26 遇到慢查询问题，可以这样思考与解决

更新时间：2020-05-11 09:27:46

> 人的影响短暂而微弱，书的影响则广泛而深远。——普希金

慢查询的确是个让人头疼的问题，而且几乎是不可避免的会遇到。我的职业生涯里，接触过的无论大小业务系统，慢查询问题几乎是没有缺席过。解决慢查询的过程其实也就是定位、优化查询的过程，由于在之前我们已经见到了很多优化查询的方法，所以，这一节的内容不会很难理解。

1. 慢查询概述

慢查询的关键就是一个“慢”字，那么，究竟为什么会慢呢 ？这里，我不再去谈一些“不合适”的查询语句，而是从 MySQL 自身去找原因。另外，慢查询会有哪些表现呢 ？也就是说这个“慢”字体现在哪些方面呢 ？围绕这两个问题，我们一起去看一看慢查询概述。

1.1 MySQL 的两个瓶颈

不可否认的是 MySQL 服务器自身存在瓶颈（硬件性能），这当然也是造成慢查询的部分原因。这确实不是 “甩锅”，我自身就体会过硬件性能的威力：那是2017年了，我们的线上服务越来越慢，完全从代码上面做优化需要大量的人力和时间，成本有些太高。所以，我们打算升级硬件，换一个方向解决问题。在代码没有做任何变动的情况下，只是将机械硬盘换成了固态硬盘，服务整体性能提高了至少50%以上。

言归正传，MySQL 服务器在 CPU 和 IO 层面存在瓶颈：

- 数据装入内存或从磁盘上读取数据时，CPU 使用率会迅速提升，此时会影响到查询或计算
- 需要载入的数据远远大于内存容量的时候，服务器就不得不使用“交换策略”往返于内存和磁盘之间

虽然，我们可以考虑更换性能更优的 CPU 和更大容量的内存，但是，面对大面积或复杂的查询，仍可能会导致慢查询的出现。所以，一方面考虑提升硬件的性能，另一方面也要考虑优化查询、优化使用。

1.2 慢查询的表现

严格意义上来说，慢查询是 SQL（查询）计算消耗的时间过多。但是，更宽泛的，我们可以把慢查询的表现归纳为以下三点：

- 查询带有大数据量返回，服务器到客户端的返回需要通过网络传输，大数据量的传输无疑会消耗大量的时间
- 发出请求到响应的时间差值超过 1s（更具体的，要结合实际业务定义阈值）的查询
- CPU 和内存瞬时使用率过高，服务器压力陡然增大，大概率是慢查询导致

通常情况下，如果你的查询符合以上描述的情况，那基本可以确定当前的查询是比较慢的或者应用程序中出现了慢查询。当然，如果你额外定义了慢查询的“条款”，也是需要把符合条件的查询并入其中。

2. 慢查询分析工具

对于一条查询是不是慢查询，以及为什么会导致它很慢，MySQL 提供了完善的工具帮助我们定位、分析问题。同时，不仅仅是针对于慢查询，你可以应用这些好用的工具 “调研” 你的任何查询，从而调优查询过程，优化系统性能。

2.1 使用 EXPLAIN 查看执行计划

由于之前在讲解 “MySQL 日志系统” 时，已经详细的对 “慢查询日志” 进行了说明。所以，我这里不再赘述通过日志去定位慢查询。MySQL 的 EXPLAIN（或者是 DESC）命令用于查看 SQL 语句的查询执行计划，它的输出结果能够让我们了解 SQL 优化器是怎样执行 SQL 语句的。它的使用方法非常简单，只需要在查询语句之前添加 DESC 就可以了。如下所示：

```
-- 使用 DESC 查看 SQL 语句的查询执行计划
mysql> DESC SELECT * FROM worker WHERE city = '宿州市' AND salary > 2000 ORDER BY name LIMIT 10;
+----+-------------+--------+------------+------+---------------+------+---------+------+------+----------+-----------------------------+
| id | select_type | table  | partitions | type | possible_keys | key  | key_len | ref  | rows | filtered | Extra                       |
+----+-------------+--------+------------+------+---------------+------+---------+------+------+----------+-----------------------------+
|  1 | SIMPLE      | worker | NULL       | ALL  | NULL          | NULL | NULL    | NULL |    1 |   100.00 | Using where; Using filesort |
+----+-------------+--------+------------+------+---------------+------+---------+------+------+----------+-----------------------------+
1 row in set, 1 warning (0.00 sec)
```

EXPLAIN 命令并不会对我们的查询提供任何调整性的建议，但是它所提供的信息可以帮助我们做出调优的决策。这个命令打印的信息有很多，我们先来看一看各个字段代表什么样的含义（由于部分字段包含的选项太多，这里只讲解最常见的情况）：

- id：SELECT 语句的编号，如果存在子查询，则编号递增
- **select_type**：查询类型
    - **SIMPLE**：简单查询，没有 UNION 或子查询
    - **PRIMARY**：如果包含关联查询或子查询，则最外层的查询标记为 PRIMARY
    - **UNION**：联合查询中第二个以及后面的查询
    - **SUBQUERY**：子查询中第一个 SELECT
- **table**：查询表名
- **partitions**：匹配的分区信息（对于分区表才会有）
- **type**：查询方式，以下介绍的选项从上到下性能递减
    - **CONST**：基于主键或唯一索引查询，最多返回一条记录

    - **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，或者表之间等值连接
    - **RANGE**：根据索引做范围扫描
    - **INDEX**：全索引扫描
    - **ALL**：逐行做全表的扫描，最差的情况是扫描到最后一行

- **possible_keys**：可能会使用到的索引，可以有多个。但是，至多只会使用一个
- **key**：真正会使用到的索引，如果没有使用任何索引，则是 NULL
- **key_len**：使用的索引的长度，如果索引是 NULL，则该值也是 NULL
- **ref**：连接查询时，表之间字段的引用关系
- **rows**：MySQL 认为执行查询时需要检查的行数
- **filtered**：这个字段是5.7版本之后才加进去的，标识存储引擎返回的数据在服务器层过滤之后，剩下多少满足查询记录数的比例
- **Extra**：标识了查询过程的详细解释（非常重要）

    - **Distinct**：找到当前记录的匹配联合结果的第一条记录之后，就不再搜索其他记录了
    - **Not exists**：MySQL 优化了 LEFT JOIN，在当前表中找到了和前一条记录符合 LEFT JOIN 条件后，就不再搜索更多的记录了
    - **Using filesort**：MySQL 需要额外的做一遍排序，可能在内存中，也可能在磁盘中
    - **Using index**：字段的信息直接从索引树中的信息取得，而不再去扫描实际的记录
    - **Using where**：使用了 WHERE 子句用来限制哪些记录匹配

可以看到，EXPLAIN 命令返回的信息量是巨大的。但实际上，我们只需要特别关注：key、Extra 这两个字段即可。例如，对于我们的查询来说，可以从执行计划中得到如下结论：

- 当前的查询没有命中（匹配）任何索引，需要考虑优化表索引结构
- 查询、排序没有命中覆盖索引，可以考虑添加联合索引

当我们根据 EXPLAIN 命令打印的信息得到结论之后，知晓了当前的查询存在的问题，就可以想办法对表结构、查询过程进行优化，将慢查询变为 “快” 查询。

2.2 使用 profiling 查看资源使用情况

通过慢查询日志我们可以直接得到执行效率低下的 SQL 语句，而通过 EXPLAIN 我们可以知道 MySQL 会怎样执行查询。最后，MySQL 还提供了 profiling 命令让我们可以获取到更准确的 SQL 执行消耗的系统资源信息。

首先，我们需要来看一看与 profiling 命令相关的三个参数：

```
mysql> SHOW VARIABLES LIKE '%profiling%';
+------------------------+-------+
| Variable_name          | Value |
+------------------------+-------+
| have_profiling         | YES   |
| profiling              | OFF   |
| profiling_history_size | 15    |
+------------------------+-------+
3 rows in set (0.01 sec)
```

其中，have_profiling 标识当前的 MySQL 中 profiling 命令是否可用，对于 5.0.37 版本之后的 MySQL 都是可用的；profiling 参数标识功能是否处于打开状态，默认是关闭的；profiling_history_size 标识保存多少条最近的记录。所以，在使用 profiling 命令之前，我们需要先把它打开。执行如下 SQL 语句：

```
-- 打开 profiling 功能
mysql> SET GLOBAL profiling = 1;
Query OK, 0 rows affected, 1 warning (0.00 sec)

-- 检验 profiling 功能是否处于打开状态（需要重新打开 MySQL 会话）
mysql> SHOW VARIABLES LIKE 'profiling';
+---------------+-------+
| Variable_name | Value |
+---------------+-------+
| profiling     | ON    |
+---------------+-------+
1 row in set (0.00 sec)
```

设置完毕之后，我们可以在客户端中执行一些查询操作，以给 profiling 准备一些历史记录。接着，我们就可以查看 profiling 记录的查询语句：

```
mysql> SHOW PROFILES;
+----------+------------+-----------------------------------------------------------------------------------------------------------+
| Query_ID | Duration   | Query                                                                                                     |
+----------+------------+-----------------------------------------------------------------------------------------------------------+
|        2 | 0.00147400 | SHOW VARIABLES LIKE '%profiling%'                                                                         |
|        3 | 0.00143700 | SHOW VARIABLES LIKE 'profiling'                                                                           |
|        4 | 0.00015900 | SELECT DATABASE()                                                                                         |
|        5 | 0.00188100 | show databases                                                                                            |
|        6 | 0.00105200 | show tables                                                                                               |
|        7 | 0.00661700 | SELECT * FROM worker WHERE city = '宿州市' AND salary > 2000 ORDER BY name LIMIT 10                          |
|        8 | 0.00011800 | SELECT DATABASE()                                                                                         |
|        9 | 0.00009100 | SELECT * FROM worker LIMTI 10                                                                             |
|       10 | 0.00021600 | SELECT * FROM worker LIMIT 10                                                                             |
|       11 | 0.00059500 | SELECT id, type FROM worker GROUP BY type                                                                 |
|       12 | 0.00153200 | SELECT type FROM worker GROUP BY type                                                                     |
|       13 | 0.00026300 | SELECT type, name, salary FROM worker WHERE city = '宿州市' ORDER BY name LIMIT 100                          |
|       14 | 0.00031700 | SELECT type, name FROM worker WHERE city = '宿州市' GROUP BY type, name ORDER BY name LIMIT 100              |
|       15 | 0.00115200 | SHOW VARIABLES LIKE 'max_length_for_sort_data'                                                            |
|       16 | 0.00025100 | SELECT type, name, salary FROM worker ORDER BY name LIMIT 100                                             |
+----------+------------+-----------------------------------------------------------------------------------------------------------+
15 rows in set, 1 warning (0.00 sec)
```

其中，Query_ID 是 profiling 递增的查询 id；Duration 是查询时间；Query 则是记录的查询语句。我们可以通过 Query_ID 来查看更加详细的执行信息，且可以自行指定想要查看的指标数据。如下所示：

```
mysql> SHOW PROFILE CPU, BLOCK IO FOR QUERY 11;
+--------------------------------+----------+----------+------------+--------------+---------------+
| Status                         | Duration | CPU_user | CPU_system | Block_ops_in | Block_ops_out |
+--------------------------------+----------+----------+------------+--------------+---------------+
| starting                       | 0.000016 | 0.000012 |   0.000004 |            0 |             0 |
| Waiting for query cache lock   | 0.000003 | 0.000002 |   0.000001 |            0 |             0 |
| starting                       | 0.000002 | 0.000002 |   0.000001 |            0 |             0 |
| checking query cache for query | 0.000381 | 0.000058 |   0.000123 |            0 |             0 |
| checking permissions           | 0.000011 | 0.000007 |   0.000004 |            0 |             0 |
| Opening tables                 | 0.000015 | 0.000014 |   0.000001 |            0 |             0 |
| init                           | 0.000053 | 0.000031 |   0.000021 |            0 |             0 |
| end                            | 0.000006 | 0.000003 |   0.000003 |            0 |             0 |
| query end                      | 0.000005 | 0.000005 |   0.000003 |            0 |             0 |
| closing tables                 | 0.000023 | 0.000011 |   0.000004 |            0 |             0 |
| freeing items                  | 0.000040 | 0.000027 |   0.000014 |            0 |             0 |
| cleaning up                    | 0.000040 | 0.000014 |   0.000025 |            0 |             0 |
+--------------------------------+----------+----------+------------+--------------+---------------+
12 rows in set, 1 warning (0.01 sec)
```

profiling 可选的指标参数有：

- ALL：显示所有信息
- **BLOCK IO**：块设备 I O输入输出的次数
- **CONTEXT SWITCHES**：上下文切换的相关开销
- **CPU**：用户和系统的 CPU 使用情况
- **IPC**：发送和接收消息的相关消耗
- **MEMEORY**：内存使用信息
- PAGE FAULTS：主要和次要页面故障的开销
- **SOURCE**：source_function，source_file 等相关开销
- **SWAPS**：交换次数的开销

通过 profiling 打印的资源使用情况，我们可以采取针对性的优化措施。由于最核心的肯定是 CPU 和内存方面的消耗，所以，只要想办法降低这两个方面，查询自然也就不会很慢了。

3. 慢查询问题优化建议

慢查询之所以会出现，基本上归咎于两点原因：查询语句过于复杂、索引设置覆盖不全。下面，我将对可能会引起慢查询的操作进行总结，并给出相应的优化建议。

3.1 SELECT 太多数据行、数据列（大数据量查询）

首先，需要说明，我们不能简单的以查询数据行、数据列的多少来判定是否是慢查询。但是，大多数情况下，这样的操作几乎不会很快，毕竟数据需要通过网络传输（从 MySQL 服务器到客户端）。另外，选取的数据列越多，覆盖索引能够命中的几率往往也会越低。

其实这个问题也很好解决，对于数据行太多而言：如果是业务型数据，使用分页控制即可；如果是报表型数据，分批查询即可。而对于数据列太多的情况，需要重新梳理业务，去除不需要使用到的数据列，精简查询语句。

3.2 没有按照索引列定义的顺序查询

为了说明这个问题，我们先来定义一张数据表 worker，建表语句如下：

```
CREATE TABLE `worker` (
  `id` bigint(20) unsigned NOT NULL AUTO_INCREMENT COMMENT 'id',
  `type` char(64) NOT NULL DEFAULT '' COMMENT '员工类型',
  `name` char(64) NOT NULL DEFAULT '' COMMENT '姓名',
  `salary` bigint(20) unsigned DEFAULT '0' COMMENT '薪水',
  `province` char(64) NOT NULL DEFAULT '' COMMENT '省份',
  `city` char(64) NOT NULL DEFAULT '' COMMENT '城市',
  PRIMARY KEY (`id`),
  KEY `city_nam_type_salary_idx` (`city`,`name`,`type`,`salary`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8 COMMENT='员工表';
```

可以看到，worker 表中定义了一个联合索引 city_nam_type_salary_idx，我们需要注意的是索引中字段的顺序。虽然 SQL 优化器可能会根据索引字段的顺序调整我们的查询语句，但是，不应该依赖于优化器。例如，对于如下的查询：

```
SELECT city, name FROM worker WHERE name IN ('tom', 'pony') AND city = '宿州市';
```

显然，`name` 和 `city` 在 WHERE 子句中定义的顺序与在索引中定义的顺序是不相符的。最好的情况是，调整下 SQL 语句的顺序（`city = '宿州市' AND name IN ('tom', 'pony')`），使之与索引匹配。

3.3 定义了索引，但是 MySQL 并没有选择

有些时候，我们虽然在某一列或几列上面定义了索引，且查询条件也符合索引的顺序，但就是 MySQL 没有使用索引才导致了慢查询。之所以会出现这种情况，还是因为你的查询“并不合适”。下面，我来总结下优化器不使用索引的一些情况和解决办法：

- 在索引列上进行函数计算，优化器大概率不会使用索引。最好的解决办法就是将计算过程转移到代码中去
- 模糊查询（LIKE）% 在前面的情况，优化器不能确定前缀，无法使用索引。可以考虑在代码中做过滤
- 存在 OR 条件连接，会导致后续条件无法使用索引。可以将 OR 修改为 UNION ALL
- 对于时间范围索引，查询的时间跨度很大，接近于全表扫描，优化器会放弃索引。这样的查询往往是不合理的，应该避免
- 索引列存在 NULL 值，维护、使用索引都很费资源，优化器很可能会放弃。索引列一定要 NOT NULL，可以填入“非法值”

3.4 没有定义索引

项目在启动之初，可能不会有很复杂的业务，所以，在索引的定义上可能不会考虑的很全面。这种事再正常不过了，但是，这也会带来很大的麻烦：查询找不到索引时，MySQL 不得不扫描全表，导致大量慢查询的出现。

但是，解决没有索引导致的慢查询问题，大多数时候，不是加上一个索引这么简单的。我在工作中就遇到过类似的问题：业务系统中的一张核心数据表，数据记录数超过了3000万行，经常由于找不到合适的索引而出现慢查询。但是，如果给它加上索引，会导致锁表时间（创建索引期间，MySQL 会获取到表锁，再去执行操作）过长，影响线上业务。那么，对于没有索引的情况，我们可以考虑以下的方法解决：

- 直接创建需要的索引，但是这仅当表中的数据量不大，不影响正常业务运转的情况下
- 表中数据量过大时，对表进行拆分（并建立必要的索引），对小表进行查询（可能会关联查询）
- 对查询的结果进行缓存，减少对数据表的访问次数

3.5 复杂查询

定义了索引，查询条件顺序也与索引顺序一致，且可以确定优化器使用了索引，但是查询执行依然很慢。出现这种问题，大概率是由于你的查询过于复杂了。我曾经排查过一个慢查询问题，一条 SQL 查询执行了一分钟，格式化之后才发现，这条 SQL 语句关联了七张数据表。

实际上，对于这种复杂查询是很难再去做优化的。最合理的方式就是将一个复杂查询拆分为多个简单查询。通常，在使用到索引时，单表或者两张表的联合查询不会执行很慢。但是，需要特别注意，拆分成多个查询之后，需要把它们都放在一个事务中。

4. 总结

可见，MySQL 早已做了一些工作来帮助你去定位、解决慢查询问题，你一定要学会并善于利用它们。另外，不仅仅是慢查询，很多问题或场景，解决方案或思考方向都一定是经验的积累。解决慢查询的同时，也一定要多去总结、思考，逐渐地，你会发现，你现在遇到的慢查询将来也一样会遇到。

5. 问题

你在工作中遇到过慢查询吗 ？ 能举个例子吗 ？ 你又是怎样解决的呢 ？

关于慢查询，你们有自己的 “条款” 吗 ？ 可以一起分享下吗 ？

我们可以缓存查询结果，减少慢查询出现的次数，但是，你知道哪些查询适合做缓存吗 ？

对于慢查询问题的相关建议，你能提出自己的一些建议或者见解吗 ？

6. 参考资料

《高性能 MySQL（第三版）》

MySQL 官方文档：SQL Statements

MySQL 官方文档：Optimizing SQL Statements

MySQL 官方文档：Optimization and Indexes

MySQL 官方文档：Optimizing for InnoDB Tables

MySQL 官方文档：Understanding the Query Execution Plan

}

25 加速 order by 查询，可以从哪些方面做优化呢？

27 服务器性能调优，你知道怎么做吗？