

# MongoDB集群和安全

---

## 课程目标

---

- MongoDB的副本集：操作、主要概念、故障转移、选举规则
- MongoDB的分片集群：概念、优点、操作、分片策略、故障转移
- MongoDB的安全认证

## 1. 副本集-Replica Sets

---

### 1.1 简介

---

MongoDB中的副本集 ( Replica Set ) 是一组维护相同数据集的mongod服务。副本集可提供冗余和高可用性，是所有生产部署的基础。

也可以说，副本集类似于有自动故障恢复功能的主从集群。通俗的讲就是用多台机器进行同一数据的异步同步，从而使多台机器拥有同一数据的多个副本，并且当主库当掉时在不需要用户干预的情况下自动切换其他备份服务器做主库。而且还可以利用副本服务器做只读服务器，实现读写分离，提高负载。

#### ( 1 ) 冗余和数据可用性

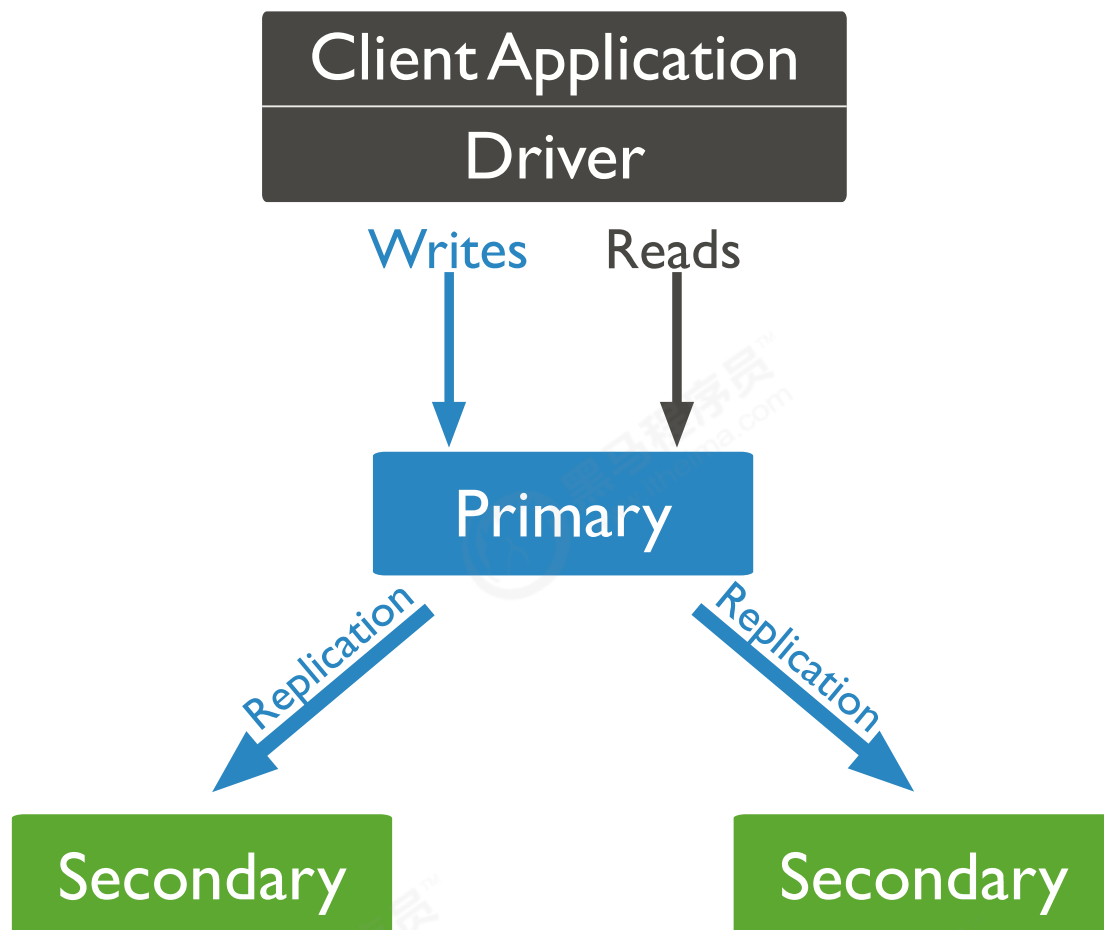
复制提供冗余并提高数据可用性。通过在不同数据库服务器上提供多个数据副本，复制可提供一定级别的容错功能，以防止丢失单个数据库服务器。

在某些情况下，复制可以提供增加的读取性能，因为客户端可以将读取操作发送到不同的服务上，在不同数据中心维护数据副本可以增加分布式应用程序的数据位置和可用性。您还可以为专用目的维护其他副本，例如灾难恢复，报告或备份。

#### ( 2 ) MongoDB中的复制

副本集是一组维护相同数据集的mongod实例。副本集包含多个数据承载节点和可选的一个仲裁节点。在承载数据的节点中，一个且仅一个成员被视为主节点，而其他节点被视为次要（从）节点。

主节点接收所有写操作。副本集只能有一个主要能够确认具有{w：“most”}写入关注的写入；虽然在某些情况下，另一个mongod实例可能暂时认为自己也是主要的。主要记录其操作日志中的数据集的所有更改，即oplog。



辅助(副本)节点复制主节点的oplog并将操作应用于其数据集，以使辅助节点的数据集反映主节点的数据集。如果主要人员不在，则符合条件的节点将举行选举以选出新的主要人员。

### (3) 主从复制和副本集区别

主从集群和副本集最大的区别就是副本集没有固定的“主节点”；整个集群会选出一个“主节点”，当其挂掉后，又在剩下的从节点中选中其他节点为“主节点”，副本集总有一个活跃点(主、primary)和一个或多个备份节点(从、secondary)。

## 1.2 副本集的三个角色

副本集有两种类型三种角色

两种类型：

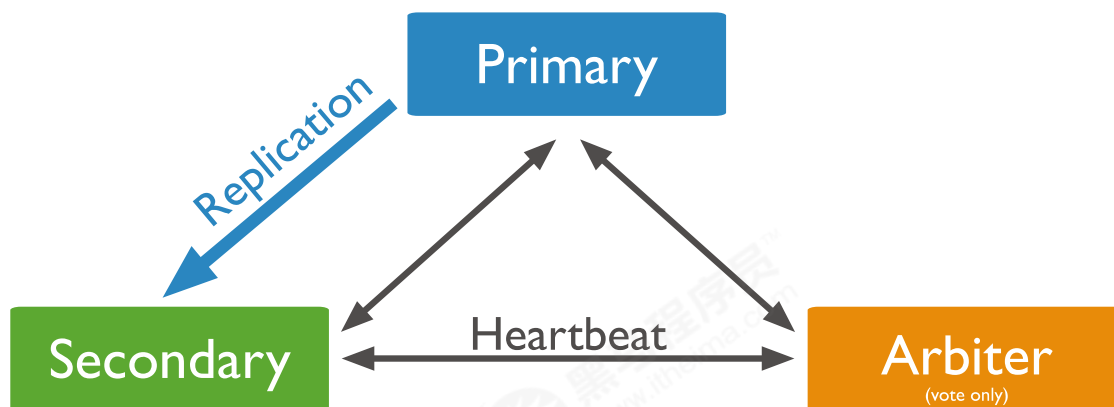
- 主节点 (Primary) 类型：数据操作的主要连接点，可读写。
- 次要 (辅助、从) 节点 (Secondaries) 类型：数据冗余备份节点，可以读或选举。

三种角色：

主要成员 (Primary)：主要接收所有写操作。就是主节点。

副本成员 (Replicate)：从主节点通过复制操作以维护相同的数据集，即备份数据，不可写操作，但可以读操作 (但需要配置)。是默认的一种从节点类型。

仲裁者 (Arbiter)：不保留任何数据的副本，只具有投票选举作用。当然也可以将仲裁服务器维护为副本集的一部分，即副本成员同时也可以作为仲裁者。也是一种节点类型。



关于仲裁者的额外说明：

您可以将额外的mongod实例添加到副本集作为仲裁者。仲裁者不维护数据集。仲裁者的目的是通过响应其他副本集成员的心跳和选举请求来维护副本集中的仲裁。因为它们不存储数据集，所以仲裁器可以是提供副本集仲裁功能的好方法，其资源成本比具有数据集的全功能副本集成员更便宜。

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。仲裁者不需要专用硬件。

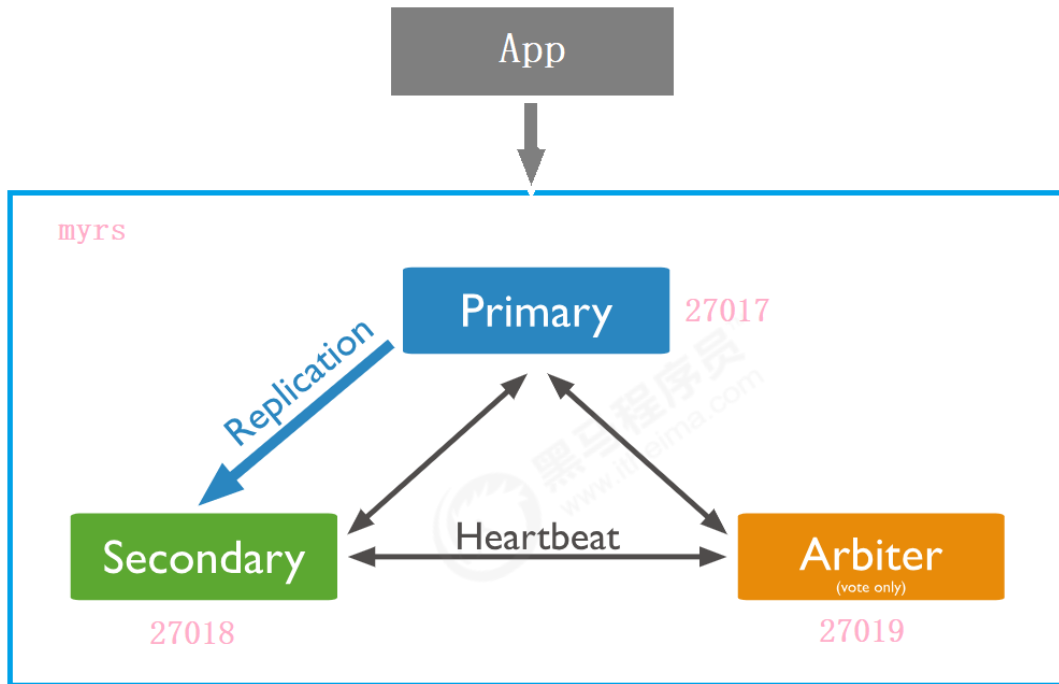
仲裁者将永远是仲裁者，而主要人员可能会退出并成为次要人员，而次要人员可能成为选举期间的主要人员。

如果你的副本+主节点的个数是偶数，建议加一个仲裁者，形成奇数，容易满足大多数的投票。

如果你的副本+主节点的个数是奇数，可以不加仲裁者。

## 1.3 副本集架构目标

一主一副本一仲裁



## 1.4 副本集的创建

### 1.4.1 第一步：创建主节点

建立存放数据和日志的目录

```
#-----myrs
#主节点
mkdir -p /mongodb/replica_sets/myrs_27017/log \ &
mkdir -p /mongodb/replica_sets/myrs_27017/data/db
```

新建或修改配置文件：

```
vim /mongodb/replica_sets/myrs_27017/mongod.conf
```

myrs\_27017：

```
systemLog:
  #MongoDB发送所有日志输出的目标指定为文件
  destination: file
  #mongod或mongos应向其发送所有诊断日志记录信息的日志文件的路径
  path: "/mongodb/replica_sets/myrs_27017/log/mongod.log"
  #当mongos或mongod实例重新启动时，mongos或mongod会将新条目附加到现有日志文件的末尾。
  logAppend: true
storage:
  #mongod实例存储其数据的目录。storage.dbPath设置仅适用于mongod。
  dbPath: "/mongodb/replica_sets/myrs_27017/data/db"
  journal:
```

```
#启用或禁用持久性日志以确保数据文件保持有效和可恢复。
enabled: true
processManagement:
  #启用在后台运行mongos或mongod进程的守护进程模式。
  fork: true
  #指定用于保存mongos或mongod进程的进程ID的文件位置，其中mongos或mongod将写入其PID
  pidFilePath: "/mongodb/replica_sets/myrs_27017/log/mongod.pid"
net:
  #服务实例绑定所有IP，有副作用，副本集初始化的时候，节点名字会自动设置为本地域名，而不是ip
  #bindIpAll: true
  #服务实例绑定的IP
  bindIp: localhost,192.168.0.2
  #bindIp
  #绑定的端口
  port: 27017
replication:
  #副本集的名称
  replSetName: myrs
```

启动节点服务：

```
[root@bobohost replica_sets]# /usr/local/mongodb/bin/mongod -f
/mongodb/replica_sets/myrs_27017/mongod.conf
about to fork child process, waiting until server is ready for connections.
forked process: 54257
child process started successfully, parent exiting
```

## 1.4.2 第二步：创建副本节点

建立存放数据和日志的目录

```
#-----myrs
#副本节点
mkdir -p /mongodb/replica_sets/myrs_27018/log \ &
mkdir -p /mongodb/replica_sets/myrs_27018/data/db
```

新建或修改配置文件：

```
vim /mongodb/replica_sets/myrs_27018/mongod.conf
```

myrs\_27018：

```
systemLog:
  #MongoDB发送所有日志输出的目标指定为文件
  destination: file
  #mongod或mongos应向其发送所有诊断日志记录信息的日志文件的路径
  path: "/mongodb/replica_sets/myrs_27018/log/mongod.log"
  #当mongos或mongod实例重新启动时，mongos或mongod会将新条目附加到现有日志文件的末尾。
```

```
logAppend: true
storage:
  #mongod实例存储其数据的目录。storage.dbPath设置仅适用于mongod。
  dbPath: "/mongodb/replica_sets/myrs_27018/data/db"
  journal:
    #启用或禁用持久性日志以确保数据文件保持有效和可恢复。
    enabled: true
processManagement:
  #启用在后台运行mongos或mongod进程的守护进程模式。
  fork: true
  #指定用于保存mongos或mongod进程的进程ID的文件位置，其中mongos或mongod将写入其PID
  pidFilePath: "/mongodb/replica_sets/myrs_27018/log/mongod.pid"
net:
  #服务实例绑定所有IP，有副作用，副本集初始化的时候，节点名字会自动设置为本地域名，而不是ip
  #bindIpAll: true
  #服务实例绑定的IP
  bindIp: localhost,192.168.0.2
  #bindIp
  #绑定的端口
  port: 27018
replication:
  #副本集的名称
  replSetName: myrs
```

启动节点服务：

```
[root@bobohost replica_sets]# /usr/local/mongodb/bin/mongod -f
/mongodb/replica_sets/myrs_27018/mongod.conf
about to fork child process, waiting until server is ready for connections.
forked process: 54361
child process started successfully, parent exiting
```

### 1.4.3 第三步：创建仲裁节点

建立存放数据和日志的目录

```
#-----myrs
#仲裁节点
mkdir -p /mongodb/replica_sets/myrs_27019/log \ &
mkdir -p /mongodb/replica_sets/myrs_27019/data/db
```

仲裁节点：

新建或修改配置文件：

```
vim /mongodb/replica_sets/myrs_27019/mongod.conf
```

myrs\_27019：

```
systemLog:
  #MongoDB发送所有日志输出的目标指定为文件
  destination: file
  #mongod或mongos应向其发送所有诊断日志记录信息的日志文件的路径
  path: "/mongodb/replica_sets/myrs_27019/log/mongod.log"
  #当mongos或mongod实例重新启动时，mongos或mongod会将新条目附加到现有日志文件的末尾。
  logAppend: true
storage:
  #mongod实例存储其数据的目录。storage.dbPath设置仅适用于mongod。
  dbPath: "/mongodb/replica_sets/myrs_27019/data/db"
  journal:
    #启用或禁用持久性日志以确保数据文件保持有效和可恢复。
    enabled: true
processManagement:
  #启用在后台运行mongos或mongod进程的守护进程模式。
  fork: true
  #指定用于保存mongos或mongod进程的进程ID的文件位置，其中mongos或mongod将写入其PID
  pidFilePath: "/mongodb/replica_sets/myrs_27019/log/mongod.pid"
net:
  #服务实例绑定所有IP，有副作用，副本集初始化的时候，节点名字会自动设置为本地域名，而不是ip
  #bindIpAll: true
  #服务实例绑定的IP
  bindIp: localhost,192.168.0.2
  #bindIp
  #绑定的端口
  port: 27019
replication:
  #副本集的名称
  replSetName: myrs
```

启动节点服务：

```
[root@bobohost replica_sets]# /usr/local/mongodb/bin/mongod -f
/mongodb/replica_sets/myrs_27019/mongod.conf
about to fork child process, waiting until server is ready for connections.
forked process: 54410
child process started successfully, parent exiting
```

## 1.4.5 第四步：初始化配置副本集和主节点

使用客户端命令连接任意一个节点，但这里尽量要连接主节点(27017节点)：

```
/usr/local/mongodb/bin/mongo --host=180.76.159.126 --port=27017
```

结果，连接上之后，很多命令无法使用，，比如 show dbs 等，必须初始化副本集才行

准备初始化新的副本集：

语法：

```
rs.initiate(configuration)
```

选项：

Parameter	Type	Description
<code>configuration</code>	document	Optional. A document that specifies <a href="#">configuration</a> for the new replica set. If a configuration is not specified, MongoDB uses a default replica set configuration.

【示例】

使用默认的配置来初始化副本集：

```
rs.initiate()
```

执行结果：

```
> rs.initiate()
{
  "info2" : "no configuration specified. Using a default configuration for
the set",
  "me" : "180.76.159.126:27017",
  "ok" : 1,
  "operationTime" : Timestamp(1565760476, 1),
  "$clusterTime" : {
    "clusterTime" : Timestamp(1565760476, 1),
    "signature" : {
      "hash" : BinData(0,"AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA="),
      "keyId" : NumberLong(0)
    }
  }
}
myrs:SECONDARY>
myrs:PRIMARY>
```

提示：

- 1) “ok”的值为1，说明创建成功。
- 2) 命令行提示符发生变化，变成了一个从节点角色，此时默认不能读写。稍等片刻，回车，变成主节点。

## 1.4.6 第五步：查看副本集的配置内容

说明：

返回包含当前副本集配置的文件。

语法：

```
rs.conf(configuration)
```

提示：



`rs.config()` 是该方法的别名。

`configuration` : 可选, 如果没有配置, 则使用默认主节点配置。

### 【示例】

在27017上执行副本集中当前节点的默认节点配置

```
myrs:PRIMARY> rs.conf()
{
  "_id" : "myrs",
  "version" : 1,
  "protocolVersion" : NumberLong(1),
  "writeConcernMajorityJournalDefault" : true,
  "members" : [
    {
      "_id" : 0,
      "host" : "180.76.159.126:27017",
      "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("5d539bdcd6a308e600d126bb")
  }
}
```

说明：

- 1) `"_id" : "myrs"` : 副本集的配置数据存储的主键值, 默认就是副本集的名字
- 2) `"members"` : 副本集成员数组, 此时只有一个: `"host" : "180.76.159.126:27017"`, 该成员不是仲裁节点: `"arbiterOnly" : false`, 优先级 (权重值) : `"priority" : 1`,
- 3) `"settings"` : 副本集的参数配置。

提示：副本集配置的查看命令，本质是查询的是 `system.replset` 的表中的数据：

```
myrs:PRIMARY> use local
switched to db local
myrs:PRIMARY> show collections
oplog.rs
replset.election
replset.invalid
replset.oplogTruncateAfterPoint
startup_log
system.replset
system.rollback.id
myrs:PRIMARY> db.system.replset.find()
{ "_id" : "myrs", "version" : 1, "protocolVersion" : NumberLong(1),
  "writeConcernMajorityJournalDefault" : true, "members" : [ { "_id" : 0, "host" :
  "180.76.159.126:27017", "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("5d539bdc6a308e600d126bb") } }
myrs:PRIMARY>
```

## 1.4.7 第六步：查看副本集状态

检查副本集状态。

说明：

返回包含状态信息的文档。此输出使用从副本集的其他成员发送的心跳包中获得的数据反映副本集的当前状态。

语法：

```
rs.status()
```

### 【示例】

在27017上查看副本集状态：

```
myrs:PRIMARY> rs.status()
{
  "set" : "myrs",
  "date" : ISODate("2019-08-14T05:29:45.161Z"),
  "myState" : 1,
  "term" : NumberLong(1),
  "syncingTo" : "",
  "syncSourceHost" : "",
  "syncSourceId" : -1,
  "heartbeatIntervalMillis" : NumberLong(2000),
  "optimes" : {
    "lastCommittedOpTime" : {
      "ts" : Timestamp(1565760578, 1),
      "t" : NumberLong(1)
    }
  }
}
```

```

    },
    "readConcernMajorityOpTime" : {
      "ts" : Timestamp(1565760578, 1),
      "t" : NumberLong(1)
    },
    },
    "appliedOpTime" : {
      "ts" : Timestamp(1565760578, 1),
      "t" : NumberLong(1)
    },
    },
    "durableOpTime" : {
      "ts" : Timestamp(1565760578, 1),
      "t" : NumberLong(1)
    }
  },
  "lastStableCheckpointTimestamp" : Timestamp(1565760528, 1),
  "members" : [
    {
      "_id" : 0,
      "name" : "180.76.159.126:27017",
      "health" : 1,
      "state" : 1,
      "stateStr" : "PRIMARY",
      "uptime" : 419,
      "optime" : {
        "ts" : Timestamp(1565760578, 1),
        "t" : NumberLong(1)
      },
      },
      "optimeDate" : ISODate("2019-08-14T05:29:38Z"),
      "syncingTo" : "",
      "syncSourceHost" : "",
      "syncSourceId" : -1,
      "infoMessage" : "could not find member to sync from",
      "electionTime" : Timestamp(1565760476, 2),
      "electionDate" : ISODate("2019-08-14T05:27:56Z"),
      "configVersion" : 1,
      "self" : true,
      "lastHeartbeatMessage" : ""
    }
  ],
  "ok" : 1,
  "operationTime" : Timestamp(1565760578, 1),
  "$clusterTime" : {
    "clusterTime" : Timestamp(1565760578, 1),
    "signature" : {
      "hash" : BinData(0,"AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA="),
      "keyId" : NumberLong(0)
    }
  }
}

```

说明：

- 1) "set" : "myrs" : 副本集的名字
- 2) "myState": 1 : 说明状态正常
- 3) "members" : 副本集成员数组，此时只有一个： "name" : "180.76.159.126:27017" ，该成员的角色是 "stateStr" : "PRIMARY"，该节点是健康的： "health" : 1。

## 1.4.8 第四步：添加副本从节点

在主节点添加从节点，将其他成员加入到副本集

语法：

```
rs.add(host, arbiterOnly)
```

选项：

Parameter	Type	Description
<code>host</code>	string or document	要添加到副本集的新成员。指定为字符串或配置文档：1) 如果是一个字符串，则需要指定新成员的主机名和可选的端口号；2) 如果是一个文档，请指定在members数组中找到的副本集成员配置文档。您必须在成员配置文档中指定主机字段。有关文档配置字段的说明，详见下方文档：“主机成员的配置文档”
<code>arbiterOnly</code>	boolean	可选的。仅在 <code>&lt;host&gt;</code> 值为字符串时适用。如果为true，则添加的主机是仲裁者。

主机成员的配置文档：

```
{
  _id: <int>,
  host: <string>,           // required
  arbiterOnly: <boolean>,
  buildIndexes: <boolean>,
  hidden: <boolean>,
  priority: <number>,
  tags: <document>,
  slaveDelay: <int>,
  votes: <number>
}
```

### 【示例】

将27018的副本节点添加到副本集中：

```

myrs:PRIMARY> rs.add("180.76.159.126:27018")
{
  "ok" : 1,
  "operationTime" : Timestamp(1565761757, 1),
  "$clusterTime" : {
    "clusterTime" : Timestamp(1565761757, 1),
    "signature" : {
      "hash" : BinData(0,"AAAAAAAAAAAAAAAAAAAAAAAAAAAA="),
      "keyId" : NumberLong(0)
    }
  }
}

```

说明：

1) "ok" : 1 : 说明添加成功。

查看副本集状态：

```

myrs:PRIMARY> rs.status()
{
  "set" : "myrs",
  "date" : ISODate("2019-08-14T05:50:05.738Z"),
  "myState" : 1,
  "term" : NumberLong(1),
  "syncingTo" : "",
  "syncSourceHost" : "",
  "syncSourceId" : -1,
  "heartbeatIntervalMillis" : NumberLong(2000),
  "optimes" : {
    "lastCommittedOpTime" : {
      "ts" : Timestamp(1565761798, 1),
      "t" : NumberLong(1)
    },
    "readConcernMajorityOpTime" : {
      "ts" : Timestamp(1565761798, 1),
      "t" : NumberLong(1)
    },
    "appliedOpTime" : {
      "ts" : Timestamp(1565761798, 1),
      "t" : NumberLong(1)
    },
    "durableOpTime" : {
      "ts" : Timestamp(1565761798, 1),
      "t" : NumberLong(1)
    }
  },
  "lastStableCheckpointTimestamp" : Timestamp(1565761798, 1),
  "members" : [
    {
      "_id" : 0,
      "name" : "180.76.159.126:27017",
      "health" : 1,
      "state" : 1,
      "stateStr" : "PRIMARY",

```

```

    "uptime" : 1639,
    "optime" : {
      "ts" : Timestamp(1565761798, 1),
      "t" : NumberLong(1)
    },
    "optimeDate" : ISODate("2019-08-14T05:49:58Z"),
    "syncingTo" : "",
    "syncSourceHost" : "",
    "syncSourceId" : -1,
    "infoMessage" : "",
    "electionTime" : Timestamp(1565760476, 2),
    "electionDate" : ISODate("2019-08-14T05:27:56Z"),
    "configVersion" : 2,
    "self" : true,
    "lastHeartbeatMessage" : ""
  },
  {
    "_id" : 1,
    "name" : "180.76.159.126:27018",
    "health" : 1,
    "state" : 2,
    "stateStr" : "SECONDARY",
    "uptime" : 48,
    "optime" : {
      "ts" : Timestamp(1565761798, 1),
      "t" : NumberLong(1)
    },
    "optimeDurable" : {
      "ts" : Timestamp(1565761798, 1),
      "t" : NumberLong(1)
    },
    "optimeDate" : ISODate("2019-08-14T05:49:58Z"),
    "optimeDurableDate" : ISODate("2019-08-14T05:49:58Z"),
    "lastHeartbeat" : ISODate("2019-08-14T05:50:05.294Z"),
    "lastHeartbeatRecv" : ISODate("2019-08-
14T05:50:05.476Z"),
    "pingMs" : NumberLong(0),
    "lastHeartbeatMessage" : "",
    "syncingTo" : "180.76.159.126:27017",
    "syncSourceHost" : "180.76.159.126:27017",
    "syncSourceId" : 0,
    "infoMessage" : "",
    "configVersion" : 2
  }
],
"ok" : 1,
"operationTime" : Timestamp(1565761798, 1),
"$clusterTime" : {
  "clusterTime" : Timestamp(1565761798, 1),
  "signature" : {
    "hash" : BinData(0,"AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA"),
    "keyId" : NumberLong(0)
  }
}
}

```

说明：

1) "name" : "180.76.159.126:27018" 是第二个节点的名字，其角色是 "stateStr" : "SECONDARY"

## 1.4.9 第五步：添加仲裁从节点

添加一个仲裁节点到副本集

语法：

```
rs.addArb(host)
```

将27019的仲裁节点添加到副本集中：

```
myrs:PRIMARY> rs.addArb("180.76.159.126:27019")
{
  "ok" : 1,
  "operationTime" : Timestamp(1565761959, 1),
  "$clusterTime" : {
    "clusterTime" : Timestamp(1565761959, 1),
    "signature" : {
      "hash" : BinData(0,"AAAAAAAAAAAAAAAAAAAAAAAAAAAA="),
      "keyId" : NumberLong(0)
    }
  }
}
```

说明：

1) "ok" : 1 : 说明添加成功。

查看副本集状态：

```
myrs:PRIMARY> rs.status()
{
  "set" : "myrs",
  "date" : ISODate("2019-08-14T05:53:27.198Z"),
  "myState" : 1,
  "term" : NumberLong(1),
  "syncingTo" : "",
  "syncSourceHost" : "",
  "syncSourceId" : -1,
  "heartbeatIntervalMillis" : NumberLong(2000),
  "optimes" : {
    "lastCommittedOpTime" : {
      "ts" : Timestamp(1565761998, 1),
      "t" : NumberLong(1)
    },
    "readConcernMajorityOpTime" : {
      "ts" : Timestamp(1565761998, 1),
      "t" : NumberLong(1)
    }
  },
}
```

```

    "appliedOpTime" : {
      "ts" : Timestamp(1565761998, 1),
      "t" : NumberLong(1)
    },
    "durableOpTime" : {
      "ts" : Timestamp(1565761998, 1),
      "t" : NumberLong(1)
    }
  },
  "lastStableCheckpointTimestamp" : Timestamp(1565761978, 1),
  "members" : [
    {
      "_id" : 0,
      "name" : "180.76.159.126:27017",
      "health" : 1,
      "state" : 1,
      "stateStr" : "PRIMARY",
      "uptime" : 1841,
      "optime" : {
        "ts" : Timestamp(1565761998, 1),
        "t" : NumberLong(1)
      },
      "optimeDate" : ISODate("2019-08-14T05:53:18Z"),
      "syncingTo" : "",
      "syncSourceHost" : "",
      "syncSourceId" : -1,
      "infoMessage" : "",
      "electionTime" : Timestamp(1565760476, 2),
      "electionDate" : ISODate("2019-08-14T05:27:56Z"),
      "configVersion" : 3,
      "self" : true,
      "lastHeartbeatMessage" : ""
    },
    {
      "_id" : 1,
      "name" : "180.76.159.126:27018",
      "health" : 1,
      "state" : 2,
      "stateStr" : "SECONDARY",
      "uptime" : 249,
      "optime" : {
        "ts" : Timestamp(1565761998, 1),
        "t" : NumberLong(1)
      },
      "optimeDurable" : {
        "ts" : Timestamp(1565761998, 1),
        "t" : NumberLong(1)
      },
      "optimeDate" : ISODate("2019-08-14T05:53:18Z"),
      "optimeDurableDate" : ISODate("2019-08-14T05:53:18Z"),
      "lastHeartbeat" : ISODate("2019-08-14T05:53:25.668Z"),
      "lastHeartbeatRecv" : ISODate("2019-08-
14T05:53:26.702Z"),
      "pingMs" : NumberLong(0),
      "lastHeartbeatMessage" : "",
      "syncingTo" : "180.76.159.126:27017",
      "syncSourceHost" : "180.76.159.126:27017",
      "syncSourceId" : 0,

```



```

        "infoMessage" : "",
        "configVersion" : 3
    },
    {
        "_id" : 2,
        "name" : "180.76.159.126:27019",
        "health" : 1,
        "state" : 7,
        "stateStr" : "ARBITER",
        "uptime" : 47,
        "lastHeartbeat" : ISODate("2019-08-14T05:53:25.668Z"),
        "lastHeartbeatRecv" : ISODate("2019-08-
14T05:53:25.685Z"),
        "pingMs" : NumberLong(0),
        "lastHeartbeatMessage" : "",
        "syncingTo" : "",
        "syncSourceHost" : "",
        "syncSourceId" : -1,
        "infoMessage" : "",
        "configVersion" : 3
    }
],
"ok" : 1,
"operationTime" : Timestamp(1565761998, 1),
"$clusterTime" : {
    "clusterTime" : Timestamp(1565761998, 1),
    "signature" : {
        "hash" : BinData(0,"AAAAAAAAAAAAAAAAAAAAAAAAAAAA="),
        "keyId" : NumberLong(0)
    }
}
}
}

```

说明：

1) "name" : "180.76.159.126:27019" 是第二个节点的名字，其角色是 "stateStr" : "ARBITER"

## 1.5 副本集的数据读写操作

目标：测试三个不同角色的节点的数据读写情况。

登录主节点27017，写入和读取数据：

```
[root@bobohost ~]# /usr/local/mongodb/bin/mongo --host 180.76.159.126 --port 27017

myrs:PRIMARY> use articledb
switched to db articledb
myrs:PRIMARY> db
articledb
myrs:PRIMARY> db.comment.insert({"articleid":"100000","content":"今天天气真好，阳光明媚","userid":"1001","nickname":"Rose","createdatetime":new Date()})
WriteResult({ "nInserted" : 1 })
myrs:PRIMARY> db.comment.find()
{ "_id" : ObjectId("5d4d2ae3068138b4570f53bf"), "articleid" : "100000", "content" : "今天天气真好，阳光明媚", "userid" : "1001", "nickname" : "Rose", "createdatetime" : ISODate("2019-08-09T08:12:19.427Z") }
```

登录从节点27018

```
[root@bobohost ~]# /usr/local/mongodb/bin/mongo --host 180.76.159.126 --port 27018

myrs:SECONDARY> show dbs;
2019-09-10T10:56:51.953+0800 E QUERY [js] Error: listDatabases failed:{"operationTime" : Timestamp(1568084204, 1), "ok" : 0, "errmsg" : "not master and slaveOk=false", "code" : 13435, "codeName" : "NotMasterNoSlaveOk", "$clusterTime" : {"clusterTime" : Timestamp(1568084204, 1), "signature" : {"hash" : BinData(0,"AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA="), "keyId" : NumberLong(0)} }
} :
_getErrorWithCode@src/mongo/shell/utils.js:25:13
Mongo.prototype.getDBs@src/mongo/shell/mongo.js:139:1
shellHelper.show@src/mongo/shell/utils.js:882:13
shellHelper@src/mongo/shell/utils.js:766:15
@(shellhelp2):1:1
```

发现，不能读取集合的数据。当前从节点只是一个备份，不是奴隶节点，无法读取数据，写当然更不行。

因为默认情况下，从节点是没有读写权限的，可以增加读的权限，但需要进行设置。

设置读操作权限：

说明：

设置为奴隶节点，允许在从成员上运行读的操作

语法：

```
rs.slaveOk()  
#或  
rs.slaveOk(true)
```

提示：

该命令是 `db.getMongo().setSlaveOk()` 的简化命令。

### 【示例】

在27018上设置作为奴隶节点权限，具备读权限：

```
rs:SECONDARY> rs.slaveOk()
```

此时，在执行查询命令，运行成功！

但仍然不允许插入。

```
myrs:SECONDARY> rs.slaveOk()  
myrs:SECONDARY> show dbs;  
admin      0.000GB  
articledb  0.000GB  
config     0.000GB  
local      0.000GB  
myrs:SECONDARY> use articledb  
switched to db articledb  
myrs:SECONDARY> show collections  
comment  
myrs:SECONDARY> db.comment.find()  
{ "_id" : ObjectId("5d7710c04cfd7eee2e3cdabe"), "articleid" : "100000",  
  "content" : "今天天气真好，阳光明媚", "userid" : "1001", "nickname" : "Rose",  
  "createdatetime" : ISODate("2019-09-10T02:56:00.467Z") }  
myrs:SECONDARY> db.comment.insert({"_id":"1","articleid":"100001","content":"我们  
不应该把清晨浪费在手机上，健康很重要，k一杯温水幸福你我  
他。","userid":"1002","nickname":"相忘于江湖","createdatetime":new Date("2019-08-  
05T22:08:15.522Z"),"likenum":NumberInt(1000),"state":"1"})  
WriteCommandError({  
  "operationTime" : Timestamp(1568084434, 1),  
  "ok" : 0,  
  "errmsg" : "not master",  
  "code" : 10107,  
  "codeName" : "NotMaster",  
  "$clusterTime" : {  
    "clusterTime" : Timestamp(1568084434, 1),  
    "signature" : {  
      "hash" : BinData(0,"AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA="),  
      "keyId" : NumberLong(0)  
    }  
  }  
})
```

现在可实现了读写分离，让主插入数据，让从来读取数据。

如果要取消作为奴隶节点的读权限：

```

myrs:SECONDARY> rs.slaveOk(false)
myrs:SECONDARY> db.comment.find()
Error: error: {
  "operationTime" : Timestamp(1568084459, 1),
  "ok" : 0,
  "errmsg" : "not master and slaveOk=false",
  "code" : 13435,
  "codeName" : "NotMasterNoSlaveOk",
  "$clusterTime" : {
    "clusterTime" : Timestamp(1568084459, 1),
    "signature" : {
      "hash" : BinData(0,"AAAAAAAAAAAAAAAAAAAAAAAAAAAA="),
      "keyId" : NumberLong(0)
    }
  }
}

```

仲裁者节点，不存放任何业务数据的，可以登录查看

```

[root@bobohost ~]# /usr/local/mongodb/bin/mongo --host 180.76.159.126 --port 27019

myrs:ARBITER> rs.slaveOk()
myrs:ARBITER> show dbs
local 0.000GB
myrs:ARBITER> use local
switched to db local
myrs:ARBITER> show collections
replset.minvalid
replset.oplogTruncateAfterPoint
startup_log
system.replset
system.rollback.id
myrs:ARBITER>

```

发现，只存放副本集配置等数据。

## 1.6 主节点的选举原则

MongoDB在副本集中，会自动进行主节点的选举，主节点选举的触发条件：

- 1) 主节点故障
- 2) 主节点网络不可达（默认心跳信息为10秒）
- 3) 人工干预（rs.stepDown(600)）

一旦触发选举，就要根据一定规则来选主节点。

选举规则是根据票数来决定谁获胜：

- 票数最高，且获得了“大多数”成员的投票支持的节点获胜。  
“大多数”的定义为：假设复制集内投票成员数量为N，则大多数为  $N/2 + 1$ 。例如：3个投票成员，则大多数的值是2。当复制集内存活成员数量不足大多数时，整个复制集将无法选举出Primary，复制集将无法提供写服务，处于只读状态。
- 若票数相同，且都获得了“大多数”成员的投票支持的，数据新的节点获胜。  
数据的新旧是通过操作日志oplog来对比的。

在获得票数的时候，优先级（priority）参数影响重大。

可以通过设置优先级（priority）来设置额外票数。优先级即权重，取值为0-1000，相当于可额外增加0-1000的票数，优先级的值越大，就越可能获得多数成员的投票（votes）数。指定较高的值可使成员更有资格成为主要成员，更低的值可使成员更不符合条件。

默认情况下，优先级的值是1

```
myrs:PRIMARY> rs.conf()
{
  "_id" : "myrs",
  "version" : 3,
  "protocolVersion" : NumberLong(1),
  "writeConcernMajorityJournalDefault" : true,
  "members" : [
    {
      "_id" : 0,
      "host" : "180.76.159.126:27017",
      "arbiterOnly" : false,
      "buildIndexes" : true,
      "hidden" : false,
      "priority" : 1,
      "tags" : {

      },
      "slaveDelay" : NumberLong(0),
      "votes" : 1
    },
    {
      "_id" : 1,
      "host" : "180.76.159.126:27018",
      "arbiterOnly" : false,
      "buildIndexes" : true,
      "hidden" : false,
      "priority" : 1,
      "tags" : {

      },
      "slaveDelay" : NumberLong(0),
      "votes" : 1
    },
    {
      "_id" : 2,
      "host" : "180.76.159.126:27019",
      "arbiterOnly" : true,
      "buildIndexes" : true,
```

```

        "hidden" : false,
        "priority" : 0,
        "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("5d539bdcd6a308e600d126bb")
}
}

```

可以看出，主节点和副本节点的优先级各为1，即，默认可以认为都已经有一票。但选举节点，优先级是0，（要注意是，官方说了，选举节点的优先级必须是0，不能是别的值。即不具备选举权，但具有投票权）

### 【了解】修改优先级

比如，下面提升从节点的优先级：

1) 先将配置导入cfg变量

```
myrs:SECONDARY> cfg=rs.conf()
```

2) 然后修改值（ID号默认从0开始）：

```
myrs:SECONDARY> cfg.members[1].priority=2
2
```

3) 重新加载配置

```
myrs:SECONDARY> rs.reconfig(cfg)
{ "ok" : 1 }
```

稍等片刻会重新开始选举。

## 1.7 故障测试

## 1.7.1 副本节点故障测试

关闭27018副本节点：

发现，主节点和仲裁节点对27018的心跳失败。因为主节点还在，因此，没有触发投票选举。

如果此时，在主节点写入数据。

```
db.comment.insert({"_id":"1","articleid":"100001","content":"我们不应该把清晨浪费在手机上，健康很重要，一杯温水幸福你我他。","userid":"1002","nickname":"相忘于江湖","createdatetime":new Date("2019-08-05T22:08:15.522Z"),"likenum":NumberInt(1000),"state":"1"})
```

再启动从节点，会发现，主节点写入的数据，会自动同步给从节点。

## 1.7.2 主节点故障测试

关闭27017节点

发现，从节点和仲裁节点对27017的心跳失败，当失败超过10秒，此时因为没有主节点了，会自动发起投票。

而副本节点只有27018，因此，候选人只有一个就是27018，开始投票。

27019向27018投了一票，27018本身自带一票，因此共两票，超过了“大多数”

27019是仲裁节点，没有选举权，27018不向其投票，其票数是0。

最终结果，27018成为主节点。具备读写功能。

在27018写入数据查看。

```
db.comment.insert({"_id":"2","articleid":"100001","content":"我夏天空腹喝凉开水，冬天喝温开水","userid":"1005","nickname":"伊人憔悴","createdatetime":new Date("2019-08-05T23:58:51.485Z"),"likenum":NumberInt(888),"state":"1"})
```

再启动27017节点，发现27017变成了从节点，27018仍保持主节点。

登录27017节点，发现是从节点了，数据自动从27018同步。

从而实现了高可用。

## 1.7.3 仲裁节点和主节点故障

先关掉仲裁节点27019，

关掉现在的主节点27018

登录27017后，发现，27017仍然是从节点，副本集中没有主节点了，导致此时，副本集是只读状态，无法写入。

为啥不选举了？因为27017的票数，没有获得大多数，即没有大于等于2，它只有默认的一票（优先级是1）

如果要触发选举，随便加入一个成员即可。

- 如果只加入27019仲裁节点成员，则主节点一定是27017，因为没得选了，仲裁节点不参与选举，但参与投票。（不演示）
- 如果只加入27018节点，会发起选举。因为27017和27018都是两票，则按照谁数据新，谁当主节点。

## 1.7.4 仲裁节点和从节点故障

先关掉仲裁节点27019，

关掉现在的副本节点27018

10秒后，27017主节点自动降级为副本节点。（服务降级）

副本集不可写数据了，已经故障了。

## 1.8 Compass连接副本集

如果使用云服务器需要修改配置中的主节点ip

```
var config = rs.config();  
config.members[0].host="180.76.159.126:27017";rs.reconfig(config)
```

compass连接：

Connect to Host

Hostname

Port

SRV Record

Authentication

Replica Set Name

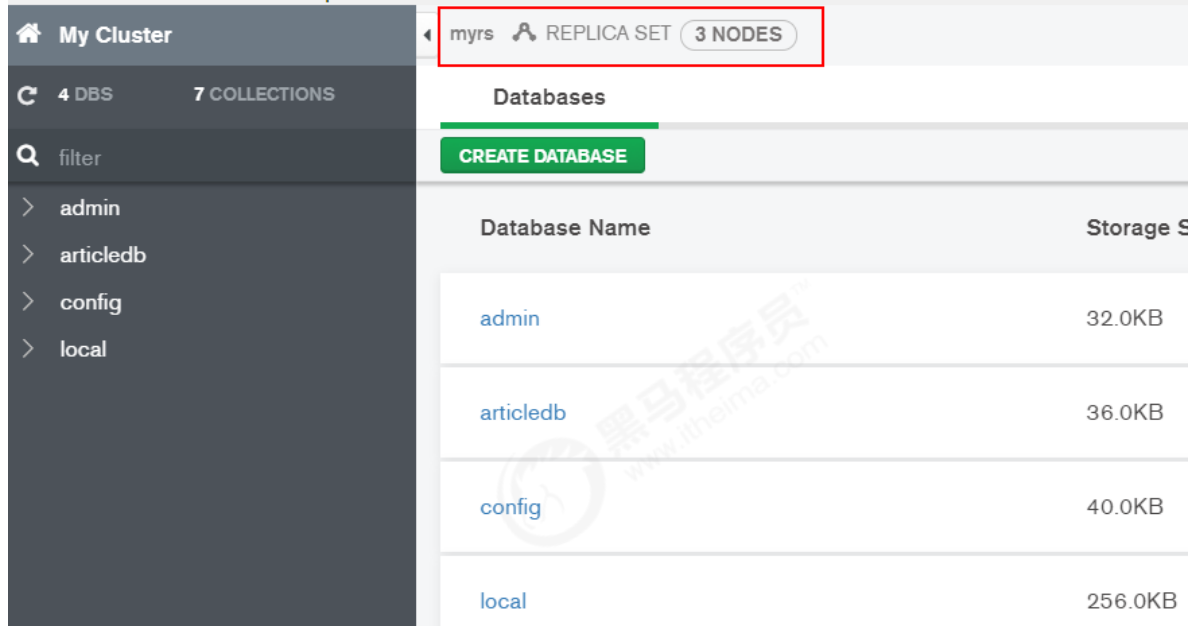
Read Preference

SSL

SSH Tunnel

Favorite Name





The screenshot shows the MongoDB Compass interface. At the top, it displays 'My Cluster' with a red box highlighting 'myrs REPLICA SET 3 NODES'. Below this, there are statistics for '4 DBS' and '7 COLLECTIONS'. A search bar with 'filter' is present. On the left, a sidebar lists databases: 'admin', 'articledb', 'config', and 'local'. The main area shows a table of databases with columns 'Database Name' and 'Storage Size'.

Database Name	Storage Size
admin	32.0KB
articledb	36.0KB
config	40.0KB
local	256.0KB

## 1.9 SpringDataMongoDB连接副本集

副本集语法：

```
mongodb://host1,host2,host3/articledb?  
connect=replicaSet&slaveOk=true&replicaSet=副本集名字
```

其中：

- slaveOk=true：开启副本节点读的功能，可实现读写分离。
- connect=replicaSet：自动到副本集中选择读写的主机。如果slaveOK是打开的，则实现了读写分离

### 【示例】

连接 replica set 三台服务器 (端口 27017, 27018, 和27019)，直接连接第一个服务器，无论是replica set一部分或者主服务器或者从服务器，写入操作应用在主服务器 并且分布查询到从服务器。

修改配置文件application.yml

```
spring:  
  #数据源配置  
  data:  
    mongodb:  
      # 主机地址  
      # host: 180.76.159.126  
      # 数据库  
      # database: articledb  
      # 默认端口是27017  
      # port: 27017  
      #也可以使用uri连接  
      #uri: mongodb://192.168.40.134:27017/articledb
```

```
# 副本集的连接字符串
```

```
uri:
```

```
mongodb://180.76.159.126:27017,180.76.159.126:27018,180.76.159.126:27019/article  
db?connect=replicaSet&slaveOk=true&replicaSet=myrs
```

注意：

主机必须是副本集中所有的主机，包括主节点、副本节点、仲裁节点。

SpringDataMongoDB自动实现了读写分离：

写操作时，只打开主节点连接：

```
2019-09-02 12:16:27.143 INFO 13336 --- [          main]  
org.mongodb.driver.connection      : Opened connection  
[connectionId{localValue:4, serverValue:60}] to 180.76.159.126:27017  
2019-09-02 12:16:27.376 INFO 13336 --- [          main]  
c.i.article.service.CommentServiceTest : Started CommentServiceTest in 5.249  
seconds (JVM running for 6.16)  
2019-09-02 12:16:27.610 INFO 13336 --- [          Thread-2]  
org.mongodb.driver.connection      : Closed connection  
[connectionId{localValue:4, serverValue:60}] to 180.76.159.126:27017 because the  
pool has been closed.
```

读操作是，同时打开主节点和从节点连接，但使用从节点获取数据：

```

2019-09-02 12:18:20.254 INFO 4168 --- [          main]
org.mongodb.driver.connection          : Opened connection
[connectionId{localValue:4, serverValue:62}] to 180.76.159.126:27017
2019-09-02 12:18:20.504 INFO 4168 --- [          main]
c.i.article.service.CommentServiceTest : Started CommentServiceTest in 5.111
seconds (JVM running for 5.95)
2019-09-02 12:18:20.694 INFO 4168 --- [          main]
org.mongodb.driver.connection          : Opened connection
[connectionId{localValue:5, serverValue:39}] to 180.76.159.126:27018
[Comment{id='5d64930fd1c1dc4ccf71eeac', content='今天天气真好, 阳光明媚',
publishtime=null, userid='1001', nickname='Rose', createdatetime=2019-08-
27T10:18:55.768, likenum=null, replynum=null, state='null', parentid='null',
articleid='100000'}, Comment{id='1', content='我们不应该把清晨浪费在手机上, 健康很重
要, 一杯温水幸福你我他.', publishtime=null, userid='1002', nickname='相忘于江湖',
createdatetime=2019-08-06T06:08:15.522, likenum=1000, replynum=null, state='1',
parentid='null', articleid='100001'}, Comment{id='2', content='我夏天空腹喝凉开水,
冬天喝温开水', publishtime=null, userid='1005', nickname='伊人憔悴',
createdatetime=2019-08-06T07:58:51.485, likenum=888, replynum=null, state='1',
parentid='null', articleid='100001'}, Comment{id='5d6c93a866ecc73210d7d600',
content='测试添加的数据2', publishtime=null, userid='1003', nickname='凯撒大帝',
createdatetime=2019-09-02T11:59:36.525, likenum=0, replynum=0, state='1',
parentid='null', articleid='100000'}]]
2019-09-02 12:18:20.740 INFO 4168 --- [      Thread-2]
org.mongodb.driver.connection          : Closed connection
[connectionId{localValue:4, serverValue:62}] to 180.76.159.126:27017 because the
pool has been closed.
2019-09-02 12:18:20.740 INFO 4168 --- [      Thread-2]
org.mongodb.driver.connection          : Closed connection
[connectionId{localValue:5, serverValue:39}] to 180.76.159.126:27018 because the
pool has been closed.

```

完整的连接字符串的参考（了解）：

MongoDB客户端连接语法：

```

mongodb://[username:password@]host1[:port1][,host2[:port2],...[,hostN[:portN]]]
[/[database][?options]]

```

- **mongodb://** 这是固定的格式，必须要指定。
- **username:password@** 可选项，如果设置，在连接数据库服务器之后，驱动都会尝试登陆这个数据库
- **host1** 必须的指定至少一个host, host1 是这个URI唯一要填写的。它指定了要连接服务器的地址。如果要连接复制集，请指定多个主机地址。
- **portX** 可选的指定端口，如果不填，默认为27017
- **/database** 如果指定username:password@，连接并验证登陆指定数据库。若不指定，默认打开test数据库。
- **?options** 是连接选项。如果不使用/database，则前面需要加上/。所有连接选项都是键值对 name=value，键值对之间通过&或(分号) 隔开

标准的连接格式包含了多个选项(options)，如下所示：

选项	描述
replicaSet=name	验证replica set的名称。 Impliesconnect=replicaSet.
slaveOk=true false	true:在connect=direct模式下，驱动会连接第一台机器，即使这台服务器不是主。在connect=replicaSet模式下，驱动会发送所有的写请求到主并且把读取操作分布在其他从服务器。false: 在connect=direct模式下，驱动会自动找寻主服务器。在connect=replicaSet 模式下，驱动仅仅连接主服务器，并且所有的读写命令都连接到主服务器。
safe=true false	true: 在执行更新操作之后，驱动都会发送getLastError命令来确保更新成功。(还要参考 wtimeoutMS).false: 在每次更新之后，驱动不会发送getLastError来确保更新成功。
w=n	驱动添加 { w : n } 到getLastError命令. 应用于safe=true。
wtimeoutMS=ms	驱动添加 { wtimeout : ms } 到 getlasterror 命令. 应用于 safe=true.
fsync=true false	true: 驱动添加 { fsync : true } 到 getlasterror 命令.应用于 safe=true.false: 驱动不会添加到getLastError命令中。
journal=true false	如果设置为 true, 同步到 journal (在提交到数据库前写入到实体中). 应用于 safe=true
connectTimeoutMS=ms	可以打开连接的时间。
socketTimeoutMS=ms	发送和接受sockets的时间。

## 2. 分片集群-Sharded Cluster

### 2.1 分片概念

分片 ( sharding ) 是一种跨多台机器分布数据的方法， MongoDB使用分片来支持具有非常大的数据集和高吞吐量操作的部署。

换句话说：分片(sharding)是指将数据拆分，将其分散存在不同的机器上的过程。有时也用分区 (partitioning)来表示这个概念。将数据分散到不同的机器上，不需要功能强大的大型计算机就可以储存更多的数据，处理更多的负载。

具有大型数据集或高吞吐量应用程序的数据库系统可以会挑战单个服务器的容量。例如，高查询率会耗尽服务器的CPU容量。工作集大小大于系统的RAM会强调磁盘驱动器的I / O容量。

有两种解决系统增长的方法：垂直扩展和水平扩展。

垂直扩展意味着增加单个服务器的容量，例如使用更强大的CPU，添加更多RAM或增加存储空间量。可用技术的局限性可能会限制单个机器对于给定工作负载而言足够强大。此外，基于云的提供商基于可用的硬件配置具有硬性上限。结果，垂直缩放有实际的最大值。

水平扩展意味着划分系统数据集并加载多个服务器，添加其他服务器以根据需要增加容量。虽然单个机器的总体速度或容量可能不高，但每台机器处理整个工作负载的子集，可能提供比单个高速大容量服务器更高的效率。扩展部署容量只需要根据需要添加额外的服务器，这可能比单个机器的高端硬件的总体成本更低。权衡是基础架构和部署维护的复杂性增加。

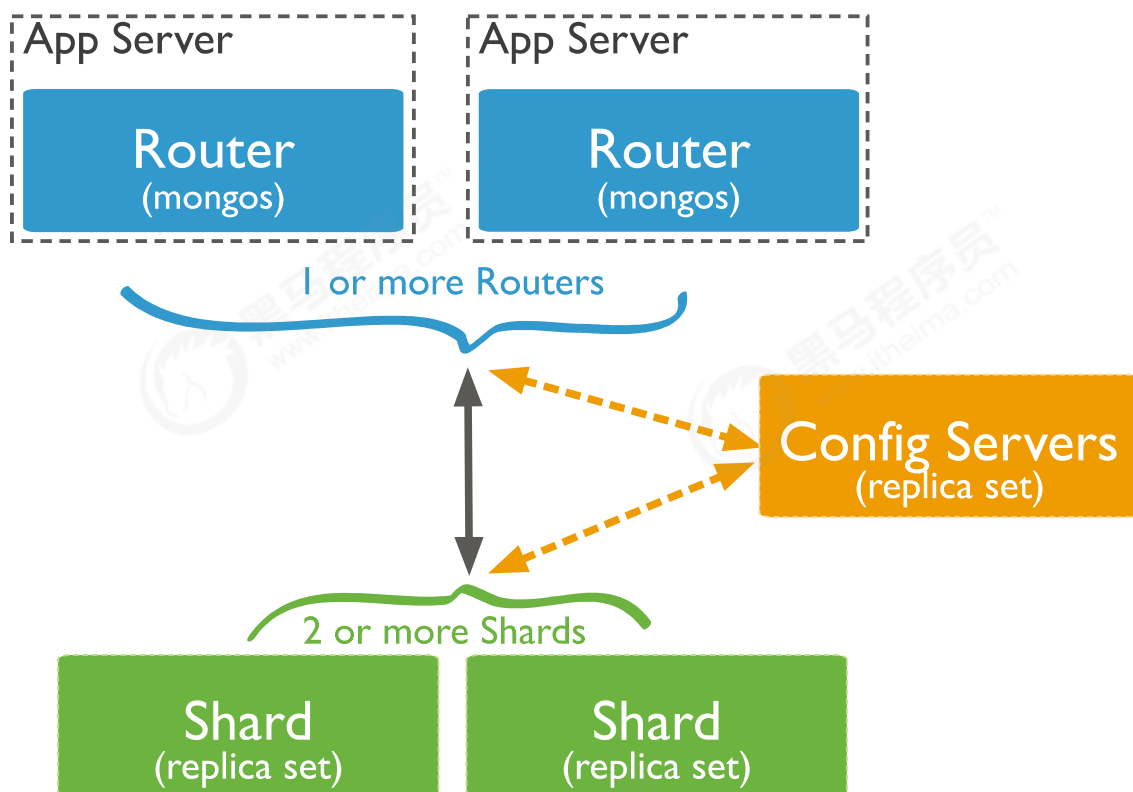
MongoDB支持通过分片进行水平扩展。

## 2.2 分片集群包含的组件

MongoDB分片集群包含以下组件：

- 分片（存储）：每个分片包含分片数据的子集。每个分片都可以部署为副本集。
- mongos（路由）：mongos充当查询路由器，在客户端应用程序和分片集群之间提供接口。
- config servers（“调度”的配置）：配置服务器存储集群的元数据和配置设置。从MongoDB 3.4开始，必须将配置服务器部署为副本集（CSRS）。

下图描述了分片集群中组件的交互：



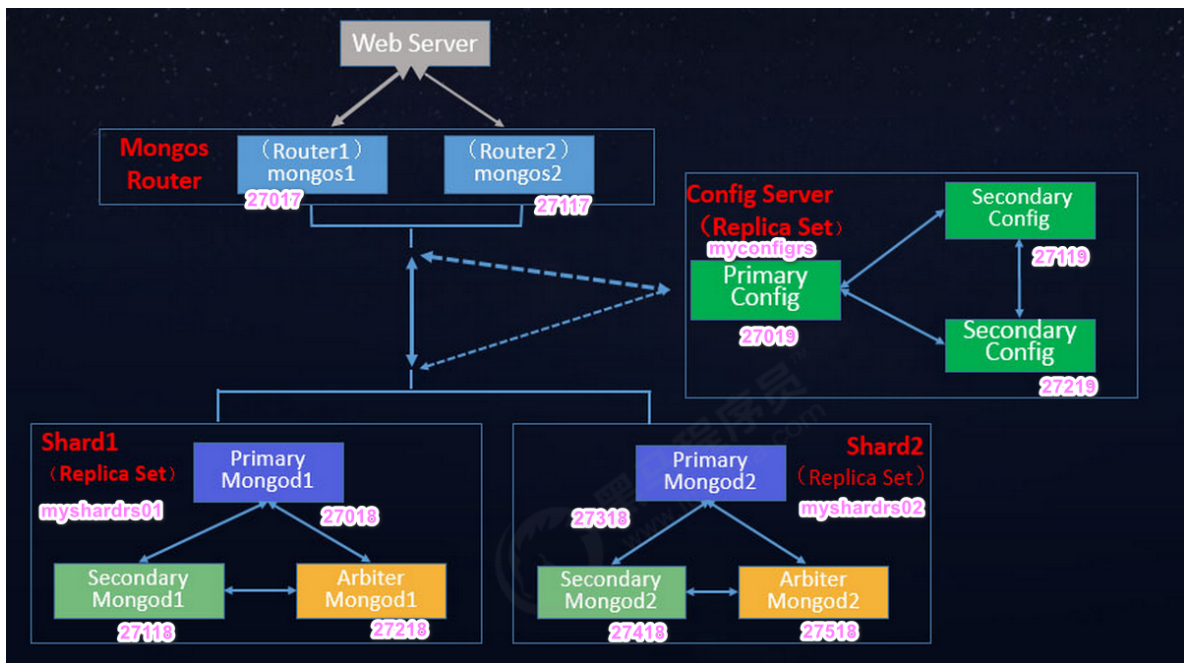
MongoDB在集合级别对数据进行分片，将集合数据分布在集群中的分片上。

27018 if mongod is a shard member ;

27019 if mongod is a config server member

## 2.3 分片集群架构目标

两个分片节点副本集（3+3）+一个配置节点副本集（3）+两个路由节点（2），共11个服务节点。



## 2.4 分片（存储）节点副本集的创建

所有的的配置文件都直接放到 `sharded_cluster` 的相应的子目录下面，默认配置文件名字：

`mongod.conf`

### 2.4.1 第一套副本集

准备存放数据和日志的目录：

```
#-----myshards01
mkdir -p /mongodb/sharded_cluster/myshards01_27018/log \ &
mkdir -p /mongodb/sharded_cluster/myshards01_27018/data/db \ &

mkdir -p /mongodb/sharded_cluster/myshards01_27118/log \ &
mkdir -p /mongodb/sharded_cluster/myshards01_27118/data/db \ &

mkdir -p /mongodb/sharded_cluster/myshards01_27218/log \ &
mkdir -p /mongodb/sharded_cluster/myshards01_27218/data/db
```

新建或修改配置文件：

```
vim /mongodb/sharded_cluster/myshards01_27018/mongod.conf
```

myshards01\_27018：

```
systemLog:
  #MongoDB发送所有日志输出的目标指定为文件
  destination: file
```

```

#mongod或mongos应向其发送所有诊断日志记录信息的日志文件的路径
path: "/mongodb/sharded_cluster/myshardrs01_27018/log/mongod.log"
#当mongos或mongod实例重新启动时，mongos或mongod会将新条目附加到现有日志文件的末尾。
logAppend: true
storage:
#mongod实例存储其数据的目录。storage.dbPath设置仅适用于mongod。
dbPath: "/mongodb/sharded_cluster/myshardrs01_27018/data/db"
journal:
#启用或禁用持久性日志以确保数据文件保持有效和可恢复。
enabled: true
processManagement:
#启用在后台运行mongos或mongod进程的守护进程模式。
fork: true
#指定用于保存mongos或mongod进程的进程ID的文件位置，其中mongos或mongod将写入其PID
pidFilePath: "/mongodb/sharded_cluster/myshardrs01_27018/log/mongod.pid"
net:
#服务实例绑定所有IP，有副作用，副本集初始化的时候，节点名字会自动设置为本地域名，而不是ip
#bindIpAll: true
#服务实例绑定的IP
bindIp: localhost,192.168.0.2
#bindIp
#绑定的端口
port: 27018
replication:
#副本集的名称
replSetName: myshardrs01
sharding:
#分片角色
clusterRole: shardsvr

```

sharding.clusterRole :

Value	Description
configsvr	Start this instance as a <a href="#">config server</a> . The instance starts on port 27019 by default.
shardsvr	Start this instance as a <a href="#">shard</a> . The instance starts on port 27018 by default.

注意：

设置sharding.clusterRole需要mongod实例运行复制。要将实例部署为副本集成员，请使用replSetName设置并指定副本集的名称。

新建或修改配置文件：

```
vim /mongodb/sharded_cluster/myshardrs01_27118/mongod.conf
```

myshardrs01\_27118 :

```

systemLog:
#MongoDB发送所有日志输出的目标指定为文件
destination: file

```

```

#mongod或mongos应向其发送所有诊断日志记录信息的日志文件的路径
path: "/mongodb/sharded_cluster/myshardrs01_27118/log/mongod.log"
#当mongos或mongod实例重新启动时，mongos或mongod会将新条目附加到现有日志文件的末尾。
LogAppend: true
storage:
#mongod实例存储其数据的目录。storage.dbPath设置仅适用于mongod。
dbPath: "/mongodb/sharded_cluster/myshardrs01_27118/data/db"
journal:
#启用或禁用持久性日志以确保数据文件保持有效和可恢复。
enabled: true
processManagement:
#启用在后台运行mongos或mongod进程的守护进程模式。
fork: true
#指定用于保存mongos或mongod进程的进程ID的文件位置，其中mongos或mongod将写入其PID
pidFilePath: "/mongodb/sharded_cluster/myshardrs01_27118/log/mongod.pid"
net:
#服务实例绑定所有IP
#bindIpAll: true
#服务实例绑定的IP
bindIp: localhost,192.168.0.2
#绑定的端口
port: 27118
replication:
replSetName: myshardrs01
sharding:
clusterRole: shardsvr

```

新建或修改配置文件：

```
vim /mongodb/sharded_cluster/myshardrs01_27218/mongod.conf
```

myshardrs01\_27218：

```

systemLog:
#MongoDB发送所有日志输出的目标指定为文件
destination: file
#mongod或mongos应向其发送所有诊断日志记录信息的日志文件的路径
path: "/mongodb/sharded_cluster/myshardrs01_27218/log/mongod.log"
#当mongos或mongod实例重新启动时，mongos或mongod会将新条目附加到现有日志文件的末尾。
LogAppend: true
storage:
#mongod实例存储其数据的目录。storage.dbPath设置仅适用于mongod。
dbPath: "/mongodb/sharded_cluster/myshardrs01_27218/data/db"
journal:
#启用或禁用持久性日志以确保数据文件保持有效和可恢复。
enabled: true
processManagement:
#启用在后台运行mongos或mongod进程的守护进程模式。
fork: true
#指定用于保存mongos或mongod进程的进程ID的文件位置，其中mongos或mongod将写入其PID
pidFilePath: "/mongodb/sharded_cluster/myshardrs01_27218/log/mongod.pid"
net:
#服务实例绑定的IP
bindIp: localhost,192.168.0.2
#绑定的端口

```



```
port: 27218
replication:
  replSetName: myshardrs01
sharding:
  clusterRole: shardsvr
```

启动第一套副本集：一主一副本一仲裁

依次启动三个mongod服务：

```
[root@bobohost bin]# /usr/local/mongodb/bin/mongod -f
/mongodb/sharded_cluster/myshardrs01_27018/mongod.conf
about to fork child process, waiting until server is ready for connections.
forked process: 123223
child process started successfully, parent exiting

[root@bobohost bin]# /usr/local/mongodb/bin/mongod -f
/mongodb/sharded_cluster/myshardrs01_27118/mongod.conf
about to fork child process, waiting until server is ready for connections.
forked process: 123292
child process started successfully, parent exiting
[root@bobohost bin]# /usr/local/mongodb/bin/mongod -f
/mongodb/sharded_cluster/myshardrs01_27218/mongod.conf
about to fork child process, waiting until server is ready for connections.
forked process: 123326
child process started successfully, parent exiting
```

查看服务是否启动：

```
[root@bobohost bin]# ps -ef |grep mongod
polkitd  61622  61604  0 7月31 ?        00:04:29 mongod --bind_ip_all
root    123223      1  1 01:10 ?        00:00:01 /usr/local/mongodb/bin/mongod
-f /mongodb/sharded_cluster/myshardrs01_27018/mongod.conf
root    123292      1  4 01:11 ?        00:00:00 /usr/local/mongodb/bin/mongod
-f /mongodb/sharded_cluster/myshardrs01_27118/mongod.conf
root    123326      1  6 01:11 ?        00:00:00 /usr/local/mongodb/bin/mongod
-f /mongodb/sharded_cluster/myshardrs01_27218/mongod.conf
```

(1) 初始化副本集和创建主节点：

使用客户端命令连接任意一个节点，但这里尽量要连接主节点：

```
/usr/local/mongodb/bin/mongo --host 180.76.159.126 --port 27018
```

执行初始化副本集命令：

```
> rs.initiate()
{
  "info2" : "no configuration specified. Using a default configuration for
the set",
  "me" : "180.76.159.126:27018",
```

```
"ok" : 1,
"operationTime" : Timestamp(1564593349, 1),
"$clusterTime" : {
  "clusterTime" : Timestamp(1564593349, 1),
  "signature" : {
    "hash" : BinData(0,"AAAAAAAAAAAAAAAAAAAAAAAAAAAA="),
    "keyId" : NumberLong(0)
  }
}
}
```

```
"me" : "bobohost.localdomain:27018",
```

查看副本集情况(节选内容)：

```
myshardrs01:SECONDARY> rs.status()
{
  "set" : "myshardrs01",
  .....
}
```

(2) 主节点配置查看：

```
myshardrs01:PRIMARY> rs.conf()
{
  "_id" : "myshardrs01",
  "version" : 1,
  "protocolVersion" : NumberLong(1),
  "writeConcernMajorityJournalDefault" : true,
  "members" : [
    {
      "_id" : 0,
      "host" : "180.76.159.126:27018",
      "arbiterOnly" : false,
      "buildIndexes" : true,
      "hidden" : false,
      "priority" : 1,
      "tags" : {
      },
      "slaveDelay" : NumberLong(0),
      "votes" : 1
    }
  ],
}
```

(3) 添加副本节点：

```

myshardrs01:PRIMARY> rs.add("180.76.159.126:27118")
{
  "ok" : 1,
  "operationTime" : Timestamp(1564593626, 1),
  "$clusterTime" : {
    "clusterTime" : Timestamp(1564593626, 1),
    "signature" : {
      "hash" : BinData(0,"AAAAAAAAAAAAAAAAAAAAAAAAAAAA="),
      "keyId" : NumberLong(0)
    }
  }
}

```

(4) 添加仲裁节点：

```

myshardrs01:PRIMARY> rs.addArb("180.76.159.126:27218")
{
  "ok" : 1,
  "operationTime" : Timestamp(1564593675, 1),
  "$clusterTime" : {
    "clusterTime" : Timestamp(1564593675, 1),
    "signature" : {
      "hash" : BinData(0,"AAAAAAAAAAAAAAAAAAAAAAAAAAAA="),
      "keyId" : NumberLong(0)
    }
  }
}

```

查看副本集的配置情况：

```

myshardrs01:PRIMARY> rs.conf()
{
  "_id" : "myshardrs01",
  "version" : 3,
  "protocolVersion" : NumberLong(1),
  "writeConcernMajorityJournalDefault" : true,
  "members" : [
    {
      "_id" : 0,
      "host" : "180.76.159.126:27018",
      "arbiterOnly" : false,
      "buildIndexes" : true,
      "hidden" : false,
      "priority" : 1,
      "tags" : {
      },
      "slaveDelay" : NumberLong(0),
      "votes" : 1
    },
    {
      "_id" : 1,

```

```

        "host" : "180.76.159.126:27118",
        "arbiterOnly" : false,
        "buildIndexes" : true,
        "hidden" : false,
        "priority" : 1,
        "tags" : {

        },
        "slaveDelay" : NumberLong(0),
        "votes" : 1
    },
    {
        "_id" : 2,
        "host" : "180.76.159.126:27218",
        "arbiterOnly" : true,
        "buildIndexes" : true,
        "hidden" : false,
        "priority" : 0,
        "tags" : {

        },
        "slaveDelay" : NumberLong(0),
        "votes" : 1
    }
],

```

## 2.4.2 第二套副本集

准备存放数据和日志的目录：

```

#-----myshardrs02
mkdir -p /mongodb/sharded_cluster/myshardrs02_27318/log \ &
mkdir -p /mongodb/sharded_cluster/myshardrs02_27318/data/db \ &

mkdir -p /mongodb/sharded_cluster/myshardrs02_27418/log \ &
mkdir -p /mongodb/sharded_cluster/myshardrs02_27418/data/db \ &

mkdir -p /mongodb/sharded_cluster/myshardrs02_27518/log \ &
mkdir -p /mongodb/sharded_cluster/myshardrs02_27518/data/db

```

新建或修改配置文件：

```
vim /mongodb/sharded_cluster/myshardrs02_27318/mongod.conf
```

myshardrs02\_27318：

```

systemLog:
    #MongoDB发送所有日志输出的目标指定为文件
    destination: file
    #mongod或mongos应向其发送所有诊断日志记录信息的日志文件的路径
    path: "/mongodb/sharded_cluster/myshardrs02_27318/log/mongod.log"
    #当mongos或mongod实例重新启动时，mongos或mongod会将新条目附加到现有日志文件的末尾。

```

```

logAppend: true
storage:
  #mongodb实例存储其数据的目录。storage.dbPath设置仅适用于mongod。
  dbPath: "/mongodb/sharded_cluster/myshardrs02_27318/data/db"
  journal:
    #启用或禁用持久性日志以确保数据文件保持有效和可恢复。
    enabled: true
processManagement:
  #启用在后台运行mongos或mongod进程的守护进程模式。
  fork: true
  #指定用于保存mongos或mongod进程的进程ID的文件位置，其中mongos或mongod将写入其PID
  pidFilePath: "/mongodb/sharded_cluster/myshardrs02_27318/log/mongod.pid"
net:
  #服务实例绑定的IP
  bindIp: localhost,192.168.0.2
  #绑定的端口
  port: 27318
replication:
  replSetName: myshardrs02
sharding:
  clusterRole: shardsvr

```

新建或修改配置文件：

```
vim /mongodb/sharded_cluster/myshardrs02_27418/mongod.conf
```

myshardrs02\_27418：

```

systemLog:
  #MongoDB发送所有日志输出的目标指定为文件
  destination: file
  #mongod或mongos应向其发送所有诊断日志记录信息的日志文件的路径
  path: "/mongodb/sharded_cluster/myshardrs02_27418/log/mongod.log"
  #当mongos或mongod实例重新启动时，mongos或mongod会将新条目附加到现有日志文件的末尾。
  logAppend: true
storage:
  #mongodb实例存储其数据的目录。storage.dbPath设置仅适用于mongod。
  dbPath: "/mongodb/sharded_cluster/myshardrs02_27418/data/db"
  journal:
    #启用或禁用持久性日志以确保数据文件保持有效和可恢复。
    enabled: true
processManagement:
  #启用在后台运行mongos或mongod进程的守护进程模式。
  fork: true
  #指定用于保存mongos或mongod进程的进程ID的文件位置，其中mongos或mongod将写入其PID
  pidFilePath: "/mongodb/sharded_cluster/myshardrs02_27418/log/mongod.pid"
net:
  #服务实例绑定所有IP
  #bindIpAll: true
  #服务实例绑定的IP
  bindIp: localhost,192.168.0.2
  #绑定的端口
  port: 27418
replication:
  replSetName: myshardrs02

```

```
sharding:
  clusterRole: shardsvr
```

新建或修改配置文件：

```
vim /mongodb/sharded_cluster/myshardrs02_27518/mongod.conf
```

myshardrs02\_27518：

```
systemLog:
  #MongoDB发送所有日志输出的目标指定为文件
  destination: file
  #mongod或mongos应向其发送所有诊断日志记录信息的日志文件的路径
  path: "/mongodb/sharded_cluster/myshardrs02_27518/log/mongod.log"
  #当mongos或mongod实例重新启动时，mongos或mongod会将新条目附加到现有日志文件的末尾。
  logAppend: true
storage:
  #mongod实例存储其数据的目录。storage.dbPath设置仅适用于mongod。
  dbPath: "/mongodb/sharded_cluster/myshardrs02_27518/data/db"
  journal:
    #启用或禁用持久性日志以确保数据文件保持有效和可恢复。
    enabled: true
processManagement:
  #启用在后台运行mongos或mongod进程的守护进程模式。
  fork: true
  #指定用于保存mongos或mongod进程的进程ID的文件位置，其中mongos或mongod将写入其PID
  pidFilePath: "/mongodb/sharded_cluster/myshardrs02_27518/log/mongod.pid"
net:
  #服务实例绑定所有IP
  #bindIpAll: true
  #服务实例绑定的IP
  bindIp: localhost,192.168.0.2
  #绑定的端口
  port: 27518
replication:
  replSetName: myshardrs02
sharding:
  clusterRole: shardsvr
```

启动第二套副本集：一主一副本一仲裁

依次启动三个mongod服务：

```
[root@bobohost bin]# /usr/local/mongodb/bin/mongod -f
/mongodb/sharded_cluster/myshardrs02_27318/mongod.conf
about to fork child process, waiting until server is ready for connections.
forked process: 123223
child process started successfully, parent exiting
```

```
[root@bobohost bin]# /usr/local/mongodb/bin/mongod -f
/mongodb/sharded_cluster/myshardrs02_27418/mongod.conf
about to fork child process, waiting until server is ready for connections.
forked process: 123292
child process started successfully, parent exiting
[root@bobohost bin]# /usr/local/mongodb/bin/mongod -f
/mongodb/sharded_cluster/myshardrs02_27518/mongod.conf
about to fork child process, waiting until server is ready for connections.
forked process: 123326
child process started successfully, parent exiting
```

查看服务是否启动：

```
[root@bobohost bin]# ps -ef |grep mongod
```

(1) 初始化副本集和创建主节点：

使用客户端命令连接任意一个节点，但这里尽量要连接主节点：

```
/usr/local/mongodb/bin/mongo --host 180.76.159.126 --port 27318
```

执行初始化副本集命令：

```
> rs.initiate()
```

查看副本集情况(节选内容)：

```
myshardrs01:SECONDARY> rs.status()
{
  "set" : "myshardrs01",
  .....
}
```

(2) 主节点配置查看：

```
myshardrs01:PRIMARY> rs.conf()
```

(3) 添加副本节点：

```

myshardrs01:PRIMARY> rs.add("180.76.159.126:27418")
{
  "ok" : 1,
  "operationTime" : Timestamp(1564593626, 1),
  "$clusterTime" : {
    "clusterTime" : Timestamp(1564593626, 1),
    "signature" : {
      "hash" : BinData(0,"AAAAAAAAAAAAAAAAAAAAAAAAAAAA="),
      "keyId" : NumberLong(0)
    }
  }
}

```

(4) 添加仲裁节点：

```

myshardrs01:PRIMARY> rs.addArb("180.76.159.126:27518")
{
  "ok" : 1,
  "operationTime" : Timestamp(1564593675, 1),
  "$clusterTime" : {
    "clusterTime" : Timestamp(1564593675, 1),
    "signature" : {
      "hash" : BinData(0,"AAAAAAAAAAAAAAAAAAAAAAAAAAAA="),
      "keyId" : NumberLong(0)
    }
  }
}

```

查看副本集的配置情况：

```

myshardrs01:PRIMARY> rs.conf()
myshardrs01:PRIMARY> rs.status()

```

```

myshardrs02:PRIMARY> rs.status()
{
  "set" : "myshardrs02",
  "date" : ISODate("2019-07-31T21:38:22.463Z"),
  "myState" : 1,
  "term" : NumberLong(1),
  "syncingTo" : "",
  "syncSourceHost" : "",
  "syncSourceId" : -1,
  "heartbeatIntervalMillis" : NumberLong(2000),
  "optimes" : {
    "lastCommittedOpTime" : {
      "ts" : Timestamp(1564609094, 1),
      "t" : NumberLong(1)
    },
    "readConcernMajorityOpTime" : {
      "ts" : Timestamp(1564609094, 1),

```



```

        "t" : NumberLong(1)
    },
    "appliedOpTime" : {
        "ts" : Timestamp(1564609094, 1),
        "t" : NumberLong(1)
    },
    "durableOpTime" : {
        "ts" : Timestamp(1564609094, 1),
        "t" : NumberLong(1)
    }
},
"lastStableCheckpointTimestamp" : Timestamp(1564609074, 1),
"members" : [
    {
        "_id" : 0,
        "name" : "180.76.159.126:27318",
        "health" : 1,
        "state" : 1,
        "stateStr" : "PRIMARY",
        "uptime" : 5086,
        "optime" : {
            "ts" : Timestamp(1564609094, 1),
            "t" : NumberLong(1)
        },
        "optimeDate" : ISODate("2019-07-31T21:38:14Z"),
        "syncingTo" : "",
        "syncSourceHost" : "",
        "syncSourceId" : -1,
        "infoMessage" : "",
        "electionTime" : Timestamp(1564604032, 2),
        "electionDate" : ISODate("2019-07-31T20:13:52Z"),
        "configVersion" : 3,
        "self" : true,
        "lastHeartbeatMessage" : ""
    },
    {
        "_id" : 1,
        "name" : "180.76.159.126:27418",
        "health" : 1,
        "state" : 2,
        "stateStr" : "SECONDARY",
        "uptime" : 4452,
        "optime" : {
            "ts" : Timestamp(1564609094, 1),
            "t" : NumberLong(1)
        },
        "optimeDurable" : {
            "ts" : Timestamp(1564609094, 1),
            "t" : NumberLong(1)
        },
        "optimeDate" : ISODate("2019-07-31T21:38:14Z"),
        "optimeDurableDate" : ISODate("2019-07-31T21:38:14Z"),
        "lastHeartbeat" : ISODate("2019-07-31T21:38:21.178Z"),
        "lastHeartbeatRecv" : ISODate("2019-07-
31T21:38:20.483Z"),
        "pingMs" : NumberLong(0),
        "lastHeartbeatMessage" : "",
        "syncingTo" : "180.76.159.126:27518",

```

```

"syncSourceHost" : "180.76.159.126:27518",
"syncSourceId" : 2,
"infoMessage" : "",
"configVersion" : 3
},
{
  "_id" : 2,
  "name" : "180.76.159.126:27518",
  "health" : 1,
  "state" : 2,
  "stateStr" : "SECONDARY",
  "uptime" : 4448,
  "optime" : {
    "ts" : Timestamp(1564609094, 1),
    "t" : NumberLong(1)
  },
  "optimeDurable" : {
    "ts" : Timestamp(1564609094, 1),
    "t" : NumberLong(1)
  },
  "optimeDate" : ISODate("2019-07-31T21:38:14Z"),
  "optimeDurableDate" : ISODate("2019-07-31T21:38:14Z"),
  "lastHeartbeat" : ISODate("2019-07-31T21:38:21.178Z"),
  "lastHeartbeatRecv" : ISODate("2019-07-
31T21:38:22.096Z"),
  "pingMs" : NumberLong(0),
  "lastHeartbeatMessage" : "",
  "syncingTo" : "180.76.159.126:27318",
  "syncSourceHost" : "180.76.159.126:27318",
  "syncSourceId" : 0,
  "infoMessage" : "",
  "configVersion" : 3
}
],

```

## 2.5 配置节点副本集的创建

第一步：准备存放数据和日志的目录：

```

#-----configs
#建立数据节点data和日志目录
mkdir -p /mongodb/sharded_cluster/myconfigs_27019/log \ &
mkdir -p /mongodb/sharded_cluster/myconfigs_27019/data/db \ &

mkdir -p /mongodb/sharded_cluster/myconfigs_27119/log \ &
mkdir -p /mongodb/sharded_cluster/myconfigs_27119/data/db \ &

mkdir -p /mongodb/sharded_cluster/myconfigs_27219/log \ &
mkdir -p /mongodb/sharded_cluster/myconfigs_27219/data/db

```

新建或修改配置文件：

```
vim /mongodb/sharded_cluster/myconfigs_27019/mongod.conf
```

myconfigs\_27019：

```
systemLog:
  #MongoDB发送所有日志输出的目标指定为文件
  destination: file
  #mongod或mongos应向其发送所有诊断日志记录信息的日志文件的路径
  path: "/mongodb/sharded_cluster/myconfigs_27019/log/mongod.log"
  #当mongos或mongod实例重新启动时，mongos或mongod会将新条目附加到现有日志文件的末尾。
  logAppend: true
storage:
  #mongod实例存储其数据的目录。storage.dbPath设置仅适用于mongod。
  dbPath: "/mongodb/sharded_cluster/myconfigs_27019/data/db"
  journal:
    #启用或禁用持久性日志以确保数据文件保持有效和可恢复。
    enabled: true
processManagement:
  #启用在后台运行mongos或mongod进程的守护进程模式。
  fork: true
  #指定用于保存mongos或mongod进程的进程ID的文件位置，其中mongos或mongod将写入其PID
  pidFilePath: "/mongodb/sharded_cluster/myconfigs_27019/log/mongod.pid"
net:
  #服务实例绑定所有IP
  #bindIpAll: true
  #服务实例绑定的IP
  bindIp: localhost,192.168.0.2
  #绑定的端口
  port: 27019
replication:
  replSetName: myconfigs
sharding:
  clusterRole: configsvr
```

新建或修改配置文件：

```
vim /mongodb/sharded_cluster/myconfigs_27119/mongod.conf
```

myconfigs\_27119

```
systemLog:
  #MongoDB发送所有日志输出的目标指定为文件
  destination: file
  #mongod或mongos应向其发送所有诊断日志记录信息的日志文件的路径
  path: "/mongodb/sharded_cluster/myconfigs_27119/log/mongod.log"
  #当mongos或mongod实例重新启动时，mongos或mongod会将新条目附加到现有日志文件的末尾。
  logAppend: true
storage:
  #mongod实例存储其数据的目录。storage.dbPath设置仅适用于mongod。
  dbPath: "/mongodb/sharded_cluster/myconfigs_27119/data/db"
  journal:
```

```
#启用或禁用持久性日志以确保数据文件保持有效和可恢复。
enabled: true
processManagement:
#启用在后台运行mongos或mongod进程的守护进程模式。
fork: true
#指定用于保存mongos或mongod进程的进程ID的文件位置，其中mongos或mongod将写入其PID
pidFilePath: "/mongodb/sharded_cluster/myconfigs_27119/log/mongod.pid"
net:
#服务实例绑定所有IP
#bindIpAll: true
#服务实例绑定的IP
bindIp: localhost,192.168.0.2
#绑定的端口
port: 27119
replication:
  replSetName: myconfigs
sharding:
  clusterRole: configsvr
```

新建或修改配置文件：

```
vim /mongodb/sharded_cluster/myconfigs_27219/mongod.conf
```

myconfigs\_27219

```
systemLog:
#MongoDB发送所有日志输出的目标指定为文件
destination: file
#mongod或mongos应向其发送所有诊断日志记录信息的日志文件的路径
path: "/mongodb/sharded_cluster/myconfigs_27219/log/mongod.log"
#当mongos或mongod实例重新启动时，mongos或mongod会将新条目附加到现有日志文件的末尾。
logAppend: true
storage:
#mongod实例存储其数据的目录。storage.dbPath设置仅适用于mongod。
dbPath: "/mongodb/sharded_cluster/myconfigs_27219/data/db"
journal:
#启用或禁用持久性日志以确保数据文件保持有效和可恢复。
enabled: true
processManagement:
#启用在后台运行mongos或mongod进程的守护进程模式。
fork: true
#指定用于保存mongos或mongod进程的进程ID的文件位置，其中mongos或mongod将写入其PID
pidFilePath: "/mongodb/sharded_cluster/myconfigs_27219/log/mongod.pid"
net:
#服务实例绑定所有IP
#bindIpAll: true
#服务实例绑定的IP
bindIp: localhost,192.168.0.2
#绑定的端口
port: 27219
replication:
  replSetName: myconfigs
sharding:
  clusterRole: configsvr
```

启动配置副本集：一主两副本

依次启动三个mongod服务：

```
[root@bobohost bin]# /usr/local/mongodb/bin/mongod -f
/mongodb/sharded_cluster/myconfigs_27019/mongod.conf
about to fork child process, waiting until server is ready for connections.
forked process: 123223
child process started successfully, parent exiting

[root@bobohost bin]# /usr/local/mongodb/bin/mongod -f
/mongodb/sharded_cluster/myconfigs_27119/mongod.conf
about to fork child process, waiting until server is ready for connections.
forked process: 123292
child process started successfully, parent exiting
[root@bobohost bin]# /usr/local/mongodb/bin/mongod -f
/mongodb/sharded_cluster/myconfigs_27219/mongod.conf
about to fork child process, waiting until server is ready for connections.
forked process: 123326
child process started successfully, parent exiting
```

查看服务是否启动：

```
[root@bobohost bin]# ps -ef |grep mongod
```

(1) 初始化副本集和创建主节点：

使用客户端命令连接任意一个节点，但这里尽量要连接主节点：

```
/usr/local/mongodb/bin/mongo --host 180.76.159.126 --port 27019
```

执行初始化副本集命令：

```
> rs.initiate()
```

查看副本集情况(节选内容)：

```
myshardrs01:SECONDARY> rs.status()
{
  "set" : "myshardrs01",
  .....
}
```

(2) 主节点配置查看：

```
myshardrs01:PRIMARY> rs.conf()
{
```

```
"_id" : "myshardrs01",
"version" : 1,
"protocolVersion" : NumberLong(1),
"writeConcernMajorityJournalDefault" : true,
"members" : [
  {
    "_id" : 0,
    "host" : "bobohost.localdomain:27018",
    "arbiterOnly" : false,
    "buildIndexes" : true,
    "hidden" : false,
    "priority" : 1,
    "tags" : {

    },
    "slaveDelay" : NumberLong(0),
    "votes" : 1
  }
],
```

(3) 添加两个副本节点：

```
myshardrs01:PRIMARY> rs.add("180.76.159.126:27119")
myshardrs01:PRIMARY> rs.add("180.76.159.126:27219")
```

查看副本集的配置情况：

```
myshardrs01:PRIMARY> rs.conf()
myshardrs01:PRIMARY> rs.status()
```

## 2.6 路由节点的创建和操作

### 2.6.1 第一个路由节点的创建和连接

第一步：准备存放数据和日志的目录：

```
#-----mongos01
mkdir -p /mongodb/sharded_cluster/mymongos_27017/log
```

mymongos\_27017节点：

新建或修改配置文件：

```
vi /mongodb/sharded_cluster/mymongos_27017/mongos.conf
```

mongos.conf

```
systemLog:
  #MongoDB发送所有日志输出的目标指定为文件
  destination: file
  #mongod或mongos应向其发送所有诊断日志记录信息的日志文件的路径
  path: "/mongodb/sharded_cluster/mymongos_27017/log/mongod.log"
  #当mongos或mongod实例重新启动时，mongos或mongod会将新条目附加到现有日志文件的末尾。
  logAppend: true
processManagement:
  #启用在后台运行mongos或mongod进程的守护进程模式。
  fork: true
  #指定用于保存mongos或mongod进程的进程ID的文件位置，其中mongos或mongod将写入其PID
  pidFilePath: /mongodb/sharded_cluster/mymongos_27017/log/mongod.pid"
net:
  #服务实例绑定所有IP，有副作用，副本集初始化的时候，节点名字会自动设置为本地域名，而不是ip
  #bindIpAll: true
  #服务实例绑定的IP
  bindIp: localhost,192.168.0.2
  #bindIp
  #绑定的端口
  port: 27017
sharding:
  #指定配置节点副本集
  configDB:
    myconfigs/180.76.159.126:27019,180.76.159.126:27119,180.76.159.126:27219
```

启动mongos：

```
[root@bobohost bin]# /usr/local/mongodb/bin/mongos -f
/mongodb/sharded_cluster/mymongos_27017/mongos.conf
about to fork child process, waiting until server is ready for connections.
forked process: 129874
child process started successfully, parent exiting
```

提示：启动如果失败，可以查看log目录下的日志，查看失败原因。

客户端登录mongos，

```
/usr/local/mongodb/bin/mongo --host 180.76.159.126 --port 27017
```

此时，写不进去数据，如果写数据会报错：

```
mongos> use aadb
switched to db aadb
mongos> db.aa.insert({aa:"aa"})
writeCommandError({
  "ok" : 0,
```

```
"errmsg" : "unable to initialize targeter for write op for collection
aa.aa :: caused by :: Database aa not found :: caused by :: No shards found",
"code" : 70,
"codeName" : "ShardNotFound",
"operationTime" : Timestamp(1564600123, 2),
"$clusterTime" : {
  "clusterTime" : Timestamp(1564600123, 2),
  "signature" : {
    "hash" : BinData(0,"AAAAAAAAAAAAAAAAAAAAAAAAAAAA="),
    "keyId" : NumberLong(0)
  }
}
})
```

原因：

通过路由节点操作，现在只是连接了配置节点，还没有连接分片数据节点，因此无法写入业务数据。

properties配置文件参考：

```
Logpath=/mongodb/sharded_cluster/mymongos_27017/log/mongos.log
Logappend=true
bind_ip_all=true
port=27017
fork=true
configdb=myconfigs/180.76.159.126:27019,180.76.159.126:27119,180.76.159.126:272
19
```

## 2.6.2 在路由节点上进行分片配置操作

使用命令添加分片：

(1) 添加分片：

语法：

```
sh.addShard("IP:Port")
```

将第一套分片副本集添加进来：



```

mongos>
sh.addShard("myshardrs01/192.168.0.2:27018,180.76.159.126:27118,180.76.159.126:27218")
{
  "shardAdded" : "myshardrs01",
  "ok" : 1,
  "operationTime" : Timestamp(1564611970, 4),
  "$clusterTime" : {
    "clusterTime" : Timestamp(1564611970, 4),
    "signature" : {
      "hash" : BinData(0,"AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA="),
      "keyId" : NumberLong(0)
    }
  }
}

```

查看分片状态情况：

```

mongos> sh.status()
--- Sharding Status ---
  sharding version: {
    "_id" : 1,
    "minCompatibleVersion" : 5,
    "currentVersion" : 6,
    "clusterId" : ObjectId("5d4211b798f3f9a48522c68b")
  }
  shards:
    { "_id" : "myshardrs01", "host" :
"myshardrs01/180.76.159.126:27018,180.76.159.126:27118", "state" : 1 }
  active mongoses:
    "4.0.10" : 1
  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 }

```

继续将第二套分片副本集添加进来：

```

mongos>
sh.addShard("myshardrs02/192.168.0.2:27318,180.76.159.126:27418,180.76.159.126:27518")
{
  "shardAdded" : "myshardrs02",
  "ok" : 1,
  "operationTime" : Timestamp(1564612147, 5),
  "$clusterTime" : {
    "clusterTime" : Timestamp(1564612147, 5),
    "signature" : {
      "hash" : BinData(0,"AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA="),
      "keyId" : NumberLong(0)
    }
  }
}

```

查看分片状态：

```

mongos> sh.status()
--- Sharding Status ---
  sharding version: {
    "_id" : 1,
    "minCompatibleVersion" : 5,
    "currentVersion" : 6,
    "clusterId" : ObjectId("5d4211b798f3f9a48522c68b")
  }
  shards:
    { "_id" : "myshardrs01", "host" :
"myshardrs01/180.76.159.126:27018,180.76.159.126:27118", "state" : 1 }
    { "_id" : "myshardrs02", "host" :
"myshardrs02/180.76.159.126:27318,180.76.159.126:27418", "state" : 1 }
  active mongoses:
    "4.0.10" : 1
  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 }

```

提示：如果添加分片失败，需要先手动移除分片，检查添加分片的信息的正确性后，再次添加分片。

移除分片参考(了解)：

```

use admin
db.runCommand( { removeShard: "myshardrs02" } )

```

注意：如果只剩下最后一个shard，是无法删除的

移除时会自动转移分片数据，需要一个时间过程。

完成后，再次执行删除分片命令才能真正删除。

(2) 开启分片功能：sh.enableSharding("库名")、sh.shardCollection("库名.集合名",{ "key":1})

在mongos上的artiledb数据库配置sharding:

```
mongos> sh.enableSharding("artiledb")
{
  "ok" : 1,
  "operationTime" : Timestamp(1564612296, 5),
  "$clusterTime" : {
    "clusterTime" : Timestamp(1564612296, 5),
    "signature" : {
      "hash" : BinData(0,"AAAAAAAAAAAAAAAAAAAAAAAAAAAA="),
      "keyId" : NumberLong(0)
    }
  }
}
```

查看分片状态：

```
mongos> sh.status()
--- Sharding Status ---
  sharding version: {
    "_id" : 1,
    "minCompatibleVersion" : 5,
    "currentVersion" : 6,
    "clusterId" : ObjectId("5d4211b798f3f9a48522c68b")
  }
  shards:
    { "_id" : "myshards01", "host" :
"myshards01/180.76.159.126:27018,180.76.159.126:27118", "state" : 1 }
    { "_id" : "myshards02", "host" :
"myshards02/180.76.159.126:27318,180.76.159.126:27418", "state" : 1 }
  active mongoses:
    "4.0.10" : 1
  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" : "artiledb", "primary" : "myshards02", "partitioned" :
true, "version" : { "uuid" : UUID("788c9a3b-bb6a-4cc2-a597-974694772986"),
"lastMod" : 1 } }
    { "_id" : "config", "primary" : "config", "partitioned" : true }
      config.system.sessions
        shard key: { "_id" : 1 }
        unique: false
        balancing: true
        chunks:
          myshards01      1
          { "_id" : { "$minKey" : 1 } } --> { "_id" : { "$maxKey"
: 1 } } } on : myshards01 Timestamp(1, 0)
```

### (3) 集合分片

对集合分片，你必须使用 `sh.shardCollection()` 方法指定集合和分片键。

语法：

```
sh.shardCollection(namespace, key, unique)
```

参数：

Parameter	Type	Description
<code>namespace</code>	string	要（分片）共享的目标集合的命名空间，格式： <code>&lt;database&gt;.&lt;collection&gt;</code>
<code>key</code>	document	用作分片键的索引规范文档。shard键决定MongoDB如何在shard之间分发文档。除非集合为空，否则索引必须在shard collection命令之前存在。如果集合为空，则MongoDB在对集合进行分片之前创建索引，前提是支持分片键的索引不存在。简单的说：由包含字段和该字段的索引遍历方向的文档组成。
<code>unique</code>	boolean	当值为true情况下，片键字段上会限制为确保是唯一索引。哈希策略片键不支持唯一索引。默认是false。

对集合进行分片时,你需要选择一个片键（Shard Key），shard key 是每条记录都必须包含的,且建立了索引的单个字段或复合字段,MongoDB按照片键将数据划分到不同的数据块中,并将数据块均衡地分布到所有分片中.为了按照片键划分数据块,MongoDB使用基于哈希的分片方式（随机平均分配）或者基于范围的分片方式（数值大小分配）。

用什么字段当片键都可以，如：nickname作为片键，但一定是必填字段。

#### 分片规则一：哈希策略

对于基于哈希的分片,MongoDB计算一个字段的哈希值,并用这个哈希值来创建数据块。

在使用基于哈希分片的系统中,拥有“相近”片键的文档很可能不会存储在同一个数据块中,因此数据的分离性更好一些。

使用nickname作为片键，根据其值的哈希值进行数据分片

```
mongos> sh.shardCollection("artikledb.comment",{"nickname":"hashed"})
{
  "collectionsharded" : "artikledb.comment",
  "collectionUUID" : UUID("ddea6ed8-ee61-4693-bd16-196acc3a45e8"),
  "ok" : 1,
  "operationTime" : Timestamp(1564612840, 28),
  "$clusterTime" : {
    "clusterTime" : Timestamp(1564612840, 28),
    "signature" : {
      "hash" : BinData(0,"AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA"),
```

```

        "keyId" : NumberLong(0)
      }
    }
  }
}

```

查看分片状态：sh.status()

```

databases:
  { "_id" : "artileddb", "primary" : "myshardrs02", "partitioned" :
true, "version" : { "uuid" : UUID("251436b7-86c2-4cd8-9a88-70874af29364"),
"lastMod" : 1 } }
  artileddb.comment
    shard key: { "nickname" : "hashed" }
    unique: false
    balancing: true
    chunks:
      myshardrs01    2
      myshardrs02    2
      { "nickname" : { "$minKey" : 1 } } --> { "nickname" :
NumberLong("-4611686018427387902") } on : myshardrs01 Timestamp(1, 0)
      { "nickname" : NumberLong("-4611686018427387902") } -->
{ "nickname" : NumberLong(0) } on : myshardrs01 Timestamp(1, 1)
      { "nickname" : NumberLong(0) } --> { "nickname" :
NumberLong("4611686018427387902") } on : myshardrs02 Timestamp(1, 2)
      { "nickname" : NumberLong("4611686018427387902") } -->
{ "nickname" : { "$maxKey" : 1 } } on : myshardrs02 Timestamp(1, 3)
  { "_id" : "config", "primary" : "config", "partitioned" : true }
  config.system.sessions
    shard key: { "_id" : 1 }
    unique: false
    balancing: true
    chunks:
      myshardrs01    1
      { "_id" : { "$minKey" : 1 } } --> { "_id" : { "$maxKey"
: 1 } } on : myshardrs01 Timestamp(1, 0)

```

分片规则二：范围策略

对于 *基于范围的分片*，MongoDB按照片键的范围把数据分成不同部分。假设有一个数字的片键：想象一个从负无穷到正无穷的直线，每一个片键的值都在直线上画了一个点。MongoDB把这条直线划分为更短的不重叠的片段，并称之为 *数据块*，每个数据块包含了片键在一定范围内的数据。

在使用片键做范围划分的系统中，拥有“相近”片键的文档很可能存储在同一个数据块中，因此也会存储在同一个分片中。

如使用作者年龄字段作为片键，按照点赞数的值进行分片：

```

mongos> sh.shardCollection("artileddb.author",{ "age":1})
{
  "collectionsharded" : "artileddb.author",
  "collectionUUID" : UUID("9a47bdaa-213a-4039-9c18-e70bfc369df7"),
  "ok" : 1,
  "operationTime" : Timestamp(1567512803, 13),

```

```

"$clusterTime" : {
  "clusterTime" : Timestamp(1567512803, 13),
  "signature" : {
    "hash" : BinData(0,"eE9QT5yE5sL1Tyr7+3U8GRy5+5Q="),
    "keyId" : NumberLong("6732061237309341726")
  }
}
}

```

注意的是：

- 1) 一个集合只能指定一个片键，否则报错。
- 2) 一旦对一个集合分片，分片键和分片值就不可改变。如：不能给集合选择不同的分片键、不能更新分片键的值。
- 3) 根据age索引进行分配数据。

查看分片状态：

```

articledb.author
      shard key: { "age" : 1 }
      unique: false
      balancing: true
      chunks:
        myshardrs01 1
        { "age" : { "$minKey" : 1 } } --> { "age" : { "$maxKey"
: 1 } } on : myshardrs01 Timestamp(1, 0)

```

基于范围的分片方式与基于哈希的分片方式性能对比：

基于范围的分片方式提供了更高效的范围查询,给定一个片键的范围,分发路由可以很简单地确定哪个数据块存储了请求需要的数据,并将请求转发到相应的分片中。

不过,基于范围的分片会导致数据在不同分片上的不均衡,有时候,带来的消极作用会大于查询性能的积极作用.比如,如果片键所在的字段是线性增长的,一定时间内的所有请求都会落到某个固定的数据块中,最终导致分布在同一个分片中.在这种情况下,一小部分分片承载了集群大部分的数据,系统并不能很好地进行扩展.

与此相比,基于哈希的分片方式以范围查询性能的损失为代价,保证了集群中数据的均衡.哈希值的随机性使数据随机分布在每个数据块中,因此也随机分布在不同分片中.但是也正由于随机性,一个范围查询很难确定应该请求哪些分片,通常为了返回需要的结果,需要请求所有分片.

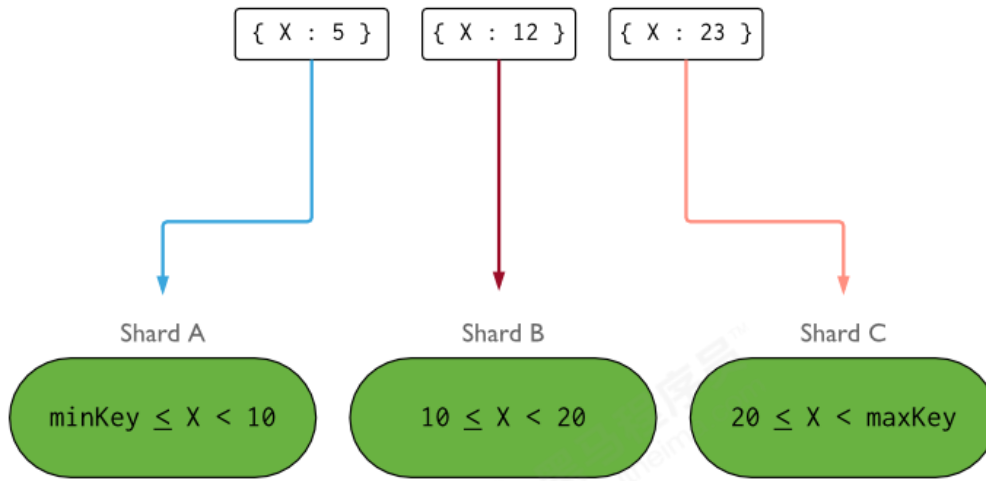
如无特殊情况，一般推荐使用 Hash Sharding。

而使用 `_id` 作为片键是一个不错的选择，因为它是必有的，你可以使用数据文档 `_id` 的哈希作为片键。

这个方案能够是的读和写都能够平均分布，并且它能够保证每个文档都有不同的片键所以数据块能够很精细。

似乎还是不够完美，因为这样的话对多个文档的查询必将命中所有的分片。虽说如此，这也是一种比较好的方案了。

理想化的 shard key 可以让 documents 均匀地在集群中分布：



显示集群的详细信息：

```
mongos> db.printShardingStatus()
```

查看均衡器是否工作（需要重新均衡时系统才会自动启动，不用管它）：

```
mongos> sh.isBalancerRunning()
false
```

查看当前Balancer状态：

```
mongos> sh.getBalancerState()
true
```

## 2.6.3 分片后插入数据测试

测试一（哈希规则）：登录mongos后，向comment循环插入1000条数据做测试：

```
mongos> use articledb
switched to db articledb
mongos> for(var i=1;i<=1000;i++)
{db.comment.insert({_id:i+"" ,nickname:"BoBo"+i})}
WriteResult({ "nInserted" : 1 })
mongos> db.comment.count()
1000
```

提示：js的语法，因为mongo的shell是一个JavaScript的shell。

注意：从路由上插入的数据，必须包含片键，否则无法插入。

分别登陆两个片的主节点，统计文档数量

第一个分片副本集：

```
/usr/local/mongodb/bin/mongo --host 180.76.159.126 --port 27018
```





如果查看状态发现没有分片，则可能是由于以下原因造成了：

1) 系统繁忙，正在分片中。

2) 数据块 ( chunk ) 没有填满，默认的数据块尺寸 ( chunksize ) 是64M，填满后才会考虑向其他片的数据块填充数据，因此，为了测试，可以将其改小，这里改为1M，操作如下：

```
use config
db.settings.save( { _id:"chunksize", value: 1 } )
```

测试完改回来：

```
db.settings.save( { _id:"chunksize", value: 64 } )
```

注意：要先改小，再设置分片。为了测试，可以先删除集合，重新建立集合的分片策略，再插入数据测试即可。

## 2.6.4 再增加一个路由节点

文件夹：

```
#-----mongos02
mkdir -p /mongodb/sharded_cluster/mymongos_27117/log
```

新建或修改配置文件：

```
vi /mongodb/sharded_cluster/mymongos_27117/mongos.conf
```

mongos.conf

```
systemLog:
  #MongoDB发送所有日志输出的目标指定为文件
  destination: file
  #mongod或mongos应向其发送所有诊断日志记录信息的日志文件的路径
  path: "/mongodb/sharded_cluster/mymongos_27117/log/mongod.log"
  #当mongos或mongod实例重新启动时，mongos或mongod会将新条目附加到现有日志文件的末尾。
  logAppend: true
processManagement:
  #启用在后台运行mongos或mongod进程的守护进程模式。
  fork: true
  #指定用于保存mongos或mongod进程的进程ID的文件位置，其中mongos或mongod将写入其PID
  pidFilePath: /mongodb/sharded_cluster/mymongos_27117/log/mongod.pid"
net:
  #服务实例绑定所有IP，有副作用，副本集初始化的时候，节点名字会自动设置为本地域名，而不是ip
  #bindIpAll: true
  #服务实例绑定的IP
  bindIp: localhost,192.168.0.2
  #bindIp
  #绑定的端口
  port: 27117
sharding:
```

```
configDB:
myconfigs/180.76.159.126:27019,180.76.159.126:27119,180.76.159.126:27219
```

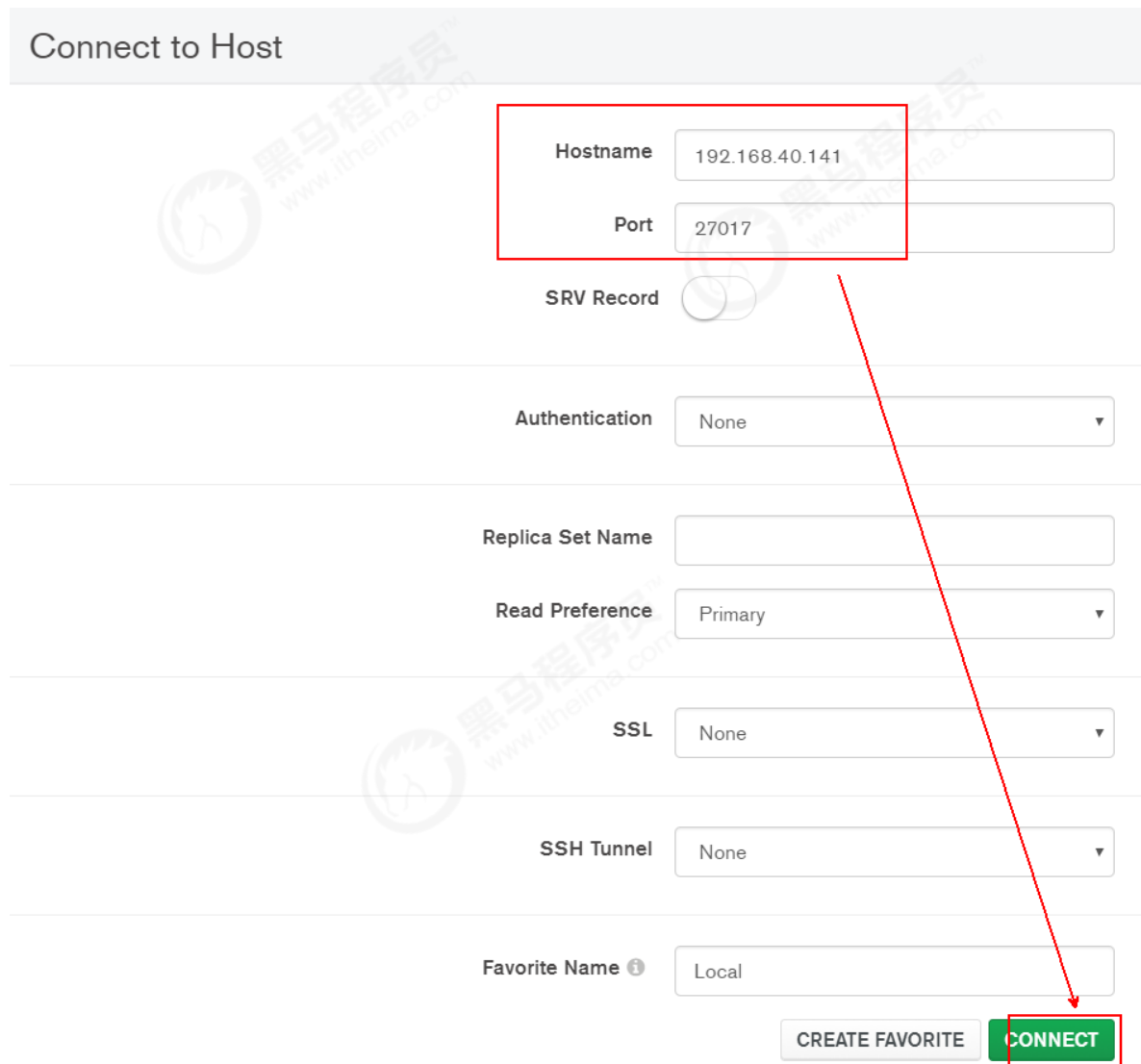
启动mongos2 :

```
[root@bobohost bin]# /usr/local/mongodb/bin/mongos -f
/mongodb/sharded_cluster/mymongos_27117/mongos.conf
about to fork child process, waiting until server is ready for connections.
forked process: 129874
child process started successfully, parent exiting
```

使用mongo客户端登录27117，发现，第二个路由无需配置，因为分片配置都保存到了配置服务器中了。

## 2.7 Compass连接分片集群

compass连接：



Connect to Host

Hostname: 192.168.40.141

Port: 27017

SRV Record:

Authentication: None

Replica Set Name:

Read Preference: Primary

SSL: None

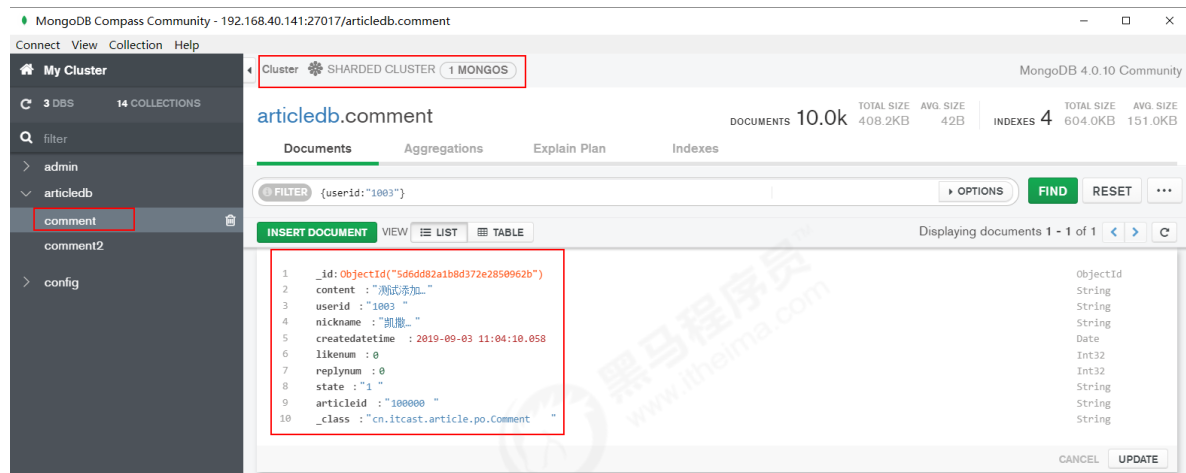
SSH Tunnel: None

Favorite Name: Local

CREATE FAVORITE CONNECT

提示：和连接单机mongod一样。

连接成功后，上方有mongos和分片集群的提示：



## 2.8 SpringDataMongoDB连接分片集群

Java客户端常用的是SpringDataMongoDB，其连接的是mongos路由，配置和单机mongod的配置是一样的。

多个路由的时候的SpringDataMongoDB的客户端配置参考如下：

```
spring:
  #数据源配置
  data:
    mongodb:
      # 主机地址
      # host: 180.76.159.126
      # 数据库
      # database: articledb
      # 默认端口是27017
      # port: 27017
      #也可以使用uri连接
      # uri: mongodb://192.168.40.134:28017/articledb
      # 连接副本集字符串
      # uri:
      mongodb://180.76.159.126:27017,180.76.159.126:27018,180.76.159.126:27019/articledb?connect=replicaSet&slaveOk=true&replicaSet=myrs
      #连接路由字符串
      uri: mongodb://180.76.159.126:27017,180.76.159.126:27117/articledb
```

通过日志发现，写入数据的时候，会选择一个路由写入：

```

2019-09-03 11:04:09.166 INFO 11816 --- [68.40.141:27117]
org.mongodb.driver.connection : Opened connection
[connectionId{localValue:2}] to 180.76.159.126:27117
2019-09-03 11:04:09.166 INFO 11816 --- [68.40.141:27017]
org.mongodb.driver.connection : Opened connection
[connectionId{localValue:1}] to 180.76.159.126:27017

2019-09-03 11:04:09.529 INFO 11816 --- [ main]
org.mongodb.driver.connection : Opened connection
[connectionId{localValue:3}] to 180.76.159.126:27117
2019-09-03 11:04:09.826 INFO 11816 --- [ main]
c.i.article.service.CommentServiceTest : Started CommentServiceTest in 7.009
seconds (JVM running for 8.043)
2019-09-03 11:04:10.173 INFO 11816 --- [ Thread-2]
org.mongodb.driver.connection : Closed connection
[connectionId{localValue:3}] to 180.76.159.126:27117 because the

```

## 2.9 清除所有的节点数据（备用）

如果在搭建分片的时候有操作失败或配置有问题，需要重新来过的，可以进行如下操作：

第一步：查询出所有的测试服务节点的进程：

```

[root@bobohost sharded_cluster]# ps -ef |grep mongo
root      10184      1  0 06:04 ?        00:01:25 /usr/local/mongodb/bin/mongod
-f /mongodb/sharded_cluster/myshardrs01_27018/mongod.conf
root      10219      1  0 06:04 ?        00:01:25 /usr/local/mongodb/bin/mongod
-f /mongodb/sharded_cluster/myshardrs01_27118/mongod.conf
root      10253      1  0 06:04 ?        00:00:46 /usr/local/mongodb/bin/mongod
-f /mongodb/sharded_cluster/myshardrs01_27218/mongod.conf
root      10312      1  0 06:04 ?        00:01:23 /usr/local/mongodb/bin/mongod
-f /mongodb/sharded_cluster/myshardrs02_27318/mongod.conf
root      10346      1  0 06:05 ?        00:01:23 /usr/local/mongodb/bin/mongod
-f /mongodb/sharded_cluster/myshardrs02_27418/mongod.conf
root      10380      1  0 06:05 ?        00:00:44 /usr/local/mongodb/bin/mongod
-f /mongodb/sharded_cluster/myshardrs02_27518/mongod.conf
root      10414      1  1 06:05 ?        00:01:36 /usr/local/mongodb/bin/mongod
-f /mongodb/sharded_cluster/myconfigs_27019/mongod.conf
root      10453      1  1 06:05 ?        00:01:37 /usr/local/mongodb/bin/mongod
-f /mongodb/sharded_cluster/myconfigs_27119/mongod.conf
root      10492      1  1 06:05 ?        00:01:38 /usr/local/mongodb/bin/mongod
-f /mongodb/sharded_cluster/myconfigs_27219/mongod.conf
root      11392      1  0 06:15 ?        00:00:24 /usr/local/mongodb/bin/mongos
--config /mongodb/sharded_cluster/mymongos_27017/mongos.cfg
root      14829      1  0 07:15 ?        00:00:13 /usr/local/mongodb/bin/mongos
--config /mongodb/sharded_cluster/mymongos_27117/mongos.cfg

```

根据上述的进程编号，依次中断进程：

```
kill -2 进程编号
```

第二步：清除所有的节点的数据：

```
rm -rf /mongodb/sharded_cluster/myconfigs_27019/data/db/*.* \ &
rm -rf /mongodb/sharded_cluster/myconfigs_27119/data/db/*.* \ &
rm -rf /mongodb/sharded_cluster/myconfigs_27219/data/db/*.* \ &
rm -rf /mongodb/sharded_cluster/myshardrs01_27018/data/db/*.* \ &
rm -rf /mongodb/sharded_cluster/myshardrs01_27118/data/db/*.* \ &
rm -rf /mongodb/sharded_cluster/myshardrs01_27218/data/db/*.* \ &
rm -rf /mongodb/sharded_cluster/myshardrs02_27318/data/db/*.* \ &
rm -rf /mongodb/sharded_cluster/myshardrs02_27418/data/db/*.* \ &
rm -rf /mongodb/sharded_cluster/myshardrs02_27518/data/db/*.* \ &
rm -rf /mongodb/sharded_cluster/mymongos_27017/data/db/*.* \ &
rm -rf /mongodb/sharded_cluster/mymongos_27117/data/db/*.*
```

第三步：查看或修改有问题的配置

第四步：依次启动所有节点，不包括路由节点：

```
/usr/local/mongodb/bin/mongod -f
/mongodb/sharded_cluster/myshardrs01_27018/mongod.conf
/usr/local/mongodb/bin/mongod -f
/mongodb/sharded_cluster/myshardrs01_27118/mongod.conf
/usr/local/mongodb/bin/mongod -f
/mongodb/sharded_cluster/myshardrs01_27218/mongod.conf
/usr/local/mongodb/bin/mongod -f
/mongodb/sharded_cluster/myshardrs02_27318/mongod.conf
/usr/local/mongodb/bin/mongod -f
/mongodb/sharded_cluster/myshardrs02_27418/mongod.conf
/usr/local/mongodb/bin/mongod -f
/mongodb/sharded_cluster/myshardrs02_27518/mongod.conf
/usr/local/mongodb/bin/mongod -f
/mongodb/sharded_cluster/myconfigs_27019/mongod.conf
/usr/local/mongodb/bin/mongod -f
/mongodb/sharded_cluster/myconfigs_27119/mongod.conf
/usr/local/mongodb/bin/mongod -f
/mongodb/sharded_cluster/myconfigs_27219/mongod.conf
```

第五步：对两个数据分片副本集和一个配置副本集进行初始化和相关配置

第六步：检查路由mongos的配置，并启动mongos

```
/usr/local/mongodb/bin/mongod -f
/mongodb/sharded_cluster/mymongos_27017/mongos.cfg
/usr/local/mongodb/bin/mongod -f
/mongodb/sharded_cluster/mymongos_27017/mongos.cfg
```

第七步：mongo登录mongos，在其上进行相关操作。

# 3. 安全认证

## 3.1 MongoDB的用户和角色权限简介

默认情况下，MongoDB实例启动运行时是没有启用用户访问权限控制的，也就是说，在实例本机服务器上都可以随意连接到实例进行各种操作，MongoDB不会对连接客户端进行用户验证，这是非常危险的。

mongodb官网上说，为了能保障mongodb的安全可以做以下几个步骤：

- 1) 使用新的端口，默认的27017端口如果一旦知道了ip就能连接上，不太安全。
- 2) 设置mongodb的网络环境，最好将mongodb部署到公司服务器内网，这样外网是访问不到的。公司内部访问使用vpn等。
- 3) 开启安全认证。认证要同时设置服务器之间的内部认证方式，同时要设置客户端连接到集群的账号密码认证方式。

为了强制开启用户访问控制(用户验证)，则需要在MongoDB实例启动时使用选项 `--auth` 或在指定启动配置文件中添加选项 `auth=true`。

在开始之前需要了解一下概念

- 1) 启用访问控制：

MongoDB使用的是基于角色的访问控制(Role-Based Access Control, RBAC)来管理用户对实例的访问。通过对用户授予一个或多个角色来控制用户访问数据库资源的权限和数据库操作的权限，在对用户分配角色之前，用户无法访问实例。

在实例启动时添加选项 `--auth` 或指定启动配置文件中添加选项 `auth=true`。

- 2) 角色：

在MongoDB中通过角色对用户授予相应数据库资源的操作权限，每个角色当中的权限可以显式指定，也可以通过继承其他角色的权限，或者两都存在的权限。

- 3) 权限：

权限由指定的数据库资源(resource)以及允许在指定资源上进行的操作(action)组成。

1. 资源(resource)包括：数据库、集合、部分集合和集群；
2. 操作(action)包括：对资源进行的增、删、改、查(CRUD)操作。

在角色定义时可以包含一个或多个已存在的角色，新创建的角色会继承包含的角色所有的权限。在同一个数据库中，新创建角色可以继承其他角色的权限，在 `admin` 数据库中创建的角色可以继承在其它任意数据库中角色的权限。

关于角色权限的查看，可以通过如下命令查询（了解）：

```
// 查询所有角色权限(仅用户自定义角色)
> db.runCommand({ rolesInfo: 1 })

// 查询所有角色权限(包含内置角色)
> db.runCommand({ rolesInfo: 1, showBuiltinRoles: true })
```

```

// 查询当前数据库中的某角色的权限
> db.runCommand({ rolesInfo: "<rolename>" })

// 查询其它数据库中指定的角色权限
> db.runCommand({ rolesInfo: { role: "<rolename>", db: "<database>" } })

// 查询多个角色权限
> db.runCommand(
  {
    rolesInfo: [
      "<rolename>",
      { role: "<rolename>", db: "<database>" },
      ...
    ]
  }
)

```

示例：

查看所有内置角色：

```

> db.runCommand({ rolesInfo: 1, showBuiltinRoles: true })
{
  "roles" : [
    {
      "role" : "__queryableBackup",
      "db" : "admin",
      "isBuiltin" : true,
      "roles" : [ ],
      "inheritedRoles" : [ ]
    },
    {
      "role" : "__system",
      "db" : "admin",
      "isBuiltin" : true,
      "roles" : [ ],
      "inheritedRoles" : [ ]
    },
    {
      "role" : "backup",
      "db" : "admin",
      "isBuiltin" : true,
      "roles" : [ ],
      "inheritedRoles" : [ ]
    },
    {
      "role" : "clusterAdmin",
      "db" : "admin",
      "isBuiltin" : true,
      "roles" : [ ],
      "inheritedRoles" : [ ]
    },
    {
      "role" : "clusterManager",

```

```
    "db" : "admin",
    "isBuiltin" : true,
    "roles" : [ ],
    "inheritedRoles" : [ ]
  },
  {
    "role" : "clusterMonitor",
    "db" : "admin",
    "isBuiltin" : true,
    "roles" : [ ],
    "inheritedRoles" : [ ]
  },
  {
    "role" : "dbAdmin",
    "db" : "admin",
    "isBuiltin" : true,
    "roles" : [ ],
    "inheritedRoles" : [ ]
  },
  {
    "role" : "dbAdminAnyDatabase",
    "db" : "admin",
    "isBuiltin" : true,
    "roles" : [ ],
    "inheritedRoles" : [ ]
  },
  {
    "role" : "dbOwner",
    "db" : "admin",
    "isBuiltin" : true,
    "roles" : [ ],
    "inheritedRoles" : [ ]
  },
  {
    "role" : "enableSharding",
    "db" : "admin",
    "isBuiltin" : true,
    "roles" : [ ],
    "inheritedRoles" : [ ]
  },
  {
    "role" : "hostManager",
    "db" : "admin",
    "isBuiltin" : true,
    "roles" : [ ],
    "inheritedRoles" : [ ]
  },
  {
    "role" : "read",
    "db" : "admin",
    "isBuiltin" : true,
    "roles" : [ ],
    "inheritedRoles" : [ ]
  },
  {
    "role" : "readAnyDatabase",
    "db" : "admin",
    "isBuiltin" : true,
```



```

        "roles" : [ ],
        "inheritedRoles" : [ ]
    },
    {
        "role" : "readwrite",
        "db" : "admin",
        "isBuiltin" : true,
        "roles" : [ ],
        "inheritedRoles" : [ ]
    },
    {
        "role" : "readwriteAnyDatabase",
        "db" : "admin",
        "isBuiltin" : true,
        "roles" : [ ],
        "inheritedRoles" : [ ]
    },
    {
        "role" : "restore",
        "db" : "admin",
        "isBuiltin" : true,
        "roles" : [ ],
        "inheritedRoles" : [ ]
    },
    {
        "role" : "root",
        "db" : "admin",
        "isBuiltin" : true,
        "roles" : [ ],
        "inheritedRoles" : [ ]
    },
    {
        "role" : "userAdmin",
        "db" : "admin",
        "isBuiltin" : true,
        "roles" : [ ],
        "inheritedRoles" : [ ]
    },
    {
        "role" : "userAdminAnyDatabase",
        "db" : "admin",
        "isBuiltin" : true,
        "roles" : [ ],
        "inheritedRoles" : [ ]
    }
],
"ok" : 1
}

```

常用的内置角色：

- 数据库用户角色：read、readWrite;
- 所有数据库用户角色：readAnyDatabase、readWriteAnyDatabase、userAdminAnyDatabase、dbAdminAnyDatabase

- 数据库管理角色：dbAdmin、dbOwner、userAdmin；
- 集群管理角色：clusterAdmin、clusterManager、clusterMonitor、hostManager；
- 备份恢复角色：backup、restore；
- 超级用户角色：root
- 内部角色：system

角色说明：

角色	权限描述
read	可以读取指定数据库中任何数据。
readWrite	可以读写指定数据库中任何数据，包括创建、重命名、删除集合。
readAnyDatabase	可以读取所有数据库中任何数据(除了数据库config和local之外)。
readWriteAnyDatabase	可以读写所有数据库中任何数据(除了数据库config和local之外)。
userAdminAnyDatabase	可以在指定数据库创建和修改用户(除了数据库config和local之外)。
dbAdminAnyDatabase	可以读取任何数据库以及对数据库进行清理、修改、压缩、获取统计信息、执行检查等操作(除了数据库config和local之外)。
dbAdmin	可以读取指定数据库以及对数据库进行清理、修改、压缩、获取统计信息、执行检查等操作。
userAdmin	可以在指定数据库创建和修改用户。
clusterAdmin	可以对整个集群或数据库系统进行管理操作。
backup	备份MongoDB数据最小的权限。
restore	从备份文件中还原恢复MongoDB数据(除了system.profile集合)的权限。
root	超级账号，超级权限

## 3.2 单实例环境

目标：对单实例的MongoDB服务开启安全认证，这里的单实例指的是未开启副本集或分片的MongoDB实例。

### 3.2.1 关闭已开启的服务（可选）

增加mongod的单实例的安全认证功能，可以在服务搭建的时候直接添加，也可以在之前搭建好的服务上添加。

本文使用之前搭建好的服务，因此，先停止之前的服务

停止服务的方式有两种：快速关闭和标准关闭，下面依次说明：

(1) 快速关闭方法（快速，简单，数据可能会出错）

目标：通过系统的kill命令直接杀死进程：

杀完要检查一下，避免有的没有杀掉。

```
#通过进程编号关闭节点  
kill -2 54410
```

### 【补充】

如果一旦是因为数据损坏，则需要进行如下操作（了解）：

1) 删除lock文件：

```
rm -f /mongodb/single/data/db/*.lock
```

2) 修复数据：

```
/usr/local/mongodb/bin/mongod --repair --dbpath=/mongodb/single/data/db
```

(2) 标准的关闭方法（数据不容易出错，但麻烦）：

目标：通过mongo客户端中的shutdownServer命令来关闭服务

主要的操作步骤参考如下：

```
//客户端登录服务，注意，这里通过localhost登录，如果需要远程登录，必须先登录认证才行。  
mongo --port 27017  
//#切换到admin库  
use admin  
//关闭服务  
db.shutdownServer()
```

## 3.2.2 添加用户和权限

(1) 先按照普通无授权认证的配置，来配置服务端的配置文件 /mongodb/single/mongod.conf：  
(参考，复用之前课程的)

```
systemLog:  
#MongoDB发送所有日志输出的目标指定为文件  
destination: file  
#mongod或mongos应向其发送所有诊断日志记录信息的日志文件的路径  
path: "/mongodb/single/log/mongod.log"  
#当mongos或mongod实例重新启动时，mongos或mongod会将新条目附加到现有日志文件的末尾。  
logAppend: true  
storage:  
#mongod实例存储其数据的目录。storage.dbPath设置仅适用于mongod。  
dbPath: "/mongodb/single/data/db"  
journal:  
#启用或禁用持久性日志以确保数据文件保持有效和可恢复。  
enabled: true  
processManagement:  
#启用在后台运行mongos或mongod进程的守护进程模式。  
fork: true  
#指定用于保存mongos或mongod进程的进程ID的文件位置，其中mongos或mongod将写入其PID
```

```
pidFilePath: "/mongodb/single/log/mongod.pid"
net:
  #服务实例绑定的IP
  bindIp: localhost,192.168.0.2
  #绑定的端口
  port: 27017
```

(2) 按之前未开启认证的方式 (不添加 `--auth` 参数) 来启动MongoDB服务:

```
/usr/local/mongodb/bin/mongod -f /mongodb/single/mongod.conf
```

提示:

在操作用户时, 启动mongod服务时尽量不要开启授权。

(3) 使用Mongo客户端登录:

```
/usr/local/mongodb/bin/mongo --host 180.76.159.126 --port 27017
```

(4) 创建两个管理员用户, 一个是系统的超级管理员 `myroot`, 一个是admin库的管理用户 `myadmin`:

```
//切换到admin库
> use admin
//创建系统超级用户 myroot, 设置密码123456, 设置角色root
//> db.createUser({user:"myroot",pwd:"123456",roles:[ { "role" : "root", "db" :
"admin" } ]})
//或
> db.createUser({user:"myroot",pwd:"123456",roles:["root"]})
Successfully added user: { "user" : "myroot", "roles" : [ "root" ] }
//创建专门用来管理admin库的账号myadmin, 只用来作为用户权限的管理
> db.createUser({user:"myadmin",pwd:"123456",roles:
[{"role":"userAdminAnyDatabase",db:"admin"}]})
Successfully added user: {
  "user" : "myadmin",
  "roles" : [
    {
      "role" : "userAdminAnyDatabase",
      "db" : "admin"
    }
  ]
}
//查看已经创建了的用户的情况:
> db.system.users.find()
```

```

{ "_id" : "admin.myroot", "userId" : UUID("9a0a698c-73ad-4c45-8f33-
e8a90d3ad689"), "user" : "myroot", "db" : "admin", "credentials" : { "SCRAM-SHA-
1" : { "iterationCount" : 10000, "salt" : "4tXXi9g9wMlrR32e+nleyA==",
"storedKey" : "78EXQoweA6lLYTTzcQrtJuwLcmg=", "serverKey" :
"xwze/lGcQ7FI5cSFoily4CW4wks=" }, "SCRAM-SHA-256" : { "iterationCount" : 15000,
"salt" : "+Hq2Y6PiNfKDEBYFertTazSWI9FqbkYGdHaFkg==", "storedKey" :
"guZ5wyl7dsjtu77Isw2dsJ+Ck4fKiKZteUh/CuJoQj4=", "serverKey" :
"4wiBApuB435LNPP49DXKwJ+YGcRaWZNVeq/Ibkr5Lxo=" } }, "roles" : [ { "role" :
"root", "db" : "admin" } ] }
{ "_id" : "admin.myadmin", "userId" : UUID("be9c832e-f894-4ffd-b76b-
62940707aab2"), "user" : "myadmin", "db" : "admin", "credentials" : { "SCRAM-
SHA-1" : { "iterationCount" : 10000, "salt" : "KIIOsnfp5kTgpUExtTKDTA==",
"storedKey" : "39509XHqiwi8HWLc6qMDf13wcSs=", "serverKey" :
"zKkJAKH3HgPL35/a6hkhBaCD1WE=" }, "SCRAM-SHA-256" : { "iterationCount" : 15000,
"salt" : "v76GZgBAKswNewB7mo6dhSE59ME3HFJ5T9UXlQ==", "storedKey" :
"CwHtBiww04Y0hychKq4VIS5QnnuAne59+iPFooIhkk=", "serverKey" :
"HQk3RTSWDABGwxyPEiEC2+iK/rGTL6ROAD0HQEJI0F8=" } }, "roles" : [ { "role" :
"userAdminAnyDatabase", "db" : "admin" } ] }
//删除用户
> db.dropUser("myadmin")
true
> db.system.users.find()

//修改密码
> db.changeUserPassword("myroot", "123456")

```

提示：

1) 本案例创建了两个用户，分别对应超管和专门用来管理用户的角色，事实上，你只需要一个用户即可。如果你对安全要求很高，防止超管泄漏，则不要创建超管用户。

2) 和其它数据库（MySQL）一样，权限的管理都差不多一样，也是将用户和权限信息保存到数据库对应的表中。Mongodb存储所有的用户信息在admin数据库的集合system.users中，保存用户名、密码和数据库信息。

3) 如果不指定数据库，则创建的指定的权限的用户在所有的数据库上有效，如 {role: "userAdminAnyDatabase", db:""} }

## (5) 认证测试

测试添加的用户是否正确

```

//切换到admin
> use admin
//密码输错
> db.auth("myroot","12345")
Error: Authentication failed.
0
//密码正确
> db.auth("myroot","123456")
1

```

## (6) 创建普通用户

创建普通用户可以在没有开启认证的时候添加，也可以在开启认证之后添加，但开启认证之后，必须使用有操作admin库的用户登录认证后才能操作。底层都是将用户信息保存在了admin数据库的集合system.users中。

```
//创建(切换)将来要操作的数据库articledb,
> use articledb
switched to db articledb
//创建用户，拥有articledb数据库的读写权限readWrite，密码是123456
> db.createUser({user: "bobo", pwd: "123456", roles: [{ role: "readwrite", db:
"articledb" } ]})
//> db.createUser({user: "bobo", pwd: "123456", roles: ["readwrite"]})
Successfully added user: {
  "user" : "bobo",
  "roles" : [
    {
      "role" : "readwrite",
      "db" : "articledb"
    }
  ]
}
//测试是否可用
> db.auth("bobo","123456")
1
```

提示：

如果开启了认证后，登录的客户端的用户必须使用admin库的角色，如拥有root角色的myadmin用户，再通过myadmin用户去创建其他角色的用户

### 3.2.3 服务端开启认证和客户端连接登录

(1) 关闭已经启动的服务

1) 使用linux命令杀死进程：

```
[root@bobohost single]# ps -ef |grep mongo
root      23482      1  0 08:08 ?        00:00:55 /usr/local/mongodb/bin/mongod
-f /mongodb/single/mongod.conf
[root@bobohost single]# kill -2 23482
```

2) 在mongo客户端中使用shutdownServer命令来关闭。

```
> db.shutdownServer()
shutdown command only works with the admin database; try 'use admin'
> use admin
switched to db admin
> db.shutdownServer()
2019-08-14T11:20:16.450+0800 E QUERY    [js] Error: shutdownServer failed: {
  "ok" : 0,
  "errmsg" : "shutdown must run from localhost when running db without
auth",
  "code" : 13,
  "codeName" : "Unauthorized"
```

```
} :
_getErrorWithCode@src/mongo/shell/utils.js:25:13
DB.prototype.shutdownServer@src/mongo/shell/db.js:453:1
@(shell):1:1
```

需要几个条件：

- 必须是在admin库下执行该关闭服务命令。
- 如果没有开启认证，必须是从localhost登陆的，才能执行关闭服务命令。
- 非localhost的、通过远程登录的，必须有登录且必须登录用户有对admin操作权限才可以。

(2) 以开启认证的方式启动服务

有两种方式开启权限认证启动服务：一种是参数方式，一种是配置文件方式。

1) 参数方式

在启动时指定参数 `--auth`，如：

```
/usr/local/mongodb/bin/mongod -f /mongodb/single/mongod.conf --auth
```

2) 配置文件方式

在mongod.conf配置文件中加入：

vim /mongodb/single/mongod.conf

```
security:
  #开启授权认证
  authorization: enabled
```

启动时可不加 `--auth` 参数：

```
/usr/local/mongodb/bin/mongod -f /mongodb/single/mongod.conf
```

(3) 开启了认证的情况下的客户端登录

有两种认证方式，一种是先登录，在mongo shell中认证；一种是登录时直接认证。

1) 先连接再认证

```
[root@bobohost bin]# /usr/local/mongodb/bin/mongo --host 180.76.159.126 --port 27017
MongoDB shell version v4.0.10
connecting to: mongodb://180.76.159.126:27017/?gssapiServiceName=mongodb
Implicit session: session { "id" : UUID("53fef661-35d6-4d29-b07c-020291d62e1a")}
MongoDB server version: 4.0.10
>
```

提示：

开启认证后再登录，发现打印的日志比较少了。

相关操作需要认证才可以：

查询admin库中的system.users集合的用户：

```
> use admin
switched to db admin
> db.system.users.find()
Error: error: {
  "ok" : 0,
  "errmsg" : "command find requires authentication",
  "code" : 13,
  "codeName" : "Unauthorized"
}
> db.auth("myroot","123456")
1
> db.system.users.find()
```

查询articledb库中的comment集合的内容：

```
> use articledb
switched to db articledb
> db.comment.find()
Error: error: {
  "ok" : 0,
  "errmsg" : "not authorized on articledb to execute command { find:
  \"comment\", filter: {}, lsid: { id: UUID(\"53fef661-35d6-4d29-b07c-
  020291d62e1a\") }, $db: \"articledb\" }",
  "code" : 13,
  "codeName" : "Unauthorized"
}
> db.auth("bobo","123456")
1
> db.comment.find()
Error: error: {
  "ok" : 0,
  "errmsg" : "too many users are authenticated",
  "code" : 13,
  "codeName" : "Unauthorized"
}
```

提示：

这里可能出现错误，说是太多的用户正在认证了。因为我们确实连续登录了两个用户了。

解决方案：退出shell，重新进来登录认证。

```
> exit
bye
[root@bobohost bin]# ./mongo --host 180.76.159.126 --port 27017
MongoDB shell version v4.0.10
connecting to: mongodb://180.76.159.126:27017/?gssapiServiceName=mongodb
```



```

Implicit session: session { "id" : UUID("329c1897-566d-4231-bcb3-b2acda301863")
}
MongoDB server version: 4.0.10
> db.auth("bobo","123456")
Error: Authentication failed.
0
> use articledb
switched to db articledb
> db.auth("bobo","123456")
1
> db.comment.find()
>

```

## 2) 连接时直接认证

对admin数据库进行登录认证和相关操作：

```

[root@bobohost ~]# /usr/local/mongodb/bin/mongo --host 180.76.159.126 --port
27017 --authenticationDatabase admin -u myroot -p 123456
MongoDB shell version v4.0.10
connecting to: mongodb://180.76.159.126:27017/?
authSource=admin&gssapiServiceName=mongodb
Implicit session: session { "id" : UUID("f959b8d6-6994-44bc-9d35-09fc7cd00ba6")
}
MongoDB server version: 4.0.10
Server has startup warnings:
2019-09-10T15:23:40.102+0800 I CONTROL [initandlisten] ** WARNING: You are
running this process as the root user, which is not recommended.
2019-09-10T15:23:40.102+0800 I CONTROL [initandlisten]
2019-09-10T15:23:40.102+0800 I CONTROL [initandlisten]
2019-09-10T15:23:40.102+0800 I CONTROL [initandlisten] ** WARNING:
/sys/kernel/mm/transparent_hugepage/enabled is 'always'.
2019-09-10T15:23:40.102+0800 I CONTROL [initandlisten] **          We suggest
setting it to 'never'
2019-09-10T15:23:40.102+0800 I CONTROL [initandlisten]
2019-09-10T15:23:40.102+0800 I CONTROL [initandlisten] ** WARNING:
/sys/kernel/mm/transparent_hugepage/defrag is 'always'.
2019-09-10T15:23:40.102+0800 I CONTROL [initandlisten] **          We suggest
setting it to 'never'
2019-09-10T15:23:40.102+0800 I CONTROL [initandlisten]
> show dbs;
admin      0.000GB
articledb 0.000GB
config    0.000GB
local     0.000GB

```

对articledb数据库进行登录认证和相关操作：

```
[root@bobohost bin]# /usr/local/mongodb/bin/mongo --host 180.76.159.126 --port 27017 --authenticationDatabase articledb -u bobo -p 123456
MongoDB shell version v4.0.10
connecting to: mongodb://180.76.159.126:27017/?
authSource=articledb&gssapiServiceName=mongodb
Implicit session: session { "id" : UUID("e5d4148f-373b-45b8-9cff-a927ce617100")
}
MongoDB server version: 4.0.10
> use articledb
switched to db articledb
> db.comment.find()
```

提示：

- `-u`：用户名
- `-p`：密码
- `--authenticationDatabase`：指定连接到哪个库。当登录是指定用户名密码时，必须指定对应的数据库！

### 3.2.4 SpringDataMongoDB连接认证

使用用户名和密码连接到 MongoDB 服务器，你必须使用

'`username:password@hostname/dbname`' 格式，'username'为用户名，'password' 为密码。

目标：使用用户bobo使用密码 123456 连接到MongoDB 服务上。

application.yml：

```
spring:
  #数据源配置
  data:
    mongodb:
      # 主机地址
      # host: 180.76.159.126
      # 数据库
      # database: articledb
      # 默认端口是27017
      # port: 27017
      #帐号
      # username: bobo
      #密码
      # password: 123456
      #单机有认证的情况下，也使用字符串连接
      uri: mongodb://bobo:123456@180.76.159.126:27017/articledb
```

提示：

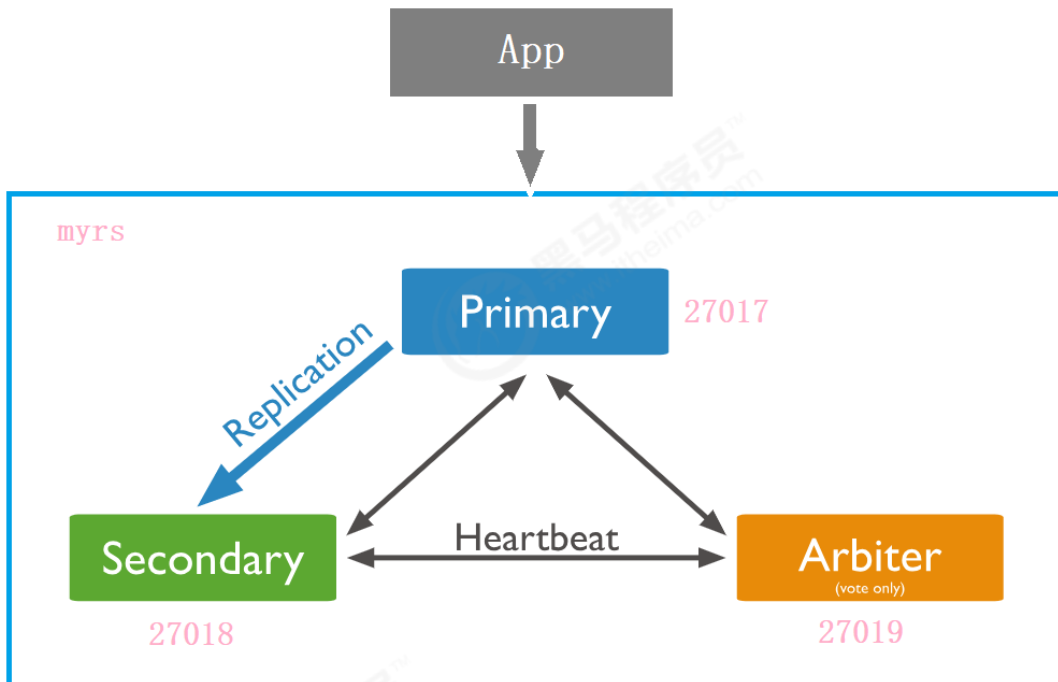
分别测试用户名密码正确以及不正确的情况。

## 3.3 副本集环境

### 3.3.1 前言

对于搭建好的mongodb副本集，为了安全，启动安全认证，使用账号密码登录。

副本集环境使用之前搭建好的，架构如下：



对副本集执行访问控制需要配置两个方面:

1) 副本集和共享集群的各个节点成员之间使用内部身份验证，可以使用密钥文件或x.509证书。密钥文件比较简单，本文使用密钥文件，官方推荐如果是测试环境可以使用密钥文件，但是正式环境，官方推荐x.509证书。原理就是，集群中每一个实例彼此连接的时候都检验彼此使用的证书的内容是否相同。只有证书相同的实例彼此才可以访问

2) 使用客户端连接到mongodb集群时，开启访问授权。对于集群外部的访问。如通过可视化客户端，或者通过代码连接的时候，需要开启授权。

在keyfile身份验证中，副本集中的每个mongod实例都使用keyfile的内容作为共享密码，只有具有正确密钥文件的mongod或者mongos实例可以连接到副本集。密钥文件的内容必须在6到1024个字符之间，并且在unix/linux系统中文件所有者必须有对文件至少有读的权限。

### 3.3.2 关闭已开启的副本集服务（可选）

增加副本集的安全认证和服务鉴权功能，可以在副本集搭建的时候直接添加，也可以在之前搭建好的副本集服务上添加。

本文使用之前搭建好的副本集服务，因此，先停止之前的集群服务

停止服务的方式有两种：快速关闭和标准关闭，下面依次说明：

(1) 快速关闭方法（快速，简单，数据可能会出错）

目标：通过系统的kill命令直接杀死进程：

依次杀死仲裁者、副本节点、主节点，直到所有成员都离线。建议主节点最后kill，以避免潜在的回滚。杀完要检查一下，避免有的没有杀掉。

```
#通过进程编号关闭节点  
kill -2 54410
```

### 【补充】

如果一旦是因为数据损坏，则需要进行如下操作（了解）：

1) 删除lock文件：

```
rm -f /mongodb/replica_sets/myrs_27017/data/db/*.lock \  
/mongodb/replica_sets/myrs_27018/data/db/*.lock \  
/mongodb/replica_sets/myrs_27019/data/db/mongod.lock \  

```

2) 依次修复数据：

```
/usr/local/mongodb/bin/mongod --repair --  
dbpath=/mongodb/replica_sets/myrs_27017/data/db  
/usr/local/mongodb/bin/mongod --repair --  
dbpath=/mongodb/replica_sets/myrs_27018/data/db  
/usr/local/mongodb/bin/mongod --repair --  
dbpath=/mongodb/replica_sets/myrs_27019/data/db
```

(2) 标准的关闭方法（数据不容易出错，但麻烦）：

目标：通过mongo客户端中的shutdownServer命令来依次关闭各个服务

关闭副本集中的服务，建议依次关闭仲裁节点、副本节点、主节点。主要的操作步骤参考如下：

```
//客户端登录服务，注意，这里通过localhost登录，如果需要远程登录，必须先登录认证才行。  
mongo --port 27017  
//告知副本集说本机要下线  
rs.stepDown()  
//切换到admin库  
use admin  
//关闭服务  
db.shutdownServer()
```

### 3.3.3 通过主节点添加一个管理员帐号

只需要在主节点上添加用户，副本集会自动同步。

开启认证之前，创建超管用户：myroot，密码：123456

```
myrs:PRIMARY> use admin  
switched to db admin  
myrs:PRIMARY> db.createUser({user:"myroot",pwd:"123456",roles:["root"]})  
Successfully added user: { "user" : "myroot", "roles" : [ "root" ] }
```

详细操作详见单实例环境的 [添加用户和权限](#) 的相关操作。

提示：

该步骤也可以在开启认证之后，但需要通过localhost登录才允许添加用户，用户数据也会自动同步到副本集。

后续再创建其他用户，都可以使用该超管用户创建。

### 3.3.4 创建副本集认证的key文件

第一步：生成一个key文件到当前文件夹中。

可以使用任何方法生成密钥文件。例如，以下操作使用openssl生成密码文件，然后使用chmod来更改文件权限，仅为文件所有者提供读取权限

```
[root@bobohost ~]# openssl rand -base64 90 -out ./mongo.keyfile
[root@bobohost ~]# chmod 400 ./mongo.keyfile
[root@bobohost ~]# ll mongo.keyfile
-r----- . 1 root root 122 8月 14 14:23 mongo.keyfile
```

提示：

所有副本集节点都必须要用同一份keyfile，一般是在一台机器上生成，然后拷贝到其他机器上，且必须有读的权限，否则将来会报错：permissions on

```
/mongodb/replica_sets/myrs_27017/mongo.keyfile are too open
```

一定要保证密钥文件一致，文件位置随便。但是为了方便查找，建议每台机器都放到一个固定的位置，都放到和配置文件一起的目录中。

这里将该文件分别拷贝到多个目录中：

```
[root@bobohost ~]# cp mongo.keyfile /mongodb/replica_sets/myrs_27017
[root@bobohost ~]# cp mongo.keyfile /mongodb/replica_sets/myrs_27018
[root@bobohost ~]# cp mongo.keyfile /mongodb/replica_sets/myrs_27019
```

### 3.3.5 修改配置文件指定keyfile

分别编辑几个服务的mongod.conf文件，添加相关内容：

```
/mongodb/replica_sets/myrs_27017/mongod.conf
```

```
security:
  #KeyFile鉴权文件
  keyFile: /mongodb/replica_sets/myrs_27017/mongo.keyfile
  #开启认证方式运行
  authorization: enabled
```

```
/mongodb/replica_sets/myrs_27018/mongod.conf
```

```
security:
  #keyFile鉴权文件
  keyFile: /mongodb/replica_sets/myrs_27018/mongo.keyfile
  #开启认证方式运行
  authorization: enabled
```

/mongodb/replica\_sets/myrs\_27019/mongod.conf

```
security:
  #keyFile鉴权文件
  keyFile: /mongodb/replica_sets/myrs_27019/mongo.keyfile
  #开启认证方式运行
  authorization: enabled
```

### 3.3.6 重新启动副本集

如果副本集是开启状态，则先分别关闭副本集中的每个mongod，从次节点开始。直到副本集的所有成员都离线，包括任何仲裁者。主节点必须是最后一个成员关闭以避免潜在的回滚。

```
#通过进程编号关闭三个节点
kill -2 54410 54361 54257
```

分别启动副本集节点：

```
/usr/local/mongodb/bin/mongod -f /mongodb/replica_sets/myrs_27017/mongod.conf
/usr/local/mongodb/bin/mongod -f /mongodb/replica_sets/myrs_27018/mongod.conf
/usr/local/mongodb/bin/mongod -f /mongodb/replica_sets/myrs_27019/mongod.conf
```

查看进程情况：

```
[root@bobohost replica_sets]# ps -ef |grep mongod
root      62425      1  5 14:43 ?          00:00:01 /usr/local/mongodb/bin/mongod
-f /mongodb/replica_sets/myrs_27017/mongod.conf
root      62495      1  7 14:43 ?          00:00:01 /usr/local/mongodb/bin/mongod
-f /mongodb/replica_sets/myrs_27018/mongod.conf
root      62567      1 11 14:43 ?          00:00:01 /usr/local/mongodb/bin/mongod
-f /mongodb/replica_sets/myrs_27019/mongod.conf
```

### 3.3.7 在主节点上添加普通账号

```
#先用管理员账号登录
#切换到admin库
use admin
#管理员账号认证
db.auth("myroot", "123456")
#切换到要认证的库
use articledb
#添加普通用户
db.createUser({user: "bobo", pwd: "123456", roles: ["readWrite"]})
```

重新连接，使用普通用户bobo重新登录，查看数据。

注意：也要使用rs.status()命令查看副本集是否健康。

### 3.3.8 SpringDataMongoDB连接副本集

使用用户名和密码连接到 MongoDB 服务器，你必须使用 'username:password@hostname/dbname' 格式，'username'为用户名，'password' 为密码。

目标：使用用户bobo使用密码 123456 连接到MongoDB 服务上。

application.yml :

```
spring:
  #数据源配置
  data:
    mongodb:
      #副本集有认证的情况下，字符串连接
      uri:
        mongodb://bobo:123456@180.76.159.126:27017,180.76.159.126:27018,180.76.159.126:27019/articledb?connect=replicaSet&slaveOk=true&replicaSet=myrs
```

## 3.4 分片集群环境(扩展)

### 3.3.1 关闭已开启的集群服务（可选）

分片集群环境下的安全认证和副本集环境下基本上一样。

但分片集群的服务器环境和架构较为复杂，建议在搭建分片集群的时候，直接加入安全认证和服务器间的鉴权，如果之前有数据，可先将之前的数据备份出来，再还原回去。

本文使用之前搭建好的集群服务，因此，先停止之前的集群服务

停止服务的方式有两种：快速关闭和标准关闭，下面依次说明：

(1) 快速关闭方法（快速，简单，数据可能会出错）

目标：通过系统的kill命令直接杀死进程：

依次杀死mongos路由、配置副本集服务，分片副本集服务，从次节点开始。直到所有成员都离线。副本集杀的时候，建议先杀仲裁者，再杀副本节点，最后是主节点，以避免潜在的回滚。杀完要检查一下，避免有的没有杀掉。

```
#通过进程编号关闭节点  
kill -2 54410
```

### 【补充】

如果一旦是因为数据损坏，则需要进行如下操作（了解）：

1) 删除lock文件：

```
rm -f /mongodb/sharded_cluster/myshardrs01_27018/data/db/*.lock \  
/mongodb/sharded_cluster/myshardrs01_27118/data/db/*.lock \  
/mongodb/sharded_cluster/myshardrs01_27218/data/db/mongod.lock \  
/mongodb/sharded_cluster/myshardrs02_27318/data/db/mongod.lock \  
/mongodb/sharded_cluster/myshardrs02_27418/data/db/mongod.lock \  
/mongodb/sharded_cluster/myshardrs02_27518/data/db/mongod.lock \  
/mongodb/sharded_cluster/myconfigs_27019/data/db/mongod.lock \  
/mongodb/sharded_cluster/myconfigs_27119/data/db/mongod.lock \  
/mongodb/sharded_cluster/myconfigs_27219/data/db/mongod.lock
```

2) 依次修复数据：

```
/usr/local/mongodb/bin/mongod --repair --  
dbpath=/mongodb/sharded_cluster/myshardrs01_27018/data/db  
/usr/local/mongodb/bin/mongod --repair --  
dbpath=/mongodb/sharded_cluster/myshardrs01_27118/data/db  
/usr/local/mongodb/bin/mongod --repair --  
dbpath=/mongodb/sharded_cluster/myshardrs01_27218/data/db  
/usr/local/mongodb/bin/mongod --repair --  
dbpath=/mongodb/sharded_cluster/myshardrs02_27318/data/db  
/usr/local/mongodb/bin/mongod --repair --  
dbpath=/mongodb/sharded_cluster/myshardrs02_27418/data/db  
/usr/local/mongodb/bin/mongod --repair --  
dbpath=/mongodb/sharded_cluster/myshardrs02_27518/data/db  
/usr/local/mongodb/bin/mongod --repair --  
dbpath=/mongodb/sharded_cluster/myconfigs_27019/data/db  
/usr/local/mongodb/bin/mongod --repair --  
dbpath=/mongodb/sharded_cluster/myconfigs_27119/data/db  
/usr/local/mongodb/bin/mongod --repair --  
dbpath=/mongodb/sharded_cluster/myconfigs_27219/data/db  
/usr/local/mongodb/bin/mongod --repair --  
dbpath=/mongodb/sharded_cluster/mymongos_27017/data/db  
/usr/local/mongodb/bin/mongod --repair --  
dbpath=/mongodb/sharded_cluster/mymongos_27117/data/db
```

(2) 标准的关闭方法（数据不容易出错，但麻烦）：

目标：通过mongo客户端中的shutdownServer命令来依次关闭各个服务



关闭分片服务器副本集中的服务，建议依次关闭仲裁节点、副本节点、主节点。主要的操作步骤参考如下：

```
//客户端登录服务，注意，这里通过localhost登录，如果需要远程登录，必须先登录认证才行。
mongo --port 27018
//告知副本集说本机要下线
rs.stepDown()
//切换到admin库
use admin
//关闭服务
db.shutdownServer()
```

关闭配置服务器副本集的服务，建议依次关闭副本节点、主节点。主要的操作步骤参考如下：

```
//客户端登录服务，注意，这里通过localhost登录，如果需要远程登录，必须先登录认证才行。
mongo --port 27019
//告知副本集说本机要下线
rs.stepDown()
//切换到admin库
use admin
//关闭服务
db.shutdownServer()
```

关闭路由服务器的服务，建议依次关闭两个路由节点。主要的操作步骤参考如下：

```
//客户端登录服务，注意，这里通过localhost登录，如果需要远程登录，必须先登录认证才行。
mongo --port 27017
//告知副本集说本机要下线
rs.stepDown()
//切换到admin库
use admin
//关闭服务
db.shutdownServer()
```

### 3.3.2 创建副本集认证的key文件

第一步：生成一个key文件到当前文件夹中。

可以使用任何方法生成密钥文件。例如，以下操作使用openssl生成密码文件，然后使用chmod来更改文件权限，仅为文件所有者提供读取权限

```
[root@bobohost ~]# openssl rand -base64 90 -out ./mongo.keyfile
[root@bobohost ~]# chmod 400 ./mongo.keyfile
[root@bobohost ~]# ll mongo.keyfile
-r----- . 1 root root 122 8月 14 14:23 mongo.keyfile
```

提示：

所有副本集节点都必须要用同一份keyfile，一般是在一台机器上生成，然后拷贝到其他机器上，且必须有读的权限，否则将来会报错：permissions on

```
/mongodb/replica_sets/myrs_27017/mongo.keyfile are too open
```

一定要保证密钥文件一致，文件位置随便。但是为了方便查找，建议每台机器都放到一个固定的位置，都放到和配置文件一起的目录中。

这里将该文件分别拷贝到多个目录中：

```
echo '/mongodb/sharded_cluster/myshardrs01_27018/mongo.keyfile
/mongodb/sharded_cluster/myshardrs01_27118/mongo.keyfile
/mongodb/sharded_cluster/myshardrs01_27218/mongo.keyfile
/mongodb/sharded_cluster/myshardrs02_27318/mongo.keyfile
/mongodb/sharded_cluster/myshardrs02_27418/mongo.keyfile
/mongodb/sharded_cluster/myshardrs02_27518/mongo.keyfile
/mongodb/sharded_cluster/myconfigs_27019/mongo.keyfile
/mongodb/sharded_cluster/myconfigs_27119/mongo.keyfile
/mongodb/sharded_cluster/myconfigs_27219/mongo.keyfile
/mongodb/sharded_cluster/mymongos_27017/mongo.keyfile
/mongodb/sharded_cluster/mymongos_27117/mongo.keyfile' | xargs -n 1 cp -v
/root/mongo.keyfile
```

### 3.3.3 修改配置文件指定keyfile

分别编辑几个服务的mongod.conf文件，添加相关内容：

/mongodb/sharded\_cluster/myshardrs01\_27018/mongod.conf

```
security:
  #KeyFile鉴权文件
  keyFile: /mongodb/sharded_cluster/myshardrs01_27018/mongo.keyfile
  #开启认证方式运行
  authorization: enabled
```

/mongodb/sharded\_cluster/myshardrs01\_27118/mongod.conf

```
security:
  #KeyFile鉴权文件
  keyFile: /mongodb/sharded_cluster/myshardrs01_27118/mongo.keyfile
  #开启认证方式运行
  authorization: enabled
```

/mongodb/sharded\_cluster/myshardrs01\_27218/mongod.conf

```
security:
  #KeyFile鉴权文件
  keyFile: /mongodb/sharded_cluster/myshardrs01_27218/mongo.keyfile
  #开启认证方式运行
  authorization: enabled
```

/mongodb/sharded\_cluster/myshardrs02\_27318/mongod.conf

```
security:  
#KeyFile鉴权文件  
keyFile: /mongodb/sharded_cluster/myshardrs02_27318/mongo.keyfile  
#开启认证方式运行  
authorization: enabled
```

/mongodb/sharded\_cluster/myshardrs02\_27418/mongod.conf

```
security:  
#KeyFile鉴权文件  
keyFile: /mongodb/sharded_cluster/myshardrs02_27418/mongo.keyfile  
#开启认证方式运行  
authorization: enabled
```

/mongodb/sharded\_cluster/myshardrs02\_27518/mongod.conf

```
security:  
#KeyFile鉴权文件  
keyFile: /mongodb/sharded_cluster/myshardrs02_27518/mongo.keyfile  
#开启认证方式运行  
authorization: enabled
```

/mongodb/sharded\_cluster/myconfigs\_27019/mongod.conf

```
security:  
#KeyFile鉴权文件  
keyFile: /mongodb/sharded_cluster/myconfigs_27019/mongo.keyfile  
#开启认证方式运行  
authorization: enabled
```

/mongodb/sharded\_cluster/myconfigs\_27119/mongod.conf

```
security:  
#KeyFile鉴权文件  
keyFile: /mongodb/sharded_cluster/myconfigs_27119/mongo.keyfile  
#开启认证方式运行  
authorization: enabled
```

/mongodb/sharded\_cluster/myconfigs\_27219/mongod.conf

```
security:
  #keyFile鉴权文件
  keyFile: /mongodb/sharded_cluster/myconfigs_27219/mongo.keyfile
  #开启认证方式运行
  authorization: enabled
```

/mongodb/sharded\_cluster/mymongos\_27017/mongos.conf

```
security:
  #keyFile鉴权文件
  keyFile: /mongodb/sharded_cluster/mymongos_27017/mongo.keyfile
```

/mongodb/sharded\_cluster/mymongos\_27117/mongos.conf

```
security:
  #keyFile鉴权文件
  keyFile: /mongodb/sharded_cluster/mymongos_27117/mongo.keyfile
```

mongos比mongod少了authorization : enabled的配置。原因是，副本集加分片的安全认证需要配置两方面的，副本集各个节点之间使用内部身份验证，用于内部各个mongo实例的通信，只有相同keyfile才能相互访问。所以都要开启 keyFile:

```
/mongodb/sharded_cluster/mymongos_27117/mongo.keyfile。
```

然而对于所有的mongod，才是真正的保存数据的分片。mongos只做路由，不保存数据。所以所有的mongod开启访问数据的授权authorization:enabled。这样用户只有账号密码正确才能访问到数据。

### 3.3.4 重新启动节点

必须依次启动配置节点、分片节点、路由节点：

```
/usr/local/mongodb/bin/mongod -f
/mongodb/sharded_cluster/myconfigs_27019/mongod.conf
/usr/local/mongodb/bin/mongod -f
/mongodb/sharded_cluster/myconfigs_27119/mongod.conf
/usr/local/mongodb/bin/mongod -f
/mongodb/sharded_cluster/myconfigs_27219/mongod.conf

/usr/local/mongodb/bin/mongod -f
/mongodb/sharded_cluster/myshardrs01_27018/mongod.conf
/usr/local/mongodb/bin/mongod -f
/mongodb/sharded_cluster/myshardrs01_27118/mongod.conf
/usr/local/mongodb/bin/mongod -f
/mongodb/sharded_cluster/myshardrs01_27218/mongod.conf

/usr/local/mongodb/bin/mongod -f
/mongodb/sharded_cluster/myshardrs02_27318/mongod.conf
/usr/local/mongodb/bin/mongod -f
/mongodb/sharded_cluster/myshardrs02_27418/mongod.conf
```

```
/usr/local/mongodb/bin/mongod -f
/mongodb/sharded_cluster/myshardrs02_27518/mongod.conf
```

```
/usr/local/mongodb/bin/mongos -f
/mongodb/sharded_cluster/mymongos_27017/mongos.conf
/usr/local/mongodb/bin/mongos -f
/mongodb/sharded_cluster/mymongos_27117/mongos.conf
```

注意：

这里有个非常特别的情况，就是启动顺序。先启动配置节点，再启动分片节点，最后启动路由节点。

如果先启动分片节点，会卡住，提示：

```
about to fork child process, waiting until server is ready for connections
```

这也许是个bug。原因未知。

### 3.3.5 创建帐号和认证

客户端mongo，通过localhost登录任意一个mongos路由，

```
[root@bobohost db]# /usr/local/mongodb/bin/mongo --port 27017
```

提示：相当于一个后门，只能在admin下添加用户。

创建一个管理员帐号：

```
mongos> use admin
switched to db admin
mongos> db.createUser({user:"myroot",pwd:"123456",roles:["root"]})
Successfully added user: { "user" : "myroot", "roles" : [ "root" ] }
```

提示：如果在开启认证之前已经创建了管理员账号，这里可以忽略

创建一个普通权限帐号：

```
mongos> use admin
switched to db admin
mongos> db.auth("myroot","123456")
1
mongos> use articledb
switched to db articledb
mongos> db.createUser({user: "bobo", pwd: "123456", roles: [{ role: "readwrite",
db: "articledb" }]])
Successfully added user: {
  "user" : "bobo",
  "roles" : [
    {
      "role" : "readwrite",
      "db" : "articledb"
    }
  ]
}
```

```
]
}
mongos> db.auth("bobo","123456")
1
```

提示：

通过mongos添加的账号信息，只会保存到配置节点的服务中，具体的数据节点不保存账号信息，因此，分片中的账号信息不涉及到同步问题。

mongo客户端登录mongos路由，用管理员帐号登录可查看分片情况：

```
mongos> use admin
switched to db admin
mongos> db.auth("myroot","123456")
1
mongos> sh.status()
```

退出连接，重新连接服务，使用普通权限帐号访问数据：

```
[root@bobohost db]# /usr/local/mongodb/bin/mongo --host 180.76.159.126 --port
27017
MongoDB shell version v4.0.10
connecting to: mongodb://180.76.159.126:27017/?gssapiServiceName=mongodb
Implicit session: session { "id" : UUID("6f84fa91-2414-407e-b3ab-c0b7eedde825")
}
MongoDB server version: 4.0.10
mongos> use articledb
switched to db articledb
mongos> db.auth("bobo","123456")
1
mongos> show collections
comment
comment2
mongos> db.comment.count()
10001
```

### 3.3.6 SpringDataMongoDB连接认证

使用用户名和密码连接到 MongoDB 服务器，你必须使用 'username:password@hostname/dbname' 格式，'username'为用户名，'password' 为密码。

目标：使用用户bobo使用密码 123456 连接到MongoDB 服务上。

application.yml：

```
spring:
  #数据源配置
  data:
    mongodb:
      # 分片集群有认证的情况下，字符串连接
      uri:
        mongodb://bobo:123456@180.76.159.126:27017,180.76.159.126:27117/articledb
```

