第1章 关系型与非关系型

```
NoSQL  not only sql
NoSQL，指的是非关系型的数据库。
NoSQL有时也称作Not Only SQL的缩写是对不同于传统的关系型数据库的数据库管理系统的统称。
对NoSQL最普遍的解释是”非关联型的”，强调Key-Value Stores和文档数据库的优点，而不是单纯的RDBMS。
NoSQL用于超大规模数据的存储。
这些类型的数据存储不需要固定的模式，无需多余操作就可以横向扩展。
今天我们可以通过第三方平台可以很容易的访问和抓取数据。
用户的个人信息，社交网络，地理位置，用户生成的数据和用户操作日志已经成倍的增加。
我们如果要对这些用户数据进行挖掘，那SQL数据库已经不适合这些应用了
NoSQL数据库的发展也却能很好的处理这些大的数据。
```

第2章 mongo和mysql数据对比

```
mysql里的数据
user库
user_info表
name     age   job
oldzhang   24  IT
cookzhang  28    cook
xiaozhang  26    IT

mongo里的数据
user库
user_info集合
{ name:"oldzhang", age:"24", job:"IT" }
{ name:"cookzhang", age:"28", job:"cook" }
{ name:"xiaozhang", age:"26", job:"IT" }
{ name:"cookya", age:"23", job:"cook" , host:"XZ" }
```


第3章 mongo的特点

```
高性能：
Mongodb提供高性能的数据持久性
尤其是支持嵌入式数据模型减少数据库系统上的I/O操作
索引支持能快的查询，并且可以包括来嵌入式文档和数组中的键

丰富的语言查询：
Mongodb支持丰富的查询语言来支持读写操作（CRUD）以及数据汇总，文本搜索和地理空间索引

高可用性：
Mongodb的复制工具，成为副本集，提供自动故障转移和数据冗余

水平可扩展性：
```

```
Mongodb提供了可扩展性，作为其核心功能的一部分，分片是将数据分，在一组计算机上

支持多种存储引擎：
WiredTiger存储引擎和、MMAPv1存储引擎和InMemory存储引擎
```

第4章 mongo的应用场景

参考网站：

```
https://www.zhihu.com/question/32071167
```

应用场景：

```
游戏场景，使用 MongoDB 存储游戏用户信息，用户的装备、积分等直接以内嵌文档的形式存储，方便查询、更新

物流场景，使用 MongoDB 存储订单信息，订单状态在运送过程中会不断更新，以 MongoDB 内嵌数组的形式来存储，一次查询就能将订单所有的变更读取出来。

社交场景，使用 MongoDB 存储存储用户信息，以及用户发表的朋友圈信息，通过地理位置索引实现附近的人、地点等功能

物联网场景，使用 MongoDB 存储所有接入的智能设备信息，以及设备汇报的日志信息，并对这些信息进行多维度的分析

视频直播，使用 MongoDB 存储用户信息、礼物信息等,用户评论

电商场景，使用 MongoDB
商城上衣和裤子两种商品，除了有共同属性，如产地、价格、材质、颜色等外，还有各自有不同的属性集，如上衣的独有属性是肩宽、胸围、袖长等，裤子的独有属性是臀围、脚口和裤长等
```

第5章 安装部署mongodb

1.下载软件

```
https://fastdl.mongodb.org/linux/mongodb-linux-x86_64-4.0.14.tgz
```

2.目录规划

```
#软件所在目录
/opt/mongodb
#单节点目录
/opt/mongo_27017/{conf,log,pid}
#数据目录
/data/mongo_27017
```

3.下载并解压

```
yum install libcurl openssl -y
#wget https://fastdl.mongodb.org/linux/mongodb-linux-x86_64-4.0.14.tgz
tar zxf mongodb-linux-x86_64-rhel70-4.0.14.tgz -C /opt/
cd /opt/
ln -s mongodb-linux-x86_64-rhel70-4.0.14 mongodb
```

4.创建目录

```
mkdir -p /opt/mongo_27017/{conf,log,pid}
mkdir -p /data/mongo_27017
```

5.创建配置文件

```
cat >/opt/mongo_27017/conf/mongodb.conf<<EOF
systemLog:
  destination: file
  logAppend: true
  path: /opt/mongo_27017/log/mongodb.log

storage:
  journal:
    enabled: true
  dbPath: /data/mongo_27017
  directoryPerDB: true
  wiredTiger:
    engineConfig:
      cacheSizeGB: 0.5
      directoryForIndexes: true
    collectionConfig:
      blockCompressor: zlib
    indexConfig:
      prefixCompression: true

processManagement:
  fork: true
  pidFilePath: /opt/mongo_27017/pid/mongod.pid

net:
  port: 27017
  bindIp: 127.0.0.1,10.0.0.51
EOF
```


6.配置文件解释

```
配置文件注解：
systemLog:
  destination: file #Mongodb 日志输出的目的地，指定一个file或者syslog，如果指定file，必须指定
  logAppend: true #当实例重启时，不创建新的日志文件， 在老的日志文件末尾继续添加
  path: /opt/mongo_27017/logs/mongodb.log #日志路径

storage:
  journal: #回滚日志
    enabled: true
  dbPath: /data/mongo_27017 #数据存储目录
  directoryPerDB: true #默认，false不适用inmemoryengine
  wiredTiger:
    engineConfig:
      cacheSizeGB: 1 #将用于所有数据缓存的最大小
      directoryForIndexes: true #默认false索引集合storage.dbPath存储在数据单独子目录
    collectionConfig:
      blockCompressor: zlib
    indexConfig:
      prefixCompression: true

processManagement: #使用处理系统守护进程的控制处理
  fork: true #后台运行
  pidFilePath: /opt/mongo_27017/pid/mongod.pid #创建 pid 文件

net:
  port: 27017 #监听端口
  bindIp: 127.0.0.1,10.0.0.51 #绑定ip
```

7.启动mongo

```
/opt/mongodb/bin/mongod -f /opt/mongo_27017/conf/mongodb.conf
```

8.检查是否启动

```
 ps -ef|grep mongo
 netstat -lntup|grep mongo
```

9.写入环境边境

```
echo 'export PATH=/opt/mongodb/bin:$PATH' >> /etc/profile
source /etc/profile
```

10.进入mongo

```
mongo
```

11.关闭命令

```
方法1:推荐
mongod -f /opt/mongo_27017/conf/mongodb.conf --shutdown

方法2：只能是使用localhost方式登陆
mongo
use admin
db.shutdownServer()

方法3：system管理
见下面的优化3
```

第6章 优化警告

1.内存不足

```
** WARNING: The configured WiredTiger cache size is more than 80% of available RAM.
   See http://dochub.mongodb.org/core/faq-memory-diagnostics-wt
```

解决方法：

```
方法1；加大机器内存
方法2：调小配置文件里缓存大小 cacheSizeGB: 0.5
```

2.没有开启访问控制

```
** WARNING: Access control is not enabled for the database.
   Read and write access to data and configuration is unrestricted.
```

解决方法：

```
开启数据库安装认证功能，见用户认证章节
```

3.不建议以root用户运行

```
** WARNING: You are running this process as the root user, which is not recommended.
```

解决方法：

```
方法1：创建普通用户mongo,然后切换到mongo用户启动
方法2：使用system方式登陆,指定运行用户为普通用户mongo
```

创建普通用户：

```
mongod -f /opt/mongo_27017/conf/mongodb.conf --shutdown
groupadd mongo -g 777
useradd mongo -g 777 -u 777 -M -s /sbin/nologin
id mongo
```

mongo的system启动文件:

```
cat >/usr/lib/systemd/system/mongod.service<<EOF
[Unit]
Description=MongoDB Database Server
Documentation=https://docs.mongodb.org/manual
After=network.target

[Service]
User=mongo
Group=mongo
ExecStart=/opt/mongodb/bin/mongod -f /opt/mongo_27017/conf/mongodb.conf
ExecStartPre=/usr/bin/chown -R mongo:mongo /opt/mongo_27017/
ExecStartPre=/usr/bin/chown -R mongo:mongo /data/mongo_27017/

PermissionsStartOnly=true
PIDFile=/opt/mongo_27017/pid/mongod.pid
Type=forking
# file size
LimitFSIZE=infinity
# cpu time
LimitCPU=infinity
# virtual memory size
LimitAS=infinity
# open files
LimitNOFILE=64000
# processes/threads
LimitNPROC=64000
# locked memory
LimitMEMLOCK=infinity
# total threads (user+kernel)
TasksMax=infinity
TasksAccounting=false
# Recommended limits for for mongod as specified in
# http://docs.mongodb.org/manual/reference/ulimit/#recommended-settings

[Install]
WantedBy=multi-user.target
```

```
EOF
```

重新启动mongo

```
systemctl daemon-reload
systemctl start mongod.service
ps -ef|grep mongo
mongo
```

4.关闭大内存页

```
** WARNING: /sys/kernel/mm/transparent_hugepage/enabled is 'always'.
        We suggest setting it to 'never'
** WARNING: /sys/kernel/mm/transparent_hugepage/defrag is 'always'.
        We suggest setting it to 'never'
```

解决方法：修改完参数后需要重启

```
echo "never" > /sys/kernel/mm/transparent_hugepage/enabled
echo "never" > /sys/kernel/mm/transparent_hugepage/defrag
systemctl stop mongod
systemctl start mongod
mongo
```

5.rlimits太低

```
** WARNING: soft rlimits too low. rlimits set to 7193 processes, 65535 files. Number of processes should be at least 32767.5 : 0.5 times number of files.
```

解决方法：

```
cat > /etc/profile<<EOF
ulimit -f unlimited
ulimit -t unlimited
ulimit -v unlimited
ulimit -n 64000
ulimit -m unlimited
ulimit -u 64000
EOF
```

生效配置:

```
source /etc/profile
```

验证：

```
systemctl stop mongod
systemctl start mongod
mongo
```

6.关闭监控服务体验

```
---
Enable MongoDB's free cloud-based monitoring service, which will then receive and display
metrics about your deployment (disk utilization, CPU, operation statistics, etc).

The monitoring data will be available on a MongoDB website with a unique URL accessible to you
and anyone you share the URL with. MongoDB may use this information to make product
improvements and to suggest MongoDB products and deployment options to you.

To enable free monitoring, run the following command: db.enableFreeMonitoring()
To permanently disable this reminder, run the following command: db.disableFreeMonitoring()
---
```

解决方法：

```
db.disableFreeMonitoring()
```

第7章 mongo数据库命令介绍

1.默认数据库说明

```
test:  登陆的时默认的库
admin: 系统预留库，Mongodb的系统管理库
local: 本地预留库，存储关键日志
config: 配置信息库
```

2.查看数据库命令

```
db：查看当前所在库
show dbs/show databases ：查看所有的数据库
show collections/show tables：查看当前库下所有的集合
use admin ：切换到不同的库
```

3.mongo特别的地方

```
1.mongo默认登陆的时候是在test库下
2.mongo不需要提前创建库和表，直接use切换就是创建库，直接插入数据就会创建表
3.使用use切换到的库，如果没有任何数据，实际上并不会真正创建，是个虚的库，所以show dbs并不会显现
```

4.shell窗口执行mongo命令

```
echo "show dbs"|mongo
```

第8章 mongo操作命令

1.插入命令

1.1插入单条

```
db.user_info.insertOne({name: "zhang",age: 29,host: "北京"})
db.user_info.insertOne({name: "yazhang",age: 29,host: "上海"})
db.user_info.insertOne({name: "yaya",age: 29,host: "深圳"})

db.user_info.insertOne(
  {
    name: "zhang",
    age: 29,
    host: "北京"
  }
)
```

1.2插入多条

```
db.user_info.insertMany([
  {name: "zhang",age: 29,host: "北京"},
  {name: "yazhang",age: 29,host: "上海"},
  {name: "yaya",age: 29,host: "深圳"},
])

db.inventory.insertMany([
   { item: "journal", qty: 25, size: { h: 14, w: 21, uom: "cm" }, status: "A" },
   { item: "notebook", qty: 50, size: { h: 8.5, w: 11, uom: "in" }, status: "A" },
   { item: "paper", qty: 100, size: { h: 8.5, w: 11, uom: "in" }, status: "D" },
   { item: "planner", qty: 75, size: { h: 22.85, w: 30, uom: "cm" }, status: "D" },
   { item: "postcard", qty: 45, size: { h: 10, w: 15.25, uom: "cm" }, status: "A" }
]);
```

2.查询语句

2.1查询单条

```
db.user_info.findOne()
select * from user_info limit 1;
```

2.2查询所有

```
db.user_info.find()
select * from user_info;
```

2.3按条件查询

```
db.user_info.find({name:"zhang"})
select * from user_info where name="zhang";
```

2.4只返回想要的字段

```
方法({条件1，条件2},{字段显示开关})
db.user_info.find({name:"zhang"},{name:1,age:1,_id:0})
select name,age from user_info where name="zhang";
```

2.5嵌套查询

```
db.inventory.find({"size.uom":"cm"})
db.inventory.find(
  {
    "size.uom":"cm"
  }
)
```

2.6逻辑查询-and

```
db.inventory.find( { status: "A", size.uom:"cm" } )

db.inventory.find(
  {
    status: "A",
    "size.uom": "cm"
  }
)

select * from inventory where status="A" AND size.uom="cm";

db.inventory.find(
```

```
  {
    status: "A",
    "size.uom": "cm"
  },
  {
    _id: 0,
    status: 1,
    "size.uom": 1
  }
)

select status,size.uom from inventory where status="A" AND size.uom="cm";
```

2.7逻辑查询-or

参考网站:

```
https://docs.mongodb.com/manual/reference/operator/query-comparison/
```

案例:

```
db.inventory.find({$or:[{条件1},{条件2}]})

db.inventory.find(
  {
    $or:[
      {条件1的key: 值},
      {条件2的key: 值}
    ]
  }
)

db.inventory.find( { $or: [ { status: "D" }, { "size.uom": "cm" } ] } )

db.inventory.find(
  {
    $or:
      [
        { status: "D" },
        { "size.uom": "cm" }
      ]
  }
)

SELECT * FROM inventory WHERE status = "D" OR size.uom = cm;

db.inventory.find(
  {
```

```
    $or:
      [
        { status: "A" },
        { qty: { $lt: 30 } }
      ]
  }
)

SELECT * FROM inventory WHERE status = "A" OR qty < 30;
```

2.8 逻辑查询+或+and+正则表达式

```
db.inventory.find(
{
     status: "A",
     $or:
       [
          { qty: { $lt: 30 } },
          { item: /^p/ }
       ]
} )

SELECT * FROM inventory WHERE status = "A" AND ( qty < 30 OR item LIKE "p%")
```

3.更新数据

3.1 按条件更改单条

```
db.inventory.updateOne({查询条件},{更改内容})
db.inventory.find({ item: /^p/ })
db.inventory.updateOne(
   { item: /^p/ },
   {
     $set: { status: "P" }
   }
)
db.inventory.find({ item: /^p/ })
```

3.2 按条件更改多条

```
db.inventory.find({ item: /^p/ })
db.inventory.updateMany(
   { item: /^p/ },
   {
     $set: { status: "P" }
   }
)
db.inventory.find({ item: /^p/ })
```

3.3 添加字段

```
db.inventory.findOne({ item: /^p/ })
db.inventory.updateOne(
   { item: /^p/ },
   {
     $set: { name: "zhangya" }
   }
)
db.inventory.find({ item: /^p/ })
```

4.索引

4.1 查看执行计划

```
db.user_info.find({age:29}).explain()
```

4.2 创建索引

```
db.user_info.createIndex(
  {
     age: 1
  },
  {
     background: true
  }
)
```

4.3 查看索引

```
db.user_info.getIndexes()
```

4.4 索引信息关键词

```
COLLSCAN  全表扫描
IXSCAN    索引扫描
```

4.5 删除索引

```
db.user_info.dropIndex("age_1")
```

4.6 其他索引类型

```
COLLSCAN — Collection scan
IXSCAN — Scan of data in index keys
FETCH — Retrieving documents
SHARD_MERGE — Merging results from shards
SORT — Explicit sort rather than using index orde
```

5.删除

5.1 删除单条

```
db.inventory.find({status:"P"})
db.inventory.deleteOne({删除条件})
db.inventory.deleteOne({status:"P"})
```

5.2 删除多条

```
db.inventory.deleteMany({status:"P"})
```

5.3 删除索引

```
db.user_info.dropIndex("age_1")
```

5,4 删除集合

```
db.user_info.drop()
```

5.5 删除库

```
db.dropDatabase()
```

第9章 基于角色的访问控制

1.官网介绍

```
https://docs.mongodb.com/manual/core/authorization/
https://docs.mongodb.com/manual/tutorial/enable-authentication/
https://docs.mongodb.com/manual/reference/built-in-roles/
```

2.与用户相关的命令

```
db.auth() 将用户验证到数据库。
db.changeUserPassword() 更改现有用户的密码。
db.createUser() 创建一个新用户。
db.dropUser() 删除单个用户。
db.dropAllUsers() 删除与数据库关联的所有用户。
db.getUser() 返回有关指定用户的信息。
db.getUsers() 返回有关与数据库关联的所有用户的信息。
db.grantRolesToUser() 授予用户角色及其特权。
db.removeUser() 已过时。从数据库中删除用户。
db.revokeRolesFromUser() 从用户中删除角色。
db.updateUser() 更新用户数据。
```

3.在未开启用户访问控制的实例下创建管理员账户

```
use admin
db.createUser(
  {
    user: "myUserAdmin",
    pwd: "123456",
    roles: [ { role: "userAdminAnyDatabase", db: "admin" }, "readWriteAnyDatabase" ]
  }
)
```

4.查看创建的用户

```
> db.getUsers()
[
        {
                "_id" : "admin.myUserAdmin",
                "userId" : UUID("bd30449e-6147-41b4-9af7-10c7f4174944"),
```

```
                "user" : "myUserAdmin",
                "db" : "admin",
                "roles" : [
                        {
                                "role" : "userAdminAnyDatabase",
                                "db" : "admin"
                        },
                        {
                                "role" : "readWriteAnyDatabase",
                                "db" : "admin"
                        }
                ],
                "mechanisms" : [
                        "SCRAM-SHA-1",
                        "SCRAM-SHA-256"
                ]
        }
]
```

5.配置访问控制

```
vim /opt/mongo_27017/conf/mongodb.conf
security:
    authorization: enabled
```

6.重启mongo

```
systemctl restart mongod
```

7.使用admin登陆

```
mongo --authenticationDatabase "admin" -u "myUserAdmin" -p
```

8.使用admin账户创建普通账户

```
use test
db.createUser(
  {
    user: "myTester",
    pwd:  "123456",
    roles: [ { role: "readWrite", db: "db1" },
             { role: "read", db: "db2" } ]
  }
)
db.getUsers()
```

9.使用admin账户创建测试数据

```
use db1
db.write.insertOne({"name":"readWrite"})

use db2
db.read.insertOne({"name":"read"})
```

10.退出admin账户，使用test账户登陆

```
mongo --authenticationDatabase "test" -u "myTester" -p
show dbs
```

11.验证普通账户权限

```
show dbs
use db1
show tables
db.write.find()                         #正常读
db.write.insertOne({name:"ok"})         #正常写

use db2
show tables
db.read.find()                          #正常读
db.read.insertOne({name:"ok"})          #不能写
```

12.使用admin用户修改普通用户权限并创建测试语句

```
mongo --authenticationDatabase "admin" -u "myUserAdmin" -p
use test
db.getUsers()
db.updateUser(
  "myTester",
  {
    roles: [ { role: "read", db: "db1" },
             { role: "readWrite", db: "db2" },
             { role: "readWrite", db: "test" }]
  }
)
db.getUsers()
db.user.insert({name:"test"})
```

13.切换普通账户登陆并测试

```
mongo --authenticationDatabase "test" -u "myTester" -p
```

```
show dbs
use db1
show tables
db.write.find()                     #正常读
db.write.insertOne({name:"ok"})     #不可写

use db2
show tables
db.read.find()                      #正常读
db.read.insertOne({name:"ok"})      #正常写

use test
show tables
db.user.find()                      #正常读
db.user.insertOne({name:"ok"})      #正常写
```

14.删除用户

```
mongo --authenticationDatabase "admin" -u "myUserAdmin" -p
use test
db.dropUser("myTester")
```

第10章 mongo副本集-replica set

1.官网地址

```
https://docs.mongodb.com/manual/replication
https://docs.mongodb.com/manual/reference/method/rs.initiate/#rs.initiate
```

2.副本集角色

```
主节点        负责读写
副本节点      同步主节点
仲裁节点      不是必需的，不存储数据，不参与竞主，只投票，不消耗什么资源
```

3.选举机制

```
大多数投票原则，存活的节点必须是副本集一半以上的数量
```

4.创建目录

```
mkdir -p /opt/mongo_2801{7,8,9}/{conf,log,pid}
mkdir -p /data/mongo_2801{7,8,9}
```

5.创建配置文件

```
cat >/opt/mongo_28017/conf/mongodb.conf <<EOF
systemLog:
 destination: file
 logAppend: true
 path: /opt/mongo_28017/log/mongodb.log

storage:
 journal:
   enabled: true
 dbPath: /data/mongo_28017
 directoryPerDB: true
 wiredTiger:
    engineConfig:
       cacheSizeGB: 0.5
       directoryForIndexes: true
    collectionConfig:
       blockCompressor: zlib
    indexConfig:
       prefixCompression: true

processManagement:
 fork: true
 pidFilePath: /opt/mongo_28017/pid/mongod.pid

net:
 port: 28017
 bindIp: 127.0.0.1,10.0.0.51

replication:
   oplogSizeMB: 1024
   replSetName: dba
EOF
```

6.复制配置文件到其他节点

```
cp /opt/mongo_28017/conf/mongodb.conf /opt/mongo_28018/conf/
cp /opt/mongo_28017/conf/mongodb.conf /opt/mongo_28019/conf/
```

7.替换端口号

```
sed -i 's#28017#28018#g' /opt/mongo_28018/conf/mongodb.conf
sed -i 's#28017#28019#g' /opt/mongo_28019/conf/mongodb.conf
```

8.启动所有节点

```
mongod -f /opt/mongo_28017/conf/mongodb.conf
mongod -f /opt/mongo_28018/conf/mongodb.conf
mongod -f /opt/mongo_28019/conf/mongodb.conf
```

9.检查服务

```
ps -ef|grep mongo
netstat -lntup|grep mongo
```

10.初始化集群

```
mongo --port 28017
rs.initiate(
   {
      _id: "dba",
      version: 1,
      members: [
         { _id: 0, host : "10.0.0.51:28017" },
         { _id: 1, host : "10.0.0.51:28018" },
         { _id: 2, host : "10.0.0.51:28019" }
      ]
   }
)
```

11.主库插入数据测试

```
mongo --port 28017
db.inventory.insertMany([
   { item: "journal", qty: 25, size: { h: 14, w: 21, uom: "cm" }, status: "A" },
   { item: "notebook", qty: 50, size: { h: 8.5, w: 11, uom: "in" }, status: "A" },
   { item: "paper", qty: 100, size: { h: 8.5, w: 11, uom: "in" }, status: "D" },
   { item: "planner", qty: 75, size: { h: 22.85, w: 30, uom: "cm" }, status: "D" },
   { item: "postcard", qty: 45, size: { h: 10, w: 15.25, uom: "cm" }, status: "A" }
]);
```

12.设置副本节点可读

方法1:临时生效

```
rs.slaveOk()
```

方法2:写入启动文件

```
echo "rs.slaveOk()" > ~/.mongorc.js
```

13.副本集常用命令

```
rs.config()
rs.status()
rs.isMaster()
rs.printReplicationInfo()
rs.printSlaveReplicationInfo()
```

第11章 模拟故障转移和权重调整

1.模拟故障转移

```
mongod -f /opt/mongo_28017/conf/mongodb.conf --shutdown
```

2.查看副本集信息

```
rs.status()
rs.config()
```

3.设置权重

```
myconfig=rs.conf()
myconfig.members[0].priority=100
rs.reconfig(myconfig)
```

4.主动降级

```
rs.stepDown()
```

5.恢复默认权重

```
myconfig=rs.conf()
myconfig.members[0].priority=1
rs.reconfig(myconfig)
```

第12章 增加节点和删除节点

1.创建新节点

```
mkdir -p /opt/mongo_28010/{conf,log,pid}
mkdir -p /data/mongo_28010
cp /opt/mongo_28017/conf/mongodb.conf /opt/mongo_28010/conf/
sed -i 's#28017#28010#g' /opt/mongo_28010/conf/mongodb.conf
mongod -f /opt/mongo_28010/conf/mongodb.conf
mongo --port 28010
```

2.集群加入新节点

```
rs.add("10.0.0.51:28010")
```

3.删除节点

```
rs.remove("10.0.0.51:28010")
rs.status()
```

第13章 仲裁节点

1.创建新节点并启动

```
mkdir -p /opt/mongo_28011/{conf,log,pid}
mkdir -p /data/mongo_28011
cp /opt/mongo_28017/conf/mongodb.conf /opt/mongo_28011/conf/
sed -i 's#28017#28011#g' /opt/mongo_28011/conf/mongodb.conf
mongod -f /opt/mongo_28011/conf/mongodb.conf
mongo --port 28011
```

2.将仲裁节点加入副本集

```
rs.addArb("10.0.0.51:28011")
```

3.登陆仲裁节点查看

```
mongo --port 28011
```

4.模拟故障转移结论

```
4个节点+1仲裁  允许坏2台机器
```

第14章 mongo状态查看工具

1.命令介绍

```
mongo        #客户端连接工具
mongod         #启动命令
mongos         #分片路由命令
mongostat      #查看mongo运行状态
mongotop       #查看mongo运行状态
mongodump      #备份
mongoexport          #备份
mongoimport          #恢复
mongorestore         #恢复
bsondump       #将bson格式导出json格式
```

2.mongostat使用说明

使命命令

```
mongostat --port 28018 -o vsize,res --humanReadable=false --noheaders -n 1
```

参数解释

```
--humanReadable=false    #将G转换为K
--noheaders            #不输出首行标题
-n 1               #只输出一次
```

3.mongostat字段解释说明

```
insert/s ： 官方解释是每秒插入数据库的对象数量，如果是slave，则数值前有*,则表示复制集操作
query/s ： 每秒的查询操作次数
update/s ： 每秒的更新操作次数
delete/s ： 每秒的删除操作次数
getmore/s: 每秒查询cursor(游标)时的getmore操作数
command: 每秒执行的命令数，在主从系统中会显示两个值(例如 3|0),分表代表 本地|复制 命令
注： 一秒内执行的命令数比如批量插入，只认为是一条命令（所以意义应该不大）
dirty: 仅仅针对WiredTiger引擎，官网解释是脏数据字节的缓存百分比
used:仅仅针对WiredTiger引擎，官网解释是正在使用中的缓存百分比
flushes:
For WiredTiger引擎：指checkpoint的触发次数在一个轮询间隔期间
For MMAPv1 引擎：每秒执行fsync将数据写入硬盘的次数
注：一般都是0，间断性会是1， 通过计算两个1之间的间隔时间，可以大致了解多长时间flush一次。flush开销是很大的，如果频繁的flush，可能就要找找原因了
vsize： 虚拟内存使用量，单位MB （这是 在mongostat 最后一次调用的总数据）
res： 物理内存使用量，单位MB （这是 在mongostat 最后一次调用的总数据）
注：这个和你用top看到的一样，vsize一般不会有大的变动， res会慢慢的上升，如果res经常突然下降，去查查是否有别的程序狂吃内存。

```

```
qr：客户端等待从MongoDB实例读数据的队列长度
qw：客户端等待从MongoDB实例写入数据的队列长度
ar：执行读操作的活跃客户端数量
aw：执行写操作的活客户端数量
注：如果这两个数值很大，那么就是DB被堵住了，DB的处理速度不及请求速度。看看是否有开销很大的慢查询。如果查询一切正常，确实是负载很大，就需要加机器了
netIn:MongoDB实例的网络进流量
netOut：MongoDB实例的网络出流量
注：此两项字段表名网络带宽压力，一般情况下，不会成为瓶颈
conn：打开连接的总数，是qr,qw,ar,aw的总和
注：MongoDB为每一个连接创建一个线程，线程的创建与释放也会有开销，所以尽量要适当配置连接数的启动参数，maxIncomingConnections，阿里工程师建议在5000以下，基本满足多数场景
```

4.案例，找出占用资源大的操作

写入循环命令

```
use oldboy
for(i=1;i<10000;i++){ db.cook.insert({"id":i,"name":"BJ","age":70,"date":new Date()}); }
```

使用mongotop和mongostat观察

```
mongostat
mongotop
```

第15章 备份与恢复

1.工具介绍

```
mongodump/mongorestore
mongoexport/mongoimport
```

2.应用场景

```
定时备份，全量备份 mongodump/mongorestore bson gzip
分析数据，迁移数据 mongoexport/mongoimport json csv
```

3.mongodump备份单点数据

全备数据库

```
mongodump --port 28017 -o mongo_backup
mongodump --host="dba/10.0.0.51:28017,10.0.0.51:28018,10.0.0.51:28019" -o mongo_backup
```

只备份某个数据库

```
mongodump --host="dba/10.0.0.51:28017,10.0.0.51:28018,10.0.0.51:28019" -d oldboy -o mongo_backup
```

只备份某个库下的某个集合

```
mongodump --host="dba/10.0.0.51:28017,10.0.0.51:28018,10.0.0.51:28019" -d oldboy -c user_info -o mongo_backup
```

压缩格式

```
mongodump --host="dba/10.0.0.51:28017,10.0.0.51:28018,10.0.0.51:28019" -o mongo_backup --gzip
```

4.mongorestore恢复

恢复bson格式的数据

```
mongorestore --host="dba/10.0.0.51:28017,10.0.0.51:28018,10.0.0.51:28019" mongo_backup
```

恢复gzip格式的数据

```
mongorestore --host="dba/10.0.0.51:28017,10.0.0.51:28018,10.0.0.51:28019" mongo_backup --gzip
```

遇到重复的删除再导入

```
mongorestore --host="dba/10.0.0.51:28017,10.0.0.51:28018,10.0.0.51:28019" mongo_backup --gzip --drop
```

模拟执行

```
mongorestore --host="dba/10.0.0.51:28017,10.0.0.51:28018,10.0.0.51:28019" mongo_backup --gzip --drop --dryRun
```

恢复到指定库

```
mongorestore --host="dba/10.0.0.51:28017,10.0.0.51:28018,10.0.0.51:28019"  --dir=./mongo_backup/oldboy -d oldboy --drop --gzip
```

恢复到指定集合: 恢复到指定集合那么数据格式必须是bson格式

```
mongorestore --host="dba/10.0.0.51:28017,10.0.0.51:28018,10.0.0.51:28019"  --dir=./mongo_backup/oldboy/cook.bson -d oldboy -c cook --drop
```

5.bsog格式转换成json格式

```
bsondump --outFile=cook.json cook.bson
```

6.mongoexport-导出成json和csv

导出指定集合为json格式

```
mongoexport --host="dba/10.0.0.51:28017,10.0.0.51:28018,10.0.0.51:28019" -d test -c user_info -o mongo_backup/test.user_info.json
```

导出成csv格式

```
mongoexport --host="dba/10.0.0.51:28017,10.0.0.51:28018,10.0.0.51:28019" -d test -c user_info --type=csv --fields=name,age,host -o mongo_backup/test.user_info.csv
```

7.mongoimport-恢复数据

从json格式恢复数据

```
mongoimport --host="dba/10.0.0.51:28017,10.0.0.51:28018,10.0.0.51:28019" -d test -c user_info mongo_backup/test.user_info.json --drop
```

从csv格式恢复数据

```
mongoimport --host="dba/10.0.0.51:28017,10.0.0.51:28018,10.0.0.51:28019" --type=csv --headerline -d test -c user_info mongo_backup/test.user_info.csv --drop
```

第17章 mongo异构数据迁移

1.mysql自定义分隔符导出成csv格式

```
select * from world.city into outfile '/var/lib/mysql/city.csv' fields terminated by ',';
```

2.手动添加CSV头部

```
ID,Name,CountryCode,District,Population
```

3.mongo导入csv格式

```
mongoimport --host="dba/10.0.0.51:28017,10.0.0.51:28018,10.0.0.51:28019" --type=csv --headerline -d world -c city /var/lib/mysql/city.csv --drop
```

4.CSV导入ES

背景：老大发给运维几个csv格式的文件，需要导入到ES里 elasticsearch

```
csv --> mongo
mongo --> json
json --> es
```

操作命令：

```
mongoimport --host="dba/10.0.0.51:28017,10.0.0.51:28018,10.0.0.51:28019" --type=csv --headerline -d world -c city /var/lib/mysql/city.csv --drop
mongoexport --host="dba/10.0.0.51:28017,10.0.0.51:28018,10.0.0.51:28019" -d world -c city -o mongo_backup/world.city.json
```

第18章 模拟误删库恢复

0.前提

```
1.只有副本集才会有oplog
2.oplog保存在local库里
3.mongodump的时候，默认不备份local库
```

1.模拟场景

```
1
2
3
全备
4
5
6
drop databases
```

2.准备数据

```
use backup
db.backup.insertOne({"id":1})
db.backup.insertOne({"id":2})
db.backup.insertOne({"id":3})
```

3.全备数据库

```
rm -rf mongo_backup/
mongodump --host="dba/10.0.0.51:28017,10.0.0.51:28018,10.0.0.51:28019" -o mongo_backup
```

4.写入增量数据

```
use backup
db.backup.insertOne({"id":4})
db.backup.insertOne({"id":5})
db.backup.insertOne({"id":6})
```

5.模拟误删除

```
use backup
db.dropDatabase()
```

6.备份oplog

```
mongodump --host="dba/10.0.0.51:28017,10.0.0.51:28018,10.0.0.51:28019" -d local -c oplog.rs -o mongo_backup
```

7.定位误删除操作的时间点

```
use local
db.oplog.rs.find({"ns" : "backup.$cmd"}).pretty()
```

8.找到误删除的位置点

```
"ts" : Timestamp(1587375719, 1)
```

9.截断oplog的drop时间点，然后恢复

```
cd mongo_backup
cp local/oplog.rs.bson oplog.bson
rm -rf local

mongorestore --host="dba/10.0.0.51:28017,10.0.0.51:28018,10.0.0.51:28019" --oplogReplay --oplogLimit=1587375719 mongo_backup --drop
```

10.深圳5期全体思路

```
1.第一天写入数据
use backup
db.backup.insertOne({"id":1})
db.backup.insertOne({"id":2})
db.backup.insertOne({"id":3})

2.第一天全备数据
rm -rf mongo_backup/
mongodump --host="dba/10.0.0.51:28017,10.0.0.51:28018,10.0.0.51:28019" -o mongo_backup
mongodump --host="dba/10.0.0.51:28017,10.0.0.51:28018,10.0.0.51:28019" -d local -o mongo_backup

3.第二天写入增量数据
use backup
db.backup.insertOne({"id":4})
db.backup.insertOne({"id":5})
db.backup.insertOne({"id":6})

4.第二天删库
use backup
db.dropDatabase()

5.查找误删库相关的时间点,找到上一次全备之后的时间点和误删除的时间点
use local
db.oplog.rs.find({"ns" : "backup.$cmd"}).pretty()

终点:
1600965303

6.切割oplog,找出上次全备到现在为止的数据
cd mongo_backup/local
bsondump --bsonFile=oplog.rs.bson --outFile=oplog.json
tail -1 oplog.json
起点:1600965264

mongodump \
--host="dba/10.0.0.51:28017,10.0.0.51:28018,10.0.0.51:28019" \
-d local \
```

```
-c oplog.rs \
-q '{ts:{$gt:Timestamp(1600965264,1)}}' \
-o mongo_backup_oplog

7.移动到全备目录里
cd mongo_backup
cp /root/mongo_backup_oplog/local/oplog.rs.bson /root/mongo_backup/oplog.bson
rm -rf local

8.在测试数据库恢复数据
mongorestore \
--host="dba/10.0.0.51:28017,10.0.0.51:28018,10.0.0.51:28019" \
--oplogReplay \
--oplogLimit=1600965303 \
./mongo_backup --drop

9.把恢复后的数据导出成json格式

10.把恢复的json文件导入到生产数据库

11.立刻做一次全备
```

第19章 生产中oplog配置多大

1.生产中需要设置多大?

```
1.差不多两个全备之间，相差时间内的数据量的大小
2.更改给定副本集成员的oplog大小或最小oplog保留期（在4.4中是新的）replSetResizeOplog不会改变副本集中任何其他成员的oplog大小。您必须replSetResizeOplog在集群中的每个副本集成员上运行，才能更改所有成员的操作日志大小或最小保留期限
```

2.官网地址

```
https://docs.mongodb.com/manual/core/replica-set-oplog/index.html
https://docs.mongodb.com/manual/reference/command/replSetResizeOplog/#dbcmd.replSetResizeOplog
```

3.相关命令

```
db.printReplicationInfo()
db.getReplicationInfo()

use admin
db.adminCommand(
  {
    replSetResizeOplog: 1,
    size: 2048,
    minRetentionHours: 24
  }
)
```