课程主题

mongodb原理分析(单机和集群)、集群搭建专题

课程目标

- 掌握mongodb的router server、config server、data server工作原理
- 掌握mongodb的replica set（副本集）工作原理
- 掌握mongodb的分片策略以及shard和chunk的理解
- 掌握mongodb的主从搭建方式
- 掌握mongodb的副本集集群搭建方式
- 掌握mongodb的混合方式集群搭建方式

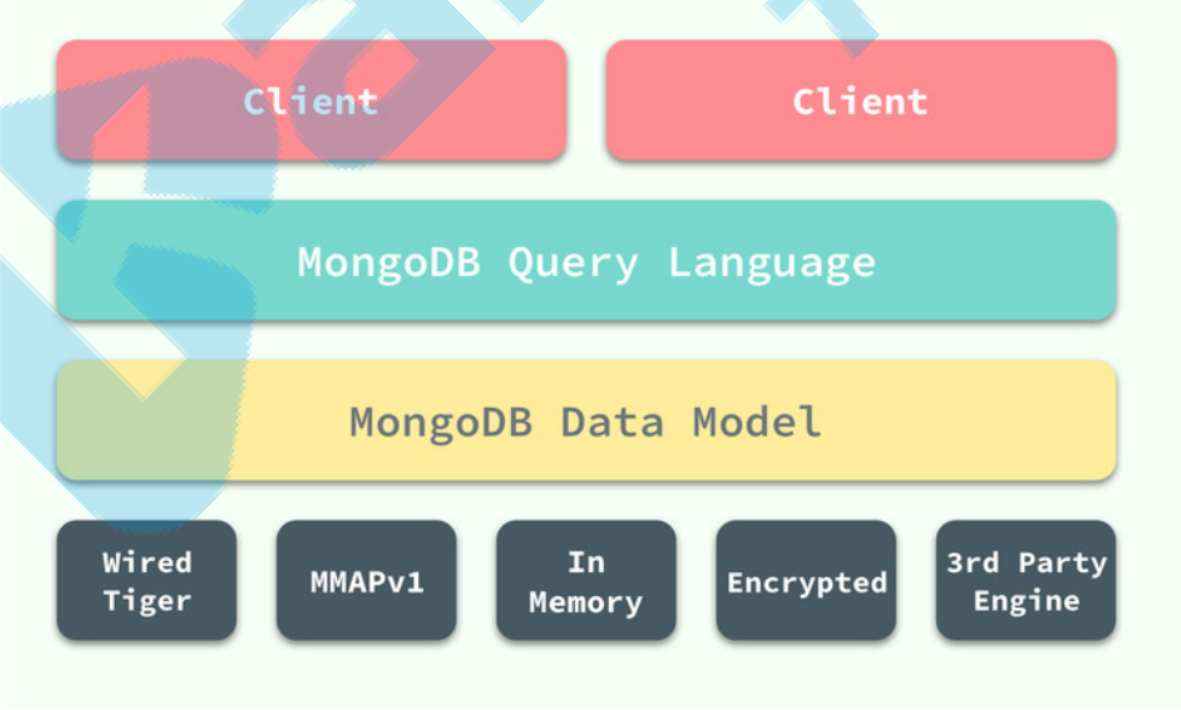
图解MongoDB底层原理

MongoDB架构

主流的 NoSQL 数据库之一

分布式文档数据库

将数据以类似 JSON 的方式（BSON）存储在磁盘上



RDBMS 与 MongoDB的区别

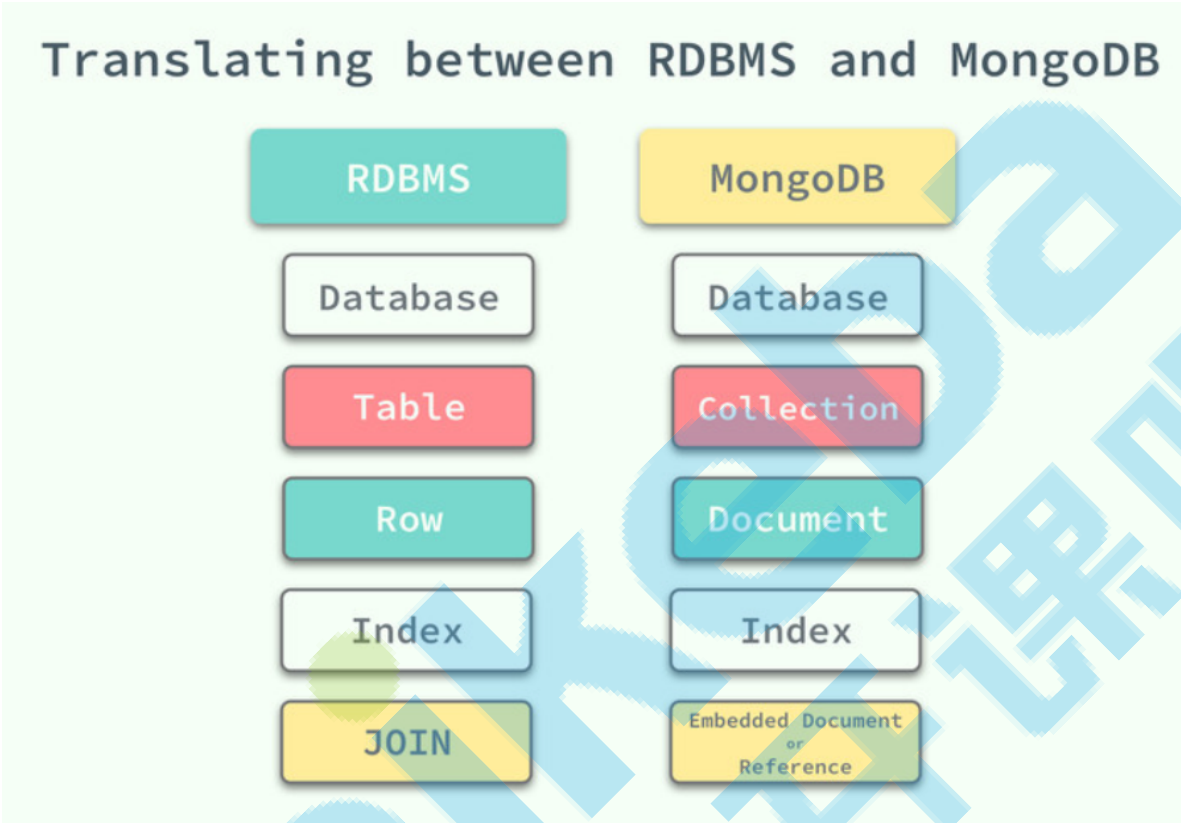
用法：

Rdbms： ACID 放业务数据，复杂的逻辑

MongoDB： 海量、高并发、低价值数据 （非核心业务）， 记录日志， 业务不固定，demo

RDBMS 其实使用 `Table` 的格式将数据逻辑地存储在一张二维的表中，表结构是固定的

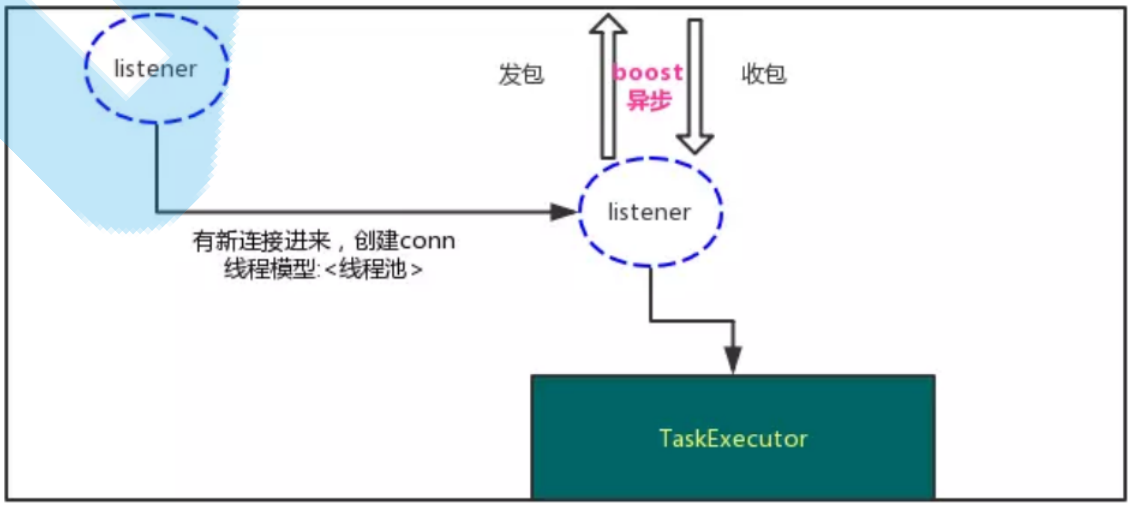
MongoDB 用 Schema 结构随时可用调整



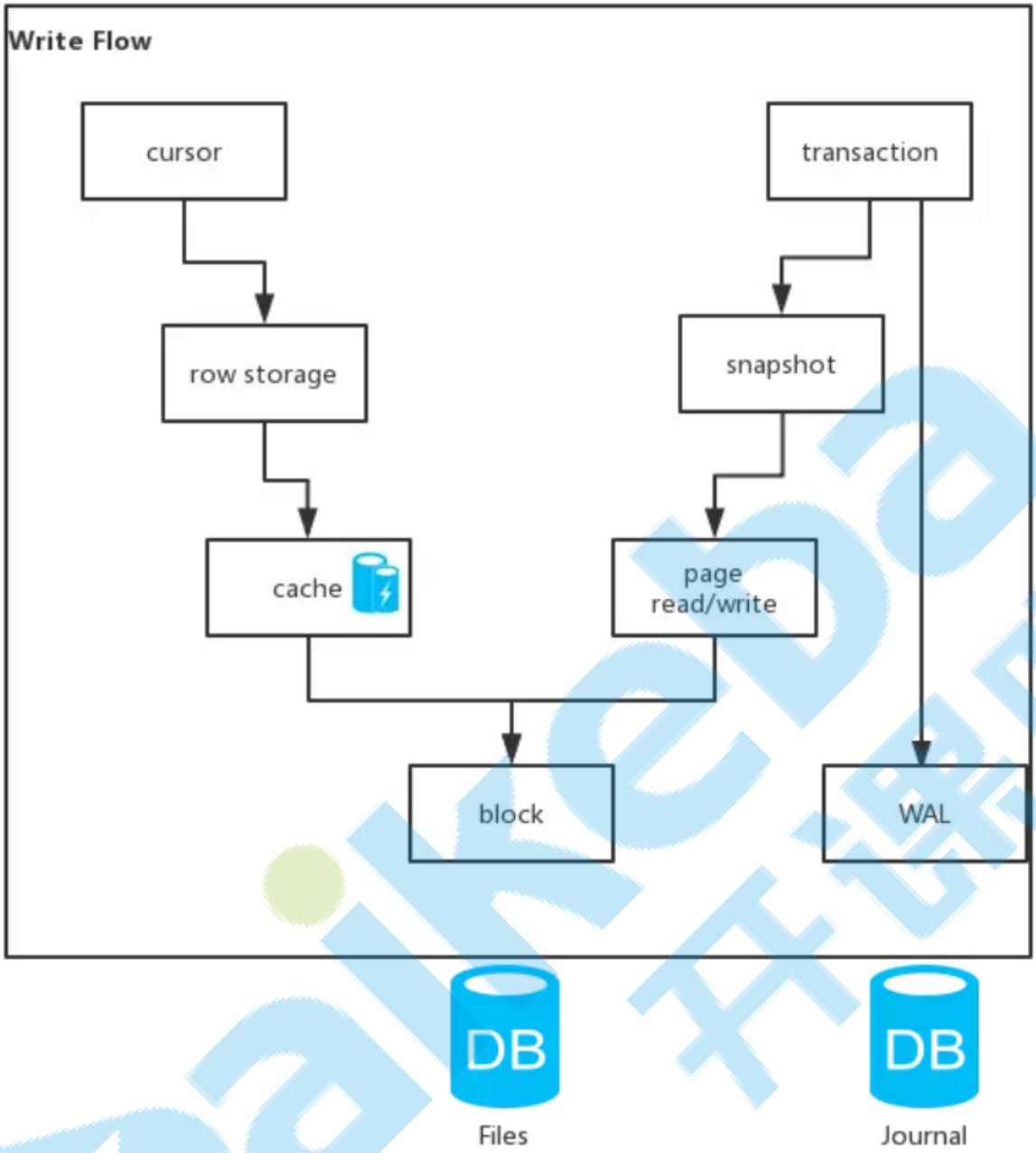
MongoDB Wiredtiger存储引擎实现原理

MongoDB2.3后默认采用WiredTiger存储引擎。(之前为MMAPV1引擎)

Transport Layer业务层



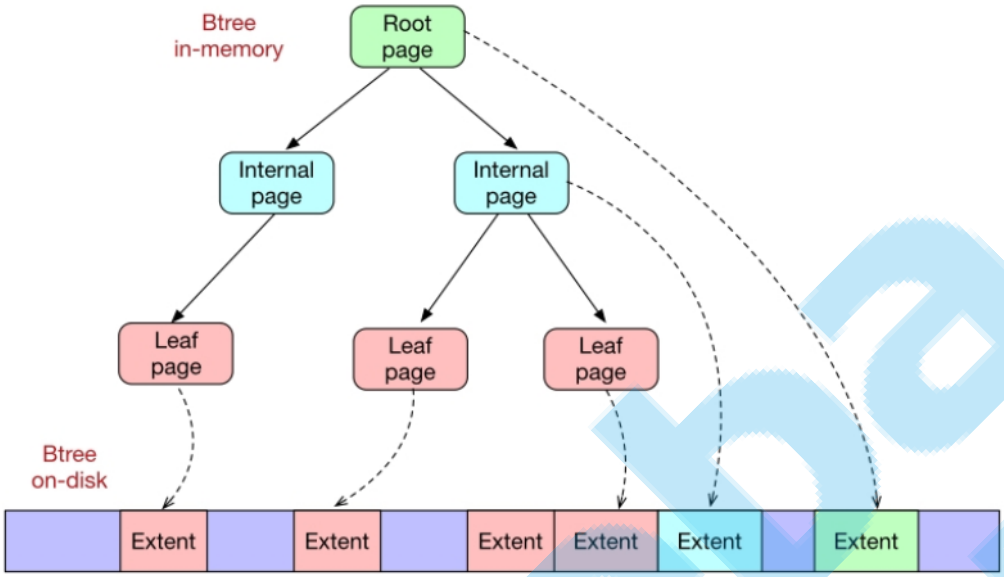
写请求



WiredTiger的写操作会默认写入`cache`,并持久化到`WAL`(Write Ahead Log)，每60s或Log文件达到2G做一次`checkpoint`，产生快照文件。WiredTiger初始化时，恢复至最新的快照状态，然后根据WAL恢复数据，保证数据的完整性。

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为单位与磁道读写。Wiredtiger采用Copy on write的方式管理修改操作（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，而是写入新分配的page，每次checkpoint都会产生一个新的root page。

Btree、Page、Extent



Journaling

为了在数据库宕机保证 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。

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

文件：

- WiredTiger.basecfg: 存储基本配置信息，与ConfigServer有关系
- WiredTiger.lock: 定义锁操作
- 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
- journal: 存储WAL

一次Checkpoint的大致流程如下:

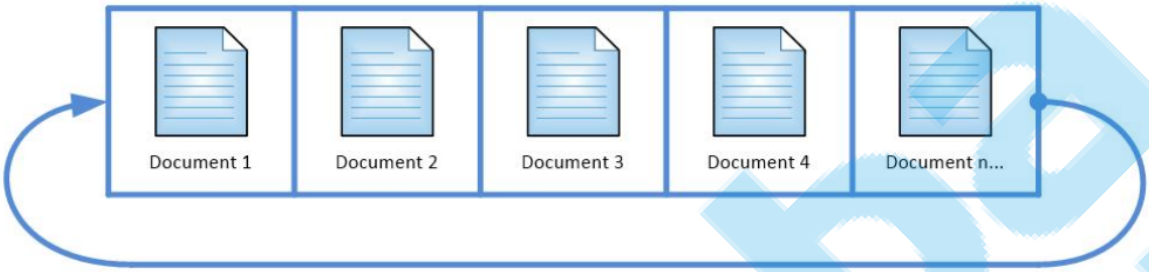
对所有的table进行一次Checkpoint，每个table的Checkpoint的元数据更新至WiredTiger.wt 对WiredTiger.wt进行Checkpoint，将该table Checkpoint的元数据更新至临时文件WiredTiger.turtle.set 将WiredTiger.turtle.set重命名为WiredTiger.turtle。 上述过程如中间失败，Wiredtiger在下次连接初始化时，首先将数据恢复至最新的快照状态，然后根据WAL恢复数据，以保证存储可靠性。

MongoDB集群

Mongodb的集群部署方案有**主从部署、副本集（主备）部署、分片部署、副本集与分片混合部署。**

主从复制原理

MongoDB Oplog是MongoDB Primary和Secondary在复制建立期间和建立完成之后的复制介质，就是Primary中所有的写入操作都会记录到MongoDB Oplog中，然后从库会来主库一直拉取Oplog并应用到自己的数据库中。这里的Oplog是MongoDB local数据库的一个集合，它是Capped collection，通俗意思就是它是固定大小，循环使用的。



OpLog结构

```
{
"ts" : Timestamp(1446011584, 2),
"h" : NumberLong("1687359108795812092"),
"v" : 2,
"op" : "i",
"ns" : "test.nosql",
"o" : { "_id" : ObjectId("563062c0b085733f34ab4129"), "name" : "mongodb",
"score" : "100" }
}

ts：  操作时间，当前timestamp + 计数器，计数器每秒都被重置
h：操作的全局唯一标识
v：oplog版本信息
op：操作类型
    i：插入操作
    u：更新操作
    d：删除操作
    c：执行命令（如createDatabase，dropDatabase）
n：空操作，特殊用途
ns：操作针对的集合
o：操作内容，如果是更新操作
o2：操作查询条件，仅update操作包含该字段
```

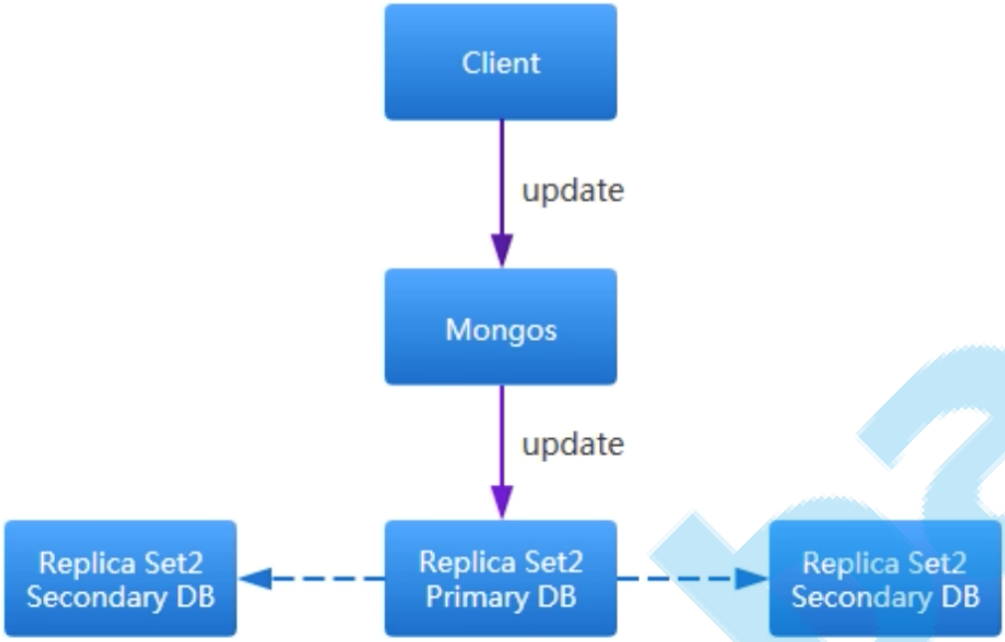
oplog.rs

```
rs001:PRIMARY> use local
rs001:PRIMARY> show tables
me
 # Primary节点写入数据，Secondary通过读取Primary的oplog得到复制信息，开始复制数据并且将复制信息写入到自己的oplog。
oplog.rs
replset.election
replset.minvalid
replset.oplogTruncateAfterPoint
```

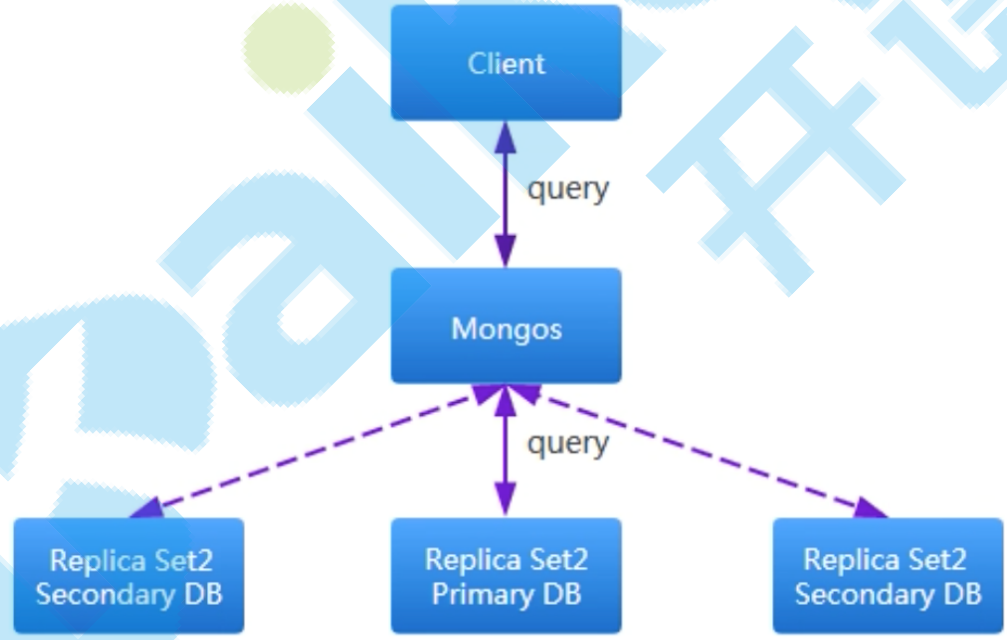
```
startup_log
system.replset
system.rollback.id
# 插入的数据
rs001:PRIMARY> db.oplog.rs.find({"op" : "i"}).pretty()
{
        "ts" : Timestamp(1561102579, 6),
        "t" : NumberLong(1),
        "h" : NumberLong("-6973786105046479584"),
        "v" : 2,
        "op" : "i",
        "ns" : "admin.system.keys",
        "ui" : UUID("bdc5bfc9-0038-4f9d-94dd-94d0e3ac6bff"),
        "wall" : ISODate("2019-06-21T07:36:19.409Z"),
        "o" : {
                "_id" : NumberLong("6704884522506256385"),
                "purpose" : "HMAC",
                "key" : BinData(0,"j0z0uU0XF5IomoMXk8rP8pYKTNo="),
                "expiresAt" : Timestamp(1568878579, 0)
        }
}
{
        "ts" : Timestamp(1561102580, 1),
        "t" : NumberLong(1),
        "h" : NumberLong("2359103001087607398"),
        "v" : 2,
        "op" : "i",
        "ns" : "admin.system.keys",
        "ui" : UUID("bdc5bfc9-0038-4f9d-94dd-94d0e3ac6bff"),
        "wall" : ISODate("2019-06-21T07:36:20.340Z"),
        "o" : {
                "_id" : NumberLong("6704884522506256386"),
                "purpose" : "HMAC",
                "key" : BinData(0,"Mg3YjbVxSWqf2hgTvDkEeIoJSvM="),
                "expiresAt" : Timestamp(1576654579, 0)
        }
}
```

副本集集群

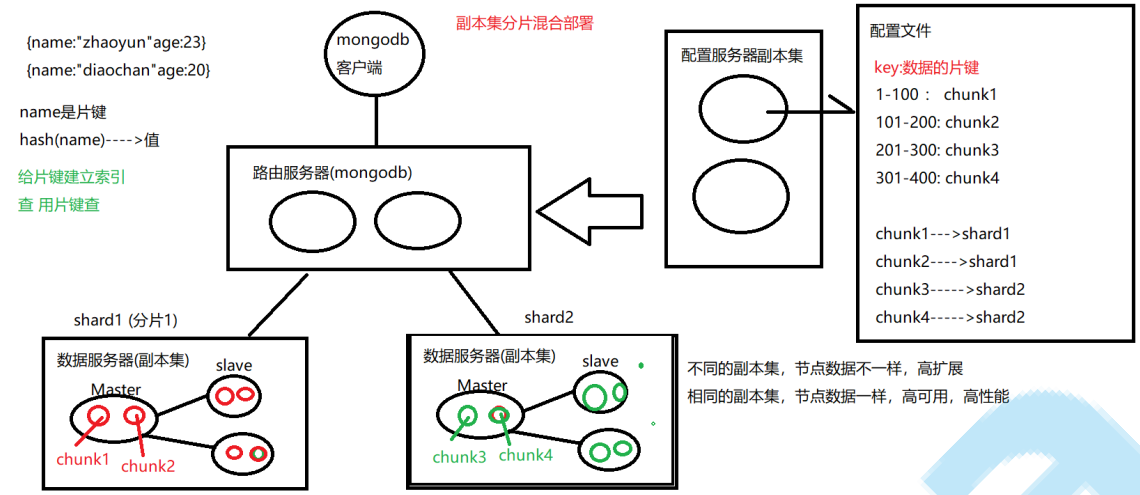
写数据：



读数据



副本集与分片混合部署

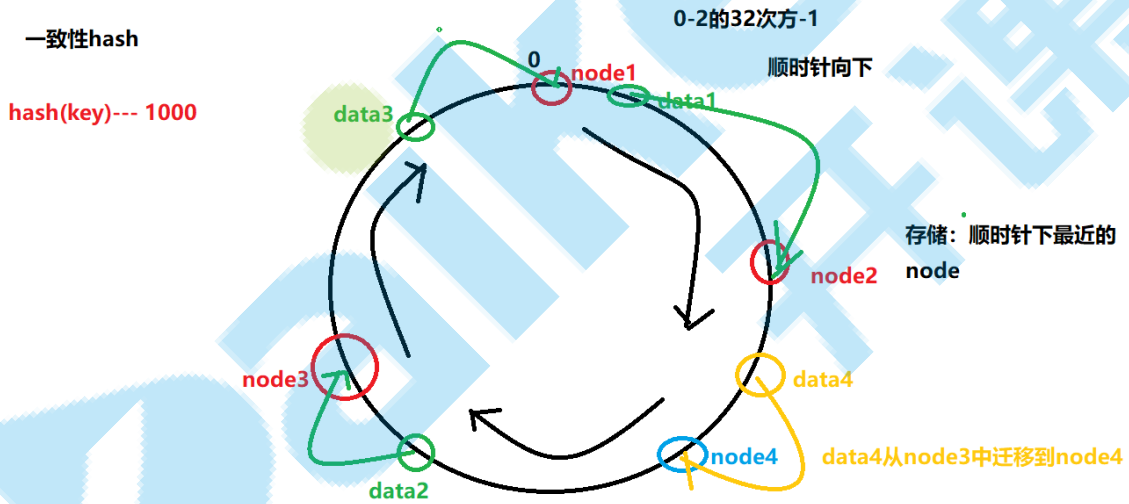


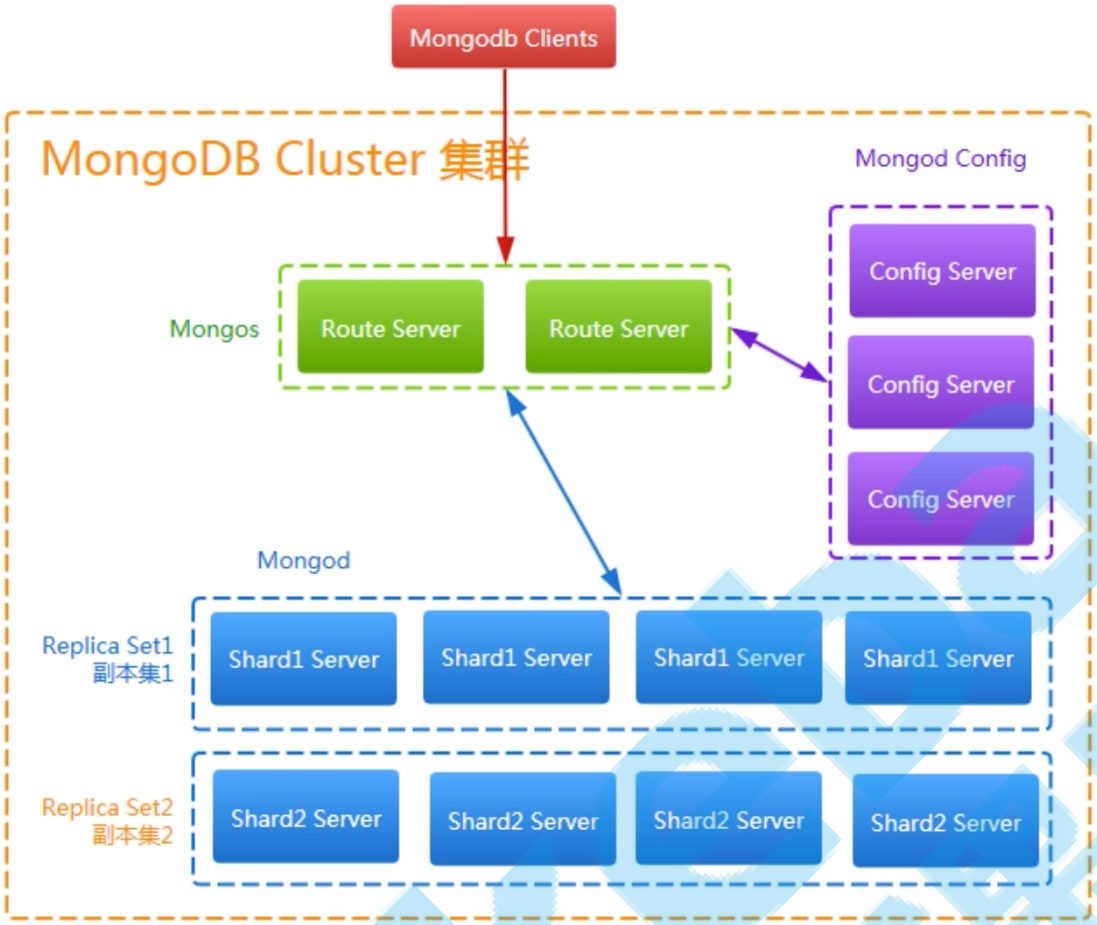
数据层： 数据服务器副本级

配置层： 配置服务器集群

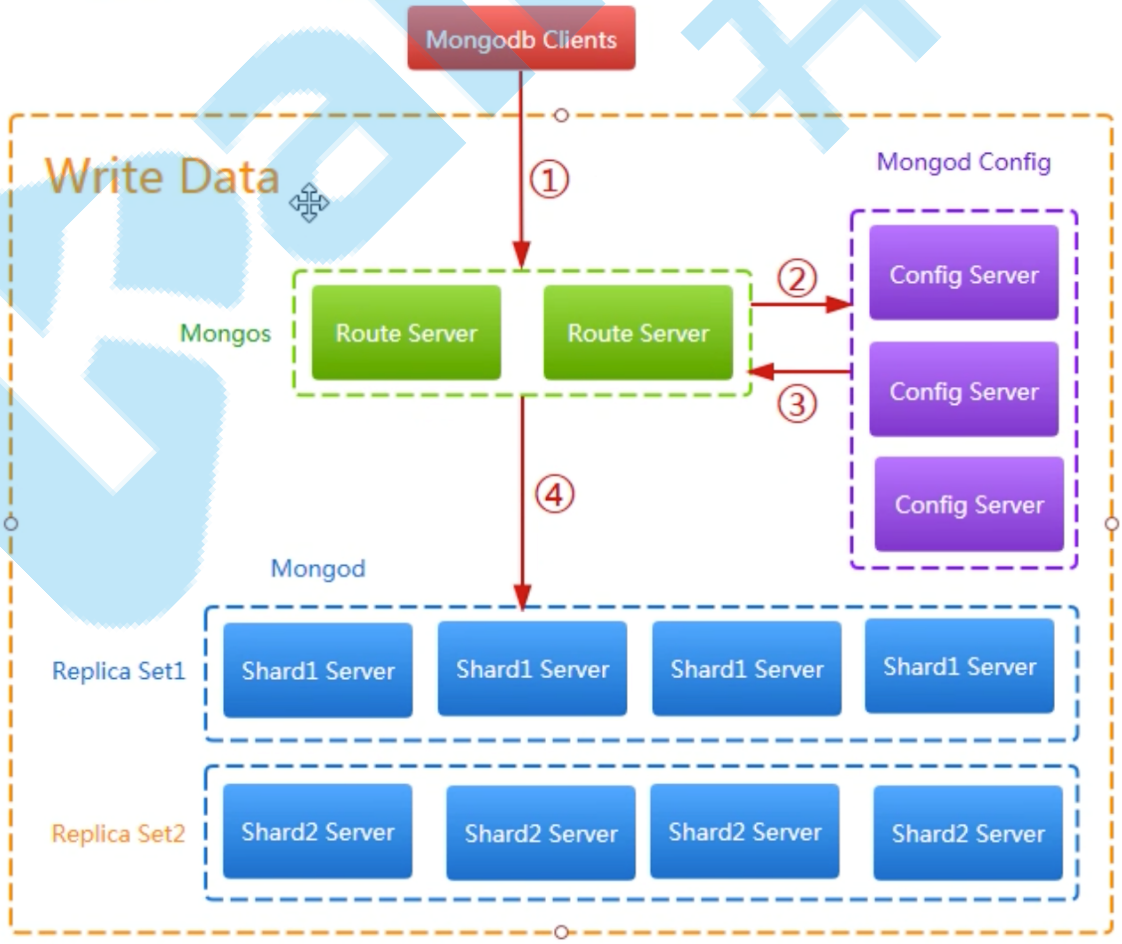
hash(key)------>chunk----->shard

一致性hash：

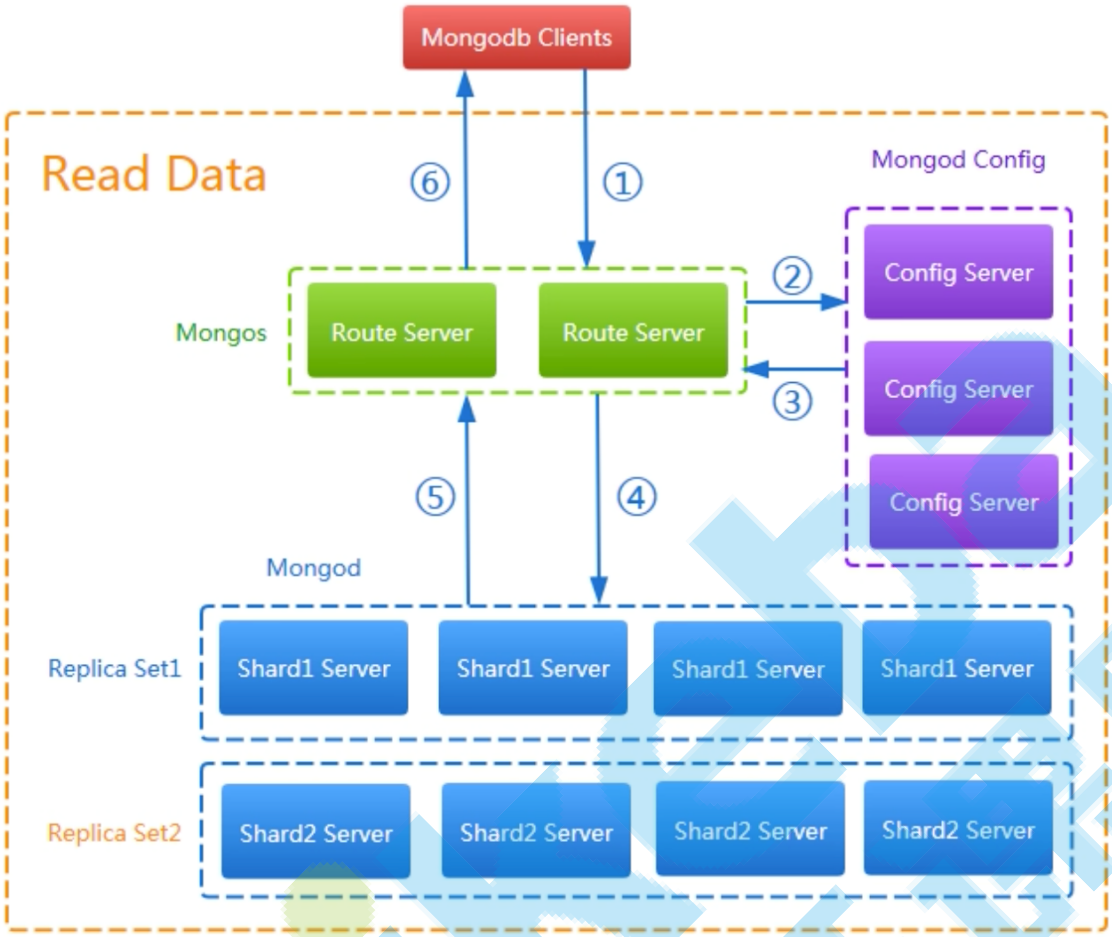


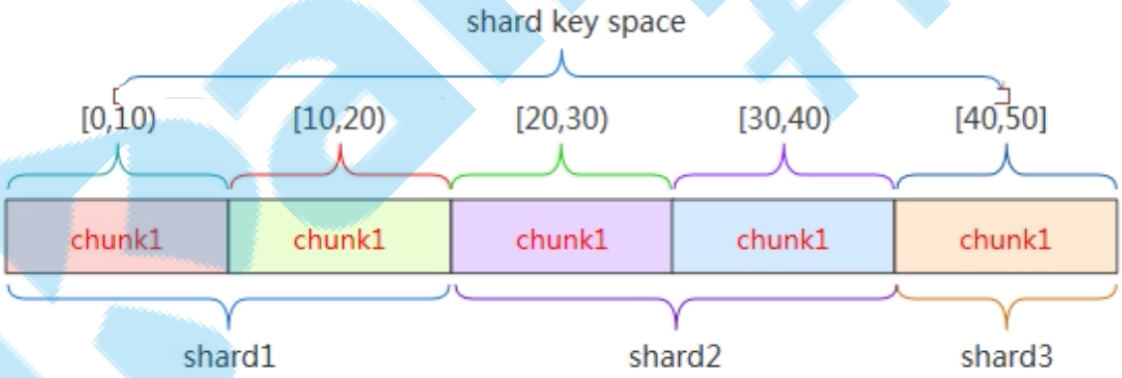


混合部署方式下向MongoDB写数据的流程如图：**



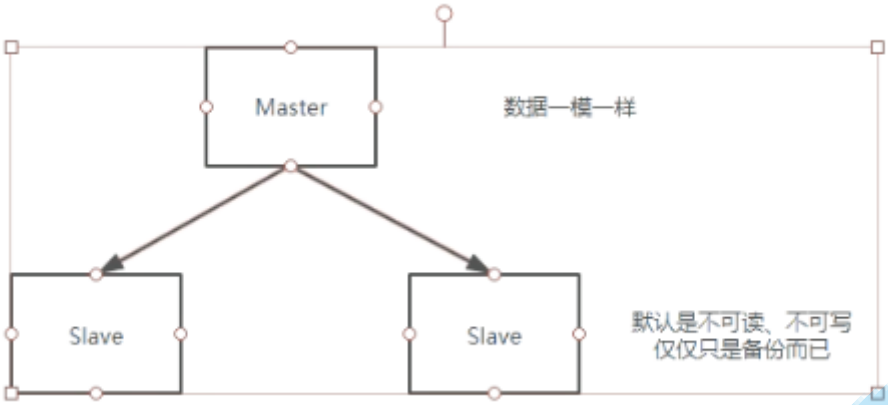
混合部署方式下读MongoDB里的数据流程如图： 按条件查询 查的就是片键 （建立索引）





MongoDB集群搭建

MongoDB主从搭建



纯主从不能高可用，主挂了，则集群挂了，不推荐

新建目录

```
[root@localhost var]# mkdir mongo-ms/master/data -p
[root@localhost var]# mkdir mongo-ms/master/logs -p
[root@localhost var]# mkdir mongo-ms/slave/logs -p
[root@localhost var]# mkdir mongo-ms/slave/data -p
```

主机配置

/var/mongo-ms/master/mongodb.cfg

```
#数据库文件位置
dbpath=/var/mongo-ms/master/data
#日志文件位置
logpath=/var/mongo-ms/master/logs/mongodb.log
# 以追加方式写入日志
logappend=true
# 是否以守护进程方式运行
fork=true
#绑定客户端访问的ip
bind_ip=192.168.24.133
# 默认27017
port=27001
# 主从模式下，指定我自身的角色是主机
master=true
# 主从模式下，从机的地址信息
source=192.168.24.133:27002
```

从机配置

/var/mongo-ms/slave/mongodb.cfg

```
# 数据库文件位置
dbpath=/var/mongo-ms/slave/data
#日志文件位置
logpath=/var/mongo-ms/slave/logs/mongodb.log
# 以追加方式写入日志
```

```
logappend=true
# 是否以守护进程方式运行
fork = true
bind_ip=192.168.24.133
# 默认27017
port = 27002
slave = true
# 主从模式下，从机的地址信息
source=192.168.24.133:27001
```

测试

启动服务

```
mongod -f /var/mongo-ms/master/mongodb.cfg
mongod -f /var/mongo-ms/slave/mongodb.cfg
```

连接测试

```
mongo 192.168.24.133:27001
mongo 192.168.24.133:27002
```

测试命令

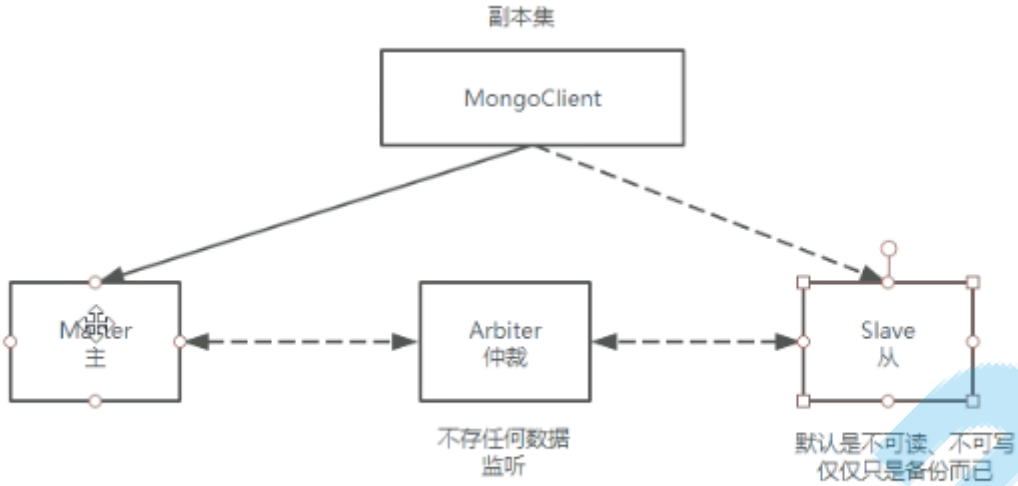
```
db.isMaster()
```

读写分离

MongoDB副本集对读写分离的支持是通过Read Preferences特性进行支持的，这个特性非常复杂和灵活。设置读写分离需要先在从节点SECONDARY 设置

```
rs.slaveOk()
```

MongoDB副本集集群



副本集中有三种角色：主节点、从节点、仲裁节点

仲裁节点不存储数据，主从节点都存储数据。

优点：

主如果宕机，仲裁节点会选举从作为新的主

如果副本集中没有仲裁节点，那么集群的主从切换依然可以进行。

缺点：

如果副本集中拥有仲裁节点，那么一旦仲裁节点挂了，集群中就不能进行主从切换了。

有仲裁节点的副本集

新建目录

```
[root@localhost var]# mkdir mongo-rs/rs01/node1/data -p
[root@localhost var]# mkdir mongo-rs/rs01/node1/logs -p
[root@localhost var]# mkdir mongo-rs/rs01/node2/data -p
[root@localhost var]# mkdir mongo-rs/rs01/node2/logs -p
[root@localhost var]# mkdir mongo-rs/rs01/node3/data -p
[root@localhost var]# mkdir mongo-rs/rs01/node3/logs -p
```

节点1配置

```
# 数据库文件位置
dbpath=/var/mongo-rs/rs01/node1/data
#日志文件位置
logpath=/var/mongo-rs/rs01/node1/logs/mongodb.log
# 以追加方式写入日志
logappend=true
# 是否以守护进程方式运行
fork = true

bind_ip=192.168.24.133
# 默认27017
port = 27003
#注意：不需要显式的去指定主从，主从是动态选举的
```

```
#副本集集群，需要指定一个名称，在一个副本集下，名称是相同的
replSet=rs001
```

节点2配置

```
# 数据库文件位置
dbpath=/var/mongo-rs/rs01/node2/data
#日志文件位置
logpath=/var/mongo-rs/rs01/node2/logs/mongodb.log
# 以追加方式写入日志
logappend=true
# 是否以守护进程方式运行
fork = true
bind_ip=192.168.24.133
# 默认27017
port = 27004
#注意：不需要显式的去指定主从，主从是动态选举的
#副本集集群，需要指定一个名称，在一个副本集下，名称是相同的
replSet=rs001
```

节点3配置

```
# 数据库文件位置
dbpath=/var/mongo-rs/rs01/node3/data
#日志文件位置
logpath=/var/mongo-rs/rs01/node3/logs/mongodb.log
# 以追加方式写入日志
logappend=true
# 是否以守护进程方式运行
fork = true
bind_ip=192.168.24.133
# 默认27017
port = 27005
#注意：不需要显式的去指定主从，主从是动态选举的
#副本集集群，需要指定一个名称，在一个副本集下，名称是相同的
replSet=rs001
```

启动副本集节点

```
mongod -f /var/mongo-rs/rs01/node1/mongodb.cfg
mongod -f /var/mongo-rs/rs01/node2/mongodb.cfg
mongod -f /var/mongo-rs/rs01/node3/mongodb.cfg
```

配置主备和仲裁

需要登录到mongodb的客户端进行配置主备和仲裁角色。

注意创建dbpath和logpath

```
mongo 192.168.24.133:27003

use admin

cfg={_id:"rs001",members: [
{_id:0,host:"192.168.24.133:27003",priority:2}, #主的可能性大
{_id:1,host:"192.168.24.133:27004",priority:1},
{_id:2,host:"192.168.24.133:27005",arbiterOnly:true}
]}

rs.initiate(cfg);
```

说明:

cfg中的_id的值是【副本集名称】

priority：数字越大，优先级越高。优先级最高的会被选举为主库

arbiterOnly:true，如果是仲裁节点，必须设置该参数

测试

```
rs.status()
```

```
qnzs:PRIMARY> rs.status()
{
        "set" : "qnzs",
        "date" : ISODate("2016-09-27T02:07:48.507Z"),
        "myState" : 1,
        "term" : NumberLong(3),
        "heartbeatIntervalMillis" : NumberLong(2000),
        "members" : [
                {
                        "_id" : 0,
                        "name" : "172.17.116.18:27017",
                        "health" : 1,
                        "state" : 1,
                        "stateStr" : "PRIMARY",
                        "uptime" : 37,
                        "optime" : {
                                "ts" : Timestamp(1474942042, 2),
                                "t" : NumberLong(3)
                        },
                        "optimeDate" : ISODate("2016-09-27T02:07:22Z"),
                        "electionTime" : Timestamp(1474942042, 1),
                        "electionDate" : ISODate("2016-09-27T02:07:22Z"),
                        "configVersion" : 1,
                        "self" : true
                },
                {
```

无仲裁副本集

和有仲裁的副本集基本上完全一样，只是在admin数据库下去执行配置的时候，不需要指定优先级和仲裁节点。这种情况，如果节点挂掉，那么他们都会进行选举。

新建目录

```
[root@localhost var]# mkdir mongo-rs/rs02/node1/data -p
[root@localhost var]# mkdir mongo-rs/rs02/node1/logs -p
[root@localhost var]# mkdir mongo-rs/rs02/node2/data -p
[root@localhost var]# mkdir mongo-rs/rs02/node2/logs -p
[root@localhost var]# mkdir mongo-rs/rs02/node3/data -p
[root@localhost var]# mkdir mongo-rs/rs02/node3/logs -p
```

节点1配置

```
# 数据库文件位置
dbpath=/var/mongo-rs/rs02/node1/data
#日志文件位置
logpath=/var/mongo-rs/rs02/node1/logs/mongodb.log
# 以追加方式写入日志
logappend=true
# 是否以守护进程方式运行
fork = true
bind_ip=192.168.24.133
# 默认27017
port = 27006
#注意：不需要显式的去指定主从，主从是动态选举的
#副本集集群，需要指定一个名称，在一个副本集下，名称是相同的
replSet=rs002
```

节点2配置

```
# 数据库文件位置
dbpath=/var/mongo-rs/rs02/node2/data
#日志文件位置
logpath=/var/mongo-rs/rs02/node2/logs/mongodb.log
# 以追加方式写入日志
logappend=true
# 是否以守护进程方式运行
fork = true
bind_ip=192.168.24.133
# 默认27017
port = 27007
#注意：不需要显式的去指定主从，主从是动态选举的
#副本集集群，需要指定一个名称，在一个副本集下，名称是相同的
replSet=rs002
```

节点3配置

```
# 数据库文件位置
dbpath=/var/mongo-rs/rs02/node3/data
#日志文件位置
logpath=/var/mongo-rs/rs02/node3/logs/mongodb.log
# 以追加方式写入日志
```

```
logappend=true
# 是否以守护进程方式运行
fork = true
bind_ip=192.168.24.133
# 默认27017
port = 27008
#注意：不需要显式的去指定主从，主从是动态选举的
#副本集集群，需要指定一个名称，在一个副本集下，名称是相同的
replSet=rs002
```

```
启动：
mongod -f /var/mongo-rs/rs02/node1/mongodb.cfg
mongod -f /var/mongo-rs/rs02/node2/mongodb.cfg
mongod -f /var/mongo-rs/rs02/node3/mongodb.cfg

mongo 192.168.24.133:27006

use admin

cfg={_id:"rs002",members: [
{_id:0,host:"192.168.24.133:27006"},
{_id:1,host:"192.168.24.133:27007"},
{_id:2,host:"192.168.24.133:27008"}
]}

rs.initiate(cfg);
```

MongoDB混合方式集群

数据服务器配置（副本集）

在副本集中每个数据节点的mongodb.cfg配置文件【追加】以下内容（**仲裁节点除外**）：

```
shardsvr=true
```

配置服务器配置(先启动配置集再启动数据副本集)

配置两个配置服务器，配置信息如下，端口和path单独指定：

```
# 数据库文件位置
dbpath=/var/mongo-conf/node1/data
#日志文件位置
logpath=/var/mongo-conf/node1/logs/mongodb.log
# 以追加方式写入日志
logappend=true
# 是否以守护进程方式运行
fork = true
bind_ip=192.168.24.133
# 默认28001
port = 28001
# 表示是一个配置服务器
configsvr=true
#配置服务器副本集名称
replSet=configsvr
```

```
# 数据库文件位置
dbpath=/var/mongo-conf/node2/data
#日志文件位置
logpath=/var/mongo-conf/node2/logs/mongodb.log
# 以追加方式写入日志
logappend=true
# 是否以守护进程方式运行
fork = true
bind_ip=192.168.24.133
# 默认28001
port = 28002
# 表示是一个配置服务器
configsvr=true
#配置服务器副本集名称
replSet=configsvr
```

注意创建dbpath和logpath

```
[root@localhost var]# mkdir mongo-conf/node1/data -p
[root@localhost var]# mkdir mongo-conf/node1/logs -p

[root@localhost var]# mkdir mongo-conf/node2/data -p
[root@localhost var]# mkdir mongo-conf/node2/logs -p

#先启动配置集再启动数据副本集
#启动配置集
mongod -f /var/mongo-conf/node1/mongodb.cfg
mongod -f /var/mongo-conf/node2/mongodb.cfg
#启动数据副本集--shard1
mongod -f /var/mongo-rs/rs01/node1/mongodb.cfg
mongod -f /var/mongo-rs/rs01/node2/mongodb.cfg
mongod -f /var/mongo-rs/rs01/node2/mongodb.cfg

#启动数据副本集--shard2
mongod -f /var/mongo-rs/rs02/node1/mongodb.cfg
mongod -f /var/mongo-rs/rs02/node2/mongodb.cfg
mongod -f /var/mongo-rs/rs02/node3/mongodb.cfg
```

配置副本集

```
mongo 192.168.24.133:28001

use admin

cfg={_id:"configsvr",members: [
{_id:0,host:"192.168.24.133:28001"},
{_id:1,host:"192.168.24.133:28002"}
]}

rs.initiate(cfg);
```

路由服务器配置

```
configdb=configsvr/192.168.24.133:28001,192.168.24.133:28002
#日志文件位置
logpath=/var/mongo-router/node01/logs/mongodb.log
# 以追加方式写入日志
logappend=true
# 是否以守护进程方式运行
fork = true
bind_ip=192.168.24.133
# 默认28001
port=30000
```

路由服务器启动（**注意这里是mongos命令而不是mongod命令**）

```
mongos -f /var/mongo-router/node1/mongodb.cfg
```

关联切片和路由

登录到路由服务器中，执行关联切片和路由的相关操作。

```
mongo  192.168.24.133:30000
#查看shard相关的命令
sh.help()
```

```
sh.addShard("切片名称/地址")
#数据副本集
sh.addShard("rs001/192.168.24.133:27003");
sh.addShard("rs002/192.168.24.133:27006");

use kkb
sh.enableSharding("kkb");
#新的集合
sh.shardCollection("kkb.citem",{name:"hashed"});

for(var i=1;i<=1000;i++) db.citem.insert({name:"iphone"+i,num:i});

#分片效果
mongos> db.citem.count()
1000
mongo 192.168.24.133:27003
use kkb
rs001:PRIMARY> db.citem.count()
516
#从库
mongo 192.168.24.133:27004
use kkb
db.getMongo().setSlaveOk() # 设置主从读写分离
rs001:SECONDARY> db.citem7.count()
516
```

```
mongo 192.168.24.133:27006
use kkb
db.citem.count()
484
```