

MongoDB 集群搭建和使用

Name : 曲中岭

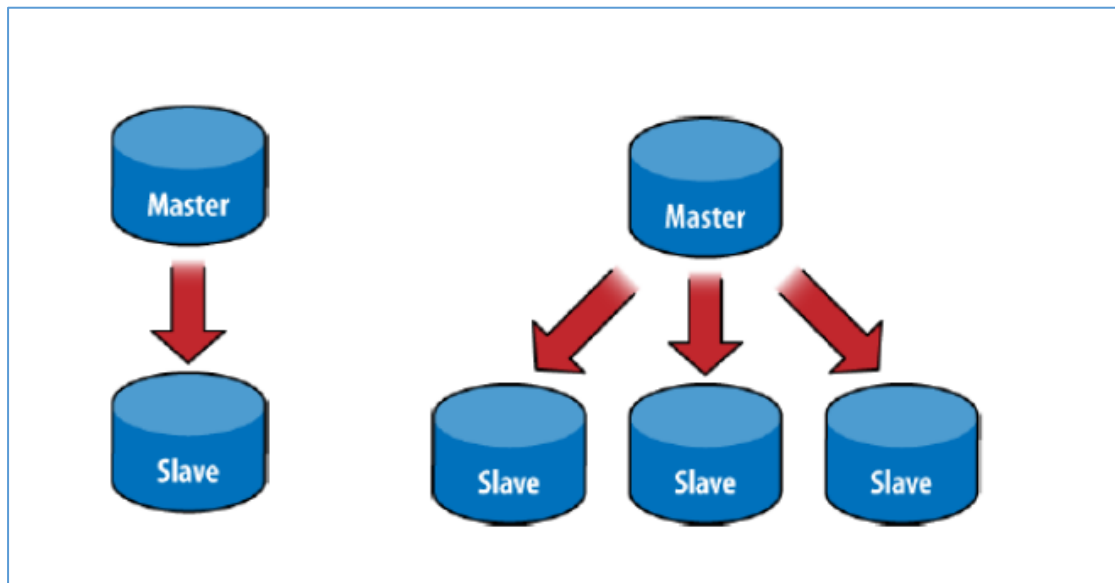
Email: zlingqu@126.com

Q Q :441869115

MongoDB 有三种集群方式，分别对应第 1-3 章。

第一章 master-slave 集群

1.1 拓扑图



主从架构一般用于备份或者做读写分离。一般有一主一从或一主多从设计。

优点：

- 1) 主节点，可读可写，从节点可读不可写，实现了读写分离
- 2) 支持一主多从，大大缓解读的压力，适用于读操作比较多的场景

缺点：

- 1) 不能实现自动主从切换，即主挂后，从不能自动成为主，集群只可以读，不可写
- 2) 不支持链式结构，即 Slave 只能直接连 Master。而不像 redis 或者 mysql 的，slave 可从另一个 slave 同步。
- 3) 数据量大的情况下，主节点可能成为性能瓶颈

1.2 测试规划

1.2.1 规划清单

OS: CentOS 6.6 x64

MongoDB: 3.6.7 (4.0 之后不再支持主从模式, 后面有介绍)

Master: 10.1.5.201

Slave1: 10.1.5.202

Slave2: 10.1.5.203

数据目录: /mongodb-data/ (自定义)

日志文件: /var/log/mongodb/mongod.log (默认)

PID 文件: /var/run/mongodb/mongod.pid (默认)

PORT: 27017 (默认)

监听地址: 0.0.0.0 (自定义)

1.2.2 版本兼容

注意: mongodb4.0.1 不再支持主从模式的集群, 可见发行说明文档:

<https://docs.mongodb.com/manual/release-notes/4.0-compatibility/>

Remove Master-Slave Replication

MongoDB 4.0 removes support for the deprecated master-slave replication. Before you can upgrade to MongoDB 4.0, if your deployment uses master-slave replication, you must upgrade to a replica set.

安装了 4 之后, 通过查询帮助, 也得到类似的结论:

```
[root@hadoop01 ~]#  
[root@hadoop01 ~]# mongod --version  
db version v4.0.1  
git version: 54f1582fc6eb01de4d4c42f26fc133e623f065fb  
openssl version: OpenSSL 1.0.1e-fips 11 Feb 2013  
allocator: tcmalloc  
modules: none  
build environment:  
  distmod: rhel62  
  distarch: x86_64  
  target_arch: x86_64  
[root@hadoop01 ~]#  
[root@hadoop01 ~]#  
[root@hadoop01 ~]# mongod -h|grep -A3 master  
--master      Master/slave replication no longer  
              supported  
--slave      Master/slave replication no longer  
             supported  
[root@hadoop01 ~]#
```

1.3 安装和部署

master、slave1、slave2 上需要执行如下步骤。

1.3.1 添加 yum 源

```
[root@hadoop01 ~]# cat /etc/yum.repos.d/Mongodb-3.6.repo
```

```
[mongodb-org-3.6]
name = MongoDB Repository
baseurl = http://repo.mongodb.org/yum/redhat/$releasever/mongodb-org/3.6/x86_64/
gpgcheck = 0
enabled = 1
```

1.3.2 安装依赖包

```
yum install openssl libcurl
```

1.3.3 安装软件包

```
yum install mongodb-org
```

将自动安装以下 4 个依赖包：

mongodb-org-server 包含 [mongod](#) 守护程序以及关联的配置和 init 脚本。

mongodb-org-mongos 包含 [mongos](#) 守护进程。

mongodb-org-shell 包含 [mongoshell](#)。

mongodb-org-tools 包含以下的 MongoDB 工具：

[mongoimport](#) [bsondump](#), [mongodump](#), [mongoexport](#),, [mongofiles](#),
[mongoperf](#), [mongorestore](#), [mongostat](#) and [mongotop](#).

依赖关系解决		
软件包	架构	版本
正在安装:		
mongodb-org	x86_64	3.6.7-1.el6
为依赖而安装:		
mongodb-org-mongos	x86_64	3.6.7-1.el6
mongodb-org-server	x86_64	3.6.7-1.el6
mongodb-org-shell	x86_64	3.6.7-1.el6
mongodb-org-tools	x86_64	3.6.7-1.el6
事务概要		
Install	5 Package(s)	
总下载量: 91 M		
Installed size: 265 M		
确定吗? [y/N]: y		

1.3.4 数据目录准备

创建数据目，并修改权限

```
mkdir /mongodb-data
chown mongod.mongod /mongodb-data/
```

1.4 修改配置文件

1.4.1 /etc/mongod.conf

master、slave1、slave2 上需要执行这一步。

默认配置文件内容如下：

```
# where to write logging data.
systemLog:
  destination: file
  logAppend: true
  path: /var/log/mongodb/mongod.log

# where and how to store data.
storage:
  dbPath: /var/lib/mongo
  journal:
    enabled: true
# engine:
# mmapv1:
# wiredTiger:

# how the process runs
processManagement:
  fork: true # fork and run in background
  pidFilePath: /var/run/mongodb/mongod.pid # location of
  timeZoneInfo: /usr/share/zoneinfo

# network interfaces
net:
  port: 27017
  bindIp: 127.0.0.1 # Enter 0.0.0.0,:: to bind to all IPv
```

配置文件使用分段形式配置，我们修改 dbpath 和 bindIP 选项，可使用如下语句快速修改：

```
sed -i 's#dbPath: /var/lib/mongo#dbPath: /mongodb-data#g' /etc/mongod.conf
sed -i 's#bindIp: 127.0.0.1#bindIp: 0.0.0.0#g' /etc/mongod.conf
```

1.4.2 /etc/init.d/mongod

master 节点执行：

vim /etc/init.d/mongod

```
# NOTE: if you change any OPTIONS here, you get wha
# this script assumes all options are in the config
CONFIGFILE="/etc/mongod.conf"
OPTIONS="-f $CONFIGFILE"
mongod=${MONGOD-/usr/bin/mongod}
```

修改为：

```
# NOTE: if you change any OPTIONS here, you get what you pay for
# this script assumes all options are in the config file.
CONFIGFILE="/etc/mongod.conf"
OPTIONS="-f $CONFIGFILE --slave --source 10.1.5.201:27017"
mongod=${MONGOD-/usr/bin/mongod}
```

可使用如下语句，快速修改：

```
sed -i 's/OPTIONS=" -f $CONFIGFILE"/OPTIONS=" -f $CONFIGFILE --slave --source 10.1.5.201:27017"/g' /etc/init.d/mongod
```

slave1、slave2 节点执行：

```
# NOTE: if you change any OPTIONS here, you get what you pay for
# this script assumes all options are in the config file.
CONFIGFILE="/etc/mongod.conf"
OPTIONS="-f $CONFIGFILE"
mongod=${MONGOD-/usr/bin/mongod}
```

修改为：

```
# NOTE: if you change any OPTIONS here, you get what you pay for
# this script assumes all options are in the config file.
CONFIGFILE="/etc/mongod.conf"
OPTIONS="-f $CONFIGFILE --master"
mongod=${MONGOD-/usr/bin/mongod}
```

可使用如下语句，快速修改：

```
sed -i 's/OPTIONS=" -f $CONFIGFILE"/OPTIONS=" -f $CONFIGFILE --master"/g' /etc/init.d/mongod
```

1.5 启动服务

master、slave1、slave2 上需要执行这一步，其中 master 节点需要先启动。

```
service mongod start
chkconfig mongod on #添加开机启动，默认已开启
```

1.6 测试

1.6.1 观察日志

```
less /var/log/mongodb/mongod.log
```

```

2018-08-22T11:51:32.874+0800 I CONTROL [initandlisten] target_arch: x86_64
2018-08-22T11:51:32.874+0800 I CONTROL [initandlisten] options: { config: "/etc/mongod.conf", net: { bindIp: "0.0.0.0", port: 27017 }, processManagement: { fork: true,
pidFilePath: "/var/run/mongod/mongod.pid", timeZoneInfo: "/usr/share/zoneinfo" }, slave: true, source: "10.1.5.201:27017", storage: { dbPath: "/mongodb-data", jour
al: { enabled: true } }, systemLog: { destination: "file", logAppend: true, path: "/var/log/mongodb/mongod.log" } }
2018-08-22T11:51:32.885+0800 I - [initandlisten] Detected data files in /mongodb-data created by the 'wiredTiger' storage engine, so setting the active storage
engine to wiredTiger
2018-08-22T11:51:32.885+0800 I STORAGE [initandlisten] ** WARNING: using the XFS filesystem is strongly recommended with the wiredTiger storage engine
2018-08-22T11:51:32.885+0800 I STORAGE [initandlisten] ** See http://dochub.mongodb.org/core/prodnotes-filesystem
2018-08-22T11:51:32.886+0800 I STORAGE [initandlisten] wiredtiger_open config: create,cache_size=1403M,session_max=20000,eviction=(threads_min=4,threads_max=4),confi
_base=false,statistics=(fast),cache_cursors=false,compatibility=(release="3.0",require_max="3.0"),log=(enabled=true,archive=true,path=journal,compressor=snappy),file_
manager=(close_idle_time=10000),statistics_log=(wait=0,verbose=(recovery_progress))
2018-08-22T11:52:53.109+0800 I STORAGE [initandlisten] WiredTiger message [1534909973:109020][2417:0x7f67456bea80], txn-recover: Main recovery loop: starting at 1/26
68
2018-08-22T11:52:53.373+0800 I STORAGE [initandlisten] WiredTiger message [1534909973:373652][2417:0x7f67456bea80], txn-recover: Recovering log 1 through 2
2018-08-22T11:52:53.811+0800 I STORAGE [initandlisten] WiredTiger message [1534909973:811781][2417:0x7f67456bea80], txn-recover: Recovering log 2 through 2
2018-08-22T11:52:53.878+0800 I STORAGE [initandlisten] WiredTiger message [1534909973:878785][2417:0x7f67456bea80], txn-recover: Set global recovery timestamp: 0
2018-08-22T11:52:54.186+0800 I CONTROL [initandlisten] ** WARNING: Access control is not enabled for the database.
2018-08-22T11:52:54.186+0800 I CONTROL [initandlisten] Read and write access to data and configuration is unrestricted.
2018-08-22T11:52:54.187+0800 I CONTROL [initandlisten]
2018-08-22T11:52:54.197+0800 I CONTROL [initandlisten] ** WARNING: /sys/kernel/mm/transparent_hugepage/enabled is 'always'.
2018-08-22T11:52:54.197+0800 I CONTROL [initandlisten] we suggest setting it to 'never'
2018-08-22T11:52:54.197+0800 I CONTROL [initandlisten]
2018-08-22T11:52:54.197+0800 I CONTROL [initandlisten] ** WARNING: /sys/kernel/mm/transparent_hugepage/defrag is 'always'.
2018-08-22T11:52:54.197+0800 I CONTROL [initandlisten] we suggest setting it to 'never'
2018-08-22T11:52:54.198+0800 I CONTROL [initandlisten]
2018-08-22T11:52:54.198+0800 I CONTROL [initandlisten] ** WARNING: soft rlimits too low. rlimits set to 1024 processes, 64000 files. Number of processes should be at
least 32000 : 0.5 times number of files.
2018-08-22T11:52:54.198+0800 I CONTROL [initandlisten]
2018-08-22T11:52:54.198+0800 I CONTROL [initandlisten] ** WARNING: This node was started in master-slave replication mode.
2018-08-22T11:52:54.198+0800 I CONTROL [initandlisten] Master-slave replication is deprecated and subject to be removed
2018-08-22T11:52:54.198+0800 I CONTROL [initandlisten] in a future version.

```

从启动日志中我们可以得到如下信息:

启动参数, 比如 slave 的参数

```

2018-08-22T11:51:32.874+0800 I CONTROL [initandlisten] options: { config: "/etc/mongod.conf",
net: { bindIp: "0.0.0.0", port: 27017 }, processManagement: { fork: true, pidFilePath: "/var/
run/mongod/mongod.pid", timeZoneInfo: "/usr/share/zoneinfo" }, slave: true, source: "10.1.5.2
01:27017", storage: { dbPath: "/mongodb-data", journal: { enabled: true } }, systemLog: { dest
ination: "file", logAppend: true, path: "/var/log/mongodb/mongod.log" } }

```

```

options:
{
  config: "/etc/mongod.conf",
  net:
  {
    bindIp: "0.0.0.0",
    port: 27017
  },
  processManagement:
  {
    fork: true,
    pidFilePath: "/var/run/mongod/mongod.pid",
    timeZoneInfo: "/usr/share/zoneinfo"
  },
  slave: true,
  source: "10.1.5.201:27017",
  storage:
  {
    dbPath: "/mongodb-data",
    journal: { enabled: true }
  },
  systemLog:
  {
    destination: "file",
    logAppend: true,
    path: "/var/log/mongodb/mongod.log"
  }
}

```

1.6.2 排错

- 1) 警告 1, 告诉我们最好使用 XFS 的文件系统, 这个我们是测试环境, 暂时忽略, 但生产环境中, 建议格式化成 XFS, 可以提高性能

```
** WARNING: Using the XFS filesystem is strongly recommended with the wiredTiger storage engine
** See http://dochub.mongodb.org/core/prodnotes-filesystem
```

- 2) 警告 2, 告诉我们没有启用访问控制, 这个自己评估是否需要, 此处测试, 不再启用

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

- 3) 警告 3, 告诉我们启用了大内存页功能, 而 mongodb 的运行环境建议应该取消, 这可以提高性能

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

如果需要修复, 可以执行如下语句:

```
cat >> /etc/rc.local << EOF
echo never > /sys/kernel/mm/transparent_hugepage/enabled
echo never > /sys/kernel/mm/transparent_hugepage/defrag
EOF
```

并将其加入/etc/rc.local

- 5) 警告 5 rlimit 被设置为了 1024, 太小了对于 mongodb 来说

```
2018-08-22T11:52:54.198+0800 I CONTROL [initandlisten] ** WARNING: soft rlimits too low. rlimits set to 1024
processes, 64000 files. Number of processes should be at least 32000 : 0.5 times number of files.
```

如果要修复, 可以使用如下语句:

```
cat >> /etc/security/limits.conf << EOF
* soft nproc 65535
* hard nproc 65535
* soft nofile 65535
* hard nofile 65535
EOF
```

并重新 ssh 服务器后并重启 mongod 服务。

- 6) 警告 6, 告诉我们 master-slave 模式将来会取消支持, 所以知道在 4.0 及其之后已经被取消了

```
WARNING: This node was started in master-slave replication mode.
Master-slave replication is deprecated and subject to be removed
in a future version.
```

1.6.3 观察主从复制

master 节点创建库, 并插入一条数据:

```
[root@hadoop01 ~]# mongo 10.1.5.201:27017
MongoDB shell version v3.6.7
connecting to: mongod://10.1.5.201:27017/test
MongoDB server version: 3.6.7
```

使用 mongos 连接 master

```
> db.printReplicationInfo()
configured oplog size: 2014.310791015625MB
log length start to end: 82072secs (22.8hrs)
oplog first event time: Tue Aug 21 2018 16:10:30 GMT+0800 (CST)
oplog last event time: Wed Aug 22 2018 14:58:22 GMT+0800 (CST)
now: Wed Aug 22 2018 14:58:28 GMT+0800 (CST)
>
> show dbs
admin 0.000GB
config 0.000GB
local 0.000GB
>
> use testdb
switched to db testdb
>
> db.testdb.insert({"name":"quzl"})
writeResult({ "nInserted" : 1 })
>
> show dbs
admin 0.000GB
config 0.000GB
local 0.000GB
testdb 0.000GB
```

命令解释:

db.printReplicationInfo(), 查看主节点信息

oplog first event time: 首次同步时间

oplog last event time: 最后一次同步时间

show dbs, 显示当前所有库

admin、config、local 是系统自带数据库

use testdb, 如果不存在就创建该库, 如果存在就进入该库

db.testdb.insert({"name":"quzl"}), 在 testdb 库中插入一条文档数据

61617 成为 master 后, 使用 8162 端口访问查看, 发现队列仍然存在。

进入 slave 库查看数据是否被同步:

```
[root@hadoop01 ~]# mongo 10.1.5.202:27017
MongoDB shell version v3.6.7
connecting to: mongod://10.1.5.202:27017/test
MongoDB server version: 3.6.7
```

进入 slave1 节点


```

> show dbs
2018-08-22T15:09:11.686+0800 E QUERY [thread1] Error: listDatabases failed:{
  "ok" : 0,
  "errmsg" : "not master and slaveOk=false",
  "code" : 13435,
  "codeName" : "NotMasterNoSlaveOk"
} :
_getErrorWithCode@src/mongo/shell/utils.js:25:13
Mongo.prototype.getDBs@src/mongo/shell/mongo.js:65:1
shellHelper.show@src/mongo/shell/utils.js:849:19
shellHelper@src/mongo/shell/utils.js:739:15
@(shellhelp2):1:1
>
> rs.slaveOk()
>
> show dbs
admin 0.000GB
config 0.000GB
local 0.000GB
testdb 0.000GB

```

第一次使用 show dbs，报 12435 的错误，那是因为 slave 库，默认是不可读的，使用 rs.slaveOk() 设置后就可以读取了。

```

> rs.printSlaveReplicationInfo()
source: 10.1.1.201:27017
syncedTo: Wed Aug 22 2018 15:24:52 GMT+0800 (CST)
5 secs (0 hrs) behind the freshest member (no primary available at the moment)
>

```

使用 rs.printSlaveReplicationInfo() 查看从库，同步状态，即显示出上一次同步的时间。


结论：主从复制正常

1.6.4 从库不可写测试

```

> use testdb2
switched to db testdb2
>
>
> show dbs
admin 0.000GB
config 0.000GB
local 0.000GB
testdb 0.000GB
>
> db.testdb2.insert({"name":"guzl"})
writeResult({ "writeError" : { "code" : 10107, "errmsg" : "not master" } })
>
>
> show dbs
admin 0.000GB
config 0.000GB
local 0.000GB
testdb 0.000GB
>

```



如上图，从库不可写

1.6.5 读写分离配置

情况 1：使用 mongo shell 进入从库

使用如下命令中的任意一条，即可实现从库可读，但只对当前 shell 有效

- 1) db.getMongo().setSlaveOk()
- 2) rs.slaveOk()

情况 2: java 连接 mongodb:

1) 在 java 代码中调用 `dbFactory.getDb().slaveOk()`;

2) 在 java 代码中调用

```
dbFactory.getDb().setReadPreference(ReadPreference.secondaryPreferred());
```

//在复制集中优先读 secondary, 如果 secondary 访问不了的时候就从 master 中读
或

```
dbFactory.getDb().setReadPreference(ReadPreference.secondary());
```

//只从 secondary 中读, 如果 secondary 访问不了的时候就不能进行查询

3) 在配置 mongo 的时候增加 `slave-ok="true"`也支持直接从 secondary 中读

```
<mongo:mongo id="mongo" host="{mongodb.host}" port="{mongodb.port}">
```

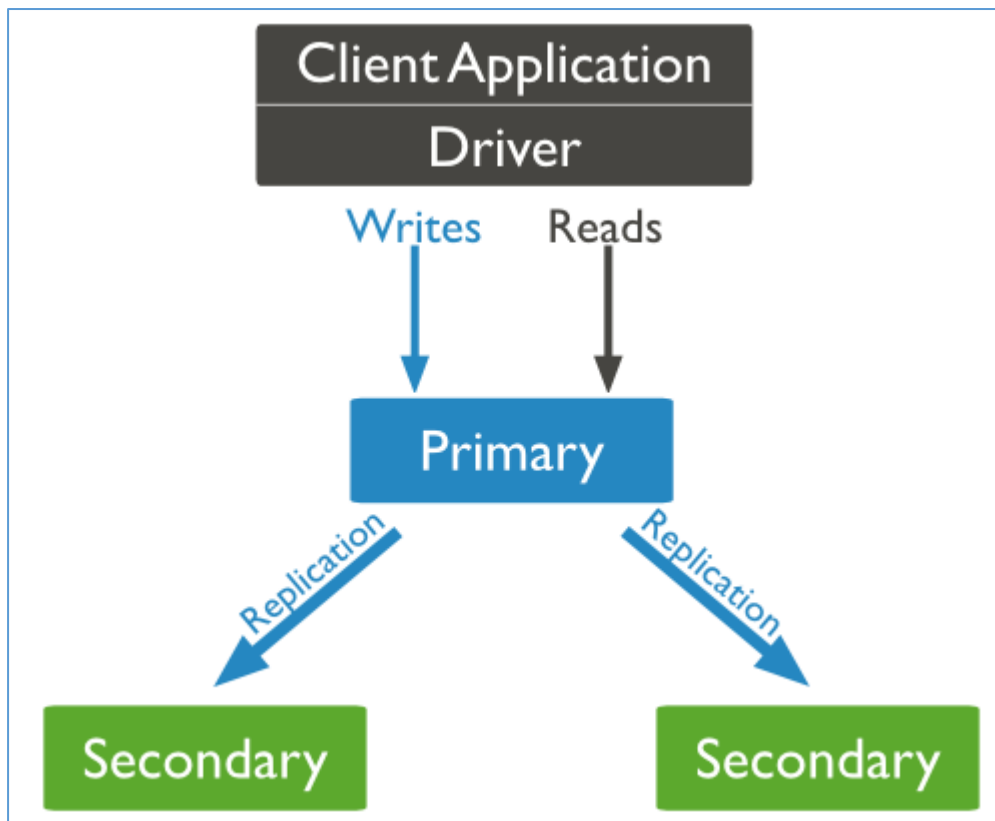
```
    <mongo:options slave-ok="true"/>
```

```
</mongo:mongo>
```

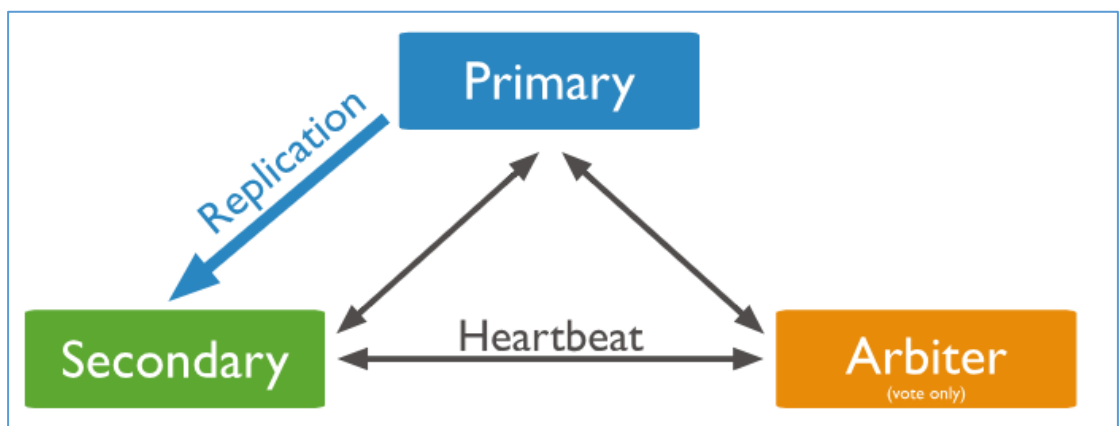
第二章 副本集集群

副本集集群，即 Replica Set 集群，可以是一主一从，也可以是一主多从，但为了保证有效性，整个集群节点个数最好是奇数个。

2.1 拓扑图



或者



(1) 主节点 (Primary)

接收所有的读、写请求，然后把修改同步到所有 **Secondary**。一个 **Replica Set** 只能有一个 **Primary** 节点，当 **Primary** 挂掉后，其他 **Secondary** 节点会重新选举出来一个成为主节点。

如果要将读请求发送到从节点，客户端可以指定读取首选项将读操作发送到辅助节点。对辅助节点的异步复制意味着从辅助节点读取可能会返回不反映主节点上数据状态的数据，即可能造成新增或修改的数据，无法获取到最新的状态。

（2）副本节点（**Secondary**）

与主节点保持同样的数据集。当主节点挂掉的时候，参与选主。

（3）仲裁者（**Arbiter**）

不保有数据，不参与选主，只进行选主投票。使用 **Arbiter** 可以减轻数据存储的硬件需求，**Arbiter** 跑起来几乎没什么大的硬件资源需求，如果您的副本集具有偶数个成员，请添加仲裁者以获得主要选举中的大多数投票，但重要的一点是，在生产环境下它和其他数据节点不要部署在同一台机器上。

优点：

- 1) 自动主从切换
- 2) 可配置读写分离

缺点：

- 1) 数据量大的情况下，主节点可能成为性能瓶颈

2.2 测试规划

这里采用 **Primary+ Secondary+ Arbiter** 的架构形式。

OS: CentOS 6.6 x64

MongoDB: 4.0.1

Primary: 10.1.5.201

Secondary: 10.1.5.202

Arbiter: 10.1.5.203

数据目录: /mongodb-data/ （自定义）

日志文件: /var/log/mongodb/mongod.log （默认）

PID 文件: /var/run/mongodb/mongod.pid （默认）

PORT: 27017 （默认）

监听地址: 0.0.0.0 （自定义）

2.3 安装和部署

本节 2.3 的所有操作，在 Primary、Secondary、Arbiter 三个节点都要执行。

2.3.1 添加 yum 源

```
[root@hadoop01 ~]# cat /etc/yum.repos.d/Mongodb-4.repo
cat > /etc/yum.repos.d/Mongodb-4.repo << EOF
[mongodb-org-4.0]
name = MongoDB Repository
baseurl = http://repo.mongodb.org/yum/redhat/$releasever/mongodb-org/4.0/x86_64/
gpgcheck = 0
enabled = 1
EOF
```

2.3.2 安装依赖包

```
yum install openssl libcurl
```

2.3.3 安装软件包

```
yum install mongodb-org
```

将自动安装以下 4 个依赖包：

mongodb-org-server 包含 [mongod](#) 守护程序以及关联的配置和 init 脚本。

mongodb-org-mongos 包含 [mongos](#) 守护进程。

mongodb-org-shell 包含 [mongo](#) shell。

mongodb-org-tools 包含以下的 MongoDB 工具：

[mongoimport](#), [bsondump](#), [mongodump](#), [mongoexport](#), [mongofiles](#),
[mongorestore](#), [mongostat](#) and [mongotop](#).

```

=====
软件包                                架构                                版本
=====
正在安装:
mongodb-org                          x86_64                             4.0.1-1.el6
为依赖而安装:
mongodb-org-mongos                   x86_64                             4.0.1-1.el6
mongodb-org-server                   x86_64                             4.0.1-1.el6
mongodb-org-shell                    x86_64                             4.0.1-1.el6
mongodb-org-tools                    x86_64                             4.0.1-1.el6
=====

事务概要
=====
Install                               5 Package(s)

总下载量: 76 M
Installed size: 232 M
确定吗? [y/N]:

```

2.3.4 数据目录准备

创建数据目录，并修改权限

```
mkdir /mongodb-data  
chown mongod.mongod /mongodb-data/
```

2.4 修改配置文件

本节 2.4 的所有操作，在 Primary、Secondary、Arbiter 三个节点都要执行。

2.4.1 /etc/mongod.conf

默认配置文件内容如下：

```
# where to write logging data.  
systemLog:  
  destination: file  
  logAppend: true  
  path: /var/log/mongodb/mongod.log  
  
# where and how to store data.  
storage:  
  dbPath: /var/lib/mongo  
  journal:  
    enabled: true  
# engine:  
# mmapv1:  
# wiredTiger:  
  
# how the process runs  
processManagement:  
  fork: true # fork and run in background  
  pidFilePath: /var/run/mongodb/mongod.pid # location of  
  timeZoneInfo: /usr/share/zoneinfo  
  
# network interfaces  
net:  
  port: 27017  
  bindIp: 127.0.0.1 # Enter 0.0.0.0,:: to bind to all IPv4 and IPv6
```

配置文件使用分段形式配置，我们修改 dbpath 和 bindIP 选项，可使用如下语句快速修改：

```
sed -i 's#dbPath: /var/lib/mongo#dbPath: /mongodb-data#g' /etc/mongod.conf  
sed -i 's#bindIp: 127.0.0.1#bindIp: 0.0.0.0#g' /etc/mongod.conf
```

2.4.2 /etc/mongod.conf

vim /etc/mongod.conf

```
# network interfaces
net:
  port: 27017
  bindIp: 0.0.0.0 # Enter 0.0.0.0,:: to bind to all IPv

#security:

#operationProfiling:

replication:
  replSetName: "my-rs1"

#sharding:
```

添加如下两行，其中 my-rs1 是副本集的名字，可以修改成其他的

```
replication:
  replSetName: "my-rs1"
```

2.5 启动集群

2.5.1 启动服务

在前面步骤都完成后，使用

```
service mongod start
```

启动三节点的 mongod 服务

2.5.2 初始化副本集-secondary 节点

使用 mongo 进入 201 节点，如果是本机，也可以直接输入 mongo

```
[root@hadoop01 ~]# mongo 10.1.5.201:27017
MongoDB shell version v3.6.7
connecting to: mongodb://10.1.5.201:27017/test
MongoDB server version: 3.6.7
```

使用如下命令初始化，其中注意 my-rs1 于 2.4.2 中配置的要一致，201、202、203 随意设置

```
rs.initiate({
  _id: "my-rs1",
  members: [
    { _id: 201, host: "10.1.5.201:27017" },
    { _id: 202, host: "10.1.5.202:27017" }
  ]
})
```

```

> rs.initiate( {
...   _id : "my-rs1",
...   members: [
...     { _id: 201, host: "10.1.5.201:27017" },
...     { _id: 202, host: "10.1.5.202:27017" }
...   ]
... })
{
  "ok" : 1,
  "operationTime" : Timestamp(1534931616, 1),
  "$clusterTime" : {
    "clusterTime" : Timestamp(1534931616, 1),
    "signature" : {
      "hash" : BinData(0,"AAAAAAAAAAAAAAAAAAAAAAAAAAAA="),
      "keyId" : NumberLong(0)
    }
  }
}
my-rs1:SECONDARY>

```

输入

rs.conf()

可以查看副本集配置信息，目前只有两个节点：

```

my-rs1:PRIMARY> rs.conf()
{
  "_id" : "my-rs1",
  "version" : 1,
  "protocolVersion" : NumberLong(1),
  "writeConcernMajorityJournalDefault" : true,
  "members" : [
    {
      "_id" : 201,
      "host" : "10.1.5.201:27017",
      "arbiterOnly" : false,
      "buildIndexes" : true,
      "hidden" : false,
      "priority" : 1,
      "tags" : {

      },
      "slaveDelay" : NumberLong(0),
      "votes" : 1
    },
    {
      "_id" : 202,
      "host" : "10.1.5.202:27017",
      "arbiterOnly" : false,
      "buildIndexes" : true,
      "hidden" : false,
      "priority" : 1,
      "tags" : {

      },
    }
  ]
}

```



```

        "slaveDelay" : NumberLong(0),
        "votes" : 1
    }
],
"settings" : {
    "chainingAllowed" : true,
    "heartbeatIntervalMillis" : 2000,
    "heartbeatTimeoutSecs" : 10,
    "electionTimeoutMillis" : 10000,
    "catchUpTimeoutMillis" : -1,
    "catchUpTakeoverDelayMillis" : 30000,
    "getLastErrorModes" : {

    },
    "getLastErrorDefaults" : {
        "w" : 1,
        "wtimeout" : 0
    },
    "replicaSetId" : ObjectId("5b7d32a0339a21d84f409ab4")
}
}

```

输入

```
rs.status()
```

可以查看副本集状态信息，其中包含了主从信息，我们可以看出 201 节点处于 **PRIMARY** 状态，202 节点处于 **SECONDARY** 状态

```
my-rs1:PRIMARY> rs.status()
```

```

{
  "set" : "my-rs1",
  "date" : ISODate("2018-08-22T09:48:38.542Z"),
  "myState" : 1,
  "term" : NumberLong(1),
  "syncingTo" : "",
  "syncSourceHost" : "",
  "syncSourceId" : -1,
  "heartbeatIntervalMillis" : NumberLong(2000),
  "optimes" : {
    "lastCommittedOpTime" : {
      "ts" : Timestamp(1534931310, 1),
      "t" : NumberLong(1)
    },
    "readConcernMajorityOpTime" : {
      "ts" : Timestamp(1534931310, 1),
      "t" : NumberLong(1)
    }
  },

```

```

    "appliedOpTime" : {
      "ts" : Timestamp(1534931310, 1),
      "t" : NumberLong(1)
    },
    "durableOpTime" : {
      "ts" : Timestamp(1534931310, 1),
      "t" : NumberLong(1)
    }
  },
  "lastStableCheckpointTimestamp" : Timestamp(1534931294, 1),
  "members" : [
    {
      "_id" : 201,
      "name" : "10.1.5.201:27017",
      "health" : 1,
      "state" : 1,
      "stateStr" : "PRIMARY",
      "uptime" : 2615,
      "optime" : {
        "ts" : Timestamp(1534931310, 1),
        "t" : NumberLong(1)
      },
      "optimeDate" : ISODate("2018-08-22T09:48:30Z"),
      "syncingTo" : "",
      "syncSourceHost" : "",
      "syncSourceId" : -1,
      "infoMessage" : "",
      "electionTime" : Timestamp(1534928893, 1),
      "electionDate" : ISODate("2018-08-22T09:08:13Z"),
      "configVersion" : 2,
      "self" : true,
      "lastHeartbeatMessage" : ""
    },
    {
      "_id" : 202,
      "name" : "10.1.5.202:27017",
      "health" : 1,
      "state" : 2,
      "stateStr" : "SECONDARY",
      "uptime" : 2435,
      "optime" : {
        "ts" : Timestamp(1534931310, 1),
        "t" : NumberLong(1)
      },
    },
  ]
}

```

```

        "optimeDurable" : {
            "ts" : Timestamp(1534931310, 1),
            "t" : NumberLong(1)
        },
        "optimeDate" : ISODate("2018-08-22T09:48:30Z"),
        "optimeDurableDate" : ISODate("2018-08-22T09:48:30Z"),
        "lastHeartbeat" : ISODate("2018-08-22T09:48:36.715Z"),
        "lastHeartbeatRecv" : ISODate("2018-08-22T09:48:38.237Z"),
        "pingMs" : NumberLong(0),
        "lastHeartbeatMessage" : "",
        "syncingTo" : "",
        "syncSourceHost" : "",
        "syncSourceId" : -1,
        "infoMessage" : "",
        "configVersion" : 2
    }
],
"ok" : 1,
"operationTime" : Timestamp(1534931310, 1),
"$clusterTime" : {
    "clusterTime" : Timestamp(1534931310, 1),
    "signature" : {
        "hash" : BinData(0,"AAAAAAAAAAAAAAAAAAAAAAAAAAAAAA="),
        "keyId" : NumberLong(0)
    }
}
}
}

```

2.5.3 初始化副本集-Arbiter 节点

接着上一步在 mongo shell 中，使用

```
rs.addArb("10.1.5.203:27017")
```

添加 arbiter 节点

```

my-rs1:PRIMARY> rs.addArb("10.1.5.203:27017")
{
    "ok" : 1,
    "operationTime" : Timestamp(1534935152, 1),
    "$clusterTime" : {
        "clusterTime" : Timestamp(1534935152, 1),
        "signature" : {
            "hash" : BinData(0,"AAAAAAAAAAAAAAAAAAAAAAAAAAAAAA="),
            "keyId" : NumberLong(0)
        }
    }
}

```

```
}
```

添加完成之后，使用 `rs.status()` 查看集群状态，可以看出 201 节点处于 **PRIMARY** 状态，202 节点处于 **SECONDARY** 状态，203 节点处于 **ARBITER** 状态，符合规划，至此，副本集集群搭建完成。

```
my-rs1:PRIMARY> rs.status()
{
  "set" : "my-rs1",
  "date" : ISODate("2018-08-22T10:52:42.776Z"),
  "myState" : 1,
  "term" : NumberLong(1),
  "syncingTo" : "",
  "syncSourceHost" : "",
  "syncSourceId" : -1,
  "heartbeatIntervalMillis" : NumberLong(2000),
  "optimes" : {
    "lastCommittedOpTime" : {
      "ts" : Timestamp(1534935152, 1),
      "t" : NumberLong(1)
    },
    "readConcernMajorityOpTime" : {
      "ts" : Timestamp(1534935152, 1),
      "t" : NumberLong(1)
    },
    "appliedOpTime" : {
      "ts" : Timestamp(1534935152, 1),
      "t" : NumberLong(1)
    },
    "durableOpTime" : {
      "ts" : Timestamp(1534935152, 1),
      "t" : NumberLong(1)
    }
  },
  "lastStableCheckpointTimestamp" : Timestamp(1534935109, 1),
  "members" : [
    {
      "_id" : 201,
      "name" : "10.1.5.201:27017",
      "health" : 1,
      "state" : 1,
      "stateStr" : "PRIMARY",
      "uptime" : 3596,
      "optime" : {
```

```

        "ts" : Timestamp(1534935152, 1),
        "t" : NumberLong(1)
    },
    "optimeDate" : ISODate("2018-08-22T10:52:32Z"),
    "syncingTo" : "",
    "syncSourceHost" : "",
    "syncSourceId" : -1,
    "infoMessage" : "",
    "electionTime" : Timestamp(1534931627, 1),
    "electionDate" : ISODate("2018-08-22T09:53:47Z"),
    "configVersion" : 2,
    "self" : true,
    "lastHeartbeatMessage" : ""
},
{
    "_id" : 202,
    "name" : "10.1.5.202:27017",
    "health" : 1,
    "state" : 2,
    "stateStr" : "SECONDARY",
    "uptime" : 3545,
    "optime" : {
        "ts" : Timestamp(1534935152, 1),
        "t" : NumberLong(1)
    },
    "optimeDurable" : {
        "ts" : Timestamp(1534935152, 1),
        "t" : NumberLong(1)
    },
    "optimeDate" : ISODate("2018-08-22T10:52:32Z"),
    "optimeDurableDate" : ISODate("2018-08-22T10:52:32Z"),
    "lastHeartbeat" : ISODate("2018-08-22T10:52:40.785Z"),
    "lastHeartbeatRecv" : ISODate("2018-08-22T10:52:42.311Z"),
    "pingMs" : NumberLong(0),
    "lastHeartbeatMessage" : "",
    "syncingTo" : "",
    "syncSourceHost" : "",
    "syncSourceId" : -1,
    "infoMessage" : "",
    "configVersion" : 2
},
{
    "_id" : 203,
    "name" : "10.1.5.203:27017",

```

```

        "health" : 1,
        "state" : 7,
        "stateStr" : "ARBITER",
        "uptime" : 9,
        "lastHeartbeat" : ISODate("2018-08-22T10:52:40.786Z"),
        "lastHeartbeatRecv" : ISODate("2018-08-22T10:52:40.833Z"),
        "pingMs" : NumberLong(1),
        "lastHeartbeatMessage" : "",
        "syncingTo" : "",
        "syncSourceHost" : "",
        "syncSourceId" : -1,
        "infoMessage" : "",
        "configVersion" : 2
    },
    ],
    "ok" : 1,
    "operationTime" : Timestamp(1534935152, 1),
    "$clusterTime" : {
        "clusterTime" : Timestamp(1534935152, 1),
        "signature" : {
            "hash" : BinData(0,"AAAAAAAAAAAAAAAAAAAAAAAAAAAAAA="),
            "keyId" : NumberLong(0)
        }
    }
}

```

2.5.4 初始化的另一种方法

其实 2.5.2 和 2.5.3 可以合并起来，初始化时直接指定 arbiter 节点，如下：

```

rs.initiate( {
  _id : "my-rs1",
  members: [
    { _id: 201, host: "10.1.5.201:27017" },
    { _id: 202, host: "10.1.5.202:27017" },
    { _id: 203, host: "10.1.5.203:27017", arbiterOnly: true}
  ]
})

```

这种方法在 3.4.2 中也使用了。

2.6 其他集群操作

2.6.1 添加 secondary 节点

操作步骤：

- 1) 配置副本集名字于现有副本集名字一致，参考 2.4.2
- 2) 启动服务
- 3) 使用 mongo 连接到 primary 节点，执行如下命令

```
rs.add( { host: "IP:PORT", priority: 0, votes: 0 } )
```

命令解析：

priority: 优先级，指是否有权限成为主，比如 Arbiter 角色此值为 0

votes: 投票权权重，一般应设置各 secondary 权限相同

注意：初始化添加时，不能设置 priority、votes 大于 0，因为这可能引发问题，详情见官方文档，<https://docs.mongodb.com/manual/tutorial/expand-replica-set/>

所以必须使用如下步骤更新这两个值

- 4) 其中 4 表示 rs.conf() 返回的 members 数组中，新加入的成员的位置，注意，数组从 0 开始编号，所以对于一个有 2 节点的副本集集群，新加入的时第三位成员，但在数组中是第 2 个成员

```
var cfg = rs.conf();
cfg.members[2].priority = 1
cfg.members[2].votes = 1
rs.reconfig(cfg)
```

另外，注意这个操作会引发重新选举，可能会有 10-20s 服务中断，所以应在业务空闲时操作

- 5) 添加完成后，使用 rs.conf() 查看是否符合需求

举例：

```
my-rs1:PRIMARY> rs.add( { host: "10.1.5.202:27017", priority: 0, votes: 0 } )
{
  "ok" : 1,
  "operationTime" : Timestamp(1534995218, 1),
  "$clusterTime" : {
    "clusterTime" : Timestamp(1534995218, 1),
    "signature" : {
      "hash" : BinData(0,"AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA"),
      "keyId" : NumberLong(0)
    }
  }
}
```

添加

```

"members" : [
  {
    "_id" : 201,
    "host" : "10.1.5.201:27017",
    "arbiterOnly" : false,
    "buildIndexes" : true,
    "hidden" : false,
    "priority" : 1,
    "tags" : {
    },
    "slaveDelay" : NumberLong(0),
    "votes" : 1
  },
  {
    "_id" : 203,
    "host" : "10.1.5.203:27017",
    "arbiterOnly" : true,
    "buildIndexes" : true,
    "hidden" : false,
    "priority" : 0,
    "tags" : {
    },
    "slaveDelay" : NumberLong(0),
    "votes" : 1
  },
  {
    "_id" : 204,
    "host" : "10.1.5.202:27017",
    "arbiterOnly" : false,
    "buildIndexes" : true,
    "hidden" : false,
    "priority" : 0,
    "tags" : {
    },
    "slaveDelay" : NumberLong(0),
    "votes" : 0
  }
],

```

查看发现，新加的成员位于数据的第 3 个，组引用位置是 2

```

my-rs1:PRIMARY> var cfg = rs.conf();
my-rs1:PRIMARY>
my-rs1:PRIMARY> cfg.members[2].votes = 1
1
my-rs1:PRIMARY> cfg.members[2].priority = 1
1
my-rs1:PRIMARY>
my-rs1:PRIMARY> rs.reconfig(cfg)
{
  "ok" : 1,
  "operationTime" : Timestamp(1534995133, 1),
  "$clusterTime" : {
    "clusterTime" : Timestamp(1534995133, 1),
    "signature" : {
      "hash" : BinData(0,"AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA"),
      "keyId" : NumberLong(0)
    }
  }
}
my-rs1:PRIMARY>

```

更新新成员权限

2.6.2 删除 secondary 节点

操作步骤：

1) 使用 mongo 连接到 primary 节点, 执行如下命令

```
rs.remove("IP:PORT")
```

根据官方文档, 还有其它方法, 但这种是最简单的。

2) 添加完成后, 使用 rs.conf() 查看是否添加成功

举例:

```
my-rs1:PRIMARY> rs.remove("10.1.5.202:27017")
{
  "ok" : 1,
  "operationTime" : Timestamp(1534994095, 1),
  "$clusterTime" : {
    "clusterTime" : Timestamp(1534994095, 1),
    "signature" : {
      "hash" : BinData(0,"AAAAAAAAAAAAAAAAAAAAAAAAAAAA"),
      "keyId" : NumberLong(0)
    }
  }
}
```

2.6.3 添加 Arbiter 节点

参考 2.5.3

2.6.4 删除 Arbiter 节点

参考 2.6.2

2.6.5 查看集群信息

```
rs.conf()
```

```
rs.status()
```

```
rs.isMaster()
```

2.7 测试

2.7.1 观察日志

1) 集群初始化的日志

```
2018-08-23T13:44:37.554+0800 I NETWORK [conn1] Skipping connection for connection # 2
2018-08-23T13:44:37.554+0800 I REPL [conn1] New replica set config in use: { _id: "my-rs1", version: 1, protocolVersion: 1, writeConcernMajorityJournalDefault: true, members: [ { _id: 201, host: "10.1.5.201:27017", arbiterOnly: false, buildIndexes: true, hidden: false, priority: 1.0, tags: {}, slaveDelay: 0, votes: 1 }, { _id: 202, host: "10.1.5.202:27017", arbiterOnly: false, buildIndexes: true, hidden: false, priority: 1.0, tags: {}, slaveDelay: 0, votes: 1 } ], settings: { chainingAllowed: true, heartbeatIntervalMillis: 2000, heartbeatTimeoutSecs: 10, electionTimeoutMillis: 10000, catchUpTimeoutMillis: -1, catchUpTakeoverDelayMillis: 30000, getLastErrorModes: {}, getLastErrorDefaults: { w: 1, wtimeout: 0 }, replicasetid: ObjectId("5b7e49c5569c9d2aeb577481") } }
2018-08-23T13:44:37.554+0800 I REPL [conn1] This node is 10.1.5.201:27017 in the config
2018-08-23T13:44:37.554+0800 I REPL [conn1] transition to STARTUP2 from STARTUP
2018-08-23T13:44:37.555+0800 I REPL [replset-0] Member 10.1.5.202:27017 is now in state STARTUP
2018-08-23T13:44:37.556+0800 I REPL [conn1] Starting replication storage threads
2018-08-23T13:44:37.556+0800 I REPL [conn1] transition to RECOVERING from STARTUP2
2018-08-23T13:44:37.556+0800 I REPL [conn1] Starting replication fetcher thread
2018-08-23T13:44:37.556+0800 I REPL [conn1] Starting replication applier thread
2018-08-23T13:44:37.556+0800 I REPL [conn1] Starting replication reporter thread
2018-08-23T13:44:37.556+0800 I REPL [rsSync-0] Starting oplog application
2018-08-23T13:44:37.556+0800 I COMMAND [conn1] command local.system.replset appName: "MongoDB Shell" command: replSetInitiate { replSetInitiate: { _id: "my-rs1", members: [ { _id: 201, host: "10.1.5.201:27017", arbiterOnly: false, buildIndexes: true, hidden: false, priority: 1.0, tags: {}, slaveDelay: 0, votes: 1 }, { _id: 202, host: "10.1.5.202:27017", arbiterOnly: false, buildIndexes: true, hidden: false, priority: 1.0, tags: {}, slaveDelay: 0, votes: 1 } ], settings: { chainingAllowed: true, heartbeatIntervalMillis: 2000, heartbeatTimeoutSecs: 10, electionTimeoutMillis: 10000, catchUpTimeoutMillis: -1, catchUpTakeoverDelayMillis: 30000, getLastErrorModes: {}, getLastErrorDefaults: { w: 1, wtimeout: 0 }, replicasetid: ObjectId("5b7e49c5569c9d2aeb577481") } } }
2018-08-23T13:44:37.556+0800 I NETWORK [conn6] received client metadata from 10.1.5.202:54033 conn6: { driver: { name: "NetworkInterfaceL", version: "4.0.1" }, os: { type: "Linux", name: "CentOS release 6.6 (final)", architecture: "x86_64", version: "Kernel 2.6.32-504.el6.x86_64" } }
2018-08-23T13:44:37.559+0800 I NETWORK [listener] connection accepted from 10.1.5.202:54032 #5 (3 connections now open)
2018-08-23T13:44:37.559+0800 I NETWORK [conn5] end connection 10.1.5.202:54032 (2 connections now open)
2018-08-23T13:44:37.559+0800 I REPL [replset-0] Member 10.1.5.202:27017 is now in state STARTUP2
2018-08-23T13:44:37.590+0800 I NETWORK [listener] connection accepted from 10.1.5.202:54033 #6 (3 connections now open)
2018-08-23T13:44:37.591+0800 I NETWORK [conn6] received client metadata from 10.1.5.202:54033 conn6: { driver: { name: "NetworkInterfaceL", version: "4.0.1" }, os: { type: "Linux", name: "CentOS release 6.6 (final)", architecture: "x86_64", version: "Kernel 2.6.32-504.el6.x86_64" } }
2018-08-23T13:44:37.602+0800 I NETWORK [listener] connection accepted from 10.1.5.202:54034 #7 (4 connections now open)
2018-08-23T13:44:37.603+0800 I NETWORK [conn7] received client metadata from 10.1.5.202:54034 conn7: { driver: { name: "NetworkInterfaceL", version: "4.0.1" }, os: { type: "Linux", name: "CentOS release 6.6 (final)", architecture: "x86_64", version: "Kernel 2.6.32-504.el6.x86_64" } }
```

2) 添加 Arbiter 节点的日志

```
2018-08-23T13:45:08.289+0800 I REPL [conn1] replSetReconfig admin command received from client: new config: { _id: "my-rs1", version: 2, protocolVersion: 1, writeConcernMajorityJournalDefault: true, members: [ { _id: 201, host: "10.1.5.201:27017", arbiterOnly: false, buildIndexes: true, hidden: false, priority: 1.0, tags: {}, slaveDelay: 0, votes: 1 }, { _id: 202, host: "10.1.5.202:27017", arbiterOnly: true }, { _id: 203, host: "10.1.5.203:27017", arbiterOnly: true } ], settings: { chainingAllowed: true, heartbeatIntervalMillis: 2000, heartbeatTimeoutSecs: 10, electionTimeoutMillis: 10000, catchUpTimeoutMillis: -1, catchUpTakeoverDelayMillis: 30000, getLastErrorModes: {}, getLastErrorDefaults: { w: 1, wtimeout: 0 }, replicasetId: ObjectId("5b7e49c5569cd2aeb577481") } }
2018-08-23T13:45:08.294+0800 I REPL [replset] New replica set config in use: { _id: "my-rs1", version: 2, protocolVersion: 1, writeConcernMajorityJournalDefault: true, members: [ { _id: 201, host: "10.1.5.201:27017", arbiterOnly: false, buildIndexes: true, hidden: false, priority: 1.0, tags: {}, slaveDelay: 0, votes: 1 }, { _id: 202, host: "10.1.5.202:27017", arbiterOnly: true, buildIndexes: true, hidden: false, priority: 0.0, tags: {}, slaveDelay: 0, votes: 1 }, { _id: 203, host: "10.1.5.203:27017", arbiterOnly: true, buildIndexes: true, hidden: false, priority: 0.0, tags: {}, slaveDelay: 0, votes: 1 } ], settings: { chainingAllowed: true, heartbeatIntervalMillis: 2000, heartbeatTimeoutSecs: 10, electionTimeoutMillis: 10000, catchUpTimeoutMillis: -1, catchUpTakeoverDelayMillis: 30000, getLastErrorModes: {}, getLastErrorDefaults: { w: 1, wtimeout: 0 }, replicasetId: ObjectId("5b7e49c5569cd2aeb577481") } }
2018-08-23T13:45:08.296+0800 I REPL [replset] This node is 10.1.5.201:27017 in the config
2018-08-23T13:45:08.298+0800 I REPL [replset] Member 10.1.5.203:27017 is now in state STARTUP
```

3) 启动日志，参考 1.6.1，及其重要，请务必仔细阅读，有些需要调整系统环境等，可提高性能

2.7.2 观察主从切换

连接 primary 节点

```
[root@hadoop01 ~]# mongo 10.1.5.201:27017
MongoDB shell version v4.0.1
connecting to: mongodb://10.1.5.201:27017/test
MongoDB server version: 4.0.1
```

插入一条数据

```
show dbs
use quzl
db.testdb.insert({"name":"quzl"})
```

```
my-rs1:PRIMARY> show dbs
admin 0.000GB
config 0.000GB
local 0.000GB
my-rs1:PRIMARY>
my-rs1:PRIMARY> use quzl
switched to db quzl
my-rs1:PRIMARY>
my-rs1:PRIMARY> db.quzl.insert({"name":"quzl"})
WriteResult({"nInserted" : 1 })
my-rs1:PRIMARY>
my-rs1:PRIMARY> show dbs
admin 0.000GB
config 0.000GB
local 0.000GB
quzl 0.000GB
my-rs1:PRIMARY>
```

停止 201 节点，并连接 202 节点查看

```
[root@hadoop01 ~]# service mongod stop
Stopping mongod:
[root@hadoop01 ~]#
[root@hadoop01 ~]#
[root@hadoop01 ~]#
[root@hadoop01 ~]# mongo 10.1.5.202:27017
MongoDB shell version v4.0.1
connecting to: mongodb://10.1.5.202:27017/test
MongoDB server version: 4.0.1
```

使用

```
rs.status()
```

查看，发现 202 成为了 primary 节点，而 201 处于不可用状态

```

{
  "_id" : 201,
  "name" : "10.1.5.201:27017",
  "health" : 0,
  "state" : 8,
  "stateStr" : "(not reachable/healthy)",
  "uptime" : 0,
  "optime" : {
    "ts" : Timestamp(0, 0),
    "t" : NumberLong(-1)
  },
  "optimeDurable" : {
    "ts" : Timestamp(0, 0),
    "t" : NumberLong(-1)
  },
  "optimeDate" : ISODate("1970-01-01T00:00:00Z"),
  "optimeDurableDate" : ISODate("1970-01-01T00:00:00Z"),
  "lastHeartbeat" : ISODate("2018-08-23T06:11:48.669Z"),
  "lastHeartbeatRecv" : ISODate("2018-08-23T06:10:49.038Z"),
  "pingMs" : NumberLong(0),
  "lastHeartbeatMessage" : "Error connecting to 10.1.5.201:27017 :: caused b",
  "syncingTo" : "",
  "syncSourceHost" : "",
  "syncSourceId" : -1,
  "infoMessage" : "",
  "configVersion" : -1
},
{
  "_id" : 202,
  "name" : "10.1.5.202:27017",
  "health" : 1,
  "state" : 1,
  "stateStr" : "PRIMARY",
  "uptime" : 1662
}

```

进入 quz1 库中查看，发现前面插入的数据仍然存在

```

my-rs1:PRIMARY>
my-rs1:PRIMARY> use quz1
switched to db quz1
my-rs1:PRIMARY>
my-rs1:PRIMARY> db.quz1.find()
{ "_id" : ObjectId("5b7e4ed87d106091d9d1b91c"), "name" : "quz1" }
my-rs1:PRIMARY>
my-rs1:PRIMARY>

```

如果我们查看 202 日志，会发现状态转换的相关日志记录

```

[replexec-11] election succeeded, assuming primary role in term 2
[replexec-11] transition to PRIMARY from SECONDARY
[replexec-11] Entering primary catch-up mode.

```

2.7.3 主从切换后操作

如果我们再次启动 201 节点，发现 201 自动成为了 secondary 节点，202 仍然是 primary 节点

```

members" : [
  {
    "_id" : 201,
    "name" : "10.1.5.201:27017",
    "health" : 1,
    "state" : 2,
    "stateStr" : "SECONDARY",
    "uptime" : 12,
    "optime" : {
      "ts" : Timestamp(1535006071, 1),
      "t" : NumberLong(2)
    },
    "optimeDate" : ISODate("2018-08-23T06:34:31Z"),
    "syncingTo" : "10.1.5.202:27017",
    "syncSourceHost" : "10.1.5.202:27017",
    "syncSourceId" : 202,
    "infoMessage" : "",
    "configversion" : 2,
    "self" : true,
    "lastHeartbeatMessage" : ""
  },
  {
    "_id" : 202,
    "name" : "10.1.5.202:27017",
    "health" : 1,
    "state" : 1,
    "stateStr" : "PRIMARY",
    "uptime" : 10,
    "optime" : {
      "ts" : Timestamp(1535006071, 1),
      "t" : NumberLong(2)
    },
    "optimeDurable" : {

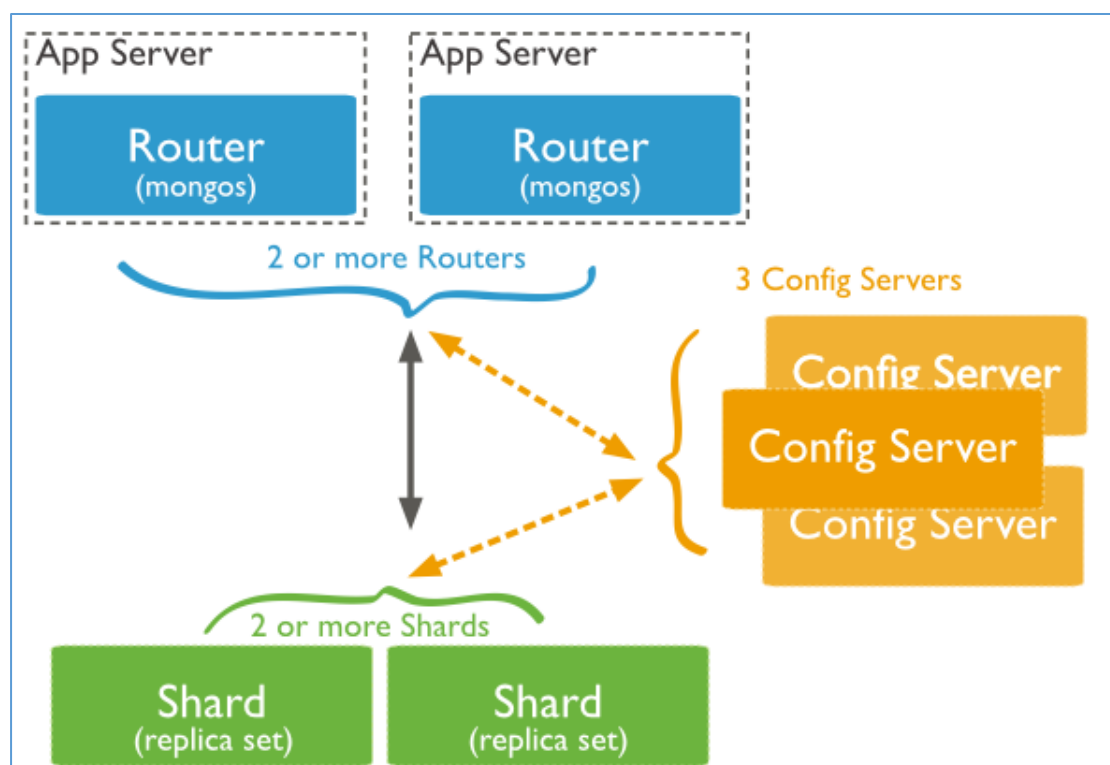
```

特别注意:

如果是生产环境中，发生了主从切换，而且主机器永久性无法起来了，可以通过 2.6.2、2.6.1 的操作步骤，删除故障节点，添加新节点处理。

第三章 分片集群

3.1 拓扑图



- **Shard:**
数据分片服务器。用于存储实际的数据块，可以是一个单独的 mongo 实例，也可以是一个副本集，在生产环境下 Shard 一般是一个 Replica Set，以防止该数据片的单点故障。
- **Config Server:**
配置服务器集群，保存集群的元数据（metadata），包含各个 Shard 的路由规则（trunk 信息）。
- **Query Routers:**
前端路由，客户端由此接入，且让整个集群看上去像单一数据库，前端应用可以透明使用，由 mongos 把读写请求路由到指定的 Shard 上去。一个 Sharding 集群，可以有一个 mongos，也可以有多 mongos 以减轻客户端请求的压力。

mongos 从配置服务器读取配置，这些信息只保留在内存中，不会有本地存储，当配置服务器的配置有变化时，会通知 mongos 更新。

3.2 测试规划

OS: CentOS 6.6 x64

MongoDB: 4.0.1

序号	类目	服务器1 10.1.5.201	服务器2 10.1.5.202	服务器3 10.1.5.203	集群名	端口	日志路径	数据目录	配置文件
1	mongos server	node 1	node 2	node 3	×	28100	/mongodb/mongos/mongs.log	×	×
2	config server	node 1	node 2	node 3	mycfg	28200	/mongodb/config/config.log	/mongodb/config/data	×
3	shard1	node 1	node 2	node 3 仲裁	rs1	27100	/mongodb/shard1/shard1.log	/mongodb/shard1/data	×
4	shard2	node 1 仲裁	node 2	node 3	rs2	27200	/mongodb/shard2/shard2.log	/mongodb/shard2/data	×
5	shard3	node 1	node 2 仲裁	node 3	rs3	27300	/mongodb/shard3/shard3.log	/mongodb/shard3/data	×

这里只是为了测试，使用了3台机器，部署了15个节点，实际生产环境中：

- 1) 将 mongos server、config server、shard 分开部署到不同机器
- 2) shard，部署成副本集形式，组的多少可横向扩展
- 3) mongos，应该是多节点的，不构成集群，各节点均可独立运行
- 4) config server，应该是多节点的，其实也是一个特殊的副本集集群
- 5) 我这里只是测试，没有单独的配置文件，使用参数启动，没指定的参数使用默认配置，简化了部署过程。生产中，应尽量使用配置文件启动，且定义为系统服务，方便维护，可参考《第二章 副本集集群》部分。

3.3 下载和部署

本节 3.3 的所有操作，在三台机器中都要执行。

3.3.2 添加 yum 源

```
[root@hadoop01 ~]# cat /etc/yum.repos.d/Mongodb-4.repo
cat > /etc/yum.repos.d/Mongodb-4.repo << EOF
[mongodb-org-4.0]
name = MongoDB Repository
baseurl = http://repo.mongodb.org/yum/redhat/$releasever/mongodb-org/4.0/x86_64/
gpgcheck = 0
enabled = 1
EOF
```

3.3.2 安装依赖包

```
yum install openssl libcurl
```

3.3.3 安装软件包

```
yum install mongodb-org
```

将自动安装以下 4 个依赖包：

mongodb-org-server 包含 `mongod` 守护程序以及关联的配置和 `init` 脚本。

mongodb-org-mongos 包含 `mongos` 守护进程。

mongodb-org-shell 包含 `mongo` shell。

mongodb-org-tools 包含以下的 MongoDB 工具：

`mongoimport`, `bsondump`, `mongodump`, `mongoexport`, `mongofiles`,
`mongorestore`, `mongostat` and `mongotop`.

软件包	架构	版本
正在安装： mongodb-org	x86_64	4.0.1-1.el6
为依赖而安装： mongodb-org-mongos	x86_64	4.0.1-1.el6
mongodb-org-server	x86_64	4.0.1-1.el6
mongodb-org-shell	x86_64	4.0.1-1.el6
mongodb-org-tools	x86_64	4.0.1-1.el6
事务概要		
Install 5 Package(s)		
总下载量: 76 M		
Installed size: 232 M		
确定吗? [y/N]:		

3.3.4 相关目录准备

可使用如下命令，一次性创建完成。

```
mkdir -pv /mongodb/{mongos/data,config/data,shard1/data,shard2/data,shard3/data}
```

如下图

```
[root@hadoop01 ~]# mkdir -pv /mongodb/{mongos/data,config/data,shard1/data,shard2/data,shard3/data}
mkdir: 已创建目录 "/mongodb/mongos"
mkdir: 已创建目录 "/mongodb/mongos/data"
mkdir: 已创建目录 "/mongodb/config"
mkdir: 已创建目录 "/mongodb/config/data"
mkdir: 已创建目录 "/mongodb/shard1"
mkdir: 已创建目录 "/mongodb/shard1/data"
mkdir: 已创建目录 "/mongodb/shard2"
mkdir: 已创建目录 "/mongodb/shard2/data"
mkdir: 已创建目录 "/mongodb/shard3"
mkdir: 已创建目录 "/mongodb/shard3/data"
[root@hadoop01 ~]#
[root@hadoop01 ~]#
[root@hadoop01 ~]# tree /mongodb
/mongodb
├── config
│   └── data
├── mongos
│   └── data
├── shard1
│   └── data
├── shard2
│   └── data
└── shard3
    └── data
10 directories, 0 files
```

这里是测试，直接使用 root 运行，如果不是 root，要修改目录权限，比如使用 mongo 运行
`chmod -R mongo.mongd /mongodb`

3.4 启动 Shard Server

该小节其实就是《第二章 副本集集群》的搭建过程，这里只写关键步骤，详细情况可参考前第二章。

3.4.1 启动所有 mongod 服务

三台机器都执行以下语句，初始化三个副本集，每个副本集 3 个节点

```
mongod --bind_ip 0.0.0.0 --port 27100 --replSet rs1 --shardsvr \  
--dbpath=/mongodb/shard1/data \  
--logpath=/mongodb/shard1/shard1.log \  
--logappend --fork
```

```
mongod --bind_ip 0.0.0.0 --port 27200 --replSet rs2 --shardsvr \  
--dbpath=/mongodb/shard2/data \  
--logpath=/mongodb/shard2/shard2.log \  
--logappend --fork
```

```
mongod --bind_ip 0.0.0.0 --port 27300 --replSet rs3 --shardsvr \  
--dbpath=/mongodb/shard3/data \  
--logpath=/mongodb/shard3/shard3.log \  
--logappend --fork
```

如下图，表示启动成功

```
mkdir: 已创建目录 '/mongodb/shard3/data'  
[root@hadoop03 ~]# mongod --bind_ip 0.0.0.0 --port 27100 --replSet rs1 --shardsvr \  
> --dbpath=/mongodb/shard1/data \  
> --logpath=/mongodb/shard1/shard1.log \  
> --logappend --fork  
2018-08-27T11:55:15.109+0800 I CONTROL [main] Automatically disabling TLS 1.0, to force-enable TLS 1.0  
about to fork child process, waiting until server is ready for connections.  
forked process: 1797  
child process started successfully, parent exiting  
[root@hadoop03 ~]#  
[root@hadoop03 ~]#  
[root@hadoop03 ~]#  
[root@hadoop03 ~]# mongod --bind_ip 0.0.0.0 --port 27200 --replSet rs2 --shardsvr \  
> --dbpath=/mongodb/shard2/data \  
> --logpath=/mongodb/shard2/shard2.log \  
> --logappend --fork  
2018-08-27T11:55:36.906+0800 I CONTROL [main] Automatically disabling TLS 1.0, to force-enable TLS 1.0  
about to fork child process, waiting until server is ready for connections.  
forked process: 1828  
child process started successfully, parent exiting  
[root@hadoop03 ~]#  
[root@hadoop03 ~]#  
[root@hadoop03 ~]#  
[root@hadoop03 ~]# mongod --bind_ip 0.0.0.0 --port 27300 --replSet rs3 --shardsvr \  
> --dbpath=/mongodb/shard3/data \  
> --logpath=/mongodb/shard3/shard3.log \  
> --logappend --fork  
2018-08-27T11:55:55.516+0800 I CONTROL [main] Automatically disabling TLS 1.0, to force-enable TLS 1.0  
about to fork child process, waiting until server is ready for connections.  
forked process: 1859  
child process started successfully, parent exiting  
[root@hadoop03 ~]#  
[root@hadoop03 ~]#
```

更多启动参数，使用

```
mongod --help
```

查看

注意：必须使用--shardsvr 参数，否则使用 mongos 连接后插入数，会提示错误如下

```
mongos>  
mongos> db.table1.save({id:1,"test1":"testval1"})  
writeResult({  
  "nInserted" : 0,  
  "writeError" : {  
    "code" : 193,  
    "errmsg" : "Cannot accept sharding commands if not started with --shardsvr"  
  }  
})  
mongos>  
mongos>
```

网上有些资料说，不需要添加这个参数，是错误的说法，也可能旧版本是可以的。经验证，4.0 版本，必须要添加这个参数。

3.4.2 创建三个分片集群

副本集 1，即 shard1,使用 mongo 连到 201 的 27100 端口。

按照规划，203:27100 是仲裁节点，集群 id 是 rs1，成员 id 随意写。

执行如下语句：

```
mongo 10.1.5.201:27100
```

```
rs.initiate( {
  _id : "rs1",
  members: [
    { _id: 0, host: "10.1.5.201:27100" },
    { _id: 1, host: "10.1.5.202:27100" },
    { _id: 2, host: "10.1.5.203:27100", arbiterOnly: true }
  ]
})
```

```
[root@hadoop01 ~]# mongo 10.1.5.201:27100
MongoDB shell version v4.0.1
connecting to: mongodb://10.1.5.201:27100/test
MongoDB server version: 4.0.1
server has startup warnings:
```

```
> rs.initiate( {
...   _id : "rs1",
...   members: [
...     { _id: 0, host: "10.1.5.201:27100" },
...     { _id: 1, host: "10.1.5.202:27100" },
...     { _id: 2, host: "10.1.5.203:27100", arbiterOnly: true }
...   ]
... })
{
  "ok" : 1,
  "operationTime" : Timestamp(1535096043, 1),
  "$clusterTime" : {
    "clusterTime" : Timestamp(1535096043, 1),
    "signature" : {
      "hash" : BinData(0,"AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA"),
      "keyId" : NumberLong(0)
    }
  }
}
```

副本集 2，即 shard2,使用 mongo 连到 202 的 27200 端口。

按照规划，201:27200 是仲裁节点，集群 id 是 rs2，成员 id 随意写。

执行如下语句：

```
mongo 10.1.5.202:27200
```

```
rs.initiate( {
  _id : "rs2",
  members: [
    { _id: 0, host: "10.1.5.201:27200", arbiterOnly: true },
    { _id: 1, host: "10.1.5.202:27200" },
    { _id: 2, host: "10.1.5.203:27200" }
  ]
})
```

```
[root@hadoop01 ~]# mongo 10.1.5.202:27200
MongoDB shell version v4.0.1
connecting to: mongodb://10.1.5.202:27200/test
MongoDB server version: 4.0.1
server has startup warnings:
```

```
> rs.initiate( {
...   _id : "rs2",
...   members: [
...     { _id: 0, host: "10.1.5.201:27200", arbiterOnly: true},
...     { _id: 1, host: "10.1.5.202:27200"},
...     { _id: 2, host: "10.1.5.203:27200"}
...   ]
... })
{
  "ok" : 1,
  "operationTime" : Timestamp(1535096881, 1),
  "$clusterTime" : {
    "clusterTime" : Timestamp(1535096881, 1),
    "signature" : {
      "hash" : BinData(0,"AAAAAAAAAAAAAAAAAAAAAAAAAAAA="),
      "keyId" : NumberLong(0)
    }
  }
}
```

副本集 3，即 shard3,使用 mongo 连到 203 的 27300 端口。
按照规划，202:27300 是仲裁节点，集群 id 是 rs3，成员 id 随意写。
执行如下语句：

```
mongo 10.1.5.203:27300
```

```
rs.initiate( {
  _id : "rs3",
  members: [
    { _id: 0, host: "10.1.5.201:27300"},
    { _id: 1, host: "10.1.5.202:27300", arbiterOnly: true},
    { _id: 2, host: "10.1.5.203:27300"}
  ]
})
```

```
[root@hadoop01 ~]# mongo 10.1.5.203:27300
MongoDB shell version v4.0.1
connecting to: mongodb://10.1.5.203:27300/test
MongoDB server version: 4.0.1
Server has startup warnings:
```

```
> rs.initiate( {
...   _id : "rs3",
...   members: [
...     { _id: 0, host: "10.1.5.201:27300"},
...     { _id: 1, host: "10.1.5.202:27300", arbiterOnly: true},
...     { _id: 2, host: "10.1.5.203:27300"}
...   ]
... })
{
  "ok" : 1,
  "operationTime" : Timestamp(1535097032, 1),
  "$clusterTime" : {
    "clusterTime" : Timestamp(1535097032, 1),
    "signature" : {
      "hash" : BinData(0,"AAAAAAAAAAAAAAAAAAAAAAAAAAAA="),
      "keyId" : NumberLong(0)
    }
  }
}
```

3.5 启动 config server

3.5.1 启动 mongod 服务

三台机器都要执行，其实这一步和 3.4.1 执行的语句区别就是加上了

--configsvr

该参数表示，创建的是一个配置服务。网上有些教程说，不用添加这个参数，如果不加，创建 config 集群也可以成功，但 mongos 服务无法起来。

```
mongod --bind_ip 0.0.0.0 --port 28200 \  
--configsvr \  
--replSet mycfg \  
--dbpath=/mongodb/config/data \  
--logpath=/mongodb/config/config.log \  
--logappend --fork
```

如下图，表示启动成功

```
[root@hadoop01 ~]#  
[root@hadoop01 ~]# mongod --bind_ip 0.0.0.0 --port 28200 --configsvr --replSet mycfg --dbpath=/mongodb/config/  
/data --logpath=/mongodb/config/config.log --logappend --fork  
2018-08-24T16:31:17.896+0800 I CONTROL [main] Automatically disabling TLS 1.0, to force-enable TLS 1.0 specif  
y --sslDisabledProtocols 'none'  
about to fork child process, waiting until server is ready for connections.  
forked process: 17082  
child process started successfully, parent exiting  
[root@hadoop01 ~]#
```

3.5.2 创建 config 集群

使用 mongo 连到 201 的 28200 端口

按照规划，集群 id 是 mycfg，成员 id 随意写。

执行如下语句：

```
mongo 10.1.5.201:28200
```

```
rs.initiate( {  
  _id : "mycfg",  
  members: [  
    { _id: 0, host: "10.1.5.201:28200"},  
    { _id: 1, host: "10.1.5.202:28200"},  
    { _id: 2, host: "10.1.5.203:28200"}  
  ]  
})
```

注意，因为 config 集群，存储的数据量很少，所以创建的是没有 arbiter 节点的副本集集群，至于区别可参考 2.1 节

```
[root@hadoop01 ~]#  
[root@hadoop01 ~]# mongo 10.1.5.201:28200  
MongoDB shell version v4.0.1  
connecting to: mongodb://10.1.5.201:28200/test  
MongoDB server version: 4.0.1
```

```
> rs.initiate( {
...   _id : "mycfg",
...   members: [
...     { _id: 0, host: "10.1.5.201:28200"},
...     { _id: 1, host: "10.1.5.202:28200"},
...     { _id: 2, host: "10.1.5.203:28200"}
...   ]
... })
{
  "ok" : 1,
  "operationTime" : Timestamp(1535098279, 1),
  "$clusterTime" : {
    "clusterTime" : Timestamp(1535098279, 1),
    "signature" : {
      "hash" : BinData(0,"AAAAAAAAAAAAAAAAAAAAAAAAAAAA="),
      "keyId" : NumberLong(0)
    }
  }
}
```

3.6 启动 mongos server

mongos server 也应该多台的，使用下面命令，在三台机器均执行

```
mongos --bind_ip 0.0.0.0 --port 28100 \
--configdb mycfg/10.1.5.201:28200,10.1.5.202:28200,10.1.5.203:28200 \
--logpath=/mongodb/mongos/mongos.log \
--logappend --fork
```

如下图所示，表示启动成功

```
[root@hadoop03 ~]#
[root@hadoop03 ~]# mongos --bind_ip 0.0.0.0 --port 28100 \
> --configdb mycfg/10.1.5.201:28200,10.1.5.202:28200,10.1.5.203:28200 \
> --logpath=/mongodb/mongos/mongos.log \
> --logappend --fork
2018-08-24T16:37:15.865+0800 I,CONTROL [main] Automatically disabling TLS 1.0, to force-enable TLS 1.0 specif
y --sslDisabledProtocols 'none'
about to fork child process, waiting until server is ready for connections.
forked process: 4265
child process started successfully, parent exiting
[root@hadoop03 ~]#
```

3.7 添加分片信息

3.7.1 添加 rs1

连接 mongos，输入以下命令

```
mongo 10.1.5.201:28100
use admin
db.runCommand( { addshard : "rs1/10.1.5.201:27100,10.1.5.203:27100,10.1.5.202:27100"});
```

如下图：

```
[root@hadoop01 ~]#
[root@hadoop01 ~]# mongo 127.0.0.1:28100
MongoDB shell version v4.0.1
connecting to: mongodb://127.0.0.1:28100/test
MongoDB server version: 4.0.1
```

```

mongos> use admin
switched to db admin
mongos>
mongos> db.runCommand( { addshard : "rs1/10.1.5.201:27100,10.1.5.203:27100,10.1.5.202:27100"});
{
  "shardAdded" : "rs1",
  "ok" : 1,
  "operationTime" : Timestamp(1535103097, 3),
  "$clusterTime" : {
    "clusterTime" : Timestamp(1535103097, 3),
    "signature" : {
      "hash" : BinData(0,"AAAAAAAAAAAAAAAAAAAAAAAAAAAA="),
      "keyId" : NumberLong(0)
    }
  }
}

```

3.7.2 添加 rs2

接着上一步，输入以下命令

```
db.runCommand( { addshard : "rs2/10.1.5.201:27200,10.1.5.202:27200,10.1.5.203:27200"});
```

如下图：

```

mongos>
mongos> db.runCommand( { addshard : "rs2/10.1.5.201:27200,10.1.5.202:27200,10.1.5.203:27200"});
{
  "shardAdded" : "rs2",
  "ok" : 1,
  "operationTime" : Timestamp(1535103487, 2),
  "$clusterTime" : {
    "clusterTime" : Timestamp(1535103487, 2),
    "signature" : {
      "hash" : BinData(0,"AAAAAAAAAAAAAAAAAAAAAAAAAAAA="),
      "keyId" : NumberLong(0)
    }
  }
}

```

3.7.3 添加 rs3

接着上一步，输入以下命令

```
db.runCommand( { addshard : "rs3/10.1.5.201:27300,10.1.5.202:27300,10.1.5.203:27300"});
```

如下图：

```

mongos> db.runCommand( { addshard : "rs3/10.1.5.201:27300,10.1.5.202:27300,10.1.5.203:27300"});
{
  "shardAdded" : "rs3",
  "ok" : 1,
  "operationTime" : Timestamp(1535103535, 4),
  "$clusterTime" : {
    "clusterTime" : Timestamp(1535103535, 4),
    "signature" : {
      "hash" : BinData(0,"AAAAAAAAAAAAAAAAAAAAAAAAAAAA="),
      "keyId" : NumberLong(0)
    }
  }
}

```

3.7.4 相关说明

- 1) 只需要在一台 mongos 中添加配置信息即可
- 2) 配置信息会通过 mongos 同步到 config 服务器，其他 mongos 再读取 config 服务器的信息，从而获得同步
- 3) 各 mongos 可以独立工作

3.8 启用分片

目前搭建了 mongod 配置服务器、路由服务器，各个分片服务器，不过应用程序连接到 mongos 路由服务器并不能使用分片机制，还需要在程序里设置分片配置，让分片生效。

3.8.1 指定分片的库

使用 mongo shell 连接其中一台 mongos，注意是 mongos，端口是 28100

```
mongo 10.1.5.203:28100
```

```
[root@hadoop03 ~]#  
[root@hadoop03 ~]# mongo 10.1.5.203:28100  
MongoDB shell version v4.0.1  
connecting to: mongodb://10.1.5.203:28100/test  
MongoDB server version: 4.0.1  
Server has startup warnings:
```

使用如下语句

```
sh.enableSharding("com")
```

指定要使用分片功能的数据库

```
mongos>  
mongos> sh.enableSharding("com")  
{  
  "ok" : 1,  
  "operationTime" : Timestamp(1535350115, 2),  
  "$clusterTime" : {  
    "clusterTime" : Timestamp(1535350115, 2),  
    "signature" : {  
      "hash" : BinData(0,"AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA"),  
      "keyId" : NumberLong(0)  
    }  
  }  
}
```

使用如下语句查看是否配置成功。其中 "partitioned" : true 表示启用分片

```
use config
```

```
db.databases.find()
```

```
mongos> use config  
switched to db config  
mongos> db.databases.find()  
{ "_id" : "com", "primary" : "rs2", "partitioned" : true, "version" : { "uuid" : UUID("f08d2e37-463f-463f-bdbf-77d3b3f46161"), "lastMod" : 1 } }  
mongos>
```

3.8.2 指定分片的集合

接着上一步，运行如下语句

```
sh.shardCollection('com.users',{ "id":"hashed" })
```

表示对 com 库的 users 集合启用分片，分片的 key 是 id，分片的方法是 hashed

另外，如果集合中已经存在数据，在选定作为 shard key 的键列必须提前创建索引；如果集合为空，mongodb 将在激活集合分片（sh.shardCollection）时创建索引。

至此，整个分片集群搭建完成。下面进行测试部分。

3.9 测试

3.9.1 观察日志

参考 1.6.1 和 2.7.1，这一步一定要看下，因为有些系统环境变量需要调整。

3.9.2 查看各副本集群配置

shard1:

```
[root@hadoop01 ~]# mongo 10.1.5.201:27100
MongoDB shell version v4.0.1
connecting to: mongodb://10.1.5.201:27100/test
MongoDB server version: 4.0.1
Server has startup warnings:

rs1:PRIMARY> rs.isMaster()
{
  "hosts" : [
    "10.1.5.201:27100",
    "10.1.5.202:27100"
  ],
  "arbiters" : [
    "10.1.5.203:27100"
  ],
  "primary" : "10.1.5.201:27100"
}
```

shard2:

```
[root@hadoop01 ~]# mongo 10.1.5.201:27200
MongoDB shell version v4.0.1
connecting to: mongodb://10.1.5.201:27200/test
MongoDB server version: 4.0.1
Server has startup warnings:

rs2:ARBITER> rs.isMaster()
{
  "hosts" : [
    "10.1.5.202:27200",
    "10.1.5.203:27200"
  ],
  "arbiters" : [
    "10.1.5.201:27200"
  ],
  "primary" : "10.1.5.202:27200"
}
```

shard3:

```
[root@hadoop01 ~]# mongo 10.1.5.201:27300
MongoDB shell version v4.0.1
connecting to: mongodb://10.1.5.201:27300/test
MongoDB server version: 4.0.1
Server has startup warnings:

rs3:SECONDARY> rs.isMaster()
{
  "hosts" : [
    "10.1.5.203:27300",
    "10.1.5.201:27300"
  ],
  "arbiters" : [
    "10.1.5.202:27300"
  ],
  "primary" : "10.1.5.203:27300"
}
```

经过查看，符合规划。

3.9.3 查看分片配置

使用

```
mongo 10.1.5.201:28100
```

登陆 mongos, 使用如下命令查看分片配置

```
use admin
```

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

如下图所示, 注意每个 shard 集群中, 仲裁节点并未显示, 这是正常现象

```
[root@hadoop01 ~]#  
[root@hadoop01 ~]# mongo 127.0.0.1:28100  
MongoDB shell version v4.0.1  
connecting to: mongod://127.0.0.1:28100/test  
MongoDB server version: 4.0.1  
  
mongos>  
mongos> use admin  
switched to db admin  
mongos>  
mongos> db.runCommand({ listshards : 1 })  
{  
  "shards" : [  
    {  
      "_id" : "rs1",  
      "host" : "rs1/10.1.5.201:27100,10.1.5.202:27100",  
      "state" : 1  
    },  
    {  
      "_id" : "rs2",  
      "host" : "rs2/10.1.5.202:27200,10.1.5.203:27200",  
      "state" : 1  
    },  
    {  
      "_id" : "rs3",  
      "host" : "rs3/10.1.5.201:27300,10.1.5.203:27300",  
      "state" : 1  
    }  
  ],  
  "ok" : 1,  
  "operationTime" : Timestamp(1535103955, 1),  
  "$clusterTime" : {  
    "clusterTime" : Timestamp(1535103955, 1),  
    "signature" : {  
      "hash" : BinData(0,"AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA="),  
      "keyId" : NumberLong(0)  
    }  
  }  
}
```

同时也可以使用

```
sh.status()
```

查看当前的分片状态


```

mongos> sh.status()
mongos> --- Sharding Status ---
  sharding version: {
    "_id" : 1,
    "minCompatibleVersion" : 5,
    "currentVersion" : 6,
    "clusterId" : ObjectId("5b7fc2a3f4bef03b44f93f53")
  }
  shards:
    { "_id" : "rs1", "host" : "rs1/10.1.5.201:27100,10.1.5.202:27100", "state" : 1 }
    { "_id" : "rs2", "host" : "rs2/10.1.5.202:27200,10.1.5.203:27200", "state" : 1 }
    { "_id" : "rs3", "host" : "rs3/10.1.5.201:27300,10.1.5.203:27300", "state" : 1 }
  active mongoses:
    "4.0.1" : 3
  autosplit:
    Currently enabled: yes
  balancer:
    Currently enabled: yes
    Currently running: no
    Failed balancer rounds in last 5 attempts: 0
    Migration Results for the last 24 hours:
      No recent migrations
  databases:
    { "_id" : "config", "primary" : "config", "partitioned" : true }
      config.system.sessions
        shard key: { "_id" : 1 }
        unique: false
        balancing: true
        chunks:
          rs1 1
          { "_id" : { "$minkey" : 1 } } --> { "_id" : { "$maxkey" : 1 } } on : rs1 Timestamp(1, 0)
mongos>

```

经查，符合规划。

3.9.4 验证是否会分片

使用

mongo 10.1.5.201:28100

登陆 mongos

```

[root@hadoop01 ~]#
[root@hadoop01 ~]# mongo 127.0.0.1:28100
MongoDB shell version v4.0.1
connecting to: mongodb://127.0.0.1:28100/test
MongoDB server version: 4.0.1

```

切换到 com 库，创建 20W 条数据，其中 id 字段是变化的

use com

for(i=1;i<=200000;i++) db.users.insert({id:i,name:'hukey',age:23})

```

mongos>
mongos> use com
switched to db com
mongos>
mongos> for(i=1;i<=200000;i++) db.users.insert({id:i,name:'hukey',age:23})
writeResult({ "nInserted" : 1 })
mongos>
mongos>

```

使用如下命令，查看 users 表的信息

use com

db.users.stats()

```

[root@hadoop01 ~]#
[root@hadoop01 ~]# mongo 10.1.5.201:28100
MongoDB shell version v4.0.1
connecting to: mongodb://10.1.5.201:28100/test
MongoDB server version: 4.0.1
Server has startup warnings:
2018-08-27T12:22:59.384+0800 I CONTROL [main]
2018-08-27T12:22:59.384+0800 I CONTROL [main] ** WARNING: Acces
2018-08-27T12:22:59.384+0800 I CONTROL [main] ** WARNING: Read
2018-08-27T12:22:59.384+0800 I CONTROL [main] ** WARNING: You
2018-08-27T12:22:59.384+0800 I CONTROL [main]
mongos>
mongos> use com
switched to db com
mongos>
mongos> db.users.stats()
{
  "sharded" : true,
  "capped" : false,
  "wiredTiger" : {
    "metadata" : {
      "formatVersion" : 1
    }
  }
}

```

如下面三图所示，20W 个 Key，被分配到了 3 个分片集群上，当然不是绝对 100%平均，这牵涉到分片的方法和技术，有兴趣的同学可以深入了解下。

```

"rs1" : {
  "ns" : "com.users",
  "size" : 4228245,
  "count" : 67115,
  "avgobjsize" : 63,
  "storageSize" : 1343488,
  "capped" : false,
  "wiredTiger" : {
    "metadata" : {
      "formatVersion" : 1
    }
  }
}

```

```

"rs2" : {
  "ns" : "com.users",
  "size" : 4184271,
  "count" : 66417,
  "avgobjsize" : 63,
  "storageSize" : 1327104,
  "capped" : false,
  "wiredTiger" : {
    "metadata" : {
      "formatVersion" : 1
    }
  }
}

```

```

"rs3" : {
  "ns" : "com.users",
  "size" : 4187484,
  "count" : 66468,
  "avgobjsize" : 63,
  "storageSize" : 1323008,
  "capped" : false,
  "wiredTiger" : {
    "metadata" : {
      "formatVersion" : 1
    }
  }
}

```

3.9.5 shared server 主从切换

参考 2.7.3

3.9.6 conf server 主从切换

其实配置服务器集群也是一个特殊的副本集集群，前面已多次说到。相关操作命令通用，可参考第二章相关内容。

主从切换的相关内容，参考 2.7.3

3.10 使用配置创建集群

使用配置文件进行创建，会有很多配置项，参考

<https://docs.mongodb.com/manual/reference/configuration-options/>

3.10.1 规划

3.10.2 mongod.conf

三个分片，有三个配置文件，分布在三个节点，所以一共有 9 个，这里只列出一个。

```
systemLog:
  destination: file
  logAppend: true
  path: /data/mongodb/shard1/mongod.log

storage:
  dbPath: /data/mongodb/shard1/data
  journal:
    enabled: true

processManagement:
  fork: true
  pidFilePath: /data/mongodb/shard1/mongod.pid
  timeZoneInfo: /usr/share/zoneinfo

net:
  port: 27100
```

```
bindIp: 0.0.0.0
```

```
#副本集相关信息
```

```
replication:
```

```
  replSetName: shard1
```

```
#分片信息
```

```
sharding:
```

```
  clusterRole: shardsvr
```

3. 10. 3 **mongod-config.conf**

三个节点都相同:

```
systemLog:
```

```
  destination: file
```

```
  logAppend: true
```

```
  path: /data/mongodb/config/mongod.log
```

```
storage:
```

```
  dbPath: /data/mongodb/config/data
```

```
  journal:
```

```
    enabled: true
```

```
processManagement:
```

```
  fork: true
```

```
  pidFilePath: /data/mongodb/config/mongod.pid
```

```
  timeZoneInfo: /usr/share/zoneinfo
```

```
net:
```

```
  port: 27400
```

```
  bindIp: 0.0.0.0
```

```
#副本集相关信息
```

```
replication:
```

```
replSetName: mongo-config
```

#分片信息

```
sharding:  
  clusterRole: configsvr
```

3. 10. 4 mongod-mongos.conf

一共有三个，都相同

```
systemLog:  
  destination: file  
  logAppend: true  
  path: /data/mongodb/mongos/mongod.log  
  
processManagement:  
  fork: true  
  pidFilePath: /data/mongodb/mongos/mongod.pid  
  timeZoneInfo: /usr/share/zoneinfo  
  
net:  
  port: 27500  
  bindIp: 0.0.0.0  
  
#mongos 检测副本集是否在线的等待时间  
replication:  
  localPingThresholdMs: 15  
  
sharding:  
  configDB: mongo-config/192.168.3.136:27400,192.168.3.137:27400,192.168.3.138:27400
```

3. 10. 5 配置副本集

```
shard1  
mongo 192.168.3.133:27100  
rs.initiate( {  
  _id : "shard1",  
  members: [  
    {  
      _id: 1,  
      host: "192.168.3.133:27100",  
      name: "shard1",  
      roles: ["PRIMARY"]  
    },  
    {  
      _id: 2,  
      host: "192.168.3.134:27100",  
      name: "shard2",  
      roles: ["PRIMARY"]  
    },  
    {  
      _id: 3,  
      host: "192.168.3.135:27100",  
      name: "shard3",  
      roles: ["PRIMARY"]  
    }  
  ]  
})
```

```
    { _id: 133, host: "192.168.3.133:27100" },  
    { _id: 134, host: "192.168.3.134:27100" },  
    { _id: 135, host: "192.168.3.135:27100" }  
  ]  
})
```

shard2

mongo 192.168.3.133:27200

```
rs.initiate( {  
  _id : "shard2",  
  members: [  
    { _id: 133, host: "192.168.3.133:27200" },  
    { _id: 134, host: "192.168.3.134:27200" },  
    { _id: 135, host: "192.168.3.135:27200" }  
  ]  
})
```

shard3

mongo 192.168.3.133:27300

```
rs.initiate( {  
  _id : "shard3",  
  members: [  
    { _id: 133, host: "192.168.3.133:27300" },  
    { _id: 134, host: "192.168.3.134:27300" },  
    { _id: 135, host: "192.168.3.135:27300" }  
  ]  
})
```

mongo-config

mongo 192.168.3.136:27400

```
rs.initiate( {  
  _id : "mongo-config",  
  members: [  
    { _id: 136, host: "192.168.3.136:27400"},  
    { _id: 137, host: "192.168.3.137:27400"},  
    { _id: 138, host: "192.168.3.138:27400"}  
  ]  
})
```

3. 10. 6 启用分片

```
mongo 192.168.3.138:27500
use admin
db.runCommand({addshard:"shard1/192.168.3.133:27100,192.168.3.134:27100,192.168.3.135:27100"});
db.runCommand({addshard:"shard2/192.168.3.133:27200,192.168.3.134:27200,192.168.3.135:27200"});
db.runCommand({addshard:"shard3/192.168.3.133:27300,192.168.3.134:27300,192.168.3.135:27300"});
```

第四章 用户和角色

数据库角色介绍:

<https://docs.mongodb.com/manual/reference/built-in-roles/>

4.1 数据库用户角色

Database User Roles 这个是针对非系统数据库和部分系统表的角色组

4.1.1 read

该角色通过授予以下操作来提供读取权限:

- changeStream
- collStats
- dbHash
- dbStats
- find
- killCursors
- listIndexes
- listCollections

4.1.2 readWrite

提供 read 角色的所有权限以及修改所有 非系统集合和 system.js 集合的数据的能力。该角色对这些集合提供以下操作:

- collStats
- convertToCapped
- createCollection
- dbHash
- dbStats
- dropCollection
- createIndex
- dropIndex
- emptycapped
- find
- insert
- killCursors
- listIndexes
- listCollections
- remove
- renameCollectionSameDB
- update

4.2 数据库管理角色

Database Administration Roles 可以操作所有数据库

4.2.1 dbadmin

数据库管理员。提供执行管理任务（如与架构相关的任务，索引和收集统计信息）的功能。此角色不授予用户和角色管理权限。

对系统库提供以下功能：

- collStats
- dbHash
- dbStats
- find
- killCursors
- listIndexes
- listCollections
- dropCollection and createCollection on system.profile only

对非系统库提供以下功能：

- bypassDocumentValidation
- collMod
- collStats
- compact
- convertToCapped
- createCollection
- createIndex
- dbStats
- dropCollection
- dropDatabase
- dropIndex
- enableProfiler
- reIndex
- renameCollectionSameDB
- repairDatabase
- storageDetails
- validate

4.2.2 dbOwner

数据库所有者

数据库所有者可以执行数据库所有管理的操作。这个角色合并了 readWrite, dbAdmin , userAdmin 角色的权限

4.2.3 usreAdmin

用户管理员

提供在当前数据库创建和修改角色和用户的能力。用户管理员角色允许用户授权任意用户的权限，包括它们自己的。

用户管理员角色明确的提供以下的操作：

- changeCustomData
- changePassword
- createRole
- createUser
- dropRole
- dropUser
- grantRole
- revokeRole
- viewRole
- viewUser

4.3 群集管理角色

Cluster Administration Roles 针对整个系统进行管理

4.3.1 clusterAdmin

提供最高集群管理权限。这个角色包括了 clusterManager, clusterMonitor, hostManager 角色的权限，这个角色提供了 dropDatabase 的操作。

可以：

- 分片所有功能；
- 删除数据库
- show dbs
- show collecitons

不能：

插入文档
查看文档内容
删除文档

4.3.2 clusterManager

集群管理者。在集群上提供管理和监视操作。一个拥有此角色用户可以有权管理分别被用来共享、复制的设置和本地数据库

把集群看成一个整体的基础上提供以下操作：

- addShard
- applicationMessage
- cleanupOrphaned
- flushRouterConfig
- listShards
- removeShard
- replSetConfigure
- replSetGetStatus
- replSetStateChange
- resync

在集群中的所有数据库提供以下方法：

- enableSharding
- moveChunk
- splitChunk
- splitVector

在配置数据库中，为 settings 集合提供以下的操作

- insert
- remove
- update

在配置数据库里，为 configuration, system.indexes, system.js, system.namespaces 集合提供以下操作：

- collStats
- dbHash
- dbStats
- find
- killCursors

在本地数据库里，为 replset 集合提供以下操作：

- collStats
- dbHash
- dbStats
- find
- killCursors

4.3.3 clusterMonitor

集群监视者。

为监视工具提供只读的权限，包括 MongoDB Cloud Manager 和 Ops Manager monitoring agent 两个工具。

把集群看成一个整体的基础上提供以下操作：

- connPoolStats
- cursorInfo
- getCmdLineOpts
- getLog
- getParameter
- getShardMap
- hostInfo
- inprog
- listDatabases
- listShards
- netstat
- replSetGetStatus
- serverStatus
- shardingState
- top

在集群中的所有数据库提供以下方法：

- collStats
- dbStats
- getShardVersion

为所有在集群里的 system.profile 集合提供 find 操作：

- collStats
- dbHash
- dbStats
- find
- killCursors

4.3.4 hostMonitor

主机管理者。

提供监视和管理服务器的能力。

把集群看成一个整体的基础上提供以下操作：

- applicationMessage
- closeAllDatabases
- connPoolSync
- cpuProfiler
- diagLogging
- flushRouterConfig

- fsync
- invalidateUserCache
- killop
- logRotate
- resync
- setParameter
- shutdown
- touch
- unlock

Provides the following actions on *all* databases in the cluster:

在集群中的所有数据库提供以下方法:

- killCursors
- repairDatabase

4.4 备份和恢复角色

Backup and Restoration Roles 备份还原角色组

4.4.1 backup

备份

4.4.2 restore

还原

4.4.3 clusterAdmin

只在 admin 数据库中可用，赋予用户所有分片和复制集相关函数的管理权限。

4.5 全数据库角色

All-Database Roles 角色里面有一些跟超管差不多了级别了，针对所有数据库的

4.4.4 readAnyDatabase

读任何数据库

4.4.5 readWriteAnyDatabase

读写任何数据库

4.4.6 userAdminAnyDatabase

在所有数据库中管理用户

4.4.7 dbAdminAnyDatabase

任意数据库管理员

4.6 root 超级用户角色

Superuser Roles 超级管理员 不用多说了
只在 admin 数据库中可用。超级账号，超级权限

4.7 内部角色

Internal Role 内部系统角色，比超管牛，别乱设哦
内部角色

__system

系统角色

4.8 小结

数据库用户角色 (Database User Roles) :

- read: 授予 User 只读数据的权限
- readWrite: 授予 User 读写数据的权限

数据库管理角色 (Database Administration Roles) :

- dbAdmin: 在当前 dB 中执行管理操作
- dbOwner: 在当前 DB 中执行任意操作
- userAdmin: 在当前 DB 中管理 User

备份和还原角色 (Backup and Restoration Roles) :

- backup
- restore

跨库角色 (All-Database Roles) :

- readAnyDatabase: 授予在所有数据库上读取数据的权限
- readWriteAnyDatabase: 授予在所有数据库上读写数据的权限
- userAdminAnyDatabase: 授予在所有数据库上管理 User 的权限
- dbAdminAnyDatabase: 授予管理所有数据库的权限

集群管理角色 (Cluster Administration Roles) :

- clusterAdmin: 授予管理集群的最高权限
- clusterManager: 授予管理和监控集群的权限, A user with this role can access the config and local databases, which are used in sharding and replication, respectively.
- clusterMonitor: 授予监控集群的权限, 对监控工具具有 readonly 的权限
- hostManager: 管理 Server

4.9 启用认证

在安装完成 mongodb 分片集群后进行用户认证, 未启用认证时, 登录 mongos, 提醒如下

```
[root@localhost ~]# mongo 127.0.0.1:27500
MongoDB shell version v4.0.8
connecting to: mongodb://127.0.0.1:27500/test?gssapiServiceName=mongodb
Implicit session: session { "id" : UUID("2706f0e0-d744-48e6-8f3c-25f7ab9490c1") }
MongoDB server version: 4.0.8
Server has startup warnings:
2019-04-12T20:00:36.976+0800 I CONTROL [main] ** WARNING: Access control is not enabled for the database.
2019-04-12T20:00:36.976+0800 I CONTROL [main] **      Read and write access to data and configuration is unrestricted.
2019-04-12T20:00:36.976+0800 I CONTROL [main] ** WARNING: You are running this process as the root user, which is not recommended.
2019-04-12T20:00:36.976+0800 I CONTROL [main]
mongo> show dbs;
admin    0.000GB
com      0.012GB
config   0.003GB
```

登录 shard 节点提醒如下

```
[root@localhost ~]# mongo 127.0.0.1:27200
MongoDB shell version v4.0.8
connecting to: mongodb://127.0.0.1:27200/test?gssapiServiceName=mongodb
Implicit session: session { "id" : UUID("0d574cea-b94c-435a-b93d-1f51159d02cf") }
MongoDB server version: 4.0.8
Server has startup warnings:
2019-04-26T09:30:10.834+0800 I CONTROL [initandlisten] ** WARNING: Access control is not enabled for the database.
2019-04-26T09:30:10.834+0800 I CONTROL [initandlisten] **      Read and write access to data and configuration is unrestricted.
2019-04-26T09:30:10.834+0800 I CONTROL [initandlisten] ** WARNING: You are running this process as the root user, which is not recommended.
2019-04-26T09:30:10.834+0800 I CONTROL [initandlisten]
---
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()
---
shard2:SECONDARY>
shard2:SECONDARY>
shard2:SECONDARY>
```

4.9.1 创建超级用户角色

登录 mongos 节点, 注意, 一定是 mongos 节点, 不是 config 或者 shard 节点, 创建 root 超级管理员角色:

```
mongo 127.0.0.1:27500
```

```
use admin
db.createUser({
  user:"root",
  pwd:"920ba0c57f7fe09b",
  roles: [ { role: "root",db:"admin"}]
})
```

4.9.2 创建 keyfile 文件

```
openssl rand -base64 753 > mongo-keyfile.key
```

```
[root@localhost ~]# openssl rand -base64 753 > mongo-keyfile.key
[root@localhost ~]# cat mongo-keyfile.key
DhJm84LO4+hntJ8VZptytNyvIpQs0rnGNrwpBDAb/UM4DfDr8uaVus7M5AqP02tX
Sh+udg9QcFVN5NfBN9GpmeuWnHHB4ASiDj1uZe4+KM0UzCuvKIU+i5CK1jj0fdc3
+byw2+1F8GxJqberMACAAz2+hH7ZIHgd/tmsLuJOREakHMMSdbYqV0ytdImSoewn
HuKfTZc1eMZ0UFHvu33KFLdi6j/A5/5jC7Le7FCs7J40YBeEn4cg0k8/luoSgYwd
XPlwjx5G1MCNiC60q4VsUjyap0SiKO+x8ftMNUhaQfoqH3Pb5YwhLP1dBVUPKFTf
uVaksd7cLXJowXK4iruP7jHBhQaal8fXYY5VwTGaf/ApCKJVIQzHbU84Z9c8djtn
l9mwt6obo+/QhpFZOOxrUv1MkmN5MmVC6yNHmacKam01+D7PVMQYw2FHMreOKfEv
rgPHiiPaOWE/8cDPHdWvwoBa5w9Jth1W0tztLRHktXY2ME/V6j9YAEA1Kf6RngtT
PwmdmVwTVYHlkyDsJtuo+yaATLVd4VxOLY+GGdO+4vj1jh4s+5p1Cc+G5E61t3761
KbtuBieRHHS0PaAJhkBSRAusexipdg/R7XNhwzRx5Znun8ZWf8o3EuCw/ph+h/OK
kscwzU2tpJjuclixQpD7OSD49QhMJcb5PFoPLOIpNjgr0yxn8SekTFuMX/pcMh60
rFi1R11lwEYv51tvNEJj4IZ9O53chgwnw1RECIn3vmJewzE/b6t0+H3HGY/+vUh
J16xL2Xn24iyB4p3McGCHPyE5hVL/qdaJCDLRSTdTk+bwQzu1sFHYiGIwVha0hsa
rSvAlpXbscsLSf2IsP03zFL25Vwpd18DirUen5FTC3u/mJcEuMtuyYZMsfla5keh
lwa+vkyY55okfy90FIeXDV8s03iCeXNEMlroFVWOUdDScv16XbMUR17WQZNIPonf
6aoQhUr1XAPDqeUEgOS/OI9xEbMSncSbfto8MyHcmC47
[root@localhost ~]#
[root@localhost ~]#
```

将 mongo-keyfile.key 拷贝到所有 shard 和 config 节点服务器上的/etc 下面

修改密钥文件权限为 600

```
chmod 600 /etc/mongo-keyfile.key
```

否正无法启动，报错如下

```
[root@localhost etc]# mongod -f /etc/mongo-config.conf
about to fork child process, waiting until server is ready for connections.
forked process: 13188
ERROR: child process failed, exited with error number 1
To see additional information in this output, start without the "--fork" option.
[root@localhost etc]#
[root@localhost etc]#
```

且日志中有如下提示，密钥文件权限太大

```
[root@localhost etc]#
[root@localhost etc]# tail /data/mongodb/config/mongod.log
2019-04-26T10:25:45.145+0800 I STORAGE [signalProcessingThread] Finished shutting down session sweeper thread
2019-04-26T10:25:45.232+0800 I STORAGE [signalProcessingThread] shutdown: removing fs lock...
2019-04-26T10:25:45.232+0800 I CONTROL [signalProcessingThread] now exiting
2019-04-26T10:25:45.232+0800 I CONTROL [signalProcessingThread] shutting down with code:0
2019-04-26T10:26:04.959+0800 I CONTROL [main] ***** SERVER RESTARTED *****
2019-04-26T10:26:04.967+0800 I CONTROL [main] Automatically disabling TLS 1.0, to force-enable TLS 1.0 specify
2019-04-26T10:26:04.985+0800 I ACCESS [main] permissions on /etc/mongo-keyfile.key are too open
2019-04-26T10:29:12.820+0800 I CONTROL [main] ***** SERVER RESTARTED *****
2019-04-26T10:29:12.826+0800 I CONTROL [main] Automatically disabling TLS 1.0, to force-enable TLS 1.0 specify
2019-04-26T10:29:12.838+0800 I ACCESS [main] permissions on /etc/mongo-keyfile.key are too open
[root@localhost etc]#
[root@localhost etc]#
```


4.9.3 启用认证

修改所有 shard 和 config 节点的配置文件，添加如下内容：

security:

keyFile: /etc/mongo-keyfile.key

clusterAuthMode: keyFile

authorization: enabled

```
[root@localhost etc]#  
[root@localhost etc]# vim mongo-shard1.conf  
systemLog:  
  destination: file  
  logAppend: true  
  path: /data/mongodb/shard1/mongod.log  
  
security:  
  keyFile: /etc/mongo-keyfile.key  
  clusterAuthMode: keyFile  
  authorization: enabled  
  
storage:  
  dbPath: /data/mongodb/shard1/data  
  journal:  
    enabled: true  
  
processManagement:  
  fork: true  
  pidFilePath: /data/mongodb/shard1/mongod.pid  
  timeZoneInfo: /usr/share/zoneinfo  
  
net:  
  port: 27100  
  bindIp: 0.0.0.0
```

修改 mongos 配置文件，添加如下两行

security:

keyFile: /etc/mongo-keyfile.key

```
[root@localhost etc]#
[root@localhost etc]# vim mongo-mongos.conf
systemLog:
  destination: file
  logAppend: true
  path: /data/mongodb/mongos/mongod.log

security:
  keyFile: /etc/mongo-keyfile.key

processManagement:
  fork: true
  pidFilePath: /data/mongodb/mongos/mongod.pid
  timeZoneInfo: /usr/share/zoneinfo
```

注意添加的内容和 shard、config 不一样

4.9.4 重启服务

停止所有服务，包括 shard、config、mongos 节点。

按照以下顺序启动各个服务：

- 1) config
- 2) shard
- 3) mongos

排错：

如果出现以下情况，一直起不来服务

```
[root@localhost etc]#
[root@localhost etc]# mongod -f /etc/mongo-shard1.conf
about to fork child process, waiting until server is ready for connections.
forked process: 13793
```

查看日志有如下错误

```
[root@localhost etc]#
[root@localhost etc]# tail /data/mongodb/shard1/mongod.log
2019-04-26T10:40:17.817+0800 W SHARDING [initandlisten] Error initializing shard
ing state, sleeping for 2 seconds and trying again :: caused by :: Unauthorized:
Error loading clusterID :: caused by :: command find requires authentication
2019-04-26T10:40:19.818+0800 W SHARDING [initandlisten] Error initializing shard
ing state, sleeping for 2 seconds and trying again :: caused by :: Unauthorized:
Error loading clusterID :: caused by :: command find requires authentication
2019-04-26T10:40:21.819+0800 W SHARDING [initandlisten] Error initializing shard
ing state, sleeping for 2 seconds and trying again :: caused by :: Unauthorized:
Error loading clusterID :: caused by :: command find requires authentication
2019-04-26T10:40:23.820+0800 W SHARDING [initandlisten] Error initializing shard
ing state, sleeping for 2 seconds and trying again :: caused by :: Unauthorized:
Error loading clusterID :: caused by :: command find requires authentication
2019-04-26T10:40:25.821+0800 W SHARDING [initandlisten] Error initializing shard
ing state, sleeping for 2 seconds and trying again :: caused by :: Unauthorized:
Error loading clusterID :: caused by :: command find requires authentication
2019-04-26T10:40:27.823+0800 W SHARDING [initandlisten] Error initializing shard
ing state, sleeping for 2 seconds and trying again :: caused by :: Unauthorized:
```

因为改节点未启用认证，无法加入集群，因为集群中其他节点已经启用了认证服务。处理方法：检查配置文件是否添加了正确的内容，且使用了集群中共同的 key 文件。

4.9.5 验证认证服务

登录 mongos 服务，看不到 db，看不到集合，且无法插入文档

```
[root@localhost etc]#
[root@localhost etc]# mongo 127.0.0.1:27500
MongoDB shell version v4.0.8
connecting to: mongodb://127.0.0.1:27500/test?gssapiServiceName=mongod
Implicit session: session { "id" : UUID("981b7de8-dab6-4bf8-97ac-4f6e381dcd09") }
MongoDB server version: 4.0.8
mongos>
mongos> show dbs;
mongos>
mongos>
mongos> show collections;
warning: unable to run listCollections, attempting to approximate collection names by parsing connectionStatus
mongos>
mongos> db
test
mongos>
mongos> db.test.insert({"name":"quz1"})
writeCommandError({
  "ok" : 0,
  "errmsg" : "command insert requires authentication",
  "code" : 13,
  "codeName" : "Unauthorized",
  "operationTime" : Timestamp(1556248456, 1),
  "$clusterTime" : {
    "clusterTime" : Timestamp(1556248456, 1),
    "signature" : {
      "hash" : BinData(0,"lCp+mtikwNDMerg/sv3MjYKYgk="),
      "keyid" : NumberLong("6678968702727094301")
    }
  }
})
mongos>
```

直接进入 shard，无法查看 db 和 collections，也无法插入文档

```
shard1:PRIMARY>
shard1:PRIMARY> show dbs;
shard1:PRIMARY>
shard1:PRIMARY> db
test
shard1:PRIMARY>
shard1:PRIMARY> show collections;
warning: unable to run listCollections, attempting to approximate collection names by parsing connectionStatus
shard1:PRIMARY>
shard1:PRIMARY> db.test.insert({"name":"quz1"})
writeCommandError({
  "operationTime" : Timestamp(1556248645, 1),
  "ok" : 0,
  "errmsg" : "not authorized on test to execute command { insert: '\\test\\', ordered: true, lsid: { id: U
amp(1556248647, 3), signature: { hash: BinData(0, 10EC8EF25FF829CC138A849A7AF86F53578607AE), keyId: 6678968702
  "code" : 13,
  "codeName" : "Unauthorized",
  "$gleStats" : {
    "lastOptTime" : Timestamp(0, 0),
    "electionId" : ObjectId("7fffffff000000000000000004")
  },
  "lastCommittedOptTime" : Timestamp(1556248645, 1),
  "$configServerState" : {
    "opTime" : {
      "ts" : Timestamp(1556248654, 2),
      "t" : NumberLong(2)
    }
  },
  "$clusterTime" : {
    "clusterTime" : Timestamp(1556248654, 2),
```

使用之前创建的 root 用户登录，一切正常

```
[root@localhost etc]#
[root@localhost etc]# mongo 127.0.0.1:27500/admin -u root -p
MongoDB shell version v4.0.8
Enter password:
connecting to: mongodb://127.0.0.1:27500/admin?gssapiServiceName=mongod
Implicit session: session { "id" : UUID("c1734f98-df18-454a-8298-4f33989f2a40") }
MongoDB server version: 4.0.8
Server has startup warnings:
2019-04-26T11:08:34.823+0800 I CONTROL [main] ** WARNING: You are running this process as the root user, which is not recommended.
2019-04-26T11:08:34.823+0800 I CONTROL [main]
mongos>
mongos> show dbs;
admin 0.000GB
com 0.012GB
config 0.003GB
mongos>
mongos>
```

同时由于，我们使用的是 root 角色，会有一个提醒。后续使用中，应该为每个库创建单独的用户，具体查看前面的角色介绍。

同时，这里登录时就要指定库，否正无法登录，比如上面，我指定了 admin 这个系统库

4.9.6 启用认证

登录 config 节点，可以看到创建的用户其实位于 admin 库里面，而不是在 shard 节点上。

```
[root@localhost ~]# mongo 127.0.0.1:27400/admin -u root -p
MongoDB shell version v4.0.8
Enter password:
connecting to: mongodb://127.0.0.1:27400/admin?gssapiServiceName=mongodb
Implicit session: session { "id" : UUID("c48497e5-5453-4b39-b87c-5af5a6d6291f") }
MongoDB server version: 4.0.8
Server has startup warnings:
```

```
@(shell):1.1
mongo-config:PRIMARY> db
admin
mongo-config:PRIMARY> show dbs
admin      0.000GB
config     0.003GB
local      0.046GB
mongo-config:PRIMARY> show users
{
  "_id" : "admin.root",
  "user" : "root",
  "db" : "admin",
  "roles" : [
    {
      "role" : "root",
      "db" : "admin"
    }
  ],
  "mechanisms" : [
    "SCRAM-SHA-1",
    "SCRAM-SHA-256"
  ]
}
```

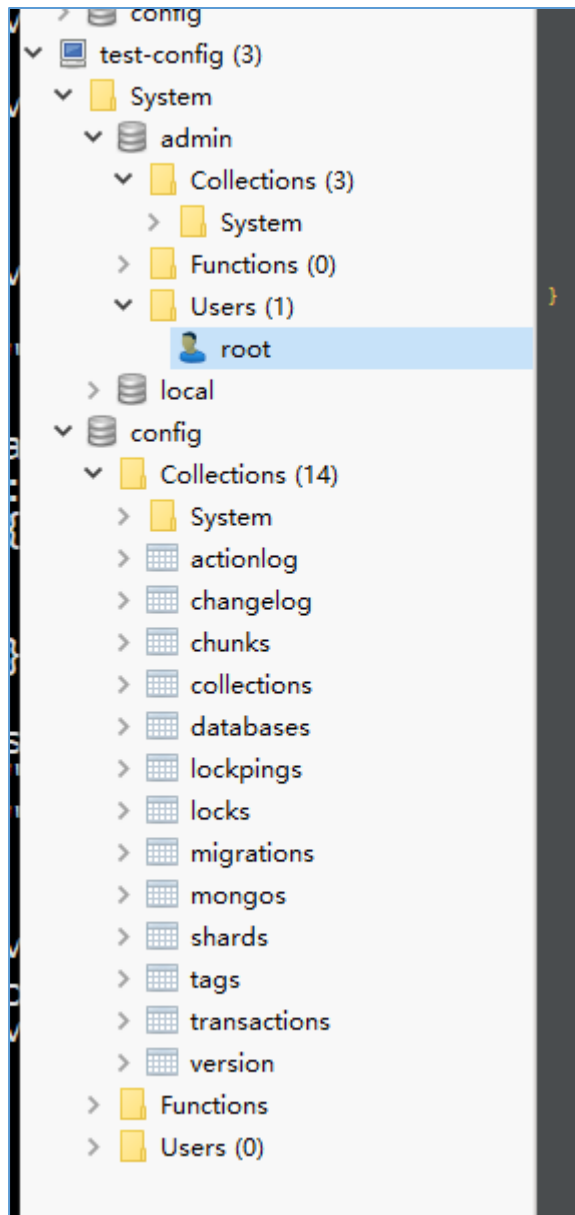
使用前面创建的 root 账号，无法直接登录 shard，因为认证信息保存在 config 上

```

[root@localhost ~]#
[root@localhost ~]# mongo 127.0.0.1:27100/admin -u root -p
MongoDB shell version v4.0.8
Enter password:
connecting to: mongodb://127.0.0.1:27100/admin?gssapiServiceName=mongodb
2019-04-26T12:01:12.493+0800 E QUERY [js] Error: Authentication failed. :
connect@src/mongo/shell/mongo.js:343:13
@(connect):2:6
exception: connect failed
[root@localhost ~]#

```

登录 config 节点



可以看到分片、数据库、日志、用户等信息。

4.10 用户管理

创建用户

```
use admin
db.createUser(
  {
    "user" : "clusteradmin",
    "pwd" : "f13760fe93f8d1c",
    roles: [
      { "role" : "clusterAdmin", "db" : "admin" },
      { role: 'userAdminAnyDatabase', db:'admin' },
    ]
  }
)
```

```
use admin
db.createUser({
  user:"root",
  pwd:"920ba0c57f7fe09b",
  roles: [ { role: "root",db:"admin"}]
})
```

```
use dm-ai
db.createUser(
...   {
...     user: "dm-ai-admin",
...     pwd: "dm-ai-admin",
...     roles: [
...       { role: "readWrite", db: "dm-ai" }
...     ]
...   }
... )
```

更新用户角色

其语法和创建用户类似，比如如下语句，不会修改密码

```
db.updateUser(
  "clusteradmin",
  {
    roles:[
```

```
    { role: 'clusterAdmin', db:'admin' },  
    { role: 'userAdminAnyDatabase', db:'admin' },  
  ]  
}  
)
```

删除用户

```
db.dropUser('dm-ai-admin')
```

第五章 mongodb 基本

5.1 简介

Mongodb 和关系型数据库对应关系

数据库 <----> 数据库

集 合 <----> 表

文 档 <----> 行

Mongodb 的特点:

- 1) 丰富的数据类型
- 2) 扩展了关系型数据库的众多重要功能, 对 sqlserver 进行了一种补充容易扩展
- 3) 丰富的功能
- 4) 不牺牲速度

5.2 系统数据库

admin 数据库: 放一些用户信息, 性能管理信息。

local 数据库: 存放一些操作日志【系统日志】。

config 数据库: 如果做 sharing 的时候, 用得到。

5.3 数据类型

- 1) 布尔
- 2) null
- 3) 字符串
- 4) 对象
- 5) 数字
- 6) 数组

缺陷:

- 1) 没有日期类型
- 2) 数字中不能区分整数和浮点数

5.4 无模式

文档: 无模式的, 就是一个集合中的多个文档的结构可以是任意的。


```
collection:
=> document:
=> {name:"jack",age:20}
    {bookname:"mongodb",date:"2015-12-19"}
```

第一个文档和第二个文档毫无关系，【无模式的概念】

5.5 ObjectId 结构

一共有 12 字节构成：

0-3: 时间戳

4-6: 机器标识，系统根据各节点的 MAC 地址、硬件标识等生产的，每台机器都不同，方便做集群

7-8: pid

9-11: 计数器

```
> db.test.find()
{ "_id" : ObjectId("5cb2a8cdd040e92c2365e8d3"), "name" : "quzl" }
```

第六章 mongo-shell

6.1 数据库操作

连接数据库

如果不指定数据库，默认是 local

```
mongodb/bin/mongo 127.0.0.1:27017
```

```
mongodb/bin/mongo 127.0.0.1:27017/admin
```

创建数据库

```
use quzl
```

进入数据库

```
use quzl
```

显示数据库(空的数据库不会显示，需要先插入数据才可以显示)

```
show dbs
```

显示当前在哪个库

```
db
```

删除数据库(先进入数据库，再执行删除命令)

```
use quzl
```

```
db.dropDatabase()
```

查看数据库状态

```
> db.stats();
```

```
{
  "db" : "test",           数据库名称
  "collections" : 4,       集合输了
  "views" : 0,             视图输了
  "objects" : 4,           集合中的文档数据
  "avgObjSize" : 175,      平均每个文档的大小， = dataSize/ objects
  "dataSize" : 700,        数据实际大小
  "storageSize" : 73728,   数据库占有磁盘大小，包括预分配的大小
  "numExtents" : 0,        数据库中所有集合包含的 extent 数量
  "indexes" : 5,           索引数据
  "indexSize" : 90112,     索引大小，单位字节
  "fsUsedSize" : 13135446016, 文件系统实际使用大小
  "fsTotalSize" : 50437033984, 为其分配了多少
  "ok" : 1
}
```

连接副本集：

```
mongodb://xmc:xmc2018@192.168.3.226:27017,192.168.3.227:27017,192.168.3.228:27017/
```

dm-xmc?replicaSet=rs0&authSource=admin

连接分片集群:

```
mongodb://mis-test:mis-  
test@192.168.3.136:27500,192.168.3.137:27500,192.168.3.138:27500/mis-  
test?authSource=mis-test
```

6.2 集合操作

创建集合

```
db.createCollection("runoob")
```

创建集合 mycol,

capped: 默认 false, 设置为 true 表示创建固定大小的文档, 当为 true 时, 必须设置 size, 可以不设置 max。达到最大值后, 最新插入的文档将会覆盖最早的文档

size: 单位字节, 表示集合的最大大小

max: 表示集合中的最大文档数量

autoIndexId:true 表示自动在 _id 字段创建索引

```
db.createCollection("mycol", { capped : true, autoIndexId : true, size : 1024000000, max :  
10000 })
```

直接插入数据, 当集合不存在时自动创建

```
db.mycol.insert({"name" : "quzl"})
```

查看有哪些集合

```
show collections;
```

删除 (hello) 集合

```
db.hello.drop()
```

6.3 文档操作

4.9.7 插入文档

```
db.col.insert({title: 'MongoDB 教程',  
description: 'MongoDB 是一个 Nosql 数据库',
```

```

    by: '菜鸟教程',
    url: 'http://www.runoob.com',
    tags: ['mongodb', 'database', 'NoSQL'],
    likes: 100
  })

```

也可以将文档内容定义为一个变量，然后插入

```

document={title: 'MongoDB 教程',
  description: 'MongoDB 是一个 Nosql 数据库',
  by: '菜鸟教程',
  url: 'http://www.runoob.com',
  tags: ['mongodb', 'database', 'NoSQL'],
  likes: 100
});
db.col.insert(document)

```

```

> document={title: 'MongoDB 教程',
...   description: 'MongoDB 是一个 Nosql 数据库',
...   by: '菜鸟教程',
...   url: 'http://www.runoob.com',
...   tags: ['mongodb', 'database', 'NoSQL'],
...   likes: 100
... };
{
  "title": "MongoDB 教程",
  "description": "MongoDB 是一个 Nosql 数据库",
  "by": "菜鸟教程",
  "url": "http://www.runoob.com",
  "tags": [
    "mongodb",
    "database",
    "NoSQL"
  ],
  "likes": 100
}
>
> db.mycol.insert(document)
WriteResult({"nInserted": 1 })
> db.mycol.find()
{"_id": ObjectId("5cb5f5084152d8e05a6868fc"), "title": "MongoDB 教程", "description": "MongoDB 是一个 Nosql 数据库", "tags": [ "mongodb", "database", "NoSQL" ], "likes": 100 }
>

```

4.9.8 查询文档

也可以通过 pretty()方法，显示为 json 格式

```

db.mycol.find()
db.mycol.find().pretty()

```

```
>
> db.mycol.find().pretty()
{
  "_id" : ObjectId("5cb5f5084152d8e05a6868fc"),
  "title" : "MongoDB 教程",
  "description" : "MongoDB 是一个 Nosql 数据库",
  "by" : "菜鸟教程",
  "url" : "http://www.runoob.com",
  "tags" : [
    "mongodb",
    "database",
    "NoSQL"
  ],
  "likes" : 100
}
```

条件查询

```
db.col.find({"likes":100})
```

使用条件操作符查询

操作	格式	范例	RDBMS中的类似语句
等于	{<key>:<value>}	db.col.find({"by":"菜鸟教程"}).pretty()	where by = '菜鸟教程'
小于	{<key>:{<\$lt>:<value>}}	db.col.find({"likes":{\$lt:50}}).pretty()	where likes < 50
小于或等于	{<key>:{<\$lte>:<value>}}	db.col.find({"likes":{\$lte:50}}).pretty()	where likes <= 50
大于	{<key>:{<\$gt>:<value>}}	db.col.find({"likes":{\$gt:50}}).pretty()	where likes > 50
大于或等于	{<key>:{<\$gte>:<value>}}	db.col.find({"likes":{\$gte:50}}).pretty()	where likes >= 50
不等于	{<key>:{<\$ne>:<value>}}	db.col.find({"likes":{\$ne:50}}).pretty()	where likes != 50

And 条件

```
db.col.find({"by":"菜鸟教程","title":"MongoDB 教程"}).pretty()
```

or 条件

```
db.col.find({$or:[{"by":"菜鸟教程"},"title":"MongoDB 教程"]}).pretty()
```

and 和 or 组合

```
db.col.find({"likes":{$gt:50}, $or: [{"by":"菜鸟教程"},"title":"MongoDB 教程"]}).pretty()
```

相当于条件'where likes>50 AND (by = '菜鸟教程' OR title = 'MongoDB 教程')'

4.9.9 查询优化显示

使用 limit()和 skip () 方法, 下面语句表示从第 101 行开始显示前 10 行

```
db.col.find().limit(10).skip(100).pretty()
```

排序, 如下-1 表示降序, 1 表示升序

```
db.col.find().sort({"likes":-1})
```

注意: skip(), limilt(), sort()三个放在一起执行的时候, 执行的顺序是先 sort(), 然后是 skip(), 最后是显示的 limit()

4.9.10 更新文档

```
db.col.update({'title':'MongoDB 教程'},{$set:{'title':'MongoDB'}})
```

前面是更新条件，后面是更新的内容

```
> db.col.find().pretty()
{
  "_id" : ObjectId("5cb5f6924152d8e05a6868fd"),
  "title" : "MongoDB 教程",
  "description" : "MongoDB 是一个 Nosql 数据库",
  "by" : "菜鸟教程",
  "url" : "http://www.runoob.com",
  "tags" : [
    "mongodb",
    "database",
    "NoSQL"
  ],
  "likes" : 100
}
>
>
> db.col.update({'title':'MongoDB 教程'},{$set:{'title':'MongoDB'}})
writeResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
>
> db.col.find().pretty()
{
  "_id" : ObjectId("5cb5f6924152d8e05a6868fd"),
  "title" : "MongoDB",
  "description" : "MongoDB 是一个 Nosql 数据库",
  "by" : "菜鸟教程",
  "url" : "http://www.runoob.com",
  "tags" : [
    "mongodb",
    "database",
    "NoSQL"
  ],
  "likes" : 100
}
>
```

4.9.11 删除文档

删除所有

```
db.col.remove({})
```

删除指定条件的

```
db.col.remove({'title':"MongoDB"})
```

只删除匹配到的第一个

```
db.col.remove({'title':"MongoDB 教程"},1)
```

```

> db.col.find({"title":"MongoDB 教程"})
{ "_id" : ObjectId("5cb5f8454152d8e05a6868fe"), "title" : "MongoDB 教程", "description" : "MongoDB 教程", "tags" : [ "mongodb", "database", "NoSQL" ], "likes" : 100 }
{ "_id" : ObjectId("5cb5f84d4152d8e05a6868ff"), "title" : "MongoDB 教程", "description" : "MongoDB 教程", "tags" : [ "mongodb", "database", "NoSQL" ], "likes" : 10000 }
>
> db.col.remove({"title":"MongoDB 教程"},1)
WriteResult({ "nRemoved" : 1 })
>
> db.col.find({"title":"MongoDB 教程"})
{ "_id" : ObjectId("5cb5f84d4152d8e05a6868ff"), "title" : "MongoDB 教程", "description" : "MongoDB 教程", "tags" : [ "mongodb", "database", "NoSQL" ], "likes" : 10000 }
>
>

```

6.4 索引操作

6.4.1 查看索引

下面表示查看当前集合有哪些索引

```
db.col.getIndexes()
```

```

> db.col.getIndexes()
[
  {
    "v" : 2,
    "key" : { "_id" : 1 },
    "name" : "_id_",
    "ns" : "test.col"
  },
  {
    "v" : 2,
    "key" : { "title" : 1 },
    "name" : "title_1",
    "ns" : "test.col"
  },
  {
    "v" : 2,
    "key" : { "likes" : 1 },
    "name" : "likes_1",
    "ns" : "test.col"
  }
]

```

如图中第一个表示，test 库的 col 集合有一个索引，索引名字是_id_，索引的 key 是_id，1 表示升序，-1 表示降序

查看索引大小，单位字节

```
db.col.totalIndexSize()
```

6.4.2 创建索引

如下创建索引

```
db.col.createIndex({"title":1})
```

创建索引会阻塞当前的所有的读写操作，如下添加 background: true，表示后台执行，不阻断当前的读写操作，该值默认是 false,当然还有很多其他参数

```
db.col.createIndex({"title":1}, {background: true})
```

创建复合索引

```
db.col.createIndex({"a":1, "b": -1}, {background: true});
```

创建 hash 索引，在 qztl 的 name 上创建 hash 索引

```
db.qztl.ensureIndex({"name": "hashed"})
```

查看创建好的 hash 索引

```
mongos>
mongos> db.users.getIndexes()
[
  {
    "v" : 2,
    "key" : {
      "_id" : 1
    },
    "name" : "_id_",
    "ns" : "a1.users"
  },
  {
    "v" : 2,
    "key" : {
      "name" : "hashed"
    },
    "name" : "name_hashed",
    "ns" : "a1.users"
  }
]
mongos>
```

6.4.3 删除索引

如下，索引名字是唯一的，所以可以根据索引名字做删除

```
db.col.dropIndex("title_1")
```



```

> db.col.dropIndex("title_1")
{ "nIndexesWas" : 3, "ok" : 1 }
>
> db.col.getIndexes()
[
  {
    "v" : 2,
    "key" : { "_id" : 1
    },
    "name" : "_id_",
    "ns" : "test.col"
  },
  {
    "v" : 2,
    "key" : { "likes" : 1
    },
    "name" : "likes_1",
    "ns" : "test.col"
  }
]

```

删除所有索引

`db.col.dropIndexes()`

6.4.4 索引的限制

- 集合中索引不能超过 64 个
- 索引名的长度不能超过 128 个字符
- 一个复合索引最多可以有 31 个字段
- 由于索引是存储在内存(RAM)中,你应该确保该索引的大小不超过内存的限制。
- 如果索引的大小大于内存的限制, MongoDB 会删除一些索引, 这将导致性能下降。

第七章 故障排查

7.1 CPU 使用率过高

现在 mongod 进程 cpu 使用率超过 500%，负载 30 以上。

使用管理员登录 mongodb,输入如下命令，可查看当前正在进行的操作

```
db.currentOp()
```

重点关注几个字段

- client: 请求是由哪个客户端发起的?
- opid: 操作的 opid, 有需要的话, 可以通过 db.killOp(opid) 直接干掉的操作
- secs_running/microsecs_running: 这个值重点关注, 代表请求运行的时间, 如果这个值特别大, 就得注意了, 看看请求是否合理
- query/ns: 这个能看出是对哪个集合正在执行什么操作
- lock*: 还有一些跟锁相关的参数, 需要了解可以看官网文档, 本文不做详细介绍

我这里就发现 microsecs_running 时间特别长, 一百多秒的都有, 进而分析语句, 创建索引后解决。