

**by volshell**

1. **简介**

mongoDB 的名称取自“ humongous ” ( 巨大的 ) 的中间部分 。由 10gen 公司为其提供商业支持。

• 它是一个开源的、面向文档的数据库，属于nosql(Not Only SQL) 数据库中的一种。

• 它可运行在 Linux 、 Windows 或 OSX 平台，支持 32 位和 64 位应用并且提供了Java ， PHP ， Ruby ，Python， C# ， C++ ， JavaScript 等多种语言的驱动程序.

1. **特性**

mongoDB 是一个可扩展、高性能的下一代数据库，由 C++ 语言编写，旨在为 web 应用提供可扩展的高性能数据存储解决方案。它的特点是高性能、易部署、易使用，存储数据非常方便，主要特性有：

• 模式自由，支持动态查询、完全索引，可轻易查询文档中内嵌的对象及数组。

• 面向集合存储，易存储对象类型的数据 , 包括文档内嵌对象及数组 。

• 高效的数据存储 , 支持二进制数据及大型对象 ( 如照片和视频 ) 。

• 支持复制和故障恢复；提供了主 - 从、主 - 主模式的数据复制及服务器之间的数据复制。

• 自动分片以支持云级别的伸缩性，支持水平的数据库集群，可动态添加额外的服务器 。

适用场景：

• 适合作为信息基础设施的持久化缓存层 。  
• 适合实时的插入，更新与查询，并具备应用程序实时数据存储所需的复制及高度伸缩性。  
• mongo 的 BSON 数据格式非常适合文档化格式的存储及查询 。  
• 适合由数十或数百台服务器组成的数据库。因为mongo 已经包含了对 MapReduce 引擎的内置支持。

不适用场景：

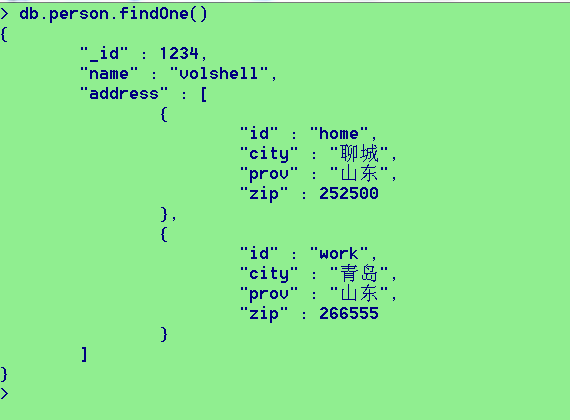
• 要求高度事务性的系统。

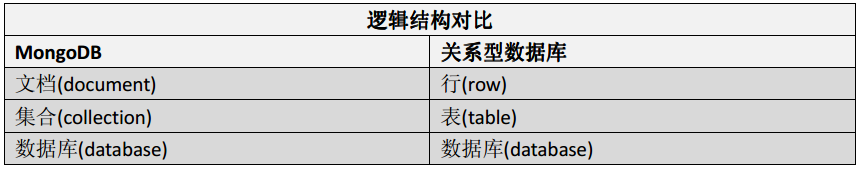
• 传统的商业智能应用。

• 复杂的跨文档 ( 表 ) 级联查询。

1. **工作方式**

**Document-oriented**





**Collection中没有行列的概念，模式自由。**

**Document 是存储单位（基本数据类型，内嵌文档，文档数组）**

**数据空间：**

数据空间采用预分配,目的是为了避免形成过多的硬盘碎片

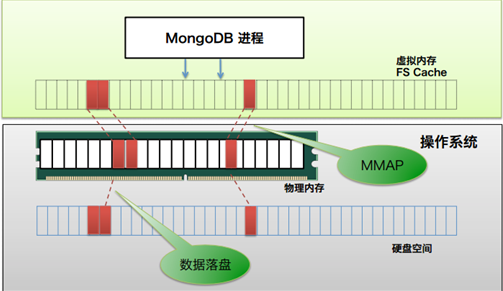
数据文件：初始64mB，2倍递增，最大2GB（32bit）

命名空间：.ns

**BSON：**

**更快的遍历速度🡪**它会将JSON的每一个元素的长度存在元素的头部

**操作更简易 🡪 数据类型**



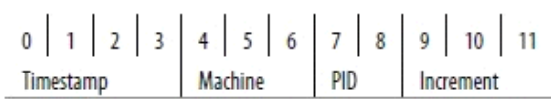
1. **数据类型**

基本数据类型，数组，内嵌文档

TimeStamp 🡪new TimeStamp()

🡪new Date()

**ObjectId** primary key default index can not modify/delete



1. **Startup & Shutdown**

**Startup**

• mongod 启动数据库进程

• --dbpath 指定数据库的目录

• --port 指定数据库的端口 , 默认是 27017

• --bind\_ip 绑定 IP

• --directoryperdb 为每个 db 创建一个独立子目录

• --logpath 指定日志存放目录

• --logappend 指定日志生成方式 ( 追加 / 覆盖 )

**Shutdown** 关闭服务：

1. Ctrl+C ***OR*** kill pid
2. db.shutdownServer()
3. **CRUD**

**连接数据库：**

mongo –port XXX dbname (default test)

**切换数据库**

**use admin**

**Create:**

db.createCollection(‘’)或者直接使用db.foo.insert()自动创建。

db.createCollection("fixedSize",{capped:true,size:100}) 创建capped collection

**Insert:**

**db.foo.insert({…}) = person = {…} + db.foo.insert(person)**

**Update:**

**db.person.save() 不存在，插入；存在，更新**

**$inc 数值递增**

{ $inc : { field : value } }

**$set 字段存在，更新；不存在，插入 🡪与之相反删除字段$unset**

{ $set : { field : value } }

**$push/$addToSet 数组中插入 存在，追加；不存在，新建。否则，报错。**

{ $push : { field : value } }

**$pushAll 数组中插入 存在，追加；不存在，新建。否则，报错。**

{ $push : { field : values\_array } }

**$pop 删除元素**

{ $pop : { field : 1 } } 删除最后一个

{ $pop : { field : -1 } } 删除第一个

**$rename:字段重命名**

{ $rename : { old\_field\_name : new\_field\_name }

**集合重命名：**

db.person.rename(‘persons’)

**Delete:**

**db.person.remove({\_id:1234})**

**删除集合：**

**db.person.drop()**

**Find:**

**查询所有文档：**

**db.person.find()**

**(多)条件查询：**

**db.person.find({\_id:1234})**

**条件表达式：$gt/$gte/$lte/$in/$nin/$or/$exists/$mod….**

**db.users.find({"age":{"$gte":18,"$lte":30}}) 18<=age<=30**

**db.person.find({\_id:{$mod:[2,0]}})**

**db.person.find({\_id:{$exists:true}})**

**数组查询：$all $size**

**db.person.find({address:{$size:2}})**

**类型查询：**

**db.person.find({\_id:{$type:1}})**

**分页查询：**limit() + skip() **OR $slice**

**skip(12).limit(10) = {$slice:[12,10]} 从第12条信息开始，返回10条**

**$where – 神器**

**db.person.find({$where:'this.\_id == 1234'})**

**db.person.find({$where:function(){return this.\_id==1234}})**

**正则表达式：**

**db.person.find({name:/v.\*l$/i})**

**db.person.find({**$regex:'v.\*l$',$options:'i'**})**

**查找指定列：**

**db.person.find({},{name:1,address:1})**

**排序：**

**db.person.find().sort({name:1})**

**游标：**

**var cur = db.person.find()**

**cur.forEach(function(x){printjson(x)})**

1. **Index**

mongoDB 中\_id 字段默认已经建立了索引，这个索引特殊，并且不可删除，不过 Capped Collections 例外。

**类型：**

**Ordinary index/Embedded Keys Index/Compound Keys Index/Unique Index/ multikeys**

**创建索引**

**db.person.ensureIndex({name:1,age:-1,address:1},{background:true},{unique:true})**

**删除：**

**db.person.dropensureIndexes()/db.person.dropIndex({name:1})**

**db.runCommand({dropIndexes:'person', index : '\*'})**

**查看索引：**

**db.person.getIndexes()**

**重建索引：**

**db.person.reIndex()/db.runCommand({reIndex:’person’})**

1. **Optimization**
2. **Profile**

**级别：**

0)：不开启（默认）

1)：慢查询 （>100ms）

2)：所有

开启设置:1.启动设置—profile=level 2.db.setProfiling(level)

**查看：**db.system.profile.find({…})

1. 建立索引
2. 限定返回结果条目数量 limit()
3. 查找使用的字段
4. Capped collection
5. Hint
6. **Security & Authority**
7. **Authority：**

用-auth 选项启动 mongod 进程

进入admin -- use admin

创建用户：db.addUser(‘admin’,’admin’,true) true 只读。

认证 db.auth(‘admin’,’admin’)

用户信息存储在admin的system.users，查看 db.system.users.find()

删除用户：db.removeUser(user\_name)

1. **Access Control：**

**bind ip , when start up mongo with** *mongod –bind\_ip*

**listen port ,when start up mongo with** *mongod –port*

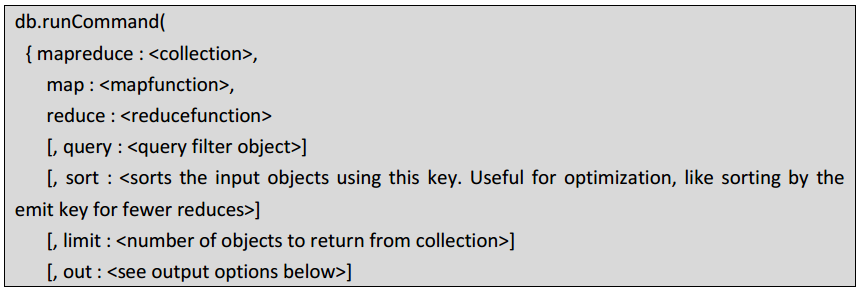
**auth user ,as above**

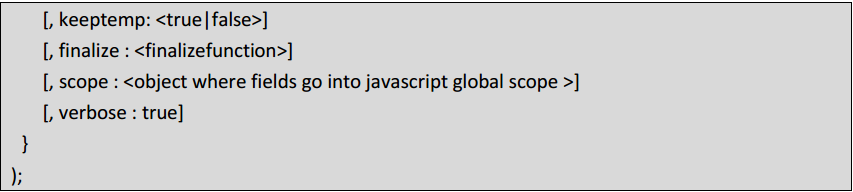
1. **MapReduce**

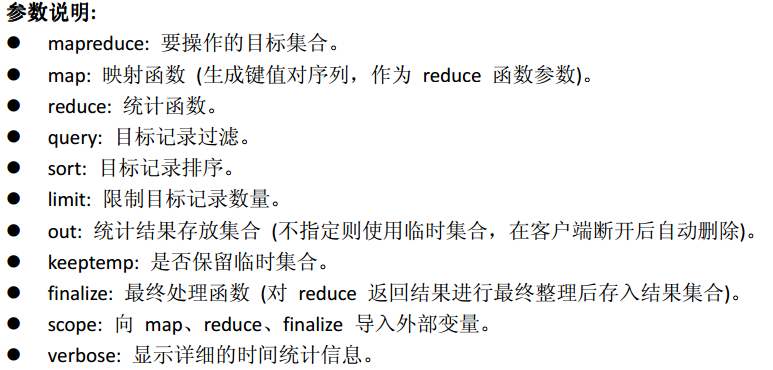
mongoDB的MapReduce中，group的时候较低，MapReduce更慢。

使用 MapReduce 要实现两个函数 map 函数和 Reduce 函数， map 函数调用 emit(key, value)，遍历 collection中所有的记录，将 key与value传递给 Reduce 函数进行处理。map函数和 Reduce函数可以使用 JavaScript 来实现，可以通过 db.runCommand 或 MapReduce 命令来执行一个。

MapReduce 的操作：







1. **Migrate**
2. **数据的导入导出**
3. **手动 使用mongoDB自带工具mongoimport/mongoexport**

**SQL 🡪 mongoDB**

1. **从SQL(mySQL)中导出**

**select \* from users into outfile “D:\\users.csv”**

**fields terminated by ‘,’**

**lines terminated by ‘\n’;**

1. **再导入mongoDB**

**mongoimport -d test -c users --type csv –headerline –file “D:\\users.csv”**

**mongoDB 🡪SQL**

* 1. **从mongoDB 导出**

**mongodbexport –d test –c users –f \_id,name --csv –o “D:\\users.csv”**

* 1. **导入SQL(mySQL)**

**load data local infile “D:\\users.csv” into table users**

**fields terminated by ‘,’**

**lines terminated by ‘\n’**

**(\_id,name,age);**

**【TIPS】表已建立；字段对应；fieds,lines terminated 相同。**

1. **Tools – mongoVEU**
2. **数据的备份恢复**

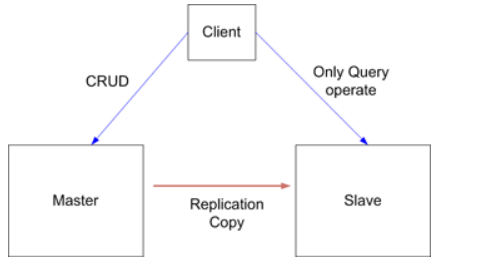
mongoDB提供了数据库和表级的备份恢复工具mongodump/mongorestore。可以实现数据库的无停机备份和恢复，但是我们失去了实时查看数据库视图的能力。

fsync ，强制服务器将缓存数据写入磁盘。同时可以选择获得写入锁。

加锁：db.runCommand({fsync:1,lock:1})

解锁: db.$cmd.sys.unlock.findOne()，db.currentOp()没有线程运行，解锁成功。

1. **Cluster**
2. **Master/Slave**

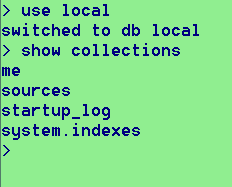
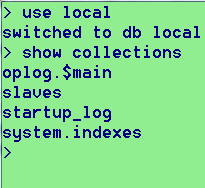


**mongod –dbpath – logpath –logappend –master –port**

**mongod –dbpath – logpath –logappend –slave –port --source loalhost:master\_port**

**mongod –dbpath – logpath –logappend –slave –port --source loalhost:master\_port**

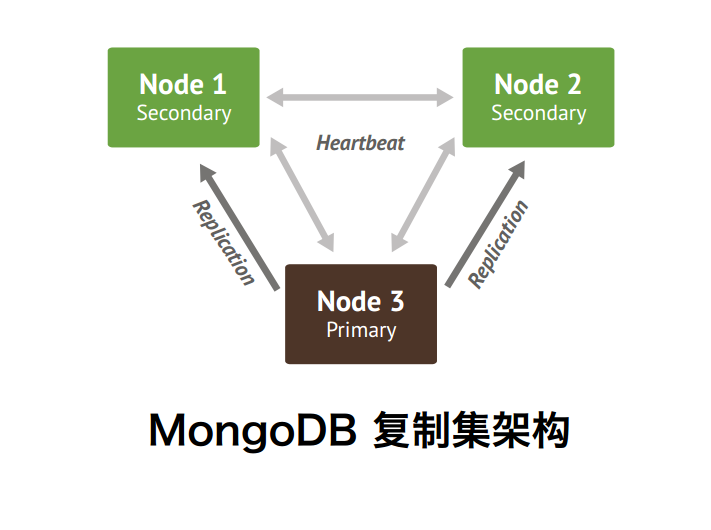
**其他参数：** 



* **Master (A)down**

**Stop B🡪delete B local 🡪startup B --Master**

1. **Replica Sets**



**使用 n 个 mongod 节点，构建具备自动容错转移(auto-failover)、自动恢复(auto-recovery) 的高可用方案。**

**节点类型：**

Standar 普通节点可以成为primary,可以参与投票。

Passive 存储完整数据，参与投票，不能成为primary

Arbiter 仅仅参与投票

对于standard 有primary和secondary之分，而且地位不是永恒的存在primary的降级和secondary的升迁。

1. **降级**

primary 节点使用心跳来跟踪集群中有多少节点对其可见。如果达不到1/2，活跃节点会自动降级为secondary。

1. **同步过程**

一个健康的secondary在运行时，会选择一个离自己最近的，数据比自己新的节点进行数据同步。

选定节点后，它会从这个节点拉取oplog同步日志，具体流程是这样的：

**执行这个op日志**

**将这个op日志写入到自己的oplog中(local.oplog.rs)**

**再请求下一个op日志**

1. **选举：**

如果X是一个 secondary，那么X会定时检测是否需要选举自己成为 primary**。** 其检测内容包括：

是否集群中有其它节点认为自己是 primary？

X节点自己是否已经是 primary？

X节点自己是否有资格成为 primary？

如果这三个问题中的任何一个回答是肯定的，那么X节点就不会试图把自己变成primary。

也就是说，只有当X节点是一个能够当 primary 的secondary，并且其它节点都不是primary时，X才会发起选举并选自己为primary。

而当X发现现在需要一个 primary 并且自己又正好可以充当时，它就会发起一轮选举：X节点会向Y、Z节点各发起一个请求包，告知他们“我认为我可以接管 primary 的角色，你们觉得怎么样？”

当Y和Z收到上面的请求包时，他们会进行下面几项检测：

他们是否已经知道集群中有一个 primary了？

他们自己的数据是否比X节点更新？

是否有其它节点的数据比X节点更新？

如果上面条件有任何一个满足，那么他们都会认为X不够资格成为 primary，他们会发送一个返回包告知X说“停止选举！”。

而如果三个条件都不成立，也就是说他们认为目前集群中确实没有 primary，并且X的数据又是最新的，那么他们会发送返回包告知X说“没问题”。

如果X收到“停止选举！”的返回，那么他会马上停止选举并保持自己为 secondary 状态。

如果X收到所有其它节点都返回说“没问题”，那么他会进入选举过程的第二阶段。

在第二阶段中，X会向其它节点发送一个包，说“我宣布我已经是 primary 了”。这时候，Y和Z节点再进行一些最终的确认：

上面的判断过的所有条件是否依然表明X可以做 primary，

如果确实如此，那么他们会在本轮 primary 选举中向X出赞成票。并且他们投完赞成票后，30秒内不会再做其它投票决定。

上面是说如果第二次确认还是通过的情况，那么如果最终确认没有通过呢。他们会投一个反对票，反对X成为 primary，如果有反对票产生，那么这一轮选举就失败了。X还是保持 secondary 的身份。

下面我们假设一种情况，如果Y给X投了赞成票，而Z给X投了反对票。那这时候Y由于投了赞成票，它在30秒内不能再进行投票。所以如果这时候Z发起选举想让自己成为 primary，那么Z这时候必须要获得X的赞成票。因为这时候Y不能投票，为了获取多数票，Z必须获得X的赞成票。

所以投票的规则是这样的：如果没有人投反对票，并且赞成票的比例过半，那么本轮选举对象就能够成为 primary。

1. **配置：**

**mongod --logpath /dev/null --dbpath /var/mongodb/0 --port 27017 --replSet myset**

**mongod --logpath /dev/null --dbpath /var/mongodb/1 --port 27018 --replSet myset**

**mongod --logpath /dev/null --dbpath /var/mongodb/2 --port 27019 --replSet myset**

**> cfg = { \_id: "myset", members: [**

**... { \_id:0, host:"localhost:27017" },**

**... { \_id:1, host:"localhost:27018" },**

**... { \_id:2, host:"localhost:27019" }**

**... ]}**

**rs.initiate(cfg)**

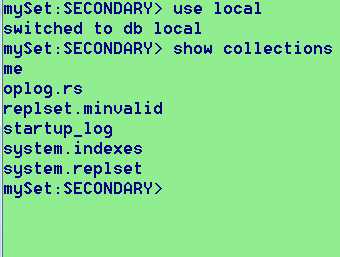
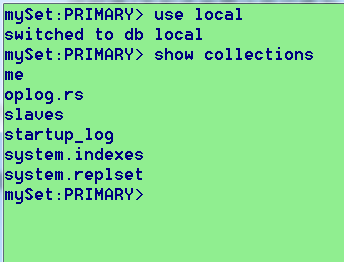
1. **管理：**

db.isMaster() rs.stats() db.printReplicationInfo()

失败节点恢复：

**mkdir /home/conan/dbs/repair/**

**mongod --dbpath=... --port 10001 --replSet setnname --logpath=... --repairpath=...**



1. **阻塞复制：**

使用getLastError的w参数来实现数据的同步性。

使用命令: db.runCommand({getLastError:1,w:N})，直到集群中的N个server完成最新数据的同步之后，阻塞解除。

1. **Sharding**
2. **三种角色**

Sharding server mongod实例，存储实际数据。

Config server mongod 实例，存储整个cluster metadata(包括chunk)

Route server mongos 实例，前端路由。不存储数据。

1. **配置**：

启动shard server

mongod –shardsvr –logpath=… dbpath … --port

启动config server

mongod –configsvr –logpath=… dbpath … --port

启动mongos

mongs –logpath –port –configdb config\_svr:port

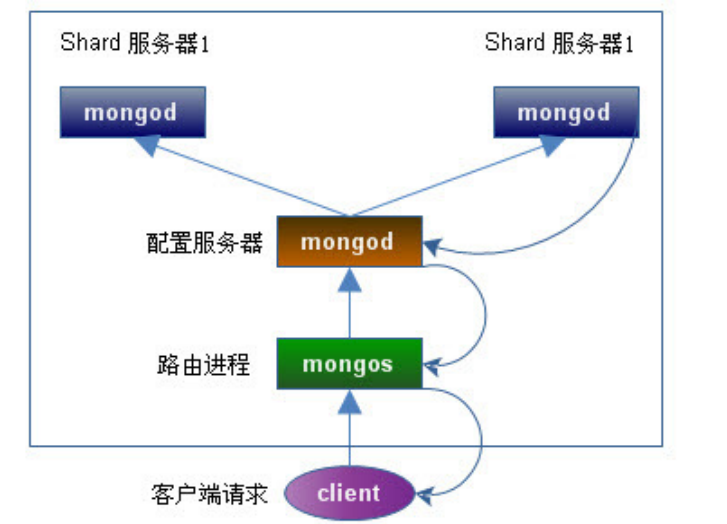
登录mongos,添加shard

Db.runCommand({addshard:’shard\_svr:port’})

配置数据库、集合以及sharding key

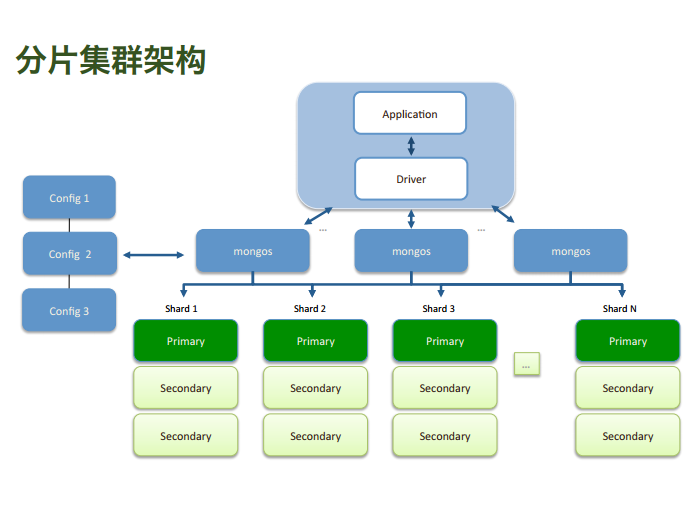
Db.runCommand({enablesharding:’db’})

Db.runCommand({shardcollection:’cl\_name’,key:{\_id:1}})



**Cluster:**

**Replica Sets + Sharding**



**配置方案：**

* **Shard:**

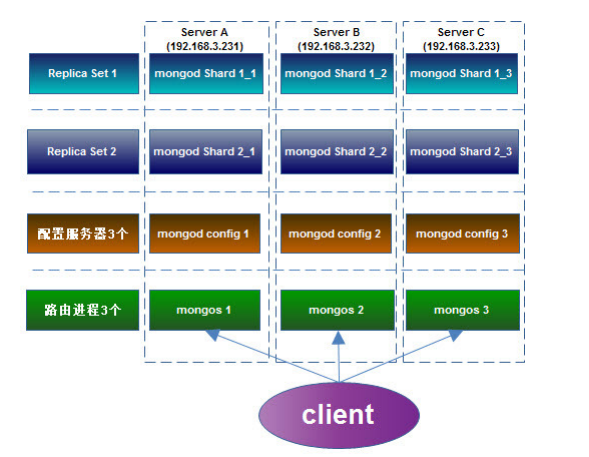
使用 Replica Sets，确保每个数据节点都具有备份、自动容错转移、自动恢复能力。

* **Config:**

使用 3 个配置服务器，确保元数据完整性

* **Route:**

使用 3 个路由进程，实现负载平衡，提高客户端接入性能



**关键**

1. **在配置路由的时候，在每一台server上添加这三台机器所有的config server。**
2. **登录路由之后，添加两个Replica set中的所有shardsvr。**