MongoDB 运维及开发

第一章：MySQL 到 MongoDB快速转型

1. MongoDB的简介

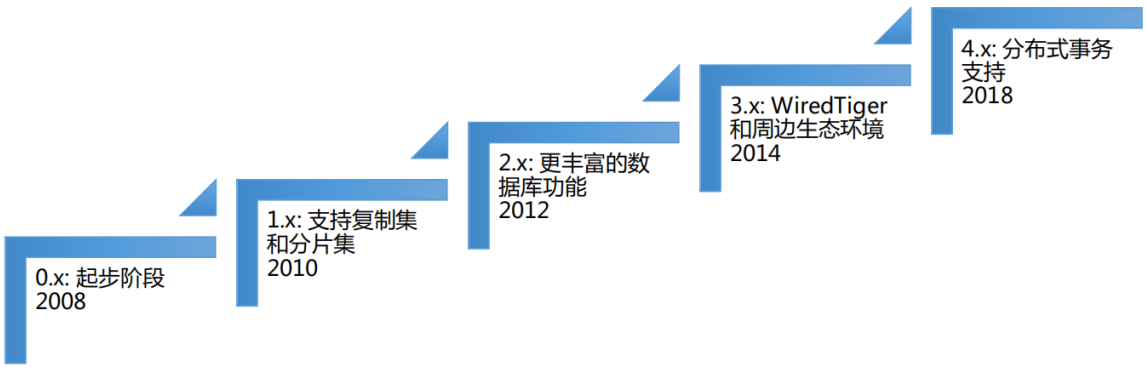
1.1 MongoDB趋势及未来展望

MongoDB：MongoDB趋势

1.2 对于MongoDB的认识Q&A

Q	A
什么是 MongoDB?	一个以 JSON 为数据模型的文档数据库
为什么叫文档数据库?	文档来自于“JSON Document”，并非我们一般理解的 PDF，WORD
谁开发 MongDB?	上市公司 MongoDB Inc.，总部位于美国纽约。
主要用途有哪些?	OLTP\OLAP数据库，类似于 Oracle, MySQL,海量数据处理，数据平台。
主要特点是什么?	无模式或可选。友好的JSON数据模型，开发方便。
MongoDB 是免费的吗?	MongoDB 有两个发布版本：社区版和企业版。企业版基于商业协议，需付费。

1.3 MongoDB 版本变迁



1.4 MongoDB vs. RDBMS

功能	MongoDB	RDBMS
数据模型	JSON	Relational
数据库类型	OLTP/OLAP	OLTP/OLAP
CRUD 操作	MQL/SQL	SQL/SQLX
高可用	原生Replica-Set	Cluster、中间件
横向扩展能力	原生MSC	分片、中间件
索引支持	B-Tree、F-text、GIS、multikey、HASH、TTL	B-Tree
开发难度	easy	hard
数据容量	无理论上限	千万、亿
扩展方式	垂直扩展+水平扩展	垂直扩展

//存储引擎比较

	MySQL InnoDB	MySQL NDB	Oracle	MongoDB MAPI	MongoDB WiredTiger
事务	YES	YES	YES	NO	NO
锁粒度	ROW-level	ROW-level	ROW-level	Collection-level	Document-level
Geospatial	YES	YES	YES	YES	YES
MVCC	YES	NO	YES	NO	NO
Replication	YES	YES	YES	YES	YES
外键	YES	YES(From 7.3)	YES	NO	NO
数据库集群	NO	YES	YES	YES	YES
B-TREE索引	YES	YES	YES	YES	YES
全文检索	YES	NO	YES	YES	YES
数据压缩	YES	NO	YES	NO	YES
存储限制	64TB	384EB	NO	NO	NO
表分区	YES	YES	YES	YES（分片）	YES（分片）

1.5 MongoDB vs. MySQL逻辑结构对比

MySQL	MongoDB
database	database
table	collection
row	document

RDBMS		MongoDB
Table, View	➜	Collection
Row	➜	Document
Index	➜	Index
Join	➜	Embedded Document
Foreign Key	➜	Reference
Partition	➜	Shard

2.MongoDB 特色及优势

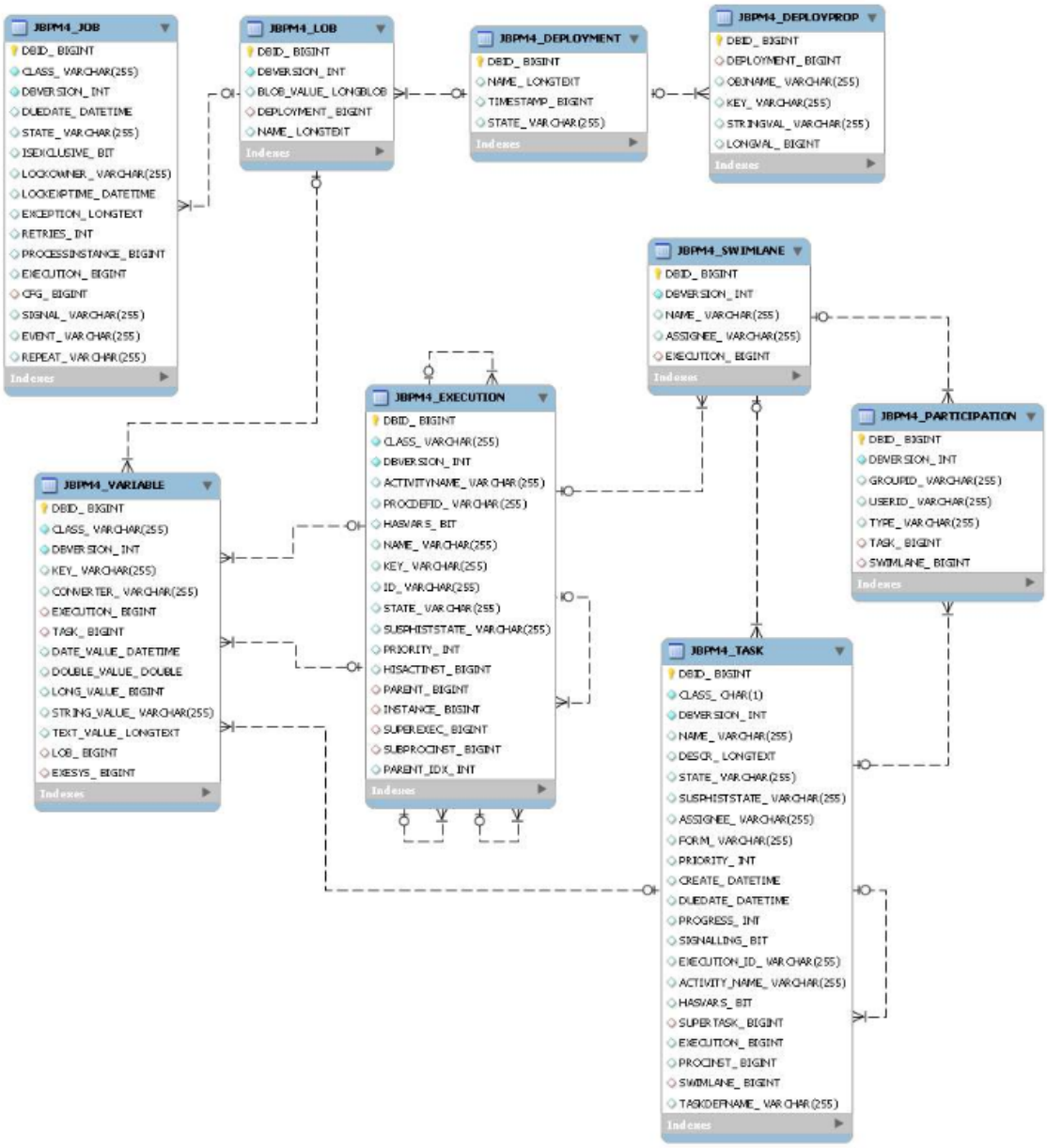
2.1 MongoDB 优势：面向开发者的易用+高效数据库

简单直观:
以自然的方式来建模，以直观的方式来与数据库交互

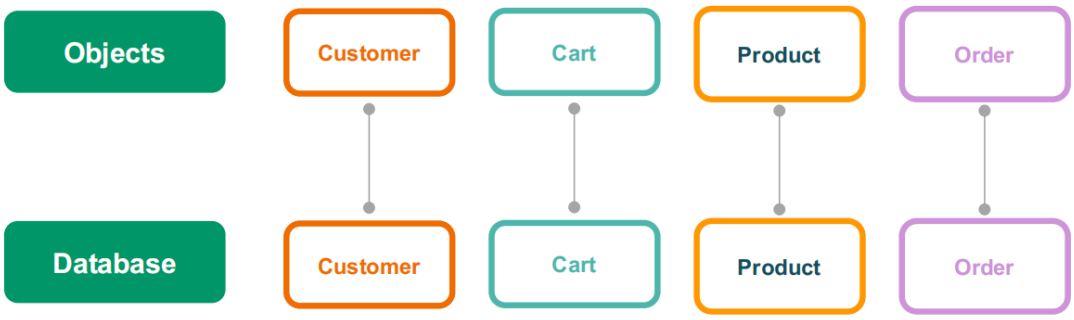
结构灵活:
弹性模式从容响应需求的频繁变化

快速开发:
做更多的事，写更少的代码

SQL模型：错综复杂

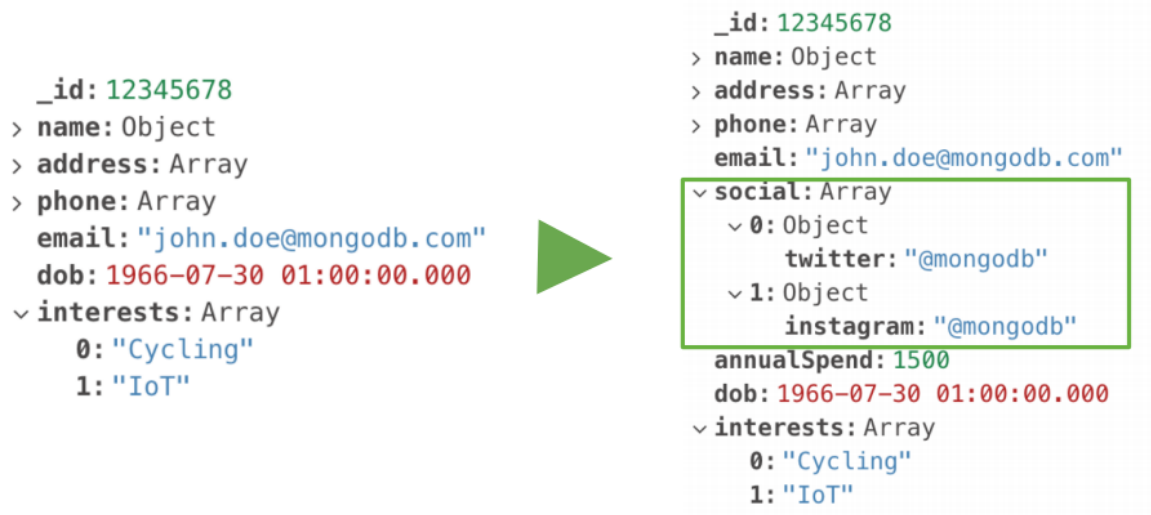


JSON模型：条理清楚



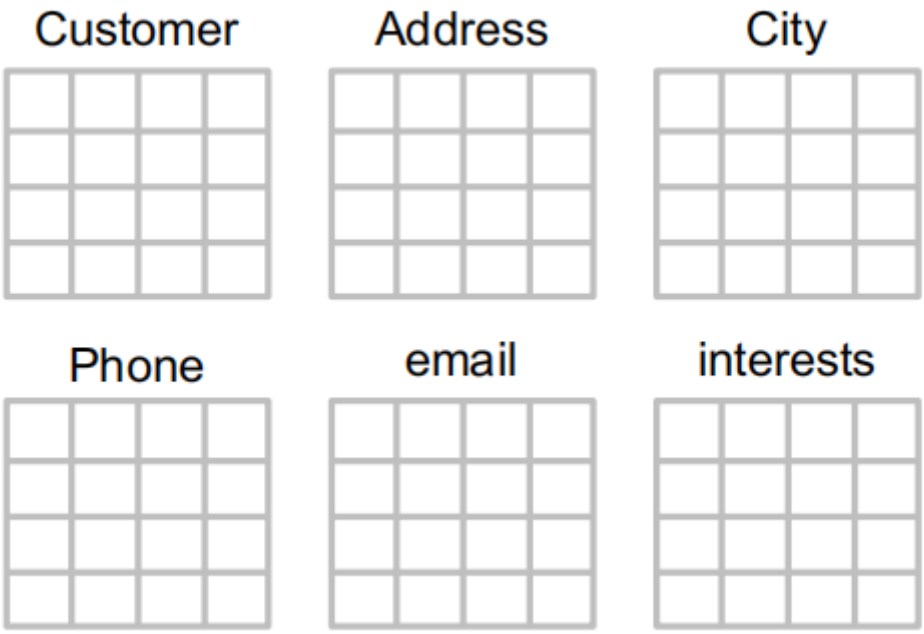
快速响应业务变化：

```
a. 多类型:
同一个Collection中,可以包含不同字段（类型）的文档对象.
b. 更灵活:
线上修改结构，应用与数据库均无须下线。
```



简洁的开发模式：

```
a. 数据库引擎只需要在一个存储区读写.
b. 反范式、无关联的组织极大优化查询速度.
c. 程序API自然，开发快速.
```




```
  _id: 12345678
> name: Object
> address: Array
> phone: Array
  email: "john.doe@mongodb.com"
  dob: 1966-07-30 01:00:00.000
v interests: Array
    0: "Cycling"
    1: "IoT"
```

SQL插入数据代码量：

```python
import mysql.connector
from mysql.connector import errorcode

def addUser(connection, user):
  cursor = connection.cursor()

  customerInsert = (
      "INSERT INTO customer (first_name, last_name, email, "
      "DOB, annual_spend) VALUES "
      "(%(first)s, %(last)s, %(email)s, %(dob)s, %
(spend)s)")

  customerData = {
      'first': user['name']['first'],
      'last': user['name']['second'],
      'email': user['email'],
      'dob': user['dob'],
      'spend': user['annualSpend']
  }

  cursor.execute(customerInsert, customerData)
  customerId = cursor.lastrowid

  cityQuery = ("SELECT city_id FROM city WHERE "
      "city = %(city)s")
  for address in user['address']:
      cursor.execute(cityQuery, {'city': address['city']})
      city_id = cursor.fetchone()[0]

      addressInsert = (
          "INSERT INTO address (address, address2, district,"
          " city_id, postal_code, customer_id, location) "
          "VALUES (%(add)s, %(add2)s, %(dist)s, %(city)s, "
          "%(post)s, %(cust)s, %(loc)s)")
```

```python
customerData = {
    'first': user['name']['first'],
    'last': user['name']['second'],
    'email': user['email'],
    'dob': user['dob'],
    'spend': user['annualSpend']
}

cursor.execute(customerInsert, customerData)
customerId = cursor.lastrowid

cityQuery = ("SELECT city_id FROM city WHERE "
    "city = %(city)s")
for address in user['address']:
    cursor.execute(cityQuery, {'city': address['city']})
    city_id = cursor.fetchone()[0]

    addressInsert = (
        "INSERT INTO address (address, address2, district,"
        " city_id, postal_code, customer_id, location) "
        "VALUES (%(add)s, %(add2)s, %(dist)s, %(city)s, "
        "%(post)s, %(cust)s, %(loc)s)")
```

```python
    addressData = {
        'add': address['number'],
        'add2': address['street'],
        'dist': address['state'],
        'city': city_id,
        'post': address['postalCode'],
        'cust': customerId,
        'loc':  address['location']
    }

    cursor.execute(addressInsert, addressData)

topicQuery = ("SELECT topics_id FROM topics "
    "WHERE subject = %(subj)s")
interestInsert = (
    "INSERT into interests (topic_id, customer_id) "
    "VALUES (%(topic)s, %(cust)s)")

for interest in user['interests']:
    topicId = 0
    topicData = {
        'subj': interest['interest']
    }
```

```python
    cursor.execute(topicQuery, topicData)
    row = cursor.fetchone()
    if row is None:
        topicInsert = ("INSERT INTO topics (subject) "
            " VALUES (%(subj)s)")
        cursor.execute(topicInsert, topicData)
        topicId = cursor.lastrowid
    else:
        topicId = row[0]

    interestData = {
        'topic': topicId,
        'cust': customerId
    }
    cursor.execute(interestInsert, interestData)

phoneInsert = (
    "INSERT INTO `phone numbers` (customer_id, "
    "phone_number, `Phone number_type`) "
    "VALUES (%(cust)s, %(num)s, %(type)s)")
for phoneNumber in user['phone']:
    phoneData = {
        'cust': customerId,
        'num': phoneNumber['number'],
        'type': phoneNumber['location']
    }
    cursor.execute(phoneInsert, phoneData)

connection.commit()
cursor.close()
return customerId
```

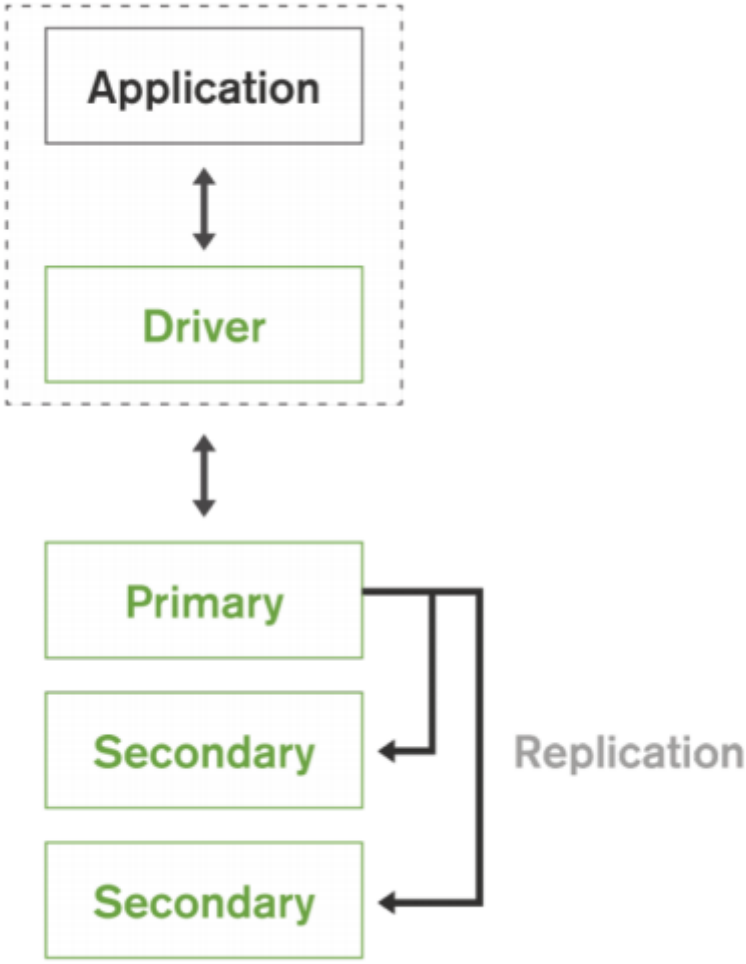
MongoDB 只需要两行代码：

```python
def addUser(database, user):
    return database.customers.insert_one(user).inserted_id
```

2.2 MongoDB 优势：原生的高可用和横向扩展能力

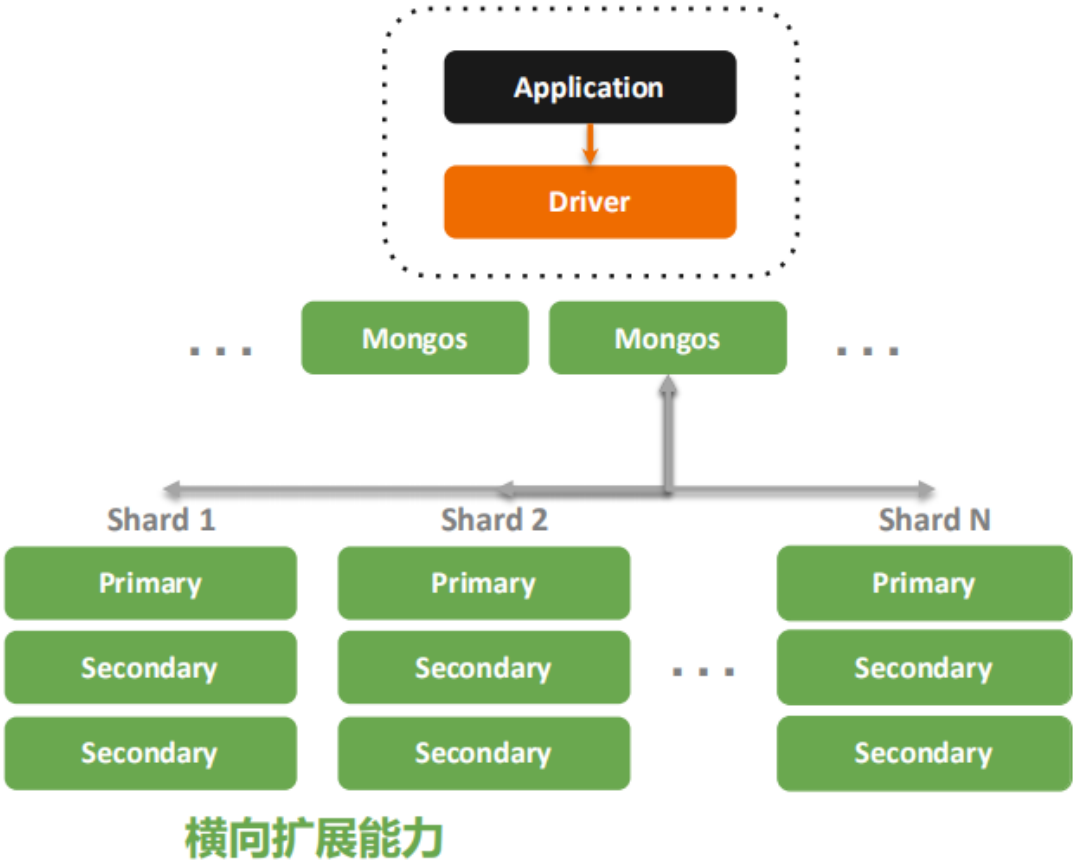
2.2.1 高可用能力

```
a. Replica Set – 2 to 50 个成员
b. 自恢复
c. 多中心容灾能力
d. 滚动服务 – 最小化服务终端
```



2.2.2 横向扩展能力

```
a. 需要的时候无缝扩展
b. 应用全透明
c. 多种数据分布策略
d. 轻松支持TB-PB数量级
```



2.3 MongoDB 技术优势总结

```
a. JSON 结构和对象模型接近，开发代码量低
b. JSON 的动态模型意味着更容易响应新的业务需求
c. 复制集提供99.999%高可用
d. 分片架构支持海量数据和无缝扩容
```

3.MongoDB的获取和安装

3.1 获取MongoDB

```
https://www.mongodb.com/try/download/community
```

3.2安装MongoDB

3.2.0 上传软件并解压

```
[root@node01 nosql]# mkdir -p /mongodb && cd /mongodb
[root@node01 nosql]# tar xf  mongodb-linux-x86_64-rhel70-4.2.8.tgz

# 关闭THP
root用户下
在vi /etc/rc.local最后添加如下代码
if test -f /sys/kernel/mm/transparent_hugepage/enabled; then
    echo never > /sys/kernel/mm/transparent_hugepage/enabled
fi
if test -f /sys/kernel/mm/transparent_hugepage/defrag; then
```

```
            echo never > /sys/kernel/mm/transparent_hugepage/defrag
fi

[root@node01 app]# cat /sys/kernel/mm/transparent_hugepage/enabled
always madvise [never]
[root@node01 app]# cat /sys/kernel/mm/transparent_hugepage/defrag
always madvise [never]

其他系统关闭参照官方文档:
https://docs.mongodb.com/manual/tutorial/transparent-huge-pages/

为什么要关闭?
Transparent Huge Pages (THP) is a Linux memory management system
that reduces the overhead of Translation Lookaside Buffer (TLB)
lookups on machines with large amounts of memory by using larger memory pages.
However, database workloads often perform poorly with THP,
because they tend to have sparse rather than contiguous memory access patterns.
You should disable THP on Linux machines to ensure best performance with MongoDB.
```

3.2.1 环境准备

```
(1)创建所需用户和组
useradd mongod
passwd mongod
(2)创建mongodb所需目录结构
mkdir -p /mongodb/conf
mkdir -p /mongodb/log
mkdir -p /mongodb/data
(3) 修改权限
chown -R mongod:mongod /mongodb
(4) 切换用户并设置环境变量
su - mongod
vi .bash_profile
export PATH=/mongodb/app/bin:$PATH
source .bash_profile
```

3.2.2 启动数据库并初始化数据

```
su - mongod
mongod --dbpath=/mongodb/data --logpath=/mongodb/log/mongodb.log --port=27017 --logappend --fork

# 登录mongodb
$ mongo
```

3.2.3 使用配置文件

```
# 普通配置文件应用
vim /mongodb/conf/mongodb.conf
logpath=/mongodb/log/mongodb.log
dbpath=/mongodb/data
port=27017
```

```
logappend=true
fork=true

# 关闭mongodb
mongod -f /mongodb/conf/mongodb.conf --shutdown
使用配置文件启动mongodb
mongod -f /mongodb/conf/mongodb.conf

# YAML 配置文件应用
--
NOTE:
YAML does not support tab characters for indentation: use spaces instead.

--系统日志有关
systemLog:
   destination: file
   path: "/mongodb/log/mongodb.log"    --日志位置
   logAppend: true                     --日志以追加模式记录
--数据存储有关
storage:
   journal:
      enabled: true
   dbPath: "/mongodb/data"            --数据路径的位置
-- 进程控制
processManagement:
   fork: true                         --后台守护进程
   pidFilePath: <string>              --pid文件的位置，一般不用配置，可以去掉这行，自动生成到data中

--网络配置有关
net:
   bindIp: <ip>                       -- 监听地址，如果不配置这行是监听在0.0.0.0
   port: <port>                       -- 端口号,默认不配置端口号，是27017

-- 安全验证有关配置
security:
  authorization: enabled              --是否打开用户名密码验证

------------------以下是复制集与分片集群有关----------------------
replication:
 oplogSizeMB: <NUM>
 replSetName: "<REPSETNAME>"
 secondaryIndexPrefetch: "all"

sharding:
   clusterRole: <string>
   archiveMovedChunks: <boolean>

---for mongos only
replication:
   localPingThresholdMs: <int>

sharding:
   configDB: <string>
---
.........
+++++++++++++++++++++
YAML例子
```

```
vim /mongodb/conf/mongo.conf
systemLog:
   destination: file
   path: "/mongodb/log/mongodb.log"
   logAppend: true
storage:
   journal:
      enabled: true
   dbPath: "/mongodb/data/"
processManagement:
   fork: true
net:
   port: 27017
   bindIp: 10.0.51,127.0.0.1

mongod -f /mongodb/conf/mongo.conf --shutdown
mongod -f /mongodb/conf/mongo.conf
++++++++++++++++++++++
mongodb的关闭方式
mongod -f mongodb.conf  --shutdown

```

3.2.4 用户基本管理

a . 注意事项

```
用户管理 *****
注意:
验证库，建立用户时use到的库，在使用用户时，要加上验证库才能登陆。
对于管理员用户,必须在admin下创建.
1. 建用户时,use到的库,就是此用户的验证库
2. 登录时,必须明确指定验证库才能登录
3. 通常,管理员用的验证库是admin,普通用户的验证库一般是所管理的库设置为验证库
4. 如果直接登录到数据库,不进行use,默认的验证库是test,不是我们生产建议的.
```

b. 基本语法

```
use admin
mongo 10.0.51/admin
db.createUser
{
    user: "<name>",
    pwd: "<cleartext password>",
    roles: [
       { role: "<role>",
     db: "<database>" } | "<role>",
    ...
    ]
}

基本语法说明:
user:用户名
pwd:密码
roles:
    role:角色名
    db:作用对象
role: root, readWrite,read
```

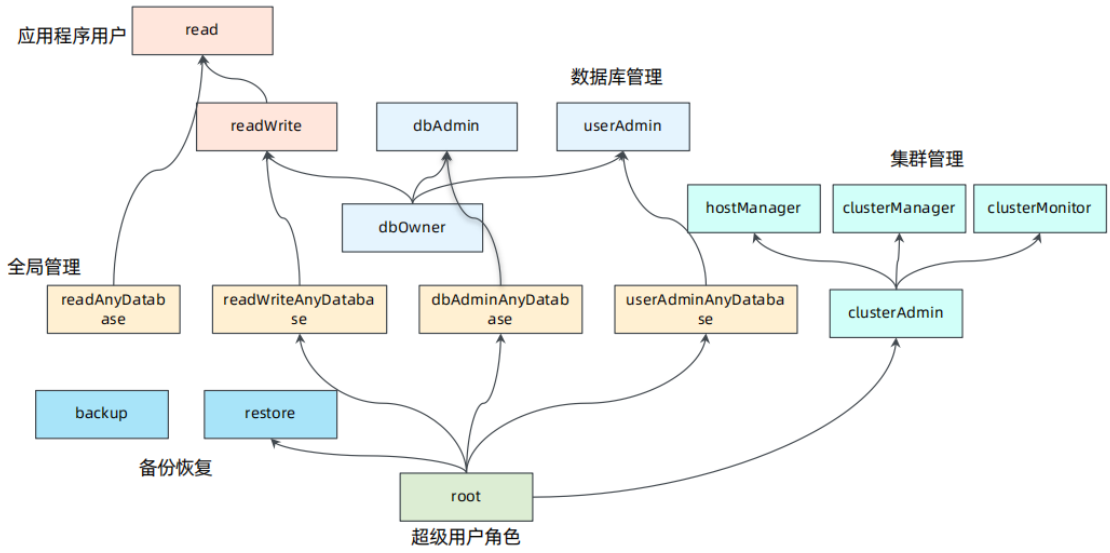
```

验证数据库：
mongo -u test -p 123 10.0.51/test

-------------
```

c. 用户实例

```

（1）--创建超级管理员：管理所有数据库（必须use admin再去创建）
$ mongo
use admin
db.createUser(
{
    user: "root",
    pwd: "root123",
    roles: [ { role: "root", db: "admin" } ]
}
)

验证用户
db.auth('root','root123')

配置文件中，加入以下配置
security:
  authorization: enabled

重启mongodb
mongod -f /mongodb/conf/mongo.conf --shutdown
mongod -f /mongodb/conf/mongo.conf

登录验证
mongo -uroot -proot123  admin
mongo -uroot -proot123  10.0.51/admin


或者
mongo
use admin
db.auth('root','root123')


查看用户:
use admin
db.system.users.find().pretty()

2、创建库管理用户
mongo -uroot -proot123  admin

use app

db.createUser(
{
user: "admin",
pwd: "admin",
roles: [ { role: "dbAdmin", db: "app" } ]
}
```

```
)

db.auth('admin','admin')


登录测试
mongo -uadmin -padmin 10.0.51/app


3、创建对app数据库，读、写权限的用户app01：
（1）超级管理员用户登陆
mongo -uroot -proot123 admin
（2）选择一个验证库
use app

(3)创建用户
db.createUser(
    {
        user: "app01",
        pwd: "app01",
        roles: [ { role: "readWrite" , db: "app" } ]
    }
)

mongo  -uapp01 -papp01 app

4、创建app数据库读写权限的用户并对test数据库具有读权限：
mongo -uroot -proot123 10.0.51/admin
use app
db.createUser(
{
user: "app03",
pwd: "app03",
roles: [ { role: "readWrite", db: "app" },
{ role: "read", db: "test" }
]
}
)

5、查询mongodb中的用户信息
mongo -uroot -proot123 10.0.51/admin
db.system.users.find().pretty()

6、删除用户（root身份登录，use到验证库）
# mongo -uroot -proot123 10.0.51/admin
use app
db.dropUser("admin")
```

d. 角色关系



4. MongoDB 基本CRUD

4.0 通用方法和帮助获取

```
a.获取帮助
help
> help
    db.help()                    help on db methods
    db.mycoll.help()             help on collection methods
    sh.help()                    sharding helpers
    rs.help()                    replica set helpers
db.help();
DB methods:
    db.adminCommand(nameOrDocument) - switches to 'admin' db, and runs command [just calls db.runCommand(...)]
    db.aggregate([pipeline], {options}) - performs a collectionless aggregation on this database; returns a cursor
    db.auth(username, password)
    db.cloneDatabase(fromhost) - will only function with MongoDB 4.0 and below
...
> db.t1.help();
DBCollection help
    db.t1.find().help() - show DBCursor help
    db.t1.bulkWrite( operations, <optional params> ) - bulk execute write operations, optional parameters are: w, wtimeout, j
    db.t1.count( query = {}, <optional params> ) - count the number of documents that matches the query, optional parameters are: limit, skip, hint, maxTimeMS
...

db.[TAB][TAB]
db.t1.[TAB][TAB]

b. 常用操作
//查看当前db版本
test> db.version()
//显示当前数据库
test> db
```

```
> db.getName()
// 查询所有数据库
test> show dbs
//切换数据库
> use local
// 显示当前数据库状态
查看local数据
test> use local
local> db.stats()
查看当前数据库的连接机器地址
> db.getMongo()

// 指定数据库进行连接：（默认连接本机test数据库）
# mongo 192.168.1.24/admin
[mongod@mongodb ~]$ mongo 192.168.1.24/admin

c. 库和表的操作
// 建库
use test
// 删除
> db.dropDatabase()
{ "dropped" : "test", "ok" : 1 }

// 创建集合(表)
方法1:
admin> use app
app> db.createCollection('a')
app> db.createCollection('b')

方法2：当插入一个文档的时候，一个集合就会自动创建。
admin> use app
switched to db app
app> db.c.insert({username:"mongodb"})
WriteResult({ "nInserted" : 1 })
app> show collections

app> db.c.find()
{ "_id" : ObjectId("5743c9a9bf72d9f7b524713d"), "username" : "mongodb" }

// 删除集合
app> use app
switched to db app
app> db.log.drop() //删除集合

// 重命名集合
app> db.log.renameCollection("log1")

```

4.1 使用 insert 完成插入操作

```
操作格式：
db.<集合>.insertOne(<JSON对象>)
db.<集合>.insertMany([<JSON 1>, <JSON 2>, …<JSON n>])
示例：
db.fruit.insertOne({name: "apple"})
db.fruit.insertMany([
{name: "apple"},
{name: "pear"},
{name: "orange"}
])
批量插入数据：
for(i=0;i<10000;i++){
db.log.insert({"uid":i,"name":"mongodb","age":6,"date":new Date()}); }
```

4.2 使用 find 查询文档

```
# 关于 find:
find 是 MongoDB 中查询数据的基本指令，相当于 SQL 中的 SELECT 。
find 返回的是游标。
# find 示例：
db.movies.find( { "year" : 1975 } ) //单条件查询
db.movies.find( { "year" : 1989, "title" : "Batman" } ) //多条件and查询
db.movies.find( { $and : [ {"title" : "Batman"}, { "category" : "action" }] } ) // and的另一种形式
db.movies.find( { $or: [{"year" : 1989}, {"title" : "Batman"}] } ) //多条件or查询
db.movies.find( { "title" : /^B/} ) //按正则表达式查找
```

4.3 查询条件对照表

SQL	MSQL
a = 1	{a: 1}
a <> 1	{a: {$ne: 1}}
a > 1	{a: {$gt: 1}}
a >= 1	{a: {$gte: 1}}
a < 1	{a: {$lt: 1}}
a <= 1	{a: {$lte: 1}}

4.4 查询逻辑对照表

SQL	MSQL
a = 1 AND b = 1	{a: 1, b: 1}或{$and: [{a: 1}, {b: 1}]}
a = 1 OR b = 1	{$or: [{a: 1}, {b: 1}]}
a IS NULL	{a: {$exists: false}}
a IN (1, 2, 3)	{a: {$in: [1, 2, 3]}}

4.5 查询逻辑运算符

```
- $lt: 存在并小于
- $lte: 存在并小于等于
- $gt: 存在并大于
- $gte: 存在并大于等于
- $ne: 不存在或存在但不等于
- $in: 存在并在指定数组中
- $nin: 不存在或不在指定数组中
- $or: 匹配两个或多个条件中的一个
- $and: 匹配全部条件
```

4.6 使用 find 搜索子文档

```
find 支持使用“field.sub_field”的形式查询子文档。假设有一个文档：
db.fruit.insertOne({
name: "apple",
from: {
country: "China",
province: "Guangdong" }
})

正确写法：
db.fruit.find( { "from.country" : "China" } )

```

4.7 使用 find 搜索数组

```
find 支持对数组中的元素进行搜索。假设有一个文档：
db.fruit.insert([
{ "name" : "Apple", color: ["red", "green" ] },
{ "name" : "Mango", color: ["yellow", "green"] }
])
查看单个条件：
db.fruit.find({color: "red"})
查询多个条件：
db.fruit.find({$or: [{color: "red"}, {color: "yellow"}]} )
```

4.8 使用 find 搜索数组中的对象

```
考虑以下文档，在其中搜索
db.movies.insertOne( {
"title" : "Raiders of the Lost Ark",
"filming_locations" : [
{ "city" : "Los Angeles", "state" : "CA", "country" : "USA" },
{ "city" : "Rome", "state" : "Lazio", "country" : "Italy" },
{ "city" : "Florence", "state" : "SC", "country" : "USA" }
]
})
// 查找城市是 Rome 的记录
db.movies.find({"filming_locations.city": "Rome"})
```

4.9 使用 find 搜索数组中的对象

```
在数组中搜索子对象的多个字段时，如果使用 $elemMatch，它表示必须是同一个
子对象满足多个条件。

考虑以下两个查询：
db.getCollection('movies').find({
"filming_locations.city": "Rome",
"filming_locations.country": "USA"
})

db.movies.insertOne( {
"title" : "11111",
"filming_locations" : [
{ "city" : "bj", "state" : "CA", "country" : "CHN" },
{ "city" : "Rome", "state" : "Lazio", "country" : "Italy" },
{ "city" : "tlp", "state" : "SC", "country" : "USA" }
]
})


db.getCollection('movies').find({
"filming_locations": {
$elemMatch:{"city":"bj", "country": "CHN"}
}
})

```

4.10 控制 find 返回的字段

```
find 可以指定只返回指定的字段；
● _id字段必须明确指明不返回，否则默认返回；
● 在 MongoDB 中我们称这为投影（projection）；
● db.movies.find({},{"_id":0, title:1})
```

```
db.movies.find({"category": "action"},{"_id":0, title:1})
```

不返回_id　　返回title

4.11使用 remove 删除文档

```
remove 命令需要配合查询条件使用；
● 匹配查询条件的的文档会被删除；
● 指定一个空文档条件会删除所有文档；
● 以下示例：
db.testcol.remove( { a : 1 } ) // 删除a 等于1的记录
db.testcol.remove( { a : { $lt : 5 } } ) // 删除a 小于5的记录
db.testcol.remove( { } ) // 删除所有记录
db.testcol.remove() //报错
```

4.12 使用 update 更新文档

```
Update 操作执行格式：db.<集合>.update(<查询条件>, <更新字段>)
● 以以下数据为例：
db.fruit.insertMany([
{name: "apple"},
{name: "pear"},
{name: "orange"}
])
db.fruit.updateOne({name: "apple"}, {$set: {from: "China"}})
```

```
db.fruit.updateOne({name: "apple"}, {$set: {from: "China"}})
```

查询 name 为 apple 的记录　　将找到记录的 from 设置为 China

4.13 使用 update 更新文档

```
● 使用 updateOne 表示无论条件匹配多少条记录，始终只更新第一条；
● 使用 updateMany 表示条件匹配多少条就更新多少条；
● updateOne/updateMany 方法要求更新条件部分必须具有以下之一，否则将报错：
    • $set/$unset
    • $push/$pushAll/$pop
    • $pull/$pullAll
    • $addToSet
● // 报错
db.fruit.updateOne({name: "apple"}, {from: "China"})
```

4.14 使用 update 更新数组

```
- $push: 增加一个对象到数组底部
- $pushAll: 增加多个对象到数组底部
- $pop: 从数组底部删除一个对象
- $pull: 如果匹配指定的值，从数组中删除相应的对象
- $pullAll: 如果匹配任意的值，从数据中删除相应的对象
- $addToSet: 如果不存在则增加一个值到数组
```

4.15 使用 drop 删除集合

```
- 使用 db.<集合>.drop() 来删除一个集合
- 集合中的全部文档都会被删除
- 集合相关的索引也会被删除
db.colToBeDropped.drop()
```

4.16 使用 dropDatabase 删除数据库

```
- 使用 db.dropDatabase() 来删除数据库
- 数据库相应文件也会被删除，磁盘空间将被释放
use tempDB
db.dropDatabase()
show collections // No collections
show dbs // The db is gone
```

5. 通过python操作mongodb

5.1 安装 Python MongoDB 驱动程序

```
在 Python 中使用 MongoDB 之前必须先安装用于访问数据库的驱动程序：
wget -O /etc/yum.repos.d/CentOS-Base.repo http://mirrors.aliyun.com/repo/Centos-7.repo
wget -O /etc/yum.repos.d/epel.repo http://mirrors.aliyun.com/repo/epel-7.repo
yum install -y python3
pip3 install pymongo
在 python 交互模式下导入 pymongo，检查驱动是否已正确安装：
import pymongo
pymongo.version
```

5.2 创建连接

```
确定 MongoDB 连接串
使用驱动连接到 MongoDB 集群只需要指定 MongoDB 连接字符串即可。其基本格式可
以参考文档: Connection String URI Format 最简单的形式是
mongodb://数据库服务器主机地址：端口号
如：mongodb://127.0.0.1:27017
- 初始化数据库连接
from pymongo import MongoClient
uri = "mongodb://root:root123@10.0.0.51:27017"
client = MongoClient(uri)
client
```

5.3 数据库操作：插入数据

```
初始化数据库和集合
db = client["eshop"]
user_coll = db["users"]
插入一条新的用户数据
new_user = {"username": "nina", "password": "xxxx", "email":
"123456@qq.com "}
result = user_coll.insert_one(new_user)
result
```

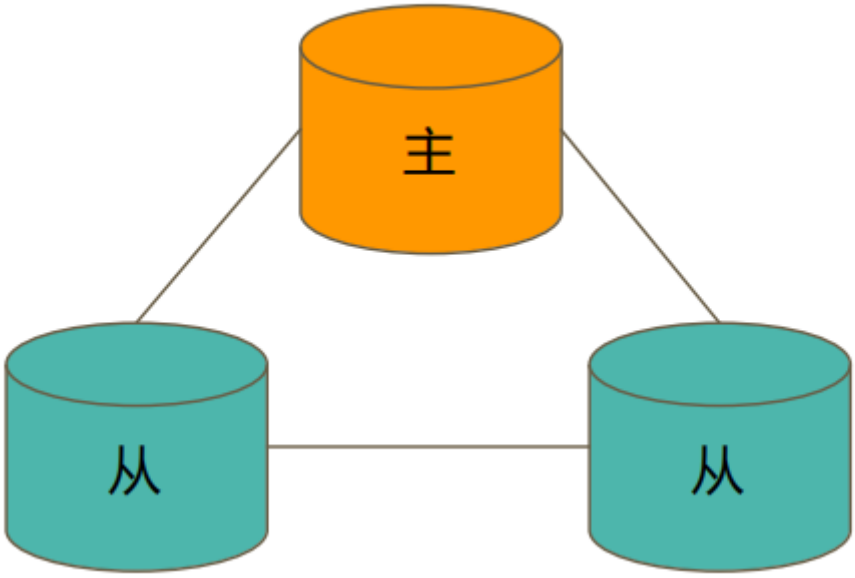
6. MongoDB RS(Replica Set)

6.1 复制集的作用

```
MongoDB 复制集的主要意义在于实现服务高可用,它的现实依赖于两个方面的功能:
• 数据写入时将数据迅速复制到另一个独立节点上
• 在接受写入的节点发生故障时自动选举出一个新的替代节点

在实现高可用的同时，复制集实现了其他几个附加作用:
• 数据分发：将数据从一个区域复制到另一个区域，减少另一个区域的读延迟
• 读写分离：不同类型的压力分别在不同的节点上执行
• 异地容灾：在数据中心故障时候快速切换到异地
```

6.2 典型复制集结构

```
一个典型的复制集由3个以上具有投票权的节点组成，包括:
• 一个主节点（PRIMARY）：接受写入操作和选举时投票
• 两个（或多个）从节点（SECONDARY）：复制主节点上的新数据和选举时投票
• Arbiter（投票节点）
```

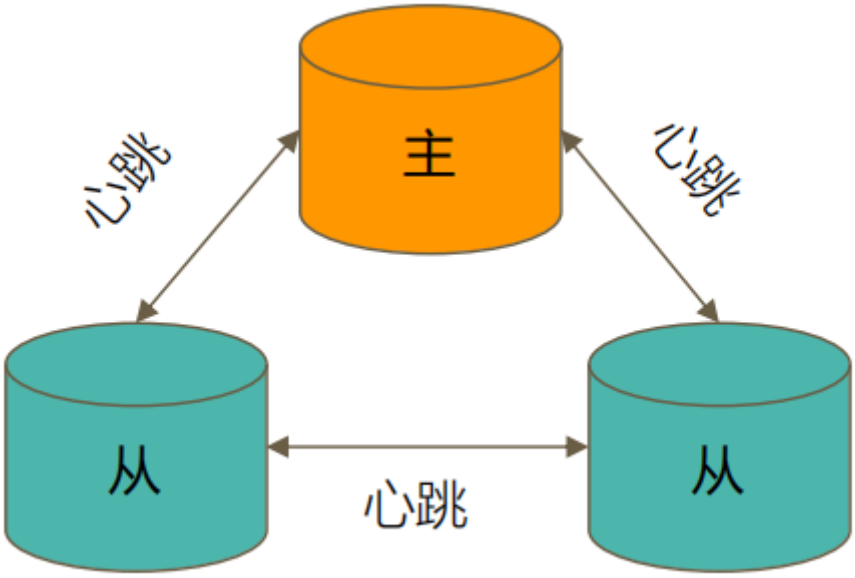


6.3 数据是如何复制的

```
当一个修改操作，无论是插入、更新或删除，到达主节点时，它对数据的操作将被记录下来（经过一些必要的转换），这些记录称为 oplog。
从节点通过在主节点上打开一个 tailable 游标不断获取新进入主节点的 oplog，并在自己的数据上回放，以此保持跟主节点的数据一致。
```

6.4 通过选举完成故障恢复

```
具有投票权的节点之间两两互相发送心跳；
- 当5次心跳未收到时判断为节点失联；
- 如果失联的是主节点，从节点会发起选举，选出新的主节点；
- 如果失联的是从节点则不会产生新的选举；
- 选举基于 RAFT 一致性算法实现，选举成功的必要条件是大多数投票节点存活；
```



6.5 影响选举的因素

```
整个集群必须有大多数节点存活；被选举为主节点的节点必须：
- 能够与多数节点建立连接
- 具有较新的 oplog
- 具有较高的优先级（如果有配置）
```

6.6 常见选项

```
复制集节点有以下常见的选配项：
• 是否具有投票权（v 参数）：有则参与投票；
• 优先级（priority 参数）：优先级越高的节点越优先成为主节点。优先级为0的节点无法成为主节点；
• 隐藏（hidden 参数）：复制数据，但对应用不可见。隐藏节点可以具有投票仅，但优先级必须为0；
• 延迟（slaveDelay 参数）：复制 n 秒之前的数据，保持与主节点的时间差。
```

6.7 复制集注意事项

```
• 关于硬件：
    • 因为正常的复制集节点都有可能成为主节点，它们的地位是一样的，因此硬件配置上必须一致；
    • 为了保证节点不会同时宕机，各节点使用的硬件必须具有独立性。
• 关于软件：
    • 复制集各节点软件版本必须一致，以避免出现不可预知的问题。
• 增加节点不会增加系统写性能！
```

6.8 Replcation Set配置过程详解

a. 规划

```
三个以上的mongodb节点（或多实例）
多实例：
（1）多个端口：28017、28018、28019、28020
（2）多套目录：
su - mongod
mkdir -p /mongodb/28017/conf /mongodb/28017/data /mongodb/28017/log
mkdir -p /mongodb/28018/conf /mongodb/28018/data /mongodb/28018/log
mkdir -p /mongodb/28019/conf /mongodb/28019/data /mongodb/28019/log
mkdir -p /mongodb/28020/conf /mongodb/28020/data /mongodb/28020/log
(3)配置文件内容准备
cat > /mongodb/28017/conf/mongod.conf <<EOF
systemLog:
```

```
  destination: file
  path: /mongodb/28017/log/mongodb.log
  logAppend: true
storage:
  journal:
    enabled: true
  dbPath: /mongodb/28017/data
  directoryPerDB: true
  #engine: wiredTiger
  wiredTiger:
    engineConfig:
      cacheSizeGB: 1
      directoryForIndexes: true
    collectionConfig:
      blockCompressor: zlib
    indexConfig:
      prefixCompression: true
processManagement:
  fork: true
net:
   port: 28017
   bindIp: 10.0.0.51,127.0.0.1
replication:
  oplogSizeMB: 2048
  replSetName: my_repl
EOF
\cp  /mongodb/28017/conf/mongod.conf  /mongodb/28018/conf/
\cp  /mongodb/28017/conf/mongod.conf  /mongodb/28019/conf/
\cp  /mongodb/28017/conf/mongod.conf  /mongodb/28020/conf/

sed 's#28017#28018#g' /mongodb/28018/conf/mongod.conf -i
sed 's#28017#28019#g' /mongodb/28019/conf/mongod.conf -i
sed 's#28017#28020#g' /mongodb/28020/conf/mongod.conf -i

------------------------------------------------------------
(4)启动多个实例备用

mongod -f /mongodb/28017/conf/mongod.conf
mongod -f /mongodb/28018/conf/mongod.conf
mongod -f /mongodb/28019/conf/mongod.conf
mongod -f /mongodb/28020/conf/mongod.conf
```

b. 配置复制集：

```
（1）1主2从，从库普通从库(PSS)
config = {_id: 'my_repl', members: [
                          {_id: 0, host: '10.0.0.51:28017'},
                          {_id: 1, host: '10.0.0.51:28018'},
                          {_id: 2, host: '10.0.0.51:28019'}]
          }

rs.initiate(config)

（2）1主1从1个arbiter(PSA)

config = {_id: 'my_repl', members: [
                          {_id: 0, host: '10.0.0.51:28017'},
```

```
                          {_id: 1, host: '10.0.0.51:28018'},
                          {_id: 2, host:
'10.0.0.51:28019',"arbiterOnly":true}]
          }

rs.initiate(config)

```

c. 复制集测试：

```
my_repl:PRIMARY> db.movies.insert([ { "title" : "Jaws", "year" : 1975,
"imdb_rating" : 8.1 },
... { "title" : "Batman", "year" : 1989, "imdb_rating" : 7.6 },
... ] );
BulkWriteResult({
    "writeErrors" : [ ],
    "writeConcernErrors" : [ ],
    "nInserted" : 2,
    "nUpserted" : 0,
    "nMatched" : 0,
    "nModified" : 0,
    "nRemoved" : 0,
    "upserted" : [ ]
})


my_repl:SECONDARY> db.movies.find().pretty();

注：在mongodb复制集当中，默认从库不允许读写。

rs.slaveOk();
my_repl:SECONDARY> db.movies.find().pretty();

```

d. 复制集管理操作：

```
（1）查看复制集状态：
rs.status();    //查看整体复制集状态
rs.isMaster(); // 查看当前是否是主节点

（2）添加删除节点
rs.add("ip:port"); // 新增从节点
rs.addArb("ip:port"); // 新增仲裁节点
rs.remove("ip:port"); // 删除一个节点

（3）特殊从节点的配置
• 优先级（priority 参数：0-1000）：
优先级越高的节点越优先成为主节点。
优先级为0的节点无法成为主节点；
• 隐藏（hidden 参数）：复制数据，但对应用不可见。隐藏节点可以具有投票仅，但优先级必须为
0；
• 延迟（slaveDelay 参数）：复制 n 秒之前的数据，保持与主节点的时间差。

配置延时节点（一般延时节点也配置成hidden）
cfg=rs.conf()
cfg.members[1].priority=0
```

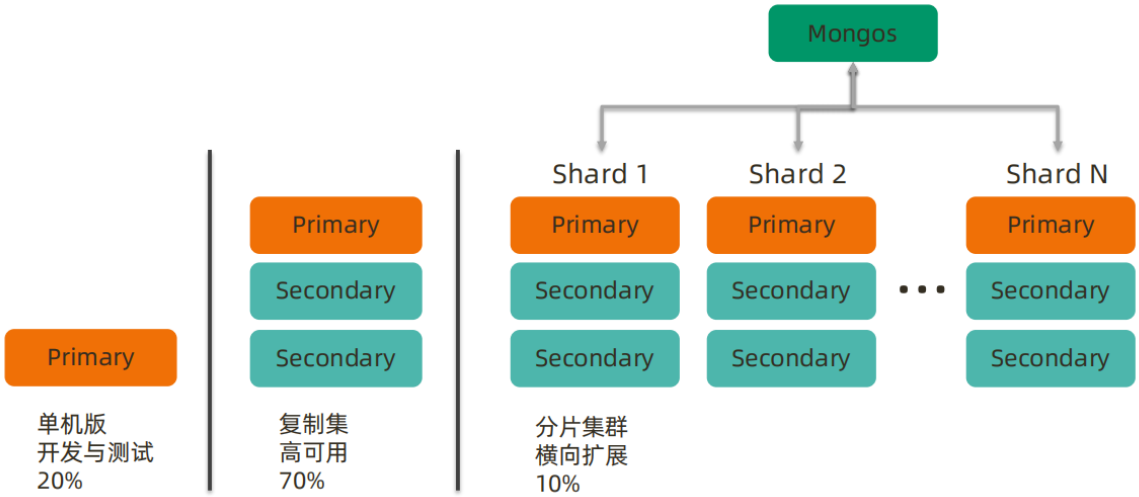
```
cfg.members[1].slaveDelay=120
cfg.members[1].hidden=true
rs.reconfig(cfg)

改回来：
cfg=rs.conf()
cfg.members[2].priority=1
cfg.members[2].slaveDelay=0
cfg.members[2].hidden=0
cfg.members[2].votes=0
rs.reconfig(cfg)
配置成功后，通过以下命令查询配置后的属性
rs.conf();
```

e. 副本集其他操作命令：

```
--查看副本集的配置信息
admin> rs.config()

--查看副本集各成员的状态
admin> rs.status()

--副本集角色切换（不要人为顺便操作，有风险）
admin> rs.stepDown()
注：
admin> rs.freeze(300) //锁定从，使其不会转变成主库
freeze()和stepDown单位都是秒。

--设置副本节点可读：在副本节点执行
admin> rs.slaveOk()

eg:
admin> use app
switched to db app
app> db.createCollection('a')
{ "ok" : 0, "errmsg" : "not master", "code" : 10107 }

--查看副本节点
admin> rs.printSlaveReplicationInfo()
source: 192.168.1.22:27017
    syncedTo: Thu May 26 2016 10:28:56 GMT+0800 (CST)
    0 secs (0 hrs) behind the primary

```

第二章：MongoDB高级运维

1.MongoDB常见架构



2.分片集群机制及原理

2.1 为什么使用分片集群

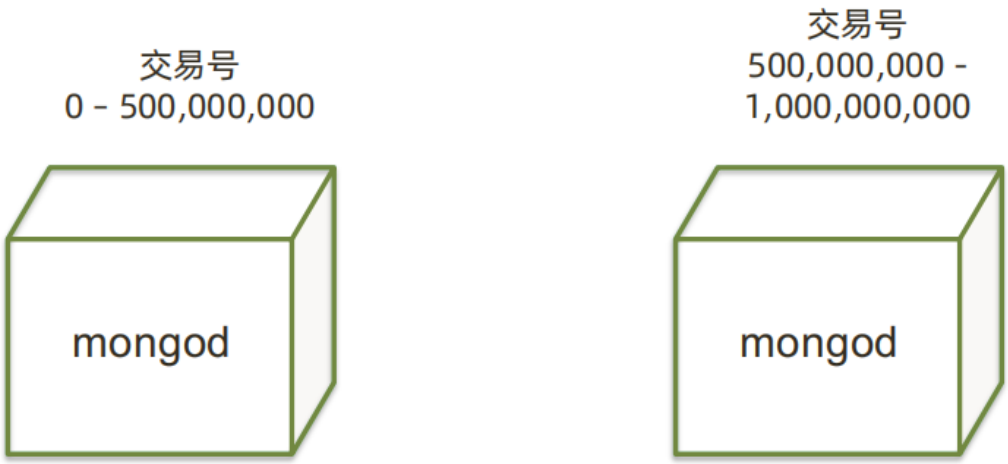
```
• 数据容量日益增大，访问性能日渐降低，怎么破？
• 新品上线异常火爆，如何支撑更多的并发用户？
• 单库已有 10TB 数据，恢复需要1-2天，如何加速？
• 地理分布数据
```

2.2 如何解决以上问题

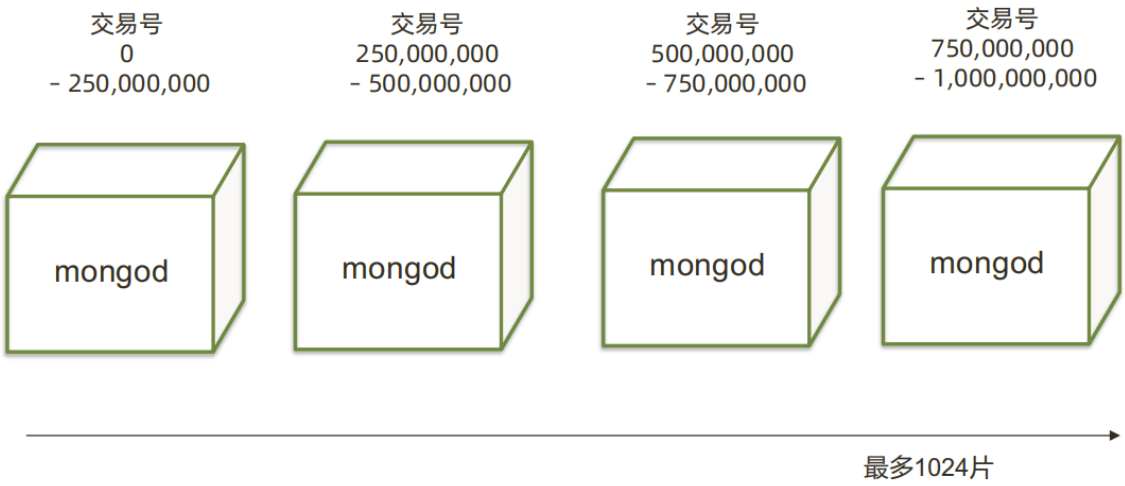
原始结构

```
•银行交易单表内10亿笔资料
•超负荷运转
```

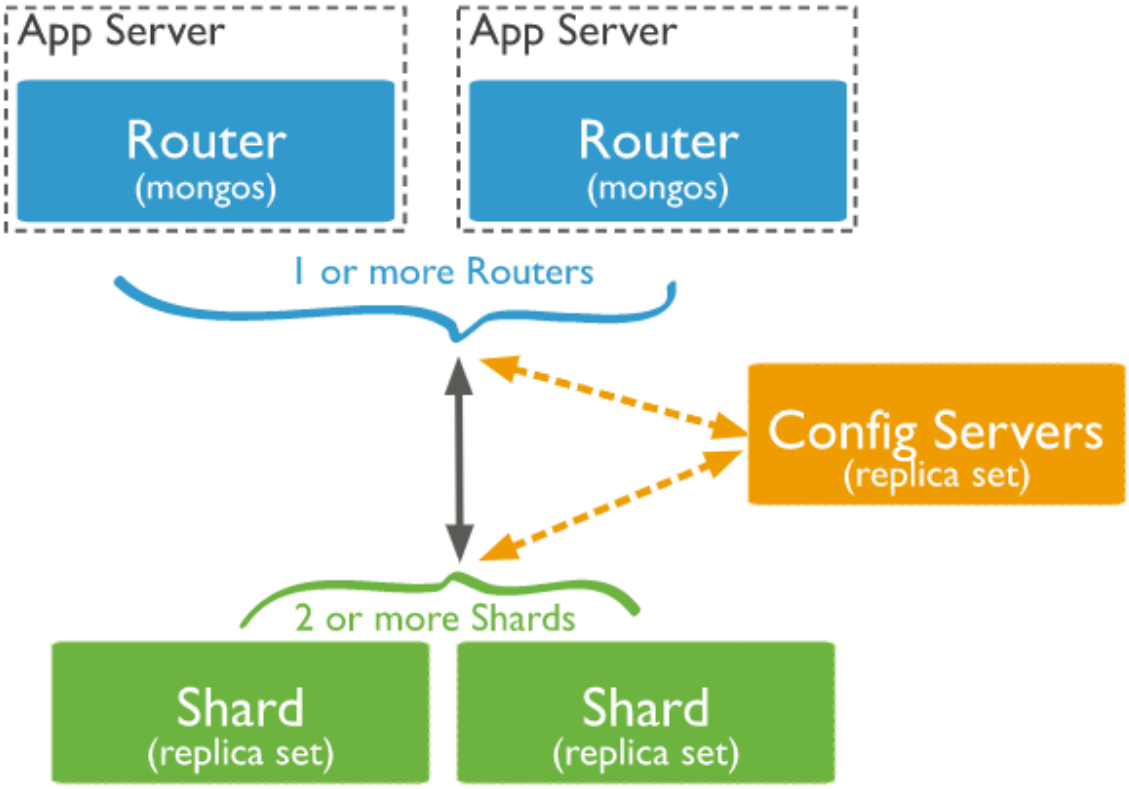
把数据分成两半



把数据分成4部分



2.3 分片架构介绍




```
• Mongos 路由节点
提供集群单一入口
转发应用端请求
选择合适数据节点进行读写
合并多个数据节点的返回
无状态
建议至少2个
• Config Servers配置节点
提供集群元数据存储
分片数据分布的映射
• Shards 数据节点
以复制集为单位
横向扩展
最大1024分片
分片之间数据不重复
所有分片在一起才可
完整工作
```

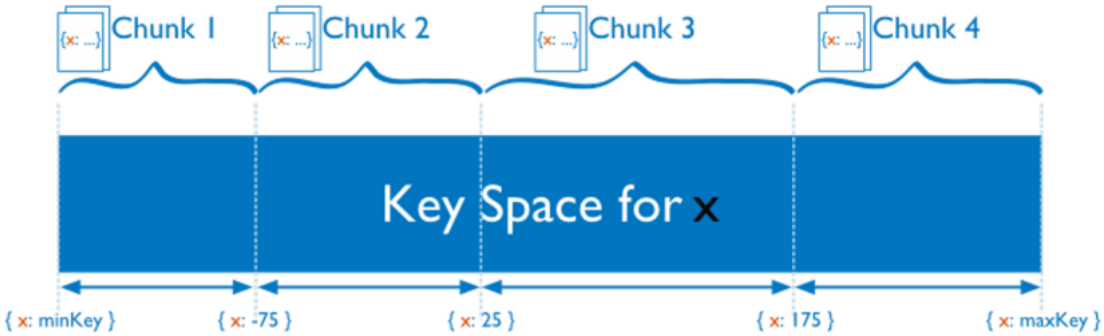
2.4 MongoDB 分片集群特点

```
• 应用全透明，无特殊处理
• 数据自动均衡
• 动态扩容，无须下线
• 提供三种分片方式
```

2.5 分片集群数据分布方式

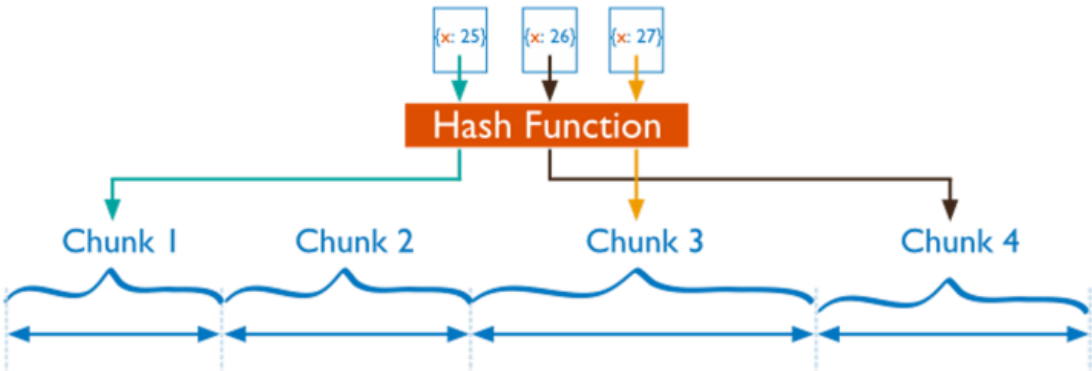
```
• 基于范围
• 基于 Hash
• 基于 zone / tag
```

分片集群数据分布方式 – 基于范围



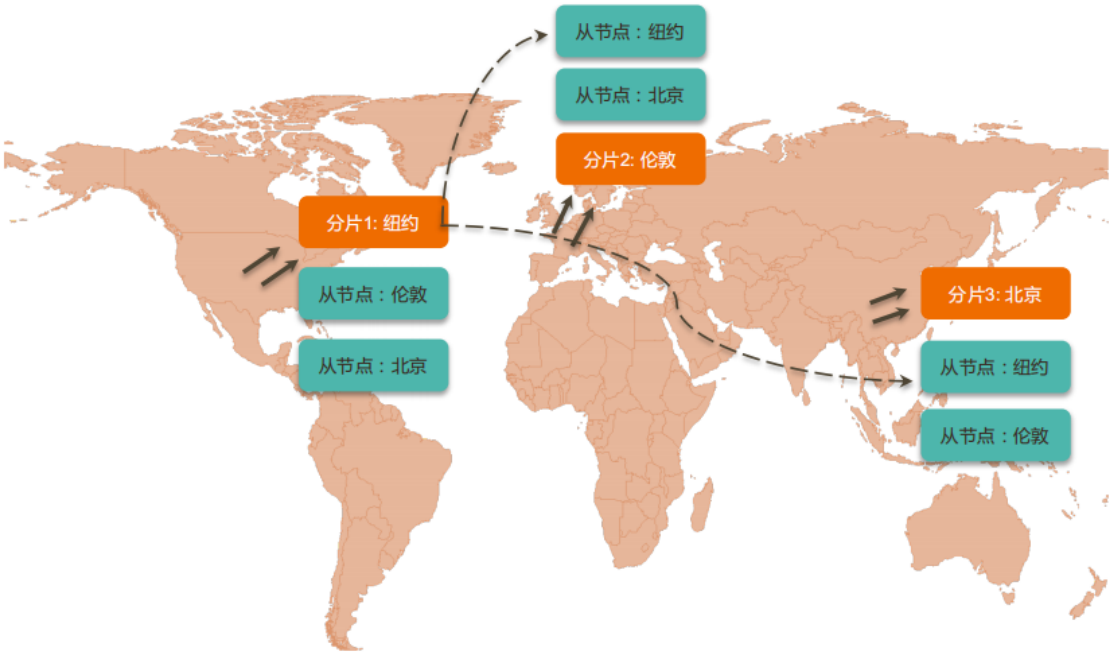
Pros	Cons
片键范围查询性能好	数据分布可能不均匀
优化读	容易有热点

分片集群数据分布方式 – 基于哈希



Pros	Cons
数据分布均匀，写优化	范围查询效率低
适用：日志，物联网等高并发场景	

分片集群数据分布方式 – 自定义Zone



3. 分片集群搭建及扩容

3.1 分片规划

```
10个实例：38017-38026
(1)configserver:
3台构成的复制集（1主两从，不支持arbiter）38018-38020
(2)shard节点：
sh1：38021-23     （1主两从，其中一个节点为arbiter，复制集名字sh1）
sh2：38024-26     （1主两从，其中一个节点为arbiter，复制集名字sh2）
(3)mongos
38017
```

3.2 配置过程

```
a. shard复制集配置：
1. 创建：
mkdir -p /mongodb/38021/conf  /mongodb/38021/log  /mongodb/38021/data
mkdir -p /mongodb/38022/conf  /mongodb/38022/log  /mongodb/38022/data
mkdir -p /mongodb/38023/conf  /mongodb/38023/log  /mongodb/38023/data
mkdir -p /mongodb/38024/conf  /mongodb/38024/log  /mongodb/38024/data
mkdir -p /mongodb/38025/conf  /mongodb/38025/log  /mongodb/38025/data
mkdir -p /mongodb/38026/conf  /mongodb/38026/log  /mongodb/38026/data

2. 配置文件：

sh1:
vi /mongodb/38021/conf/mongodb.conf
===============
根据需求修改相应参数：
systemLog:
  destination: file
  path: /mongodb/38021/log/mongodb.log
```

```
  logAppend: true
storage:
  journal:
    enabled: true
  dbPath: /mongodb/38021/data
  directoryPerDB: true
  #engine: wiredTiger
  wiredTiger:
    engineConfig:
      cacheSizeGB: 1
      directoryForIndexes: true
    collectionConfig:
      blockCompressor: zlib
    indexConfig:
      prefixCompression: true
net:
  bindIp: 10.0.0.51,127.0.0.1
  port: 38021
replication:
  oplogSizeMB: 2048
  replSetName: sh1
sharding:
  clusterRole: shardsvr
processManagement:
  fork: true

===============
cp  /mongodb/38021/conf/mongodb.conf  /mongodb/38022/conf/
cp  /mongodb/38021/conf/mongodb.conf  /mongodb/38023/conf/

sed 's#38021#38022#g' /mongodb/38022/conf/mongodb.conf -i
sed 's#38021#38023#g' /mongodb/38023/conf/mongodb.conf -i

sh2:
vi /mongodb/38024/conf/mongodb.conf
========
根据需求修改相应参数:
systemLog:
  destination: file
  path: /mongodb/38024/log/mongodb.log
  logAppend: true
storage:
  journal:
    enabled: true
  dbPath: /mongodb/38024/data
  directoryPerDB: true
  wiredTiger:
    engineConfig:
      cacheSizeGB: 1
      directoryForIndexes: true
    collectionConfig:
      blockCompressor: zlib
    indexConfig:
      prefixCompression: true
net:
  bindIp: 10.0.0.51,127.0.0.1
  port: 38024
replication:
```

```
  oplogSizeMB: 2048
  replSetName: sh2
sharding:
  clusterRole: shardsvr
processManagement:
  fork: true
========


cp  /mongodb/38024/conf/mongodb.conf  /mongodb/38025/conf/
cp  /mongodb/38024/conf/mongodb.conf  /mongodb/38026/conf/

sed 's#38024#38025#g' /mongodb/38025/conf/mongodb.conf -i
sed 's#38024#38026#g' /mongodb/38026/conf/mongodb.conf -i


3. 所有节点，并搭建复制集：

mongod -f  /mongodb/38021/conf/mongodb.conf
mongod -f  /mongodb/38022/conf/mongodb.conf
mongod -f  /mongodb/38023/conf/mongodb.conf
mongod -f  /mongodb/38024/conf/mongodb.conf
mongod -f  /mongodb/38025/conf/mongodb.conf
mongod -f  /mongodb/38026/conf/mongodb.conf



mongo --port 38021

use  admin
config = {_id: 'sh1', members: [
                          {_id: 0, host: '10.0.0.51:38021'},
                          {_id: 1, host: '10.0.0.51:38022'},
                          {_id: 2, host: '10.0.0.51:38023',"arbiterOnly":true}]
           }

rs.initiate(config)


 mongo --port 38024
 use admin
config = {_id: 'sh2', members: [
                          {_id: 0, host: '10.0.0.51:38024'},
                          {_id: 1, host: '10.0.0.51:38025'},
                          {_id: 2, host: '10.0.0.51:38026',"arbiterOnly":true}]
           }

rs.initiate(config)

=-=----=-=-=-
```

```

2. 修改配置文件：

[mongod@server1 ~]$ vi /mongodb/38018/conf/mongodb.conf

systemLog:
  destination: file
  path: /mongodb/38018/log/mongodb.conf
  logAppend: true
storage:
  journal:
    enabled: true
  dbPath: /mongodb/38018/data
  directoryPerDB: true
  #engine: wiredTiger
  wiredTiger:
    engineConfig:
      cacheSizeGB: 1
      directoryForIndexes: true
    collectionConfig:
      blockCompressor: zlib
    indexConfig:
      prefixCompression: true
net:
  bindIp: 10.0.0.51,127.0.0.1
  port: 38018
replication:
  oplogSizeMB: 2048
  replSetName: configReplSet
sharding:
  clusterRole: configsvr
processManagement:
  fork: true


cp /mongodb/38018/conf/mongodb.conf /mongodb/38019/conf/
cp /mongodb/38018/conf/mongodb.conf /mongodb/38020/conf/

sed 's#38018#38019#g' /mongodb/38019/conf/mongodb.conf -i
sed 's#38018#38020#g' /mongodb/38020/conf/mongodb.conf -i


3. 启动节点，并配置复制集


mongod -f /mongodb/38018/conf/mongodb.conf
mongod -f /mongodb/38019/conf/mongodb.conf
mongod -f /mongodb/38020/conf/mongodb.conf


mongo --port 38018
use  admin

 config = {_id: 'configReplSet', members: [
                          {_id: 0, host: '10.0.0.51:38018'},
                          {_id: 1, host: '10.0.0.51:38019'},
                          {_id: 2, host: '10.0.0.51:38020'}]
           }
```

```
rs.initiate(config)


注：configserver 可以是一个节点，官方建议复制集。configserver不能有arbiter。
新版本中，要求必须是复制集。
注：mongodb 3.4之后，虽然要求config server为replica set，但是不支持arbiter

================================
c. mongos节点配置：
1. 创建目录：
mkdir -p /mongodb/38017/conf  /mongodb/38017/log

2配置文件：
vi /mongodb/38017/conf/mongos.conf
systemLog:
  destination: file
  path: /mongodb/38017/log/mongos.log
  logAppend: true
net:
  bindIp: 10.0.0.51,127.0.0.1
  port: 38017
sharding:
  configDB: configReplSet/10.0.0.51:38018,10.0.0.51:38019,10.0.0.51:38020
processManagement:
  fork: true

3. 启动mongos

mongos -f /mongodb/38017/conf/mongos.conf

d. 分片集群操作：

连接到其中一个mongos（10.0.0.51），做以下配置
（1）连接到mongs的admin数据库
# su - mongod
$ mongo 10.0.0.51:38017/admin

（2）添加分片
db.runCommand( { addshard : "sh1/10.0.0.51:38021,10.0.0.51:38022,10.0.0.51:38023",name:"shard1"} )
db.runCommand( { addshard : "sh2/10.0.0.51:38024,10.0.0.51:38025,10.0.0.51:38026",name:"shard2"} )

（3）列出分片
mongos> db.runCommand( { listshards : 1 } )

（4）整体状态查看
mongos> sh.status();

==================================

e. 使用分片集群

##RANGE分片配置及测试
test库下的vast大表进行手工分片

1、激活数据库分片功能
mongo --port 38017 admin
```

```
admin>  ( { enablesharding : "数据库名称" } )

eg:
admin> db.runCommand( { enablesharding : "test" } )

2、指定分片建对集合分片
eg：范围片键
--创建索引
use test
> db.vast.ensureIndex( { id: 1 } )

--开启分片
use admin
> db.runCommand( { shardcollection : "test.vast",key : {id: 1} } )

3、集合分片验证
admin> use test

test> for(i=1;i<500000;i++){
db.vast.insert({"id":i,"name":"shenzheng","age":70,"date":new Date()}); }

test> db.vast.stats()

4、分片结果测试

shard1:
mongo --port 38021
db.vast.count();


shard2:
mongo --port 38024
db.vast.count();

-----------------------------------------------------
f. Hash分片例子：
对test库下的vast大表进行hash
创建哈希索引
（1）对于test开启分片功能
mongo --port 38017 admin
use admin
admin> db.runCommand( { enablesharding : "test" } )

（2）对于test库下的vast表建立hash索引
use test
test> db.vast.ensureIndex( { id: "hashed" } )
（3）开启分片
use admin
admin > sh.shardCollection( "test.vast", { id: "hashed" } )

（4）录入10w行数据测试
use test
for(i=1;i<100000;i++){
db.vast.insert({"id":i,"name":"shenzheng","age":70,"date":new Date()}); }

（5）hash分片结果测试
mongo --port 38021
use test
```

```
db.vast.count();

mongo --port 38024
use test
db.vast.count();
---------------------------
g. 分片的管理
1、判断是否Shard集群
admin> db.runCommand({ isdbgrid : 1})

2、列出所有分片信息
admin> db.runCommand({ listshards : 1})

3、列出开启分片的数据库
admin> use config

config> db.databases.find( { "partitioned": true } )
或者：
config> db.databases.find() //列出所有数据库分片情况

4、查看分片的片键
config> db.collections.find().pretty()
{
    "_id" : "test.vast",
    "lastmodEpoch" : ObjectId("58a599f19c898bbfb818b63c"),
    "lastmod" : ISODate("1970-02-19T17:02:47.296Z"),
    "dropped" : false,
    "key" : {
        "id" : 1
    },
    "unique" : false
}

5、查看分片的详细信息
admin> db.printShardingStatus()
或
admin> sh.status()

6、删除分片节点（谨慎）
（1）确认blance是否在工作
sh.getBalancerState()

（2）删除shard2节点(谨慎)
mongos> db.runCommand( { removeShard: "shard2" } )

注意：删除操作一定会立即触发blancer。

7、balancer操作

介绍：
mongos的一个重要功能，自动巡查所有shard节点上的chunk的情况，自动做chunk迁移。
什么时候工作？
1、自动运行，会检测系统不繁忙的时候做迁移
2、在做节点删除的时候，立即开始迁移工作
3、balancer只能在预设定的时间窗口内运行

有需要时可以关闭和开启blancer（备份的时候）
mongos> sh.stopBalancer()
```

```
mongos> sh.startBalancer()

8、自定义 自动平衡进行的时间段
https://docs.mongodb.com/manual/tutorial/manage-sharded-cluster-balancer/#schedule-the-balancing-window
// connect to mongos

use config
sh.setBalancerState( true )
db.settings.update({ _id : "balancer" }, { $set : { activeWindow : { start : "3:00", stop : "5:00" } } }, true )

sh.getBalancerWindow()
sh.status()

```

4. 企业中分片集群设计

4.1分片的基本标准

```
• 关于数据：数据量不超过3TB，尽可能保持在2TB一个片；
• 关于索引：常用索引必须容纳进内存；
• 按照以上标准初步确定分片后，还需要考虑业务压力，随着压力增大，CPU、RAM、磁盘中的任何一项出现瓶颈时，都可以通过添加更多分片来解决。
```

4.2 如何粗略判断需要多少分片

条件	分片个数
B : 工作集大小 / 单服务器内存容量	400GB / （256G * 0.6） = 3
A : 所需存储总量 / 单服务器可挂载容量	8TB / 2TB = 4
C : 并发量总数 / （单服务器并发量 * 0.7)	30000 / (9000*0.7) = 6
D: 额外开销	?

```
分片数量 = max(A, B, C)+D =?
```

4.3 额外的考量

```
考虑分片的分布：
• 是否需要跨机房分布分片？
• 是否需要容灾？
• 高可用的要求如何？
```

4.4 选择片键的正确姿势

```
影响片键效率的主要因素：
• 取值基数（Cardinality）；
• 取值分布；
• 分散写，集中读；
• 被尽可能多的业务场景用到；
• 避免单调递增或递减的片键；
```

a. 选择基数大的片键

```
对于小基数的片键：
• 因为备选值有限，那么块的总数量就有限；
• 随着数据增多，块的大小会越来越大；
• 水平扩展时移动块会非常困难；

例如：存储一个高中的师生数据，以年龄（假设年龄范围为15~65岁）作为片键，
15<=年龄<=65，且只为整数,所以最多只会有51个 chunk

结论：取值基数要大！
```

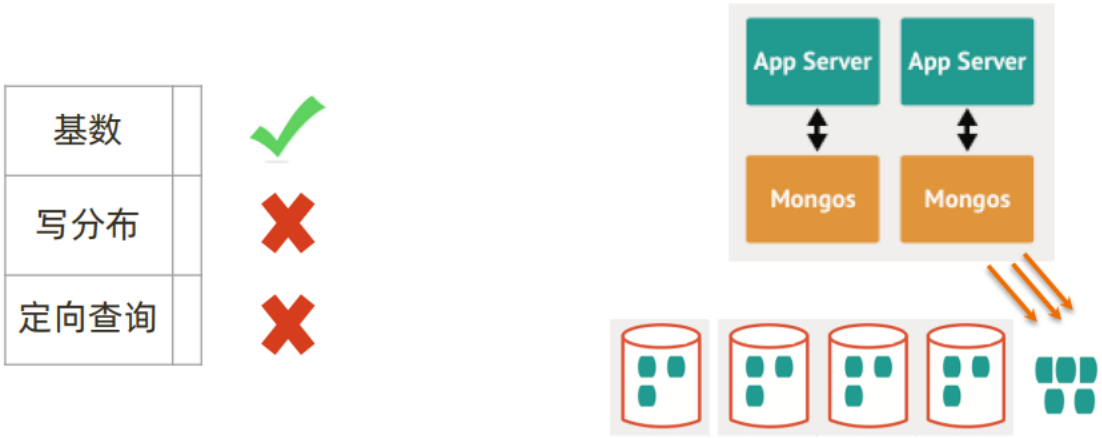
b.选择分布均匀的片键

```
对于分布不均匀的片键：
• 造成某些块的数据量急剧增大
• 这些块压力随之增大
• 数据均衡以 chunk 为单位，所以系统无能为力 • 例如：存储一个学校的师生数据，以年龄（假设年龄范围为15~65岁）作为片键，
那么：
• 15<=年龄<=65，且只为整数
• 大部分人的年龄范围为15~18岁（学生） • 15、16、17、18四个 chunk 的数据量、访问压力远大于其他 chunk
结论：取值分布应尽可能均匀
```

c. 通过一个例子来看片键选择

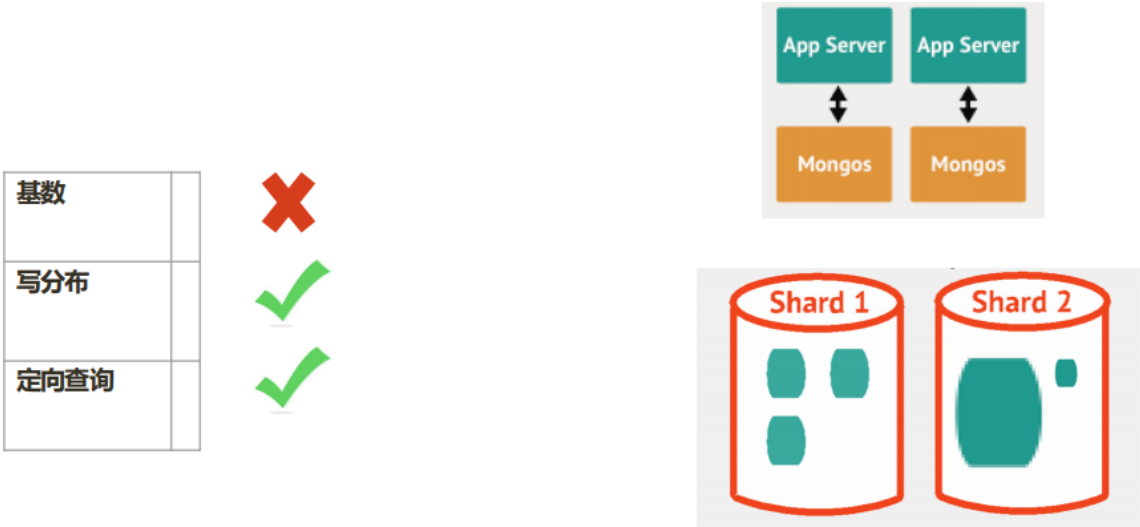
```
{
_id: ObjectId(),
user: 123,
time: Date(),
subject: “...”,
recipients: [],
body: “...”,
attachments: []
}
```

选择片键： { _id: 1}



选择片键： { _id: ”hashed”}

选择片键：{ user_id: 1 }



选择片键：{ user_id: 1, time:1 }

4.5 如何规划硬件

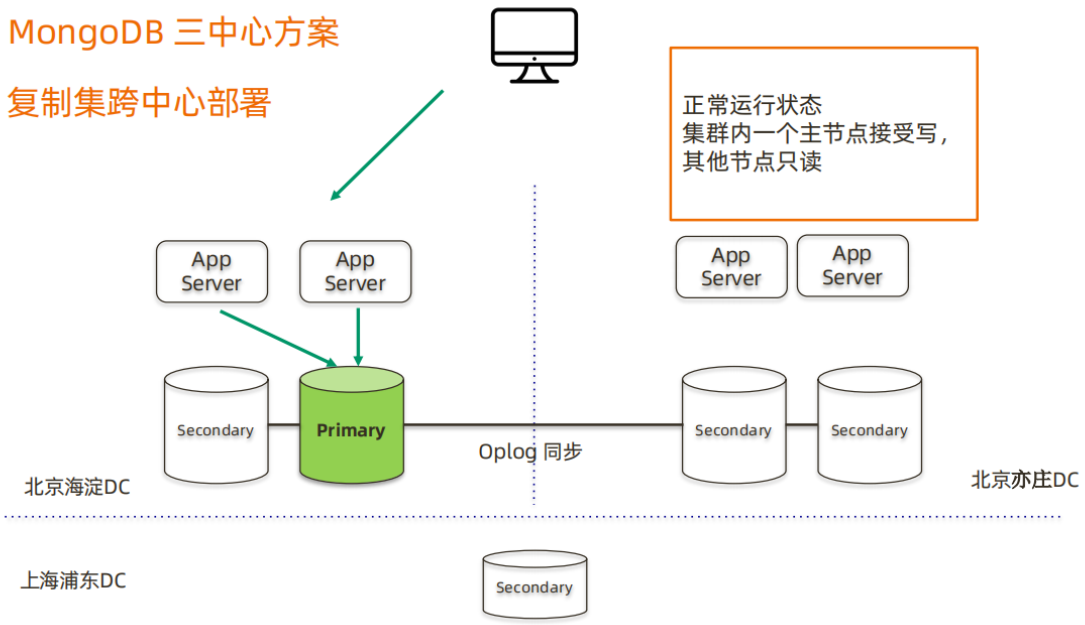
```
• mongos 与 config 通常消耗很少的资源，可以选择低规格虚拟机；
• 资源的重点在于 shard 服务器：
• 需要足以容纳热数据索引的内存；
• 正确创建索引后 CPU 通常不会成为瓶颈，除非涉及非常多的计算；
• 磁盘尽量选用 SSD；
实际测试是最好的检验，来看你的资源配置是否完备。
另外，即使项目初期已经具备了足够的资源，仍然需要考虑在合适的时候扩展。建议监控
各项资源使用情况，无论哪一项达到60%以上，则开始考虑扩展。
• 扩展需要新的资源，申请新资源需要时间；
• 扩展后数据需要均衡，均衡需要时间。应保证新数据入库速度慢于均衡速度
• 均衡需要资源，如果资源即将或已经耗尽，均衡也是会很低效的。
```

5. 高级集群设计：两地三中心

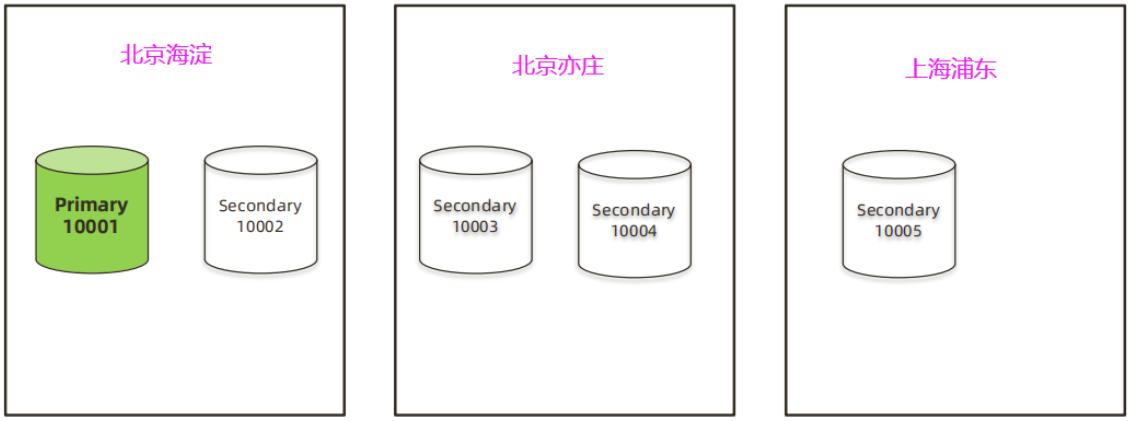
5.1 容灾级别

级别	方式	RPO	RTO
L0	**无备源中心**: 没有灾难恢复能力，只在本地进行数据备份	24小时+	4小时+
L1	**本地备份+异地保存**: 本地将关键数据备份，然后送到异地保存。 灾难发生后，按预定数据恢复程序恢复系统和数据。	24小时+	8小时+
L2	**双中心主备模式**: 在异地建立一个热备份点，通过网络进行数据备份。 当出现灾难时，备份站点接替主站点的业务，维护业务连续性	秒级	数分钟到半小时
L3	**双中心双活** 在相隔较远的地方分别建立两个数据中心，进行相互数据备份。 当某个数据中心发生灾难时，另一个数据中心接替其工作任务。	秒级	秒级
L4	**双中心双活 + 异地热备 = 两地三中心** 在同城分别建立两个数据中心，进行相互数据备份。 当该城市的2个中心同时不可用（地震/大面积停电/网络等），快速切换到异地	秒级	分钟级

5.2 MongoDB 两地三中心集群实现



5.3 两地三中心规划及实施



a. 准备虚拟机及数据库实例

```
1. 规划
# 10.0.0.51
primary:10.0.0.51:10001
s1     :10.0.0.51:10002

# 10.0.0.52
s3     :10.0.0.52:10003
s4     :10.0.0.52:10004
# 10.0.53
s5     :10.0.53:10005

2. 准备实例
# 配置文件-db01
su - mongod
mkdir -p /mongodb/10001/conf /mongodb/10001/data /mongodb/10001/log
mkdir -p /mongodb/10002/conf /mongodb/10002/data /mongodb/10002/log
cat > /mongodb/10001/conf/mongod.conf <<EOF
systemLog:
  destination: file
  path: /mongodb/10001/log/mongodb.log
  logAppend: true
storage:
  journal:
    enabled: true
  dbPath: /mongodb/10001/data
  directoryPerDB: true
  #engine: wiredTiger
  wiredTiger:
    engineConfig:
      cacheSizeGB: 0.5
      directoryForIndexes: true
    collectionConfig:
      blockCompressor: zlib
    indexConfig:
      prefixCompression: true
processManagement:
  fork: true
net:
   port: 10001
   bindIp: 10.0.0.51,127.0.0.1
replication:
```

```
  oplogSizeMB: 2048
  replSetName: my_repl
EOF
\cp  /mongodb/10001/conf/mongod.conf  /mongodb/10002/conf/
sed 's#10001#10002#g' /mongodb/10002/conf/mongod.conf -i

mongod -f /mongodb/10001/conf/mongod.conf
mongod -f /mongodb/10002/conf/mongod.conf


# 配置文件-db02
su - mongod
mkdir -p /mongodb/10003/conf /mongodb/10003/data /mongodb/10003/log
mkdir -p /mongodb/10004/conf /mongodb/10004/data /mongodb/10004/log
cat > /mongodb/10003/conf/mongod.conf <<EOF
systemLog:
  destination: file
  path: /mongodb/10003/log/mongodb.log
  logAppend: true
storage:
  journal:
    enabled: true
  dbPath: /mongodb/10003/data
  directoryPerDB: true
  #engine: wiredTiger
  wiredTiger:
    engineConfig:
      cacheSizeGB: 0.5
      directoryForIndexes: true
    collectionConfig:
      blockCompressor: zlib
    indexConfig:
      prefixCompression: true
processManagement:
  fork: true
net:
   port: 10003
   bindIp: 10.0.0.52,127.0.0.1
replication:
  oplogSizeMB: 2048
  replSetName: my_repl
EOF
\cp  /mongodb/10003/conf/mongod.conf  /mongodb/10004/conf/
sed 's#10003#10004#g' /mongodb/10004/conf/mongod.conf -i

mongod -f /mongodb/10003/conf/mongod.conf
mongod -f /mongodb/10004/conf/mongod.conf


# 配置文件-db03
su - mongod
mkdir -p /mongodb/10005/conf /mongodb/10005/data /mongodb/10005/log
cat > /mongodb/10005/conf/mongod.conf <<EOF
systemLog:
  destination: file
  path: /mongodb/10005/log/mongodb.log
  logAppend: true
storage:
```

```
  journal:
    enabled: true
  dbPath: /mongodb/10005/data
  directoryPerDB: true
  #engine: wiredTiger
  wiredTiger:
    engineConfig:
      cacheSizeGB: 0.5
      directoryForIndexes: true
    collectionConfig:
      blockCompressor: zlib
    indexConfig:
      prefixCompression: true
processManagement:
  fork: true
net:
   port: 10005
   bindIp: 10.0.53,127.0.0.1
replication:
  oplogSizeMB: 2048
  replSetName: my_repl
EOF
mongod -f /mongodb/10005/conf/mongod.conf


config = {_id: 'my_repl', members: [
                           {_id: 0, host: '10.0.0.51:10001'},
                           {_id: 1, host: '10.0.0.51:10002'},
                           {_id: 2, host: '10.0.0.52:10003'},
                           {_id: 3, host: '10.0.0.52:10004'},
                           {_id: 4, host: '10.0.53:10005'}
                           ]
          }
rs.initiate(config)
```

b. 两地三中心定制化配置

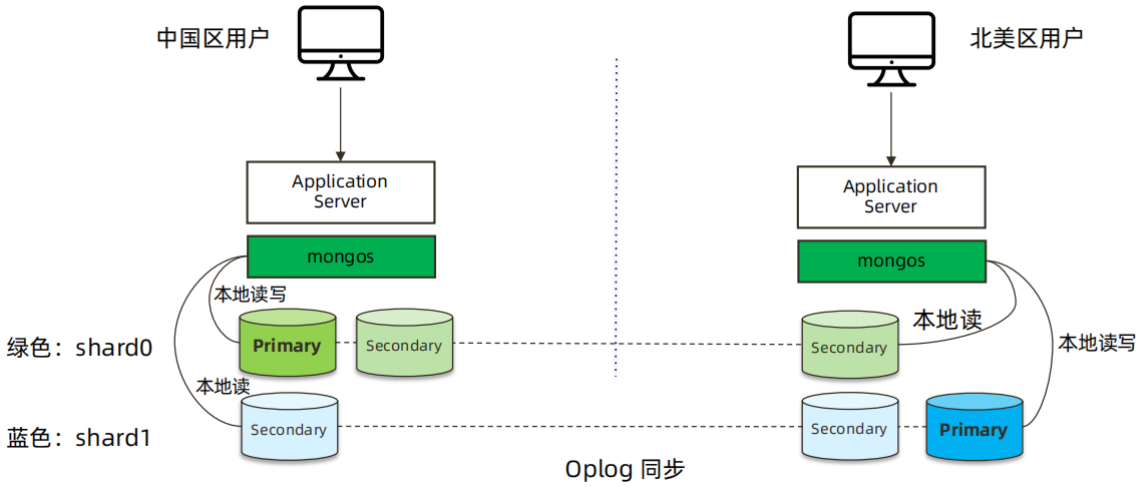
```

cfg = rs.conf()
cfg.members[1].priority = 20
cfg.members[2].priority = 10
cfg.members[3].priority = 10
rs.reconfig(cfg)
```

c. 复制集安全加固

```
# db01
openssl rand -base64 756 > /mongodb/10001/conf/keyfile
cp -a  /mongodb/10001/conf/keyfile /mongodb/10002/conf
chmod 600 /mongodb/10001/conf/keyfile /mongodb/10002/conf/keyfile
scp /mongodb/10001/conf/keyfile 10.0.0.52:/mongodb/10003/conf
scp /mongodb/10001/conf/keyfile 10.0.0.52:/mongodb/10004/conf
scp /mongodb/10001/conf/keyfile 10.0.53:/mongodb/10005/conf

```

```
每个节点开启验证：
cat >> /mongodb/10001/conf/mongod.conf<<EOF
security:
  keyFile: /mongodb/10001/conf/keyfile
EOF
cat >>/mongodb/10002/conf/mongod.conf<<EOF
security:
  keyFile: /mongodb/10002/conf/keyfile
EOF
cat >> /mongodb/10003/conf/mongod.conf <<EOF
security:
  keyFile: /mongodb/10003/conf/keyfile
EOF
cat >> /mongodb/10004/conf/mongod.conf <<EOF
security:
  keyFile: /mongodb/10004/conf/keyfile
EOF
cat >> /mongodb/10005/conf/mongod.conf <<EOF
security:
  keyFile: /mongodb/10005/conf/keyfile
EOF

use admin
db.shutdownServer()
+++++
Shut down each mongod in the replica set, starting with the secondaries. Continue until all members of the replica set are offline, including any arbiters. The primary must be the last member shut down to avoid potential rollbacks.
+++++


启动所有节点，在主节点添加用户：

use admin
db.createUser(
{
    user: "root",
    pwd: "root123",
    roles: [ { role: "root", db: "admin" } ]
}
)

手工交互式输入密码
db.createUser(
{
    user: "root1",
    pwd: passwordPrompt(),
    roles: [ { role: "root", db: "admin" } ]
}
)
手工交互式验证
my_repl:PRIMARY> use admin
switched to db admin
my_repl:PRIMARY> db.auth("root1",passwordPrompt())
Enter password:

```

6. 高级集群设计：全球多写

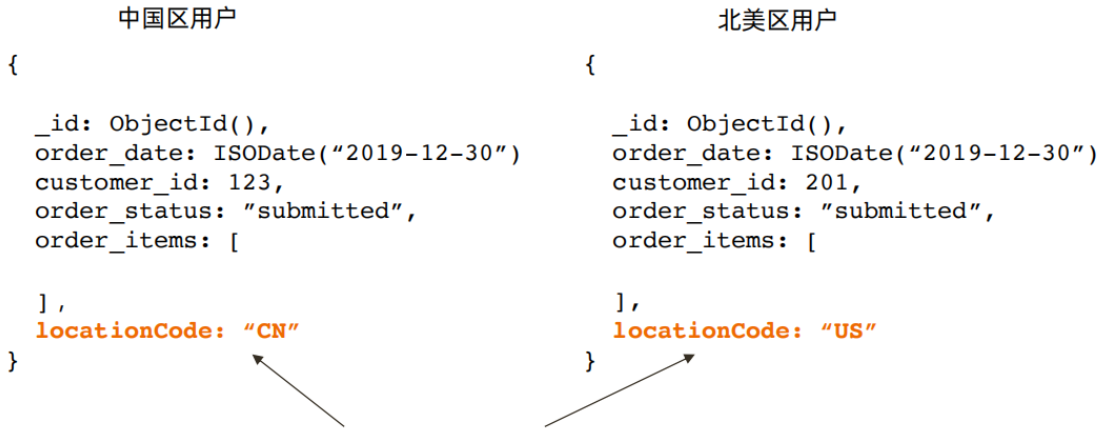



```
# db01:
su - mongod
mkdir -p /mongodb/20001/conf /mongodb/20001/data /mongodb/20001/log
mkdir -p /mongodb/20002/conf /mongodb/20002/data /mongodb/20002/log
mkdir -p /mongodb/20003/conf /mongodb/20003/data /mongodb/20003/log
cat > /mongodb/20001/conf/mongod.conf <<EOF
systemLog:
  destination: file
  path: /mongodb/20001/log/mongodb.log
  logAppend: true
storage:
  journal:
    enabled: true
  dbPath: /mongodb/20001/data
  directoryPerDB: true
  #engine: wiredTiger
  wiredTiger:
    engineConfig:
      cacheSizeGB: 0.5
      directoryForIndexes: true
    collectionConfig:
      blockCompressor: zlib
    indexConfig:
      prefixCompression: true
processManagement:
  fork: true
net:
   port: 20001
   bindIp: 10.0.0.51,127.0.0.1
replication:
  oplogSizeMB: 2048
  replSetName: CN_sh
sharding:
  clusterRole: shardsvr
EOF
\cp  /mongodb/20001/conf/mongod.conf  /mongodb/20002/conf/
```

```
\cp  /mongodb/20001/conf/mongod.conf  /mongodb/20003/conf/
sed 's#20001#20002#g' /mongodb/20002/conf/mongod.conf -i
sed 's#20001#20003#g' /mongodb/20003/conf/mongod.conf -i


mongod -f /mongodb/20001/conf/mongod.conf
mongod -f /mongodb/20002/conf/mongod.conf
mongod -f /mongodb/20003/conf/mongod.conf



# db02:
su - mongod
mkdir -p /mongodb/20001/conf /mongodb/20001/data /mongodb/20001/log
mkdir -p /mongodb/20002/conf /mongodb/20002/data /mongodb/20002/log
mkdir -p /mongodb/20003/conf /mongodb/20003/data /mongodb/20003/log
cat > /mongodb/20001/conf/mongod.conf <<EOF
systemLog:
  destination: file
  path: /mongodb/20001/log/mongodb.log
  logAppend: true
storage:
  journal:
    enabled: true
  dbPath: /mongodb/20001/data
  directoryPerDB: true
  #engine: wiredTiger
  wiredTiger:
    engineConfig:
      cacheSizeGB: 0.5
      directoryForIndexes: true
    collectionConfig:
      blockCompressor: zlib
    indexConfig:
      prefixCompression: true
processManagement:
  fork: true
net:
   port: 20001
   bindIp: 10.0.0.52,127.0.0.1
replication:
  oplogSizeMB: 2048
  replSetName: US_sh
sharding:
  clusterRole: shardsvr
EOF
\cp  /mongodb/20001/conf/mongod.conf  /mongodb/20002/conf/
\cp  /mongodb/20001/conf/mongod.conf  /mongodb/20003/conf/
sed 's#20001#20002#g' /mongodb/20002/conf/mongod.conf -i
sed 's#20001#20003#g' /mongodb/20003/conf/mongod.conf -i


mongod -f /mongodb/20001/conf/mongod.conf
mongod -f /mongodb/20002/conf/mongod.conf
mongod -f /mongodb/20003/conf/mongod.conf

# db01:
mkdir -p /mongodb/20004/conf /mongodb/20004/data /mongodb/20004/log
```

```
mkdir -p /mongodb/20005/conf /mongodb/20005/data /mongodb/20005/log
mkdir -p /mongodb/20006/conf /mongodb/20006/data /mongodb/20006/log
cat > /mongodb/20004/conf/mongod.conf <<EOF
systemLog:
  destination: file
  path: /mongodb/20004/log/mongodb.log
  logAppend: true
storage:
  journal:
    enabled: true
  dbPath: /mongodb/20004/data
  directoryPerDB: true
  #engine: wiredTiger
  wiredTiger:
    engineConfig:
      cacheSizeGB: 0.5
      directoryForIndexes: true
    collectionConfig:
      blockCompressor: zlib
    indexConfig:
      prefixCompression: true
processManagement:
  fork: true
net:
   port: 20004
   bindIp: 10.0.0.51,127.0.0.1
replication:
  oplogSizeMB: 2048
  replSetName: config
sharding:
  clusterRole: configsvr
EOF
\cp  /mongodb/20004/conf/mongod.conf  /mongodb/20005/conf/
\cp  /mongodb/20004/conf/mongod.conf  /mongodb/20006/conf/
sed 's#20004#20005#g' /mongodb/20005/conf/mongod.conf -i
sed 's#20004#20006#g' /mongodb/20006/conf/mongod.conf -i


mongod -f /mongodb/20004/conf/mongod.conf
mongod -f /mongodb/20005/conf/mongod.conf
mongod -f /mongodb/20006/conf/mongod.conf


复制集配置


config = {_id: 'CN_sh', members: [
                          {_id: 0, host: '10.0.0.51:20001'},
                          {_id: 1, host: '10.0.0.51:20002'},
                          {_id: 2, host: '10.0.0.52:20003'}]
          }

rs.initiate(config)


config = {_id: 'US_sh', members: [
                          {_id: 0, host: '10.0.0.52:20001'},
                          {_id: 1, host: '10.0.0.52:20002'},
```

```
                          {_id: 2, host: '10.0.0.51:20003'}]
          }

rs.initiate(config)



config = {_id: 'config', members: [
                          {_id: 0, host: '10.0.0.51:20004'},
                          {_id: 1, host: '10.0.0.51:20005'},
                          {_id: 2, host: '10.0.0.51:20006'}]
          }

rs.initiate(config)


# mongos配置

mkdir -p /mongodb/20010/conf  /mongodb/20010/log

cat > /mongodb/20010/conf/mongos.conf<<EOF
systemLog:
  destination: file
  path: /mongodb/20010/log/mongos.log
  logAppend: true
net:
  bindIp: 10.0.0.51,127.0.0.1
  port: 20010
sharding:
  configDB: config/10.0.0.51:20004,10.0.0.51:20005,10.0.0.51:20006
processManagement:
  fork: true
EOF
mongos -f /mongodb/20010/conf/mongos.conf



mkdir -p /mongodb/20011/conf  /mongodb/20011/log

cat>  /mongodb/20011/conf/mongos.conf <<EOF
systemLog:
  destination: file
  path: /mongodb/20011/log/mongos.log
  logAppend: true
net:
  bindIp: 10.0.0.52,127.0.0.1
  port: 20011
sharding:
  configDB: config/10.0.0.51:20004,10.0.0.51:20005,10.0.0.51:20006
processManagement:
  fork: true
EOF
mongos -f /mongodb/20011/conf/mongos.conf

db.runCommand( { addshard :
"CN_sh/10.0.0.51:20001,10.0.0.51:20002,10.0.0.52:20003",name:"CN_sh"} )
db.runCommand( { addshard :
"US_sh/10.0.0.52:20001,10.0.0.52:20002,10.0.0.51:20003",name:"US_sh"} )
```




```
sh.addShardTag("CN_sh", "ASIA")
sh.addShardTag("US_sh", "AMERICA")

sh.addTagRange( "crm.orders", { "locationCode" : "CN", "order_id" : MinKey
},
{ "locationCode" : "CN", "order_id" : MaxKey } ,
"ASIA" )
sh.addTagRange( "crm.orders",
{ "locationCode" : "US", "order_id" : MinKey },
{ "locationCode" : "US", "order_id" : MaxKey } ,
"AMERICA" )
sh.addTagRange( "crm.orders",
{ "locationCode" : "CA", "order_id" : MinKey },
{ "locationCode" : "CA", "order_id" : MaxKey } ,
"AMERICA" )


```

7. MongoDB 备份与恢复及迁移

7.1 备份恢复工具介绍:

```
（1）**   mongoexport/mongoimport
（2）***** mongodump/mongorestore
```

7.2 备份工具区别在哪里?

```
mongoexport/mongoimport  导入/导出的是JSON格式或者CSV格式，
mongodump/mongorestore导入/导出的是BSON格式。

应用场景:
mongoexport/mongoimport:json csv
1、异构平台迁移  mysql  <---> mongodb
2、同平台，跨大版本：mongodb 2  ----> mongodb 3
mongodump/mongorestore
日常备份恢复时使用.
```

7.3 导出工具mongoexport

```
Mongodb中的mongoexport工具可以把一个collection导出成JSON格式或CSV格式的文件。
可以通过参数指定导出的数据项，也可以根据指定的条件导出数据。
（1）版本差异较大
（2）异构平台数据迁移

mongoexport具体用法如下所示：
$ mongoexport --help
参数说明：
-h:指明数据库宿主机的IP
-u:指明数据库的用户名
-p:指明数据库的密码
-d:指明数据库的名字
-c:指明collection的名字
-f:指明要导出那些列
-o:指明到要导出的文件名
-q:指明导出数据的过滤条件
--authenticationDatabase admin

1.单表备份至json格式
use test
for(i=0;i<10000;i++){ db.log.insert({"uid":i,"name":"mongodb","age":6,"date":new Date()}); }
mongoexport -uroot -proot123 --port 27017 --authenticationDatabase admin -d test -c log -o /mongodb/log.json
注：备份文件的名字可以自定义，默认导出了JSON格式的数据。
2. 单表备份至csv格式
如果我们需要导出CSV格式的数据，则需要使用--type=csv参数：
 mongoexport -uroot -proot123 --port 27017 --authenticationDatabase admin -d test -c log --type=csv -f uid,name,age,date  -o /mongodb/log.csv
```

7.4 导入工具mongoimport

```
Mongodb中的mongoimport工具可以把一个特定格式文件中的内容导入到指定的collection中。该工具可以导入JSON格式数据，也可以导入CSV格式数据。具体使用如下所示：
$ mongoimport --help
参数说明：
-h:指明数据库宿主机的IP
-u:指明数据库的用户名
-p:指明数据库的密码
-d:指明数据库的名字
-c:指明collection的名字
-f:指明要导入那些列
```

```
-j, --numInsertionWorkers=<number>  number of insert operations to run concurrently                                                  (defaults to 1)
//并行


数据恢复:
1.恢复json格式表数据到log1
mongoimport -uroot -proot123 --port 27017 --authenticationDatabase admin -d test -c log1 /mongodb/log.json

2.恢复csv格式的文件到log2
上面演示的是导入JSON格式的文件中的内容，如果要导入CSV格式文件中的内容，则需要通过--type参数指定导入格式，具体如下所示：
错误的恢复

注意:
（1）csv格式的文件头行，有列名字
mongoimport   -uroot -proot123 --port 27017 --authenticationDatabase admin  -d test -c log2 --type=csv --headerline --file  /mongodb/log.csv

（2）csv格式的文件头行，没有列名字
mongoimport   -uroot -proot123 --port 27017 --authenticationDatabase admin  -d test -c log3 -j 4 --type=csv -f id,name,age,date --file /mongodb/log.csv

--headerline:指明第一行是列名，不需要导入。

3. 异构平台迁移案例
mysql   -----> mongodb
world数据库下city表进行导出，导入到mongodb

（1）mysql开启安全路径
vim /etc/my.cnf   --->添加以下配置
secure-file-priv=/data/backup/
--重启数据库生效
/etc/init.d/mysqld restart

（2）导出mysql的city表数据
select * from test.t100w into outfile '/tmp/t100w.csv' fields terminated by ','   ENCLOSED BY '"' ;

（3）获取列信息
mysql> select table_name,group_concat(column_name)  from information_schema.columns where table_schema='test' group by table_name order by null ;
+------------+--------------------------+
| TABLE_NAME | group_concat(column_name) |
+------------+--------------------------+
| t100w      | dt,id,k1,k2,num           |
+------------+--------------------------+
1 row in set (0.07 sec)

(4)在mongodb中导入备份
mongoimport -uroot -proot123 --port 27017 --authenticationDatabase admin -d test -c t100w  --type=csv -f id,num，k1,k2,,dt --file  /tmp/t100w.csv
use world
db.t100w.find({});
```

```

4. 彩蛋----如何将MySQL大量表迁移到MongoDB
痛点:
（1） 批量从MySQL导出多张表
mysqldump   --fields-terminated-by ',' --fields-enclosed-by '"'  world -T /tmp/
cd /data/backup
rm -rf /data/backup/*.sql
find ./  -name "*.txt" | awk -F "." '{print $2}' | xargs -i -t mv ./{}.txt ./{}.csv

(2) 拼接语句
select concat("mongoimport -uroot -proot123 --port 27017 --authenticationDatabase admin -d ",table_schema, " -c  ",table_name ," --type=csv "," -f ", group_concat(column_name) ," --file /data/backup/",table_name ,".csv")
from information_schema.columns where table_schema='world' group by table_name;

(3) 导入数据
[mongod@db01 backup]$ ll
total 256
-rwxrwxrwx 1 mysql mysql 184355 Jul 22 13:45 city.csv
-rwxrwxrwx 1 mysql mysql  38659 Jul 22 13:45 country.csv
-rwxrwxrwx 1 mysql mysql  26106 Jul 22 13:45 countrylanguage.csv
-rwxrwxrwx 1 mysql mysql    656 Jul 22 13:45 import.sh
[mongod@db01 backup]$ sh import.sh


```

7.5 mongodump和mongorestore

a. 介绍

```
mongodump能够在Mongodb运行时进行备份，它的工作原理是对运行的Mongodb做查询，然后将所有查到的文档写入磁盘。但是存在的问题时使用mongodump产生的备份不一定是数据库的实时快照，如果我们在备份时对数据库进行了写入操作，则备份出来的文件可能不完全和Mongodb实时数据相等。另外在备份时可能会对其它客户端性能产生不利的影响。
```

b. mongodump参数

```
$ mongodump --help
参数说明:
-h:指明数据库宿主机的IP
-u:指明数据库的用户名
-p:指明数据库的密码
-d:指明数据库的名字
-c:指明collection的名字
-o:指明到要导出的文件名
-q:指明导出数据的过滤条件
-j, --numParallelCollections=  number of collections to dump in parallel (4 by default)
--oplog  备份的同时备份oplog

```

c. mongodump和mongorestore基本使用

```
全库备份
mkdir /mongodb/backup -p
mongodump  -uroot -proot123 --port 27017 --authenticationDatabase admin -o /mongodb/backup

--备份world库
$ mongodump   -uroot -proot123 --port 27017 --authenticationDatabase admin -d test -o /mongodb/backup/

--备份test库下的log集合
$ mongodump   -uroot -proot123 --port 27017 --authenticationDatabase admin -d test -c log -o /mongodb/backup/

 --压缩备份
$ mongodump   -uroot -proot123 --port 27017 --authenticationDatabase admin -d abc -o /mongodb/backup/ --gzip
 mongodump   -uroot -proot123 --port 27017 --authenticationDatabase admin -o /mongodb/backup/ --gzip
$ mongodump   -uroot -proot123 --port 27017 --authenticationDatabase admin -d app -c vast -o /mongodb/backup/ --gzip

--全备中恢复单库
$ mongorestore   -uroot -proot123 --port 27017 --authenticationDatabase admin -d world1  /mongodb/backup/world

--全备中恢复单表
[mongod@db01 backup]$ mongorestore   -uroot -proot123 --port 27017 --authenticationDatabase admin -d a -c t1   /mongodb/backup/world/t5.bson.gz --gzip

--drop表示恢复的时候把之前的集合drop掉(危险)
$ mongorestore  -uroot -proot123 --port 27017 --authenticationDatabase admin -d test --drop  /mongodb/backup/test
```

d. mongodump和mongorestore高级企业应用（--oplog)

```
注意：这是replica set模式专用
--oplog
 use oplog for taking a point-in-time snapshot

# oplog介绍
在replica set中oplog是一个定容集合（capped collection），它的默认大小是磁盘空间的5%（可以通过--oplogSizeMB参数修改）.

位于local库的db.oplog.rs，有兴趣可以看看里面到底有些什么内容。
其中记录的是整个mongod实例一段时间内数据库的所有变更（插入/更新/删除）操作。
当空间用完时新记录自动覆盖最老的记录。
其覆盖范围被称作oplog时间窗口。需要注意的是，因为oplog是一个定容集合，
所以时间窗口能覆盖的范围会因为你单位时间内的更新次数不同而变化。
想要查看当前的oplog时间窗口预计值，可以使用以下命令：

 mongod -f /mongodb/28017/conf/mongod.conf
```

```
 mongod -f /mongodb/28018/conf/mongod.conf
 mongod -f /mongodb/28019/conf/mongod.conf
 mongod -f /mongodb/28020/conf/mongod.conf

 use local
 db.oplog.rs.find().pretty()


    "ts" : Timestamp(1553597844, 1),
    "op" : "n"
    "o"  :

    "i": insert
    "u": update
    "d": delete
    "c": db cmd

test:PRIMARY> rs.printReplicationInfo()
configured oplog size:   1561.5615234375MB <--集合大小
log length start to end: 423849secs (117.74hrs) <--预计窗口覆盖时间
oplog first event time:  Wed Sep 09 2015 17:39:50 GMT+0800 (CST)
oplog last event time:   Mon Sep 14 2015 15:23:59 GMT+0800 (CST)
now:                     Mon Sep 14 2015 16:37:30 GMT+0800 (CST)


# oplog企业级应用
（1）实现热备，在备份时使用--oplog选项
注：为了演示效果我们在备份过程，模拟数据插入
（2）准备测试数据
use test

for(var i = 1 ;i < 100; i++) {
    db.foo.insert({a:i});
}

my_repl:PRIMARY> db.oplog.rs.find({"op":"i"}).pretty()


oplog 配合mongodump实现热备
mongodump --port 28017 --oplog -o /mongodb/backup

作用介绍：--oplog 会记录备份过程中的数据变化。会以oplog.bson保存下来

恢复
mongorestore  --port 28017 --oplogReplay /mongodb/backup


!!!!!!!!!!oplog高级应用  ==========binlog应用

背景：每天0点全备，oplog恢复窗口为48小时
某天，上午10点world.city 业务表被误删除。

恢复思路：
    0、停应用
    2、找测试库
    3、恢复昨天晚上全备
    4、截取全备之后到world.city误删除时间点的oplog，并恢复到测试库
    5、将误删除表导出，恢复到生产库
```

```


恢复步骤:
模拟故障环境:
1、全备数据库
模拟原始数据
mongo --port 28017
use wo
for(var i = 1 ;i < 20; i++) {
    db.ci.insert({a: i});
}

全备:
rm -rf /mongodb/backup/*
mongodump --port 28017 --oplog -o /mongodb/backup
--oplog功能:在备份同时,将备份过程中产生的日志进行备份
文件必须存放在/mongodb/backup下,自动命令为oplog.bson
再次模拟数据
db.ci1.insert({id:1})
db.ci2.insert({id:2})
2、上午10点:删除wo库下的ci表
10:00时刻,误删除

db.ci.drop()
show tables;

3、备份现有的oplog.rs表
mongodump --port 28017 -d local -c oplog.rs  -o /mongodb/backup

4、截取oplog并恢复到drop之前的位置
更合理的方法:登陆到原数据库
[mongod@db03 local]$ mongo --port 28017
my_repl:PRIMARY> use local
db.oplog.rs.find({op:"c"}).pretty();
{
    "ts" : Timestamp(1600489082, 1),
    "t" : NumberLong(1),
    "h" : NumberLong(0),
    "v" : 2,
    "op" : "c",
    "ns" : "wo.$cmd",
    "ui" : UUID("875f2a41-57e3-4b6b-b738-469fad032b18"),
    "o2" : {
        "numRecords" : 19
    },
    "wall" : ISODate("2020-09-19T04:18:02.717Z"),
    "o" : {
        "drop" : "ci"
    }
}

获取到oplog误删除时间点位置:
1600489082, 1

 5、恢复备份+应用oplog

[mongod@db03 backup]$ cd /mongodb/backup/local/
[mongod@db03 local]$ ls
```

```
oplog.rs.bson  oplog.rs.metadata.json
[mongod@db03 local]$ cp oplog.rs.bson ../oplog.bson
rm -rf /mongodb/backup/local/

mongorestore --port 28017  --oplogReplay --oplogLimit "1600489082:1"  --drop   /mongodb/backup/



6. 分片集群的备份思路
6.1 需要备份的数据
shard  、  configserver

6.2 痛点
chunk 迁移 ，关闭或者调整balancer时间窗口
备份出来的数据时间不一致。

6.3
a.使用Ops Manager
b.
balancer  关闭--->
同一时刻config、shard其中一个节点脱离集群--->
开始备份节点数据 --->
把节点恢复到集群

PSS
PSA


```

9. MongoDB 监控实践

9.1 常用的监控工具及手段

```
• MongoDB Ops Manager
• Percona
• 通用监控平台
• 程序脚本
```

9.2 如何获取监控数据

```
监控信息的来源：
• db.serverStatus()（主要）
• db.isMaster()（次要）
• mongostats 命令行工具（只有部分信息）
说明：db.serverStatus() 包含的监控信息是从上次开机到现在为止的累计数据.
```

9.3 serverStatus() Output

```
• host
• version
• process
• pid
• uptime
```

```
• uptimeMillis
• uptimeEstimate
• localTime
• asserts
• connections
• electionMetrics
• extra_info
• flowControl
• freeMonitoring
• flowControl
• freeMonitoring
• globalLock
• locks
• network
• opLatencies
• opReadConcernCounters
• opcounters
• opcountersRepl
• oplogTruncation
• repl
• storageEngine
• tcmalloc
• trafficRecording
• transactions
• transportSecurity
• wiredTiger
```

9.4 serverStatus() 主要信息

```
• connections: 关于连接数的信息；
• locks: 关于 MongoDB 使用的锁情况；
• network: 网络使用情况统计；
• opcounters: CRUD 的执行次数统计；
• repl: 复制集配置信息；
• wiredTiger: 包含大量 WirdTiger 执行情况的信息：
    block-manager: WT 数据块的读写情况；
    session: session 使用数量；
    concurrentTransactions: Ticket 使用情况；
• mem: 内存使用情况；
• metrics: 一系列性能指标统计信息；
```

9.5 监控报警的考量

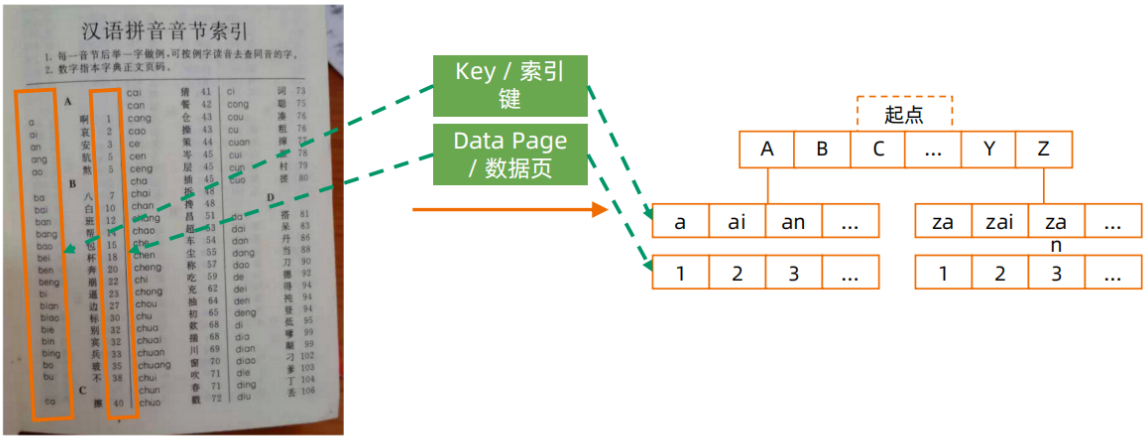
```
• 具备一定的容错机制以减少误报的发生；
• 总结应用各指标峰值；
• 适时调整报警阈值；
• 留出足够的处理时间；
```

9.6 建议监控指标说明

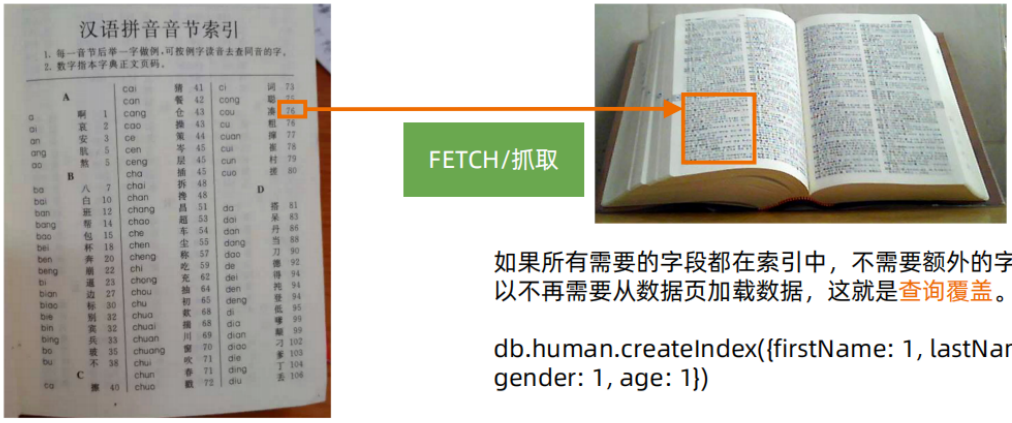
指标	功能	采集方法
opcounters（操作计数器）	查询、更新、插入、删除、getmore 和其他命令的的数量。	db.serverStatus().opcounters
tickets（令牌）	对 WiredTiger 存储引擎的读/写令牌数量。令牌数量表示了可以进入存储引擎的并发操作数量。	db.serverStatus().wiredTiger.concurrentTransactions
replication lag（复制延迟）	这个指标代表了写操作到达从结点所需要的最小时间。过高的 replication lag 会减小从结点的价值并且不利于配置了写关 注 w>1 的那些操作。	db.adminCommand({'replSetGetStatus': 1})
oplog window（复制时间窗）	这个指标代表oplog可以容纳多长时间的写操作。它表示了一个从结点可以离线多长时间仍能够追上主节点。通常建议该值应大于24小时为佳。	db.oplog.rs.find().sort({$natural: -1}).limit(1).next().ts -db.oplog.rs.find().sort({$natural: 1}).limit(1).next().ts
connections（连接数）	连接数应作为监控指标的一部分，因为每个连接都将消耗资源。应该计算低峰/正常/高峰时间的连接数，并制定合理的报警阈值范围。	db.serverStatus().connections
Query targeting（查询专注度）	索引键/文档扫描数量比返回的文档数量，按秒平均。如果该值比较高表示查询系需要进行很多低效的扫描来满足查询。这个情况通常代表了索引不当或缺少索引来支持查询。	var status = db.serverStatus() status.metrics.queryExecutor.scanned / status.metrics.document.returned status.metrics.queryExecutor.scannedObjects / status.metrics.document.returned
Scan and Order（扫描和排序）	每秒内内存排序操作所占的平均比例。内存排序可能会十分昂贵，因为它们通常要求缓冲大量数据。如果有适当索引的情况下，内存排序是可以避免的。	var status = db.serverStatus() status.metrics.operation.scanAndOrder / status.opcounters.query
复制集节点状态	每个节点的运行状态。如果节点状态不是PRIMARY、SECONDARY、ARBITER 中的一个，或无法执行上述命令则报警	db.runCommand("isMaster")
dataSize（数据大小）	整个实例数据总量（压缩前）	每个 DB 执行db.stats();
StorageSize（磁盘空间 大小）	已使用的磁盘空间占总空间的百分比。	每个 DB 执行db.stats();

10.MongoDB 索引管理

10.1 Index/Key/DataPage——索引/键/数据页?



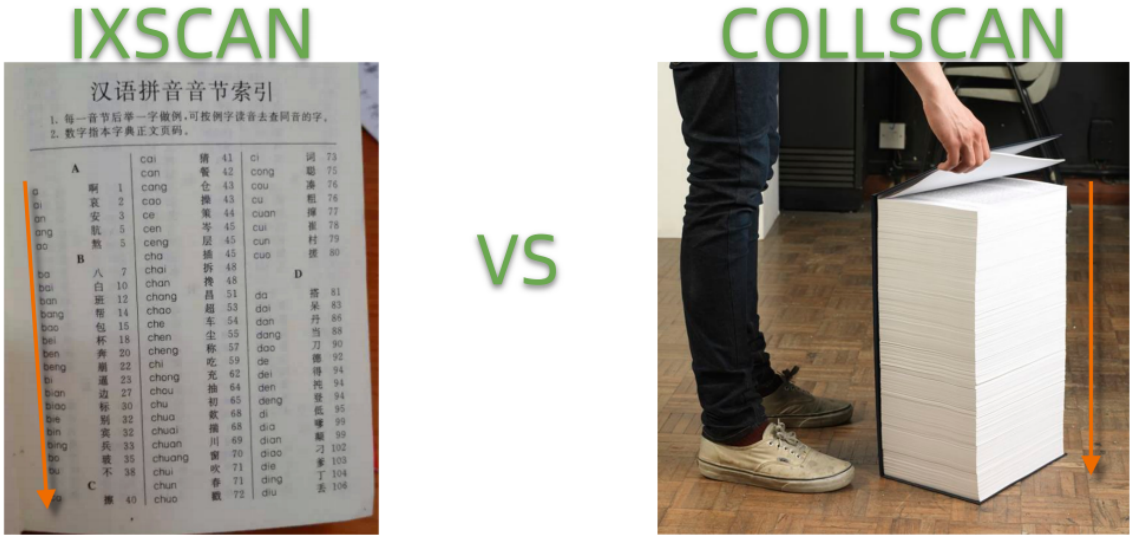
10.2 Covered Query



如果所有需要的字段都在索引中，不需要额外的字段，就可以不再需要从数据页加载数据，这就是查询覆盖。

db.human.createIndex({firstName: 1, lastName: 1, gender: 1, age: 1})

10.3 IXSCAN/COLLSCAN



10.4 复合索引

```
db.human.createIndex({firstName: 1, lastName: 1, gender: 1, age: 1})
```

以上索引的全部前缀包括：

- {firstName: 1}
- {firstName: 1, lastName: 1}
- {firstName: 1, lastName: 1, gender: 1}

所有索引前缀都可以被该索引覆盖，没有必要针对这些查询建立额外的索引

10.5 Selectivity——过滤性

```
在一个有10000条记录的集合中：
• 满足 gender= F 的记录有4000 条
• 满足 city=LA 的记录有 100 条
• 满足 ln='parker' 的记录有 10 条
查询条件：
ln=10 city=SZ  gender=F
条件 ln 能过滤掉最多的数据，city
其次，gender 最弱。
所以 ln 的过滤性（selectivity）> city > gender。

```

10.6 获取执行计划

```
-- 写入10000条文档
for (var i=1;i<100000; i++)
        db.col.insert(
                {name:i, age:i,
date:new Date() } )

-- 查询
db.col.find({name:1111}).explain(true)
```

```
>>"executionStats" :
{       "executionSuccess" : true,
        "nReturned" : 1,
        "executionTimeMillis" : 58,
        "totalKeysExamined" : 0,
        "totalDocsExamined" : 99999,
        "executionStages" : {
                "stage" : "COLLSCAN",
                 "filter" : {"name" : {"$eq" : 1111}},
                "nReturned" : 1,
                "executionTimeMillisEstimate" : 53,
                "works" : 100001,
                "advanced" : 1,
                "needTime" : 99999,
                "needYield" : 0,
                "saveState" : 783,
                "restoreState" : 783,
                "isEOF" : 1,
                "invalidates" : 0,
                "direction" : "forward",
                "docsExamined" : 99999
```

10.6 优化后的执行计划

```
-- 写入10000条文档
for (var i=1;i<100000; i++)
        db.col.insert(
                {name:i, age:i,
date:new Date() } )


-- 创建name索引
db.col.createIndex({name:1})


-- 查询
db.col.find({name:1111}).explain(true)
```

```
>> "executionStats" : {
                "executionSuccess" : true,
                "nReturned" : 1,
                "executionTimeMillis" : 3,
                "totalKeysExamined" : 1,
                "totalDocsExamined" : 1,
                "executionStages" : {
                        "stage" : "FETCH",
                        "nReturned" : 1,

                "executionTimeMillisEstimate" : 0,
                        "docsExamined" : 1,
                        "alreadyHasObj" : 0,
                        "inputStage" : {
                                "stage" :
"IXSCAN",

        "nReturned" : 1,

        "executionTimeMillisEstimate" : 0,
                                "works" : 2,

        "advanced" : 1,
                ...
}
```

10.7 组合索引的最佳方式：ESR原则

```
组合索引的最佳方式：ESR原则
• 精确（Equal）匹配的字段放最前面
• 排序（Sort）条件放中间
• 范围（Range）匹配的字段放最后面
同样适用： ES, ER
请看一下查询条件：
db.members.find({ gender: “F”， age: {$gte: 18}}).sort(“join_date:1”)

{ gender: 1, age: 1, join_date: 1 }
{ gender: 1, join_date:1, age: 1 }
{ join_date: 1, gender: 1, age: 1 }
{ join_date: 1, age: 1, gender: 1 }
{ age: 1, join_date: 1, gender: 1}
{ age: 1, gender: 1, join_date: 1}
这么多候选的，用哪一个？


```

10.8 部分索引

```
-- 创建部分索引

>> db.<collection_name>.createIndex(
            {'a': 1},
            { partialFilterExpression:
                        {a:
                           {$gte:5}
                        }
)
```

```
-- 只对有wechat字段的建索引:

>> db.<collection_name>.createIndex(
            {'wechat': 1},
            { partialFilterExpression:
                        {wechat:
                           {$exists: true}
                        }
)
```

```
-- 索引目标文档

{ "_id" : ObjectId("5c7f4d8723a59b2f55f58ca5"), "a" : 1 }
{ "_id" : ObjectId("5c7f4d8723a59b2f55f58ca6"), "a" : 2 }
{ "_id" : ObjectId("5c7f4d8723a59b2f55f58ca7"), "a" : 3 }
{ "_id" : ObjectId("5c7f4d8723a59b2f55f58ca8"), "a" : 4 }
{ "_id" : ObjectId("5c7f4d8723a59b2f55f58ca9"), "a" : 5 }
{ "_id" : ObjectId("5c7f4d8723a59b2f55f58caa"), "a" : 6 }
{ "_id" : ObjectId("5c7f4d8723a59b2f55f58cab"), "a" : 7 }
{ "_id" : ObjectId("5c7f4d8723a59b2f55f58cac"), "a" : 8 }
{ "_id" : ObjectId("5c7f4d8723a59b2f55f58cad"), "a" : 9 }
{ "_id" : ObjectId("5c7f4d8723a59b2f55f58cae"), "a" : 10 }
```

11.MongoDB 性能诊断

11.1 问题诊断工具 - mongostat

```
insert query update delete getmore command dirty used flushes vsize   res qrw arw net_in net_out conn set repl                time
    *0    *0     *0     *0       0     2|0  0.1% 2.8%       0 5.11G 32.0M 0|0 1|0   174b   55.9k    1 rs0  PRI Oct 25 10:53:46.502
    *0    *0     *0     *0       0     1|0  0.1% 2.8%       0 5.11G 32.0M 0|0 1|0   157b   50.5k    1 rs0  PRI Oct 25 10:53:47.504
    *0    *0     *0     *0       0     2|0  0.1% 2.8%       0 5.11G 32.0M 0|0 1|0   158b   50.7k    1 rs0  PRI Oct 25 10:53:48.502
    *0    *0     *0     *0       0     1|0  0.1% 2.8%       0 5.11G 32.0M 0|0 1|0   157b   50.5k    1 rs0  PRI Oct 25 10:53:49.505
    *0    *0     *0     *0       0     2|0  0.1% 2.8%       0 5.11G 32.0M 0|0 1|0   158b   50.7k    1 rs0  PRI Oct 25 10:53:50.504
    *0    *0     *0     *0       0     2|0  0.1% 2.8%       0 5.11G 32.0M 0|0 1|0   158b   50.8k    1 rs0  PRI Oct 25 10:53:51.501
    *0    *0     *0     *0       0     2|0  0.1% 2.8%       0 5.11G 32.0M 0|0 1|0   158b   50.7k    1 rs0  PRI Oct 25 10:53:52.501
```

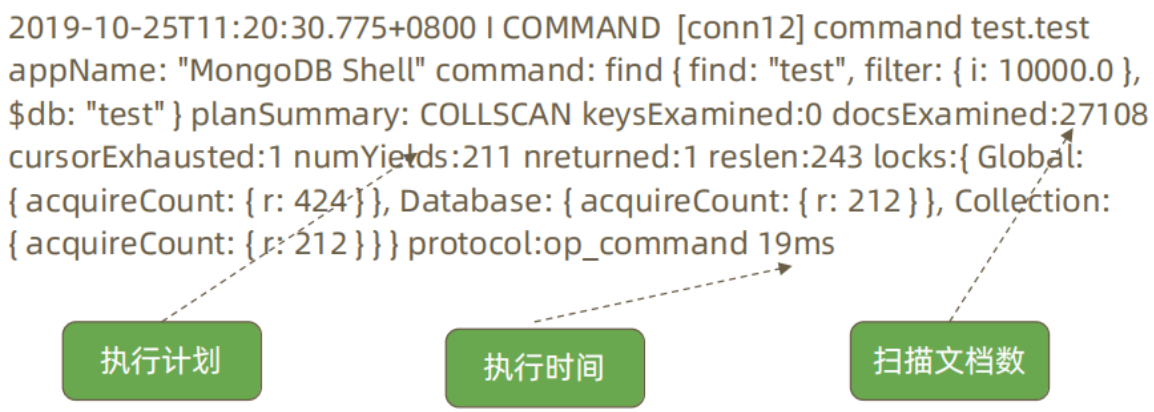
超过20%时阻塞新请求　　超过95%时阻塞新请求　　排队的请求　　连接数量

```
主要关注点：
dirty：
内存中的脏页数量百分比。默认在5%以下，每分钟在后台刷盘；超过5%后台进程频繁刷盘，超过20%会阻止新请求。
used：
分配给mongodb的内存使用的百分比。低于80%,不会触发内存回收,超过80%,触发LRU回收内存，默认超过95%会阻止新请求。
qrw: 排队的请求数量，超过10以上，需要关注下积压的操作是什么（慢查询、锁等）。
```

11.2 问题诊断工具 - mongotop

11.3 问题诊断 – mongod 慢日志

```
配置方法：
方法一：启动命令行
启动MongoDB时加上--profile=级别

方法二：配置文件
operationProfiling:
   mode: slowOp
   slowOpThresholdMs:  10

方法三：在线配置
db.setProfilingLevel(2);
db.setProfilingLevel( 1 , 10 );

上面profile的级别可以取0，1，2  三个值，他们表示的意义如下：
0 -  不开启
1 -  记录慢命令 (默认为>100ms)
2 -  记录所有命令

查询 Profiling 记录 ：
列出执行时间长于某一限度(100 ms)的 Profile 记录：
    db.system.profile.find( { millis : { $gt : 100 } } )

说明：  mongod的日志也会记录慢日志信息，会更加详细。如下面例子：
```



12. MongoDB的生产上线及版本升级

12.1 上线前：性能测试

```
模拟真实压力，对集群完成充分的性能测试，了解集群概况。
性能测试的输出：
• 压测过程中各项指标表现，例如 CRUD 达到多少，连接数达到多少等。
• 根据正常指标范围配置监控阈值；
• 根据压测结果按需调整硬件资源；
```

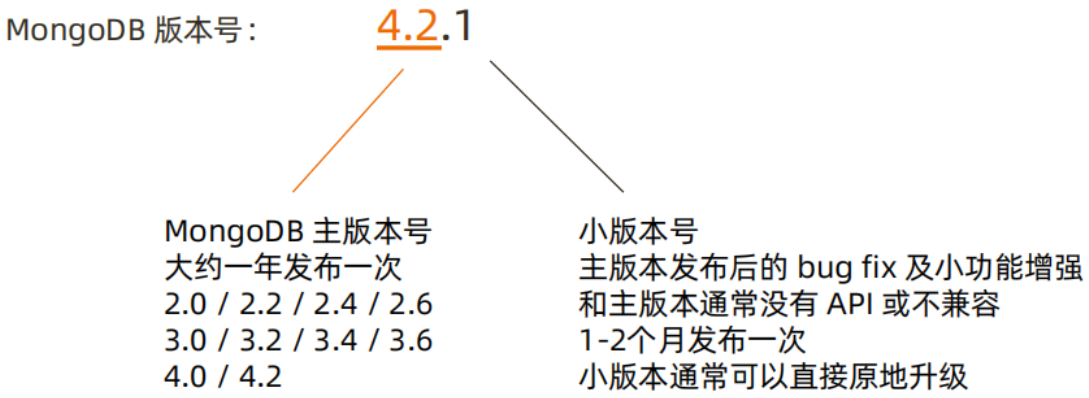
12.2 上线前：环境检查

```
按照最佳实践要求对生产环境所使用的操作系统进行检查和调整。最常见的需要调整的参数包括：
• 禁用 NUMA，否则在某些情况下会引起突发大量swap交换；
• 禁用 Transparent Huge Page，否则会影响数据库效率；
• tcp_keepalive_time 调整为120秒，避免一些网络问题；
• ulimit -n，避免打开文件句柄不足的情况；
• 关闭 atime，提高数据文件访问效率；
```

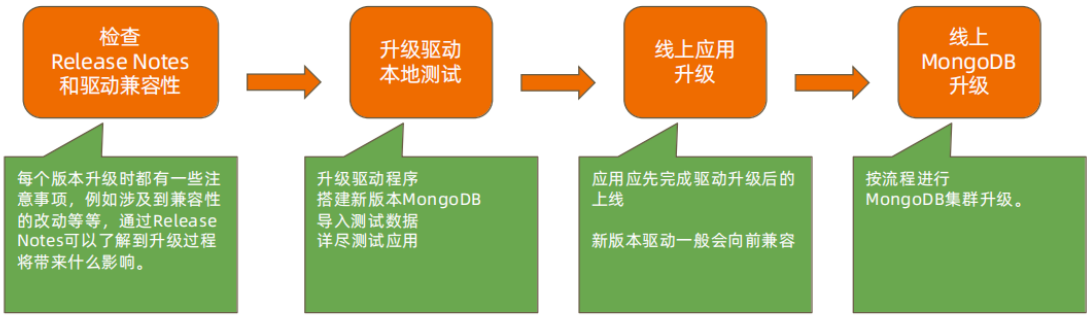
12.3 上线后

```
性能监控
为防止突发状况，应对常见性能指标进行监控以及时发现问题。
性能监控请参考前述章节的内容
定期健康检查
• mongod 日志；
• 环境设置是否有变动；
• MongoDB 配置是否有变动；
```

12.4 MongoDB 版本发布规律

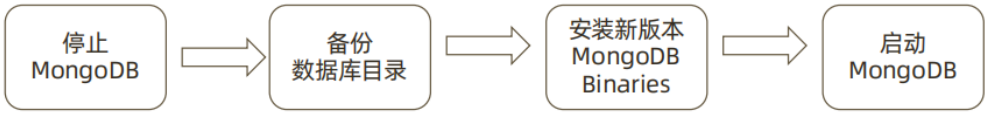


12.5 主版本升级流程




```
参考 ： https://docs.mongodb.com/ecosystem/drivers/driver-compatibility-reference/
```

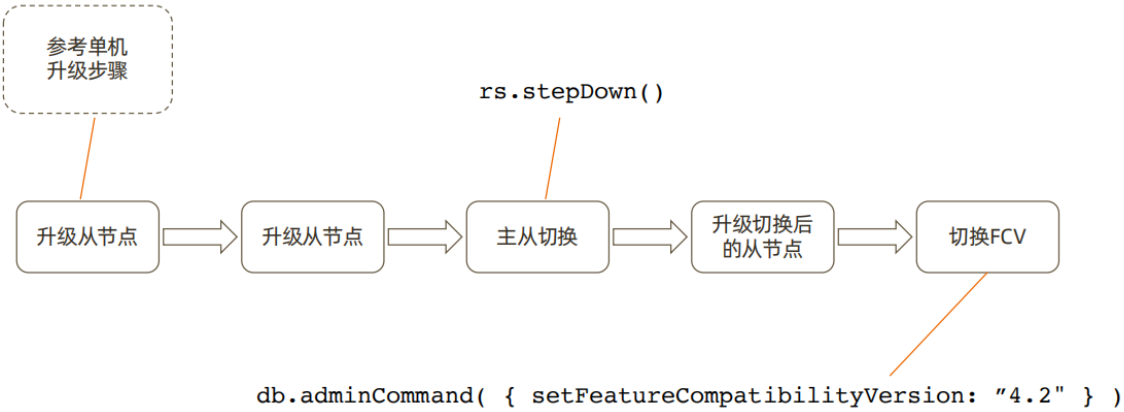
12.6 MongoDB 单机升级流程




```
# 升级模拟4.0.18升级4.2.8版本
1. 安装高版本软件至/data/app/mongodb
2. 停低版本实例
[mongod@db01 ~]$ /opt/mongodb/bin/mongod -f /data/mongo40/conf/mongo.conf --shutdown

killing process with pid: 4860
3. 启动后自动升级
[mongod@node01 ~]$ mongod -f /mongodb/conf/mongo.conf
about to fork child process, waiting until server is ready for connections.
forked process: 6376
child process started successfully, parent exiting
4. 检测版本
```

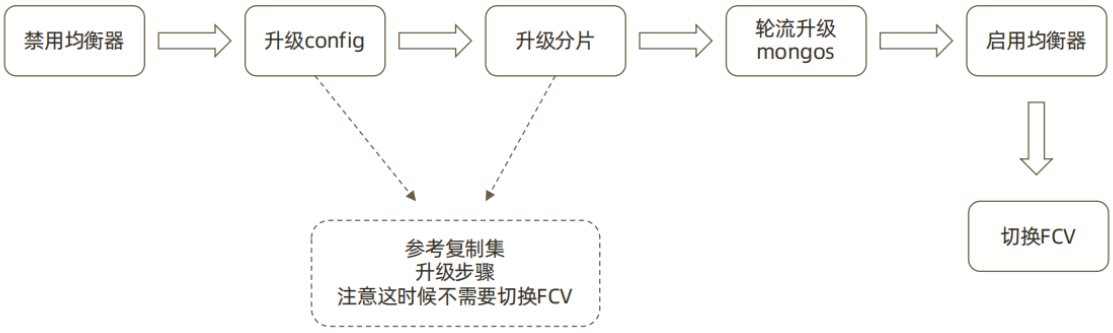
```
> db.version()
4.2.8

5. 切换FCV
> db.adminCommand( { setFeatureCompatibilityVersion: "4.2" } )
{ "ok" : 1 }
> db.adminCommand( { getParameter: 1, featureCompatibilityVersion: 1 } )
{ "featureCompatibilityVersion" : { "version" : "4.2" }, "ok" : 1 }


```

12.7 MongoDB 复制集升级流程




```
1、升级从节点
如单机升级步骤。
2、将主节点身份切换并升级
rs.stepdown()
升级步骤如单机升级流程。
3、切换FCV
> db.adminCommand( { getParameter: 1, featureCompatibilityVersion: 1 } )
> db.adminCommand( { setFeatureCompatibilityVersion: "4.4" } )
{ "ok" : 1 }
> db.adminCommand( { getParameter: 1, featureCompatibilityVersion: 1 } )
{ "featureCompatibilityVersion" : { "version" : "4.4" }, "ok" : 1 }


```

12.8 MongoDB 分片集群升级流程



```
1、禁用均衡器
sh.stopBalancer()
2、升级config（从-->主）
参考复制集升级过程
3、升级分片
参考复制集升级过程
4、升级mongos(建议多个mongos)
轮流升级
5. 启动均衡器
sh.startBalancer()
```

12.9 版本升级：在线升级注意

```
MongoDB支持在线升级，即升级过程中不需要间断服务；
升级过程中虽然会发生主从节点切换，存在短时间不可用，但是：
• 3.6版本开始支持自动写重试可以自动恢复主从切换引起的集群暂时不可写；
• 4.2开始支持的自动读重试则提供了包括主从切换在内的读问题的自动恢复；
升级需要逐版本完成，不可以跳版本：
• 正确：3.2->3.4->3.6->4.0->4.2
• 错误：3.2->4.2
原因：
• MongoDB复制集仅仅允许相邻版本共存
• 有一些升级内部数据格式如密码加密字段，需要在升级过程中由mongo进行转换
```

12.10 降级

```
• 如果升级无论因何种原因失败，则需要降级到原有旧版本。在降级过程中：
• 滚动降级过程中集群可以保持在线，仅在切换节点时会产生一定的不可写时间；
• 降级前应先去除已经用到的新版本特性。例如：用到了 NumberDecimal 则应把所有使用 NumberDecimal 的文档先去除该字段；
• 通过设置 FCV（Feature Compatibility Version）可以在功能上降到与旧版本兼容；
• FCV 设置完成后再滚动替换为旧版本。
```

13、异构平台在线迁移

13.1 为什么要迁移至MongoDB

13.1.1 MongoDB数据库定位

```
• OLTP /OLAP 数据库
• 原则上 Oracle 和 MySQL 能做的事情，MongoDB 都能做（包括 ACID 事务）
• 优点：横向扩展能力，数据量或并发量增加时候架构可以自动扩展
• 优点：灵活模型，适合迭代开发，数据模型多变场景
• 优点：JSON 数据结构，适合微服务/REST API
```

13.1.2 基于功能选择 MongoDB

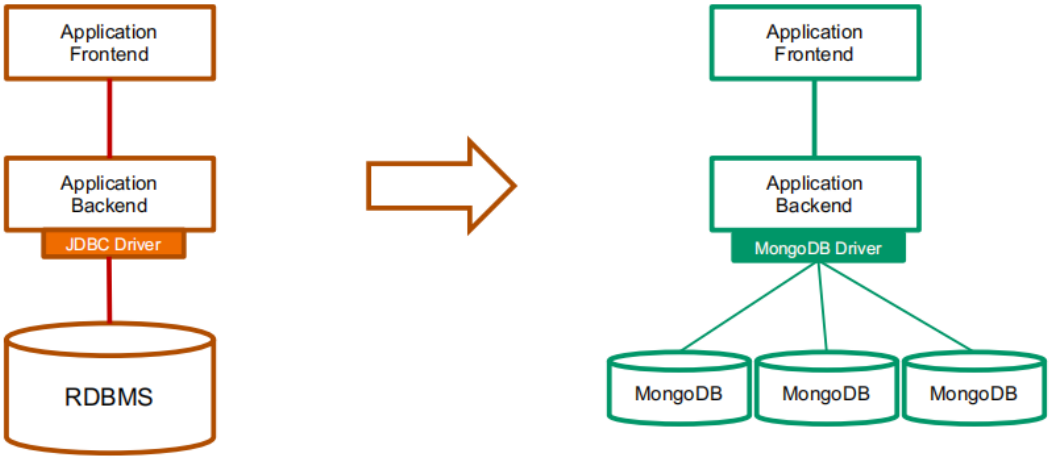
	MongoDB	关系型数据库
亿级以上数据量	轻松支持	要努力一下，分库分表
灵活表结构	轻松支持	Entity Key /Value 表，关联查询比较痛苦
高并发读	轻松支持	需要优化
高并发写	轻松支持	需要优化
跨地区集群	轻松支持	需要定制方案
分片集群	轻松支持	需要中间件
地理位置查询	比较完整的地理位置	PG 还可以，其他数据库略麻烦
聚合计算	功能很强大	使用 Group By 等，能力有限
异构数据	轻松支持	使用 EKV 属性表
大宽表	轻松支持	性能受限

13.1.3 基于场景选择 MongoDB

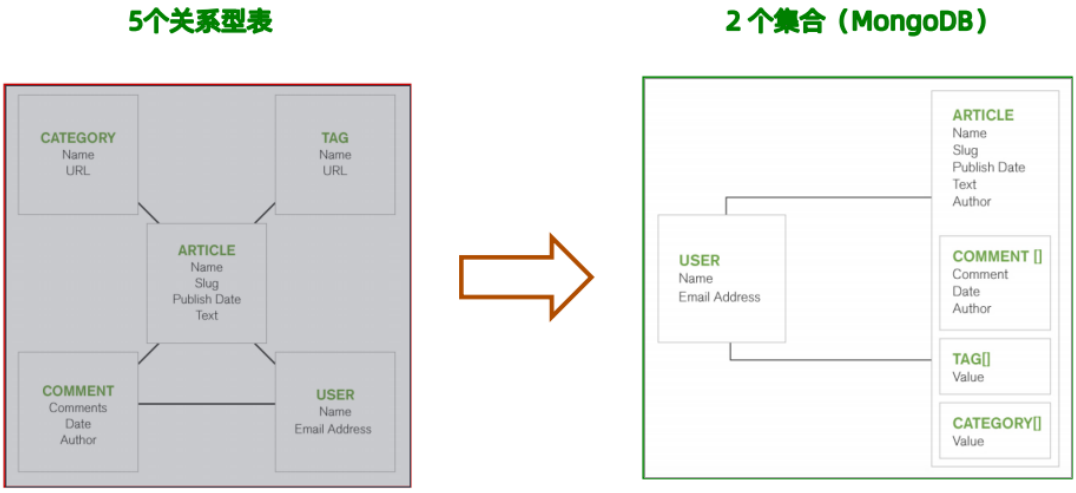
13.2 迁移痛点

```
关系型到关系型 - 相对简单
• Oracle -> MySQL, Oracle - PostgreSQL
关系型到文档型 - 相对复杂
• Oracle -> MongoDB
需要考虑:
• 总体架构 (从单体式到分布式)
• 模式设计（从关系模型到文档模型）
• SQL 语句 / 存储过程 / JDBC / ORM
• 数据迁移 （如何处理已有数据？）
```

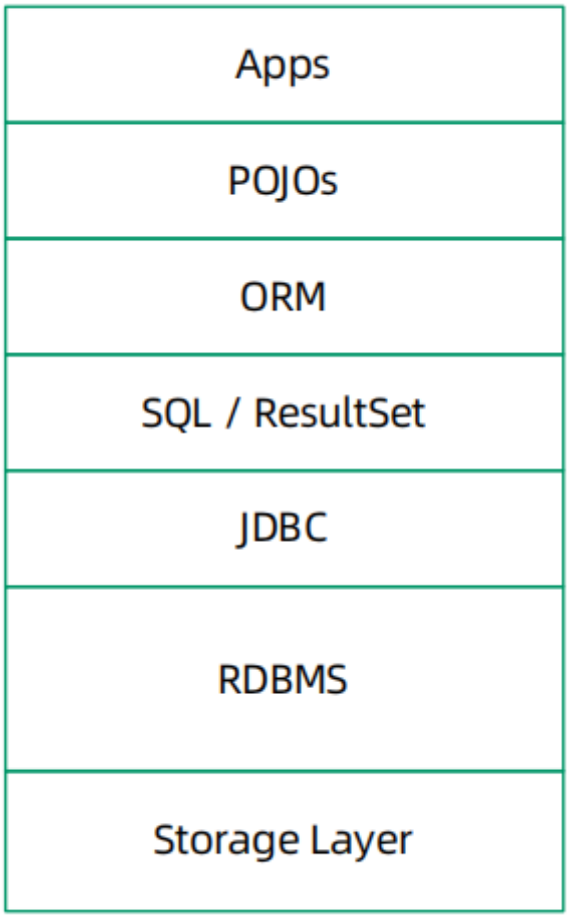
单点到分布式


Application Frontend
Application Backend
JDBC Driver
RDBMS
Application Frontend
Application Backend
MongoDB Driver
MongoDB
MongoDB
MongoDB

模式转换


5个关系型表
2 个集合（MongoDB）
CATEGORY
Name
URL
TAG
Name
URL
ARTICLE
Name
Slug
Publish Date
Text
COMMENT
Comments
Date
Author
USER
Name
Email Address
ARTICLE
Name
Slug
Publish Date
Text
Author
USER
Name
Email Address
COMMENT []
Comment
Date
Author
TAG[]
Value
CATEGORY[]
Value

迁移主流程

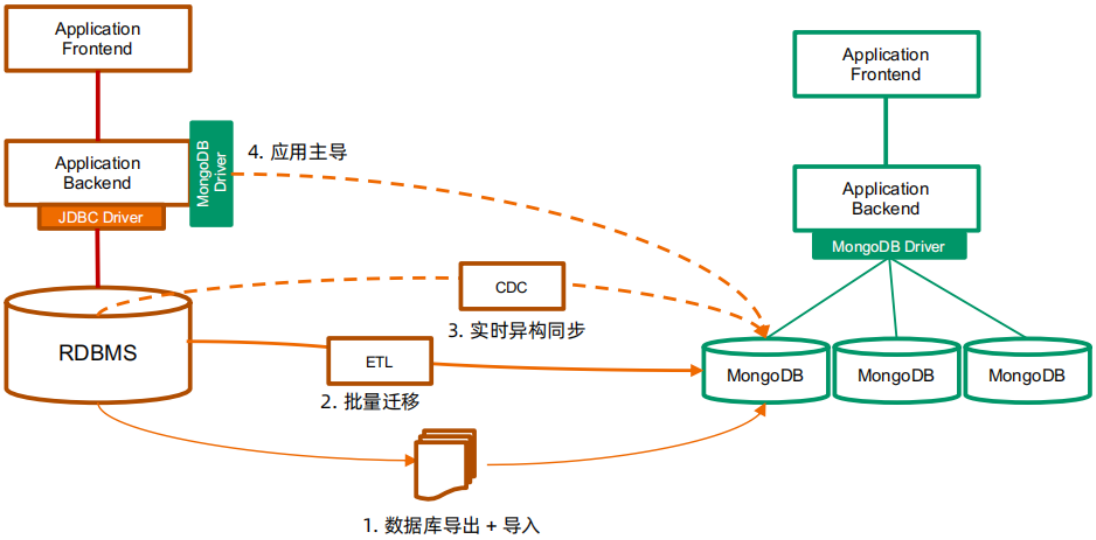



```
RDBMS:
• 存储过程
• 运维工具、脚本
• 权限设置
• 数据库监控备份及恢复
Storage:
• 典型的关系型数据库部署在 SAN 上
• MongoDB支持 SAN, 但是使用本地存储
• 可以最大化的提高性能
JDBC:
MongoDB 没有原生态 JDBC, 而是采用自带的驱动程序:
• 自带连接池管理
• 事务支持
SQL/RS:
MongoDB 不支持SQL的增删改，结果集也不是 ResultSet

等..
```

13.3 DBA主要关注--数据迁移

迁移方式

```
• 数据库导出+导入
• 批量迁移工具
• 实时同步工具
• 应用主导迁移
```

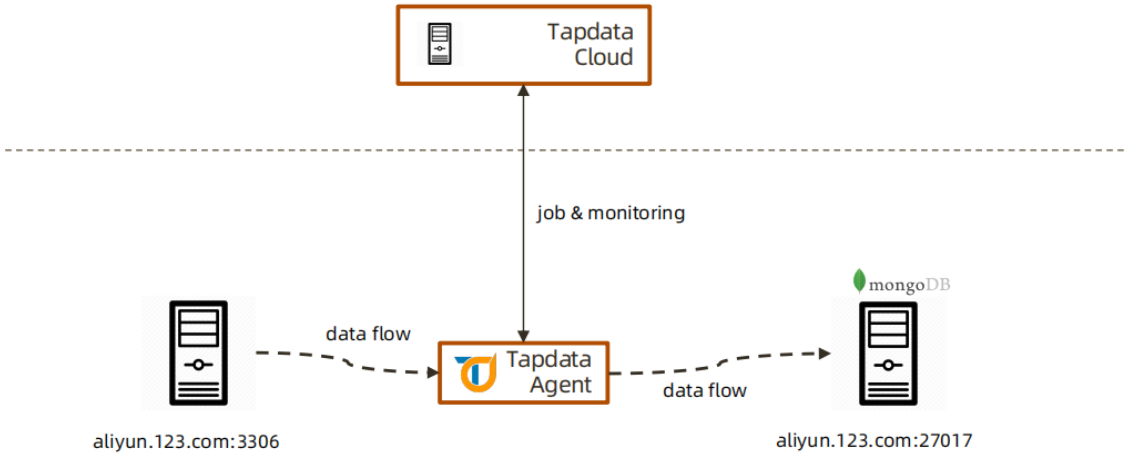


各种迁移方式的特点

```
1. 数据库导出导入
步骤:
- 停止现有的基于 RDBMS 的应用
- 使用 RDBMS 的数据库导出工具，将数据库表导出到 CSV 或者 JSON（如 mysqldump） - 使用 mongoimport 将 CSV 或者 JSON 文件导入 MongoDB 数据库
- 启动新的 MongoDB 应用
备注:
- 适用于一次性数据迁移
- 需要应用/数据库下线，较长的下线时间

2. 批量同步
步骤:
- 安装同步工具（如 Kettle / Talend）
- 创建输入源（关系型数据库）
- 创建输出源（MongoDB） - 编辑数据同步任务
- 执行
备注:
- 适用批量同步，定期更新，特别是每晚跑批的场景
- 支持基于时间戳的增量同步，需要源表有合适的时间戳支持
- 对源库有较明显的性能影响，不宜频繁查询
- 不支持实时同步

3. 实时同步
步骤:
- 安装实时同步工具（如Informatica / Tapdata）
- 创建输入源（关系型数据库）
- 创建输出源（MongoDB） - 编辑实时数据同步任务
- 执行
备注:
- 基于源库的日志文件解析机制，可以实现秒级数据的同步
- 对源库性能影响较小
- 可以支持应用的无缝迁移

4. 应用主导迁移
步骤:
1. 升级已有应用支持 MongoDB
2. 数据插入请求直接进入 MongoDB
```

```
3. 数据查询和更新请求首先定向到 MongoDB
4. 如果记录不存在，从 RDBMS 读出来并写入到 MongoDB
5. 重复步骤3
6. 当步骤4在限定时间段（一星期、一个月）没有被调用，认为迁移完成
备注：
- 需要研发团队配合 ，有一定开发和测试量
- 为保证不遗漏数据，仍然先要执行一次批量同步
```

13.4 实时同步方案-TPDATA应用



第三章 MongoDB 集群中的数据一致性和隔离性保证

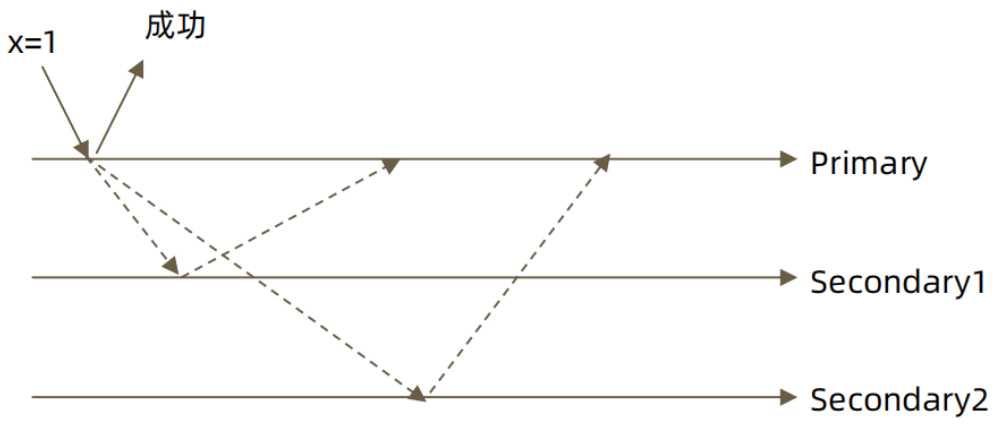
1.writeConcern

1.1 什么是 writeConcern

```
writeConcern 决定一个写操作落到多少个节点上才算成功。类似于MySQL中的半同步复制。主要保证数据最终一致性
• 0：默认值。发起写操作，不关心是否成功；
• 1: 集群最大数据节点数：写操作需要被复制到指定节点数才算成功；
• majority：写操作需要被复制到大多数节点上才算成功。
```

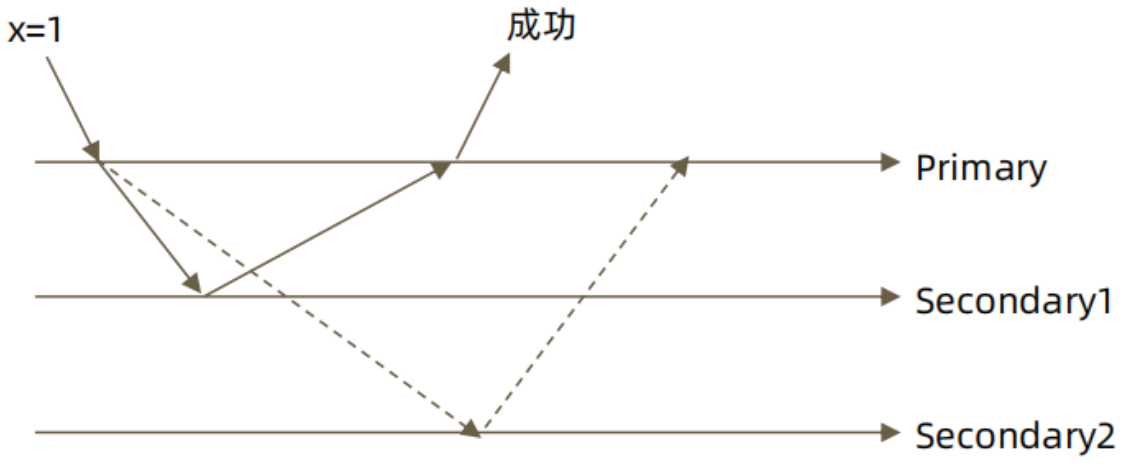
默认行为

```
3 节点复制集不作任何特别设定（默认值）。
```



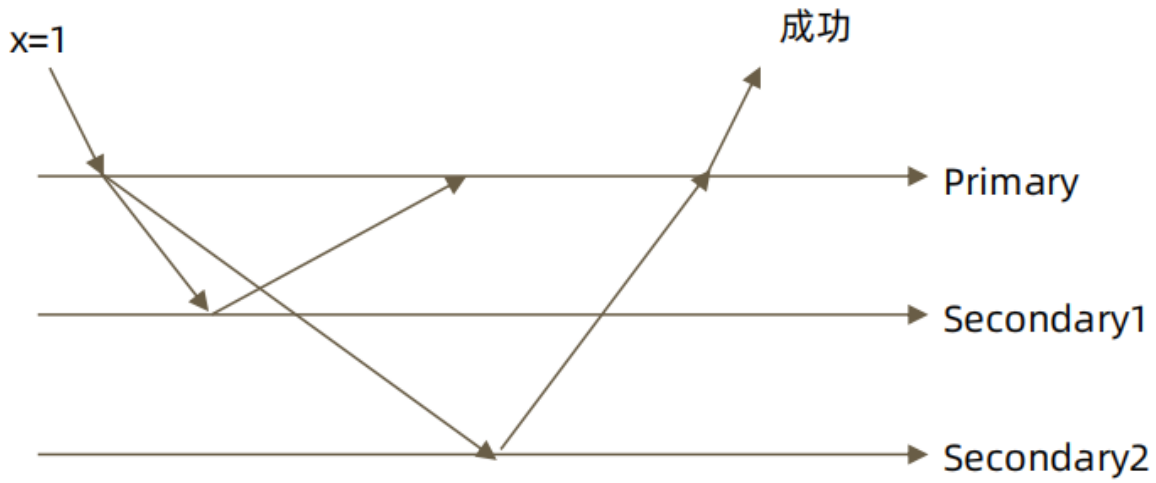
w: "majority"

```
大多数节点确认模式,其中包括主节点
```



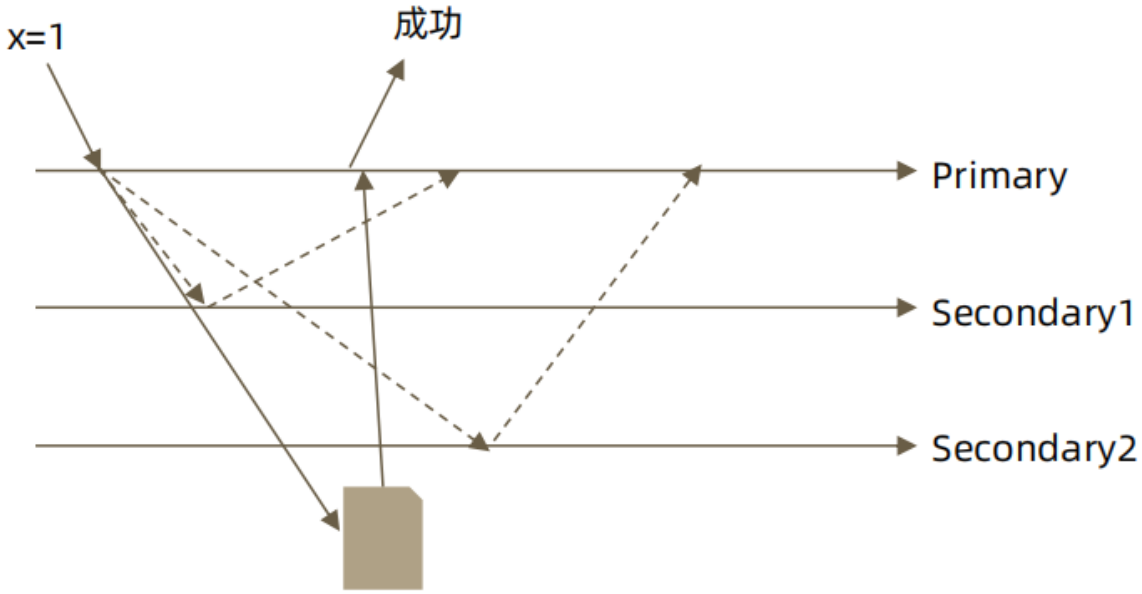
w: "all"

```
全部节点确认模式。
```



1.2 journal日志

```
writeConcern 可以决定写操作到达多少个节点才算成功，journal 则定义如何才算成功。journal日志类似于MySQL中的redo日志。是在WriteConcern基础上对于数据安全的进一步保证。
取值包括：
• true: 写操作落到journal 文件中才算成功；
• false: 写操作到达内存即算作成功。
```



1.3 writeConcern 测试

```
在复制集测试writeConcern参数
db.test.insert( {count: 1}, {writeConcern: {w: "majority"}})
db.test.insert( {count: 1}, {writeConcern: {w: 3 }})
db.test.insert( {count: 1}, {writeConcern: {w: 4 }})
配置延迟节点，模拟网络延迟（复制延迟）
conf=rs.conf()
conf.members[2].slaveDelay = 5
conf.members[2].priority = 0
rs.reconfig(conf)
观察复制延迟下的写入，以及timeout参数
db.test.insert( {count: 1}, {writeConcern: {w: 3})
db.test.insert( {count: 1}, {writeConcern: {w: 3, wtimeout:3000 }})
```

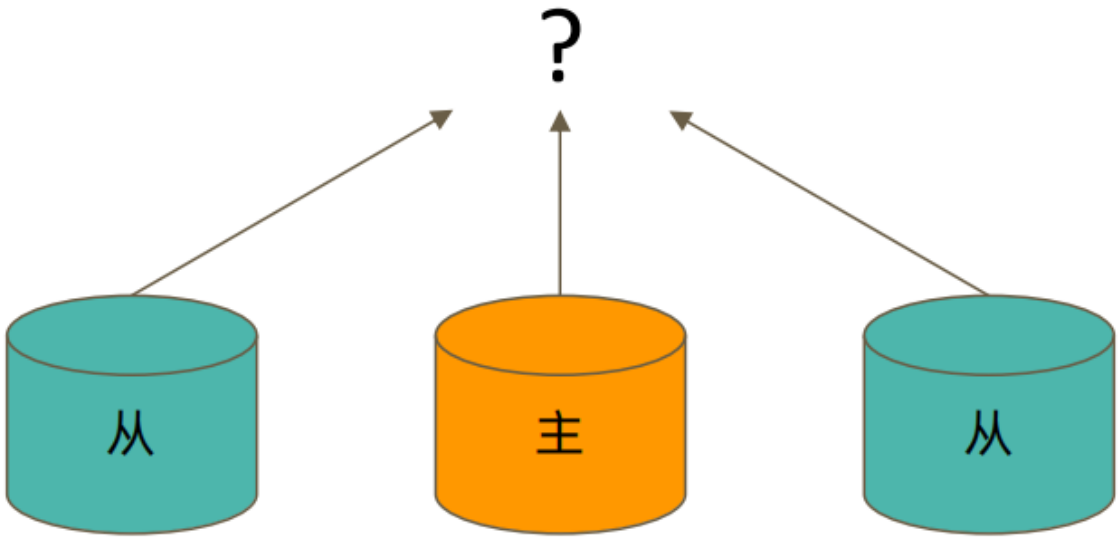
1.4 注意事项

```
• 虽然多于半数的 writeConcern 都是安全的，但通常只会设置 majority，因为这是等待写入延迟时间最短的选择；
• 不要设置 writeConcern 等于总节点数，因为一旦有一个节点故障，所有写操作都将失败；
• writeConcern 虽然会增加写操作延迟时间，但并不会显著增加集群压力，因此无论是否等待，写操作最终都会复制到所有节点上。设置 writeConcern 只是让写操作等待复制后再返回而已；
• 应对重要数据应用 {w: “majority”}，普通数据可以应用 {w: 1} 以确保最佳性能。
```

2.readPreference读配置

2.1 介绍

```
readPreference 决定使用哪一个节点来满足正在发起的读请求。
可选值包括：
• primary: 只选择主节点
• primaryPreferred：优先选择主节点，如果不可用则选择从节点
• secondary：只选择从节点
• secondaryPreferred：优先选择从节点，如果从节点不可用则选择主节点
• nearest：选择最近的节点
```

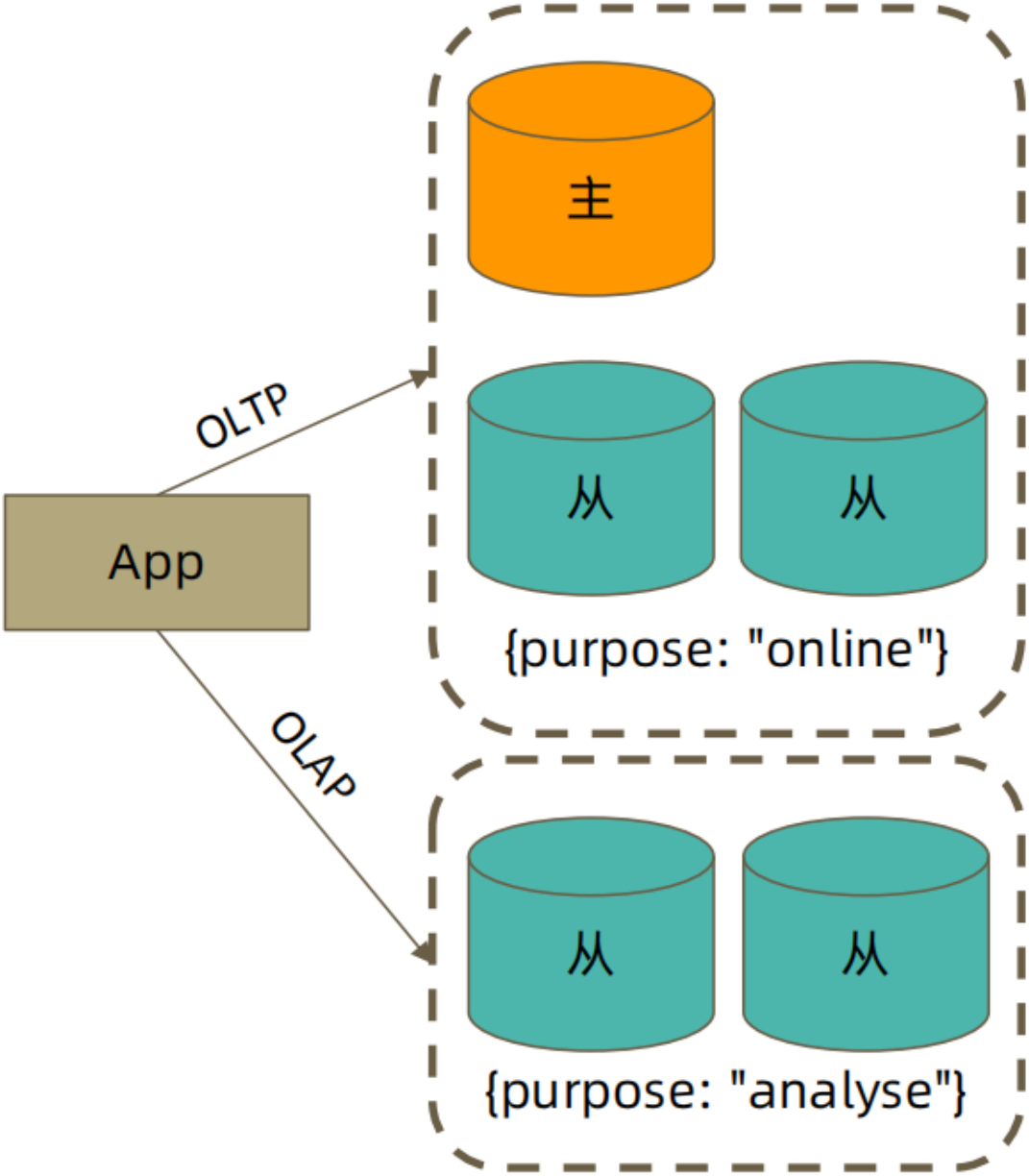


2.2. readPreference 场景举例

```
• 用户下订单后马上将用户转到订单详情页——primary/primaryPreferred。因为此时从节点可能还没复制到新订单；
• 用户查询自己下过的订单——secondary/secondaryPreferred。查询历史订单对时效性通常没有太高要求；
• 生成报表——secondary。报表对时效性要求不高，但资源需求大，可以在从节点单独处理，避免对线上用户造成影响；
• 将用户上传的图片分发到全世界，让各地用户能够就近读取——nearest。每个地区的应用选择最近的节点读取数据。
```

2.3 readPreference 与 Tag

```
readPreference 只能控制使用一类节点。Tag 则可以将节点选择控制到一个或几个节点。
考虑以下场景：
• 一个 5 个节点的复制集
• 3 个节点硬件较好，专用于服务线上客户
• 2 个节点硬件较差，专用于生成报表
可以使用 Tag 来达到这样的控制目的
• 为 3 个较好的节点打上 {purpose: "online"}
• 为 2 个较差的节点打上 {purpose: "analyse"}
• 在线应用读取时指定 online，报表读取时指定reporting
更多信息请参考官方文档：readPreference
```



2.4 readPreference 配置

```
通过 MongoDB 的连接串参数:
• mongodb://host1:27107,host2:27107,host3:27017/?replicaSet=rs&readPreference=secondary
通过 MongoDB 驱动程序 API:
• MongoCollection.withReadPreference(ReadPreference readPref)
Mongo Shell:
• db.collection.find({}).readPref( “secondary” )
```

2.5 readPreference 实验: 从节点读

```
• 主节点写入 {x:1}，观察该条数据在各个节点均可见
• 在两个从节点分别执行 db.fsyncLock() 来锁定写入（同步）
• 主节点写入 {x:2}
• db.test.find()
• db.test.find().readPref("secondary")
• 解除从节点锁定 db.fsyncUnlock()
• db.test.find().readPref("secondary")
```

2.6 注意事项

```
• 指定 readPreference 时也应注意高可用问题。例如将 readPreference 指定 primary，则发生故障转移不存在 primary 期间将没有节点可读。如果业务允许，则应选择 primaryPreferred；
• 使用 Tag 时也会遇到同样的问题，如果只有一个节点拥有一个特定 Tag，则在这个节点失效时将无节点可读。这在有时候是期望的结果，有时候不是。例如：
• 如果报表使用的节点失效，即使不生成报表，通常也不希望将报表负载转移到其他节点上，此时只有一个节点有报表 Tag 是合理的选择；
• 如果线上节点失效，通常希望有替代节点，所以应该保持多个节点有同样的 Tag；
• Tag 有时需要与优先级、选举权综合考虑。例如做报表的节点通常不会希望它成为主节点，则优先级应为 0。
```

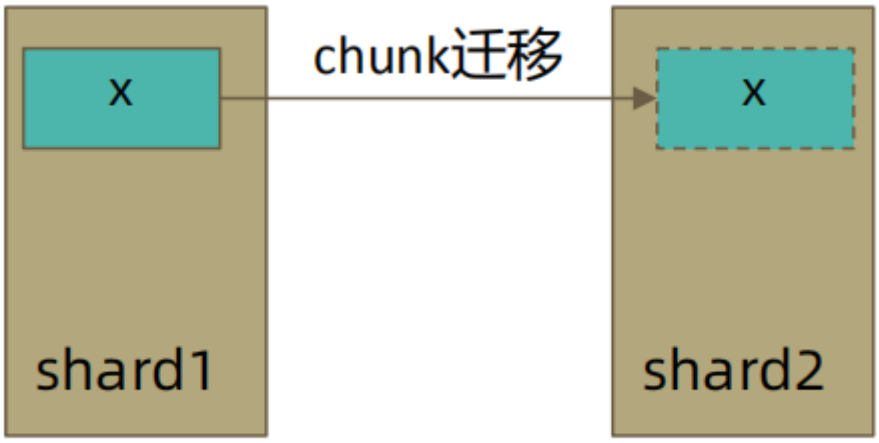
3. readConcern读隔离性保证

3.1 什么是 readConcern?

```
在 readPreference 选择了指定的节点后，readConcern 决定这个节点上的数据哪些是可读的，类似于关系数据库的隔离级别。可选值包括：
• available：读取所有可用的数据；
• local：读取所有可用且属于当前分片的数据；
• majority：读取在大多数节点上提交完成的数据；
• snapshot：读取最近快照中的数据；
• linearizable：可线性化读取文档；

```

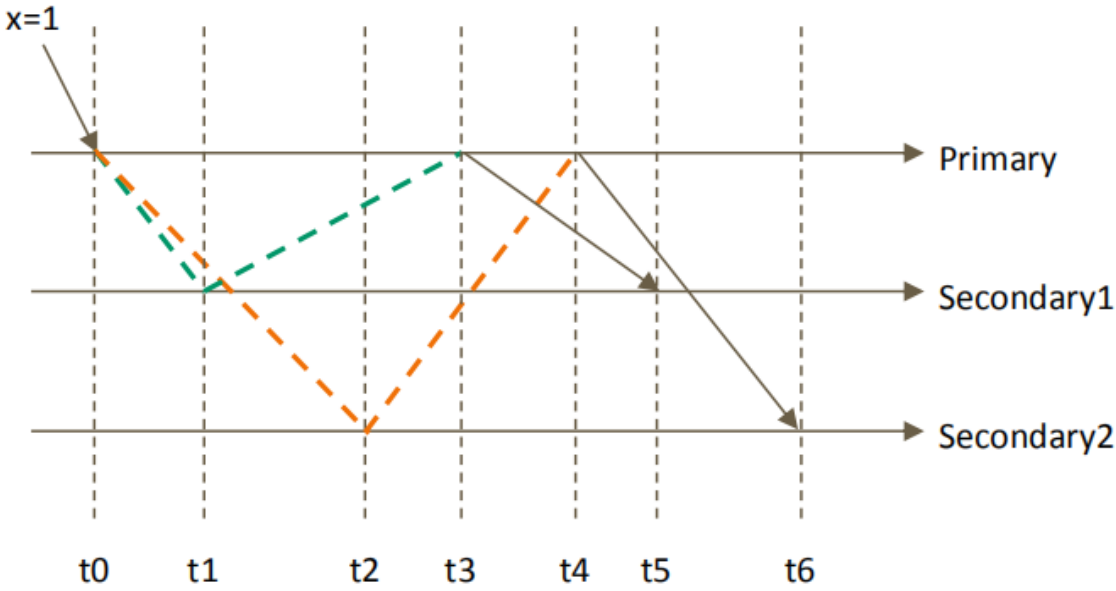
3.2 readConcern: local 和 available

```
在复制集中 local 和 available 是没有区别的。两者的区别主要体现在分片集上。
考虑以下场景：
• 一个 chunk x 正在从 shard1 向 shard2 迁移；
• 整个迁移过程中 chunk x 中的部分数据会在 shard1 和 shard2 中同时存在，但源分片shard1仍然是chunk x 的负责方；
• 所有对 chunk x 的读写操作仍然进入 shard1；
• config 中记录的信息 chunk x 仍然属于 shard1；
• 此时如果读 shard2，则会体现出 local 和 available 的区别：
    local：只取应该由 shard2 负责的数据（不包括 x）；
    available：shard2 上有什么就读什么（包括 x）；
注意事项：
• 虽然看上去总是应该选择 local，但毕竟对结果集进行过滤会造成额外消耗。在一些无关紧要的场景（例如统计）下，也可以考虑 available；
• MongoDB <=3.6 不支持对从节点使用 {readConcern: "local"}；
• 从主节点读取数据时默认 readConcern 是 local，从从节点读取数据时默认readConcern是available（向前兼容原因）。
```



3.3 readConcern: majority

```
只读取大多数数据节点上都提交了的数据。考虑如下场景：
● 集合中原有文档 {x: 0}；
● 将x值更新为 1；
```

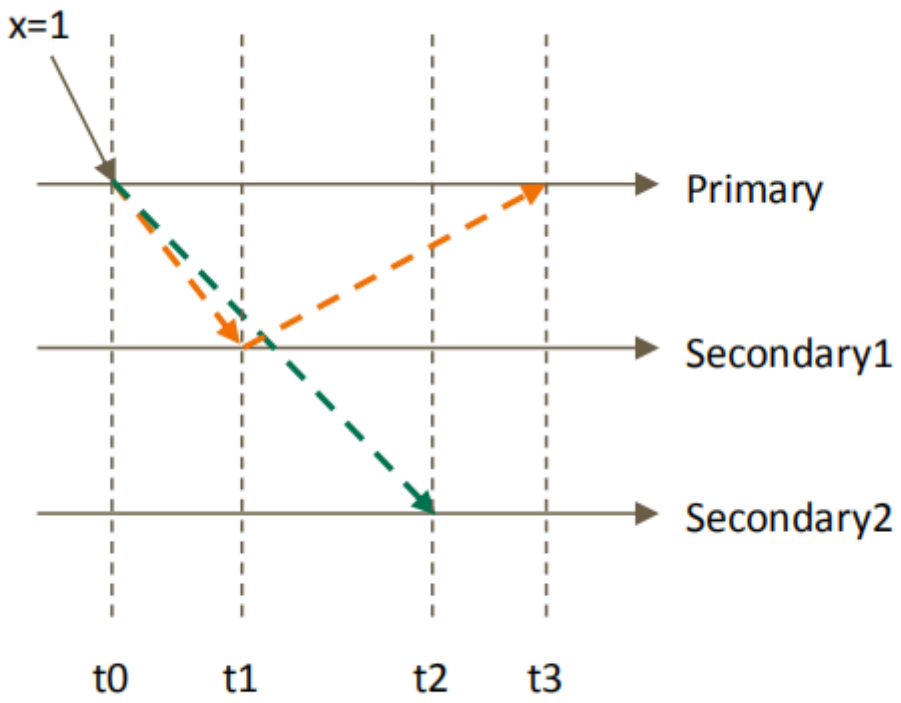


如果在各节点上应用
{readConcern: “majority” } 来读取数据：

	P	S1	S2
t0	x=0	x=0	x=0
t1	x=0	x=0	x=0
t2	x=0	x=0	x=0
t3	x=1	x=0	x=0
t4	x=1	x=0	x=0
t5	x=1	x=1	x=0
t6	x=1	x=1	x=1

3.4 readConcern: majority 的实现方式

```
考虑 t3 时刻的 Secondary1，此时：
• 对于要求 majority 的读操作，它将返回 x=0；
• 对于不要求 majority 的读操作，它将返回 x=1；
如何实现？
节点上维护多个 x 版本，MVCC 机制MongoDB 通过维护多个快照来链接不同的版本：
• 每个被大多数节点确认过的版本都将是一个快照；
• 快照持续到没有人使用为止才被删除；
```



3.5 实验： readConcern : ”majority” vs “local”

```
安装 3 节点复制集。
• 注意配置文件内 server 参数 enableMajorityReadConcern
• 将复制集中的两个从节点使用 db.fsyncLock() 锁住写入（模拟同步延迟）
```

```
replication:
  replSetName: rs0
  enableMajorityReadConcern: true
```

3.6 readConcern 验证

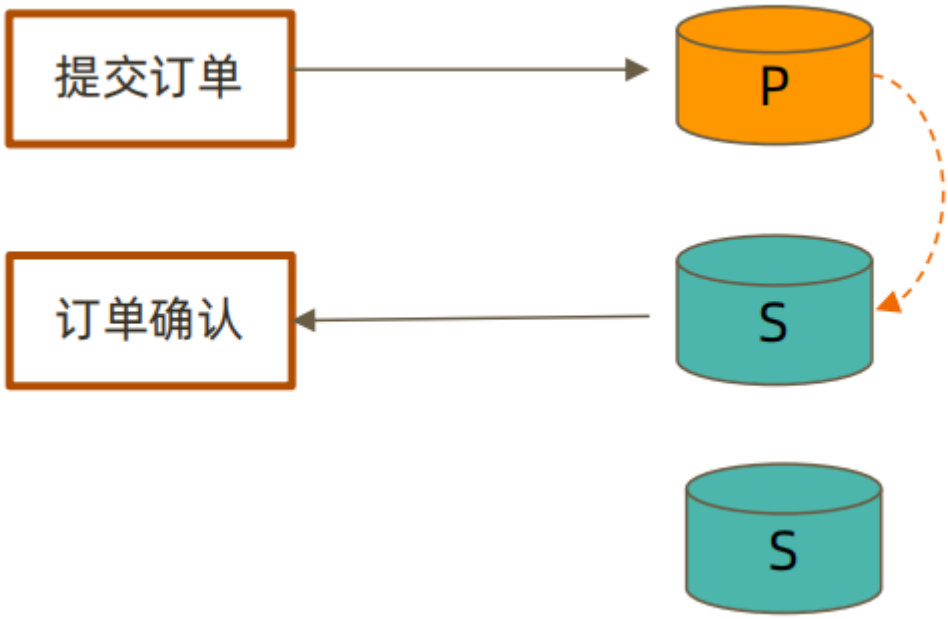
```
• db.test.save({“A”:1})
• db.test.find().readConcern(“local”)
• db.test.find().readConcern(“majority”)
• 在从节点上执行 db.fsyncUnlock()
结论：
• 使用 local 参数，则可以直接查询到写入数据
• 使用 majority，只能查询到已经被多数节点确认过的数据
• update 与 remove 与上同理。
```

3.7 readConcern: majority 与脏读

```
MongoDB 中的回滚：
• 写操作到达大多数节点之前都是不安全的，一旦主节点崩溃，而从节还没复制到该次操作，刚才的写操作就丢失了；
• 把一次写操作视为一个事务，从事务的角度，可以认为事务被回滚了。所以从分布式系统的角度来看，事务的提交被提升到了分布式集群的多个节点级别的“提交”，而不再是单个节点上的“提交”。在可能发生回滚的前提下考虑脏读问题
• 如果在一次写操作到达大多数节点前读取了这个写操作，然后因为系统故障该操作回滚了，则发生了脏读问题；使用 {readConcern: “majority”} 可以有效避免脏读
```

3.8 readConcern: 如何实现安全的读写分离

```
考虑如下场景：
向主节点写入一条数据；立即从从节点读取这条数据。如何保证自己能够读到刚刚写入的数据？
下述方式有可能读不到刚写入的订单：
db.orders.insert({ oid: 101, sku: ”kite", q: 1})
db.orders.find({oid:101}).readPref("secondary")

使用 writeConcern + readConcern majority 来解决：
db.orders.insert({ oid: 101, sku: "kiteboar", q: 1}, {writeConcern:{w: "majority”}})
db.orders.find({oid:101}).readPref(“secondary”).readConcern("majority")

```



读隔离性和MySQL的对比

```
readConcern 主要关注读的隔离性， ACID 中的 Isolation，但是是分布式数据库里特有的概念
readCocnern： majority 对应于MySQL事务中隔离级别中的哪一级？
```

Read Uncommited

Read Committed

Repeatable Read

Seriazable

4.MongoDB的ACID事务支持

4.1 介绍

```
MongoDB 虽然已经在 4.2 开始全面支持了多文档事务，但并不代表大家应该毫无节制地使用它。相反，对事务的使用原则应该是：能不用尽量不用。
通过合理地设计文档模型，可以规避绝大部分使用事务的必要性
为什么？
事务 = 锁，节点协调，额外开销，性能影响.
```

4.2 MongoDB ACID 多文档事务支持

事务属性	支持程度
Atomocity 原子性	单表单文档： 1.x 就支持 复制集多表多行：4.0 复制集 分片集群多表多行4.2
Consistency 一致性	writeConcern, readConcern (3.2)
Isolation 隔离性	readConcern (3.2)
Durability 持久性	Journal and Replication

4.3 使用方法

```
MongoDB 多文档事务的使用方式与关系数据库非常相似:
try (ClientSession clientSession = client.startSession()) {
clientSession.startTransaction();
collection.insertOne(clientSession, docOne);
collection.insertOne(clientSession, docTwo);
clientSession.commitTransaction();
}
```

4.4 事务的隔离级别

```
- 事务完成前，事务外的操作对该事务所做的修改不可访问
- 如果事务内使用 {readConcern: “snapshot”}，则可以达到可重复读 Repeatable Read
```

4.5 实验：启用事务后的隔离性

```
db.tx.insertMany([{ x: 1 }, { x: 2 }]);
var session = db.getMongo().startSession();
session.startTransaction();
var coll = session.getDatabase('test').getCollection("tx");
coll.updateOne({x: 1}, {$set: {y: 1}});
coll.findOne({x: 1});
db.tx.findOne({x: 1});
session.abortTransaction();
```

```
db.tx.insertMany([{ x: 1 }, { x: 2 }]);
var session = db.getMongo().startSession();
session.startTransaction();
var coll = session.getDatabase('test').getCollection("tx");
coll.updateOne({x: 1}, {$set: {y: 1}});
coll.findOne({x: 1});                                  事务内操作 {x:1, y:1}
db.tx.findOne({x: 1});                                 // 事务外的操作 {x:1}
session.abortTransaction();
```

4.6 实验：可重复读 Repeatable Read

```
var session = db.getMongo().startSession();
session.startTransaction({
readConcern: {level: "snapshot"},
writeConcern: {w: "majority"}});
var coll = session.getDatabase('test').getCollection("tx");
coll.findOne({x: 1}); // 返回：{x: 1}
db.tx.updateOne({x: 1}, {$set: {y: 1}});
db.tx.findOne({x: 1}); // 返回：{x: 1, y: 1}
coll.findOne({x: 1}); // 返回：{x: 1}
session.abortTransaction();
```

```
var session = db.getMongo().startSession();
session.startTransaction({
        readConcern: {level: "snapshot"},
        writeConcern: {w: "majority"}});
var coll = session.getDatabase('test').getCollection("tx");

coll.findOne({x: 1});  // 返回：{x: 1}
db.tx.updateOne({x: 1}, {$set: {y: 1}});
db.tx.findOne({x: 1});  // 返回：{x: 1, y: 1}
coll.findOne({x: 1});  // 返回：{x: 1}
session.abortTransaction();
```

Repeatable Read

4.7 事务写机制

```
MongoDB 的事务错误处理机制不同于关系数据库：
● 当一个事务开始后，如果事务要修改的文档在事务外部被修改过，则事务修改这个文档时会触发 Abort 错误，因为此时的修改冲突了；
● 这种情况下，只需要简单地重做事务就可以了；
● 如果一个事务已经开始修改一个文档，在事务以外尝试修改同一个文档，则事务以外的修改会等待事务完成才能继续进行
```

4.8 实验：写冲突

```
继续使用上个实验的tx集合
开两个 mongo shell 均执行下述语句
var session = db.getMongo().startSession();
session.startTransaction({ readConcern: {level: "snapshot"},
writeConcern: {w: "majority"}});
var coll = session.getDatabase('test').getCollection("tx");
```

窗口1：

```
coll.updateOne({x: 1}, {$set: {y: 1}});
// 正常结束
```

窗口2：

```
coll.updateOne({x: 1}, {$set: {y: 2}});
// 异常 - 解决方案：重启事务
```

```
窗口1：第一个事务，正常提交
coll.updateOne({x: 1}, {$set: {y: 1}});
窗口2：另一个事务更新同一条数据，异常
coll.updateOne({x: 1}, {$set: {y: 2}});
窗口3：事务外更新，需等待
db.tx.updateOne({x: 1}, {$set: {y: 3}});
```

4.9 注意事项

- 可以实现和关系型数据库类似的事务场景
- 必须使用与 MongoDB 4.2 兼容的驱动；
- 事务默认必须在 60 秒（可调）内完成，否则将被取消；
- 涉及事务的分片不能使用仲裁节点；
- 事务会影响 chunk 迁移效率。正在迁移的 chunk也可能造成事务提交失败（重试即可）；
- 多文档事务中的读操作必须使用主节点读；
- readConcern 只应该在事务级别设置，不能设置在每次读写操作上。
- 必须是WT引擎才支持事务。