留. 我推荐设置为 1-10 天. 保存更长的时间并没有太多用处,因为从备份中恢复会快得多



binlog\_format=ROW

修改为基于行的复制，可以通过减少资源锁定提高性能



character-set-server=utf8mb4 和 collation-server=utf8mb4\_general\_ci

utf8 编码对新应用来说是更好的默认选项. 您还可以设置 skip-character-set-client-handshake 以忽略应用程序想要设置的其他字符集(character-set).

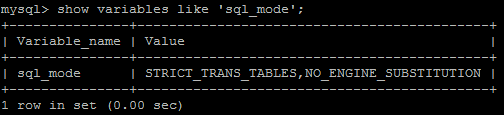
sql-mode

MySQL默认对不规范的数据很宽容,并且会静默地截断数据. 在我 之前的一篇文章中, 我提到新应用程序最好设置为： STRICT\_TRANS\_TABLES,ERROR\_FOR\_DIVISION\_BY\_ZERO,

NO\_AUTO\_CREATE\_USER,NO\_AUTO\_VALUE\_ON\_ZERO,

NO\_ENGINE\_SUBSTITUTION,NO\_ZERO\_DATE,

NO\_ZERO\_IN\_DATE,ONLY\_FULL\_GROUP\_BY.



external-locking

使用skip-external-locking选项以避免外部锁定。该选项默认开启，允许外部文件级别的锁. 打开文件锁会对性能造成负面影响。所以只有在你在同样的文件上运行多个数据库实例时才使用此选项(注意仍会有其他约束!)或者你在文件层面上使用了其他一些软件依赖来锁定MyISAM表建议设为external-locking = FALSE，阿里云mysql设为skip-external-locking

skip-name-resolve

禁用反向域名解析. DNS解析在某些系统上可能有点慢/不稳定,所以如果不需要基于主机名的授权,我建议避免这种解析

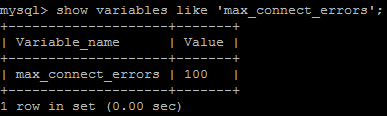
max-connections

默认值是151。16核的机器在IO阻塞的情况下也只有大约 2x~10x 的连接执行能力。你可能希望,许多打开的连接都是空闲并休眠的. 但如果他们都处于活跃状态的话,可能会创建大量新的线程(thread-thrash)。如果条件允许,可以为应用程序配置优化数据库连接池(connection-pools)来解决这个问题,而不是打开并保持大量连接; 当然那些不使用连接池(non-pooled ), 迅速打开,执行任务后又尽可能快地关闭连接的应用也是可行的. MySQL服务器允许的最大连接数16384。

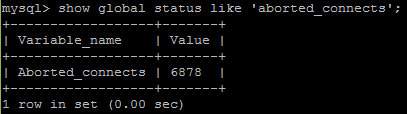
read\_buffer\_size

max\_connect\_errors

默认100，设置每个主机的连接请求异常中断的最大次数，当超过该次数，MYSQL服务器将禁止host的连接请求，直到mysql服务器重启或通过flush hosts命令清空此host的相关信息。

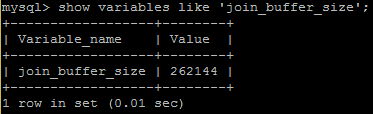


查看 “aborted\_connects” 状态来获取全局计数器.

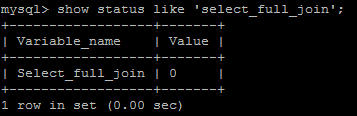


join\_buffer\_size

此缓冲被使用来优化全联合(full JOINs 不带索引的联合)，类似的联合在极大多数情况下有非常糟糕的性能表现，但是将此值设大能够减轻性能影响，默认是256K

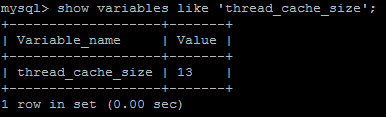


通过 “Select\_full\_join” 状态变量查看全联合的数量，当全联合发生时,在每个线程中分配

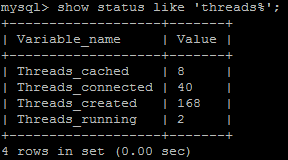


thread\_cache\_size

在cache中保留多少线程用于重用，默认值是13。当一个客户端断开连接后,如果cache中的线程还少于thread\_cache\_size,则客户端线程被放入cache中.如果线程重新被请求，那么请求将从缓存中读取,如果缓存中是空的或者是新的请求，那么这个线程将被重新创建这可以在你需要大量新连接的时候极大的减少线程创建的开销 (一般来说如果你有好的线程模型的话,这不会有明显的性能提升.)

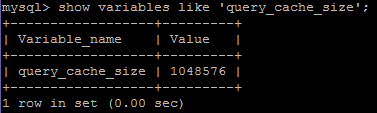


通过比较 Connections 和 Threads\_created 状态的变量，可以看到这个变量的作用。

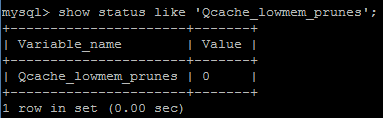


query\_cache\_size

query cache（查询缓存）的工作原理：一个SELECT查询在DB中工作后，DB会把该语句缓存下来，当同样的一个SQL再次来到DB里调用时，DB在该表没发生变化的情况下把结果从缓存中返回给Client。这里有一个关建点，就是DB在利用Query\_cache工作时，要求该语句涉及的表在这段时间内没有发生变更。那如果该表在发生变更时，Query\_cache里的数据又怎么处理呢？首先要把Query\_cache和该表相关的语句全部置为失效，然后在写入更新。那么如果Query\_cache非常大，该表的查询结构又比较多，查询语句失效也慢，一个更新或是Insert就会很慢，这样看到的就是Update或是Insert怎么这么慢了。所以在数据库写入量或是更新量也比较大的系统，该参数不适合分配过大。而且在高并发，写入量大的系统，建议把该功能禁掉。该值默认是1M。 最佳选项是将其从一开始就停用，设置query\_cache\_size = 0（现在MySQL 5.6的默认值）并利用其他方法加速查询：优化索引、增加拷贝分散负载或者启用额外的缓存（比如memcache或redis）。如果你已经为你的应用启用了query cache并且还没有发现任何问题，query cache可能对你有用。这时如果你想停用它，那就得小心了。

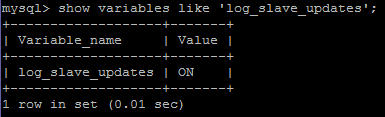


查询缓冲常被用来缓冲 SELECT 的结果并且在下一次同样查询的时候不再执行直接返回结果， 打开查询缓冲可以极大的提高服务器速度, 如果你有大量的相同的查询并且很少修改表。查看 “Qcache\_lowmem\_prunes” 状态变量来检查是否当前值对于你的负载来说是否足够高，注意: 在你表经常变化的情况下或者如果你的查询原文每次都不同，查询缓冲也许引起性能下降而不是性能提升.



log\_slave\_updates

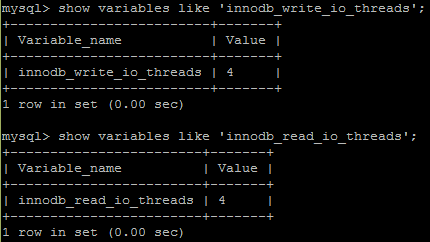
如果你在使用链式从服务器结构的复制模式 (A->B->C)，你需要在服务器B上打开此项。此选项打开在从线程上重做过的更新的日志，并将其写入从服务器的二进制日志.



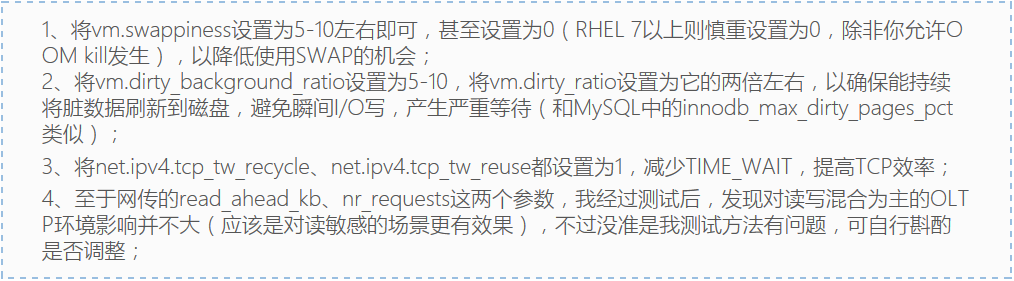
innodb\_write\_io\_threads

innodb\_read\_io\_threads

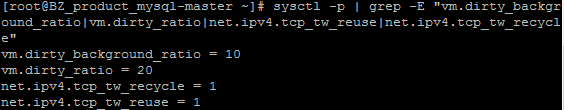
innodb使用后台线程处理数据页上的读写 I/O(输入输出)请求,根据你的 CPU 核数来更改,默认是4。注:这两个参数不支持动态改变,需要把该参数加入到my.cnf里，修改完后重启MySQL服务



系统级优化



sysctl -p | grep -E "vm.dirty\_background\_ratio|vm.dirty\_ratio|net.ipv4.tcp\_tw\_reuse|net.ipv4.tcp\_tw\_recycle"



Centos6.5默认vm.dirty\_background\_ratio = 10、vm.dirty\_ratio = 20

配置研究

#研究设置 InnoDB settings

#innodb\_buffer\_pool\_size=1G

#innodb\_log\_file\_size=1G

#innodb\_flush\_method=O\_DIRECT

#innodb\_io\_capacity=200

#innodb\_io\_capacity\_max=500

#innodb\_lru\_scan\_depth=200

#研究设置 replication

#log-bin

#sync\_binlog=1

#sync\_relay\_log=1

#relay-log-info-repository=TABLE

#master-info-repository=TABLE

#expire\_logs\_days=10

#binlog\_format=ROW

#transaction-isolation=READ-COMMITTED

#innodb\_autoinc\_lock\_mode = 2

#研究设置 Other

#character-set-server=utf8

#collation-server=utf8\_general\_ci

#sql-mode="STRICT\_TRANS\_TABLES,

# ERROR\_FOR\_DIVISION\_BY\_ZERO,

# NO\_AUTO\_CREATE\_USER,

# NO\_AUTO\_VALUE\_ON\_ZERO,

# NO\_ENGINE\_SUBSTITUTION,

# NO\_ZERO\_DATE,

# NO\_ZERO\_IN\_DATE,

# ONLY\_FULL\_GROUP\_BY"

#skip-name\_resolve

#max-connect-errors=100000

#max-connections=500

配置建议

1. 当写入时，使用 innodb\_flush\_method=O\_DIRECT 来避免双缓冲.

2. 分配足够的 innodb\_buffer\_pool\_size 来加载整个 InnoDB 文件到内存中

3. 不要将 innodb\_log\_file\_size 参数设置太大， 这样可以更快同时有更多的磁盘空间 – 丢掉多的日志通常是好的，在数据库崩溃后可以降低恢复数据库的时间.

4. 不要混用 innodb\_thread\_concurrency 和 thread\_concurrency 参数– 这2个值是不兼容的.

5. 分配一个极小的数量给 max\_connections 参数 – 太多的连接会用尽RAM并锁定MySQL服务.

6. 保持 thread\_cache 在一个相对较高的数字，大约 16 – 防止打开连接时缓慢.

7. 使用skip-name-resolve参数 – 去掉 DNS 查找.

8. 如果你的查询都是重复的，并且数据不常常发生变化，那么可以使用查询缓存. 但是如果你的数据经常发生变化，那么使用查询缓存会让你感到失望.

9. 增大temp\_table\_size值，以防止写入磁盘

10. 增大max\_heap\_table\_size值，以防止写入磁盘

11. 不要把sort\_buffer\_size值设置的太高，否则的话你的内存将会很快耗尽

12. 根据key\_read\_requests和key\_reads值来决定key\_buffer的大小，一般情况下key\_read\_requests应该比key\_reads值高，否则你不能高效的使用key\_buffer

13. 将innodb\_flush\_log\_at\_trx\_commit设置为0将会提高性能，但是如果你要保持默认值（1）的话，那么你就要确保数据的完整性，同时你也要确保复制不会滞后.

阿里mysql配置

[mysqld]

……

skip-external-locking

log-error=/alidata/log/mysql/error.log

key\_buffer\_size = 16M

max\_allowed\_packet = 1M

table\_open\_cache = 64

sort\_buffer\_size = 512K

net\_buffer\_length = 8K

read\_buffer\_size = 256K

read\_rnd\_buffer\_size = 512K

myisam\_sort\_buffer\_size = 8M

log-bin=mysql-bin

binlog\_format=mixed

server-id = 1

sql\_mode=NO\_ENGINE\_SUBSTITUTION,STRICT\_TRANS\_TABLES

[mysqldump]

quick

max\_allowed\_packet = 16M

[mysql]

no-auto-rehash

[myisamchk]

key\_buffer\_size = 20M

sort\_buffer\_size = 20M

read\_buffer = 2M

write\_buffer = 2M

[mysqlhotcopy]

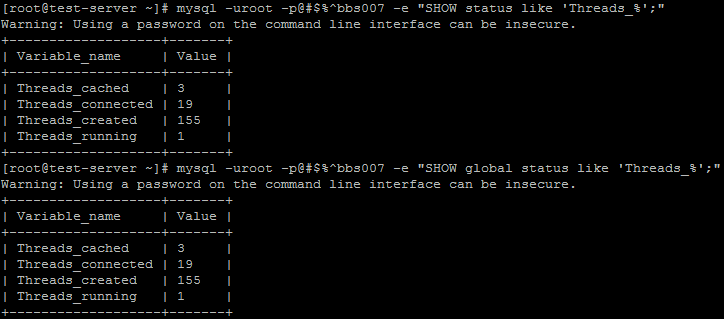
interactive-timeout

expire\_logs\_days = 5

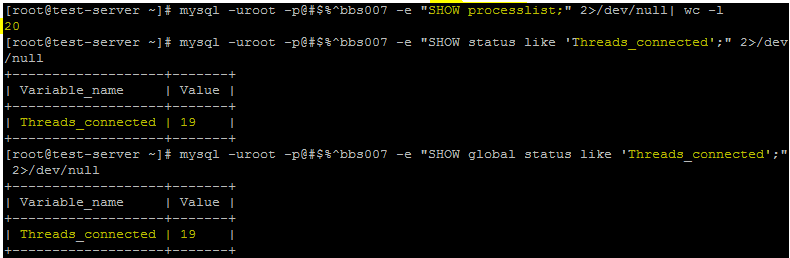
max\_binlog\_size = 1000M

thread

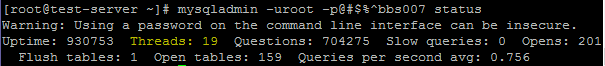
"SHOW status like 'Threads\_%';"和"SHOW global status like 'Threads\_%';"执行的结果一样，都反映了mysql的线程



而Threads\_connected的值，正是"SHOW processlist;"统计的值



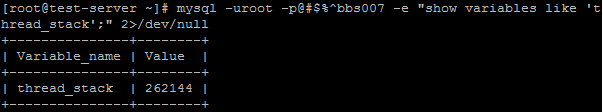
同时也是mysqladmin status中Threads的值



其实这都是mysql的连接数。

thread\_stack

设置MYSQL每个线程的堆栈大小，默认值为256KB，可满足普通操作。



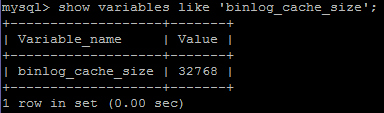
thread\_concurrency

设置thread\_concurrency的值的正确与否,会导致mysql能否充分利用多cpu(或多核), 系统默认为10，即同一时刻能有5个cpu(或核)在工作。thread\_concurrency应设为CPU核数的2倍，2个双核的cpu, thread\_concurrency的值应为8

cache

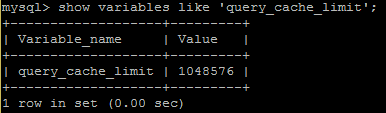
binlog\_cache\_size

在一个事务中binlog为了记录SQL状态所持有的cache大小，如果你经常使用大的,多声明的事务,你可以增加此值来获取更大的性能。所有从事务来的状态都将被缓冲在binlog缓冲中然后在提交后一次性写入到binlog中；如果事务比此值大, 会使用磁盘上的临时文件来替代。此缓冲在每个连接的事务第一次更新状态时被创建，默认是32K



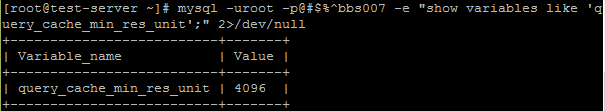
query\_cache\_limit

指定单个查询能够使用的缓冲区大小，缺省为1M。此设置用来保护查询缓冲，防止一个极大的结果集将其他所有的查询结果都覆盖，只有小于此设定值的结果才会被缓冲

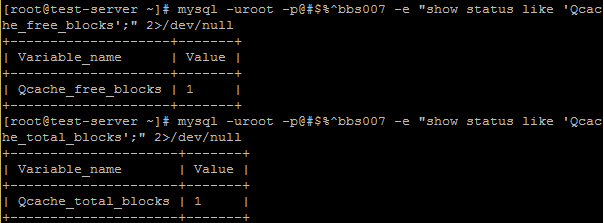


query\_cache\_min\_res\_unit

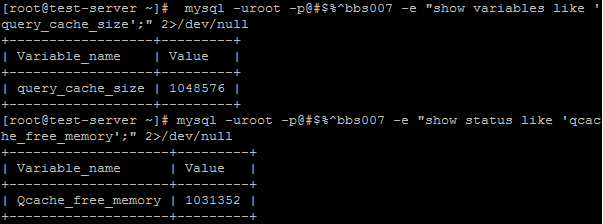
默认是4KB，设置值大对大数据查询有好处，但如果你的查询都是小数据查询，就容易造成内存碎片和浪费。如果查询缓存碎片率超过20%，可以用FLUSH QUERY CACHE整理缓存碎片，或者试试减小query\_cache\_min\_res\_unit，如果你的查询都是小数据量的话。查询缓存利用率在25%以下的话说明query\_cache\_size设置的过大，可适当减小;查询缓存利用率在80%以上而且Qcache\_lowmem\_prunes > 50的话说明query\_cache\_size可能有点小，要不就是碎片太多。



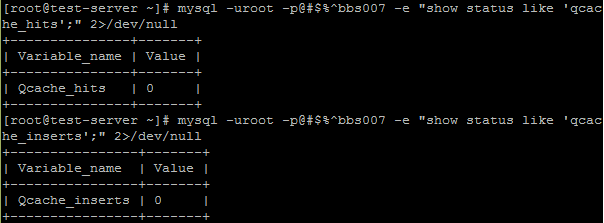
查询缓存碎片率 = Qcache\_free\_blocks / Qcache\_total\_blocks \* 100%，这里是100%



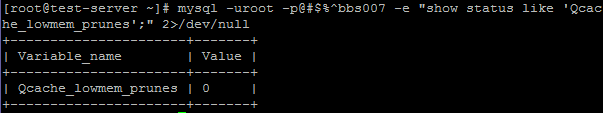
查询缓存利用率 = (query\_cache\_size – Qcache\_free\_memory) / query\_cache\_size \* 100%，这里是1.64%



查询缓存命中率 = (Qcache\_hits – Qcache\_inserts) / Qcache\_hits \* 100%，这里是0

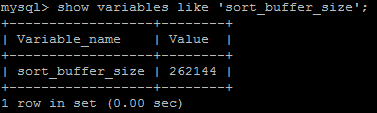


Qcache\_lowmem\_prunes为0

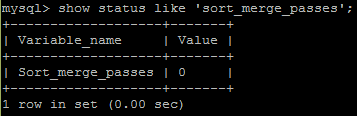


sort\_buffer\_size

排序缓冲被用来处理类似ORDER BY以及GROUP BY队列所引起的排序，如果排序后的数据无法放入排序缓冲，一个用来替代的基于磁盘的合并分类会被使用。系统默认256k。阿里mysql在[mysqld]项中设为sort\_buffer\_size = 512K，在[myisamchk]项中设为sort\_buffer\_size = 20M，在[mysqld]项中设置myisam\_sort\_buffer\_size = 8M

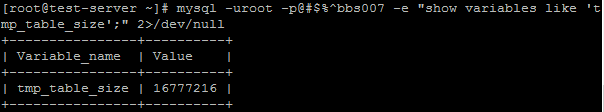


查看 “sort\_merge\_passes” 状态变量，在排序发生时由每个线程分配



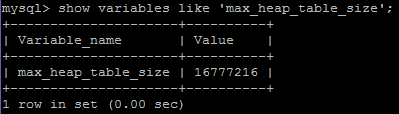
tmp\_table\_size

tmp\_table\_size 的默认大小是 16M。MySQL的heap（堆积）表缓冲大小。所有联合在一个DML指令内完成，并且大多数联合甚至可以不用临时表即可以完成。大多数临时表是基于内存的(HEAP)表。具有大的记录长度的临时表 (所有列的长度的和)或包含BLOB列的表存储在硬盘上。如果某个内部heap（堆积）表大小超过tmp\_table\_size，MySQL可以根据需要自动将内存中的heap表改为基于硬盘的MyISAM表。还可以通过设置tmp\_table\_size选项来增加临时表的大小。也就是说，如果调高该值，MySQL同时将增加heap表的大小，可达到提高联接查询速度的效果。此限制是针对单个表的，而不是总和。如果一张临时表超出该大小，MySQL产生一个 The table tbl\_name is full 形式的错误，如果你做很多高级 GROUP BY 查询，建议增加 tmp\_table\_size 值。



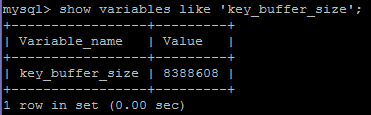
max\_heap\_table\_size

默认值是16M



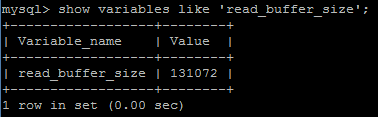
key\_buffer\_size

默认是8M，批定用于索引的缓冲区大小，增加它可以得到更好的索引处理性能，对于内存在4GB左右的服务器来说，该参数可设置为256MB或384MB。



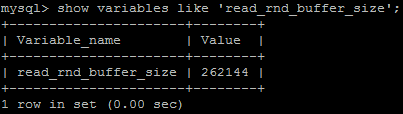
read\_buffer\_size

mySql读入缓冲区大小，默认128K。对表进行顺序扫描的请求将分配一个读入缓冲区，MySql会为它分配一段内存缓冲区。read\_buffer\_size变量控制这一缓冲区的大小。如果对表的顺序扫描请求非常频繁，并且你认为频繁扫描进行得太慢，可以通过增加该变量值以及内存缓冲区大小提高其性能。和sort\_buffer\_size一样，该参数对应的分配内存也是每个连接独享。



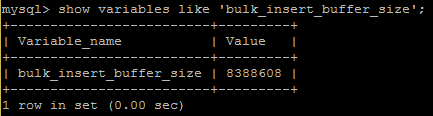
read\_rnd\_buffer\_size

MySql的随机读（查询操作）缓冲区大小，默认256K。当按任意顺序读取行时(例如，按照排序顺序)，将分配一个随机读缓存区。进行排序查询时，MySql会首先扫描一遍该缓冲，以避免磁盘搜索，提高查询速度，如果需要排序大量数据，可适当调高该值。但MySql会为每个客户连接发放该缓冲空间，所以应尽量适当设置该值，以避免内存开销过大。



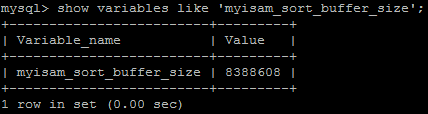
bulk\_insert\_buffer\_size

批量插入缓存大小， 这个参数是针对MyISAM存储引擎来说的。适用于在一次性插入100-1000+条记录时， 提高效率。默认值是8M。可以针对数据量的大小，翻倍增加。



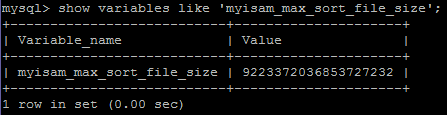
myisam\_sort\_buffer\_size

MyISAM设置恢复表之时使用的缓冲区的尺寸，当在REPAIR TABLE或用CREATE INDEX创建索引或ALTER TABLE过程中排序 MyISAM索引分配的缓冲区，默认8M



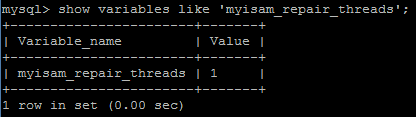
myisam\_max\_sort\_file\_size

MySQL重建索引时所允许的最大临时文件的大小 (当 REPAIR, ALTER TABLE 或者 LOAD DATA INFILE).如果文件大小比此值更大,索引会通过键值缓冲创建(更慢) 。如果临时文件会变得超过索引，不要使用快速排序索引方法来创建一个索引。注释：这个参数以字节的形式给出。



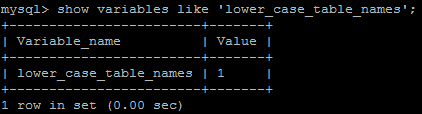
myisam\_repair\_threads

默认为1。如果一个表拥有超过一个索引, MyISAM 可以通过并行排序使用超过一个线程去修复他们.这对于拥有多个CPU以及大量内存情况的用户,是一个很好的选择.



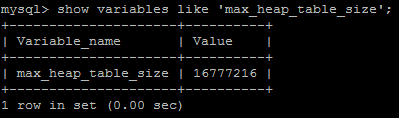
lower\_case\_table\_names

默认为1



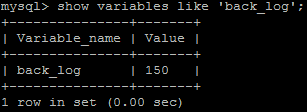
max\_heap\_table\_size

独立的内存表所允许的最大容量，此选项为了防止意外创建一个超大的内存表导致永尽所有的内存资源。默认是16M



back\_log

back\_log 是操作系统在监听队列中所能保持的连接数，队列保存了在MySQL连接管理器线程处理之前的连接。如果MySQL的连接数据达到max\_connections时，新来的请求将会被存在堆栈中以等待某一连接释放资源，该堆栈的数量即back\_log，如果等待连接的数量超过back\_log，将不被授予连接资源。默认值150。如果你有非常高的连接率并且出现”connection refused” 报错，你就应该增加此处的值。



back\_log值指出在MySQL暂时停止回答新请求之前的短时间内多少个请求可以被存在堆栈中。也就是说，如果MySql的连接数据达到max\_connections时，新来的请求将会被存在堆栈中，以等待某一连接释放资源，该堆栈的数量即back\_log，如果等待连接的数量超过back\_log，将不被授予连接资源。将会报：unauthenticated user | xxx.xxx.xxx.xxx | NULL | Connect | NULL | login | NULL 的待连接进程时.

back\_log值不能超过TCP/IP连接的侦听队列的大小。若超过则无效，查看当前系统的TCP/IP连接的侦听队列的大小命令：cat /proc/sys/net/ipv4/tcp\_max\_syn\_backlog目前系统为1024。对于Linux系统推荐设置为小于512的整数。

修改系统内核参数，）http://www.51testing.com/html/64/n-810764.html

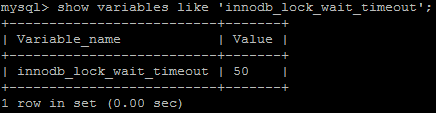
查看mysql 当前系统默认back\_log值，命令：

show variables like 'back\_log'; 查看当前数量

Innodb

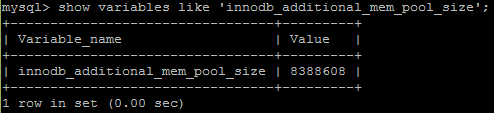
innodb\_lock\_wait\_timeout 和 innodb\_rollback\_on\_timeout

从官方文档中看，这个值是针对innodb引擎的，是innodb中行锁的等待超时时间，默认为50秒。如果超时，则当前语句会回滚。如果设置了innodb\_rollback\_on\_timeout，则会回滚整个事务，否则，只回滚事务等待行锁的这个语句。更深入了解参考http://www.jianshu.com/p/a5747a382a0a



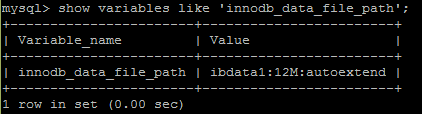
innodb\_additional\_mem\_pool\_size

这个参数用来设置 InnoDB 存储的数据目录信息和其它内部数据结构的内存池大小，类似于Oracle的library cache。默认是8M。



innodb\_data\_file\_path

innodb的表空间文件、重要数据，默认大小是12 M

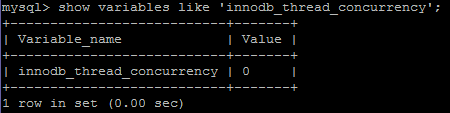


可实际上，它已经76MB了



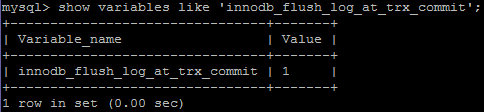
innodb\_thread\_concurrency

默认是0，表示不限制并发数，这里推荐设置为0，更好去发挥CPU多核处理能力，提高并发量



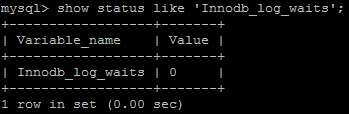
innodb\_flush\_log\_at\_trx\_commit

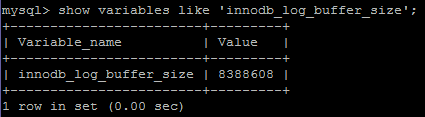
默认值为1，表示InnoDB完全支持ACID特性。当你的主要关注点是数据安全的时候这个值是最合适的，比如在一个主节点上。但是对于磁盘（读写）速度较慢的系统，它会带来很巨大的开销，因为每次将改变flush到redo日志都需要额外的fsyncs。将它的值设置为2会导致不太可靠（reliable）因为提交的事务仅仅每秒才flush一次到redo日志，但对于一些场景是可以接受的，比如对于主节点的备份节点这个值是可以接受的。如果值为0速度就更快了，但在系统崩溃时可能丢失一些数据：只适用于备份节点。



innodb\_log\_buffer\_size

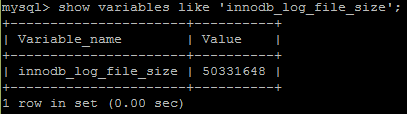
这项配置决定了为尚未执行的事务分配的缓存。其默认值（8MB）一般来说已经够用了，但是如果你的事务中包含有二进制大对象或者大文本字段的话，这点缓存很快就会被填满并触发额外的I/O操作。看看Innodb\_log\_waits状态变量，如果它不是0，增加innodb\_log\_buffer\_size。缓冲区更大能提高性能，但意外的故障将会丢失数据，默认是8M。





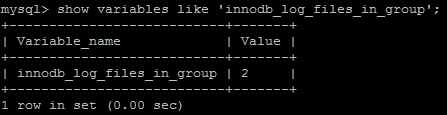
innodb\_log\_file\_size

这是redo日志的大小，默认48M。redo日志被用于确保写操作快速而可靠并且在崩溃时恢复。一直到MySQL 5.5，崩溃恢复的性能的到了很大提升，redo日志的总尺寸被限定在4GB(默认可以有2个log文件)。这在MySQL 5.6里被提高。一开始就把innodb\_log\_file\_size设置成512M(这样有1GB的redo日志)会使你有充裕的写操作空间。如果你知道你的应用程序需要频繁的写入数据，可以一开始就把它设置成4G。



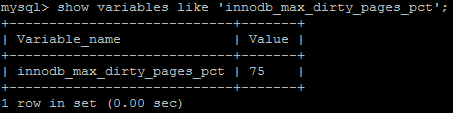
innodb\_log\_files\_in\_group

为提高性能，MySQL可以以循环方式将日志文件写到多个文件，默认是2



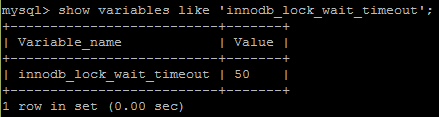
innodb\_max\_dirty\_pages\_pct

该参数直接控制了Dirty\_Page在Buffer\_Pool中所占的比率，默认为75，直接影响InnoDB的关闭时间。所以，在关闭InnoDB之前先将innodb\_max\_dirty\_pages\_pct调小，强制数据块Flush一段时间，则能够大大缩短 MySQL关闭的时间.



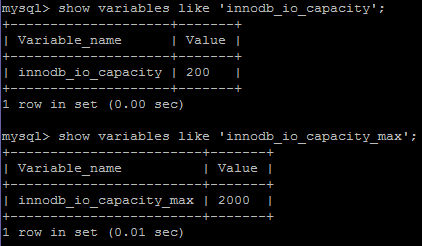
innodb\_lock\_wait\_timeout

InnoDB事务在被回滚之前可以等待一个锁定的超时秒数。InnoDB在它自己的锁定表中自动检测事务死锁并且回滚事务。InnoDB用LOCK TABLES语句注意到锁定设置。默认值是50秒。



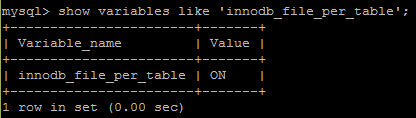
innodb\_io\_capacity 和 innodb\_io\_capacity\_max

这些设置会影响InnoDB每秒在后台执行多少操作. 在 以前的一篇文章 里我描述了大多数写IO(除了写InnoDB日志)是后台操作的. 如果你深度了解硬件性能(如每秒可以执行多少次IO操作),则使用这些功能是很可取的,而不是让它闲着. 有一个很简单的计算: 如果每个磁盘每秒读写(IOPS)可以达到 200次, 则拥有10个磁盘的 RAID10 磁盘阵列IOPS理论上 =(10/2)\* 200 = 1000. 我说它“很简单”,是因为RAID控制器通常能够提供额外的合并,并有效提高IOPS能力. 对于SSD磁盘,IOPS可以轻松达到好几千.将这两个值设置得太大可能会存在某些风险,你肯定不希望后台操作妨碍了前台任务IO操作的性能. 过去的经验表明,将这两个值设置的太高,InnoDB持有的内部锁会导致性能降低(按我了解到的信息,在MySQL5.6中这得到了很大的改进).



innodb\_file\_per\_table

这项设置告知InnoDB是否需要将所有表的数据和索引存放在共享表空间里（innodb\_file\_per\_table = OFF） 或者为每张表的数据单独放在一个.ibd文件（innodb\_file\_per\_table = ON）。每张表一个文件允许你在drop、truncate或者rebuild表时回收磁盘空间。这对于一些高级特性也是有必要的，比如数据压缩。但是它不会带来任何性能收益。你不想让每张表一个文件的主要场景是：有非常多的表（比如10k+）。这个属性默认值是ON，它只对新创建的表有影响。



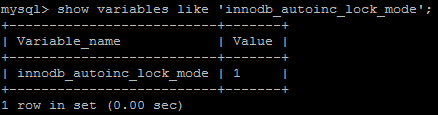
innodb\_lru\_scan\_depth

默认值为 1024. 这是mysql 5.6中引入的一个新选项. Mark Callaghan 提供了 一些配置建议. 简单来说,如果增大了 innodb\_io\_capacity 值, 应该同时增加 innodb\_lru\_scan\_depth.

innodb\_autoinc\_lock\_mode

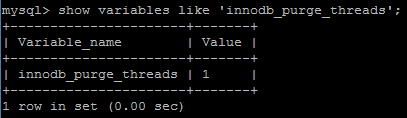
这是InnoDB为了解决自增主键锁表的问题，引入的参数。innodb使用新的方式解决自增字段一致性问题，对于可以预判行数的insert语句，innodb使用一个轻量级的互斥量。如：某一insert语句1执行前，表的AUTO\_INCREMENT=1，语句1的插入行数已知为3，innodb在语句1的实际插入操作执行前就预分配给该语句三个自增值，当有一个新的insert语句2要执行时，读取的AUTO\_INCREMENT=4，这样虽然语句1可能还没有执行完，语句2就可直接执行无需等待语句1。这种方式对于可预判插入行数的插入语句有效，如：insert和replace。对于无法提前获知插入行数的语句，如：insert...select...、replace...select...和load data则innodb还是使用表锁。insert语句中有时会显示的设置自增字段的值，对于这种情况innodb还是会预分配给语句总行数的自增值而不是只有实际使用系统自增的行。因而有可能会造成自增字段的值不连续。这种方式对于可预判插入行数的插入语句有效，如：insert和replace。对于无法提前获知插入行数的语句，如：insert...select...、replace...select...和load data则innodb还是使用表锁。INSERT ... ON DUPLICATE KEY UPDATE语句与之前一种情况类似，也会造成自增字段的值不连续,而且如果使用新的自增互斥方式，对于replication应该避免使用INSERT ... ON DUPLICATE KEY UPDATE语句。设置新自增互斥方式：通过配置选项：innodb\_autoinc\_lock\_mode，它是专门用来在使用auto\_increment的情况下调整锁策略的，目前有三种选择：innodb\_autoinc\_lock\_mode = 0 (“traditional” lock mode：全部使用表锁)；

innodb\_autoinc\_lock\_mode = 1 (默认)(“consecutive” lock mode：可预判行数时使用新方式，不可时使用表锁)；innodb\_autoinc\_lock\_mode = 2 (“interleaved” lock mode：全部使用新方式，不安全，不适合replication)



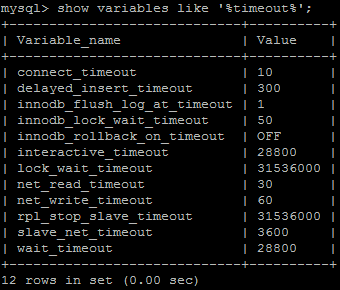
innodb\_purge\_threads

InnoDB中的清除操作是一类定期回收无用数据的操作，该项设置为 1 时表示使用单独的清除线程。默认即为1。



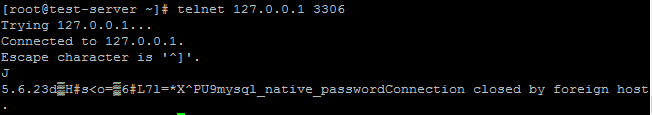
timeout

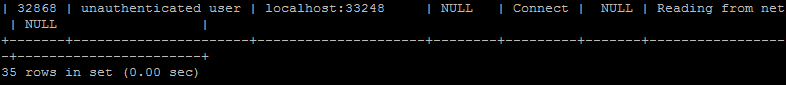
有关timeout相关变量有12个：



connect\_timeout

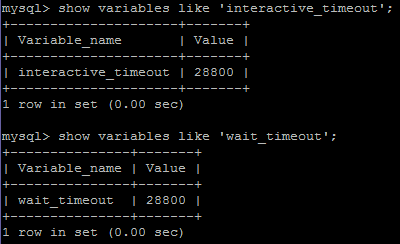
connect\_timeout指的是验证客户端连接过程中握手的超时时间，在5.0.52以后默认为10秒，之前版本默认是5秒。如果客户端请求连接，在10s内没有发生验证密码，服务器就会断开连接。





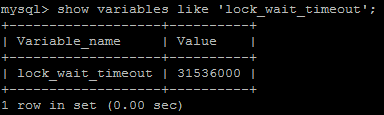
interactive\_timeout和wait\_timeout

这两个值默认都是28800s，即8小时。在MySQL中执行数据库操作主要有两种方式，interactive（交互式）和non-interactive（非交互式），当然执行命令会有超时时间（可能由于网络原因，数据没传递过来造成客户端超时；或由于服务端处理此操作非常耗时超过了服务端设置的超时时间造成的服务端超时），但是这里说的超时时间是指数据库连接在指定的时间内都没有被使用，过后使用时的超时。交互式超时——interactive\_timeout，即在交互式命令行界面执行sql时数据库连接长时间不用导致超时；非交互式超时——wait\_timeout，即如web应用的数据库连接（池）就属于这类，如果长时间都没有使用这条连接，就会报"the last packet successfully received from the server was xxx milliseconds ago"。这两个参数对于show processlist影响至深。



lock\_wait\_timeout

简单说来lock\_wait\_timeout是元数据锁等待超时，任意锁元数据的语句都会用到这个超时参数，默认为一年。元数据锁可以参加mysql metadata lock，为了保证事务可串行化，不管是myisam还是innodb引擎的表，只要是开始一个事务，就会获取操作表的元数据锁，这时候如果另一个事务要对表的元数据进行修改，则会阻塞直到超时。



net\_read\_timeout 和 net\_write\_timeout

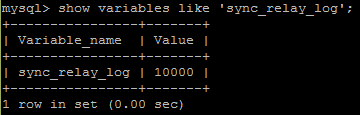
这两个参数在网络条件不好的情况下起作用。比如我在客户端用load data infile的方式导入很大的一个文件到数据库中，然后中途用iptables禁用掉mysql的3306端口，这个时候服务器端该连接状态是reading from net，在等待net\_read\_timeout后关闭该连接。同理，在程序里面查询一个很大的表时，在查询过程中同样禁用掉端口，制造网络不通的情况，这样该连接状态是writing to net，然后在net\_write\_timeout后关闭该连接。slave\_net\_timeout类似。

总结来说，connect\_timeout在握手认证阶段（authenticate）起作用，interactive\_timeout 和wait\_timeout在连接空闲阶段（sleep）起作用，而net\_read\_timeout和net\_write\_timeout则是在连接繁忙阶段（query）或者网络出现问题时起作用。参考：<http://www.jianshu.com/p/a5747a382a0a>

replication

sync\_relay\_log

启用二进制日志后， 默认情况下二进制日志不是事故安全的(not crash safe)。默认值竟是10000，可修改为sync\_relay\_log=1。

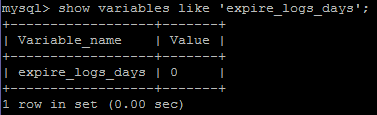


relay-log-info-repository=TABLE 和 master-info-repository=TABLE

配合sync\_relay\_log=1做出的优化

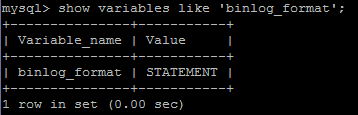
expire-logs-days

默认值是0，旧日志会一直保留. 我推荐设置为 1-10 天. 保存更长的时间并没有太多用处,因为从备份中恢复会快得多



binlog\_format

默认是STATEMENT，可修改为binlog\_format=ROW基于行的复制，可以通过减少资源锁定提高性能。阿里配置为mixed。



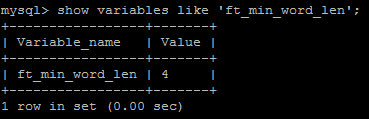
transaction-isolation

MySQL支持4种事务隔离级别，他们分别是： READ-UNCOMMITTED（读未提交）, READ-COMMITTED（读已提交）, REPEATABLE-READ（可重复读）, SERIALIZABLE（串行）.如没有指定，MySQL默认采用的是REPEATABLE-READ，ORACLE默认的是READ-COMMITTED

Other

ft\_min\_word\_len

被全文检索索引的最小的字长，默认是4。如果你需要搜索更短字的时候，可以减少它，但是修改此值后，需要重建你的 FULLTEXT 索引。



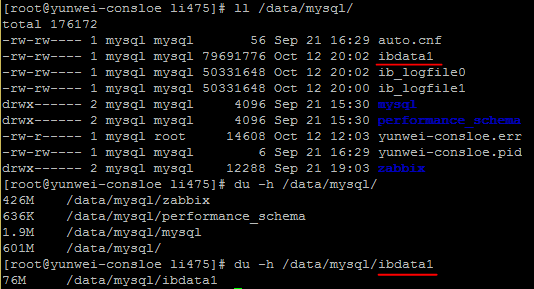
max\_allowed\_packet

服务所能处理的请求包的最大大小以及服务所能处理的最大的请求大小(当与大的BLOB字段一起工作时相当必要)，每个连接独立的大小。系统默认值 为4MB，最大值是1GB，必须设置1024的倍数。阿里mysql设为[mysqldump]项：max\_allowed\_packet = 16M

physical file

ibdata1

在mysql的数据目录下，固定有ibdata1文件，它是InnoDB 的共享表空间，非常占磁盘空间



当启用了 innodb\_file\_per\_table，表被存储在他们自己的表空间里，但是共享表空间仍然在存储其它的 InnoDB 内部数据：

数据字典，也就是 InnoDB 表的元数据

变更缓冲区

双写缓冲区

撤销日志

可以通过 innodb\_ibuf\_max\_size 设置最大变更缓冲区，或设置 innodb\_doublewrite\_file 来将双写缓冲区存储到一个分离的文件。MySQL 5.6 版中你也可以创建外部的撤销表空间，所以它们可以放到自己的文件来替代存储到 ibdata1。可以看看这个文档。

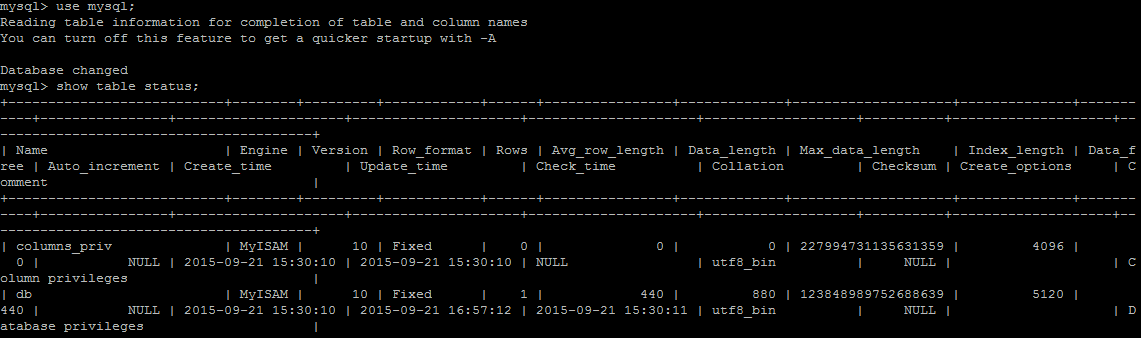
show status

zabbix数据库Root密码：Rot889@u!^V

显示一些系统特定资源的信息，例如，正在运行的线程数量

show table status;

查看该库的所有表的Engine、Row\_format、Avg\_row\_length、Data\_length、Max\_data\_length、Index\_length等等



show status 主要项目说明：

Aborted\_clients 由于客户没有正确关闭连接已经死掉，已经放弃的连接数量。

Aborted\_connects 尝试已经失败的MySQL服务器的连接的次数。

Connections 试图连接MySQL服务器的次数。

Created\_tmp\_tables 当执行语句时，已经被创造了的隐含临时表的数量。

Delayed\_insert\_threads 正在使用的延迟插入处理器线程的数量。

Delayed\_writes 用INSERT DELAYED写入的行数。

Delayed\_errors 用INSERT DELAYED写入的发生某些错误(可能重复键值)的行数。

Flush\_commands 执行FLUSH命令的次数。

Handler\_delete 请求从一张表中删除行的次数。

Handler\_read\_first 请求读入表中第一行的次数。

Handler\_read\_key 请求数字基于键读行。

Handler\_read\_next 请求读入基于一个键的一行的次数。

Handler\_read\_rnd 请求读入基于一个固定位置的一行的次数。

Handler\_update 请求更新表中一行的次数。

Handler\_write 请求向表中插入一行的次数。

Key\_blocks\_used 用于关键字缓存的块的数量。

Key\_read\_requests 请求从缓存读入一个键值的次数。

Key\_reads 从磁盘物理读入一个键值的次数。

Key\_write\_requests 请求将一个关键字块写入缓存次数。

Key\_writes 将一个键值块物理写入磁盘的次数。

Max\_used\_connections 同时使用的连接的最大数目。

Not\_flushed\_key\_blocks 在键缓存中已经改变但是还没被清空到磁盘上的键块。

Not\_flushed\_delayed\_rows 在INSERT DELAY队列中等待写入的行的数量。

Open\_tables 打开表的数量。

Open\_files 打开文件的数量。

Open\_streams 打开流的数量(主要用于日志记载）

Opened\_tables 已经打开的表的数量。

Questions 发往服务器的查询的数量。

Slow\_queries 要花超过long\_query\_time时间的查询数量。

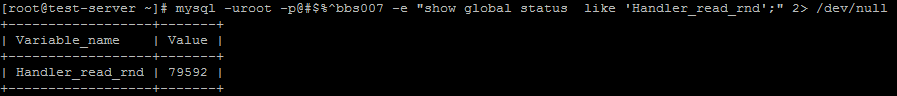
Threads\_connected 当前打开的连接的数量。

Threads\_running 不在睡眠的线程数量。

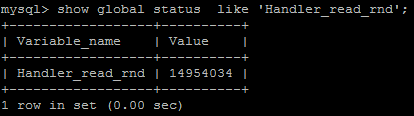
Uptime 服务器工作了多少秒。

Handler\_read\_rnd

show global status like 'Handler\_read\_rnd';



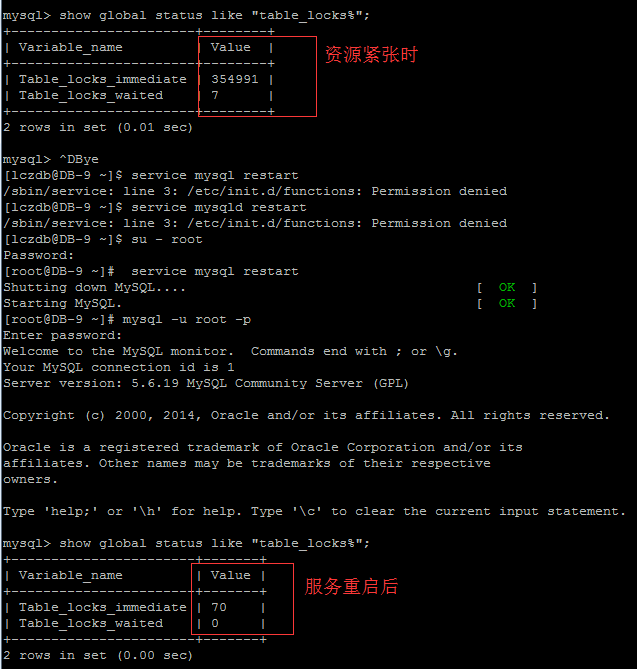
如果Handler\_read\_rnd太大，那么你很可能有大量的查询需要MySQL扫描整个表或你有没正确使用键值的联结(join)



table\_locks\_waited

show global status like "table\_locks%";

Table\_locks\_immediate 指的是能够立即获得表级锁的次数，而Table\_locks\_waited指的是不能立即获取表级锁而需要等待的次数。如果 Table\_locks\_waited的值比较大的话，并且你遇到了性能问题，你可能希望将大表切分成小表；例如，你可以为一个定制模块创建一个专有的缓存（cache）表，或者通过其它方式来减小表的大小，或者降低表级锁命令调用的频率。对于一些表，比如cache\_\*，Watchdog， 和 accesslog表，减少表的大小的一种方式是减少数据的生命周期。



show engine innodb status

监控对象：

mysql -uzabbix -pZabbix87@3#^G -e "show engine innodb status\G" 2>/dev/null | grep "History list length"|awk '{print $4}'

mysql -uzabbix -pZabbix87@3#^G -e "show engine innodb status\G" 2>/dev/null | grep "Last checkpoint at"|awk '{print $4}'

mysql -uzabbix -pZabbix87@3#^G -e "show engine innodb status\G" 2>/dev/null | grep "Log sequence number"|awk '{print $4}'

mysql -uzabbix -pZabbix87@3#^G -e "show engine innodb status\G" 2>/dev/null | grep "Log flushed up to"|awk '{print $5}'

mysql -uzabbix -pZabbix87@3#^G -e "show engine innodb status\G" 2>/dev/null | grep "read views open inside InnoDB"|awk '{print $1}'

mysql -uzabbix -pZabbix87@3#^G -e "show engine innodb status\G" 2>/dev/null | grep "queries inside InnoDB"|awk '{print $1}'

mysql -uzabbix -pZabbix87@3#^G -e "show engine innodb status\G" 2>/dev/null | grep "queries in queue"|awk '{print $5}'

mysql -uzabbix -pZabbix87@3#^G -e "show engine innodb status\G" 2>/dev/null | grep "hash searches"|awk '{print $1}'

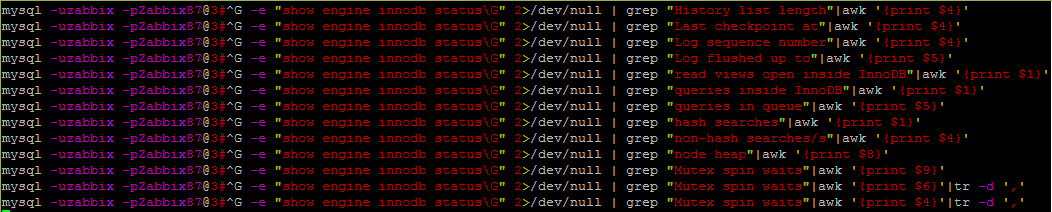
mysql -uzabbix -pZabbix87@3#^G -e "show engine innodb status\G" 2>/dev/null | grep "non-hash searches/s"|awk '{print $4}'

mysql -uzabbix -pZabbix87@3#^G -e "show engine innodb status\G" 2>/dev/null | grep "node heap"|awk '{print $8}'

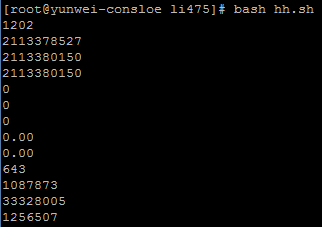
mysql -uzabbix -pZabbix87@3#^G -e "show engine innodb status\G" 2>/dev/null | grep "Mutex spin waits"|awk '{print $9}'

mysql -uzabbix -pZabbix87@3#^G -e "show engine innodb status\G" 2>/dev/null | grep "Mutex spin waits"|awk '{print $6}'|tr -d ','

mysql -uzabbix -pZabbix87@3#^G -e "show engine innodb status\G" 2>/dev/null | grep "Mutex spin waits"|awk '{print $4}'|tr -d ','



执行结果



“History list length（历史记录列表长度）”展示了一些等待清除操作。这种情况下问题经常发生，因为清除线程（或者老版本的主线程）不能像这些记录进来的速度一样快地处理撤销。

参考：

结合源码分析show engine innodb status输出内容(一)

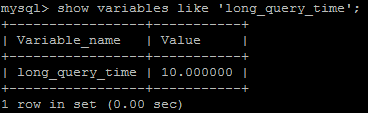
<http://blog.csdn.net/happy_life123/article/details/45971349>

show engine innodb status 详解

<http://blog.sina.com.cn/s/blog_73a36be80101cse2.html>

log\_slow\_queries

记录慢速查询. 慢速查询是指消耗了比 “long\_query\_time” 定义的更多时间的查询。如果 log\_long\_format 被打开，那些没有使用索引的查询也会被记录。在my.cnf加上”log\_slow\_queries“选项，即开启了慢查询。long\_query\_time默认是10s，所有的使用了比这个时间(以秒为单位)更多的查询会被认为是慢速查询.不要在这里使用”1″, 否则会导致所有的查询,甚至非常快的查询页被记录下来(由于MySQL 目前时间的精确度只能达到秒的级别)

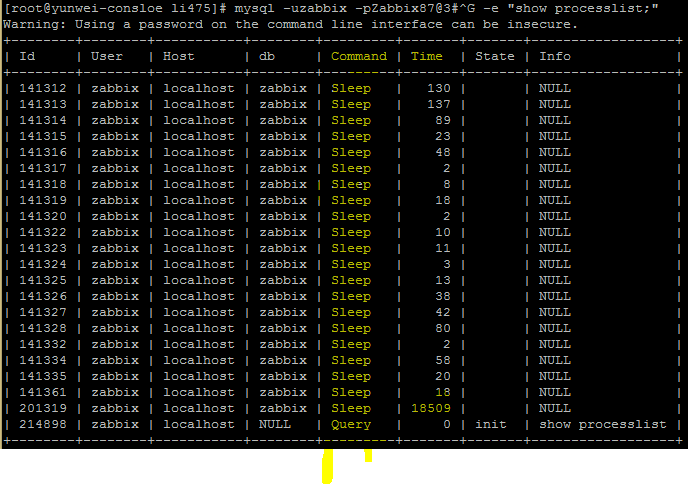


log\_long\_format

打开此项会记录使得那些没有使用索引的查询也被作为到慢速查询附加到慢速日志里，从而在慢速日志中记录更多的信息.

show processlist

processlist命令的输出结果显示了有哪些线程在运行，而mysql默认为一个连接数创建一个线程，线程数即连接数



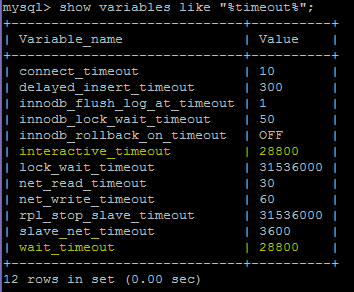
command列，显示当前连接的执行的命令，一般就是休眠（sleep），查询（query），连接（connect）。time列，此这个状态持续的时间，单位是秒。state列，显示使用当前连接的sql语句的状态，很重要的列，后续会有所有的状态的描述，请注意，state只是语句执行中的某一个状态，一个sql语句，已查询为例，可能需要经过copying to tmp table，Sorting result，Sending data等状态才可以完成，info列，显示这个sql语句，因为长度有限，所以长的sql语句就显示不全。

通过以上的信息，我们知道了该数据库有22个连接，其中有21个线程处于Sleep状态，而有一个线程处理时间长达18508s



这会促进我们的思考：1）为什么这些线程会处于Sleep状态怎么久？2）如果这些Sleep的线程已经无效，能否kill掉？

Mysql线程销毁的时间，由interactive\_timeout和wait\_timeout两个参数决定：



MySQL中的配置参数interactive\_timeout和wait\_timeout(可能导致过多sleep进程的两个参数)

1)interactive\_timeout:

参数含义：服务器关闭交互式连接前等待活动的秒数。交互式客户端定义为在mysql\_real\_connect()中使用CLIENT\_INTERACTIVE选项的客户端。

参数默认值：28800秒（8小时）

(2)wait\_timeout:

参数含义：服务器关闭非交互连接之前等待活动的秒数。

在线程启动时，根据全局wait\_timeout值或全局interactive\_timeout值初始化会话wait\_timeout值，取决于客户端类型(由mysql\_real\_connect()的连接选项CLIENT\_INTERACTIVE定义)。

参数默认值：28800秒（8小时）

MySQL服务器所支持的最大连接数是有上限的，因为每个连接的建立都会消耗内存，因此我们希望客户端在连接到MySQL Server处理完相应的操作后，应该断开连接并释放占用的内存。如果你的MySQL Server有大量的闲置连接，他们不仅会白白消耗内存，而且如果连接一直在累加而不断开，最终肯定会达到MySQL Server的连接上限数，这会报'too many connections'的错误。对于wait\_timeout的值设定，应该根据系统的运行情况来判断。在系统运行一段时间后，可以通过show processlist命令查看当前系统的连接状态，如果发现有大量的sleep状态的连接进程，则说明该参数设置的过大，可以进行适当的调整小些。

问题：

如果在配置文件my.cnf中只设置参数wait\_timeout=100,则重启服务器后进入，执行:

Mysql> show variables like “%timeout%”;

会发现参数设置并未生效，仍然为28800（即默认的8个小时）。

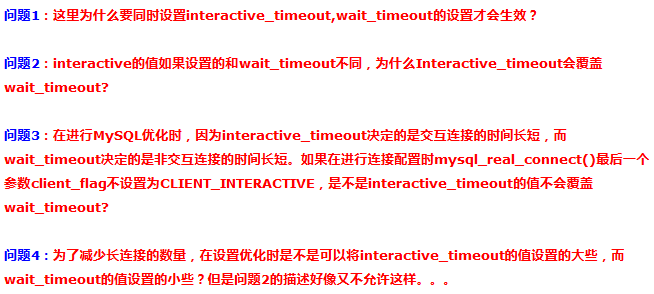
查询资料后，要同时设置interactive\_timeout和wait\_timeout才会生效。

【mysqld】

wait\_timeout=100

interactive\_timeout=100

重启MySQL Server进入后，查看设置已经生效。



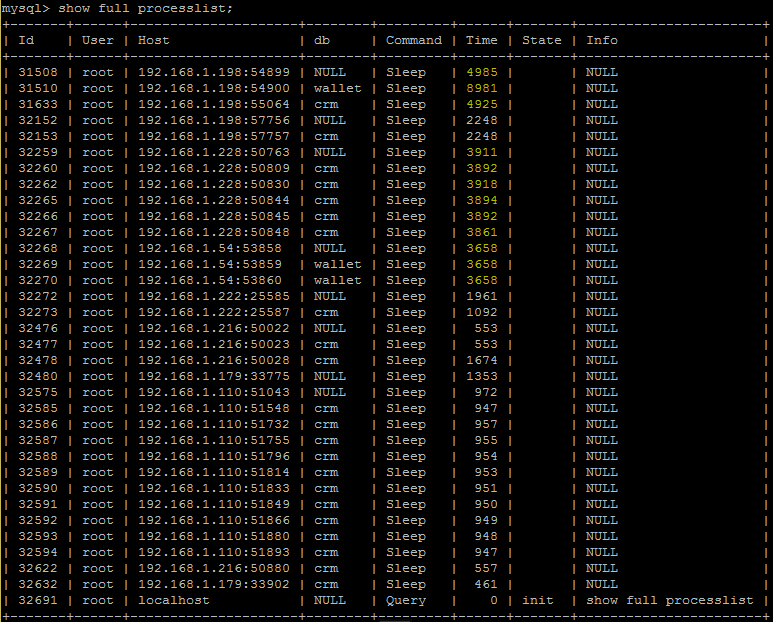
参考：<http://www.cnblogs.com/jiunadianshi/articles/2475475.html>

也可以临时修改

set global interactive\_timeout = 300;

set global wait\_timeout = 300;

这是内部测试环境mysql的线程状态



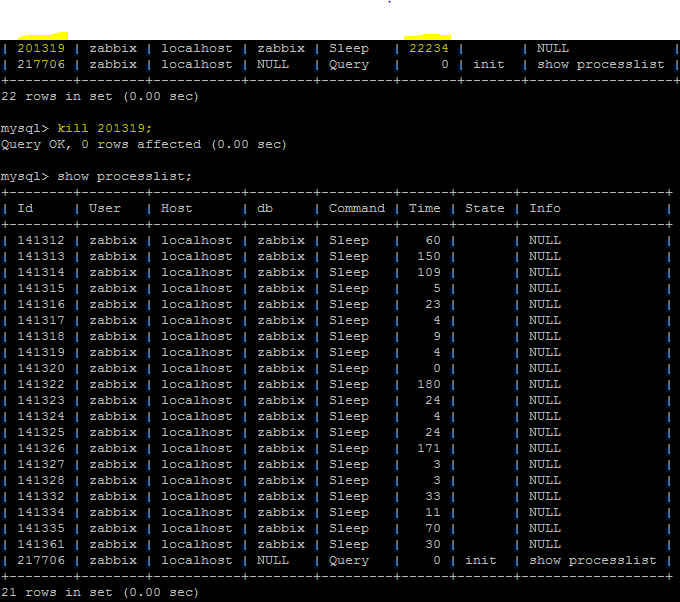
连接数有34个



参考：http://cenalulu.github.io/mysql/mysql-timeout/



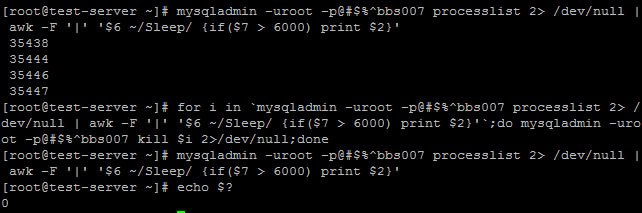
Kill线程很简单，确定该死的线程的pid后，执行kill命令



能不能批量杀死线程呢，当然可以

For循环批量杀死Time > 6000s的线程：

for i in `mysqladmin -uroot -p@#$%^bbs007 processlist 2> /dev/null | awk -F '|' '$6 ~/Sleep/ {if($7 > 6000) print $2}'`;do mysqladmin -uroot -p@#$%^bbs007 kill $i 2>/dev/null;done

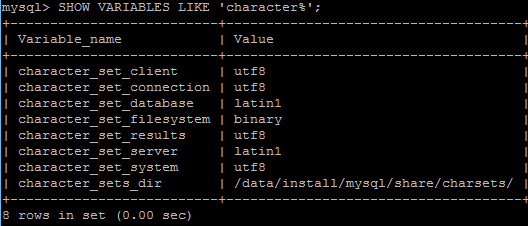


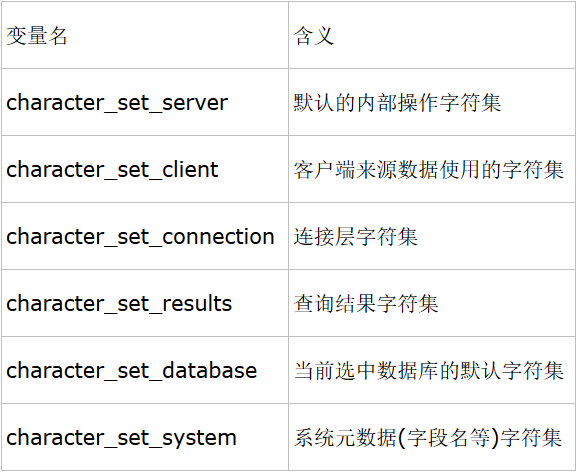
mysql字符集

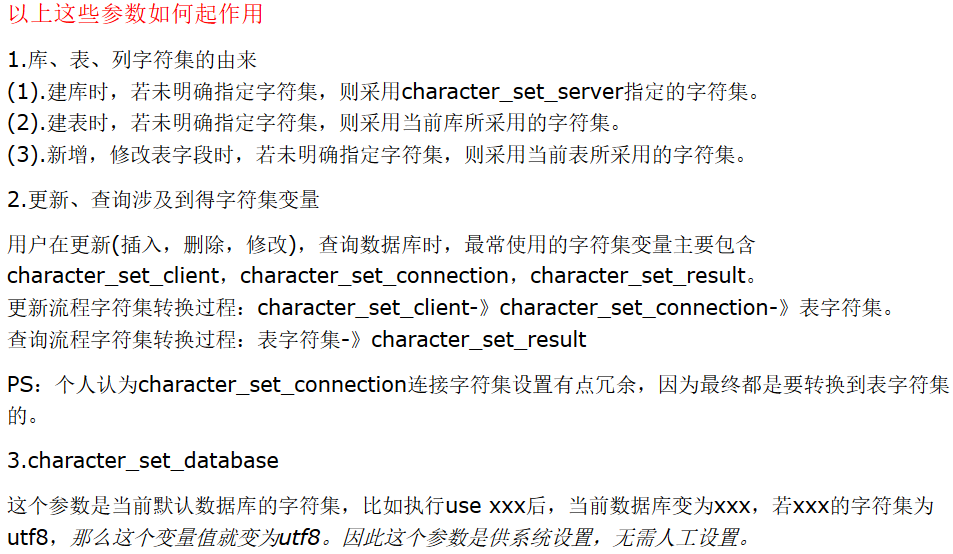
show character set; #查看mysql支持的全部字符集

SHOW COLLATION; #查看mysql的字符序列(排序方式)

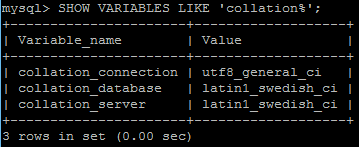
SHOW VARIABLES LIKE 'character%'; 查看mysql系统默认字符集设置值



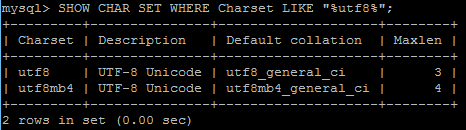




SHOW VARIABLES LIKE 'collation%'; 查看mysql系统默认的字符序列(排序方式) 设置值



SHOW CHAR SET WHERE Charset LIKE "%utf8%"; 确认mysql server是否支持 utf8mb4 编码

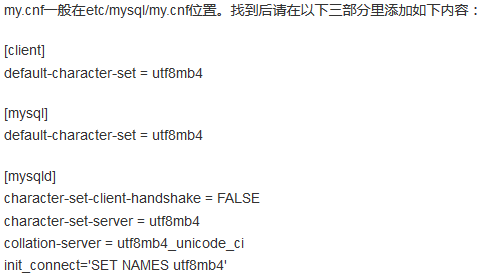


修改database、table和column（字段）字符集

ALTER DATABASE database\_name CHARACTER SET = utf8mb4 COLLATE = utf8mb4\_unicode\_ci;

ALTER TABLE table\_name CONVERT TO CHARACTER SET utf8mb4 COLLATE utf8mb4\_unicode\_ci;

ALTER TABLE table\_name CHANGE column\_name VARCHAR(191) CHARACTER SET utf8mb4 COLLATE utf8mb4\_unicode\_ci;



在线set golbal character\_set\_database=utf8mb4;

注意事项

• my.cnf中的default\_character\_set设置只影响mysql命令连接服务器时的连接字符集，不会对使用libmysqlclient库的应用程序产生任何作用！

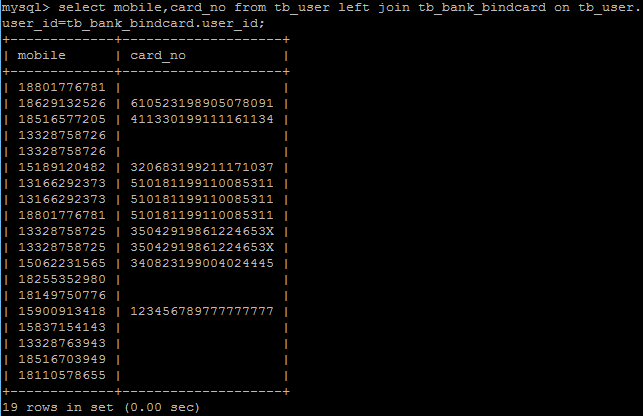
• 对字段进行的SQL函数操作通常都是以内部操作字符集进行的，不受连接字符集设置的影响。

• SQL语句中的裸字符串会受到连接字符集或introducer设置的影响，对于比较之类的操作可能产生完全不同的结果，需要小心！

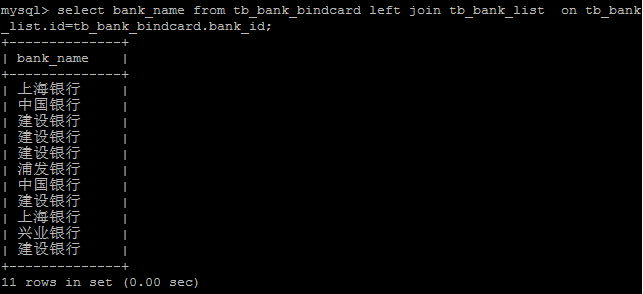
SQL语句基础

left join

select mobile,card\_no from tb\_user left join tb\_bank\_bindcard on tb\_user.user\_id=tb\_bank\_bindcard.user\_id;

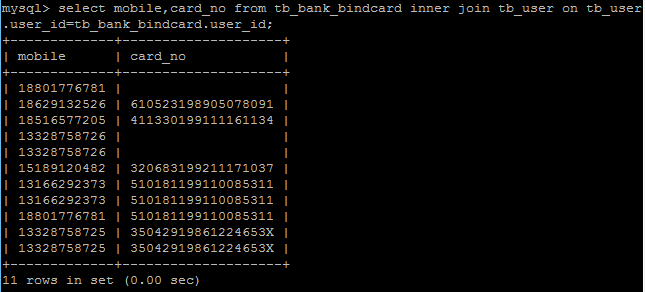


select bank\_name from tb\_bank\_bindcard left join tb\_bank\_list on tb\_bank\_list.id=tb\_bank\_bindcard.bank\_id;

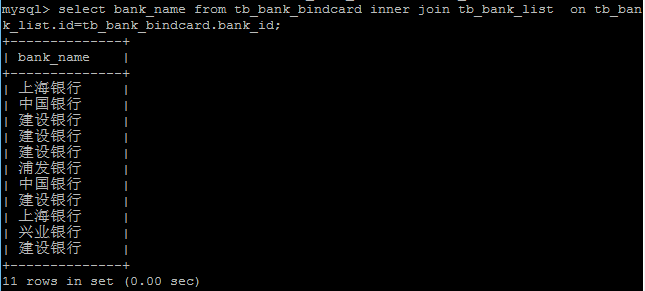


Inner join

select mobile,card\_no from tb\_bank\_bindcard inner join tb\_user on tb\_user.user\_id=tb\_bank\_bindcard.user\_id;

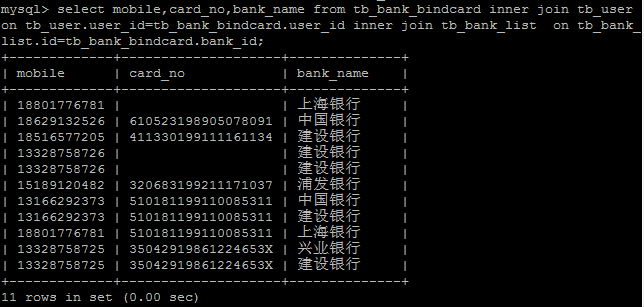


select bank\_name from tb\_bank\_bindcard inner join tb\_bank\_list on tb\_bank\_list.id=tb\_bank\_bindcard.bank\_id;



三表查询

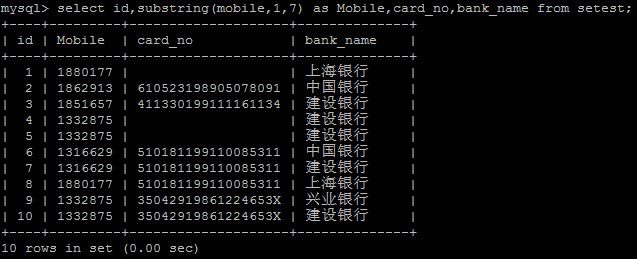
select mobile,card\_no,bank\_name from tb\_bank\_bindcard inner join tb\_user on tb\_user.user\_id=tb\_bank\_bindcard.user\_id inner join tb\_bank\_list on tb\_bank\_list.id=tb\_bank\_bindcard.bank\_id;



substring

截取列值，从第1位到第7位

select id,substring(mobile,1,7) as Mobile,card\_no,bank\_name from setest;



Trouble shutting

1. 主从slave重启报错

首先确定master和slave在my.cnf的log-bin=mysqlmaster-bin.log配置相同，在slave上执行reset salve;

master\_host='10.162.106.134',master\_user='slave',master\_password='BZ\_salve007',master\_log\_file='mysqlmaster-bin.000003',master\_log\_pos=2658;

start slave;

OK，问题解决

MySQL自动化运维工具

[一个集审核、执行、备份及生成回滚语句于一身的MySQL自动化运维工具](https://github.com/mysql-inception/inception)

MySQL级联复制(A->B->C)

mysql置成级联复制，要特别注意第二级mysql配置开启log\_slave\_updates，log\_slave\_updates是将从服务器从主服务器收到的更新记入到从服务器自己的二进制日志文件中。

在配置文件里加上“log-slave-updates”

