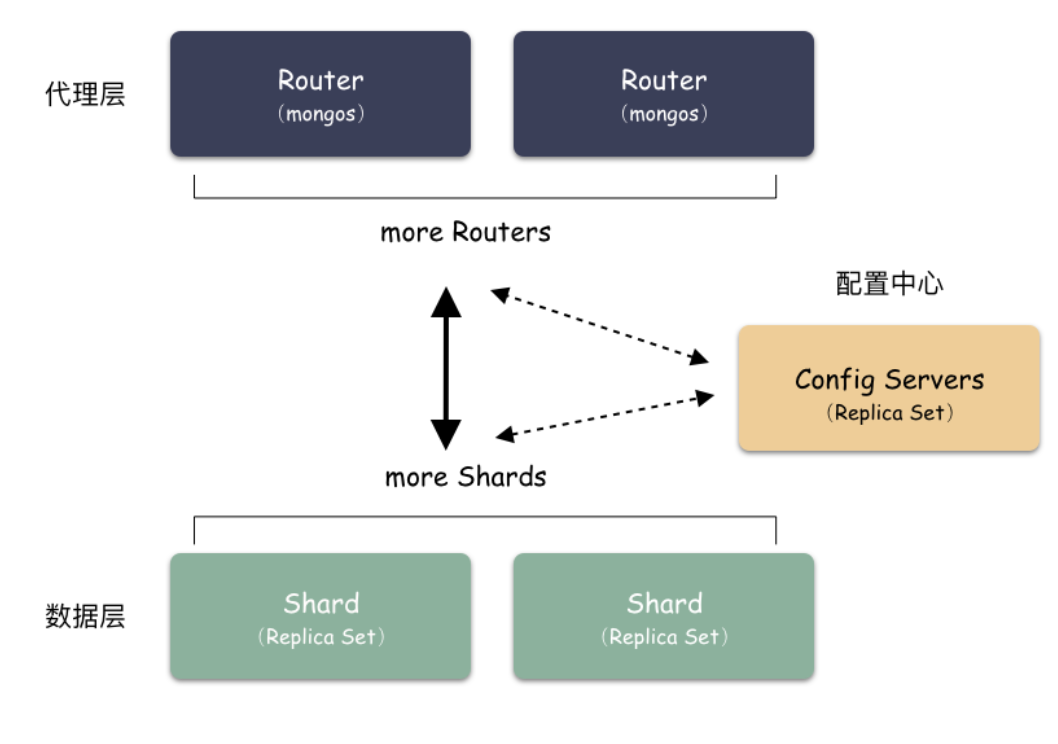
# Mongodb集群

## 简介

Mongodb通过分片的方式将数据分散到各个节点，通过副本的方式变得高可用。

Mongodb有三个组件：

* **shard**：mongodb的分片组件，在服务器节点上配置shard组件，指定主节点，副本节点，然后启动各个服务器节点的shard组件即可；
* **mongos:** 数据库请求的入口，所有请求都需要经过mongos进行协调，无需在应用层面利用程序来进行路由选择，为防止mongos单节点故障，一般需要配置多个mongos；
* **conf server:** 配置服务器，存储所有数据库元数据 (分片、路由)的配置，mongos在第一次启动或后期重启时候，就会从config server中加载配置信息，如果配置服务器信息发生更新会通知所有的mongos来更新自己的状态。为防止配置文件服务器存在单点问题，需要做高可用。



下面搭建一个单分片一主一副一仲裁节点的集群。Mongodb版本为4.4.18

假设将主机节点分别为192.168.2.136,192.168.2.137,192.168.2.138。

## 安装mongodb

### 下载Mongodb

<https://www.mongodb.com/try/download/community-kubernetes-operator>

### 解压tar包

tar -zxvf mongodb-linux-x86\_64-ubuntu1804-4.4.18.tgz

mv mongodb-linux-x86\_64-ubuntu1804-4.4.18 mongodb

### 创建各个组件的目录和文件

mkdir -p {mongodb目录}/mongos/{log,conf}

mkdir -p {mongodb目录}/mongoconf/{data,log,conf}

mkdir -p {mongodb目录}/shard1/{data,log,conf}

touch {mongodb目录}/mongos/log/mongos.log

touch {mongodb目录}/mongoconf/log/mongoconf.log

touch {mongodb目录}/shard1/log/shard1.log

### 配置config server组件

在三个节点都进行以下配置：

创建 {mongodb目录}/mongoconf/conf/mongoconf.conf文件，并添加以下内容：

dbpath={mongodb目录}/conf/data

logpath={mongodb目录}/conf/log/mongoconf.log

logappend=true

bind\_ip=0.0.0.0

port=21000

maxConns=1000

journal=true

journalCommitInterval=200

fork=true

syncdelay=60

oplogSize=1000

configsvr=true

replSet=replconf

启动 ./bin/mongod -f ./mongoconf/conf/mongoconf.conf

登录 ./bin/mongo --port 21000

配置config server集群

use admin;

config = {\_id:"replconf",members:[

{\_id:0,host:"192.168.2.136:21000"},

{\_id:1,host:"192.168.2.137:21000"},

{\_id:2,host:"192.168.2.138:21000"},]

}

rs.initiate(config);

查看配置情况：

rs.status();

### 配置shard组件

新建{mongodb目录}/shard1/conf/shard.conf，并填写内容：

dbpath={mongodb目录}/shard1/data

logpath = {mongodb目录}/shard1/log/shard1.log

bind\_ip=0.0.0.0

port=22001

logappend=true

fork=true

oplogSize=4096

shardsvr=true

replSet=shard1

启动shard服务： mongod -f {mongodb目录}/shard1/conf/shard.conf

登录192.168.2.136的shard1服务： ./bin/mongo {host}:{ip}

use admin;

config={\_id:"shard1",members:[

{\_id:0,host:"192.168.2.136:22001"},

{\_id:1,host:"192.168.2.137:22001",arbiterOnly:true},

{\_id:2,host:"192.168.2.138:22001"},]

}

rs.initiate(config);

查看集群状态

rs.status();

### 配置mongos组件

新建{mongodb目录}/mongos/conf/mongos.conf，并填写内容：

logpath={mongodb目录}/mongos/log/mongos.log

logappend=true

port=20000

maxConns=1000

configdb=replconf/192.168.2.136:21000,192.168.2.137:21000,192.168.2.138:21000

fork=true

启动mongos服务: mongos -f {mongodb目录}/mongos/conf/mongos.conf

加入分片配置：

如果有多个mongos，只需要在其中一个添加分片配置即可

登录mongos： mongo {host}:{ip}

use admin;

db.runCommand({addshard:"shard1/192.168.2.136:22001,192.168.2.137:22001,192.168.2.138:22001"})

# Mongodb基本操作

## Mongodb对象

### 数据库-db

MongoDB也存在“数据库”概念。可以使用use <databasename>的指令切换到某个数据库并新增数据，如果不存在此数据库，那么会自动创建此数据库；若没有切换数据库就新增数据，那么数据会建在默认的数据库”test”下.

### 集合-collection

文档时MongoDB存储数据的最基本单元，相当于关系型数据库库中的表。文档内包含多个键值对，值可以是数值，字符串，数组或子文档等等。

### 字段-Field

文档中的元素。在MongdoDB中，字段可以弹性地增加或减少，并不需要提前定义。

### 索引-index

MongoDB索引的使用方法和原理都与关系型数据库的索引相同。

### 用户-User

如果MongoDB开启了安全认证，则会在数据库中产生用户表。在MongoDB中，每个数据库都有独立的用户表。在用户权限配置方面，MongoDB与关系型数据库基本一致。

## 文档数据格式

MongoDB的文档数据格式基于JSON进行了改良，采用了BSON(Binary JSON)格式进行存储。

BSON与JSON同样为无模式化的存储形式，可以支持内嵌的文档以及数据。BSON的值类型比JSON更加全面，支持Date、二进制、Decimal、ObjectID(对象ID)类型。

## MongoDB的数据类型

### 数据类型

MongoDB每个类型都有对应的数字，如果要修改字段的类型，那么需要依照对应的名称或数字来修改。

double – 1

string – 2

object – 3

array – 4

binData – 5(二进制类型)

objectId – 7

bool – 8

date - 9

null – 10

regex – 11

javascript – 13

javascriptWithScope – 15

int – 16

timestamp – 17

long – 18

decimal – 19

### 对象ID- ObjectId

MongoDB文档中有一个自动生成的字段--\_id, 此字段会被自动指定为一个不重复的值，若不对其进行定义，则MongoDB的驱动程序在写入数据时会自动生成一个类型为ObjectId的唯一键。

ObjectId长度为12字节，由几个2-4字节的链组成。每个链代表并指定文档身份的具体内容。

1. 1-4 字节：表示4字节的自Unix纪元以来的秒数；
2. 5-7 字节：表示3字节的机器标识符；
3. 8-9字节：表示2字节的进程ID；
4. 10-12字节：表示3字节的由计数器生成的值



### 数组

MongoDB可以将数组作为一种数据类型赋值给某个字段。数组由零个至多个元素组合而成，放在中括号[]中，元素间用逗号隔开，且每个元素都可以是不同的数据类型，比如[1,2,”foo”,”bar”]

### 日期和时间

在MongoDB中可以使用多种方法来创建日期，可以使用Date(),new Date(), ISODate()的方式得到时间。

Date()：得到字符串类型的时间

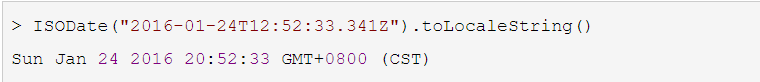
new Date(): 得到日期对象类型的日期(UTC时间)

ISODate(): 得到日期对象类型的日期(UTC时间)



MongoDB存储时间类型数据，都是先转换为UTC时间，然后存储到数据库中。

toLocaleString()可以返回当前时区的时间字符串



## 文档操作

### 数据库

创建

use <db\_name>: 自动创建数据库并切换；

查看当前数据库

db

查看所有数据库

show dbs

删除当前数据库

db.dropDatabase()

### collection

创建

db.createCollection(“<collection\_name>”)



删除

db.<collection\_name>.drop()



查看collection列表

show collections



### 文档

#### 插入

插入文档有三个特点：

* 如果没有指定数据库，则在插入文档时会自动创建一个名为“test”的数据库；
* 在插入文档时，如果没有此集合，则MongoDB会自动创建这个集合名称；
* 在插入文档时，若没有指定“\_id”字段，则MongoDB会自动新增一个“\_id”字段，默认是ObjectId类型。如果指定”\_id”字段，则其值是不能重复的，可以是数字或字符串类型。

##### insert()

此方法可以插入单笔或多笔文档。此方法已过期

格式： db.<collection\_name>.insert(

[{bson文档1},{bson文档2}]

)

例子：

db.Person.insertMany(

[

{name: 'dao4', age: 18},

{name: 'dao5', age: 20}

],

{

writeConcern: {wtimeout:5000},

ordered:false

})

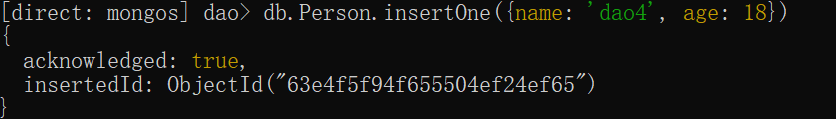
writeConcern: 可选参数，表示是否使用写入策略；

ordered: 可选参数，表示插入文档时是否为有序插入，默认为true。若为有序插入，则当插入文档中有一个发生错误时，MongoDB将不再继续插入后续的文档；若为无序插入，则当插入的文档有一个文档发生错误时，MongoDB将会继续插入后续的文档。

##### insertOne()

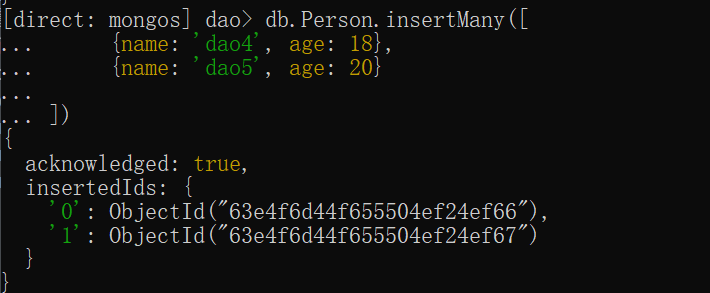
插入一条数据

例子：



##### insertMany()

insertMany()可以插入多个文档



#### 更新

##### updateOne()

updateOne()也是一种更新文档的操作方法。

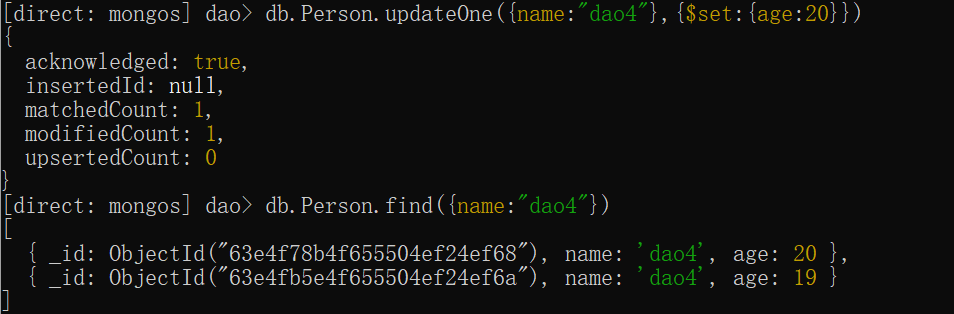
格式: db.<collection\_name>.update(

<query>,

<update>

)

通过query内容查询，通过update内容更新文档。

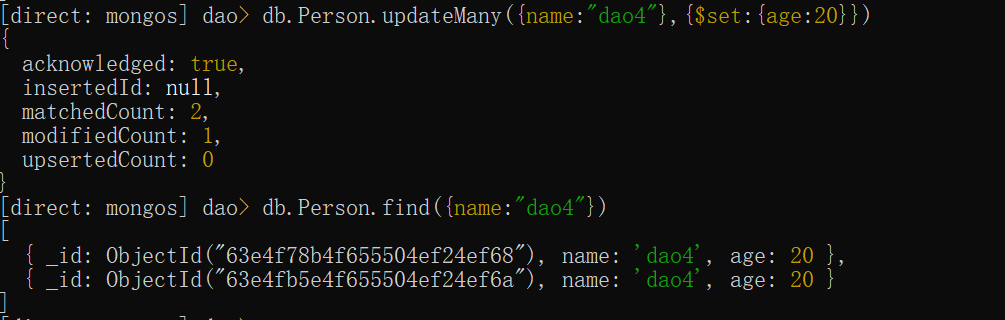


$set: 表示更新指定的字段

$inc: 表示自增指定的字段数值

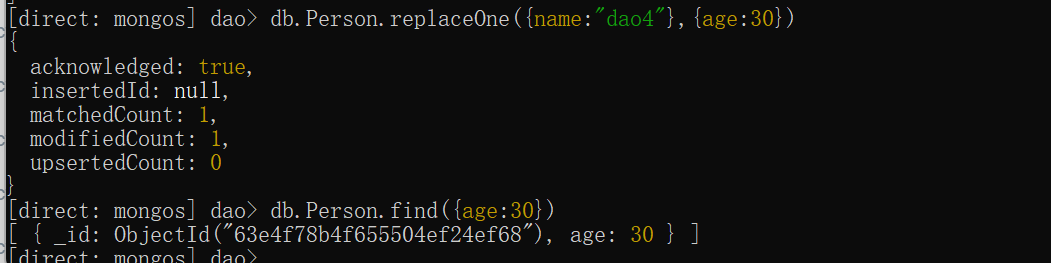
##### updateMany()

updateMany()是更新多条文档的操作方法。与updateOne()的不同是会更新所有匹配到的文档。



##### replaceOne()

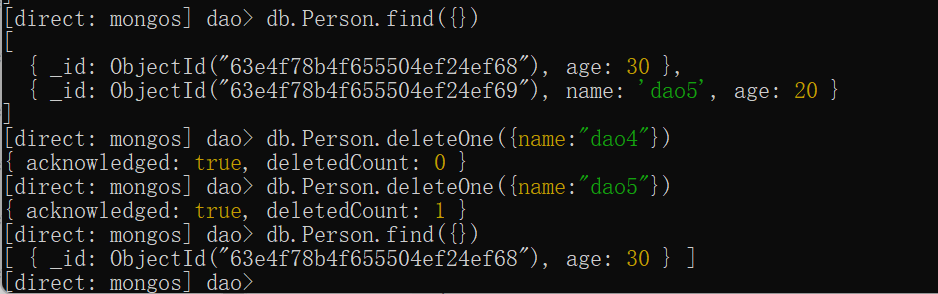
替换单个文档。



#### 删除

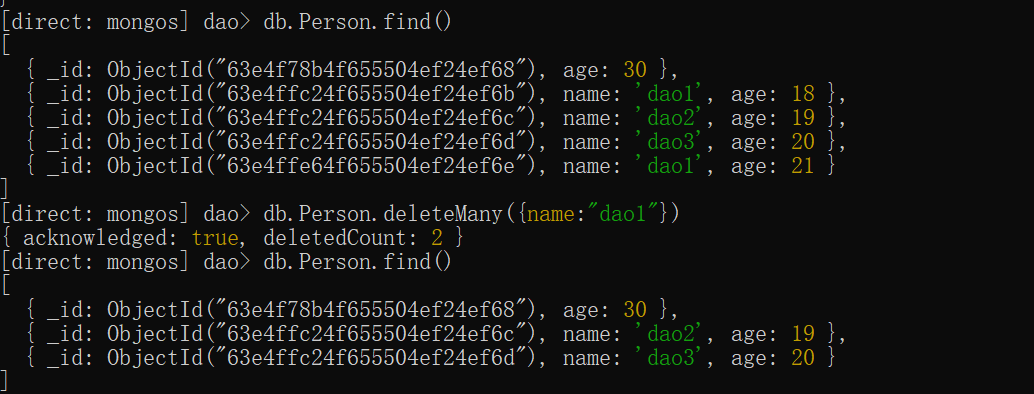
##### deleteOne()

删除单个文档。



##### deleteMany()

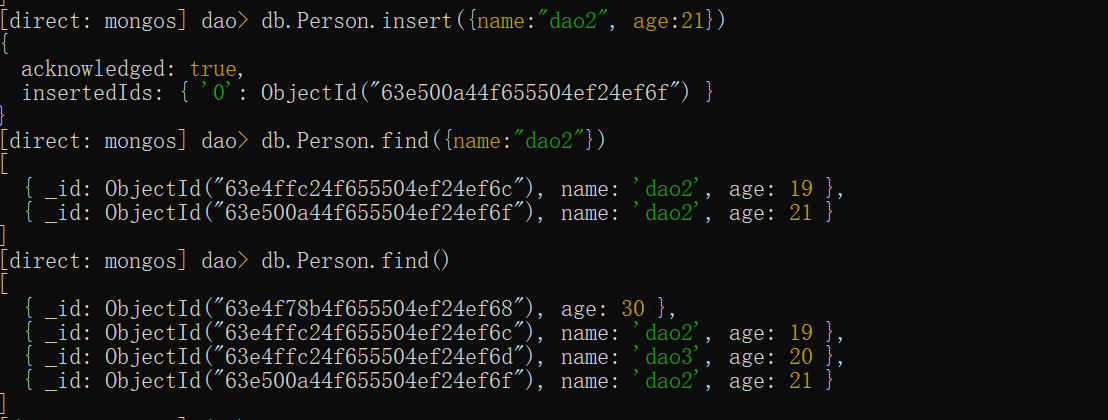
删除多个文档



#### 查询

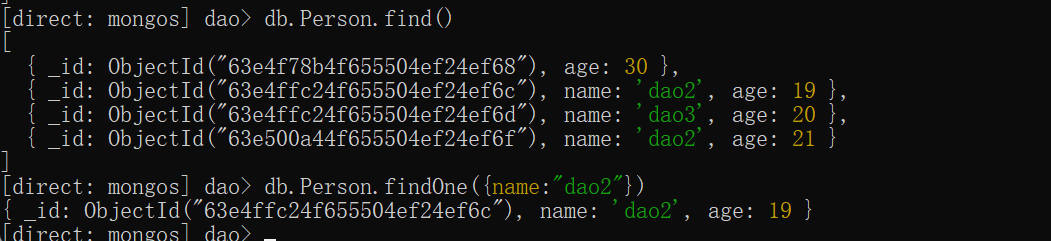
##### find()

可以查询出文档里的所有内容



##### findOne()

查询单个文档。



#### 条件查询

##### 比较操作符

$eq: 表示查询出与条件值相同的文档；

$ne: 查询出与条件值不相等或不存在的文档；

$gt: 查询出大于条件值的文档；

$gte: 查询出大于或等于条件值的文档；

$lt: 查询出小于条件值的文档

$lte: 查询出小于或等于条件值的文档

$ln: 此操作符使用的条件值必须用数组表示，表示筛选出字段值与数组中任一元素吻合的

$nin: 此操作符使用的条件必须用数组表示，表示筛选出字段值不在数组元素中的文档。

例子：

db.Product.find( {“Wight”:{“$gt”:200}} )

db.Product.find( {“Color”:{$in:[“White”,”Black”]}})

##### 逻辑操作符

$and: 用于连接多个查询条件，表示查询的文档必须符合所有条件；

$nor: 用于连接多个查询条件，表示查询的文档必须不符合所有条件；

$or: 用于连接多个查询条件，表示查询的文档只满足其中一个条件即可；

$not: 只可用于一个查询条件，表示查询的文档必须不符合该条件。当字段不存在时也符合条件。

例子：

db.Product.find( { $and:[{“Color”:{$eq: “Red”}}, {“Weight”:{$gte:200}}] } )

db.Product.find( {“SysNo”:{$not: {$eq:2971}} } )

##### 元素操作符

$exists: 可依据该字段是否存在来筛选数据。

$type: 可依据该字段的类型来筛选数据。

例子：  
 db.Product.find({“Color”:{$exists: true}})

db.Product.find( {“Color”: {$type: “array”}} )

##### 评估操作符

$expr: 用来对文档中的两个字段进行比较，进而查询出符合条件的文档；

$mod: 取余数的条件操作符，表示查询出符合余数条件的文档。

例子：

db.Sales.find( {$expr: { $lt: [“$totalAmount”, “$OriginalAmount” ] }} ): 查询字段“totalAmount”的值小于字段“OriginalAmount”的值的文档；

db.Product.find( {“Weight”: {$mod: [5,0]} } ): 查询集合中字段“Weight”的值除以5余数等于0的文档。

##### 正则表达式

格式： {<field>: {$regex:/pattern/, $options: ‘<options>’}}

options是可选参数，有4个值：

“i”表示忽略大小写差异；

“x”表示忽略空格；

“m”表示在数据为多行时，任何一行符合条件即可；

“s”表示会将多行数据视为一行。

例子：  
 db.Product.find( {“Name”: {$regex:”rose”, “$options”:”i”}} )

#### 内嵌文档查询

例子： db.Carts.find( {“product.ProductName”: “Note3”, “product.SysNo”:7329} )

##### 查询优化诊断

在查询语句后加入explain()可以诊断查询效率。

db.Product.find( {“Weight”:{$lt:200}} ).explain()

返回字段说明：  
 shards.shardName: 分片名称

shards.connectionString: 此分片上的服务器信息

shards.serverInfo: 服务器信息

shards.plannerVersion: 查询计划版本

shards.namespace: “数据库”.”集合”

shards.indexFilterSet: 如果显示false，则表示没有对查询语句指定使用索引查询

shards.parseQuery: 显示查询语句的条件

shards.winningPlan: 显示是否通过索引显示文档和索引信息

shards.winningPlan.stage: 在执行查询时会使用哪些操作方式查询文档。

COLLSCAN: 查询字段并没有建立索引，采用全表扫描的方式找出文档；

FETCH: 查询的字段建立了索引，所以查询时会通过索引的位置找到对应的文档；

IXSCAN: 使用索引来找出对应的文档；

TEXT: 使用了全文索引；

COUNTSCAN: 计算总数时没有使用索引；

COUNT\_SCAN: 计算总数时会使用索引；

SUBPLA: 使用“$or”操作符查询时没有使用索引；

SORT: 查询时使用了排序，但没有使用索引；

SKIP: 查询时跳过了几个文档；

IDHACK: 查询时针对特定的“\_id”进行查询；

LIMIT: 限制查询结果的数量；

PROJECTION: 查询时限制显示特定字段；

#### 原子性操作

原子性操作是指在保存文档时只有两种情况：一种是文档全部被保存，另一种则是文档全部被回滚。当一个文档正在进行写操作时，其他对于此文档的操作是不可以进行的。

MongoDB通过锁机制来避免并发操作：

* 当一个使用者对文档进行读操作时，会读取一个“读”锁。此时，其他使用者可以读此文档，但不可以对此文档进行写操作；
* 当一个使用者对文档进行写操作时，会取得一个“写”锁。此时，其他使用者不可以对此文档进行读/写操作。

#### 常用修改操作

##### $set

修改文档。

db.Product.update(

{“SysNo”:2971, “Weight”:{&gt:200}},

{$set:{Weight:300}}

)

修改weight字段为300

##### $unset

$unset操作符可以删除指定字段。

db.Product.update(

{“SysNo”:2971},

{“unset”:{Weight:””}}

)

删除Weight字段

##### $inc

$inc操作符可以对指定字段的值进行加减运算。

db.Product.update(

{“SysNo”:2971, “Weight”:{$gt:200}},

{$inc: {Weight: -100}}

)

将weight的值减少100。

##### $push

$push操作符可以在指定文档的数组中插入一个值。

db.Product.update(

{\_id: ObjectId(“5ad9378a”)},

{$push: {“Color”:”Orange”}}

)

在指定文档的“Color”数组中加入“Orange”字符串。

##### $pull

$push操作符可以在指定文档的数组中删除一个值。

db.Product.update(

{\_id: ObjectId(“5ad9378a”)},

{$pull: {“Color”:”Orange”}}

)

在Color数组中删除Orange字符串

##### $pop

$pop操作符可以删除指定文档数组中的第1个或最后一个值。$pop可以设定为两个值。“1”表示删除数组中的最后一个值；“-1”表示删除数组中的第一个值。

db.Product.update(

{\_id:ObjectId(“5ad9378a380”)},

{$pop: {“Color”:1}}

)

删除Color数组中的最后一个值。

##### $rename

$rename可以更改文档中的字段名

db.Product.update(

{\_id: ObjectId(“5ad9378a”)},

{$rename: {“Color”: “ProductColor”}}

)

### 索引

如果在没有索引的情况下查找数据，则MongoDB会扫描所有文档，再将符合查询条件的文档筛选出来。如果集合中存在许多文档，为了避免扫描全部文档的这种情况，通常会创建索引。

MongoDB的索引分为单字段索引、复合索引、地理空间索引。

#### 单字段索引

##### 创建索引

db.collection.createIndex(<keys>,<options>)

keys: 此参数用于设定索引的字段和排序规则，1表示升序，-1表示降序。“field”:1表示field字段建索引且升序排序。

options：可选参数。

* background: 创建索引是否会在后台进行。默认为false；
* unique: 设定此索引是否为唯一索引，默认为false；
* name: 自定义创建的索引名称；
* partialFilterExpression: 索引只用于筛选条件匹配的文档；
* sparse: 索引只用于指定字段的文档；
* expireAfterSeconds: 此参数用在TTL索引中，可以控制文档保留在集合的时间；
* storageEngine: 指定索引使用的存储引擎。

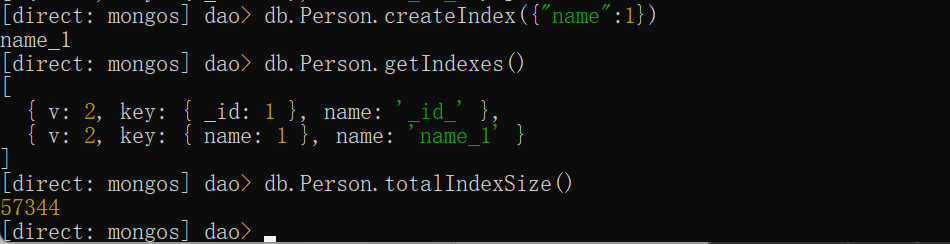
例子：

db.Person.createIndex({“Weight”:1})

##### 查询索引

db.<collection>.getIndexes()：可查看所有的索引。

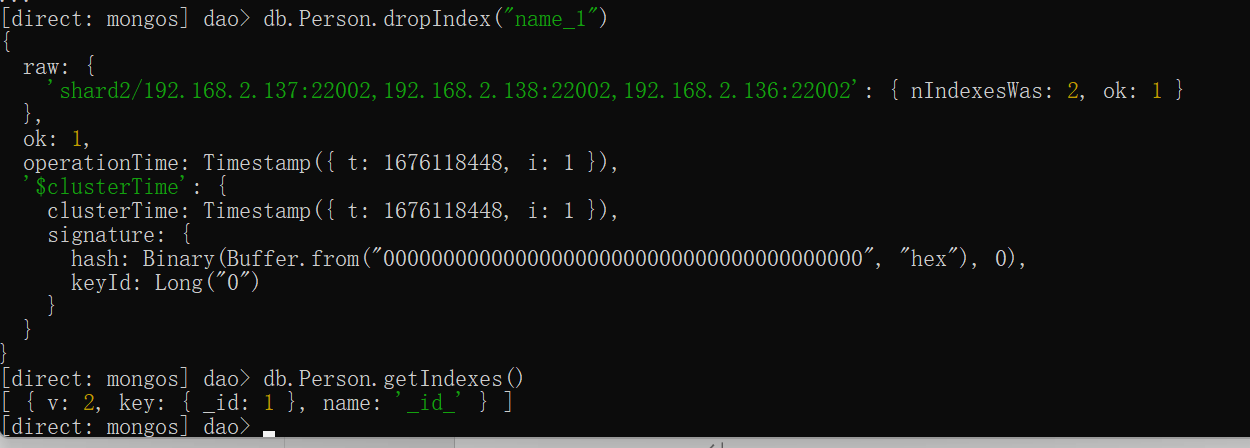
db.<collection>.totalIndexSize()：可查看集合中所有索引的大小。



##### 删除索引

db.<collection>.dropIndex(<index>):删除已经建立的某个索引。

db.<collection>.dropIndexes(): 删除除了\_id字段以外的所有索引。



#### 复合索引

复合索引是将多个字段组合成一个索引，这个索引整体上按第1个字段进行排序，再用第2个字段进行排序，依次类推。

例子： db.Product.createIndex( {“Weight”:-1, “SysNo”:1} )

#### TTL索引

该索引可以针对文档中的时间类型的字段设定一个时间值。如果此字段的时间超过设定的时间值，则该文档就会被删除。

* TTL索引不可以是复合索引；
* 如果创建TTL索引的字段不是时间类型，则此文档将不会被删除；
* TTL索引每60s运行一次，移出过期的文档；
* 如果时间类型的字段已经被设定为其他索引，则无法通过TTL索引来删除此文档。

例子： db.createIndex( {“CreateDate”:-1}, {expireAfterSeconds: 60\*60\*24\*7} )

文档在“CreateDate”字段时间的7天后删除，并以“CreateOrder” 字段进行降序排序。

# 高可用

## 副本集

副本集是数据高可用的必要配置，它主要包含三种角色：  
 主节点与副节点：

主节点与副节点均属于数据节点，其中保存了完整的数据；

“数据写入的需求”由主节点接收并处理，然后通过同步机制同步到所有的副节点上，使得数据得以复制保存；

节点间通过“心跳”机制进行沟通，确保所有节点都能正常运行。

## 仲裁节点

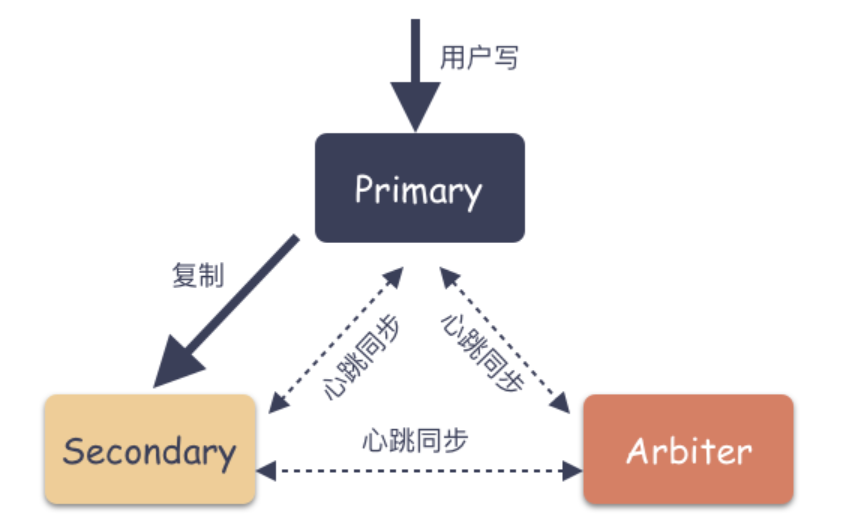
它是副本集中的一个投票者，负责在主节点出现异常需要切换时，发挥副本集(投票)的角色。

副本集中的每个副本都可以拥有投票权，但仲裁节点的不同之处在于：仲裁节点只具备投票权，不会保存数据，对服务器的资源要求较低；

这种设计可以在硬件资源有限的情况下，我们仍可以实现灾难还原。

## 副本集的基本架构

副本集的最低要求为：1个主节点+1个副节点+1个仲裁节点。3个具有投票权的节点即可进行有效的投票。典型的副本集架构有：“一主二副”以及“一主一副一仲裁”。



## 心跳机制

心跳是成员间确认彼此是否还存活的依据。心跳的频率可以在配置副本集时通过settings.heartbeatIntervalMillis参数来决定，默认是2s。

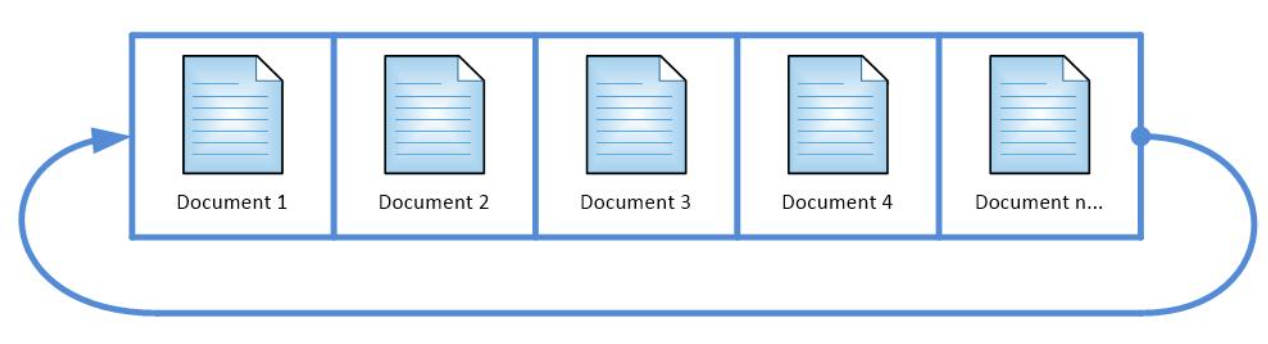
## 选举和投票机制

在主节点出现故障时，会重新进行选举，只有得到大多数节点的投票的节点才能成为主节点。一般是奇数节点(和偶数节点的效果一致但能节约一台资源)。

## Oplog同步

MongoDB Oplog是MongoDB 节点之间在复制建立期间和建立完成之后的复制介质，Primary节点中的所有写入操作都会被记录到MongoDB的Oplog中，然后从库会来主库一直拉取Oplog并应用到自己的数据库中。

MongoDB Oplog是local库下的一个固定集合，它是固定大小的，循环使用。



MongoDB节点之间的同步分为两种：

* initial sync: 这个过程发生在副本集中创建一个新的数据库或其中某个节点刚从宕机中恢复，或者向副本集中添加新的成员时。
* replication: 在初始化后这个操作会一直持续的进行，以保持各个secondary节点之间的数据同步。

**回滚：**在对MongoDB副本集进行写操作时，若主节点在接收写操作后尚未将写操作同步至副本节点便发生宕机或网络断开，则新的主节点仍有新的写操作，但当旧的主节点恢复后，便会产生数据不一致的状况，这时可以通过回滚机制，将旧的主节点数据恢复至最后一致的状态。

若要避免发生回滚的状态，可使用写入策略确保数据在写入多个节点后才回传成功，这样可以避免“副节点尚未从主节点同步数据”就发生主节点宕机或网络断开而造成数据丢失。

## 写入策略

MongoDB的写入策略可以在写入数据时指定，通过客户端的Write Concern配置来做到这一点，它包含以下字段：

{w: <值>, J: <布尔值>, wtimeout: <数字>}

w字段：表示在反馈“操作完成”信息给应用程序前必须同步到N个副本节点上

* 1: 表示只需写入主节点即可返回
* 0：表示不需要等待任何节点写入完成的响应即可返回；
* 大于1的某数：表示只要写入达到这个数目的节点数成功，即可返回。需要注意集群节点数量需大于次数，否则会报错。

majority: 表示，只有当写入超过“大多数”的可投票节点时才会反馈信息给应用端；如果节点中有仲裁节点，则可能出现即使所有数据节点都写入完成，但仍无法满足“超过大多数”节点的条件而出现异常。

j字段：表示“写”操作是否需要记录到日志文件中。若设置为记录到日志中，则在服务发生意外时可通过日志恢复数据；

wtimeout字段：表示等待时间的限制，单位为ms。若超过此限制则写入失败，只有当“w”字段的值大于1时，这个属性才会生效。

示例： db.Product.insertOne( {name: “Jason”} , {writeConcern:{w:0,j:true}})

## 读取策略

类似于写入策略，MongoDB对数据读取也提供了弹性的配置，分别是Read Concern和Read Preference。

**Read Concern**

Read Concern的作用是让客户端指定什么样的数据是可以读取的。

使用例子：db.Person.find().readConcern(<Level>)。

Level的取值：

* local: 从主节点读取时的默认值；
* available: 从副节点读取时的默认值；
* majority: 数据操作必须更新至大多数节点后才能被读取；
* linearizable：只允许在主节点读取。

**Read Preference**

通过”Read Preference”配置可以让客户端驱动知道要从哪个节点去读取数据，此配置可实现读写分离、就近读取数据的负载均衡效果。

格式：mongo.setReadPref( <mode>, <tagSet>)

mode的取值：

* primary: 默认值，从主节点读取；
* primaryPreferred: 优先从主节点读取。若主节点无法读取，则从副节点读取；
* secondary: 只从副节点读取；
* secondaryPreferred: 优先从副节点读取。若无法从副节点读取，则从主节点读取。
* nearest: 根据网络情况从最近的节点读取

## 分片集

分片机制可以将数据进行水平扩展，将数据分布到不同节点上。

### 分片键

MongoDB数据分片并不是完全自动，在分片前还需要通过配置让MongoDB知道数据分配的规则，然后再根据这些规则将数据以数据块(chunk)的形式分布存储到各个分片上。这个规则就是片键。

对数据库启用分片：sh.enableSharding(<database>)

设置集合的分片：sh.shardCollection(<namespace>,<key>,<unique>,<options>)

namespace: 要进行的分片集合。指定对应的数据库与集合(database.collection)即可。

key: 用作分片依据的片键，让MongoDB知道如何在分片间分发文档。此处指定作为片键的字段及排序依据：1为正向，-1为反向，hashed为散列片键。可配置复合字段作为片键。

unique: 可选参数。若为true，表示强制执行索引的唯一约束；

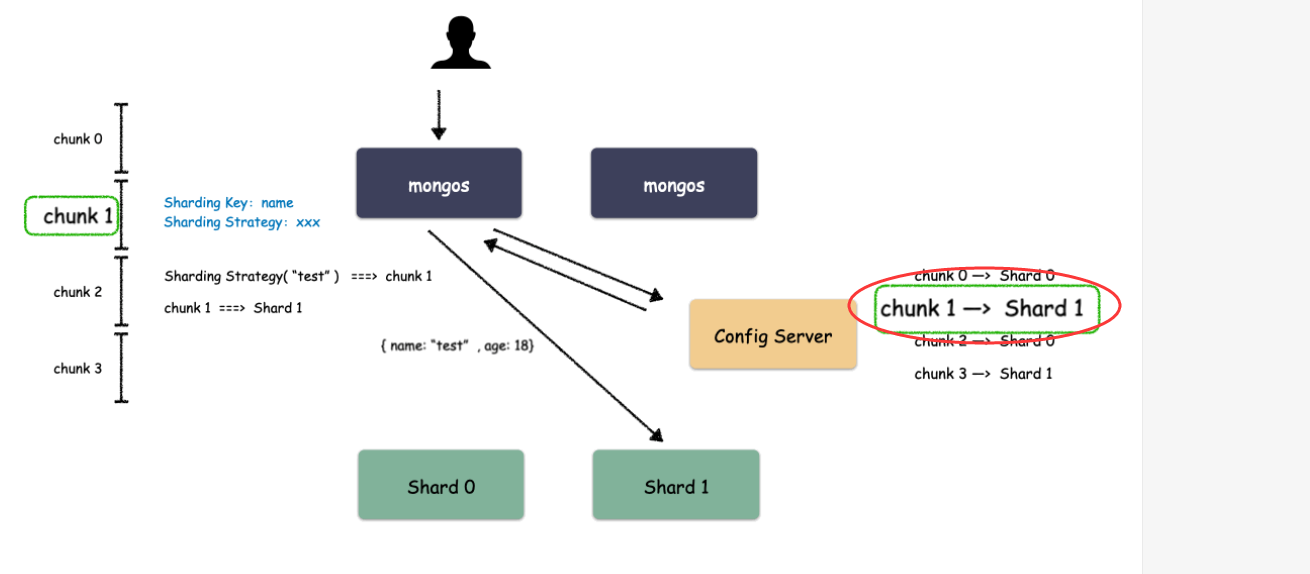
options: 可选参数，包含numInitialChunks和collation。

* numlnitialChunks是一个整数值，可设置空集合初始化数据库(chunk)的数量，但每个分片必须小于8192个chunk。
* collation用来指定字符串的排序规则

使用例子：sh.shardCollection(“E-commerce.Carts”,{CustomerSysNo:1}, false, {numlnitiaChunks:5, collation: {locale: “sample”}})

### 平衡器

集合设置分片后，数据是以Chunk为单位分散在各个节点上的，当写入数据时，会先根据分片策略计算属于哪个数据块(Chunk),然后写入到Chunk对应的节点上。



平衡器会周期性地检查分片是否均匀分布，这个检查考虑的是每个分片chunk的数量，并非文档的数据大小和数量。如果分片分布不均匀，则会发生搬迁，一般发生在chunk数量最多和最少的分片之间。

在搬迁的过程中，chunk中的数据会被复制到目的端的分片上，此时应用端的读写请求仍会被送到来源端分片的chunk上，知道chunk搬迁完且元数据也完成更新，mongos才会重新指向目的端分片。