云数据库 MongoDB

最佳实践

产品文档

腾讯云

文档目录

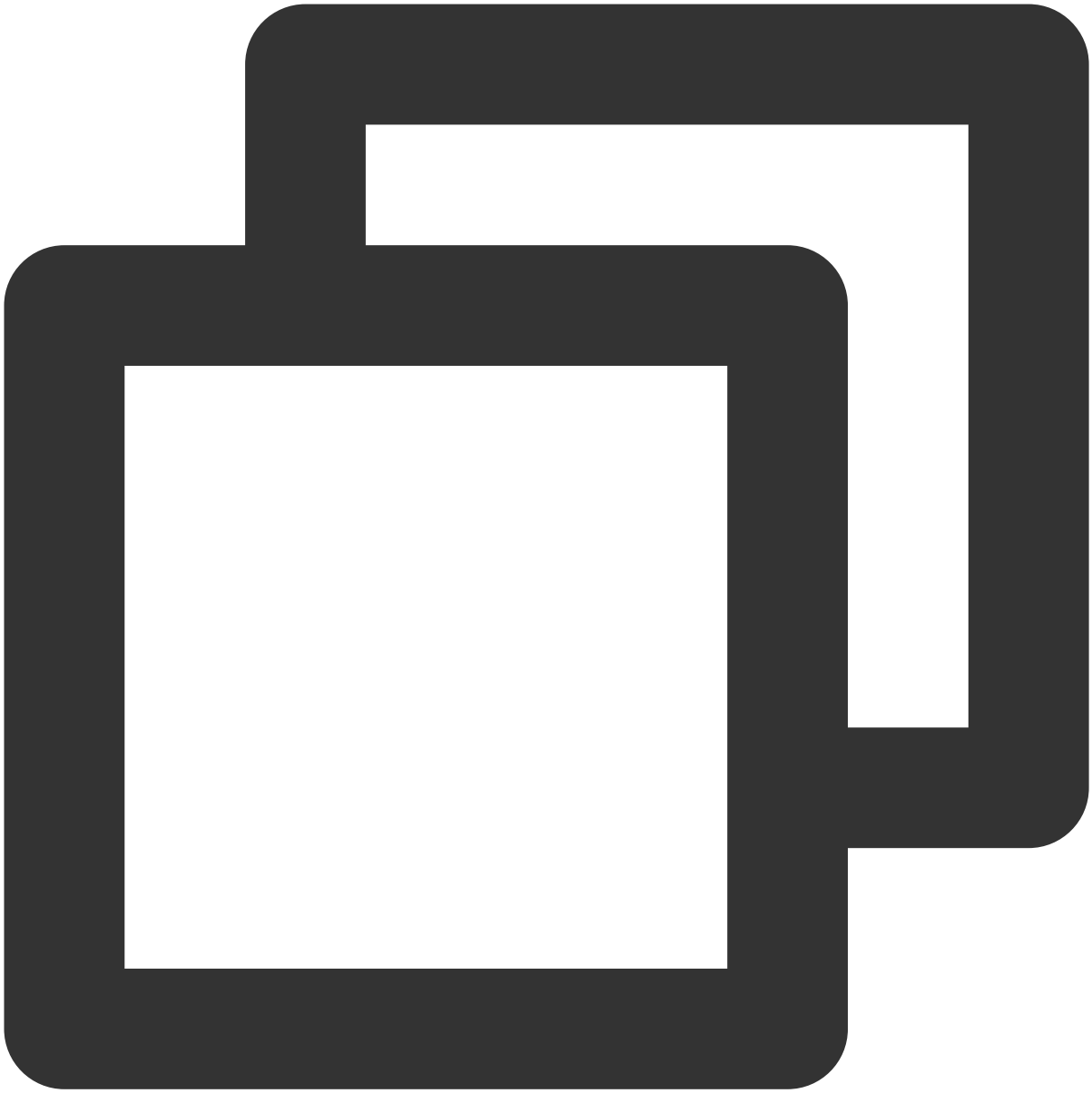
最佳实践
索引优化解决读写性能瓶颈

最近更新时间：2024-01-12 10:43:28

索引对 MongoDB 数据库查询性能起着至关重要的作用，用最少索引满足用户查询需求，可极大提升数据库性能，减少存储成本。本文介绍一系列索引优化分析过程，助力您解决数据库读写性能瓶颈问题。

异常现象

日常运维，登录 MongoDB 控制台，单击实例 ID 进入**实例详情**页面，可查看如下信息。
选择**系统监控**页签，检查实例的监控数据。
发现集群 Mongod 节点 CPU 消耗过高，CPU 使用率经常接近90%，甚至100%。
磁盘每秒读写次数持续偏高，IO 消耗过高，单节点 IO 资源消耗占整服务器60%。
选择**数据库管理**页签，再选择**慢日志查询**页签，查看慢日志。
实例存在大量慢日志，且慢日志中包含大量不同类型 find 和 update 请求，高峰期每秒可达数千条。
慢日志类型各不相同，查询条件众多，所有慢查询都有匹配索引。其内容如下所示。



```
Mon Aug  2 10:34:24.928 I COMMAND  [conn10480929] command xxx.xxx command: find { f

Mon Aug  2 10:34:22.965 I COMMAND  [conn10301893] command xx.txxx command: find { f
```

原因分析

分析慢日志，发现查询请求均有使用 { alxxxId: 1.0, itemTagList: 1.0 } 索引。该索引扫描的 keysExamined 为1498行，扫描的 docsExamined 为1498行，但是返回的 doc 文档数却只有 nreturned =3 行。即满足条件的数据只有3条，

但是却扫描了1498行数据和索引。可见，影响读写性能的关键原因在于索引设置不合理性。

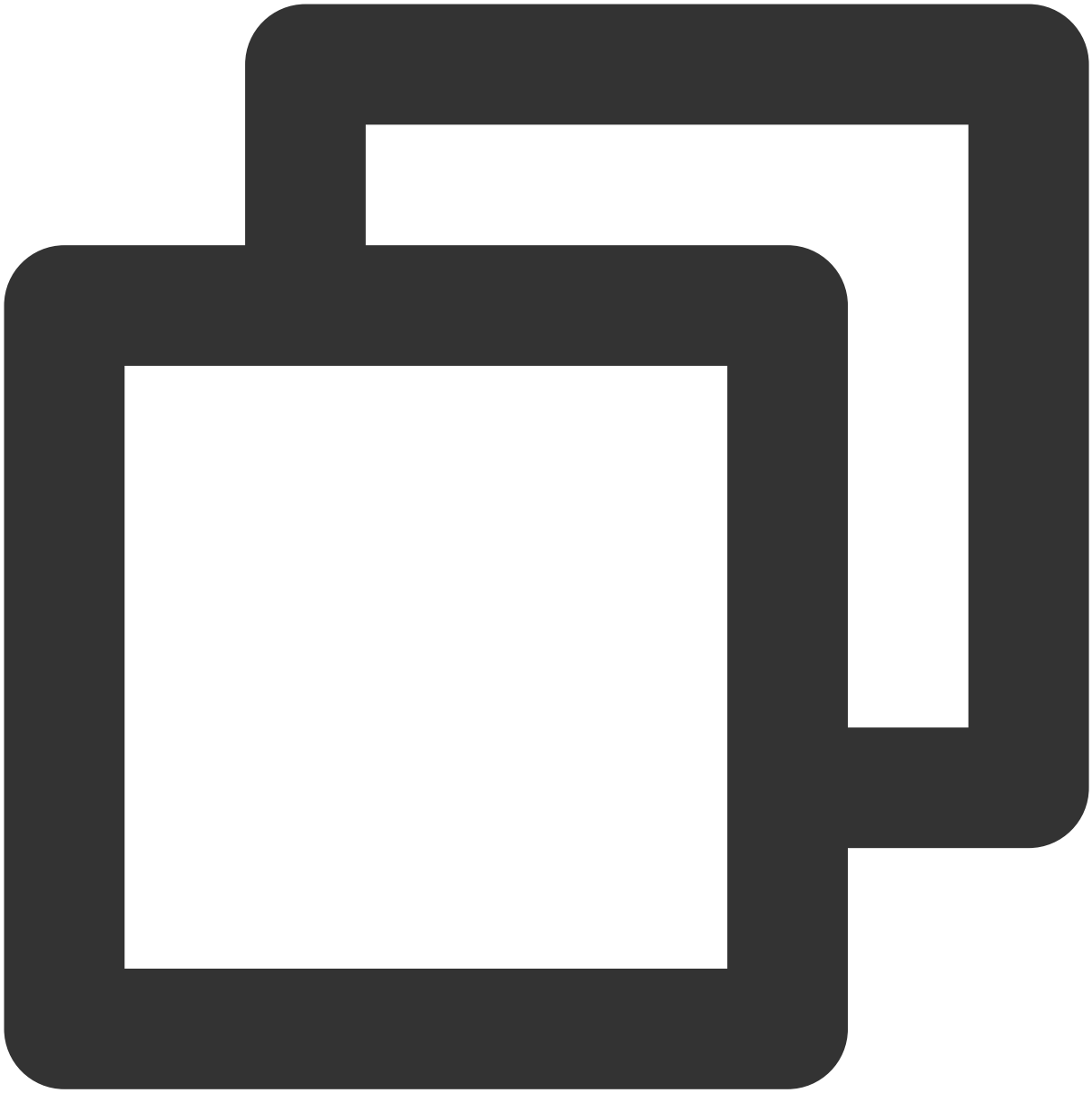
说明：

keysExamined 指明索引扫描条目。docsExamined 代表文档扫描条目。keysExamined 和 docsExamined 越大，说明没有建索引或者索引的区分度不高。

索引优化过程

步骤1：收集用户数据模式

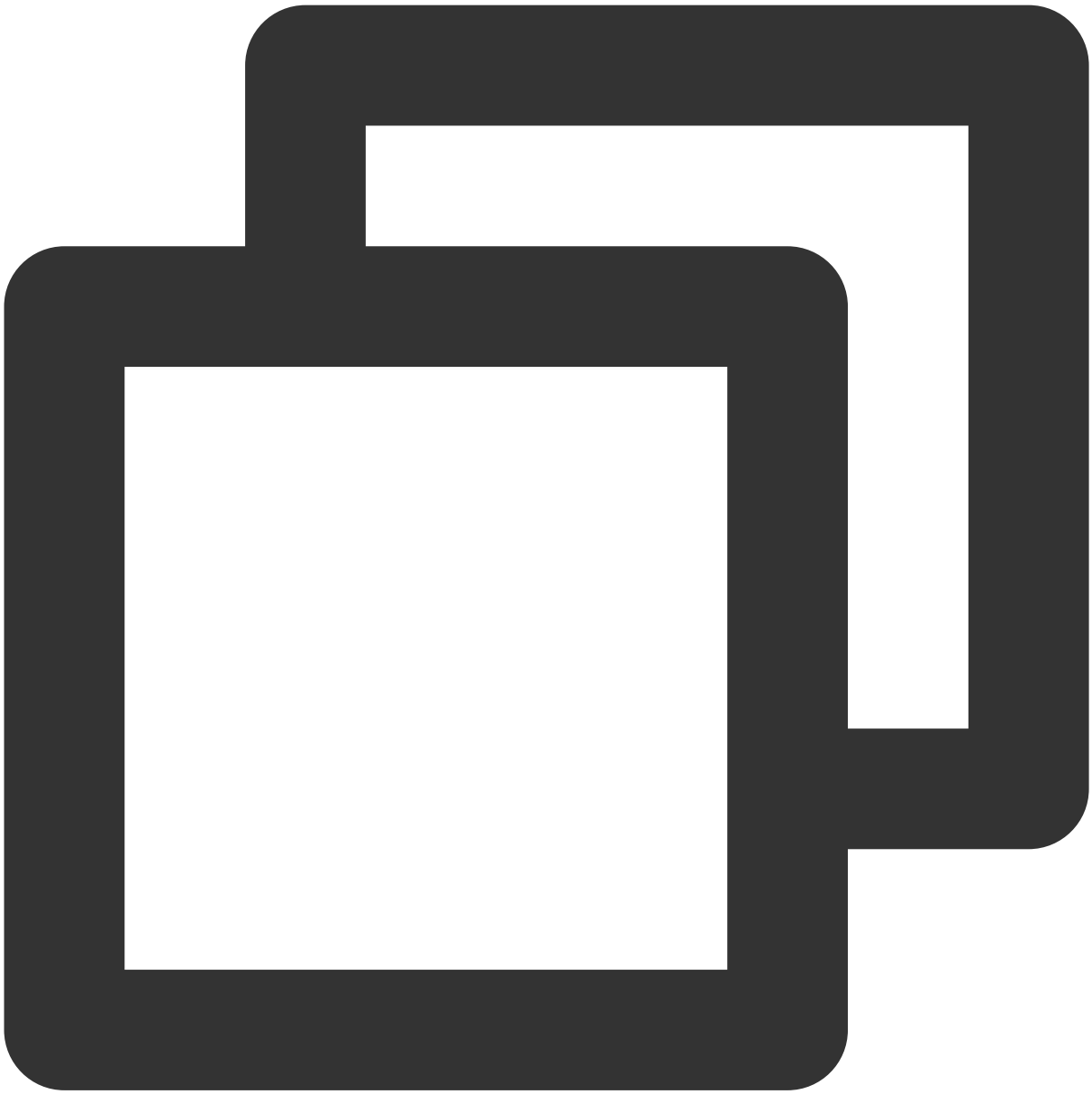
业务常用查询、更新类 SQL，如下所示：



基于 AlxxxId(用户ID) + itxxxId(单个或多个)
基于 AlxxxId 查询 count
基于 AlxxxId 通过时间范围(createTime)进行分页查询，部分查询会拼接 state 及其他字段
基于 AlxxxId, ParentAlxxxId, parentItxxxId, state 组合查询
基于 ItxxxId(单个或多个) 查询数据
基于 AlxxxId, state, updateTime 组合查询
基于 AlxxxId, state, createTime, totalStock(库存数量) 组合查询
基于 AlxxxId(用户ID) + itxxxId(单个或多个) + 任意其他字段组合
基于 AlxxxId, digitalxxxrmarkId(水印ID), state 进行查询
基于 AlxxxId, itemTagList(标签ID), state 等进行查询
基于 AlxxxId + itxxxId(单个或多个) + 其他任意字段进行查询

其他查询

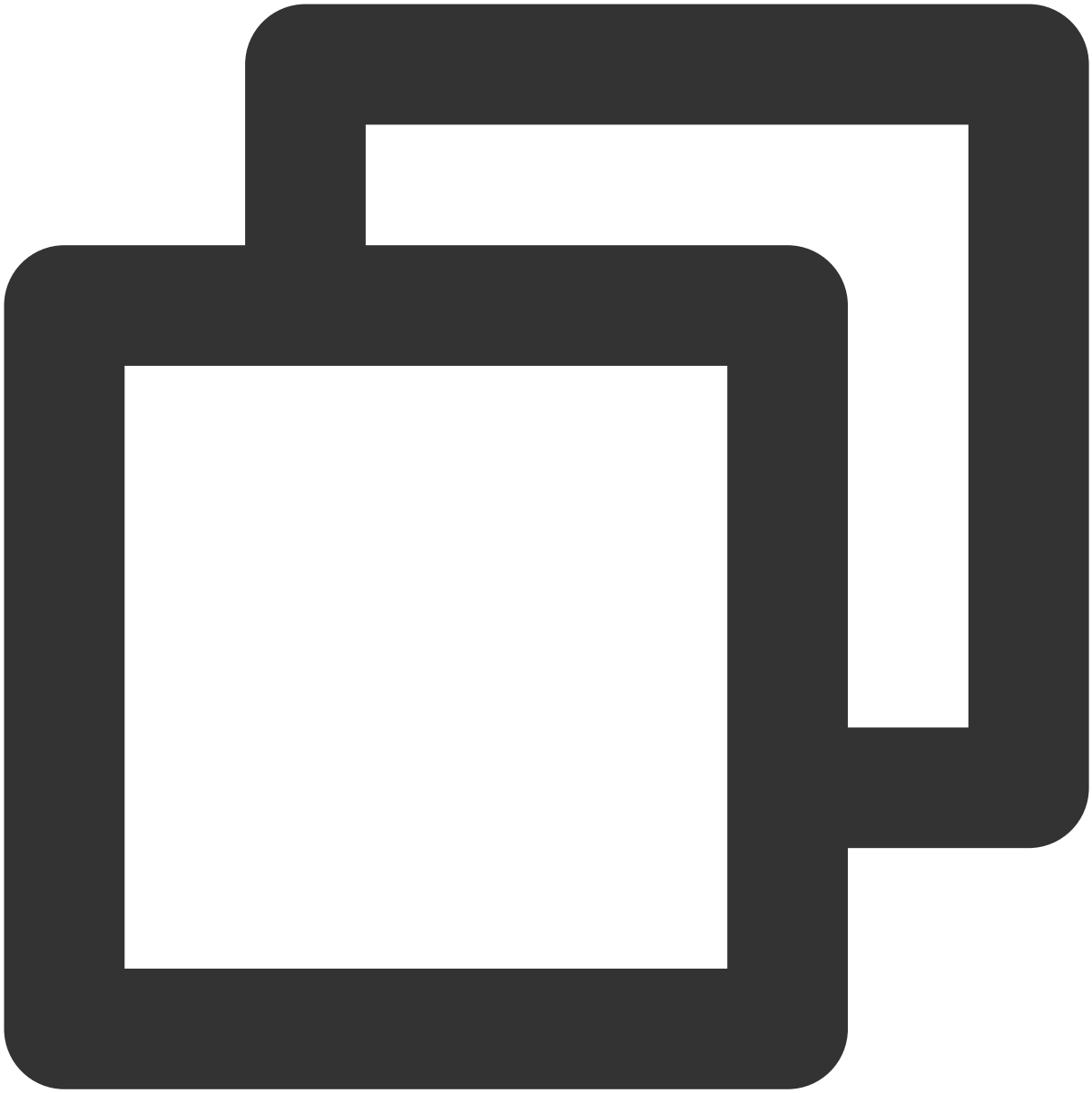
业务常用统计类 count 查询SQL，如下所示：



```
AlxxxId, state, persxxal 组合
AlxxxId, state, itemType 组合
AlxxxId(用户ID) + itxxxId(单个或多个) + 任意其他字段组合
```

步骤2：获取集群已有索引

通过 db.xxx.getindex() 获取到该表索引信息，查询复杂，索引众多，总计30个索引，如下所示。



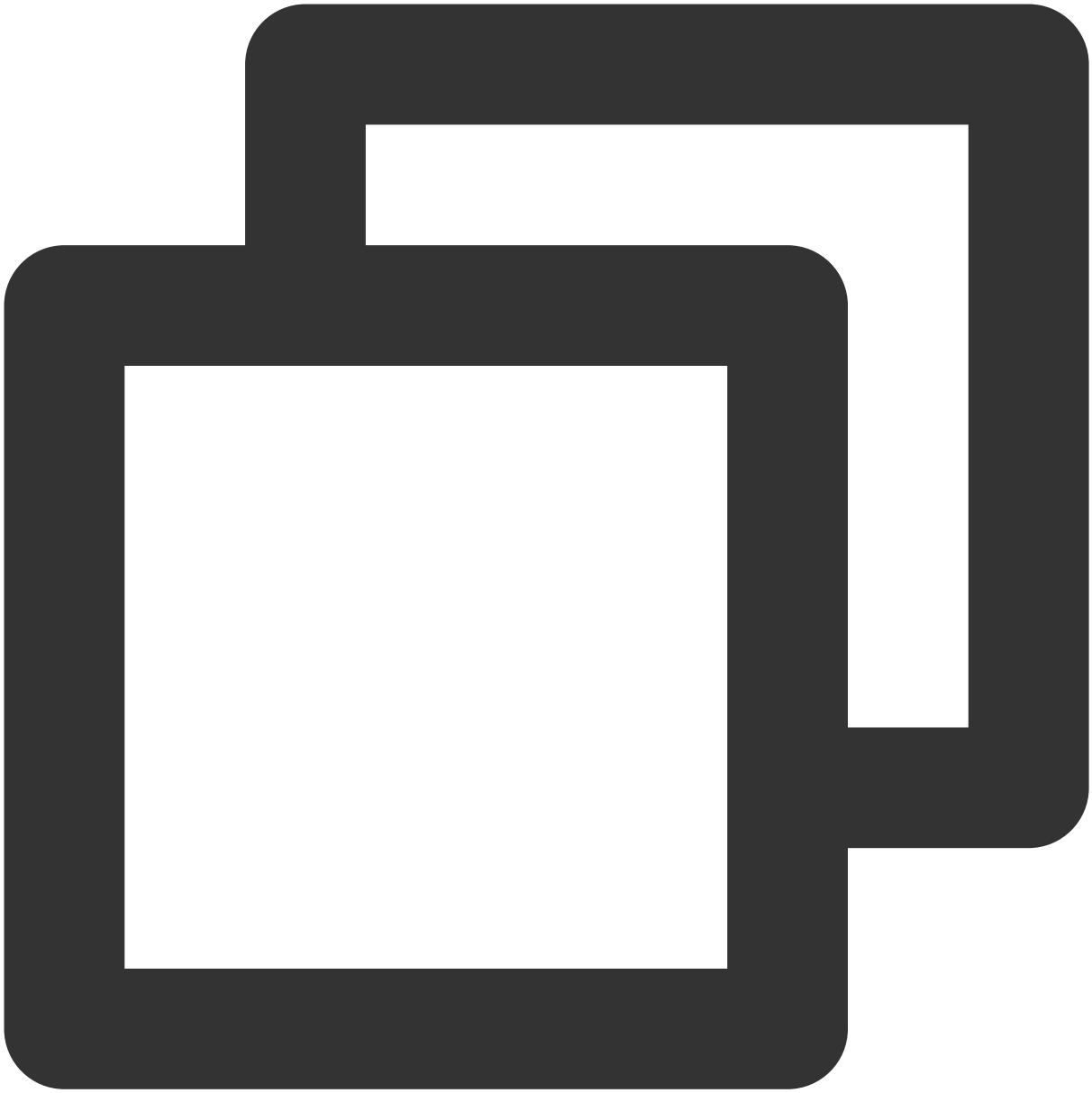
```
{ "alxxxId" : 1, "state" : -1, "updateTime" : -1, "itxxxId" : -1, "persxxal" : 1, "
{ "alxxxId" : 1, "image" : 1 }
{ "itexxxList.vidxxCheck" : 1, "itemType" : 1, "state" : 1 }
{ "alxxxId" : 1, "state" : -1, "newsendTime" : -1, "itxxxId" : 1, "persxxal" : 1 }
{ "_id" : 1 }
{ "alxxxId" : 1, "createTime" : -1, "checkStatus" : 1 }
{ "alxxxId" : 1, "parentItxxxId" : -1, "state" : -1, "updateTime" : -1, "persxxal"
{ "alxxxId" : 1, "state" : -1,  "parentItxxxId" : 1, "updateTime" : -1, "persxxal"
{ "srcItxxxId" : 1 }
{ "createTime" : 1 }
{ "itexxxList.boyunState" : -1, "itexxxList.wozhituUploadServerId": -1, "itexxxList
```

```
{ "alxxxId" : 1, "state" : 1, "digitalxxxrmarkId" : 1, "updateTime" : -1 }
{ "itxxxId" : -1 }
{ "alxxxId" : 1, "parentItxxxId" : 1, "parentAlxxxId" : 1, "state" : 1 }
{ "alxxxId" : 1, "videoCover" : 1 }
{ "alxxxId" : 1, "itemType" : 1 }
{ "alxxxId" : 1, "state" : -1, "itemType" : 1, "persxxal" : 1, "updateTime" : 1 }
{ "alxxxId" : 1, "itxxxId" : 1 }
{ "itxxxId" : 1, "alxxxId" : 1 }
{ "alxxxId" : 1, "parentAlxxxId" : 1, "state" : 1 }
{ "alxxxId" : 1, "itemTagList" : 1 }
{ "itexxxList.photoQiniuUrl" : 1, "itexxxList.boyunState" : -1, "itexxxList.sourceT
{ "alxxxId" : 1, "parentItxxxId" : 1, "state" : 1 }
{ "alxxxId" : 1, "parentItxxxId" : 1, "updateTime" : 1 }
{ "updateTime" : 1 }
{ "itemPhoxxIdList" : -1 }
{ "alxxxId" : 1, "state" : -1, "isTop" : 1 }
{ "alxxxId" : 1, "state" : 1, "itemResxxxIdList" : 1, "updateTime" : -1 }
{ "alxxxId" : 1, "state" : -1, "itexxxList.photoQiniuUrl" : 1 }
{ "itexxxList.qiniuStatus" : 1, "itexxxList.photoNetUrl" : 1, "itexxxList.photoQini
{ "itemResxxxIdList" : 1  }
```

步骤3：索引优化

删除无用索引

MongoDB 支持通过索引统计命令获取各个索引命中的次数，该命令如下：



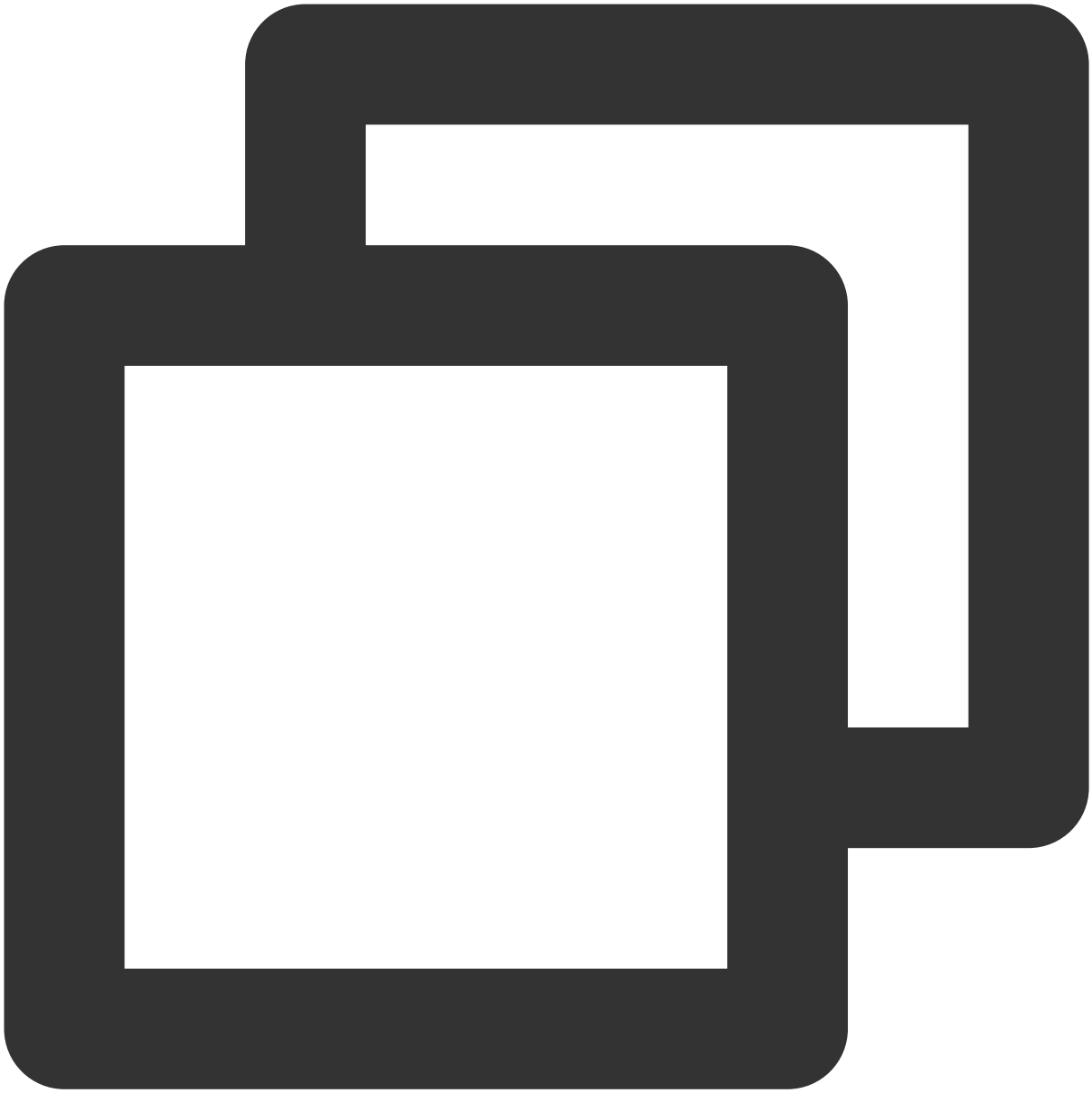
```
> db.xxxxx.aggregate({"$indexStats":{}})
{ "name" : "alxxxId_1_parentItxxxId_1_parentAlxxxId_1", "key" : { "alxxxId" : 1, "p
```

字段含义解释如下。

name：索引名，针对该索引名进行统计。

ops：索引命中次数，即所有查询中，使用本索引作为查询请求命中的次数。 如果命中次数为0或者很小，说明该索引很少被选为最优索引，可认为其为无用索引。

使用该索引统计命令获取所有索引命中的次数，如下所示。命中次数为0或者很小，直接删除。同时，业务已运行一段时间，ops 小于10000也删除。总计可删除11个无用索引，剩余有用索引30 - 11 = 19个。



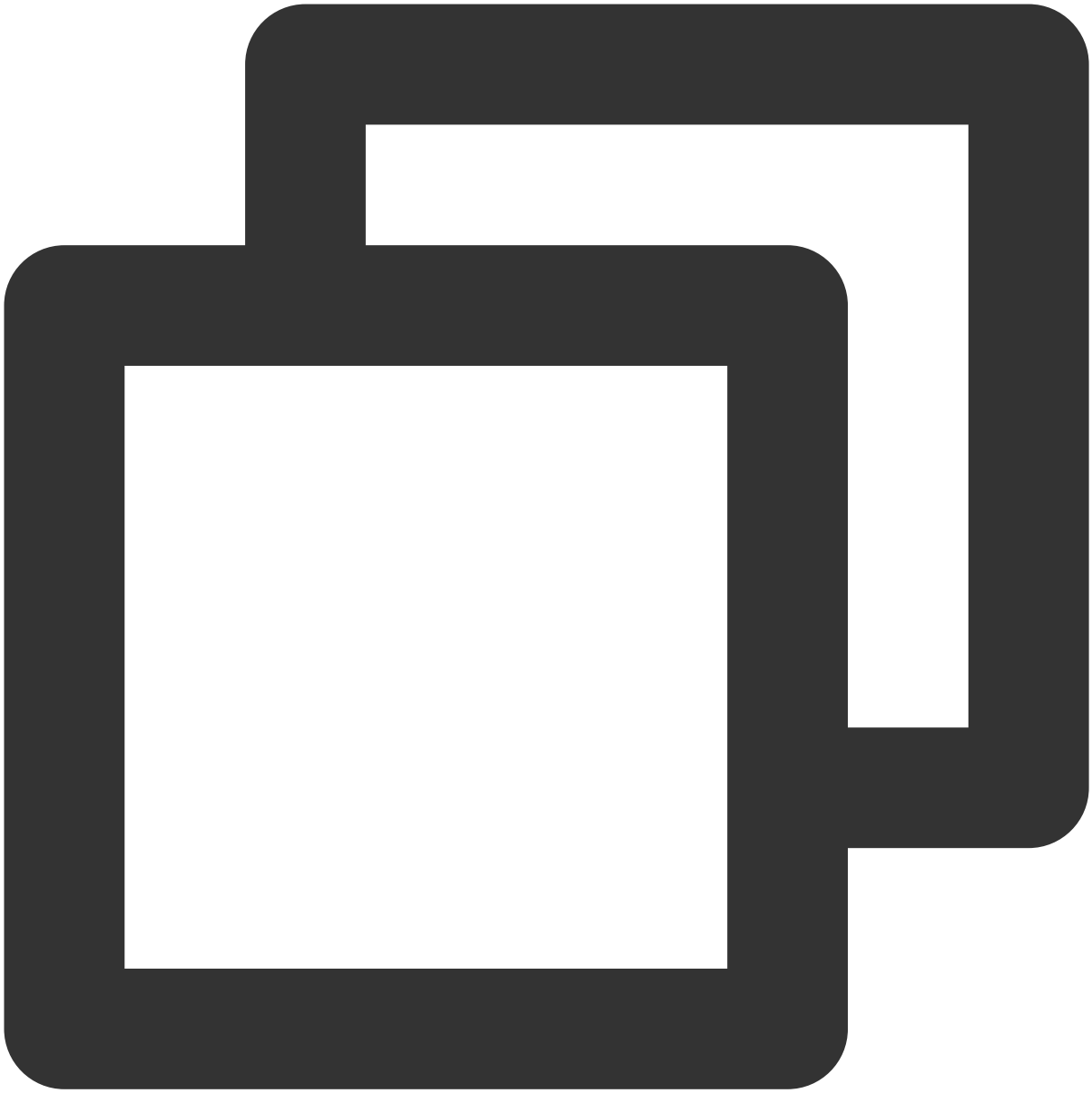
```
db.xxx.aggregate({"$indexStats":{}})
{ "alxxxId" : 1, "state" : -1, "updateTime" : -1, "itxxxId" : -1, "persxxal" : 1, "
{ "alxxxId" : 1, "image" : 1 }                                  "ops" : NumberLong(293104
{ "itexxxList.vidxxCheck" : 1, "itemType" : 1, "state" : 1 }    "ops" : NumberLong(
{ "alxxxId" : 1, "state" : -1, "newsendTime" : -1, "itxxxId" : -1, "persxxal" : 1 }
{ "_id" : 1 }                                                     "ops" : NumberLong(3987)
 { "alxxxId" : 1, "createTime" : 1, "checkStatus" : 1 }      "ops" : NumberLong(200
{ "alxxxId" : 1, "parentItxxxId" : -1, "state" : -1, "updateTime" : -1, "persxxal"
{ "alxxxId" : 1, "state" : -1,  "parentItxxxId" : 1, "updateTime" : -1, "persxxal"
{ "itxxxId" : -1 }      "ops" : NumberLong(38854593)
{ "srcItxxxId" : -1 }                                       "ops" : NumberLong(0)
```

```
{ "createTime" : 1 }                                "ops" : NumberLong(62)
{ "itexxxList.boyunState" : -1, "itexxxList.wozhituUploadServerId" : -1, "itexxxLis
{ "alxxxId" : 1, "state" : 1, "digitalxxxrmarkId" : 1, "updateTime" : -1 }
{ "itxxxId" : -1 }                  "ops" : NumberLong(38854593)
{ "alxxxId" : 1, "parentItxxxId" : 1, "parentAlxxxId" : 1, "state" : 1 }    "ops" :
{ "alxxxId" : 1, "videoCover" : 1 }         { "ops" : NumberLong(2921857)
{ "alxxxId" : 1, "itemType" : 1 }           { "ops" : NumberLong(457)
{ "alxxxId" : 1, "state" : -1, "itemType" : 1, "persxxal" : 1, " itxxxId " : 1 }
{ "alxxxId" : 1, "itxxxId" : 1 }        "ops" : NumberLong(232360252)
{ "itxxxId" : 1, "alxxxId" : 1 }        "ops" : NumberLong(145640252)
{ "alxxxId" : 1, "parentAlxxxId" : 1, "state" : 1 }          "ops" : NumberLong(689
{ "alxxxId" : 1, "itemTagList" : 1 }                    "ops" : NumberLong(28986936
{ "itexxxList.photoQiniuUrl" : 1, "itexxxList.boyunState" : 1, "itexxxList.sourceTy
{ "alxxxId" : 1, "parentItxxxId" : 1, "state" : 1 }                "ops" : NumberLo
{ "alxxxId" : 1, "parentItxxxId" : 1, "updateTime" : 1 }          "ops" : NumberLon
{ "updateTime" : 1 }                                          "ops" : NumberLong(139
{ "itemPhoxxIdList" : -1 }         "ops" : NumberLong(0)
{ "alxxxId" : 1, "state" : -1, "isTop" : 1 }        "ops" : NumberLong(213305)
{ "alxxxId" : 1, "state" : 1, "itemResxxxIdList" : 1, "updateTime" : 1 }       "ops
{ "alxxxId" : 1, "state" : 1, "itexxxList.photoQiniuUrl" : 1}  "ops" : NumberLong(2
{ "itexxxList.qiniuStatus" : 1, "itexxxList.photoNetUrl" : 1, "itexxxList.photoQini
{ "itemResxxxIdList" : 1  }                "ops" : NumberLong(7)
```

删除重复索引

查询顺序引起的索引重复。

该业务不同开发写了两个索引，如下所示。 通过分析，这两个 SQL 索引的目的是一致的，创建其中任何一个索引即可。

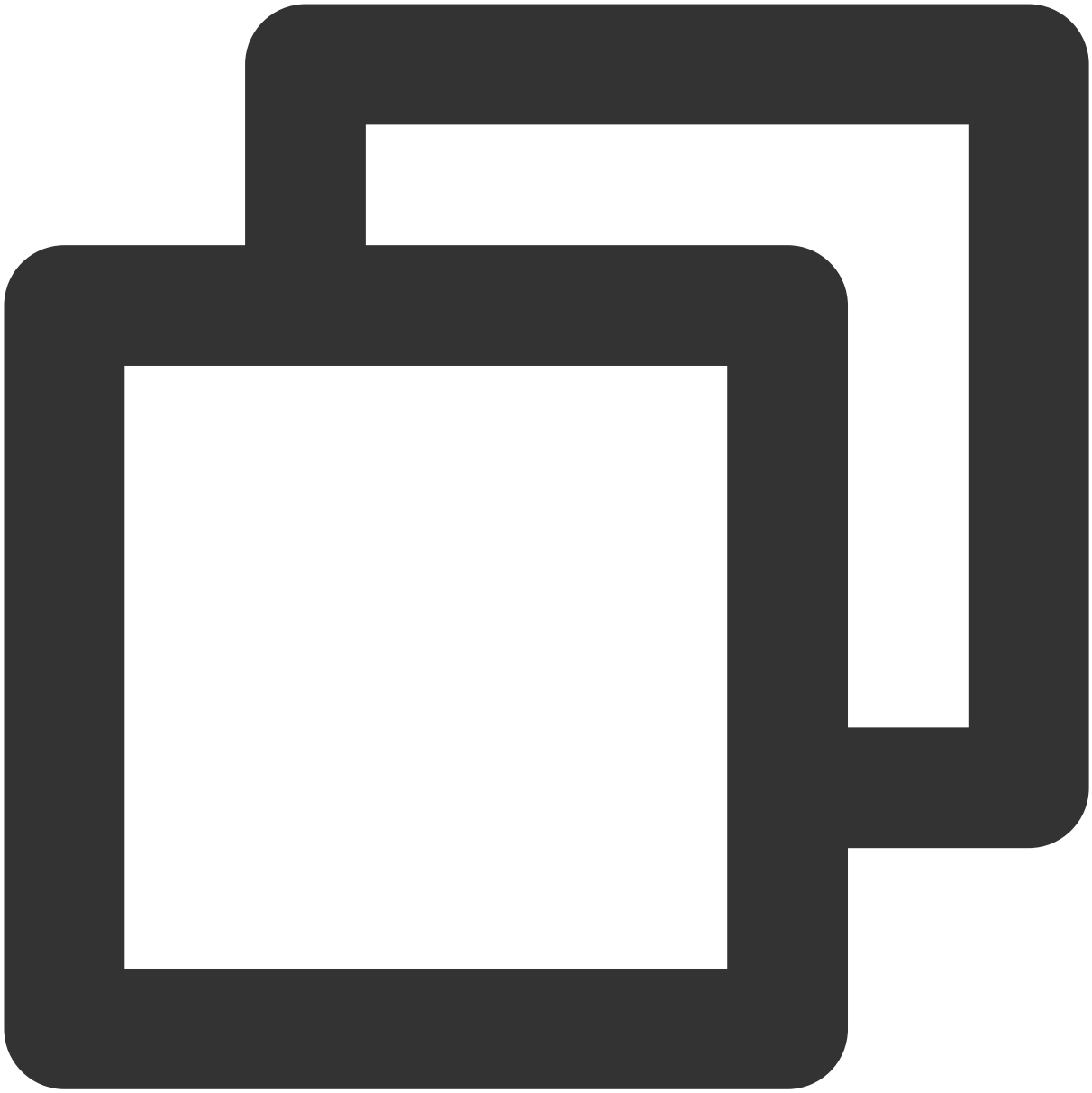


```
db.xxxx.find({{ "alxxxId" : xxx, "itxxxId" : xxx }})
db.xxxx.find({{ " itxxxId " : xxx, " alxxxId " : xxx }})
```

最左原则匹配引起的索引重复。

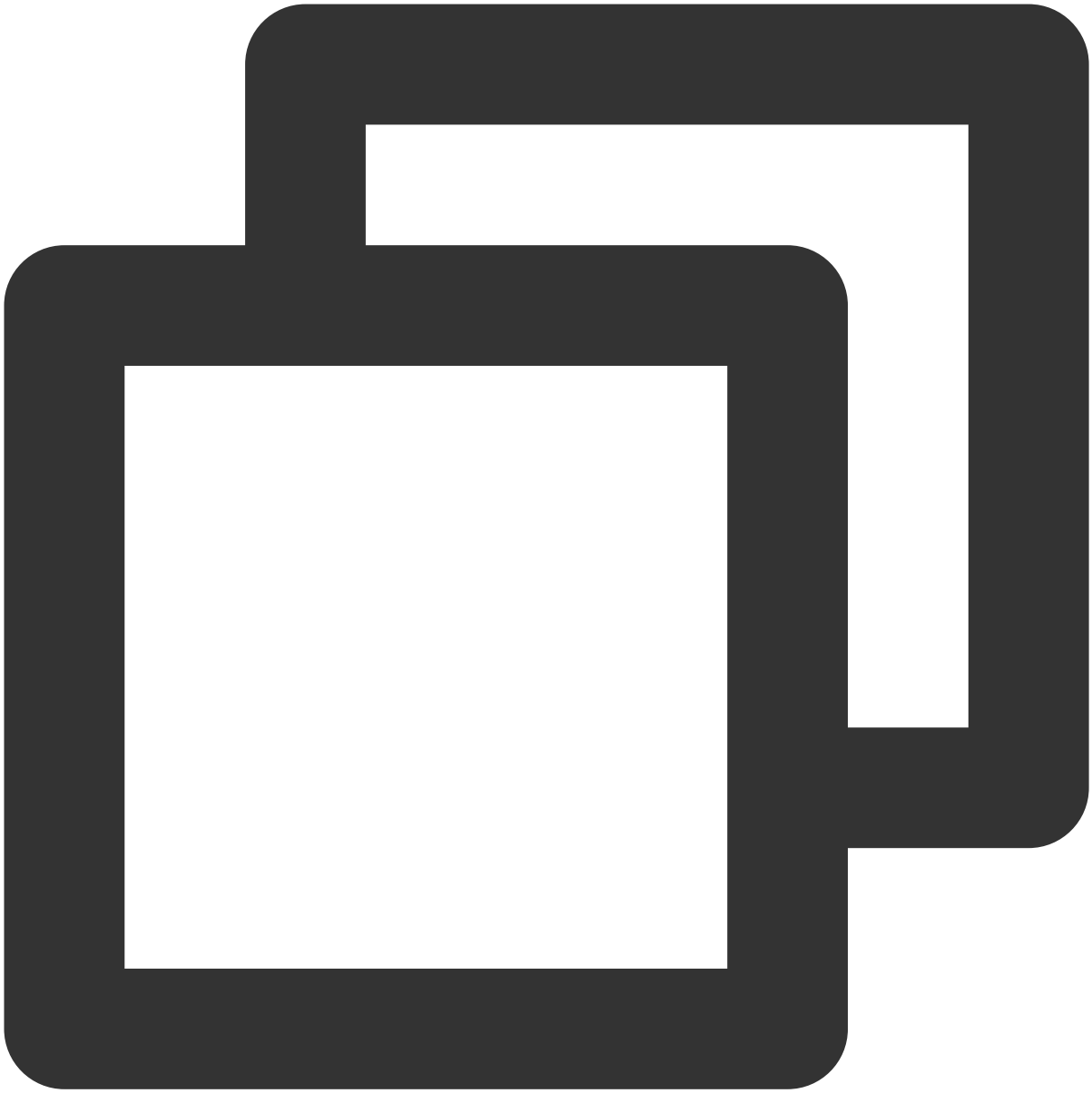
{ itxxxId:1, alxxxId:1 } 和 { itxxxId :1} 这两个索引，{ itxxxId :1} 即为重复索引。

包含关系引起的索引重复。



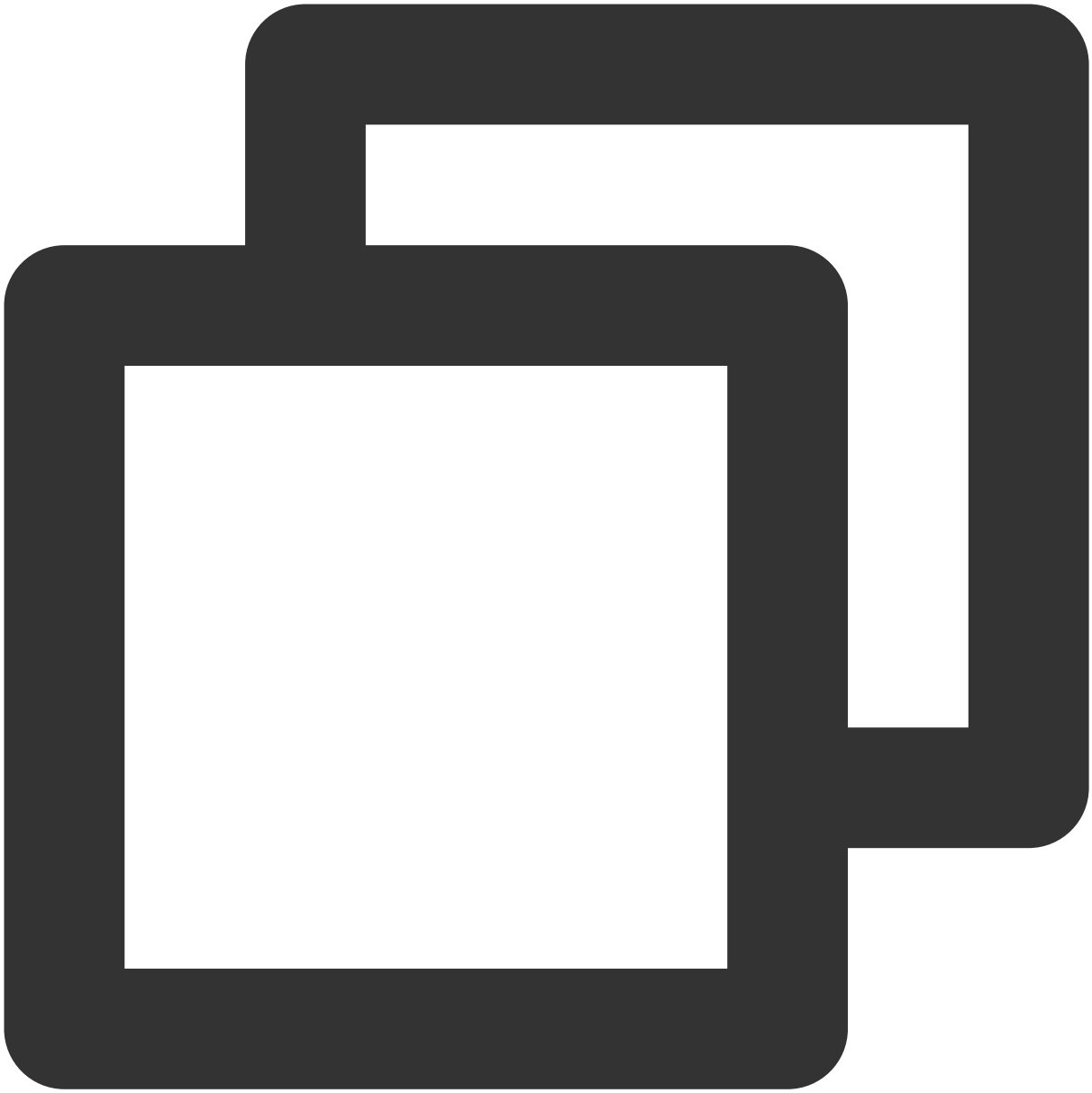
```
{ "alxxxId" : 1, "parentItxxxId" : 1, "parentAlxxxId" : 1, "state" : 1 }
{ "alxxxId" : 1, "parentAlxxxId" : 1, "state" : 1 }
{ "alxxxId" : 1, " state " : 1 }
```

这三个索引，存在如下三个查询：



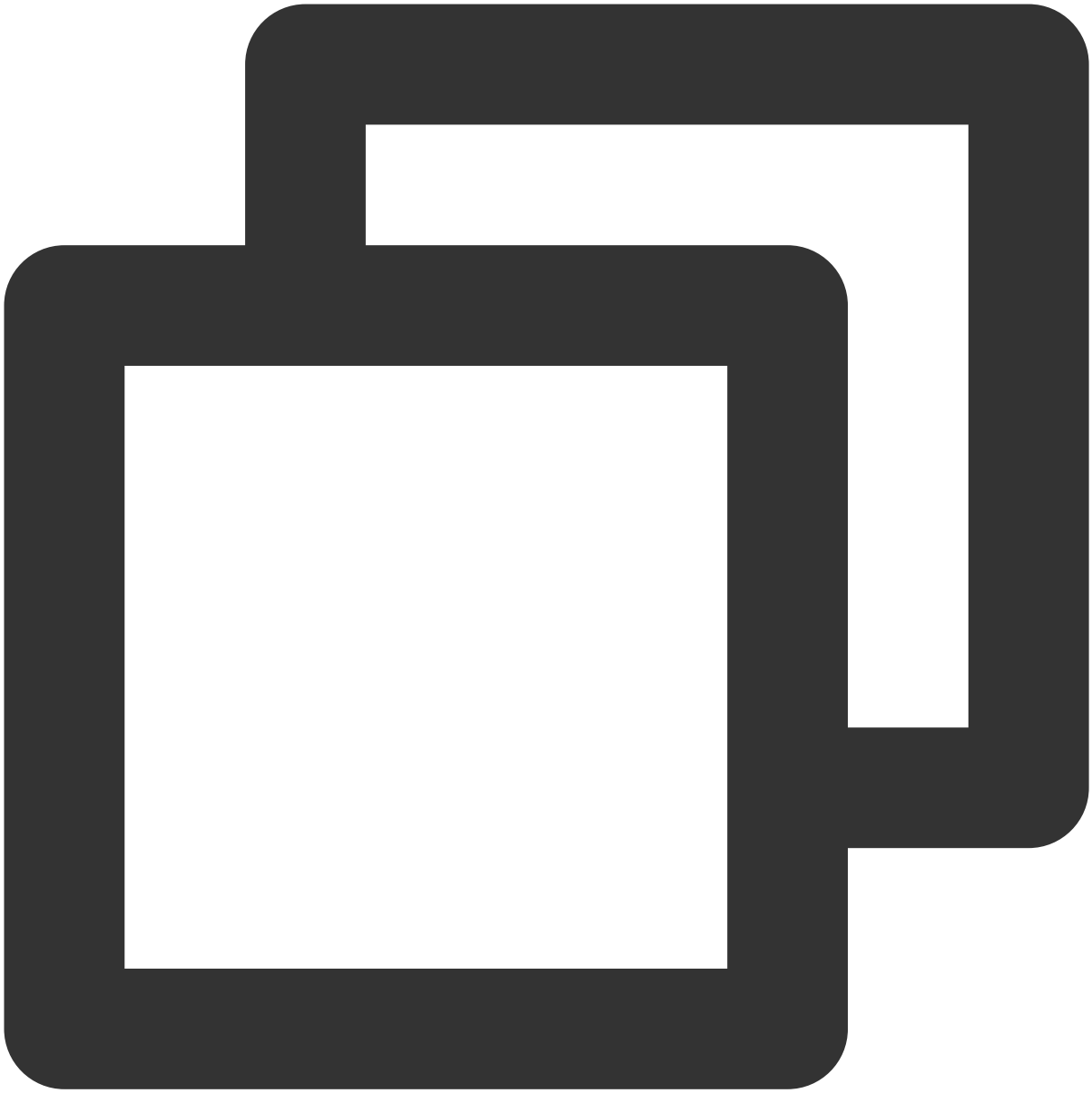
```
Db.xxx.find({ "alxxxId" : xxx, "parentItxxxId" : xx, "parentAlxxxId" : xxx, "state"
Db.xxx.find({ "alxxxId" : xxx, " parentAlxxxId " : xx, " state " : xxx })
Db.xxx.find({ "alxxxId" : xxx,  " state " : xxx })
```

这几个查询都包含公共字段，因此可以合并为一个索引来满足这两类 SQL 的查询，合并后的索引如下：



```
{ "alxxxId" : 1, " state " : 1, " parentAlxxxId " : 1, parentItxxxId :1}
```

重复索引，经合并清理，可保留如下2个索引。

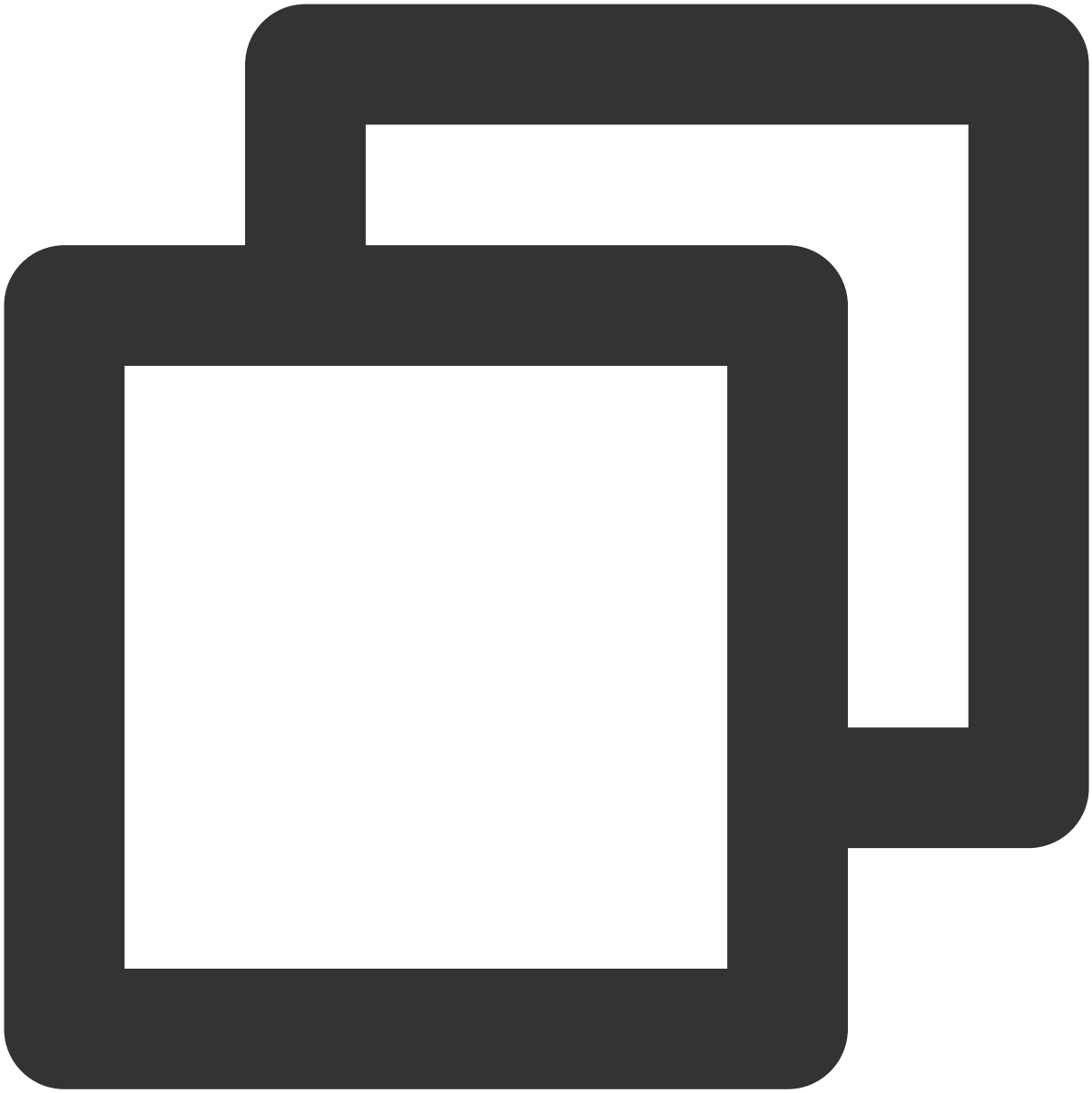


```
{ itxxxId:1, alxxxId:1 }
{ "alxxxId" : 1, "parentItxxxId" : 1, "parentAlxxxId" : 1, "state" : 1 }
```

分析索引唯一性，去除重合索引

分析表中数据各个字段模块组合，可发现 alxxxId 和 itxxxId 字段为高频字段。分析 Schema 信息，随机抽取一部分数据，发现这两个字段组合是唯一的。

经确认，这两个字段的任意组合都代表一条唯一的数据。那么，这两个字段和任何字段的组合都是唯一的。因此，下面的几个索引可以合并为一个索引：{ itxxxId:1, alxxxId:1 }。

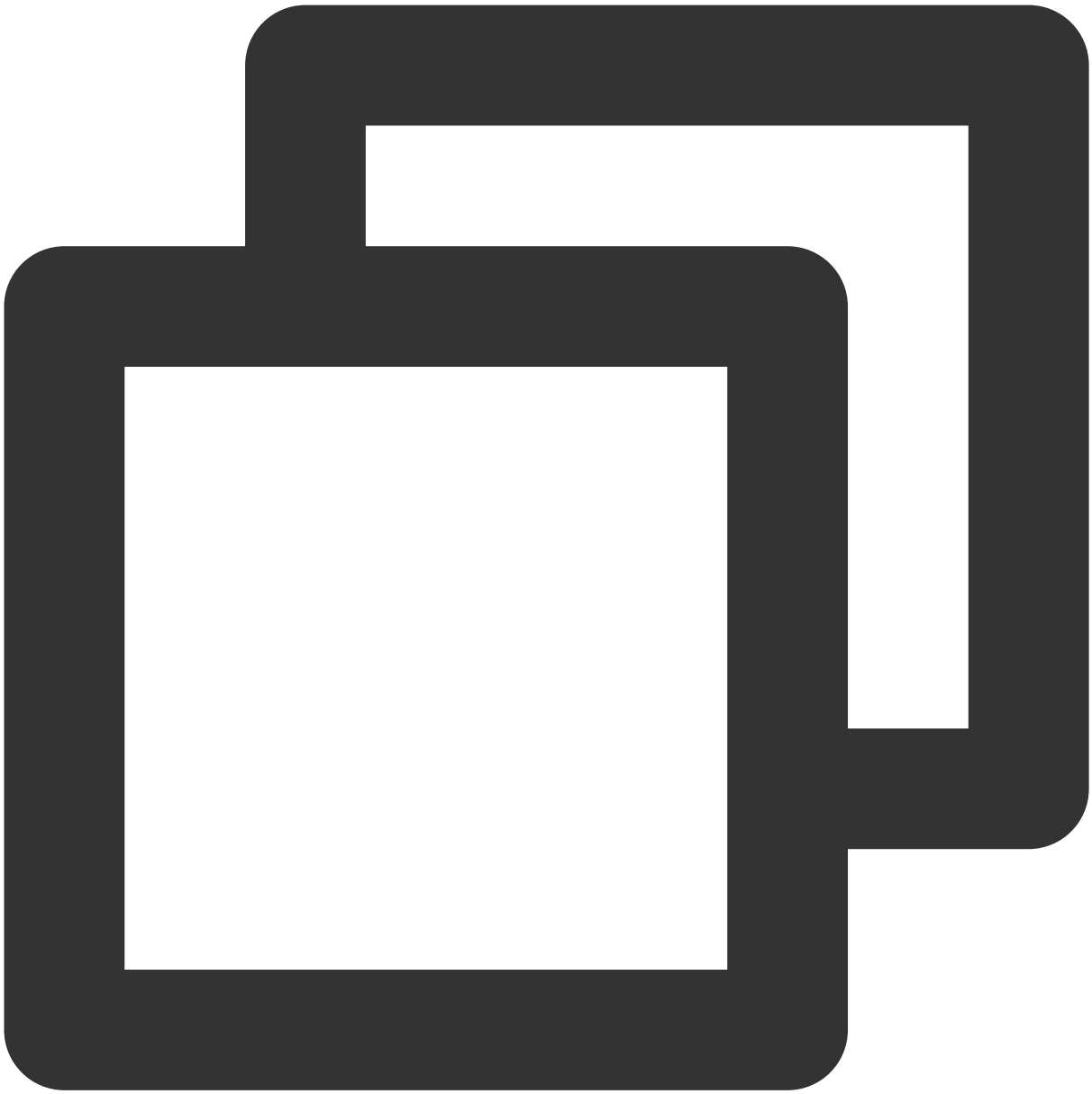


```
{ "alxxxId" : 1, "state" : -1, "updateTime" : -1, "itxxxId" : 1, "persxxal" : 1, "s
{ "alxxxId" : 1, "state" : -1, "itemType" : 1, "persxxal" : 1, " itxxxId " : 1 }
{ "alxxxId" : 1, "state" : -1, "newsendTime" : -1, "itxxxId" : 1, "persxxal" : 1 }
{ "alxxxId" : 1, "state" : 1, "itxxxId" : 1, "updateTime" : -1 }
{ itxxxId:1, alxxxId:1 }
```

非等值查询引起的无用索引优化

从前面的30个索引可以看出，索引中有部分为时间类型字段，如 createTime、updateTime，经确认，这些字段用于各种范围查询。范围查询属于非等值查询，如果范围查询字段出现在索引字段前面，则后面字段无法走索引，如下

所示。

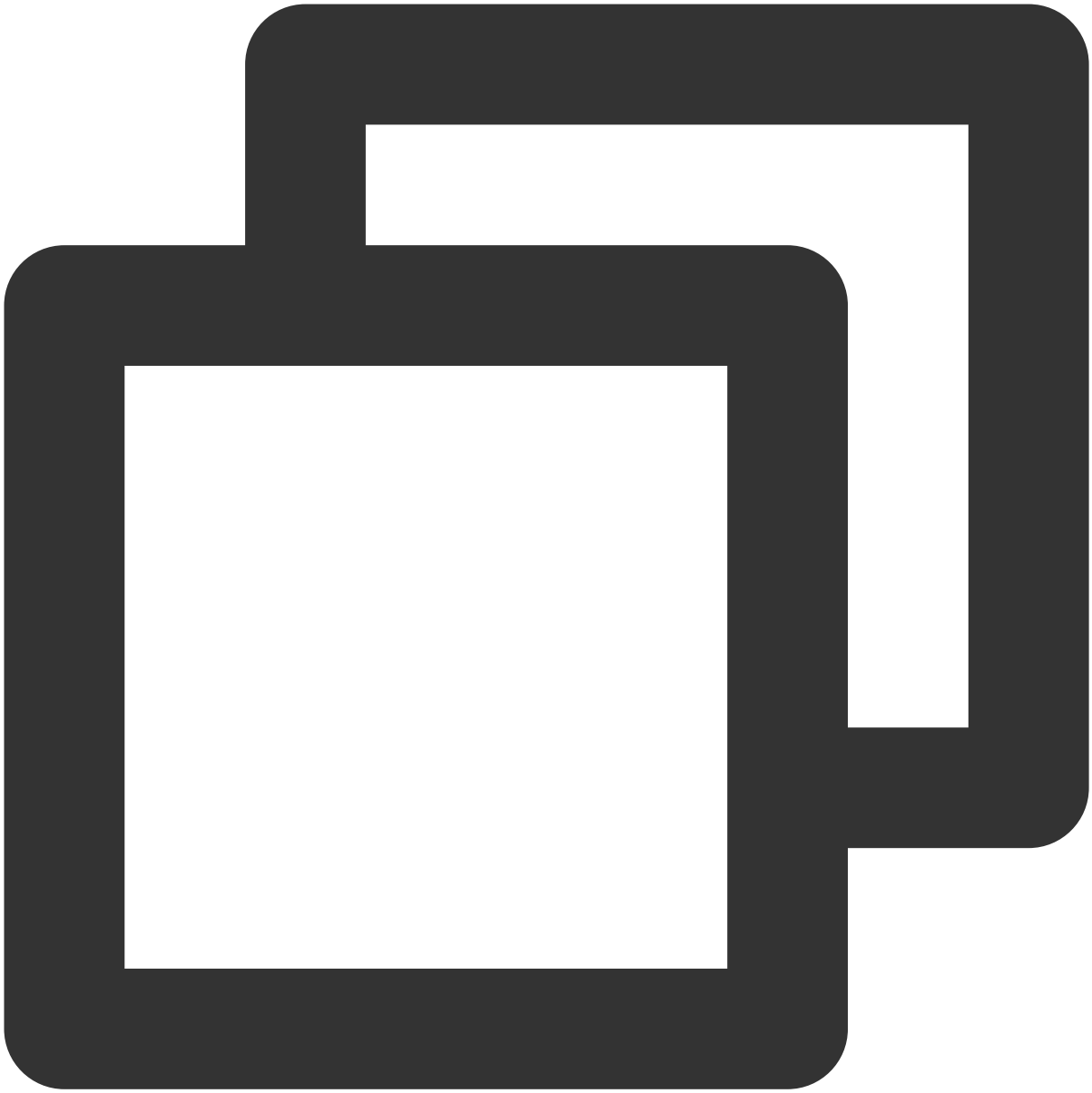


```
db.collection.find({{ "alxxxId" : xx, "parentItxxxId" : xx, "state" : xx, "updateTi

db.collection.find({{ "alxxxId" : xx, "state" : xx, "parentItxxxId" : xx, "updateTi
```

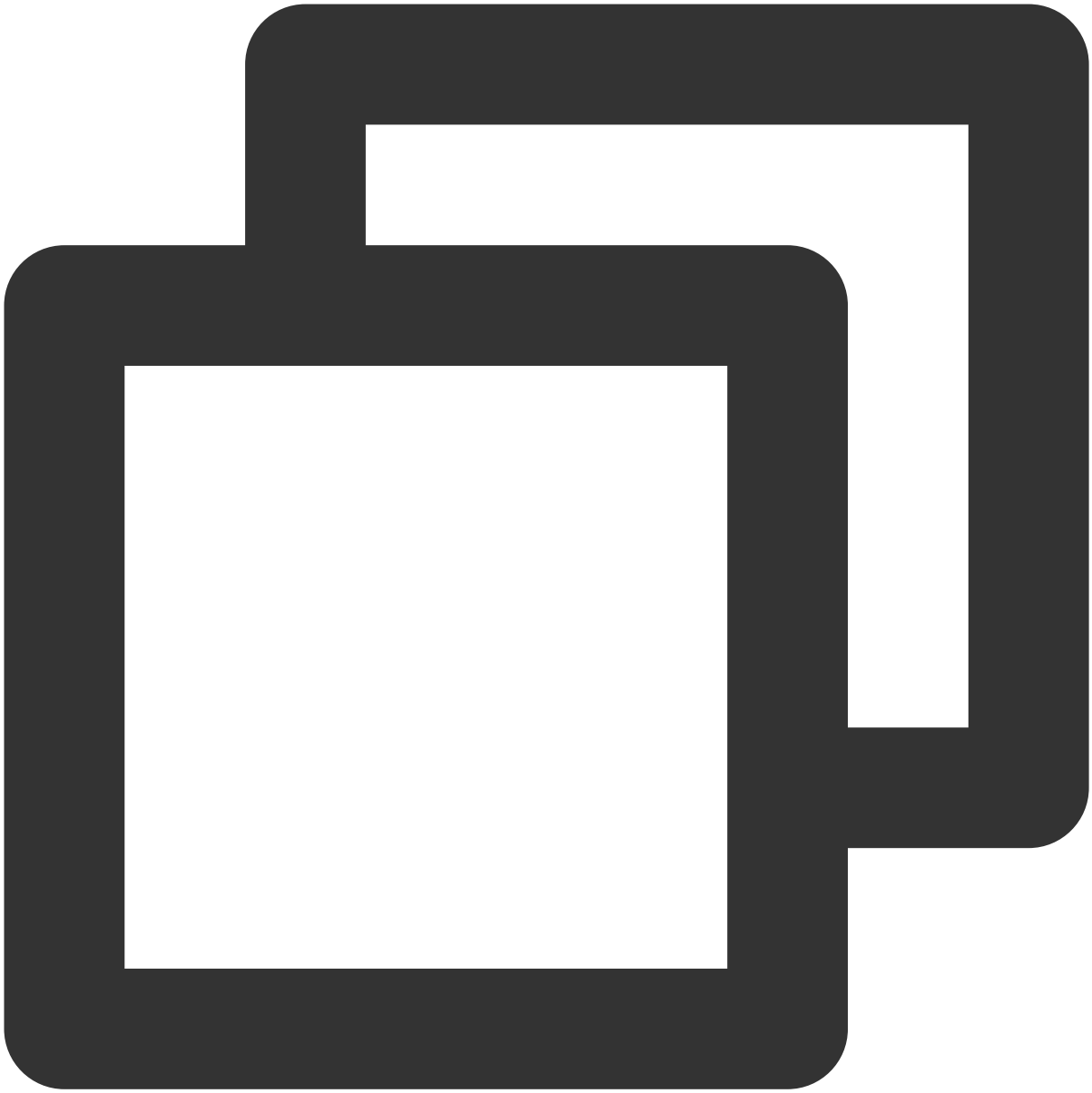
这两个查询都包含 updateTime 字段，并进行范围查询。除了 updateTime 字段以外的字段都是等值查询，updateTime 右边的字段无法使用索引，也第一个索引 persxxal 和 srcItxxxId 字段无法匹配索引，第二个索引 persxxal 字段无法匹配索引。

用户为这两个查询设置以下两个索引。



```
{ "alxxxId" : 1, "parentItxxxId" : -1, "state" : -1, "updateTime" : -1, "persxxal"
{ "alxxxId" : 1, "state" : -1,  "parentItxxxId" : 1, "updateTime" : -1, "persxxal"
```

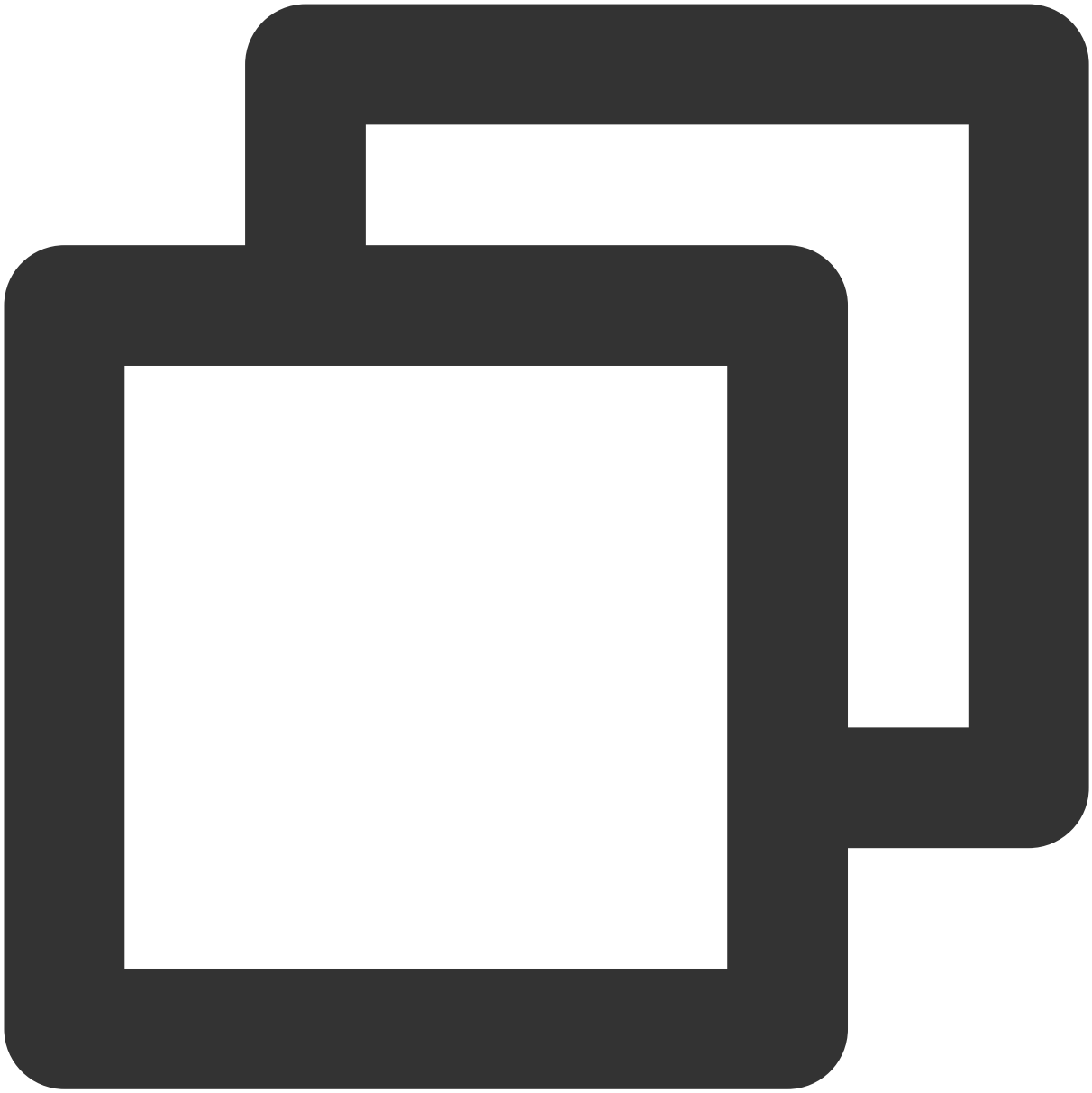
这两个索引字段基本相同，可优化为如下一个索引，确保更多字段能够匹配到索引。



```
{ "alxxxId" : 1, "state" : -1,  "parentItxxxId" : 1,  "persxxal" : -1, "updateTime"
```

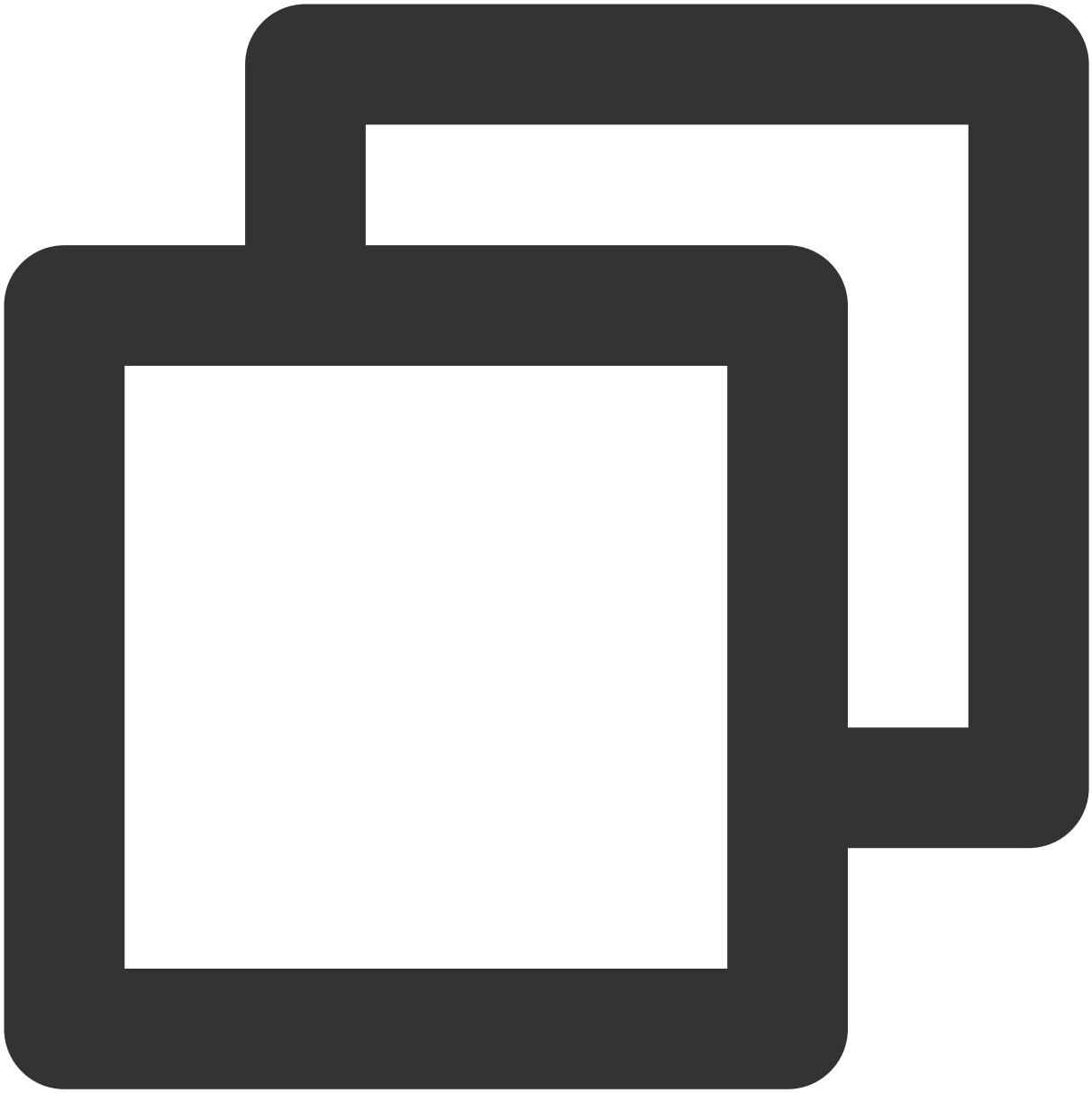
去除查询频率较低字段对应的索引

删除无用索引时，清理掉命中率低于10000次以下的索引。但是，还有一部分索引相比高频命中次数（数十亿次）命中次数，也相对较低（命中次数只有几十万）。这部分较低频命中次数的索引如下所示，分别包含 image 与 videoCover 字段。



```
{ "alxxxId" : 1, "image" : 1 }          "ops" : NumberLong(293104)
{ "alxxxId" : 1, "videoCover" : 1 }     "ops" : NumberLong(292857)
```

登录 MongoDB 控制台，在**慢日志查询**页签，将慢日志时延阈值调低，分析这两个查询对应日志，如下所示。



```
Mon Aug  2 10:56:46.533 I COMMAND  [conn5491176] command xxxx.tbxxxxx command: coun

Mon Aug  2 10:47:53.262 I COMMAND  [conn10428265] command xxxx.tbxxxxx command: fin
```

Image 字段：分析日志，可发现用户请求中的 image 都是和 alxxxId，itxxxId 进行组合查询，而 alxxxId，itxxxId 组合是唯一的，image 字段完全没有被索引，则 { "alxxxId" : 1, "ixxxge" : 1 } 索引可以删除。

videoCover 字段：分析日志，发现查询条件中没有携带 videoCover，只是部分查询匹配 { alxxxId: 1, videoCover: 1 } 索引，并且 keysExamined、docsExamined 与 nreturned 不相同，可确认，实际只匹配了 alxxxId 索引字段。因此，该索引 { alxxxId: 1, videoCover: 1 } 也可以删除。

分析日志高频查询，添加高频查询最优索引

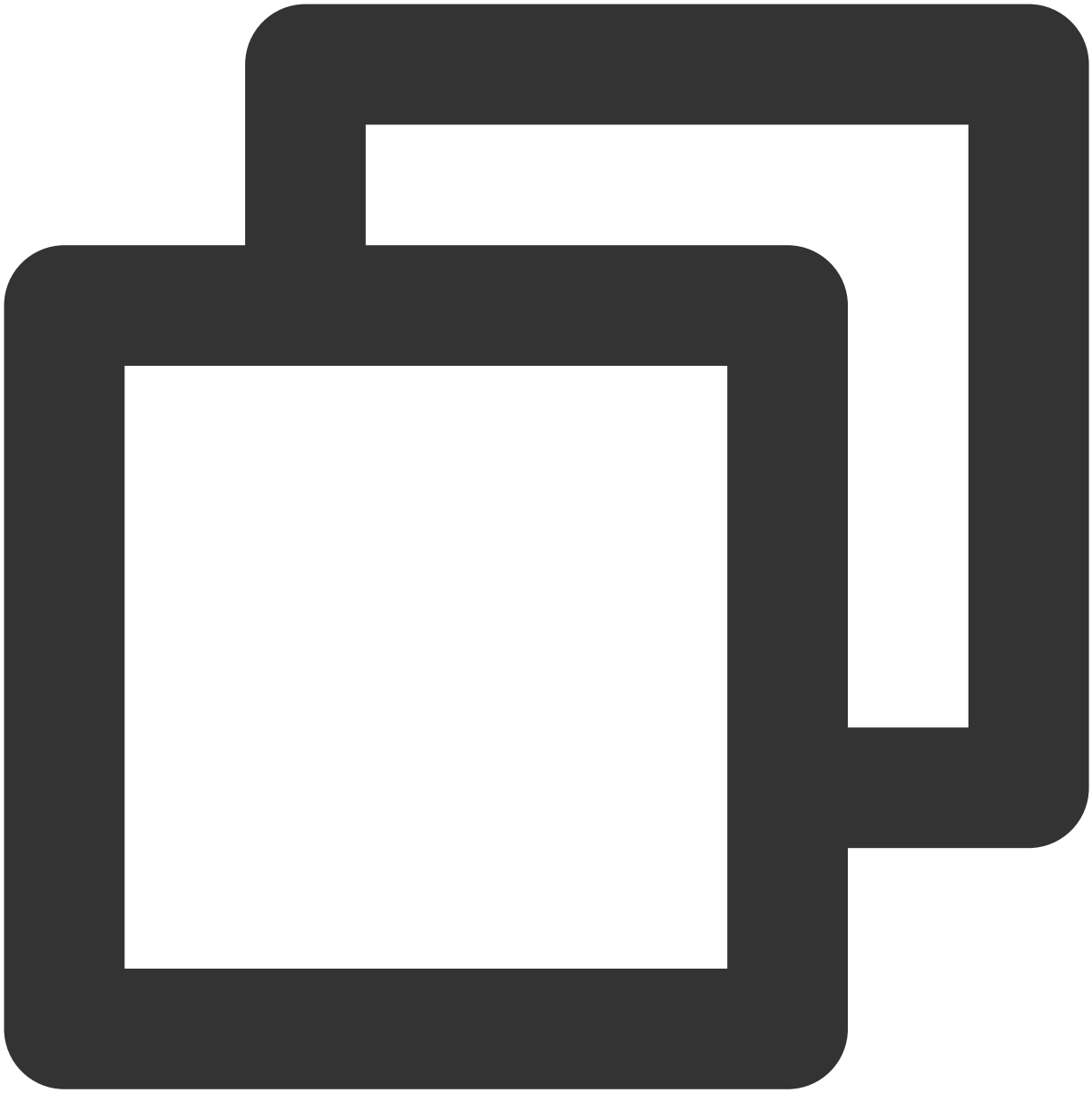
登录 MongoDB 控制台，在**慢日志查询**页签，将慢日志时延阈值调低，通过 mtools 工具分析一段时间的查询，获取到如下热点查询信息：

```
     source: xxx_slow.log
       host: unknown
      start: 2021 Aug 03 15:54:20.884
        end: 2021 Aug 04 11:13:19.295
date format: ctime
   timezone: UTC
     length: 236205
     binary: unknown
    version: >= 2.4.x ctime (milliseconds present)
    storage: unknown

QUERIES
namespace        operation    pattern

[conn5050235]    find         {"$and": [{"alxxxId": 1}, {"state": 1}, {"itexxagList": 1}, {"persxxal": 1}]}
[conn4352017]    find         {"$and": [{"alxxxId": 1}, {"state": 1}, {"itexxagList": 1}, {"persxxal": 1}]}
[conn4923398]    find         {"$and": [{"alxxxId": 1}, {"state": 1}, {"itexxagList": 1}, {"persxxal": 1}]}
[conn5050236]    find         {"$and": [{"alxxxId": 1}, {"state": 1}, {"itexxagList": 1}, {"persxxal": 1}]}
[conn4478402]    find         {"$and": [{"alxxxId": 1}, {"state": 1}, {"itexxagList": 1}, {"persxxal": 1}]}
[conn5051636]    find         {"$and": [{"alxxxId": 1}, {"state": 1}, {"itexxagList": 1}, {"persxxal": 1}]}
[conn5050288]    find         {"$and": [{"alxxxId": 1}, {"state": 1}, {"itexxagList": 1}, {"persxxal": 1}]}
[conn4766912]    find         {"$and": [{"alxxxId": 1}, {"state": 1}, {"itexxagList": 1}, {"persxxal": 1}]}
[conn4923401]    find         {"$and": [{"alxxxId": 1}, {"state": 1}, {"itexxagList": 1}, {"persxxal": 1}]}
[conn5050237]    find         {"$and": [{"alxxxId": 1}, {"state": 1}, {"itexxagList": 1}, {"persxxal": 1}]}
[conn4725774]    find         {"$and": [{"alxxxId": 1}, {"state": 1}, {"itexxagList": 1}, {"persxxal": 1}]}
[conn4017540]    find         {"$and": [{"alxxxId": 1}, {"state": 1}, {"itexxagList": 1}, {"persxxal": 1}]}
[conn5050308]    find         {"$and": [{"alxxxId": 1}, {"state": 1}, {"itexxagList": 1}, {"persxxal": 1}]}
[conn5050239]    find         {"$and": [{"alxxxId": 1}, {"state": 1}, {"itexxagList": 1}, {"persxxal": 1}]}
[conn4725778]    find         {"$and": [{"alxxxId": 1}, {"state": 1}, {"itexxagList": 1}, {"persxxal": 1}]}
[conn4923396]    find         {"$and": [{"alxxxId": 1}, {"state": 1}, {"itexxagList": 1}, {"persxxal": 1}]}
[conn4923404]    find         {"$and": [{"alxxxId": 1}, {"state": 1}, {"itexxagList": 1}, {"persxxal": 1}]}
[conn4263629]    find         {"$and": [{"alxxxId": 1}, {"state": 1}, {"itexxagList": 1}, {"persxxal": 1}]}
[conn3852107]    find         {"$and": [{"alxxxId": 1}, {"state": 1}, {"itexxagList": 1}, {"persxxal": 1}]}
[conn4017582]    find         {"$and": [{"alxxxId": 1}, {"state": 1}, {"itexxagList": 1}, {"persxxal": 1}]}
[conn5004287]    find         {"$and": [{"alxxxId": 1}, {"state": 1}, {"itexxagList": 1}, {"persxxal": 1}]}
[conn5004325]    find         {"$and": [{"alxxxId": 1}, {"state": 1}, {"itexxagList": 1}, {"persxxal": 1}]}
[conn5049548]    find         {"$and": [{"alxxxId": 1}, {"state": 1}, {"itexxagList": 1}, {"persxxal": 1}]}
[conn4867278]    find         {"$and": [{"alxxxId": 1}, {"state": 1}, {"itexxagList": 1}, {"persxxal": 1}]}
```

这部分高频热点查询几乎占用了99%以上的查询。分析该类查询对应日志，得到如下信息。

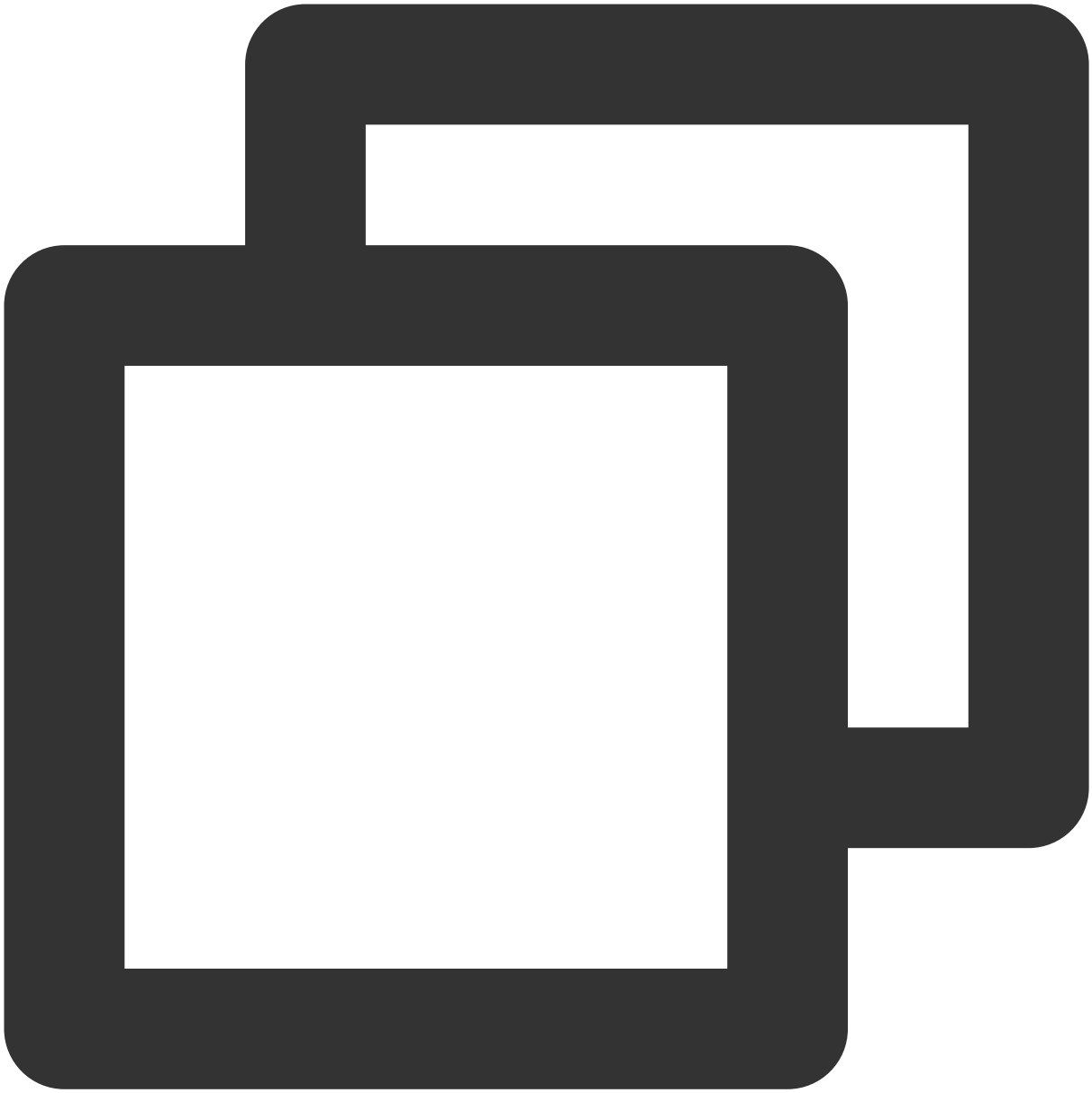


```
Mon Aug  2 10:47:58.015 I COMMAND  [conn4352017] command xxxx.xxx command: find { f
```

分析日志可以看出，该高频查询匹配 { alxxxId: 1.0, itexxagList: 1.0 } 索引，扫描数据行数和最终返回的数据行数差距很大，扫描了1327行，最终只获取到了3条数据。

该索引不是最优索引，该高频查询是四字段的等值查询，只有两个字段走了索引，可以把该索引优化为如下索引 { alxxxId: 1.0, itexxagList: 1.0 , persxxal:1.0, stat:1.0}。

此外，从日志可以看出，该高频查询实际上还有个 sort 排序和 limit 限制，整个查询原始 SQL 如下：

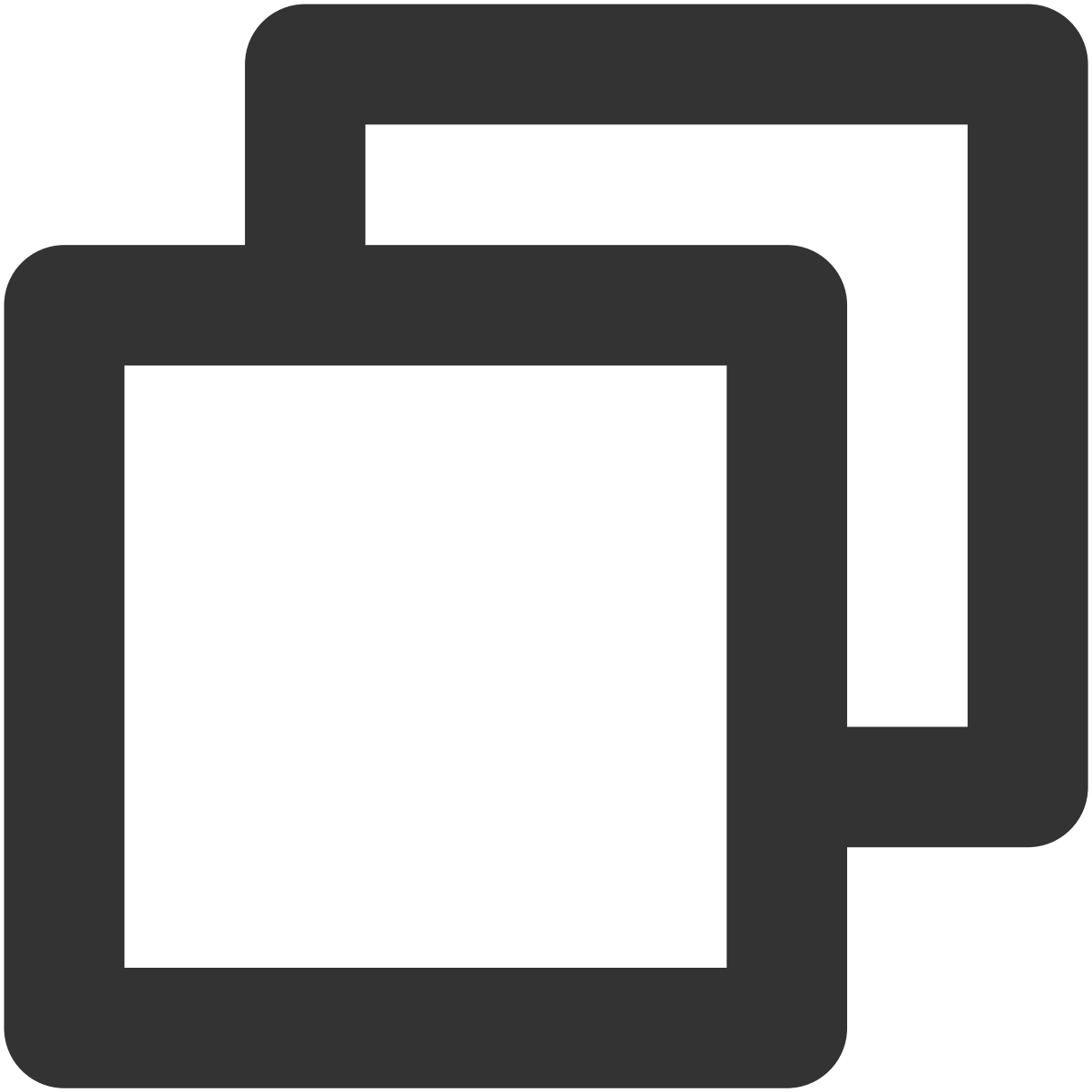


```
db.xxx.find({ $and: [ { alxxxId:"xxxx" }, { state: 0 }, { itexxagList: { $in: [ xxx
```

该查询模型为普通多字段等值查询 + sort 排序类查询 + limit 限制。该类查询最优索引可能是下面两个索引中的一个：

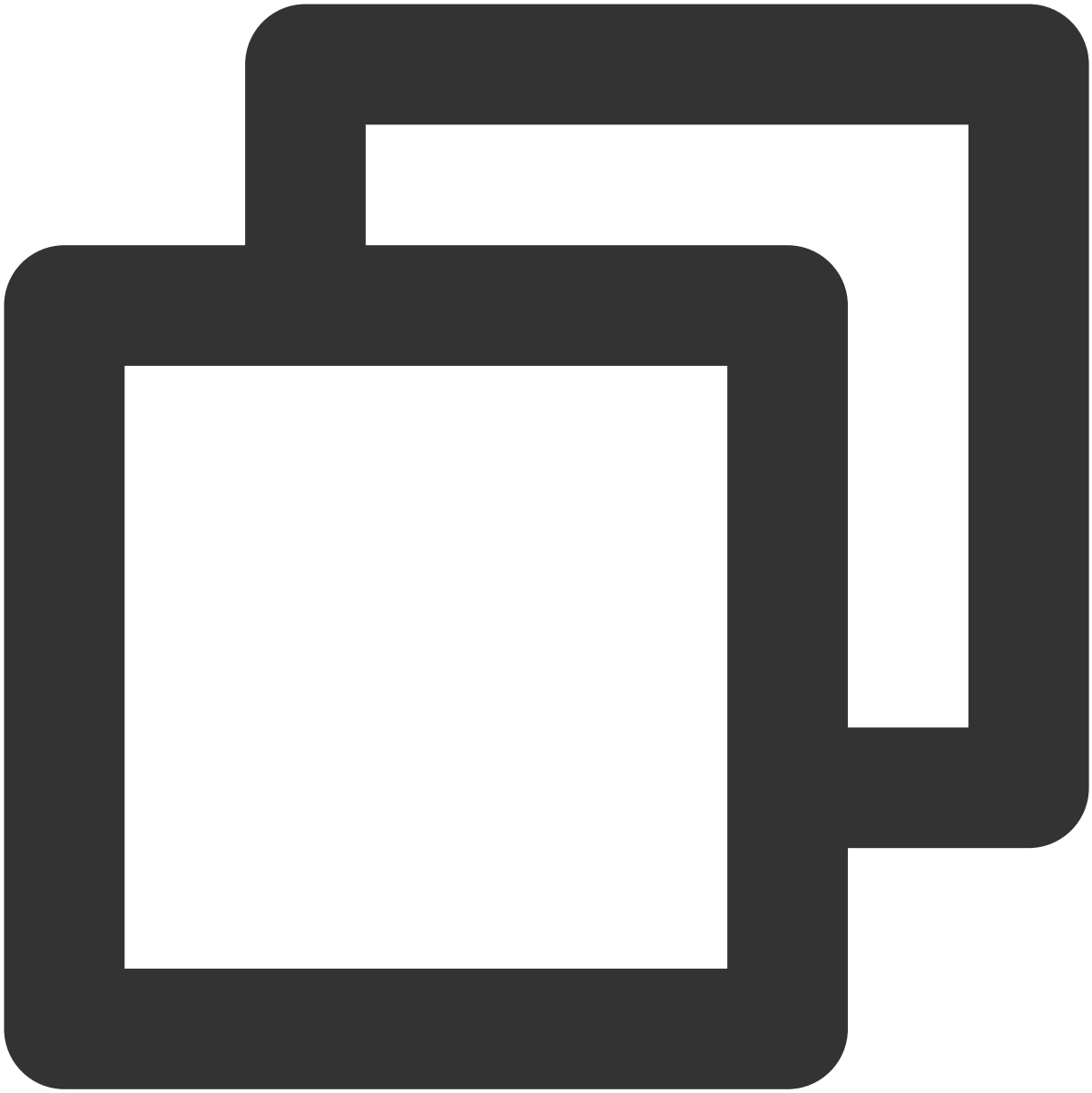
索引1：普通多字段等值查询对应索引

分析其查询条件：



```
{ $and: [ { alxxxId:"xxx" }, { state: 0 }, { itexxagList: { $in: [ xxxx ] } }, { pe
```

该 SQL 四个字段都为等值查询，按照散列度创建最优索引，取值越散列的字段放最左边，可以得到如下最优索引：



```
{ alxxxId: 1.0, itexxagList: 1.0 , persxxal:1.0, stat:1.0}
```

如果选择该索引作为最优索引，则整个普通多字段等值查询 + sort 排序类查询 + limit 限制查询，执行流程如下：

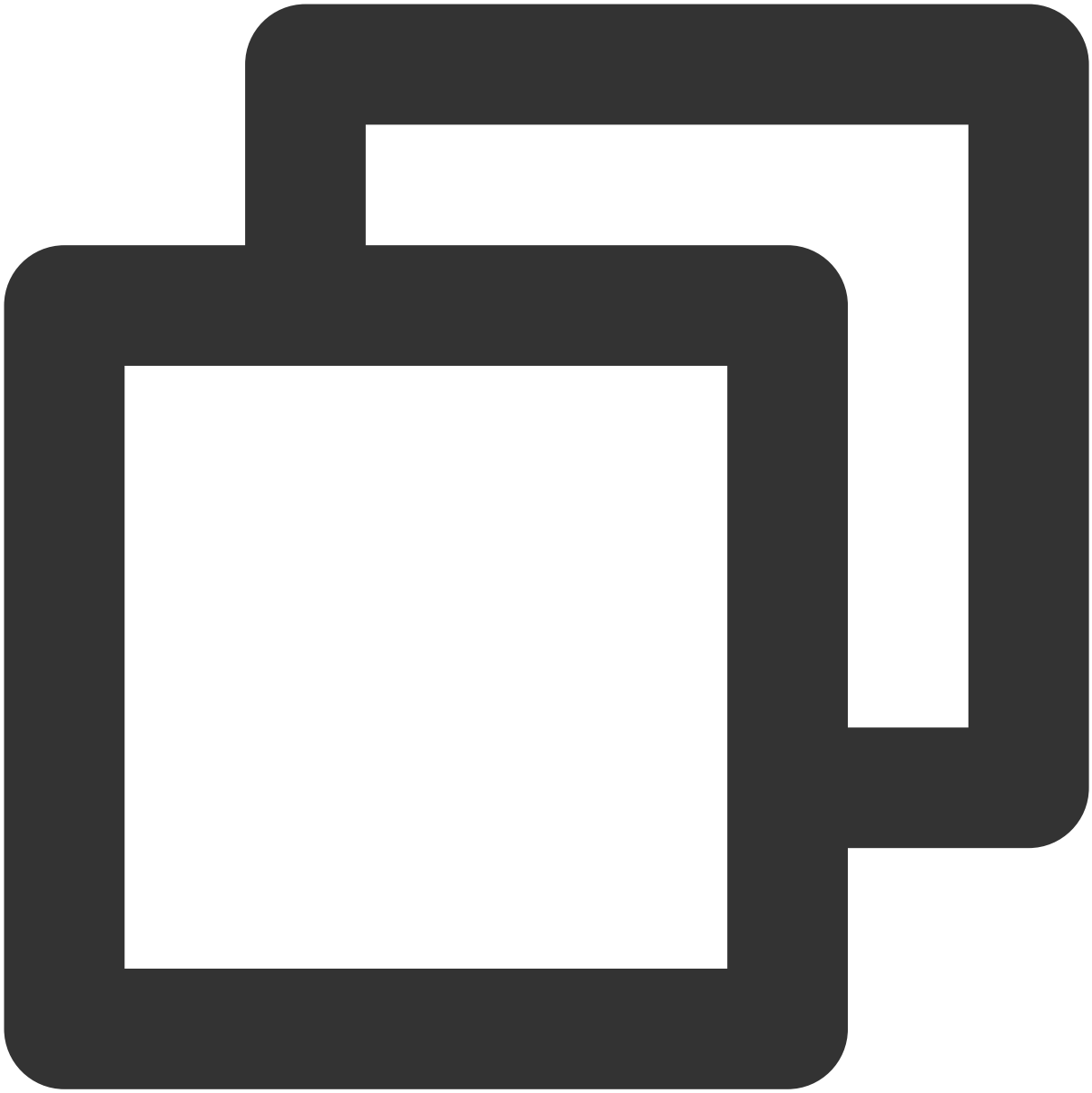
通过 { alxxxId: 1.0, itexxagList: 1.0 , persxxal:1.0, stat:1.0} 索引找出满足 { $and: [{ alxxxId:"xxxx" }, { state: 0 }, { itexxagList: { $in: [xxxx] } }, { persxxal: 0 }] } 条件的所有数据。

对这些满足条件的数据进行内存排序。

取排序好的前三条数据。

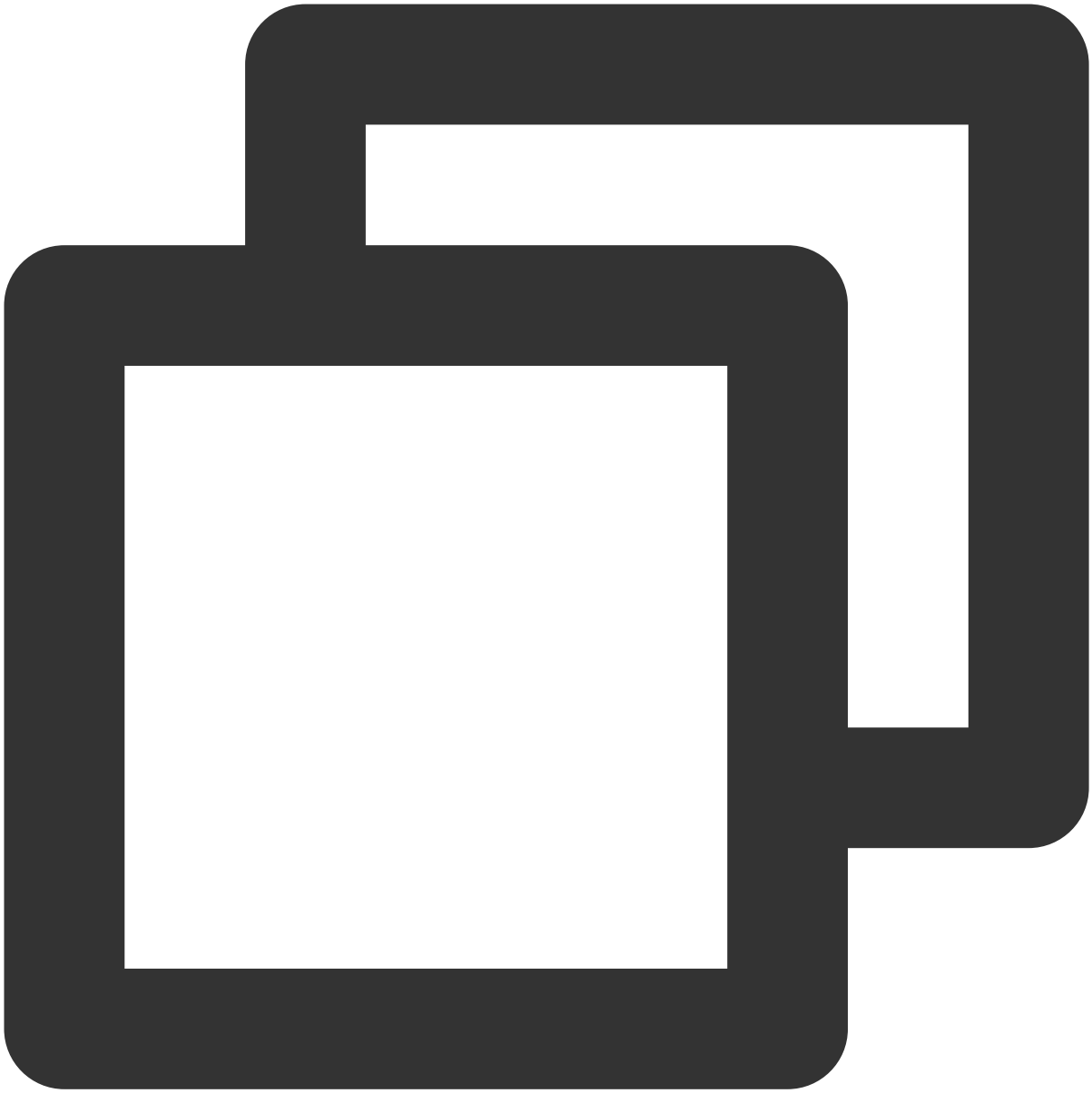
索引2：等值查询 + sort 排序对应最优索引

sort 排序查询中带有 limit，找出该高频排序 SQL，如下：



```
{ $and: [ { alxxxId:"xxxx" }, { state: 0 }, { itexxagList: { $in: [ xxxx ] } }, { p
```

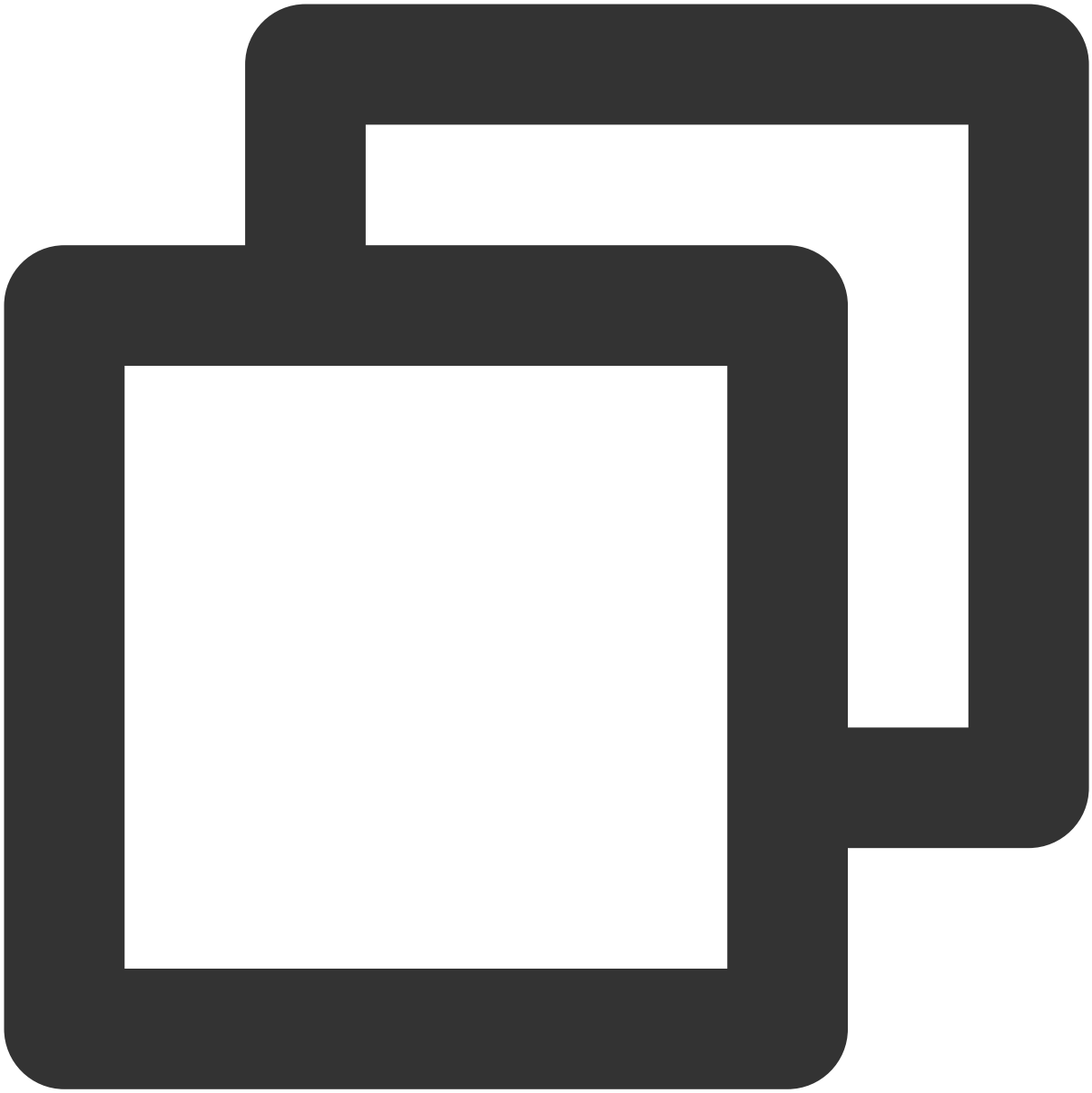
该查询为超高频查询，建议这类 SQL 添加如下索引：



```
{ alxxxId: 1.0, itexxagList: 1.0 , persxxal:1.0, stat:1.0, updateTime ：1}
```

步骤4：梳理最终保留的索引

通过以上优化，保留如下索引。



```
{ "itxxxId" : 1, "alxxxId" : 1 }
{ "alxxxId" : 1, "state" : 1, "digitalxxxrmarkId" : 1, "updateTime" : 1 }
{ "alxxxId" : 1, "state" : -1,  "parentItxxxId" : 1, "persxxal" : -1, "updateTime"
{ "alxxxId" : 1, "parentAlxxxId" : 1, "state" : 1"parentItxxxId" : 1}
{ alxxxId: 1.0, itexxagList: 1.0 , persxxal:1.0, stat:1.0, updateTime:1}
{ "alxxxId" : 1, "createTime" : -1}
```

索引优化后收益

CPU 资源节省90%以上。

CPU 峰值消耗从之前的90%多降到优化后的10%以内。

磁盘 IO 资源节省85%。

磁盘 IO 消耗从之前的60% - 70%降低到10%以内。

盘存储成本节省20%。

每个索引都对应一个磁盘索引文件，索引从30个减少到8个，数据 + 索引最终真实磁盘消耗减少20%左右。

慢日志减少99%。

索引优化前慢日志每秒数千条，优化后慢日志条数降低到数十条。

分片集群 Mongos 负载不均解析及应对方案

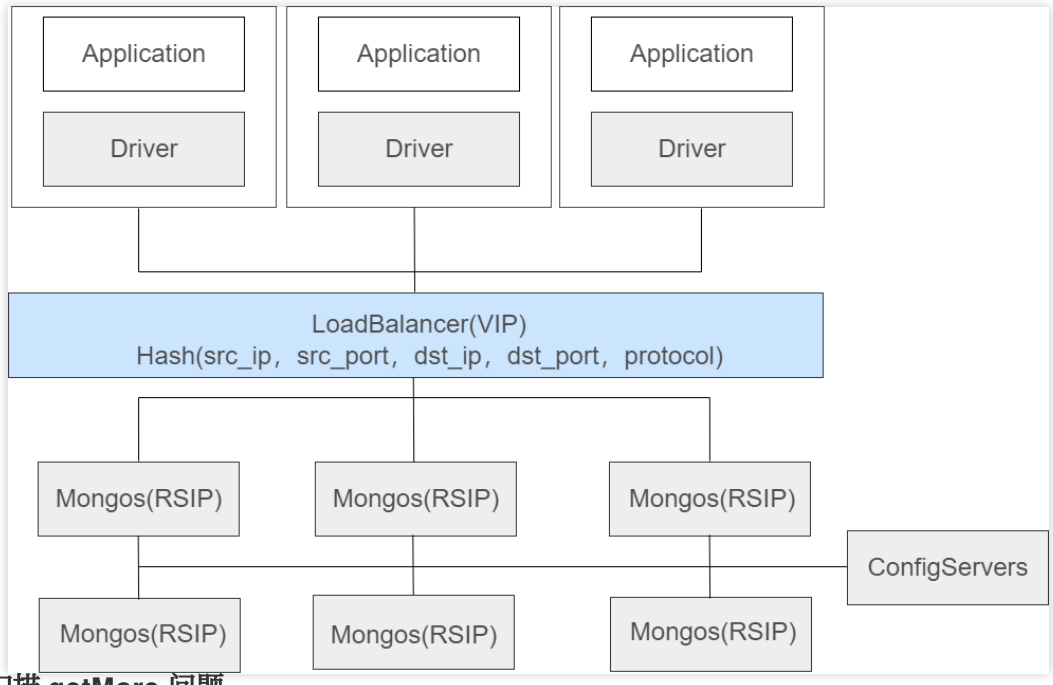
最近更新时间：2024-04-08 11:34:18

业务背景

云数据库 MongoDB 分片集群提供了多个 Mongos 节点，负责接收所有客户端应用程序的连接查询请求，并将请求路由到集群内部对应的分片上，同时会把接收到的响应拼装起来返回到客户端。用户通过一个 VIP 或多个 VIP 等方式接入负载均衡，将流量自动分发至不同的 Mongos 执行，以加强网络数据处理能力、增加吞吐量、提高网络的可用性和灵活性。

负载均衡原理

用户程序连接到一个负载均衡服务（VIP），通过一个 VIP 来屏蔽后端的多个 RSIP（ Real Server IP ）。云数据库 MongoDB 负载均衡服务通过源 IP、源端口、目标 IP、目标端口、通信协议5元组 Hash 策略，将不同请求源 IP 路由于不同的 Mongos 节点。如果后台出现 RSIP 变更，会有自动化的流程变更 VIP 和 RSIP 的映射关系，对用户无感知，接入简单。



批量扫描 getMore 问题

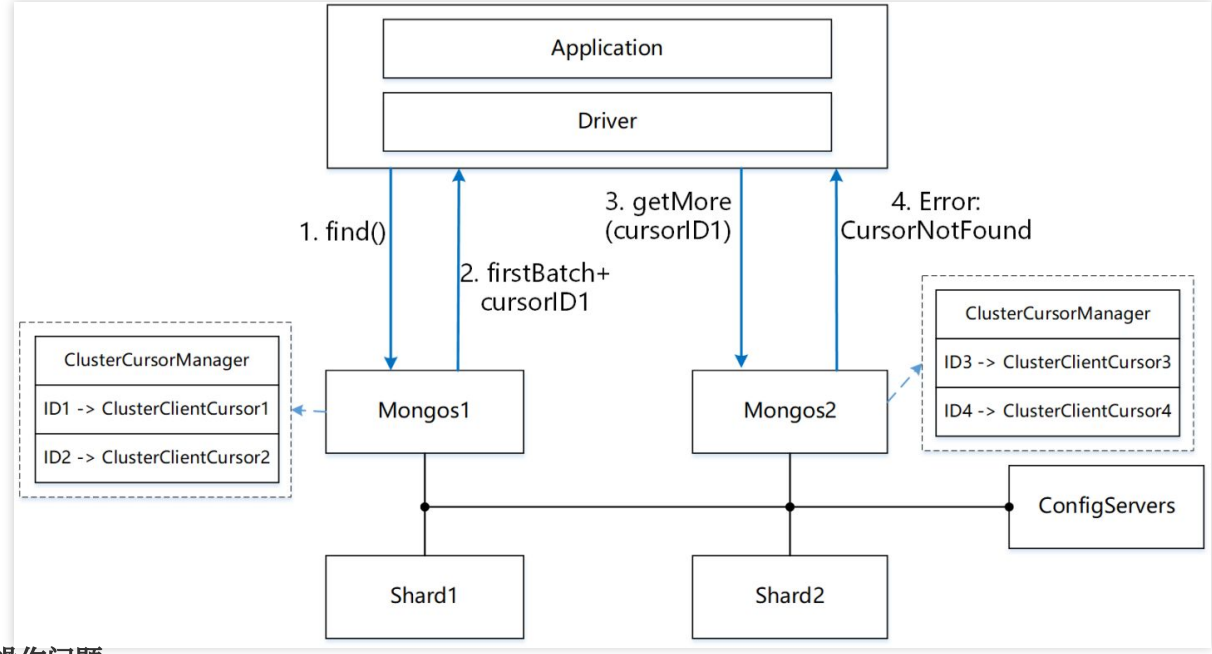
当 MongoDB 无法将 find 结果一次性返回时，会优先返回第一批数据 + cursorID，客户端通过这个 cursorID 不断 getMore 迭代剩余的数据。所以一次批量扫描请求可能会对应1次 find 请求和多次 getMore 请求，并通过 cursorID 关联。

每个 Mongos 节点在内存中维护了一个全局的 ClusterCursorManager，通过 HashMap 维护了 cursorID 和 ClusterClientCusor 的映射关系。其中，cursorID 是一个 int64 的随机数，ClusterClientCursor 则维护了请求的执行计划、当前状态等信息。

如果查询结果不能一次性返回（如超过了16MB限制），则会生成一个非0的 cursorID，并将这个 ID 和 ClusterClientCusor 本身 注册到 ClusterCursorManager 中。

如果客户端需要后续的结果，可以携带前面返回的 cursorID 进行 getMore 请求，Mongos 会 找到之前缓存的 ClusterClientCusor，继续执行查询计划并返回结果。ID 和 cursor 的信息独立存在于各个 Mongos 节点中。

因此必须要保证 find 及其相关联的 getMore 请求发往同一个 Mongos 节点。而如果 getMore 请求发给了其他的 Mongos 节点，会因为找不到 Cursor 返回 CursorNotFound 错误，如下图所示。



事务操作问题

MongoDB 在 4.2 版本支持了分布式事务，用户可以连接 Mongos 节点发起事务操作。在 startTransaction 和 commitTransaction/abortTransaction 之间可以执行多次 读写操作，Mongos 在内存中记录了事务中每次请求携带的 logicalSessionId 和 txnId 等元数据来维护上下文关系。因此，MongoDB 的设计 决定了需要保证事务中的每个操作都发到同一个 Mongos 上执行。

云数据库 MongoDB 负载均衡策略

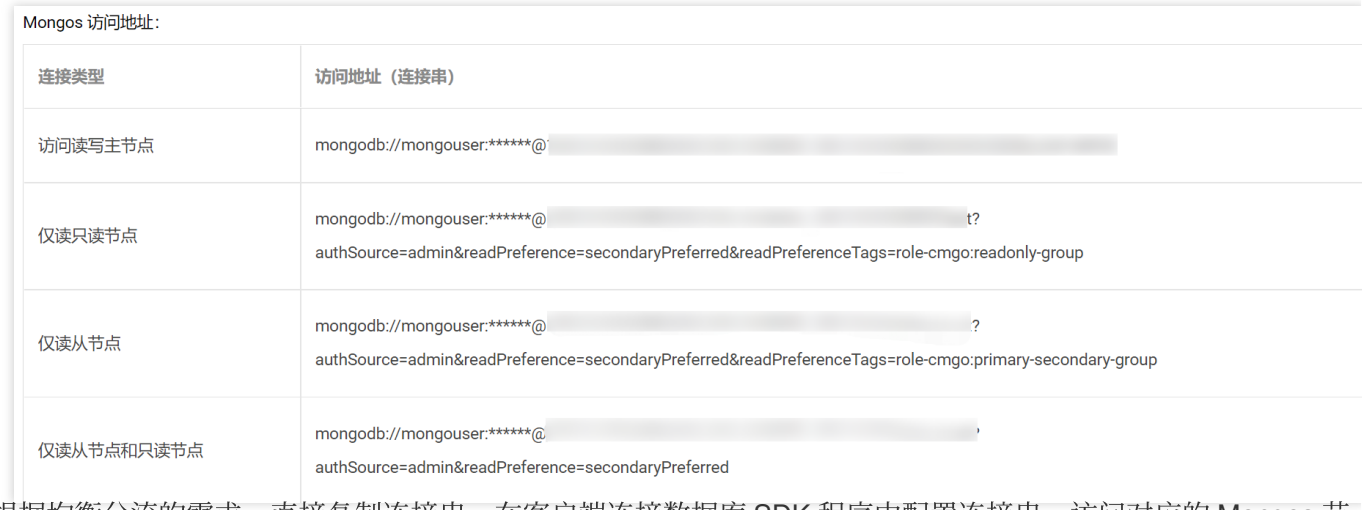
基于批量扫描 getMore 问题及其事务操作问题的考虑，云数据库 MongoDB 负载均衡 Hash 策略根据访问端（一般是 CVM）IP 信息来均衡分流：一个源 IP 的请求都会落在同一个 Mongos 上，保证 getMore() 和事务操作在同一个上下文进行。

一般生产环境中访问端 IP 较多，这种策略效果较好。但是当访问端 IP 较少的情况下，特别是在压测场景下，容易引起 Mongos 负载不均衡的问题。

Mongos 负载不均解决方案

如果您不想使用默认的云数据库 MongoDB 的负载均衡策略，可开通 Mongos 访问地址。在实例当前的 VIP 下面，系统将给不同的 Mongos 节点绑定不同的 VPORT，用户可灵活控制 Mongos 的请求分配。并且，Mongos 故障后系统将重新绑定新的 Mongos 进程，VIP 和 VPORT 地址不会变化，不影响原有的负载均衡访问地址。具体操作，请参见 开通 Mongos 访问地址。

开通之后，在控制台**实例详情**页面的**网络配置**区域的**访问地址**中，可查看 Mongos 访问地址，展示不同连接类型的连接串。每一个连接串中配置了实例所有 Mongos 节点，通过参数 authSource、 readPreference 与 readPreferenceTags 控制访问节点类型，如下图所示。


Mongos 访问地址：

连接类型	访问地址（连接串）
访问读写主节点	mongodb://mongouser:******@
仅读只读节点	mongodb://mongouser:******@ t? authSource=admin&readPreference=secondaryPreferred&readPreferenceTags=role-cmgo:readonly-group
仅读从节点	mongodb://mongouser:******@ ? authSource=admin&readPreference=secondaryPreferred&readPreferenceTags=role-cmgo:primary-secondary-group
仅读从节点和只读节点	mongodb://mongouser:******@ , authSource=admin&readPreference=secondaryPreferred

您可根据均衡分流的需求，直接复制连接串，在客户端连接数据库 SDK 程序中配置连接串，访问对应的 Mongos 节点。具体连接方式，请参见 连接 MongoDB 实例。然而，连接串较长，请仔细操作。

需要注意的是，如果您在连接串中配置所有了 Mongos 节点，当您调整了实例中的 Mongos 节点数后，则需要在应用侧同步更新连接串。

分片集群使用注意事项

最近更新时间：2024-02-20 16:24:04

分片集群为 MongoDB 的分布式版本，相较副本集，分片集群数据被均衡的分布在不同分片中， 不仅大幅提升了整个集群的数据容量上限，也将读写的压力分散到不同分片，以解决副本集性能瓶颈的难题，但分片集群的架构更加复杂，本文重点介绍使用腾讯云 MongoDB 分片集群时的注意事项。

分片集群组件

一个 MongoDB 分片集群由如下三个组件构成，缺一不可：
shard：每个分片是整体数据的一部分子集，每个分片都部署为副本集。
mongos：充当查询路由器，提供客户端应用程序和分片集群之间的接口。
config servers：配置服务器存储集群的元数据和配置，包括权限认证相关。

分片集群 sharding 方式及性能影响

MongoDB 分片集群提供三种 Sharding（数据分布）方式，分别为基于范围、基于 Hash、基于 zone/tag。不同的 Sharding 方式使用不同的业务，也会对性能产生不同的影响。

基于范围

优势：分片键范围查询性能较好，读性能较好。
劣势：数据分布可能不均匀，存在热点。

基于 Hash

优势：数据分布均匀，写性能较好，适用于日志、物联网等高并发场景。
劣势：范围查询效率较低。

基于 zone/tag

若数据具备一些天然的区分，如基于地域、时间等标签，数据可以基于标签来做区分。
优势：数据分布较为合理。

分片键的选择

分片键是文档中的某一个字段，用来进行路由查询。
选择合适的片键对 sharding 效率影响很大，主要基于如下四个因素：

取值基数

取值基数建议尽可能大，如果用小基数的片键，因为备选值有限，那么块的总数量就有限，随着数据增多，块的大小会越来越大，导致水平扩展时移动块会非常困难。

例如：选择年龄做一个基数，范围最多只有100个，随着数据量增多，同一个值分布过多时，导致 chunck 的增长超出 chuncksize 的范围，引起 jumbo chunk，从而无法迁移，导致数据分布不均匀，性能瓶颈。

取值分布

取值分布建议尽量均匀，分布不均匀的片键会造成某些块的数据量非常大，同样有上面数据分布不均匀，性能瓶颈的问题。

查询带分片

查询时建议带上分片，使用分片键进行条件查询时，mongos 可以直接定位到具体分片，否则 mongos 需要将查询分发到所有分片，再等待响应返回。

避免单调递增或递减

单调递增的 sharding key，数据文件挪动小，但写入会集中，导致最后一篇的数据量持续增大，不断发生迁移，递减同理。

综上，在选择片键时要考虑以上4个条件，尽可能满足更多的条件，才能降低 MoveChuncks 对性能的影响，从而获得最优的性能体验。

修改分片健值

MongoDB 4.2 之前的版本，文档的分片键字段值不可变。

从 MongoDB 4.2 版本开始，除非分片键字段是不可变的 `_id` 字段，否则您可以更新文档的分片键值。 若要更新，请使用以下方式来更新文档的分片键值：

命令	方法
update with multi: false	db.collection.replaceOne() db.collection.updateOne() db.collection.update() with multi: false
findAndModify	db.collection.findOneAndReplace() db.collection.findOneAndUpdate() db.collection.findAndModify()
-	db.collection.bulkWrite() Bulk.find.updateOne() 如果分片键修改导致将文档移动到另一个分片，则在批量操作中不能指定多个分片键修改；即批量大小为 `1` 。 如果分片键修改不会导致将文档移动到另一个分片，则可以在批量操作中指定多个分片键修改。

修改分片健时需要注意：

必须在事务中或以可重试写入方式在 mongos 上运行，不要直接在分片上执行操作。

您必须在查询过滤器的完整分片键上包含相等条件。 例如，如果一个分片集合内使用 `{country：1，userid：1}` 作为分片键，要想更新文档的分片键，则必须在查询过滤器中包含 `country：，userid：` ，也可以根据需要在查询中包括其他字段。

分片集群 balance 介绍及相关参数

在一个分片集群内部，MongoDB 会把数据分为 chunks，后台进程 balancer 负责 chunk 的迁移，从而均衡各个 shard server 的负载，每个 chunk 包含一部分数据，chunk 的产生和迁移会导致 balance 的产生。

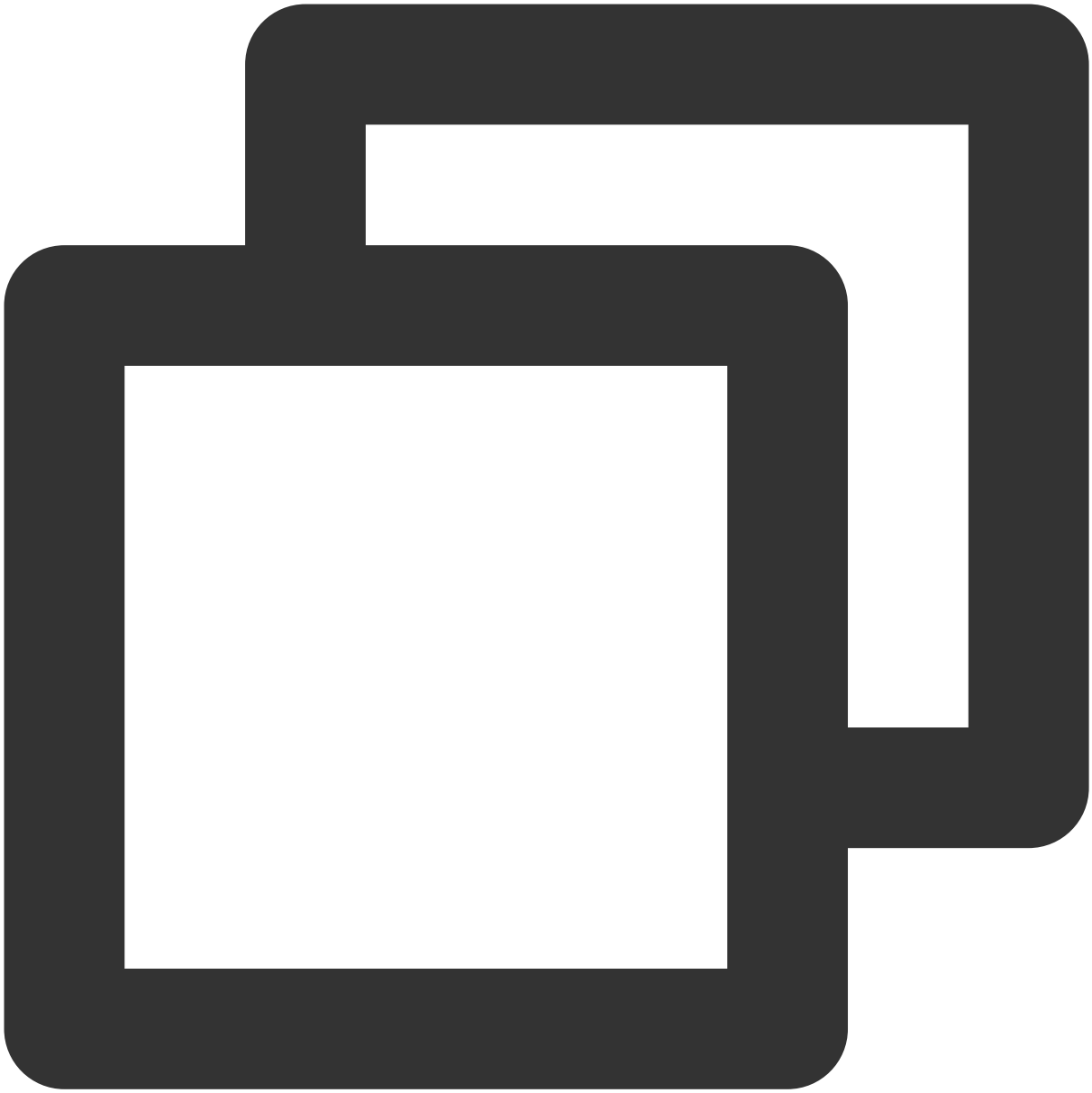
说明：

系统初始仅1个 chunk，chunk size 默认值64MB。

chunck 迁移时会造成集群的读写性能下降，因此需要通过适当配置 balance 活动窗口来避免 balance 对业务高峰期的影响，也可以通过命令来关闭 balance。

下面介绍管理 balance 的相关命令，若某些指令无权限执行，请 提交工单 联系我们处理。

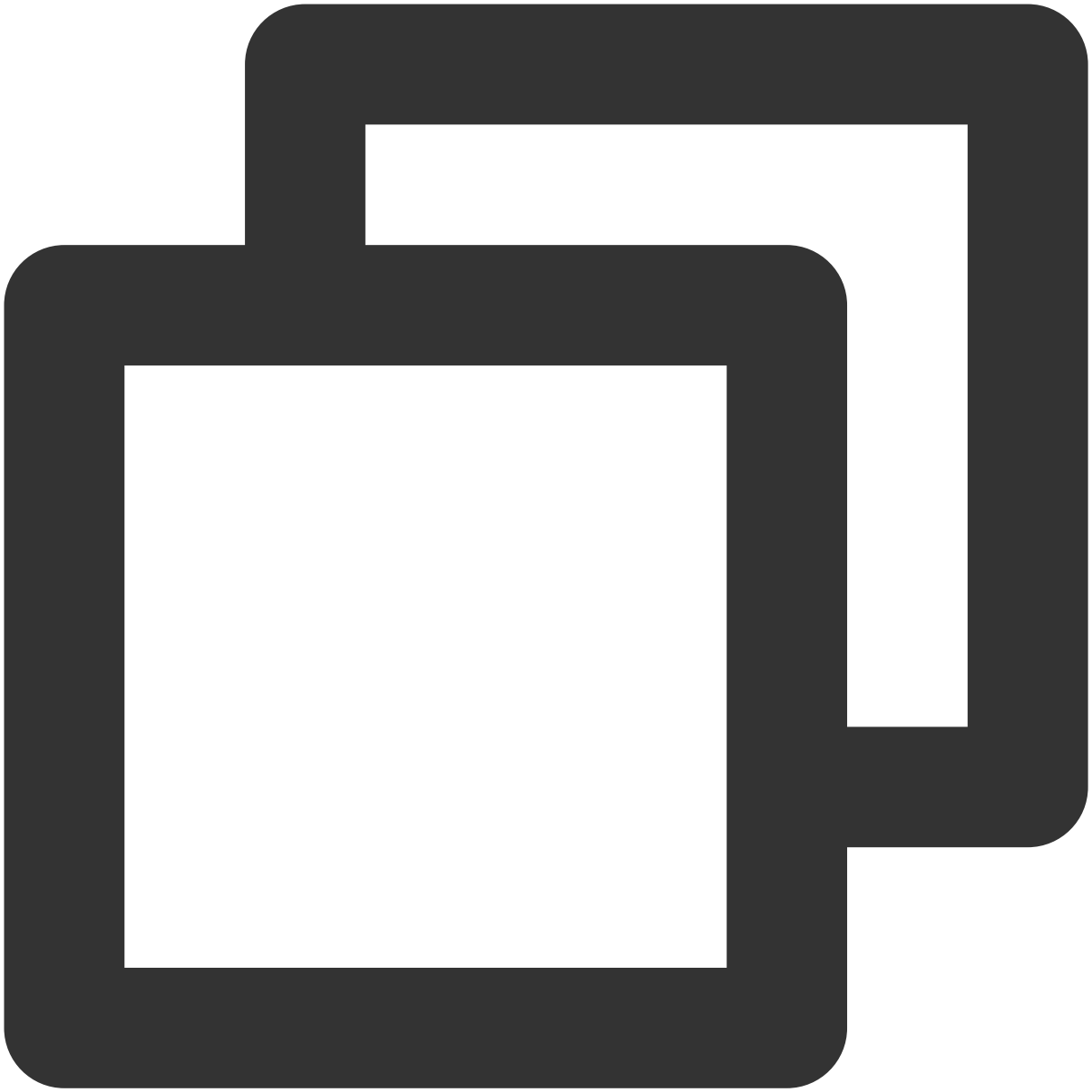
查看 mongo 集群是否开启了 balance



```
mongos> sh.getBalancerState()
true
```

也可通过执行 sh.status() 查看 balance 状态。

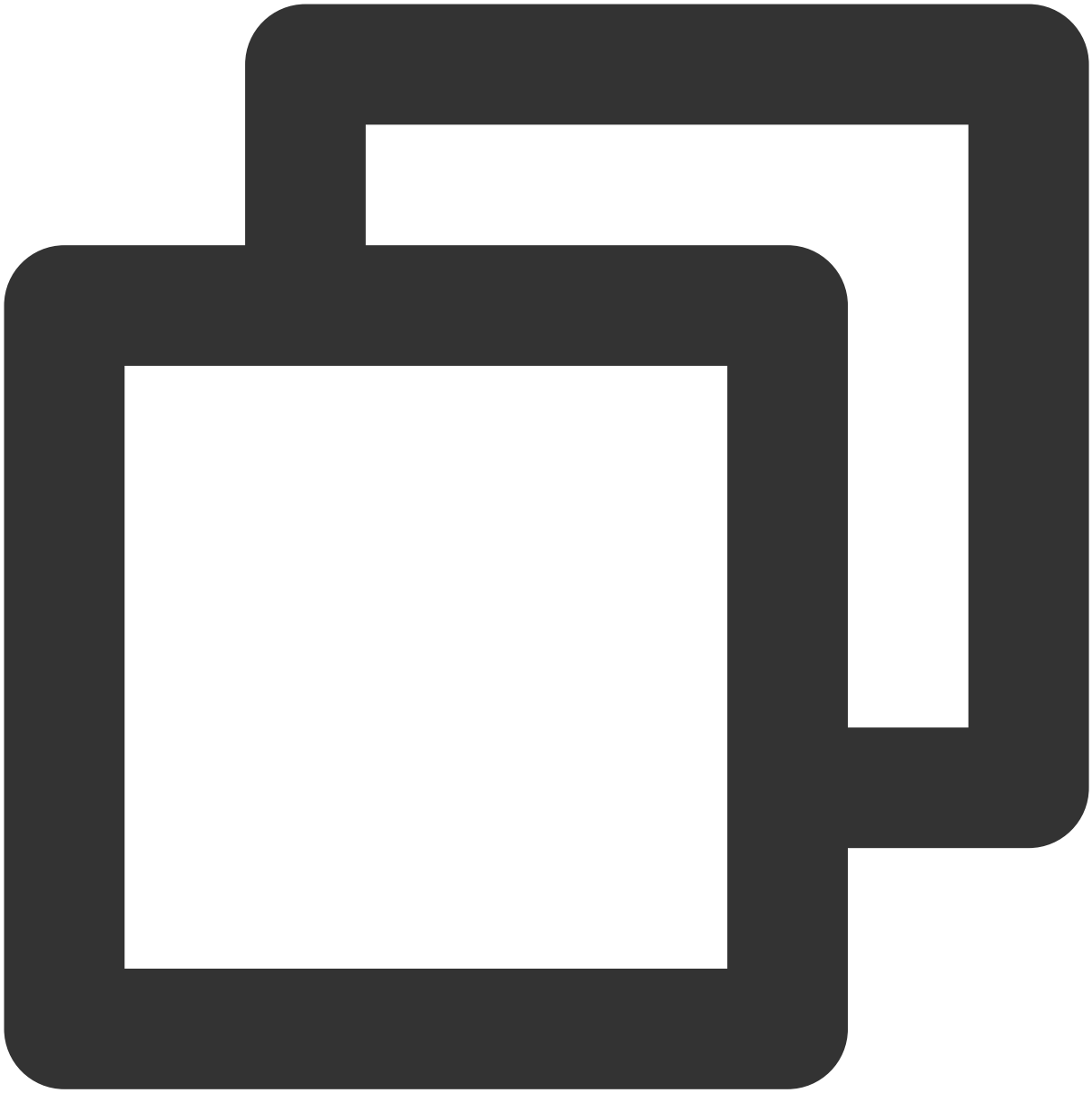
查看是否正在有数据的迁移



```
mongos> sh.isBalancerRunning()
false
```

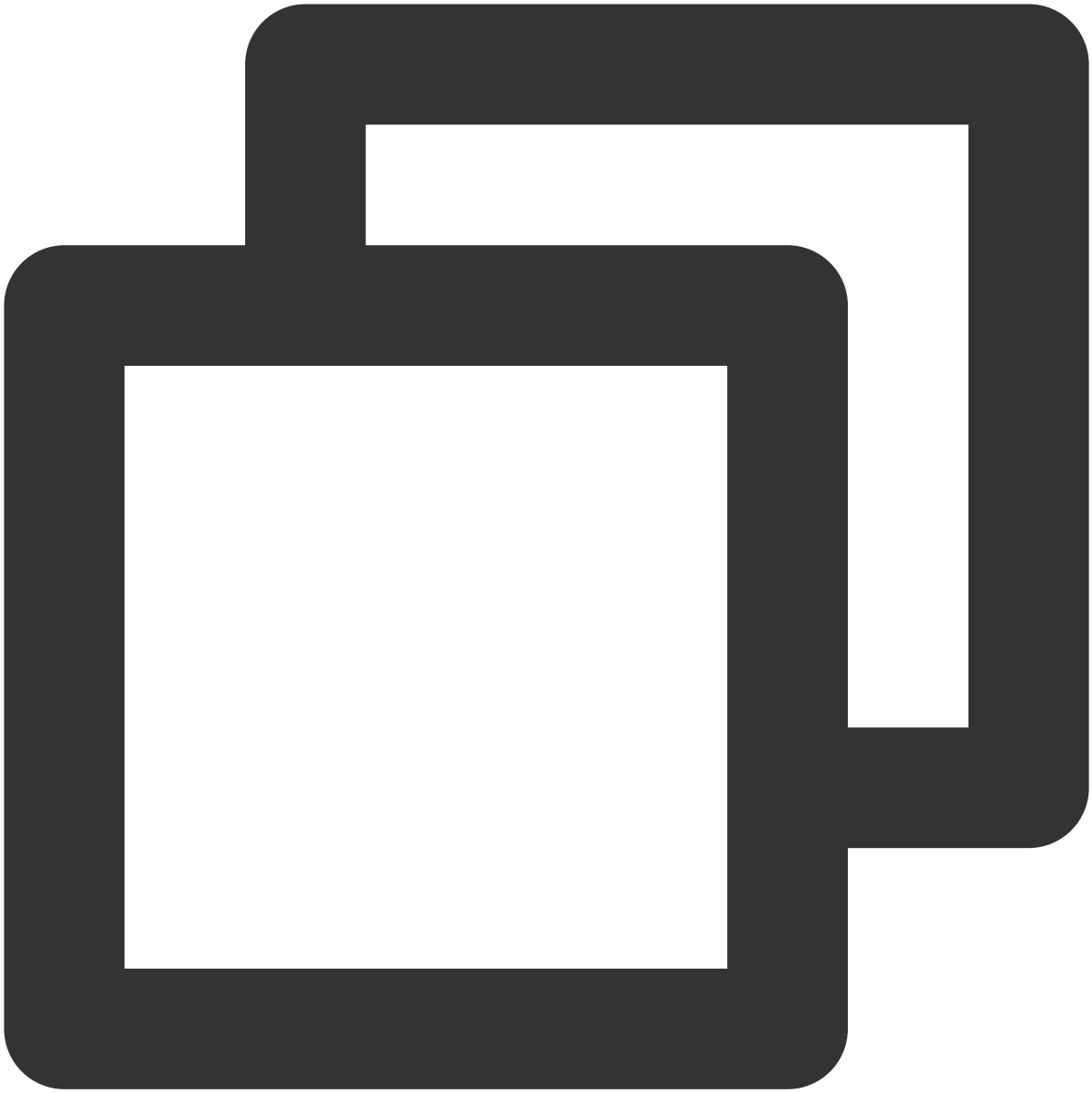
设置 balance 窗口

修改 balance 窗口的时间：



```
db.settings.update(
   { _id: "balancer" },
   { $set: { activeWindow : { start : "<start-time>", stop : "<stop-time>" } } },
   { upsert: true }
)
```

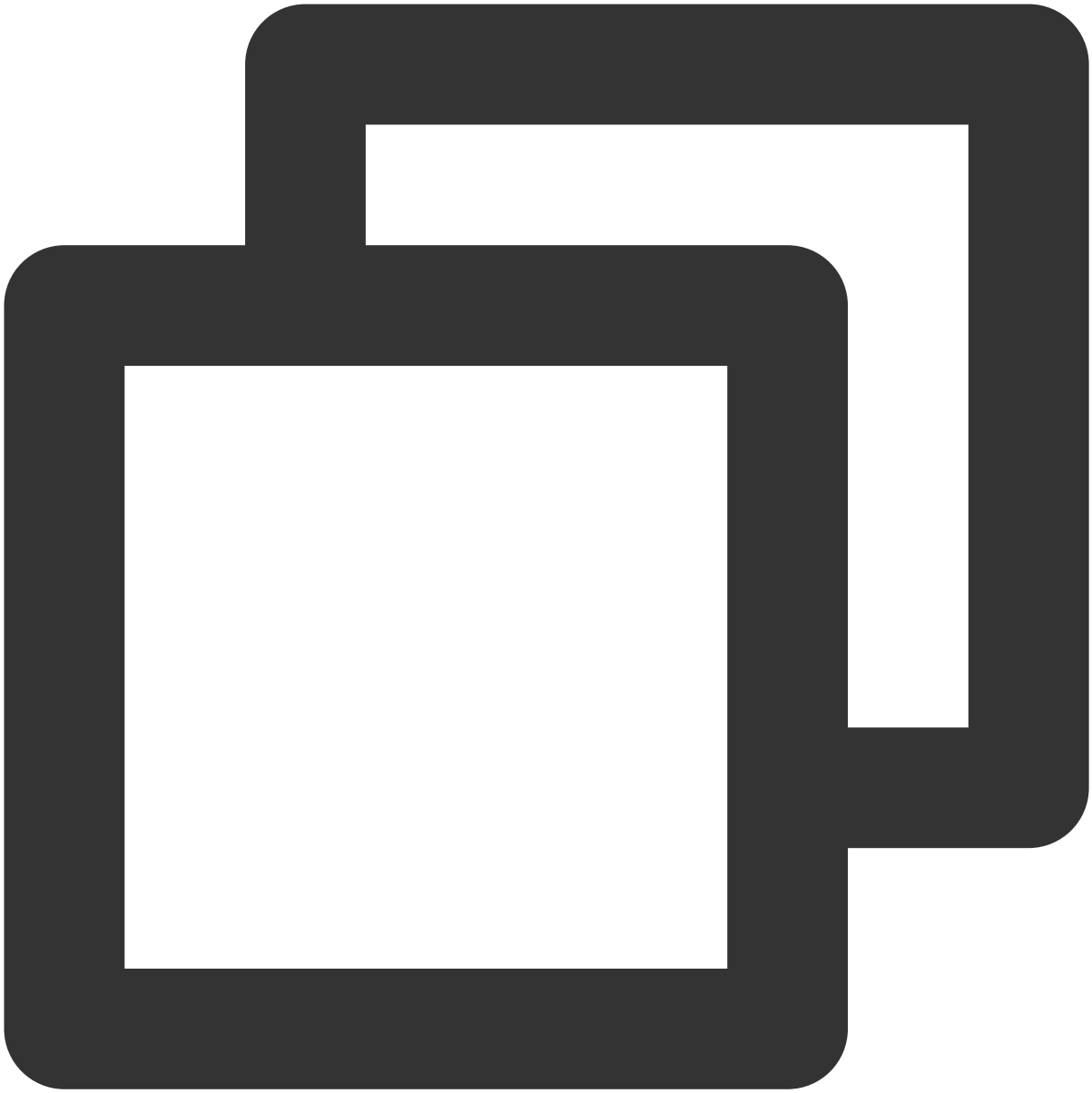
删除 balance 窗口：



```
use config
db.settings.update({ _id : "balancer" }, { $unset : { activeWindow : true } })
```

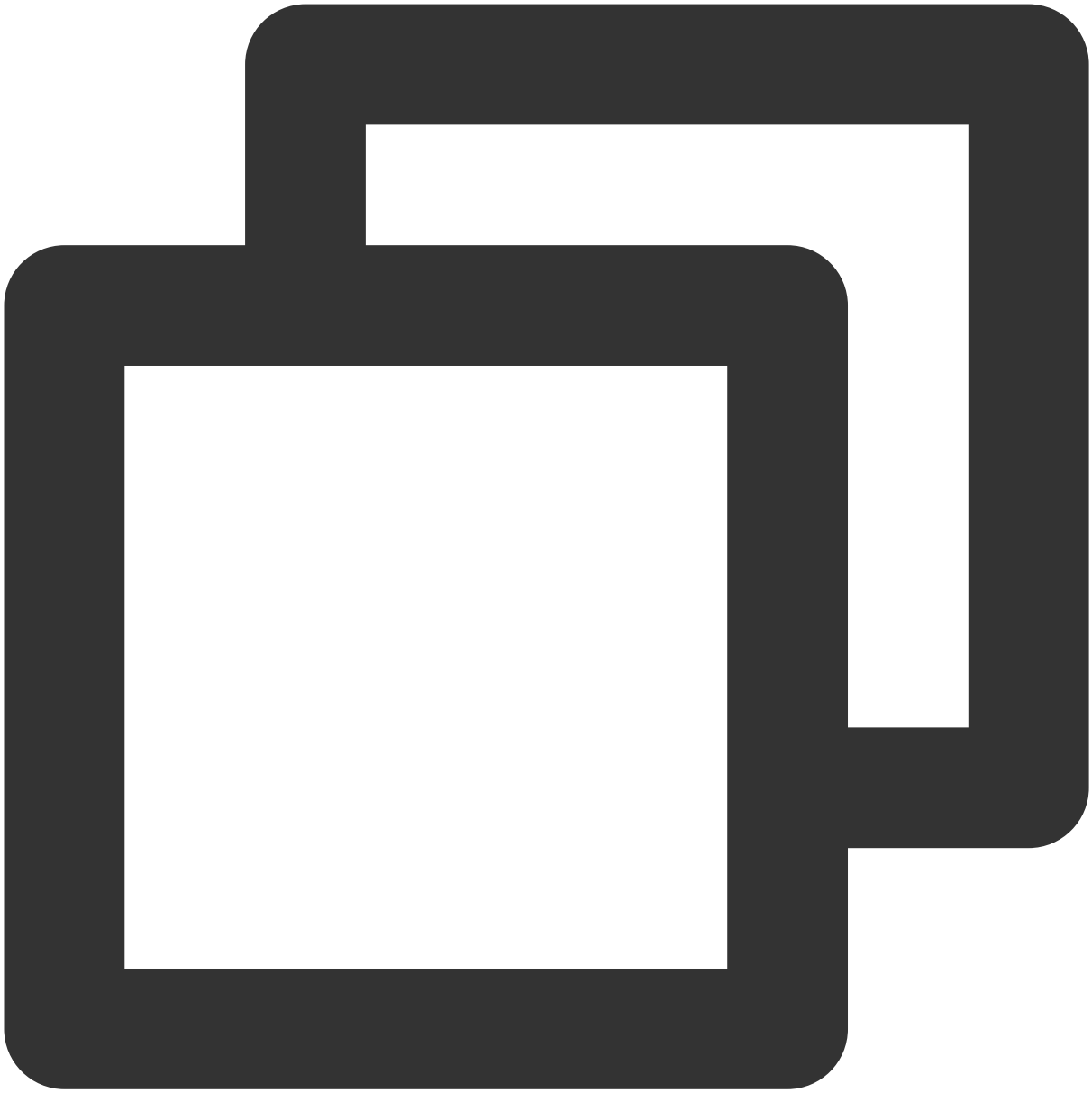
关闭 balance

默认 balance 的运行可以在任何时间，迁移只需要迁移的 chunk，如需关闭 balance，可执行下列命令：



```
sh.stopBalancer()
sh.getBalancerState()
```

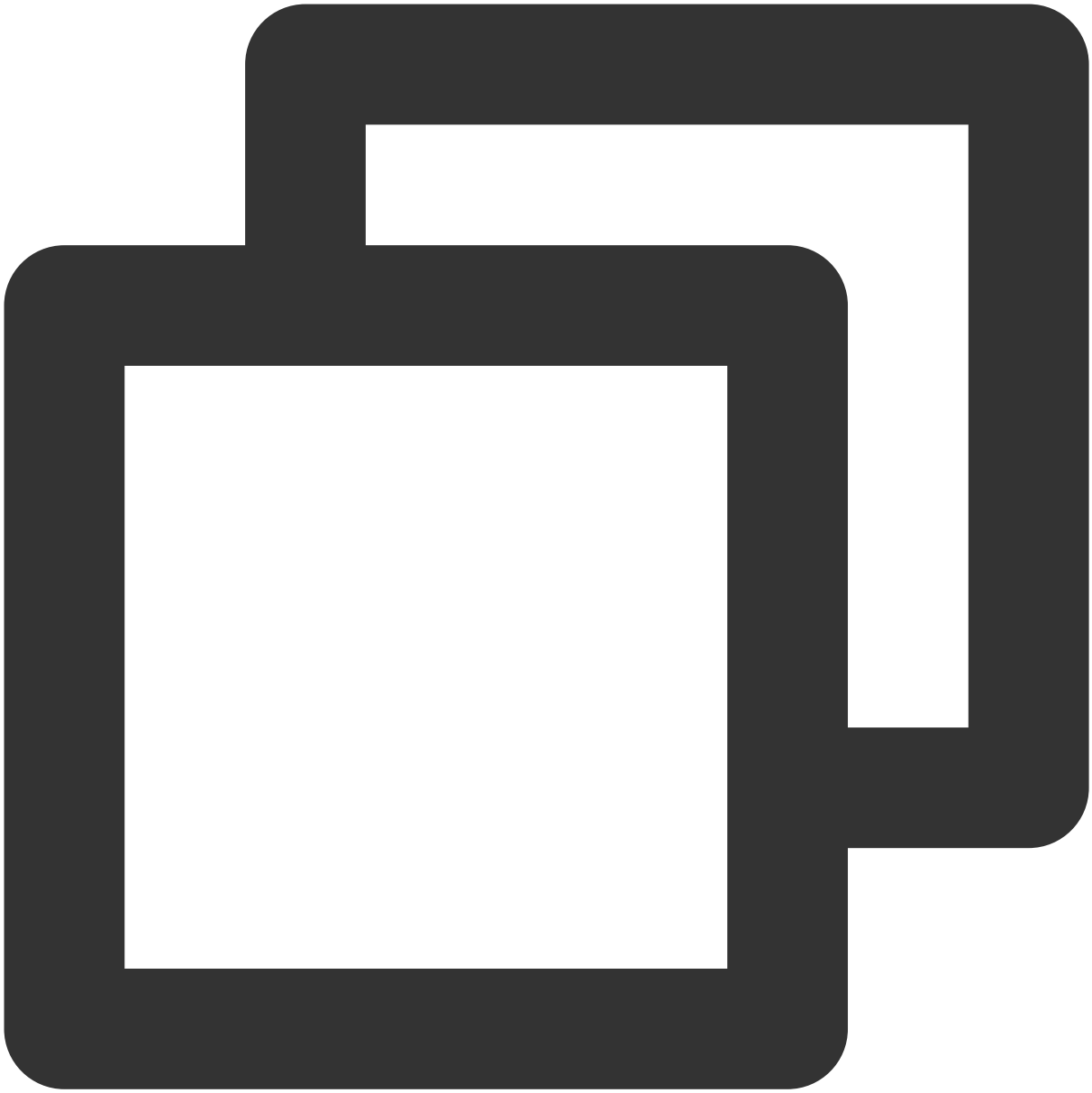
停止 balance 后，查看是否有迁移进程正在执行，可执行下列命令：



```
use config
while( sh.isBalancerRunning() ) {
          print("waiting...");
          sleep(1000);
}
```

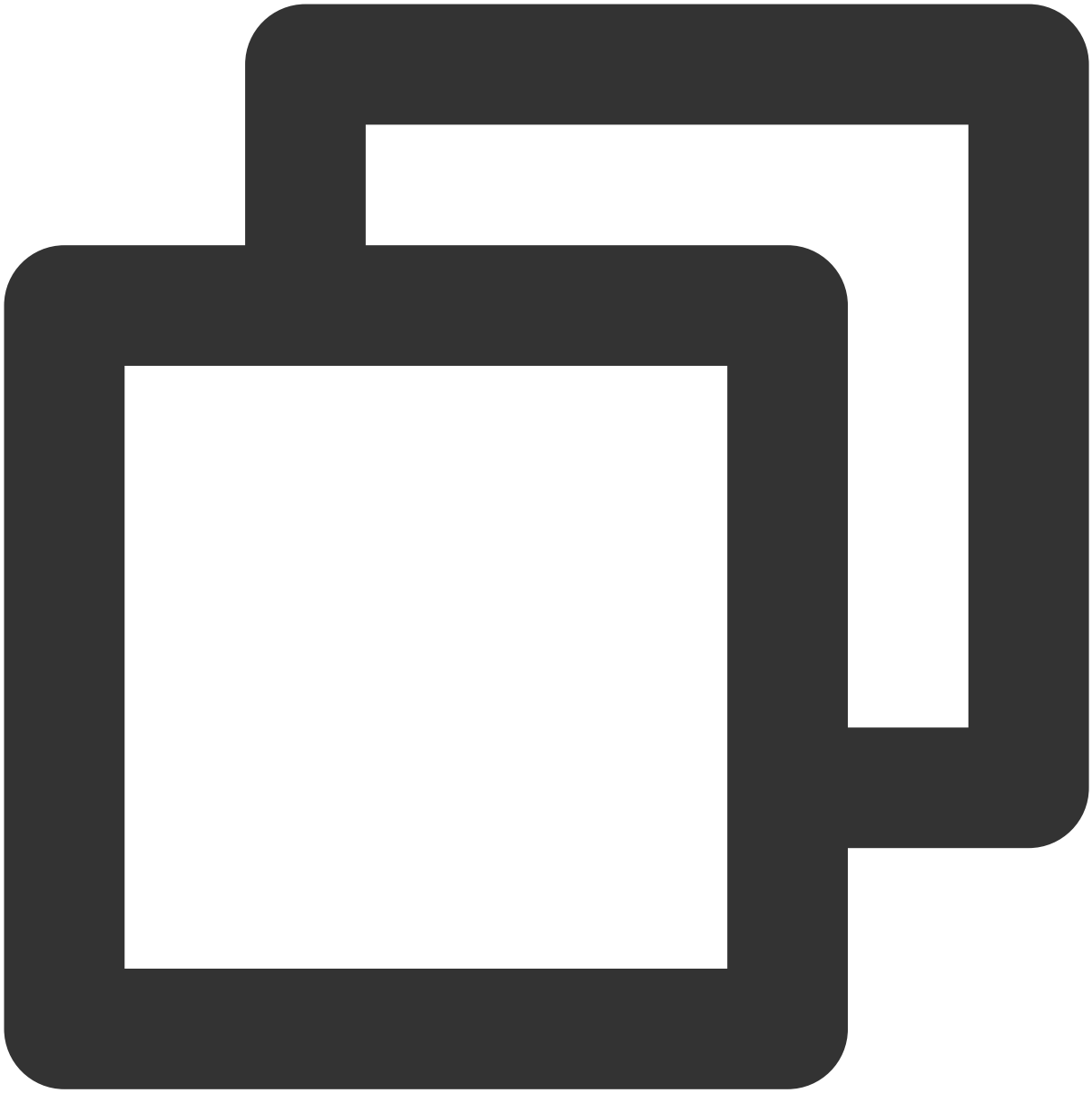
打开 balance

如您需要准备重新打开 balance，可执行下列命令：



```
sh.setBalancerState(true)
```

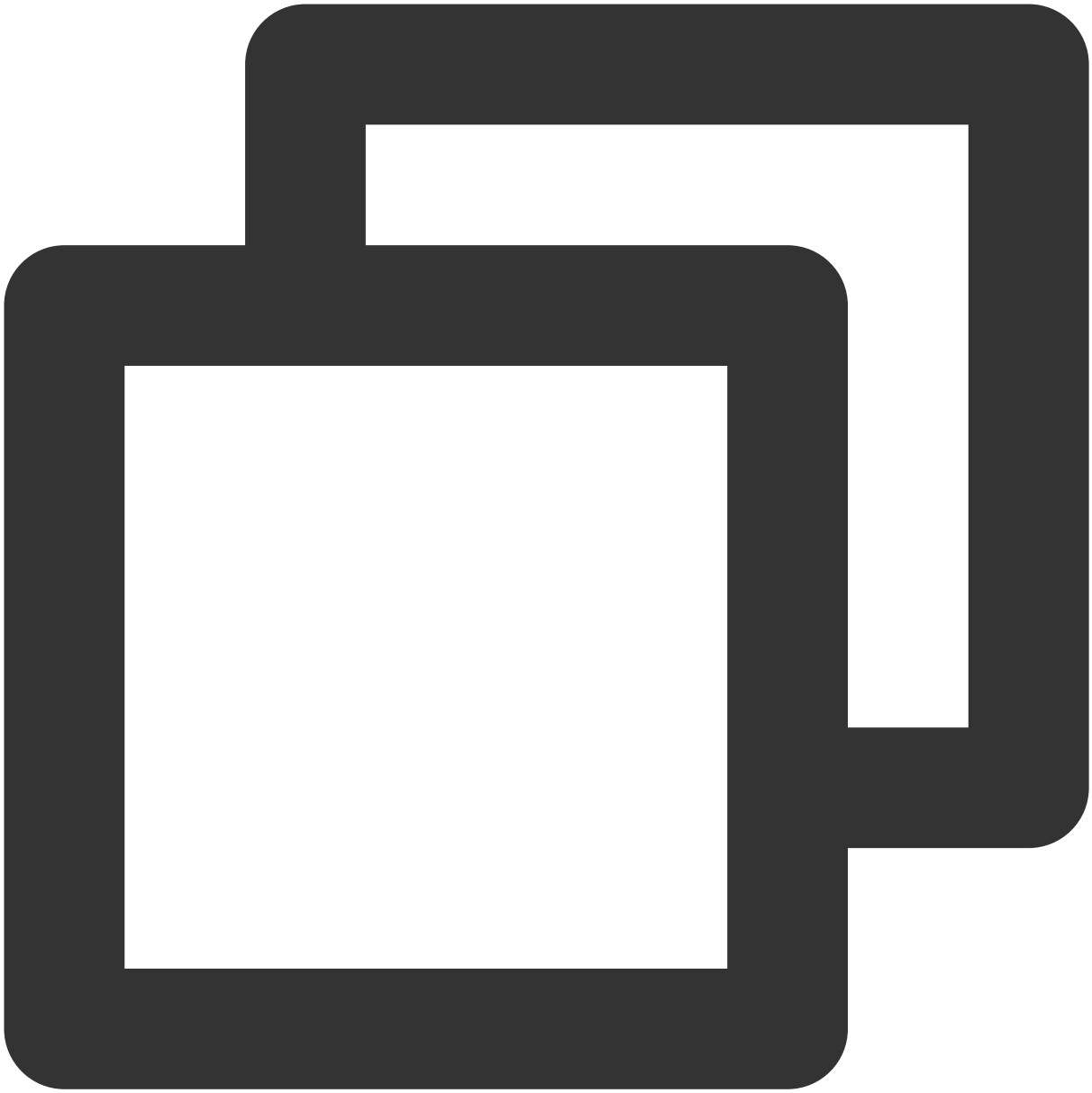
当驱动版本不支持 sh.startBalancer() 时，可执行下列命令来重新打开 balance：



```
use config
db.settings.update( { _id: "balancer" }, { $set : { stopped: false } } , { upsert:
```

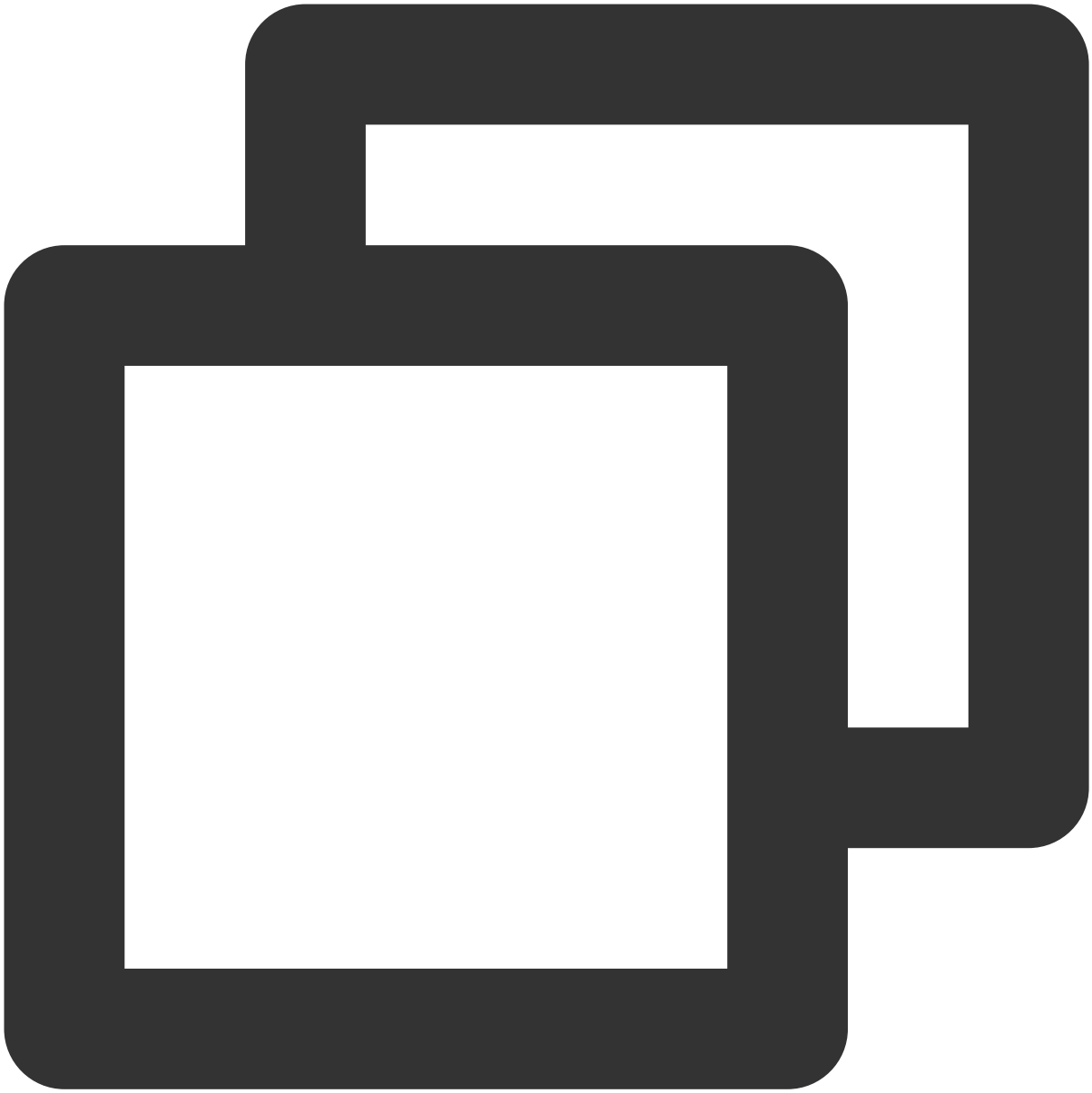
集合的 balance

关闭某个集合的 balance：



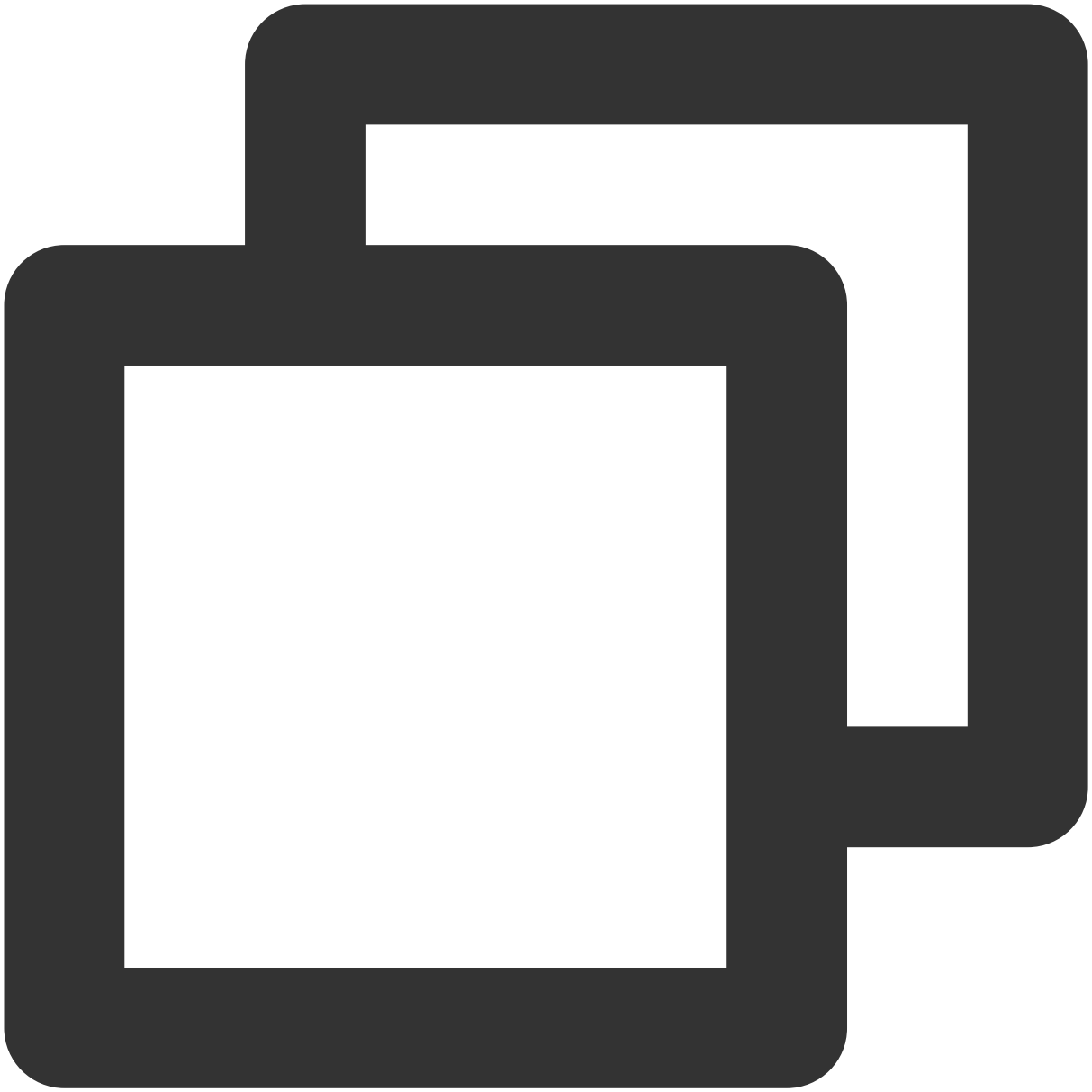
```
sh.disableBalancing("students.grades")
```

打开某个集合的 balance：



```
sh.enableBalancing("students.grades")
```

查看某个集合是否开启了 balance：

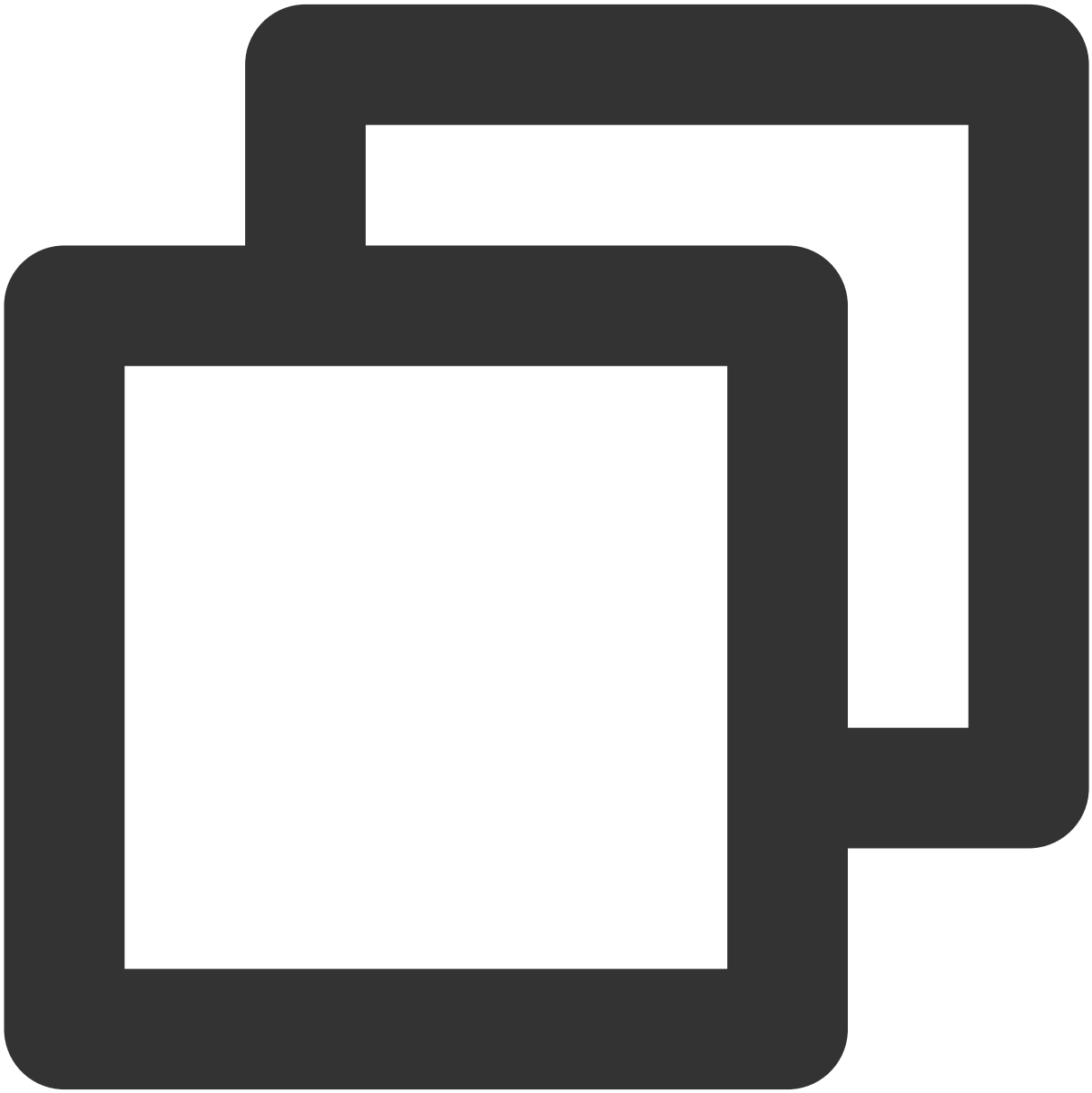


```
db.getSiblingDB("config").collections.findOne({_id : "students.grades"}).noBalance
```

MongoDB 协议实例读写示例

最近更新时间：2024-01-12 10:46:07

本文以 Python 代码示例来演示 MongoDB 分片集群的数据基本读写操作。首先在控制台创建分片集群实例，创建完成之后，在业务侧补充下述代码：
示例代码：



```
#!/usr/bin/python
import pymongo
```

```
import random


mongodbUri = 'mongodb://mongouser:1234567a@10.66.153.111:27017/admin'

client = pymongo.MongoClient(mongodbUri)
db = client.test

if 'num' in db.collection_names():
    db.drop_collection('num')

#create database and shardkey,shardkey is name
db_admin=client.admin
db_admin.command('enableSharding', 'test')
db_admin.command('shardCollection', 'test.num', key = {'name':1})

#insert data
print 'insert docs'
db.num.insert_one({'id':1, 'name':'R9', 'des':'pretty'})
db.num.insert_one({'id':2, 'name':'BOY', 'des':'handsome'})
db.num.insert_one({'id':3, 'name':'cat', 'des':'nice'})
db.num.insert_one({'id':4, 'name':'dog', 'des':'clever'})
print 'list all docs'
for i in db.num.find(): print i

#insert update doc
print 'update R9 and delete BOY'
db.num.update_one({"name":"R9"},{"$set":{"des":"good"}})
db.num.delete_one({"name":"BOY"})
db.num.update_one({"id":3}, {"$set":{"des":"kind"}})

print 'print R9'
for i in db.num.find({"name":"R9"}): print i
print 'list all docs'
for i in db.num.find(): print i
```

运行结果：

```
[root@VM_63_228_centos distribute_test]# python demo.py
insert docs
list all docs
{u'_id': ObjectId('589c62e99d89702a48ebb10c'), u'des': u'pretty', u'id': 1, u'name': u'R9'}
{u'_id': ObjectId('589c62e99d89702a48ebb10e'), u'des': u'nice', u'id': 3, u'name': u'cat'}
{u'_id': ObjectId('589c62e99d89702a48ebb10f'), u'des': u'clever', u'id': 4, u'name': u'dog'}
{u'_id': ObjectId('589c62e99d89702a48ebb10d'), u'des': u'handsome', u'id': 2, u'name': u'BOY'}
update R9 and delete BOY
print R9
{u'_id': ObjectId('589c62e99d89702a48ebb10c'), u'des': u'good', u'id': 1, u'name': u'R9'}
list all docs
{u'_id': ObjectId('589c62e99d89702a48ebb10c'), u'des': u'good', u'id': 1, u'name': u'R9'}
{u'_id': ObjectId('589c62e99d89702a48ebb10e'), u'des': u'kind', u'id': 3, u'name': u'cat'}
{u'_id': ObjectId('589c62e99d89702a48ebb10f'), u'des': u'clever', u'id': 4, u'name': u'dog'}
```

基于 CVM 连接 MongoDB 进行数据导入导出的方法

最近更新时间：2024-05-07 10:10:46

通过云服务器 CVM 连接云数据库 MongoDB 可以进行数据导入和导出，请注意使用最新版本的 MongoDB 客户端套件，具体操作可参见 连接实例。

注意：

local 数据库主要存储副本集的配置信息、oplog 等元数据；admin 数据库则主要存储用户、角色等信息。为了防止数据错乱、鉴权失败等现象发生，云数据库 MongoDB 禁止将 local 和 admin 数据库导入实例。

导出导入命令

MongoDB 官方提供了两套数据导入导出工具：

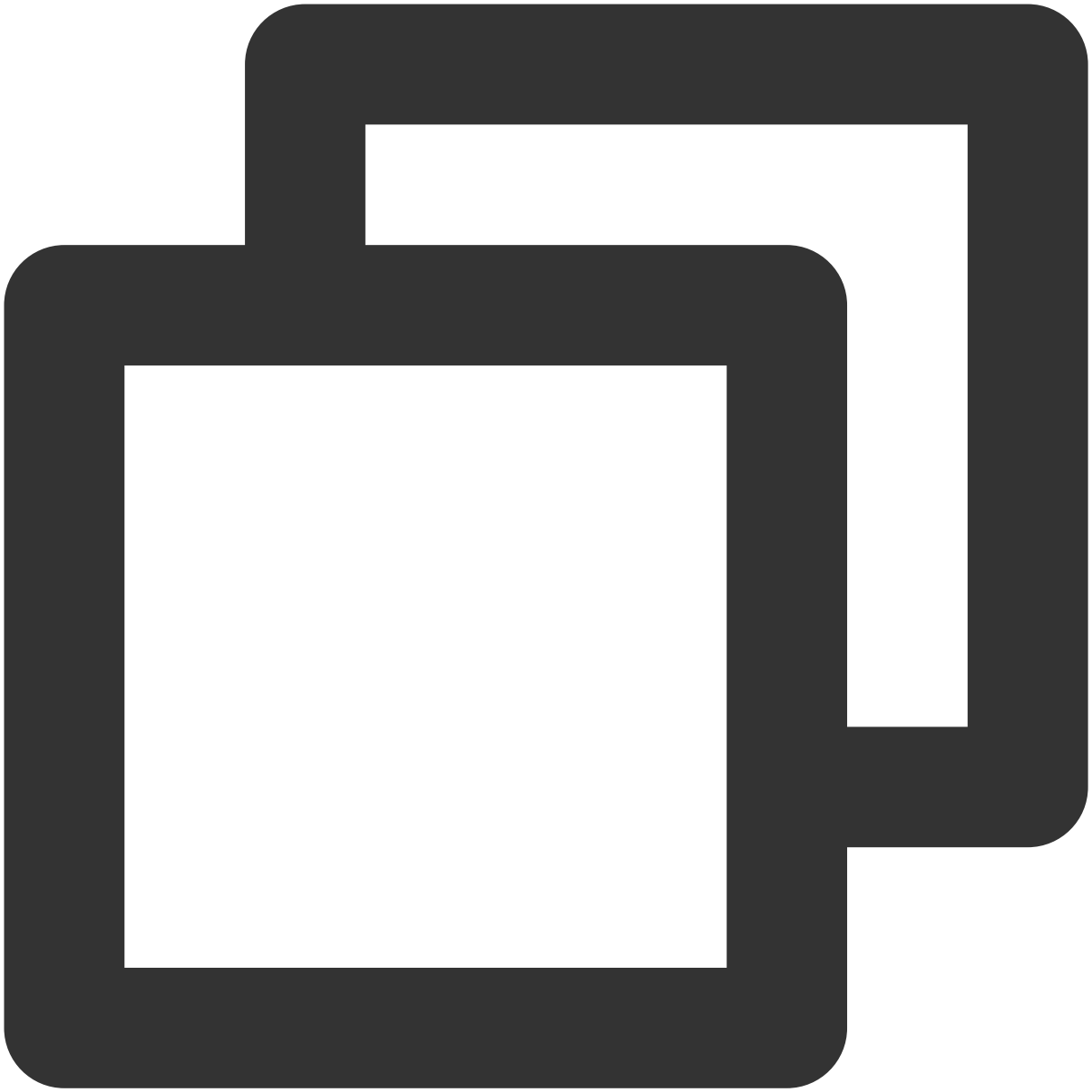
mongodump 和 mongorestore

mongoexport 和 mongoimport

mongodump 和 mongorestore

进行整库导出导入时，通常使用 mongodump 和 mongorestore，这一对组合操作的数据是 BSON 格式，进行大量 dump 和 restore 时效率较高。

mongodump 导出命令如下：

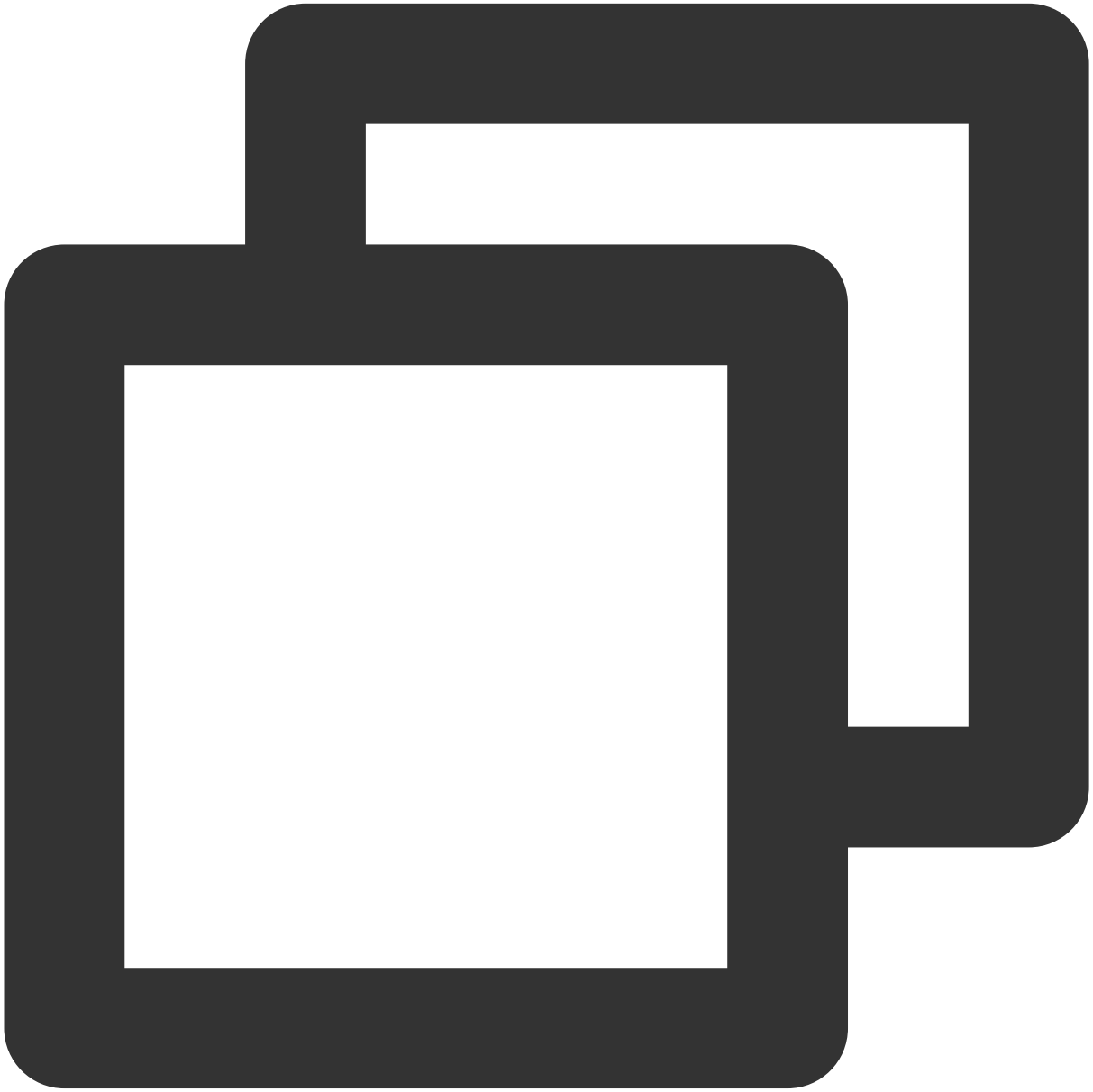


```
mongodump --host 10.66.187.127:27017 -u mongouser -p thepasswordA1 --authentication
```

如下图所示，则执行成功：

```
#: ./mongodump --host 10.66.187.127:27017 -u mongouser -p thepasswordA1
 --db=testdb -o /data/dump_testdb
2016-11-16T12:12:06.114+0800    writing testdb.system.indexes to
2016-11-16T12:12:06.116+0800    done dumping testdb.system.indexes (1 d
2016-11-16T12:12:06.116+0800    writing testdb.testcollection to
2016-11-16T12:12:06.118+0800    done dumping testdb.testcollection (3 d
```

mongorestore 导入命令如下：



```
mongorestore --host 10.66.187.127:27017 -u mongouser -p thepasswordA1 --authenticat
```

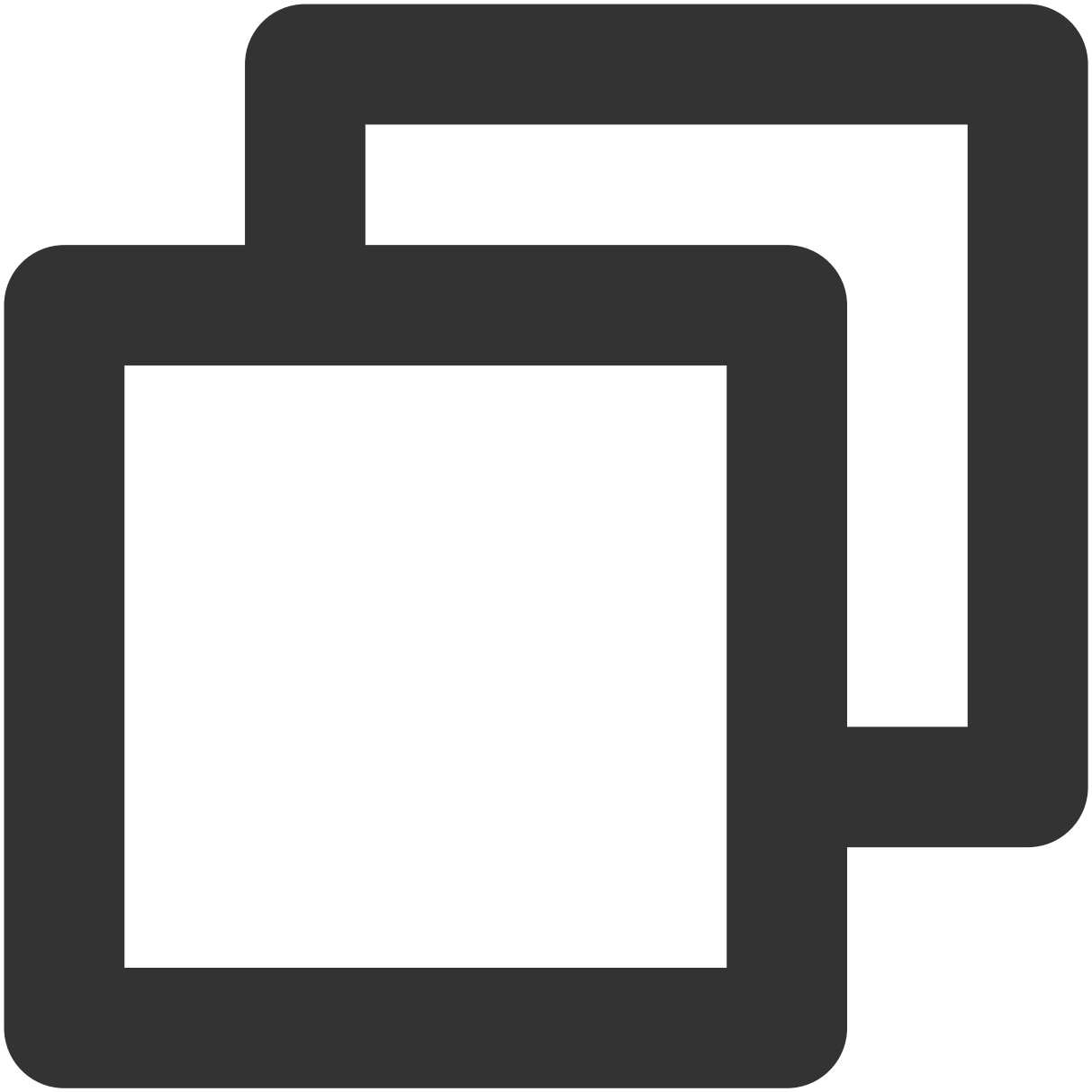
如下图所示，则执行成功：

```
#: ./mongorestore --host 10.66.187.127:27017 -u mongouser -p thepasswor
min --dir=/data/dump_testdb
2016-11-16T12:13:23.654+0800    building a list of dbs and collections
db dir
2016-11-16T12:13:23.678+0800    reading metadata for testdb.testcollect
db/testcollection.metadata.json
2016-11-16T12:13:23.678+0800    restoring testdb.testcollection from /d
ection.bson
2016-11-16T12:13:23.740+0800    restoring indexes for collection testdb
2016-11-16T12:13:23.740+0800    finished restoring testdb.testcollectio
2016-11-16T12:13:23.741+0800    done
#:
```

mongoexport 和 mongoimport

进行单个集合导出导入时，通常使用 mongoexport 和 mongoimport，这一对组合操作的数据是 JSON 格式，可读性较高。

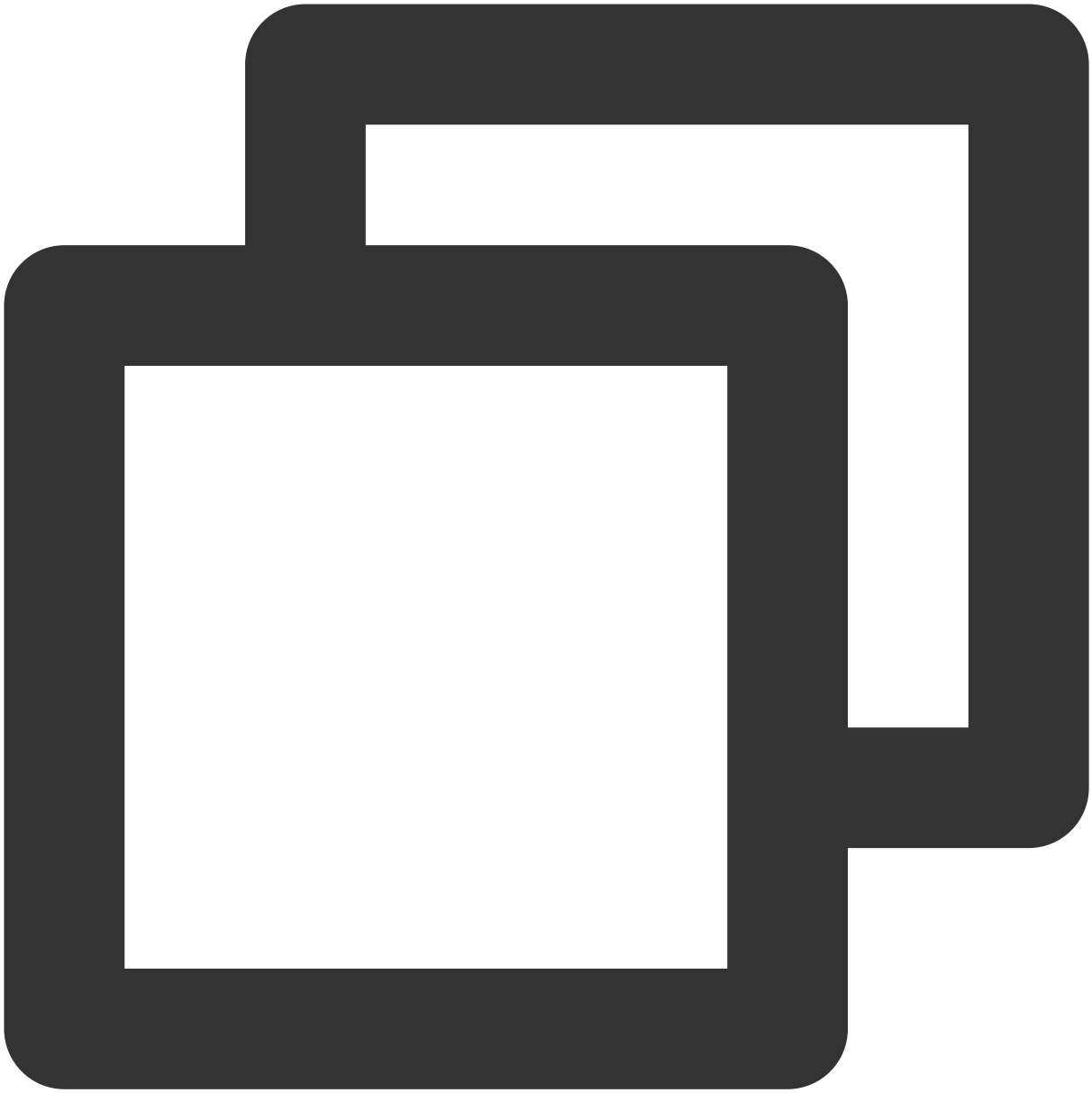
mongoexport 导出命令如下：



```
mongoexport --host 10.66.187.127:27017 -u mongouser -p thepasswordA1 --authenticati
```

另外您也可以加上 -f 参数指定需要的字段，-q 参数指定一个查询条件来限定要导出的数据。

mongoimport 导入命令如下：



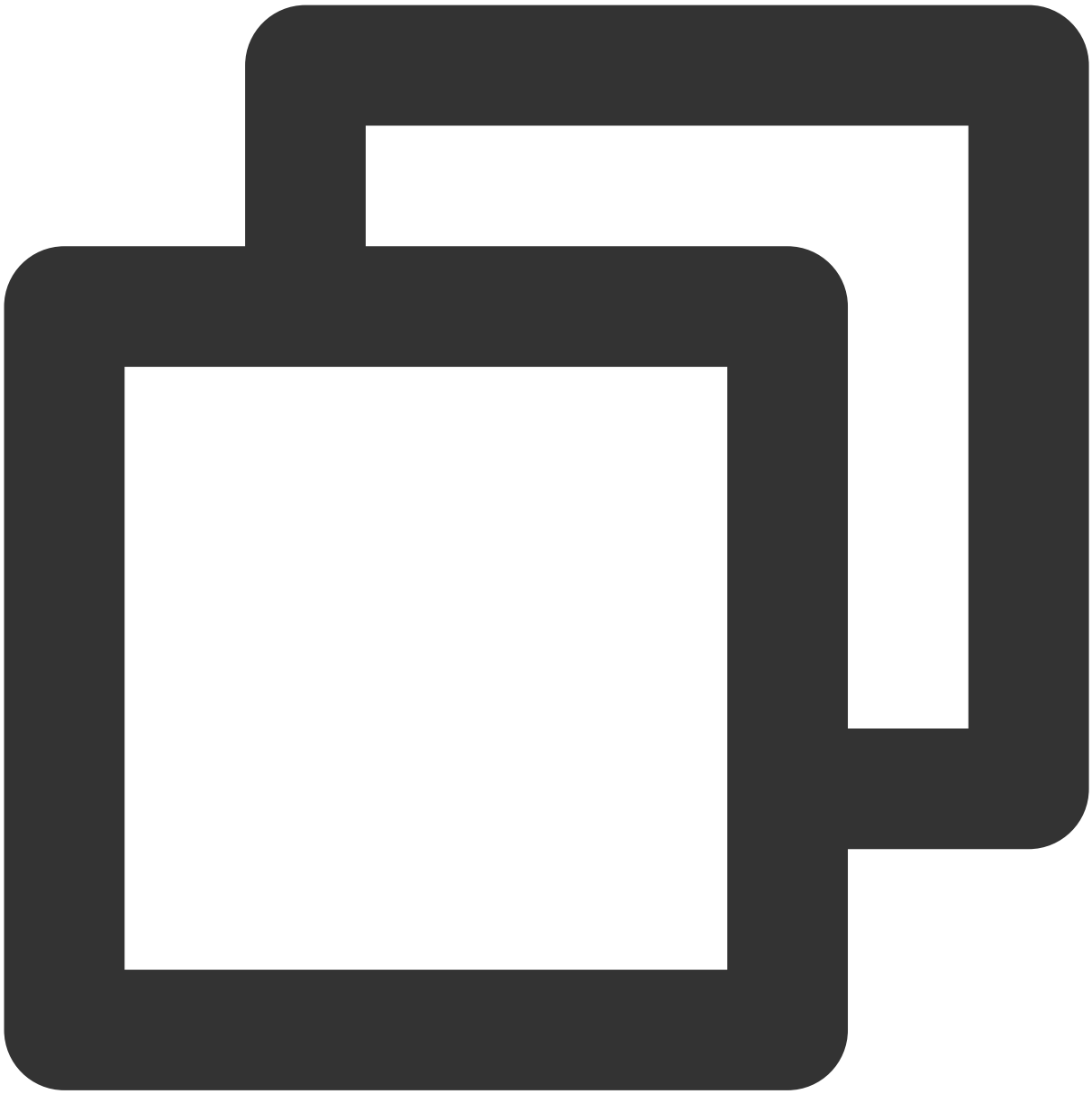
```
mongoimport --host 10.66.187.127:27017 -u mongouser -p thepasswordA1 --authenticati
```

多种认证方式的参数说明

在 连接示例 中有说明，云数据库 MongoDB 默认提供了 “rwuser” 和 “mongouser” 两个用户名分别支持 “MONGODB-CR” 和 “SCRAM-SHA-1” 两种认证方式。
对于 “mongouser” 以及在控制台创建的所有新用户，在使用导出导入命令工具时，根据上文示例操作即可。

对于 “rwuser”，需要在每个命令里加入参数 “--authenticationMechanism=MONGODB-CR”。

mongodump 示例：



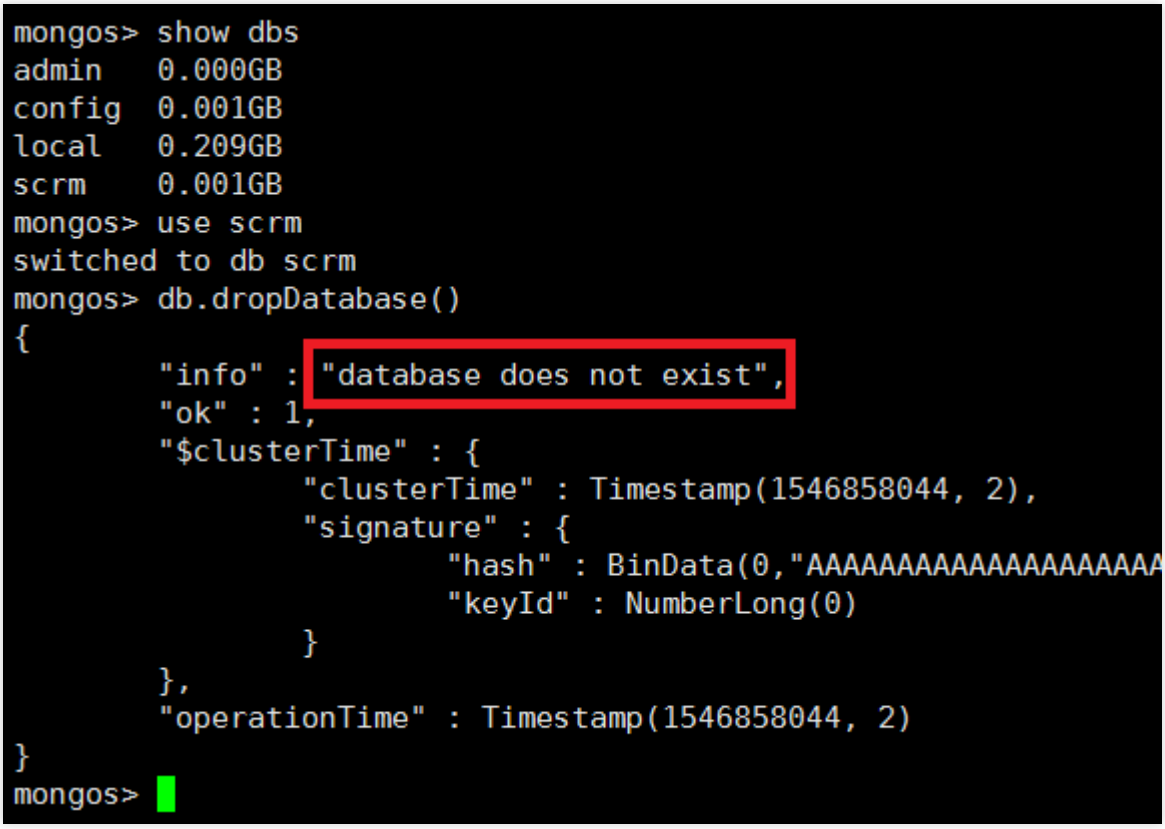
```
mongodump --host 10.66.187.127:27017 -u rwuser -p thepasswordA1 --authenticationDat
```

3.6版本实例反复创建和删除同名数据库时报错怎么办

最近更新时间：2024-05-07 17:39:08

问题描述

MongoDB 3.6版本实例如果反复 drop 一个 Database 然后创建一个相同名字的 Database，读写或者 drop 该 Database 时可能会报错 'database does not exist'，具体类似下图所示：



解决办法

该问题是一个共性问题。出现该问题的原因是 mongos 可能没有更新其 metadata cache。具体请参考官方 说明。如下图所示：

> **WARNING:**
> If you drop a database and create a new database with the same name, you must either restart a
> instances, or use the `flushRouterConfig` command on all `mongos` instances before reading
> that database. This action ensures that the `mongos` instances refresh their metadata cache, inclu
> location of the primary shard for the new database. Otherwise, the `mongos` may miss data on re
> write data to a wrong shard.

解决办法有两种，请选择其中一种：

1. 重启 mongos，该操作可以在 控制台 实例列表进行。
2. 或者运行 flushRouterConfig 命令，内附详细说明。

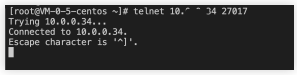
无法连接 MongoDB 解决方法

最近更新时间：2024-04-12 14:16:16

场景描述

使用云服务器 CVM 通过自动分配给云数据库的内网地址连接云数据库 MongoDB，具体连接方式，请参见 连接 MongoDB 实例。若连接失败，请参见下表进行排查与解决。

排查与解决

序号	可能原因	排查方法	解决方法
1	云服务器 CVM 和 数据库 MongoDB 内网不互通。云服务器和数据库不属于同一私有网络内。云服务器与数据库务必在同一账号同一个 VPC 内，或同在基础网络内，内网才能直接互通。 安全组配置错误。若想使用 CVM 连接 MongoDB，需在 CVM 安全组中配置出站规则，当出站规则的目标配置不为 0.0.0.0/0且协议端口不为	1. 登录 CVM 控制台，在实例列表的**实例配置**查看 CVM 网络信息。 2. 登录 MongoDB 控制台，在实例列表查看 MongoDB 的网络信息。具体操作，请参见 查看实例详情。 3. 比较 CVM 与 MongoDB 是否属于同一网络。 4. 登录 CVM，使用 `telnet 10.x.x.34 27017` 确认 MongoDB 网络端口是否可正常访问。 连接链路不通，如下图所示。  连接链路成功，如下图所示。 	如下情况，都可能因网络 致连接失败。 云服务器（CVM）采用私 （VPC），MongoDB 采 网络。建议将 MongoDB 网络切换为 VPC 网络， 切换实例网络。 CVM 采用基础网络，Mo 采用 VPC。建议将 CVM 网络切换为 VPC 网络， 切换私有网络服务。 CVM 与 MongoDB 在同一 内，但属于不同的 VPC 建议将 MongoDB 迁移到 所在的 VPC 网络，请参 实例网络。 CVM 与 MongoDB 不在同 内，属于不同的 VPC 网 议在两个 VPC 网络之间 联网。 CVM 与 MongoDB 账号不 于不同的 VPC 网络。建 个 VPC 网络之间建立 云 **CVM 安全组配置有误**

	ALL 时，需要把 MongoDB 的 IP 及端口添加到出站规则中。 若想指定的 CVM 连接 MongoDB 实例，需要在 MongoDB 安全组中配置入站规则，当入站规则的源端配置不为 0.0.0.0/0且协议端口不为 ALL时，需要把 CVM 的 IP 及端口添加到入站规则中。		1. 登录 安全组控制台，在 列表中，找到 CVM 所绑 全组，单击安全组名，进 CVM 绑定的安全组详情 2. 选择**出站规则**页签，单 **规则**。 **类型**选择**自定义**。 **目标**填写您 MongoDB 的 （段）。 **协议端口**填写 MongoDB 口。 **策略**选择**允许**。 **MongoDB 安全组配置有** 1. 登录 安全组控制台，在 列表中，找到 MongoDB 绑定的安全组，单击安全 进入 MongoDB 绑定的安 情页。 2. 选择**入站规则**页签，单 **规则**。 填写您允许连接的 IP 地址 （段）及需要放通的端口 （MongoDB内网端口）， 许放通。 **类型**选择自定义。 **来源**填写您 CVM 的 IP 地 （段）。 **协议端口**填写 MongoDB 口。 **策略**选择**允许**。
2	用户名与密码输入错误。	登录 CVM，连接数据库实例，提示账号密码错误。例如：提示 `Error: Authentication failed`，说明用户名密码输入错误。	登录 MongoDB 控制台， **详情**页面，选择**数据库管** 签，进入 **账号管理**页面，查看当前 所有账号信息，重置密码 操作，请参见 账号管理。
3	数据库访问密码中包含%、@等特殊字符。	密码中存在%与@特殊字符，驱动或者 MongoShell 等客户端没有自动转义这些特殊字符，引起这些特殊字符与连接串地址冲突，而导致用户名或密码出错。提示 `Password cannot properly be URL decoded` 或者 `Error: Authentication failed` 错误信息。	将访问密码的特殊字符按 转义规则进行处理： 感叹号“!”: 转义为 %21 at “@” : 转义为 %40 警号“#”：转义为 %23 百分号“%”: 转义为 %25 插入号“^” : 转义为 %5e 星号“*”: 转义为 %2a

			左括号“(”: 转义为 %28 右括号“)”: 转义为 %29 下划线“_”: 转义为 %5f 例如，如果原始密码为：`^%@132121a`，则的密码应为：`^%25%40132121a`
4	Mongoshell 版本过低	登录 CVM，执行 `mongo --version` 确认版本信息。	为保障鉴权成功，请安装 Shell 3.0及以上版本。安骤，请参见 官方文档。
5	客户端的连接串中，没有正确使用鉴权库。 控制台创建的用户：云数据库 MongoDB 统一使用 admin 库作为登录鉴权的认证数据库，在 URI 中端口后面必须加上“/admin”以指定认证库，通过认证后再切换到具体业务数据库进行读写操作。 命令行创建的用户：直接指定对应的数据库认证即可。例如在 test 库下建立的用户登录时，指定的认证库为 test。	控制台创建的用户：检查客户端程序中配置的连接串是否有包含“/admin”或者 authSource=admin。 命令行创建的用户：请您自行检查连接串中的认证库是否为正确的数据库名称。	登录 MongoDB控制台，**详情**页面的**网络配置**区域账号连接请直接复制 URI 连接串。其他账户，请您改为正确的认证库再尝试具体操作，请参见 连接 MongoDB 实例。 如果以上方法仍未解决问还可以 在线咨询 联系售
6	存在阻塞其他请求的操作。若在业务繁忙时段，进行了前台建索引操	业务侧自行排查索引创建的方式。	采用后台方式建索引。但索引方式也是有代价的，导致索引创建时间变长。建索引的选项，请参见 MongoDB 官网。同时，

	作（background 选项的值为 false ）。该操作将阻塞其他的所有操作，导致请求被锁住，直到前台完成索引创建。具体创建索引的方式，请参见 MongoDB 官网。		`currentOp` 命令来查创建索引的进度，具体的下：
			`db.currentOp(`
			`{`
			`$or: [`
			`{ op: "command", "query.createIndex $exists: true } },`
			`{ op: "in ns: /\\.system\\.index }`
			`]`
			`}`
			`)`

7	检查客户连接数是否已经达到上限。每个实例都有连接数上限的限制，超过限制，则无法连接。	登录 MongoDB控制台 ，在数据库管理页面，选择**连接数管理**页签，查看实例**最大连接数**、**实时连接数**、**连接使用率**。具体操作，请参见 连接数管理。	连接使用率偏高解决方法见 连接使用率偏高异常

注意：

在官网控制台创建的用户其认证库均为 admin，因此用户登录时需要制定认证库为 admin。用命令行创建的用户，例如在 test 库下建立的用户登录时指定的认证库为 test。

性能调优
连接使用率偏高异常分析及解决方法

最近更新时间：2024-01-12 10:47:04

场景描述

MongoDB 的服务模型是每个网络连接由一个单独的线程（one-thread-per-connection）来处理，当网络连接数太多时，过多的线程会导致上下文切换开销变大，同时内存开销也会上涨。每次请求都建立连接和鉴权会极大的影响性能。因此，限制实例的连接数，使用完毕规范化及时释放连接，是保障数据库稳定的必要条件。
登录 MongoDB控制台，在**系统监控**页面，查看实例监控指标**连接百分比**的趋势变化图。连接百分比指当前集群的连接数量与最大连接数的比例。若达到最大连接数，将会导致连接响应变慢，甚至连接失败的现象。因此，当连接使用率超过85%，请及时进行排查处理。

排查与解决

序号	可能原因	排查方式	解决方法
1	若使用连接池，可能因连接参数配置不合理而导致大量连接资源被占用。	请您自行排查客户端连接池配置参数是否适合业务场景。	连接池使用建议 进行连接池参数配置。
2	业务侧存在较多无实际业务请求的连接。	请借助**数据库智能管家**（TencentDB for DBbrain，DBbrain）的**诊断优化**功能，在**实时会话**页面，排查业务侧所有连接的客户端信息，是否为业务实际真实所需要的连接。	对于不需要的连接，可在**数据库智能管家**（TencentDB for DBbrain，DBbrain）的**诊断优化**中，在**实时会话**页面，直接进行 Kill 操作，进行清理。具体操作，请参见 诊断优化。
3	存在大量慢查询，连接一直占用未释放。	1. 请借助**数据库智能管家**（TencentDB for DBbrain，DBbrain）的**诊断优化**功能，在**慢 SQL 分析**页面，查询数据库当前的所有慢日志的记录和执行信息统计数据及视	针对慢查询，请进行索引优化。可借助数据库智能管家（TencentDB for DBbrain，DBbrain）索引推荐，选择最优的索引。 请参见最佳实践 索引优化解决读写性能瓶颈，提升数据库性能。

		图。具体信息，请参见 诊断优化。 2. 请登录 MongoDB 控制台，单击实例 ID，进入**实例详情**页面，选择**数据库管理**页签，单击**慢日志查询**，通过抽象排查，排查具体的慢查询信息。具体查询方法，请参见 慢日志管理。	
4	连接泄露，存在未释放的链接。	重启 mongos 实例， 导致实例所有的连接在重启的一瞬间中断，业务直接进行重连即可，不存在持续影响业务的可能。若重启后业务连接数迅速增加又导致连接使用率100%，则说明业务确实存在大量有效连接，不属于连接泄漏的场景。具体操作，请参见 重启实例。	直接在控制台提升连接，临时解决业务突发的状况，具体操作，请参见 连接数管理。 调整实例配置规格，副本集实例，提升Mongod 的 CPU 与内存配置，可同步提升实例的最大连接数量。具体信息，请参见 变更 Mongod 节点配置规格。分片集群，请提升 Mongos 的节点规格，或增加分片数量。具体操作，请参见 变更 Mongos 节点配置规格 与 调整分片数量。
5	业务量突增，当前实例配额不足	请参见序号2的排查方法。	

连接池使用建议

以 GO 语言为例，说明客户端通过连接池连接数据库时需配置的参数。具体信息，请参见下表。其他语言类型，请找到对应的连接池参数进行配置。不同语言类型的更多参数信息，请参见 MongoDB 官网。

参数	单位	参数含义	配置建议
maxPoolSize	数量	配置连接池每个客户端所能申请的最大连接数量。	该参数与客户端数量的乘积务必小于实例的最大连接数，避免连接数量过多而导致无法连接。
minPoolSize	数量	配置连接池每个客户端所能申请的最小连接数量	该参数与客户端数量的乘积小于实例的最大连接数，避免业务突发需要新建太多连接而后端实例资源消耗供应不足。
socketTimeoutMS	毫秒	配置发送和接受 sockets 等待响应的超时时间。默认为 0，指不超时。	建议根据业务实际场景设置，避免当 MongoDB 服务端异常故障引起主备切换之后，客户端一直等不到服务端响应的消息包，而导致此无效连接资源一直被占用。

maxIdleTimeMS	毫秒	配置一个空闲连接在被删除或者关闭之前存在的最大时间。	建议业务调整为1小时内，避免空闲连接一直占用连接资源。
heartbeatFrequencyMS	毫秒	配置客户端给服务端发送心跳的频率。用于客户端定期检查与后端数据库连接的存活情况。	建议配置10s内，便于第一时间识别服务端的运行状况，避免产生无效连接。

CPU 使用率偏高异常排查方法

最近更新时间：2024-05-07 17:21:28

问题描述

日常运维，登录 MongoDB 控制台，单击实例 ID 进入**实例详情**页面，选择**系统监控**页签，检查实例的监控指标详情，发现数据库 **CPU 监控类** 指标明显持续偏高。

原因分析及解决方法

序号	可能原因	原因分析	排查方法	解决方法
1	频繁建立短连接	实例大量资源消耗在处理频繁短连接上，引起 CPU 使用率较高，连接数较高，然而 QPS（集群每秒访问次数）未达到预期。	借助数据库智能管家 DBbrain 的**诊断优化**功能，分析数据库实例的实时会话统计视图及数据，确认是否存在连接数突增的现象。具体查询方式，请参见 实时会话。	将短连接调整为长连接，不同语言类型的更多参数信息，请参见 MongoDB 官网。
2	业务侧存在较多非预期的请求耗时较长	非预想的请求较多，或者被恶意请求，可能会导致 CPU 资源过度浪费。	请借助**数据库智能管家**（TencentDB for DBbrain，DBbrain）的**诊断优化**功能，在**实时会话**页面，排查业务侧所有连接的客户端请求，是否为业务实际真实所需。	对于非预期的请求，可在**数据库智能管家**（TencentDB for DBbrain，DBbrain）的**诊断优化**中，在**实时会话**页面，直接进行 Kill 操作，进行清理。具体操作，请参见 实时会话。 在**数据库智能管家**（TencentDB for DBbrain，DBbrain）的**诊断优化**中，在**SQL 限流**页面，创建 SQL 限流任务，自主设置 SQL 类型、最大并发数、限流时间、SQL 关键词，来控制数据库的请求访问量和 SQL 并发量，从而达到服务的可用性。具体操作及应用案例，请参见 SQL限流。
3	存在高		高复杂的查询请求，通	针对慢查询，请进行索引优化。

4 5	复杂的请求	高复杂的请求，可能因为如下原因导致 CPU 资源过度消耗。 排序操作：在查询过程中进行排序，本身将消耗较大 CPU资源。 全表扫描：若数据库没有使用索引，请求将可能进行全表扫描，对 CPU 和 IO 资源造成较大的压力，从而导致性能下降。 不合理的索引：可能会因为覆盖的字段不足，索引字段的顺序不合理，而导致大量的物理读和逻辑读，占有较多 CPU 资源	常比较耗时，会相应产生慢日志。可借助数据库智能管家 DBbrain 的**诊断优化**功能进行**慢日志分析**，在慢日志信息列表中，查看是否存在相关的关键字。具体操作，请参见 慢 SQL 分析。 排序相关关键字：Sort、hasSortStage 等。 全表扫描关键字 COLLSCAN：说明该查询进行了全表扫描。 docsExamined：代表文档扫描条目。 docsExamined 值越大，扫描的文档条目就越多，消耗 CPU 资源越多。 不合理索引关键字 IXSCAN：代表进行了索引扫描。 keysExamined：指明索引扫描条目。 keysExamined 值越大，扫描的条目就越多，消耗CPU资源越多。	可借助**数据库智能管家**（TencentDB for DBbrain，DBbrain）索引推荐，选择最优的索引。 请参见最佳实践 索引优化解决读写性能瓶颈，提升数据库性能。
6	业务上涨，资源不足	业务量上涨，数据规模增长，当前实例的 CPU 规格不足。	登录 MongoDB 控制台，单击**实例 ID** 进入**实例详情**页面，选择**系统监控**页签，查看实例总请求指标 QPS，查看节点每秒被请求的次数是否明显偏高。	调整实例配置规格。 Mongod 节点：提升Mongod 的 CPU 与内存配置。具体信息，请参见 变更 Mongod 节点配置规格。 Mongos 节点：如果 Mongos 达到瓶颈，请提升 Mongos 的节点规格，或增加 Mongos 数量。具体操作，请参见 变更 Mongos 节点配置规格 与 新增 Mongos 节点 。

内存调优方法

最近更新时间：2024-04-08 10:02:49

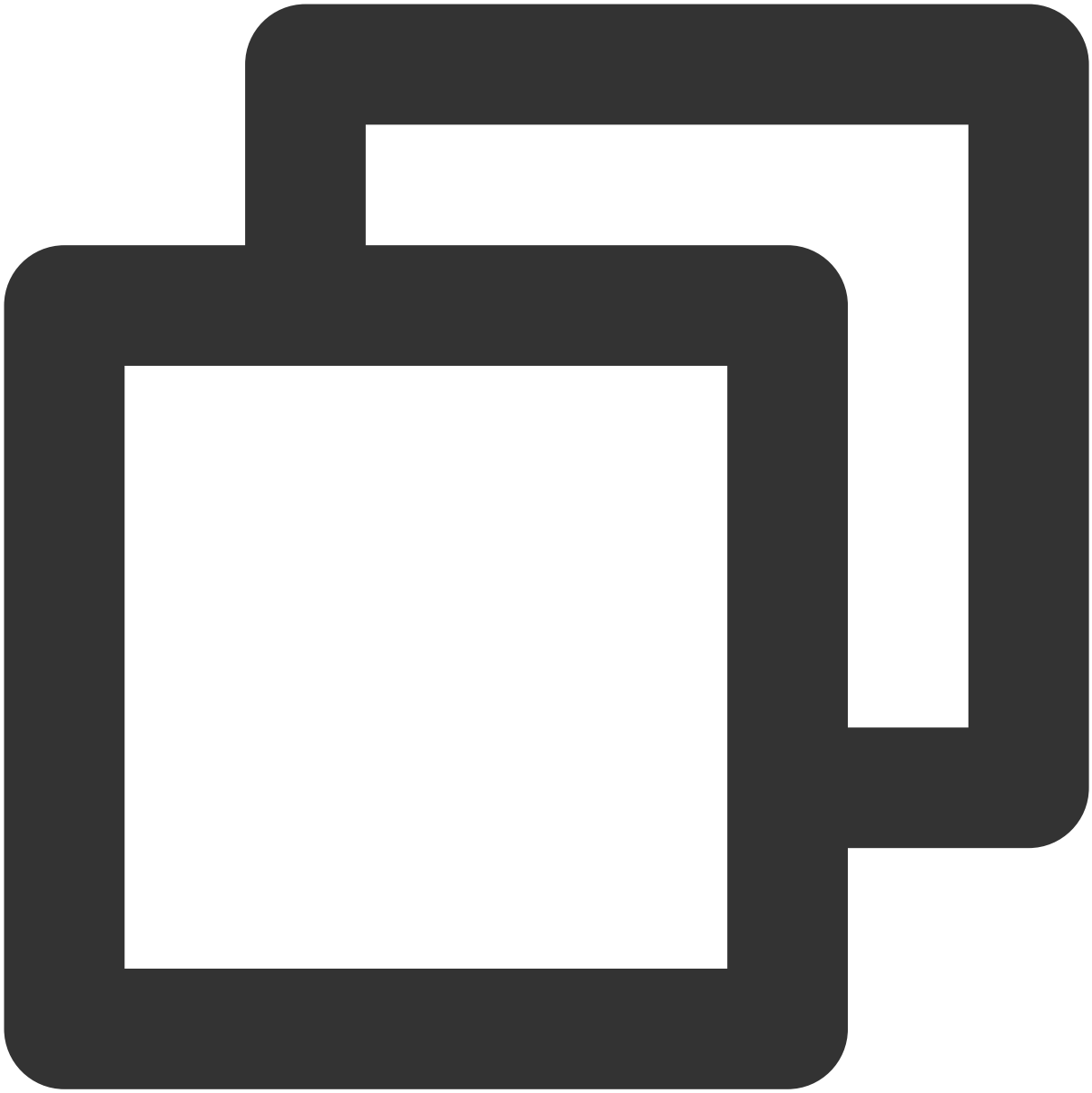
内存优化并不是简单地减少内存使用，而是在保证系统性能正常的前提下，让内存使用足够且稳定，并在机器资源和性能之间达到一个最佳的平衡方案。本文从云数据库 MongoDB 通常内存开销的类别出发，说明其内存占用原因，并给出排查方法及解决建议，便于您及时调优数据库性能。

存储引擎内存

云数据库 MongoDB 将存储引擎 WiredTiger 的缓存大小限定为实际申请的实例内存规格大小的60%。如果存储引擎缓存接近于限定范围的95%，说明实例负载已经很高。存储引擎脏数据缓存占比如果达到20%，将会阻塞用户线程。

排查方法

在 MongoDB Shell 中通过 `db.serverStatus().wiredTiger.cache` 命令可查看引擎内存的使用情况。返回信息中 `bytes currently in the cache` 后的值为缓存数据的大小。如下图所示：



```
{
    ......
    "bytes belonging to page images in the cache":6511653424,
    "bytes belonging to the cache overflow table in the cache":65289,
    "bytes currently in the cache":8563140208,
    "bytes dirty in the cache cumulative":NumberLong("369249096605399"),
    ......
}
```

在**数据库智能管家 DBbrain** 性能趋势的 MongoStatus 功能，可实时查看存储引擎 Cache 脏数据占比。具体信息，请参见 mongostat。

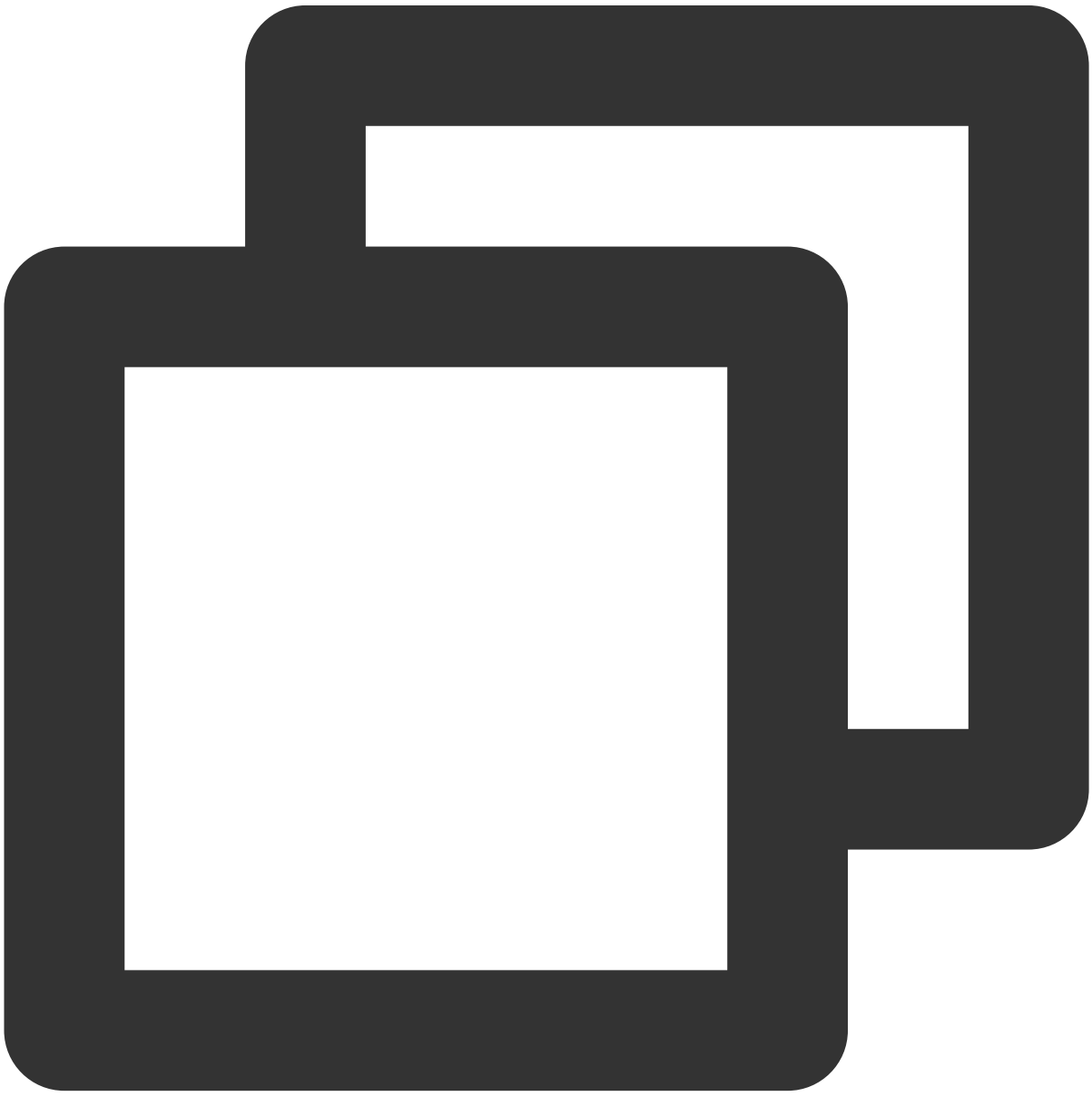
使用建议

如果存储引擎 Cache Dirty 持续性升高超过20%，请按照如下步骤处理：

a. 控制单位时间写入的数据量。

b. 提升实例 Mongod 节点内存规格。

c. 提升清理脏数据线程数量。其线程数量越高，越消耗实例资源，请谨慎调整合适的数值。默认为 `threads_max=4,threads_min=1` ，其设置方式如下：



```
db.runCommand({"setParameter":1, "wiredTigerEngineRuntimeConfig":"eviction=(threads
```

说明：

通过 db.runCommand 调整清理脏数据的线程数量，仅适合副本集架构 4.0 及其之后的版本。

如果存储引擎 Cache Used 持续性升高超过95%，请按照如下步骤处理：

a. 通过**数据库智能管家 DBbrain** 的 慢 SQL 分析 对数据库中存在的慢日志进行分析。

b. 提升实例 Mongod 节点内存规格。

c. 提升清理脏数据线程数。

连接和请求内存

每个连接都需要对应一个请求线程进行处理，过多的请求线程会引起请求上下文频繁切换而导致内存开销增加。每个线程最多可以使用1MB的线程栈，通常情况下内存在几十KB到几KB之间。

每当接收到一个请求命令时，都会创建一个请求上下文，整个过程中可能会分配很多临时缓冲区，例如请求包、应答包和排序的临时缓冲区等。这些缓冲区在请求结束时都会被释放。但是，此时释放只是将其归还给内存分配器 `tcmalloc` 。 `tcmalloc` 会优先将这些缓冲区返回到自己的缓存中，然后逐步将它们归还给操作系统。很多情况下，内存使用率高的原因是 **`tcmalloc`** 未能及时将内存归还给操作系统，导致内存最大可能达到几十GB。

排查方法

`tcmalloc` 未归给操作系统的内存大小，可以通过命令 `db.serverStatus().tcmalloc.tcmalloc.formattedString` 或者 `db.serverStatus().tcmalloc` 查看。如下图所示，使用 `db.serverStatus().tcmalloc.tcmalloc.formattedString` 查询内存信息。其中的 `Bytes in use by application` 对应的内存指 Mongod 节点实际消耗的内存， `Bytes in page heap freelist` 为未归还给操作系统的内存。

```
MALLOC:    15842638328 (15108.7 MiB) Bytes in use by application
MALLOC: +     60239872 (   57.4 MiB) Bytes in page heap freelist
MALLOC: +    234037232 (  223.2 MiB) Bytes in central cache freelist
MALLOC: +      4681024 (    4.5 MiB) Bytes in transfer cache freelist
MALLOC: +      5683416 (    5.4 MiB) Bytes in thread cache freelists
MALLOC: +    132604160 (  126.5 MiB) Bytes in malloc metadata
MALLOC:   ------------
MALLOC: =  16279884032 (15525.7 MiB) Actual memory used (physical + swap)
MALLOC: +    618319872 (  589.7 MiB) Bytes released to OS (aka unmapped)
MALLOC:   ------------
MALLOC: =  16898203904 (16115.4 MiB) Virtual address space used
MALLOC:
MALLOC:        1990277              Spans in use
MALLOC:            114              Thread heaps in use
MALLOC:           4096              Tcmalloc page size
```

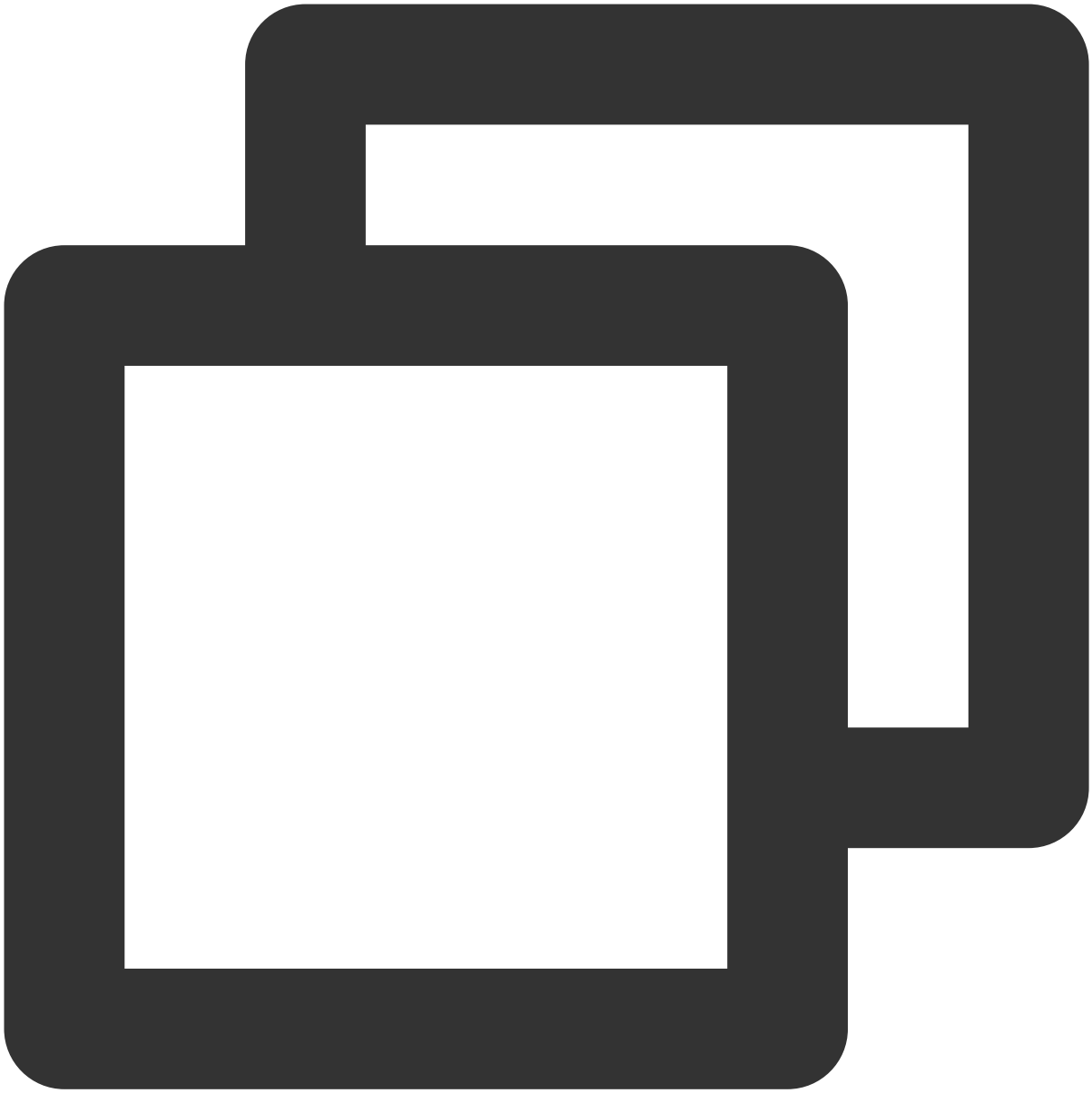
登录 MongoDB控制台，在**系统监控**页面，查看实例监控指标**连接百分比**的趋势变化图。连接百分比指当前集群的连接数量与最大连接数的比例。

使用建议

1. 快速回收内存。

`tcmallocReleaseRate` 是一个用于控制内存释放的参数，它指定了当内存占用超过一定比例时，应该释放多少内存。 `tcmallocReleaseRate` 的值越高，MongoDB 释放内存的速度就越快，但是也会对性能产生一定的影响。在配置 `tcmallocReleaseRate` 时需要根据实际情况进行调整，以平衡内存使用和性能需求。执行如下命令，可配置 `tcmallocReleaseRate` 的值，其取值范围为[1,10]。

`diagnosticDataCollectionVerboseTCMalloc` 为 `true` 时，可直接暴力回收内存，执行方式，如下所示：



```
db.adminCommand( { setParameter: 1, tcmallocReleaseRate: 5.0 } )
db.runCommand( { setParameter: 1, diagnosticDataCollectionVerboseTCMalloc:true} )
```

说明：

`tcmallocReleaseRate` 适用于云数据库 MongoDB 4.2及其之上的版本，且数据库实例为副本集。

`diagnosticDataCollectionVerboseTCMalloc` 适用于云数据库 MongoDB 4.4及其之上的版本，且数据库实例为副本集。

2. 控制并发连接数。

建议数据库中最大创建100个长连接，默认 MongoDB Driver 可以和后端建立100个连接池。当存在很多客户端时，降低每个客户端的连接池大小，一般建议与整个数据库建立的长连接控制在1000以内。连接百分比较高，请参见 连接使用率偏高异常分析及解决方法 进行调优。

3. 降低单次请求的内存开销。

例如通过创建索引减少集合的扫描、内存排序等。创建索引，请参见 MongoDB 官方 Indexes。

4. 升级内存规格。

在连接数合适的情况下内存使用率持续升高，建议升级内存规格，避免内存溢出和大量清除缓存而导致系统性能急剧下滑。

提升 Mongod 的 CPU 与内存配置。具体信息，请参见 变更 Mongod 节点配置规格。

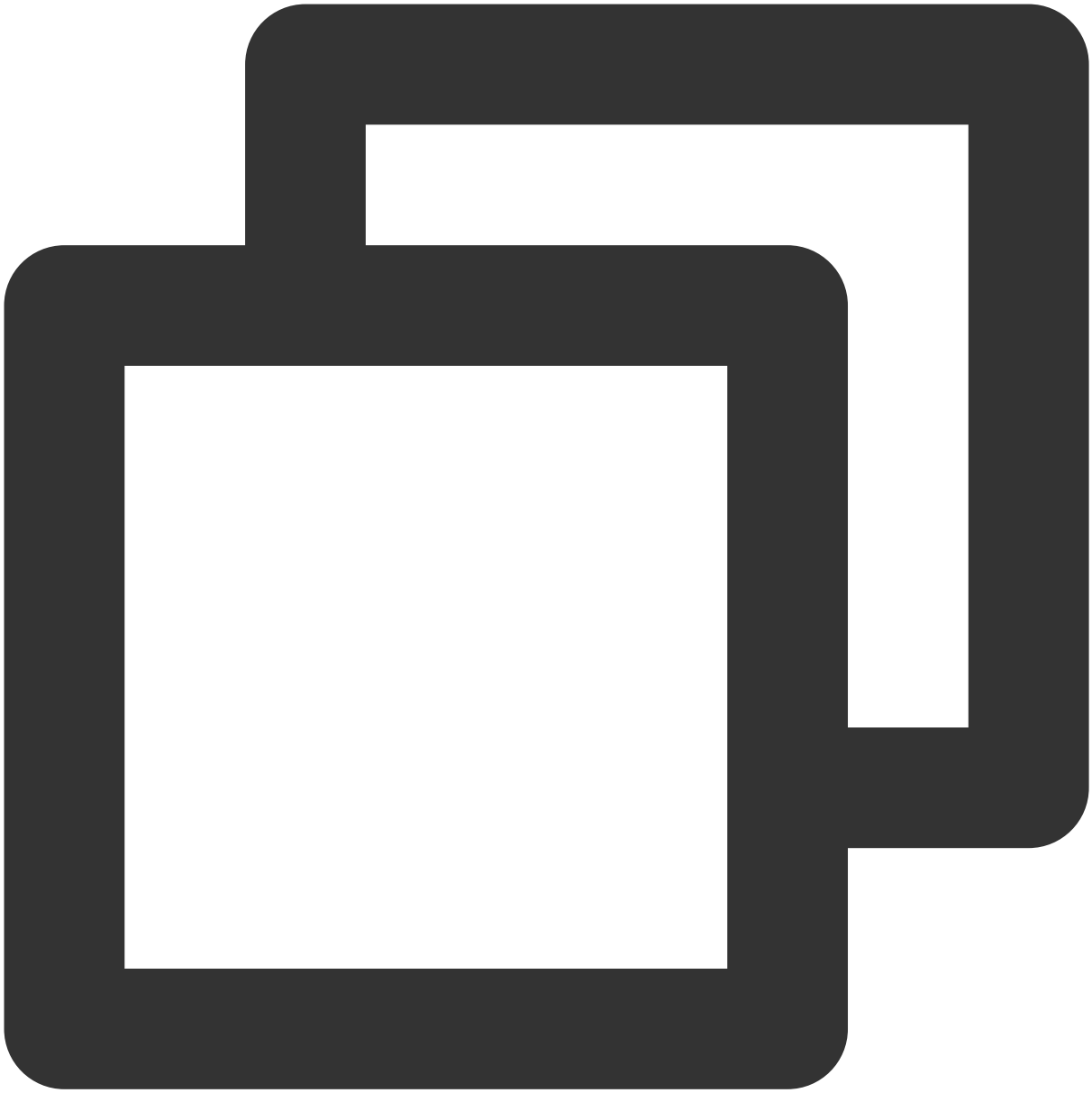
分片集群，可增加分片数量。具体操作，请参见 调整分片数量。

元数据信息内存

数据库 MongoDB 为每个集合维护一些元数据，例如索引信息、文档数量、存储引擎选项等。这些信息的管理会占用一定的系统资源。如果集合、索引等内存元数据数量很多，将会占用大量内存。特别是在云数据库 MongoDB 4.0之前的版本中，全量逻辑备份期间会打开大量文件句柄，但未能及时归还给操作系统，导致内存快速上涨。此外，在低版本的云数据库 MongoDB 中，大量删除集合后可能未能删除文件句柄，也会导致内存泄漏。

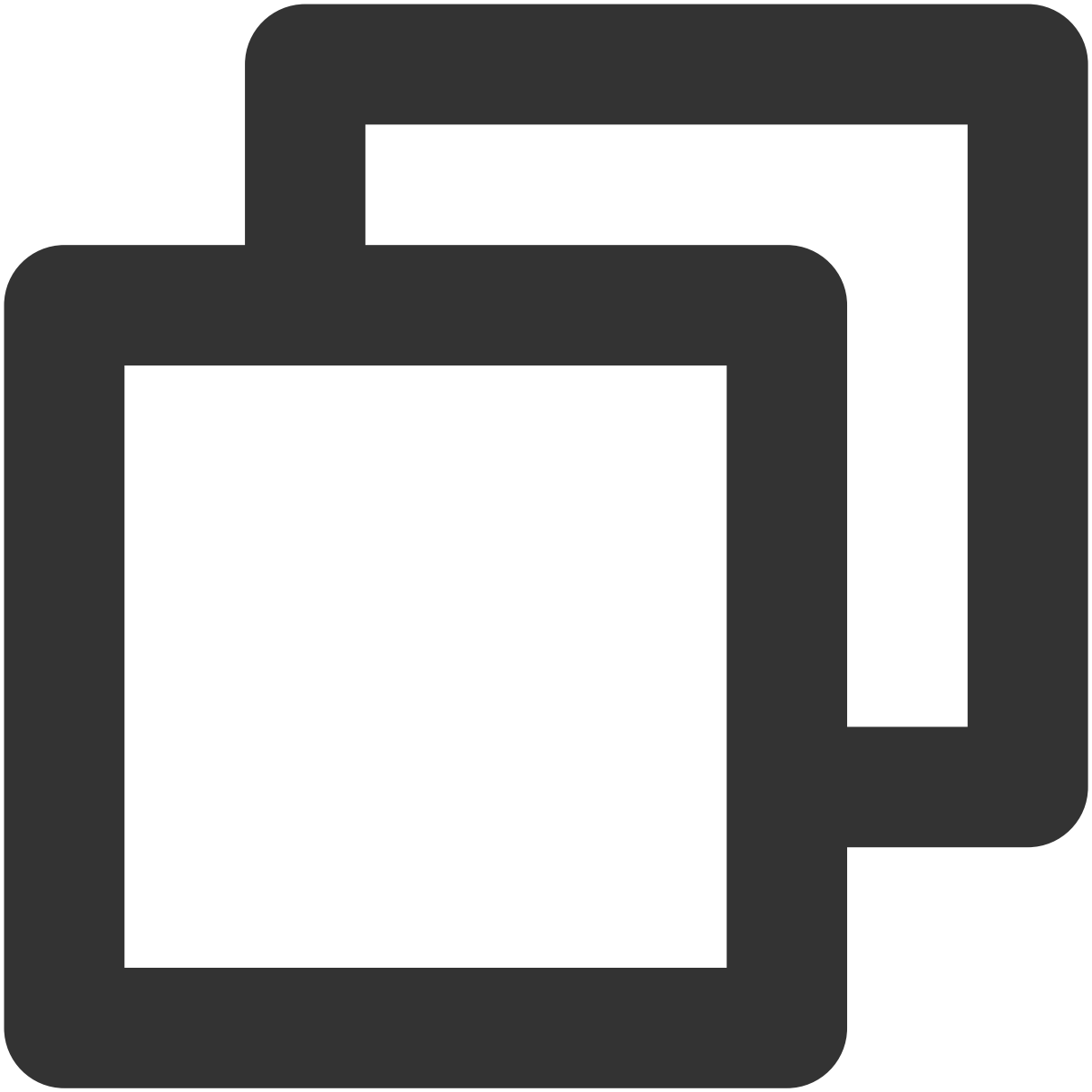
排查方法

统计云数据库 MongoDB 中的集合数量与索引数量，请使用如下命令分别统计：



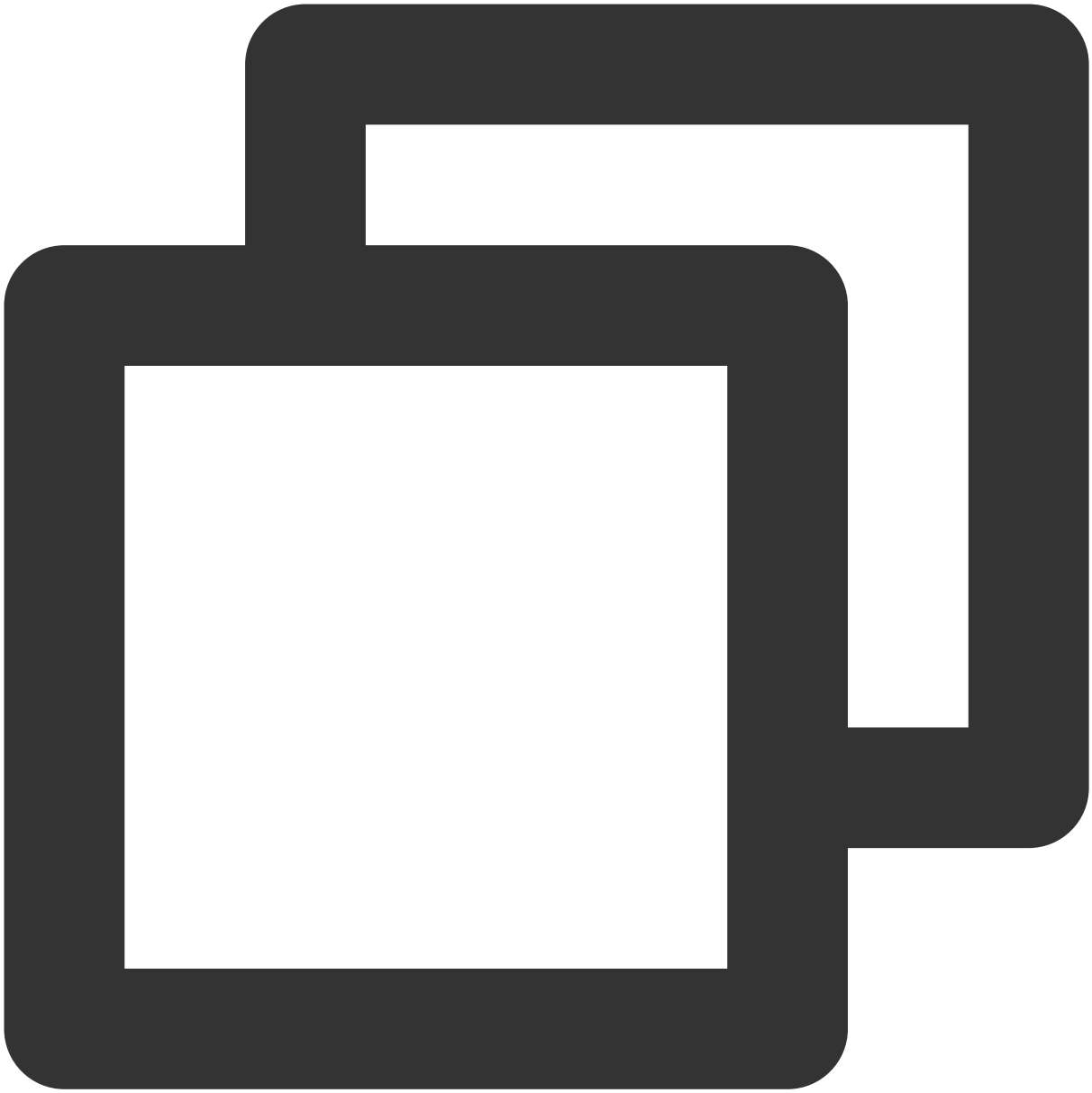
```
use <database>;
db.getCollectionNames().filter(function(c) { return !c.startsWith("system."); }).le
db.getCollectionNames().forEach(function(col) {     print("Indexes for " + col + ":
```

查询集合数量，执行示例如下所示：



```
mongos> db.getCollectionNames().filter(function(c) { return !c.startsWith("system."
4
```

查询每个集合的索引数量，执行示例如下所示：



```
mongos> db.getCollectionNames().forEach(function(col) {     print("Indexes for " +
Indexes for nba: 3
Indexes for t1: 1
Indexes for test: 1
Indexes for user: 1
```

使用建议

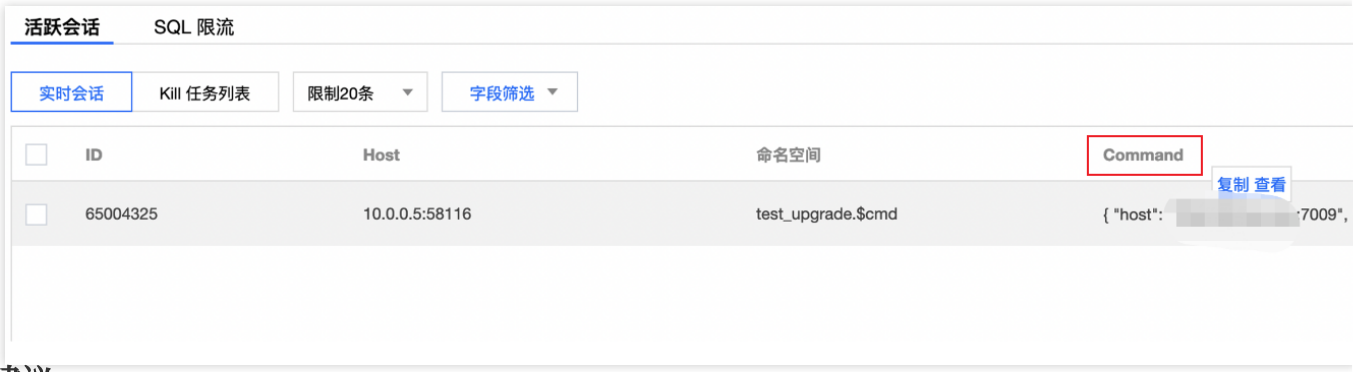
在 MongoDB 中，建议将集合的个数保持在数千个以内。否则，导致 MongoDB 管理这些集合时的性能显著下降。

创建索引过程的内存消耗

MongoDB 在创建索引时可能会存在占用大量内存的情况。在正常的业务数据写入情况下，从节点会维持一个约256M的 `buffer` 用于数据回放。在 MongoDB 4.2版本之前，主节点通过非后台方式创建索引时，后端回放创建索引是串行的，最多可能消耗500M内存。而在 MongoDB 4.2版本之后，默认废弃了 background 选项，允许从节点并行回放创建索引，导致消耗更多的内存。因此，如果在多个索引同时创建时，可能会导致实例内存溢出。另外，从节点在数据回放过程中也可能会消耗更多的内存，特别是在主节点完成索引创建后。

排查方法

在**数据库智能管家 DBbrain** 的 实时会话 页面的**活跃会话**页签，查看是否有多个正在添加索引的进程。如下图所示，可以在 **Command** 列查询到 CreateIndex 关键字段。



使用建议

避免在高峰期同时创建多个大型索引，可以将索引的创建时间分散开来。
尽量使用后台方式创建索引，以避免占用过多的内存。具体创建索引的方式，请参见 MongoDB 官网。
如果内存不足，可以考虑增加系统内存。具体操作，请参见 变更 Mongod 节点配置规格。
可以考虑使用更高效的索引类型，如文本搜索索引、地理位置索引等，以减少内存占用。请通过数据库智能管家（TencentDB for DBbrain，DBbrain）的 索引推荐 功能，选择最优索引。
对于较大的集合，可以考虑使用分片技术来分散数据和索引的存储，以减少单个实例的内存压力。

逻辑备份与主从切换产生的内存

逻辑备份通常会产生大量的数据扫描，导致内存占用较多。如果逻辑备份选择了从节点或者隐藏节点，将会出现从节点内存明显高于主节点的现象。
主从节点切换，原来的主节点变更为从节点，那么当前从节点内存较当前的主节点会明显偏高。

排查方法

1. 登录 MongoDB 控制台，请在**任务管理**页面确认是否有正在进行的备份任务。当前正在执行的任务通常在任务列表的最上面。单击**操作**列的**任务详情**，确认是否为**逻辑备份**。具体操作，请参见 任务管理。

2. 在**任务管理**页面的**任务类型**中，搜索**切换主节点，**确认业务侧是否进行了主从节点切换。
3. 单击**系统监控**页签，在左侧集群总览下拉菜单中，分别查看主节点与从节点的**内存使用率**。

使用建议

物理备份是通过物理文件拷贝来进行备份的。因此备份过程中消耗的内存相对较小。如果对从节点内存要求较高，请选择物理备份方式。各个版本所支持的备份方式，请参见 备份数据。
如果主从切换之后，业务侧没有设置读取从节点，请在控制台重启对应的从节点来释放内存，如果从节点有请求，建议在流量低峰期操作。

慢查询增多时延偏高原因分析与解决方法

最近更新时间：2024-05-07 16:12:47

现象描述

慢查询日志是用于记录查询时间超过阈值的查询日志，当系统中出现大量的慢查询时，会导致系统性能下降，响应时间变长，甚至可能导致系统崩溃。因此，需要对慢查询进行优化，减少慢查询的数量，提高系统性能。
登录 MongoDB 控制台，单击实例 ID 进入**实例详情**页面，选择**系统监控**页签，检查实例的监控数据。发现数据库时延监控类指标明显变长。时延监控指标主要反馈的是请求到达接入层直至处理完请求返回客户端的时间。具体监控指标项，请参见 监控概述。

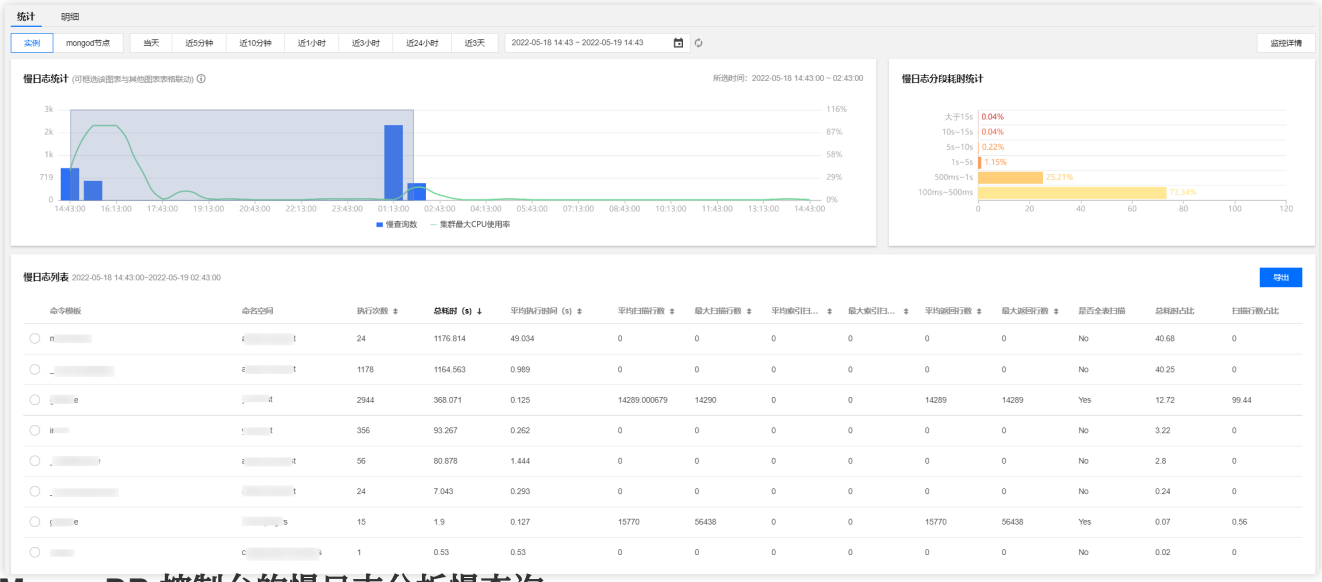
可能原因

通过 $lookup 运算符查询，不使用索引或者使用的索引不支持该查询，需要遍历整个数据库进行完整的扫描，导致检索效率很低。
集合中的文档使用了大量的搜索和索引的大型数组字段，搜索和索引数据集过大，导致系统负载过高。

分析慢查询

基于数据库智能管家 DBbrain 的慢 SQL 分析排查慢查询（推荐）

数据库智能管家（TencentDB for DBbrain，DBbrain）是腾讯云推出的一款为用户提供数据库性能优化、安全、管理等功能的数据库自治云服务。其中，针对 MongoDB 的慢 SQL 分析专用于分析 MongoDB 操作过程中产生的慢日志。具体诊断数据直观、易于查找，如下图所示。更多信息，请参见 慢 SQL 分析。



基于 MongoDB 控制台的慢日志分析慢查询

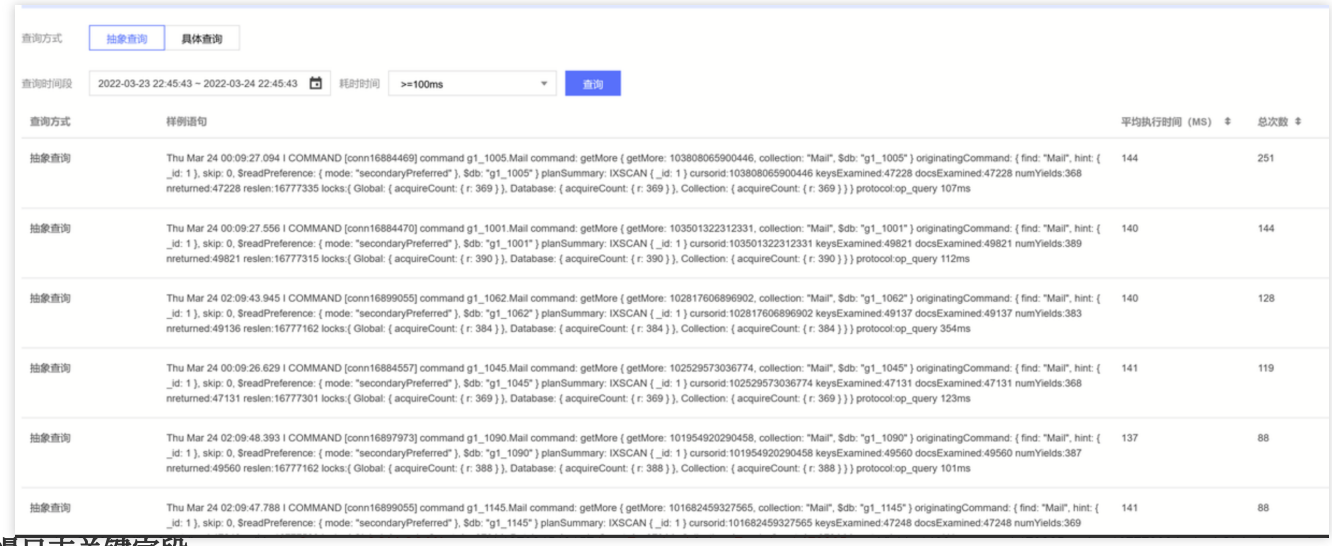
在 MongoDB 未接入数据库智能管家（TencentDB for DBbrain，DBbrain）之前，可在 MongoDB 控制台获取慢日志，逐一分析关键字段来排查慢查询的原因。

获取慢日志

1. 登录 MongoDB 控制台。
2. 在左侧导航栏 **MongoDB** 的下拉列表中，选择**副本集实例**或者**分片实例**。副本集实例与分片实例操作类似。
3. 在右侧实例列表页面上方，选择地域。
4. 在实例列表中，找到目标实例。
5. 单击目标实例 ID，进入**实例详情**页面。
6. 选择**数据库管理**页签，再选择**慢日志查询**页签。
7. 在**慢日志查询**页签，分析慢日志，系统会记录执行时间超过100毫秒的操作，慢日志保留时间为7天，同时支持下载慢日志文件，具体操作，请参见 慢日志管理。

抽象查询：经过对查询条件的模糊处理后的统计值，这里可以看到按平均执行时长排序的慢查询统计，我们建议先优化 Top5 的请求。

具体查询：记录完整的用户执行请求，包括：执行计划，扫描的行数，执行时长，锁等待的一些信息。



分析慢日志关键字段

在慢日志中查看关键字段。常见的关键字段的含义，请参见下表。更多字段描述，请参见 MongoDB 官网。

关键字段	字段含义
command	指慢日志中记录的请求操作。
COLLSCAN	指该查询进行了全表扫描。如果扫描的行数低于100，全表扫表的速度也会很快。
IXSCAN	说明进行了索引扫描。该字段后面会输出具体使用的哪一个索引。有可能一个表有多个索引，当这里的索引不符合预期时，应该考虑优化索引或者通过 hint()来改造查询语句。
keysExamined	指扫描索引的条目。"keysExined" : 0, # MongoDB 为执行操作而扫描的索引键的数量为0。
docsExamined	说明扫描的集合中的文档数。
planSummary	用于描述查询执行的摘要信息。每个 MongoDB 查询都会生成一个执行计划，该计划包含有关查询的详细信息，例如使用的索引、扫描的文档数量、查询的执行时间等。例如："planSummary" : "IXSCAN { a: 1, _id: -1 }" 表示 MongoDB 使用了索引扫描（IXSCAN）的查询计划。具体而言，它使用了名为 "a" 的索引以及默认的 "_id" 索引，并按照升序（1）的方式对 "a" 索引进行了扫描。这是一种常见的查询计划，表示查询使用了索引来获取所需的数据。
numYield	该字段表示操作在执行过程中让出锁的次数。当一个操作需要等待某些资源（例如磁盘 I/O 或锁）时，它可能会放弃 CPU 控制权，以便其他操作可以继续执行。这个过程被称为“让步”。numYield 的值越高，通常表示系统的负载越大，因为操作需要更多的时间来完成。通常，进行文档搜索的操作（查询、更新和删除）可交出锁。只有在其他操作列队等待该操作所持有的锁时，它才有可能交出自己的锁。通过优化系统中的让步次数，可以提高系统的并发性能和吞吐量，减少锁竞争，从而提高系统的稳定性和可靠性。

nreturned	指查询请求返回的文档数。这个值越大，代表返回的行数越多。如果 keysExamined 值很大，nreturned 返回的文档很少，说明索引有待优化。
millis	从 MongoDB 操作开始到结束耗费的时间，这个值越大，代表执行越慢。
IDHACK	用于加速查询或更新操作。在 MongoDB 中，每个文档都有一个 _id 字段，它是一个唯一标识符。在某些情况下，如果查询或更新操作中包含了 _id 字段，MongoDB 可以使用 IDHACK 技术来加速操作。具体来说， IDHACK 技术可以利用 _id字段的特殊性质，将查询或更新操作转换为更高效的操作。例如，如果查询条件是一个精确匹配的 _id值，IDHACK 可以直接使用 _id 索引来查找文档，而不需要扫描整个集合。
FETCH	该字段表示在执行查询操作时，MongoDB 从磁盘读取的文档数量。当执行查询操作时，MongoDB 会根据查询条件和索引等信息，从磁盘中读取匹配的文档。 FETCH 字段记录了这个过程中读取的文档数量。通常情况下 FETCH 字段的值越小，表示查询性能越好。因为 MongoDB 可以尽可能地利用索引等技术，减少从磁盘中读取文档的次数，从而提高查询性能。

解决方法

清理慢查询

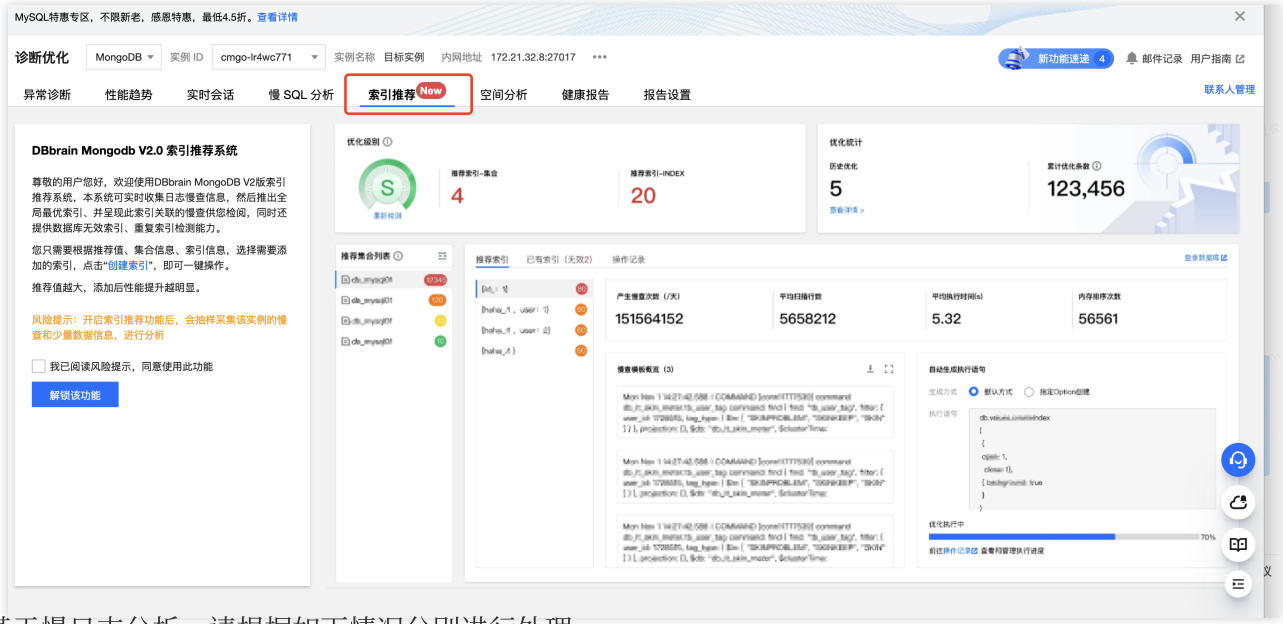
1. 选择**数据库管理** > **慢查询管理**页签，列表会展示当前实例正在执行的请求（包括从节点的请求），您可单击**批量 Kill** 对不需要的慢查询请求进行 Kill 操作。具体操作，请参见 慢日志管理。
2. 对于非预期的请求，可在**数据库智能管家**（TencentDB for DBbrain，DBbrain）的**诊断优化**中，在**实时会话**页面，直接进行 Kill 操作，进行清理。具体操作，请参见 实时会话。

SQL 限流

云数据库 MongoDB 4.0 版本，在**数据库智能管家**（TencentDB for DBbrain，DBbrain）的**诊断优化**的**SQL 限流**页面，创建 SQL 限流任务，自主设置 SQL 类型、最大并发数、限流时间、SQL 关键词，来控制数据库的请求访问量和 SQL 并发量，从而达到服务的可用性。具体操作及应用案例，请参见 SQL 限流。

使用索引

若是基于数据库智能管家（TencentDB for DBbrain，DBbrain）的慢 SQL 分析，请在慢日志列表中，关注扫描行数是否很大，说明有大扫描的请求或者长时间运行的请求。
全表扫描引起的慢查询增多，请通过创建索引可减少集合的扫描、内存排序等。创建索引，请参见 MongoDB 官方 Indexes。
若使用了索引，而索引的扫描行数为 0，而扫描的行数大于0，说明索引需要优化，请通过数据库智能管家（TencentDB for DBbrain，DBbrain）的 索引推荐 功能，选择最优索引。索引推荐通过实时日志慢查信息的收集，进行自动分析，推出全局最优索引，并按照性能影响进行排列，推荐值越大操作后性能提升越显著。

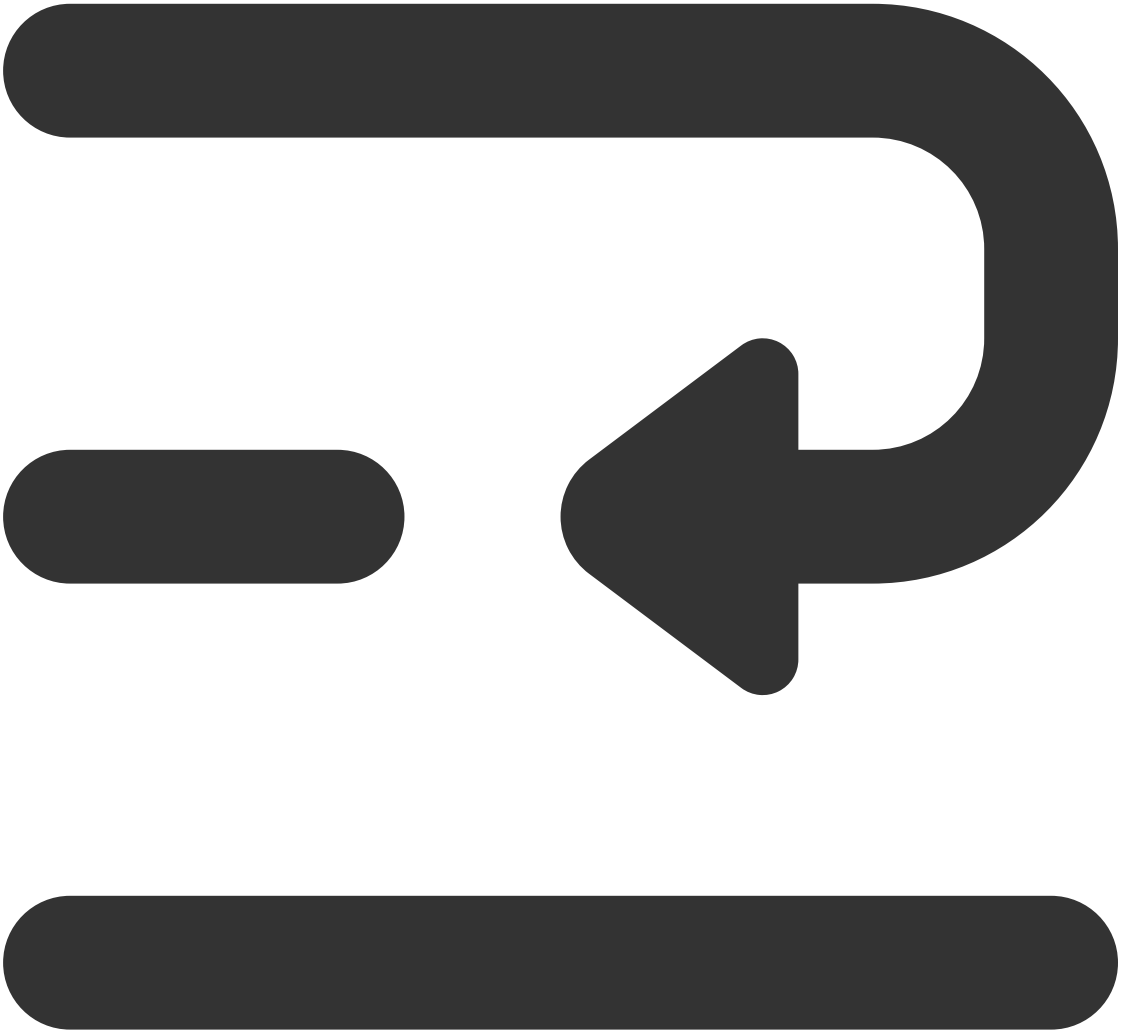


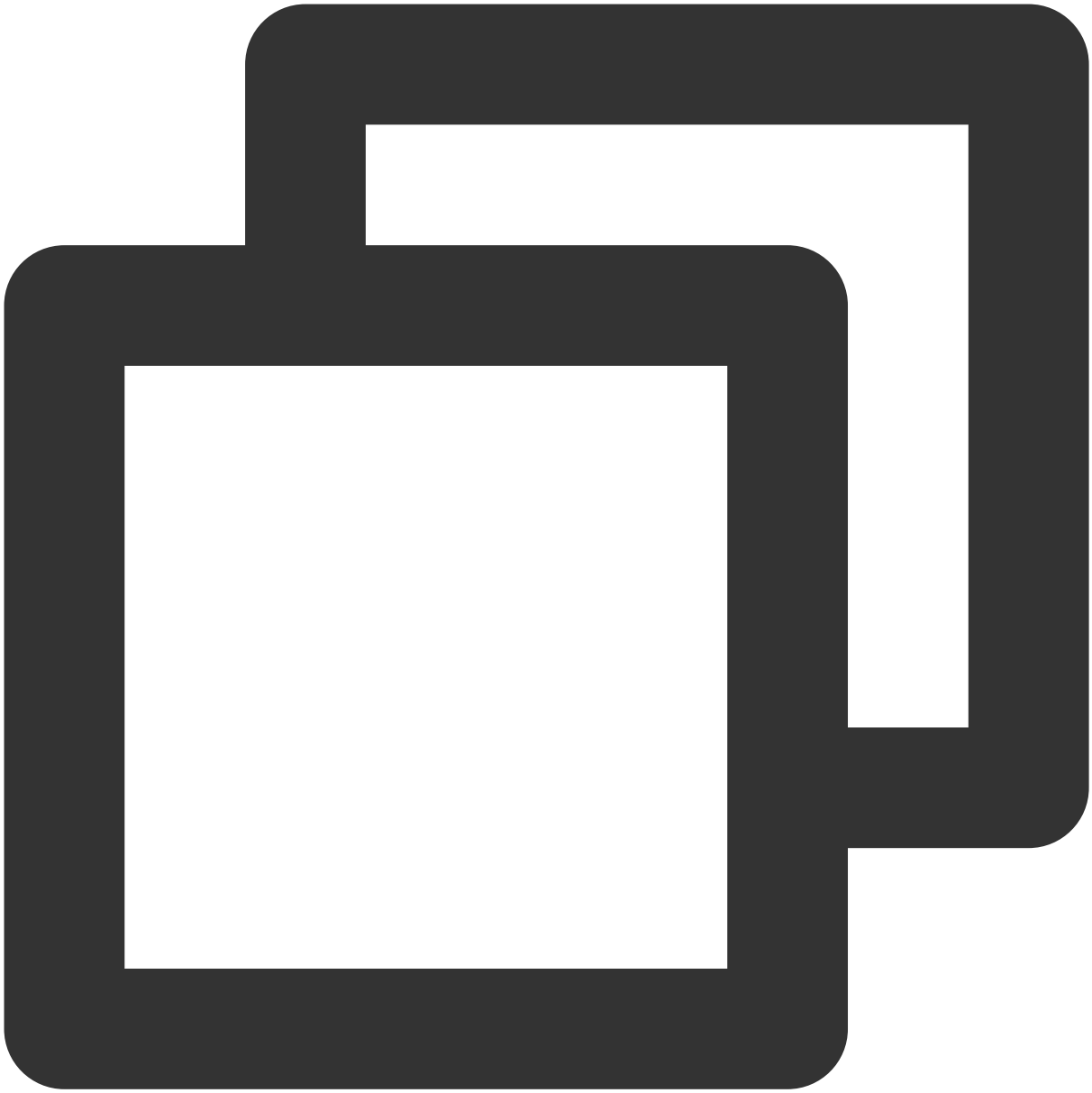
若是基于慢日志分析，请根据如下情况分别进行处理。

keysExamined = 0，而 docsExamined > 0 ，planSummary 为 COLLSCAN ，说明执行了全表扫表，引起查询延迟明显，如下所示。创建索引，请参见 MongoDB 官方 Indexes。

keysExamined > 0，而 docsExamined > 0，planSummary 为 IXSCAN，说明有部分查询条件或者返回的字段，索引里面并不包含，需进行索引优化。请通过数据库智能管家（TencentDB for DBbrain，DBbrain）的 索引推荐 功能，选择最优索引。

关键字段 keysExamined > 0，而 docsExamined = 0 planSummary 为 IXSCAN，说明查询条件或返回的字段，索引已经包含，如果 keysExamined 值很大，建议优化索引的字段顺序、或者添加更合适的索引进行过滤。更多信息，请参见 索引优化解决读写性能瓶颈。





```
Thu Mar 24 01:03:01.099 I COMMAND [conn8976420] command tcoverage.ogid_mapping_info
```

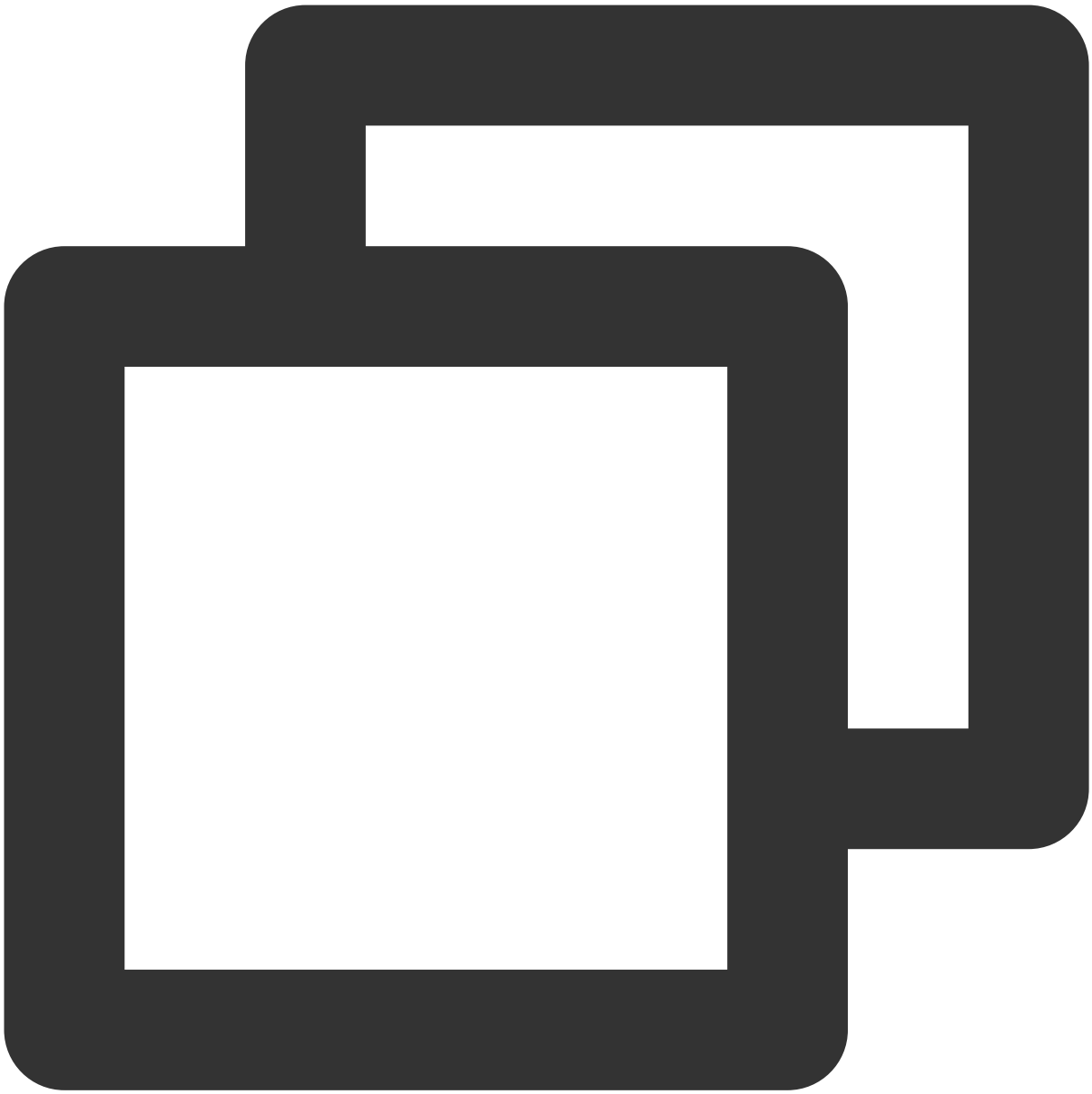
如果 MongoDB 为 4.2 之前版本，已确认业务查询所用索引并无问题，请确认当前是否在业务繁忙时段，进行了前台方式创建索引操作，请更改为后台方式。

说明：

前台方式创建索引：MongoDB 4.2 之前的版本一个集合创建索引时默认方式为前台方式，即将参数 background 选项的值设置为 false，该操作将阻塞其他所有操作，直到前台完成索引创建。

后台方式创建索引：即将 background 选项的值设置为 true，则在创建索引期间，MongoDB 依旧可以正常提供读写操作服务。然而，后台方式建索引可能会导致索引创建时间变长。具体创建索引的方式，请参见 MongoDB 官网。

后台方式创建索引，请通过 `currentOp` 命令来查看当前创建索引的进度。具体的命令，如下所示。



```
db.adminCommand(
    {
      currentOp: true,
      $or: [
        { op: "command", "command.createIndexes": { $exists: true }  },
        { op: "none", "msg" : /^Index Build/ }
      ]
    }
)
```

返回如下图所示，`msg` 字段指明当前创建索引的进度，`locks` 字段代表该操作的锁类型。更多锁说明信息，请参见 MongoDB 官网。

连接数超限解决方法

最近更新时间：2024-05-07 10:19:05

现象描述

针对连接数超限问题，用户可使用 控制台 的连接数管理、重启功能，以及优化业务侧服务来进行排查处理。

可能原因

连接泄露，客户端不规范的编码，连接池配置不合理，出现较多未正常释放连接池的问题。
存在大量并发应用程序请求，数据库当前的规格无法满足当前需求量，连接数配置不足。

处理步骤

方式一：提升连接数

1. 登录 MongoDB 控制台，单击实例 ID，进入实例管理页面。
2. 选择**数据库管理** > **连接数管理**页，查看连接的来源 IP 和对应的连接数，进行业务端的排查。

说明：

若您的连接数达到80%或以上，影响到新连接的建立，可通过控制台一键**提升连接数**功能，在6小时内提升连接数的上限至150%。
若提升连接数至150%还不能解决您的问题，请联系售后或 提交工单 处理。

方式二：重启实例

1. 登录 MongoDB 控制台，单击实例 ID，进入实例管理页面。
2. 选择**数据库管理** > **连接数管理**页，必要时您可单击右侧的**重启**对 Mongos 进行重启，以解决连接数超限的问题。

说明：

4.0 版本副本集实例无 Mongos。

重启 Mongod 为高危操作，会出现连接闪断，如果同时有数据正写入，可能会造成 rollback 进而丢失数据，默认不开放，如需开放请联系售后或 提交工单。

方式三：优化业务侧服务

通过对业务端的排查对业务侧服务进行优化。具体排查方法，请参见 连接使用率偏高异常分析及解决方法。

磁盘空间利用率偏高解决方法

最近更新时间：2024-05-07 16:17:03

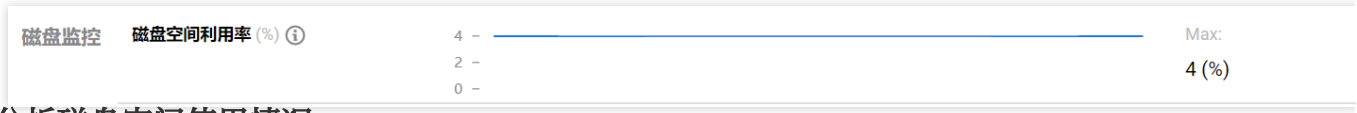
操作场景

MongoDB 的磁盘主要存储数据和索引，以及一些系统文件和日志文件。磁盘空间利用率是一个非常重要的监控指标，当磁盘空间被完全使用时，MongoDB 实例将无法继续写入新的数据，这将导致实例出现故障并停止工作。因此，监控磁盘利用率并及时采取措施来释放磁盘空间，是确保 MongoDB 实例正常运行的关键。

查看磁盘空间使用情况

快速查看监控指标

登录 MongoDB 控制台，在**系统监控**页签，可查看**磁盘空间利用率**的变化趋势视图。具体操作，请参见 查看监控数据。



详细分析磁盘空间使用情况

登录 MongoDB 控制台，在左侧导航栏，选择**诊断优化**，再选择**空间分析**页签。可通过数据库智能管家（TencentDB for DBbrain，DBbrain）的空间分析功能，进一步分析数据库的磁盘空间的使用详情，包括数据库的集合空间、索引空间、物理文件空间大小，以及数据库大小、数据占比、集合的行数等分析数据及视图等。具体操作，请参见 空间分析。

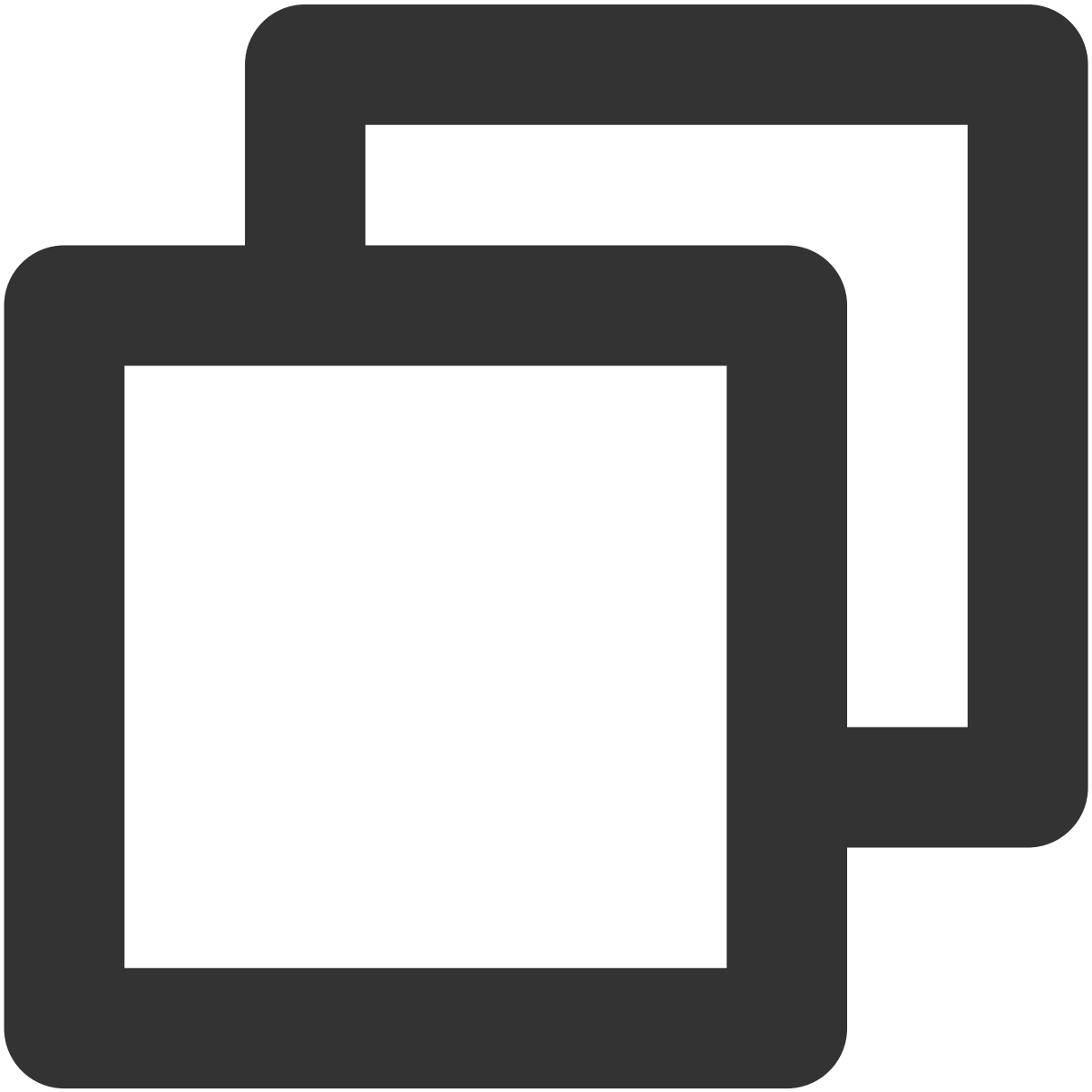
通过 MongoDB 自身提供的命令 `db.stats()` 和 `db.$collection_name.stats()` 分析磁盘空间使用情况。具体信息，请参见下表。

分析命令	命令含义
db.stats()	该命令用于获取当前数据库的统计信息。执行 `db.stats()` 命令将返回一个文档，其中包含有关当前数据库的各种信息，如：数据库名称、数据大小、索引大小、集合数量等。
db.collection.stats()	该命令用于获取指定集合的统计信息。执行 `db.collection.stats()` 命令将返回一个文档，其中包含有关指定集合的各种信息，如：集合名称、文档数量、数据大小、索引数量等。
db.collection.storageSize()	该命令用于获取指定集合占用的存储空间大小。执行 `db.collection.storageSize()` 命令将返回指定集合的存储空间大小，单位为字节。该命令所返回的存储空间大小包括集合中的数据和索引等占用的空间，但不包括 MongoDB 实例的其他开销，如日志文件和临时文件等。
db.collection.totalIndexSize()	该命令用于获取指定集合的所有索引占用的存储空间大小。执行 `db.collection.totalIndexSize()` 命令将返回指定集合的所有索引占用的存储空间大小，单位为字节。该命令所返回的存储空间大小不包括集合中的数据占用的空间，只包括集合的所有索引占用的空间。
db.collection.totalSize()	该命令用于获取指定集合占用的总存储空间大小。执行 `db.collection.totalSize()` 命令将返回指定集合占用的总存储空间大小，单位为字节。该命令所返回的存储空间大小包括集合中的数据和索引等占用的空间，以及 MongoDB 实例的其他开销，如日志文件和临时文件等。

问题分析

云数据库 MongoDB 默认使用的是 WiredTiger 引擎，删除文档时，并不会直接回收磁盘空间。当插入新的数据时，MongoDB 会重用之前占用的空间，而不会继续额外占用新的磁盘空间。随着删除的操作增多，碎片也会越来越多。如下代码，可一次性查看指定数据库的所有 collection 碎片率。

当实例磁盘空间利用率达到80%~85%以上时，可通过降低数据库实际占用空间或扩容存储空间的方法避免空间占满的风险。



##生成查看碎片率的函数

```
function getCollectionDiskSpaceFragRatio(dbname, coll) {
    var res = db.getSiblingDB(dbname).runCommand({
        collStats: coll
    });
    var totalStorageUnusedSize = 0;
    var totalStorageSize = res['storageSize'] + res['totalIndexSize'];
    Object.keys(res.indexDetails).forEach(function(key) {
        var size = res['indexDetails'][key]['block-manager']['file bytes available
        print("index table " + key + " unused size: " + size);
        totalStorageUnusedSize += size;
    });
    var size = res['wiredTiger']['block-manager']['file bytes available for reuse']
    print("collection table " + coll + " unused size: " + size);
    totalStorageUnusedSize += size;
    print("collection and index table total unused size: " + totalStorageUnusedSize
    print("collection and index table total file size: " + totalStorageSize);
    print("Fragmentation ratio: " + ((totalStorageUnusedSize * 100.0) / totalStorag
}
##指定数据库，查看所有集合的碎片率
use xxxdb
db.getCollectionNames().forEach((c) => {print("\\n\\n" + c); getCollectionDiskSpace
```

解决方法

磁盘使用率高且碎片率高

云数据库 MongoDB 为 4.4及以上版本

若磁盘使用率高（一般80%~85%以上），碎片率较高（一般超过25%以上，才有做回收的收益），云数据库 MongoDB 为 4.4及以上版本副本集架构，请使用命令 `db.runCommand({compact:"collectionName"})` 对指定的集合文档进行压缩来释放磁盘空间。其中，`collectionName` 为集合名称，请根据实际情况替换。

说明：

当 MongoDB 执行 Compact 操作时，会对整个数据库进行压缩。因此在此期间，集合的创建和删除、索引的创建和删除等操作会被阻塞，而其他操作（如查询）则不会受到影响，但是存在负载影响，请求会有时延。建议在业务低峰期执行该操作。

云数据库 MongoDB 为4.4之前版本

若磁盘使用率高（一般80%~85%以上），碎片率较高（一般超过25%以上，才有做回收的收益），云数据库 MongoDB 为 4.4以下版本，不建议操作 `compact`，会阻塞实例的所有请求，可能会出现 Compact 索引无效的问题，解决方案如下：

升级版本，建议升级到最新版本以获得更好的性能和稳定性。具体操作，请参见 版本升级。

如果不想升级版本，可以通过逻辑迁移来重建节点达到收缩空间的效果，操作过程中会存在多次闪断。具体操作，请联系售后或 提交工单。

磁盘使用率高但碎片率不高

若磁盘使用率高（一般80%~85%以上），而碎片率并不高，碎片率低于20%，不建议进行 `compact` 操作，因为 MongoDB 会复用这部分空间。此时请扩容磁盘空间，具体操作，请参见 变更 Mongod 节点配置规格。