**目录**

[目录](#9239-1610814307569)

[mysql优化过程](#1140-1610814304144)

[查询优化](#1746-1610814287264)

[小表驱动大表](#8295-1610814287264)

[exists和in对比:](#4670-1610814287264)

[order by 优化](#9740-1610814287264)

[group by 优化](#6489-1610814287264)

[慢查询日志](#8360-1610814287264)

[慢查询日志是什么？](#7661-1610814287264)

[慢查询日志开启](#8690-1610814287264)

[日志分析工具 mysqldumpslow](#6840-1610814287264)

[批量数据脚本](#7680-1610814287264)

[Show Profile](#5630-1610814624475)

[全局查询日志](#7858-1610814850694)

**mysql优化过程**

**explain**

----分析 ----

1. 观察，至少跑一天，看看生产的慢sql情况

2. 开启慢查询日志，设置阙值，比如超过5秒就是慢sql，并将它抓出来。

3. explain+慢sql分析

4. show profile

5. 进行mysql数据库服务器的参数调优

----总结 ----

1. 慢查询的开启并捕获

2. explain+慢sql分析

3. show profile查询sql在Mysql服务器里面的执行细节和生命周期。

4. mysql数据库服务器的参数调优

**查询优化**

**小表驱动大表**

问题：为什么要小表驱动大表呢？

类似循环嵌套

for(int i=5;.......)
{
for(int j=1000;......)
{}
}

如果小的循环在外层，对于数据库连接来说就只连接5次，进行5000次操作，如果1000在外，则需要进行1000次数据库连接，从而浪费资源，增加消耗。这就是为什么要小表驱动大表

**exists和in对比**:

**in**

确定给定的值是否与子查询或列表中的值相匹配。in在查询的时候，首先查询子查询的表，然后将内表和外表做一个笛卡尔积，然后按照条件进行筛选。所以相对内表比较小的时候，in的速度较快。

select \* from A where id in(select id from B)

以上查询使用了in语句,in()只执行一次,它查出B表中的所有id字段并缓存起来.之后,检查A表的id是否与B表中的id相等,如果相等则将A表的记录加入结果集中,直到遍历完A表的所有记录.

**exists**

exists 指定一个子查询，检测行的存在。遍历循环外表，然后看外表中的记录有没有和内表的数据一样的。匹配上就将结果放入结果集中。sql如下：

select a.\* from A a where exists(select 1 from B b where a.id=b.id)

exists()会执行A.length次,它并不缓存exists()结果集,因为exists()结果集的内容并不重要,重要的是结果集中是否有记录,如果有则返回true,没有则返回false.

当B表比A表数据大时适合使用exists(),因为它没有那么遍历操作,只需要再执行一次查询就行。

in 是把外表和内表作hash 连接，而exists是对外表作loop循环，每次loop循环再对内表进行查询。一直以来认为exists比in效率高的说法是不准确的

**总结**: in后面跟的是小表，exists后面跟的是大表

**not in 和not exists**

　　如果查询语句使用了not in 那么内外表都进行全表扫描，没有用到索引；而not extsts 的子查询依然能用到表上的索引。所以无论那个表大，用not exists都比not in要快。

**order by 优化**

1. ORDER BY子句，尽量使用Index方式排序(指排序走索引)，避免使用FileSort方式排序

2. 如果对where条件字段未建索引，只对排序字段建索引，是不会使用索引的.

**结论**

MySQL支持二种方式的排序，FileSort和Index，**Index效率高，它指MySQL扫描索引本身完成排序**，FileSort方式效率较低。

ORDER BY满足两情况（最佳左前缀原则），会使用Index方式排序

ORDER BY语句使用索引最左前列

使用where子句与OrderBy子句条件列组合满足索引最左前列

尽可能在索引列上完成排序操作，遵照索引建的**最佳左前缀.**

**如果未在索引列上完成排序，mysql 会启动 filesort 的两种算法：双路排序和单路排序**

双路排序

MySQL4.1之前是使用双路排序，字面意思是两次扫描磁盘，最终得到数据。读取行指针和将要进行orderby操作的列，对他们进行排序，然后扫描已经排序好的列表，按照列表中的值重新从列表中读取对应的数据传输

从磁盘取排序字段，在buffer进行排序，再从磁盘取其他字段。

单路排序

取一批数据，要对磁盘进行两次扫描，众所周知，I/O是很耗时的，所以在mysql4.1之后，出现了改进的算法，就是单路排序。

从磁盘读取查询需要的所有列，按照将要进行orderby的列，在sort buffer对它们进行排序，然后扫描排序后的列表进行输出，它的效率更快一些，避免了第二次读取数据，并且把随机IO变成顺序IO，但是它会使用更多的空间，因为它把每一行都保存在内存中了。

结论及引申出的问题：

由于单路是改进的算法，总体而言好过双路

在sort\_buffer中，方法B比方法A要多占用很多空间，因为方法B是把所有字段都取出，所以有可能取出的数据的总大小超出了sort\_buffer的容量，导致每次只能取sort\_buffer容量大小的数据，进行排序（创建tmp文件，多路合并），排完再取取sort\_buffer容量大小，再排…… 从而会导致多次I/O。

结论：本来想省一次I/O操作，反而导致了大量的/O操作，反而得不偿失。

更深层次的优化策略：

增大sort\_buffer\_size参数的设置

增大max\_length\_for\_sort\_data参数的设置

**遵循如下规则，可提高Order By的速度**

Order by时select \*是一个大忌，只Query需要的字段，这点非常重要。在这里的影响是：

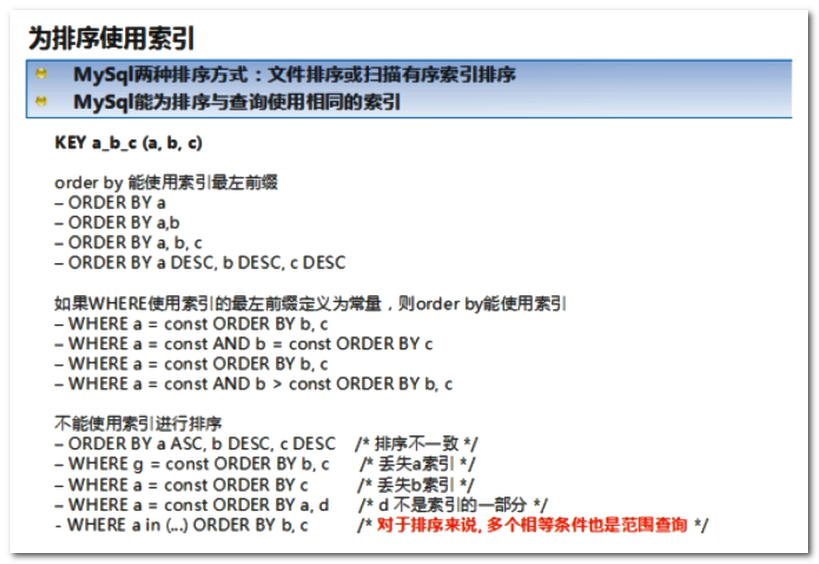
当Query的字段大小总和小于max\_length\_for\_sort\_data，而且排序字段不是TEXT|BLOB类型时，会用改进后的算法——单路排序，否则用老算法——多路排序。

两种算法的数据都有可能超出sort\_buffer的容量，超出之后，会创建tmp文件进行合并排序，导致多次I/O，但是用单路排序算法的风险会更大一些，所以要提高sort\_buffer\_size。

尝试提高 sort\_buffer\_size不管用哪种算法，提高这个参数都会提高效率，当然，要根据系统的能力去提高，因为这个参数是针对每个进程的

尝试提高max\_length\_for\_sort\_data提高这个参数，会增加用改进算法的概率。但是如果设的太高，数据总容量超出sort\_buffer\_size的概率就增大，明显症状是高的磁盘I/O活动和低的处理器使用率。

**Order By 排序索引优化的总结**



**group by 优化**

group by实质是**先排序后进行分组，遵照索引的最佳左前缀**

当无法使用索引列，增大max\_length\_for\_sort\_data参数的设置+增大sort\_buffer\_size参数的设置

where高于having，能写在where限定的条件就不要去having限定了

其余的规则均和 order by 一致.

**慢查询日志**

**慢查询日志是什么？**

慢查询日志是MySQL提供的一种日志记录，它用来**记录在MySQL中响应时间超过阀值的语句**，具体指运行时间超过**long\_query\_time**值的SQL，则会被记录到慢查询日志中。

long\_query\_time的默认值为10，意思是运行10秒以上的SQL语句会被记录下来

由他来查看哪些SQL超出了我们的最大忍耐时间值，比如一条sql执行超过5秒钟，我们就算慢SQL，希望能收集超过5秒的sql，结合之前explain进行全面分析。

**慢查询日志开启**

默认情况下，MySQL数据库没有开启慢查询日志，需要我们手动来设置这个参数。

当然，如果不是调优需要的话，一般不建议启动该参数，因为开启慢查询日志会或多或少带来一定的性能影响。慢查询日志支持将日志记录写入文件。

**查看慢查询日志是否开启**

默认情况下slow\_query\_log的值为OFF，表示慢查询日志是禁用的

可以通过设置**slow\_query\_log**的值来开启

通过SHOW VARIABLES LIKE '%slow\_query\_log%';查看 mysql 的慢查询日志是否开启

mysql> SHOW VARIABLES LIKE '%slow\_query\_log%';
+---------------------+-------------------------------+
| Variable\_name | Value |
+---------------------+-------------------------------+
| slow\_query\_log | OFF |
| slow\_query\_log\_file | /var/lib/mysql/Heygo-slow.log |
+---------------------+-------------------------------+
2 rows in set (0.00 sec)

**如何开启开启慢查询日志**

set global slow\_query\_log = 1; 开启慢查询日志

使用set global slow\_query\_log=1 开启了慢查询日志只对当前数据库生效，如果MySQL重启后则会失效。

mysql> set global slow\_query\_log = 1;
Query OK, 0 rows affected (0.07 sec)
mysql> SHOW VARIABLES LIKE '%slow\_query\_log%';
+---------------------+-------------------------------+
| Variable\_name | Value |
+---------------------+-------------------------------+
| slow\_query\_log | ON |
| slow\_query\_log\_file | /var/lib/mysql/Heygo-slow.log |
+---------------------+-------------------------------+
2 rows in set (0.00 sec)

如果要永久生效，就必须修改配置文件my.cnf（其它系统变量也是如此）

修改my.cnf文件，[mysqld]下增加或修改参数：slow\_query\_log和slow\_query\_log\_file后，然后重启MySQL服务器。也就是将如下两行配置进my.cnf文件：

[mysqld]
slow\_query\_log =1
slow\_query\_log\_file=/var/lib/mysql/Heygo-slow.log

关于慢查询的参数slow\_query\_log\_file，它指定慢查询日志文件的存放路径，系统默认会给一个缺省的文件host\_name-slow.log（如果没有指定参数slow\_query\_log\_file的话）。

**开启慢查询日志后，什么样的SQL参会记录到慢查询里面**

这个是由参数long\_query\_time控制，默认情况下long\_query\_time的值为10秒，命令：SHOW VARIABLES LIKE 'long\_query\_time%';查看慢 SQL 的阈值

mysql> SHOW VARIABLES LIKE 'long\_query\_time%';
+-----------------+-----------+
| Variable\_name | Value |
+-----------------+-----------+
| long\_query\_time | 10.000000 |
+-----------------+-----------+
1 row in set (0.01 sec)

可以使用命令修改，也可以在my.cnf参数里面修改.(以下举例使用命令设置为3秒)，设置后阈值后，需要重新连接或者新开一个回话才能看到修改值，

mysql> set global long\_query\_time=3;
Query OK, 0 rows affected (0.00 sec)

假如运行时间正好等于long\_query\_time的情况，并不会被记录下来。也就是说，在mysql源码里是判断大于long\_query\_time，而非大于等于。

查询当前系统中有多少条慢查询记录：show global status like '%Slow\_queries%';

mysql> show global status like '%Slow\_queries%';
+---------------+-------+
| Variable\_name | Value |
+---------------+-------+
| Slow\_queries | 1 |
+---------------+-------+
1 row in set (0.00 sec)

**配置版的慢查询日志**

在 /etc/my.cnf 文件的 [mysqld] 节点下配置

slow\_query\_log=1；
slow\_query\_log\_file=/var/lib/mysql/Heygo-slow.log
long\_query\_time=3；
log\_output=FILE

**日志分析工具 mysqldumpslow**

**mysqldumpslow是什么？**

在生产环境中，如果要手工分析日志，查找、分析SQL，显然是个体力活，MySQL提供了日志分析工具mysqldumpslow。

**查看 mysqldumpslow的帮助信息**

[root@Heygo mysql]# mysqldumpslow --help
Usage: mysqldumpslow [ OPTS... ] [ LOGS... ]
Parse and summarize the MySQL slow query log. Options are
--verbose verbose
--debug debug
--help write this text to standard output
-v verbose
-d debug
-s ORDER what to sort by (al, at, ar, c, l, r, t), 'at' is default
al: average lock time
ar: average rows sent
at: average query time
c: count
l: lock time
r: rows sent
t: query time
-r reverse the sort order (largest last instead of first)
-t NUM just show the top n queries
-a don't abstract all numbers to N and strings to 'S'
-n NUM abstract numbers with at least n digits within names
-g PATTERN grep: only consider stmts that include this string
-h HOSTNAME hostname of db server for \*-slow.log filename (can be wildcard),
default is '\*', i.e. match all
-i NAME name of server instance (if using mysql.server startup script)
-l don't subtract lock time from total time

**mysqldumpshow 参数解释**

s：是表示按何种方式排序

c：访问次数

l：锁定时间

r：返回记录

t：查询时间

al：平均锁定时间

ar：平均返回记录数

at：平均查询时间

t：即为返回前面多少条的数据

g：后边搭配一个正则匹配模式，大小写不敏感的

**常用参数手册**

得到返回记录集最多的10个SQL

mysqldumpslow -s r -t 10 /var/lib/mysql/Heygo-slow.log

得到访问次数最多的10个SQL

mysqldumpslow -s c- t 10/var/lib/mysql/Heygo-slow.log

得到按照时间排序的前10条里面含有左连接的查询语句

mysqldumpslow -s t -t 10 -g "left join" /var/lib/mysql/Heygo-slow.log

另外建议在使用这些命令时结合 | 和more使用，否则有可能出现爆屏情况

mysqldumpslow -s r -t 10 /var/lib/mysql/Heygo-slow.log | more

**批量数据脚本**

创建函数，假如报错：This function has none of DETERMINISTIC………

由于开启过慢查询日志，因为我们开启了bin-log，我们就必须为我们的function指定一个参数。

log\_bin\_trust\_function\_creators = OFF ，默认必须为 function 传递一个参数

mysql> show variables like 'log\_bin\_trust\_function\_creators';
+---------------------------------+-------+
| Variable\_name | Value |
+---------------------------------+-------+
| log\_bin\_trust\_function\_creators | OFF |
+---------------------------------+-------+
1 row in set (0.00 sec)

通过 set global log\_bin\_trust\_function\_creators=1;我们可以不用为 function 传参

mysql> set global log\_bin\_trust\_function\_creators=1;
Query OK, 0 rows affected (0.00 sec)
mysql> show variables like 'log\_bin\_trust\_function\_creators';
+---------------------------------+-------+
| Variable\_name | Value |
+---------------------------------+-------+
| log\_bin\_trust\_function\_creators | ON |
+---------------------------------+-------+
1 row in set (0.00 sec)

这样添加了参数以后，如果mysqld重启，上述参数又会消失，永久方法在配置文件中修改：

windows下：my.ini --> [mysqld] 节点下加上 log\_bin\_trust\_function\_creators=1

linux下：/etc/my.cnf --> [mysqld] 节点下加上 log\_bin\_trust\_function\_creators=1

**此处未学习，创建函数与存储过程等等。。。。。。**

**Show Profile**

**Show Profile 是什么**

是mysql提供可以用来分析当前会话中语句执行的资源消耗情况。可以用于SQL的调优测量。

官网：http://dev.mysql.com/doc/refman/5.5/en/show-profile.html。

默认情况下，参数处于关闭状态，并保存最近15次的运行结果。

**分析步骤**

1. 查看是当前的SQL版本是否支持Show Profile

show variables like ‘profiling%’; 查看 Show Profile 是否开启

mysql> show variables like 'profiling%';
+------------------------+-------+
| Variable\_name | Value |
+------------------------+-------+
| profiling | OFF |
| profiling\_history\_size | 15 |
+------------------------+-------+
2 rows in set (0.01 sec)

2. 开启功能 Show Profile ，默认是关闭，使用前需要开启

set profiling=on; 开启 Show Profile

mysql> set profiling=on;
Query OK, 0 rows affected, 1 warning (0.00 sec)
mysql> show variables like 'profiling%';
+------------------------+-------+
| Variable\_name | Value |
+------------------------+-------+
| profiling | ON |
| profiling\_history\_size | 15 |
+------------------------+-------+
2 rows in set (0.00 sec)

3. 运行SQL

正常 SQL

select \* from tbl\_emp;
select \* from tbl\_emp e inner join tbl\_dept d on e.deptId = d.id;
select \* from tbl\_emp e left join tbl\_dept d on e.deptId = d.id;

慢 SQL

select \* from emp group by id%10 limit 150000;
select \* from emp group by id%10 limit 150000;
select \* from emp group by id%10 order by 5;

4. 查看结果

通过 show profiles; 指令查看结果

mysql> show profiles;
+----------+------------+----------------------------------------------------------------------+
| Query\_ID | Duration | Query |
+----------+------------+----------------------------------------------------------------------+
| 1 | 0.00052700 | show variables like 'profiling%' |
| 2 | 0.00030300 | select \* from tbl\_emp |
| 3 | 0.00010650 | select \* from tbl\_emp e inner join tbl\_dept d on e.'deptId' = d.'id' |
| 4 | 0.00031625 | select \* from tbl\_emp e inner join tbl\_dept d on e.deptId = d.id |
| 5 | 0.00042100 | select \* from tbl\_emp e left join tbl\_dept d on e.deptId = d.id |
| 6 | 0.38621875 | select \* from emp group by id%20 limit 150000 |
| 7 | 0.00014900 | select \* from emp group by id%20 order by 150000 |
| 8 | 0.38649000 | select \* from emp group by id%20 order by 5 |
| 9 | 0.06782700 | select COUNT(\*) from emp |
| 10 | 0.35434400 | select \* from emp group by id%10 limit 150000 |
+----------+------------+----------------------------------------------------------------------+
10 rows in set, 1 warning (0.00 sec)

5. 诊断SQL

show profile cpu, block io for query SQL编号; 查看 SQL 语句执行的具体流程以及每个步骤花费的时间。

mysql> show profile cpu, block io for query 2;
+----------------------+----------+----------+------------+--------------+---------------+
| Status | Duration | CPU\_user | CPU\_system | Block\_ops\_in | Block\_ops\_out |
+----------------------+----------+----------+------------+--------------+---------------+
| starting | 0.000055 | 0.000000 | 0.000000 | 0 | 0 |
| checking permissions | 0.000007 | 0.000000 | 0.000000 | 0 | 0 |
| Opening tables | 0.000011 | 0.000000 | 0.000000 | 0 | 0 |
| init | 0.000024 | 0.000000 | 0.000000 | 0 | 0 |
| System lock | 0.000046 | 0.000000 | 0.000000 | 0 | 0 |
| optimizing | 0.000018 | 0.000000 | 0.000000 | 0 | 0 |
| statistics | 0.000008 | 0.000000 | 0.000000 | 0 | 0 |
| preparing | 0.000019 | 0.000000 | 0.000000 | 0 | 0 |
| executing | 0.000003 | 0.000000 | 0.000000 | 0 | 0 |
| Sending data | 0.000089 | 0.000000 | 0.000000 | 0 | 0 |
| end | 0.000004 | 0.000000 | 0.000000 | 0 | 0 |
| query end | 0.000003 | 0.000000 | 0.000000 | 0 | 0 |
| closing tables | 0.000005 | 0.000000 | 0.000000 | 0 | 0 |
| freeing items | 0.000006 | 0.000000 | 0.000000 | 0 | 0 |
| cleaning up | 0.000006 | 0.000000 | 0.000000 | 0 | 0 |
+----------------------+----------+----------+------------+--------------+---------------+
15 rows in set, 1 warning (0.00 sec)

参数备注：

ALL：显示所有的开销信息

BLOCK IO：显示块IO相关开销

CONTEXT SWITCHES：上下文切换相关开销

CPU：显示CPU相关开销信息

IPC：显示发送和接收相关开销信息

MEMORY：显示内存相关开销信息

PAGE FAULTS：显示页面错误相关开销信息

SOURCE：显示和Source\_function，Source\_file，Source\_line相关的开销信息

SWAPS：显示交换次数相关开销的信息

**日常开发需要注意的结论**

converting HEAP to MyISAM：查询结果太大，内存都不够用了往磁盘上搬了。

Creating tmp table：创建临时表，mysql 先将拷贝数据到临时表，然后用完再将临时表删除

Copying to tmp table on disk：把内存中临时表复制到磁盘，危险！！！

locked：锁表

**全局查询日志**

**永远不要在生产环境开启这个功能。（**该功能用于保存所有的sql执行记录**）**

**配置启用全局查询日志**

在mysql的my.cnf中，设置如下：

# 开启
general\_log=1
# 记录日志文件的路径
general\_log\_file=/path/logfile
# 输出格式
log\_output=FILE

**编码启用全局查询日志（重启mysql失效）**

执行如下指令开启全局查询日志

set global general\_log=1;
set global log\_output='TABLE';

此后，你所执行的sql语句，将会记录到mysql库里的general\_log表，可以用下面的命令查看

select \* from mysql.general\_log;

mysql> select \* from mysql.general\_log;
+---------------------+---------------------------+-----------+-----------+--------------+-----------------------------------------------+
| event\_time | user\_host | thread\_id | server\_id | command\_type | argument |
+---------------------+---------------------------+-----------+-----------+--------------+-----------------------------------------------+
| 2020-08-05 14:41:07 | root[root] @ localhost [] | 14 | 0 | Query | select \* from emp group by id%10 limit 150000 |
| 2020-08-05 14:41:12 | root[root] @ localhost [] | 14 | 0 | Query | select COUNT(\*) from emp |
| 2020-08-05 14:41:30 | root[root] @ localhost [] | 14 | 0 | Query | select \* from mysql.general\_log |
+---------------------+---------------------------+-----------+-----------+--------------+-----------------------------------------------+
3 rows in set (0.00 sec)