

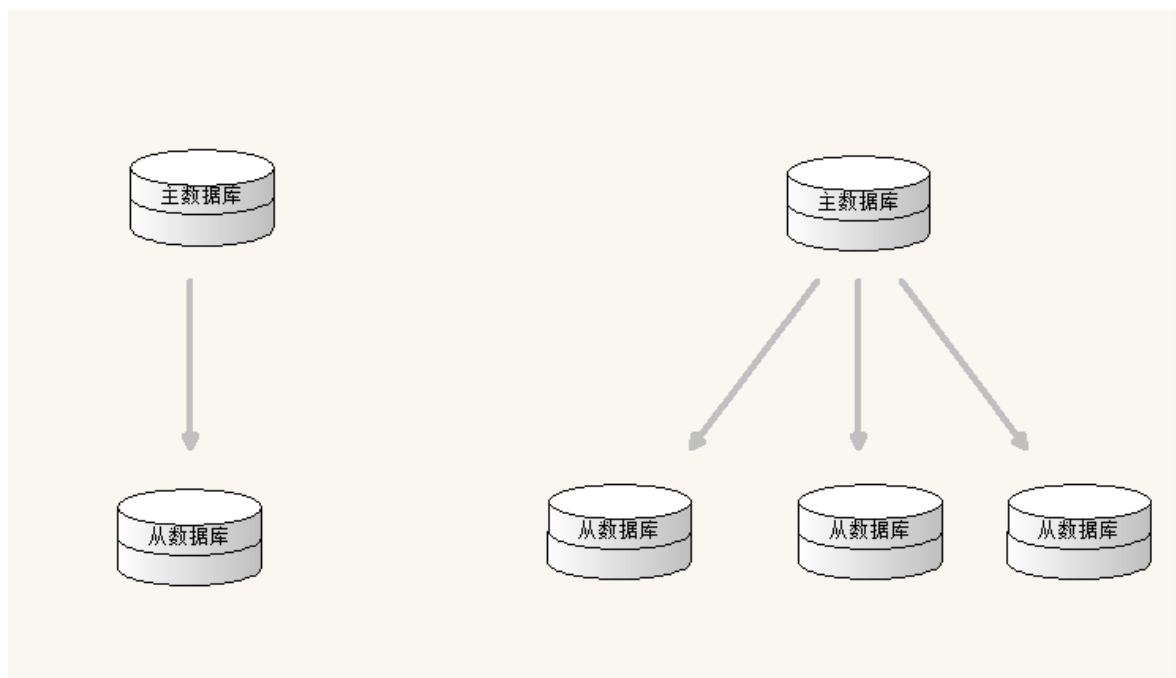
Mongo高级

--- 老孙

7、搭建高可用集群

7.1 MongoDB主从复制架构原理和缺陷

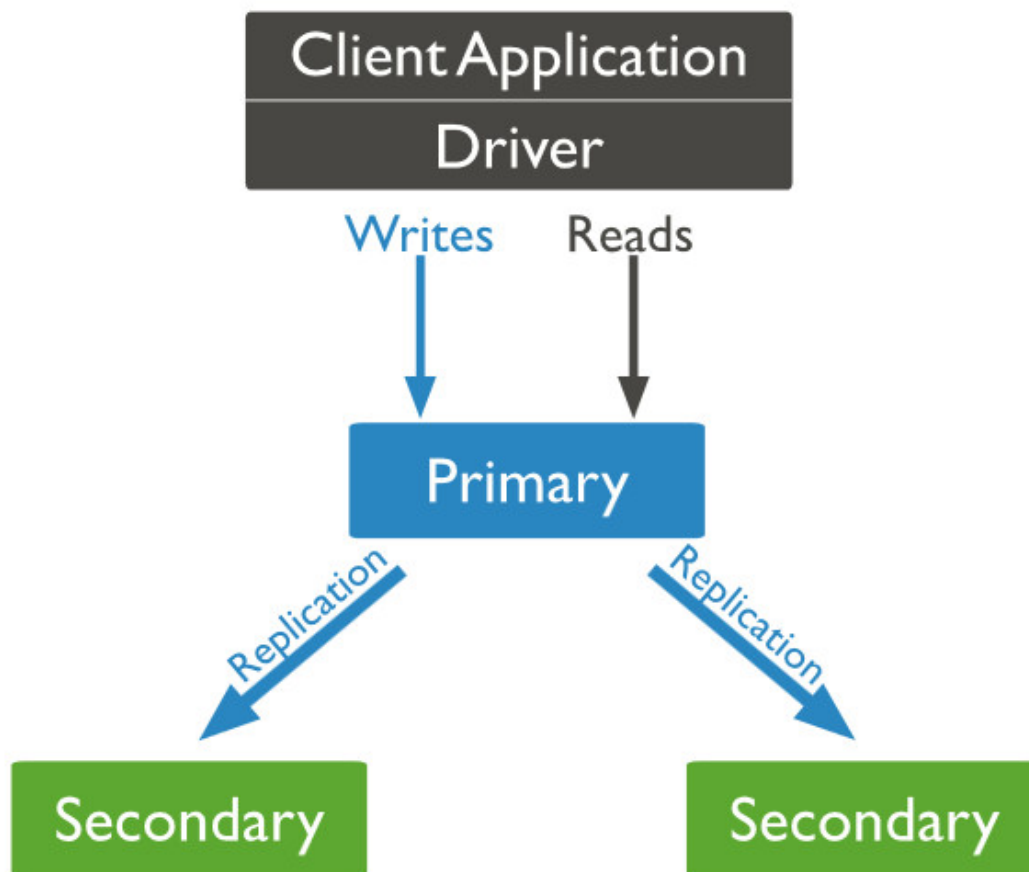
- 在主从结构中，主节点的操作记录成为oplog (operation log)。
- oplog存储在一个系统数据库local的集合oplog.\$main中，这个集合的每个文档都代表主节点上执行的一个操作。
- 从服务器会定期从主服务器中获取oplog记录，然后在本机上执行！
- 对于存储oplog的集合，MongoDB采用的是固定集合，也就是说随着操作过多，新的操作会覆盖旧的操作！
- mongodb支持传统的master-slave架构。master节点负责数据的读写，slave没有写入权限。



- 没有自动故障转移功能，需要指定master和slave端，不推荐在生产中使用
- mongodb4.0后不再支持主从复制！

[main] Master/slave replication is no longer supported

7.2 复制集 replica sets



7.2.1 什么是复制集（副本集）

- MongoDB 的复制集不同于以往的主从模式。
 - 在集群Master故障的时候，副本集可以自动投票，选举出新的Master，并引导其余的Slave服务器连接新的Master，而这个过程对于应用是透明的。
 - 可以说MongoDB的复制集是**自带故障转移**功能的主从复制。
 - 相对于传统主从模式的优势传统的主从模式，需要手工指定集群中的 Master。如果 Master 发生故障，一般都是人工介入，指定新的 Master。这个过程对于应用一般不是透明的，往往伴随着应用重新修改配置文件，重启应用服务器等。
 - 而 MongoDB 副本集，集群中的任何节点都可能成为 Master 节点。

7.2.2 为什么要使用复制集

1. 高可用

防止设备（服务器、网络）故障。

提供自动 failover（故障转移）功能。

技术来保证高可用

2. 灾难恢复

当发生故障时，可以从其他节点恢复用于备份。

3. 功能隔离

我们可以在备节点上执行，减少主节点的压力

比如:用于分析、报表，数据挖掘，系统任务等。

7.2.3 复制集集群架构原理

- 一个副本集即为服务于同一数据集的多个 MongoDB 实例，其中一个为主节点，其余的都为从节点。
- 主节点上能够完成读写操作，从节点仅能用于读操作。
- 主节点需要记录所有改变数据库状态的操作，这些记录保存在 oplog 中，这个文件存储在 local 数据库，各个从节点通过此 oplog 来复制数据并应用于本地，保持本地的数据与主节点的一致。
- oplog 具有幂等性，即无论执行几次其结果一致，这个比 mysql 的二进制日志更好用。
- 集群中的各节点还会通过传递心跳信息来检测各自的健康状况。
- 当主节点故障时，多个从节点会触发一次新的选举操作，并选举其中的一个成为新的主节点(通常谁的优先级更高，谁就是新的主节点)，心跳信息默认每 2 秒传递一次。

心跳检测：

整个集群需要保持一定的通信才能知道哪些节点活着哪些节点挂掉。

mongodb节点会向副本集中的其他节点每两秒就会发送一次**pings**包，如果其他节点在**10**秒钟之内没有返回就标示为不能访问。

每个节点内部都会维护一个状态映射表，表明当前每个节点是什么角色、日志时间戳等关键信息。

如果是主节点，除了维护映射表外还需要检查自己能否和集群中内大部分节点通讯，如果不能，则把自己降级为**secondary**只读节点。

数据同步副本集同步分为初始化同步和**keep**复制。

初始化同步指全量从主节点同步数据，如果主节点数据量比较大同步时间会比较长。

而**keep**复制指初始化同步过后，节点之间的实时同步一般是增量同步。

初始化同步不只是在第一次才会被触发，有以下两种情况会触发：

- 1) **secondary**第一次加入，这个是肯定的。
- 2) **secondary**落后的数据量超过了**oplog**的大小，这样也会被全量复制。

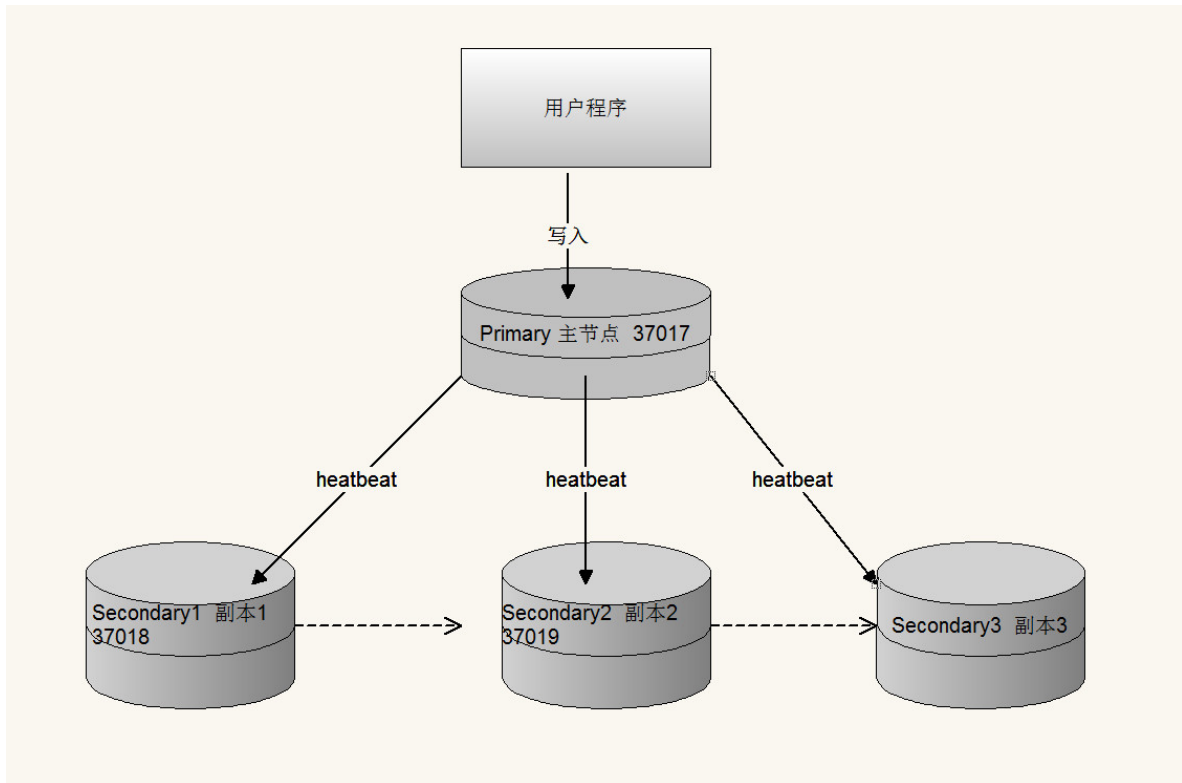
- 副本集中的副本节点在主节点挂掉后通过心跳机制检测到后，就会在集群内发起主节点的选举机制，自动选举出一位新的主服务器，从而保证系统可以正常运行

1) 主节点负责处理客户端请求，读、写数据，记录在其上所有操作的**oplog**；

2) 从节点定期轮询主节点获取这些操作，然后对自己的数据副本执行这些操作，从而保证从节点的数据与主节点一致。默认情况下，从节点不支持外部读取，但可以设置；

3) 仲裁节点不复制数据，仅参与投票。由于它没有访问的压力，比较空闲，因此不容易出故障。由于副本集出现故障的时候，存活的节点必须大于副本集节点总数的一半，否则无法选举主节点，或者主节点会自动降级为从节点，整个副本集变为只读。因此，增加一个不容易出故障的仲裁节点，可以增加有效选票，降低整个副本集不可用的风险。仲裁节点可多于一个。也就是说只参与投票，不接收复制的数据，也不能成为活跃节点。

- 一个复制集里至少包含一个主节点。
- 如下图中，三个节点，写操作只能在主节点操作。
 - 主节点收到数据操作时，会将操作记录在操作日志里。
 - 所有的从节点都会在数据同步的过程中，从主节点读取操作日志，从而保证从节点和主节点的数据统一。



7.2.4 无仲裁节点 复制集搭建

伪集群：在一台服务器下，创建N个mongo实例形成的集群

opt目录下创建replica_set目录

```
[root@A ~]# mkdir /opt/replica_set
```

mongo压缩包copy进去，然后解压

```
[root@A opt]# cp mongodb-linux-x86_64-4.1.3.tgz
/opt/replica_set/
[root@A opt]# cd replica_set/
[root@A replica_set]# tar -zxvf mongodb-linux-
x86_64-4.1.3.tgz
[root@A replica_set]# cd mongodb-linux-x86_64-4.1.3
[root@A mongodb-linux-x86_64-4.1.3]# vim
mongo_37017.conf
```

进入mongo目录，创建下面3个配置文件，并修改其内容如下：

1. 主节点配置

```
# 主节点配置 mongo_37017.conf
dbpath=/data/mongo/data/server1
bind_ip=0.0.0.0
port=37017
fork=true
logpath=/data/mongo/logs/server1.log
replSet=lagouCluster
```

2. 从节点1配置

```
# mongo_37018.conf
dbpath=/data/mongo/data/server2
bind_ip=0.0.0.0
port=37018
fork=true
logpath=/data/mongo/logs/server2.log
replSet=lagouCluster
```

3. 从节点2配置

```
# mongo_37019.conf
dbpath=/data/mongo/data/server3
bind_ip=0.0.0.0
port=37019
fork=true
logpath=/data/mongo/logs/server3.log
replSet=lagouCluster
```

- 保证配置中目录的存在

```
mkdir -p /data/mongo/data/server1
mkdir -p /data/mongo/data/server2
mkdir -p /data/mongo/data/server3
mkdir -p /data/mongo/logs
```

4. 初始化节点配置

启动三个节点

```
[root@A mongodb-linux-x86_64-4.1.3]# ./bin/mongod -f
mongo_37017.conf
[root@A mongodb-linux-x86_64-4.1.3]# ./bin/mongod -f
mongo_37018.conf
[root@A mongodb-linux-x86_64-4.1.3]# ./bin/mongod -f
mongo_37019.conf

[root@A mongodb-linux-x86_64-4.1.3]# ps -ef | grep
mongo
```

然后进入任意一个节点（我们进入37017），运行如下命令：

```
[root@A mongodb-linux-x86_64-4.1.3]# ./bin/mongo --
port 37017
```

先做37017和37018两台机器的集群，一主一从


```
var cfg ={
  "_id":"lagouCluster",
  "protocolVersion" : 1,
  "members":[

{"_id":1,"host":"192.168.204.141:37017","priority":10},
    {"_id":2,"host":"192.168.204.141:37018"}
  ]
}
```

rs.initiate(cfg)// 执行完成显示

lagouCluster:SECONDARY> 表示当前节点是从节点，稍等一会，回车刷新，37017会变成主节点lagouCluster:PRIMARY>

rs.status()

- _id：集群标识
- protocolVersion：协议版本
- priority：优先级，整数1~1000，越大，优先级越高，默认为1，设置为0则永不会选举成主
 - 如果都没有设置优先级，则默认为最先启动的节点为主节点（根据启动的时间戳）

5. 节点的动态增删

- 在主节点下进行操作

// 增加节点

```
rs.add("192.168.204.141:37019")
```

// 删除slave 节点

```
rs.remove("192.168.204.141:37019")
```

6. 复制集操作演示

- 进入主节点 ----- 插入数据 ----- 进入从节点验证
- 注意：默认节点下从节点不能读取数据。调用slaveOk解决

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

7. 复制集的选举机制

- 为了保证高可用，在集群当中如果主节点挂掉后，会自动在从节点中选举一个重新做为主节点。
 - kill掉37017，37018会自动上位成主节点
 - 当37017归来，37018会自动让位给37017，因为37017的优先级高
- 在MongoDB 中通过在集群配置中的 rs，属性值大小来决定选举谁做为主节点

7.2.5 有仲裁节点 复制集搭建

- 设置arbiterOnly 为 true 表示做为裁判节点用于执行选举操作，该配置下的节点永远不会被选举为主节点和从节点。

举例：加入37020，并全部启动

```
var cfg = {
  "_id": "lagouCluster",
  "protocolVersion" : 1,
  "members": [
```

```
{ "_id":1,"host":"192.168.204.141:37017","priority":10},  
  
{ "_id":2,"host":"192.168.204.141:37018","priority":0},  
  
{ "_id":3,"host":"192.168.204.141:37019","priority":5},  
  
{ "_id":4,"host":"192.168.204.141:37020","arbiterOnly":true}  
    ]  
};
```

// 重新装载配置，并重新生成集群节点。

```
rs.reconfig(cfg)
```

//重新查看集群状态

```
rs.status()
```

- 节点说明:

- **PRIMARY** 节点：（主节点）可以查询和新增数据
- **SECONDARY** 节点：（从节点）只能查询不能新增，基于 priority 权重可以被选为主节点
- **ARBITER** 节点：（裁判/仲裁节点）不能查询数据和新增数据，不能变成主节点

- 和上面的配置步骤相同，另一种方式增加特殊的仲裁节点
- 注入节点执行 `rs.addArb("IP:端口");`
- 删除节点和之前一样

```
rs.addArb("192.168.204.141:37020")  
rs.remove("192.168.204.141:37020")
```

7.3 分片集群 Sharded Cluster

7.3.1 什么是分片

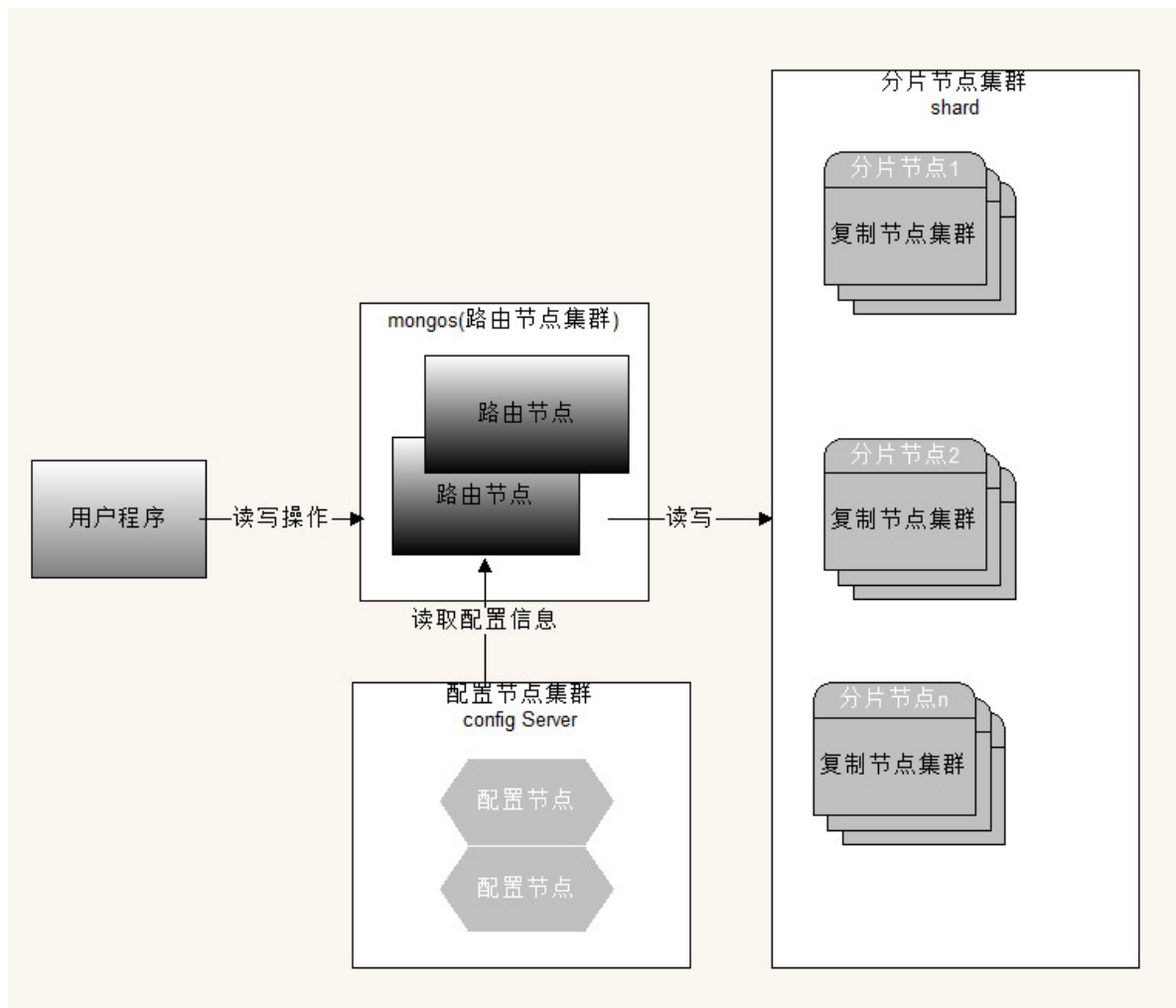
- 分片，是指将数据拆分，将其分散到不同的机器上。
- 这样的好处就是，不需要功能强大的大型计算机也可以存储更多的数据，处理更大的负载。
- mongoDB 的分片，是将collection 的数据进行分割，然后将不同的部分分别存储到不同的机器上。
- 当 collection 所占空间过大时，我们需要增加一台新的机器，分片会自动将 collection 的数据分发到新的机器上。

7.3.2 为什么要分片

- 存储容量需求超出单机**磁盘**容量
- 活跃的数据集超出单机**内存**容量，导致很多请求都要从磁盘读取数据，影响性能
- 写IOPS（每秒处理IO量）超出单个MongoDB节点的写服务能力随着数据的增长，**单机实例的瓶颈是很明显的**。可以通过复制的机制应对压力
- 但MongoDB中单个集群的节点数量限制到了50个（之前是12个）以内，所以需要通过分片进一步横向扩展。
- 此外分片也可节约磁盘的存储。
- 分片技术，使得集合中的数据分散到多个分片集中。使得MongoDB具备横向的发展。

7.3.3 分片的工作原理

- 分片集群由以下3个服务组成：
 - Shards Server：**分片服务**，每个shard由一个或多个 mongod进程（复制集集群）组成，用于存储数据
 - Config Server：**配置服务**，用于存储集群的Metadata信息，包括每个Shard的信息和chunks信息
 - Route Server：**路由服务**，由Client连接，使整个Cluster看起来像单个DB服务器。



- 要构建一个MongoDB Sharding Cluster（分片集群），需要三种角色：
 - mongos**（路由进程，应用程序接入 mongos 再查询到具体分片）
 - 数据库集群请求的入口，所有的请求都通过 mongos 进行协调，不需要在应用程序添加一个路由选择器，mongos 自己

就是一个请求分发中心，它负责把对应的数据转发到对应的 shard 服务器上。

- 在生产环境通常有多个 mongos 作为请求的入口，防止其中一个挂掉所有的 mongodb 请求都没有办法操作。

2、config server (路由表服务，每一台都具有全部 chunk 的路由信息)

- 顾名思义为配置服务器，存储所有数据库元信息(路由、分片)的配置。
- mongos 本身没有物理存储分片服务器和数据路由信息，只是缓存在内存里，配置服务器则实际存储这些数据。
- mongos 第一次启动或者关掉重启就会从 config server 加载配置信息，以后如果配置服务器信息发生变化，就会通知到所有的 mongos 更新自己的状态，这样 mongos 就能继续准确路由。
- 在生产环境通常有多个 config server 配置服务器，因为它存储了分片路由的元数据，这个可不能丢失！就算挂掉其中一台，只要还有存货，mongodb 集群就不会挂掉。

3、shard (为数据存储分片。每一片都可以是复制集(replica set))

- 这就是传说中的分片了。一台机器的一个数据表 Collection1 存储了 1T 数据，压力太大了！再分给 4 个机器后，每个机器都是 256G，则分摊了集中在一台机器的压力。
- 事实上，上图4个分片如果没有副本集(replica set)是个不完整架构，假设其中的一个分片挂掉那四分之一的数据就丢失了，所以在高可用性的分片架构还需要对于每一个分片构建 replica set 副本集保证分片的可靠性（高可用）。
- 生产环境通常是 2 个副本 + 1 个仲裁。

7.3.4 shard和chunk

片键 (shard key)

- 为了在集合中分配文档，MongoDB使用分片主键分割集合。
- 分片主键由不重复的字段或者字段集合组成。
- 对一个集合分片时，你要选择分片主键，分片主键在分片以后不能修改。
- 一个分片集合只有一个分片主键。
- 为了对非空的集合进行分片，集合必须有一个以分片主键开头的索引。
- 对于空集合，如果集合对于分片主键没有一个合适的索引，MongoDB将创建索引。
- 分片主键的选择将影响分片集群的性能、效果和扩展能力。一个最佳的硬件和基础设施的集群的瓶颈取决于分片主键的选择
- 分片主键的选择将影响你的集群使用的分片策略。

区块 (chunks)

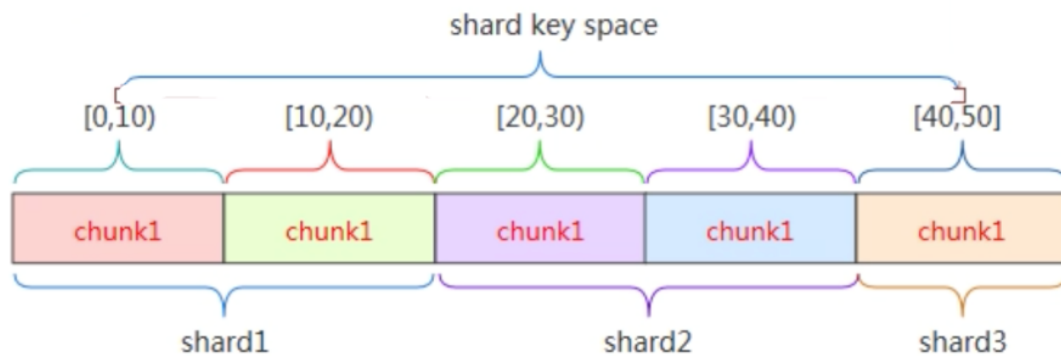
- MongoDB分割分片数据到区块，每一个区块包含基于分片主键的左闭右开的区间范围。
- 在分片集群中，MongoDB通过分片迁移区块，使用分片集群权衡器。
- 权衡器视图完成一个公平的区块平衡，通过集群中所有的分片。

7.3.5 分片策略

范围分片 (Range based sharding)

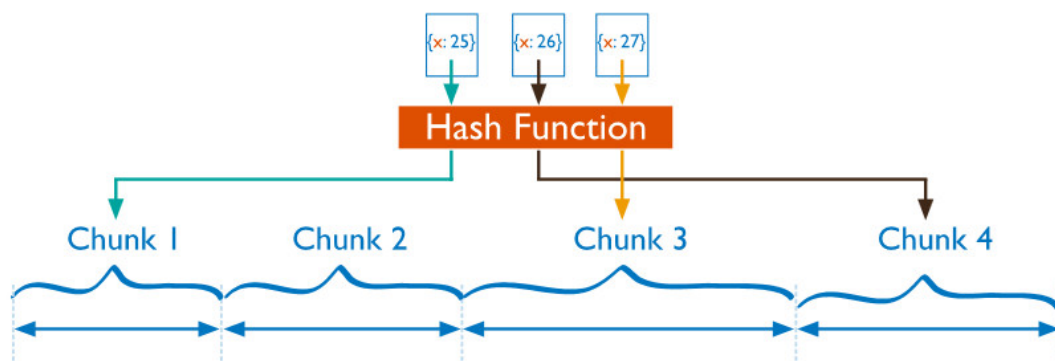
- 范围分片是基于分片主键的值切分数据，每一个区块将会分配到一个范围。

- 范围分片适合满足在一定范围内的查找，例如查找X的值在【100-200】之间的数据，mongo 路由根据Config server中存储的元数据，可以直接定位到指定的shard的Chunk中
- 缺点：如果shardkey有明显递增（或者递减）趋势，则新插入的文档多会分布到同一个chunk，无法扩展写的能力



哈希分片 (Hash based sharding)

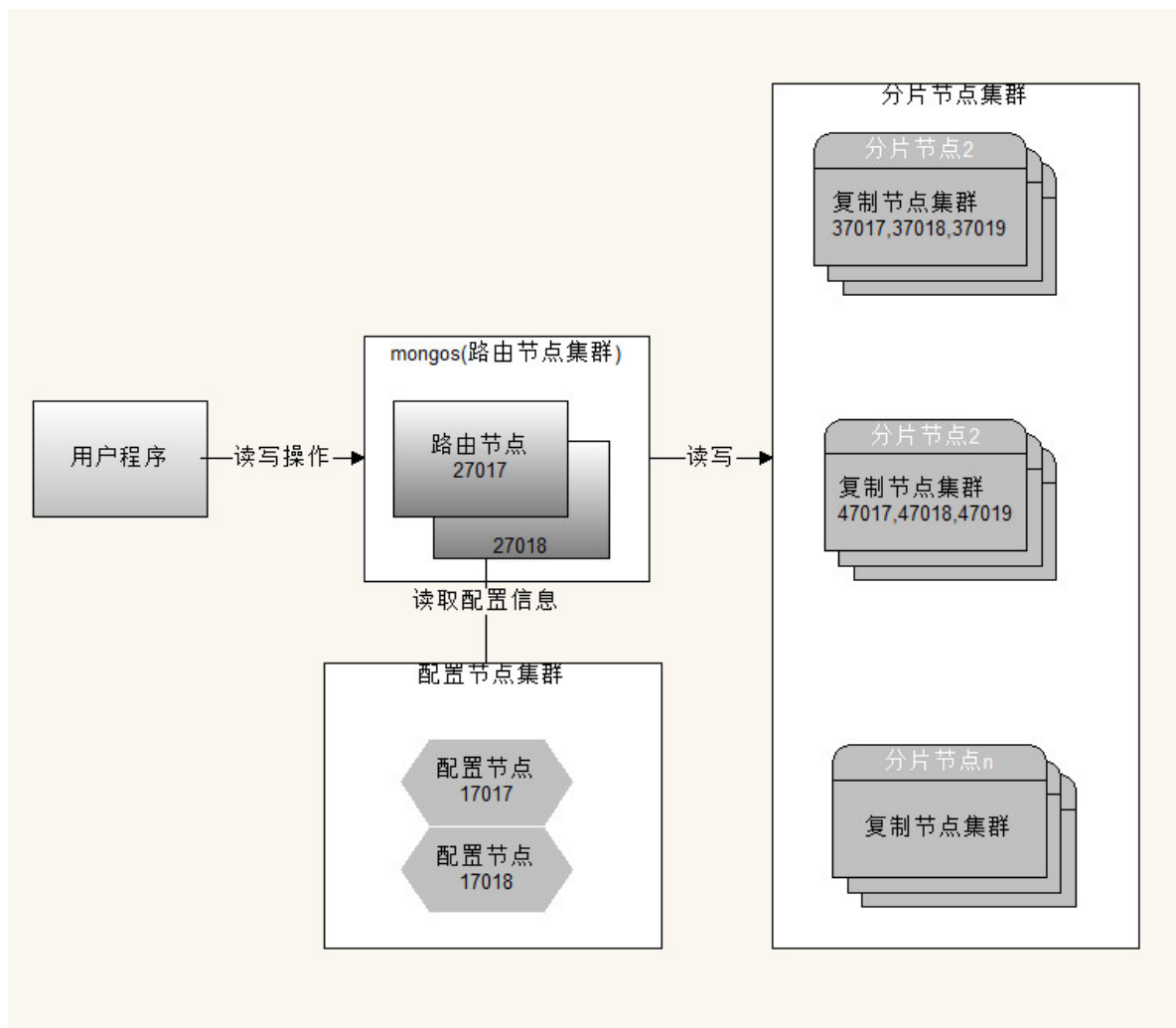
- Hash分片是计算一个分片主键的hash值，每一个区块将分配一个范围的hash值。
- Hash分片与范围分片互补，能将文档随机的分散到各个chunk，充分的扩展写能力，弥补了范围分片的不足
- 缺点：不能高效的服务范围查询，所有的范围查询要分发到后端所有的Shard才能找出满足条件的文档。



合理的选择shard key

- 选择shard key时，要根据业务的需求及『范围分片』和『Hash分片』2种方式的优缺点合理选择，要根据字段的实际原因对数据进行分片，否则会产生过大的Chunk（热块）

7.3.6 分片搭建架构图



7.3.7 分片集群的搭建过程

1. 配置 config 节点集群

- 将mongo解压到根目录，改名为shard_cluster
- 进入shard_cluster并创建config目录用来存集群节点的配置文件

```
[root@A opt]# cp mongodb-linux-x86_64-4.1.3.tgz /
[root@A opt]# tar -zxvf mongodb-linux-x86_64-4.1.3.tgz
[root@A /]# mv mongodb-linux-x86_64-4.1.3 shard_cluster
[root@A /]# cd shard_cluster
[root@A shard_cluster]# mkdir config
[root@A shard_cluster]# cd config
[root@A config]# vim config-17017.conf
[root@A config]# vim config-17018.conf
```

```
dbpath=/data/mongo/config1
bind_ip=0.0.0.0
port=17017
fork=true
logpath=/data/mongo/logs/config1.log
replset=configCluster
configsvr=true
```

```
dbpath=/data/mongo/config2
bind_ip=0.0.0.0
port=17018
fork=true
logpath=/data/mongo/logs/config2.log
replset=configCluster
configsvr=true
```

- 配置文件中的目录需要手动创建出来

```
[root@A config]# mkdir -p /data/mongo/config1
[root@A config]# mkdir -p /data/mongo/config2
[root@A config]# mkdir /data/mongo/logs
```

- 启动两台mongo，并进入任意节点shell 并添加配置节点集群，注意，要进入admin库操作

```
./bin/mongo --port 17017
```

```
use admin
```

```
var cfg = {  
  "_id": "configCluster",  
  "protocolVersion" : 1,  
  "members": [  
    {"_id": 0, "host": "192.168.204.141:17017"},  
    {"_id": 1, "host": "192.168.204.141:17018"}  
  ]  
};  
  
rs.initiate(cfg)
```

2. 配置 shard 节点集群

- 分片1：shard1集群搭建37017到37019
- 创建data数据目录
- shard1-37017.conf

```
dbpath=/data/mongo/shard1-37017  
bind_ip=0.0.0.0  
port=37017  
fork=true  
logpath=/data/mongo/logs/shard1-37017.log  
replSet=shard1  
shardsvr=true
```

```
dbpath=/data/mongo/shard1-37018
bind_ip=0.0.0.0
port=37018
fork=true
logpath=/data/mongo/logs/shard1-37018.log
replSet=shard1
shardsvr=true
```

```
dbpath=/data/mongo/shard1-37019
bind_ip=0.0.0.0
port=37019
fork=true
logpath=/data/mongo/logs/shard1-37019.log
replSet=shard1
shardsvr=true
```

```
var cfg = {
  "_id": "shard1",
  "protocolVersion" : 1,
  "members": [
    {"_id": 1, "host": "192.168.204.141:37017"},
    {"_id": 2, "host": "192.168.204.141:37018"},
    {"_id": 3, "host": "192.168.204.141:37019"}
  ]
};
rs.initiate(cfg)
rs.status()
```

- 分片2：shard2集群搭建47017到47019
- 创建data数据目录

```
dbpath=/data/mongo/shard2-47017
bind_ip=0.0.0.0
port=47017
fork=true
logpath=/data/mongo/logs/shard2-47017.log
replSet=shard2
shardsvr=true
```

```
dbpath=/data/mongo/shard2-47018
bind_ip=0.0.0.0
port=47018
fork=true
logpath=/data/mongo/logs/shard2-47018.log
replSet=shard2
shardsvr=true
```

```
dbpath=/data/mongo/shard2-47019
bind_ip=0.0.0.0
port=47019
fork=true
logpath=/data/mongo/logs/shard2-47019.log
replSet=shard2
shardsvr=true
```

```
var cfg = {
  "_id": "shard2",
  "protocolVersion" : 1,
  "members": [
    {"_id": 1, "host": "192.168.204.141:47017"},
    {"_id": 2, "host": "192.168.204.141:47018"},
    {"_id": 3, "host": "192.168.204.141:47019"}
  ]
};

rs.initiate(cfg)
rs.status()
```

3. 配置 mongos 路由节点

- route-27017.conf

```
port=27017
bind_ip=0.0.0.0
fork=true
logpath=/data/mongo/logs/route.log
configdb=configCluster/192.168.204.141:17017,192.168.204.141:17018
```

4. mongos中添加分片节点

- 启动路由使用mongos命令，而不是mongo或mongod

```
[root@A shard_cluster]# ./bin/mongos -f
config/route-27017.conf
```

- 进入路由和普通的节点一样，使用mongo

```
[root@A shard_cluster]# ./bin/mongo --port 27017
```

```
// 查看分片集群的状态用sh对象
sh.status()
// 在admin库下操作
use admin
// 注意连接字符串的格式拼写
sh.addShard("shard1/192.168.204.141:37017,192.168.204.141:37018,192.168.204.141:37019");
sh.addShard("shard2/192.168.204.141:47017,192.168.204.141:47018,192.168.204.141:47019");
```

5. 指定片键 并 设置分片大小

- 使用mongoos完成分片开启和分片大小设置

```
// 为数据库开启分片功能
sh.enableSharding("lagou")

// 为指定集合开启分片功能
// sh.shardCollection("lagou.emps",{"片键字段名如
_id":索引说明})
sh.shardCollection("lagou.emps",{"name":"hashed"})
```

- 数据明显分配得不均匀，因为默认chunk的大小是64M，而我们刚才插入的数据量不大，估计也不会产生几个chunks，而且chunk迁移需要满足一定的条件：

每个shard块数量	相差阈值	备注
小于20个	2	块最多的分片与块最少的分片，相差2个以上，进行块迁移
20-79	4	相差4个以上进行迁移
大于等于80	8	相差8个以上进行迁移

- 所以，为了能够测试看的很清楚，我们调整下chunk的大小为1M：

```
// 修改分片大小：
use config
db.settings.save({_id:"chunksize",value:1})
db.settings.find()
```

6. 插入数据测试

- 通过路由（在路由节点下操作）循环向集合中添加数据

```
mongos>
for(var i = 1 ; i <= 1000 ; i++){
    db.emps.insert({"_id":i,"name":"test"+i});
}
```

- 测试
 - 在shard1和shard2分片中，分别执行，观察结果

```
shard1 PRIMARY> use lagou
switched to db lagou
shard1:PRIMARY> db.emps.find();
{ "_id" : 3, "name" : "test3" }
{ "_id" : 4, "name" : "test4" }
{ "_id" : 5, "name" : "test5" }
{ "_id" : 9, "name" : "test9" }
```

```
shard2 PRIMARY> use lagou
switched to db lagou
shard2:PRIMARY> db.emps.find();
{ "_id" : 1, "name" : "test1" }
{ "_id" : 2, "name" : "test2" }
{ "_id" : 6, "name" : "test6" }
{ "_id" : 7, "name" : "test7" }
{ "_id" : 8, "name" : "test8" }
```

8、安全认证

8.1 安全认证概述

- MongoDB 默认是没有账号的，可以直接连接，无须身份验证。
- 实际项目中肯定是要权限验证的，否则后果不堪设想。
- 从2016年开始发生了多起MongoDB黑客赎金事件，大部分MongoDB安全问题暴露出的短板其实是用户
- 首先用户对于数据库的安全不重视，其次用户在使用过程中可能没有养成定期备份的好习惯，最后是企业可能缺乏有经验和技术的专业人员。
- 所以对MongoDB进行安全认证是必须要做的。

8.2 用户相关操作

8.2.1 切换到admin数据库对用户的添加

- 创建 MongoDB 登录用户以及分配权限的方法

```
db.createUser({
  user : "账号",
  pwd  : "密码",
  roles : [
    { role: "父亲角色", db: "养育孩子" },
    { role: "职员角色", db: "为企业工作" }
  ]
});
```

- user : 创建的用户名称，如 admin、root、lagou
- pwd : 用户登录的密码
- roles : 为用户分配的角色，不同的角色拥有不同的权限，参数是数组，可以同时设置多个
- role : 角色，MongoDB 已经约定好的角色，不同的角色对应不同的权限 后面会对role做详细解释
- db : 数据库实例名称，如 MongoDB 4.0.2 默认自带的有 admin、local、config、test 等，即为哪个数据库实例 设置用户

举例:

1. 无认证的启动mongo，执行创建用户等操作
2. 关闭mongo，以安全认证的方式启动mongo

```
use admin

db.createUser({
  user : "laosun",
  pwd : "123123",
  roles : [
    { role:"root" , db:"admin" }
  ]
});
```

8.2.2 修改密码

```
db.changeUserPassword('laosun','newPassword');
```

8.2.3 用户添加角色

```
db.grantRolesToUser('用户名', [{ role: '角色名', db:
'数据库名' }])
```

8.2.4 以auth 方向启动mongod

```
./bin/mongod -f conf/mongo.conf --auth
```

- 也可以在mongo.conf 中添加auth=true 参数)

8.2.5 验证用户

```
db.auth("账号","密码")
```

8.2.6 删除用户

```
db.dropUser("用户名")
```

8.3 角色

8.3.1 数据库内置的权限

- read：允许用户读取指定数据库
- readWrite：允许用户读写指定数据库
- dbAdmin：允许用户在指定数据库中执行管理函数，如索引创建、删除，查看统计或访问system.profile
- userAdmin：允许用户向system.users集合写入，可以找指定数据库里创建、删除和管理用户
- clusterAdmin：只在admin数据库中可用，赋予用户所有分片和复制集相关函数的管理权限。
- readAnyDatabase：只在admin数据库中可用，赋予用户所有数据库的读权限
- readWriteAnyDatabase：只在admin数据库中可用，赋予用户所有数据库的读写权限
- userAdminAnyDatabase：只在admin数据库中可用，赋予用户所有数据库的userAdmin权限
- dbAdminAnyDatabase：只在admin数据库中可用，赋予用户所有数据库的dbAdmin权限。
- root：只在admin数据库中可用。超级账号，超级权限

8.3.2 角色对应的权限

- 数据库用户角色：read、readWrite;
- 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 ;
- 所有数据库角色：readAnyDatabase、readWriteAnyDatabase、userAdminAnyDatabase、dbAdminAnyDatabase
- 超级用户角色：root
- 这里还有几个角色间接或直接提供了系统超级用户的访问 (dbOwner 、 userAdmin、userAdminAnyDatabase)

总结：生产环境用read和readWrite。开发人员用root，运维人员backup和restore；

8.4 安全认证实现流程

8.4.1 创建管理员

- MongoDB 服务端开启安全检查之前，**至少需要有一个管理员账号**，admin 数据库中的用户都被视为管理员
- 如果 admin 库没有任何用户的话，即使在其他数据库中创建了用户，并启用身份验证，默认的连接方式依然会有超级权限，即仍然可以不验证账号密码照样能进行 CRUD，安全认证相当于无效。

```
use admin

db.createUser({
  user:"root",
  pwd:"123123",
  roles:[{role:"root",db:"admin"}]
})
```

8.4.2 创建普通用户

- 如下所示 mydb是自己新建的数据库，没安全认证之前可以随意 CRUD
- 其余的都是 mongoDB 4.0.2 自带的数据库

```
>show dbs
admin    0.000GB
config   0.000GB
local    0.000GB
mydb     0.000GB
```

- 为 admin 库创建管理员之后，现在来为普通数据库创建普通用户，以 mydb 为例，方式与创建管理员一致，切换到指定数据库进行创建即可。
- 如下所示，为 mydb 数据库创建了两个用户，zhangSan 拥有读写权限，liSi 拥有只读权限，密码都是 123123
- 注意：大小写敏感

```
use mydb

db.createUser({
  user:"zhangsan",
  pwd:"123123",
  roles:[{role:"readwrite",db:"mydb"}]
})

db.createUser({
  user:"lisi",
  pwd:"123123",
  roles:[{role:"read",db:"mydb"}]
})
```

- 在客户端关闭 MongoDB 服务

```
> use admin
switched to db admin
> db.shutdownServer()
server should be down...
```

8.4.3 MongoDB 安全认证方式启动

```
mongod --dbpath=数据库路径 --port=端口 --auth
```

8.4.4 分别以普通用户和管理员登录验证权限

- 普通用户现在仍然像以前一样进行登录，如下所示直接登录进入 mydb 数据库中，登录是成功的，只是登录后日志少了很多东西，而且执行 show dbs 命令，以及 show tables 等命令都是失败的，即使没有被安全认证的数据库，用户同样操作不了，这都是因为权限不足

- 一句话：用户只能在自己权限范围内的数据库中进行操作

```
MongoDB shell version v4.1.3
connecting to: mongodb://127.0.0.1:27020/
Implicit session: session { "id" : UUID("a5131732-
0a4f-4122-842e-75098c7e67e8") }
MongoDB server version: 4.1.3
> show dbs
```

报错内容如下：

```
2021-08-31T16:57:16.760+0800 E QUERY [js] Error:
listDatabases failed:{
  "ok" : 0,
  "errmsg" : "command listDatabases requires
authentication",
  "code" : 13,
  "codeName" : "Unauthorized"
} :
_getErrorWithCode@src/mongo/shell/utils.js:25:13
Mongo.prototype.getDBs@src/mongo/shell/mongo.js:67:1
shellHelper.show@src/mongo/shell/utils.js:879:19
shellHelper@src/mongo/shell/utils.js:769:15
@(shellhelp2):1:1
```

- 如下所示，登录之后必须使用 db.auth("账号","密码") 方法进行安全认证，认证通过，才能进行权限范围内的操作
- show dbs 已经看不到其它未拥有操作权限的数据库了，显然它也操作不了其它未授权的数据库
- 因为创建的用户 zhangSan 拥有对 mydb 的读写权限，所以如下所示 读写正常

```
Error: error: {
  "ok" : 0,
  "errmsg" : "too many users are
authenticated",
  "code" : 13,
  "codeName" : "Unauthorized"
}
> exit
```

如上错误码所示，是因为mongo的shell命令的bug，因为切换了太多用户。是一个坑。

exit退出mongo，然后重新进入，
use mydb
db.auth认证

- db.auth("账号","密码") 返回 1 表示验证成功，0 表示验证失败
- 注意 db.auth 认证必须在对应的数据库下进行，比如为 db1 数据库单独创建的用户 dbUser
- 不能在 db2 数据库下执行 db.auth 验证操作，否则是失败的，只能到 db1 下去验证。

```
> use mydb
switched to db mydb
> db.auth("zhangSan","123123")
1
> db.student.find()
{ "_id" : ObjectId("612dedf9934f78f43ad82c86"),
  "name" : "aaa" }
{ "_id" : ObjectId("612dedfc934f78f43ad82c87"),
  "name" : "bbb" }
> db.student.insert({name:"ccc"})
writeResult({ "nInserted" : 1 })
> db.student.find()
```



```
{ "_id" : ObjectId("612dedf9934f78f43ad82c86"),
"name" : "aaa" }
{ "_id" : ObjectId("612dedfc934f78f43ad82c87"),
"name" : "bbb" }
{ "_id" : ObjectId("612df0c31c31a7fd1b60f1eb"),
"name" : "ccc" }
> exit
bye
[root@A mongodb-linux-x86_64-4.1.3]# ./bin/mongo --
port 27020
MongoDB shell version v4.1.3
connecting to: mongodb://127.0.0.1:27020/
Implicit session: session { "id" : UUID("2959f4c2-
61d4-4e2c-b9d9-1c9fdb739cc6") }
MongoDB server version: 4.1.3
> use mydb
switched to db mydb
> db.auth("lisi","123123")
1
> db.student.find()
{ "_id" : ObjectId("612dedf9934f78f43ad82c86"),
"name" : "aaa" }
{ "_id" : ObjectId("612dedfc934f78f43ad82c87"),
"name" : "bbb" }
{ "_id" : ObjectId("612df0c31c31a7fd1b60f1eb"),
"name" : "ccc" }
> db.student.insert({name:"ddd"})
WriteCommandError({
  "ok" : 0,
  "errmsg" : "not authorized on mydb to execute
command { insert: \"student\", ordered: true, lsid:
{ id: UUID(\"2959f4c2-61d4-4e2c-b9d9-1c9fdb739cc6\")
}, $db: \"mydb\" }",
  "code" : 13,
  "codeName" : "Unauthorized"
})
```

- 除了登录后再使用 `db.auth("账号","密码")` 方法进行验证之外，也可以像 Mysql 一样，在登录的同时指定 账号与密码

```
mongo localhost:27017/mydb1 -u "账号" -p "密码"
```

- 客户端管理员登录如下所示 管理员 root 登录，安全认证通过后，拥有对所有数据库的所有权限。

```
> use admin
switched to db admin
> db.auth("root","123123")
1
> show dbs
...
```

8.5 分片集群认证

8.5.1 认证之前 进入路由

```
[root@A shard_cluster]# ./bin/mongo --port 27017
```

- 创建管理员（完全参考8.4即可）

```
mongos> use admin

db.createUser({
  user:"root",
  pwd:"123123",
  roles:[{role:"root",db:"admin"}]
})
```

- 普通用户

```
mongos> use laosundb
```

```
db.createUser({  
  user:"laosun",  
  pwd:"123123",  
  roles:[{role:"readwrite",db:"laosundb"}]  
})
```

8.5.2 关闭所有的节点

- 配置节点、分片节点、路由节点
- 一个一个关闭太累了，搞一个小工具，可以快速关闭进程

```
[root@A ~]# yum install psmisc  
[root@A ~]# killall mongod  
[root@A ~]# killall mongo  
[root@A ~]# killall mongos
```

8.5.3 生成密钥文件 并修改权限

- 密钥文件是各个节点直接通信的凭证（**通关文牒**）
- 另一种通关文牒：509数字证书（[百度自行查阅](#)）
- **赋予权限只能是600**

```
[root@A ~]# mkdir -p /data/mongodb  
[root@A ~]# openssl rand -base64 756 >  
/data/mongodb/testKeyFile.file  
[root@A ~]# chmod 600 /data/mongodb/testKeyFile.file
```

OpenSSL简介

- 在所有的类 Unix 发行版、Solaris、Mac OS X 和 Windows 中默认都用openssl这个工具来生成**高强度**随机密码（这个是系统自带，使用率最高）

```
[root@A ~]# openssl rand 6 -base64  
vaJgqrLe
```

- 注：上面的命令将生成一个随机的、长度为 6 个字符的高强度密码，这种方式不支持同时生成多个密码。
- 我们强烈推荐你生成 14 个字符的密码。当然你可以使用 OpenSSL 生成任意长度的密码。
- 不是说6位长度吗？怎么显示的不是6位？因为你生成的位数不足以达到高强度密码要求，所以默认情况下openssl会给你添加字符，使其保证密码高强度！

8.5.4 配置节点和分片节点 设置密钥文件

- 向 xxx.conf 文件中追加如下内容

```
auth=true  
keyFile=/data/mongodb/testKeyFile.file
```

8.5.5 路由节点 设置密钥文件

- 切记：路由不需要认证

```
keyFile=/data/mongodb/testKeyFile.file
```

8.5.6 启动所有节点

- 挨个启动太累，可以编写一个shell 脚本批量启动

```
[root@A /]# cd /shard_cluster
[root@A shard_cluster]# vim startup.sh
[root@A shard_cluster]# chmod +x startup.sh
[root@A shard_cluster]# ./startup.sh
```

- 下面内容拷贝到startup.sh文件中

```
./bin/mongod -f config/config-17017.conf
./bin/mongod -f config/config-17018.conf
./bin/mongod -f config/shard1-37017.conf
./bin/mongod -f config/shard1-37018.conf
./bin/mongod -f config/shard1-37019.conf
./bin/mongod -f config/shard2-47017.conf
./bin/mongod -f config/shard2-47018.conf
./bin/mongod -f config/shard2-47019.conf
./bin/mongos -f config/route-27017.conf
```

8.5.7 认证测试

- 进入路由节点（测试管理账号）

```
mongos> use admin
mongos> show dbs 报错
mongos> db.auth("root","123123")
mongos> show dbs 成功
```

- 测试普通用户（避免认证用户过多的错误，退出路由重新进）

```
mongos> use laosundb
mongos> show dbs 报错
mongos> db.auth("laosun","123123")
mongos> show dbs 成功
```

8.5.7 Spring boot 连接安全认证的分片集群

`spring.data.mongodb.username=账号` (不要使用root, 使用普通用户, 否则报错认证失败)

`spring.data.mongodb.password=密码`

uri方式: 注意帐号密码在@前面

`spring.data.mongodb.uri=mongodb://账号:密码@IP:端口/数据库名`

9、MongoDB监控

9.1 MongoDB Ops Manager 简介

- MongoDB Ops Manager(MMS) 是用于**监控和备份**MongoDB的基础设施服务。
- MongoDB OPS Manager是一个Web应用程序, 它需要1个mongodb数据库, 这个数据库是用来支持本身的MongoDB OPS Manager来运行的。因此, 如果我们想要MongoDB OPS Manager运行起来, 最少也需要安装一个MongoDB数据库。

9.2 Ops Manager 作用

- 简易的自动化数据库部署、扩展、升级和任务管理;
- 通过 OPS 平台提供的超过 100 项仪表、图表, 可以对mongodb 进行多种监控;
- 支持单节点、分片集群的备份和恢复;

9.3 安装 Ops Manager

9.3.1 下载安装包

- 安装包的大小根据版本的不同略有差异，但维持在1GB左右
- 官网：<https://www.mongodb.com/subscription/downloads/archived>
- 往下滚动，各种版本都有，mongo4对应mms4，mongo5对应mms5，**版本要对应**

9.3.2 上传到服务器并解压

```
[root@A opt]# tar -zxvf mongodb-mms-4.1.3.53428.20190304T2149Z-1.x86_64.tar.gz
```

- 解压完成后改个名字，名字太长了

```
[root@A opt]# mv mongodb-mms-4.1.3.53428.20190304T2149Z-1.x86_64 mongodb-mms
```

9.3.3 编辑配置文件

- conf-mms.properties

```
[root@A opt]# cd mongodb-mms/conf  
[root@A conf]# vim conf-mms.properties
```

- 根据自己的mongodb 进行配置，但必须先启动一个27020的mongodb实例

```
mongo.mongoUri=mongodb://127.0.0.1:27020/?
maxPoolSize=150
mongo.ssl=false
```

- mms.conf

```
[root@A conf]# vim mms.conf
```

```
BASE_PORT=8080
BASE_SSL_PORT=8443

# JVM configurations
JAVA_MMS_UI_OPTS="${JAVA_MMS_UI_OPTS} -Xss328k -
Xmx2048m -Xms2048m -XX:NewSize=600m -Xmn1500m -
XX:ReservedCodeCacheSize=128m -XX:-
OmitStackTraceInFastThrow"
```

- 一般修改端口 和 内存 如果虚拟机内存不太够 可以适当减少内存配置
- 比如 `-Xmx4352m -Xms4352m` 改成 `-Xmx2048m -Xms2048m`

9.3.4 启动 mms

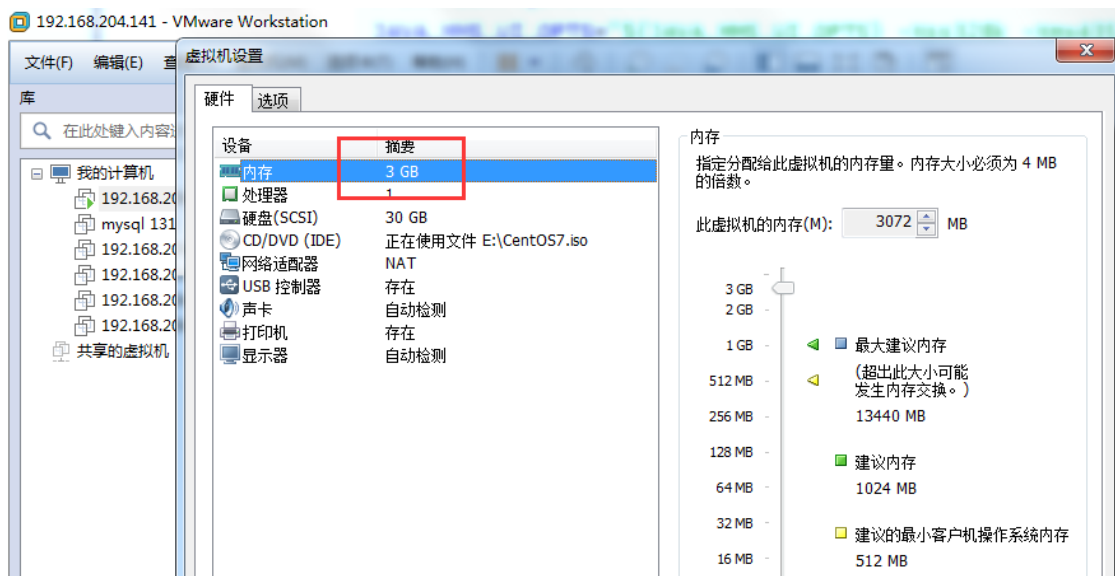
- 启动之前确保 Ops Manager对应的 MongoDB 数据库已经启动

```
[root@A mongodb-mms]# ./bin/mongodb-mms start
```

- 启动时间漫长，多等一会
- 如果启动失败，请查看 `/opt/mongodb-mms/logs/mms0-startup.log` 日志文件查看错误信息
- 如果错误是由于空间不足引起的

```
[root@A mongodb-mms]# cat logs/mms0-startup.log
OpenJDK 64-Bit Server VM warning: Option UseConcMarkSweepGC was deprecated in version 9.0 and will likely be removed in a future release.
OpenJDK 64-Bit Server VM warning: INFO: os::commit_memory(0x000000074dc00000, 2990538752, 0) failed; error=Not enough space (errno=12)
#
# There is insufficient memory for the Java Runtime Environment to continue.
# Native memory allocation (mmap) failed to map 2990538752 bytes for committing reserved memory.
# An error report file with more information is saved as:
# /opt/mongodb-mms/hs_err_pid7124.log
```


- 我们可以通过修改虚拟机的内存为3G，然后修改mms.conf中将内存大小为2G，重启虚拟机，一切重新来过再试



```
WARNING: ALL illegal access operations will be denied in
Successfully finished pre-flight checks
```

```
Migrate Ops Manager data
Running migrations... [ OK ]
Start Ops Manager server
Instance 0 starting..... [ OK ]
Starting pre-flight checks
WARNING: An illegal reflective access operation has occur
WARNING: Illegal reflective access by com.google.inject.i
java.lang.ClassLoader.defineClass(java.lang.String,byte[
WARNING: Please consider reporting this to the maintainer
WARNING: Use --illegal-access=warn to enable warnings of
WARNING: All illegal access operations will be denied in
Successfully finished pre-flight checks

Start Backup Daemon... [ OK ]
[root@01 mangedb mms1#
```

启动成功！

9.3.5 访问 mms 首页

- 这里的端口是在 mms.conf 中配置的：`http://主机:端口`

MongoDB Ops Manager

Login

Username

Usually, this is the email you used to register.

Password

Forgot password?

Login

Register for a new account

9.4 配置 Ops Manager

9.4.1 注册账号

- 帐号虽然说要邮箱，但可以不用邮箱 laosun
- 密码的要求较高，Abc_123123

Register for MongoDB Ops Manager

Email Address

laosun

Password

.....

✓ 8-characters minimum

✓ One number

✓ One letter

✓ One special character

First Name

wukong

Last Name

sun

☒ I agree to the Evaluation Agreement

Create Account

9.4.2 OPS Manager的配置页面

- 使用创建的用户登录。来到OPS Manager的配置页面。

Web Server

* required ** restart required

URL To Access Ops Manager *

Fully qualified URL, including the port number, of the Ops Manager Application.

Example: `http://opsmanager.example.com:8080`

http://192.168.204.141:8080

"From" Email Address *

The email address used for sending the general emails, such as alerts. You can include an alias with the email address.

Example: `Ops Manager<opsmanager@example.com>`

laosun@163.com

"Reply To" Email Address *

The email address to send replies to general emails

laosun@163.com

Admin Email Address *

The email address of the Ops Manager admin. The email address will receive emails related to problems with Ops Manager itself

laosun@163.com

Email Delivery Method Configuration *

The method to send email

SMTP Email Server

Transport *

Transfer protocol (smtp or smtps) as specified by your email provider

smtp

SMTP Server Hostname *

Hostname of the SMTP server

laosun@163.com

SMTP Server Port *

Port of the SMTP server

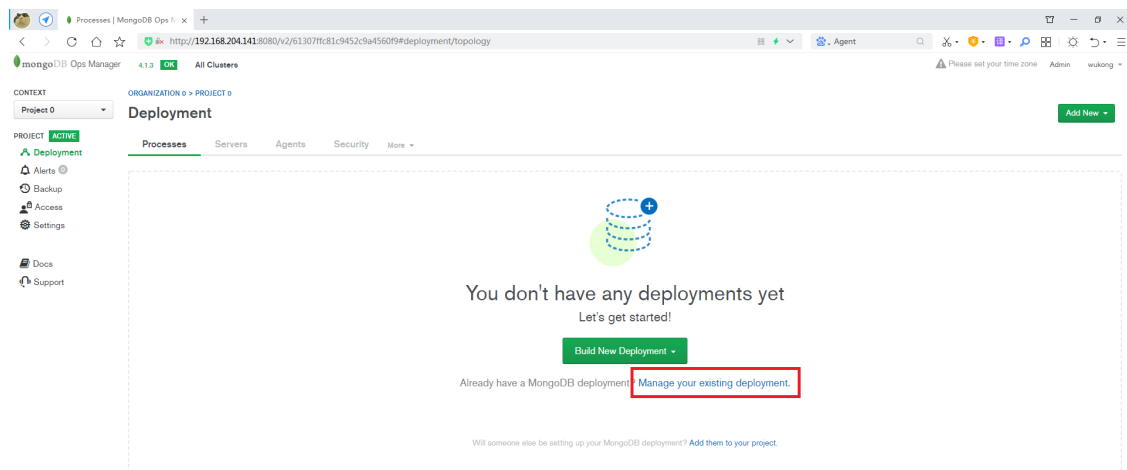
25

- 配置的过程很多，选项也很多，只填写黄色星号的即可。
- 95%的选项选择默认值即可
- 需要自己创建一个mms的版本目录

```
[root@A mongodb-mms]# mkdir /opt/mongodb-mms/versions
```

9.5 配置 MongoDB Ops Manager Agent

1. 点击下面界面中的 Manage your existing deployment

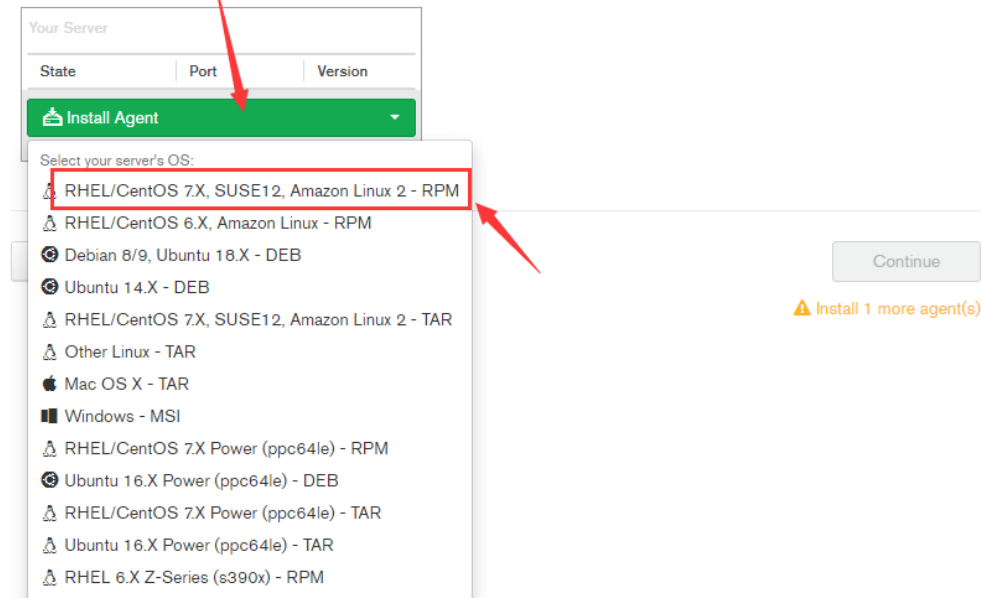


2. 单击绿色 install Agent

根据我们目标数据库的操作系统，我们可以选择相应的Agent来安装。（PS：实际的操作系统版本可能更多，大致上一致即可，只要能装上）

Install an Automation Agent on a single server

Ops Manager uses lightweight Agents to allow secure communications between your MongoDB deployment and Ops Manager. To get started, install an Automation Agent on any single server within your deployment.



3. 根据弹出的安装步骤 安装 agent

Automation Agent Installation Instructions

To save time, you can repeat each step of these instructions in parallel across servers with the same OS

1. Download the agent



```
curl -OL http://192.168.204.141:8080/download/agent/automation/mongodb-mms-automation-agent
```

and install the package.



```
sudo rpm -U mongodb-mms-automation-agent-manager-6.3.1.5645-1.x86_64.rhel7.rpm
```

2. Create a new Agent API Key. After being generated, keys will only be shown once. Treat this API Key like a password.

✓ Key Generated

3. Next, open the config file



```
sudo vi /etc/mongodb-mms/automation-agent.config
```

and enter your API key, Project ID, and Ops Manager Base URL as shown below.



```
mmsGroupId=61307ffc81c9452c9a4560f9
```



```
mmsApiKey=613095eb81c9452c9a4572c5644e79cd299b891da270707fc451f5cd
```



```
mmsBaseUrl=http://192.168.204.141:8080
```

To manage your API keys, visit the [Agent API Keys](#) tab.

4. Start the agent.



```
sudo systemctl start mongodb-mms-automation-agent.service
```

On SUSE, it may be necessary to run:




```
sudo /sbin/service mongodb-mms-automation-agent start
```

- 按照上面的提示的步骤一步步操作即可,最后继续点击 Verify Agent

Install an Automation Agent on a single server

Ops Manager uses lightweight Agents to allow secure communications between your MongoDB deployment and Ops Manager. To get started, install an Automation Agent on any single server within your deployment.

A		
State	Port	Version
 Automation Agent		6.3.1.5645
Automation Agent Successfully Verified		

Go Back

已经验证通过




Continue

✓ Agents Verified

- 等待全自动安装完成！如下图所示

Set up Ops Manager Monitoring

The Automation Agent will now install a Monitoring Agent on your server. The Monitoring Agent will be used to collect health and performance metrics from your MongoDB deployment. A Backup Agent will also be installed, but will lie dormant until you choose to enable our Backup feature.

A		
State	Port	Version
 Automation Agent		6.3.1.5645
 Monitoring Agent		7.0.0.481
 Backup Agent		7.5.0.1051
Monitoring Agent Successfully Verified		

Go Back

Continue

9.6 监控现有的Sharding Cluster服务

9.6.1 确保 shard 集群已启动

- 确保路由、配置集群、分片集群都已经启动
 - 每个节点的配置文件中（注意dbpath和logpath使用绝对路径，否则监控有警告）
 - 取消注释安全认证的配置




```
dbpath=/data/mongo/config1
bind_ip=0.0.0.0
port=17017
fork=true
logpath=/data/mongo/logs/config1.log
replSet=configCluster
configsvr=true
#auth=true
#keyFile=/data/mongodb/testKeyFile.file
```

- 启动所有节点

9.6.2 在安装好的agent界面点击继续

Set up Ops Manager Monitoring

The Automation Agent will now install a Monitoring Agent on your server. The Monitoring Agent will be used to collect health and performance metrics from your MongoDB deployment. A Backup Agent will also be installed, but will lie dormant until you choose to enable our Backup feature.

A		
State	Port	Version
 Automation Agent		6.3.1.5645
 Monitoring Agent		7.0.0.481
 Backup Agent		7.5.0.1051
Monitoring Agent Successfully Verified		

Go Back

Continue

9.6.3 配置监控服务 单机实例或者集群实例都可以

Import your deployment for monitoring

To get started, we'll need some basic information about one item in your MongoDB deployment before we can start monitoring it. From this single seed item, Ops Manager will automatically discover all connected processes. If you have a sharded cluster, seed Ops Manager with a single mongos. If you have a replica set, seed Ops Manager with the primary.

Hostname

This is the hostname of the seed MongoDB process as seen from the Ops Manager agent. This hostname must be unique and resolvable from any server within your deployment. Do not use 'localhost'.

192.168.204.141

Port

This is the port that the seed MongoDB process is running on. To test that you have chosen the right hostname and port, login to the server on which you installed the Ops Manager Agent and test connecting via the MongoDB shell:

```
mongo hostname:port
```

27017

Enable Authentication

Does your MongoDB deployment require authentication?

☐ NO

Use TLS/SSL

Do you want to use TLS/SSL for MongoDB connections? ⓘ

☐ NO

点击继续后，等待agent发现集群

Go Back

Continue


9.6.4 配置完成点击continue 会进行集群发现

Enable Authentication

Does your MongoDB deployment require authentication?

☐ NO

Use TLS/SSL

Do you want to use TLS/SSL for MongoDB connections? 

☐ NO

Deployment found

发现集群

Go Back















Continue

继续continue

Adding your deployment to Ops Manager

Ops Manager is discovering all MongoDB processes in your deployment. This may take up to 10 minutes, please be patient. Once you see all of the processes in your deployment here, you can hit continue. If not all processes are discovered, make sure that each process is able to accept connections from the server on which you installed the Ops Manager Agent.

Make sure you see all of the processes in your deployment before continuing




A 		
OS Name:  CentOS Linux release 7.4.1708 (Core)		
RAM: 2831 MB		
State	Port	Version
 Automation Agent		6.3.1.5645
 Monitoring Agent		7.0.0.481
 Backup Agent		7.5.0.1051
 shard1	37017	4.1.3
 shard1	37018	4.1.3
 shard1	37019	4.1.3
 shard2	47017	4.1.3
 shard2	47018	4.1.3
 shard2	47019	4.1.3
 config	17017	4.1.3
 config	17018	4.1.3
 mongos	27017	4.1.3

Go Back

Continue

9 processes and 1 server discovered.

9.6.5 集群发现要求配置节点和分片节点必须是绝对路径 注意一下即可



You have successfully installed an Automation and Monitoring Agent on one of your servers.

You will be required to install an Automation Agent on each remaining server in your deployment on the next step.

... but first, your deployment must meet the following requirements to add Automation:

- ✓ The fully qualified domain name `hostname -f` for each server in the deployment must be resolvable from every other server. For example, if you login to Server A, and `hostname -f` returns `a.example.com`, then this address must be resolvable from Server B, C, and D as well.
- ✓ Your MongoDB processes must be running as the same system user as the Automation Agent. For example, if your MongoDB process is running as the "mongod" user, the Automation Agent must also be running as the "mongod" user.
- ✓ Your deployment makes use only of supported MongoDB configuration file options. [See docs here](#).

☐ I understand that this requires the installation of an automation agent on each of the servers in my deployment and I have read the requirements/risks above

⚠ We are unable to add automation to this deployment item. Please ensure that your deployment has been successfully imported for monitoring and that it meets the requirements above. **Error: dbPath must be an absolute path for process 192.168.211.136:17017**

Previous

No, Just Monitor

Continue

9.6.6 根据提示做修改 直到出现下面的界面

Do you want to add Ops Manager Automation to this deployment?

You've already connected your database and set it up to be monitored by Ops Manager. You can now take your Ops Manager integration a step further and unlock the power of managing your database with the click of a button. You will be required to install an Automation Agent on each remaining server in your deployment on the next step...



You have successfully installed an Automation and Monitoring Agent on one of your servers.



You will be required to install an Automation Agent on each remaining server in your deployment on the next step.



... but first, your deployment must meet the following requirements to add Automation:

- ✓ The fully qualified domain name `hostname -f` for each server in the deployment must be resolvable from every other server. For example, if you login to Server A, and `hostname -f` returns `a.example.com`, then this address must be resolvable from Server B, C, and D as well.
- ✓ Your deployment makes use only of supported MongoDB configuration file options. [See docs here](#).
- ✓ Your MongoDB processes must be running as the same system user as the Automation Agent. For example, if your MongoDB process is running as the "mongod" user, the Automation Agent must also be running as the "mongod" user.

☒ I understand that this requires the installation of an automation agent on each of the servers in my deployment and I have read the requirements/risks above

Previous

No, Just Monitor

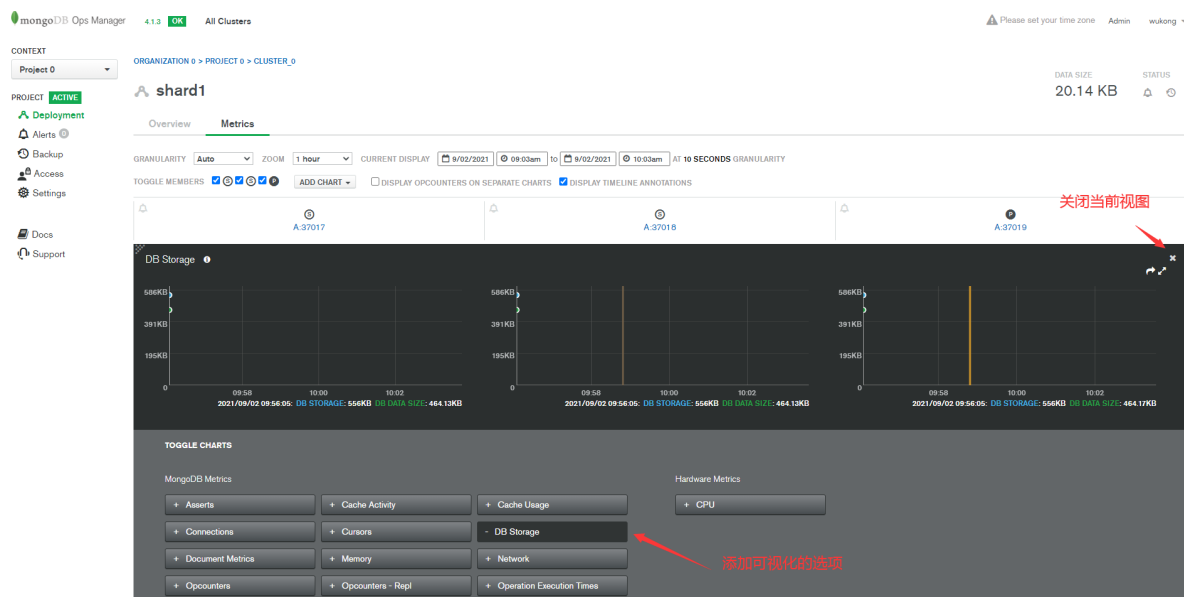
Continue

9.6.7 点击 No , JustMonitor 即可看到集群信息

The screenshot shows the MongoDB Ops Manager interface. The top navigation bar includes the MongoDB logo, version 4.1.3, and a status indicator 'OK'. The main content area is titled 'Deployment' and shows details for 'Cluster_0'. The 'Metrics' tab is selected, displaying a graph of the cluster's performance over time. The graph shows a sharp spike in the 'Reads' metric around 10:01 AM. Below the graph, a table lists the cluster's components, including 'shard1' and 'shard2', with their respective data sizes and status indicators.

Shard Name	Alerts	Data Size	Show	Primarys	Secondaries	All	Reads	Writes	Queued
shard1		20.14 KB							
shard2		17.86 KB							

9.6.8 查看具体的分片 和 监控指标



10、数据备份与恢复

10.1 备份的目的

- 防止硬件故障引起的数据丢失
- 防止人为错误误删数据
- 时间回溯（时光机）
- 监管要求

10.2 数据备份

- 在MongoDB 中我们使用 `mongodump` 命令来备份MongoDB数据。
- 该命令可以导出所有数据到指定目录中。

- `mongodump` 命令可以通过参数指定导出的数据库或者集合。
- `mongodump` 命令脚本语法如下：

```
> mongodump -h dbhost -d dbname -o dbdirectory
```

-h：MongoDB所在**服务器地址**

例如：127.0.0.1，当然也可以指定端口号：127.0.0.1:37017

--db 或者 -d：需要备份的**数据库实例**

例如：laosundb

-o：备份的**数据存放位置**

例如：/root/bdatas 在备份完成后，自动建立/root/bdatas 目录，这个目录里面存放备份数据。

语法	描述	实例
mongodump --host HOST_NAME --port PORT_NUMBER	该命令将备份所有 MongoDB 数据	mongodump --host 192.168.204.141 -- port 37017
mongodump --db DB_NAME --out BACKUP_DIRECTORY	备份指定的 数据库	mongodump --port 37017 --d laosundb -- out /data/backup/
mongodump -- collection COLLECTION --db DB_NAME	该命令将备份指定数据库的集合	mongodump -- collection emps -d laosundb

举例：

```
[root@a mongodb]# ./bin/mongodump -h
192.168.204.141:27020 -d test -o /data/bdatas
[root@a mongodb]# ./bin/mongodump --
host=192.168.204.141 --port=27020 -d test -c emps -
o=/data/emps_bak
```

- 导出的数据目录中：一个数据库对应一个目录，目录里面一个集合对应2个文件（bson和metadata.json）
 - bson文件中记录数据
 - metadata.json文件中记录集合的结构等元数据

10.3 数据恢复

- mongod使用 `mongorestore` 命令来恢复备份的数据。
- `mongorestore` 命令脚本语法如下：

```
> mongorestore -h <hostname>[:port] -d dbname
<path>
```

--host <:port> 或 -h <:port>： MongoDB所在服务器地址

默认为：localhost:37017

--db 或 -d： 需要恢复的数据库实例

例如：test，当然这个名称也可以和备份时候的不一样，比如test2

--drop： 恢复的时候，先删除当前数据，然后恢复备份的数据。

就是说，恢复后，备份后添加修改的数据都会被删除，慎用哦！

<path/>：

mongorestore 最后的一个参数，设置备份数据所在位置，例

如：/root/bdatas/laosundb

你不能同时指定 <path/> 和 --dir 选项，--dir也可以设置备份目录。

注意：恢复指定的数据库 需要在恢复的路径中出现数据库的名字

--dir：指定备份的目录

你不能同时指定 <path/> 和 --dir 选项。

举例：

```
[root@a mongodb]# ./bin/mongorestore -h
127.0.0.1:27020 -d test /data/emp_bak/test
[root@a mongodb]# ./bin/mongorestore -h
127.0.0.1:27020 /data/bdatas
```

- 如果指定-d参数，则数据目录必须指定到库目录
- 如果没指定库的目录，那么-d参数也不可以指定
- 总之：-d和库目录test要么同时存在，要么同时删除

10.4 全量加增量备份和恢复案例

注意的问题：

- 删除复制集中原来的数据文件目录/data/mongo/data/server1
重新建立数据目录data
- 重新启动复制集中的实例，进入37017重新进行复制集的配置

```

var cfg = {
  "_id" : "lagouCluster",
  "protocolVersion" : 1,
  "members" : [

{"_id":1,"host":"192.168.204.141:37017","priority":1
0},
    { "_id":2,"host":"192.168.204.141:37018"},
    { "_id":3,"host":"192.168.204.141:37019"}
  ]
}

rs.initiate(cfg)

```

1. 进入mongodb 插入两条数据

```

use laosundb
db.emps.insert({name:"aaa",salary:1})
db.emps.insert({name:"bbb",salary:2})

```

2. 进行全量备份

备份之前：将fullbackup和oplog_bak目录删掉（没有的话，请忽略删除目录的步骤）

```

[root@a mongodb]# rm -rf fullbackup/ oplog_bak/
[root@a mongodb]# ./bin/mongodump --
host=192.168.204.141 --port=37017 --
out=/mongo/fullbackup

```

3. 继续插入数据 并更新

```
db.emps.insert({name:"ccc",salary:3})
db.emps.insert({name:"ddd",salary:4})
db.emps.insert({name:"eee",salary:5})

db.emps.update({name:"ccc"},{$set:{salary:333}})
db.emps.update({name:"ddd"},{$set:{salary:444}})
```

4. 做增量备份

```
[root@a mongodb]# ./bin/mongodump --
host=192.168.204.141 --port=37017 -d local -c
oplog.rs -o=/mongo/oplog_bak
```

5. 删除所有的数据

```
db.emps.remove({})
db.emps.find()
```

6. 先恢复全量数据

```
[root@a mongodb]# ./bin/mongorestore --
host=192.168.204.141 --port=37017 --
dir=/mongo/fullbackup
```

```
db.emps.find()
// 此时，恢复了前两条数据
```

```
lagouCluster:PRIMARY> db.emps.find();
{ "_id" : ObjectId("6136281fd91287aa5cbe76f3"), "name" : "aaa", "salary" : 1 }
{ "_id" : ObjectId("6136281fd91287aa5cbe76f4"), "name" : "bbb", "salary" : 2 }
```

7. 恢复数据到指定的时间点

修改数据备份文件

oplog.rs.bson -> oplog.bson （去掉中间的rs）

并且删除meta元数据文件

```
[root@a ~]# cd /mongo/oplog_bak/local
[root@a local]# mv oplog.rs.bson oplog.bson
[root@a local]# rm -rf oplog.rs.metadata.json
```

找出第一次更新的时间

- op : operation
- u : update
- ts : timestamp
- 1 : 升序（时间最近的考前）

```
use local
db.oplog.rs.find({"op" : "u"}).sort({"ts":1})
```

恢复到指定的时间点的数据

```
[root@a mongodb]# ./bin/mongorestore --
host=192.168.204.141 --port=37017 --oplogReplay -
-oplogLimit "实际查询出来的时间"
/mongo/oplog_bak/local
```

实际查询出来的时间 = `Timestamp(1630935854, 1)`

```
[root@a mongodb]# ./bin/mongorestore --
host=192.168.204.141 --port=37017 --oplogReplay -
-oplogLimit "1630935854:1" /mongo/oplog_bak/local
```

查看数据恢复

```
use laosundb
db.emps.find()
```

```
lagouCluster:PRIMARY> db.emps.find()
{ "_id" : ObjectId("6136281fd91287aa5cbe76f3"), "name" : "aaa", "salary" : 1 }
{ "_id" : ObjectId("6136281fd91287aa5cbe76f4"), "name" : "bbb", "salary" : 2 }
{ "_id" : ObjectId("6136283fd91287aa5cbe76f5"), "name" : "ccc", "salary" : 3 }
{ "_id" : ObjectId("6136283fd91287aa5cbe76f6"), "name" : "ddd", "salary" : 4 }
{ "_id" : ObjectId("6136283fd91287aa5cbe76f7"), "name" : "eee", "salary" : 5 }
```

8. 恢复所有的增量数据

```
[root@a mongodb]# ./bin/mongorestore --
host=192.168.204.141 --port=37017 --oplogReplay
/mongo/oplog_bak/local
```

查看数据恢复

```
use laosundb
db.emps.find()
```

```
lagouCluster:PRIMARY> db.emps.find()
{ "_id" : ObjectId("6136281fd91287aa5cbe76f3"), "name" : "aaa", "salary" : 1 }
{ "_id" : ObjectId("6136281fd91287aa5cbe76f4"), "name" : "bbb", "salary" : 2 }
{ "_id" : ObjectId("6136283fd91287aa5cbe76f5"), "name" : "ccc", "salary" : 333 }
{ "_id" : ObjectId("6136283fd91287aa5cbe76f6"), "name" : "ddd", "salary" : 444 }
{ "_id" : ObjectId("6136283fd91287aa5cbe76f7"), "name" : "eee", "salary" : 5 }
```

10.5 定时备份

1. 创建备份数据的存放目录

mongo_bak_now : 数据备份的临时目录

mongo_bak_list : 备份数据压缩文件的存放目录

```
[root@a /]# mkdir -p
/mongo/backup/mongo_bak/mongo_bak_now
/mongo/backup/mongo_bak/mongo_bak_list
```

2020-1-2 2 : 2 : 3

2. 编写备份脚本

```
[root@a /]# vi /mongo/backup/mongobk.sh
```

```
#!/bin/sh
# dump 命令执行路径，根据mongodb安装路径而定
DUMP=/opt/replica_set/mongodb/bin/mongodump
# 临时备份路径
OUT_DIR=/mongo/backup/mongo_bak/mongo_bak_now
# 压缩后的备份存放路径
TAR_DIR=/mongo/backup/mongo_bak/mongo_bak_list
# 当前系统时间
DATE=`date +%Y_%m_%d_%H_%M_%S`
# 数据库账号
#DB_USER=user
# 数据库密码
#DB_PASS=password
# 代表删除7天前的备份，即只保留近 7 天的备份
DAYS=7
# 最终保存的数据库备份文件名称，例如
mongod_bak_2020_08_02_10_20_35.tar.gz
TAR_BAK="mongod_bak_${DATE}.tar.gz"
cd $OUT_DIR
rm -rf $OUT_DIR/*
mkdir -p $OUT_DIR/$DATE
$DUMP -h 127.0.0.1 --port 37017 -o $OUT_DIR/$DATE
# 压缩格式为 .tar.gz 格式
tar -zPcvf $TAR_DIR/$TAR_BAK $OUT_DIR/$DATE
# 删除 7 天前的备份文件
find $TAR_DIR/ -mtime +$DAYS -delete
exit
```

3. 修改脚本权限

```
[root@a mongodb]# chmod +x
/mongo/backup/mongobk.sh
```

4. 编辑crontab

```
[root@a mongodb]# crontab -e
```

#表示每天凌晨2点30执行备份

```
30 2 * * * /mongo/backup/mongobk.sh
```

测试的时候 可以改成 一分钟备份一次

```
* * * * * /mongo/backup/mongobk.sh
```

5. 查看crontab 的状态

```
[root@a mongodb]# service crond status
```

如果没有启动，可以使用下面的命令启动定时服务和加入开机自启动

启动定时任务

```
[root@a mongodb]# service crond start
```

加入开机自动启动，设定crond在等级3和5为开机运行服务（系统服务分为0-6个级别），on表示启动

```
[root@a mongodb]# chkconfig --level 35 crond on
```

6. 查看定时任务和删除定时任务

```
[root@a mongodb]# crontab -l      # 查看当前定时任务  
list
```

```
[root@a mongodb]# crontab -r      # 删除定时任务  
remove
```

```
[root@a mongodb]# crontab -e      # 编辑定时任务  
edit
```

11、底层存储引擎剖析

11.1 概述

- 存储引擎是MongoDB的核心组件，负责管理数据如何存储在硬盘和内存上。
- MongoDB支持的存储引擎有MMAPv1，WiredTiger和InMemory。
 - InMemory存储引擎用于将数据只存储在内存中，只将少量的元数据(meta-data)和诊断日志（Diagnostic）存储到硬盘文件中，由于不需要Disk的IO操作，就能获取所需的数据
 - InMemory存储引擎大幅度降低了数据查询的延迟（Latency）。
- 从mongodb3.2开始默认的存储引擎是WiredTiger，3.2版本之前的默认存储引擎是MMAPv1
- mongodb4.x版本不再支持MMAPv1存储引擎。

```
storage:
  journal:
    enabled: true
  dbPath: /data/mongo/
  ##是否一个库一个文件夹
  directoryPerDB: true
  ##数据引擎
  engine: wiredTiger
  ## WT引擎配置
  wiredTiger:
    engineConfig:
      ##WT最大使用cache（根据服务器实际情况调节）
```



```
cacheSizeGB: 2
##是否将索引也按数据库名单独存储
directoryForIndexes: true
journalCompressor:none (默认snappy)
##表压缩配置
collectionConfig:
    blockCompressor: zlib (默认snappy,还可选
none、zlib)
##索引配置
indexConfig:
    prefixCompression: true
```

11.2 WiredTiger存储引擎优势

1. 文档空间分配方式

WiredTiger使用的是BTree存储，MMAPV1 线性存储 需要Padding

2. 并发级别

WiredTiger 文档级别锁（并发性能更优秀），MMAPV1引擎使用表级锁

3. 数据压缩

snappy (默认) 和 zlib ,相比MMAPV1(无压缩) 空间节省数倍。

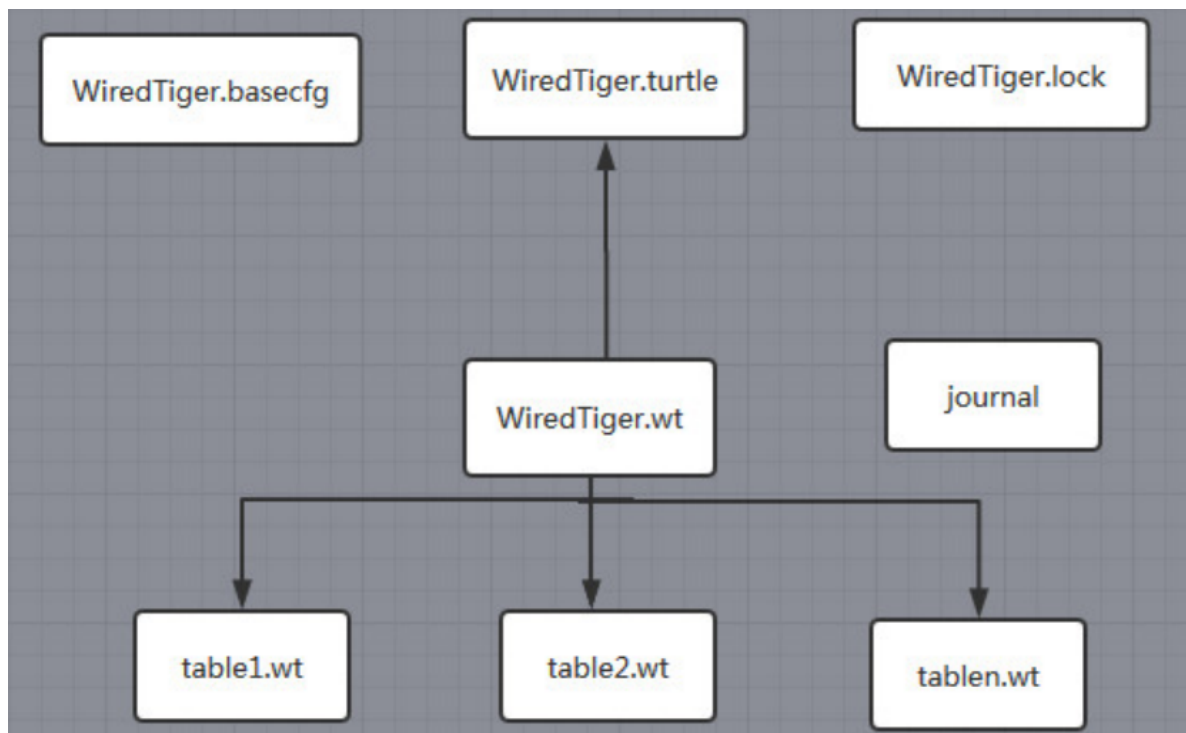
4. 内存使用

WiredTiger 可以指定内存的使用大小。而MMAPv1会占用机器的全部内存

5. Cache使用

WT引擎使用了二阶缓存WiredTiger Cache , File System Cache 来保证Disk上的数据的最终一致性。而MMAPv1只有journal 日志。

11.3 WiredTiger引擎包含的文件和作用



- table*.wt : 存储各张表的数据
- WiredTiger.wt : 存储table* 的**元数据**
- WiredTiger.turtle : 存储WiredTiger.wt的元数据
- WiredTiger.basecfg : 存储基本配置信息 , 与 ConfigServer有关系
- WiredTiger.lock : 定义锁操作
- journal : 存储WAL (Write Ahead Log : 前置写日志)

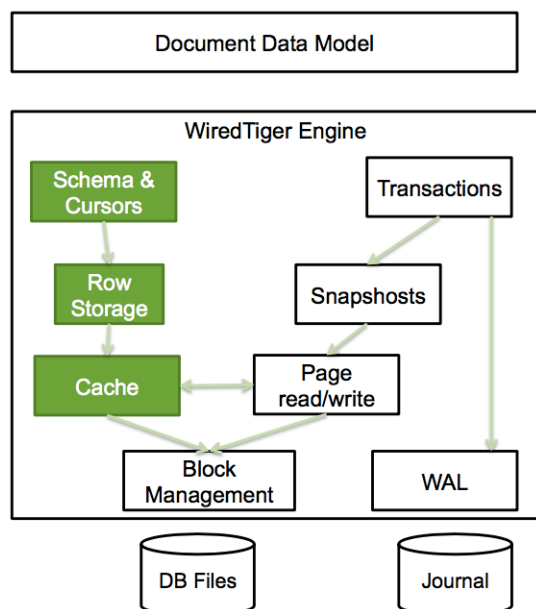
11.4 WiredTiger存储引擎实现原理

11.4.1 写请求

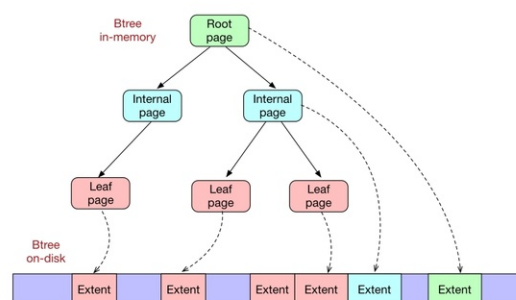
- WiredTiger的写操作会默认写入 Cache，并持久化到 **WAL (Write Ahead Log : 预写日志)**
- 每60s或Log文件达到2G做一次 checkpoint
- 当然我们也可以通过在写入时传入 `j:true` 的参数强制 journal 文件的同步并产生快照文件。

```
writeConcern { w:<value>, j:<boolean>, wtimeout:<number> }
```

- WiredTiger初始化时，恢复至最新的快照状态，然后再根据 WAL恢复数据，保证数据的完整性。

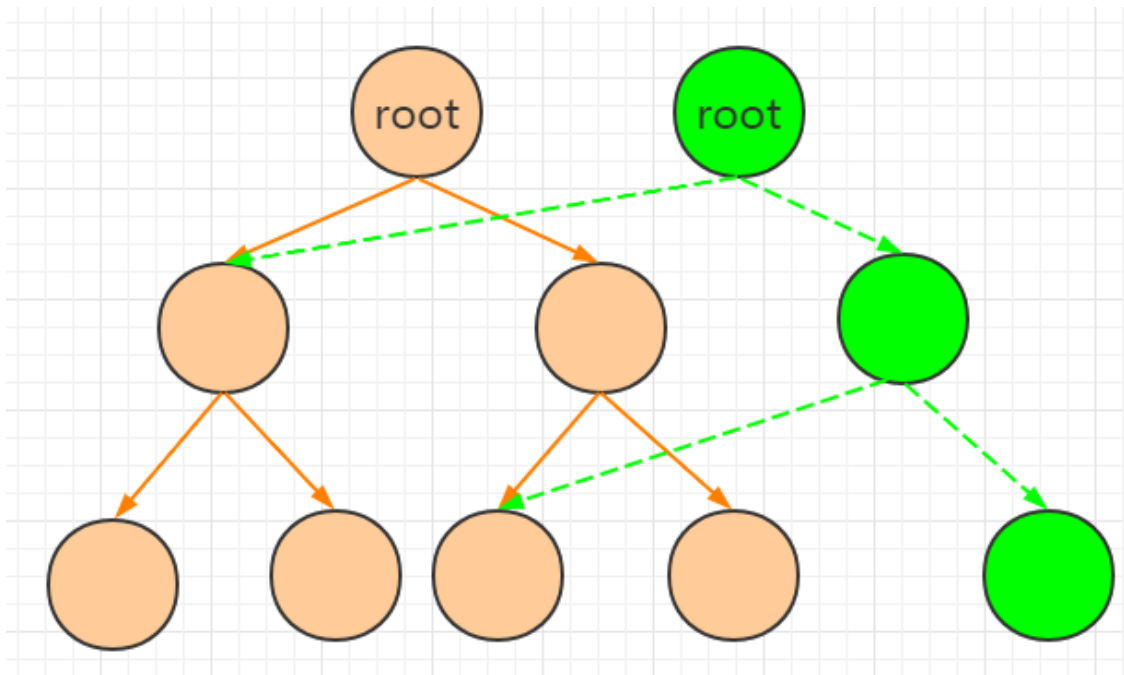


Btree、Page、Extent



- **Cache是基于BTree的**
- 节点是一个page，root page是根节点，internal page是中间索引节点，leaf page真正存储数据
- 数据以page为单位读写。WiredTiger采用Copy on write的方式管理写操作 (insert、update、delete)

- 修改操作会先缓存在cache里，持久化时，修改操作不会在原来的leaf page上进行，写入新分配的page，每次checkpoint都会产生一个新的root page。



11.4.2 checkpoint流程

1. 对所有的table进行一次checkpoint，每个table的checkpoint的元数据更新至WiredTiger.wt
2. 对WiredTiger.wt进行checkpoint，将该table checkpoint的元数据更新至临时文件WiredTiger.turtle.set
3. 将WiredTiger.turtle.set重命名为WiredTiger.turtle
4. 上述过程如果中间失败，WiredTiger在下次连接初始化时，首先将数据恢复至最新的快照状态，然后根据WAL恢复数据，以保证存储可靠性。

（假设100s的数据，60s时产生快照，系统启动直接恢复即可，后面40s的数据通过WAL日志中的时间点进行恢复）

11.4.3 Journaling

- 在数据库宕机时，为保证 MongoDB 中数据的持久性，MongoDB 使用了 Write Ahead Logging 向磁盘上的 journal 文件预先进行写入。
- 除了 journal 日志，MongoDB 还使用检查点（checkpoint）来保证数据的一致性，当数据库发生宕机时，我们就需要 **checkpoint** 和 **journal** 文件协作完成数据的恢复工作。
 - 在数据文件中查找上一个检查点的标识符
 - 在 journal 文件中查找标识符对应的记录
 - 重做对应记录之后的全部操作

