# [通过查看mysql 配置参数、状态来优化你的mysql](https://www.cnblogs.com/zengkefu/p/5634858.html)

mysql的监控方法大致分为两类：

1.连接到mysql数据库内部，使用show status，show variables，flush status 来查看mysql的各种性能指标。

2. 直接使用mysqladmin查看其性能指标，例如：

UserParameter=mysql.uptime,mysqladmin -uroot status|cut -f2 -d":"|cut -f1 -d"T"

mysqladmin两个参数，status，extended-status

shell > mysqladmin  -uroot -ppassword  variables status

可得到以下信息（后面详解）

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

Uptime: 4557887      #mysql运行的秒数

Threads: 1                #连接数

Questions: 1684130    #The number of questions (queries) from clients since the server was started.

Slow queries: 0      #The number of queries that have taken more than long\_query\_time seconds

Opens: 221872     #The number of tables the server has opened.

Flush tables: 1     #The number of flush-\*, refresh, and reload commands the server has executed.

Open tables: 64     #The number of tables that currently are open.

Queries per second avg: 0.369  #从上次运行开始计算，每秒钟平均查询次数

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

Questions = Com\_\* + Qcache\_hits

最完整的信息

shell > mysqladmin  -uroot -ppassword  variables extended-status

其他的信息

shell > /usr/libexec/mysqld --verbose --help （这个命令生成所有mysqld选项和可配置变量的列表 ）

mysql>SHOW STATUS;   （服务器状态变量，运行服务器的统计和状态指标）

mysql> SHOW VARIABLES;（服务器系统变量，实际上使用的变量的值）

或者

mysql>SHOW STATUS LIKE  '%变量名% ' ;

对配置参数的说明：

配置参数的格式如下：（shell > mysqladmin  -uroot -ppassword  variables extended-status）

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

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

| Variable\_name                           |              Value                                                      |

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

| auto\_increment\_increment                |          1                                                          |

| auto\_increment\_offset                   |              1                                                          |

| automatic\_sp\_privileges                 |            ON                                                         |

.........

注：value 值的单位是byte ，要得到M ，需除以2次1024

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

Uptime                             4405546

MySQL服务器已经运行的秒数

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

auto\_increment\_increment             1

auto\_increment\_offset                    1

两个变量值都只能为1到65,535之间的整数值。设置为非整数值，则会给出错误。

这两个变量影响AUTO\_INCREMENT列。

auto\_increment\_increment控制列中的值的增量值（步进量）。

auto\_increment\_offset确定AUTO\_INCREMENT列值的初始值。

一般不去更改。更改方法：mysql> SET @auto\_increment\_offset=5;

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

max\_connections                    100

table\_cache                             64

open\_files\_limit                       1024

Open\_tables                           64

Opened\_tables                       187690

几个参数的关系：

table\_cache \* 2 + max\_connections＝max\_open\_files

max\_connections

默认为100

mysql>show processlist;

mysql>show full processlist;

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

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

max\_open\_files 由 open\_files\_limit 参数决定。

mysql打开的最大文件数，受两个参数的影响：系统打开的最大文件数（ulimit -n）和   open\_files\_limit 。

加大max\_open\_files的值

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

在/etc/my.cnf加入open\_files\_limit=8192

在/etc/security/limits.conf添加

\*               soft    nofile          8192

\*               hard    nofile          8192

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

最好用sysctl或者修改/etc/sysctl.conf文件，同时还要在配置文件中把open\_files\_limit这个参数增大，对于4G内存服务器，open\_files\_limit至少要增大到4096，非特殊情况，设置成8192就可以了。

table\_cache

MySQL 5.0升级到5.1，table\_cache 改名table\_open\_cache

设置表高速缓存的数目。

表缓存的说明：

当 Mysql 访问一个表时，如果该表在缓存中已经被打开，则可以直接访问缓存；如果还没有被缓存，但是在 Mysql 表缓冲区中还有空间，那么这个表就被打开并放入表缓冲区；如果表缓存满了，则会按照一定的规则将当前未用的表释放，或者临时扩大表缓存来存放，使用表缓存的好处是可以更快速地访问表中的内容。

每个连接进来，都会至少打开一个表缓存。因此， table\_cache 的大小应与 max\_connections 的设置有关。例如，对于 200 个并行运行的连接，应该让表的缓存至少有 200 × N ，这里 N 是网站程序一次查询所用到的表的最大值。

每个线程会独自持有一个数据文件的文件描述符，而索引文件的文件描述符是公用的。当table cache不够用的时候，MySQL会采用LRU算法踢掉最长时间没有使用的表。如果table\_cache设置过小，MySQL就会反复打开、关闭 frm文件，造成一定的性能损失。如果table\_cache设置过大，MySQL将会消耗很多CPU去做 table cache的算法运算。

而InnoDB的元数据管理是放在共享表空间里面做的，所以获取表的结构不需要去反复解析frm文件，这是比MyISAM强的地方。即使 table\_cache设置过小，对于InnoDB的影响也是很小的，因为它根本不需要反复打开、关闭frm文件去获取元数据。

合理设置table\_cache的大小：通过查看open\_tables，Opened\_tables，Flush tables 的值来比较。

察看当前的表缓存情况：

shell > mysqladmin  -uroot -ppassword  variables status

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

Opens: 221872   则是已经打开的表的数量。

Flush tables: 1

Open tables: 64   是当前打开的表的数量

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

mysql> show global status like 'open%\_tables';

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

open\_tables 是当前打开的表的数量，

Opened\_tables 表示打开过的表数量

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

清空表缓存

mysql> flush tables;

如果发现 open\_tables 接近 table\_cache 的时候，如果 Opened\_tables 随着重新运行 SHOW STATUS 命令快速增加，就说明缓存命中率不够。并且多次执行FLUSH TABLES(通过shell > mysqladmin  -uroot -ppassword  variables status )，那就说明可能 table\_cache 设置的偏小，经常需要将缓存的表清出，将新的表放入缓存，这时可以考虑增加这个参数的大小来改善访问的效率。

如果 Open\_tables 比 table\_cache 设置小很多，就说明table\_cache 设的太大了。

table\_cache的值在2G内存以下的机器中的值默认时256到512，如果机器有4G内存,则默认这个值是2048，但这决意味着机器内存越大，这个值应该越大，因为table\_cache加大后，使得mysql对SQL响应的速度更快了，不可避免的会产生更多的死锁（dead lock），这样反而使得数据库整个一套操作慢了下来，严重影响性能。

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。

对于有1G内存的机器，推荐值是128－256。

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

key\_buffer\_size                         67108864（/1024/1024=64M）

Key\_read\_requests                   40944  从缓存读键的数据块的请求数。

Key\_reads                                 2711     从硬盘读取键的数据块的次数。

Key\_write\_requests    将键的数据块写入缓存的请求数。

Key\_writes       向硬盘写入将键的数据块的物理写操作的次数。

（获得信息：

shell > mysqladmin  -uroot -ppassword  variables extended-status

shell>mysqladmin -uroot -ppassword  variable status

mysql> show status like '%key\_read%';

）

key\_buffer\_size设置索引块（index blocks）缓存的大小，保存了 MyISAM 表的索引块。它被所有线程共享，决定了数据库索引处理的速度，尤其是索引读的速度。理想情况下，对于这些块的请求应该来自于内存，而不是来自于磁盘。

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。

key\_buffer\_size: 如果不使用MyISAM存储引擎，16MB足以，用来缓存一些系统表信息等。如果使用 MyISAM存储引擎，在内存允许的情况下，尽可能将所有索引放入内存，简单来说就是“越大越好”

合理设置key\_buffer\_size的方法：

查看Key\_read\_requests和Key\_reads的比例，

Key\_reads 代表命中磁盘的请求个数， Key\_read\_requests 是总数。命中磁盘的读请求数除以读请求总数就是不中比率。如果每 1,000 个请求中命中磁盘的数目超过 1 个，就应该考虑增大关键字缓冲区了。

key\_reads / key\_read\_requests的值应该尽可能的低，比如1:100，1:1000 ，1:10000。

对于内存在4GB左右的服务器该参数可设置为256M或384M。

注意：该参数值设置的过大反而会是服务器整体效率降低！

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

256MB内存和许多表，想要在中等数量的客户时获得最大性能，应使用：

shell> mysqld\_safe --key\_buffer\_size=64M --table\_cache=256 --sort\_buffer\_size=4M --read\_buffer\_size=1M &

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

每个连接到MySQL服务器的线程都需要有自己的缓冲，默认为其分配256K。事务开始之后，则需要增加更多的空间。运行较小的查询可能仅给指定的线程增加少量的内存消耗，例如存储查询语句的空间等。但如果对数据表做复杂的操作比较复杂，例如排序则需要使用临时表，此时会分配大约read\_buffer\_size，sort\_buffer\_size，read\_rnd\_buffer\_size，tmp\_table\_size大小的内存空间。不过它们只是在需要的时候才分配，并且在那些操作做完之后就释放了。

myisam\_sort\_buffer\_size                 8388608

当在REPAIR TABLE或用CREATE INDEX创建索引或ALTER TABLE过程中排序 MyISAM索引分配之缓冲区。

sort\_buffer\_size                         2097144

每个排序线程分配的缓冲区的大小。增加该值可以加快ORDER BY或GROUP BY操作。

注意：该参数对应的分配内存是每个连接独享，如果有100个连接，那么实际分配的总共排序缓冲区大小为100 × 6 ＝ 600MB。所以，对于内存在4GB左右的服务器推荐设置为6-8M。

mysql> SHOW STATUS LIKE "sort%";

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

Sort\_merge\_passes  1

 Sort\_range         79192

 Sort\_rows          2066532

 Sort\_scan          44006

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

 如果 sort\_merge\_passes 很大，就表示需要注意 sort\_buffer\_size。

当 MySQL 必须要进行排序时，就会在从磁盘上读取数据时分配一个排序缓冲区来存放这些数据行。如果要排序的数据太大，那么数据就必须保存到磁盘上的临时文件中，并再次进行排序。如果 sort\_merge\_passes 状态变量很大，这就指示了磁盘的活动情况。

read\_buffer\_size                          131072

（show variables like 'read%';）

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

read\_buffer\_size      1048576

read\_rnd\_buffer\_size  524288

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

read\_buffer\_size是MySql读入缓冲区大小。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。

每个线程连续扫描时为扫描的每个表分配的缓冲区的大小(字节)。如果进行多次连续扫描，可能需要增加该值， 默认值为131072。和sort\_buffer\_size一样，该参数对应的分配内存也是每连接独享。

read\_rnd\_buffer\_size

read\_rnd\_buffer\_size是MySql的随机读缓冲区大小。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。该参数对应的分配内存也是每连接独享。

join\_buffer\_size                          131072

联合查询操作所能使用的缓冲区大小，和sort\_buffer\_size一样，该参数对应的分配内存也是每个连接独享。

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

max\_allowed\_packet                      1048576

net\_buffer\_length                           16384

包消息缓冲区初始化为net\_buffer\_length字节，但需要时可以增长到max\_allowed\_packet字节。该值默认很小，以捕获大的(可能是错误的)数据包。

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

 thread\_stack                            196608

每个线程的堆栈大小。用crash-me测试检测出的许多限制取决于该值。 默认值足够大，可以满足普通操作。

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

thread\_cache\_size                        0

query\_cache\_size                         0

tmp\_table\_size                          33554432

innodb\_thread\_concurrency             8

max\_connections                         100

max\_connect\_errors                      10

（获得信息：

shell > mysqladmin  -uroot -ppassword  variables extended-status

shell>mysqladmin -uroot -ppassword  variable status

）

thread\_cache\_size

（

mysql> show status LIKE 'threads%';

）

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

 Threads\_cached     27

 Threads\_connected  15

 Threads\_created    838610

 Threads\_running    3

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

线程缓存。mysqld 在接收连接时会根据需要生成线程。在一个连接变化很快的繁忙服务器上，对线程进行缓存便于以后使用可以加快最初的连接。

此处重要的值是 Threads\_created，每次 mysqld 需要创建一个新线程时，这个值都会增加。如果这个数字在连续执行 SHOW STATUS 命令时快速增加，就应该尝试增大线程缓存。

query\_cache\_size

mysql> SHOW VARIABLES LIKE 'have\_query\_cache';

mysql> show variables like '%query%';

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

 ft\_query\_expansion\_limit      20

 have\_query\_cache              YES

 long\_query\_time               10.000000

 query\_alloc\_block\_size        8192

 query\_cache\_limit             1048576

 query\_cache\_min\_res\_unit      4096

 query\_cache\_size              0

 query\_cache\_type              ON

 query\_cache\_wlock\_invalidate  OFF

 query\_prealloc\_size           8192

 slow\_query\_log                OFF

 slow\_query\_log\_file           /var/run/mysqld/mysqld-slow.log

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

have\_query\_cache              是否有查询缓存

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\_type            变量影响其工作方式。这个变量可以设置为下面的值：

0 或OFF 将阻止缓存或查询缓存结果。

1 或ON 将允许缓存，以SELECT SQL\_NO\_CACHE 开始的查询语句除外。

2 或DEMAND ，      仅对以SELECT SQL\_CACHE 开始的那些查询语句启用缓存。

如果全部使用innodb存储引擎，建议为0，如果使用MyISAM 存储引擎，建议为2

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。

query\_cache\_size   为了存储老的查询结果而分配的内存数量 (以字节指定) 。如果设置它为 0 ，查询缓冲将被禁止(缺省值为 0 )。 根据 命中率(Qcache\_hits/(Qcache\_hits+Qcache\_inserts)\*100))进行调整，一般不建议太大，256MB可能已经差不多了，大型的配置型静态数据可适当调大

mysql> SHOW STATUS LIKE 'qcache%';

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

 Qcache\_free\_blocks       5216

 Qcache\_free\_memory       14640664

 Qcache\_hits              2581646882

 Qcache\_inserts           360210964

 Qcache\_lowmem\_prunes     281680433

 Qcache\_not\_cached        79740667

 Qcache\_queries\_in\_cache  16927

 Qcache\_total\_blocks      47042

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

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。 未命中然后插入。

Qcache\_lowmem\_prunes  的值非常大，则表明经常出现缓冲不够的情况,同时Qcache\_hits的值非常大，则表明查询缓冲使用非常频繁，此时需要增加缓冲大小，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。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。

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。

Total number of queries = Qcache\_inserts + Qcache\_hits + Qcache\_not\_cached.

查询命中率=Qcache\_hits -Qcache\_inserts /Qcache\_hits

查询插入率=Qcache\_inserts / Com\_select;

未插入率 = Qcache\_not\_cached / Com\_select;

很多 LAMP 应用程序都严重依赖于数据库，但却会反复执行相同的查询。每次执行查询时，数据库都必须要执行相同的工作 —— 对查询进行分析，确定如何执行查询，从磁盘中加载信息，然后将结果返回给客户机。MySQL 有一个特性称为查询缓存，查询缓存会存储一个 SELECT 查询的文本与被传送到客户端的相应结果。如果之后接收到一个同样的查询，服务器将从查询缓存中检索结果，而不是再次分析和执行这个同样的查询。在很多情况下，这会极大地提高性能。不过，问题是查询缓存在默认情况下是禁用的。

通常，间隔几秒显示这些变量就可以看出区别，这可以帮助确定缓存是否正在有效地使用。运行 FLUSH STATUS 可以重置一些计数器，如果服务器已经运行了一段时间，这会非常有帮助。

使用非常大的查询缓存，期望可以缓存所有东西，这种想法非常诱人。但如果表有变动时，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，建系把该功能禁掉。

作为一条规则，如果 FLUSH QUERY CACHE 占用了很长时间，那就说明缓存太大了。

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

wait\_timeout                            28800

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

connect\_timeout                        10

mysqld服务器用Bad handshake响应前等待连接包的秒数。

interactive\_timeout                      28800

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。

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

mysql> SHOW STATUS LIKE "com\_select";

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

 Com\_select     318243

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

mysql> SHOW STATUS LIKE "handler\_read\_rnd\_next";

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

Handler\_read\_rnd\_next  165959471

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

MySQL 也会分配一些内存来读取表。理想情况下，索引提供了足够多的信息，可以只读入所需要的行，但是有时候查询（设计不佳或数据本性使然）需要读取表中大量数据。要理解这种行为，需要知道运行了多少个 SELECT 语句，以及需要读取表中的下一行数据的次数（而不是通过索引直接访问）。

Handler\_read\_rnd\_next / Com\_select 得出了表扫描比率 —— 在本例中是 521:1。如果该值超过 4000，就应该查看 read\_buffer\_size，例如 read\_buffer\_size = 4M。如果这个数字超过了 8M，就应该与开发人员讨论一下对这些查询进行调优了！

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

mysql> SHOW STATUS LIKE 'created\_tmp%';

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

 Created\_tmp\_disk\_tables  30660

 Created\_tmp\_files        2

 Created\_tmp\_tables       32912

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

临时表可以在更高级的查询中使用，其中数据在进一步进行处理（例如 GROUP BY 字句）之前，都必须先保存到临时表中；理想情况下，在内存中创建临时表。但是如果临时表变得太大，就需要写入磁盘中。

每次使用临时表都会增大 Created\_tmp\_tables；基于磁盘的表也会增大 Created\_tmp\_disk\_tables。对于这个比率，并没有什么严格的规则，因为这依赖于所涉及的查询。长时间观察 Created\_tmp\_disk\_tables 会显示所创建的磁盘表的比率，您可以确定设置的效率。 tmp\_table\_size 和 max\_heap\_table\_size 都可以控制临时表的最大大小，因此请确保在 my.cnf 中对这两个值都进行了设置。

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

日志相关

log-bin=mysql-bin

binlog\_format=mixed

mysql-bin.000001、mysql-bin.000002等文件是数据库的操作日志，例如UPDATE一个表，或者DELETE一些数据，即使该语句没有匹配的数据，这个命令也会存储到日志文件中，还包括每个语句执行的时间，也会记录进去的。

=================================================================================================

 InnoDB

innodb\_buffer\_pool\_size

　　innodb\_buffer\_pool\_size 定义了 InnoDB 存储引擎的表数据和索引数据的最大内存缓冲区大小。和 MyISAM 存储引擎不同， MyISAM 的 key\_buffer\_size 只能缓存索引键，而 innodb\_buffer\_pool\_size 却可以缓存数据块和索引键。适当的增加这个参数的大小，可以有效的减少 InnoDB 类型的表的磁盘 I/O 。为Innodb加速优化首要参数。默认值8M

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。如果你的数据量不大，并且不会暴增，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

mysql>show variables like 'innodb%';

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

innodb\_adaptive\_hash\_index               ON

 innodb\_additional\_mem\_pool\_size          2097152

 innodb\_autoextend\_increment              8

 innodb\_autoinc\_lock\_mode                 1

 innodb\_buffer\_pool\_size                  67108864

 innodb\_checksums                         ON

 innodb\_commit\_concurrency                0

 innodb\_concurrency\_tickets               500

 innodb\_data\_file\_path                    ibdata1:10M:autoextend

 innodb\_data\_home\_dir                     /var/lib/mysql

 innodb\_doublewrite                       ON

 innodb\_fast\_shutdown                     1

 innodb\_file\_io\_threads                   4

 innodb\_file\_per\_table                    OFF

 innodb\_flush\_log\_at\_trx\_commit           1

 innodb\_flush\_method

 innodb\_force\_recovery                    0

 innodb\_lock\_wait\_timeout                 50

 innodb\_locks\_unsafe\_for\_binlog           OFF

 innodb\_log\_buffer\_size                   8388608

 innodb\_log\_file\_size                     16777216

 innodb\_log\_files\_in\_group                2

 innodb\_log\_group\_home\_dir                /var/lib/mysql

 innodb\_max\_dirty\_pages\_pct               90

 innodb\_max\_purge\_lag                     0

 innodb\_mirrored\_log\_groups               1

 innodb\_open\_files                        300

 innodb\_rollback\_on\_timeout               OFF

 innodb\_stats\_method                      nulls\_equal

 innodb\_stats\_on\_metadata                 ON

 innodb\_support\_xa                        ON

 innodb\_sync\_spin\_loops                   20

 innodb\_table\_locks                       ON

 innodb\_thread\_concurrency                8

 innodb\_thread\_sleep\_delay                10000

 innodb\_use\_legacy\_cardinality\_algorithm  ON

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

mysql>show status like 'innodb%';

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

Innodb\_buffer\_pool\_pages\_data     2559       分配出去， 正在被使用页的数量，包括脏页。单位是page

 Innodb\_buffer\_pool\_pages\_dirty    0            脏页但没有被flush除去的页面数。单位是page

 Innodb\_buffer\_pool\_pages\_flushed  795       已经flush的页面数。单位是page

 Innodb\_buffer\_pool\_pages\_free     1473     当前空闲页面数。单位是page

 Innodb\_buffer\_pool\_pages\_misc     64         缓存池中当前已经被用作管理用途或hash index而不能用作为普通数据页的数目。单位是page

 Innodb\_buffer\_pool\_pages\_total    4096       缓冲区总共的页面数。单位是page

 Innodb\_buffer\_pool\_read\_ahead\_rnd 8        随机预读的次数

 Innodb\_buffer\_pool\_read\_ahead\_seq 1        顺序预读的次数

 Innodb\_buffer\_pool\_read\_requests  1725871    从缓冲池中读取页的次数

 Innodb\_buffer\_pool\_reads          2108              从磁盘读取页的次数。缓冲池里面没有， 就会从磁盘读取

 Innodb\_buffer\_pool\_wait\_free      0                缓冲池等待空闲页的次数，当需要空闲块而系统中没有时，就会等待空闲页面

 Innodb\_buffer\_pool\_write\_requests 2296       缓冲池总共发出的写请求次数

 Innodb\_data\_fsyncs                695              总共完成的fsync次数

 Innodb\_data\_pending\_fsyncs        0            innodb当前等待的fsync次数

 Innodb\_data\_pending\_reads         0           innodb当前等待的读的次数

 Innodb\_data\_pending\_writes        0            innodb当前等待的写的次数

 Innodb\_data\_read                  44044288          总共读入的字节数

 Innodb\_data\_reads                 2191              innodb完成的读的次数

 Innodb\_data\_writes                1296              innodb完成的写的次数

 Innodb\_data\_written               26440192     总共写出的字节数

 Innodb\_dblwr\_pages\_written        795

 Innodb\_dblwr\_writes               90

 Innodb\_log\_waits                  0                   因日志缓存太小而必须等待其被写入所造成的等待数。单位是次。

 Innodb\_log\_write\_requests         263

 Innodb\_log\_writes                 410

 Innodb\_os\_log\_fsyncs              500

 Innodb\_os\_log\_pending\_fsyncs      0

 Innodb\_os\_log\_pending\_writes      0

 Innodb\_os\_log\_written             343552

 Innodb\_page\_size                  16384

 Innodb\_pages\_created              4

 Innodb\_pages\_read                 2555

 Innodb\_pages\_written              795

 Innodb\_row\_lock\_current\_waits     0

 Innodb\_row\_lock\_time              0

 Innodb\_row\_lock\_time\_avg          0

 Innodb\_row\_lock\_time\_max          0

 Innodb\_row\_lock\_waits             0

 Innodb\_rows\_deleted               0

 Innodb\_rows\_inserted              352

 Innodb\_rows\_read                  818617

 Innodb\_rows\_updated               88

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

 命中率=innodb\_buffer\_pool\_read\_requests / (innodb\_buffer\_pool\_read\_requests + innodb\_buffer\_pool\_read\_ahead + innodb\_buffer\_pool\_reads)

innodb\_buffer\_pool\_size: 如果不使用InnoDB存储引擎，可以不用调整这个参数，如果需要使用，在内存允许的情况下，尽可能将所有的InnoDB数据文件存放如内存中，同样将但来说也是“越大越好”

 innodb\_additional\_pool\_size

这个值不用分配太大，系统可以自动调。不用设置太高。通常比较大数据设置16Ｍ够用了，如果表比较多，可以适当的增大。如果这个值自动增加，会在error log有中显示的。20M足够了。

innodb\_log\_file\_size

作用：指定日志的大小

分配原则：几个日志成员大小加起来差不多和你的innodb\_buffer\_pool\_size相等。在高写入负载尤其是大数据集的情况下很重要。这个值越大则性能相对越高，但是要注意到可能会增加恢复时间。

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。

innodb\_log\_buffer\_size：

作用：事务在内存中的缓冲。

分配原则：控制在2-8M.这个值不用太多的。他里面的内存一般一秒钟写到磁盘一次。具体写入方式和你的事务提交方式有关。一般最大指定为3M比较合适。

参考：Innodb\_os\_log\_written(show global status 可以拿到)

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blog字段，可以考虑增加这个参数的值。

 默认的设置在中等强度写入负载以及较短事务的情况下，服务器性能还可以。如果存在更新操作峰值或者负载较大，就应该考虑加大它的值了。如果它的值设置太高了，可能会浪费内存 -- 它每秒都会刷新一次，因此无需设置超过1秒所需的内存空间。通常 8-16MB 就足够了。越小的系统它的值越小。

innodb\_flush\_logs\_at\_trx\_commit

作用：控制事务的提交方式

分配原则：这个参数只有3个值，0，1，2请确认一下自已能接受的级别。默认为1，主库请不要更改了。性能更高的可以设置为0或是2，但会丢失一秒钟的事务。

值为1时：innodb 的事务LOG在每次提交后写入日志文件，并对日志做刷新到磁盘。这个可以做到不丢任何一个事务。

值为2事，也就是不把日志刷新到磁盘上，而只刷新到操作系统的缓存上。日志仍然会每秒刷新到磁盘中去，因此通常不会丢失每秒1-2次更新的消耗。如果设置为 0 就快很多了，不过也相对不安全了 -- MySQL服务器崩溃时就会丢失一些事务。设置为 2 只会丢失刷新到操作系统缓存的那部分事务。

 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。

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。

 InnoDB 默认会将所有的数据库InnoDB引擎的表数据存储在一个共享空间中：ibdata1，增删数据库的时候，ibdata1文件不会自动收缩，单个数据库的备份也将成为问题。通常只能将数据使用mysqldump 导出，然后再导入解决这个问题。

查看是否开启：

mysql> show variables like ‘%per\_table%’;

开启

innodb\_file\_per\_table=1

请适当的增加innodb\_open\_files

innodb\_open\_files

作用：限制Innodb能打开的表的数据。

分配原则：如果库里的表特别多的情况，请增加这个。这个值默认是300。这个值必须超过你配置的innodb\_data\_file\_path个数。

请适当的增加table\_cache

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。UNIX可以设置：fdatasync (默认设置)，O\_DIRECT，和O\_DSYNC

O\_DIRECT跳过了操作系统的文件系统Disk Cache，让MySQL直接读写磁盘。 有数据表明，如果是大量随机写入操作，O\_DIRECT会提升效率。但是顺序写入和读取效率都会降低。

innodb\_max\_dirty\_pages\_pct

作用：控制Innodb的脏页在缓冲中在那个百分比之下，值在范围1-100,默认为90.

O\_DIRECT的flush\_method更适合于操作系统内存有限的情况下（可以避免不必要的对交换空间的读写操作），否则，它会由于禁用了os的缓冲降低对数据的读写操作的效能。

使用memlock可以避免MySQL内存进入swap