mongo杂谈及其在评论中的应用

- 分享人：青木铃一
- 部　门：社交平台产品中心

- 时　间：2020年12月

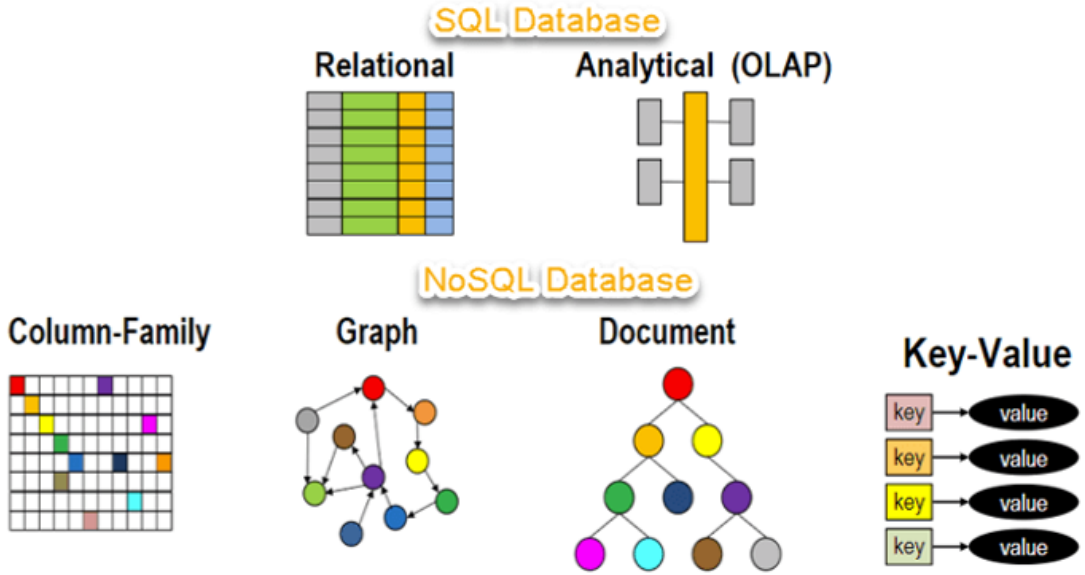
内容

1. mongo简介
2. mongo架构和原理
3. mongo索引
4. mongo在评论中的应用
5. 总结

mongo简介

RDBMS vs NoSQL

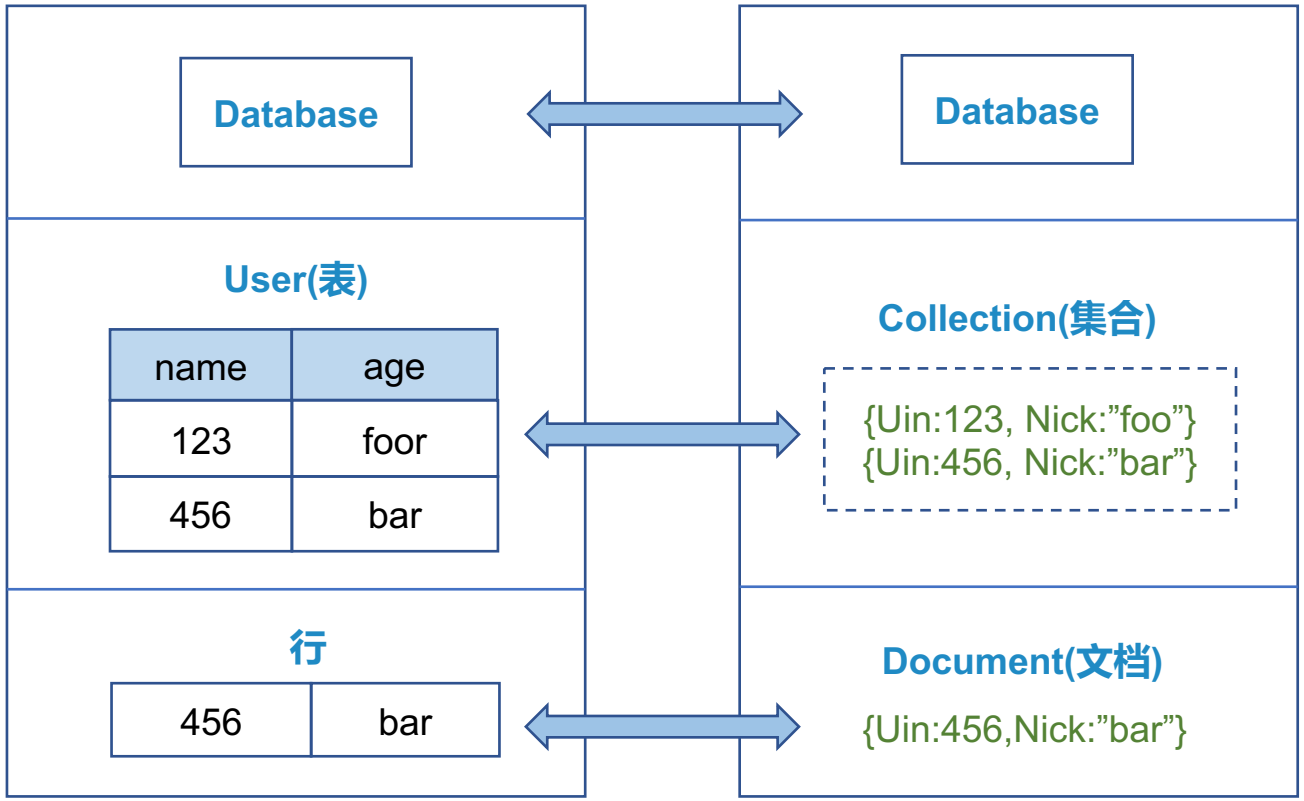
- NoSQL（Not Only SQL）是对不同于传统关系数据库的数据库管理系统的统称。
- 常见的类型有：列存储、键值、图像存储、文档存储



存储类型	NoSQL	
键值存储	内存键值存储	Memcached、Redis
	持久化键值存储	BigTable、LevelDB
文档存储	MongoDB、CouchDB	
图存储	Neo4JI、FlockDB	
列存储	Hbase、Cassandra	

MySql vs MongoDb

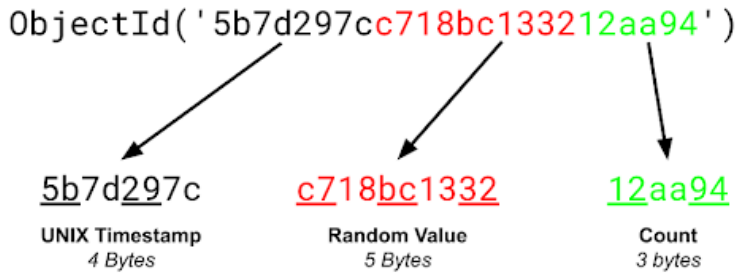
MongoDb是一种面向文档的NoSQL型分布式数据库，是非关系数据库当中功能最丰富，最像关系型数据库的。



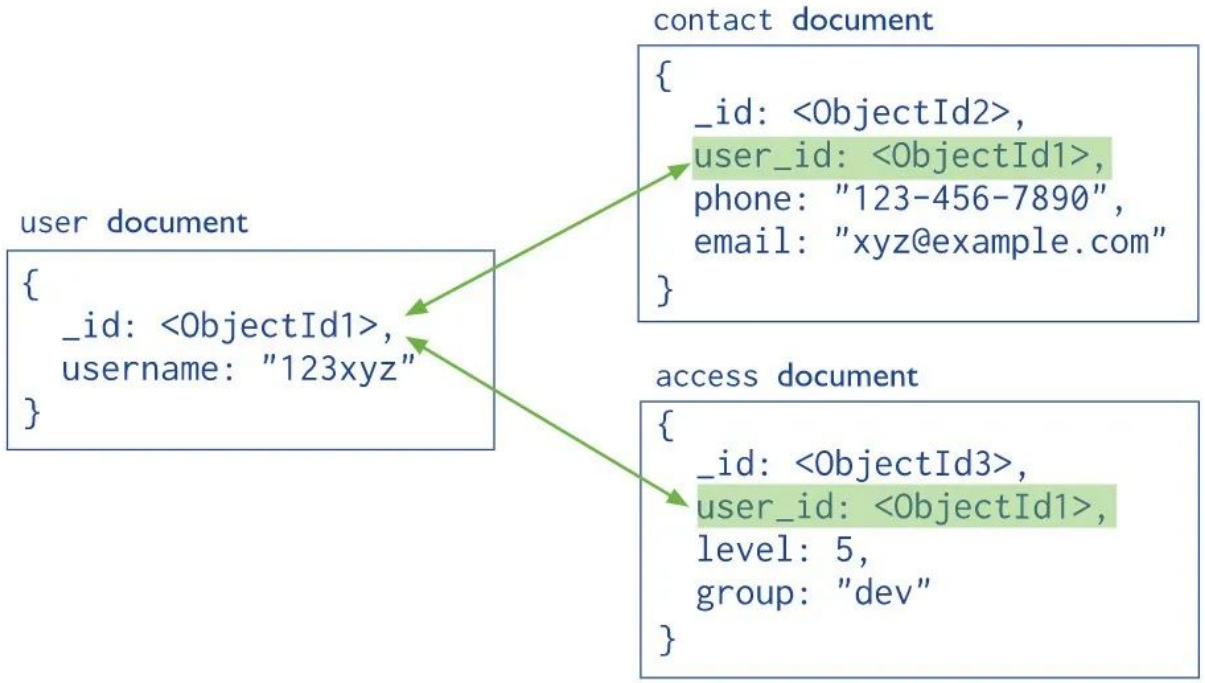
MySql术语	MongoDB术语
database	database
表（table）	集合（collection）
行（row)	文档（document）
列（column）	字段（field）
index	index
table joins	embedded documents and linking
主键（唯一值的列或者多列的组合）	_id字段
aggregation	aggregation pipeline

MySql和MongoDb的简单类比

常见数据模型



3.4版本以上的ObjectId格式



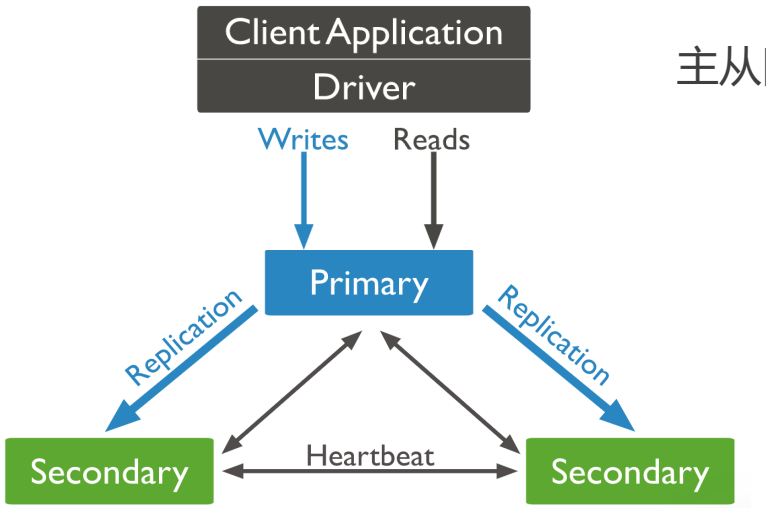
（1）引用

（2）内嵌

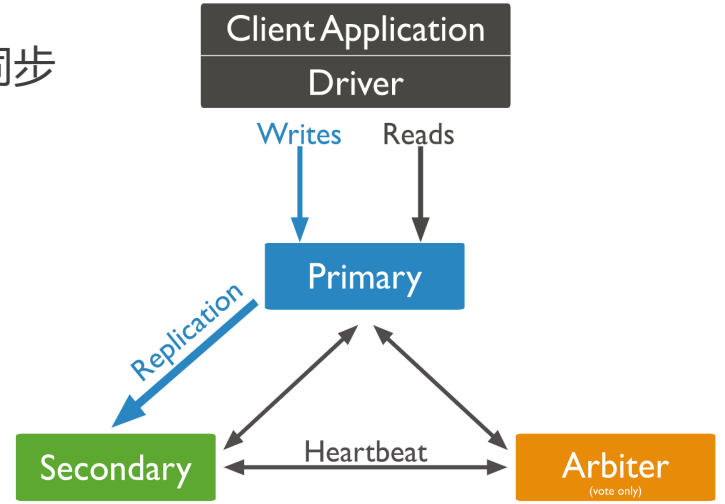
mongo架构和原理

复制集

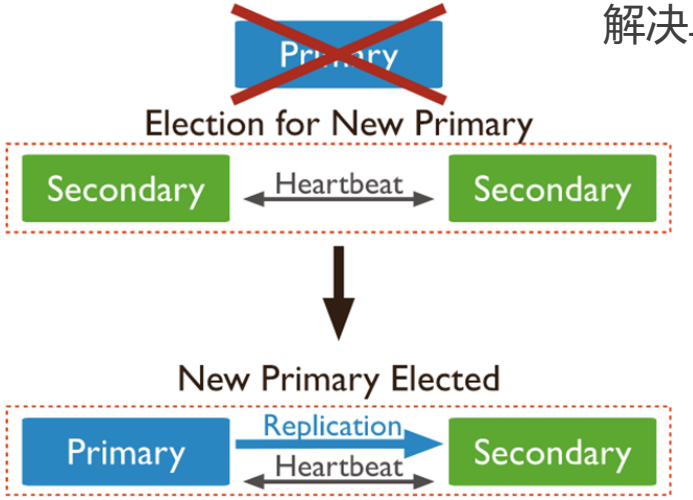
- 数据冗余和高可靠性
- 横向水平扩展
- 高可用保证
- 有primary和secondary等多种角色

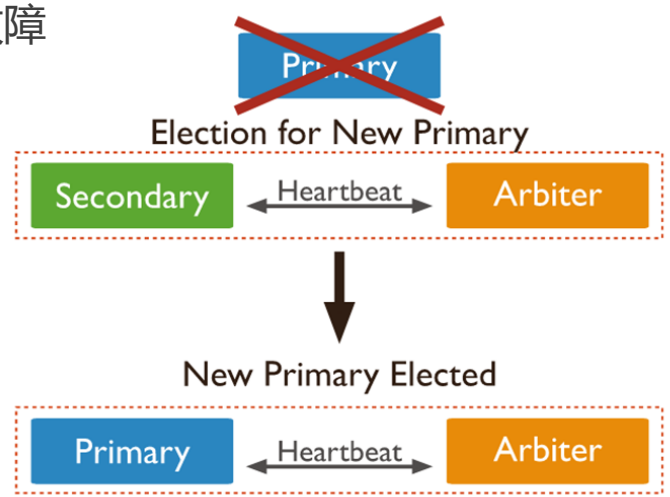


主从同步



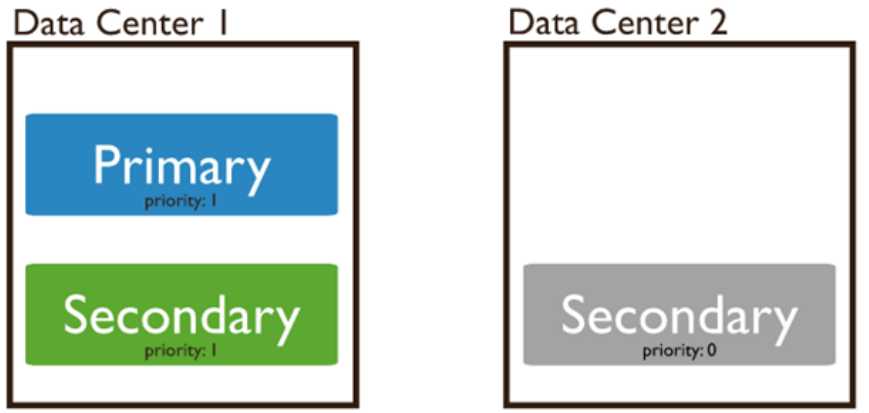
解决单点故障





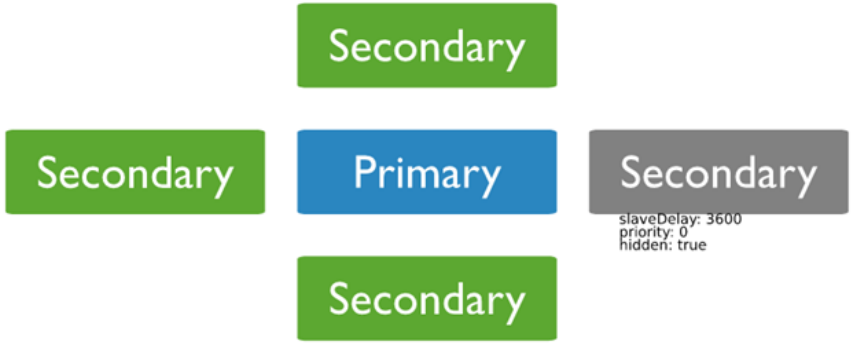
复制集的角色成员

成员	说明
Secondary	可参与Primary选举，并从Primary同步最新写入的数据。可提供读服务，增加Secondary节点可以提供复制集的读服务能力，同时提升复制集的可用性。
Arbiter	只参与投票，不能被选为Primary，并且不从Primary同步数据。Arbiter本身不存储数据，是非常轻量级的服务，当复制集成员为偶数时，最好加入一个Arbiter节点，以提升复制集可用性。
Priority0	Priority0节点的选举优先级为0，不会被选举为Primary。
Hidden	Hidden节点不能被选为主，并且对Driver不可见。因此Hidden节点不会接受Driver的请求，可用来做一些数据备份、离线计算的任务，不会影响复制集的服务。
Delayed	Delayed节点必须是Hidden节点，并且其数据落后与Primary一段时间（时间可配置）。当错误或者无效的数据写入Primary时，可通过Delayed节点的数据来恢复到之前的时间点。



（1）priority0

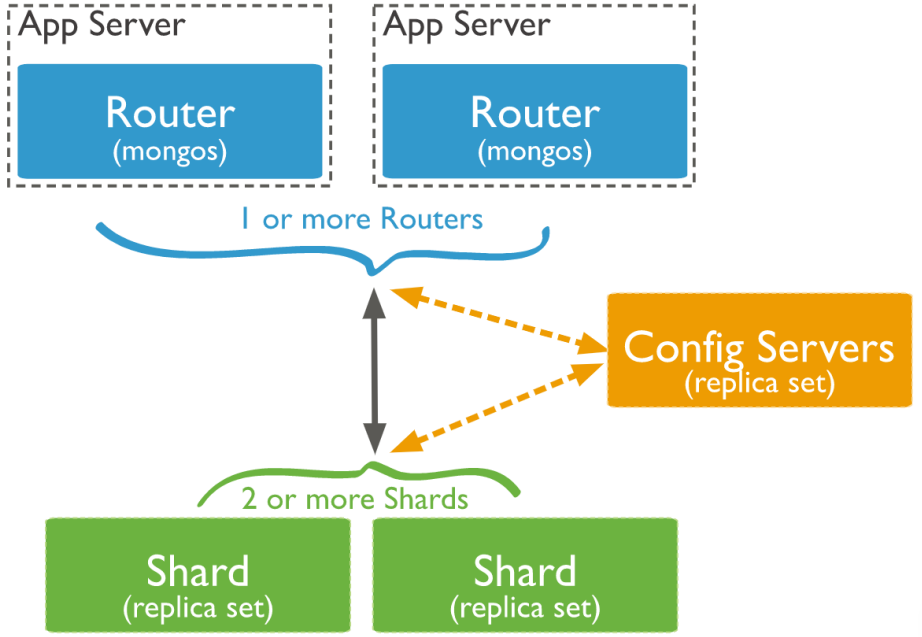
（2）hidden



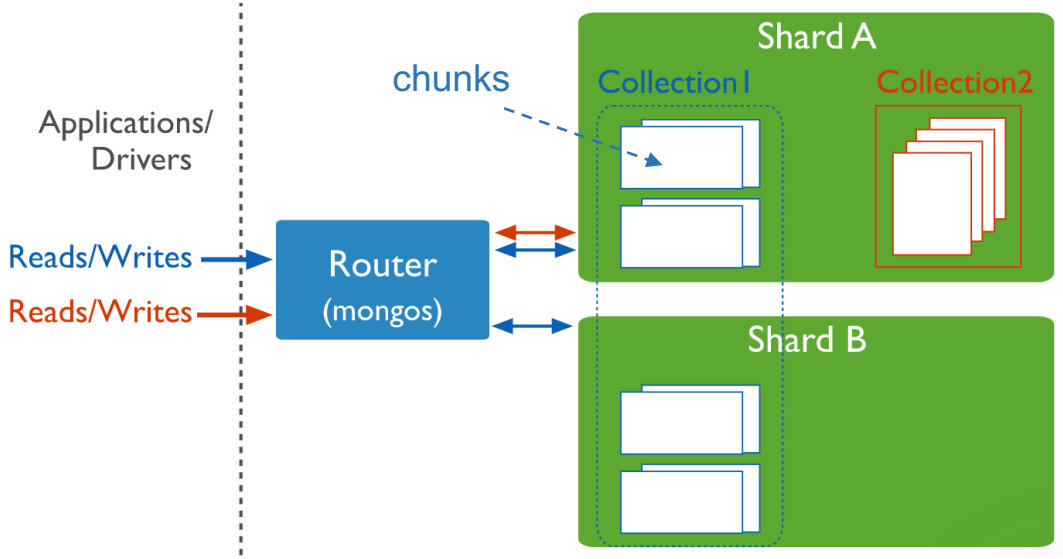
（3）delayed

分片集群

- 单复制集只能垂直扩展，吞吐量和存储容量存在上限。
- 分片集群提供了水平扩展的能力。

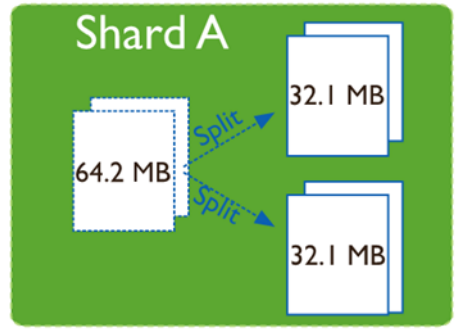


分片组件	说明
Config Server	存储集群所有分片节点、分片路由信息等元数据。
Mongos	和客户端打交道的路由模块，mongos本身不存储数据，其会将客户端的读写请求根据分片键的不同路由到不同的分片复制集上，同时会把接收到的响应拼装起来返回给客户端。
Mongod	实际存储文档和索引数据的节点，以chunk为单位进行存储。

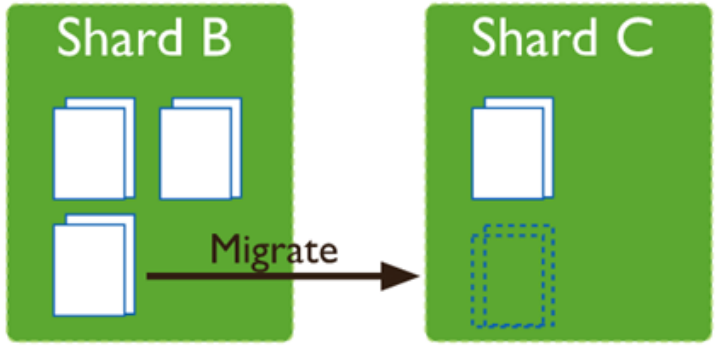


chunk和chunk均衡

- shard server会将一个集合的数据划分为多个chunks来存储。
- chunk的大小默认是64M，当超过64M，chunk会进行分裂。
- chunk会被自动均衡，由balancer后台进程进行迁移。
- chunk size是可调节的，过小容易出现jumbo chunk而无法迁移（即shard key的某个值出现频率很高，这些文档只能放到一个chunk 里，无法再分裂）；chunkSize过大则可能导致chunk内文档数过多而无法迁移。
- chunk的分裂和迁移非常消耗IO资源，插入和更新操作都有可能触发。



chunk裂变一分为二



chunk的迁移

balance相关操作

- **获取分片集群balance开启状态**

```
sh.getBalancerState()
```

- **查看分片集群balance是否开启**

```
sh.isBalancerRunning()
```

- **开启balance**

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

- **设置balance窗口**

```
use config
db.settings.update(
{ _id : "balancer" },
{ $set: {activeWindow:{ start:"00:00", stop:"5:00" }}}, true)
```

- **删除balance窗口**

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

- **停止balance**

```
sh.stopBalancer()
```

- **关闭某个集合的balance**

```
sh.disableBalancing("cm.comments")
```

- **打开某个集合的balance**

```
sh.enableBalancing("cm.comments")
```

```
mongos> use config;
switched to db config
mongos> show tables;
actionlog
changelog
chunks
collections
databases
lockpings
locks
migrations
mongos
settings
shards
system.sessions
tags
transactions
version
```

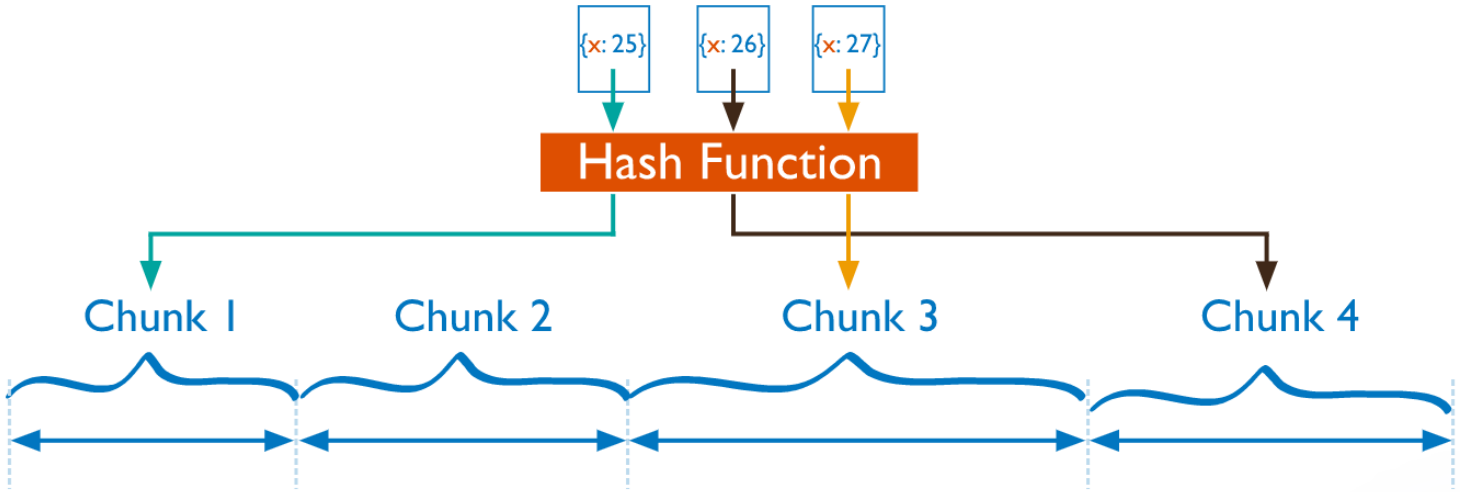
config db

分片键

- mongodb是根据片键（shard key）来对集合中的数据进行拆分的。
- 分片键一旦确定就不能再改变，且必须要有索引，键的大小最大为512bytes。
- 分片键可用于路由查询，避免请求广播。
- 对于片键，mongodb有范围分片和哈希分片两种方式。
- 集合指定片键后，其每个文档都必须包含片键字段。

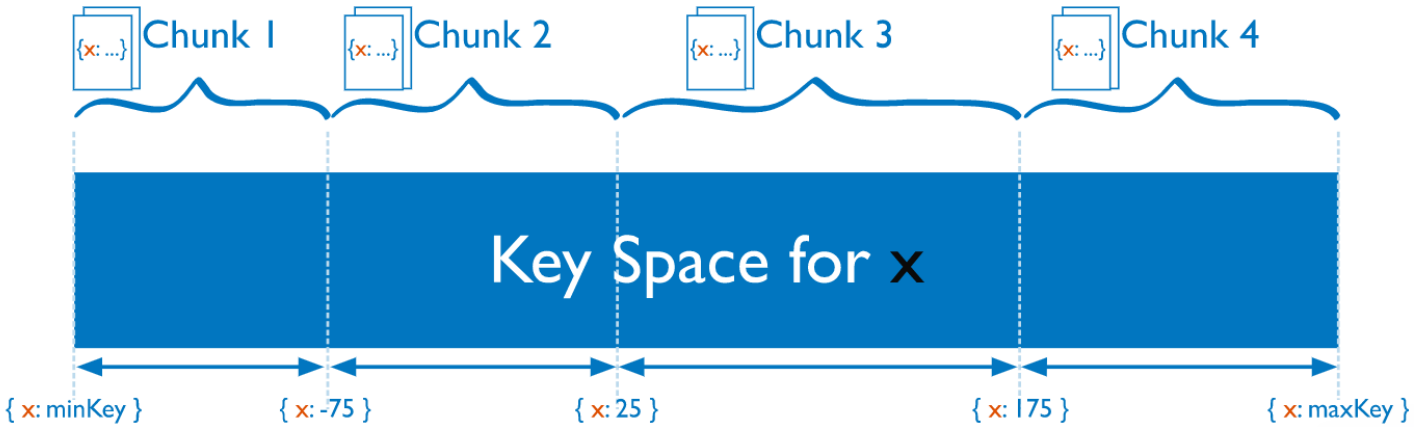
哈希分片

- 基于hash索引键来进行分片，数据在各个节点分布基本均匀。
- 片键值“相似”的文档很可能不会存储在同一个数据块，数据的分离性更好。
- 能将文档随机的分散到各个chunk，充分的扩展写能力。
- 片键只能使用一个字段。



范围分片

- 基于片键的取值范围来进行chunk的划分，片键可以由多个字段组合而成。
- 取值“相似”的片键很有可能存储在同一个chunk上。
- 集中写入带有相邻片键值的文档时，会受限于单个shard的写入性能，并有可能触发频繁的chunk分裂和迁移。
- 查询某一片键范围内的值会相对高效，减少了请求广播。



片键设计

针对不同的业务场景选择不同的分片策略，设计片键需要考虑的要素：

1. 读请求和写请求的分布。
2. 数据块的大小。
3. 每个查询需要路由的分片数。

一个好片键的特点：

1. 将插入数据均匀分布到各个分片上。
2. 保证CRUD能利用局限性。
3. 有足够的粒度进行块拆分。

案例：

1. 单key范围分片

```
sh.shardCollection(“user.log", { Ts: 1 })
```

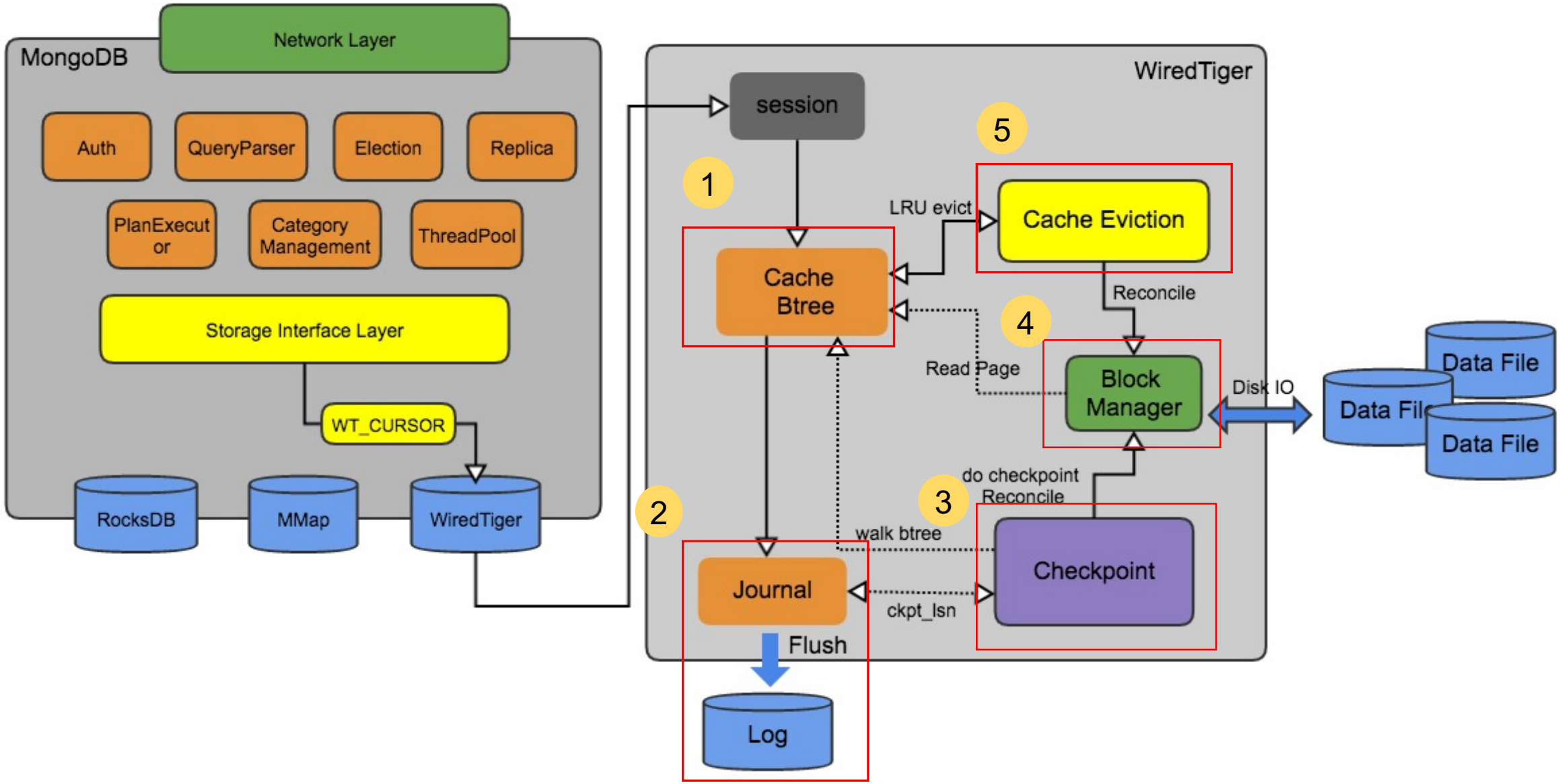
2. 单key hash分片

```
sh.shardCollection(“user.log", {"Uin":"hashed"})
```

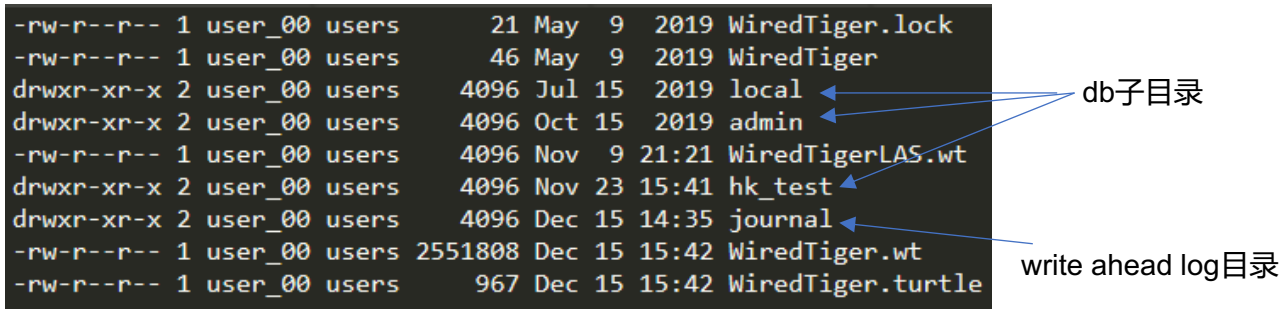
3. 组合key范围分片

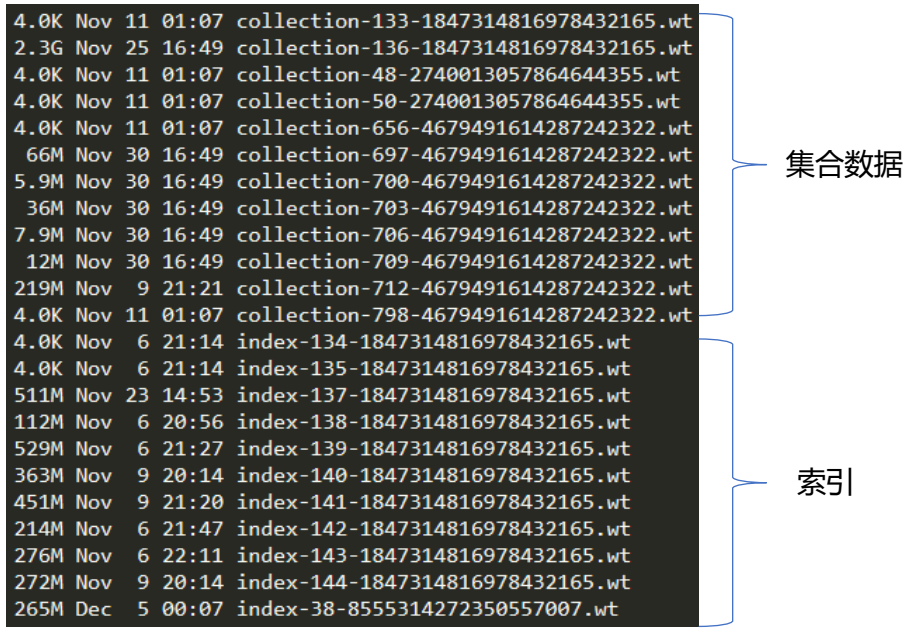
```
sh.shardCollection(“user.log", { “Uin”: 1, “Ts”:1 })
```

WiredTiger存储引擎



mongodb磁盘文件





每个db子目录下的数据和索引文件

- **WiredTiger.wt文件**

存储的是所有集合（包含系统自带的集合）相关数据文件和索引文件的checkpoint信息。

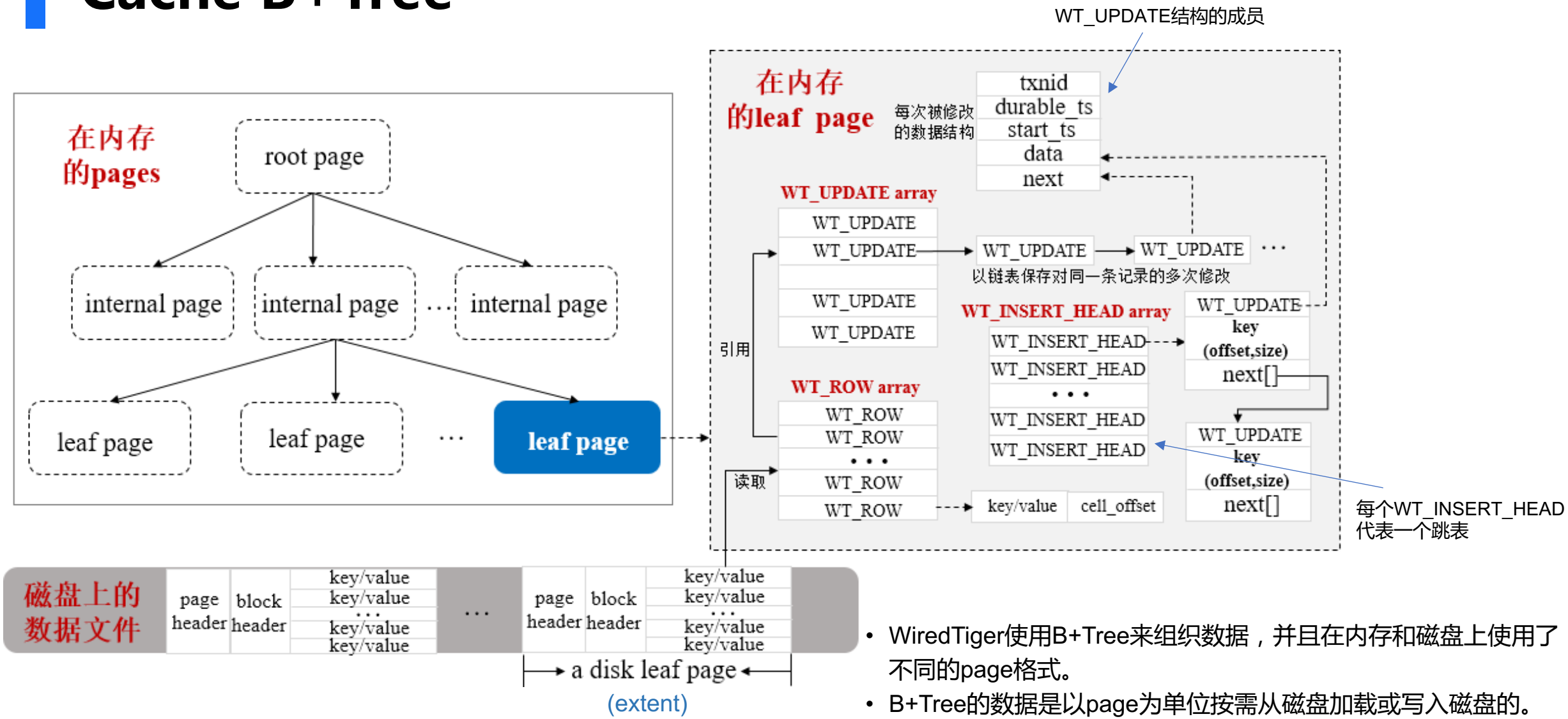
- **collection-xxx.wt和index-xxx.wt类文件**

数据库中集合所对应的数据文件和索引文件。

- **journal文件夹**

Jouranl日志功能开启后，该目录存放write ahead log事务日志，当数据库意外crash重启时，可通过checkpoint和WAL log来恢复数据。

Cache B+Tree



- WiredTiger使用B+Tree来组织数据，并且在内存和磁盘上使用了不同的page格式。
- B+Tree的数据是以page为单位按需从磁盘加载或写入磁盘的。
- 采用copy on write的方式管理修改操作（CUD），修改结果会先缓存在cache里。

Checkpoint

- **allocated list pages**

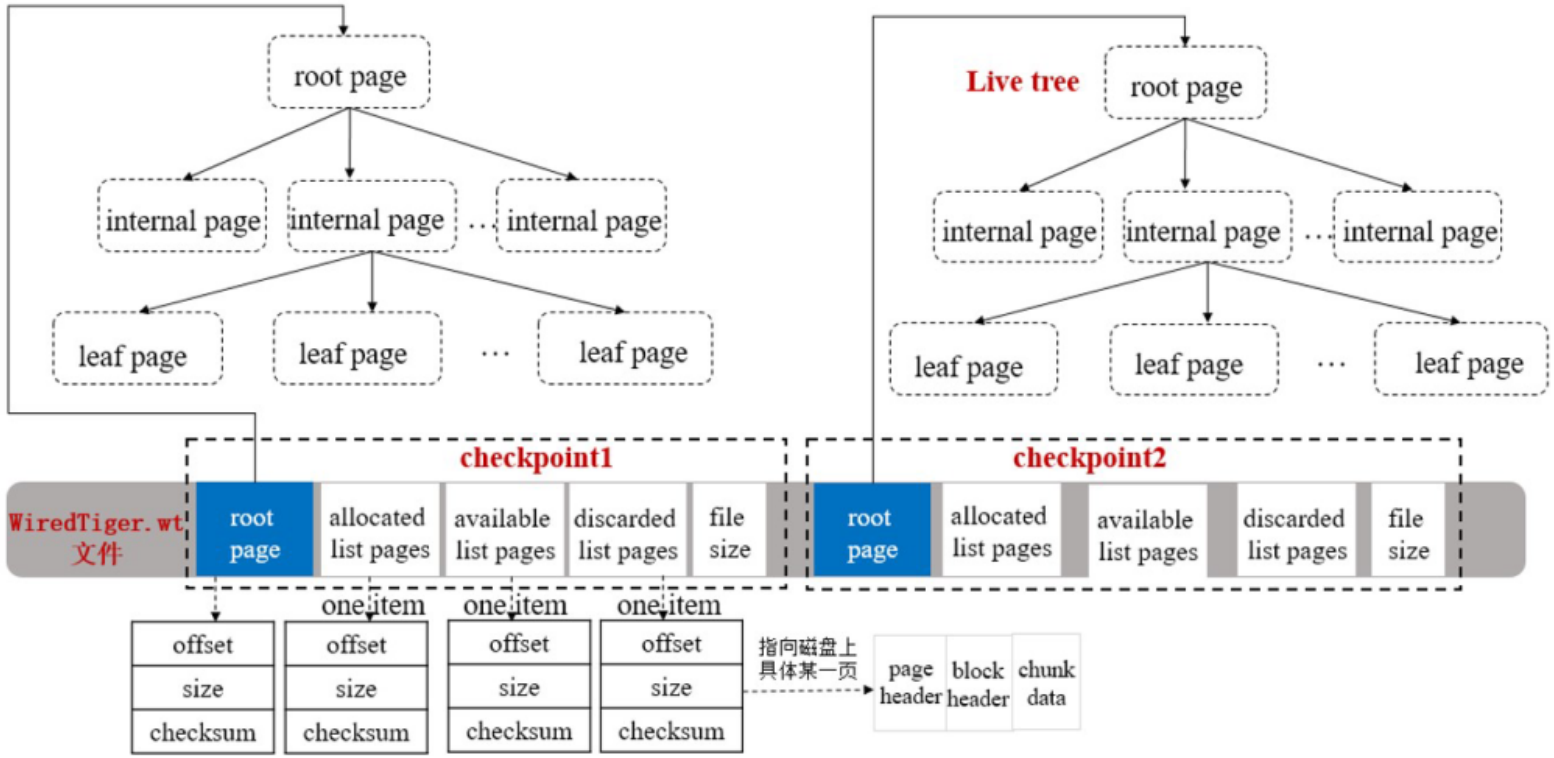
记录上一次checkpoint之后，这次checkpoint执行时，由WiredTiger块管理器新分配的pages。

- **discarded list pages**

记录上一次checkpoint之后，这次checkpoint执行时，丢弃的不再使用的pages，

- **available list pages**

记录这次checkpoint执行时，由WiredTiger块管理器分配但还没被使用的pages；



Checkpoint时机

1. 按一定时间周期：默认60s执行一次；
2. Journal日志文件大小达到2GB，执行一次checkpoint；
3. 任何打开的数据文件被修改，关闭时将自动执行一次checkpoint；

WiredTiger数据写入流程

应用层

1. 往集合 db.coll 发起一个文档插入请求。

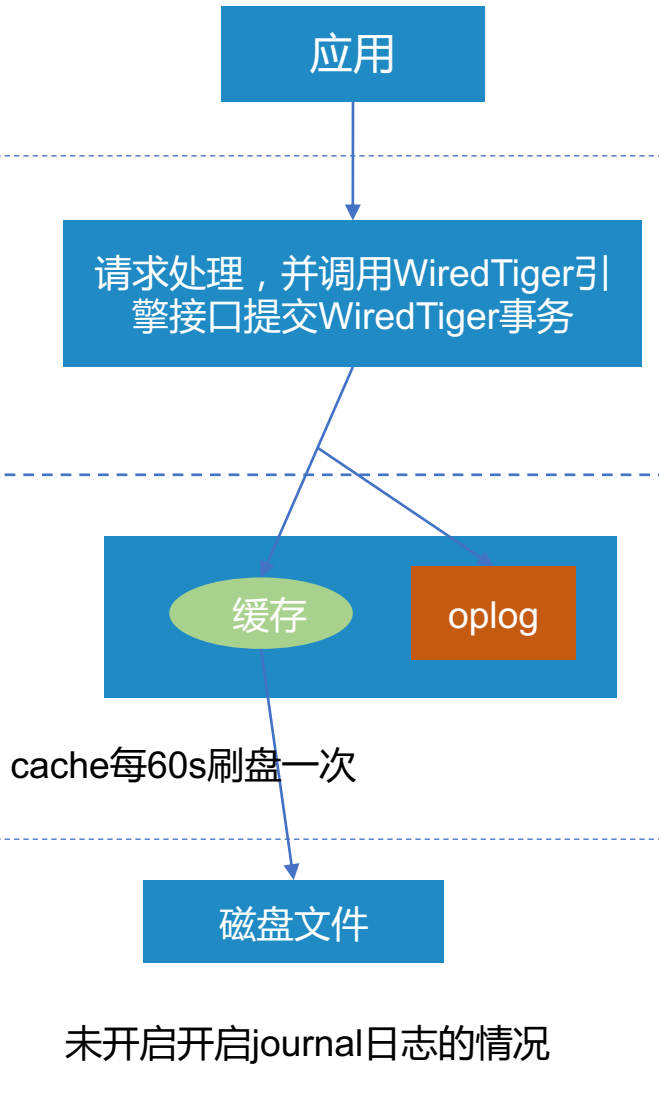
请求处理层

2. 开启事务。
3. 将文档插入 db.coll 集合中。
4. 更新 db.coll的索引。
5. 往 local.oplog.rs 集合写入oplog。
6. 提交事务给WiredTiger。

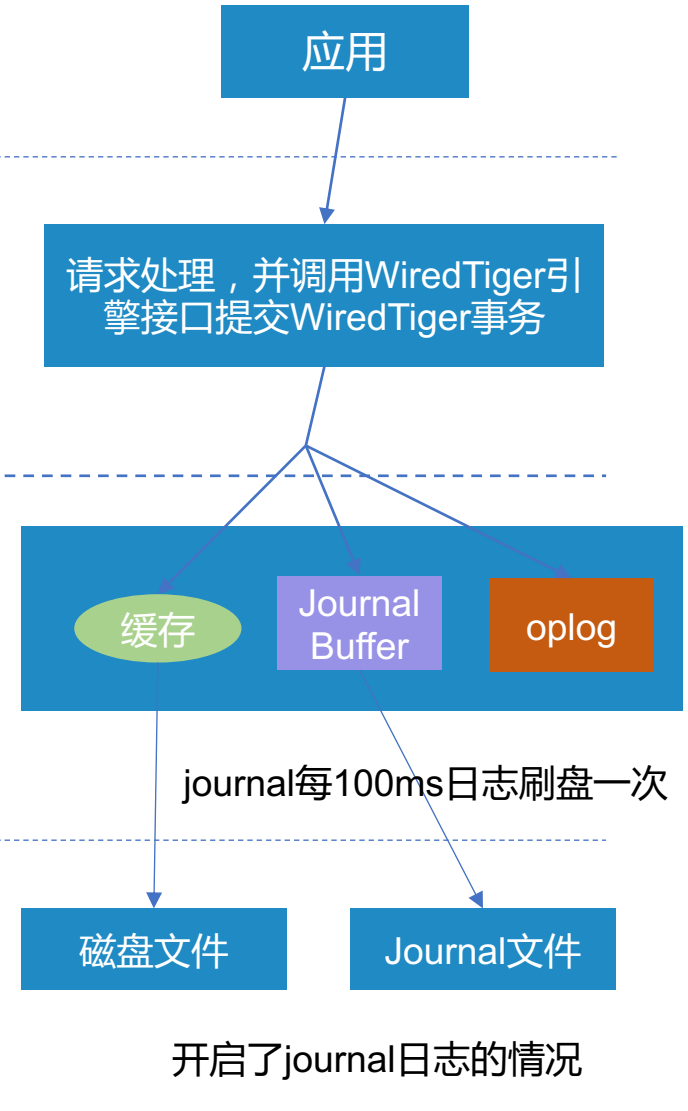
WiredTiger引擎层

7. 写日志缓冲。
8. 写数据缓冲 (B+Tree)。
9. 写oplog (Capped Collection)。

外部存储



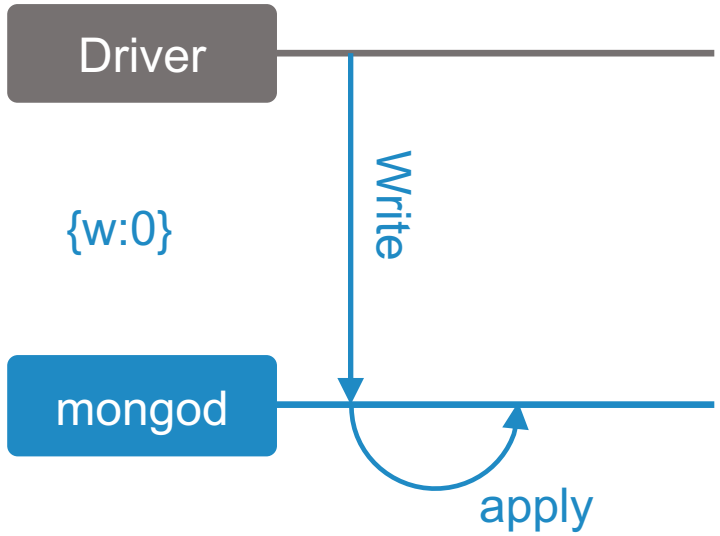
未开启开启journal日志的情况



开启了journal日志的情况

WriteConcern

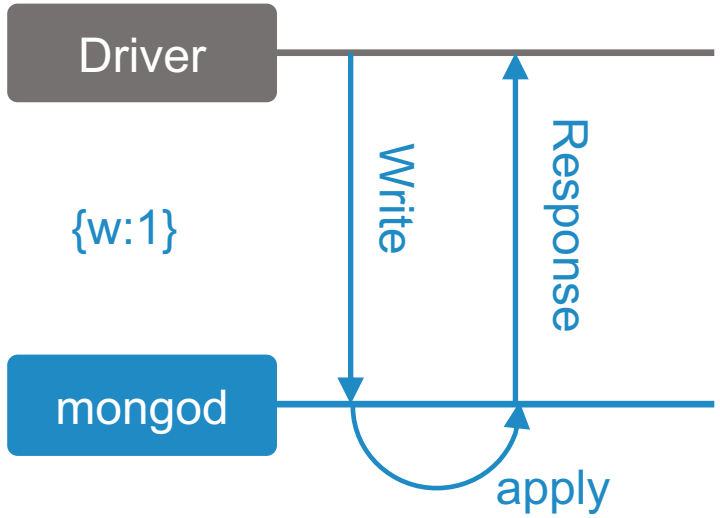
- 用来指定mongod对写操作的回执行为。
- writeConcern有三个字段，分别为 w，j 和 wtimeout：
  - w可选值: **0 | 1 | N | majority**。（N可取值2、3…等）
  - j 可选值: **false | true**



unacknowledged

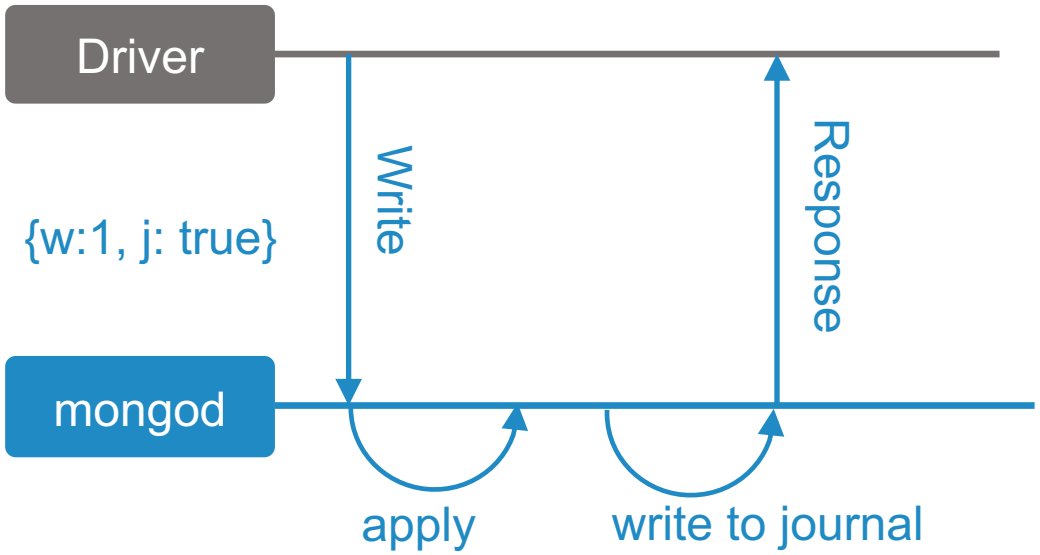
示例：

```
db.test.insert( { _id: “foo}, { writeConcern: { w: 1, wtimeout: 5000 } } )
```

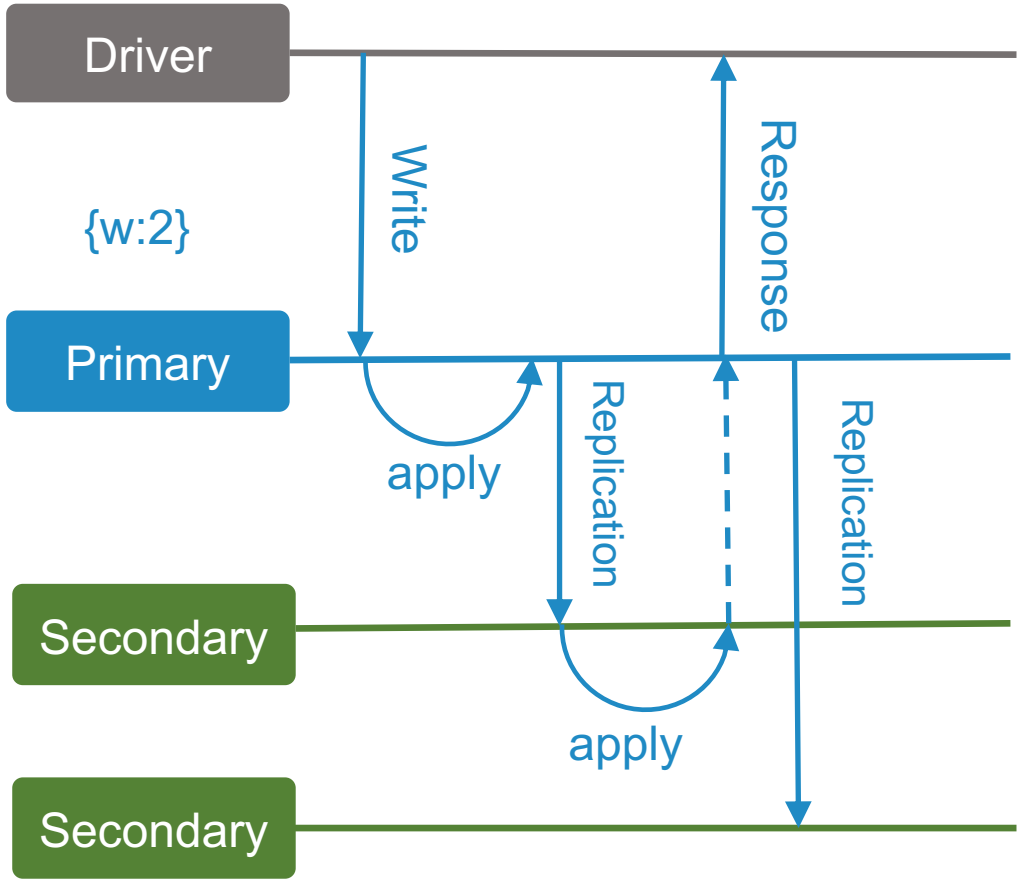


acknowledged

WriteConcern



journaled



2/N/majority replica acknowledged

ReadPreference

- ReadPreference主要包含 read preference mode，和一个可选的 tag set，3.4版本或以上还支持配置 maxStalenessSeconds。

  - read preference mode

模式	说明
primary	只从复制集的主节点读。
primaryPreferred	优先从复制集的主节点读，如果主节点不可用，则读从其他从节点。
Secondary	只从复制集的从节点读。
secondaryPreferred	有限从复制集的从节点读，如果从节点不可用，则读主节点。
nearest	优先从网络通信时延最小的节点读

ReadPreference

- tag set
  - 复制集的节点可以指定关联一系列的tags值，tags的取值格式为：
    { "<tag1>": "<string1>", "<tag2>": "<string2>",... }
  - 客户端在发起读请求时，可以在read preference中指定相应的tags值，以便将请求路由到满足tag条件的节点上。
  - 当read mode为primary时，tags值不起作用。

示例：

```
// 1. 读请求路由到同时满足 dc是shanghai，usage是production两个条件的从节点。
  db.collection.find({}).readPref( "secondary", [ { "dc": "shanghai", "usage": "production" } ] )
// 2. 读请求先路由到满足 dc是shanghai的从节点，如果找不到节点，则尝试路由到usage是production的从节点。
  db.collection.find({}).readPref( "secondary", [ { "dc": " shanghai "}, { "usage": "production" } ] )
```

- maxStalenessSeconds

mongo3.4或以上版本支持的选项，用来控制当从节点数据同步滞后主节点超过指定时间时，客户端停止使用该节点来进行数据读取。

```
ugc_test_0:PRIMARY> conf = rs.conf();
{
        "_id" : "ugc_test_0",
        "version" : 76,
        "protocolVersion" : NumberLong(1),
        "members" : [
                {
                        "_id" : 7,
                        "host" : "9.25.159.216:7005",
                        "arbiterOnly" : false,
                        "buildIndexes" : true,
                        "hidden" : false,
                        "priority" : 1,
                        "tags" : {
                                "dc" : "shanghai",
                                "usage" : "production"
                        },
                        "slaveDelay" : NumberLong(0),
                        "votes" : 1
                },
                {
                        "_id" : 8,
                        "host" : "9.25.159.216:7011",
                        "arbiterOnly" : false,
                        "buildIndexes" : true,
                        "hidden" : false,
                        "priority" : 1,
                        "tags" : {
                                "dc" : "shenzhen",
                                "usage" : "reporting"
                        },
                        "slaveDelay" : NumberLong(0),
                        "votes" : 1
                },
```

mongo索引

索引类型（Index Types）

索引类型	说明	使用示例
单字段索引 （Single Field）	针对单个字段建立的索引，可指定升序或者降序，每个集合默认会对_id字段建立一个索引。	1. {CmId:”xxxxx”, Uin:”123456”} 2. db.comments.createIndex({Uin:1}) 3. db.comments.find({Uin:” 123456”})
复合索引 (Compound)	针对多个字段联合创建索引，先按第一个字段排序，第一个字段相同的再按第二个字段排序，复合索引查询时遵循“最左匹配原则”。	1. {TpId:“song_237773700”, “Seq”：1000)} 2. db.comments.createIndex({TpId:1,Seq:-1}) 3. db.comments.find({TpId:” song_237773700”,Seq:{$lt:2000}})
多key索引 （Multikey）	这是针对数组类型的字段建立的索引，数组中的每个元素都会添加到索引中。	1. {“CmId" : “xxx", Label: [“good, poor"]} 2. db. comments.createIndex({Label: 1}) 3. db. comments.find({Label: "good"})
地理位置索引 （2dsphere ）	2dsphere 索引以GeoJSON对象或者普通坐标对的方式存储数据，支持在一个类地球的球面上进行几何计算。例如：查找附近的人等场景。	1. {loc : { type: "Point", coordinates: [-73, 40]}, name:"Central Park"} 2. db.parks.createIndex({loc: "2dsphere" }) 3. db.parks.find({loc: { $near:{ $geometry:{ type:"Point",coordinates:[-60,50] } , $maxDistance:10}}})
哈希索引 （Hashed）	基于某个字段的hash值来建立索引，目前主要用在hash分片的场景，这类索引只能进行字段完全匹配的查询。	1. {Uin:” 123456”, AddC: 2} 2. db.uin_flows.createIndex({Uin:"hashed"}) 3. db.uin_flows.find({Uin:” 123456”})

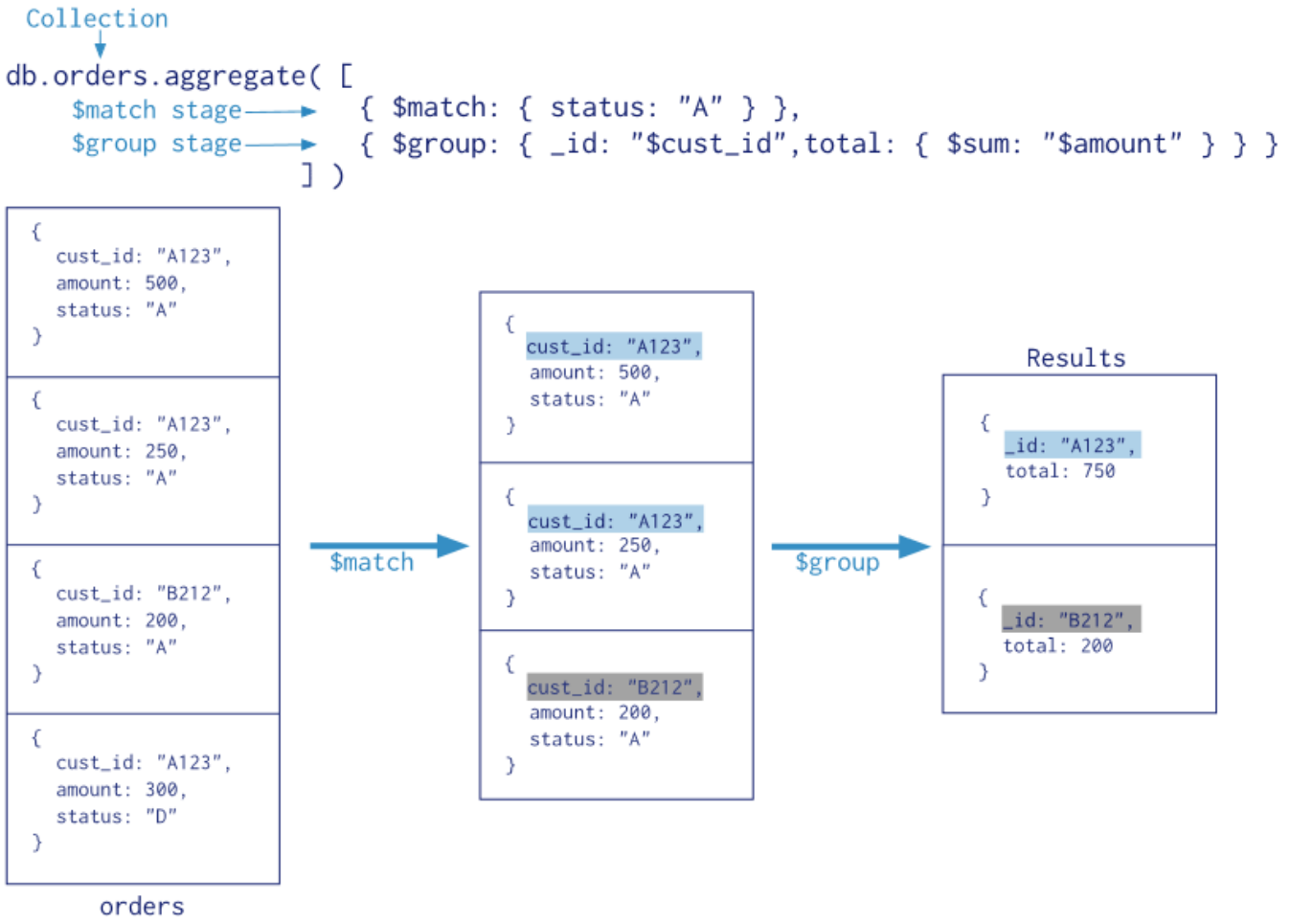
索引属性 (Index Properties)

索引属性	说明	使用示例
唯一索引 （Unique）	用来限制索引字段不出现相同的取值	db.comments.ensureIndex({"TpId":1,"_id":1},{unique: true})
TTL索引 （Time To Live）	针对日期类型的字段，指定文档的过期时间（可以指定多长时间后过期 或 在将来的某个时间点过期）	db.uin_flows.createIndex({ "Ts": 1 }, { expireAfterSeconds: 31536000 })
部分索引 (Partial)	只针对符合某个特定条件的文档建立索引，3.2或以上版本才支持。目前只支持以下几种条件： • 判等表达式，eg: {field:value} or {field:{$eq:value}} • $exists: true • $gt, $gte, $lt, $lte表达式 • $type表达式 • 顶层使用的 $and 操作符	db.comments.createIndex({TpId:1,HoSc:-1,Seq:-1}, { partialFilterExpression:{ $and:[{St:1},{HoSc:{$exists:true}}]})
稀疏索引 (Sparse)	只针对存在索引字段的文档建立索引，可看做是部分索引的一种特殊情况。	db. comments.createIndex({“RtId”:1}, {sparse: true})

聚合Pipeline

常用管道	说明
$group	对文档进行分组，可用于分组统计
$match	条件过滤，只处理条件符合的文档
$project	修改文档的结构 (例如重命名、增、删字段等)
$sort	对输出结果进行排序
$limit	限制管道输出结果的数目
$skip	略过指定数目的输出结果
$unwind	将数组类型的字段进行拆分

常用表达式	含义
$sum	{$sum: 1} 表示返回总和×1的值 (即总的条目数) {$sum: “$字段”} 计算指定字段值的总和
$avg	求平均值
$min	求最小值
$max	求最大值
$push	将文档结果插入到一个数组中
$first	根据文档的排序结果获取第一个文档的数据
$last	根据文档的排序结果获取最后一个文档的数据



一个聚合统计案例

mongo在评论中的应用

Q音评论产品形态的演变

mongo改造前评论数据的存储

评论id, eg: song_123_845381442_1608084008

评论详情数据，eg: Comment{ strID:"xxxx", mapContent:xxxx }

查询顺序

旧的评论数据存储，存在的问题：

1. ckv单key的值大小有上限，会导致个人主页评论id无法写入，同时也使得热门榜评论id需要做上限限制，无法对所有的评论按点赞排序。
2. 评论id和评论详情数据存在两种异构存储中，需要单独发送网络请求来更新，存在id和详情数据不一致的情况。
3. 无法满足盖楼等多样化的列表排序和复杂的条件过滤查询场景。
4. 在产品更新迭代代时，无法方便对所有存量数据进行处理，例如增加推荐分、热门分等新字段。

score

845381442_1608084099	1608084099
1850389556_1608084088	1608084088
1850389556_1608084077	1608084077

ckv+中的时间倒序评论id列表
(zset，key: comment_zset_song_123)

song_123_845381442_1608084099
song_123_2441509871_1608084066
song_123_1850389556_1608084077
song_123_1850389556_1608084088

ckv中的热门榜评论ids
(kv，key: toplist_song_123)

song_123_1850389556_1608084088
song_123_1850389556_1608084077

ckv中时间倒序个人主页评论ids
(kv，key: self_comment_song_123)

ckv+中单条评论详情数据
(kv, key: song_123_845381442_1608084008)

Comment{…} Comment{…} Comment{…}

ugc中单个TopicId下的列表数据

Comment{…}
Comment{…}
Comment{…}
Comment{…}

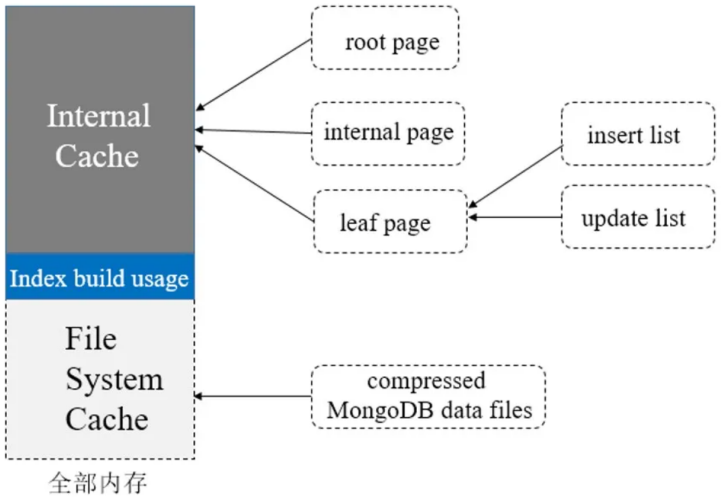
评论分片集群

primary节点，主要用在评论写入和数据同步等场景

secondary节点，主要用在各种评论列表拉取、评论详情查询等场景。

hidden节点，用于数据分析和挖掘

分片名	实例名	IP	PORT	角色	cpu配额(cores)	CPU使用率	内存配额(MB)	内存使用率	存储配额(GB)	存储使用率	同步源
music_comment_0	18890161	10.62.84.26	7007	Primary	4	2%	8192	68%	292	13%	
music_comment_0	23192985	9.186.15.113	7003	Secondary	4	57%	8192	65%	292	13%	10.62.84.26:7007
music_comment_0	23192987	10.56.88.239	7000	Secondary	4	39%	8192	64%	292	13%	10.62.84.26:7007
music_comment_0	37841656	100.97.23.119	7000	Secondary	4	0%	8192	83%	292	13%	10.62.84.26:7007
music_comment_1	18890205	10.165.22.154	7001	Primary	4	1%	8192	67%	292	13%	
music_comment_1	23192944	10.55.214.61	7005	Secondary	4	40%	8192	65%	292	13%	10.165.22.154:7001
music_comment_1	27303729	10.56.85.55	7001	Secondary	4	45%	8192	63%	292	13%	10.165.22.154:7001
music_comment_1	37841721	100.97.23.119	7001	Secondary	4	0%	8192	87%	292	13%	10.165.22.154:7001
music_comment_2	18890266	9.22.14.172	7010	Primary	4	0%	8192	69%	292	13%	
music_comment_2	18890271	9.44.14.26	7001	Secondary	4	46%	8192	62%	292	13%	9.22.10.125:7005
music_comment_2	23192946	9.22.10.125	7005	Secondary	4	38%	8192	65%	292	13%	9.22.14.172:7010
music_comment_2	37841884	100.97.23.119	7002	Secondary	4	1%	8192	88%	292	13%	9.22.14.172:7010
music_comment_3	23258102	9.44.13.78	7000	Primary	4	0%	8192	68%	292	13%	
music_comment_3	18890383	9.44.11.112	7001	Secondary	4	46%	8192	66%	292	13%	10.56.87.108:7005
music_comment_3	23193039	10.56.87.108	7005	Secondary	4	92%	8192	66%	292	13%	9.44.13.78:7000
music_comment_3	37842035	100.97.23.119	7003	Secondary	4	0%	8192	82%	292	13%	9.44.13.78:7000
music_comment_4	18890440	9.44.11.67	7005	Primary	4	1%	8192	69%	292	14%	
music_comment_4	23192969	9.22.10.60	7007	Secondary	4	49%	8192	62%	292	14%	9.44.11.67:7005
music_comment_4	23192967	10.56.87.108	7004	Secondary	4	44%	8192	66%	292	14%	9.44.11.67:7005
music_comment_4	37842258	100.97.23.119	7004	Secondary	4	0%	8192	89%	292	14%	9.44.11.67:7005

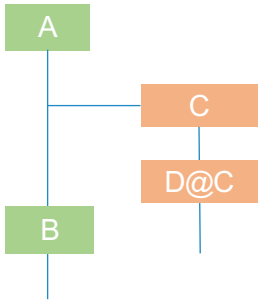


评论存储迁移到mongo

评论分片集合创建（4.0版）

```
sh.shardCollection(“cm.comments", {“TpId":"hashed"}, false, {numInitialChunks:8000})
```

主要字段	说明
CmId	评论id，eg：song_107685_737702042_1563142999
TpId	TopicId，eg：song_107685，分片键
Seq	用SnowFlake方法生成的分布式唯一自增序号
Uin	用户的musicid
St	评论状态，正常、自己可见和已删除等
Txt	Base64加密的评论文本
RplCnt	回复数
PId	父评论id
RtId	根评论id
RkSc	盖楼热门序排序分，目前只存了点赞数
HoSc	一级全局热门序排序分，高位存置顶序，低位存点赞数
RcSc	通过公式计算得到的综合推荐分



评论列表层级结构

```
// mongo中的评论bson对象
struct MongoComment {
    // 必备字段
    0 optional string CmId;
    1 optional string TpId;
    2 optional long Seq;
    3 optional string Uin;
    4 optional int St;
    5 optional string Txt;
    6 optional int RplCnt;

    // 可选字段（业务无关）
    7 optional string PId;
    8 optional string RtId;
    9 optional int SubSt;
    10 optional int NoSf;

    // 可选字段（业务相关）
    11 optional int HitT;
    12 optional int HitRs;
    13 optional int CommitSt;

    // 关联的拓扩展信息
    14 optional string RltBizId;
    15 optional string TaogeTopic;
    16 optional string TaogeUrl;
    17 optional int SongType;
    18 optional string RltCmId;
    19 optional string SongId;

    20 optional long RkSc;
    21 optional int AuP;

    22 optional long HoSc;
    23 optional double RcSc;
};
```

comments集合的文档结构

comments集合索引设计

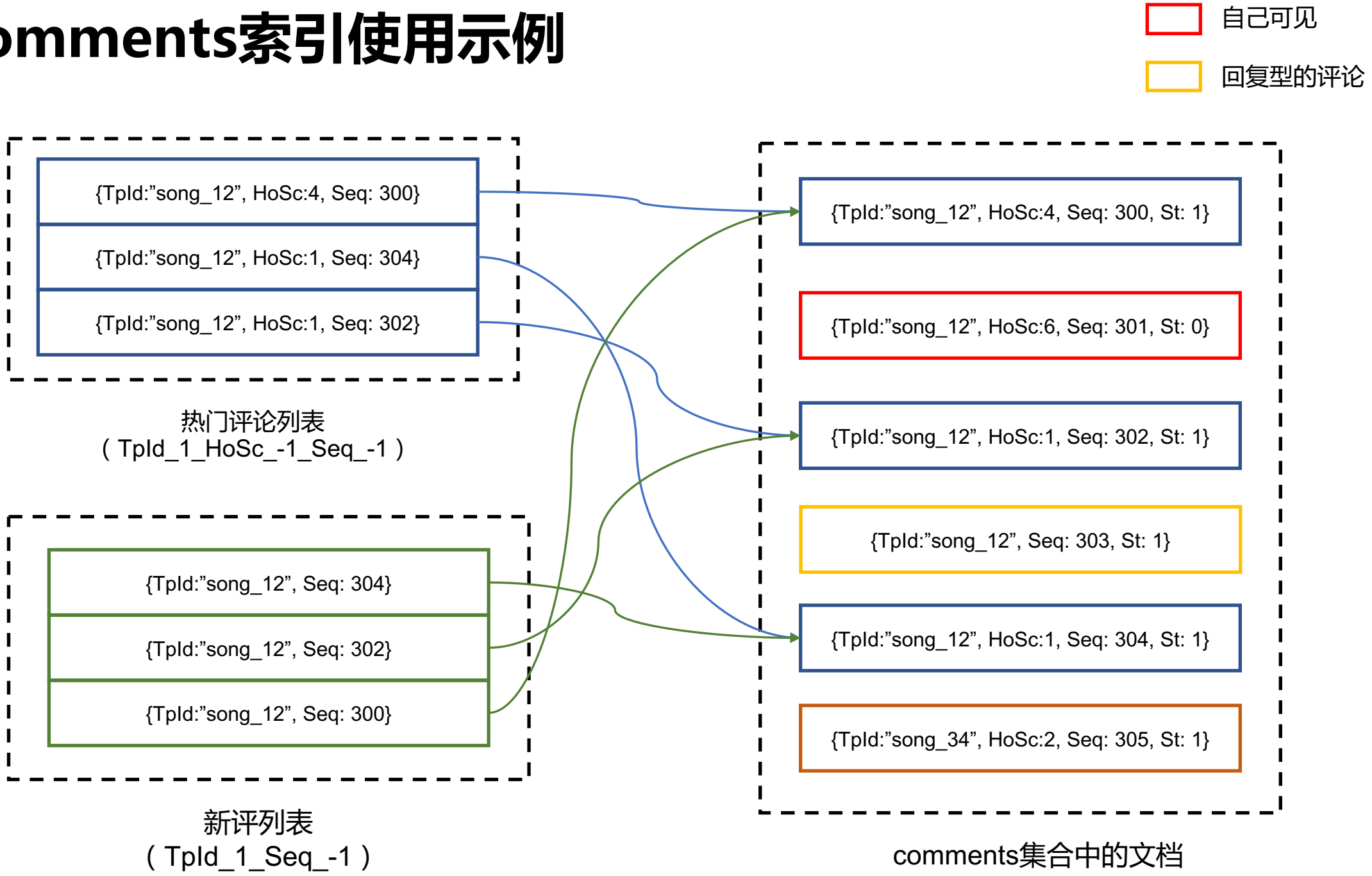
索引	说明
_id_	评论id，eg：song_107685_737702042_1563142999
TpId_1	TopicId，eg：song_107685，分片键，hash索引
TpId_1__id_1	组合唯一索引，限制评论id的唯一性
Seq_1	全局时间有序且唯一的分布式序号
TpId_1_Seq_-1	某TopicId下，时间降序的全部评论
Uin_1_Seq_-1	某Uin下，时间降序的个人评论
RtId_1_Seq_1	某根评论id下，时间升序的盖楼评论
RtId_1_RkSc_-1_Seq_1	某根评论id下，优先按热度降序，其次按时间升序的盖楼评论，RkSc目前存储的点赞数
TpId_1_HoSc_-1_Seq_-1	某TopicId下，优先按置顶时间降序，其次按点赞降序，最后按时间降序的全部评论，HoSc是64位整数，高位存置顶序，低位存点赞数。采用的是条件索引
TpId_1_RcSc_-1_Seq_-1	某TopicId下，优先按推荐分降序，其次按时间降序的推荐评论。采用的是条件索引

HoSc字段结构

索引创建语句：

- db.comments.ensureIndex({"TpId":1,"_id":1},{unique: true}) // 唯一索引
- db.comments.ensureIndex({"TpId":1,"Seq":-1},{background: true}) // 时间页
- db.comments.ensureIndex({"Uin":1,"Seq":-1},{background: true}) // 个人主页
- db.comments.ensureIndex({“Seq”:1},{background: true}) // 全局时间序
- db.comments.ensureIndex({“RtId”:1, “RkSc”: -1, “Seq”:1}, {background: true})
  db.comments.ensureIndex({"RtId":1, "Seq":1},{background: true})
- db.comments.createIndex({TpId:1,HoSc:-1,Seq:-1},
  { partialFilterExpression:{ $and:[{St:1},{HoSc:{$exists:true}}]}, background:true })
- db.comments.createIndex({TpId:1,RcSc:-1,Seq:-1},
  { partialFilterExpression:{ $and:[{St:1},{RcSc:{$exists:true}}]}, background:true })

comments索引使用示例



评论任务系统

内容创作/日	计算方式	上限次数	上限/分
发评论	+5	5	25
看评论/3min	+3	5	15
分享评论	+3	5	15
回复评论	+1	5	5
内容被肯定/月	**计算方式**	**上限次数**	**上限/分**
精彩评论	+10	20	200
置顶评论	+20	10	200
内容被消费/月	**计算方式**	**上限次数**	**上限/分**
评论被回复	+1/1条	300	300
评论被赞数	+50/一条评论达到50个赞	评论被赞数	评论被赞任务每月累计最多奖励500分
评论被分享	+3/1次	100	300

（1）月任务和日任务，奖励有限额

编号id	UIN	认证名字	总评论学分↓	置顶评论次数	精彩评论次数	开始日期	结束日期
5	321840491	sir	86	0	1	2020-12-05 00:00:00	2020-12-07 20:02:28
6	1851776221	AntonioMelissa	25	0	0	2020-12-02 00:00:00	2020-12-15 17:30:49

（2）以日为单位进行任务流水聚合统计

需要解决以下问题：

1. 日流水只保留近一年的，超出时间的自动过期。
2. 支持日粒度的范围统计。
3. 支持当天日或者月任务发放限额。
4. 点赞简单防刷。

评论任务数据的存储

uin_flows集合

索引	说明
Uin_hashed	以uin作为hash分片键
Uin_1_Ts_1_StT_1	组合唯一索引，保证用户某一天某一种粒度统计条目只有一条，同时为了聚合操作能使用到索引。
Ts_1	以日为粒度的日期，TTL索引，一年过期

St: 1-日粒度流水类型，2-月粒度流水类型

```
{
    "_id" : ObjectId("5fccc55029dca07aaabe8c18"),
    "StT" : NumberLong(1),
    "Ts" : ISODate("2020-12-05T16:00:00Z"),
    "Uin" : NumberLong(1842734989),
    "AddR" : NumberLong(1),
    "AddC" : NumberLong(1),
    "BRpl" : NumberLong(2)
}
```

用户为主体的任务流水

cmid_flows集合

索引	说明
Uin_hashed	以uin作为hash分片键
Uin_1_Ts_1_CmId_1_StT_1	组合唯一索引，保证用户某一天某一条评论某一种粒度统计条目只有一条，同时为了聚合操作能使用到索引。
Ts_1	以日为粒度的日期，TTL索引，一年过期

```
{
    "_id" : ObjectId("5fccc54a00dcf65040057f33"),
    "CmId" : "song_4931121_1152921505018718032_1594097510",
    "StT" : 2, "Ts" : ISODate("2020-11-30T16:00:00Z"),
    "Uin" : NumberLong("1152921505018718032"),
    "BPr" : NumberLong(3),
    "PrUins" : [ NumberLong("2351509995"), NumberLong(1732091969), NumberLong(617757292) ]
}
```

评论id为主体的任务流水

批量和原子操作的使用

一次到mongos的网络请求，完成多个操作

1. 如果指定评论的日或者月流水不存在则创建。
2. 同时进行日流水和月流水的被点赞数的原子自增。
3. 同时完成数组的push和slice。

```
writeModels := []mongo.WriteModel{
    mongo.NewUpdateOneModel().SetFilter(bson.M{
        mongoProxy.MgoCmTaskCmIdFlowsFieldUin:       praisedUin, // 被点赞人
        mongoProxy.MgoCmTaskCmIdFlowsFieldCommentId: praisedCmIdSt.ID,
        mongoProxy.MgoCmTaskCmIdFlowsFieldTimestamp: dayDate,
        mongoProxy.MgoCmTaskCmIdFlowsFieldStatType:  mongoProxy.MgoCmTaskStatTypeDay,
    }).SetUpdate(bson.M{
        "$inc": bson.M{mongoProxy.MgoCmTaskCmIdFlowsFieldBePraised: int64(1)},
    }).SetUpsert( upsert: true), // 评论日被点赞流水 ( 统计用)
    mongo.NewUpdateOneModel().SetFilter(bson.M{
        mongoProxy.MgoCmTaskCmIdFlowsFieldUin:       praisedUin, // 被点赞人
        mongoProxy.MgoCmTaskCmIdFlowsFieldCommentId: praisedCmIdSt.ID,
        mongoProxy.MgoCmTaskCmIdFlowsFieldTimestamp: monthDate,
        mongoProxy.MgoCmTaskCmIdFlowsFieldStatType:  mongoProxy.MgoCmTaskStatTypeMonth,
    }).SetUpdate(bson.M{
        "$inc": bson.M{mongoProxy.MgoCmTaskCmIdFlowsFieldBePraised: int64(1)},
        "$push": bson.M{mongoProxy.MgoCmTaskCmIdFlowsFieldPraiseUins: bson.M{
            "$each":  []int64{uin},
            "$slice": int64(-20),  // 保留最近20个点赞用户
        }},
    }).SetUpsert( upsert: true), // 评论月被点赞流水 （ 判断是否应该发放当月的被点赞奖励分 ）
}
```

总结

一点心得

- Primary只用来写或者用作数据对账时的读，让从机对外提供查询能力

这样在因为突发流量或者大量慢查询导致从机挂掉的情况也不响应数据的写入。保证写的稳定性。

- 建立hash分片时提前指定初始chunk的大小。

sh.shardCollection(“cm.comments", {“TpId":"hashed"}, false, {numInitialChunks:8000})

- 能用原子或者批量命令尽可能使用，减少不一致或者版本的冲突的情况。

mongo支持bulkWrite批量写入，findAndModify，原子自增$inc，位操作$bit等操作，$push 和 $slice结合。

- 尽量应用条件索引。

这样在条件翻页时，不会因为条目扫描过多导致超时。

- 注意语言类型和mongo类型的对应关系。

使用Go版的mongo库进行数据插入或者更新时，要注意Go类型和mongo类型是有对应关系的，插入通过insert、update等更新操作指定整型常量值时，最好明确指定类型，否则默认是double。

- 在对存量数据建新索引时，注意要带上 background 参数。

例如：db.comments.ensureIndex({“RtId”:1, “RcSc”: -1, “Seq”:-1}, {background: true})

- _id字段默认会自动生成并创建单key唯一索引，可以用业务中更有意义的数据id值替换之。

mongo4.2新特性

- 分布式事务。

  让用户修改分片key的内容成为可能。

- 支持中文全文索引。

  直接与 Lucene 引擎整合，搜索功能增强。

- update功能增强。

  update能根据文档现有的字段内容来生成新的更新内容。

- 查询语言表达能力、分析和聚合能力方面的增量。

谢谢