SHOW ENGINE INNODB解释

```
mysql> show engine innodb status \G;
*************************** 1. row ***************************
Type: InnoDB
Name:
Status:
=====================================
2015-10-08 13:39:09 7f5b1730c700 INNODB MONITOR OUTPUT
=====================================
Per second averages calculated from the last 56 seconds
-----------------
BACKGROUND THREAD
-----------------
srv_master_thread loops: 331131 srv_active, 0 srv_shutdown, 2605615 srv_idle
srv_master_thread log flush and writes: 2936746
```

主循环进行了331131次

如果有高并发的工作负载，就要关注它，包含了两种数据：事件计数器，和可选的当前等待线程列表，如果有性能上的瓶颈可以使用它些信息找出来。

SEMAPHORES (信号量)

OS WAIT ARRAY INFO: reservation count 481149(保留计数)
OS WAIT ARRAY INFO: signal count 499176(信号计数)单位/次

上面两行是操作系统等待数组的信息，它是一个插曹数组，innodb在数组里为信号量保留一些插曹，操作系统用这些信号量给线程发送信号，让线程可以继续运行来完成它们等着做的事情。这两行显示出innodb使用了多少次操作系统的等待。保留计数reservation count显示了Innodb分配插曹的频度，而信号计数signal count衡量的是线程通过数组得到的信号频度。操作系统的等待要比空转等待要昂贵。

--Thread 233… has waited at ./../include/bufobuf.in line 630 for 0.00 second the semaphore:
Mutex at 0x2a9578…. created file bufobuf.c line 517,lock var 0(lock var表示)
waiter flag 0显示了多少个等待者正在等同一个互斥量
wait is ending意味着这个互斥量实际已被释放，但操作系统还没把线程调度过来运行
此处和上面一样....

Mutex spin waits 778996, rounds 9754423, OS waits 307799 自旋锁等待，它是一个低成本的，不过它是一个活跃的会浪费CPU资源，如果太多就会浪费CPU时间和无谓的上下文切换，可以用innodb_sync_spin_loops来平衡。

```
RW-shared spins 108785, rounds 2279317, OS waits 72907
RW-excl spins 99416, rounds 3538224, OS waits 97820
Spin rounds per wait: 12.52 mutex, 20.95 RW-shared, 35.59 RW-excl
------------
TRANSACTIONS
------------
Trx id counter 230962655
Purge done for trx's n:o < 230962649 undo n:o < 0 state: running but idle
History list length 1833
LIST OF TRANSACTIONS FOR EACH SESSION:
---TRANSACTION 0, not started
MySQL thread id 6644993, OS thread handle 0x7f5b1730c700, query id 21533069 localhost root init
show engine innodb status
```

如某应用的删除和更新操作的TPS(transaction per second)为1000，为每个事务分一个undo页那么一分钟就要1000*60个页，约要1GB惹每秒puerge页的数量20这样对磁盘空间有着相当高的要求，所以innodb要对undo页进行重用。当事务提交时，先将undo log放入链表中，然后判断undo页的使用空间是否小于3/4，若是则表示该undo页可以被重用，之后的新undo log记录在当前undo log的后面。由于存放undo log的列表是以记录进行组织的，而undo页可能存放着不同事务的undo log因此purge操作需要涉及磁盘的离散读取操作，是一个比较慢的过程。可以在show innodb status来看链表中undo log的数量：**History list length 1833**

当purge操作会减少该值。

```
--------
FILE I/O
--------
I/O thread 0 state: waiting for completed aio requests (insert buffer thread)
I/O thread 1 state: waiting for completed aio requests (log thread)
I/O thread 2 state: waiting for completed aio requests (read thread)
I/O thread 3 state: waiting for completed aio requests (read thread)
I/O thread 4 state: waiting for completed aio requests (read thread)
I/O thread 5 state: waiting for completed aio requests (read thread)
I/O thread 6 state: waiting for completed aio requests (write thread)
I/O thread 7 state: waiting for completed aio requests (write thread)
I/O thread 8 state: waiting for completed aio requests (write thread)
I/O thread 9 state: waiting for completed aio requests (write thread)
```

Innsert buffer thread 负责插入缓冲合并，如记录被从插入缓冲合并到表空间中
Log thread 异步刷新日志
Read thread 执行预读操作以尝试先读取innodb预感需要的数据
Write thread 刷脏缓冲

```
Pending normal aio reads: 0 [0, 0, 0, 0] , aio writes: 0 [0, 0, 0, 0] ,
 ibuf aio reads: 0, log i/o's: 0, sync i/o's: 0
Pending flushes (fsync) log: 0; buffer pool: 0
```

上面三行显示辅助线程挂起操作的数量

```
193966 OS file reads, 19421053 OS file writes, 7327496 OS fsyncs
```

上面一行显示了读写和fsync调用执行的数量
0.00 reads/s, 0 avg bytes/read, 0.00 writes/s, 0.00 fsyncs/s 显示头部时间段里每一秒执行各种操作的平均值次数

```
-------------------------------------
INSERT BUFFER AND ADAPTIVE HASH INDEX
-------------------------------------
Ibuf: size 1, free list len 19025, seg size 19027, 7229 merges
merged operations:
 insert 8331, delete mark 48206, delete 26 change buffer里的各种buffer
```

discarded operations:表示当change buffer发生merge 时表已被删除，此时就不用再将记录合并到辅助索引中了

```
 insert 0, delete mark 0, delete 0
Hash table size 10624727, node heap has 5310 buffer(s)
```

0.00 hash searches/s, 0.00 non-hash searches/s #HASH对于相等和不等有效率但范围就不行了，所以出现了non-hash searches/s可以理解为使用HSAH索引后的效率，可以通过innodb_adaptive_hash_index来禁用或启动此特性，默认为开启

```
---
LOG
---
Log sequence number 756455713849 日志序号LSN
Log flushed up to   756455713846 已经刷新到的位置
```

可以用日志号减去已经刷新的得出有多少字节没有写入到日志文件中(756455713849-756455713846 = 3字节)

```
Pages flushed up to 756455713849
```

Last checkpoint at 756455713849上一检测点，一个检测点表示一个数据和日志文件都处于已知状态，可以用于恢复
如果上一个检测点表落后日志序号太多，并差异接近该日志文件的大小，innodb会触发”疯狂刷”对影响性能。

```
0 pending log writes, 0 pending chkp writes
6563899 log i/o's done, 0.00 log i/o's/second
```

以上显示了挂起的日志操作和统计，可以将其与FILE I/O部分的值相比较，了解IO有多少是由日志子系统引起，有多少是其它原因

```
----------------------
```

BUFFER POOL AND MEMORY

Total memory allocated 5494538240; in additional pool allocated 0 显示总大小和分配的额外的大小
Dictionary memory allocated 11725807
Buffer pool size   327672 缓冲池总大小
Free buffers   8206 空闲页并可用于缓冲数据的缓冲段个数
Database pages   314156 数据库页
Old database pages 115803
Modified db pages 0 脏页 modified db*16k/1024=就是没有写到数据文件
以上显示了缓冲池度量值，以页为单位
innodb_max_dirty_pages_pct(75):当脏页的百分比超过了这个值，innodb将快速地刷写脏页，尝试让脏页的数量更低。当事务日志没有足够的空间时，innodb也将进入"激烈刷写(Furious Flushing)"模式，这就是大日志可以提升性能的一个原因。
show indoor status里的LOG里log sequence number(日志号)和log flushed up to(刷新日志)如果相差大，和BUFFER POLL AND MEMORY的修改的页modified db*16k/1024=就是没有写到数据文件。(如果是写入量大，而写入的数据不是太活跃，可以考虑把这个值设的低一点，如果写入或是更新的数据也就是热数据就可以把这个值设为95%)
Pending reads 0 等待中的读请求个数，应该保持在最低水平
Pending writes: LRU 0, flush list 0, single page 0 等待中写讲求个数
以上是显示了挂起的读写数量，这些值不与FILE IO部分的值相匹配，因为innodb可能合并许多的逻辑操作到一个物理IO操作中，它是通过LUR冲刷不ipkhet的页来释放空间给常用页的一种方法，单页的写是独立的页面写，不会被合并
Pages made young 119674, not young 1666266
0.00 youngs/s, 0.00 non-youngs/s
Pages read 188552, created 1139152, written 12386758
0.00 reads/s, 0.00 creates/s, 0.00 writes/s
No buffer pool page gets since the last printout
Pages read ahead 0.00/s, evicted without access 0.00/s, Random read ahead 0.00/s
LRU len: 314156, unzip_LRU len: 0
I/O sum[0]:cur[0], unzip sum[0]:cur[0]
Old database pages 115803：在旧区域存放着多少个页。
Pages made young：移动到新区域的有多少个页。
Pages made not young：没有移动到新区域的有多少个页。
youngs/s：每秒移动到新区域的有多少个页。
non-youngs/s：每秒没有移动到新区域的有多少个页。
young-making rate：移动到新区域的比例。
young-making not rate：没有移动到新区域的比例。
如果你没有全表扫描，发现youngs/s的值很小，那么就应该增大innodb_old_blocks_pct或者减少innodb_old_blocks_time。如果你进行了全表扫描，发现non-youngs/s的值很小，那么就应该增大innodb_old_blocks_time。

INDIVIDUAL BUFFER POOL INFO

---BUFFER POOL 0
Buffer pool size   40959
Free buffers   1026
Database pages   39266
Old database pages 14474
Modified db pages 0
Pending reads 0
Pending writes: LRU 0, flush list 0, single page 0
Pages made young 14201, not young 201852
0.00 youngs/s, 0.00 non-youngs/s
Pages read 25059, created 143586, written 1545870

```
0.00 reads/s, 0.00 creates/s, 0.00 writes/s
No buffer pool page gets since the last printout
Pages read ahead 0.00/s, evicted without access 0.00/s, Random read ahead 0.00/s
LRU len: 39266, unzip_LRU len: 0
I/O sum[0]:cur[0], unzip sum[0]:cur[0]
---BUFFER POOL 1
Buffer pool size   40959
Free buffers       1025
Database pages     39272
Old database pages 14476
Modified db pages  0
Pending reads 0
Pending writes: LRU 0, flush list 0, single page 0
Pages made young 17937, not young 129610
0.00 youngs/s, 0.00 non-youngs/s
Pages read 23711, created 142725, written 1714014
0.00 reads/s, 0.00 creates/s, 0.00 writes/s
No buffer pool page gets since the last printout
Pages read ahead 0.00/s, evicted without access 0.00/s, Random read ahead 0.00/s
LRU len: 39272, unzip_LRU len: 0
I/O sum[0]:cur[0], unzip sum[0]:cur[0]
--------------
ROW OPERATIONS
--------------
0 queries inside InnoDB, 0 queries in queue
0 read views open inside InnoDB
Main thread process no. 2373, id 140036239288064, state: sleeping
Number of rows inserted 82204507, updated 539367, deleted 77, read 8001710559
0.00 inserts/s, 0.00 updates/s, 0.00 deletes/s, 0.00 reads/s  每秒扫描的行数
单位为次
----------------------------
END OF INNODB MONITOR OUTPUT如果没看到这句说明有一个很大的死锁存在，它截断了这些
信息的输入
============================

1 row in set (0.07 sec)

ERROR:
No query specified
```