五、性能分析和性能优化篇

内容： 性能分析和性能优化

目标： 背诵性能分析步骤、 能够配置、背诵出性能优化的思路、做sql优化

性能分析的思路

1. 首先需要使用【慢查询日志】功能，去获取所有查询时间比较长的SQL语句
2. 其次【查看执行计划】查看有问题的SQL的执行计划 explain
3. 最后可以使用【show profile[s]】 查看有问题的SQL的性能使用情况

慢查询日志

慢查询日志介绍

*数据库查询快慢*是影响项目性能的一大因素，对于数据库，我们除了要优化 SQL，更重要的是得**先找到需要优化的SQL。**MySQL数据库有一个**“慢查询日志”**功能，用来**记录**查询时间**超过某个设定值的SQL语句**，这将极大程度帮助我们**快速定位到症结所在**，以便对症下药。

至于查询时间的多少才算慢，每个项目、业务都有不同的要求。

> 比如说传统企业的软件允许查询时间高于某个值，但是把这个标准放在互联网项目或者访问量大的网站上，估计就是一个bug，甚至可能升级为一个功能性缺陷。

MySQL的慢查询日志功能**默认是关闭的，需要手动开启。**

- **永久开启慢查询功能**

修改/etc/my.cnf配置文件，重启 MySQL, 这种永久生效.

```
[mysqld]
slow_query_log = 1 #开启慢查询日志
slow_query_log_file = /var/lib/mysql/slow.log
long_query_time = 1 # 大于等于该时间
```

分析慢查询日志的工具

使用mysqldumpslow工具

> mysqldumpslow是MySQL自带的慢查询日志工具。
>
> 可以使用mysqldumpslow工具搜索慢查询日志中的SQL语句。

得到按照时间排序的前10条里面含有左连接的查询语句：

```
[root@localhost mysql]# mysqldumpslow -s t -t 10 -g “left join” /var/log/mysql/slow.log
```

常用参数说明：

-s：是表示按照何种方式排序

c：访问计数

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：是top n的意思，即为返回前面多少条的数据

-g：后边可以写一个正则匹配模式，大小写不敏感的

使用percona-toolkit工具

percona-toolkit是一组高级命令行工具的集合，可以查看当前服务的摘要信息，磁盘检测，分析慢查询日志，查找重复索引，实现表同步等等。

- 下载

https://www.percona.com/downloads/percona-toolkit/3.0.11/binary/tarball/percona-toolkit-3.0.11_x86_64.tar.gz

```
wget https://www.percona.com/downloads/percona-toolkit/3.0.11/binary/tarball/percona-toolkit-3.0.11_x86_64.tar.gz
```

- 安装

```
tar -xf percona-toolkit-3.0.11_x86_64.tar.gz
cd percona-toolkit-3.0.11
perl Makefile.PL
make
make install
```

- 调错

  Can't locate ExtUtils/MakeMaker.pm in @INC 错误的解决方式:

  先执行再安装

```
yum install -y perl-ExtUtils-CBuilder perl-ExtUtils-MakeMaker
```

Can't locate Time/HiRes.pm in @INC

```
yum install -y perl-Time-HiRes
```

Can't locate Digest/MD5.pm in @INC

```
yum install perl-Digest-MD5.x86_64
```

- 使用pt-query-digest查看慢查询日志

```
pt-query-digest /var/lib/mysql/localhost-slow.log
```

用法示例

1.直接分析慢查询文件:

```
pt-query-digest  slow.log > slow_report.log
```

2.分析最近12小时内的查询:

```
pt-query-digest  --since=12h  slow.log > slow_report2.log
```

3.分析指定时间范围内的查询:

```
pt-query-digest slow.log --since '2017-01-07 09:30:00' --until '2017-01-07 10:00:00'> >
slow_report3.log
```

4.分析指含有select语句的慢查询

```
pt-query-digest --filter '$event->{fingerprint} =~ m/^select/i' slow.log> slow_report4.log
```

5.针对某个用户的慢查询

```
pt-query-digest --filter '($event->{user} || "") =~ m/^root/i' slow.log> slow_report5.log
```

6.查询所有所有的全表扫描或full join的慢查询

```
pt-query-digest --filter '(($event->{Full_scan} || "") eq "yes") ||(($event->{Full_join} ||
"") eq "yes")' slow.log> slow_report6.log
```

7.把查询保存到query_review表

```
pt-query-digest --user=root -password=abc123 --review  h=localhost,D=test,t=query_review--
create-review-table  slow.log
```

8.把查询保存到query_history表

```
pt-query-digest  --user=root -password=abc123 --review  h=localhost,D=test,t=query_history--
create-review-table  slow.log_0001
pt-query-digest  --user=root -password=abc123 --review  h=localhost,D=test,t=query_history--
create-review-table  slow.log_0002
```

9.通过tcpdump抓取mysql的tcp协议数据，然后再分析

```
tcpdump -s 65535 -x -nn -q -tttt -i any -c 1000 port 3306 > mysql.tcp.txt
pt-query-digest --type tcpdump mysql.tcp.txt> slow_report9.log
```

profile分析语句

介绍

Query Profiler是MySQL自带的一种**query诊断分析工具**，通过它可以分析出一条SQL语句的**硬件性能瓶颈**在什么地方。

通常我们是使用的explain,以及slow query log都无法做到精确分析，但是Query Profiler却可以定位出一条SQL语句执行的各种资源消耗情况，比如CPU，IO等，以及该SQL执行所耗费的时间等。不过该工具只有在MySQL 5.0.37以及以上版本中才有实现。

默认的情况下，MYSQL的该功能没有打开，需要自己手动启动。

语句使用

- **show profile** 和 **show profiles** 语句可以展示**当前会话**(退出session后,profiling重置为0) 中执行语句的资源使用情况.
- **show profiles** :以列表形式显示最近发送到服务器上执行的语句的**资源使用情况**.显示的记录数由变量:*profiling_history_size* 控制,默认15条

```
mysql> show profiles;
+----------+------------+------------------------------+
| Query_ID | Duration   | Query                        |
+----------+------------+------------------------------+
|        1 | 0.00023200 | select count(*) from sys_user |
```

- **show profile**: 展示最近一条语句执行的详细资源占用信息,默认显示 **Status和Duratio**n两列

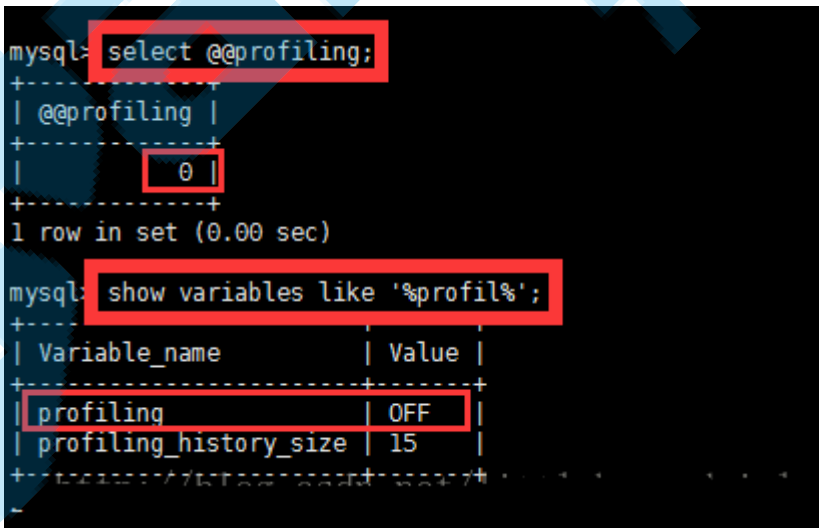
```
mysql> show profile;
+--------------------+----------+
| Status             | Duration |
+--------------------+----------+
| starting           | 0.000068 |
| Opening tables     | 0.000031 |
| System lock        | 0.000004 |
| Table lock         | 0.000007 |
| init               | 0.000014 |
| optimizing         | 0.000006 |
| statistics         | 0.000010 |
| preparing          | 0.000008 |
| executing          | 0.000005 |
| Sending data       | 0.000058 |
| end                | 0.000004 |
| query end          | 0.000002 |
| freeing items      | 0.000012 |
| logging slow query | 0.000001 |
| cleaning up        | 0.000002 |
+--------------------+----------+
15 rows in set (0.00 sec)
```

开启Profile功能

- Profile 功能由MySQL会话变量 : **profiling**控制,默认是**OFF**关闭状态。
- 查看是否开启了Profile功能:

```
select @@profiling;

show variables like ‘%profil%’;
```



- 开启profile功能

```
set profiling=1; --1是开启、0是关闭
```

服务器层面优化

将数据保存在内存中，保证从内存读取数据

- 设置足够大的`innodb_buffer_pool_size`，将数据读取到内存中。

```
建议innodb_buffer_pool_size设置为总内存大小的3/4或者4/5.
默认128M
```

内存预热

将磁盘数据在MySQL Server启动的时候，读取到内存中。

降低磁盘写入次数

- 对于生产环境来说，很多日志是不需要开启的，比如：**通用查询日志、慢查询日志、错误日志**
- 使用足够大的写入缓存 **innodb_log_file_size**

```
推荐 innodb_log_file_size 设置为 0.25 * innodb_buffer_pool_size
```

- 设置合适的**innodb_flush_log_at_trx_commit**，和日志落盘有关系。

SQL设计层面优化

面对人群：

```
懂技术并且了解需求的程序员。
```

具体优化方案如下：

- **设计中间表**，一般针对于**统计分析**功能，或者实时性不高的需求（OLTP、OLAP）
- 为减少关联查询，创建合理的**冗余字段**（考虑数据库的三范式和查询性能的取舍，创建冗余字段还需要注意**数据一致性问题**）
- 对于字段太多的大表，考虑**拆表**（比如一个表有100多个字段） 人和身份证
- 对于表中经常不被使用的字段或者存储数据比较多的字段，考虑拆表（比如商品表中会存储商品介绍，此时可以将商品介绍字段单独拆解到另一个表中，使用商品ID关联）
- 每张表建议都要有一个主键（**主键索引**），而且主键类型最好是**int类型**，建议自增主键（**不考虑分布式系统的情况下**）。

SQL语句优化（开发人员）

索引优化

为搜索字段（**where中的条件**）、排序字段、select查询列，创建合适的索引

LIMIT优化

如果预计SELECT语句的查询结果是一条，最好使用 **LIMIT 1**，可以停止全表扫描。

其他优化

- 尽量不使用count(*)、尽量使用count（主键）其他优化
    - 尽量不使用count(*)、尽量使用count（主键）

建表语句（1千万条）

```
CREATE PROCEDURE test_insert()
BEGIN
DECLARE i INT DEFAULT 1;
WHILE i<=10000000
DO
insert into tuser2
VALUES(null,concat('zy',i),concat('zhaoyun',i),23,'1',1,'beijing');
SET i=i+1;
END WHILE ;
commit;
END;
```