

NSD NOSQL DAY05

1. [案例1：配置MongoDB副本集](#)
2. [案例2：文档管理](#)

1 案例1：配置MongoDB副本集

1.1 问题

- 具体要求：
- 准备3台mongodb服务器
- 配置副本集服务
- 验证副本集配置

1.2 方案

准备三台虚拟机，配置mongodb副本集，ip分别为192.158.4.51，192.168.4.52，192.168.4.53其中一个为主节点，负责处理客户端请求，其余的都是从节点，负责复制主节点上的数据，实现存储数据副本，提高了数据的可用性，具体分配如表-1所示：

表-1

主机名	IP 地址	端口
mongodb51	192.158.4.51	27077
mongodb51	192.158.4.52	27078
mongodb53	192.158.4.53	27079

1.3 步骤

实现此案例需要按照如下步骤进行。

步骤一：创建mongodb副本集

1) 三台主机安装mongodb (以4.51为例)

```

01. [root@mongodb51 ~]# cd mongodb/
02. [root@mongodb51 mongodb]# mkdir /usr/local/mongodb
03. [root@mongodb51 mongodb]# cd /usr/local/mongodb
04. [root@mongodb51 mongodb]# cp -r \
05. /root/mongodb/mongodb-linux-x86_64-rhel70-3.6.3/bin/ .
06. [root@mongodb51 mongodb]# ls
07. bin
08. [root@mongodb51 mongodb]# mkdir etc
09. [root@mongodb51 mongodb]# mkdir log
10. [root@mongodb51 mongodb]# mkdir -p data/db
11. [root@mongodb51 mongodb]# vim etc/mongodb.conf
  
```

[Top](#)

12. dbpath=/usr/local/mongodb/data/db/
13. logpath=/usr/local/mongodb/log/mongodb.log
14. logappend=true
15. fork=true
16. bind_ip=192.168.4.51
17. port=27077
18. replSet=rs1
19. //加入到副本集，rs1名字随便起，想知道谁和我在一个副本集里，三台机器的名字一样

2) 设置PATH变量

01. [root@mongodb51 mongodb] # vim /etc/profile
02. export PATH=/usr/local/mongodb/bin: \$PATH
03. [root@mongodb51 mongodb] # source /etc/profile

3) 由于启动和停止服务名字太长，可以起一个别名 给停止服务起一个别名

01. [root@mongodb51 mongodb] # alias cmdb='mongod -- dbpath=/usr/local/mongodb/data/

给启动服务起一个别名

01. [root@mongodb51 mongodb] # alias smdb='mongod -f /usr/local/mongodb/etc/mongodb

4) 启动服务并连接

01. [root@mongodb51 ~] # smdb
02. about to fork child process, waiting until server is ready for connections.
03. forked process: 5656
04. child process started successfully, parent exiting
05. [root@mongodb51 ~] # mongo -- host 192.168.4.51 -- port 27077
06. MongoDB shell version v3.6.3
07. connecting to: mongodb://192.168.4.51:27077/
08. MongoDB server version: 3.6.3

[Top](#)

- 09. ...
- 10. >

5) 配置集群信息，任意一台都可以，在这里在51上面操作

```

01. > rs1_config = {           //rs1_config随便起变量名,要记住
02.   _id: "rs1",              //必须为rs1这个，三台主机集群名，配置文件里面写的是这个
03.   members: [
04.     { _id: 0, host: "192.168.4.51: 27077" },           //_id值随意，host值固定
05.     { _id: 1, host: "192.168.4.52: 27078" },
06.     { _id: 2, host: "192.168.4.53: 27079" }
07.   ]
08. };           //回车，出现下面情况为成功
09. {
10.   "_id" : "rs1",
11.   "members" : [
12.     {
13.       "_id" : 0,
14.       "host" : "192.168.4.51: 27077"
15.     },
16.     {
17.       "_id" : 1,
18.       "host" : "192.168.4.52: 27078"
19.     },
20.     {
21.       "_id" : 2,
22.       "host" : "192.168.4.53: 27079"
23.     }
24.   ]
25. }
26. >

```

6) 初始化Replica Sets环境

```

01. > rs.initiate( rs1_config)
02. {
03.   "ok" : 1,
04.   "operationTime" : Timestamp( 1538187475, 1),
05.   "$clusterTime" : {

```

[Top](#)

```

06.      "clusterTime" : Timestamp( 1538187475, 1),
07.      "signature" : {
08.        "hash" : BinData( 0,"AAAAAAAAAAAAAAAAAAAAAAAAAAAA="),
09.        "keyId" : NumberLong( 0)
10.      }
11.    }
12.  }
13.
14.  rs1: SECONDARY>
15.  rs1: PRIMARY>      //提示PRIMARY，51为主

```

7) 在52和53上面查看

```

01.  [root@mongodb52 ~]# mongo -- host 192.168.4.52 -- port 27078
02.  MongoDB shell version v3.6.3
03.  connecting to: mongodb://192.168.4.52:27078/
04.  MongoDB server version: 3.6.3
05.  ...
06.  ...
07.  rs1: SECONDARY>      //提示SECONDARY，52为从
08.  rs1: SECONDARY>
09.  rs1: SECONDARY>
10.
11.  [root@192 ~]# mongo -- host 192.168.4.53 -- port 27079
12.  MongoDB shell version v3.6.3
13.  connecting to: mongodb://192.168.4.53:27079/
14.  MongoDB server version: 3.6.3
15.  ...
16.  ...
17.  rs1: SECONDARY>      //提示SECONDARY，53为从
18.  rs1: SECONDARY>

```

注意：如果初始化错误，重启服务登陆之后重新设置变量，之后再重新初始化

8) 查看状态信息

```

01.  rs1: PRIMARY> rs.status()
02.  ...
03.      "members" : [
04.        {

```

[Top](#)

```

05.         "_id" : 0,
06.         "name" : "192.168.4.51: 27077",
07.         "health" : 1,
08.         "state" : 1,
09.         "stateStr" : "PRIMARY",
10.         "uptime" : 2295,
11.         ...
12.     ...
13. },
14. {
15.     "_id" : 1,
16.     "name" : "192.168.4.52: 27078",
17.     "health" : 1,
18.     "state" : 2,
19.     "stateStr" : "SECONDARY",
20.     "uptime" : 384,
21.     ...
22.     ...
23. },
24. {
25.     "_id" : 2,
26.     "name" : "192.168.4.53: 27079",
27.     "health" : 1,
28.     "state" : 2,
29.     "stateStr" : "SECONDARY",
30.     ...
31.     ...

```

9) 查看是否是master库

```

01. rs1: PRIMARY> rs.isMaster( )
02. {
03.     "hosts" : [
04.         "192.168.4.51: 27077",
05.         "192.168.4.52: 27078",
06.         "192.168.4.53: 27079"
07.     ],
08.     "setName" : "rs1",
09.     "setVersion" : 1,
10.     "ismaster" : true,      //主库

```

[Top](#)

```

11.     "secondary" : false,
12.     "primary" : "192.168.4.51:27077",
13.     "me" : "192.168.4.51:27077",
14.     ...
15.     ...

```

10) 验证副本集，同步数据验证 (51上面写数据)

```

01.  rs1: PRIMARY> use gamedb2
02.  switched to db gamedb2
03.  rs1: PRIMARY> db.a.save({ name: "yaya", age: 75, em: "p@.com" })
04.  WriteResult({ "nInserted" : 1 })

```

52上面查看

```

01.  [root@mongodb52 ~]# mongo --host 192.168.4.52 --port 27078
02.  rs1: SECONDARY> db.getMongo().setSlaveOk()
03.  rs1: SECONDARY> show dbs      //有gamedb2库
04.  admin      0.000GB
05.  config     0.000GB
06.  ddsdb       0.000GB
07.  gamedb2     0.000GB
08.  local       0.000GB
09.  test        0.000GB

```

步骤三：切换主库验证

1) 自动切换主库验证

关闭51

```

01.  [root@mongodb51 ~]# cmdb      //之前设置的别名
02.  killing process with pid: 5656

```

查看52和53

[Top](#)

```

01.  [root@mongodb52 ~]# mongo --host 192.168.4.52 --port 27078
02.  MongoDB shell version v3.6.3

```

```

03. connecting to: mongodb://192.168.4.52:27078/
04. MongoDB server version: 3.6.3
05. ...
06. ...
07. rs1: PRIMARY> //52为主
08. rs1: PRIMARY>
09.
10. [ root@mongodb53 ~] # mongo -- host 192.168.4.53 -- port 27079
11. MongoDB shell version v3.6.3
12. connecting to: mongodb://192.168.4.53:27079/
13. MongoDB server version: 3.6.3
14. ...
15. ...
16. rs1: SECONDARY> //53为从

```

启动51，启动后不会再变为主，会成为52的从

```

01. [ root@mongodb51 ~] # smdb
02. about to fork child process, waiting until server is ready for connections.
03. forked process: 6598
04. child process started successfully, parent exiting
05. rs1: SECONDARY> rs.isMaster()
06. {
07.   "hosts" : [
08.     "192.168.4.51:27077",
09.     "192.168.4.52:27078",
10.     "192.168.4.53:27079"
11.   ],
12.   "setName" : "rs1",
13.   "setVersion" : 1,
14.   "ismaster" : false,
15.   "secondary" : true,
16.   "primary" : "192.168.4.52:27078",
17.   "me" : "192.168.4.51:27077",
18.   ...

```

2 案例2：文档管理

[Top](#)

2.1 问题

- 基于MongoDB环境完成下列练习：
- 插入文档
- 查询文档
- 更新文档
- 删除文档

2.2 步骤

实现此案例需要按照如下步骤进行。

步骤一：管理文档

1) 把系统用户信息/etc/passwd存储到mdb库下的user集合里

```

01.  rs1: PRIMARY> use mdb
02.  switched to db mdb
03.  rs1: PRIMARY> db.user.save({ name: "yaya", password: "x", uid: 9999, gid: 9999, comment: "",
04.  WriteResult({ "nInserted" : 1 })
05.  rs1: PRIMARY> exit
06.  bye
07.
08.  [ root@mongodb52 ~] # mongoexport -- host 192.168.4.52 -- port 27078 - d mdb - c user -
09.  2018-09-29T11:04:14.967+0800 connected to: 192.168.4.52:27078
10.  2018-09-29T11:04:14.968+0800 exported 1 record
11.
12.  [ root@mongodb52 ~] # cp /etc/passwd /tmp
13.  [ root@mongodb52 ~] # sed -i 's:/,/g' /tmp/passwd
14.  [ root@mongodb52 ~] # sed -i '$r /tmp/passwd' /tmp/user.csv
15.  [ root@mongodb52 ~] # mongoimport -- host 192.168.4.52 -- port 27078 - d mdb - c user --
16.  2018-09-29T11:06:08.355+0800 connected to: 192.168.4.52:27078
17.  2018-09-29T11:06:08.363+0800 imported 41 documents

```

2) 查看

```

01.  [ root@mongodb52 ~] # mongo -- host 192.168.4.52 -- port 27078
02.  rs1: PRIMARY> use mdb
03.  switched to db mdb
04.  rs1: PRIMARY> db.user.findOne()
05.  {
06.    "_id" : ObjectId("5baeeb37ce3cc5539aa21f38"),
07.    "name" : "yaya",
08.    "password" : "x",

```

[Top](#)


```

09.     "uid" : 9999,
10.     "gid" : 9999,
11.     "comment" : "",
12.     "homdir" : "/home",
13.     "shell" : "/bin/bash"
14. }

```

db.user.find ({条件},{定义显示的字段}) #指定查询条件并指定显示的字段

```

01.  rs1: PRIMARY> db.user.find()
02.  { "_id" : ObjectId( "5baeeb37ce3cc5539aa21f38" ), "name" : "y a y a", "password" : "x", "ui
03.  ...
04.  Type "it" for more           //出现这个按it，默认出现20行

```

查看每行的name字段

```

01.  rs1: PRIMARY> db.user.find( {}, { name: 1 } )
02.  { "_id" : ObjectId( "5baeeb37ce3cc5539aa21f38" ), "name" : "y a y a" }
03.  ...
04.  ...
05.  { "_id" : ObjectId( "5baeec2001805180a1011843" ), "name" : "rpc" }
06.  Type "it" for more

```

不看_id字段

```

01.  rs1: PRIMARY> db.user.find( {}, { _id: 0 } )
02.  { "name" : "y a y a", "password" : "x", "uid" : 9999, "gid" : 9999, "comment" : "", "homdir
03.  ...
04.  ...
05.  { "name" : "rpc", "password" : "x", "uid" : 32, "gid" : 32, "comment" : "Rpcbind Daemon"
06.  Type "it" for more

```

不看_id那一列，看name那一列

[Top](#)

```

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

```

02. { "name" : "y a y a" }
03. ...
04. ...
05. { "name" : "rpc" }
06. Type "it" for more

查看以a开头的name字段

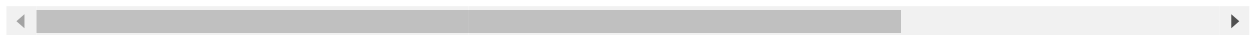
01. rs1: PRIMARY> db.user.find({ name:/^a/},{_id:0})
02. { "name" : "adm", "password" : "x", "uid" : 3, "gid" : 4, "comment" : "adm", "homdir" : ' ' }
03. { "name" : "abrt", "password" : "x", "uid" : 173, "gid" : 173, "comment" : "", "homdir" : ' ' }
04. { "name" : "avahi", "password" : "x", "uid" : 70, "gid" : 70, "comment" : "Avahi mDNS/Daemon", "homdir" : ' ' }



显示查询结果的前一行

limit 数字

01. rs1: PRIMARY> db.user.find({ name:/^a/},{_id:0}).limit (1)
02. { "name" : "adm", "password" : "x", "uid" : 3, "gid" : 4, "comment" : "adm", "homdir" : ' ' }



显示name字段以a开头的第一行

01. rs1: PRIMARY> db.user.findOne({ name:/^a/},{_id:0,name:1,shell:1,uid:1})
02. { "name" : "adm", "uid" : 3, "shell" : "/sbin/nologin" }

跳过几行显示 (2行)

skip 数字

01. rs1: PRIMARY> db.user.find({ name:/^a/},{_id:0,name:1,shell:1}).skip (2)
02. { "name" : "avahi", "shell" : "/sbin/nologin" }

默认升序排序

sort 字段名

[Top](#)

```

01  rs1: PRIMARY> db.user.find({ name:/^a/},{_id:0,name:1,shell:1,uid:1}).sort({ uid:1 })
02  { "name" : "adm", "uid" : 3, "shell" : "/sbin/nologin" }
03  { "name" : "avahi", "uid" : 70, "shell" : "/sbin/nologin" }
04  { "name" : "abrt", "uid" : 173, "shell" : "/sbin/nologin" }

```

降序排序

```

01  rs1: PRIMARY> db.user.find({ name:/^a/},{_id:0,name:1,shell:1,uid:1}).sort({ uid:-1 })
02  { "name" : "abrt", "uid" : 173, "shell" : "/sbin/nologin" }
03  { "name" : "avahi", "uid" : 70, "shell" : "/sbin/nologin" }
04  { "name" : "adm", "uid" : 3, "shell" : "/sbin/nologin" }

```

显示name字段以a开头和uid为3的所有行

```

01  rs1: PRIMARY> db.user.find({ name:/^a/,uid:3},{_id:0,name:1,shell:1,uid:1})
02  { "name" : "adm", "uid" : 3, "shell" : "/sbin/nologin" }

```

3) 条件判断的表示方式

\$in 在...里

```

01  rs1: PRIMARY> db.user.find({ uid:{ $in:[ 1,6,9]}})      //uid的为1或者6或者9的匹配
02  { "_id" : ObjectId( "5baeec2001805180a1011833"), "name" : "bin", "password" : "x", "uid"
03  { "_id" : ObjectId( "5baeec2001805180a1011838"), "name" : "shutdown", "password" : "x"

```

\$nin 不在...里

```

01  rs1: PRIMARY> db.user.find({ uid:{ $nin:[ 1,6,9]}},{_id:0,name:1,uid:1})
02  { "name" : "yaya", "uid" : 9999 }
03  ...
04  ...
05  { "name" : "saslauth", "uid" : 996 }
06  Type "it" for more

```

[Top](#)

\$or 条件满足任意其中一个即可

```

01. rs1: PRIMARY> db.user.find({ $or:[ { name:"root"}, { uid: 1 } ], { _id: 0, name: 1, uid: 1 } )
02. { "name" : "root", "uid" : 0 }
03. { "name" : "bin", "uid" : 1 }

```

4) 正则匹配 , 以a开头的name字段

```

01. rs1: PRIMARY> db.user.find({ name:/^a/ }, { _id: 0, name: 1, uid: 1 } )
02. { "name" : "adm", "uid" : 3 }
03. { "name" : "abrt", "uid" : 173 }
04. { "name" : "avahi", "uid" : 70 }

```

5) 数值比较

\$lt(小于) \$lte (小于等于) \$gt (大于) \$gte (大于等于) \$ne (不等于)

```

01. rs1: PRIMARY> db.user.find({ uid:{ $gte: 10, $lte: 40 } }, { _id: 0, name: 1, uid: 1 } )
02. { "name" : "operator", "uid" : 11 }
03. { "name" : "games", "uid" : 12 }
04. { "name" : "ftp", "uid" : 14 }
05. { "name" : "rpc", "uid" : 32 }
06. { "name" : "rpcuser", "uid" : 29 }
07. { "name" : "ntp", "uid" : 38 }

```

匹配null : 可以匹配没有的字段 , 也可以检查这个字段有没有

```

01. rs1: PRIMARY> db.user.save({ name: null, uid: null })
02. WriteResult({ "nInserted" : 1 })
03. rs1: PRIMARY> db.user.find({ name: null })
04. { "_id" : ObjectId( "5baef385f9f3bf625ea1dbd6" ), "name" : null, "uid" : null }
05.
06. rs1: PRIMARY> db.user.find({ shell: null }) //表示此条文档没有shell字段
07. { "_id" : ObjectId( "5baef385f9f3bf625ea1dbd6" ), "name" : null, "uid" : null }
08. rs1: PRIMARY>

```

6) save和insert的区别

[Top](#)

相同点 : 当集合不存在时创建集合 , 并插入记录

不同点：save() _id字段值已经存在时，修改文档字段值

insert() _id字段值已经存在时，放弃修改文档字段值

```

01.  rs1: PRIMARY> db.t1.save({ name: "bob", age: 19 })
02.  WriteResult({ "nInserted" : 1 })
03.  rs1: PRIMARY> db.t1.insert({ name: "bob", age: 19 })
04.  WriteResult({ "nInserted" : 1 })
05.
06.  rs1: PRIMARY> db.t1.save({ _id: 7, name: "bob", age: 19 })
07.  WriteResult({ "nMatched" : 0, "nUpserted" : 1, "nModified" : 0, "_id" : 7 })
08.  rs1: PRIMARY> db.t1.find()
09.  ...
10.  ...
11.  { "_id" : 7, "name" : "bob", "age" : 19 }
12.  rs1: PRIMARY> db.t1.save({ _id: 7, name: "tom", age: 19 })      //把上一条的记录直接修改
13.  WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
14.  rs1: PRIMARY> db.t1.find()
15.  ...
16.  ...
17.  { "_id" : 7, "name" : "tom", "age" : 19 }
18.  rs1: PRIMARY>
19.
20.
21.  rs1: PRIMARY>
22.  rs1: PRIMARY> db.t1.insert({ _id: 8, name: "tom", age: 19 })    //可以存上
23.  WriteResult({ "nInserted" : 1 })
24.  rs1: PRIMARY> db.t1.insert({ _id: 8, name: "tom", age: 19 })    //存不上
25.  WriteResult({
26.    "nInserted" : 0,
27.    "writeError" : {
28.      "code" : 11000,
29.      "errmsg" : "E11000 duplicate key error collection: mdb.t1 index: _id_ dup key: { : 8.0
30.    }
31.  })

```

7) 插入多行文档

[Top](#)

```

01.  rs1: PRIMARY> db.t1.insertMany([ { name: "xiaojiu" }, { name: "laoshi" } ])
02.  {

```

```

03.     "acknowledged" : true,
04.     "insertedIds" : [
05.         ObjectId( "5baef 526f 9f 3bf 625ea1dbd9" ),
06.         ObjectId( "5baef 526f 9f 3bf 625ea1dbda" )
07.     ]
08. }
09.
10. rs1: PRIMARY> db.t1.find()
11. ...
12. ...
13. { "_id" : ObjectId( "5baef 526f 9f 3bf 625ea1dbd9" ), "name" : "xiaojiu" }
14. { "_id" : ObjectId( "5baef 526f 9f 3bf 625ea1dbda" ), "name" : "laoshi" }

```

8) update修改

```

01. rs1: PRIMARY> db.user.update( { name: "root" }, { password: "XXX" })
02. //如果这一列不写完整，这一行除了password这一行，这一列的其他值都没有了相当于
03. WriteResult( { "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 } )
04. rs1: PRIMARY> db.t1.find( { name: "root" } )
05. rs1: PRIMARY> db.user.find( { name: "root" } ) //没有东西，除了password: "XXX"

```

9) \$set 条件匹配时，修改指定字段的值(局部修改)

```

01. rs1: PRIMARY> db.user.update( { name: "adm" }, { $set: { password: "AAA" } } )
02. WriteResult( { "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 } )
03. rs1: PRIMARY> db.user.find( { name: "adm" } ) //还存在
04. { "_id" : ObjectId( "5baeec2001805180a1011835" ), "name" : "adm", "password" : "AAA",
05.
06.
07. rs1: PRIMARY> db.user.update( { name: /^r/ }, { $set: { password: "FFF" } } )
08. //默认修改匹配条件的第一行
09. WriteResult( { "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 } )
10. rs1: PRIMARY> db.user.update( { name: /^a/ }, { $set: { password: "FFF" } }, false, true )
11. //改匹配到的所有
12. WriteResult( { "nMatched" : 3, "nUpserted" : 0, "nModified" : 3 } )

```

[Top](#)

10) \$unset 删除与条件匹配文档的字段

```

01  rs1: PRIMARY> db.user.update({ name: "sync", { $unset: { password: 1 } })
02  //删除password字段
03  WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })

```

11) 数组

```

01  rs1: PRIMARY> db.user.insert({ name: "bob", like: [ "a", "b", "c", "d", "e", "f", ] })
02  WriteResult({ "nInserted" : 1 })

```

\$pop 删除数组末尾一个元素，1删除最后一个，-1删除第一个

```

01  rs1: PRIMARY> db.user.update({ name: "bob", { $pop: { like: 1 } })
02  //删除匹配的第一条的最后一个
03  WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
04  rs1: PRIMARY> db.user.update({ name: "bob", { $pop: { like: - 1 } })
05  //删除匹配的第一条的第一个
06  WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })

```

\$push 向数组中添加新元素

```

01  rs1: PRIMARY> db.user.update({ name: "bob", { $push: { like: "Z" } }) //默认添加到最后
02  WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
03  rs1: PRIMARY> db.user.update({ name: "bob", { $push: { like: "W" } })
04  WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
05  rs1: PRIMARY> db.user.find({ name: "bob" })
06  { "_id" : ObjectId( "5baef7b2034891a205de2959"), "name" : "bob", "like" : [ "b", "c", "d"

```

\$addToSet 避免重复添加

```

01  rs1: PRIMARY> db.user.update({ name: "bob", { $addToSet: { like: "W" } })
02  WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 0 })
03  rs1: PRIMARY> db.user.find({ name: "bob" })
04  { "_id" : ObjectId( "5baef7b2034891a205de2959"), "name" : "bob", "like" : [ "b", "c", "d"

```

[Top](#)

\$pull 删除数组里的指定元素，若有两个bob可以用_id值定义把name:"bob"换成id值

```

01. rs1: PRIMARY> db.user.update({ name: "bob" }, { $pull: { like: "c" } })
02. WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
03. rs1: PRIMARY> db.user.find({ name: "bob" })
04. { "_id" : ObjectId("5baef7b2034891a205de2959"), "name" : "bob", "like" : [ "b", "d", "e"
05. rs1: PRIMARY> db.user.update({ "_id": ObjectId("5afc1a717eff45e9cf c57ed3") }, { $push: { lik
06. WriteResult({ "nMatched" : 0, "nUpserted" : 0, "nModified" : 0 })
07. rs1: PRIMARY>

```

\$inc 条件匹配时，字段值自加或自减

```

01. rs1: PRIMARY> db.user.update({ uid: { $lte: 10 } }, { $inc: { uid: 2 } })
02. //设置字段值自加2,默认改第一行
03. WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
04.
05. rs1: PRIMARY> db.user.update({ uid: { $lte: 10 } }, { $inc: { uid: 2 } }, false, true)
06. //设置字段值自加2, false, true改全部
07. WriteResult({ "nMatched" : 8, "nUpserted" : 0, "nModified" : 8 })
08. rs1: PRIMARY>
09.
10. rs1: PRIMARY> db.user.update({ uid: { $lte: 10 } }, { $inc: { uid: - 1 } })
11. //负数时是自减1, 默认改第一行
12. WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })

```

12) 删除文档

remove()与drop()的区别

remove()删除文档时不删除索引

drop()删除集合的时候同时删除索引

```

01. rs1: PRIMARY> db.t1.remove({})
02. WriteResult({ "nRemoved" : 6 })
03.
04. rs1: PRIMARY> db.user.remove({ name: "/^a/" }) //删除以a开头的记录
05. WriteResult({ "nRemoved" : 0 })
06.
07. rs1: PRIMARY> db.t1.drop() //删除集合t1

```

[Top](#)

- 08. true
- 09. rs1: PRIMARY>

[Top](#)