[MySQL优化案例]系列 — discuz!热帖翻页优化

写在前面：discuz!作为首屈一指的社区系统，为广大站长提供了一站式网站解决方案，而且是开源的（虽然部分代码是加密的），它为这个垂直领域的行业发展作出了巨大贡献。尽管如此，discuz!系统源码中，还是或多或少有些坑。其中最著名的就是默认采用MyISAM引擎，以及基于MyISAM引擎的抢楼功能，session表采用memory引擎等，可以参考后面几篇历史文章。本次我们要说说discuz!在应对热门帖子翻页逻辑功能中的另一个问题。

在我们的环境中，使用的是 MySQL-5.6.6 版本。

在查看帖子并翻页过程中，会产生类似下面这样的SQL：

```
mysql> desc SELECT * FROM pre_forum_post WHERE
 tid=8201301 AND `invisible` IN('0','-2') ORDER BY dateline DESC LIMIT 15\G
 *************************** 1. row ***************************
 id: 1
 select_type: SIMPLE
 table: pre_forum_post
 type: ref
 possible_keys: tid,displayorder,first
 key: displayorder
 key_len: 3
 ref: const
 rows: 593371
 Extra: Using index condition; Using where; Using filesort
```

这个SQL执行的代价是：

```
-- 根据索引访问行记录次数，总体而言算是比较好的状态
| Handler_read_key           | 16     |

-- 根据索引顺序访问下一行记录的次数，通常是因为根据索引的范围扫描，或者全索引扫描，总体而言也算是比较好的状态
| Handler_read_next          | 329881 |

-- 按照一定顺序读取行记录的总次数。如果需要对结果进行排序，该值通常会比较大。当发生全表扫描或者多表join无法使用索引时，该值也会比较大
| Handler_read_rnd           | 15     |
```

而当遇到热帖需要往后翻很多页时，例如：

```
mysql> desc SELECT * FROM pre_forum_post WHERE
 tid=8201301 AND `invisible` IN('0','-2') ORDER BY dateline  LIMIT 129860, 15\G
 *************************** 1. row ***************************
 id: 1
 select_type: SIMPLE
 table: pre_forum_post
 type: ref
 possible_keys: displayorder
 key: displayorder
 key_len: 3
 ref: const
 rows: 593371
 Extra: Using where; Using filesort
```

这个SQL执行的代价则变成了（可以看到Handler_read_key、Handler_read_rnd大了很多）：

```
| Handler_read_key           | 129876 | -- 因为前面需要跳过很多行记录
```

```
| Handler_read_next          | 329881 | -- 同上
| Handler_read_rnd           | 129875 | -- 因为需要先对很大一个结果集进行排序
```

可见，遇到热帖时，这个SQL的代价会非常高。如果该热帖被大量的访问历史回复，或者被搜索引擎一直反复请求并且历史回复页时，很容易把数据库服务器直接压垮。

小结：这个SQL不能利用 `displayorder` 索引排序的原因是，索引的第二个列 `invisible` 采用范围查询（RANGE），导致没办法继续利用联合索引完成对 `dateline` 字段的排序需求（而如果是 WHERE tid =? AND invisible IN(?, ?) AND dateline =? 这种情况下是完全可以用到整个联合索引的，注意下二者的区别）。

知道了这个原因，相应的优化解决办法也就清晰了：
创建一个新的索引 idx_tid_dateline，它只包括 tid、dateline 两个列即可（根据其他索引的统计信息，item_type 和 item_id 的基数太低，所以没包含在联合索引中。当然了，也可以考虑一并加上）。

我们再来看下采用新的索引后的执行计划：

```
mysql> desc SELECT * FROM pre_forum_post WHERE
 tid=8201301 AND `invisible` IN('0','-2') ORDER BY dateline  LIMIT 15\G
 *************************** 1. row ***************************
 id: 1
 select_type: SIMPLE
 table: pre_forum_post
 type: ref
 possible_keys: tid,displayorder,first,idx_tid_dateline
 key: idx_tid_dateline
 key_len: 3
 ref: const
 rows: 703892
 Extra: Using where
```

可以看到，之前存在的 **Using filesort** 消失了，可以通过索引直接完成排序了。

不过，如果该热帖翻到较旧的历史回复时，相应的SQL还是不能使用新的索引：

```
mysql> desc SELECT * FROM pre_forum_post WHERE
 tid=8201301 AND `invisible` IN('0','-2') ORDER BY dateline LIMIT 129860,15\G
 *************************** 1. row ***************************
 id: 1
 select_type: SIMPLE
 table: pre_forum_post
 type: ref
 possible_keys: tid,displayorder,first,idx_tid_dateline
 key: displayorder
 key_len: 3
 ref: const
 rows: 593371
 Extra: Using where; Using filesort
```

对比下如果建议优化器使用新索引的话，其执行计划是怎样的：

```
mysql> desc SELECT * FROM pre_forum_post use index(idx_tid_dateline) WHERE
 tid=8201301 AND `invisible` IN('0','-2') ORDER BY dateline  LIMIT 129860,15\G
 *************************** 1. row ***************************
 id: 1
 select_type: SIMPLE
```

```
 table: pre_forum_post
 type: ref
 possible_keys: idx_tid_dateline
 key: idx_tid_dateline
 key_len: 3
 ref: const
 rows: 703892
 Extra: Using where
```

可以看到，因为查询优化器认为后者需要扫描的行数远比前者多了11万多，因此认为前者效率更高。

事实上，在这个例子里，排序的代价更高，因此我们要优先消除排序，所以应该强制使用新的索引，也就是采用后面的执行计划，在相应的程序中指定索引。

最后，我们来看下热帖翻到很老的历史回复时，两个执行计划分别的profiling统计信息对比：

1、采用旧索引（displayorder）：

```
mysql> SELECT * FROM pre_forum_post WHERE
 tid=8201301 AND `invisible` IN('0','-2') ORDER BY dateline LIMIT 129860,15;

#查看profiling结果
 | starting             | 0.020203 |
 | checking permissions | 0.000026 |  检查“权限”
 | Opening tables       | 0.000036 |  打开“表”
 | init                 | 0.000099 |  初始化
 | System lock          | 0.000092 |  系统锁
 | optimizing           | 0.000038 |  优化
 | statistics           | 0.000123 |  统计
 | preparing            | 0.000043 |
 | Sorting result       | 0.000025 |
 | executing            | 0.000023 |
 | Sending data         | 0.000045 |
 | Creating sort index  | 0.941434 |  创建“排序索引”
 | end                  | 0.000077 |
 | query end            | 0.000044 |
 | closing tables       | 0.000038 |
 | freeing items        | 0.000056 |
 | cleaning up          | 0.000040 |
```

2、如果是采用新索引（idx_tid_dateline）：

```
mysql> SELECT * FROM pre_forum_post use index(idx_tid_dateline) WHERE
 tid=8201301 AND `invisible` IN('0','-2') ORDER BY dateline LIMIT 129860,15;

#对比查看profiling结果
 | starting             | 0.000151 |
 | checking permissions | 0.000033 |
 | Opening tables       | 0.000040 |
 | init                 | 0.000105 |
 | System lock          | 0.000044 |
 | optimizing           | 0.000038 |
 | statistics           | 0.000188 |
 | preparing            | 0.000044 |
 | Sorting result       | 0.000024 |
 | executing            | 0.000023 |
 | Sending data         | 0.917035 |
 | end                  | 0.000074 |
 | query end            | 0.000030 |
```

```
| closing tables       | 0.000036 |
| freeing items        | 0.000049 |
| cleaning up          | 0.000032 |
```

可以看到，效率有了一定提高，不过不是很明显，因为确实需要扫描的数据量更大，所以 Sending data 阶段耗时更多。

这时候，我们可以再参考之前的一个优化方案：[MySQL优化案例]系列 — 分页优化

然后可以将这个SQL改写成下面这样：

```
mysql> EXPLAIN SELECT * FROM pre_forum_post t1 INNER JOIN (
 SELECT id FROM pre_forum_post use index(idx_tid_dateline) WHERE
 tid=8201301 AND `invisible` IN('0','-2') ORDER BY
 dateline  LIMIT 129860,15) t2
 USING (id)\G
 *************************** 1. row ***************************
 id: 1
 select_type: PRIMARY
 table:
 type: ALL
 possible_keys: NULL
 key: NULL
 key_len: NULL
 ref: NULL
 rows: 129875
 Extra: NULL
 *************************** 2. row ***************************
 id: 1
 select_type: PRIMARY
 table: t1
 type: eq_ref
 possible_keys: PRIMARY
 key: PRIMARY
 key_len: 4
 ref: t2.id
 rows: 1
 Extra: NULL
 *************************** 3. row ***************************
 id: 2
 select_type: DERIVED
 table: pre_forum_post
 type: ref
 possible_keys: idx_tid_dateline
 key: idx_tid_dateline
 key_len: 3
 ref: const
 rows: 703892
 Extra: Using where
```

再看下这个SQL的 profiling 统计信息：

```
| starting             | 0.000209 |
| checking permissions | 0.000026 |
| checking permissions | 0.000026 |
| Opening tables       | 0.000101 |
| init                 | 0.000062 |
| System lock          | 0.000049 |
| optimizing           | 0.000025 |
| optimizing           | 0.000037 |
```

```
| statistics           | 0.000106 |
| preparing            | 0.000059 |
| Sorting result       | 0.000039 |
| statistics           | 0.000048 |
| preparing            | 0.000032 |
| executing            | 0.000036 |
| Sending data         | 0.000045 |
| executing            | 0.000023 |
| Sending data         | 0.225356 |
| end                  | 0.000067 |
| query end            | 0.000028 |
| closing tables       | 0.000023 |
| removing tmp table   | 0.000029 |
| closing tables       | 0.000044 |
| freeing items        | 0.000048 |
| cleaning up          | 0.000037 |
```

可以看到，效率提升了1倍以上，还是挺不错的。

最后说明下，这个问题只会在热帖翻页时才会出现，一般只有1,2页回复的帖子如果还采用原来的执行计划，也没什么问题。

因此，建议discuz!官方修改或增加下新索引，并且在代码中判断是否热帖翻页，是的话，就强制使用新的索引，以避免性能问题。

扩展阅读：

1、MySQL优化 之 Discuz论坛优化
2、MySQL优化 之 Discuz论坛优化 — 续
3、MySQL优化 之 Discuz论坛MySQL通用优化

最后稍微吐槽一下：最近几天遇到了几起关于MySQL查询优化器的BUG，挺让人摸不着头脑的：（