**NoSQL-day05**

# MongoDB副本集

# 文档管理

## MONGODB

### 环境准备

副本集——也称为MongoDB复制

— 指在多个服务器上存储数据副本，并实现数据同步

— 提高数据可用性、安全性，方便数据故障恢复

### MongoDB复制原理

1. **副本集工作过程**

— 至少需要两个节点。其中一个是主节点，负责处理客户端请求，其余是从节点，负责复制主节点数据

— 常见搭配方式：一主以从、一主多从

— 主节点记录所有操作oplog，从节点定期轮询主节点获取这些操作，然后对自己的数据副本执行这些操作，从而保证从节点的数据与主节点一致

1. **副本集实现方式**

Master-Slave主从复制

— 启动器胎服务器时加上“-master” 参数，作为主节点

— 启动其他服务器时加上“-slave”和“-source”参数，作为从节点

1. **主从复制的优点**

— 从节点可以提供数据查询，降低主节点的访问压力

— 由从节点执行备份，避免锁定主节点数据

— 当主节点故障是，可快速切换到从节点，实现高可用

1. **Replica Sets副本集**

— 从1.6版本开始支持，优于之前的replication

— 支持故障自动切换、自动修复成员节点，降低运维成本

— Replica Sets 副本集的结构类似高可用集群

### 配置Replica Sets

运行MongoDB

启动服务时，指定主机所在的副本集名称

— 所有副本集成员使用相同的副本集名称

— --replSet rs1 //指定副本集名称

1. **51-53主机作为副本集成员，执行以下相同操作**

[root@mysql**51** ~]# vim /usr/local/mongodb/etc/mongodb.conf

logpath=/usr/local/mongodb/log/mongodb.log

logappend=true

dbpath=/usr/local/mongodb/data/db

fork=true

port=**27051**

bind\_ip=**192.168.4.51**

replSet=rs1 //指定副本集名称

1. **配置节点信息**
   1. 在任意一台主机连接mongod服务，执行如下操作

> config = {

... \_id:"rs1",

... members:[

... {\_id:0,host:"192.168.4.51:27051"},

... {\_id:1,host:"192.168.4.52:27052"},

... {\_id:2,host:"192.168.4.53:27053"}

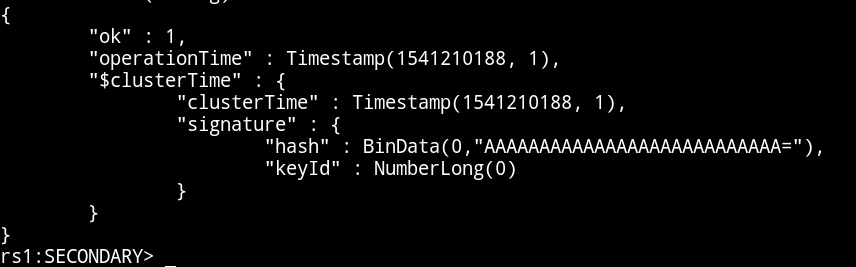
... ]

... };



b) 初始化Replica Sets环境

> rs.initiate(config)



c) 查看状态信息

rs1:SECONDARY> rs.status()

d) 查看是否是master库

rs1:PRIMARY> rs.isMaster()

1. **测试集群（验证副本集配置）**
2. **客户端连接主库存取数据**

]# /usr/local/mongodb/bin/mongo --host 192.168.4.51 --port 27051

rs1:PRIMARY> use db5

rs1:PRIMARY> db.t1.save({name:"bob",age:19})

rs1:PRIMARY> db.t1.find()

1. **客户端连接从库存取数据(52|53)**

[root@mysql52 ~]# mongo --host 192.168.4.52 --port 27052

rs1:SECONDARY> db.getMongo().setSlaveOk() //连接从库时，允许操作数据

rs1:SECONDARY> show dbs

rs1:SECONDARY> use db5

rs1:SECONDARY> show tables

rs1:SECONDARY> db.t1.find()

1. **测试集群高可用功能**

停止当前主库51，在2个从库里选取新的主库；对比数据的完整性，谁的数据最新最完整则谁当主库

## MONgoDB文档管理

[root@mysql51 ~]# bsondump /mdb2/db4/user.bson

[root@mysql51 ~]# mongoimport --host 192.168.4.51 --port 27051 -d db5 -c user /mdb2/db4/user.bson

### 插入文档 save() 和 insert() 或 InsertMany()

格式：> db.集合名.save({key:”值”,key:”值”})

> db.集合名insert({key:”值”,key:”值”})

注意

— 集合不存在时创建集合，然后再插入记录

— save(): \_id字段值已存在时，修改文档字段值

— insert(): \_id字段值已存在时，放弃插入

— \_id字段值不存在时，插入文档

InsertMany() //插入多条记录

> db.集合名.insertMany(

[ {name:”xiaojiu”,age:19},

{name:”laoshi”,email:”yaya@tedu.cn”}

] )

rs1:PRIMARY> db.ta.insertMany( [ {name:"jerry",age:19},

... {name:"alice",email:"x@163.com"},

... {name:"bob",class:"nsd7"}

... ] )

### 查询文档find()

显示所有行: db.集合名.find()

（默认输出20行，输出it可显示后续行）

rs1:PRIMARY> db.user.find()

显示第一行: db.集合名.findOne()

rs1:PRIMARY> db.user.findOne()

指定查询条件并制定显示的字段

db.集合名.find({条件},{定义显示的字段})

rs1:PRIMARY> db.user.find({},{\_id:0,name:1})

rs1:PRIMARY> db.user.find({},{\_id:0,name:1,uid:1})

rs1:PRIMARY> db.user.find({},{name:1})

rs1:PRIMARY> db.user.findOne({},{\_id:0,name:1})

rs1:PRIMARY> db.user.find({shell:"/bin/bash"},{\_id:0})

### 行数显示限制

1. **limit(数字)**  //显示前几行

> db.集合名.find().limit(3)

rs1:PRIMARY> db.user.find({shell:"/sbin/nologin"},{\_id:0,name:1,shell:1}).limit(1)

1. **skip(数字)** //跳过前及行

> db.集合名.find().skip(3)

rs1:PRIMARY> db.user.find(

... {shell:"/sbin/nologin"},{\_id:0,name:1,uid:1}).skip(2)

1. **sort(字段名)** //1升序，-1降序

> db.集合名.find().sort({uid:1|-1})

rs1:PRIMARY> db.user.find({shell:"/sbin/nologin"},

... {\_id:0,name:1,uid:1}

... ).sort({uid:-1})

**综合查询：跳过前10行，升序排列查询结果，并显示查询结果的前3行**

rs1:PRIMARY> db.user.find({shell:"/sbin/nologin"},

... {\_id:0,name:1,uid:1}

... ).skip(10).sort({uid:1}).limit(3)

{ "name" : "ntp", "uid" : 38 }

{ "name" : "gdm", "uid" : 42 }

{ "name" : "tss", "uid" : 59 }

rs1:PRIMARY>db.user.find({shell:"/sbin/nologin"},

{\_id:0,name:1,uid:1,gid:1,password:1}

).skip(5).limit(10).sort({uid:-1})

### 

### 匹配条件

**简单条件**

> db.集合名.find({key:”值”})

> db.集合名.find({key:”值”, keyname:”值”})

1. **范围比较**

$in --------------->在...里

$nin--------------->不在...里

$or----------------->或

1. **相等比较：**

rs1:PRIMARY> db.user.find( {shell:"/bin/bash"} )

rs1:PRIMARY> db.user.find( {uid:10}, {\_id:0,name:1,uid:1})

rs1:PRIMARY> db.user.find( {name:"root",uid:0,shell:"/bin/bash"},{\_id:0} )

1. **在...里....**

rs1:PRIMARY> db.user.find({uid:{$in:[3,7,9,11]}},{\_id:0,name:1,uid:1})

rs1:PRIMARY>db.user.find({name:{$in:["mysql","apache","daemon","adm"]}},{\_id:0,name:1,uid:1}

1. **不再....里.....**

rs1:PRIMARY>db.user.find({shell:{$nin:["/bin/bash","/sbin/nologin"]}}, {\_id:0,name:1,shell:1} )

1. **$or或比较**

rs1:PRIMARY> db.user.find(

... {$or:[{uid:10},{name:"adm"},{shell:"/bin/bash"}]},

... {\_id:0,name:1,shell:1,uid:1})

### 正则匹配

>db.user.find({name:/^a/})

rs1:PRIMARY> db.user.find( {name:/^a/}, {\_id:0,name:1})

### 数值比较($lt[<] $lte[<=] $gt[>] $gte[>=] $ne[!=])

rs1:PRIMARY> db.user.find( {uid:{$gte:1000,$lte:9999}},

... {\_id:0,name:1,uid:1})

rs1:PRIMARY> db.user.find( {uid:{$lte:10}}, {\_id:0,name:1,uid:1})

1. **匹配null,也可以匹配没有的字段**

rs1:PRIMARY> db.user.save({name:null,uid:1002})

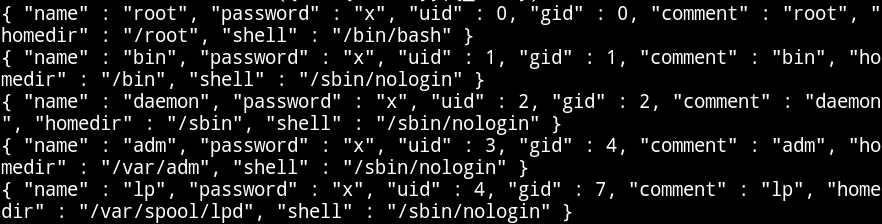
rs1:PRIMARY> db.user.find( {name:null},{\_id:0,name:1,uid:1})

### 更新文档update()

**只修改与条件匹配的指定字段，其他每指定字段全部删除**

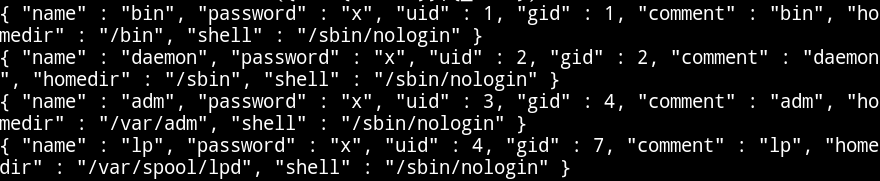
> dn.集合名.update({条件}{修改的字段})

rs1:PRIMARY> db.user.find({uid:{$lte:4}},{\_id:0})



rs1:PRIMARY> db.user.update({uid:{$lte:4}},{password:"A"})

rs1:PRIMARY> db.user.find({uid:{$lte:4}},{\_id:0})



rs1:PRIMARY> db.user.find({password:"A"})

{ "\_id" : ObjectId("5bdc0eb96e64f674df6ff545"), "password" : "A" }

**默认修改：把文档的其他字段都删除，只留下password字段，且只修改与条件匹配的第1行**

**多文档更新**

语法格式：默认值更新与条件匹配的第1行

> db.user.update({条件}, {$set:{修改的字段}},false,true)

> db.user.update({name:”bin”},

{$set:{password:”abc12123”}},false,true)

rs1:PRIMARY> db.user.find({uid:{$gte:3,$lte:6}},{\_id:0})

rs1:PRIMARY> db.user.update(

... {uid:{$gte:3,$lte:6}},

... {$set:{password:"B",shell:"No"}},false,true)

rs1:PRIMARY> db.user.find({uid:{$gte:3,$lte:6}},{\_id:0})

**$set/$unset**

1. **$set 条件匹配时,修改指定字段的值**

db.user.update({ 条件 },$set: { 修改的字段 })

db.user3.update({name:"bin"},{$set:{password:"A"}}

1. **$unset 删除与条件匹配文档的字段**

db. 集合名 .update({ 条件 },{$unset:{key:values}})

db.user3.update({name:"bin"},{$unset:{passwor

d:"A"}})

**$inc**

**$inc 条件匹配时,字段值自加或自减**

– db. 集合名 .update({ 条件 },{$inc:{ 字段名 : 数字 }})

– db.user.update({name:"bin"},{$inc:{uid:2}}) 字段值自加 2

– db.user.update({name:“bin”},{$inc:{uid:-1}}) 字段值自减1

**正整数自加 负整数自减!!!**

**$push / $addToSet**

1. **$push 向数组中添加新元素**

– db. 集合名 .update({ 条件 },{$push:{ 数组名 :“ 值” }})

– db.user.insert({name:"bob",likes:["a","b","c","d","e","f"]})

– db.user.update({name:“bob”},{$push:{likes:“w"}})

1. **$addToSet 避免重复添加**

– db. 集合名 .update({ 条件 },{$addToSet:{ 数组名 :”值” }})

db.user.update({name:"bob"},{$addToSet:{likes:“f"}})

rs1:PRIMARY> db.user.save({name:"alice",like:["a","b","c","d","e"]})

rs1:PRIMARY> db.user.find({name:"alice"},{\_id:0})

rs1:PRIMARY> db.user.update( {name:"alice"},{$push:{like:"e"}})

rs1:PRIMARY> db.user.update( {name:"alice"},{$push:{like:"e"}})

rs1:PRIMARY> db.user.find({name:"alice"},{\_id:0})

rs1:PRIMARY> db.user.update({name:"alice"},{$addToSet:{like:"e"}})

rs1:PRIMARY> db.user.update({name:"alice"},{$addToSet:{like:"G"}})

rs1:PRIMARY> db.user.find({name:"alice"},{\_id:0})

**$pop /$pull**

**$pop 从数组头部删除一个元素**

– db. 集合名 .update({ 条件 },{$pop:{ 数组名 : 数字 }})

– db.user.update({name:"bob"},{$pop:{likes:1}})

– db.user.update({name:"bob"},{$pop:{likes:-1}})

**注：1 删除数组尾部元素 -1 删除数组头部元素**

**$pull 删除数组指定元素**

– db. 集合名 .update({ 条件 },{$pull:{ 数组名 : 值 }})

– db.user.update({name:"bob"},{$pull:{likes:"b"}})

rs1:PRIMARY> db.user.find({name:"alice"},{\_id:0})

rs1:PRIMARY> db.user.update({name:"alice"},{$pop:{like:1}})

rs1:PRIMARY> db.user.update({name:"alice"},{$pop:{like:-1}})

rs1:PRIMARY> db.user.update({name:"alice"},{$pull:{like:"e"}})

### 删除文档 $drop/$remove

$drop //删除集合的同时删除索引

– db. 集合名 .drop( )

– db.user.drop( )

remove() //删除文档时不删除索引

– db. 集合名 .remove({}) // 删除所有文档

– db. 集合名 .remove({ 条件 }) // 删除与条件匹配的文档

– db.user.remove({uid:{$lte:10}})

– db.user.remove({})

rs1:PRIMARY> db.ta.remove({})

rs1:PRIMARY> db.user.remove({uid:{$lte:10}})

rs1:PRIMARY> db.user.find({uid:{$lte:10}})