MySQL数据库引擎

介绍:

存储引擎概念

MySQL中的数据用各种不同的技术存储在文件（或者内存）中。这些技术中的每一种技术都使用不同的存储机制、索引技巧、锁定水平并且最终提供广泛的不同的功能和能力。通过选择不同的技术，你能够获得额外的速度或者功能，从而改善你的应用的整体功能。

引擎（engine）能够决定程序管理和数据操作的程序或程序段。数据库引擎就是操作数据库的一段程序或程序段

简单说来，一个数据库的存储引擎就是用来对数据表进行信息存储、处理、查询等操作。

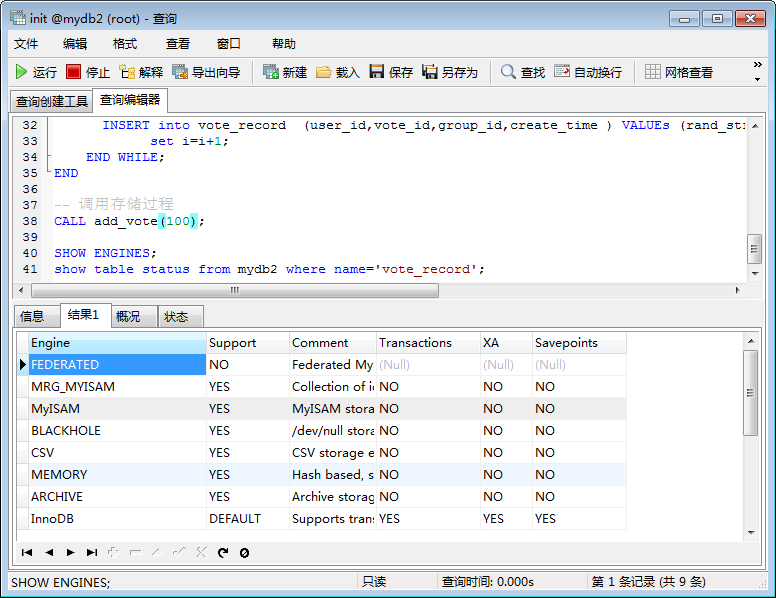
MySQL数据库引擎取决于MySQL在安装的时候是如何被编译的。要添加一个新的引擎，就必须重新编译MYSQL。在缺省情况下，MYSQL支持三个引擎：ISAM、MYISAM和HEAP。另外两种类型INNODB和BERKLEY（BDB），也常常可以使用。

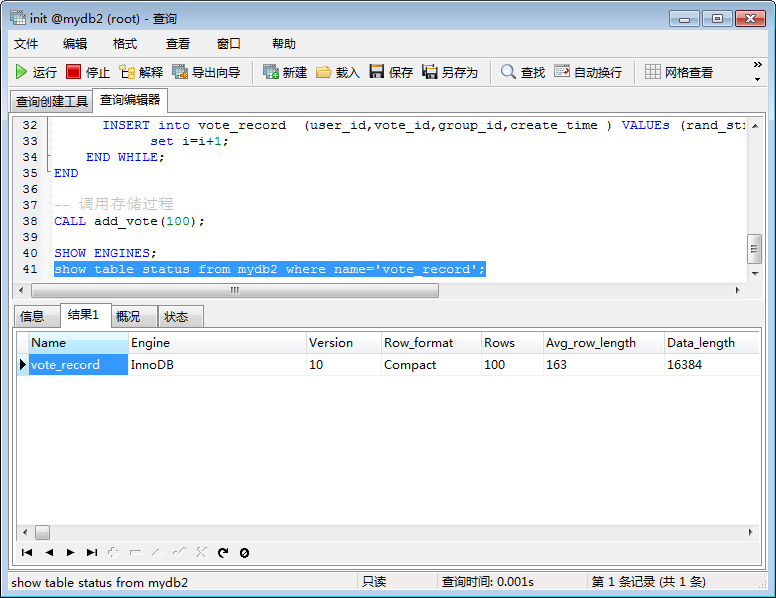
5.1可以使用innodb插件  
5.5.5以上默认Engine是Innodb，其他版本默认是MyISAM

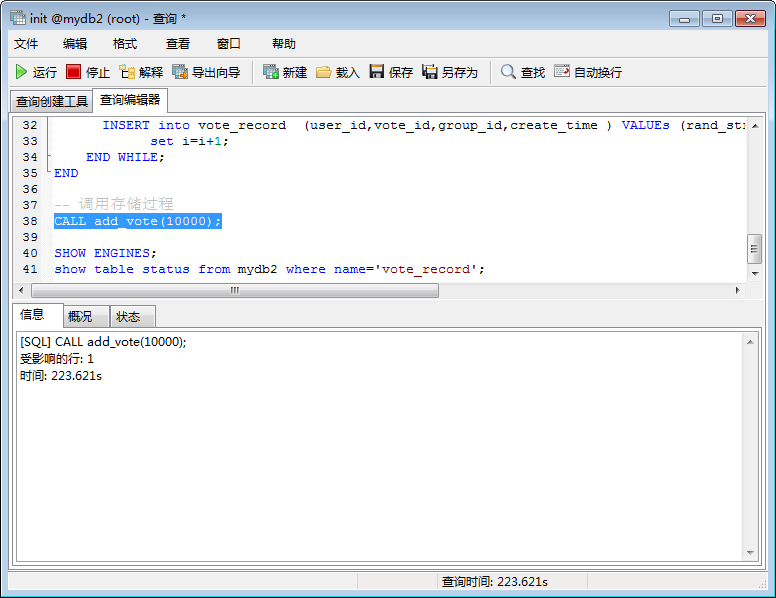
MyISAM是MySQL的默认[数据库](http://lib.csdn.net/base/mysql" \o "MySQL知识库" \t "http://blog.csdn.net/u014082617/article/details/_blank)引擎（5.5版之前），由早期的ISAM（Indexed Sequential Access Method：有索引的顺序访问方法）所改良。虽然性能极佳，但却有一个缺点：不支持事务处理（transaction）。不过，在这几年的发展下，MySQL也导入了InnoDB（另一种数据库引擎），以强化参考完整性与并发违规处理机制，后来就逐渐取代MyISAM。

InnoDB，是MySQL的数据库引擎之一，为MySQL AB发布binary的标准之一。InnoDB由Innobase Oy公司所开发，2006年五月时由甲骨文公司并购。与传统的ISAM与MyISAM相比，InnoDB的最大特色就是支持了ACID兼容的事务（Transaction）功能，类似于PostgreSQL。目前InnoDB采用双轨制授权，一是GPL授权，另一是专有软件授权。

1. MyISAM
2. InnoDB
3. HEAP
4. INNODB
5. BERKLEYDB

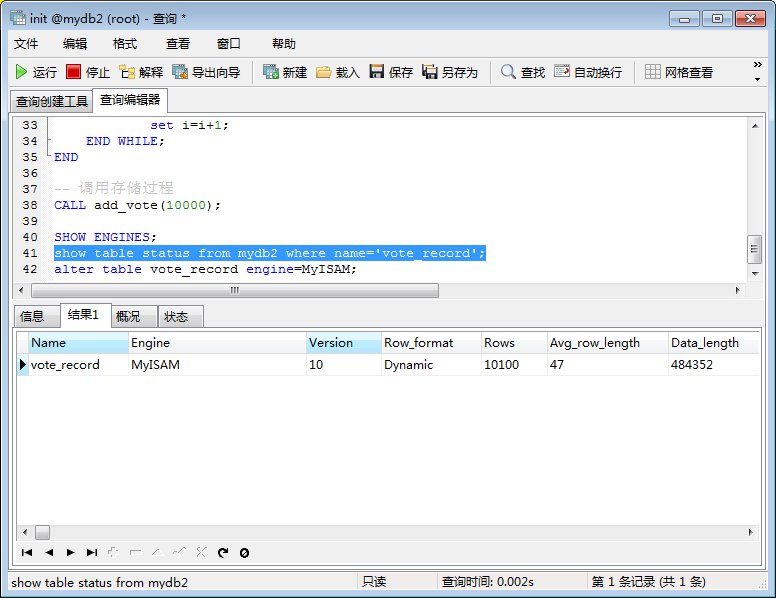


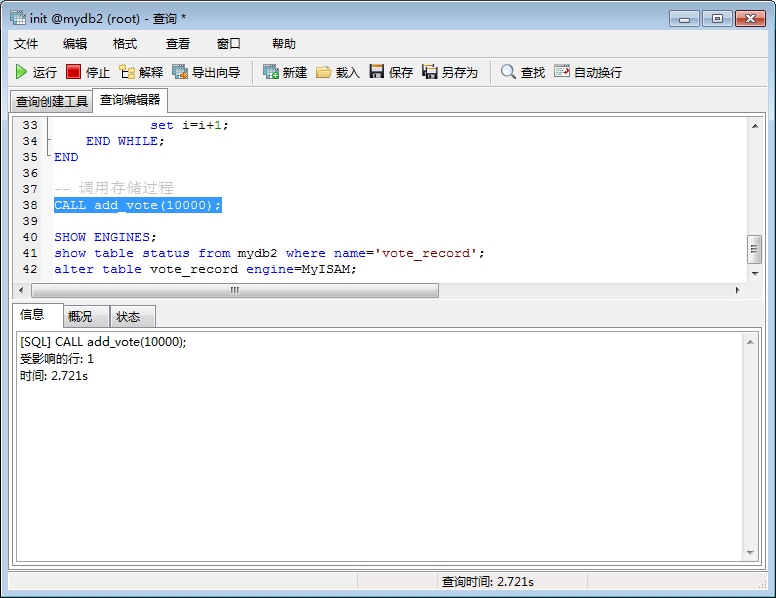




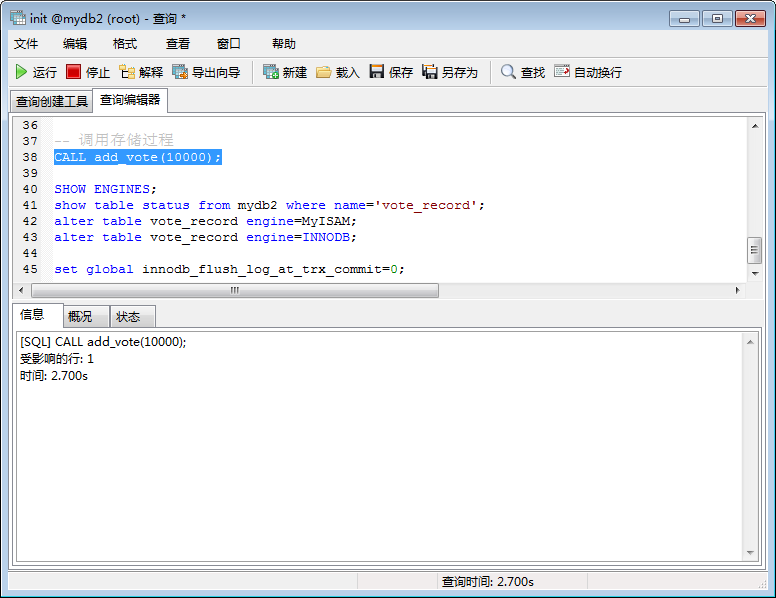
alter table vote\_record engine=MyISAM;

show table status from mydb2 where name='vote\_record';

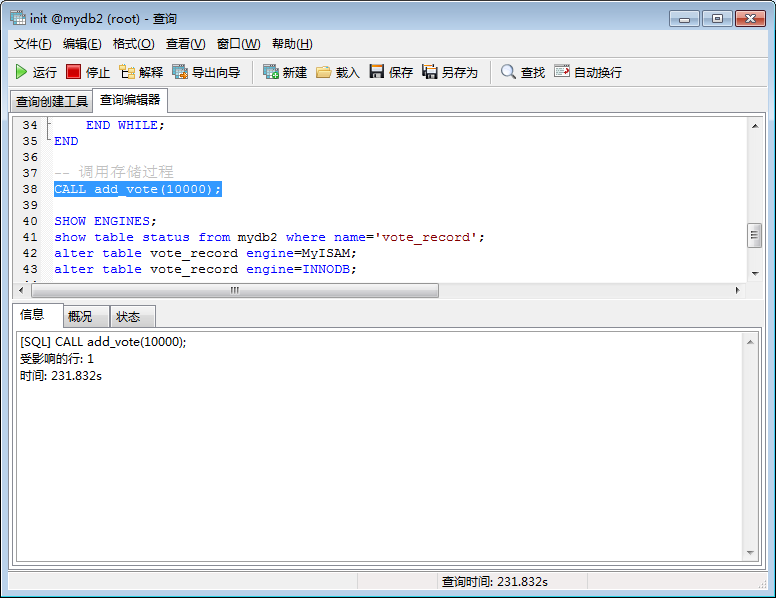




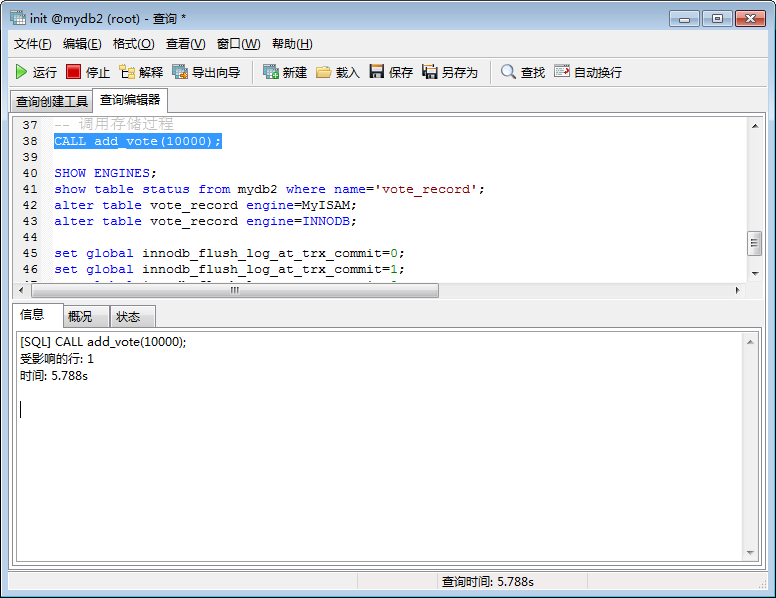
set global innodb\_flush\_log\_at\_trx\_commit=0;



set global innodb\_flush\_log\_at\_trx\_commit=1;



set global innodb\_flush\_log\_at\_trx\_commit=2;



## 配置项说明

### 0

如果innodb\_flush\_log\_at\_trx\_commit的值为0,log buffer每秒就会被刷写日志文件到磁盘，提交事务的时候不做任何操作。

### 1

当设为默认值1的时候，每次提交事务的时候，都会将log buffer刷写到日志。

### 2

如果设为2,每次提交事务都会写日志，但并不会执行刷的操作。每秒定时会刷到日志文件。要注意的是，并不能保证100%每秒一定都会刷到磁盘，这要取决于进程的调度。

默认值1是为了保证完整的[ACID](http://en.wikipedia.org/wiki/ACID" \t "http://blog.csdn.net/liqfyiyi/article/details/_blank)。当然，你可以将这个配置项设为1以外的值来换取更高的性能，但是在系统崩溃的时候，你将会丢失1秒的数据。设为0的话，mysqld进程崩溃的时候，就会丢失最后1秒的事务。设为2,只有在**[操作系统](http://lib.csdn.net/base/operatingsystem" \o "操作系统知识库" \t "http://blog.csdn.net/liqfyiyi/article/details/_blank)**崩溃或者断电的时候才会丢失最后1秒的数据。InnoDB在做恢复的时候会忽略这个值。

提升InnoDB性能的方法：   
MyISAM和InnoDB存储引擎性能差别并不是很大，针对InnoDB来说，影响性能的主要是 innodb\_flush\_log\_at\_trx\_commit 这个选项，如果设置为1的话，那么每次插入数据的时候都会自动提交，导致性能急剧下降，应该是跟刷新日志有关系，设置为0效率能够看到明显提升，当然，同 样你可以SQL中提交“SET AUTOCOMMIT = 0”来设置达到好的性能。另外，还听说通过设置innodb\_buffer\_pool\_size能够提升InnoDB的性能，但是我测试发现没有特别明显 的提升。

1. 下面是对线上mysql5.6版本的数据库的配置进行的优化分析记录：  
   1）内存利用方面：  
   innodb\_buffer\_pool\_size   
   这个是Innodb最重要的参数，和MyISAM的key\_buffer\_size有相似之处，但也是有差别的。   
   这个参数主要缓存innodb表的索引，数据，插入数据时的缓冲。  
   该参数分配内存的原则：  
   这个参数默认分配只有8M，可以说是非常小的一个值。  
   如果是一个专用DB服务器，那么他可以占到内存的70%-80%。  
   这个参数不能动态更改，所以分配需多考虑。分配过大，会使Swap占用过多，致使Mysql的查询特慢。  
   如果你的数据比较小，那么可分配是你的数据大小+10%左右做为这个参数的值。  
   例如：数据大小为50M,那么给这个值分配innodb\_buffer\_pool\_size＝64M  
   设置方法，在my.cnf文件里：  
   innodb\_buffer\_pool\_size=4G

注意：  
在Mysql5.7版本之前，调整innodb\_buffer\_pool\_size大小必须在my.cnf配置里修改，然后重启mysql进程才可以生效。  
如今到了Mysql5.7版本，就可以直接动态调整这个参数，方便了很多。

尤其是在服务器内存增加之后，运维人员不能粗心大意，要记得调大Innodb\_Buffer\_Pool\_size这个参数。  
数据库配置后，要注意检查Innodb\_Buffer\_Pool\_size这个参数的设置是否合理

需要注意的地方：  
在调整innodb\_buffer\_pool\_size 期间，用户的请求将会阻塞，直到调整完毕，所以请勿在白天调整，在凌晨3-4点低峰期调整。  
调整时，内部把数据页移动到一个新的位置，单位是块。如果想增加移动的速度，需要调整innodb\_buffer\_pool\_chunk\_size参数的大小，默认是128M。

innodb\_additional\_mem\_pool\_size   
用来存放Innodb的内部目录，这个值不用分配太大，系统可以自动调。通常设置16Ｍ够用了，如果表比较多，可以适当的增大。  
设置方法，在my.cnf文件里：  
innodb\_additional\_mem\_pool\_size = 16M

2）关于日志方面：  
innodb\_log\_file\_size  
作用：指定在一个日志组中，每个log的大小。  
结合innodb\_buffer\_pool\_size设置其大小，25%-100%。避免不需要的刷新。  
注意：这个值分配的大小和数据库的写入速度，事务大小，异常重启后的恢复有很大的关系。一般取256M可以兼顾性能和recovery的速度。  
分配原则：几个日值成员大小加起来差不多和你的innodb\_buffer\_pool\_size相等。上限为每个日值上限大小为4G.一般控制在几个Log文件相加大小在2G以内为佳。具体情况还需要看你的事务大小，数据大小为依据。  
说明：这个值分配的大小和数据库的写入速度，事务大小，异常重启后的恢复有很大的关系。  
设置方法：在my.cnf文件里：  
innodb\_log\_file\_size = 256M

innodb\_log\_files\_in\_group   
作用：指定你有几个日值组。  
分配原则：　一般我们可以用2-3个日值组。默认为两个。  
设置方法：在my.cnf文件里：  
innodb\_log\_files\_in\_group=3

innodb\_log\_buffer\_size：  
作用：事务在内存中的缓冲，也就是日志缓冲区的大小， 默认设置即可，具有大量事务的可以考虑设置为16M。  
如果这个值增长过快，可以适当的增加innodb\_log\_buffer\_size  
另外如果你需要处理大理的TEXT，或是BLOB字段，可以考虑增加这个参数的值。  
设置方法：在my.cnf文件里：  
innodb\_log\_buffer\_size=3M

innodb\_flush\_logs\_at\_trx\_commit  
作用：控制事务的提交方式,也就是控制log的刷新到磁盘的方式。  
分配原则：这个参数只有3个值（0，1，2）.默认为1，性能更高的可以设置为0或是2，这样可以适当的减少磁盘IO（但会丢失一秒钟的事务。），游戏库的MySQL建议设置为0。主库请不要更改了。  
其中：  
0：log buffer中的数据将以每秒一次的频率写入到log file中，且同时会进行文件系统到磁盘的同步操作，但是每个事务的commit并不会触发任何log buffer 到log file的刷新或者文件系统到磁盘的刷新操作；  
1：（默认为1）在每次事务提交的时候将logbuffer 中的数据都会写入到log file，同时也会触发文件系统到磁盘的同步；  
2：事务提交会触发log buffer 到log file的刷新，但并不会触发磁盘文件系统到磁盘的同步。此外，每秒会有一次文件系统到磁盘同步操作。  
说明：  
这个参数的设置对Ｉｎｎｏｄｂ的性能有很大的影响，所以在这里给多说明一下。  
当这个值为1时：innodb 的事务LOG在每次提交后写入日值文件，并对日值做刷新到磁盘。这个可以做到不丢任何一个事务。  
当这个值为2时：在每个提交，日志缓冲被写到文件，但不对日志文件做到磁盘操作的刷新,在对日志文件的刷新在值为2的情况也每秒发生一次。但需要注意的是，由于进程调用方面的问题，并不能保证每秒１００％的发生。从而在性能上是最快的。但操作系统崩溃或掉电才会删除最后一秒的事务。  
当这个值为0时：日志缓冲每秒一次地被写到日志文件，并且对日志文件做到磁盘操作的刷新，但是在一个事务提交不做任何操作。mysqld进程的崩溃会删除崩溃前最后一秒的事务。   
从以上分析，当这个值不为１时，可以取得较好的性能，但遇到异常会有损失，所以需要根据自已的情况去衡量。   
设置方法：在my.cnf文件里：  
innodb\_flush\_logs\_at\_trx\_commit=1

3）文件IO分配，空间占用方面  
innodb\_file\_per\_table  
作用：使每个Innodb的表，有自已独立的表空间。如删除文件后可以回收那部分空间。默认是关闭的，建议打开（innodb\_file\_per\_table=1）  
分配原则：只有使用不使用。但DB还需要有一个公共的表空间。  
设置方法：在my.cnf文件里：  
innodb\_file\_per\_table=1

innodb\_file\_io\_threads  
作用：文件读写IO数，这个参数只在Windows上起作用。在Linux上只会等于4，默认即可！  
设置方法：在my.cnf文件里：  
innodb\_file\_io\_threads=4

innodb\_open\_files  
作用：限制Innodb能打开的表的数据。  
分配原则：这个值默认是300。如果库里的表特别多的情况，可以适当增大为1000。innodb\_open\_files的大小对InnoDB效率的影响比较小。但是在InnoDBcrash的情况下，innodb\_open\_files设置过小会影响recovery的效率。所以用InnoDB的时候还是把innodb\_open\_files放大一些比较合适。  
设置方法：在my.cnf文件里：  
innodb\_open\_files=800

innodb\_data\_file\_path   
指定表数据和索引存储的空间，可以是一个或者多个文件。最后一个数据文件必须是自动扩充的，也只有最后一个文件允许自动扩充。这样，当空间用完后，自动扩充数据文件就会自动增长（以8MB为单位）以容纳额外的数据。  
例如： innodb\_data\_file\_path=/disk1/ibdata1:900M;/disk2/ibdata2:50M:autoextend 两个数据文件放在不同的磁盘上。数据首先放在ibdata1 中，当达到900M以后，数据就放在ibdata2中。  
设置方法，在my.cnf文件里：  
innodb\_data\_file\_path =ibdata1:1G;ibdata2:1G;ibdata3:1G;ibdata4:1G;ibdata5:1G;ibdata6:1G:autoextend

innodb\_data\_home\_dir   
放置表空间数据的目录，默认在mysql的数据目录，设置到和MySQL安装文件不同的分区可以提高性能。  
设置方法，在my.cnf文件里：（比如mysql的数据目录是/data/mysql/data，这里可以设置到不通的分区/home/mysql下）  
innodb\_data\_home\_dir = /home/mysql

4）其它相关参数（适当的增加table\_cache）   
这里说明一个比较重要的参数：  
innodb\_flush\_method  
作用：Innodb和系统打交道的一个IO模型  
分配原则：  
Windows不用设置。  
linux可以选择：O\_DIRECT   
直接写入磁盘，禁止系统Cache了  
设置方法：在my.cnf文件里：  
innodb\_flush\_method=O\_DIRECT

innodb\_max\_dirty\_pages\_pct   
作用：在buffer pool缓冲中，允许Innodb的脏页的百分比，值在范围1-100,默认为90，建议保持默认。  
这个参数的另一个用处：当Innodb的内存分配过大，致使Swap占用严重时，可以适当的减小调整这个值，使达到Swap空间释放出来。建义：这个值最大在90%，最小在15%。太大，缓存中每次更新需要致换数据页太多，太小，放的数据页太小，更新操作太慢。  
设置方法：在my.cnf文件里：  
innodb\_max\_dirty\_pages\_pct＝90  
动态更改需要有管理员权限：  
set global innodb\_max\_dirty\_pages\_pct=50;

innodb\_thread\_concurrency  
同时在Innodb内核中处理的线程数量。建议默认值。  
设置方法，在my.cnf文件里：  
innodb\_thread\_concurrency = 16

5）公共参数调优  
skip-external-locking  
MyISAM存储引擎也同样会使用这个参数，MySQL4.0之后，这个值默认是开启的。  
作用是避免MySQL的外部锁定(老版本的MySQL此参数叫做skip-locking)，减少出错几率增强稳定性。建议默认值。  
设置方法，在my.cnf文件里：  
skip-external-locking

skip-name-resolve   
禁止MySQL对外部连接进行DNS解析（默认是关闭此项设置的，即默认解析DNS），使用这一选项可以消除MySQL进行DNS解析的时间。  
但需要注意，如果开启该选项，则所有远程主机连接授权都要使用IP地址方式，否则MySQL将无法正常处理连接请求！如果需要，可以设置此项。  
设置方法，在my.cnf文件里：（我这线上mysql数据库中打开了这一设置）  
skip-name-resolve

max\_connections   
设置最大连接（用户）数，每个连接MySQL的用户均算作一个连接，max\_connections的默认值为100。此值需要根据具体的连接数峰值设定。  
设置方法，在my.cnf文件里：  
max\_connections = 3000

query\_cache\_size   
查询缓存大小，如果表的改动非常频繁，或者每次查询都不同，查询缓存的结果会减慢系统性能。可以设置为0。  
设置方法，在my.cnf文件里：  
query\_cache\_size = 512M

sort\_buffer\_size   
connection级的参数，排序缓存大小。一般设置为2-4MB即可。  
设置方法，在my.cnf文件里：  
sort\_buffer\_size = 1024M

read\_buffer\_size   
connection级的参数。一般设置为2-4MB即可。  
设置方法，在my.cnf文件里：  
read\_buffer\_size = 1024M

max\_allowed\_packet  
网络包的大小，为避免出现较大的网络包错误，建议设置为16M  
设置方法，在my.cnf文件里：  
max\_allowed\_packet = 16M

table\_open\_cache  
当某一连接访问一个表时，MySQL会检查当前已缓存表的数量。如果该表已经在缓存中打开，则会直接访问缓存中的表，以加快查询速度；如果该表未被缓存，则会将当前的表添加进缓存并进行查询。  
通过检查峰值时间的状态值Open\_tables和Opened\_tables，可以决定是否需要增加table\_open\_cache的值。  
如果发现open\_tables等于table\_open\_cache，并且opened\_tables在不断增长，那么就需要增加table\_open\_cache的值;设置为512即可满足需求。  
设置方法，在my.cnf文件里：  
table\_open\_cache = 512

myisam\_sort\_buffer\_size  
实际上这个myisam\_sort\_buffer\_size参数意义不大，这是个字面上蒙人的参数，它用于ALTER TABLE, OPTIMIZE TABLE, REPAIR TABLE 等命令时需要的内存。默认值即可。  
设置方法，在my.cnf文件里：  
myisam\_sort\_buffer\_size = 8M

thread\_cache\_size   
线程缓存，如果一个客户端断开连接，这个线程就会被放到thread\_cache\_size中（缓冲池未满），SHOW STATUS LIKE 'threads%';如果 Threads\_created 不断增大，那么当前值设置要改大，改到 Threads\_connected 值左右。（通常情况下，这个值改善性能不大），默认8即可  
设置方法，在my.cnf文件里：  
thread\_cache\_size = 8

innodb\_thread\_concurrency  
线程并发数，建议设置为CPU内核数\*2  
设置方法，在my.cnf文件里：  
innodb\_thread\_concurrency = 8

key\_buffer\_size   
仅作用于 MyISAM存储引擎，用来设置用于缓存 MyISAM存储引擎中索引文件的内存区域大小。如果我们有足够的内存，这个缓存区域最好是能够存放下我们所有的 MyISAM 引擎表的所有索引，以尽可能提高性能。不要设置超过可用内存的30%。即使不用MyISAM表，也要设置该值8-64M，用于临时表。  
设置方法，在my.cnf文件里：  
key\_buffer\_size = 8M

-----------------------------------------------------------------------------------------------------------------------------------------------  
下面是线上mysql（innodb）的my.cnf配置参考：  
[client]  
port = 3306  
socket = /usr/local/mysql/var/mysql.sock

[mysqld]  
port = 3306  
socket = /usr/local/mysql/var/mysql.sock

basedir = /usr/local/mysql/  
datadir = /data/mysql/data  
pid-file = /data/mysql/data/mysql.pid  
user = mysql  
bind-address = 0.0.0.0  
server-id = 1  
sync\_binlog=1  
log\_bin = mysql-bin

skip-name-resolve  
back\_log = 600

max\_connections = 3000  
max\_connect\_errors = 3000  
table\_open\_cache = 512  
max\_allowed\_packet = 16M  
binlog\_cache\_size = 16M  
max\_heap\_table\_size = 16M  
tmp\_table\_size = 256M

read\_buffer\_size = 1024M  
read\_rnd\_buffer\_size = 1024M  
sort\_buffer\_size = 1024M  
join\_buffer\_size = 1024M  
key\_buffer\_size = 8192M

thread\_cache\_size = 8

query\_cache\_size = 512M  
query\_cache\_limit = 1024M

ft\_min\_word\_len = 4  
  
binlog\_format = mixed  
expire\_logs\_days = 30

log\_error = /data/mysql/data/mysql-error.log  
slow\_query\_log = 1  
long\_query\_time = 1  
slow\_query\_log\_file = /data/mysql/data/mysql-slow.log

performance\_schema = 0  
explicit\_defaults\_for\_timestamp

skip-external-locking

default\_storage\_engine = InnoDB  
innodb\_file\_per\_table = 1  
innodb\_open\_files = 500  
innodb\_buffer\_pool\_size = 1024M  
innodb\_write\_io\_threads = 1000  
innodb\_read\_io\_threads = 1000  
innodb\_thread\_concurrency = 8  
innodb\_purge\_threads = 1  
innodb\_flush\_log\_at\_trx\_commit = 2  
innodb\_log\_buffer\_size = 4M  
innodb\_log\_file\_size = 32M  
innodb\_log\_files\_in\_group = 3  
innodb\_max\_dirty\_pages\_pct = 90  
innodb\_lock\_wait\_timeout = 120

bulk\_insert\_buffer\_size = 8M  
myisam\_sort\_buffer\_size = 8M  
myisam\_max\_sort\_file\_size = 10G  
myisam\_repair\_threads = 1

interactive\_timeout = 28800  
wait\_timeout = 28800

[mysqldump]  
quick  
max\_allowed\_packet = 16M

[myisamchk]  
key\_buffer\_size = 8M  
sort\_buffer\_size = 8M  
read\_buffer = 4M  
write\_buffer = 4M

sql\_mode=NO\_ENGINE\_SUBSTITUTION,STRICT\_TRANS\_TABLES   
port = 3306

--------------------------------------------------------------------------------------------------------------------------------------

下面分享一个mysql5.6下my.cnf的优化配置，能使mysql性能大大提升：  
# For advice on how to change settings please see  
# http://dev.mysql.com/doc/refman/5.6/en/server-configuration-defaults.html  
# \*\*\* DO NOT EDIT THIS FILE. It's a template which will be copied to the  
# \*\*\* default location during install, and will be replaced if you  
# \*\*\* upgrade to a newer version of MySQL.  
[mysqld]  
# Remove leading # and set to the amount of RAM for the most important data  
# cache in MySQL. Start at 70% of total RAM for dedicated server, else 10%.  
# innodb\_buffer\_pool\_size = 128M  
# Remove leading # to turn on a very important data integrity option: logging  
# changes to the binary log between backups.  
# log\_bin  
# These are commonly set, remove the # and set as required.  
# basedir = .....  
# datadir = .....  
# port = .....  
# server\_id = .....  
# socket = .....  
  
# Remove leading # to set options mainly useful for reporting servers.  
# The server defaults are faster for transactions and fast SELECTs.  
# Adjust sizes as needed, experiment to find the optimal values.  
##################################################  
#innodb  
user=mysql  
innodb\_buffer\_pool\_size=6G  
innodb\_log\_file\_size=4G  
innodb\_log\_buffer\_size = 8M  
innodb\_flush\_log\_at\_trx\_commit=2  
innodb\_file\_per\_table=1  
innodb\_file\_io\_threads=4  
innodb\_flush\_method=O\_DIRECT  
innodb\_io\_capacity=2000  
innodb\_io\_capacity\_max=6000  
innodb\_lru\_scan\_depth=2000  
innodb\_thread\_concurrency = 0  
innodb\_additional\_mem\_pool\_size=16M  
innodb\_autoinc\_lock\_mode = 2  
##################################################  
# Binary log/replication  
log-bin  
sync\_binlog=1  
sync\_relay\_log=1  
relay-log-info-repository=TABLE  
master-info-repository=TABLE  
expire\_logs\_days=7  
binlog\_format=ROW  
transaction-isolation=READ-COMMITTED  
#################################################  
#cache  
tmp\_table\_size=512M  
character-set-server=utf8  
collation-server=utf8\_general\_ci  
skip-external-locking  
back\_log=1024  
key\_buffer\_size=1024M  
thread\_stack=256k  
read\_buffer\_size=8M  
thread\_cache\_size=64  
query\_cache\_size=128M  
max\_heap\_table\_size=256M  
query\_cache\_type=1  
binlog\_cache\_size = 2M  
table\_open\_cache=128  
thread\_cache=1024  
thread\_concurrency=8  
wait\_timeout=30  
join\_buffer\_size = 1024M  
sort\_buffer\_size = 8M  
read\_rnd\_buffer\_size = 8M  
#################################################  
#connect  
max-connect-errors=100000  
max-connections=1000  
#################################################  
explicit\_defaults\_for\_timestamp=true  
sql\_mode=NO\_ENGINE\_SUBSTITUTION,STRICT\_TRANS\_TABLES  
##################################################

参数解释：

# Binary log/replication(这里主要是复制功能，也就是主从,提前配置好，后面讲主从配置)

#二进制日志  
log-bin  
#为了在最大程序上保证复制的InnoDB事务持久性和一致性  
sync\_binlog=1  
sync\_relay\_log=1  
#启用此两项,可用于实现在崩溃时保证二进制及从服务器安全的功能  
relay-log-info-repository=TABLE  
master-info-repository=TABLE  
#设置清除日志时间  
expire\_logs\_days=7  
#行复制  
binlog\_format=ROW  
#mysql数据库事务隔离级别有四种(READ UNCOMMITTED,READ COMMITTED,REPEATABLE READ,SERIALIZABLE)  
transaction-isolation=READ-COMMITTED  
  
#cache  
#内部内存临时表的最大值  
tmp\_table\_size=512M  
character-set-server=utf8  
collation-server=utf8\_general\_ci  
#即跳过外部锁定  
skip-external-locking  
#MySQL能暂存的连接数量(根据实际设置)  
back\_log=1024  
#指定索引缓冲区的大小,只对MyISAM表起作用,这里写上也没有关系  
key\_buffer\_size=1024M  
#这条指令限定用于每个数据库线程的栈大小  
thread\_stack=256k  
#当一个查询不断地扫描某一个表,MySQL会为它分配一段内存缓冲区  
read\_buffer\_size=8M  
#线程缓存  
thread\_cache\_size=64  
#查询缓存大小  
query\_cache\_size=128M  
#内部内存临时表的最大值,每个线程都要分配  
max\_heap\_table\_size=256M  
#将查询结果放入查询缓存中  
query\_cache\_type=1  
#代表在事务过程中容纳二进制日志SQL语句的缓存大小  
binlog\_cache\_size = 2M  
#同样是缓存表大小  
table\_open\_cache=128  
#缓存线程  
thread\_cache=1024  
#推荐设置为服务器 CPU核数的2倍  
thread\_concurrency=8  
wait\_timeout=30  
#表和表联接的缓冲区的大小  
join\_buffer\_size = 1024M  
#是一个connection级参数,在每个connection第一次需要使用这个buffer的时候,一次性分配设置的内存  
sort\_buffer\_size=8M  
#随机读取数据缓冲区使用内存  
read\_rnd\_buffer\_size = 8M  
  
#connect  
#是一个MySQL中与安全有关的计数器值,它负责阻止过多尝试失败的客户端以防止暴力破解密码  
max-connect-errors=100000  
#连接数  
max-connections=1000  
#开启查询缓存  
explicit\_defaults\_for\_timestamp=true  
#mysql服务器能够工作在不同的模式下,并能针对不同的客户端以不同的方式应用这些模式  
sql\_mode=NO\_ENGINE\_SUBSTITUTION,STRICT\_TRANS\_TABLES  
  
下面列出了对性能优化影响较大的主要变量，主要分为连接请求的变量和缓冲区变量。  
1.连接请求的变量：  
1) max\_connections  
MySQL的最大连接数，增加该值增加mysqld 要求的文件描述符的数量。如果服务器的并发连接请求量比较大，建议调高此值，以增加并行连接数量，当然这建立在机器能支撑的情况下，因为如果连接数越多， 介于MySQL会为每个连接提供连接缓冲区，就会开销越多的内存，所以要适当调整该值，不能盲目提高设值。  
数值过小会经常出现ERROR 1040: Too many connections错误，可以过’conn%’通配符查看当前状态的连接数量，以定夺该值的大小。  
show variables like ‘max\_connections’ 最大连接数  
show status like ‘max\_used\_connections’响应的连接数  
如下：  
mysql> show variables like ‘max\_connections‘;  
+———————–+——-+  
| Variable\_name　| Value |  
+———————–+——-+  
| max\_connections | 256　　|  
+———————–+——-+  
mysql> show status like ‘max%connections‘;  
+———————–+——-+  
| Variable\_name　 | Value |  
+—————————-+——-+  
| max\_used\_connections | 256|  
+—————————-+——-+  
max\_used\_connections / max\_connections \* 100% （理想值≈ 85%）   
如果max\_used\_connections跟max\_connections相同 那么就是max\_connections设置过低或者超过服务器负载上限了，低于10%则设置过大。  
2) back\_log  
MySQL能暂存的连接数量。当主要MySQL线程在一个很短时间内得到非常多的连接请求，这就起作用。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。  
back\_log值指出在MySQL暂时停止回答新请求之前的短时间内有多少个请求可以被存在堆栈中。只有如果期望在一个短时间内有很多连接，你需要增加它，换句话说，这值对到来的TCP/IP连接的侦听队列的大小。  
当观察你主机进程列表（mysql> show full processlist），发现大量264084 | unauthenticated user | xxx.xxx.xxx.xxx | NULL | Connect | NULL | login | NULL 的待连接进程时，就要加大back\_log 的值了。  
默认数值是50，可调优为128，对系统设置范围为小于512的整数。   
3) interactive\_timeout  
一个交互连接在被服务器在关闭前等待行动的秒数。一个交互的客户被定义为对mysql\_real\_connect()使用CLIENT\_INTERACTIVE 选项的客户。   
默认数值是28800，可调优为7200。   
2. 缓冲区变量  
全局缓冲：  
4) key\_buffer\_size  
key\_buffer\_size指定索引缓冲区的大小，它决定索引处理的速度，尤其是索引读的速度。通过检查状态值 Key\_read\_requests和Key\_reads，可以知道key\_buffer\_size设置是否合理。比例key\_reads / key\_read\_requests应该尽可能的低，至少是1:100，1:1000更好（上述状态值可以使用SHOW STATUS LIKE ‘key\_read%’获得）。  
key\_buffer\_size只对MyISAM表起作用。即使你不使用MyISAM表，但是内部的临时磁盘表是MyISAM表，也要使用该值。可以使用检查状态值created\_tmp\_disk\_tables得知详情。  
举例如下：  
mysql> show variables like ‘key\_buffer\_size‘;  
+——————-+————+  
| Variable\_name | Value |  
+———————+————+  
| key\_buffer\_size | 536870912 |  
+———— ———-+————+  
key\_buffer\_size为512MB，我们再看一下key\_buffer\_size的使用情况：  
mysql> show global status like ‘key\_read%‘;  
+————————+————-+  
| Variable\_name　 | Value |  
+————————+————-+  
| Key\_read\_requests| 27813678764 |  
| Key\_reads　　　| 6798830 |  
+————————+————-+  
一共有27813678764个索引读取请求，有6798830个请求在内存中没有找到直接从硬盘读取索引，计算索引未命中缓存的概率：  
key\_cache\_miss\_rate ＝Key\_reads / Key\_read\_requests \* 100%，设置在1/1000左右较好  
默认配置数值是8388600(8M)，主机有4GB内存，可以调优值为268435456(256MB)。  
5) query\_cache\_size  
使用查询缓冲，MySQL将查询结果存放在缓冲区中，今后对于同样的SELECT语句（区分大小写），将直接从缓冲区中读取结果。  
通过检查状态值Qcache\_\*，可以知道query\_cache\_size设置是否合理（上述状态值可以使用SHOW STATUS LIKE ‘Qcache%’获得）。如果Qcache\_lowmem\_prunes的值非常大，则表明经常出现缓冲不够的情况，如果Qcache\_hits的值也 非常大，则表明查询缓冲使用非常频繁，此时需要增加缓冲大小；如果Qcache\_hits的值不大，则表明你的查询重复率很低，这种情况下使用查询缓冲反 而会影响效率，那么可以考虑不用查询缓冲。此外，在SELECT语句中加入SQL\_NO\_CACHE可以明确表示不使用查询缓冲。  
  
与查询缓冲有关的参数还有query\_cache\_type、query\_cache\_limit、query\_cache\_min\_res\_unit。  
query\_cache\_type指定是否使用查询缓冲，可以设置为0、1、2，该变量是SESSION级的变量。  
query\_cache\_limit指定单个查询能够使用的缓冲区大小，缺省为1M。  
query\_cache\_min\_res\_unit是在4.1版本以后引入的，它指定分配缓冲区空间的最小单位，缺省为4K。检查状态值 Qcache\_free\_blocks，如果该值非常大，则表明缓冲区中碎片很多，这就表明查询结果都比较小，此时需要减小 query\_cache\_min\_res\_unit。  
举例如下：  
mysql> show global status like ‘qcache%‘;  
+——————————-+—————–+  
| Variable\_name | Value　 |  
+——————————-+—————–+  
| Qcache\_free\_blocks　 | 22756　 |  
| Qcache\_free\_memory　 | 76764704 |  
| Qcache\_hits　　　　　 | 213028692 |  
| Qcache\_inserts　　　　 | 208894227 |  
| Qcache\_lowmem\_prunes | 4010916 |  
| Qcache\_not\_cached　| 13385031 |  
| Qcache\_queries\_in\_cache | 43560　|  
| Qcache\_total\_blocks | 111212　 |  
+——————————-+—————–+  
mysql> show variables like ‘query\_cache%‘;  
+————————————–+————–+  
| Variable\_name　　　　　 | Value　 |  
+————————————–+———–+  
| query\_cache\_limit　　　　　 | 2097152 |  
| query\_cache\_min\_res\_unit　 | 4096　　 |  
| query\_cache\_size　　　　　 | 203423744 |  
| query\_cache\_type　　　　　 | ON　 |  
| query\_cache\_wlock\_invalidate | OFF　 |  
+————————————–+—————+  
查询缓存碎片率= Qcache\_free\_blocks / Qcache\_total\_blocks \* 100%  
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= (query\_cache\_size – Qcache\_free\_memory) / query\_cache\_size \* 100%  
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。  
查询缓存命中率= (Qcache\_hits – Qcache\_inserts) / Qcache\_hits \* 100%  
示例服务器查询缓存碎片率＝20.46％，查询缓存利用率＝62.26％，查询缓存命中率＝1.94％，命中率很差，可能写操作比较频繁吧，而且可能有些碎片。  
每个连接的缓冲  
6) record\_buffer\_size  
每个进行一个顺序扫描的线程为其扫描的每张表分配这个大小的一个缓冲区。如果你做很多顺序扫描，你可能想要增加该值。  
默认数值是131072(128K)，可改为16773120 (16M)  
7) read\_rnd\_buffer\_size  
随机读缓冲区大小。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。  
一般可设置为16M   
8) sort\_buffer\_size  
每个需要进行排序的线程分配该大小的一个缓冲区。增加这值加速ORDER BY或GROUP BY操作。  
默认数值是2097144(2M)，可改为16777208 (16M)。  
9) join\_buffer\_size  
联合查询操作所能使用的缓冲区大小  
record\_buffer\_size，read\_rnd\_buffer\_size，sort\_buffer\_size，join\_buffer\_size为每个线程独占，也就是说，如果有100个线程连接，则占用为16M\*100  
10) table\_cache  
表高速缓存的大小。每当MySQL访问一个表时，如果在表缓冲区中还有空间，该表就被打开并放入其中，这样可以更快地访问表内容。通过检查峰值时间的状态值Open\_tables和Opened\_tables，可以决定是否需要增加table\_cache的值。如 果你发现open\_tables等于table\_cache，并且opened\_tables在不断增长，那么你就需要增加table\_cache的值了 （上述状态值可以使用SHOW STATUS LIKE ‘Open%tables’获得）。注意，不能盲目地把table\_cache设置成很大的值。如果设置得太高，可能会造成文件描述符不足，从而造成性能 不稳定或者连接失败。  
1G内存机器，推荐值是128－256。内存在4GB左右的服务器该参数可设置为256M或384M。  
11) max\_heap\_table\_size  
用户可以创建的内存表(memory table)的大小。这个值用来计算内存表的最大行数值。这个变量支持动态改变，即set @max\_heap\_table\_size=#  
这个变量和tmp\_table\_size一起限制了内部内存表的大小。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。  
12) tmp\_table\_size  
通过设置tmp\_table\_size选项来增加一张临时表的大小，例如做高级GROUP BY操作生成的临时表。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，建议尽量优化查询，要确保查询过程中生成的临时表在内存中，避免临时表过大导致生成基于硬盘的MyISAM表。  
mysql> show global status like ‘created\_tmp%‘;  
+——————————–+———+  
| Variable\_name　　 | Value　|  
+———————————-+———+  
| Created\_tmp\_disk\_tables | 21197 |  
| Created\_tmp\_files　　　| 58　　|  
| Created\_tmp\_tables　　| 1771587 |  
+——————————–+———–+  
每次创建临时表，Created\_tmp\_tables增加，如果临时表大小超过tmp\_table\_size，则是在磁盘上创建临时 表，Created\_tmp\_disk\_tables也增加,Created\_tmp\_files表示MySQL服务创建的临时文件文件数，比较理想的配 置是：  
Created\_tmp\_disk\_tables / Created\_tmp\_tables \* 100% <= 25%比如上面的服务器Created\_tmp\_disk\_tables / Created\_tmp\_tables \* 100% ＝1.20%，应该相当好了  
默认为16M，可调到64-256最佳，线程独占，太大可能内存不够I/O堵塞  
13) thread\_cache\_size  
可以复用的保存在中的线程的数量。如果有，新的线程从缓存中取得，当断开连接的时候如果有空间，客户的线置在缓存中。如果有很多新的线程，为了提高性能可以这个变量值。  
通过比较 Connections和Threads\_created状态的变量，可以看到这个变量的作用。  
默认值为110，可调优为80。   
14) thread\_concurrency  
推荐设置为服务器 CPU核数的2倍，例如双核的CPU, 那么thread\_concurrency的应该为4；2个双核的cpu, thread\_concurrency的值应为8。默认为8  
15) wait\_timeout  
指定一个请求的最大连接时间，对于4GB左右内存的服务器可以设置为5-10。  
3. 配置InnoDB的几个变量  
innodb\_buffer\_pool\_size  
对于InnoDB表来说，innodb\_buffer\_pool\_size的作用就相当于key\_buffer\_size对于MyISAM表的作用一样。InnoDB使用该参数指定大小的内存来缓冲数据和索引。对于单独的MySQL数据库服务器，最大可以把该值设置成物理内存的80%。  
根据MySQL手册，对于2G内存的机器，推荐值是1G（50%）。

innodb\_flush\_log\_at\_trx\_commit  
主要控制了innodb将log buffer中的数据写入日志文件并flush磁盘的时间点，取值分别为0、1、2三个。0，表示当事务提交时，不做日志写入操作，而是每秒钟将log buffer中的数据写入日志文件并flush磁盘一次；1，则在每秒钟或是每次事物的提交都会引起日志文件写入、flush磁盘的操作，确保了事务的 ACID；设置为2，每次事务提交引起写入日志文件的动作，但每秒钟完成一次flush磁盘操作。  
实际测试发现，该值对插入数据的速度影响非常大，设置为2时插入10000条记录只需要2秒，设置为0时只需要1秒，而设置为1时则需要229秒。因此，MySQL手册也建议尽量将插入操作合并成一个事务，这样可以大幅提高速度。  
根据MySQL手册，在允许丢失最近部分事务的危险的前提下，可以把该值设为0或2。

innodb\_log\_buffer\_size  
log缓存大小，一般为1-8M，默认为1M，对于较大的事务，可以增大缓存大小。  
可设置为4M或8M。

innodb\_additional\_mem\_pool\_size  
该参数指定InnoDB用来存储数据字典和其他内部数据结构的内存池大小。缺省值是1M。通常不用太大，只要够用就行，应该与表结构的复杂度有关系。如果不够用，MySQL会在错误日志中写入一条警告信息。  
根据MySQL手册，对于2G内存的机器，推荐值是20M，可适当增加。

innodb\_thread\_concurrency=8  
推荐设置为 2\*(NumCPUs+NumDisks)，默认一般为8  
  
MySQL 5.6相比于前代GA版本性能提升显著，但默认缓存设置对于小型站点并不合理。通过修改my.ini文件中的performance\_schema\_max\_table\_instances参数，能够有效降低内存占用。  
以下是5.6默认的设置  
performance\_schema\_max\_table\_instances 12500  
table\_definition\_cache 1400  
table\_open\_cache 2000  
可以调成，或者在小点都可以。  
  
performance\_schema\_max\_table\_instances=600  
table\_definition\_cache=400  
table\_open\_cache=256  
  
performance\_schema\_max\_table\_instances  
The maximum number of instrumented table objects 检测的表对象的最大数目。  
table\_definition\_cache  
The number of table definitions (from .frm files) that can be stored in the definition cache. If you use a large number of tables, you can create a large table definition cache to speed up opening of tables. The table definition cache takes less space and does not use file descriptors, unlike the normal table cache. The minimum and default values are both 400.   
缓存frm文件

table\_open\_cache  
The number of open tables for all threads. Increasing this value increases the number of file descriptors that mysqld requires.   
table\_open\_cache 指的是缓存数据文件的描述符(Linux/Unix)相关信息  
这个很重要啊，之前mount个单独的文件，数据库一直不成功，原来是这个在作怪啊。  
chcon -R -t mysqld\_db\_t /home/myusqldata  
  
  
mysql> show variables;  
一、慢查询  
mysql> show variables like '%slow%';  
+------------------+-------+  
| Variable\_name | Value |  
+------------------+-------+  
| log\_slow\_queries | ON |  
| slow\_launch\_time | 2 |  
+------------------+-------+

mysql> show global status like '%slow%';  
+---------------------+-------+  
| Variable\_name | Value |  
+---------------------+-------+  
| Slow\_launch\_threads | 0 |  
| Slow\_queries | 4148 |  
+---------------------+-------+  
配置中打开了记录慢查询，执行时间超过2秒的即为慢查询，系统显示有4148个慢查询，你可以分析慢查询日志，找出有问题的SQL语句，慢查询时间不宜设置过长，否则意义不大，最好在5秒以内，如果你需要微秒级别的慢查询，可以考虑给MySQL打补丁：[http://www.percona.com/docs/wiki/release:start](http://www.percona.com/docs/wiki/release:start" \t "http://www.cnblogs.com/kevingrace/p/_blank)，记得找对应的版本。  
打开慢查询日志可能会对系统性能有一点点影响，如果你的MySQL是主－从结构，可以考虑打开其中一台从服务器的慢查询日志，这样既可以监控慢查询，对系统性能影响又小。  
二、连接数  
经常会遇见”MySQL: ERROR 1040: Too manyconnections”的情况，一种是访问量确实很高，MySQL服务器抗不住，这个时候就要考虑增加从服务器分散读压力，另外一种情况是MySQL配置文件中max\_connections值过小：  
mysql> show variables like 'max\_connections';  
+-----------------+-------+  
| Variable\_name | Value |  
+-----------------+-------+  
| max\_connections | 256 |  
+-----------------+-------+  
这台MySQL服务器最大连接数是256，然后查询一下服务器响应的最大连接数：  
mysql> show global status like 'Max\_used\_connections';  
+----------------------+-------+  
| Variable\_name | Value |  
+----------------------+-------+  
| Max\_used\_connections | 245 |  
+----------------------+-------+  
MySQL服务器过去的最大连接数是245，没有达到服务器连接数上限256，应该没有出现1040错误，比较理想的设置是：  
Max\_used\_connections / max\_connections \* 100% ≈ 85%  
最大连接数占上限连接数的85％左右，如果发现比例在10%以下，MySQL服务器连接数上限设置的过高了。  
三、Key\_buffer\_size  
key\_buffer\_size是对MyISAM表性能影响最大的一个参数，下面一台以MyISAM为主要存储引擎服务器的配置：  
mysql> show variables like 'key\_buffer\_size';  
+-----------------+------------+  
| Variable\_name | Value |  
+-----------------+------------+  
| key\_buffer\_size | 536870912 |  
+-----------------+------------+  
分配了512MB内存给key\_buffer\_size，我们再看一下key\_buffer\_size的使用情况：  
mysql> show global status like 'key\_read%';  
+------------------------+-------------+  
| Variable\_name | Value |  
+------------------------+-------------+  
| Key\_read\_requests | 27813678764 |  
| Key\_reads | 6798830 |  
+------------------------+-------------+  
一共有27813678764个索引读取请求，有6798830个请求在内存中没有找到直接从硬盘读取索引，计算索引未命中缓存的概率：  
key\_cache\_miss\_rate ＝ Key\_reads / Key\_read\_requests \* 100%  
比 如上面的数据，key\_cache\_miss\_rate为0.0244%，4000个索引读取请求才有一个直接读硬盘，已经很BT 了，key\_cache\_miss\_rate在0.1%以下都很好（每1000个请求有一个直接读硬盘），如果key\_cache\_miss\_rate在 0.01%以下的话，key\_buffer\_size分配的过多，可以适当减少。  
MySQL服务器还提供了key\_blocks\_\*参数：  
mysql> show global status like 'key\_blocks\_u%';  
+------------------------+-------------+  
| Variable\_name | Value |  
+------------------------+-------------+  
| Key\_blocks\_unused | 0 |  
| Key\_blocks\_used | 413543 |  
+------------------------+-------------+  
Key\_blocks\_unused 表示未使用的缓存簇(blocks)数，Key\_blocks\_used表示曾经用到的最大的blocks数，比如这台服务器，所有的缓存都用到了，要么 增加key\_buffer\_size，要么就是过渡索引了，把缓存占满了。比较理想的设置：  
Key\_blocks\_used / (Key\_blocks\_unused + Key\_blocks\_used) \* 100% ≈ 80%  
四、临时表  
mysql> show global status like 'created\_tmp%';  
+-------------------------+---------+  
| Variable\_name | Value |  
+-------------------------+---------+  
| Created\_tmp\_disk\_tables | 21197 |  
| Created\_tmp\_files | 58 |  
| Created\_tmp\_tables | 1771587 |  
+-------------------------+---------+  
每次创建临时表，Created\_tmp\_tables增加，如果是在磁盘上创建临时表，Created\_tmp\_disk\_tables也增加,Created\_tmp\_files表示MySQL服务创建的临时文件文件数，比较理想的配置是：  
Created\_tmp\_disk\_tables / Created\_tmp\_tables \* 100% <= 25%  
比如上面的服务器Created\_tmp\_disk\_tables / Created\_tmp\_tables \* 100% ＝ 1.20%，应该相当好了。我们再看一下MySQL服务器对临时表的配置：  
mysql> show variables where Variable\_name in ('tmp\_table\_size', 'max\_heap\_table\_size');  
+---------------------+-----------+  
| Variable\_name | Value |  
+---------------------+-----------+  
| max\_heap\_table\_size | 268435456 |  
| tmp\_table\_size | 536870912 |  
+---------------------+-----------+  
只有256MB以下的临时表才能全部放内存，超过的就会用到硬盘临时表。  
五、Open Table情况  
mysql> show global status like 'open%tables%';  
+---------------+-------+  
| Variable\_name | Value |  
+---------------+-------+  
| Open\_tables | 919 |  
| Opened\_tables | 1951 |  
+---------------+-------+  
Open\_tables 表示打开表的数量，Opened\_tables表示打开过的表数量，如果Opened\_tables数量过大，说明配置中 table\_cache(5.1.3之后这个值叫做table\_open\_cache)值可能太小，我们查询一下服务器table\_cache值：  
mysql> show variables like 'table\_cache';  
+---------------+-------+  
| Variable\_name | Value |  
+---------------+-------+  
| table\_cache | 2048 |

+---------------+-------+  
比较合适的值为：  
Open\_tables / Opened\_tables \* 100% >= 85%  
Open\_tables / table\_cache \* 100% <= 95%

六、进程使用情况  
mysql> show global status like 'Thread%';  
+-------------------+-------+  
| Variable\_name | Value |  
+-------------------+-------+  
| Threads\_cached | 46 |  
| Threads\_connected | 2 |  
| Threads\_created | 570 |  
| Threads\_running | 1 |  
+-------------------+-------+  
如 果我们在MySQL服务器配置文件中设置了thread\_cache\_size，当客户端断开之后，服务器处理此客户的线程将会缓存起来以响应下一个客户 而不是销毁（前提是缓存数未达上限）。Threads\_created表示创建过的线程数，如果发现Threads\_created值过大的话，表明 MySQL服务器一直在创建线程，这也是比较耗资源，可以适当增加配置文件中thread\_cache\_size值，查询服务器 thread\_cache\_size配置：  
mysql> show variables like 'thread\_cache\_size';  
+-------------------+-------+  
| Variable\_name | Value |  
+-------------------+-------+  
| thread\_cache\_size | 64 |  
+-------------------+-------+  
示例中的服务器还是挺健康的。  
七、查询缓存(query cache)  
mysql> show global status like 'qcache%';  
+-------------------------+-----------+  
| Variable\_name | Value |  
+-------------------------+-----------+  
| Qcache\_free\_blocks | 22756 |  
| Qcache\_free\_memory | 76764704 |  
| Qcache\_hits | 213028692 |  
| Qcache\_inserts | 208894227 |  
| Qcache\_lowmem\_prunes | 4010916 |  
| Qcache\_not\_cached | 13385031 |  
| Qcache\_queries\_in\_cache | 43560 |  
| Qcache\_total\_blocks | 111212 |  
+-------------------------+-----------+  
MySQL查询缓存变量解释：  
Qcache\_free\_blocks：缓存中相邻内存块的个数。数目大说明可能有碎片。FLUSH QUERY CACHE会对缓存中的碎片进行整理，从而得到一个空闲块。  
Qcache\_free\_memory：缓存中的空闲内存。  
Qcache\_hits：每次查询在缓存中命中时就增大  
Qcache\_inserts：每次插入一个查询时就增大。命中次数除以插入次数就是不中比率。  
Qcache\_lowmem\_prunes： 缓存出现内存不足并且必须要进行清理以便为更多查询提供空间的次数。这个数字最好长时间来看；如果这个数字在不断增长，就表示可能碎片非常严重，或者内存 很少。（上面的 free\_blocks和free\_memory可以告诉您属于哪种情况）  
Qcache\_not\_cached：不适合进行缓存的查询的数量，通常是由于这些查询不是 SELECT 语句或者用了now()之类的函数。  
Qcache\_queries\_in\_cache：当前缓存的查询（和响应）的数量。  
Qcache\_total\_blocks：缓存中块的数量。  
我们再查询一下服务器关于query\_cache的配置：  
mysql> show variables like 'query\_cache%';  
+------------------------------+-----------+  
| Variable\_name | Value |  
+------------------------------+-----------+  
| query\_cache\_limit | 2097152 |  
| query\_cache\_min\_res\_unit | 4096 |  
| query\_cache\_size | 203423744 |  
| query\_cache\_type | ON |  
| query\_cache\_wlock\_invalidate | OFF |  
+------------------------------+-----------+  
各字段的解释：  
query\_cache\_limit：超过此大小的查询将不缓存  
query\_cache\_min\_res\_unit：缓存块的最小大小  
query\_cache\_size：查询缓存大小  
query\_cache\_type：缓存类型，决定缓存什么样的查询，示例中表示不缓存 select sql\_no\_cache 查询  
query\_cache\_wlock\_invalidate：当有其他客户端正在对MyISAM表进行写操作时，如果查询在query cache中，是否返回cache结果还是等写操作完成再读表获取结果。  
query\_cache\_min\_res\_unit的配置是一柄”双刃剑”，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。  
查询缓存碎片率 = Qcache\_free\_blocks / Qcache\_total\_blocks \* 100%  
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 = (query\_cache\_size - Qcache\_free\_memory) / query\_cache\_size \* 100%  
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。  
查询缓存命中率 = (Qcache\_hits - Qcache\_inserts) / Qcache\_hits \* 100%  
示例服务器 查询缓存碎片率 ＝ 20.46％，查询缓存利用率 ＝ 62.26％，查询缓存命中率 ＝ 1.94％，命中率很差，可能写操作比较频繁吧，而且可能有些碎片。  
八、排序使用情况  
mysql> show global status like 'sort%';  
+-------------------+------------+  
| Variable\_name | Value |  
+-------------------+------------+  
| Sort\_merge\_passes | 29 |  
| Sort\_range | 37432840 |  
| Sort\_rows | 9178691532 |  
| Sort\_scan | 1860569 |  
+-------------------+------------+  
Sort\_merge\_passes 包括两步。MySQL 首先会尝试在内存中做排序，使用的内存大小由系统变量Sort\_buffer\_size 决定，如果它的大小不够把所有的记录都读到内存中，MySQL 就会把每次在内存中排序的结果存到临时文件中，等MySQL 找到所有记录之后，再把临时文件中的记录做一次排序。这再次排序就会增加 Sort\_merge\_passes。实际上，MySQL会用另一个临时文件来存再次排序的结果，所以通常会看到 Sort\_merge\_passes增加的数值是建临时文件数的两倍。因为用到了临时文件，所以速度可能会比较慢，增加 Sort\_buffer\_size 会减少Sort\_merge\_passes 和 创建临时文件的次数。但盲目的增加 Sort\_buffer\_size 并不一定能提高速度，  
另外，增加read\_rnd\_buffer\_size(3.2.3是record\_rnd\_buffer\_size)的值对排序的操作也有一点的好处，  
九、文件打开数(open\_files)  
mysql> show global status like 'open\_files';  
+---------------+-------+  
| Variable\_name | Value |  
+---------------+-------+  
| Open\_files | 1410 |  
+---------------+-------+

mysql> show variables like 'open\_files\_limit';  
+------------------+-------+  
| Variable\_name | Value |  
+------------------+-------+  
| open\_files\_limit | 4590 |  
+------------------+-------+  
比较合适的设置：Open\_files / open\_files\_limit \* 100% <= 75％  
十、表锁情况  
mysql> show global status like 'table\_locks%';  
+-----------------------+-----------+  
| Variable\_name | Value |  
+-----------------------+-----------+  
| Table\_locks\_immediate | 490206328 |  
| Table\_locks\_waited | 2084912 |  
+-----------------------+-----------+  
Table\_locks\_immediate 表示立即释放表锁数，Table\_locks\_waited表示需要等待的表锁数，如果Table\_locks\_immediate / Table\_locks\_waited >5000，最好采用InnoDB引擎，因为InnoDB是行锁而MyISAM是表锁，对于高并发写入的应用InnoDB效果会好些。示例中的服务 器Table\_locks\_immediate / Table\_locks\_waited ＝ 235，MyISAM就足够了。  
十一、表扫描情况  
mysql> show global status like 'handler\_read%';  
+-----------------------+-------------+  
| Variable\_name | Value |  
+-----------------------+-------------+  
| Handler\_read\_first | 5803750 |  
| Handler\_read\_key | 6049319850 |  
| Handler\_read\_next | 94440908210 |  
| Handler\_read\_prev | 34822001724 |  
| Handler\_read\_rnd | 405482605 |  
| Handler\_read\_rnd\_next | 18912877839 |  
+-----------------------+-------------+  
各字段解释参见,调出服务器完成的查询请求次数：  
mysql> show global status like 'com\_select';  
+---------------+-----------+  
| Variable\_name | Value |  
+---------------+-----------+  
| Com\_select | 222693559 |  
+---------------+-----------+  
计算表扫描率：  
表扫描率 ＝ Handler\_read\_rnd\_next / Com\_select  
如果表扫描率超过4000，说明进行了太多表扫描，很有可能索引没有建好，增加read\_buffer\_size值会有一些好处，但最好不要超过8MB。  
  
要查看死锁，你要show engine innodb status\G;  
在MySQL5.6版本，在my.cnf配置文件里，加入  
innodb\_print\_all\_deadlocks = 1  
就可以把死锁信息打印到错误日志里

innodb\_flush\_log\_at\_trx\_commit和sync\_binlog是MySQL innodb引擎的两个重要的参数，其中innodb\_flush\_log\_at\_trx\_commit是将事务日志从innodb log buffer写入到redo log中，sync\_binlog是将二进制日志文件刷新到磁盘上。  
  
innodb事务日志redo,binlog逻辑过程如下:  
1.事务写入redo log buffer中；  
2.将log buffer刷新到redo log中，不过会先写一个TX PREPARE标记；  
3.写binlog  
4.在redo log中写入TX COMMIT标记；  
5.将写binlog成功的标记写入redo log。  
  
参数解析如下：   
innodb\_flush\_log\_at\_trx\_commit = N：   
  
N=0    每隔一秒，把事务日志缓存区的数据写到日志文件中，以及把日志文件的数据刷新到磁盘上；  
   log buffer 会 每秒写入到日志文件并刷写（flush）到磁盘。但每次事务提交不会有任何影响，也就是 log buffer 的刷写操作和事务提交操作没有关系。在这种情况下，MySQL性能最好，但如果 mysqld 进程崩溃，通常会导致最后 1s 的日志丢失。  
     
N=1    每个事务提交时候，把事务日志从缓存区写到日志文件中，并且刷新日志文件的数据到磁盘上；   
    当取值为 1 时，每次事务提交时，log buffer 会被写入到日志文件并刷写到磁盘。这也是默认值。这是最安全的配置，但由于每次事务都需要进行磁盘I/O，所以也最慢。  
  
N=2    每事务提交的时候，把事务日志数据从缓存区写到日志文件中；每隔一秒，刷新一次日志文件，但不一定刷新到磁盘上，而是取决于操作系统的调度；   
    当取值为 2 时，每次事务提交会写入日志文件，但并不会立即刷写到磁盘，日志文件会每秒刷写一次到磁盘。这时如果 mysqld 进程崩溃，由于日志已经写入到系统缓存，所以并不会丢失数据；在操作系统崩溃的情况下，通常会导致最后 1s 的日志丢失。   
上面说到的「最后 1s」并不是绝对的，有的时候会丢失 更多数据。有时候由于调度的问题，每秒刷写（once-per-second flushing）并不能保证 100% 执行。对于一些数据一致性和完整性要求不高的应用，配置为 2 就足够了；如果为了最高性能，可以设置为 0。有些应用，如支付服务，对一致性和完整性要求很高，所以即使最慢，也最好设置为 1.   
      当我们设置为2 的时候，Log Thread 会在我们每次事务结束的时候将数据写入事务日志，但是这里的写入仅仅是调用了文件系统的文件写入操作。而我们的文件系统都是有缓存机制的，所以Log Thread 的这个写入并不能保证内容真的已经写入到物理磁盘上面完成持久化的动作。文件系统什么时候会将缓存中的这个数据同步到物理磁盘文件Log Thread 就完全不知道了。所以，当设置为2 的时候，MySQL Crash 并不会造成数据的丢失，但是OS Crash 或者是主机断电后可能丢失的数据量就完全控制在文件系统上了。各种文件系统对于自己缓存的刷新机制各不一样，大家可以自行参阅相关的手册。   
       
sync\_binlog =  N：   
  
N>0    每向二进制日志文件写入N条SQL或N个事务后，则把二进制日志文件的数据刷新到磁盘上；   
N=0    不主动刷新二进制日志文件的数据到磁盘上，而是由操作系统决定；   
  
推荐配置组合：   
  
N=1,1  — 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，比如充值消费系统；   
N=1,0  — 适合数据安全性要求高，磁盘IO写能力支持业务不富余，允许备库落后或无复制；   
N=2,0或2,m(0<m<100)  — 适合数据安全性有要求，允许丢失一点事务日志，复制架构的延迟也能接受；   
N=0,0  — 磁盘IO写能力有限，无复制或允许复制延迟稍微长点能接受，例如：日志性登记业务；   
  当两个参数设置为双1的时候，写入性能最差，sync\_binlog=N (N>1 ) innodb\_flush\_log\_at\_trx\_commit=2 时，(在当前模式下)MySQL的写操作才能达到最高性能。   
  
数据安全性  
   
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，都为1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，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。

MySQL[数据库](http://www.2cto.com/database/" \t "http://www.2cto.com/database/201409/_blank)提供了十三种不同的存储引擎来应对不同的数据处理。大部分使用过MySQL的人应该都知道MySQL所常用的两个存储引擎：MyISAM 和 InnoDB。大多数情况下，MyISAM 是作为MySQL默认的存储引擎而存在的。事实上我们大多数人在使用MySQL的时候都没有去配置存储引擎而直接采用MySQL默认的MyISAM 。如果你需要在特定的表上面使用特定的存储引擎，你可以使用下面的命令行：

[?](http://www.2cto.com/database/201409/333956.html)

|  |  |
| --- | --- |
| 1 | **CREATE** **TABLE** tablename (column1, column2, [etc...]) ENGINE = [storage\_engine]. |

那么什么是数据库引擎呢？  
简单说来，一个数据库的存储引擎就是用来对数据表进行信息存储、处理、查询等操作。这个世界上不存在十全十美的存储引擎，但是对于大部分的应用来说，默认的MyISAM 就够用了。在MySQL内置的10种不同的存储引擎中，并不是所有的引擎都是可用的。你可以使用下面的命令来查询你的MySQL服务器提供的存储引擎：  
代码:

[?](http://www.2cto.com/database/201409/333956.html)

|  |  |
| --- | --- |
| 1  2  3 | mysql -uroot -p  **Password**:  mysql> show engines; |

这个命令会列出你的服务器所支持的存储引擎列表：  
MyISAM  
InnoDB  
MERGE  
MEMORY (HEAP)  
BDB (BerkeleyDB)  
EXAMPLE  
ARCHIVE  
CSV  
BLACKHOLE  
ISAM 你可以根据你自己的应用需求来选择对于的存储引擎。例如，如果你需要存储大量的日志数据，那么你可以使用只支持INSERT和SELECT操作的ARCHIVE 存储引擎。你可以对你的数据库服务器、数据库或者表格设置特定的存储引擎，这种灵活的可配置性也是MySQL比起那些只支持单一存储引擎的数据库更加受欢迎的原因。  
  
**MyISAM**

MyISAM 事实上是 ISAM 存储引擎的一个分支。如果你的应用不需要TRANSACTION，也不需要行级别的锁（它只提供表级别的锁），那么这是选择MyISAM是最优方案了。MyISAM 特别适合于INSERT/UPDATE操作少而SELECT 操作多的应用场景。当你的应用需要进行大量的INSERT/UPDATE操作时，你需要考虑是否应该换一个存储引擎，因为表级别的锁会让这个操作造成性能问题。

MyISAM 支持的最大行数为 ~4.29E+09，每个表的索引数目限制为64。MyISAM 同时也为 TEXT/BLOB类型的列提供了全索引支持，这样很方便进行搜索等操作。  
  
**InnoDB**

相对于MyISAM, InnoDB提供了更多特性来提升[系统](http://www.2cto.com/os/" \t "http://www.2cto.com/database/201409/_blank)性能。这也导致了InnoDB在初始化的时候会比MyISAM花更多的时间，但是这样带来的好处远比初始化额外花的时间要多。其中一个主要的不同就是InnoDB提供了行级别的锁。这就为并发的 INSERT/UPDATE/DELETE 操作提供了可能，不像MyISAM那样要等一个操作完成才能进行下一个操作。

同时, InnoDB 还提供了外键功能。这可以保证你在往表二插入数据之前表一的相关数据已经存在。而且这还会阻止你在表一中删除表二所依赖的数据。

InnoDB还提供了数据的缓存已经在内存和磁盘上建立索引，这样可以大大的提升系统性能。也许对于那些低内存的系统来说，这不是一个理想的解决方案，但是对于内存足够的系统来说就不是问题了。

**MERGE**  
MERGE ，从MySQL 3.23.25 开始添加的存储引擎。它让用户可以对一个MyISAM 集合进行统一操作，就像是操作一个表一样。但是使用这个引擎有一些约束，比如所有的表格都必须是统一的定义。  
  
**MEMORY (HEAP)**  
HEAP 存储引擎，也叫做 MEMORY，允许在内存里面建表。MySQL 服务器会保留表的格式，这样可以快速的创建一个“trash”表，然后快速的读取数据以便更好的处理。不过这也使得这个存储引擎不适合长期的数据处理。  
  
**BDB (BerkeleyDB)**

BerkeleyDB存储引擎处理事务安全的表，并以哈希为基础的存储系统。这个存储引擎适合快速的读写某些数据，特别是不同key的数据。但是这个存储引擎有不少缺点，比如在没有索引的列上速度很慢。也因为这个原因人们都忽视了它。我还是相信它有它的用武之地。

**EXAMPLE**

EXAMPLE，从MySQL 4.1.3 开始添加的存储引擎。这个存储引擎主要是为程序员服务的。EXAMPLE 提供了创建表的能力，但是无法插入或者查询信息。

**ARCHIVE**  
ARCHIVE，从MySQL 4.1.3 开始添加的存储引擎，它经常被用来存储大规模数据（无需索引）。这个存储引擎只支持 INSERT 和 SELECT操作，而且所有的信息都是被压缩过的。这些特性让ARCHIVE很适合用来存储日志、交易记录、账户等。不过需要注意的是，在读取数据的时候，整个表都需要在数据返回之前解压并读取。因此这个存储引擎最适合使用频率较低的场景。  
  
**CSV**  
CSV, 从MySQL 4.1.4开始添加的存储引擎，以逗号分隔的字符串存储数据。因此这个存储引擎不适合大规模的数据存储或者需要索引的数据表等场景。这个存储引擎适合将数据转成spreadsheet 文件。  
  
**BLACKHOLE**  
这个存储引擎似乎看起来没什么用处，因为它不允许任何数据的存储或者查询。因此BLACKHOLE 通常被用来测试数据库的结构、索引和擦查询。你仍然可以使用 INSERTS 命令来插入，只不过所有的数据都是void  
  
**ISAM**  
最原始的存储引擎就是ISAM, 它管理着非事务性表。后来它就被MyISAM 代替了，而且MyISAM 是向后兼容的，因此你可以忘记这个ISAM存储引擎。  
  
**总结**  
总的说来，没有十全十美的数据库存储引擎。对于大多数的应用和DBA来说，选用InnoDB 和 MyISAM 基本就够用了。但是也要记住，InnoDB 和 MyISAM 虽然比较通用，但是它们并不是对所有的场景都是完美支持的，也许其他的存储引擎可以很好地支持你的应用。

MyISAM与InnoDB的区别  
　　InnoDB和MyISAM是许多人在使用MySQL时最常用的两个表类型，这两个表类型各有优劣，视具体应用而定。基本的差别为：MyISAM类型不支持事务处理等高级处理，而InnoDB类型支持。MyISAM类型的表强调的是性能，其执行数度比InnoDB类型更快，但是不提供事务支持，而InnoDB提供事务支持已经外部键等高级数据库功能。

**以下是一些细节和具体实现的差别：**1.InnoDB不支持FULLTEXT类型的索引。  
2.InnoDB 中不保存表的具体行数，也就是说，执行select count(\*) fromtable时，InnoDB要扫描一遍整个表来计算有多少行，但是MyISAM只要简单的读出保存好的行数即可。注意的是，当count(\*)语句包含where条件时，两种表的操作是一样的。  
3.对于AUTO\_INCREMENT类型的字段，InnoDB中必须包含只有该字段的索引，但是在MyISAM表中，可以和其他字段一起建立联合索引。  
4.DELETE FROM table时，InnoDB不会重新建立表，而是一行一行的删除。  
5.LOAD TABLE FROMMASTER操作对InnoDB是不起作用的，解决方法是首先把InnoDB表改成MyISAM表，导入数据后再改成InnoDB表，但是对于使用的额外的InnoDB特性(例如外键)的表不适用。

另外，InnoDB表的行锁也不是绝对的，假如在执行一个SQL语句时MySQL不能确定要扫描的范围，InnoDB表同样会锁全表，例如updatetable set num=1 where name like “a%”  
两种类型最主要的差别就是Innodb支持事务处理与外键和行级锁.而MyISAM不支持.所以MyISAM往往就容易被人认为只适合在小项目中使用。  
我作为使用MySQL的用户角度出发，Innodb和MyISAM都是比较喜欢的，但是从我目前运维的数据库平台要达到需求：99.9%的稳定性，方便的扩展性和高可用性来说的话，MyISAM绝对是我的首选。

**原因如下：**1、首先我目前平台上承载的大部分项目是读多写少的项目，而MyISAM的读性能是比Innodb强不少的。  
2、MyISAM的索引和数据是分开的，并且索引是有压缩的，内存使用率就对应提高了不少。能加载更多索引，而Innodb是索引和数据是紧密捆绑的，没有使用压缩从而会造成Innodb比MyISAM体积庞大不小。  
3、从平台角度来说，经常隔1，2个月就会发生应用开发人员不小心update一个表where写的范围不对，导致这个表没法正常用了，这个时候MyISAM的优越性就体现出来了，随便从当天拷贝的压缩包取出对应表的文件，随便放到一个数据库目录下，然后dump成sql再导回到主库，并把对应的binlog补上。如果是Innodb，恐怕不可能有这么快速度，别和我说让Innodb定期用导出xxx.sql机制备份，因为我平台上最小的一个数据库实例的数据量基本都是几十G大小。  
4、从我接触的应用逻辑来说，select count(\*) 和order by是最频繁的，大概能占了整个sql总语句的60%以上的操作，而这种操作Innodb其实也是会锁表的，很多人以为Innodb是行级锁，那个只是where对它主键是有效，非主键的都会锁全表的。  
5、还有就是经常有很多应用部门需要我给他们定期某些表的数据，MyISAM的话很方便，只要发给他们对应那表的frm.MYD,MYI的文件，让他们自己在对应版本的数据库启动就行，而Innodb就需要导出xxx.sql了，因为光给别人文件，受字典数据文件的影响，对方是无法使用的。  
6、如果和MyISAM比insert写操作的话，Innodb还达不到MyISAM的写性能，如果是针对基于索引的update操作，虽然MyISAM可能会逊色Innodb,但是那么高并发的写，从库能否追的上也是一个问题，还不如通过多实例分库分表**[架构](http://lib.csdn.net/base/architecture" \o "大型网站架构知识库" \t "http://blog.csdn.net/a764340703/article/details/_blank)**来解决。  
7、如果是用MyISAM的话，merge引擎可以大大加快应用部门的开发速度，他们只要对这个merge表做一些selectcount(\*)操作，非常适合大项目总量约几亿的rows某一类型(如日志，调查统计)的业务表。  
当然Innodb也不是绝对不用，用事务的项目如模拟炒股项目，我就是用Innodb的，活跃用户20多万时候，也是很轻松应付了，因此我个人也是很喜欢Innodb的，只是如果从数据库平台应用出发，我还是会首MyISAM。  
另外，可能有人会说你MyISAM无法抗太多写操作，但是我可以通过架构来弥补，说个我现有用的数据库平台**容量：**主从数据总量在几百T以上，每天十多亿pv的动态页面，还有几个大项目是通过数据接口方式调用未算进pv总数，(其中包括一个大项目因为初期memcached没部署,导致单台数据库每天处理9千万的查询)。而我的整体数据库服务器平均负载都在0.5-1左右。

**一般来说，MyISAM适合：**(1)做很多count 的计算；  
(2)插入不频繁，查询非常频繁；  
(3)没有事务。

**InnoDB适合：**(1)可靠性要求比较高，或者要求事务；  
(2)表更新和查询都相当的频繁，并且表锁定的机会比较大的情况指定数据引擎的创建  
让所有的灵活性成为可能的开关是提供给ANSI SQL的MySQL扩展——TYPE参数。MySQL能够让你在表格这一层指定数据库引擎，所以它们有时候也指的是table formats。下面的示例代码表明了如何创建分别使用MyISAM、ISAM和HEAP引擎的表格。要注意，创建每个表格的代码是相同的，除了最后的 TYPE参数，这一参数用来指定数据引擎。

**以下为引用的内容：**

**[sql]** [view plain](http://blog.csdn.net/a764340703/article/details/64129772" \o "view plain) [copy](http://blog.csdn.net/a764340703/article/details/64129772" \o "copy)

1. **CREATE** **TABLE** tblMyISAM (
2. id **INT** NOT NULL AUTO\_INCREMENT,
3. **PRIMARY** **KEY** (id),
4. value\_a TINYINT
5. ) TYPE=MyISAM
6. **CREATE** **TABLE** tblISAM (
7. id **INT** NOT NULL AUTO\_INCREMENT,
8. **PRIMARY** **KEY** (id),
9. value\_a TINYINT
10. ) TYPE=ISAM
11. **CREATE** **TABLE** tblHeap (
12. id **INT** NOT NULL AUTO\_INCREMENT,
13. **PRIMARY** **KEY** (id),
14. value\_a TINYINT
15. ) TYPE=Heap

你也可以使用ALTER TABLE命令，把原有的表格从一个引擎移动到另一个引擎。下面的代码显示了如何使用ALTER TABLE把MyISAM表格移动到InnoDB的引擎：

**以下为引用的内容：**

**[sql]** [view plain](http://blog.csdn.net/a764340703/article/details/64129772" \o "view plain) [copy](http://blog.csdn.net/a764340703/article/details/64129772" \o "copy)

1. **ALTER** **TABLE** tblMyISAM CHANGE TYPE=InnoDB

MySQL用三步来实现这一目的。首先，这个表格的一个副本被创建。然后，任何输入数据的改变都被排入队列，同时这个副本被移动到另一个引擎。最后，任何排入队列的数据改变都被送交到新的表格里，而原来的表格被删除。

**[sql]** [view plain](http://blog.csdn.net/a764340703/article/details/64129772" \o "view plain) [copy](http://blog.csdn.net/a764340703/article/details/64129772" \o "copy)

1. **ALTER** **TABLE**捷径

如果只是想把表格从ISAM更新为MyISAM，你可以使用MySQL\_convert\_table\_format命令，而不需要编写ALTER TABLE表达式。

你可以使用SHOW TABLE命令（这是MySQL对ANSI标准的另一个扩展）来确定哪个引擎在管理着特定的表格。SHOW TABLE会返回一个带有多数据列的结果集，你可以用这个结果集来查询获得所有类型的信息：数据库引擎的名称在Type字段里。下面的示例代码说明了 SHOW TABLE的用法：

**[sql]** [view plain](http://blog.csdn.net/a764340703/article/details/64129772" \o "view plain) [copy](http://blog.csdn.net/a764340703/article/details/64129772" \o "copy)

1. SHOW **TABLE** STATUS **FROM** tblInnoDB

你可以用SHOW CREATE TABLE [TableName]来取回SHOW TABLE能够取回的信息。   
一般情况下，MySQL会默认提供多种存储引擎，可以通过下面的查看:  
（1）看你的MySQL现在已提供什么存储引擎: mysql> show engines；  
（2）看你的MySQL当前默认的存储引擎: mysql> show variables like '%storage\_engine%'；  
（3）你要看某个表用了什么引擎(在显示结果里参数engine后面的就表示该表当前用的存储引擎): mysql> show create table 表名；  
最后，如果你想使用没有被编译成MySQL也没有被激活的引擎，那是没有用的，MySQL不会提示这一点。而它只会给你提供一个缺省格式（MyISAM）的表格。除了使用缺省的表格格式外，还有办法让MySQL给出错误提示，但是就现在而言，如果不能肯定特定的数据库引擎是否可用的话，你要使用SHOW TABLE来检查表格格式。  
更多的选择意味着更好的性能  
用于特定表格的引擎都需要重新编译和追踪，考虑到这种的额外复杂性，为什么你还是想要使用非缺省的数据库引擎呢？答案很简单：要调整数据库来满足你的要求。  
可以肯定的是，MyISAM的确快，但是如果你的逻辑设计需要事务处理，你就可以自由使用支持事务处理的引擎。进一步讲，由于MySQL能够允许你在表格这一层应用数据库引擎，所以你可以只对需要事务处理的表格来进行性能优化，而把不需要事务处理的表格交给更加轻便的MyISAM引擎。对于 MySQL而言，灵活性才是关键。

**性能[测试](http://lib.csdn.net/base/softwaretest" \o "软件测试知识库" \t "http://blog.csdn.net/a764340703/article/details/_blank)**所有的性能测试在：Micrisoft window xp sp2 ， Intel(R) Pentinum(R) M processor 1.6oGHz 1G 内存的电脑上测试。  
测试方法：连续提交10个query， 表记录总数：38万 ， 时间单位 s  
引擎类型MyISAMInnoDB  性能相差  
count 0.00083573.01633609  
查询主键   0.005708  0.157427.57  
查询非主键   24.01 80.37  3.348  
更新主键   0.008124  0.8183100.7  
更新非主键   0.004141  0.02625  6.338  
插入   0.004188  0.369488.21  
（1）加了索引以后，对于MyISAM查询可以加快：4 206.09733倍，对InnoDB查询加快510.72921倍，同时对MyISAM更新速度减慢为原来的1/2，InnoDB的更新速度减慢为原来的1/30。要看情况决定是否要加索引，比如不查询的log表，不要做任何的索引。  
（2）如果你的数据量是百万级别的，并且没有任何的事务处理，那么用MyISAM是性能最好的选择。  
（3）InnoDB表的大小更加的大，用MyISAM可省很多的硬盘空间。

在我们测试的这个38w的表中，表占用空间的情况如下：

引擎类型MyISAM   InnoDB  
数据  53,924 KB   58,976 KB  
索引 13,640 KB   21,072 KB

占用总空间 67,564 KB   80,048 KB

另外一个176W万记录的表， 表占用空间的情况如下：  
引擎类型MyIsam   InnorDB  
数据  56,166 KB   90,736 KB  
索引  67,103 KB   88,848 KB

占用总空间  123,269 KB179,584 KB  
  
**其他**  
    MySQL 官方对InnoDB是这样解释的：InnoDB给MySQL提供了具有提交、回滚和崩溃恢复能力的事务安全（ACID兼容）存储引擎。InnoDB锁定在行级并且也在SELECT语句提供一个**[Oracle](http://lib.csdn.net/base/oracle" \o "Oracle知识库" \t "http://blog.csdn.net/a764340703/article/details/_blank)**风格一致的非锁定读，这些特色增加了多用户部署和性能。没有在InnoDB中扩大锁定的需要，因为在InnoDB中行级锁定适合非常小的空间。InnoDB也支持FOREIGN KEY强制。在SQL查询中，你可以自由地将InnoDB类型的表与其它MySQL的表的类型混合起来，甚至在同一个查询中也可以混合。  
    InnoDB是为处理巨**[大数据](http://lib.csdn.net/base/hadoop" \o "Hadoop知识库" \t "http://blog.csdn.net/a764340703/article/details/_blank)**量时的最大性能设计，它的CPU效率可能是任何其它基于磁盘的关系数据库引擎所不能匹敌的。  
    InnoDB存储引擎被完全与MySQL服务器整合，InnoDB存储引擎为在主内存中缓存数据和索引而维持它自己的缓冲池。InnoDB存储它的表＆索引在一个表空间中，表空间可以包含数个文件（或原始磁盘分区）。这与MyISAM表不同，比如在MyISAM表中每个表被存在分离的文件中。InnoDB 表可以是任何尺寸，即使在文件尺寸被限制为2GB的**[操作系统](http://lib.csdn.net/base/operatingsystem" \o "操作系统知识库" \t "http://blog.csdn.net/a764340703/article/details/_blank)**上。  
    InnoDB默认地被包含在MySQL二进制分发中。Windows Essentials installer使InnoDB成为Windows上MySQL的默认表。  
    InnoDB被用来在众多需要高性能的大型数据库站点上产生。著名的Internet新闻站点Slashdot.org运行在InnoDB上。 Mytrix, Inc.在InnoDB上存储超过1TB的数据，还有一些其它站点在InnoDB上处理平均每秒800次插入/更新的.