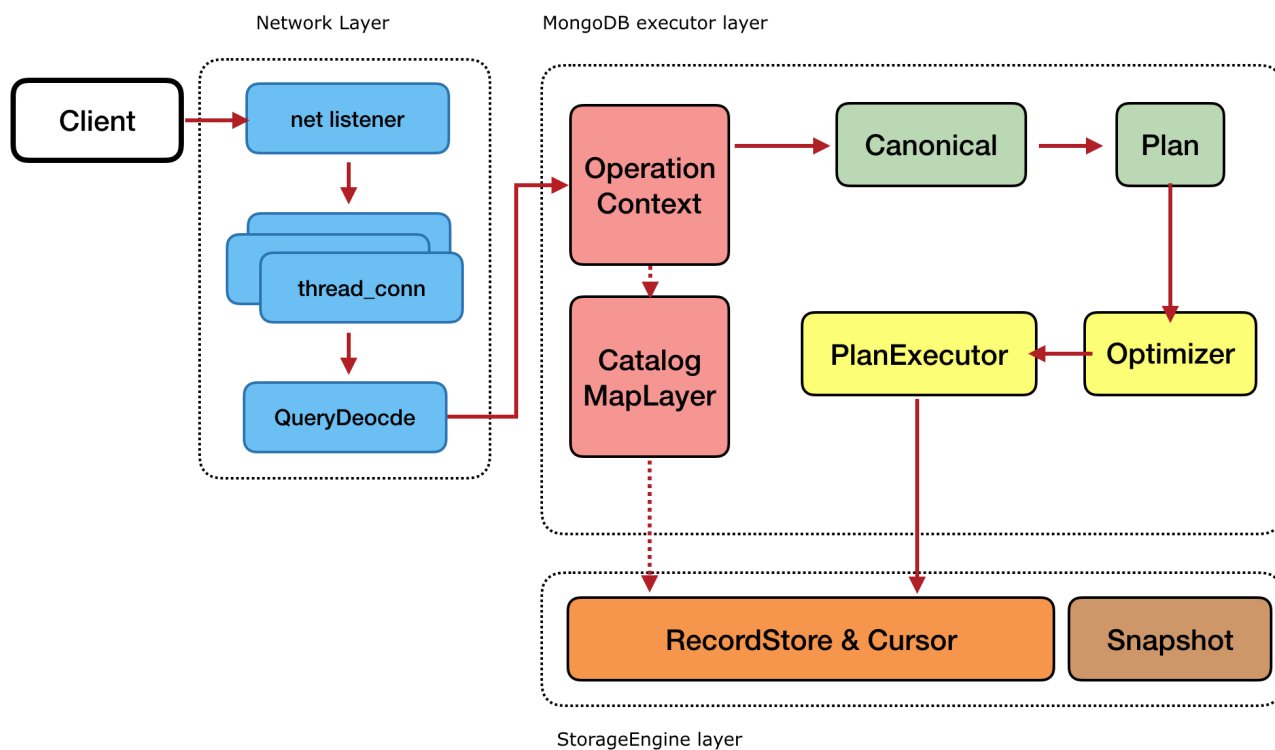


查询优化概要

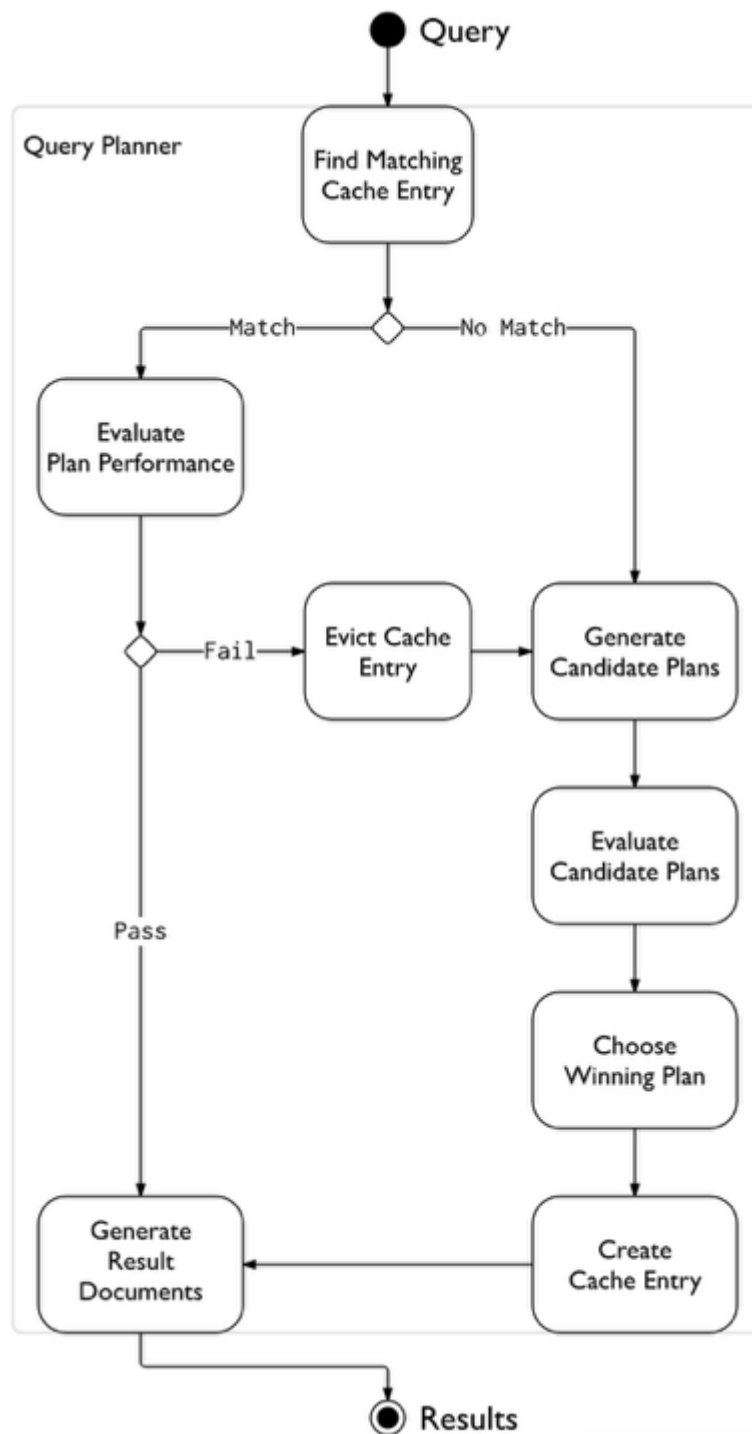
mongodb 查询逻辑

Query在执行层的逻辑，其他模块的逻辑进行了精简：



- Client按照MongoDB的网络协议，请求建立连接，并友单独新建的thread处理所有请求。
- 有些Query(如insert)，本身不需要执行计划和优化，这直接通过接口和引擎交互(通过RecordStore写表)
- Query会进行简单的处理(标准化)，并构造一些上下文数据结构变成CanonicalQuery(标准化Query)。
- Plan模块会负责生成该Query的多个执行计划，然后丢给Optimizer去选择最优的，丢给PlanExecutor。
- PlanExecutor按照执行计划一步一步迭代，获得最终的数据(或执行update修改数据)

查询分析器



优化时所需要用到的命令

```

mongo shell
db.collection.find().explain()           // 查询分析器
db.serverStatus()                        // 查询数据库状态
db.collection.stats()                    // 查询表状态
db.currentOp()                           // 查询数据库当前运行命令
db.collection.getIndexes()               // 查询索引
db.setProfile()                           // 设置慢查询命令，可查看日志信息
tail -f /mongodb/logs/mongod.log         // 查看mongodb 运行日志
db.getCollection("system.profile").find({}) // db.setProfile(1) 时生效 默认慢命令为 100ms

```

优化监控工具

```

mongostat           // mongodb 整体性能监控
mongotop            // 读写性能

```

性能优化指标

计费话单条件查询**

```

2019 - 02 - 22T09 : 32 : 16.777 + 0800 I COMMAND[conn163]command gzb sms.ucCallRecord command : find {
  find : "ucCallRecord",
  filter : {
    $and : [{
      tenementId : {
        $in : ["880001"]}, {calling : /^.*290727.*$/}, {called : /^.*1369.*$/}, {direction : "in"}, {trunkId : "SIPT_4_880001"},
        {wireNumber : /^.*0551.*$/}, {beginTime : {$gte : 1548000000000}}, {beginTime : {$lte : 1550764799000}}
      },
      sort : {
        beginTime : -1
      },
      limit : 15
    }
  },
  planSummary : IXSCAN {
    direction : 1
  }
}
keysExamined : 3511273 docsExamined : 3511273 hasSortStage : 1 cursorExhausted : 1 numYields : 27508 nreturned : 0 reslen : 107 locks : {
  Global : {acquireCount : {r : 55018},acquireWaitCount : {r : 114},timeAcquiringMicros : {r : 136482}},
  Database : {acquireCount : {r : 27509}},
  Collection : {acquireCount : {r : 27509}}
}
protocol : op_query 20453ms

```

如上图所示:

- 查询条件: 企业ID: **880001**, 主叫: **290727**, 被叫: **1369**, 入局, 中继: **cop**, 外线号: **0551**, 时间范围: **2019-01-21 -- 2019-2-21**
- 分页数量为 **15**条
- 扫描数据量为**3511273** 个文档, **3511273** 个索引
- 查询到的数据为 **0** 条
- 耗时 **20.45**秒

查询慢的原因

- 查询条件多, 需要过滤的文档多, 导致扫描数据大
- 没有命中排序索引, 需要排序的数据量大 (模糊查询导致)
- 条件索引多, 选举耗时的耗长
- 内存不够

优化思路：

- 分词：把模糊查询优化成，**分词匹配**，确保索引能够精确命中（模糊查询会扫描所有的索引记录）
- 对于使用模糊查询时，若查找不存在的数据使用 **布隆过滤器** 进行过滤
- 保证内存空间：确保索引都在内存中扫描
- 分表/分库：尽可能地使扫描数据记录减少

优化选择

- 优化成本：硬件>系统配置>数据库表结构>SQL及索引
- 优化效果：硬件<系统配置<数据库表结构<SQL及索引

SQL 及其索引优化

优化项目对比

- 原始模糊查询（索引优化）
- 最左前缀查询（sql优化）
- 查询分词查询（sql优化）
 - 因为需要使用到查询条件分词，会对模糊查询的字段进行分词如：

分词前：

```
{
  "_id" : ObjectId("5b23e627aeee1022a08b8e8b"),
  "_class" : "com.ejiahe.bms.bean.mongobean.CallRecord",
  "tenementId" : "880001",
  "callingNumID" : "249996",
  "calledNumID" : "249999",
  "callingUserId" : "u110003",
  "calledUserId" : "u110002",
  "direction" : "inner",
  "beginTime" : NumberLong("1529079268000"),
  "endTime" : NumberLong("1529079320000"),
  "duration" : 52,
  "billUserId" : "u110003",
  "calledRes" : "OK",
  "calling" : "张冠华",
  "called" : "123",
  "durationTime" : "00:00:52",
  "createTime" : NumberLong("1529079335665"),
}
```

分词后:

```
{
  "_id" : ObjectId("5b23e627aeee1022a08b8e8b"),
  "_class" : "com.ejiahe.bms.bean.mongobean.CallRecord",
  "tenementId" : "880001",
  "callingNumID" : "249996",
  "calledNumID" : "249999",
  "callingUserId" : "u110003",
  "calledUserId" : "u110002",
  "direction" : "inner",
  "beginTime" : NumberLong("1529079268000"),
  "endTime" : NumberLong("1529079320000"),
  "duration" : 52,
  "billUserId" : "u110003",
  "calledRes" : "OK",
  "calling" : "张冠华",
  "called" : "123",
  "durationTime" : "00:00:52",
  "createTime" : NumberLong("1529079335665"),
  "callingarr" : ["冠","冠华","华","张","张冠","张冠华"],
  "calledarr" : ["1","12","23","2","3","123"]
}
```

o

数据量










记录数量：2700W

占用空间










优化方式	数据库占用内存	数据库虚拟内存	表数据大小	预分配文件大小	索引大小
模糊查询	7.1G	7.4G	21.5G	6.5G	2.9G
最左匹配	7.1G	7.4G	21.5G	6.5G	2.9G
分词查询	4.1G	4.5G	68.9G	33.5G	57.6G

索引分配

模糊查询，左前缀查询

 _id_	250,855,424 (250.9 M)
 beginTime_-1	212,881,408 (212.9 M)
 direction_1_beginTime_-1	244,805,632 (244.8 M)
 trunkId_1_beginTime_-1	234,655,744 (234.7 M)
 calling_1_beginTime_-1	722,743,296 (722.7 M)
 called_1_beginTime_-1	856,559,616 (856.6 M)
 wireNumber_1_beginTime_-1	408,252,416 (408.3 M)
 tenementId_1_beginTime_-1	213,897,216 (213.9 M)
 direction_1_trunkId_1_beginTime_-1	246,054,912 (246.1 M)

分词:

 indexSizes	{ 8 fields }
 _id_	250,773,504 (250.8 M)
 monthSetId	111,460,352 (111.5 M)
 direction_1_beginTime_-1	245,837,824 (245.8 M)
 trunkId_1_beginTime_-1	234,205,184 (234.2 M)
 calledarr_1_beginTime_-1	14,759,927,808 (14.8 G)
 beginTime_-1	212,713,472 (212.7 M)
 callingarr_1_beginTime_-1	22,524,891,136 (22.5 G)
 wireNumberarr_1_beginTime_-1	23,502,827,520 (23.5 G)

查询条件

模糊查询

```
db.ucCallRecord.find({
  tenementId : {
    $in : ["880001"]
  },
  calling : {
    $regex : /^.*290727.*$/
  },
  direction : "in",
  trunkId : "SIPT_4_880001",
  beginTime : {
    $gte : 1548000000000,
    $lte : 1550764799000
  }
}).sort({
  beginTime : -1
}).limit(15).explain('executionStats');
```

左前缀查询

```
db.ucCallRecord.find({
  tenementId : {
    $in : ["880001"]
  },
  calling : {
    $regex : /^290727.*$/
  },
  direction : "in",
  trunkId : "SIPT_4_880001",
  beginTime : {
    $gte : 1548000000000,
    $lte : 1550764799000
  }
}).sort({
  beginTime : -1
}).limit(15).explain('executionStats');
```

分词查询

```
db.ucCallRecord.find({
  tenementId : {
    $in : ["880001"]
  },
  callingarr : '290727',
  direction : "in",
  trunkId : "SIPT_4_880001",
  beginTime : {
    $gte : 1548000000000,
    $lte : 1550764799000
  }
}).sort({
  beginTime : -1
}).limit(15).explain('executionStats');
```

查询条件	模糊查询	左前缀	分词
主呼叫: 290727 企业: 880001 方向: 入局 中继: SIPT_4_880001	索引扫描: 653948, 文档扫描: 653948, 耗时: 91625 ms, 返回: 2	索引扫描: 1537, 文档扫描: 1541, 耗时: 177ms, 返回: 0	索引扫描: 2034, 文档扫描: 2034, 耗时: 65ms, 返回: 2
主呼叫: 290727 企业: 880001 中继: SIPT_4_880001	索引扫描: 8539, 文档扫描: 8539, 耗时: 129ms, 返回: 15	索引扫描: 1537, 文档扫描: 1541, 耗时: 51ms, 返回: 15	索引扫描: 16, 文档扫描: 16, 耗时: 15ms, 返回: 15
主呼叫: 290727 中继: SIPT_4_880001	索引扫描: 8539, 文档扫描: 8539, 耗时: 81ms, 返回: 15	索引扫描: 1541, 文档扫描: 1537, 耗时: 25ms, 返回: 15	索引扫描: 16, 文档扫描: 16, 耗时: 0ms, 返回: 15
主呼叫: 290727	索引扫描: 12669, 文档扫描: 12669, 耗时: 73ms, 返回: 15	索引扫描: 1541, 文档扫描: 1537, 耗时: 14ms, 返回: 15	索引扫描: 15, 文档扫描: 15, 耗时: 0ms, 返回: 15

查询匹配越多

查询条件	模糊查询	左前缀	分词
主呼叫: 290727 企业: 880001 方向: 入局 中继: SIPT_4_880001	索引扫描: 653948, 文档扫描: 653948, 耗时: 91625ms, 返回: 2	索引扫描: 1540, 文档扫描: 1537, 耗时: 472 ms, 返回: 0	索引扫描: 2034, 文档扫描: 2034, 耗时: 65ms, 返回: 2
主呼叫: 2907 企业: 880001 方向: 入局 中继: SIPT_4_880001	索引扫描: 28159, 文档扫描: 28159, 耗时: 1311ms, 返回: 15	索引扫描: 36798, 文档扫描: 36560, 耗时: 5249ms, 返回: 0	索引扫描: 2377, 文档扫描: 2377, 耗时: 148 ms, 返回: 15
主呼叫: 290 企业: 880001 方向: 入局 中继: SIPT_4_880001	索引扫描: 3707, 文档扫描: 3707, 耗时: 145ms, 返回: 15	索引扫描: 344784, 文档扫描: 342444, 耗时: 17502 ms, 返回: 2	索引扫描: 2570, 文档扫描: 2570, 耗时: 532ms, 返回: 15
主呼叫: 29 企业: 880001 方向: 入局 中继: SIPT_4_880001	索引扫描: 36, 文档扫描: 36, 耗时: 23 ms, 返回: 15	索引扫描: 399454, 文档扫描: 396748, 耗时: 21831 ms, 返回: 15	索引扫描: 338, 文档扫描: 338, 耗时: 26 ms, 返回: 15

- 条件越少, 查询速度越快
- 扫描数据越少, 查询速度越快

SQL优化小结

模糊查询

- **优点:**
 - 无需做业务上的修改，后期开发灵活，只需要查询时使用正则表达式即可
 - 内存，存储压力小
- **缺点**
 - 扫描记录数量大：需要按顺序扫描所有的索引文档直到满足查询条数为止
 - 查询慢，消耗mongodb 服务器性能大

左前缀匹配索引方式进行模糊查询

- **优点:**
 - 在查询越精确的数据速度越快，若没有查询到匹配的记录会返回很快，基本上都是毫秒级别
 - 内存，存储压力较少
 - 能匹配前缀索引
- **缺点**
 - 当查询匹配到的记录越多时，返回数据越慢（因为需要对查询到的记录进行排序）

分词模糊查询

- **优点:**
 - 数据按照索引顺序，进行精确匹配，查询速度很快
- **缺点**
 - 空间换时间：数据大小增加,所需要内存增大（若内存不足可能会很大程度上影响查询速度）
 - 逻辑复杂：需要对模糊匹配字段进行分词，同时需要添加数组索引，在逻辑复杂度增加
 - 操作不灵活：当需求变更是，若添加/删除模糊字段，需要重新定义数据结构
 - 若是有多模糊查询需要命中索引，则会扫描匹配分词数组，测试若分词数量大时，查询会相对的慢很多（建议不要同时使用两个以上的模糊查询）
 - 数据插入速度慢：分词之后为一个数组字段，数据大大的增大索引也随之增加（当前环境数8G内存，据插入速度：30条/s）

对于需求方的建议

- 组合查询条件尽可能简单，条件越少，越易于走索引扫描，若针对复杂业务尽量使用分类查询
- 模糊查询数据量大是，使用分词（模糊查询的字段建议不超过6位[语义分词另算]）
- 排序字段若也是条件时，建议为必选字段

是否上述查询就已经满足我们所需了，是否还有优化空间呢？

答案：肯定有

数据库表结构优化

优化项目对比（垂直分表）

- 原始模糊查询（全覆盖索引分表优化）
 - 把所有条件提出来，作为覆盖索引进行查询

```
{
  "_id" : ObjectId("5b23e627aeee1022a08b8e8b"),
  "tenementId" : "880001",
  "direction" : "inner",
  "beginTime" : NumberLong("1529079268000"),
  "calling" : "张冠华",
  "called" : "123",
  "wireNumber" : "234556"
}
```

- 最左前缀查询（sql优化）
- 查询分词查询（sql优化）
 - 因为需要使用到查询条件分词，会对模糊查询的字段进行分词如:

分词分表后:

```
{
  "_id" : ObjectId("5b23e627aeee1022a08b8e8b"),
  "tenementId" : "880001",
  "trunk_Id" : "249996",
  "direction" : "inner",
  "beginTime" : NumberLong("1529079268000"),
  "createTime" : NumberLong("1529079335665"),
  "callingarr" : ["冠", "冠华", "华", "张", "张冠", "张冠华"],
  "calledarr" : ["1", "12", "23", "2", "3", "123"],
  "wireNumber": ["1", "2", "12"]
}
```

◦

数据量






























记录数量: 2700W

占用空间

优化方式	数据库占用内存	数据库虚拟内存	表数据大小	预分配文件大小	索引大小
模糊查询	8.1G	8.4G	6.4G	1.9G	13.4G
最左匹配	8.1G	8.4G	6.4G	1.9G	13.4G
分词查询	7.1G	7.4G	16.4G	7.1G	10G

索引分配

模糊查询，左前缀查询

 nindexes	25
 totalIndexSize	14,412,627,968 (13.4 GiB)
  indexSizes	{ 25 fields }
 _id_	258,404,352 (258.4 M)
 direction_1_trunkId_1_tenementId_1_	701,853,696 (701.9 M)
 direction_1_trunkId_1_beginTime_-1_	699,719,680 (699.7 M)
 direction_1_beginTime_-1_calling_1	699,179,008 (699.2 M)
 trunkId_1_beginTime_-1_calling_1	691,613,696 (691.6 M)
 trunkId_1_tenementId_1_beginTime_-	477,241,344 (477.2 M)
 trunkId_1_tenementId_1_beginTime_-	606,556,160 (606.6 M)
 direction_1_trunkId_1_beginTime_-1_	488,882,176 (488.9 M)
 direction_1_trunkId_1_tenementId_1_	489,852,928 (489.9 M)
 tenementId_1_beginTime_-1_calling_	647,000,064 (647.0 M)
 direction_1_tenementId_1_beginTime_	611,397,632 (611.4 M)
 direction_1_beginTime_-1_wireNumbr	487,555,072 (487.6 M)
 direction_1_tenementId_1_beginTime_	699,351,040 (699.4 M)
 direction_1_tenementId_1_beginTime_	488,275,968 (488.3 M)
 trunkId_1_beginTime_-1_wireNumber	475,828,224 (475.8 M)
 trunkId_1_tenementId_1_beginTime_-	693,452,800 (693.5 M)
 tenementId_1_beginTime_-1_wireNun	467,181,568 (467.2 M)
 direction_1_trunkId_1_tenementId_1_	613,474,304 (613.5 M)
 direction_1_trunkId_1_beginTime_-1_	612,134,912 (612.1 M)
 direction_1_beginTime_-1_called_1	610,861,056 (610.9 M)
 trunkId_1_beginTime_-1_called_1	604,004,352 (604.0 M)
 tenementId_1_beginTime_-1_called_1	590,372,864 (590.4 M)
 beginTime_-1_calling_1	644,337,664 (644.3 M)
 beginTime_-1_called_1	588,677,120 (588.7 M)
 beginTime_-1_wireNumber_1	465,420,288 (465.4 M)

分词:

nindexes	5
totalIndexSize	10,904,760,320 (10.2 GiB)
indexSizes	{ 5 fields }
id	257,171,456 (257.2 M)
tenementId_1_beginTime_-1	187,822,080 (187.8 M)
direction_1_beginTime_-1	227,536,896 (227.5 M)
trunkId_1_beginTime_-1	213,745,664 (213.7 M)
calling_1_beginTime_-1	10,018,484,224 (10.0 G)
ok	1

查询条件

模糊查询

```
db.ucCallRecord.find({
  tenementId : {
    $in : ["880001"]
  },
  calling : {
    $regex : /^.*290727.*$/
  },
  direction : "in",
  trunkId : "SIPT_4_880001",
  beginTime : {
    $gte : 1548000000000,
    $lte : 1550764799000
  }
}).sort({
  beginTime : -1
}).limit(15).explain('executionStats');
```

左前缀查询

```
db.ucCallRecord.find({
  tenementId : {
    $in : ["880001"]
  },
  calling : {
    $regex : /^290727.*$/
  },
  direction : "in",
  trunkId : "SIPT_4_880001",
  beginTime : {
    $gte : 1548000000000,

```

```
    $lte : 1550764799000
  }
}).sort({
  beginTime : -1
}).limit(15).explain('executionStats');
```

分词查询

```
db.ucCallRecord.find({
  tenementId : {
    $in : ["880001"]
  },
  callingarr : '290727',
  direction : "in",
  trunkId : "SIPT_4_880001",
  beginTime : {
    $gte : 1548000000000,
    $lte : 1550764799000
  }
}).sort({
  beginTime : -1
}).limit(15).explain('executionStats');
```

查询结果分析

查询条件	模糊查询	左前缀	分词
主呼叫: 290727 企业: 880001 方向: 入局 中继: SIPT_4_880001	索引扫描: 653772, 文档扫描: 2, 耗时: 1325 ms 返回: 2	索引扫描: 439078, 文档扫描: 0, 耗时: 1203 ms 返回: 0	索引扫描: 2034, 文档扫描: 2034, 耗时: 12 ms 返回: 15
主呼叫: 2907 企业: 880001 中继: SIPT_4_880001	索引扫描: 8539, 文档扫描: 15, 耗时: 14 ms 返回: 15	索引扫描: 6565, 文档扫描: 15, 耗时: 17 ms 返回: 15	索引扫描: 16, 文档扫描: 16, 耗时: 1ms 返回: 15
主呼叫: 290 中继: SIPT_4_880001	索引扫描: 8539, 文档扫描: 15, 耗时: 22ms 返回: 15	索引扫描: 6565, 文档扫描: 15, 耗时: 22 ms 返回: 15	索引扫描: 16, 文档扫描: 15, 耗时: 0ms 返回: 15
主呼叫: 29	索引扫描: 12670, 文档扫描: 15, 耗时: 116ms 返回: 15	索引扫描: 8344, 文档扫描: 15, 耗时: 17ms 返回: 15	索引扫描: 15, 文档扫描: 15, 耗时: 0 ms 返回: 15

查询条件	模糊查询	左前缀	分词
主呼叫：290727 企业：880001 方向：入局 中继： SIPT_4_880001	索引扫描： 653772， 文档扫描：2， 耗时：1325 ms 返回：2	索引扫描： 439078， 文档扫描：0， 耗时：1203 ms 返回：0	索引扫描：2034， 文档扫描：2034， 耗时：17 ms 返回：15
主呼叫：2907 企业：880001 方向：入局 中继： SIPT_4_880001	索引扫描：28159， 文档扫描：15， 耗时：56 ms 返回：15	索引扫描： 439078， 文档扫描：0， 耗时：1084 ms 返回：0	索引扫描：2377， 文档扫描：2377， 耗时：20 ms 返回：15
主呼叫：290 企业：880001 方向：入局 中继： SIPT_4_880001	索引扫描：3707， 文档扫描：3707， 耗时：8ms 返回：15	索引扫描： 439080， 文档扫描： 439080， 耗时：1060 ms 返回：2	索引扫描：2570， 文档扫描：2570， 耗时：12 ms 返回：15
主呼叫：29 企业：880001 方向：入局 中继： SIPT_4_880001	索引扫描：485， 文档扫描：15， 耗时：1ms 返回：15	索引扫描： 372989， 文档扫描：15， 耗时：948 ms 返回：15	索引扫描：338， 文档扫描：338， 耗时：7 ms 返回：15

表结构修改小结

模糊查询（覆盖索引）

- 优点
 - 空间占用小
 - 因为全部扫描覆盖索引，数据紧密查询速度快 0 -100w 查询结果时建议使用该方案
 - 查询速度 指标: 扫描： 3801900条索引 需要 7614ms即可查寻出来，当前机器内存为：8G，
- 缺点
 - 随着数据量增大，查询扫描索引越大，查询越慢（内存足够另算）
 - 当扫描一条不存在的数据时会进行全表的索引扫描（**建议使用布隆过滤器进行排除**）
 - 单线程插入速度：插入10w 条数据需要 884s，换算下来：113条/s
 - 因为全部都是覆盖索引的原因，导致每条查询语句都需要指定索引查询，开发成本高，程序业务应变能力差

分词查询

- 优点
 - 数据按照索引顺序，进行精确匹配，查询速度很快
- 缺点
 - 空间占用大，扫描文档消耗增加
 - 分词字段因为都分成数组的原因，单挑数据的索引大，不合适对其他多个字段进行灵活组合

- 逻辑复杂: 需要对模糊匹配字段进行分词, 同时需要添加数组索引, 在逻辑复杂增加
- 索引复杂, 数据插入慢 50条/s
- 由于mongo 特性的原因: 不能同时使用两个或者两个以上的数组作为索引

垂直分表之后还能继续优化吗?

优化项目 (水平拆分: 分词)

续上次讨论遗留下来的问题

为什么要读写分离

- 物理服务器增加, 负荷增加
- 考虑并发性能 (读和写连接使用的抢占)
- 主从只负责各自的写和读, 极大程度的缓解X锁和S锁争用
- 对于读或者写库做一些针对性的个性优化

count 和explain 的区别:

- **explain**

首先找到查询第一个记录所在的page (记为 P_{Left}), 统计 P_{Left} 里的记录数 (记为 $Records_P_{Left}$), 之后找到最后一个记录所在的page (记为 P_{Right}), 统计 P_{Right} 的记录数 ($Records_P_{Right}$), 之后将 $Records_P_{Left}$ 与 $Records_P_{Right}$ 取平均, 最后乘以总共的page数目 (记为 $Page_Num$)。公式如下:

$$Rows = ((Records_P_{Left} + Records_P_{Right})/2) * Page_Num$$

- **count**

count 则是扫描所有数据记录

参考资料:

<https://blog.csdn.net/samjustin1/article/details/52640125>

<https://www.cnblogs.com/LBSer/p/3333881.html>

<https://yq.aliyun.com/articles/647563>

<https://mp.weixin.qq.com/s/-lru6FAhkXTo7gLCCfWlyg>

http://stor.zol.com.cn/222/2223038_all.html#p2223045