
mongoing
中文社区
錦木
mongoDB
MongoDB中文社区北京大会
MongoDB中文社区 & 锦木信息技术有限公司联合主办
2018.08.25 北京
mongoing
中文社区
IT大咖说
知识共享平台

MongoDB

讲师：胡国青

Mongodb

HA集群架构

01 Mongodb 命名规则

02 Mongodb 复制集

03 Mongodb 分片集群

04 Mongodb 集群模式典型场景

05 研发特别注意

ONE

Mongodb命名约束规则

SQL到MongoDB映射表

SQL 术语/概念	MongoDB 术语/概念
数据库	database
表	collection
行	document or BSON 文档
列	field
索引	index
表连接	内嵌文档和链接
主键 指定任何唯一列或复合列作为主键。	primary key 在MongoDB里，主键会自动设置为 _id 列。
聚合（e.g. group by）	聚合管道 查看文档 SQL to Aggregation Mapping Chart。

Mongodb开发命名规则

I. 复制集的读写设置

➢Read Preference

➢默认情况下，复制集的所有读请求都发到Primary，Driver可通过设置Read Preference来将读请求路由到其他的节点。

- ✓primary： 默认规则，所有读请求发到Primary
- ✓primaryPreferred： Primary优先，如果Primary不可达，请求Secondary
- ✓secondary： 所有的读请求都发到secondary
- ✓secondaryPreferred：Secondary优先，当所有Secondary不可达时，请求Primary

II. nearest：读请求发送到最近的可达节点上(通过ping探测得出最近的节点)

统计分组函数优化

➢db.props.aggregate([{$group:{_id:"$extra.uc_event.batchId",count:{$sum:1}}},{$match:{count:{$gt:1}}}],{ allowDiskUse: true })

- ◆在secondary库执行
- ◆分析该语句可能出现的问题
  - ✓做统计分析，查询超过了mongodb限制的16M大小；
  - ✓mongo内存限制。 aggregate函数 使用$group时，数据大小必须小于16945KB
  - ✓添加allowDiskUse: true解决

Mongodb开发命名规则

III. 3.Mongodb的创建索引
- ✓需要和DBA沟通
- ✓在后台创建索引—不影响业务正常的DML操作

```
b.works.createIndex({plan:1,trgpoints:1,cOrder:1,sValue:1},{background:true
})
```

IV. 删除字段、修改字段值等不清楚的和DBA沟通

V. 库名全部小写，禁止使用任何`_`以外的特殊字符，比如我们线上lp-pmm数据库

VI. 集合名全部小写，禁止使用任何`_`以外的特殊字符

VII. 如果评估单集合数据量较大，比如8亿以上的集合，可以将一个大集合拆分为多个小集合，即mongodb的分库分表-sharding；

VIII. MongoDB的集合拥有“自动清理过期数据”的功能
- ✓需在该集合中文档的时间字段增加一个TTL索引即可实现该功能
- ✓但需要注意的是该字段的类型则必须是mongoDate()
- ✓一定要结合实际业务设计是否需要

Mongodb开发命名规则

IX. 文档设计

✓文档中的key禁止使用任何`_`以外的特殊字符

✓禁止使用_id，如：向_id中写入自定义内容

X. 查询中的某些 $ 操作符可能会导致性能低下

✓$exist：因为松散的文档结构导致查询必须遍历每一个文档

✓$ne：如果当取反的值为大多数，则会扫描整个索引

✓$not：可能会导致查询优化器不知道应当使用哪个索引，所以会经常退化为全表扫描

✓$nin：全表扫描

✓$or：有多少个条件就会查询多少次，最后合并结果集，所以尽可能的使用 $in

XI. 不要一次取出太多的数据进行排序

✓MongoDB 目前支持对32MB以内的结果集进行排序

✓如果需要排序，那么请尽量限制结果集中的数据量

Teambition MongoDB语句黑名单

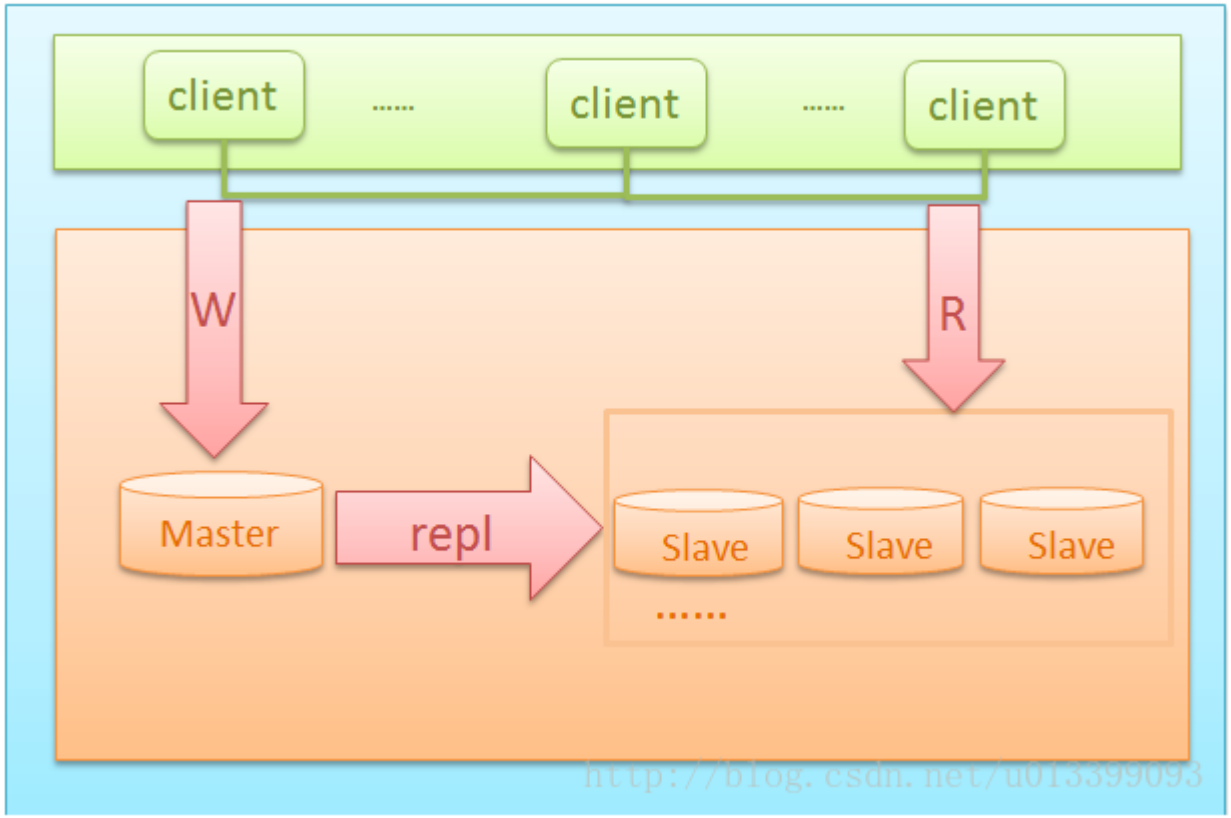
任何线上业务场景下，数据库语句需遵循如下规则（持续更新）：

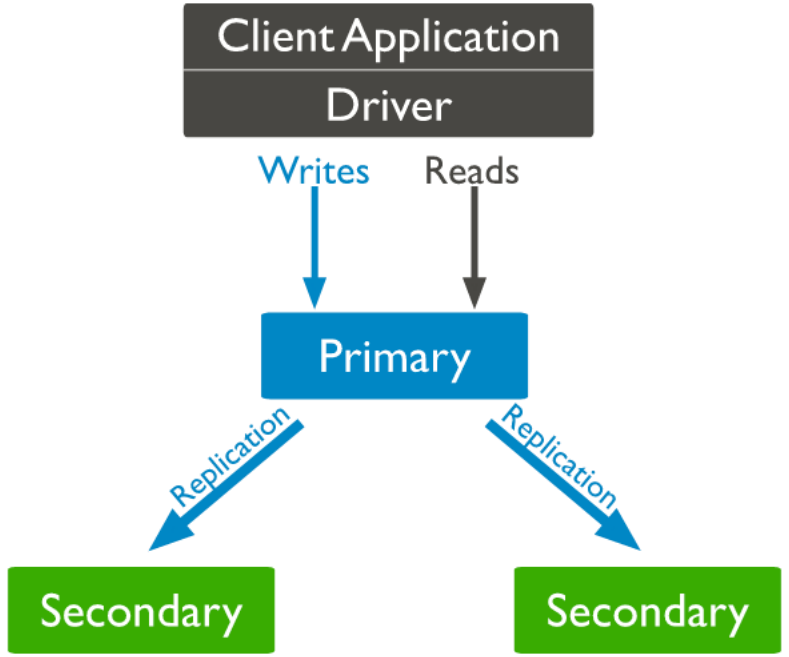
- 线上业务不得使用MapReduce语句 （请使用aggregate来代替或者代码层进行处理）。
- 不得使用不带条件的的update或者query语句。
- update语句中一定要使用$set。
- aggregate的第一层一定要使用$match,$group的层数需不得大于3层。
- find和aggregate的使用中要养成只返回需要数据字段的习惯。
- 对于数组字段的查询与update条件需要谨慎，数组如果可以为空数组，此时条件如果也为空数组则会匹配到空数组的数据。如数据为{a:[]},这时候{a:[]}的查询就会匹配到a为空数组的数据，如果没有这类空数组需求，业务上需要用in或者其他操作符来使用。

TWO

Mongodb副本集介绍

Mongodb复制集HA集群





工作机制

首先复制集客户端动通过isMaster命令，来判断复制集哪个节点是当前主节点；
判断之后，将writes操作，指向primary节点；
Mongodb也属于异步复制，就是写完主库操作以后，将这个操作记录下来，
在主库的oplog里，oplog主从都有；secondary节点都是通过复制主节点的oplog到本机的oplog里进行应用

半初始化同步-初始化同步过程

以存在的复制集，添加新节点的工作原理。
在克隆的时候，会将数据抓取到内存里面，会对primary有影响。

复制集成员

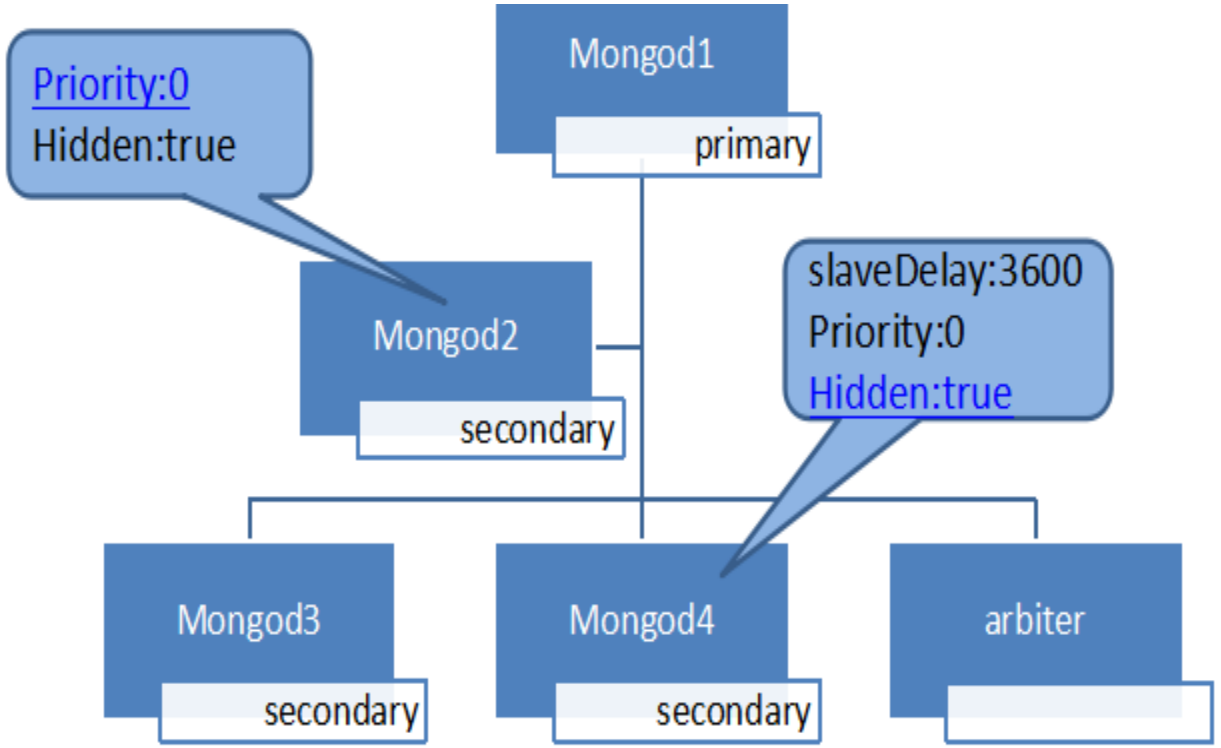
- 主节点

主节点是唯一能够接收写请求的节点。MongoDB在 主节点 上进行写操作，并会将这些操作记录到主节点的 oplog 中。 从节点 会将oplog复制到其本机并将这些操作应用到其自己的数据集上。

- 从节点

优先级为0的成员作为备用节点
隐藏节点
延时节点

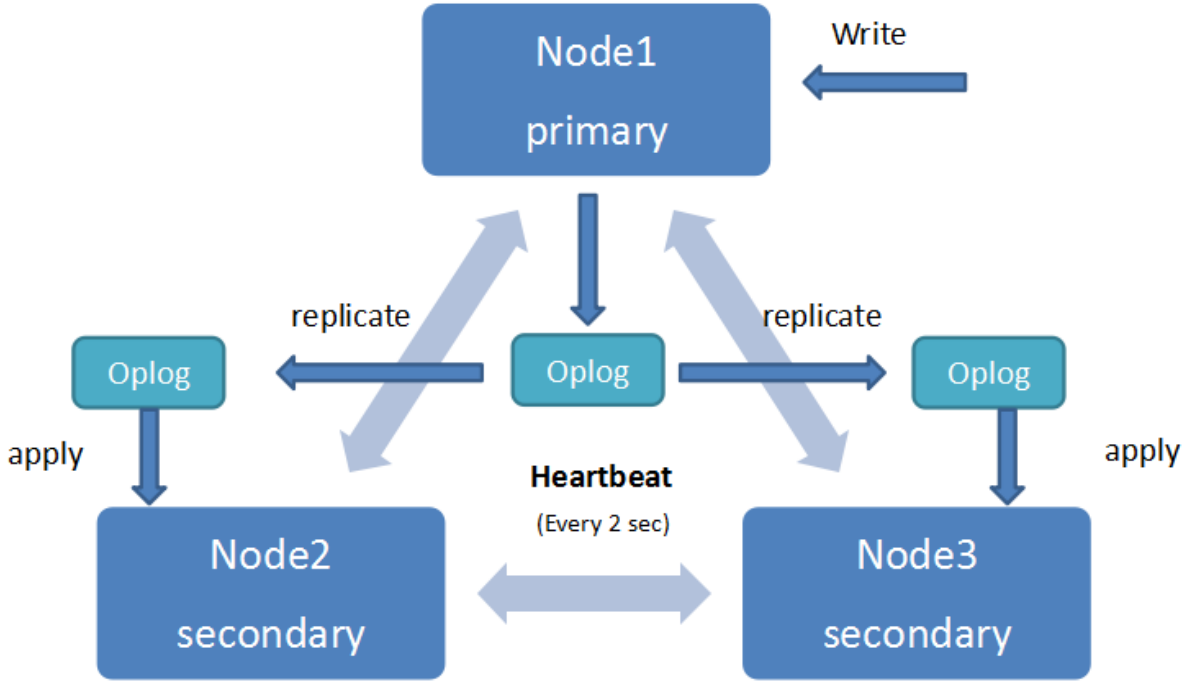
- 投票节点



复制集Oplog

Oplog 是Mongodb 复制集的纽带

作用：
当Primary进行写操作的时候，会将这些写操作记录写入Primary的Oplog 中，而后Secondary会将Oplog 复制到本机并应用这些操作，从而实现Replication的功能。
同时由于其记录了Primary上的写操作，故还能将其用作数据恢复。



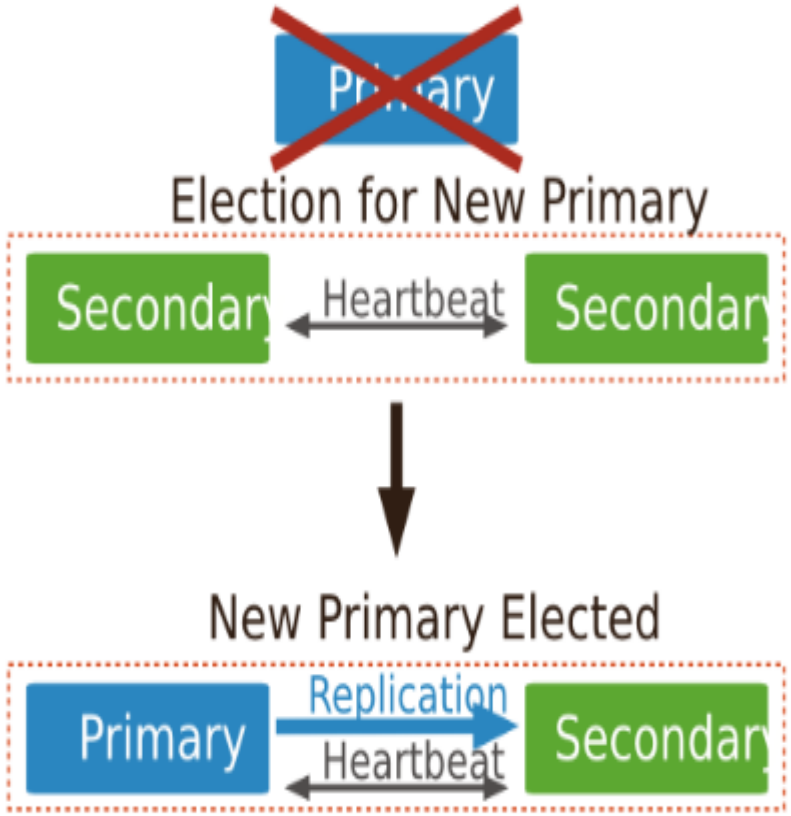
复制集高可用

1：复制集成员每两秒向复制集中其他成员进行心跳检测。
如果某个节点在10秒内没有返回，那么它将被标记为不可用。

2：一个从节点无法与主节点进行连接。
当从节点们无法与主节点进行沟通的时候将会触发选举

3：如果复制集中的某个节点不能连接上其他多数节点，
那么它将不能升职为主节点。在选举中，
多数是指多数 投票 而不是多数节点个数。

4：我们可以通过设定 priority 来加重某个或者某些特殊的
节点在选举中获得选票的优先级。比如，当我们有一个
异地分布式架构的复制集，我们可以通过设置优先级来
使只有特定数据中心中的节点能够升职为主节点

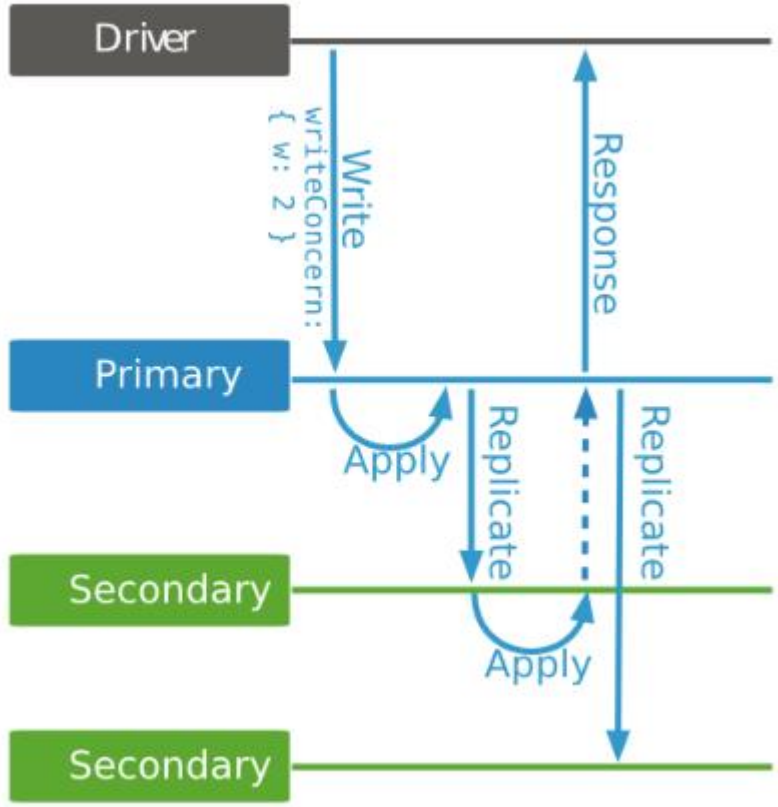


复制集部署：举个例子


Data Center 1
Primary
priority: 1
Secondary
priority: 1
Data Center 2
Secondary
priority: 0.5
Secondary
priority: 0.5
Data Center 3
Secondary
priority: 0

复制集的安全写级别

```
db.products.insert(
   { item: "envelopes", qty : 100, type: "Clasp" },
   { writeConcern: { w: 2, wtimeout: 5000 } }
)
```



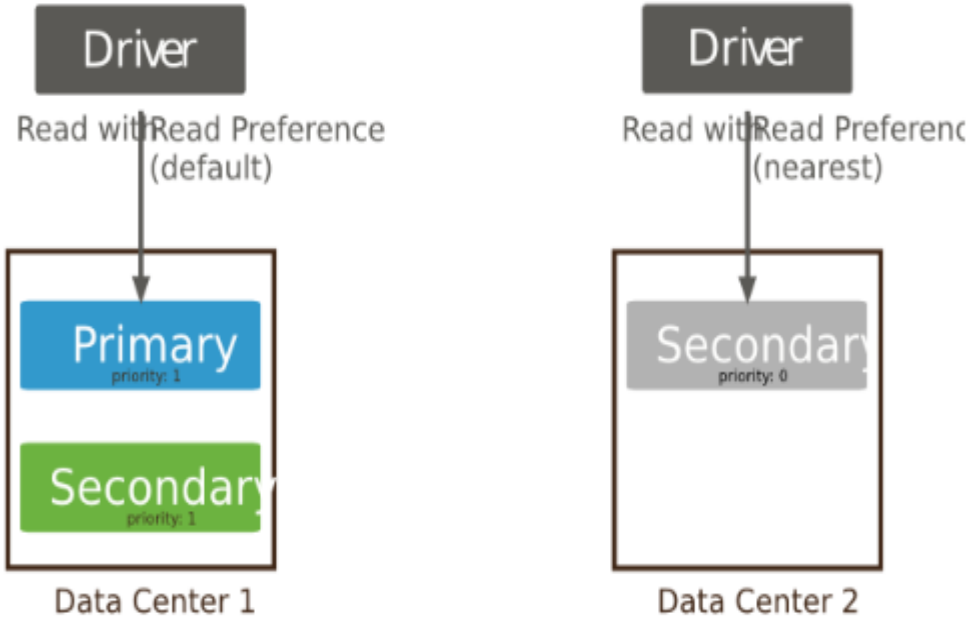
1：如果在执行写操作的时候指定了安全写级别，那么该写操作将会使用指定的安全写级别而不是默认的

2：我们可以在每次写操作的时候指定安全写级别来规避默认的安全写级别

3：我们可以在安全写级别中设定超时限制。这样可以在写操作无法到达目标服务器的时候造成的堵塞。

复制集读选项

复制集读选项模式	详细说明
primary	默认模式，所有的读操作都在复制集的 主节点 进行的。
primaryPreferred	在大多数情况时，读操作在 主节点 上进行，但是如果主节点不可用了，读操作就会转移到 从节点 上执行。
secondary	All operations read from the secondary members of the replica set.
secondaryPreferred	在大多数情况下，读操作都是在 从节点 上进行的，但是当 从节点 不可用了，读操作会转移到 主节点 上进行。
nearest	读操作会在 复制集 中网络延时最小的节点上进行，与节点类型无关。



除了 primary 模式以外的复制集读选项都有可能返回非最新的数据，因为复制过程是异步的，
从节点 上应用操作可能会比主节点有所延后。如果我们不使用 primary 模式，
请确保业务允许数据存在可能的不一致。

案例

1.如果只有一台服务器，也要以单节点模式启动副本集.

如果部署一台主机，要有远见，配置成单实例复制集，方便以后添加节点，rs.add，不用停机直接添加节点。

```
> config={_id:"rsOrder",members:[{_id:1,host:"10.47.209.200:27017",priority:136,tags:{'use':'order-history-1'}}]}
{
        "_id" : "rsOrder",
        "members" : [
                {
                        "_id" : 1,
                        "host" : "10.47.209.200:27017",
                        "priority" : 136,
                        "tags" : {
                                "use" : "order-history-1"
                        }
                }
        ]
}
> rs.initiate(config)
```

```
rs.add({_id:2,host:"10.47.209.221:27017",priority:123,tags:{'use':'order-history-2'}})
```

案例

Oplog设计要满足日常维护的大小—添加节点；

如果该节点最新的oplog时间戳，比所有节点最旧的oplog时间戳还要小，该节点将找不到同步源，会一直处于RECOVERING而不能服务；反之，如果能找到同步源，则直接进入replication阶段，不断的应用新的oplog；

一般给于40~50G左右；

1.数据量大小

2.QPS/TPS峰值

THREE

Mongodb分片集群

分片的作用

分片，是指将数据拆分，将其分散到不同的机器上。这样的好处就是，不需要功能强大的大型计算机也可以存储更多的数据，处理更大的负载。

mongoDB的分片，是将collection的数据进行分割，然后将不同的部分分别存储到不同的机器上。当collection所占空间过大时，我们需要增加一台新的机器，分片会自动将collection的数据分发到新的机器上。

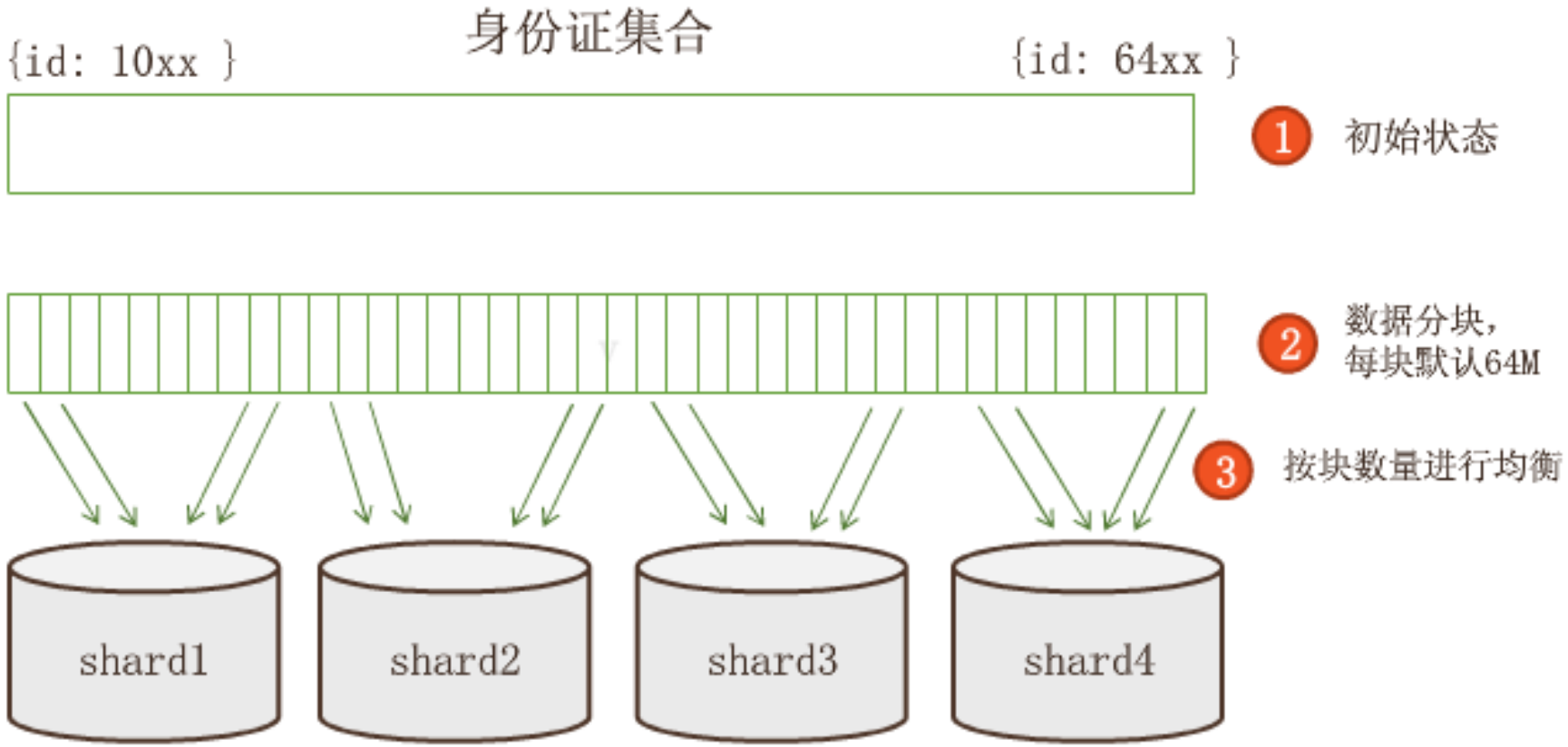
1：写性能扩展

2：读性能扩展

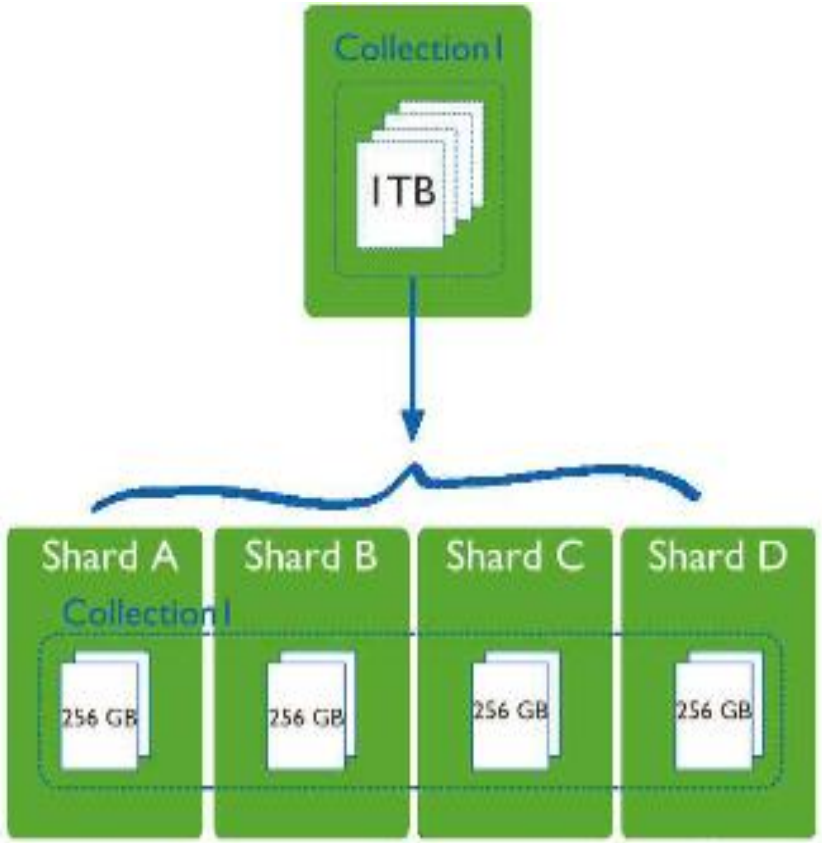
3：地理分布数据

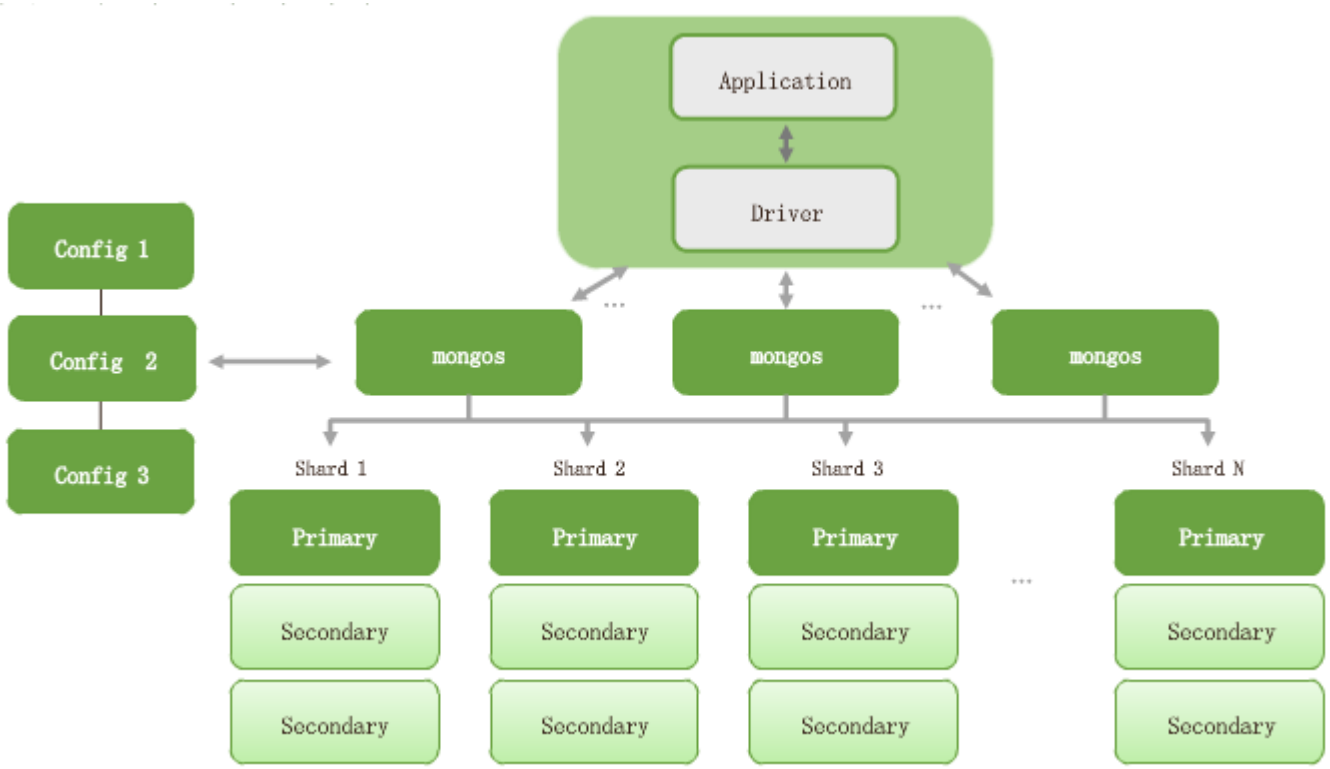
4：备份的快速恢复

分片原理


身份证集合
{id: 10xx }
{id: 64xx }
1
初始状态
2
数据分块，
每块默认64M
3
按块数量进行均衡
shard1
shard2
shard3
shard4

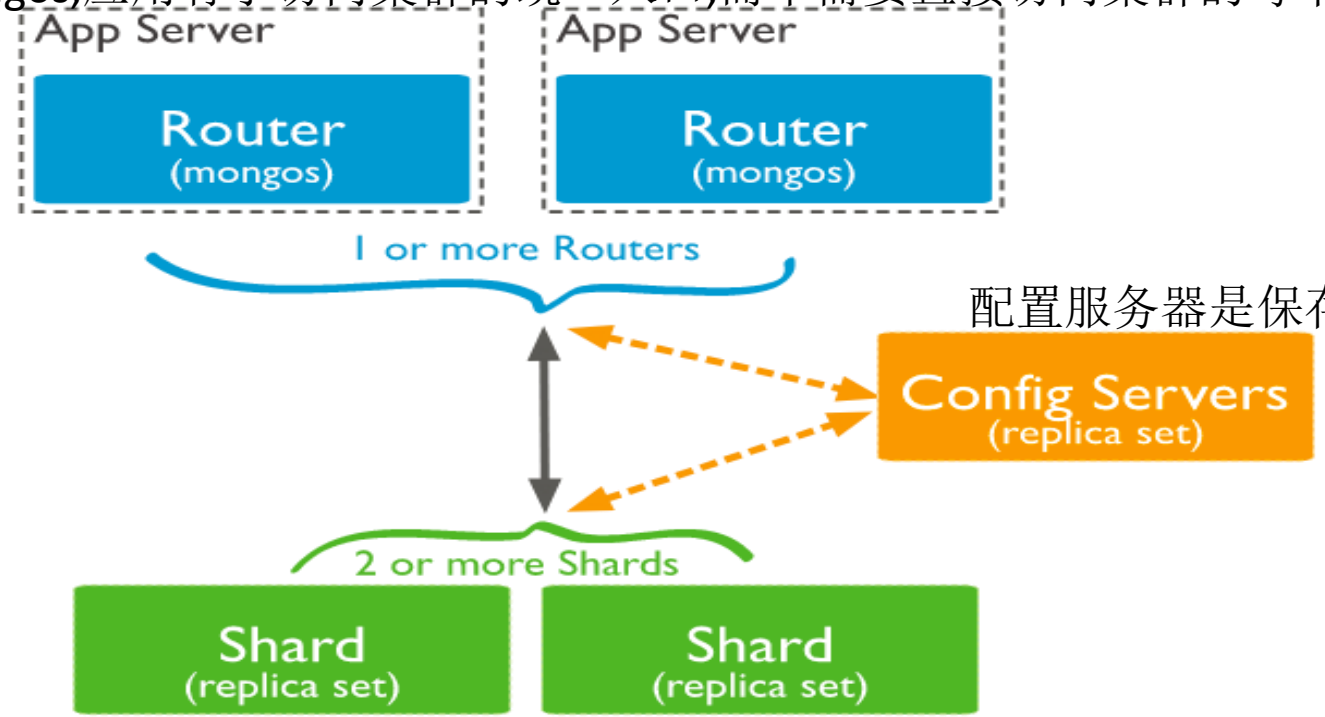
Mongodb分片集群





Mongodb集群组件

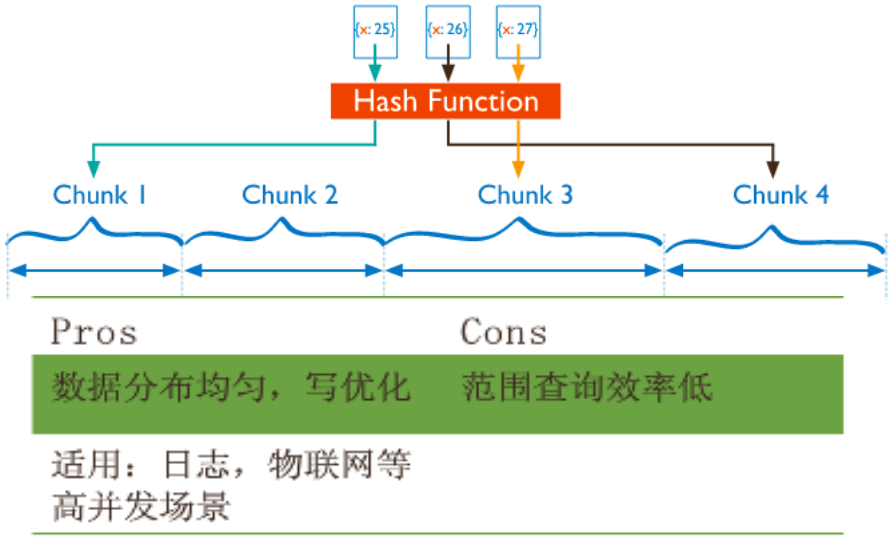
mongos 负责将查询与写入分发到 分片 中.
使用 mongos,应用有了访问集群的统一入口,而不需要直接访问集群的每个分片.

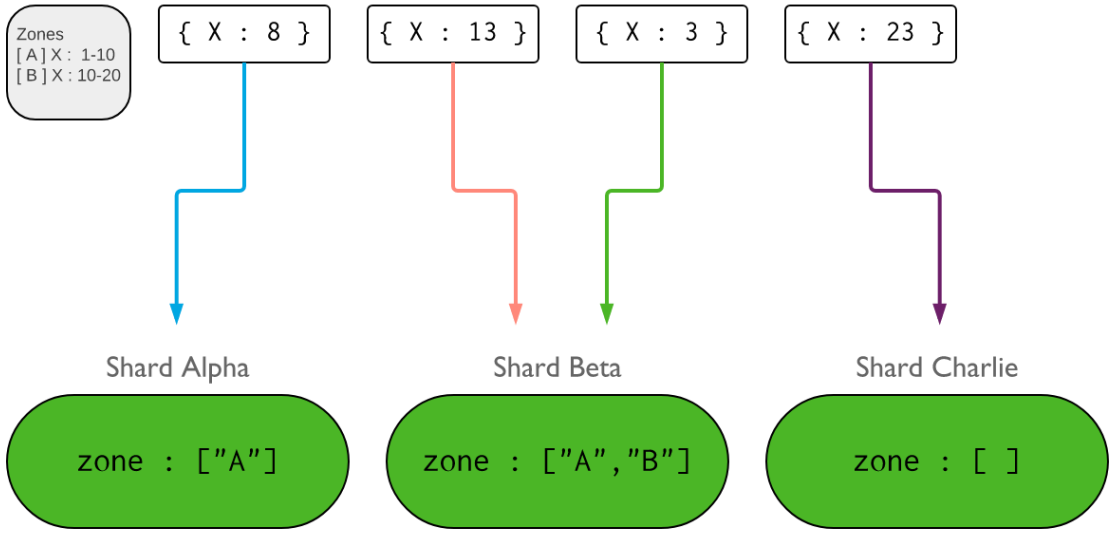


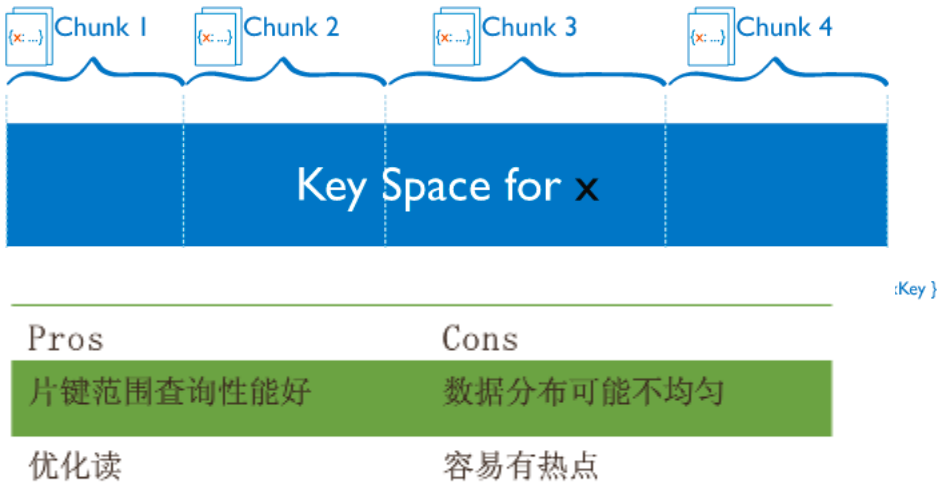
配置服务器是保存集群中 元信息 的特殊 mongod .

分片是存储了集群一部分数据的 mongod
或者复制集.所有分片存储了集群的全部数据.

Mongodb分片策略







Mongodb如何选择分片键Key

- 基数要大
- 写操作分布均匀
- 查询定向性好

举个例子

```
{
        _id: ObjectId(),
       user: 123,
       time: Date(),
       subject: “...”,
       recipients: [],
       body: “...”,
       attachments: []
}
```

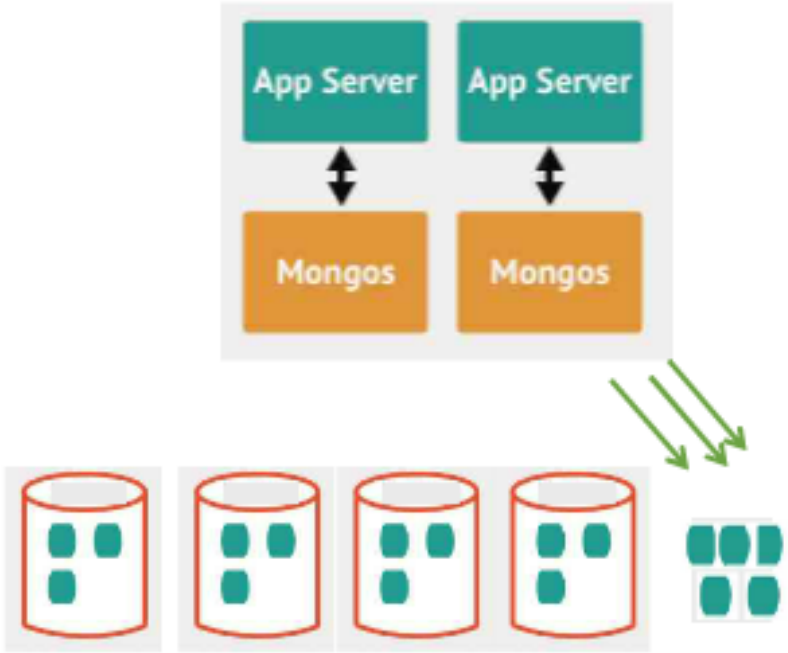
举个例子

片键： { _id: 1}

基数 ✓

写分布 ✗

定向查询 ✗



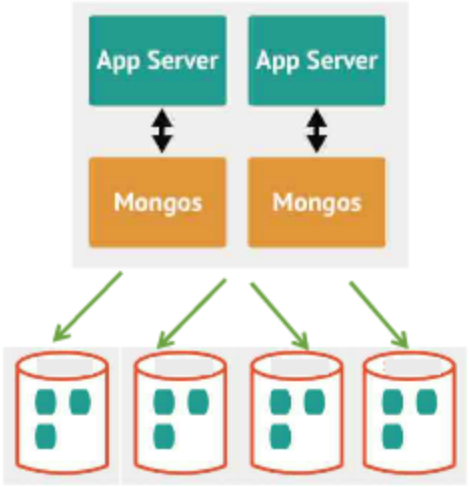
举个例子

片键： { _id: “hashed” }

基数 ✓

写分布 ✓

定向查询 ✗



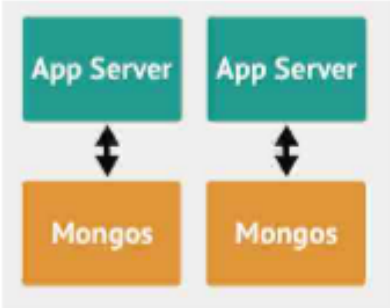
举个例子

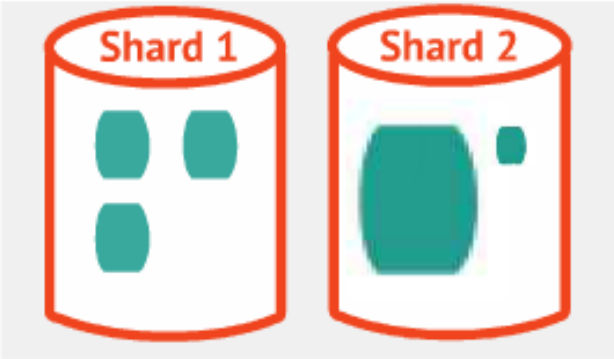
片键： { userid: 1}

基数 ✗

写分布 ✓

定向查询 ✓





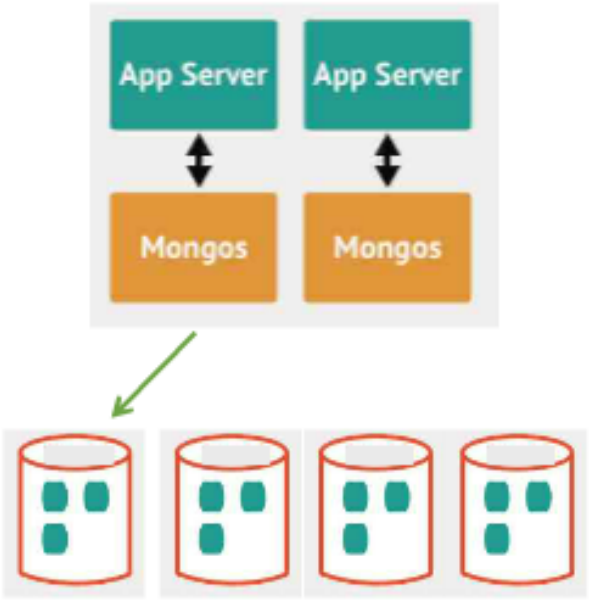
举个例子

片键： { userid: 1, time:1}

基数 ✓

写分布 ✓

定向查询 ✓



片键选择

1 .片键的选择？
原则：结合业务需求，尽可能选择存储分片均匀的键
(1)字段的值有限
　　结果：理论上分片有限；缺点就是严重制约了业务扩展的能力；
(2)时间戳或者自动增长ID
　结果：数据大多都逻辑连续和顺序地插入到一个分片上，对于插入等，没有充分利用磁盘I/O性能；
　优点：无限数量的字段值，提供无限数量的分片，对业务扩展不需要担心；
　缺点：单一且不可分散的热点
　使用时间分片：
　　这个跟用_id（包含记录插入时的时间戳）类似，坏处是你最近的写入都会分到某个特定的『分片』
　　写入IO 后面还有数据迁移（均衡）带来IO

片键选择

(3)随机片键
　　优点：分散热点
　　缺点：数据越大，性能越差
(4)结合业务特点，组合的搜索条件就是片键
　结果：高效
 往往结合自动增长ID和一个类似随机片键组合在一起。

如果以日期做为分片键，为了避免大的chunk，我们可以把日期精确到 时分秒 然后做分片键

shardkey 如果用hash分片，只能包含一个字段
比如{somekey: "hashed"}

什么时候需要分片

(1)单点数据库服务器存储成为瓶颈

(2)单点数据库服务器的性能成为瓶颈

(3)大型应用分散数据以充分利用内存

案例

Linux环境

主机IP	OS	备注
10.10.23.121	RH7.2	48c 256G 3T SSD
10.10.23.122	RH7.2	48c 256G 3T SSD
10.10.23.123	RH7.2	48c 256G 3T SSD

MongoDB版本：**mongodb3.4.10**

案例

IP及端口规划

实例	10.10.23.121	10.10.23.122	10.10.23.123
Router	mongos1（17615）	mongos2（17615）	mongos3（17615）
Config	config server1（20000）	config server2（20000）	config server3（20000）
Shard1	Shard1-主（28011）	Shard1-从（28011）	Shard1-从（28011）
Shard2	Shard2-从（28012）	Shard2-主（28012）	Shard2-从（28012）
Shard3	Shard3-从（28013）	Shard3-从（28013）	Shard3-主（28013）

案例

mongod.conf参数

```
# where to writelogging data.
systemLog:
  destination: file
  logAppend: true
  logRotate: reopen
  path: /data/mongodb/logs/37017.log
# Where and howto store data.
storage:
  dbPath: /data/mongodb/data37017
  journal:
    enabled: true
  directoryPerDB: true
  engine: wiredTiger
```

```
#########storage.wiredTigerOptions
  wiredTiger:
    engineConfig:
      cacheSizeGB: 32
      directoryForIndexes: true
    indexConfig:
      prefixCompression: true
###########operationProfilingOptions
#operationProfiling:
   #slowOpThresholdMs: 50
   #mode: "all"
########ProcessManagementOptions
processManagement:
  fork: true
  pidFilePath:
/data/mongodb/data37017/37017.pid
```

案例

```
# networkinterfaces
net:
  port: 37017
  #bindIp:
  maxIncomingConnections: 27600
security:
  keyFile: /data/mongodb/keyFilers0.key
###########replicationOptions
replication:
  replSetName: rs37017
  oplogSizeMB: 20480
  secondaryIndexPrefetch: all
sharding:
  clusterRole: shardsvr    #configsvr or shardsvr
```

```
##Enterprise-Only Options
#snmp:
setParameter:
  enableLocalhostAuthBypass: true
  replWriterThreadCount: 32
  wiredTigerConcurrentReadTransactions: 1000
  wiredTigerConcurrentWriteTransactions: 1000
```

案例

```
config={_id:"rs37017",members:[{_id:1,host:“ip:37017",prio
rity:36,tags:{'use':‘order-1'}}]}
rs.initiate(config)
rs.add({_id:2, host:" ip:37017", priority:33, tags:{'use':‘order-
2'}})
rs.add({_id:3, host:" ip:37017 ", priority:30,
tags:{'use':‘order-3'}})
```

先在各个副本集上创建了账号

```
use admin
db.createUser(
{
   user: "admin",
   pwd: "Sp106#Rt9wL",
   roles:
   [
    {role: "userAdminAnyDatabase", db: "admin"},
    { role: "readAnyDatabase", db: "admin" },
    { role: "dbOwner", db: "admin" },
    { role: "userAdmin", db: "admin" },
    { role: "root", db: "admin" },
    { role: "dbAdmin", db: "admin" },
   ]
 }
)
```

案例

config.conf参数

```
# where to writelogging data.
systemLog:
  destination: file
  logAppend: true
  logRotate: reopen
  path:
/data/mongodb/logs/configsvr.log
# Where and howto store data.
storage:
  dbPath: /data/mongodb/confg30005
  journal:
    enabled: true
  directoryPerDB: true
  engine: wiredTiger
```

```
#########storage.wiredTigerOptions
  wiredTiger:
    engineConfig:
      cacheSizeGB: 1
      directoryForIndexes: true
    indexConfig:
      prefixCompression: true
###########operationProfilingOptions
#operationProfiling:
   #slowOpThresholdMs: 50
   #mode: "all"
########ProcessManagementOptions
processManagement:
  fork: true
  pidFilePath:
/data/mongodb/confg30005/30005.pid
```

案例

```
# networkinterfaces
net:
  port: 30005
  #bindIp:
  maxIncomingConnections: 8500
#security:
  #keyFile:
/data/mongodb/keyFilers0.key
##########replicationOptions
replication:
  replSetName: configReplSet
  oplogSizeMB: 2048
  secondaryIndexPrefetch: all
sharding:
  clusterRole: configsvr    #configsvr or
shardsvr
```

配置为副本集模式

```
 rs.initiate( {_id: "configReplSet",
   configsvr: true,
   members: [
      { _id: 1, host:
“ip:30005", priority:70, tags:{'use':'3212
-70' }},
      { _id: 2, host:
"ip:30005", priority:60, tags:{'use':'3212
-60' }},
      { _id: 3, host:
"ip:30005", priority:50, tags:{'use':'3212
-50' }}
   ]
} )

configReplSet:PRIMARY> rs.status()
```

案例

mongos.conf　三台一样

```
systemLog:
  destination: file
  path: /data/logs/mongos.log
  logAppend: true
processManagement:
  fork: true
  pidFilePath: /data/mongos/mongos.pid
net:
  port: 20000
sharding:
  configDB: configReplSet/ip:30005, ip:30005, ip:30005
security:
  keyFile: "/data/keyfile/security"
```

说明 configReplSet是副本集名称

Keyfile给于600权限

```
mongos> use admin
添加账号(和mognod的一样即可)
use admin
db.createUser(
{
    user: "admin",
    pwd: "",
    roles:
    [
      {role: "userAdminAnyDatabase", db: "admin"},
      { role: "readAnyDatabase", db: "admin" },
      { role: "dbOwner", db: "admin" },
      { role: "userAdmin", db: "admin" },
      { role: "root", db: "admin" },
      { role: "dbAdmin", db: "admin" },
    ]
  }
)
```

案例

```
switched to db admin
mongos> db.runCommand({addshard:"rs37017/ip:37017, ip:37017, ip:37017",name:"rs37017_tag"});
mongos> db.runCommand({addshard:"rs37018/ip:37018,ip:37018,ip:37018",name:"rs37018_tag"});
mongos> db.runCommand({addshard:"rs37019/ip:37019,ip:37019,ip:37019",name:"rs37019_tag"});
```

案例

注意：在mongos上创建的账号，在mongod是看不到的，而且在config上是可以看到的

部署完后，然后在停止各个进程 mongod config mongos
将配置文件的keyfile认证文件打开
然后重启mongod config mongos
登录后 会通过认证访问

分片案例

db.ORDER_CUSTOMER_SUCCESS.count() *** 发送[ORDER_CUSTOMER]成功
ygzrs1:PRIMARY> db.ORDER_CUSTOMER_SUCCESS.count()
211981030
数据集合启动分片
mongos>
db.ORDER_CUSTOMER_SUCCESS.createIndex({“ORDER_TIME":1},{background:true})
mongos>
db.runCommand({shardcollection:“ORDER.ORDER_CUSTOMER_SUCCESS",key:{“ORDER_TIM
E":1}})

以hash作为分片[*** 发送[ORDER_CUSTOMER]失败]
db.ORDER_CUSTOMER_FAILED.createIndex({"_id":"hashed"},{background:true})
db.runCommand({shardcollection:“ORDER.ORDER_CUSTOMER_FAILED”,key:{“_id”:hashed},n
umInitialChunks:300}) 集合为空的时候使用；
每个节点分配300，每个chunk 64M 然后再乘以shard数，就是分配的总大小空间;

根据业务查询最多的接口设计;

索引案例

db.ORDER_CUSTOMER_SUCCESS.find({"DATAGRAM_TYPE" :NumberInt(23),"RETURN_CODE" :{"$in":[408,704,705,802]}}).sort({"POST_TIME":1}).limit(100000).explain()

解析：

第一个条件是=

第二个条件 范围 或者$lt $gt

第三个条件是 排序

创建索引规则：先等值 ,在排序,然后在范围

综合性能分析案例

一：慢查分析和ip来源分析
先安装pip pymongo 在安装mtools
#tar xvf pip-1.5.4.tar.gz
cd pip-1.5.4
python setup.py install

tar xvf pymongo-2.3.tar.gz
cd pymongo-2.3
python setup.py install

tar xvf mtools-1.1.8.tar.gz
cd mtools-1.1.8
python setup.py install
部署完毕
会自动安装两个工具 mlogfilter mloginfo
工具使用
1.分析日志信息 查找某个时间出现问题的日志
注意时间根据主机生产的，可以先more看一下mongodb日志的时间格式
sed -n '/2018-06-12T09:00.*/,/2018-06-12T10:00.*/p' mongodb日志 > 180612_09-10.log
2.通过mloginfo统计查看日志中慢查询的分类
mloginfo --queries 180612_09-10.log --重点
3.通过mloginfo统计日志中connections的来源情况 查找ip等信息
mloginfo 88.log --connections 或者使用操作系统命令 tcpdump
注意：
如果在做mloginfo时出错，则执行切分日志 操作系统命令
split -b 1024m 180612_09-10.log 每个日志在1G左右，如果1g还出错，则调整1024m为再小的值

综合性能分析案例

二.监控整体状况和直观发现是否有问题

mongostat --host= --port=

关注qr/aw值，小于20左右即可

```
insert query update delete getmore command % dirty % used flushes vsize   res qr|qw ar|aw netIn netOut conn    set repl     time
   *70  1338   *126     *0       0   116|0     0.9  100.0       0 40.4G 33.7G 273|3 125|0  149k     1m 1041 rsOrder  SEC 22:06:46
    *0  3771     *0     *0       0   395|0     0.8  100.0       0 40.4G 33.7G 242|3 128|0  435k     4m 1053 rsOrder  SEC 22:06:48
    *0  1813     *0     *0       0   104|0     0.8  100.0       0 40.4G 33.7G 257|3 128|0  200k     2m 1025 rsOrder  SEC 22:06:50
    *0   495     *0     *0       0    69|0     0.8  100.0       0 40.4G 33.7G 275|3 127|0   63k   526k 1030 rsOrder  SEC 22:06:52
    *0  2871     *0     *0       0    81|0     0.6  100.0       1 40.4G 33.7G 258|3 128|0  304k     3m 1024 rsOrder  SEC 22:06:53
    *0  2052     *0     *0       0    82|0     0.5  100.0       0 40.4G 33.7G 245|3 128|0  226k     2m 1018 rsOrder  SEC 22:06:54
    *0  2154     *0     *0       0   115|0     0.2  100.0       0 40.4G 33.7G 250|3 126|0  235k     2m 1023 rsOrder  SEC 22:06:56
    *0  3966     *0     *0       0   198|0     0.0  100.0       0 40.4G 33.7G 261|3 127|0  433k     3m 1017 rsOrder  SEC 22:06:59
    *0  1211     *0     *0       0    45|0     0.0  100.0       0 40.4G 33.7G 261|3 128|0  130k     1m 1026 rsOrder  SEC 22:07:00
    *0  1335     *0     *0       0   101|0     0.0  100.0       0 40.4G 33.7G 257|3 127|0  153k     1m 1041 rsOrder  SEC 22:07:02
```

综合性能分析案例

三.mongodb日志切割(每天生成一个)
先看一下# ps -ef | grep log 看有没有类似如下的命令
root 1224 1 00:00:00 /sbin/rsyslogd -i /var/run/syslogd.pid -c 5

vim /etc/logrotate.d/mongo

```
/data/log/mongodb/mongod.log {
    daily
    rotate 30
    dateext
    compress
    missingok
    notifempty
    sharedscripts
    copytruncate
    postrotate
        /bin/kill -SIGUSR1 `cat /data/mongodb/mongod.lock 2> /dev/null` 2> /dev/null || true
    endscript
}
```

综合性能分析案例

参数说明
/data/log/mongodb/mongod.log为mongodb日志路径 rotate保留日志30天
/data/mongodb/mongod.lock为mongodb启动后都有一个lock文件
根据贵公司需求修改即可

测试ok，先单独执行如下命令看是否切换日志(手动执行)
/bin/kill -SIGUSR1 `cat /data/mongodb/mongod.lock 2> /dev/null` 2> /dev/null || true

注意：不会影响线上活动
启动 这个在后端执行，相当于crontab
logrotate -vf /etc/logrotate.conf
[
logrotate [-vf] logfile选项与参数：
v: 启动显示模式，会显示 logrotate 执行的过程
f: 不论是否符合配置文件地规则，强制每个日志都进行 rotate 的动作
]
会有一个进程启动
ps -ef | grep log
root 1224 1 00:00:00 /sbin/rsyslogd -i /var/run/syslogd.pid -c 5

监控

1.mongod的实例是否存在
监控端口或进程id是否存在即可
2.副本集是否存活
3.tps/s 每秒的tps执行
insert+update+delete
4.每秒的qps
query
5.qr-读队列积压情况
阈值>20
6.qw-写队列积压情况
阈值>20
7.内存使用情况
取 res值即可
8.mongodb每个实例的连接数
conn

9.主机监控
a.主机内存空闲比率
free+cache
b.磁盘使用比率 iostat -xmt 2
监控值 await<=10
%util <=85%
c.负载 top
load average: 0.00, 0.04, 0.03
根据cpu 比如每个服务器24核
cpu(24*0.7) 负载可以设置为 17左右
d. 磁盘使用空间
< 85%
e.cpu监控
%us 用户使用cpu < 80%

FOUR

Mongodb集群模式典型场景

Mongodb场景

- 网站动态数据，需要实时的插入，更新与查询。
- 存储大尺寸，低价值的数据。
- 高伸缩性的集群场景。
- 文档化结构的数据存储及查询。
- 实时数据存储
- BSON数据格式非常适合文档化格式的存储和查询(评论、作业)
- 数十台或上百台的数据库集群
- 审计日志—audit表等目前存储在mysql
- GridFS--文件存储：小文件和图片等
- ……

Mongodb集群优劣对别

复制集群

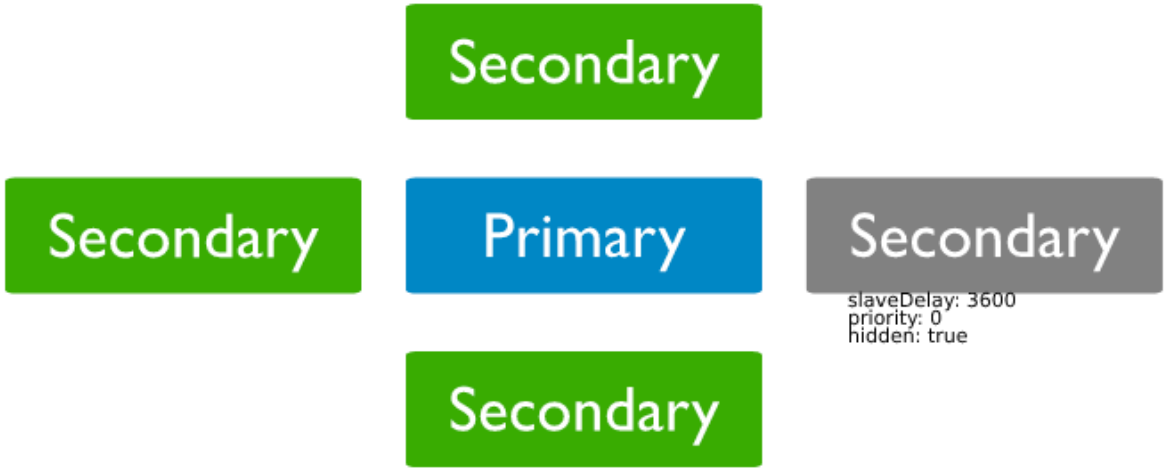
- 快速构建
- 维护简单
- 容量与处理能力受到单机限制

分片Sharding

- 可以无限水平扩展容量与处理能力
- 维护成本相对复制集较高

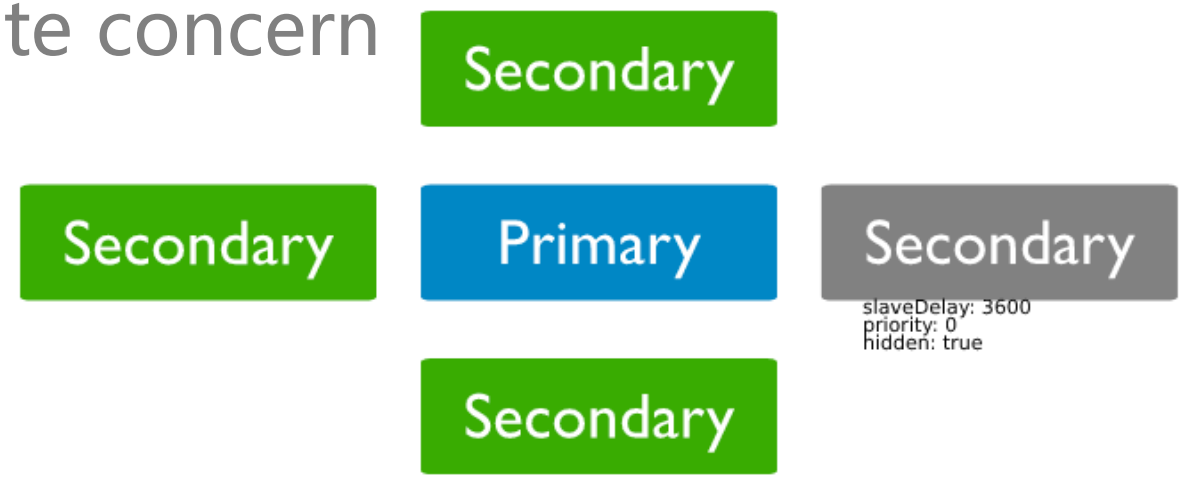
常规互联网场景下集群架构设计

- 强一致场景从主上读，如交易相关以及核心数据
- 非强一致场景可考虑分离至从，比如异步逻辑等。
- 离线业务、与其他数据层的同步在延时节点上进行。

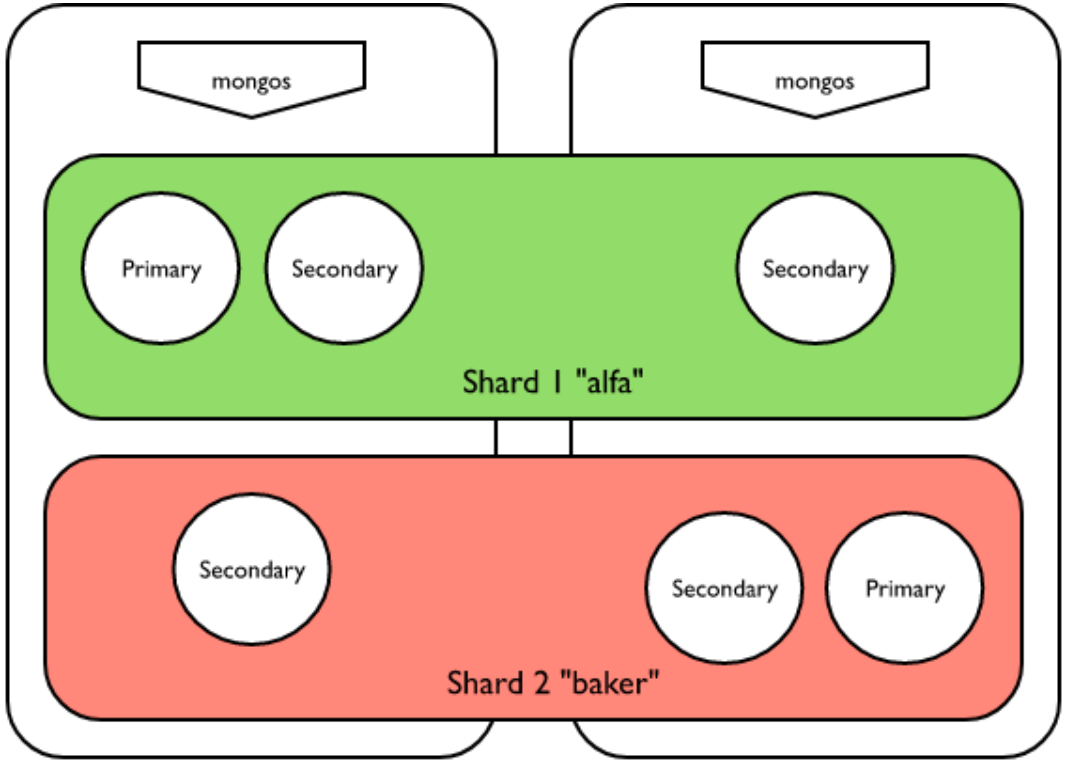


进阶的常规场景场景

- 节点分布在多个机房
- 保证主机房拥有多数个节点
- 业务应用跨机房部署在读的时候选择nearest
- 重要业务的写，可以定义majority的write concern 或根据机房定义repl set tag而后定义多机房的write concern



物联网场景



业务以{ datacenter : 1, carid : 1 }作为联合片键，sharding架构分布在2个datacenter中alfa 和 bravo。

```
{
   "_id" : ObjectId("56f08c447fe58b2e96f595fa"),
   "message_id" : 329620,
   "datacenter" : "alfa",
   "carid" : 123,
   ...
}
```

```
sh.addShardTag("shard0000", "alfa")
sh.addShardTag("shard0001", "bravo")

sh.addTagRange(
  "iot.teaambition",
  { "datacenter" : "alfa", "carid" : MinKey },
  { "datacenter" : "alfa", "carid" : MaxKey },
  "alfa"
)

sh.addTagRange(
  "iot.teaambition",
  { "datacenter" : "bravo", "carid" : MinKey },
  { "datacenter" : "bravo", "carid" : MaxKey },
  "bravo"
)
```

默认情况下业务都往本地shard进行写入，如果本地的shard节点挂了或者超过返回时间，业务内部维护一个简单的状态表将该shard置为down，则向另一个shard中写入并且与此同时修改数据中的datacenter字段。

FIVE

研发特别注意

研发特别注意

使用修改器

通常文档只需要部分更新，可使用原子的更新修改器，使得文档更新起来更高效。更新修改器是中特殊的键，用来指定复杂的操作，比如增加，删除或者调整键，还可能是操作数组或者内嵌文档。

- $unset修改器用于删除键

将集合infos的title列删除

```
:PRIMARY> db.spampic.update({},{$unset:{"matStr":1}}, 0, 1)
```

- 修改字段值

true值修改为false

```
:PRIMARY> db.antiams.count()
3286
keepRS:PRIMARY> db.antiams.findOne()
{
        "_id" : ObjectId("5734737d252ef06b96f51e24"),
        "modified" : ISODate("2016-06-02T03:27:05.249Z"),
        "refe" : ObjectId("57345bd839b59de33671cca8"),
        "action" : {
                "from" : "normal",
                "to" : "autoBan"
        },
        "isDeal" : true
```

研发特别注意

- 修改表全部数据

db.antiams.update({spam:"true"},{$set:{spam:true}},{multi:true})
db.antiams.update({spam:"false"},{$set:{spam:false}},{multi:true})
:PRIMARY> db.antiams.find({spam:false}).count()
1643
:PRIMARY> db.antiams.find({spam:true}).count()
1498

- 根据achievement条件更新groupName字段值 -全量更新

:SECONDARY>
db.userachie.find({achievement:ObjectId("564001415372f4dfc94dd1eb")}).count()
591879
db.userachie.update({achievement:ObjectId("564001415372f4dfc94dd1eb")},
{$set:{groupName:'eventAchievement'}}, false, true)
通过achievement索引查询后更新groupName，这样是比较快的。

- 更新多条数据

db.users.update({'stateValue':0}, {'$set':{'stateValue':10, 'state':"normal"}}, {'multi':true})

- 更新一条

db.userachie.update({achievement:ObjectId("564001415372f4dfc94dd1eb")},
{$set:{groupName:'eventAchievement'}})

Mongoing
中文社区

谢 谢