**目录**

[1. 概述 2](#_Toc32242670)

[2. 搭建分片存储集群——初级 3](#_Toc32242671)

[3. 搭建分片集群——高级 7](#_Toc32242672)

[3.1整体准备 7](#_Toc32242673)

[3.2启动并配置2个分片副本集 8](#_Toc32242674)

[3.2.1配置第1个分片副本集 8](#_Toc32242675)

[3.2.2配置第2个分片副本集 9](#_Toc32242676)

[3.3启动并配置配置结点 9](#_Toc32242677)

[3.4启动并配置mongos结点 10](#_Toc32242678)

[3.5指定片键、测试分片效果 10](#_Toc32242679)

[3.6动态添加一个分片副本集 10](#_Toc32242680)

[3.6.1准备要添加的分片副本集 11](#_Toc32242681)

[3.6.2添加分片到集群中 11](#_Toc32242682)

[3.6.3添加数据观察新分片上的数据 11](#_Toc32242683)

[ 移动数据块 13](#_Toc32242684)

[3.7关于mongod、mongos结点配置建议 13](#_Toc32242685)

[3.7.1mongos 13](#_Toc32242686)

[3.7.2mongod 13](#_Toc32242687)

[ 作为数据存储shards结点 13](#_Toc32242688)

[ 作为配置服务config server结点 13](#_Toc32242689)

[4. 聊几个Problem 14](#_Toc32242690)

[4.1如何优雅地删除分片结点 14](#_Toc32242691)

[4.2关于数据均衡 15](#_Toc32242692)

[4.2.1均衡器——自动均衡 15](#_Toc32242693)

[ 启用关闭均衡器的一些场景 15](#_Toc32242694)

[ 设置/删除均衡器的时间窗口 15](#_Toc32242695)

[ 禁用和开启均衡器 16](#_Toc32242696)

[ 均衡规则 16](#_Toc32242697)

[4.2.2手动均衡 17](#_Toc32242698)

[ 查看分片情况 17](#_Toc32242699)

[ 查看chunk\_id情况 18](#_Toc32242700)

[4.3配置信息查看 20](#_Toc32242701)

[4.4关于均衡器 21](#_Toc32242702)

[4.4.1关闭均衡器、确认分片、指定片键 21](#_Toc32242703)

[4.4.2插入大量数据、查看分片情况 22](#_Toc32242704)

[4.4.3如果在添加结点、删除结点的时候关闭了均衡器会出现什么情况呢？ 24](#_Toc32242705)

[ 添加结点 24](#_Toc32242706)

[ 删除结点 24](#_Toc32242707)

[ 建议 24](#_Toc32242708)

[4.5关于片键 24](#_Toc32242709)

[4.5.1片键不能修改（暂未测） 24](#_Toc32242710)

[4.5.2片键种类 24](#_Toc32242711)

[4.5.3片键选取建议 25](#_Toc32242712)

[4.6正在更新的文档所属的块正在被迁移 26](#_Toc32242713)

[4.7删除moveChunk目录下的老数据安全吗？ 26](#_Toc32242714)

[4.8mongs处理连接 26](#_Toc32242715)

[4.9mongs的配置服务 27](#_Toc32242716)

[4.10mongd启动选项 27](#_Toc32242717)

[4.11未分片的数据 27](#_Toc32242718)

[4.12先插入数据后进行分片 27](#_Toc32242719)

[4.13分片集群中的读写分离 27](#_Toc32242720)

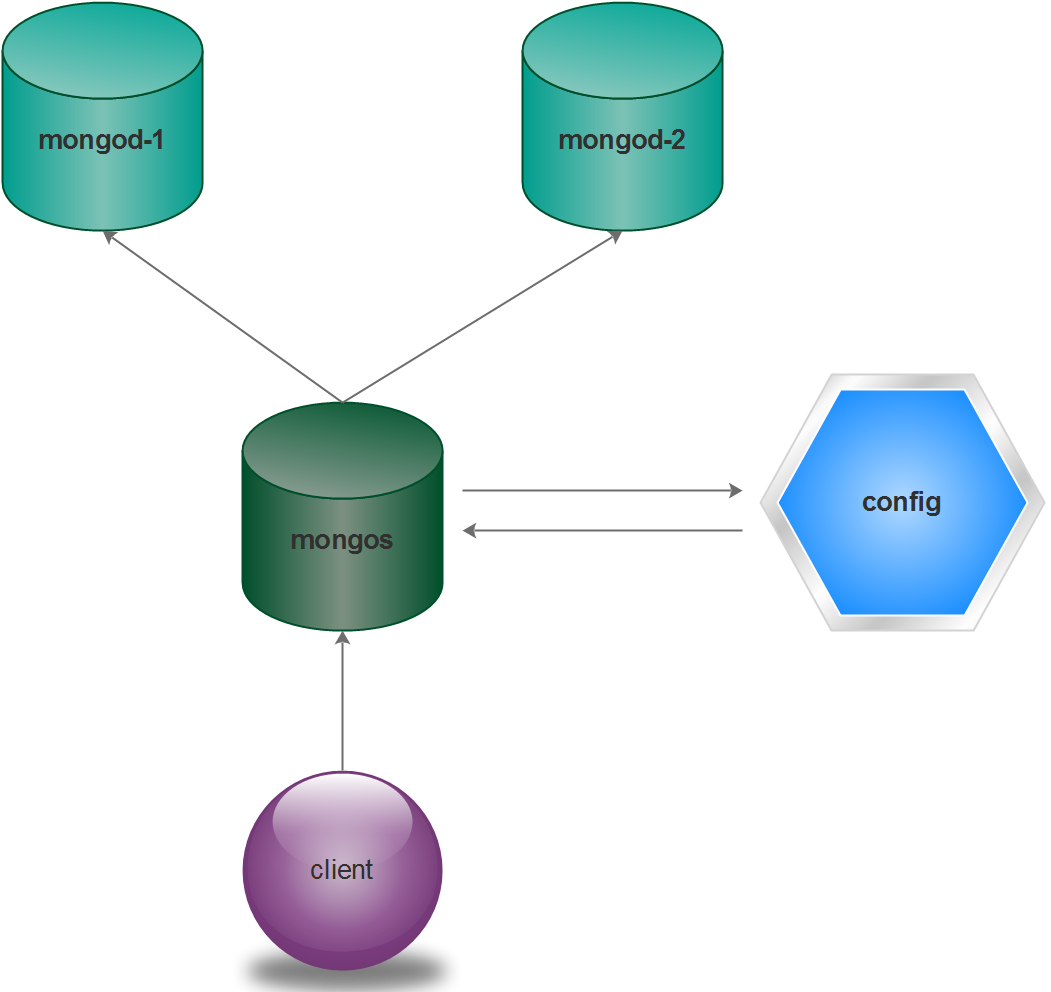
[5. 参考资料 29](#_Toc32242721)

**jar cvfm MyWordCount.jar manifest.data \*.class**

# 概述

分片方式存储从一定程度来讲解决了单击存储磁盘容量受限的问题。

在MongoDB中分片方式中，最基本的一个分片集群如下设计：



**client**：是我们访问MongoDB的客户端，无论你的MongoDB集群如何设计，客户端是毫不关心的。

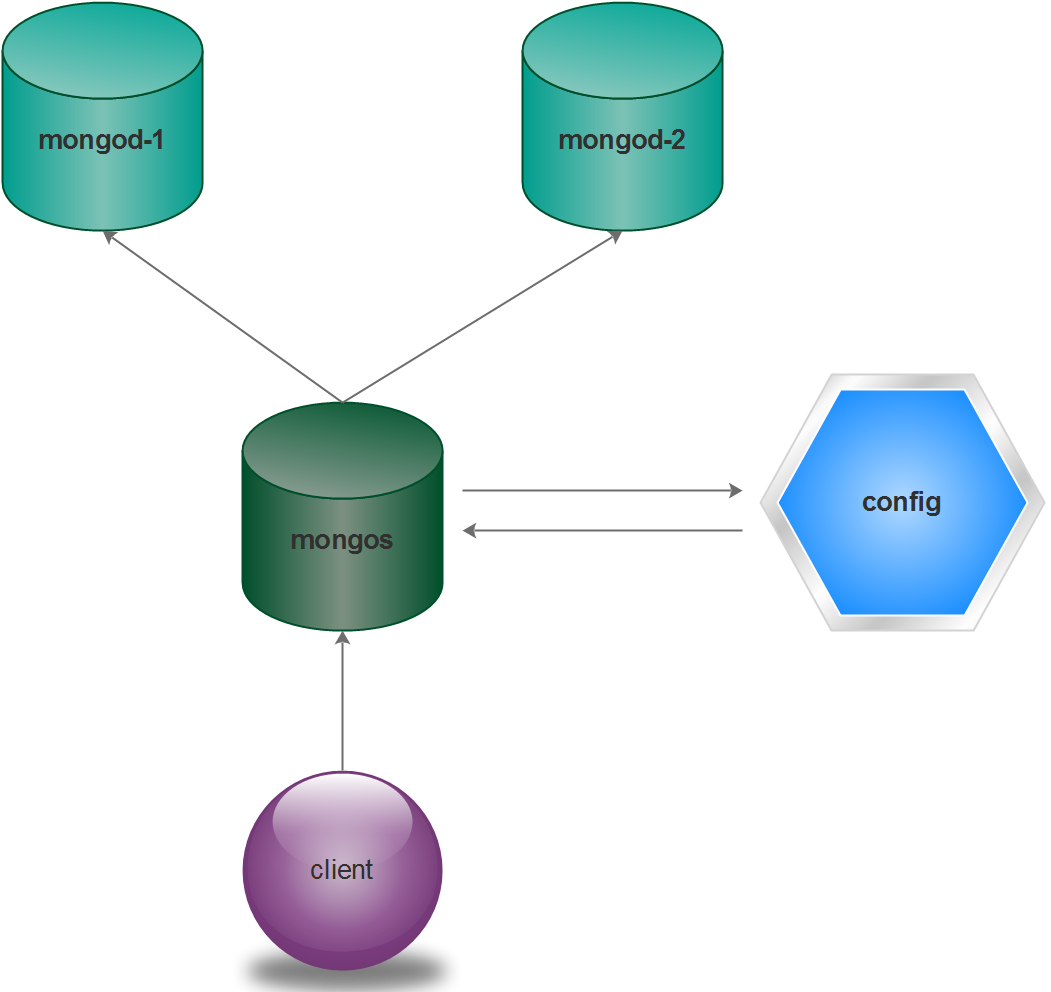
**config**：分片集群的配置服务器，保存集群的配置信息。

**mongos**：首先我们要了解”片键“的概念，也就是说拆分集合的依据是什么？按照什么键值进行拆分集合.... 好了，mongos就是一个路由服务器，它会根据管理员设置的“片键”将数据分摊到自己管理的mongod集群，数据和片的对应关系以及相应的配置信息保存在"config服务器"上。（在mongos启动的时候，规定config的数量是1个或者3个就这样规定，爱用不用。）

**mongod**：一个普通的MongoDB数据库实例，如果不分片的话，直接连接mongod即可。

# 搭建分片存储集群——初级

我们搭建的集群模型就按照如下图：



我们这次依然在一个机器使用多个端口来搭建这个集群，这里我们需要4个MongoDB结点，分别启动：config、mongos、mongod-1、mongod-2。4个目录分别是：J:\MongoDB 2.6 Standard-config、J:\MongoDB 2.6 Standard-mongos、J:\MongoDB 2.6 Standard-mongod-1、J:\MongoDB 2.6 Standard-mongod-2。

2.1启动4个结点

我们依次config、mongos、mongod-1、mongod-2，依次使用10000、20000、30000、40000端口：

**mongod.exe --dbpath="J:\MongoDB 2.6 Standard-config\data" --port=10000**

**mongos.exe --port=20000 --configdb=127.0.0.1:10000**

**mongod.exe --dbpath="J:\MongoDB 2.6 Standard-mongod-1\data" --port=30000**

**mongod.exe --dbpath="J:\MongoDB 2.6 Standard-mongod-2\data" --port=40000**

2.2服务配置

2.2.1添加mongod分片到mongos中

登录到mongos上，我们将mongod的信息添加到mongos中，如下：

**mongo 127.0.0.1:20000**

**use admin;**

**db.runCommand({"addshard":"127.0.0.1:30000", allowLocal:true});**

**db.runCommand({"addshard":"127.0.0.1:40000", allowLocal:true});**

我们将mongod-1、mongod-2都添加到其中，然后查看一下：

**mongos> show dbs;**

**admin (empty)**

**config 0.063GB**

**mongos> use config;**

**switched to db config**

**mongos> show tables;**

**changelog**

**chunks**

**databases**

**lockpings**

**locks**

**mongos**

**settings**

**shards**

**system.indexes**

**version**

**mongos> db.shards.find();**

**{ "\_id" : "shard0000", "host" : "127.0.0.1:30000" }**

**{ "\_id" : "shard0001", "host" : "127.0.0.1:40000" }**

**mongos>**

2.2.2配置片键

2.2.1已经配置了分片，但是mongos还不知道如何将数据分布到哪一个分片中，下面我们配置片键：

（1）开启数据库的分片功能，enablesharding()，分片的最小单位数据库；

（2）指定集合中分片的键，这里我们为school数据进行分片，student.name作为片键；

**mongos> use admin;**

**switched to db admin**

**mongos> db.runCommand({"enablesharding":"school"});**

**{ "ok" : 1 }**

**mongos>db.runCommand({"shardcollection":"school.student", "key":{"name":1}});**

**{ "collectionsharded" : "school.student", "ok" : 1 }**

**mongos>**

**注意**如果库中已经有数据，那么再分片，会出现错误信息"please create an index that starts with the shard key before sharding."，所以可以在在没有数据的情况下进行分片。如果在**已存在的集合中进行分片**，需保证片键上有索引，如果没有，需要先创建。

而且做了片键的字段会被自动创建索引，我们将school.student中的name字段作为片键，我们查看指定完片键之后，查看索引情况如下：

**mongos> use school;**

**switched to db school**

**mongos> db.student.getIndexes();**

**[**

**{**

**"v" : 1,**

**"key" : {**

**"\_id" : 1**

**},**

**"name" : "\_id\_",**

**"ns" : "school.student"**

**},**

**{**

**"v" : 1,**

**"key" : {**

**"name" : 1**

**},**

**"name" : "name\_1",**

**"ns" : "school.student"**

**}**

**]**

2.3查看效果

我们向school.student表中插入若干条数据，然后查看分片效果：

**use school;**

**for(var i=0; i<500000; i++){ db.student.insert({"name":"zhangsan"+i}) }**

**db.printShardingStatus();**

**mongos> db.printShardingStatus();**

**--- Sharding Status ---**

**sharding version: {**

**"\_id" : 1,**

**"version" : 4,**

**"minCompatibleVersion" : 4,**

**"currentVersion" : 5,**

**"clusterId" : ObjectId("5760ffdde976909f286f715f")**

**}**

**shards: # 当前集群中分片的数量**

**{ "\_id" : "shard0000", "host" : "127.0.0.1:30000" }**

**{ "\_id" : "shard0001", "host" : "127.0.0.1:40000" }**

**databases:**

**{ "\_id" : "admin", "partitioned" : false, "primary" : "config" }**

**{ "\_id" : "school", "partitioned" : true, "primary" : "shard0000" } # partitioned表示db是否被分区**

**school.student**

**shard key: { "name" : 1 }**

**chunks: # 数据被切分为4段，如下[无穷小，zhangsan0)、[无穷小，zhangsan0)、[zhangsan0，zhangsan257283)、**

**# [zhangsan257283，zhangsan9999)、[zhangsan9999，无穷大)**

**shard0001 2 # shard0001分片存储了2个段**

**shard0000 2 # shard0000分片存储了2个段**

**{ "name" : { "$minKey" : 1 } } -->> { "name" : "zhangsan0" } on : shard0001 Timestamp(2, 0)**

**{ "name" : "zhangsan0" } -->> { "name" : "zhangsan257283" } on : shard0000 Timestamp(3, 2)**

**{ "name" : "zhangsan257283" } -->> { "name" : "zhangsan9999" } on : shard0000 Timestamp(3, 3)**

**{ "name" : "zhangsan9999" } -->> { "name" : { "$maxKey" : 1 } } on : shard0001 Timestamp(3, 0)**

**{ "\_id" : "class\_info", "partitioned" : false, "primary" : "shard0001" }**

**mongos>**

2.4添加结点

我们启动一个新的mongod，然后在先用的mongos上添加该节点：

**use admin;**

**db.runCommand({"addshard":"127.0.0.1:50000", allowLocal:true});**

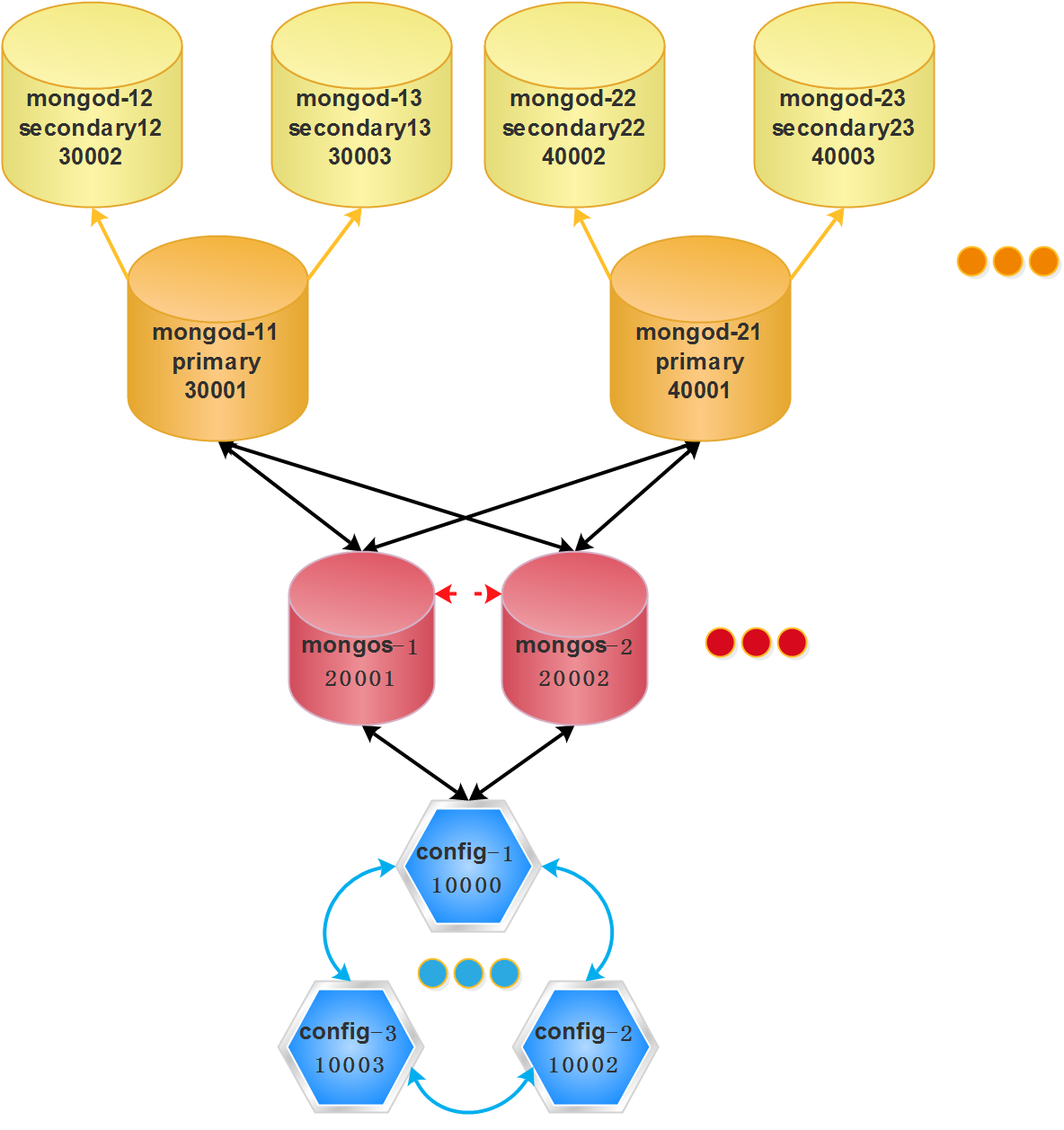
默认情况下，启动的集群会开启均衡器，我们可以如下查看集群中的均衡器状态：

**mongos> sh.getBalancerState();**

**true**

# 搭建分片集群——高级

在前面我们搭建了一个简单的分片集群，当然这个只是基本的。在前面几节，我们接触了副本集的应用。在此为了搭建一个稳定、安全的集群，我们将如下设计：



## 3.1整体准备

（1）根据上图，为了便于识别，我们需要10个结点；

（2）mongod使用副本集，在此我们不设置仲裁结点，要设置仲裁结点也行，整体配置方法不变；

（3）分片副本集、mongos、config都可以有多个；

我们准备的结点如下：

mongod副本集分片1

G:\MongoDB 2.6 Standard-mongod-30001、使用30001端口

G:\MongoDB 2.6 Standard-mongod-30002、使用30002端口

G:\MongoDB 2.6 Standard-mongod-30003、使用30003端口

mongod副本集分片2

G:\MongoDB 2.6 Standard-mongod-40001、使用40001端口

G:\MongoDB 2.6 Standard-mongod-40002、使用40002端口

G:\MongoDB 2.6 Standard-mongod-40003、使用40003端口

mongod配置结点

G:\MongoDB 2.6 Standard-config-10000、使用10000端口

G:\MongoDB 2.6 Standard-config-10001、使用10001端口

G:\MongoDB 2.6 Standard-config-10002、使用10002端口

**！！！注意：config server配置的在不同版本有不同。在2.x/3.0版本中，即使你配置了3个config server，一旦有一个config server挂掉了，那么此时整个集群的元数据将变为只读状态。在3.2=+的版本中，我们可以将3个config server配置为一个replset来解决这个问题。**

**这一点非常坑，也就是在2.x/3.0版本中，多个config server仅仅保证了config不稳定的数据完整性，至于业务层面还是无法做到稳定的。在3.2=+中可以搞定，可以将config server设置为1个replset来解决这个问题，参考这里[https://docs.mongodb.com/manual/core/sharded-cluster-config-servers/#csrs](https://docs.mongodb.com/manual/core/sharded-cluster-config-servers/" \l "csrs)将config server配置为replset模式。**

mongos结点

G:\MongoDB 2.6 Standard-mongos-20000、使用20000端口

G:\MongoDB 2.6 Standard-mongos-20002、使用20001端口

## 3.2启动并配置2个分片副本集

一些说明：

（1）--shardsvr：声明这个服务是一个分片服务；

（2）没有配置oplogsize、logpath等选项，这些在线上配置时需要添加；

### 3.2.1配置第1个分片副本集

**"G:\MongoDB 2.6 Standard-mongod-30001\bin\mongod" --port=30001 --dbpath="G:\MongoDB 2.6 Standard-mongod-30001\data" --replSet lt\_replset\_shard\_01 --shardsvr**

**"G:\MongoDB 2.6 Standard-mongod-30002\bin\mongod" --port=30002 --dbpath="G:\MongoDB 2.6 Standard-mongod-30002\data" --replSet lt\_replset\_shard\_01 --shardsvr**

**"G:\MongoDB 2.6 Standard-mongod-30003\bin\mongod" --port=30003 --dbpath="G:\MongoDB 2.6 Standard-mongod-30003\data" --replSet lt\_replset\_shard\_01 --shardsvr**

**mongo 127.0.0.1:30001**

**use admin;**

**config = {\_id:"lt\_replset\_shard\_01", members:[{\_id:0, host:"127.0.0.1:30001"}, {\_id:1, host:"127.0.0.1:30002"}, {\_id:2, host:"127.0.0.1:30003"}]};**

**rs.initiate(config); #执行之后等待1-2分钟，再查看，各个节点才会就绪为SECONDARN、PRIMARY类型**

**rs.status(); #查看显示3个结点中1个Primary、2个Secondary，说明副本集配置成功**

### 3.2.2配置第2个分片副本集

**"G:\MongoDB 2.6 Standard-mongod-40001\bin\mongod" --port=40001 --dbpath="G:\MongoDB 2.6 Standard-mongod-40001\data" --replSet lt\_replset\_shard\_02 --shardsvr**

**"G:\MongoDB 2.6 Standard-mongod-40002\bin\mongod" --port=40002 --dbpath="G:\MongoDB 2.6 Standard-mongod-40002\data" --replSet lt\_replset\_shard\_02 --shardsvr**

**"G:\MongoDB 2.6 Standard-mongod-40003\bin\mongod" --port=40003 --dbpath="G:\MongoDB 2.6 Standard-mongod-40003\data" --replSet lt\_replset\_shard\_02 --shardsvr**

**mongo 127.0.0.1:40001**

**use admin;**

**config = {\_id:"lt\_replset\_shard\_02", members:[{\_id:0, host:"127.0.0.1:40001"}, {\_id:1, host:"127.0.0.1:40002"}, {\_id:2, host:"127.0.0.1:40003"}]};**

**rs.initiate(config); #执行之后等待1-2分钟，再查看，各个节点才会就绪为SECONDARN、PRIMARY类型**

**rs.status(); #查看显示3个结点中1个Primary、2个Secondary，说明副本集配置成功**

## 3.3启动并配置配置结点

按照如下命令依次启动即可：

**"G:\MongoDB 2.6 Standard-config-10000\bin\mongod.exe" --dbpath="G:\MongoDB 2.6 Standard-config-10000\data" --port=10000 --configsvr**

**"G:\MongoDB 2.6 Standard-config-10001\bin\mongod.exe" --dbpath="G:\MongoDB 2.6 Standard-config-10001\data" --port=10001 --configsvr**

**"G:\MongoDB 2.6 Standard-config-10002\bin\mongod.exe" --dbpath="G:\MongoDB 2.6 Standard-config-10002\data" --port=10002 --configsvr**

## 3.4启动并配置mongos结点

多个mongos的工作依靠--configdb来同步（注意，--configdb的内容，各个ip:port之间不能有任何空格，比如：--configdb=”127.0.0.1:11111, 127.0.0.1:11112”这个就会出错）

**"G:\MongoDB 2.6 Standard-mongos-20000\bin\mongos.exe" --port=20000 --configdb="127.0.0.1:10000,127.0.0.1:10001,127.0.0.1:10002"**

**mongo 127.0.0.1:20000**

**use admin;**

**db.runCommand({ addshard : "lt\_replset\_shard\_01/127.0.0.1:30001,127.0.0.1:30002,127.0.0.1:30003"});**

**db.runCommand({ addshard : "lt\_replset\_shard\_02/127.0.0.1:40001,127.0.0.1:40002,127.0.0.1:40003"});**

**db.runCommand({ listshards : 1 });**

当我们配置好上面的mongos之后，只要其他的mongos将--configdb配置为与上面相同的地址，新的mongos就拥有了集群相关的信息。

**"G:\MongoDB 2.6 Standard-mongos-20001\bin\mongos.exe" --port=20001 --configdb="127.0.0.1:10000,127.0.0.1:10001,127.0.0.1:10002"**

**mongo 127.0.0.1:20001**

**use admin;**

**db.runCommand({listshards:1}); #我们同样可以看到如上配置的集群信息**

## 3.5指定片键、测试分片效果

我们指定片键，之后添加50w的数据，如下操作：

**use admin;**

**db.runCommand({"enablesharding":"school"});**

**db.runCommand({"shardcollection":"school.student", "key":{"\_id":"hashed"}});**

**use school;**

**for(var i=0; i<500000; i++){ db.student.insert({"name":i}) }**

**sh.status();**

查看之后，我们可以看到数据被分散在各个分片上，我们也可以登录到各个分片上进行查看，也可以在将副本设置为可读（登录到副本结点，执行db.getMongo().setSlaveOk();即可），在分片的副本上查看数据数量。

注意2.4之下的版本不支持hashedkey。

## 3.6动态添加一个分片副本集

本文其他地方已经提到了均衡器的作用，添加结点我们开启均衡器，观察添加后数据块在各个分片上的分布情况：

### 3.6.1准备要添加的分片副本集

**"G:\MongoDB 2.6 Standard-mongod-50001\bin\mongod" --port=50001 --dbpath="G:\MongoDB 2.6 Standard-mongod-50001\data" --replSet lt\_replset\_shard\_03 --shardsvr**

**"G:\MongoDB 2.6 Standard-mongod-50002\bin\mongod" --port=50002 --dbpath="G:\MongoDB 2.6 Standard-mongod-50002\data" --replSet lt\_replset\_shard\_03 --shardsvr**

**"G:\MongoDB 2.6 Standard-mongod-50003\bin\mongod" --port=50003 --dbpath="G:\MongoDB 2.6 Standard-mongod-50003\data" --replSet lt\_replset\_shard\_03 --shardsvr**

**mongo 127.0.0.1:50001**

**use admin;**

**config = {\_id:"lt\_replset\_shard\_03", members:[{\_id:0, host:"127.0.0.1:50001"}, {\_id:1, host:"127.0.0.1:50002"}, {\_id:2, host:"127.0.0.1:50003"}]};**

**rs.initiate(config); #执行之后等待1-2分钟，再查看，各个节点才会就绪为SECONDARN、PRIMARY类型**

**rs.status();**

### 3.6.2添加分片到集群中

登录任意一个mongos都行。

**mongo 127.0.0.1:20001**

**use admin;**

**db.runCommand({ addshard : "lt\_replset\_shard\_03/127.0.0.1:50001,127.0.0.1:50002,127.0.0.1:50003"});**

**db.runCommand({listshards:1})**

### 3.6.3添加数据观察新分片上的数据

添加数据之后，均衡器会根据当前规则是否满足数据块迁移，为我们调整数据块到新的分片中。

**mongos> use school;**

**switched to db school**

**mongos> for(var i=500000; i<600000; i++){ db.student.insert({"name":i}) }**

**mongos> sh.status();**

**--- Sharding Status ---**

**sharding version: {**

**"\_id" : 1,**

**"version" : 4,**

**"minCompatibleVersion" : 4,**

**"currentVersion" : 5,**

**"clusterId" : ObjectId("57635d8d56f81cb8ae31575f")**

**}**

**shards:**

**{ "\_id" : "lt\_replset\_shard\_01", "host" : "lt\_replset\_shard\_01/127.0.0.1:30001,127.0.0.1:30002,127.0.0.1:30003" }**

**{ "\_id" : "lt\_replset\_shard\_02", "host" : "lt\_replset\_shard\_02/127.0.0.1:40001,127.0.0.1:40002,127.0.0.1:40003" }**

**{ "\_id" : "lt\_replset\_shard\_03", "host" : "lt\_replset\_shard\_03/127.0.0.1:50001,127.0.0.1:50002,127.0.0.1:50003" }**

**databases:**

**{ "\_id" : "admin", "partitioned" : false, "primary" : "config" }**

**{ "\_id" : "school", "partitioned" : true, "primary" : "lt\_replset\_shard\_01" }**

**school.student**

**shard key: { "\_id" : "hashed" }**

**chunks:**

**lt\_replset\_shard\_03 1**

**lt\_replset\_shard\_01 1**

**lt\_replset\_shard\_02 2**

**{ "\_id" : { "$minKey" : 1 } } -->> { "\_id" : NumberLong("-4611686018427387902") } on : lt\_replset\_shard\_03 Timestamp(3, 0)**

**{ "\_id" : NumberLong("-4611686018427387902") } -->> { "\_id" : NumberLong(0) } on : lt\_replset\_shard\_01 Timestamp(3, 1)**

**{ "\_id" : NumberLong(0) } -->> { "\_id" : NumberLong("4611686018427387902") } on : lt\_replset\_shard\_02 Timestamp(2, 4)**

**{ "\_id" : NumberLong("4611686018427387902") } -->> { "\_id" : { "$maxKey" : 1 } } on :lt\_replset\_shard\_02 Timestamp(2, 5)**

如果不能理解观察到新分片上数据的效果，我们可以手动移动数据块以观察效果。我们手动将某个数据块移动到新的副本集中（最好先关闭均衡器sh.stopBalancer();），然后再次插入部分数据再观察数据分布情况。

#### 移动数据块

**mongo 127.0.0.1:20000**

**db.adminCommand({moveChunk:"school.student", find:{\_id:"** **4611686018427387902"}, to:"lt\_replset\_shard\_03"});**

**db.adminCommand({moveChunk:"school.student", find:{\_id:"** **4611686018427387902"}, to:"127.0.0.1:50001"});**

注意：移动到的分片可以是**分片名称（不能有空格）**或者是**Primary节点的IP:Port**。

## 3.7关于mongod、mongos结点配置建议

### 3.7.1mongos

mongos是数据库集群请求的入口，所有的请求通过mongos协调。mongos不占用太多的磁盘空间，占用一定的内存（用来缓存部分shard-key与sharding server的对应关系，便于路由），占用一定的CPU用来处理请求。通常配置mongos的结点不需要有很高的性能，另外由于mongos的集群访问的入口，通常会设置多个来容灾。

### 3.7.2mongod

#### 作为数据存储shards结点

mongod负责实际的数据读写存储，是构成MongoDB集群的基本单位，副本集、分片、配置服务等等都与其有关。作为数据存储结点而言，对于机器的I/O性能、磁盘空间等都需要有较高的需求。

#### 作为配置服务config server结点

配置结点存储的信息至关重要，保存了所有数据库的元信息（路由、分片等）的配置。mongos本身没有物理存储分片信息和数据路由信息，仅仅是将config-server中的信息还存在了自己的内存中。mongos在第一次启动或者重启就会从config-server中加载配置信息，配置服务信息的变化会通知mongos更新内存中的状态。

由于mongos保存了许多元信息，因此在正式环境中通常都会配置多个config server。

# 聊几个Problem

## 4.1如何优雅地删除分片结点

移除分片不是立刻实现的，他需要一个把分片上的数据转移到其他分片的过程，当转移完成后该分片才会被正式踢下线。

移除命令是：db.runCommand({"removeshard":"ip+port"})或者db.runCommand({"removeshard":"\_id"})，注意是这里需要用admin数据库来执行操作。删除分片的动作分为三个状态：start、ongoing、complete。

！注意：如果要删除的结点分片是数据库的主分片，我们会发现要删除的分片结点一直处于ongoing状态，此时我们需要再次执行db.runCommand({"removeshard":"\_id"})动作查看状态的时候，会有如下提示：

**mongos> db.runCommand({"removeshard":"shard0000"})**

**{**

**"msg" : "draining ongoing",**

**"state" : "ongoing",**

**"remaining" : {**

**"chunks" : NumberLong(0),**

**"dbs" : NumberLong(1)**

**},**

**"note" : "you need to drop or movePrimary these databases",**

**"dbsToMove" : [**

**"school"**

**],**

**"ok" : 1**

**}**

告诉我们需要将school数据库的主分片移动到其他分片中，我们接着如下操作将其移动到另一个分片中：

**db.adminCommand({"movePrimary":"school", "to":"shard0001"})**

然后再次执行db.runCommand({"removeshard":"\_id"})，查看分片：

**mongos> db.runCommand({"removeshard":"shard0000"})**

**{**

**"msg" : "removeshard completed successfully",**

**"state" : "completed",**

**"shard" : "shard0000",**

**"ok" : 1**

**}**

已经是completed状态，此时我们使用sh.status()再次查看会发现分片中这个结点已经消失了而不是"draining" : true，这就说明已经将分片移除成功了。另外，我们还可以登录到每个分片上查看其数据量，或者直接在mongos上查看（**db.collection.stats()**）。

我们可以通过如下命令来查询数据库的主分片在哪个分片结点上：

**mongos> use config;**

**switched to db config**

**mongos> db.databases.find();**

**{ "\_id" : "admin", "partitioned" : false, "primary" : "config" }**

**{ "\_id" : "school", "partitioned" : true, "primary" : "shard0000" }**

## 4.2关于数据均衡

### 4.2.1均衡器——自动均衡

MongoDB提供了自动均衡器，在我们默认创建完成分片集群时，这个均衡器就是默认打开的，它负责将数据均匀的分散开。我们可以通过如下命令来查看均衡器的状态：

**sh.getBalancerState(); # 均衡是否打开**

**sh.setBalancerState(true/false); # 设置均衡器的状态**

**sh.isBalancerRunning(); # 均衡器是否在运行**

**use config; db.locks.find({"\_id":"balancer"}); # 查看均衡器，state为0表示未激活，1或者2表示已激活**

在集群中同一时刻只有一个均衡器工作，均衡器每次处理一个chunks。如果再禁用均衡器时，它恰巧在运行，禁用会等待运行完毕再执行。

（理论上来讲，因为测试结果不一样）当我们关闭了均衡器的时候，数据就不会自动的均衡分布，只有等待某个空间不足或者其他情况，数据才会向其他分片迁移。即使开启了均衡器，在没有达到触发条件的情况下，它也是不是执行均衡操作的（毕竟均衡操作对于服务性能有影响）。

#### 启用关闭均衡器的一些场景

1. 数据备份：数据备份时我们需要**关闭**均衡器，避免导致数据不一致的情况；
2. 添加/删除结点：**开启**均衡器做数据迁移，后面在[“此处”](#_4.4.3如果在添加结点、删除结点的时候关闭了均衡器会出现什么情况呢？)也详细提到了；

#### 设置/删除均衡器的时间窗口

设置时间窗口之后，均衡器会在指定的时间段内执行。

**use config;**

**sh.setBalancerState(true);**

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

start-time/stoptime：HH:MM，HH在00-23，MM在00-59。在MongoDB的官方文档中建议：如果我们设置了时间窗口，那么就不要使用sh.startBalancer()方法。

我们可以通过如下命令来删除均衡器的时间窗口：

**use config;**

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

#### 禁用和开启均衡器

sh.stopBalancer()或者sh.setBalancerState(false);或者use config; db.settings.update( { \_id: "balancer" }, { $set : { stopped: true } } , { upsert: true } )用来禁用均衡器；

sh.startBalancer()或者sh.setBalancerState(true);或者db.settings.update( { \_id: "balancer" }, { $set : { stopped: false } } , { upsert: true } )用来启用均衡器；

sh.getBalancerState()查看均衡器的状态。

PS：如果初次在config的settings中看不到均衡器的状态，那是因为有些版本的bug，只要将均衡器停止再开启即可看见。

#### 均衡规则

均衡器均衡数据会将数据块（chunks）多的分片上的数据均衡到数据块（chunks）较少的分片上。在MongoDB的官方文档中，关于均衡器进行数据均衡的规则描述如下：

**迁移阈值**

为了使均衡对集群性能的影响减小到最低，在不同分片之间的数据块数量差异达到一定的阈值时，[balancer](http://docs.mongoing.com/manual-zh/reference/glossary.html" \l "term-balancer) 才会开始进行均衡动作。迁移阈值是指集群中 [chunks](http://docs.mongoing.com/manual-zh/reference/glossary.html" \l "term-chunk) 最多的分片与chunks最少的分片之间数据块数量的差，均衡器有以下阈值:

|  |  |  |  |
| --- | --- | --- | --- |
|  | 数据块的数量 | 迁移阈值 | 说明 |
| 规则1 | 少于20个 | 2 | 当集群中总chunks数目在(min，20)之间时，如果有两个分片上chunks的数量差达到了2，此时均衡器便会开始工作。 |
| 规则2 | 20-79 | 4 | 当集群中chunks数目在[20，79]之间时，如果有两个分片上的chunks的数量达到了4，此时均衡器开始工作。 |
| 规则3 | 80 and greater | 8 | 当集群中chunks的数目在[80，max)之间时，如果有两个分片上的chunks达到了8，此时均衡器便会开始工作。 |

一旦一个均衡过程开始,除非集群中,这个集合在不同分片的数据块数目差 少于2个 或者数据块迁移失败,均衡过程不会停止。

**分片数据量大小**

MongoDB默认的chunks的大小是64M，这也是部署集群最合适的大小，chunks的大小必须在[1，2014]这个范围。官方文档提到，合理的缩小chunks-size可以提高自动迁移的效率。

通常不建议在启动的时候设置chunks的大小，建议使用save方法来修改chunksize的值：

**use config;**

**db.settings.save({ \_id:"chunksize", value:<sizeInMB>})**

### 4.2.2手动均衡

除了让均衡器进行数据块的移动之外，我们还可以手动进行数据块在分片之间的均衡操作。

手动均衡使用的命令是：

db.adminCommand({moveChunk:" **<db.collection>**", find:{\_id:"**<chunks\_id>** "}, to:"**<shard\_id/ip>**"});

db.collection：要移动的数据库的表；

chunks\_id：要移动的chunk的id编号，这个编号可以通过sh.status()查看，或者use config; db.chunks.find();查看；

shard\_id/ip：要移动到的分片id或者地址；

手动均衡示例：

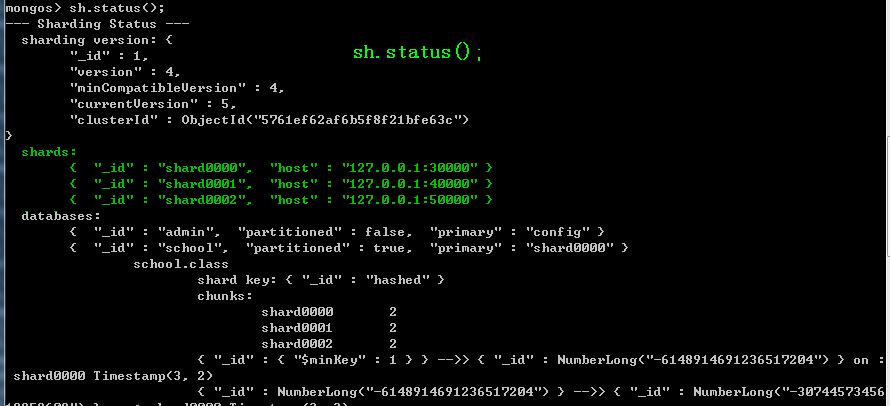
**db.adminCommand({moveChunk:"school.student", find:{\_id:"school.student-\_id\_-2426208201423656718"}, to:"shard0002"});**

**db.adminCommand({moveChunk:"school.student", find:{\_id:"1011079072357423905"}, to:"shard0000"});**

**db.adminCommand({moveChunk:"school.student", find:{\_id:"-7295109993169927424"}, to:"127.0.0.1:30000"});**

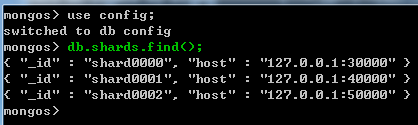
#### 查看分片情况

sh.status();



use config;

db.shards.find();



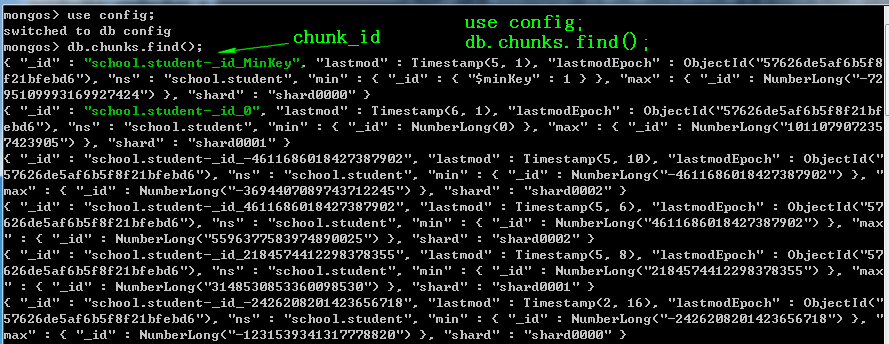
#### 查看chunk\_id情况

sh.status();



use config;

db.chunks.find();



这两种chunk\_id都可以。

## 4.3配置信息查看

配置信息通常存储在config和admin数据库。

|  |  |  |
| --- | --- | --- |
| 操作 | 用途 | 其他 |
| use config;  db.shards.find(); | 查看分片信息，可罗列出当前集群的所有分配信息。 |  |
| use config;  db.databases.find(); | 查看所有数据库信息，显示数据是否分片、当前主信息存储主节点信息。 |  |
| use config;  db.collections.find(); | 查看所有文档信息，显示所有表的ns全称、是否已经删除、主键信息、最后修改信息。 |  |
| use config;  db.mongos.find(); | 查看mongos的状态信息，当前集群中的所有mongos信息。 |  |
| use config;  db.locks.find(); | 查看均衡器锁信息，记录集群中所有的锁，可知道那个mongos是均衡器。也可看到当前所有锁的使用情况。 |  |
| use config;  db.chunks.find(); | 查看所有的块信息，块是数据切分的单位。 |  |
| use config;  db.changelog.find(); | 查看所有分片操作记录 |  |
| use admin;  db.runCommand("getShardMap"); | 查看集群的机器信息，集群配置机器信息，集群分片机器信息等都可在此查看。 |  |
| use admin;  db.runCommand("getCmdLineOpts"); | 查看mongos启动配置信息。 |  |

## 4.4关于均衡器

由于对MongoDB分散数据的方式不够了解，开始不理解均衡器与mongos散发数据到分片有何关系。参考资料和请教相关人士终于弄清楚了。

1. 均衡器的作用在于让数据分布的均匀一些，它的触发需要一定的条件，均衡器与mongos将数据散发到各个分片没有关系；
2. 写入到集群中的一条数据被分在那个分片上，有mongos根据所定的片键决定。

即：如果我们的片键足够均匀，我们也是可以不开启均衡器的。下面的例子用来测试：**关闭均衡器不影响mongos分散数据到各个分片**。

### 4.4.1关闭均衡器、确认分片、指定片键

关闭均衡器：

**mongos> sh.stopBalancer();**

**Waiting for active hosts...**

**Waiting for the balancer lock...**

**Waiting again for active hosts after balancer is off...**

**mongos> sh.getBalancerState();**

**false**

**mongos>**

由于"key":{"\_id":1}自增片键，分割不明显，在此我们使用哈希片键（hash片键在MongoDB2.4+才有）。

**use admin;**

**db.runCommand({"enablesharding":"school"});**

**db.runCommand({"shardcollection":"school.student", "key":{"\_id":"hashed"}});**

下面我们查看一下当前分片情况：

**mongos> use admin;**

**switched to db admin**

**mongos> db.runCommand({"shardcollection":"school.student", "key":{"\_id":"hashed"}});**

**{ "collectionsharded" : "school.student", "ok" : 1 }**

**mongos> sh.status();**

**--- Sharding Status ---**

**sharding version: {**

**"\_id" : 1,**

**"version" : 4,**

**"minCompatibleVersion" : 4,**

**"currentVersion" : 5,**

**"clusterId" : ObjectId("5761ef62af6b5f8f21bfe63c")**

**}**

**shards:**

**{ "\_id" : "shard0000", "host" : "127.0.0.1:30000" }**

**{ "\_id" : "shard0001", "host" : "127.0.0.1:40000" }**

**databases:**

**{ "\_id" : "admin", "partitioned" : false, "primary" : "config" }**

**{ "\_id" : "school", "partitioned" : true, "primary" : "shard0000" }**

**school.student**

**shard key: { "\_id" : "hashed" }**

**chunks:**

**shard0000 2**

**shard0001 2**

**{ "\_id" : { "$minKey" : 1 } } -->> { "\_id" : NumberLong("-4611686018427387902") } on : shard0000 Timestamp(2, 2)**

**{ "\_id" : NumberLong("-4611686018427387902") } -->> { "\_id" : NumberLong(0) } on : shard0000 Timestamp(2, 3)**

**{ "\_id" : NumberLong(0) } -->> { "\_id" : NumberLong("4611686018427387902") } on : shard0001 Timestamp(2, 4)**

**{ "\_id" : NumberLong("4611686018427387902") } -->> { "\_id" : { "$maxKey" : 1 } } on :shard0001 Timestamp(2, 5)**

**mongos>**

我们看到默认的分片以及数据块如上。注意，当我们删除一个collection的时候，它的片键配置也会消失。

### 4.4.2插入大量数据、查看分片情况

我们插入20w条数据：

**mongos> use school;**

**switched to db school**

**mongos> for(var i=0; i<200000; i++){ db.student.insert({"name":i}) }**

**WriteResult({ "nInserted" : 1 })**

**mongos> db.student.find().count();**

**200000**

**mongos> sh.status();**

**--- Sharding Status ---**

**sharding version: {**

**"\_id" : 1,**

**"version" : 4,**

**"minCompatibleVersion" : 4,**

**"currentVersion" : 5,**

**"clusterId" : ObjectId("5761ef62af6b5f8f21bfe63c")**

**}**

**shards:**

**{ "\_id" : "shard0000", "host" : "127.0.0.1:30000" }**

**{ "\_id" : "shard0001", "host" : "127.0.0.1:40000" }**

**databases:**

**{ "\_id" : "admin", "partitioned" : false, "primary" : "config" }**

**{ "\_id" : "school", "partitioned" : true, "primary" : "shard0000" }**

**school.student**

**shard key: { "\_id" : "hashed" }**

**chunks:**

**shard0000 3**

**shard0001 4**

**{ "\_id" : { "$minKey" : 1 } } -->> { "\_id" : NumberLong("-4611686018427387902") } on : shard0000 Timestamp(2, 2)**

**{ "\_id" : NumberLong("-4611686018427387902") } -->> { "\_id" : NumberLong("-2426208201423656718") } on : shard0000 Timestamp(2, 8)**

**{ "\_id" : NumberLong("-2426208201423656718") } -->> { "\_id" : NumberLong(0) } on : shard0000 Timestamp(2, 9)**

**{ "\_id" : NumberLong(0) } -->> { "\_id" : NumberLong("2184574412298378355") } on : shard0001 Timestamp(2, 6)**

**{ "\_id" : NumberLong("2184574412298378355") } -->> { "\_id" : NumberLong("4611686018427387902") } on : shard0001 Timestamp(2, 7)**

**{ "\_id" : NumberLong("4611686018427387902") } -->> { "\_id" : NumberLong("6669710830083149403") } on : shard0001 Timestamp(2, 10)**

**{ "\_id" : NumberLong("6669710830083149403") } -->> { "\_id" : { "$maxKey" : 1 } } on :shard0001 Timestamp(2, 11)**

**mongos>**

好，我们看到了，事实确实如此，均衡器确实不会影响mongos对于数据的分散录入。

### 4.4.3如果在添加结点、删除结点的时候关闭了均衡器会出现什么情况呢？

#### 添加结点

如果你在添加结点的时候，没有开启均衡器，那么后来录入的数据是不会被均衡到这个新节点的。当然你可以选择手动对数据块进行均衡操作，手动操作将某些分片移动之后，我们如果再进行数据录入操作，此时新节点便会分配数据了。

当然了，添加结点后创建的片键，是可以享用该分片的存储权限的。

#### 删除结点

如果你已经手动迁移了这些数据，删除就行。如果没有手动迁移，这样做会导致mongos数据访问出错。

#### 建议

删除结点的场景比较少，添加结点往往及时为了均衡数据，因此添加结点通常不建议你关闭均衡器。

## 4.5关于片键

片键是文档的一个属性字段或者一个符合索引字段，一旦建立不能改变，片键是数据分片的关键，片键的选取直接影响集群的性能。片键也是查询时一个常用的索引。

### 4.5.1片键不能修改（暂未测）

片键一旦添加不能修改，如果要修改片键，需要通过如下繁琐的方式来进行：

1. 导出所有数据；
2. 删除原来的分片集合；
3. 创建集合，创建的同时指定要分成多少个块（目的是为了提高插入数据时的性能）；
4. 重新导入数据；

### 4.5.2片键种类

* 递增片键

这类片键常用在时间戳、日期、自增的主键、ObjectId，\_id等，这类片键写入会集中在一个分片服务器上，写入不具有分散性，会导致单台服务器的压力比较大，这台服务器可能会成为性能瓶颈。但是非常容易分割。

如下是一个以时间戳创建的递增片键：

**mongos> use foo**

**mongos> db.bar.ensureIndex({"timestamp":1}) # 为该列创建递增索引**

**mongos> sh.enableSharding("foo") # 使能该库的分片功能**

**{ "ok" : 1 }**

**mongos> sh.shardCollection("foo.bar",{"timestamp":1}) # 指定片键**

**{ "collectionsharded" : "foo.bar", "ok" : 1 }**

当然了，均衡器会协助进群将数据块均匀的分散在集群的每个分片上。

* 哈希片键

使用hash片键，最大的优点是使得数据均匀的分布在各个分片上，没有单个服务器的压力存在。但是它也有缺点由于数据分散的随机性，数据读取也是随机的，对于这类片键的数据可能会命中更多的分片，一般具有随机性的片键（密码、hash、MD5等）查询隔离性能比较差。

如下是一个创建hash片键的例子：

**mongos> db.bar.ensureIndex({"files\_id":"hashed"}) # 为该列创建hash索引**

**mongos> sh.enableSharding("foo") # 使能该库的分片功能**

**{ "ok" : 1 }**

**mongos> sh.shardCollection("foo.fs.chunks",{"files\_id":"hashed"}) # 指定片键**

**{ "collectionsharded" : " foo.fs.chunks ", "ok" : 1 }**

* 组合片键

数据库中没有比较合适的片键供选择，或者是打算使用的片键基数太小（即变化少如星期只有7天可变化），可以选另一个字段使用组合片键，甚至可以添加冗余字段来组合。一般是粗粒度+细粒度进行组合。

**组合片键的创建，**对GridFS的chunks集合使用files\_id和n组合分片

**mongos> sh.enableSharding("foo")**

**{ "ok" : 1 }**

**mongos> sh.shardCollection("foo.fs.chunks",{"files\_id":1, "n":1})**

**{ "collectionsharded" : " foo.fs.chunks ", "ok" : 1 }**

### 4.5.3片键选取建议

一个良好的片键应该具备如下特点：

1. 可以讲数据均匀的分布到各个分片上；
2. 保证CURD可利用局部性；

（3）有足够的粒度进行块拆分。

满足这些要求的的片键通常由两个字段组成，**第一个是粗粒度，第二个是粒度较细。那么我们需要使用复合片键{userid：1，\_id:1}**。当用户同时插入数据时，我们可以预见大多数情况下，这些数据会被均匀的分布到所有的片上，而且分片里的唯一字段\_id能保证对任意一个文档的查询和更新始终都能指向单个分片。如果对用户ID执行更复杂的查询，那么路由也只会将查询路由包含此用户ID存在的片上，而不会发到所有分片。

大致了解了片键的种类，那么怎么选择片键呢？无非从两个方面考虑，数据的查询和写入，最好的效果就是数据查询时能命中更少的分片，数据写入时能够随机的写入每个分片，关键在于如何**权衡性能和负载**。

如何选择片键主要从下面几个问题考虑：

（1）首先确定一个经常性查询的字段

（2）找到影响这些操作性能的关键点

（3）如果选的字段基数比较小，添加一个粒度细的字段

      怎么选择，如何权衡，跟具体的业务以及数据类型都有关系，具体问题具体分析。

## 4.6正在更新的文档所属的块正在被迁移

跟新操作会发生在老的分片上，待更新完成后，新的数据会被复制到新分片的块中。

## 4.7删除moveChunk目录下的老数据安全吗？

是可以删除的，这些数据是在均衡的时候在后台创建的备份文件，一旦迁移完成这些文件可以删除。

## 4.8mongs处理连接

mongos采用连接池来处理客户端的连接，无状态连接。另外，mongos和mongod之间有一个长连接。

## 4.9mongs的配置服务

mongos会延迟更新自己的本地缓存配置，可以通过flushRouterConfig来强制刷新本地缓存。db.adminCommand(flushRouterConfig);。

分片的相关命令参考：https://docs.mongodb.com/manual/reference/command/flushRouterConfig/#dbcmd.flushRouterConfig

## 4.10mongd启动选项

**mongod --nojournal选项、oplogSize选项、--fork选项、logpath选项在mongod启动时的添加。**

**块拆分/块迁移的触发条件。**

## 4.11未分片的数据

没有分片的数据均存储在主分区上，所有的数据库都有自己的主分区，可以在config中查看：

**mongos> use config;**

**switched to db config**

**mongos> db.databases.find();**

**{ "\_id" : "admin", "partitioned" : false, "primary" : "config" }**

**{ "\_id" : "school", "partitioned" : true, "primary" : "lt\_replset\_shard\_01" }**

**mongos>**

## 4.12先插入数据后进行分片

这是可以的，和普通过程相同。但是要注意前面提到的，作为片键的列必须是有索引的。没有索引需要先创建，创建之后再进行片键等等的指定。

已经有的数据，创建索引和片键后并不会立即分片，在测试时发现需要继续写入部分数据才会产生分片。对于后分片的触发条件，暂未做深入了解。

## 4.13分片集群中的读写分离

在副本集模型中，我们可以在client中选择性的对副本集中的成员进行读写设置。但是在分片集群中，我们连接mongos，此时如何进行读写分离的工作呢？

首先可以进行读写分离，关于读写分离MongoDB提供了五种模式，可设置优先读取的结点类型：

|  |  |
| --- | --- |
| 读优先结点 | 含义 |
| primary | 这也是默认模式，所有的读操作都从primary节点读取。 |
| primaryPreferred | 大多数情况下，操作从primary节点读取数据，当primary出现问题时从secondary节点读取数据。 |
| secondary | 这也是默认模式，所有的读操作都从secondary节点读取。 |
| secondaryPreferred | 大多数情况下，操作从secondary节点读取数据，当secondary出现问题时从primary节点读取数据。 |
| nearest | 从副本集中网络延迟最小的结点读取数据，不管结点类型。 |

对于读写分离操作的设置，我们需要在调用对应语言的SDK时去设置。如下是python版本的设置：

#encoding:utf=8

import pymongo

import string

import struct

import os

import time

import datetime

import sys

from optparse import OptionParser

from pymongo import MongoClient

from pymongo import ReadPreference

def get\_mongo\_conn(host, port, user, password):

conn = MongoClient(host, port, readPreference='secondaryPreferred')

print conn.read\_preference

conn["admin"].authenticate(user,password)

return conn

def get\_info\_by\_md5(param\_objectid):

conn = get\_mongo\_conn('10.58.127.56', 7333, 'mongodb', 'zhangsan')

…

if \_\_name\_\_ == '\_\_main\_\_':

usage\_str = "usage: %prog [options]"

version\_str = "%prog 1.0.0"

parser = OptionParser(usage=usage\_str, version=version\_str)

(options, args) = parser.parse\_args()

nargs = len(args)

if nargs != 1:

print 'usage parameter error'

print 'usage: python get\_info\_by\_md5.py md5\_str'

exit()

md5\_str\_file = str(args[0])

level = get\_info\_by\_md5(md5\_str\_file)

print md5\_str\_file, level

# 参考资料

1. http://www.cnblogs.com/huangxincheng/archive/2012/03/07/2383284.html
2. http://www.cnblogs.com/lazyboy/archive/2012/11/26/2789401.html
3. http://www.cnblogs.com/spnt/archive/2012/07/27/2611540.html
4. http://www.cnblogs.com/spnt/archive/2012/07/25/2608804.html
5. http://www.cnblogs.com/zhoujinyi/p/4668218.html
6. http://docs.mongoing.com/manual-zh/core/sharding-balancing.html

https://docs.mongodb.com/manual/core/sharding-balancing/中英文对照着看，中文版翻译个别地方有问题。

1. http://nosqldb.org/p/5148460edba1dfc408016889
2. http://docs.mongoing.com/manual-zh/tutorial/migrate-chunks-in-sharded-cluster.html
3. https://docs.mongodb.com/manual/tutorial/migrate-chunks-in-sharded-cluster/
4. http://www.lanceyan.com/tech/arch/mongodb\_shard1.html
5. http://my.oschina.net/u/1251522/blog/337889
6. http://blog.chinaunix.net/uid-15795819-id-3521990.html
7. http://www.cnblogs.com/cyhe/p/5519830.html
8. http://www.cnblogs.com/gaobing/p/5025983.html
9. http://stackoverflow.com/questions/12301753/need-list-of-config-servers-mongodb