02 快速学会分析SQL执行效率（上）

更新时间：2019-07-30 10:13:59

> 勤学如春起之苗，不见其增，日有所长。
>
> ——陶潜

从开篇词我们了解到，本专栏首先会一起讨论一下 SQL 优化，而优化 SQL 的前提是能定位到慢 SQL 并对其进行分析，因此在专栏的开始，会跟大家分享**如何定位慢查询和如何分析 SQl 执行效率**。在前面两节，会用一些简单的例子让大家学会这些分析技巧。

在工作中可能会遇到某个新功能在测试时需要很久才返回结果，这时就应该分析是不是慢查询导致的。如果确实有慢查询，又应该怎么去分析 SQL 执行效率呢？这一篇文章我们就来学习怎么找到慢查询和怎么分析 SQL 执行效率。

1 定位慢 SQL

当我们实际工作中，碰到某个功能或者某个接口需要很久才能返回结果，我们就应该去确定是不是慢查询导致的。定位慢 SQL 有如下两种解决方案：

- 查看慢查询日志确定已经执行完的慢查询
- show processlist 查看正在执行的慢查询

我们一起来了解下这两种方法的使用场景和使用技巧吧！

1.1 通过慢查询日志

如果需要定位到慢查询，一般的方法是通过慢查询日志来查询的，MySQL 的慢查询日志用来记录在 MySQL 中响应时间超过参数 long_query_time（单位秒，默认值 10）设置的值并且扫描记录数不小于 min_examined_row_limit（默认值0）的语句，**能够帮我们找到执行完的慢查询，方便我们对这些 SQL 进行优化**。

> 知识扩展：
>
> 默认情况下，慢查询日志中不会记录管理语句，可通过设置 log_slow_admin_statements = on 让管理语句中的慢查询也会记录到慢查询日志中。
>
> 默认情况下，也不会记录查询时间不超过 long_query_time 但是不使用索引的语句，可通过配置 log_queries_not_using_indexes = on 让不使用索引的 SQL 都被记录到慢查询日志中（即使查询时间没超过 long_query_time 配置的值）。

如果需要使用慢查询日志，一般分为四步：开启慢查询日志、设置慢查询阀值、确定慢查询日志路径、确定慢查询日志的文件名。下面我们来学习下:

首先开启慢查询日志，由参数 slow_query_log 决定是否开启，在 MySQL 命令行下输入下面的命令：

```
mysql> set global slow_query_log = on;

Query OK, 0 rows affected (0.00 sec)
```

默认环境下，慢查询日志是关闭的。

设置慢查询时间阀值

```
mysql> set global long_query_time = 1;

Query OK, 0 rows affected (0.00 sec)
```

> 知识扩展：
> **MySQL 中 long_query_time 的值如何确定呢？**
>
> 线上业务一般建议把 long_query_time 设置为 1 秒，如果某个业务的 MySQL 要求比较高的 QPS，可设置慢查询为 0.1 秒。发现慢查询及时优化或者提醒开发改写。
>
> 一般测试环境建议 long_query_time 设置的阀值比生产环境的小，比如生产环境是 1 秒，则测试环境建议配置成 0.5 秒。便于在测试环境及时发现一些效率低的 SQL。
>
> 甚至某些重要业务测试环境 long_query_time 可以设置为 0，以便记录所有语句。并留意慢查询日志的输出，上线前的功能测试完成后，分析慢查询日志每类语句的输出，重点关注 Rows_examined（语句执行期间从存储引擎读取的行数），提前优化。

确定慢查询日志路径

慢查询日志的路径默认是 MySQL 的数据目录

```
mysql> show global variables like "datadir";

+---------------+-----------------------+
| Variable_name | Value                 |
+---------------+-----------------------+
| datadir       | /data/mysql/data/3306/ |
+---------------+-----------------------+

1 row in set (0.00 sec)
```

确定慢查询日志的文件名

```
mysql> show global variables like "slow_query_log_file";

+---------------------+----------------+
| Variable_name       | Value          |
+---------------------+----------------+
| slow_query_log_file | mysql-slow.log |
+---------------------+----------------+

1 row in set (0.00 sec)
```

根据上面的查询结果，可以直接查看 /data/mysql/data/3306/mysql-slow.log 文件获取已经执行完的慢查询

```
[root@mysqltest ~]# tail -n5 /data/mysql/data/3306/mysql-slow.log

Time: 2019-05-21T09:15:06.255554+08:00

User@Host: root[root] @ localhost []  Id: 8591152

Query_time: 10.000260  Lock_time: 0.000000 Rows_sent: 1  Rows_examined: 0

SET timestamp=1558401306;
select sleep(10);
```

这里对上方的执行结果详细描述一下：

- tail -n5：只查看慢查询文件的最后5行
- Time：慢查询发生的时间
- User@Host：客户端用户和IP
- Query_time：查询时间
- Lock_time：等待表锁的时间
- Rows_sent：语句返回的行数
- Rows_examined：语句执行期间从存储引擎读取的行数

上面这种方式是用系统自带的慢查询日志查看的，如果觉得系统自带的慢查询日志不方便查看，小伙伴们可以使用 pt-query-digest 或者 mysqldumpslow 等工具对慢查询日志进行分析，由于本节重点是找到慢查询，这里就不一一示例了。

1.2 通过 show processlist;

有时慢查询正在执行，已经导致数据库负载偏高了，而由于慢查询还没执行完，因此慢查询日志还看不到任何语句。此时可以使用 show processlist 命令判断正在执行的慢查询。show processlist 显示哪些线程正在运行。如果有 PROCESS 权限，则可以看到所有线程。否则，只能看到当前会话的线程。

> 知识扩展：如果不使用 FULL 关键字，在 info 字段中只显示每个语句的前 100 个字符，如果想看语句的全部内容可以使用 full 修饰（show full processlist）。

执行结果如下：

```
mysql> show processlist\G`
`......`
`*************************** 10. row ***************************`
    `Id: 7651833`
  `User: one`
  `Host: 192.168.1.251:52154`
    `db: ops`
`Command: Query`
  `Time: 3`
 `State: User sleep`
  `Info: select sleep(10)`
`......`
`10 rows in set (0.00 sec)`
```

这里对上面结果解释一下：

- Time：表示执行时间
- Info：表示 SQL 语句

我们这里可以通过它的执行时间（Time）来判断是否是慢 SQL。

2 使用 explain 分析慢查询

分析 SQL 执行效率是优化 SQL 的重要手段，通过上面讲的两种方法，定位到慢查询语句后，我们就要开始分析 SQL 执行效率了，子曾经曰过：“工欲善其事，必先利其器”，我们可以通过 explain、show profile 和 trace 等诊断工具来分析慢查询。本节先讲解 explain 的使用，在下节将分享 show profile 和 trace 的使用。

Explain 可以获取 MySQL 中 SQL 语句的执行计划，比如语句是否使用了关联查询、是否使用了索引、扫描行数等。可以帮我们选择更好地索引和写出更优的 SQL 。使用方法：在查询语句前面加上 explain 运行就可以了。

这也是分析 SQL 时最常用的，也是作者最推荐的一种分析慢查询的方式。下面我们来看下示例~~

为了便于理解，先创建两张测试表（方便第 1、2 节实验使用），建表及数据写入语句如下：

```
CREATE DATABASE muke;         /* 创建测试使用的database，名为muke */
use muke;                     /* 使用muke这个database */
drop table if exists t1;      /* 如果表t1存在则删除表t1 */

CREATE TABLE `t1` (           /* 创建表t1 */
  `id` int(11) NOT NULL auto_increment,
  `a` int(11) DEFAULT NULL,
  `b` int(11) DEFAULT NULL,
  `create_time` datetime NOT NULL DEFAULT CURRENT_TIMESTAMP COMMENT '记录创建时间',
  `update_time` datetime NOT NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP COMMENT '记录更新时间',
  PRIMARY KEY (`id`),
  KEY `idx_a` (`a`),
  KEY `idx_b` (`b`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;

drop procedure if exists insert_t1; /* 如果存在存储过程insert_t1，则删除 */
delimiter ;;
create procedure insert_t1()      /* 创建存储过程insert_t1 */
begin
  declare i int;                  /* 声明变量i */
  set i=1;                        /* 设置i的初始值为1 */
  while(i<=1000)do                /* 对满足i<=1000的值进行while循环 */
    insert into t1(a,b) values(i, i); /* 写入表t1中a、b两个字段，值都为i当前的值 */
    set i=i+1;                    /* 将i加1 */
  end while;
end;;
delimiter ;               /* 创建批量写入1000条数据到表t1的存储过程insert_t1 */
call insert_t1();         /* 运行存储过程insert_t1 */

drop table if exists t2;   /* 如果表t2存在则删除表t2 */
create table t2 like t1;   /* 创建表t2，表结构与t1一致 */
insert into t2 select * from t1;   /* 将表t1的数据导入到t2 */
```

下面尝试使用 explain 分析一条 SQL，例子如下：

```
mysql> explain select * from t1 where b=100;
```

```
mysql> explain select * from t1 where b=100;
+----+-------------+-------+------------+------+---------------+-------+---------+-------+------+----------+-------+
| id | select_type | table | partitions | type | possible_keys | key   | key_len | ref   | rows | filtered | Extra |
+----+-------------+-------+------------+------+---------------+-------+---------+-------+------+----------+-------+
|  1 | SIMPLE      | t1    | NULL       | ref  | idx_b         | idx_b | 5       | const |    1 |   100.00 | NULL  |
+----+-------------+-------+------------+------+---------------+-------+---------+-------+------+----------+-------+
1 row in set, 1 warning (0.00 sec)
```

Explain 的结果各字段解释如下：

加粗的列为需要重点关注的项。

列名	解释
id	查询编号
select_type	查询类型：显示本行是简单还是复杂查询
table	涉及到的表
partitions	匹配的分区：查询将匹配记录所在的分区。仅当使用 partition 关键字时才显示该列。对于非分区表，该值为 NULL。
type	本次查询的表连接类型
possible_keys	可能选择的索引
key	实际选择的索引
key_len	被选择的索引长度：一般用于判断联合索引有多少列被选择了
ref	与索引比较的列
rows	预计需要扫描的行数，对 InnoDB 来说，这个值是估值，并不一定准确
filtered	按条件筛选的行的百分比
Extra	附加信息

表1-explain 各字段解释

其中 explain 各列都有各种不同的值，这里介绍几个比较重要列常包含的值:包含 select_typ、type 和 Extra。

下面将列出它们常见的一些值，可稍微过一遍，不需要完全记下来，在后续章节分析 SQL 时，可以返回查询本节内容并对比各种值的区别。

2.1 select_type

select_type 的值	解释
SIMPLE	简单查询(不使用关联查询或子查询)
PRIMARY	如果包含关联查询或者子查询，则最外层的查询部分标记为primary
UNION	联合查询中第二个及后面的查询
DEPENDENT UNION	满足依赖外部的关联查询中第二个及以后的查询
UNION RESULT	联合查询的结果
SUBQUERY	子查询中的第一个查询
DEPENDENT SUBQUERY	子查询中的第一个查询，并且依赖外部查询
DERIVED	用到派生表的查询
MATERIALIZED	被物化的子查询
UNCACHEABLE SUBQUERY	一个子查询的结果不能被缓存，必须重新评估外层查询的每一行
UNCACHEABLE UNION	关联查询第二个或后面的语句属于不可缓存的子查询

表2-select_type 各项值解释

2.2 type

type的值	解释
system	查询对象表只有一行数据,且只能用于 MyISAM 和 Memory 引擎的表，这是最好的情况
const	基于主键或唯一索引查询，最多返回一条结果
eq_ref	表连接时基于主键或非 NULL 的唯一索引完成扫描
ref	基于普通索引的等值查询，或者表间等值连接
fulltext	全文检索
ref_or_null	表连接类型是 ref，但进行扫描的索引列中可能包含 NULL 值
index_merge	利用多个索引
unique_subquery	子查询中使用唯一索引
index_subquery	子查询中使用普通索引
range	利用索引进行范围查询
index	全索引扫描
ALL	全表扫描

表3-type 各项值解释

上表的这些情况，查询性能从上到下依次是最好到最差。

2.3 Extra

Extra 常见的值	解释	例子
Using filesort	将用外部排序而不是索引排序，数据较小时从内存排序，否则需要在磁盘完成排序	explain select * from t1 order by create_time;
Using temporary	需要创建一个临时表来存储结构，通常发生对没有索引的列进行 GROUP BY 时	explain select * from t1 group by create_time;
Using index	使用覆盖索引	explain select a from t1 where a=111;
Using where	使用 where 语句来处理结果	explain select * from t1 where create_time='2019-06-18 14:38:24';
Impossible WHERE	对 where 子句判断的结果总是 false 而不能选择任何数据	explain select * from t1 where 1<0;
Using join buffer (Block Nested Loop)	关联查询中，被驱动表的关联字段没索引	explain select * from t1 straight_join t2 on (t1.create_time=t2.create_time);
Using index condition	先条件过滤索引，再查数据	explain select * from t1 where a >900 and a like "%9";

Extra 常见的值	解释	例子
Select tables optimized away	使用某些聚合函数（比如 max、min）来访问存在索引的某个字段是	explain select max(a) from t1;

表4-Extra 常见值解释及举例

3 总结

今天我分享的关于定位慢 SQL 及使用 explain 分析慢 SQL 到这里就结束了。本节知识点总结如下：

本节首先讲到如何定位慢 SQL:

- 一种方法是查看慢查询日志
- 另一种方法是 show process 查看正在执行的 SQL

再讲到通过 explain 分析慢 SQL，explain 会返回很多字段，其中 select_type、type、key、rows、Extra 是重点关注项。

在工作中及面试时，SQL 性能优化都是我们经常遇到的问题，要想做好性能优化，我们必须学会使用 SQL 优化时需要的工具，进行定位和分析。由于篇幅的问题，本小节只介绍了 explain 工具的使用，在下节将补充另外两种分析慢查询的工具：show profile 和 trace。在后面我会再讲解 SQL 优化的一些知识点，相信小伙伴们 SQL 性能优化时一定可以越来越熟练。

最后小伙伴们可以将处理问题时的心得体会进行总结，也欢迎给我留言分享，我们一起来交流、学习、进步。

4 问题

你在平常工作中是怎么降低慢查询在生产环境出现的概率？

5 参考资料

《深入浅出 MySQL》（第2版）：第 18 章第 1 节

}

01 开篇词

03 快速学会分析SQL执行效率（下）