MySQL 性能优化技巧

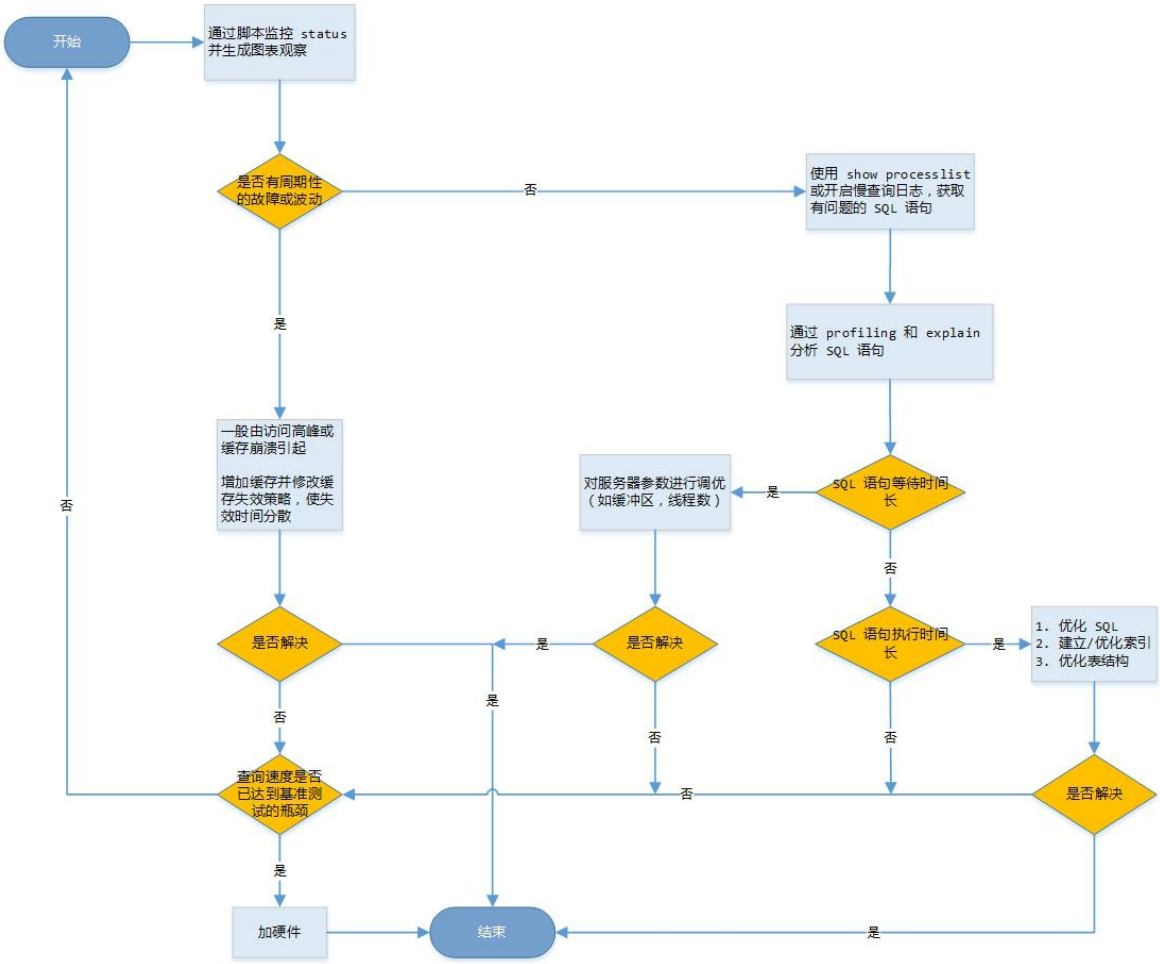
2017-10-07 | 数据库 | 1226

一、背景

最近公司项目添加新功能，上线后发现有些功能的列表查询时间很久。原因是新功能用到旧功能的接口，而这些旧接口的 SQL 查询语句关联5,6张表且编写不够规范，导致 MySQL 在执行 SQL 语句时索引失效，进行全表扫描。原本负责优化的同事有事请假回家，因此优化查询数据的问题落在笔者手中。笔者在查阅网上 SQL 优化的资料后成功解决了问题，在此从**全局角度**记录和总结 MySQL 查询优化相关技巧。

二、优化思路

数据查询慢，不代表 SQL 语句写法有问题。 首先，我们需要找到问题的源头才能"对症下药"。笔者用一张流程图展示 MySQL 优化的思路：



(http://images.extlight.com/mysql-optimized.jpg)

无需更多言语，从图中可以清楚地看出，导致数据查询慢的原因有多种，如：缓存失效，在此一段时间内由于高并发访问导致 MySQL 服务器崩溃；SQL 语句编写问题；MySQL 服务器参数问题；硬件配置限制 MySQL 服务性能问题等。

三、查看 MySQL 服务器运行的状态值

如果系统的并发请求数不高，且查询速度慢，可以忽略该步骤直接进行 SQL 语句调优步骤。

执行命令：

```
show status
```

由于返回结果太多，此处不贴出结果。其中，再返回的结果中，我们主要关注 “Queries”、“Threads_connected” 和 “Threads_running” 的值，即查询次数、线程连接数和线程运行数。

我们可以通过执行如下脚本监控 MySQL 服务器运行的状态值

```
#!/bin/bash
while true
do
mysqladmin -uroot -p"密码" ext | awk '/Queries/{q=$4}/Threads_connected/{c=$4}/Threads_running/{r=$4}END{printf("%d %d %d
sleep 1
done
```

执行该脚本 24 小时，获取 status.txt 里的内容，再次通过 awk 计算**每秒请求 MySQL 服务的次数**

```
awk '{q=$1-last;last=$1}{printf("%d %d %d\n",q,$2,$3)}' status.txt
```

复制计算好的内容到 Excel 中生成图表观察数据周期性。

如果观察的数据有周期性的变化，如上图的解释，需要修改缓存失效策略。

例如：

通过随机数在[3,6,9] 区间获取其中一个值作为缓存失效时间，这样分散了缓存失效时间，从而节省了一部分内存的消耗。

当访问高峰期时，一部分请求分流到未失效的缓存，另一部分则访问 MySQL 数据库，这样减少了 MySQL 服务器的压力。

四、获取需要优化的 SQL 语句

4.1 方式一：查看运行的线程

执行命令：

```
show processlist
```

返回结果：

```
mysql> show processlist;
+----+------+-----------+------+---------+------+----------+------------------+
| Id | User | Host      | db   | Command | Time | State    | Info             |
+----+------+-----------+------+---------+------+----------+------------------+
|  9 | root | localhost | test | Query   |    0 | starting | show processlist |
+----+------+-----------+------+---------+------+----------+------------------+
1 row in set (0.00 sec)
```

从返回结果中我们可以了解该线程执行了什么命令/SQL 语句以及执行的时间。实际应用中，查询的返回结果会有 N 条记录。

其中，**返回的 State 的值是我们判断性能好坏的关键**，其值出现如下内容，则该行记录的 SQL 语句需要优化：

```
Converting HEAP to MyISAM # 查询结果太大时，把结果放到磁盘，严重
Create tmp table #创建临时表，严重
Copying to tmp table on disk  #把内存临时表复制到磁盘，严重
locked #被其他查询锁住，严重
loggin slow query #记录慢查询
Sorting result #排序
```

State 字段有很多值，如需了解更多，可以参看文章末尾提供的链接。

4.2 方式二：开启慢查询日志

在配置文件 my.cnf 中的 [mysqld] 一行下边添加两个参数：

```
slow_query_log = 1
slow_query_log_file=/var/lib/mysql/slow-query.log
long_query_time = 2

log_queries_not_using_indexes = 1
```

其中，slow_query_log = 1 表示开启慢查询；

slow_query_log_file 表示慢查询日志存放的位置；

long_query_time = 2 表示查询 >=2 秒才记录日志；

log_queries_not_using_indexes = 1 记录没有使用索引的 SQL 语句。

注意：slow_query_log_file 的路径不能随便写，否则 MySQL 服务器可能没有权限将日志文件写到指定的目录中。建议直接复制上文的路径。

修改保存文件后，重启 MySQL 服务。在 /var/lib/mysql/ 目录下会创建 slow-query.log 日志文件。连接 MySQL 服务端执行如下命令可以查看配置情况。

```
show variables like 'slow_query%';

show variables like 'long_query_time';
```

测试慢查询日志：

```
mysql> select sleep(2);
+----------+
| sleep(2) |
+----------+
|        0 |
+----------+
1 row in set (2.00 sec)
```

打开慢查询日志文件

```
[root@localhost mysql]# vim /var/lib/mysql/slow-query.log
/usr/sbin/mysqld, Version: 5.7.19-log (MySQL Community Server (GPL)). started with:
Tcp port: 0  Unix socket: /var/lib/mysql/mysql.sock
Time                 Id Command    Argument
# Time: 2017-10-05T04:39:11.408964Z
# User@Host: root[root] @ localhost []  Id:     3
# Query_time: 2.001395  Lock_time: 0.000000 Rows_sent: 1  Rows_examined: 0
use test;
SET timestamp=1507178351;
select sleep(2);
```

我们可以看到刚才执行了 2 秒的 SQL 语句被记录下来了。

虽然在慢查询日志中记录查询慢的 SQL 信息，但是日志记录的内容密集且不易查阅。因此，我们需要通过工具将 SQL 筛选出来。

MySQL 提供 mysqldumpslow 工具对日志进行分析。我们可以使用 mysqldumpslow --help 查看命令相关用法。

常用参数如下：

```
  -s：排序方式，后边接着如下参数
    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：平均返回记录书
  at：平均查询时间
  -t：返回前面多少条的数据
  -g：翻遍搭配一个正则表达式，大小写不敏感
```

案例：

```
获取返回记录集最多的10个sql
mysqldumpslow -s r -t 10 /var/lib/mysql/slow-query.log

获取访问次数最多的10个sql
mysqldumpslow -s c -t 10 /var/lib/mysql/slow-query.log

获取按照时间排序的前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/slow-query.log
```

五、分析 SQL 语句

 # 5.1 方式一：explain

筛选出有问题的 SQL，我们可以使用 MySQL 提供的 explain 查看 SQL 执行计划情况（关联表，表查询顺序、索引使用情况等）。

用法：

```
explain select * from category;
```

返回结果：

```
mysql> explain select * from category;
+----+-------------+----------+------------+------+---------------+------+---------+------+------+----------+-------+
| id | select_type | table    | partitions | type | possible_keys | key  | key_len | ref  | rows | filtered | Extra |
+----+-------------+----------+------------+------+---------------+------+---------+------+------+----------+-------+
|  1 | SIMPLE      | category | NULL       | ALL  | NULL          | NULL | NULL    | NULL |    1 |   100.00 | NULL  |
+----+-------------+----------+------------+------+---------------+------+---------+------+------+----------+-------+
1 row in set, 1 warning (0.00 sec)
```

字段解释：

1. id：select 查询序列号。id相同，执行顺序由上至下；id不同，id值越大优先级越高，越先被执行
2. select_type：查询数据的操作类型，其值如下：

```
simple：简单查询，不包含子查询或 union
primary:包含复杂的子查询，最外层查询标记为该值
subquery：在 select 或 where 包含子查询，被标记为该值
derived：在 from 列表中包含的子查询被标记为该值，MySQL 会递归执行这些子查询，把结果放在临时表
union：若第二个 select 出现在 union 之后，则被标记为该值。若 union 包含在 from 的子查询中，外层 select 被标记为 derived
union result：从 union 表获取结果的 select
```

3. table：显示该行数据是关于哪张表
4. partitions：匹配的分区
5. type：表的连接类型，其值，性能由高到底排列如下：

```
system：表只有一行记录，相当于系统表
const：通过索引一次就找到，只匹配一行数据
eq_ref：唯一性索引扫描，对于每个索引键，表中只有一条记录与之匹配。常用于主键或唯一索引扫描
ref：非唯一性索引扫描，返回匹配某个单独值的所有行。用于=、< 或 > 操作符带索引的列
range：只检索给定范围的行，使用一个索引来选择行。一般使用between、>、<情况
index：只遍历索引树
ALL：全表扫描，性能最差
```

注：前5种情况都是理想情况的索引使用情况。通常优化至少到range级别，最好能优化到 ref

6. possible_keys：指出 MySQL 使用哪个索引在该表找到行记录。如果该值为 NULL，说明没有使用索引，可以建立索引提高性能
7. key：显示 MySQL 实际使用的索引。如果为 NULL，则没有使用索引查询
8. key_len：表示索引中使用的字节数，通过该列计算查询中使用的索引的长度。在不损失精确性的情况下，长度越短越好 显示的是索引字段的最大长度，并非实际使用长度
9. ref：显示该表的索引字段关联了哪张表的哪个字段
10. rows：根据表统计信息及选用情况，大致估算出找到所需的记录或所需读取的行数，数值越小越好
11. filtered：返回结果的行数占读取行数的百分比，值越大越好
12. extra： 包含不合适在其他列中显示但十分重要的额外信息，常见的值如下：

```
using filesort：说明 MySQL 会对数据使用一个外部的索引排序，而不是按照表内的索引顺序进行读取。出现该值，应该优化 SQL
using temporary：使用了临时表保存中间结果，MySQL 在对查询结果排序时使用临时表。常见于排序 order by 和分组查询 group by。
using index：表示相应的 select 操作使用了覆盖索引，避免了访问表的数据行，效率不错
using where：where 子句用于限制哪一行
using join buffer：使用连接缓存
distinct：发现第一个匹配后，停止为当前的行组合搜索更多的行
```

注意：出现前 2 个值，SQL 语句必须要优化。

 # 5.2 方式二：profiling

使用 profiling 命令可以了解 SQL 语句消耗资源的详细信息（每个执行步骤的开销）。

 # 5.2.1 查看 profile 开启情况

```
select @@profiling;
```

返回结果：

```
mysql> select @@profiling;
+-------------+
| @@profiling |
+-------------+
|           0 |
+-------------+
1 row in set, 1 warning (0.00 sec)
```

0 表示关闭状态,1 表示开启

 # 5.2.2 启用 profile

```
set profiling = 1;
```

返回结果：

```
mysql> set profiling = 1;
Query OK, 0 rows affected, 1 warning (0.00 sec)

mysql> select @@profiling;
+-------------+
| @@profiling |
+-------------+
|           1 |
+-------------+
1 row in set, 1 warning (0.00 sec)
```

在连接关闭后，profiling 状态自动设置为关闭状态。

 # 5.2.3 查看执行的 SQL 列表

```
show profiles;
```

返回结果：

```
mysql> show profiles;
+----------+------------+------------------------------+
| Query_ID | Duration   | Query                        |
+----------+------------+------------------------------+
|        1 | 0.00062925 | select @@profiling           |
|        2 | 0.00094150 | show tables                  |
|        3 | 0.00119125 | show databases               |
|        4 | 0.00029750 | SELECT DATABASE()            |
|        5 | 0.00025975 | show databases               |
|        6 | 0.00023050 | show tables                  |
|        7 | 0.00042000 | show tables                  |
|        8 | 0.00260675 | desc role                    |
|        9 | 0.00074900 | select name,is_key from role |
+----------+------------+------------------------------+
9 rows in set, 1 warning (0.00 sec)
```

该命令执行之前，需要执行其他 SQL 语句才有记录。

 # 5.2.4 查询指定 ID 的执行详细信息

```
show profile for query Query_ID;
```

返回结果：

```
mysql> show profile for query 9;
+----------------------+----------+
| Status               | Duration |
+----------------------+----------+
| starting             | 0.000207 |
| checking permissions | 0.000010 |
| Opening tables       | 0.000042 |
| init                 | 0.000050 |
| System lock          | 0.000012 |
| optimizing           | 0.000003 |
| statistics           | 0.000011 |
| preparing            | 0.000011 |
| executing            | 0.000002 |
| Sending data         | 0.000362 |
| end                  | 0.000006 |
| query end            | 0.000006 |
| closing tables       | 0.000006 |
| freeing items        | 0.000011 |
| cleaning up          | 0.000013 |
+----------------------+----------+
15 rows in set, 1 warning (0.00 sec)
```

每行都是状态变化的过程以及它们持续的时间。Status 这一列和 show processlist 的 State 是一致的。因此，需要优化的注意点与上文描述的一样。

其中，Status 字段的值同样可以参考末尾链接。

5.2.5 获取 CPU、 Block IO 等信息

```
show profile block io,cpu for query Query_ID;

show profile cpu,block io,memory,swaps,context switches,source for query Query_ID;

show profile all for query Query_ID;
```

六、优化手段

主要以查询优化、索引使用和表结构设计方面进行讲解。

6.1 查询优化

1. 避免 SELECT *，需要什么数据，就查询对应的字段。
2. 小表驱动大表，即小的数据集驱动大的数据集。如：以 A，B 两表为例，两表通过 id 字段进行关联。

```
当 B 表的数据集小于 A 表时，用 in 优化 exist；使用 in ，两表执行顺序是先查 B 表，再查 A 表
select * from A where id in (select id from B)

当 A 表的数据集小于 B 表时，用 exist 优化 in；使用 exists，两表执行顺序是先查 A 表，再查 B 表
select * from A where exists (select 1 from B where B.id = A.id)
```

3. 一些情况下，可以使用连接代替子查询，因为使用 join，MySQL 不会在内存中创建临时表。
4. 适当添加冗余字段，减少表关联。
5. 合理使用索引（下文介绍）。如：为排序、分组字段建立索引，避免 filesort 的出现。

6.2 索引使用

6.2.1 适合使用索引的场景

1. 主键自动创建唯一索引
2. 频繁作为查询条件的字段
3. 查询中与其他表关联的字段
4. 查询中排序的字段
5. 查询中统计或分组字段

6.2.2 不适合使用索引的场景

1. 频繁更新的字段

2. where 条件中用不到的字段
3. 表记录太少
4. 经常增删改的表
5. 字段的值的差异性不大或重复性高

 # 6.2.3 索引创建和使用原则

1. 单表查询：哪个列作查询条件，就在该列创建索引
2. 多表查询：left join 时，索引添加到右表关联字段；right join 时，索引添加到左表关联字段
3. 不要对索引列进行任何操作（计算、函数、类型转换）
4. 索引列中不要使用 !=，<> 非等于
5. 索引列不要为空，且不要使用 is null 或 is not null 判断
6. 索引字段是字符串类型，查询条件的值要加''单引号,避免底层类型自动转换

违背上述原则可能会导致索引失效，具体情况需要使用 explain 命令进行查看

 # 6.2.4 索引失效情况

除了违背索引创建和使用原则外，如下情况也会导致索引失效：

1. 模糊查询时，以 % 开头
2. 使用 or 时，如：字段1（非索引）or 字段2（索引）会导致索引失效。
3. 使用复合索引时，不使用第一个索引列。

index(a,b,c) ，以字段 a,b,c 作为复合索引为例：

语句	索引是否生效
where a = 1	是，字段 a 索引生效
where a = 1 and b = 2	是，字段 a 和 b 索引生效
where a = 1 and b = 2 and c = 3	是，全部生效
where b = 2 或 where c = 3	否
where a = 1 and c = 3	字段 a 生效，字段 c 失效
where a = 1 and b > 2 and c = 3	字段 a，b 生效，字段 c 失效
where a = 1 and b like 'xxx%' and c = 3	字段 a，b 生效，字段 c 失效

 # 6.3 数据库表结构设计

 # 6.3.1 选择合适的数据类型

1. 使用可以存下数据最小的数据类型
2. 使用简单的数据类型。int 要比 varchar 类型在mysql处理简单
3. 尽量使用 tinyint、smallint、mediumint 作为整数类型而非 int
4. 尽可能使用 not null 定义字段，因为 null 占用4字节空间
5. 尽量少用 text 类型,非用不可时最好考虑分表
6. 尽量使用 timestamp 而非 datetime
7. 单表不要有太多字段，建议在 20 以内

 # 6.3.2 表的拆分

当数据库中的数据非常大时，查询优化方案也不能解决查询速度慢的问题时，我们可以考虑拆分表，让每张表的数据量变小，从而提高查询效率。

1. 垂直拆分：将表中多个列分开放到不同的表中。例如用户表中一些字段经常被访问，将这些字段放在一张表中，另外一些不常用的字段放在另一张表中。 插入数据时，使用事务确保两张表的数据一致性。
2. 水平拆分：按照行进行拆分。例如用户表中，使用用户ID，对用户ID取10的余数，将用户数据均匀的分配到0~9的10个用户表中。查找时也按照这个规则查询数据。

 # 6.3.3 读写分离

一般情况下对数据库而言都是“读多写少”。换言之，数据库的压力多数是因为大量的读取数据的操作造成的。我们可以采用数据库集群的方案，使用一个库作为主库，负责写入数据；其他库为从库，负责读取数据。这样可以缓解对数据库的访问压力。

七、服务器参数调优

 # 7.1 内存相关

sort_buffer_size 排序缓冲区内存大小

join_buffer_size 使用连接缓冲区大小

read_buffer_size 全表扫描时分配的缓冲区大小

7.2 IO 相关

Innodb_log_file_size 事务日志大小

Innodb_log_files_in_group 事务日志个数

Innodb_log_buffer_size 事务日志缓冲区大小

Innodb_flush_log_at_trx_commit 事务日志刷新策略，其值如下：

0：每秒进行一次 log 写入 cache，并 flush log 到磁盘

1：在每次事务提交执行 log 写入 cache，并 flush log 到磁盘

2：每次事务提交，执行 log 数据写到 cache，每秒执行一次 flush log 到磁盘

7.3 安全相关

expire_logs_days 指定自动清理 binlog 的天数

max_allowed_packet 控制 MySQL 可以接收的包的大小

skip_name_resolve 禁用 DNS 查找

read_only 禁止非 super 权限用户写权限

skip_slave_start 级你用 slave 自动恢复

7.4 其他

max_connections 控制允许的最大连接数

tmp_table_size 临时表大小

max_heap_table_size 最大内存表大小

笔者并没有使用这些参数对 MySQL 服务器进行调优，具体详情介绍和性能效果请参考文章末尾的资料或另行百度。

八、硬件选购和参数优化

硬件的性能直接决定 MySQL 数据库的性能瓶颈，直接决定 MySQL 数据库的运行数据和效率。

作为软件开发程序员，我们主要关注软件方面的优化内容，以下硬件方面的优化作为了解即可

8.1 内存相关

内存的 IO 比硬盘的速度快很多，可以增加系统的缓冲区容量，使数据在内存停留的时间更长，以减少磁盘的 IO

8.2 磁盘 I/O 相关

1. 使用 SSD 或 PCIe SSD 设备，至少获得数百倍甚至万倍的 IOPS 提升
2. 购置阵列卡同时配备 CACHE 及 BBU 模块，可以明显提升 IOPS
3. 尽可能选用 RAID-10，而非 RAID-5

8.3 配置 CUP 相关

在服务器的 BIOS 设置中，调整如下配置：

1. 选择 Performance Per Watt Optimized（DAPC）模式，发挥 CPU 最大性能
2. 关闭 C1E 和 C States 等选项，提升 CPU 效率
3. Memory Frequency（内存频率）选择 Maximum Performance

九、参考资料

- **https://dev.mysql.com/doc/refman/5.7/en/show-status.html (https://dev.mysql.com/doc/refman/5.7/en/show-status.html)** show status 语法
- **https://dev.mysql.com/doc/refman/5.7/en/show-processlist.html (https://dev.mysql.com/doc/refman/5.7/en/show-processlist.html)**
  show processlist 语法
- **https://dev.mysql.com/doc/refman/5.7/en/general-thread-states.html (https://dev.mysql.com/doc/refman/5.7/en/general-thread-states.html)**
  线程状态
- **https://dev.mysql.com/doc/refman/5.7/en/explain-output.html (https://dev.mysql.com/doc/refman/5.7/en/explain-output.html)** explain 语法
- **https://dev.mysql.com/doc/refman/5.7/en/show-profile.html (https://dev.mysql.com/doc/refman/5.7/en/show-profile.html)** show profile 语法

- **http://blog.csdn.net/nightelve/article/details/17393631 (http://blog.csdn.net/nightelve/article/details/17393631)** MySQL 服务器参数调优
- **http://blog.csdn.net/qq_22929803/article/details/51237056 (http://blog.csdn.net/qq_22929803/article/details/51237056)** MySQL 服务器参数调优
- **http://blog.chinaunix.net/uid-11640640-id-3426908.html (http://blog.chinaunix.net/uid-11640640-id-3426908.html)**
- **https://segmentfault.com/a/1190000006158186 (https://segmentfault.com/a/1190000006158186)**
- **http://blog.csdn.net/gzh0222/article/details/7976127 (http://blog.csdn.net/gzh0222/article/details/7976127)**

上一篇：《Centos 7.2 安装和卸载 MySQL 5.7》 (/2017/10/02/Centos-7.2-安装和卸载-MySQL-5.7/)
下一篇：《原生 Javascript 编写俄罗斯方块》 (/2017/10/08/原生-Javascript-编写俄罗斯方块/)

评论