课程主题

mongodb原理分析、集群搭建专题

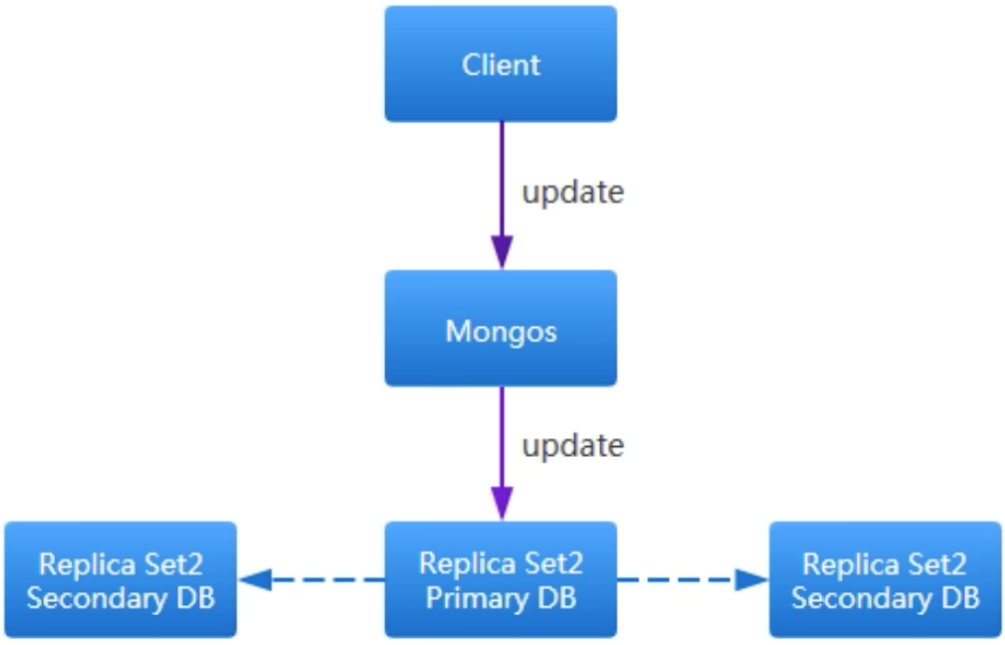
课程目标

- 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的常用命令
- 掌握mongodb的spring data mongodb应用
- 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

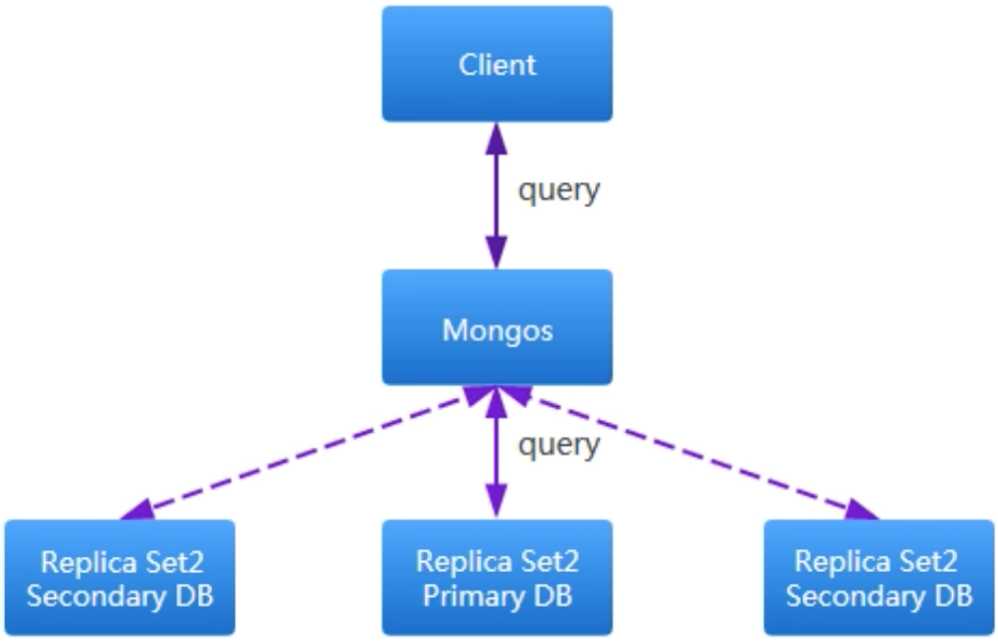
Mongodb的部署方案有**单机部署、主从部署、副本集（主备）部署、分片部署、副本集与分片混合部署。**

副本集集群

对于副本集集群，又有主和从两种角色，写数据和读数据也是不同，写数据的过程是只写到主结点中，由主 结点以异步的方式同步到从结点中：



而读数据则只要从任一结点中读取，具体到哪个结点读取是可以指定的：



副本集与分片混合部署

Mongodb的集群部署方案有三类角色：**实际数据存储节点，配置文件存储节点和路由接入节点。**

- **实际数据存储节点**的作用就是存储数据，
- **路由接入节点**的作用是在分片的情况下起到负载均衡的作用。
- **存储配置存储节点**的作用其实存储的是片键与chunk 以及chunk 与server 的映射关系，用上面的数据表 示的配置结点存储的数据模型如下表：

map1

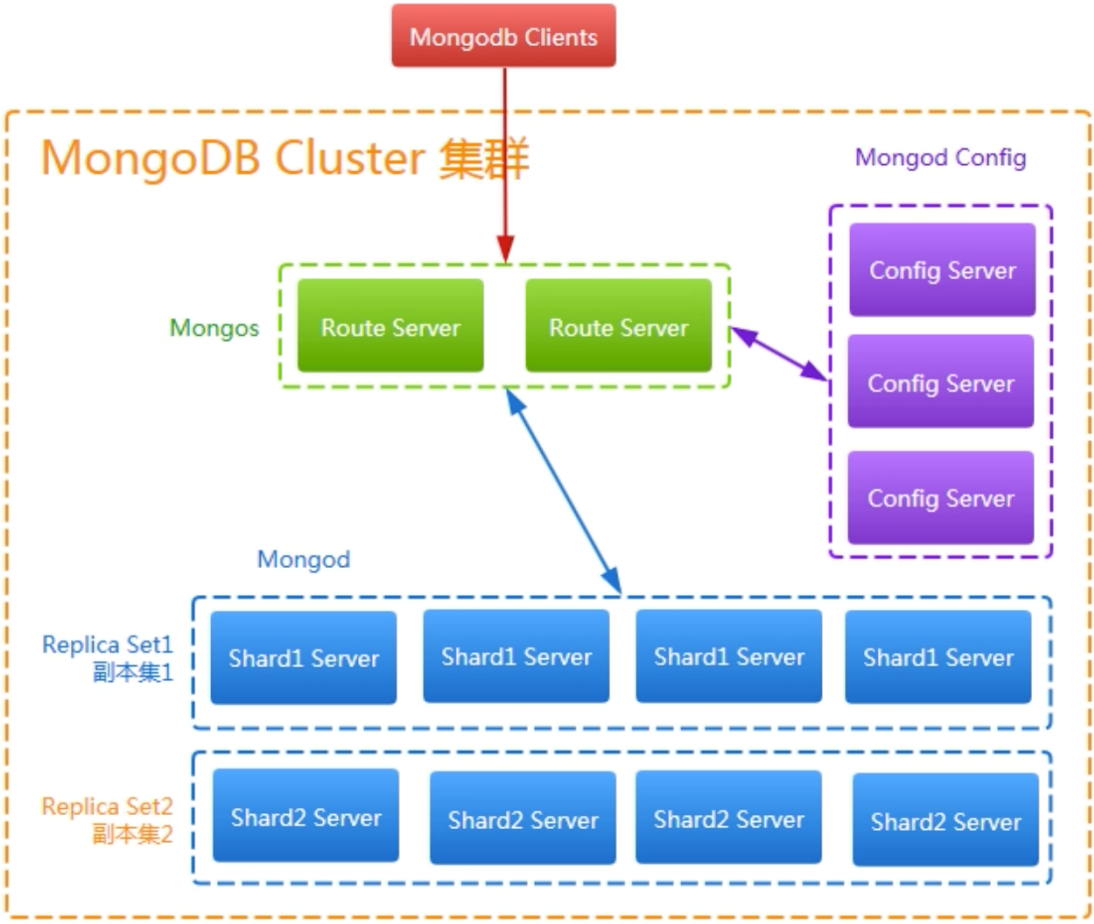
key range	chunk
[0,10}	chunk1
[10,20}	chunk2
[20,30}	chunk3
[30,40}	chunk4
[40,50}	chunk5

map2

chunk	shard
chunk1	shard1
chunk2	shard2
chunk3	shard3
chunk4	shard4
chunk5	shard5

MongoDB的客户端直接与路由节点相连，从配置节点上查询数据，根据查询结果到实际的存储节点上查询和存储数据。

副本集与分片混合部署方式如图：

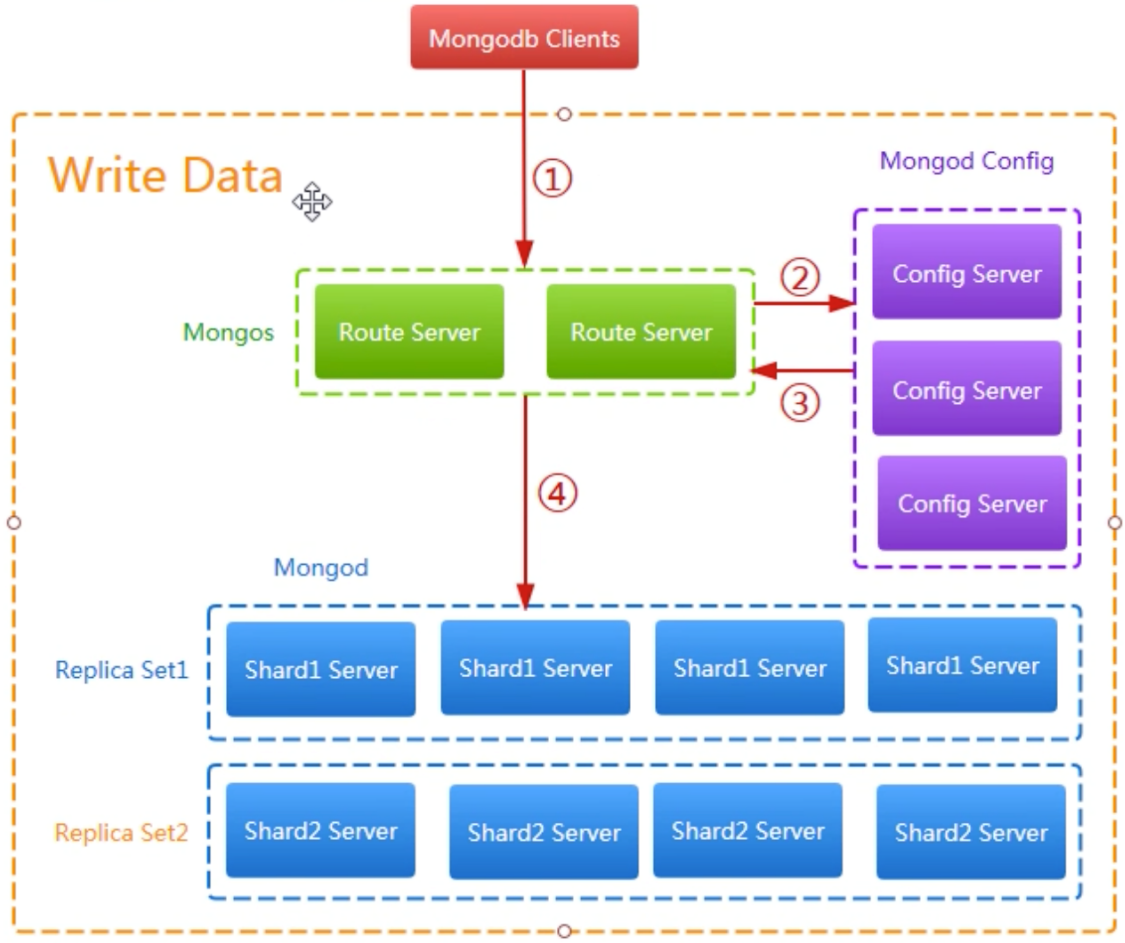


相同的副本集中的节点存储的数据是一样的，副本集中的节点是分为主节点、从节点、仲裁节点（非必须）三种角色。【这种设计方案的目的，主要是为了高性能、高可用、数据备份。】

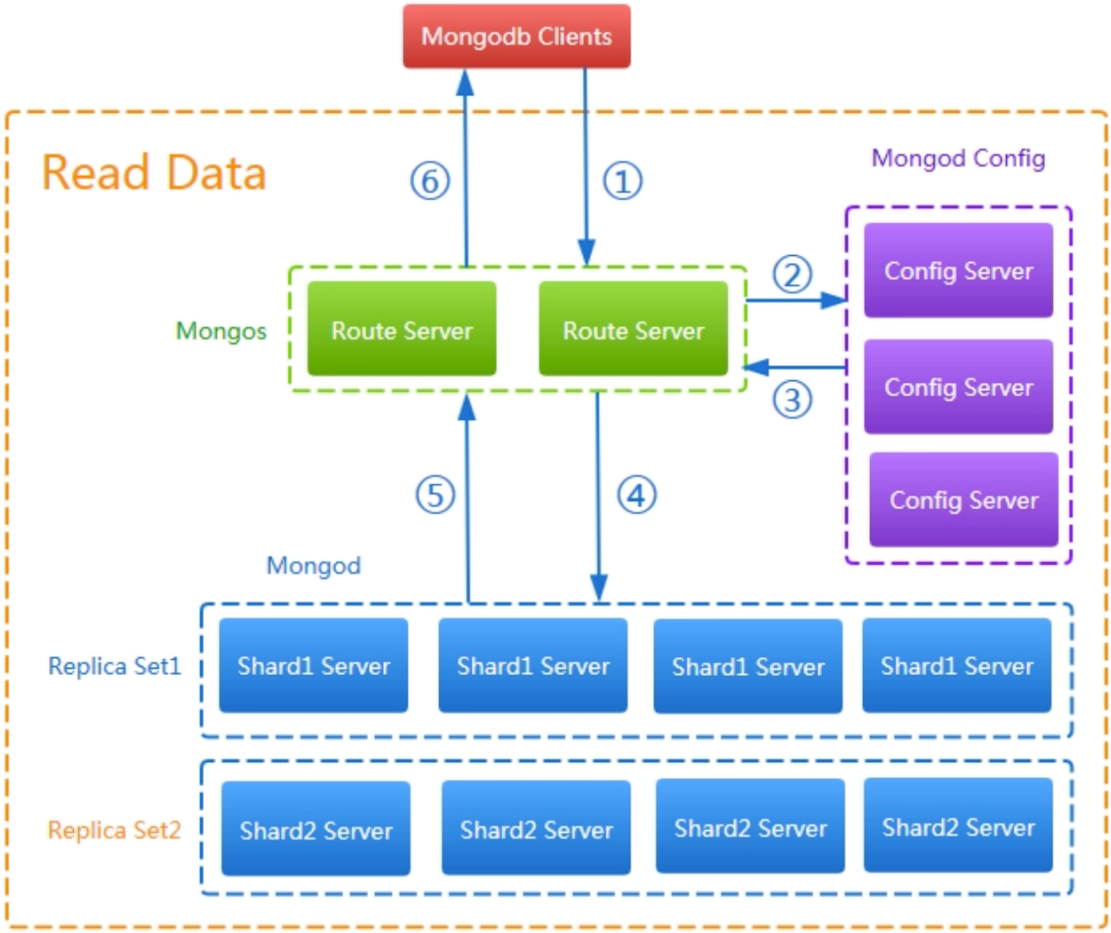
不同的副本集中的节点存储的数据是不一样，【这种设计方案，主要是为了解决高扩展问题，理论上是可以无限扩展的。】

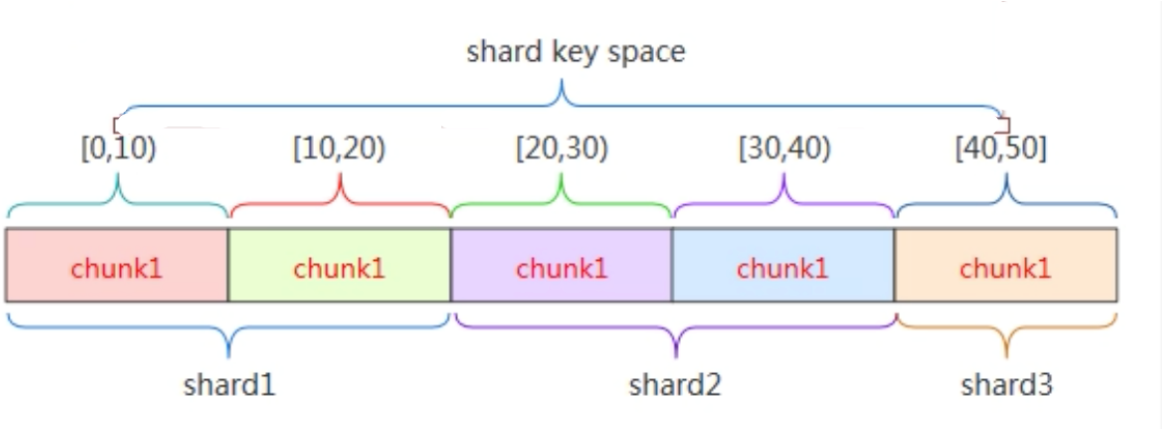
每一个副本集可以看成一个shard（分片），多个副本集共同组成一个逻辑上的大数据节点。通过对shard上面进行逻辑分块chunk（块），每个块都有自己存储的数据范围，所以说客户端请求存储数据的时候，会去读取config server中的映射信息，找到对应的chunk（块）存储数据。

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的应用场景和不适用场景

适用场景

更高的写入负载

默认情况下，MongoDB更侧重高数据写入性能，而非事务安全，MongoDB很适合业务系统中有大量"低价值"数据的场景。但是应当避免在高事务安全性的系统中使用MongoDB，除非能从架构设计上保证事务安全。

高可用性

MongoDB的复副集(Master-Slave)配置非常简洁方便，此外，MongoDB可以快速响应的处理单节点故障，自动、安全的完成故障转移。这些特性使得MongoDB能在一个相对不稳定（如云主机）的环境中，保持高可用性。

数据量很大或者未来会变得很大

依赖数据库(MySQL)自身的特性，完成数据的扩展是较困难的事，**在MySQL中，当一个单表达到5-10GB时会出现明显的性能降级**，此时需要通过数据的水平和垂直拆分、库的拆分完成扩展，使用MySQL通常需要借助驱动层或代理层完成这类需求。而MongoDB内建了多种数据分片的特性，可以很好的适应大数据量的需求。

基于位置的数据查询

MongoDB支持二维空间索引，因此可以快速及精确的从指定位置获取数据。

表结构不明确，且数据在不断变大

在一些传统RDBMS中，增加一个字段会锁住整个数据库/表，或者在执行一个重负载的请求时会明显造成其它请求的性能降级。通常发生在数据表大于1G的时候（当大于1TB时更甚）。 因MongoDB是文档型数据库，为非结构货的文档增加一个新字段是很快速的操作，并且不会影响到已有数据。另外一个好处当业务数据发生变化时，是将不在需要由DBA修改表结构。

没有DBA支持

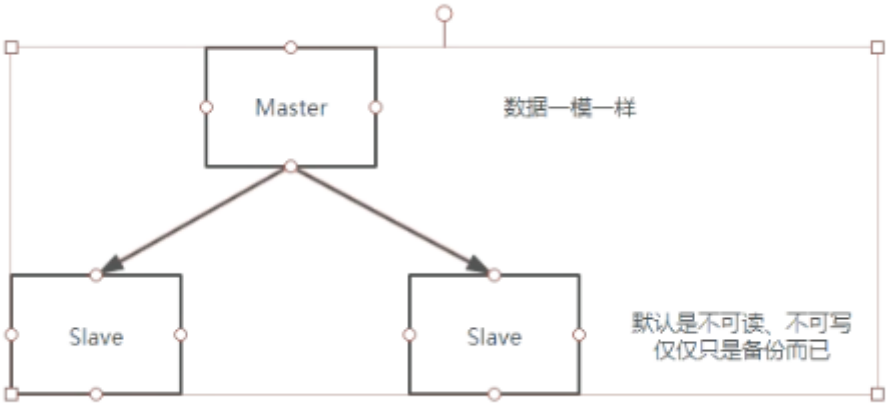
如果没有专职的DBA，并且准备不使用标准的关系型思想（结构化、连接等）来处理数据，那么MongoDB将会是你的首选。MongoDB对于对像数据的存储非常方便，类可以直接序列化成JSON存储到MongoDB中。 但是需要先了解一些最佳实践，避免当数据变大后，由于文档设计问题而造成的性能缺陷。

不适用场景

在某些场景下，MongoDB作为一个非关系型数据库有其局限性。MongoDB不支持事务操作，所以需要用到事务的应用建议不用MongoDB，另外MongoDB目前不支持join操作，需要复杂查询的应用也不建议使用MongoDB。

MongoDB主从搭建

MongoDB的主从集群，其实官方已经不推荐了，但是要理解主从集群的一些特性：**默认从机是不可操作的，只是作为数据备份的。如果需要从机对外提供读的操作，需要单独发送指令。**



伪分布式搭建：在同一台机器，使用多个不同的端口，去启动多个实例。组成一个分布式系统。

真正的分布式搭建：在不同机器，使用相同的端口，分别启动实例。如果是真正的分布式搭建，一定要保证网络畅通和防火墙问题。

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 192.168.10.135:27001
mongo 192.168.10.135:27002
```

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。

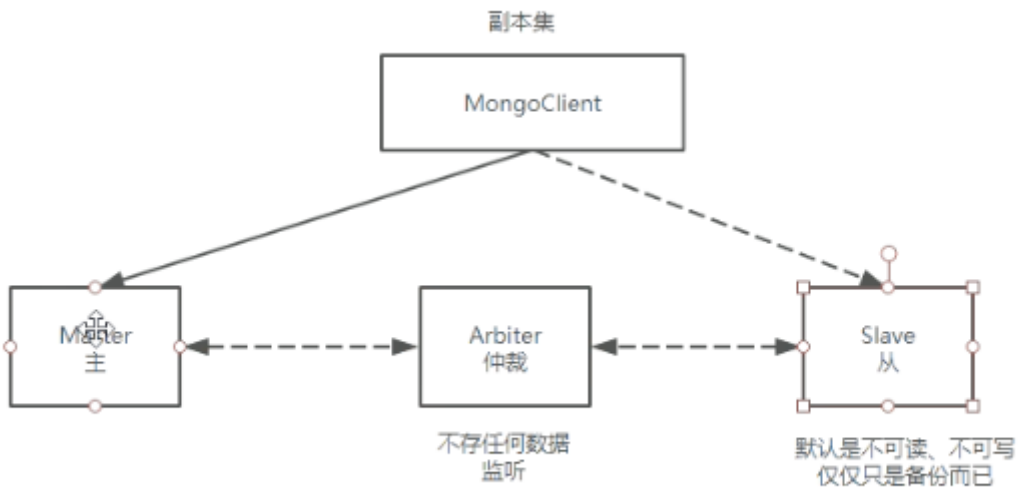
优点：

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。



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},
{_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()
```

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
```

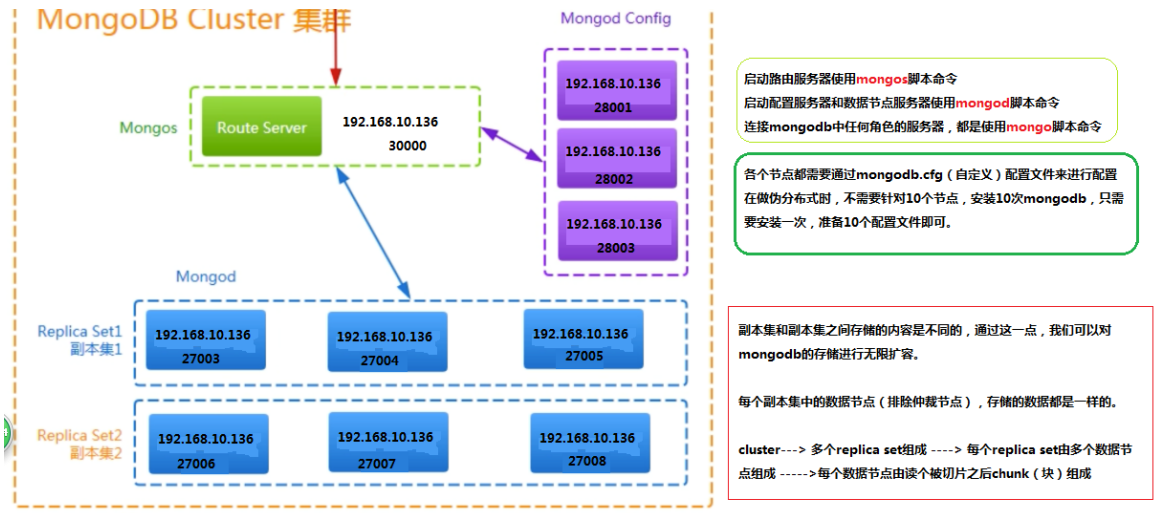
节点3配置

```
# 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

部署图



数据服务器配置

副本集1的配置

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**）：

```
# 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

shardsvr=true
```

```
# 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

shardsvr=true
```

配置服务器配置

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

mongod -f /var/mongo-conf/node1/mongodb.cfg
mongod -f /var/mongo-conf/node2/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")

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
db.citem.count()
516
mongo 192.168.24.133:27006
use kkb
db.citem.count()
484
```