文档数据库 MongoDB

内容概述

- MongoDB 介绍
- MongoDB 安装
- MongoDB 管理
- MongoDB 复制集
- MongoDB 分片集群
- MongoDB 备份和恢复

1 MongoDB 介绍

1.1 MongoDB 介绍

MongoDB并非芒果mango的意思，而是源于 Humongous（巨大）一词

MongoDB是由总部位于美国纽约的上市公司MongoDB Inc (原名10gen)基于C++编写的分布式文档数据库。

之所以称为文档数据库，是因为MongoDB保存的是"JSON Document"，并非一般的PDF，WORD文档

MongoDB内部使用类似于Json的bson格式。

内部执行引擎为JS解释器。把文档存储成bson结构,在查询时转换为JS对象,并可以通过熟悉的js语法来操作

MongoDB被称为最像RDBMS 的NoSQL,支持事务，锁，索引类似于MySQL

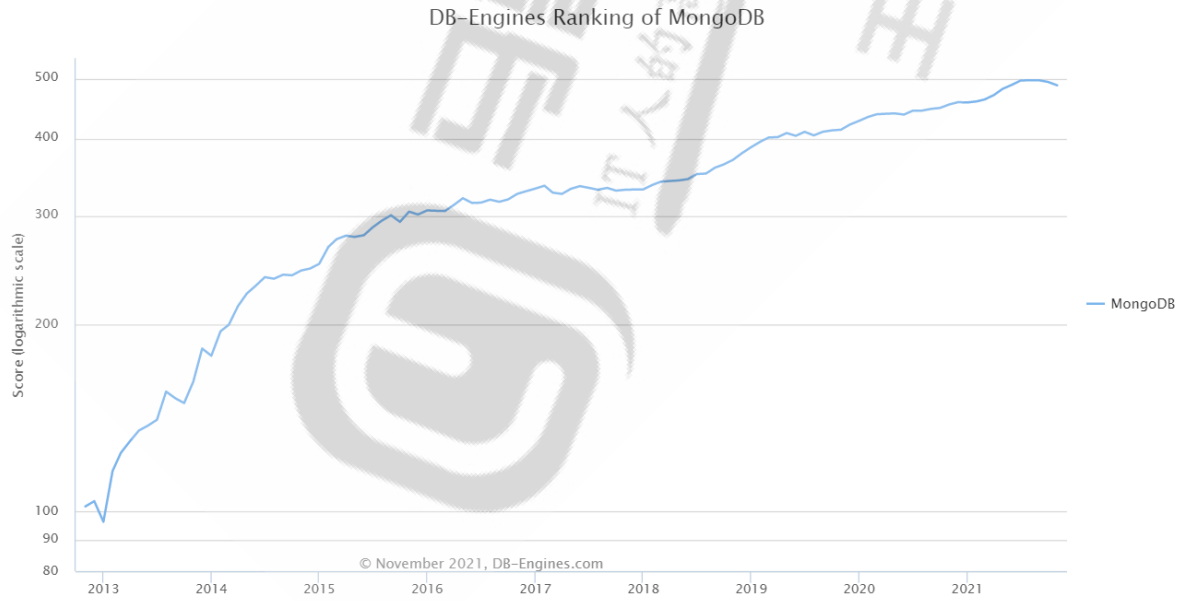
官网: http://www.mongodb.com/

GitHub URL：https://github.com/mongodb

```
https://db-engines.com/en/ranking
https://db-engines.com/en/ranking_trend/system/MongoDB
```

381 systems in ranking, November 2021

Rank Nov 2021	Rank Oct 2021	Rank Nov 2020	DBMS	Database Model	Score Nov 2021	Score Oct 2021	Score Nov 2020
1.	1.	1.	Oracle	Relational, Multi-model	1272.73	+2.38	-72.27
2.	2.	2.	MySQL	Relational, Multi-model	1211.52	-8.25	-30.12
3.	3.	3.	Microsoft SQL Server	Relational, Multi-model	954.29	-16.32	-83.35
4.	4.	4.	PostgreSQL	Relational, Multi-model	597.27	+10.30	+42.22
5.	5.	5.	MongoDB	Document, Multi-model	487.35	-6.21	+33.52
6.	6.	↑ 7.	Redis	Key-value, Multi-model	171.50	+0.15	+16.08
7.	7.	↓ 6.	IBM Db2	Relational, Multi-model	167.52	+1.56	+5.90
8.	8.	8.	Elasticsearch	Search engine, Multi-model	159.09	+0.84	+7.54
9.	9.	9.	SQLite	Relational	129.80	+0.43	+6.48
10.	10.	10.	Cassandra	Wide column	120.88	+1.61	+2.13
11.	11.	11.	Microsoft Access	Relational	119.24	+2.86	+2.01
12.	12.	12.	MariaDB	Relational, Multi-model	102.19	-0.41	+9.90
13.	13.	13.	Splunk	Search engine	92.31	+1.69	+2.60
14.	14.	↑ 15.	Hive	Relational	83.31	-1.43	+13.05
15.	15.	↑ 17.	Microsoft Azure SQL Database	Relational, Multi-model	81.32	+1.60	+14.33
16.	16.	16.	Amazon DynamoDB	Multi-model	76.99	+0.43	+8.09
17.	17.	↓ 14.	Teradata	Relational, Multi-model	69.59	-0.24	-6.01
18.	18.	↑ 42.	Snowflake	Relational	64.19	+5.93	+54.09
19.	19.	↑ 20.	Neo4j	Graph	57.98	+0.11	+4.45
20.	20.	↓ 19.	SAP HANA	Relational, Multi-model	55.53	+0.26	+1.95



主要用途：

- 应用数据库，比如：存储银行，保险公司流水信息，类似于Oracle,MySQL海量数据处理，大数据分析
- 网站数据、缓存等大尺寸、低价值的数据
- 在高伸缩性的场景，用于对象及JSON数据的存储。

主要特点

- 灵活动态的文档建模
- JSON数据模型比较适合开发者
- 支持高可用

- 水平扩展可以支撑很大数据量和并发
- 存储空间占用相对关系型数据库会小很多

缺点：

- 多表关联： 仅仅支持Left Outer Join
- SQL 语句支持： 查询为主，部分支持
- 多表原子事务： 不支持
- 多文档原子事务：不支持
- 16MB 文档大小限制，不支持中文排序
- 服务端 Javascript 性能欠佳

MongoDB 历史

- 2007年10月，MongoDB由10gen团队开发
- 2009年2月首度推出第一个版本
- 2012年05月23日，MongoDB2.1 开发分支发布了! 该版本采用全新架构，包含诸多增强
- 2012年06月06日，MongoDB 2.0.6 发布，分布式文档数据库
- 2013年04月23日，MongoDB 2.4.3 发布，此版本包括了一些性能优化，功能增强以及bug修复
- 2013年08月20日，MongoDB 2.4.6 发布
- 2013年11月01日，MongoDB 2.4.8 发布

语言支持

MongoDB有官方的驱动如下：

- C
- C++
- C# / .NET
- Erlang
- Haskell
- Java
- JavaScript
- Lisp
- node.JS
- Perl
- PHP
- Python
- Ruby
- Scala
- Go

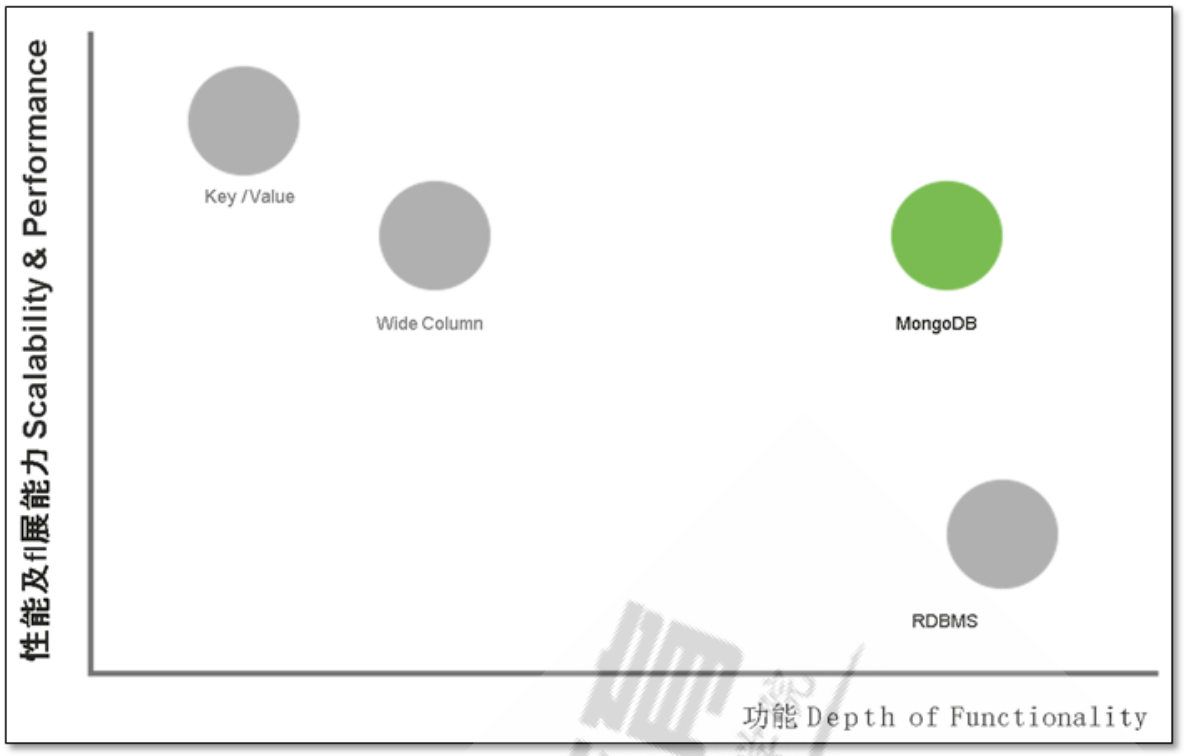
1.2 对比关系型数据库

基于功能选择MongoDB

	MongoDB	关系型数据库
数据模型	文档模型	关系模型
数据库类型	OLTP	OLTP
CRUD操作	MQL/SQL	SQL
高可用	复制集	集群模式
数据容量	没有理论上限	千万、亿
扩展方式	垂直扩展+水平扩展	垂直扩展
索引支持	B-树、全文索引、地理位置索引、多键(multikey)B树索引引、TTL索引	B-树
横向扩展能力	通过原生分片完善支持	数据分区或者应用侵入式
开发难度	容易	困难
亿级以上数据量	轻松支持	分库分表
灵活表结构	轻松支持	Entity Key /value表，关联查询比较复杂
高并发读写	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	轻松支持功能很强大	使用Group By 等，能力有限
大宽表	轻松支持	性能受限
异构数据	轻松支持	使用EKV属性表

1.3 数据库功能和性能对比

由下图可以看出MongoDB数据库的性能扩展能力及功能都较好，都能够在数据库中，站立一足之地。



1.4 什么时候使用MongoDB

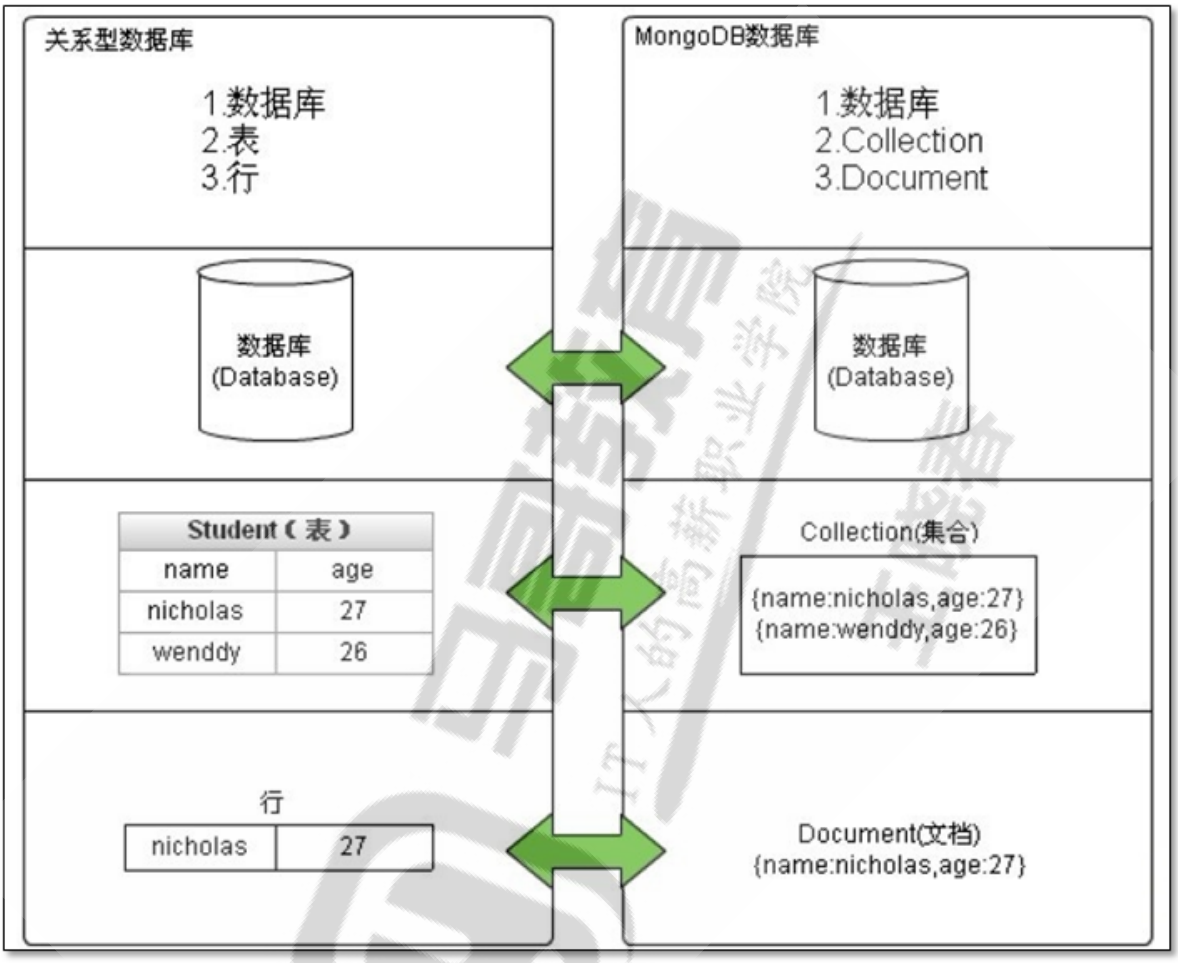
- 数据量是有亿万级或者需要不断扩容
- 需要2000-3000以上的读写每秒
- 新应用，需求会变，数据模型无法确定
- 需要整合多个外部数据源
- 系统需要99.999%高可用
- 系统需要大量的地理位置查询
- 系统需要提供最小的latency
- 管理的主要数据对象<10

满足上面一条，可以考虑MongoDB；当有2个以上的时候：MongoDB是不二的选择!

1.5 逻辑结构

id	user_name	email	age	city
1	Mark Hanks	mark@abc.com	25	Los Angeles
2	Richard Peter	richard@abc.com	31	Dallas

```
{
    "_id": ObjectId("5146bb52d8524270060001f3"),
    "age": 25,
    "city": "Los Angeles",
    "email": "mark@abc.com",
    "user_name": "Mark Hanks"
}
{
    "_id": ObjectId("5146bb52d8524270060001f2"),
    "age": 31,
    "city": "Dallas",
    "email": "richard@abc.com",
    "user_name": "Richard Peter"
}
```



MySQL逻辑结构	Mongodb逻辑结构
库: database	库:database
表:table	集合:collection
记录：record	文档:document
列: column	键: Key
Primary Key	主键 (MongoDB提供了key为 _id)
字段值：value	值：value

MongoDB和关系型数据库最大的不同:

- 传统型数据库:结构化数据，定好了表结构后,每一行的内容必须符合表结构，就是说一列的个数和类型都一样

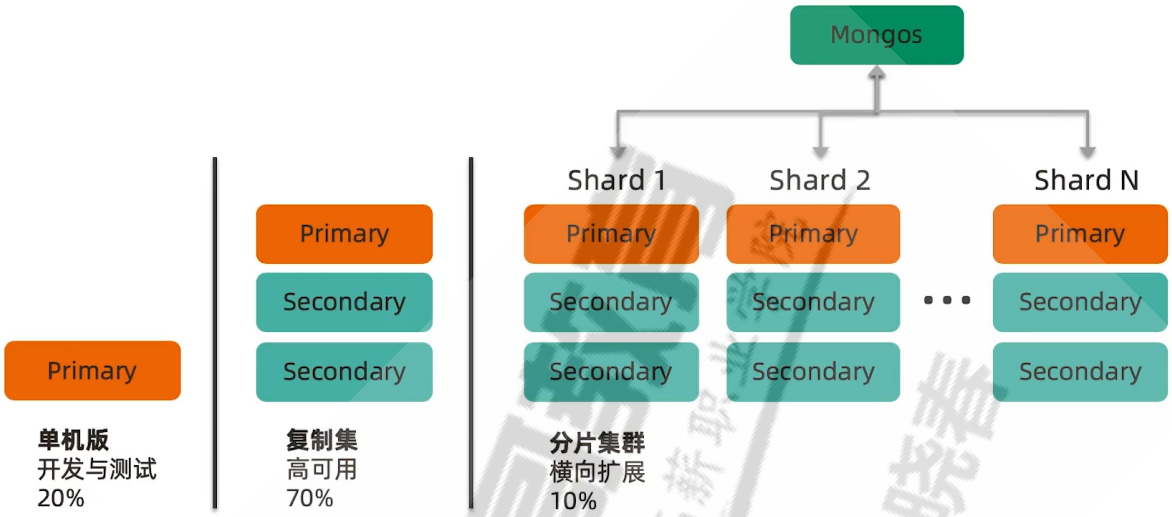
- MongoDB文档型数据库:表中的每个文档都可以有自己独特的结构，即json对象都可以有自己独特的属性和值

1.6 版本

MongoDB有两个发布版本:

- 社区版：社区版是基于SSPL,一种和AGPL基本类似的开源协议
- 企业版：企业版是基于商业协议，需付费使用

1.7 MongoDB 常见部署架构



1.8 MongoDB 相关工具和应用

软件	说明
mongod	MongoDB数据库软件
mongo	MongoDB命令行工具，管理MongoDB数据库
mongos	MongoDB路由进程，分片环境下使用
mongodump / mongorestore	命令行数据库备份与恢复工具
mongoexport / mongoimport	CSV/JSON 导入与导出，主要用于不同系统间数据迁移
compass	MongoDB GUI管理工具
Ops Manager(企业版)	MongoDB集群管理软件,支持对分片集群的备份还原
BI Connector(企业版)	SQL解释器/BI套接件
MongoDB Charts(企业版)	MongoDB可视化软件
Atlas(付费及免费)	MongoDB云托管服务，包括永久免费云数据库

1.9 MongoDB 参考资料

```
http://docs.mongodb.com
http://university.mongodb.com
http://mongoing.com/
https://www.cnblogs.com/clsn/p/8214194.html
https://www.runoob.com/mongodb/mongodb-intro.html
```

2 MongoDB 部署和访问

官方安装说明

```
https://docs.mongodb.com/manual/installation/
https://docs.mongodb.com/manual/administration/install-community/
https://docs.mongodb.com/manual/administration/install-on-linux/
```

MongoDB 支持在各种系统上安装，包括：

- Linux
- MacOS
- Windows

2.1 安装方法

MongoDB 提供多种安装方法

- 基于rpm/deb包安装
- 基于二进制包安装
- 基于docker安装

2.2 基于rpm/deb包安装

官方仓库路径

```
https://repo.mongodb.org/
```

国内镜像

```
https://mirrors.tuna.tsinghua.edu.cn/mongodb/
https://mirrors.aliyun.com/mongodb/
```

2.2.1 基于CentOS安装

官方说明

```
https://docs.mongodb.com/manual/tutorial/install-mongodb-on-red-hat/
```

范例：

```
cat > /etc/yum.repos.d/mongodb-org-5.0.repo <<EOF
[mongodb-org-5.0]
name=MongoDB Repository
baseurl=https://repo.mongodb.org/yum/redhat/\$releasever/mongodb-org/5.0/x86_64/
gpgcheck=1
enabled=1
```

```
gpgkey=https://www.mongodb.org/static/pgp/server-5.0.asc
EOF

#安装最新版
yum install -y mongodb-org

#安装指定版本
yum install -y mongodb-org-5.0.2 mongodb-org-database-5.0.2 mongodb-org-server-
5.0.2 mongodb-org-shell-5.0.2 mongodb-org-mongos-5.0.2 mongodb-org-tools-5.0.2
```

2.2.2 基于Ubuntu安装

```
https://docs.mongodb.com/manual/tutorial/install-mongodb-on-ubuntu/
```

范例：

```
#导入公钥
wget -qO - https://www.mongodb.org/static/pgp/server-5.0.asc | sudo apt-key add
-

#配置仓库文件
#基于Ubuntu20.04
echo "deb [ arch=amd64,arm64 ] https://repo.mongodb.org/apt/ubuntu
focal/mongodb-org/5.0 multiverse" | sudo tee /etc/apt/sources.list.d/mongodb-
org-5.0.list

#基于Ubuntu18.04
echo "deb [ arch=amd64,arm64 ] https://repo.mongodb.org/apt/ubuntu
bionic/mongodb-org/5.0 multiverse" | sudo tee /etc/apt/sources.list.d/mongodb-
org-5.0.list

#更新索引
sudo apt-get update

#安装最新版本
sudo apt-get install -y mongodb-org

#安装指定版本
sudo apt-get install -y mongodb-org=5.0.2 mongodb-org-database=5.0.2 mongodb-
org-server=5.0.2 mongodb-org-shell=5.0.2 mongodb-org-mongos=5.0.2 mongodb-org-
tools=5.0.2
```

2.3 二进制包安装

二进制包下载链接

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

2.3.1 安装前准备

- 关闭iptables防火墙
- 关闭 SELinux
- 安装相关包

```
#Red Hat/CentOS：
yum install libcurl openssl

#Ubuntu 18.04 LTS ("Bionic")/Debian 10 "Buster"：
sudo apt-get install libcurl4 openssl

#Ubuntu 16.04 LTS ("Xenial")/Debian 9 "Stretch"：
sudo apt-get install libcurl3 openssl
```

- 关闭大页内存机制,其他系统关闭参照官方文档:

```
https://docs.mongodb.com/manual/tutorial/transparent-huge-pages/
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 enabled, because
they tend to have sparse rather than contiguous memory access patterns. When
running MongoDB on Linux, THP should be disabled for best performance.
```

- 内核参数ulimit

```
https://docs.mongodb.com/manual/tutorial/install-mongodb-on-red-hat/
Starting in MongoDB 4.4, a startup error is generated if the `ulimit` value
for number of open files is under `64000`.
```

范例: 内核优化

```
cat >> /etc/rc.local <<EOF
echo never > /sys/kernel/mm/transparent hugepage/enabled
EOF

chmod +x /etc/rc.local
```

2.3.2 二进制安装 MongoDB

配置文件官方说明

```
https://docs.mongodb.com/manual/reference/configuration-options/
```

范例： 二进制安装MongoDB

```
#创建所需用户和组
useradd  mongod

#创建mongodb所需目录结构
mkdir -p /mongodb/{conf,data,log}

#创建YAML格式的配置文件,早期3.0版本以前是普通文本格式
cat > /mongodb/conf/mongo.conf <<EOF
#日志相关
systemLog:
  destination: file
  path: "/mongodb/log/mongodb.log"  #日志位置
  logAppend: true                   #追加日志
```

```
#数据存储有关
storage:
  dbPath: "/mongodb/data/"              #数据路径的位置

#进程控制
processManagement:
  fork : true                           #后台守护进程

#网络配置有关
net:
  port: 27017                           #端口号,默认不配置端口号，是27017
  bindIp: 0.0.0.0                       #监听地址自MongoDB 3.6版本后默认监听在localhost

#安全验证有关配置
security:
  authorization: enabled              #是否打开用户名密码验证,默认此项为关掉
EOF

cat /mongodb/conf/mongo.conf
systemLog:
  destination: file
  path: "/mongodb/log/mongodb.log"
  logAppend: true
storage:
  dbPath: "/mongodb/data/"
processManagement:
  fork: true
net:
  port: 27017
  bindIp: 0.0.0.0

chown -R  mongod.mongod /mongodb/

tar xf mongodb-linux-x86_64-rhel70-v3.6-latest.tgz -C /usr/local
ln -s /usr/local/mongodb-linux-x86_64-rhel70-3.6.23-8-gc2609ed/
/usr/local/mongodb

#设置PATH变量
echo PATH=/usr/local/mongodb/bin/:'$PATH' > /etc/profile.d/mongodb.sh
. /etc/profile.d/mongodb.sh

#启动
su - mongod
mongod --dbpath /mongodb/data --bind_ip_all --port 27017 --logpath
/mongodb/log/mongod.log --logappend  --fork
mongod -f  /mongodb/conf/mongo.conf


#登录mongodb
mongo

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

#mongodb使用systemd管理
cat > /lib/systemd/system/mongod.service <<EOF
[Unit]
```

```
Description=mongodb
After=network.target remote-fs.target nss-lookup.target

[Service]
Type=forking
User=mongod
Group=mongod
ExecStart=/usr/local/mongodb/bin/mongod --config /mongodb/conf/mongo.conf
ExecReload=/bin/kill -s HUP \$MAINPID
ExecStop=/usr/local/mongodb/bin/mongod --config  /mongodb/conf/mongo.conf --
shutdown
PrivateTmp=true
# file size
LimitFSIZE=infinity
# cpu time
LimitCPU=infinity
# virtual memory size
LimitAS=infinity
# open files
LimitNOFILE=64000
# processes/threads
LimitNPROC=64000
# locked memory
LimitMEMLOCK=infinity
# total threads (user+kernel)
TasksMax=infinity
TasksAccounting=false
# Recommended limits for mongod as specified in
# https://docs.mongodb.com/manual/reference/ulimit/#recommended-ulimit-settings

[Install]
WantedBy=multi-user.target
EOF

systemctl daemon-reload
systemctl enable --now  mongod
```

2.4 客户端连接

2.4.1 MongoDB 后台管理 Shell

2.4.1.1 mongo

如果需要进入 mongodb 后台管理，可以先打开 mongodb 装目录的下的 bin 目录，然后执行 mongo 命令文件。

MongoDB Shell 是 MongoDB 自带的交互式 Javascript shell，用来对 MongoDB 进行操作和管理的交互式环境。

mongo 是MongoDB早期版本使用的shell工具

当mongo进入 mongoDB 后台后，它默认会链接到 test 数据库

```
[root@ubuntu1804 ~]#/usr/local/mongodb/bin/mongo
/usr/local/mongodb/bin/mongo: /lib64/libcurl.so.4: no version information
available (required by /usr/local/mongodb/bin/mongo)
MongoDB shell version v5.0.2
```

```
connecting to: mongodb://127.0.0.1:27017/?
compressors=disabled&gssapiServiceName=mongodb
Implicit session: session { "id" : UUID("4259ec1e-9df6-4f46-b51f-2df105e171b6")
}
MongoDB server version: 5.0.2
......
> help
    db.help()                    help on db methods
    db.mycoll.help()             help on collection methods
    sh.help()                    sharding helpers
    rs.help()                    replica set helpers
    help admin                   administrative help
    help connect                 connecting to a db help
    help keys                    key shortcuts
    help misc                    misc things to know
    help mr                      mapreduce

    show dbs                     show database names
    show collections             show collections in current database
    show users                   show users in current database
    show profile                 show most recent system.profile entries with
time >= 1ms
    show logs                    show the accessible logger names
    show log [name]              prints out the last segment of log in memory,
'global' is default
    use <db_name>                set current database
    db.mycoll.find()             list objects in collection mycoll
    db.mycoll.find( { a : 1 } )  list objects in mycoll where a == 1
    it                           result of the last line evaluated; use to
further iterate
    DBQuery.shellBatchSize = x   set default number of items to display on shell
    exit                         quit the mongo shell
> db
test
```

2.4.1.2 mongosh

官方说明

```
https://docs.mongodb.com/mongodb-shell/install/
#下载路径
https://www.mongodb.com/try/download/shell
```

```
#mongo提示将被mongosh代替
[root@ubuntu1804 ~]#mongo
MongoDB shell version v5.0.3
connecting to: mongodb://127.0.0.1:27017/?
compressors=disabled&gssapiServiceName=mongodb
Implicit session: session { "id" : UUID("e726b0cd-d021-4092-856e-f8fa7b9e7b3b")
}
MongoDB server version: 5.0.3
================
Warning: the "mongo" shell has been superseded by "mongosh",
which delivers improved usability and compatibility.The "mongo" shell has been
deprecated and will be removed in
an upcoming release.
We recommend you begin using "mongosh".
For installation instructions, see
https://docs.mongodb.com/mongodb-shell/install/
```

范例：安装mongosh

```
[root@rocky8 ~]#cat /etc/yum.repos.d/mongosh.repo
[mongodb-org-5.0]
name=MongoDB Repository
baseurl=https://repo.mongodb.org/yum/redhat/$releasever/mongodb-org/5.0/x86_64/
gpgcheck=1
enabled=1
gpgkey=https://www.mongodb.org/static/pgp/server-5.0.asc

[root@rocky8 ~]#yum install -y mongodb-mongosh
[root@rocky8 ~]#rpm -ql mongodb-mongosh
/usr/bin/mongosh
/usr/libexec/mongocryptd-mongosh
/usr/share/doc/mongodb-mongosh-1.1.2
/usr/share/doc/mongodb-mongosh-1.1.2/README
/usr/share/doc/mongodb-mongosh-1.1.2/THIRD_PARTY_NOTICES
/usr/share/doc/mongodb-mongosh-1.1.2/mongosh.1.gz
/usr/share/licenses/mongodb-mongosh-1.1.2
/usr/share/licenses/mongodb-mongosh-1.1.2/LICENSE-mongocryptd
/usr/share/licenses/mongodb-mongosh-1.1.2/LICENSE-mongosh

[root@rocky8 ~]#mongosh
Current Mongosh Log ID: 6196240e80c1fcef4feef8fd
Connecting to:      mongodb://127.0.0.1:27017/?
directConnection=true&serverSelectionTimeoutMS=2000
Using MongoDB:      5.0.3
Using Mongosh:      1.1.2

For mongosh info see: https://docs.mongodb.com/mongodb-shell/

------
   The server generated these startup warnings when booting:
   2021-11-18T17:55:45.669+08:00: Access control is not enabled for the
database. Read and write access to data and configuration is unrestricted
   2021-11-18T17:55:45.670+08:00: /sys/kernel/mm/transparent_hugepage/enabled is
'always'. We suggest setting it to 'never'
------
```

```
Warning: Found ~/.mongorc.js, but not ~/.mongoshrc.js. ~/.mongorc.js will not be
loaded.
  You may want to copy or rename ~/.mongorc.js to ~/.mongoshrc.js.
test>
```

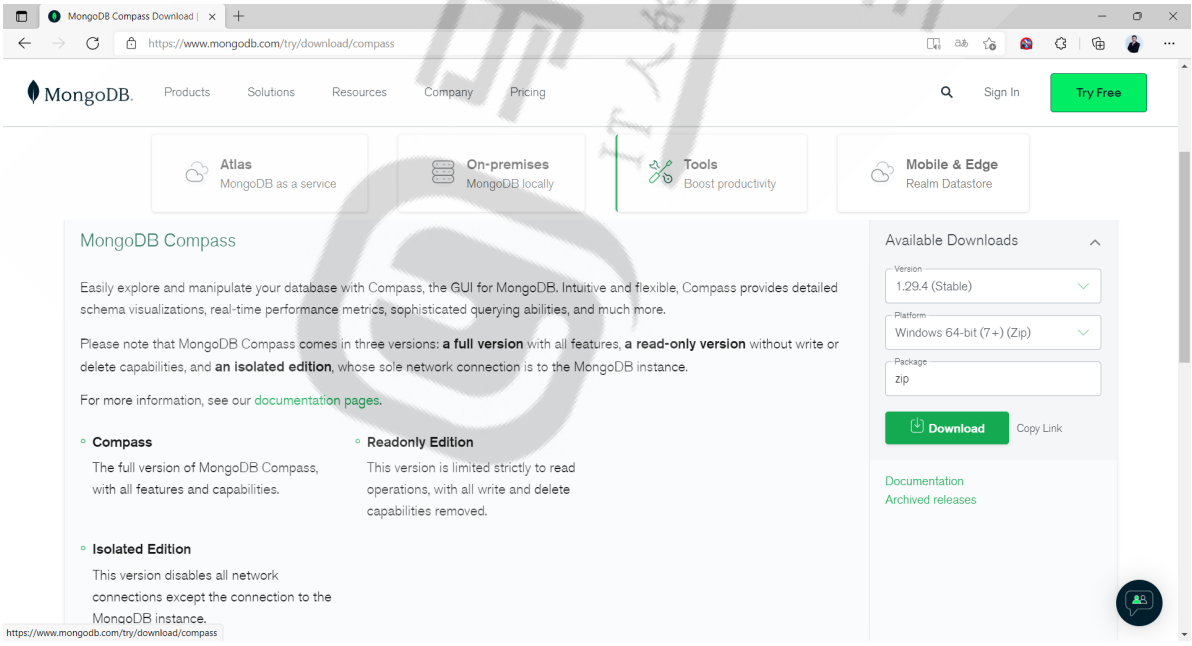
```
[root@rocky8 ~]#mongosh
Current Mongosh Log ID: 61962325175b8f29b73c7b14
Connecting to:          mongodb://127.0.0.1:27017/?directConnection=true&serverSelectionTimeoutMS=2000
Using MongoDB:          5.0.3
Using Mongosh:          1.1.2

For mongosh info see: https://docs.mongodb.com/mongodb-shell/

------
   The server generated these startup warnings when booting:
   2021-11-18T17:55:45.669+08:00: Access control is not enabled for the database. Read and write access to dat
a and configuration is unrestricted
   2021-11-18T17:55:45.670+08:00: /sys/kernel/mm/transparent_hugepage/enabled is 'always'. We suggest setting
it to 'never'
------

Warning: Found ~/.mongorc.js, but not ~/.mongoshrc.js. ~/.mongorc.js will not be loaded.
  You may want to copy or rename ~/.mongorc.js to ~/.mongoshrc.js.
test>
```

2.4.2 使用 MongoDB Compass 连接

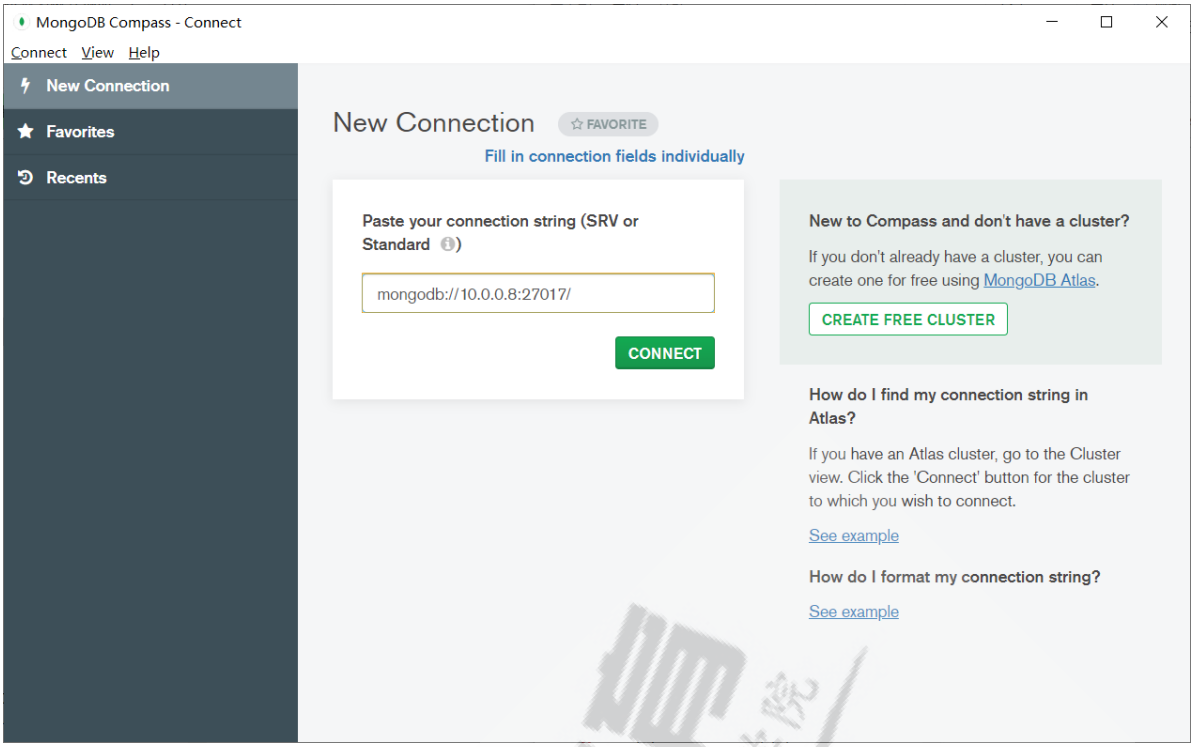
MongoDB Compass是MongoDB官网提供的一个可视化管理工具,可以实现创建数据库、管理集合和文档、运行临时查询、评估和优化查询、性能图表、构建地理查询等功能

MongoDB Compass下载官方地址：

```
https://www.mongodb.com/download-center/compass
https://downloads.mongodb.com/compass/mongodb-compass-1.29.4-win32-x64.msi
https://downloads.mongodb.com/compass/mongodb-compass-1.29.4.x86_64.rpm
```



下载完成之后，打开MongoDB Compass应用程序，进入Compass后首先出现连接界面如下：



2.4.3 Python 访问 MongoDB

```
[root@rocky8 ~]#yum -y install python3
[root@rocky8 ~]#pip3 install pymongo
[root@rocky8 ~]#cat  test_mongodb.py
#!/usr/bin/python3
from pymongo import MongoClient

# 客户端连接
#client = MongoClient(host='127.0.0.1', port=27017)
client = MongoClient('mongodb://127.0.0.1:27017')
print(client)

# 指定数据库 # Database类
```

```
#db = client.test
db = client['test']
print(db)

# 集合 Collection类
#users = db['users']
users = db.users
print(users)

for i in range(10):
    user = {'id':i, 'name':'wang' + str(i), 'age':20}
    users.insert_one(user)
for x in users.find():
    print(x)
client.close()

[root@rocky8 ~]#python3 test_mongodb.py
MongoClient(host=['127.0.0.1:27017'], document_class=dict, tz_aware=False,
connect=True)
Database(MongoClient(host=['127.0.0.1:27017'], document_class=dict,
tz_aware=False, connect=True), 'test')
Collection(Database(MongoClient(host=['127.0.0.1:27017'], document_class=dict,
tz_aware=False, connect=True), 'test'), 'users')
{'_id': ObjectId('6195adb943ee2188fe347d99'), 'id': 0, 'name': 'wang0', 'age':
20}
{'_id': ObjectId('6195adb943ee2188fe347d9a'), 'id': 1, 'name': 'wang1', 'age':
20}
{'_id': ObjectId('6195adb943ee2188fe347d9b'), 'id': 2, 'name': 'wang2', 'age':
20}
{'_id': ObjectId('6195adb943ee2188fe347d9c'), 'id': 3, 'name': 'wang3', 'age':
20}
{'_id': ObjectId('6195adb943ee2188fe347d9d'), 'id': 4, 'name': 'wang4', 'age':
20}
{'_id': ObjectId('6195adb943ee2188fe347d9e'), 'id': 5, 'name': 'wang5', 'age':
20}
{'_id': ObjectId('6195adb943ee2188fe347d9f'), 'id': 6, 'name': 'wang6', 'age':
20}
{'_id': ObjectId('6195adb943ee2188fe347da0'), 'id': 7, 'name': 'wang7', 'age':
20}
{'_id': ObjectId('6195adb943ee2188fe347da1'), 'id': 8, 'name': 'wang8', 'age':
20}
{'_id': ObjectId('6195adb943ee2188fe347da2'), 'id': 9, 'name': 'wang9', 'age':
20}
```

3 MongoDB 管理

3.1 MongoDB 操作数据库

3.1.1 MongoDB 默认数据库介绍

MongoDB 默认存在的库

- admin库:系统预留库，MongoDB系统管理库
- local库:本地预留库，存储关键日志
- config库:MongoDB配置信息库
- test:登录时默认存在的测试库,生产中可以将之删除

3.1.2 命令种类

官方帮助

```
https://docs.mongodb.com/manual/
```

db对象相关命令

```
show dbs|show databases     #显示数据库,相当于MySQL中的show databases
use <库名>                   #切换数据库
use admin
> db                    #查看当前库相当于MySQL中的select database()
> db.shutdownServer()   #关闭服务

show collections        #显示表列表,相当于MySQL中的show tables
show tables
```

库级命令

```
db.[TAB][TAB]
db.help()
```

表和文档

```
db.<collection>.[TAB][TAB]
db.<collection>.help ()
```

rs 复制集有关(replication set) :

```
rs.[TAB][TAB]
rs.help ()
```

sh分片集群(sharding cluster)

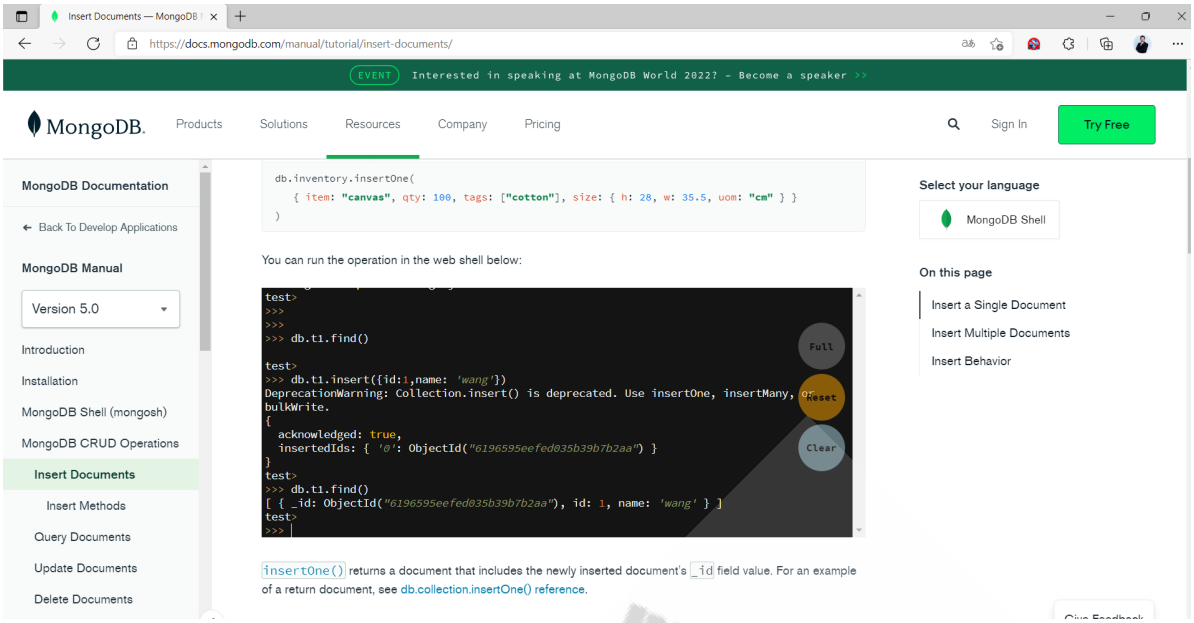
```
sh.[TAB][TAB]
sh.help ()
```

3.1.3 mongodb 对象操作

```
mongo  mysql
库 ----->库
集合----->表
文档----->数据行
```

官方帮助

```
https://docs.mongodb.com/manual/crud/
```



3.1.3.1 库的操作

```
#use可以自动建库,如果库内没有数据,则不使用时会自动删除库
>use test
#删除当前所在库
>db.dropDatabase()
{ "dropped" : "test", "ok" : 1 }
 #可以直接切换至不存在的库
>use testdb1
>db
testdb1
```

3.1.3.2 集合(表)的操作

```
#建表方法1:直接创建表
app>db.createCollection ('student')

#建表方法2:当插入一个文档的时候，一个集合就会自动创建。
use magedu
db.test.insert ({name: " wang"})
db.student.insert({id:1, name : "wang" ,age:20, gender: "m" })
db.student.insert ({id:2, name: "zhao"})
db.student.insert([{id:3,name: "tom",age: 22},{id:4,name: "jerry",age: 23}])
db.student.insert ({a: "b",c: "d"})
db.student.insert ({a:1,c:2})

#查看库中的集合列表
show tables
show collections

#查看集合的信息
db.student.stats()

#删除集合
db.student.drop()
```

3.1.3.3 记录(文档)操作

```
#数据录入:
for(i=0;i<10000;i++){db.student.insert({uid:i, name: "wang"+i , "age":18,
"date": new Date()})}
for(i=0;i<10000;i++){db.users.insert({uid:i, name: "wang"+i ,
"age":i,address:"beijing"})}
for(i=0;i<10000;i++){db.random.insertOne({num: Math.random()* 100000})}

#查询数据行数:
db.student.count()

#全表查询,默认每页显示20行
db.student.find()
it  #继续显示下面行

#只显示前3个
db.student.find().limit(3)

#跳过前面10个显示后面的文档
db.student.find().skip(10)

#每页显示50条记录:
DBQuery.shellBatchsize=50

#按照条件查询
db.student.find({uid: 10})
#查询大于等于18并正序排序users集合
db.users.find({age: { $gte: 18 }}).sort({age: 1})

#查询users集合中的小于18的前5个文档中两个key:name和address
db.users.find({age: {$lt: 18 }},{ name: 1, address: 1}).limit(5)

#以标准的json格式显示数据
db.student.find({uid:10}).pretty()
#显示效果如下
{
    "id": objectId ( "8bc516e60d13144c89dead66"),
    "uid": 10,
    "name": "wang10",
    "age": 18,
    "date": ISODate ( "2018-01-12T02:38:57.113Z")
}

#修改文档,如果没有加multi: true.默认只修改第一个符合条件的文档
db.student.update({uid:1},{$set:{age: 20}},{multi: true})

#删除指定的文档
db.student.remove({uid: 10})

#删除集合中所有记录,但表还存在
db.student.remove({})


#查看集合存储信息,包括集合中索引和数据压缩存储后的大小
db.student.totalSize()
```

3.2 用户及权限管理

MongoDB数据库默认是没有用户名及密码的，即无权限访问限制。为了方便数据库的管理和安全，应启用认证和创建数据库用户

3.2.1 关于用户验证库

- 创建用户时,use所在的库就是此用户的验证库
- 登录时，必须明确指定验证库才能登录
- 一个数据库可以成为多个用户的验证库,但一个用户只能使用一个验证库
- 对于管理员用户，必须在admin下创建,即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
- 如果直接登录到数据库,不进行use,默认的验证库是test
- 从3.6版本开始，配置文件中不添加bindIp参数，默认不允许远程登录，只能本地管理员登录。

3.2.2 用户管理

官方帮助

```
https://docs.mongodb.com/manual/tutorial/create-users/
```

3.2.2.1 开启用户认证

MongoDB数据库默认是无认证功能,可以直接登录做任务操作,为了安全需启用认证功能

```
#注意:启用认证后,仍然可以直接登录,但无权限做数据操作,但可以创建用户
#方法1:命令行方式通过“--auth”参数
mongod --auth --port 27017 --dbpath /data/db

#方法2,修改配置文件中，加入以下配置
cat >> /mongodb/conf/mongo.conf <<EOF
security:
  authorization: enabled
EOF

systemctl restart  mongod
```

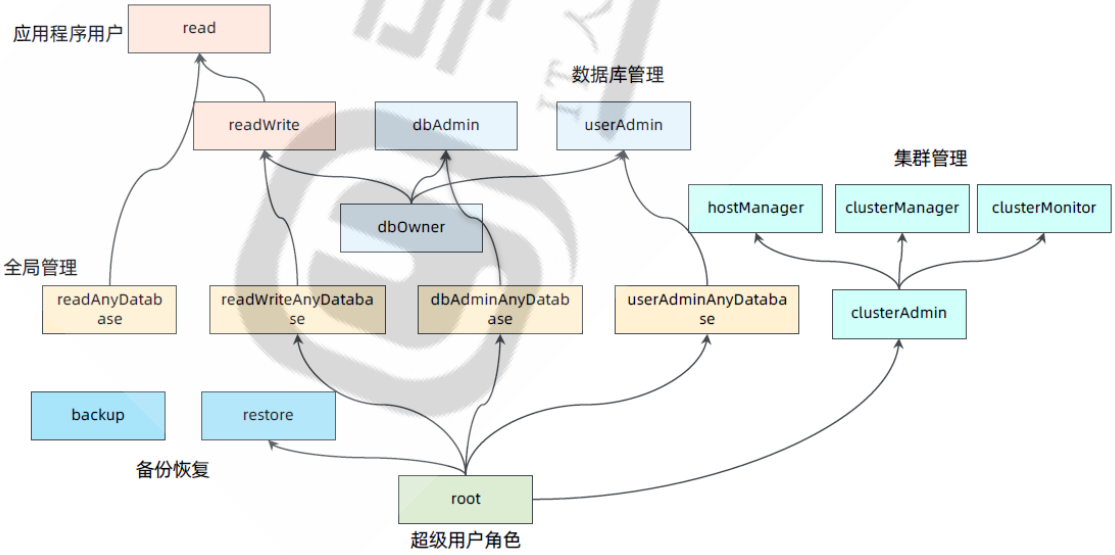
3.2.2.2 用户管理说明

用户的创建,需要基于指定数据库,即用户是存放于特定数据库的,即验证库

```
#创建用户
use <db> #切换至需要创建用户的验证库后,再创建用户
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,比如:root, readWrite, read
  db:上面角色作用的数据库对象

#验证用户
use <验证库>
db.auth('用户名', '密码')

#用户保存在admin库的users集合中
use admin
db.system.users.find()

#登录验证
mongo -u用户名 -p密码  <验证库>                                  #本地登录,如果验证库admin省略,登录后默认在test库
mongo -u用户名 -p密码  <MongoDB主机IP>/<验证库>     #远程登录,验证库不可以省略


#修改密码
db.changeUserPassword("用启名", "新密码")

#删除用户
use <验证库>
db.dropUser("用户名")
```

系统内置角色的说明



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

更多关于用户权限的说明参照：https://docs.mongodb.com/manual/core/security-built-in-roles/

3.2.2.3 范例

范例: 创建MongoDB的超级管理员

```
$mongo
#创建超级管理员root(也可以是任意用户名)管理所有数据库（必须use admin再去创建,因为超级用户是存放在admin库中的）
use admin

db.createUser ( {user: "root",pwd: "123456",roles: [{role: "root", db: "admin"
}]})

#验证用户
use admin
db.auth('root', '123456')

#修改配置文件中，加入以下配置
cat >> /mongodb/conf/mongo.conf <<EOF
security:
  authorization: enabled
EOF

#重启mongodb
systemctl restart mongod
#或者
mongod -f /mongodb/conf/mongo.conf --shutdown
mongod -f /mongodb/conf/mongo.conf

#登录验证
mongo -uroot -p123456  admin  #本地登录,如果验证库admin省略,登录后默认在test库而非admin
```

```
mongo -uroot -p123456  10.0.0.100/admin  #远程登录,验证库admin不可以省略

#或者交互式验证
mongo
use admin
db.auth('root','123456')

#在admin库中system.users表中查看所有的用户列表
use admin
db.system.users.find().pretty()

#查看当前库中用户
admin> use test
test> db.getUsers()
```

范例: 创建普通用户

```
use test
db.createUser(
  {
    user: "myTester",
    pwd:  passwordPrompt(),   //交互式指定密码
    roles: [ { role: "readWrite", db: "test" },
             { role: "read", db: "reporting" } ]
  }
)

mongosh --port 27017 -u "myTester"  --authenticationDatabase "test" -p
```

范例: 创建应用用户

```
use magedu
db.createUser (
{
    user: "app01",
    pwd: "123456",
    roles: [{role: "readWrite", db: "magedu"}]
}
)

db.createUser (
{
    user: "app02",
    pwd: "123456",
    roles: [{role: "readWrite", db: "magedu"}]
}
)

#连接测试
mongo -uapp01 -p123456
mongo -uapp01 -p123456 magedu
mongo -uapp01 -p123456 10.0.0.100/magedu
mongo -uapp02 -p123456 10.0.0.100/magedu

#查询mongodb中的用户信息
mongo -uroot -p123456 10.0.0.100/admin
```

```
db.system.users.find().pretty()
```

范例:删除用户

```
#以管理员root身份登录，use到被删除用户的验证库才能删除用户
use magedu
db.createUser ( {user: "app02" , pwd: "app02", roles:[ { role: "readWrite" , db:
"magedu" }]})

mongo -uroot -p123456 10.0.0.100/admin
use magedu
db.dropUser ("app02")
```

4 MongoDB 复制集

4.1 基本原理

官方说明

```
https://docs.mongodb.com/manual/replication/
```

4.1.1 什么是复制集 Replica Set

MongDB 像MySQL一样,支持类似的主从复制架构，但无法实现自动故障转移，所以官方推荐使用复制集

MongoDB复制集是将数据同步在多个服务器的过程

复制集提供了数据的冗余备份，并在多个服务器上存储数据副本，保证数据的安全性

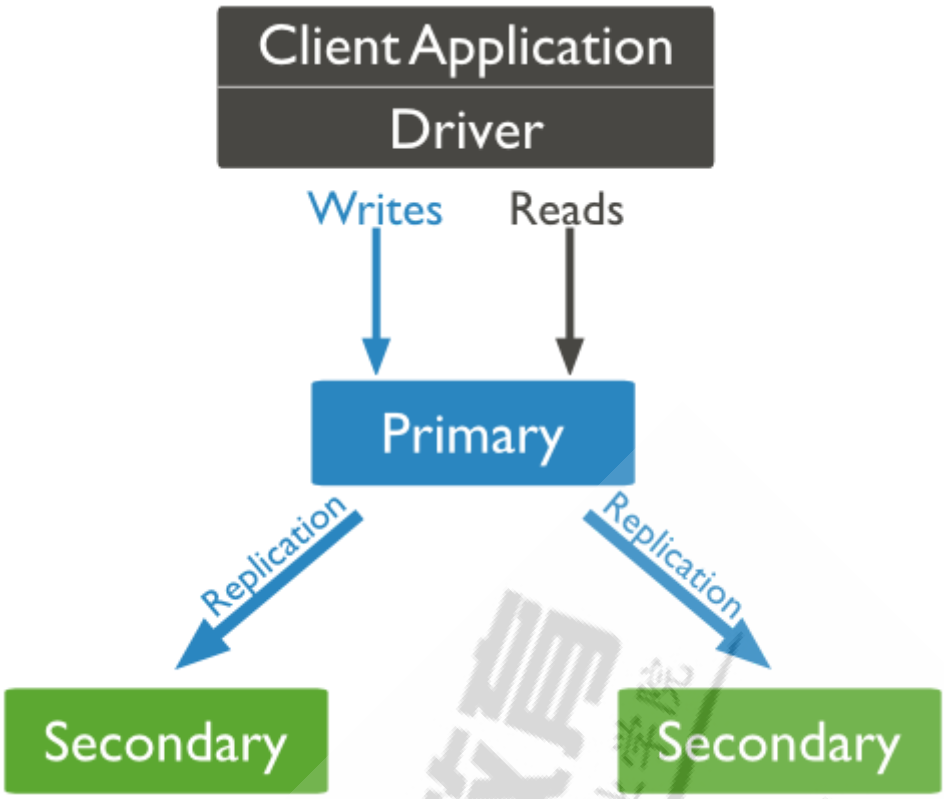
复制集还允许从硬件故障和服务中断中恢复数据。当故障时,会自动选举新master节点,实现集群的高可用

复制集功能

- 数据高可用性
- 异地容灾，实现灾难恢复
- 无需停机维护（如备份，重建索引，压缩）
- 分布式读取数据，读操作的负载均衡

4.1.2 复制集架构

MongoDB复制结构图如下所示



MongoDB的复制至少需要两个节点。其中一个是主节点，负责处理客户端请求，其余的都是从节点，负责复制主节点上的数据

MongoDB各个节点常见的搭配方式为：一主一从、一主两从此方式最多

主节点记录在其上的所有操作oplog，从节点定期轮询主节点获取这些操作，然后对自己的数据副本执行这些操作，从而保证从节点的数据与主节点一致

客户端写入数据到主节点时， 主节点与从节点进行数据交互保障数据的一致性

复制集特征：

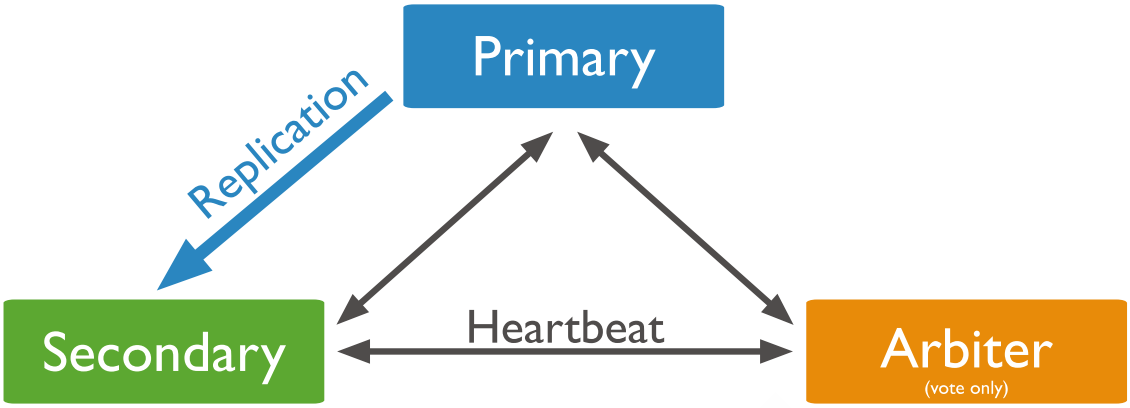
- N 个奇数节点的集群
- 基于选举机制，任何节点可作为主节点
- 所有写入操作都在主节点上，所以增加节点不会提高系统写性能，可以提升读性能
- 主节点故障时，会自动选举出新节点代替，自动故障转移

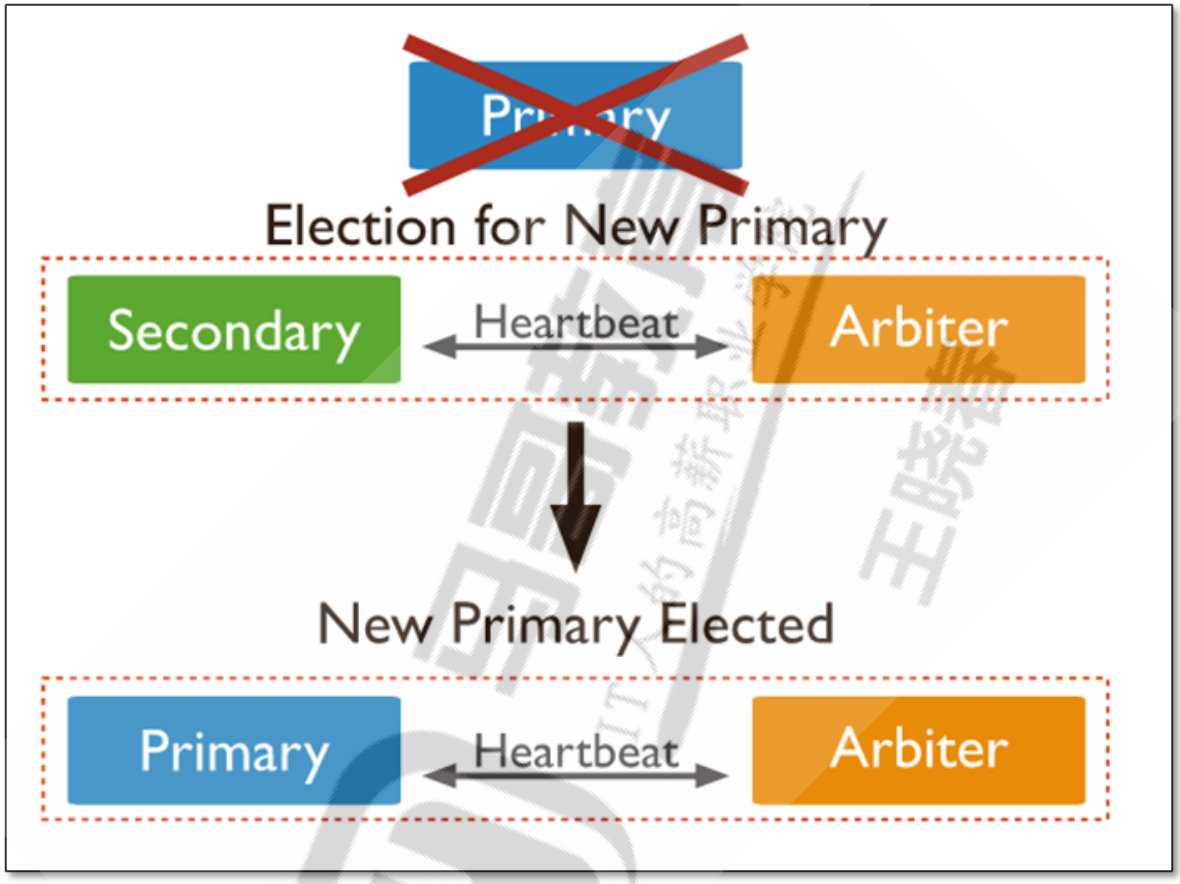
多个节点数据同步的实现

- 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

4.1.3 复制集模式中主要角色和选举

4.1.3.1 复制集的主要角色





主节点[Primary]

接收所有的写请求，然后把修改同步到所有Secondary。一个Replica Set只能有一个Primary节点，当Primary挂掉后，其他Secondary或者Arbiter节点会重新选举出来一个主节点。默认读请求也是发到Primary节点处理的，如果需要转发到Secondary,需要在客户端修改一下连接配置。

副本节点[Secondary]

与主节点保持同样的数据集。当主节点挂掉的时候，可以参与选举出新主节点。

仲裁者[Arbiter]

不保存数据，不参与选主，只进行选主投票。使用Arbiter可以减轻数据存储的硬件需求，Arbiter跑起来几乎没什么大的硬件资源需求，在生产环境下它和其他数据节点不要部署在同一台机器上。实际生产环境当前不推存使用Arbiter节点类型

MongoDB 使用Raft选举机制,而MySQL MGR 用的是Paxos,而Raft本质是Paxos的变种

注意，一个自动failover的Replica Set节点数必须为奇数，目的是选主投票的时候要有一个大多数才能进行选主决策。

4.1.3.2 Primary选举实现

复制集通过replSetInitiate命令（或mongo shell的rs.initiate()）进行初始化，初始化后各个成员间开始发送心跳消息，并发起Primary选举操作，获得大多数成员投票支持的节点，会成为Primary，其余节点成为Secondary。

选举实现

- 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
- 复制集中最多可以有50个节点，但具有投票权的节点最多7个，且为奇数个投票成员

大多数的定义

假设复制集内投票成员（后续介绍）数量为N，则大多数为 N/2 + 1，当复制集内存活成员数量不足大多数时，整个复制集将无法选举出Primary，复制集将无法提供写服务，处于只读状态。

投票成员数	大多数	容忍失效数
1	1	0
2	2	0
3	2	1
4	3	1
5	3	2
6	4	2
7	4	3

通常建议将复制集成员数量设置为奇数，从上表可以看出3个节点和4个节点的复制集都只能容忍1个节点失效，从服务可用性的角度看，其效果是一样的。（但无疑4个节点能提供更可靠的数据存储）

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）

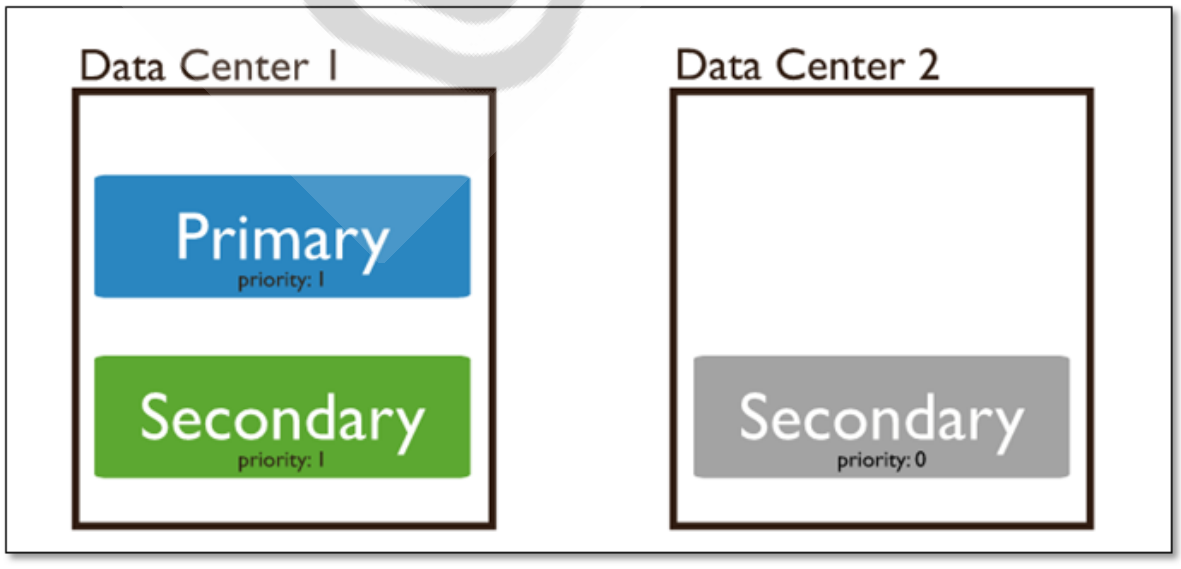
4.1.3.3 复制集中其它角色和选项说明

4.1.3.3.1 角色说明

成员	说明
Secondary	正常情况下，复制集的Seconary会参与Primary选举（自身也可能会被选为Primary），并从Primary同步最新写入的数据，以保证与Primary存储相同的数据。Secondary可以提供读服务，增加Secondary节点可以提供复制集的读服务能力，同时提升复制集的可用性。另外，Mongodb支持对复制集的Secondary节点进行灵活的配置，以适应多种场景的需求。
Arbiter	Arbiter节点只参与投票，不能被选为Primary，并且不从Primary同步数据。比如你部署了一个2个节点的复制集，1个Primary，1个Secondary，任意节点宕机，复制集将不能提供服务了（无法选出Primary），这时可以给复制集添加一个Arbiter节点，即使有节点宕机，仍能选出Primary。Arbiter本身不存储数据，是非常轻量级的服务，当复制集成员为偶数时，最好加入一个Arbiter节点，以提升复制集可用性。
Priority0	默认Priority为1，值最大优先级越高。设置Priority为0节点的选举优先级为0，不会被选举为Primary，但可以投票，比如:跨机房A、B部署了一个复制集，并且想指定Primary必须在A机房，这时可以将B机房的复制集成员Priority设置为0，这样Primary就一定会是A机房的成员。注意：如果这样部署，最好将大多数节点部署在A机房，否则网络分区时可能无法选出Primary
Vote0	Mongodb 3.0里，复制集成员最多50个，参与Primary选举投票的成员最多7个，其他成员的vote属性必须设置为0，即不参与投票。
Hidden	Hidden节点不能被选为主（Priority必须为0），并且对应用不可见。因Hidden节点不会接受Driver的请求，可使用Hidden节点做一些数据备份、离线计算的任务，不会影响复制集的服务。
Delayed	Delayed节点必须是Hidden节点，并且其数据落后与Primary一段时间（可配置，比如1个小时）。因Delayed节点的数据比Primary落后一段时间，当错误或者无效的数据写入Primary时，可通过Delayed节点的数据来恢复到之前的时间点。

4.1.3.3.2 Priority 0节点

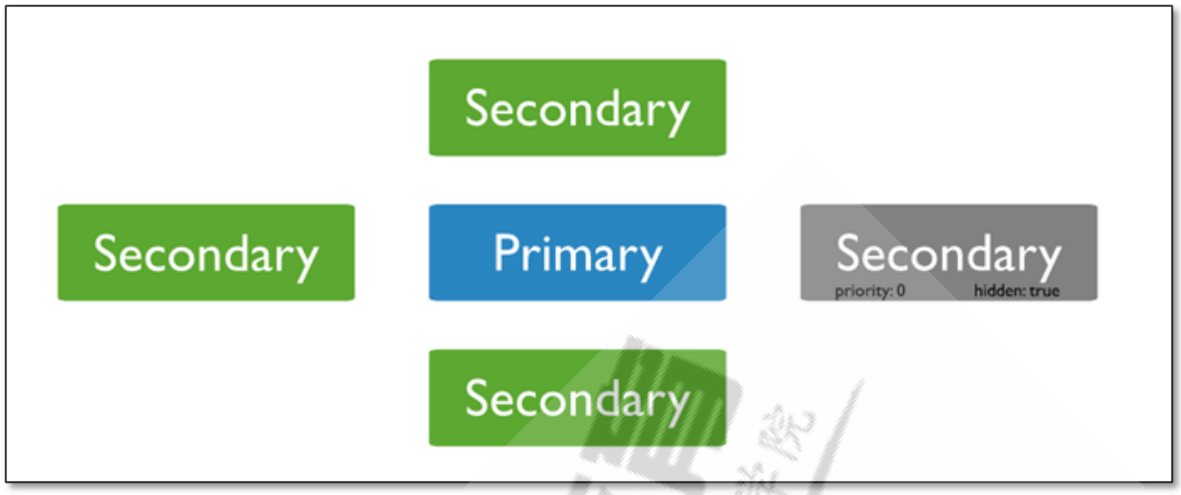
作为一个辅助可以作为一个备用。在一些复制集中，可能无法在合理的时间内添加新成员的时候。备用成员保持数据的当前最新数据能够替换不可用的成员。



4.1.3.3.3 Hidden 节点

客户端将不会把读请求分发到隐藏节点上，即使设定了复制集读选项 。

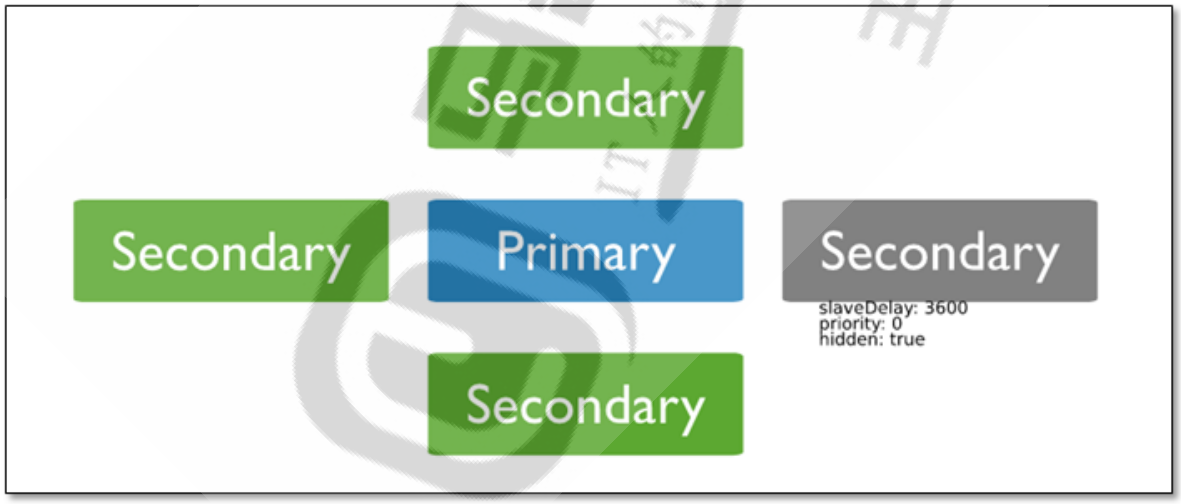
这些隐藏节点将不会收到来自应用程序的请求。可以将隐藏节点专用于报表节点或是备份节点。 延时节点也应该是一个隐藏节点。



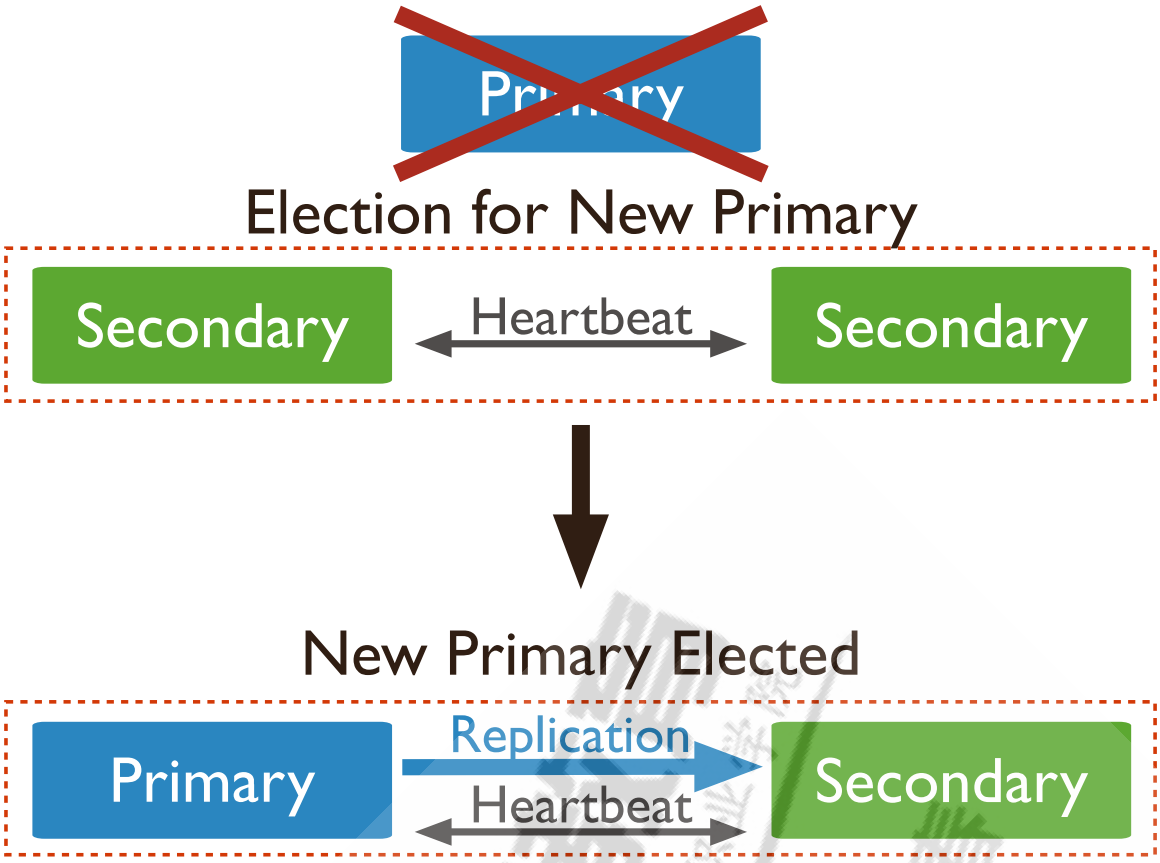
4.1.3.3.4 Delayed 节点

延时节点的数据集是延时的，因此它可以帮助我们在人为误操作或是其他意外情况下恢复数据。

举个例子，当应用升级失败，或是误操作删除了表和数据库时，可以通过延时节点进行数据恢复。



4.1.4 自动的故障恢复



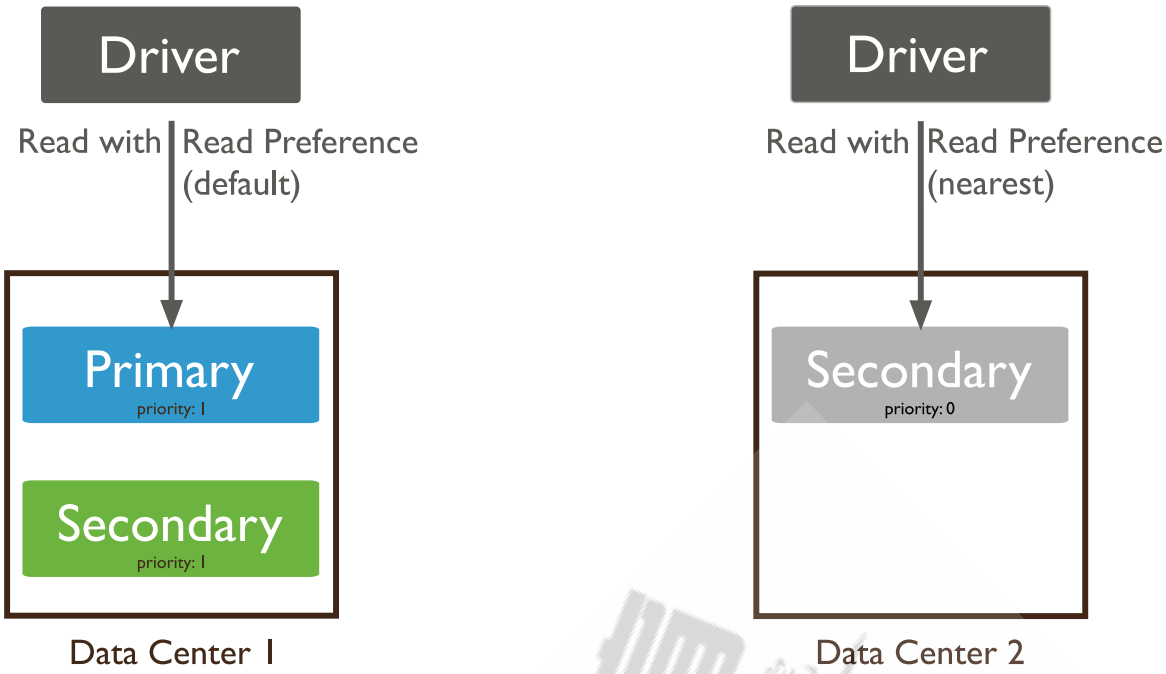
如果发生主库宕机，复制集内部会进行投票选举，选择一个新的主库替代原有主库对外提供服务。同时复制集会自动通知客户端程序，主库已经发生切换了。应用就会连接到新的主库

4.1.5 read preference 模式

默认情况下，应用程序将其读取操作指向复制集的主节点（即读取首选项模式"primary"）。但是，客户端可以通过read preference 模式指定将读取操作发送给到从节点

read preference 模式官方说明

```
https://docs.mongodb.com/manual/core/read-preference/
```



Read Preference Mode	Description
primary	主节点，默认模式，读操作只在主节点，如果主节点不可用，报错或者抛出异常。
primaryPreferred	首选主节点，大多情况下读操作在主节点，如果主节点不可用，如故障转移，读操作在从节点。
secondary	从节点，读操作只在从节点， 如果从节点不可用，报错或者抛出异常。
secondaryPreferred	首选从节点，大多情况下读操作在从节点，特殊情况（如单主节点架构）读操作在主节点。
nearest	最邻近节点，读操作在最邻近的成员，可能是主节点或者从节点。

4.2 复制集实现

4.2.1 复制集环境规划

复制集要求三个以上的 MongoDB节点或者实例

下面案例以多实例,采用多个端口: 28017、28018、28019、28020实现

4.2.2 准备配置文件

```
su - mongod
mkdir -p /mongodb/{28017,28018,28019,28020}/{conf,data,log}

#多套配置文件
/mongodb/28017/conf/mongod.conf
/mongodb/28018/conf/mongod.conf
/mongodb/28019/conf/mongod.conf
/mongodb/28020/conf/mongod.conf

#配置文件内容
```

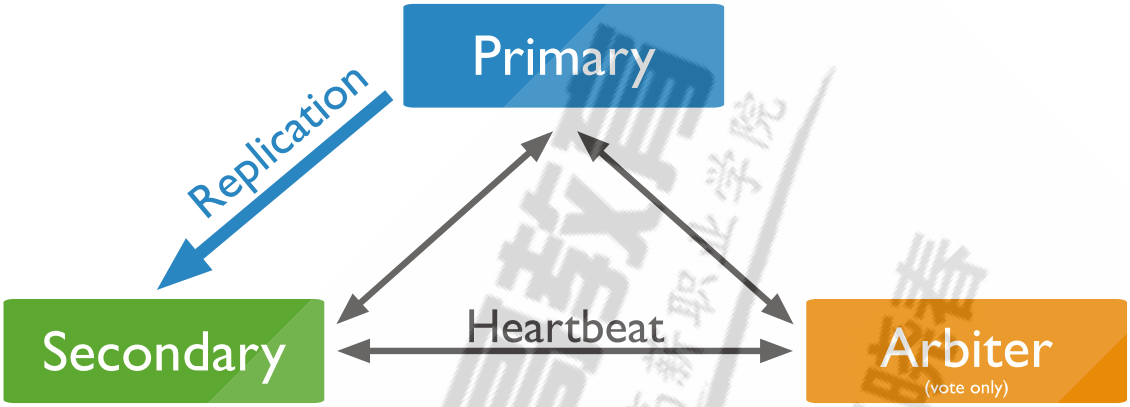
```
cat > /mongodb/28017/conf/mongod.conf <<EOF
systemLog:
  destination: file
  path: /mongodb/28017/log/mongodb.log
  logAppend: true
storage:
  dbPath: /mongodb/28017/data
processManagement:
  fork: true
net:
  bindIp: 0.0.0.0
  port: 28017
replication:
  replSetName: myrepl  #指定复制集名称,所有复制集成员此名称要一致
EOF

#说明:
3.x版本以上默认是wiredtiger引擎，类似于MySQL InnoDB支持事务，文档锁
2.x版本中，默认的是MMAPv1引擎，相当于MySQL MyISAM引擎

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


#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
```

4.2.3 配置复制集: 1主2从

```
#方法1
mongo --port 28017 admin
#指定复制集的所有成员信息
config = { _id: 'myrepl', members: [
    {_id: 0, host: '10.0.0.100:28017'},
    {_id: 1, host: '10.0.0.100:28018'} ,
    {_id: 2, host: '10.0.0.100:28019'}]
    }

#以json格式显示变量内容
printjson(config)

#初始化并启动复制集
rs.initiate(config)

#方法2
mongo --port 28017
>rs.initiate ( )
>rs.add("10.0.0.100:28018")
>rs.add("10.0.0.100:28019")
```

```
#查询复制集状态
rs.status()
rs.hello()
rs.isMaster()

#显示复制信息
myrepl:SECONDARY> db.printReplicationInfo()
configured oplog size:   4968.0986328125MB
log length start to end: 495secs (0.14hrs)
oplog first event time:  Tue Nov 16 2021 10:27:35 GMT+0800 (CST)
oplog last event time:   Tue Nov 16 2021 10:35:50 GMT+0800 (CST)
now:                     Tue Nov 16 2021 10:35:51 GMT+0800 (CST)
```

4.2.4 配置复制集: 1主1从1Arbiter




```
mongo -port 28017 admin

config = { _id: 'myrepl', members: [
          { _id: 0, host: '10.0.0.100:28017'},
          { _id: 1, host: '10.0.0.100:28018'},
          { _id: 2, host: '10.0.0.100:28019', "arbiterOnly": true}]
         }

rs.initiate (config)

#查询复制集状态
rs.status()
```

4.2.5 修改已有集群为1主1从1个Arbiter

将已有节点28019，替换为arbiter

```
[mongod@mongodb ~]$ mongo --port 28017 admin
myrepl:PRIMARY> rs.remove ("10.0.0.100:28019")
myrepl:PRIMARY> rs.addArb ("10.0.0.100:28019")

#上述操作会关闭节点,所以需要重新启动指定节点的monogodb
[mongod@mongodb ~]$mongod -f /mongodb/28019/conf/mongod.conf
```

4.3 复制集管理操作

官方帮助

```
https://docs.mongodb.com/manual/reference/replication/
```

4.3.1 查看复制集状态

```
#查看整体复制集状态
rs.status()

#查看当前是否是主节点
rs.isMaster()

#查看复制集配置信息
rs.conf()
```

4.3.2 添加和删除节点

```
#删除一个节点
rs.remove ("ip: port")

#新增从节点
rs.add ("ip: port")

#新增仲裁节点
rs.addArb ("ip: port")
```

范例: 增加和删除一个节点

```
#添加一个节点
myrepl:PRIMARY> rs.add ("10.0.0.100:28019")

#删除一个节点
myrepl:PRIMARY> rs.remove ("10.0.0.100:28019")
{ "ok" : 1 }
myrepl:PRIMARY>rs.isMaster()
```

范例: 添加arbiter节点

```
#1、连接到主节点
[mongod@mongodb ~]$ mongo --port 28018 admin

#2、添加仲裁节点
myrepl: PRIMARY> rs.addArb ("10.0.0.100:28020")

#3、查看节点状态
myrepl: PRIMARY>rs.isMaster()
{

    "hosts": [
        "10.0.0.100:28017",
        "10.0.0.100:28018",
        "10.0.0.100:28019"
     ],
     "arbiters" : [
     "10.0.0.100:28020"
     ],
```

```
}
```

4.3.3 特殊从节点管理

arbiter节点:主要负责选主过程中的投票，但是不存储任何数据，也不提供任何服务

hidden节点:隐藏节点，不参与选主，也不对外提供服务。

delay节点:延时节点，数据落后于主库一段时间，因为数据是延时的，也不应该提供服务或参与选主，所以通常会配合hidden (隐藏),可用于防止逻辑删除,一股情况下会将delay+hidden一起配置使用,配置延时节点(一般延时节点也配置成hidden)

优先级为0的节点的特点

```
此节点丧失了当选Primary的机会。永远不会上位
此节点虽然不能当选Primary但是却可以投票
此节点正常参与Primary产生的oplog的读取，进行数据备份和命令执行
此节点正常参与客户端对于数据的读取，进行担当负载均衡的工作
```

范例： 实现特殊节点功能

```
#在主节点执行下面操作配置特殊节点并保存
config=rs.conf()                        #定义变量
config.members[3].hidden=true           #设置hidden,3表示第4个成员,从0开始表示第1个,是
rs.conf()显示的顺序,不是_id的值
config.members[3].priority=0            #优先级0表示不参选master，但可以投票,值越大优先级越
高
config.members[3].arbiterOnly=true      #也可将延时节点配置为arbiter节点
config.members[3].slaveDelay=120        #延时时长,以秒为单位,生产可延时3-6小时
config.members[3].votes=0               #持有票数为0
rs.reconfig(config)                     #加载配置

#取消以上配置
config=rs.conf()
config.members[3].priority=1
config.members[3].hidden=false
config.members[3].slaveDelay=0
config.members[3].votes=1
rs.reconfig(config)

#配置成功后,通过以下命令查询配置后的属性
rs.conf();
```

4.3.4 管理从节点

注：在mongodb复制集当中，默认从库不允许读。在从库打开读配置

```
#打开从节点读支持
#新版命令
myrepl:SECONDARY> rs.secondaryOk()

#旧版命令已废弃
myrepl:SECONDARY> rs.slaveOk()
```

5 MongoDB 分片集群

5.1 MongoDB 分片集群说明

分片（sharding）是MongoDB用来将大型集合分割到不同服务器（或者说一个集群）上所采用的方法。尽管分片起源于关系型数据库分区，但MongoDB分片完全又是另一回事。

和MySQL分区方案相比，MongoDB Sharding Cluster (MSC) 的最大区别在于它几乎能自动完成所有事情，只要告诉MongoDB要分配数据，它就能自动维护数据在不同服务器之间的均衡。

5.1.1 MongoDB分片介绍

5.1.1.1 分片的目的

高数据量和吞吐量的数据库应用会对单机的性能造成较大压力,大的查询量会将单机的CPU耗尽,大的数据量对单机的存储压力较大,最终会耗尽系统的内存而将压力转移到磁盘IO上。

为了解决这些问题,有两个基本的方法: 垂直扩展和水平扩展。

- 垂直扩展：增加更多的CPU和存储资源来扩展容量。
- 水平扩展：将数据集分布在多个服务器上。水平扩展即分片

分片的缺点

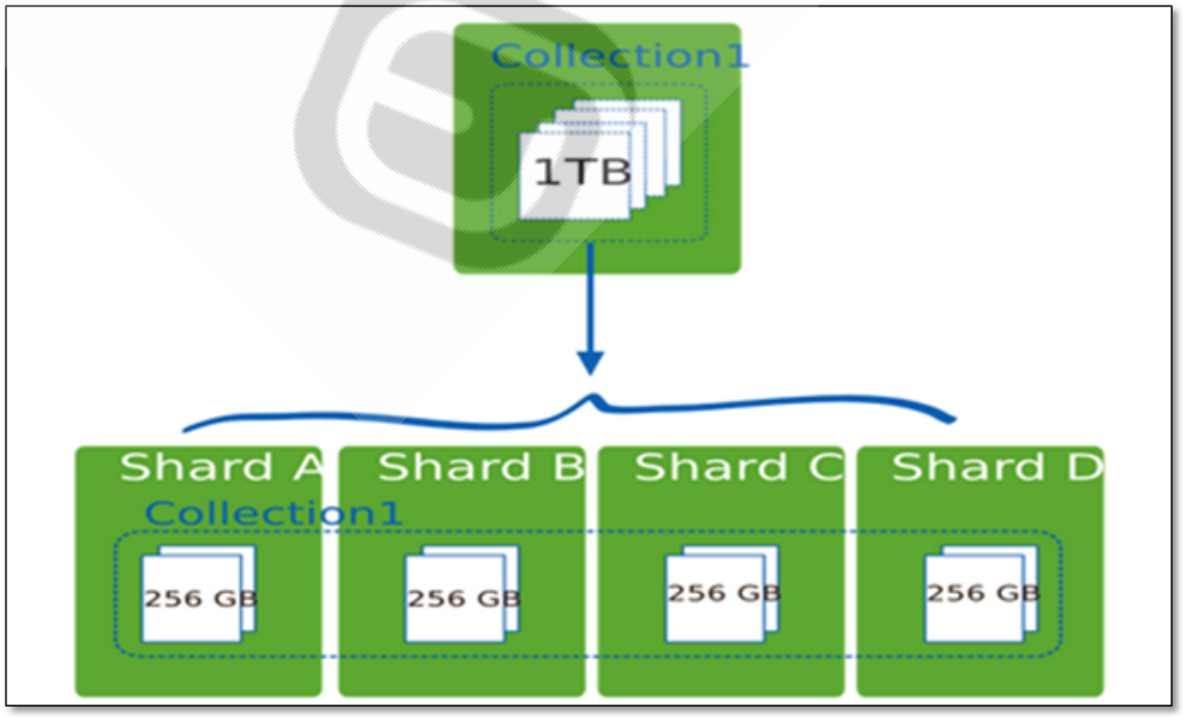
- 分片额外消耗较多,约相当于单机性能的70%左右
- 管理复杂，尽量不要分片

5.1.1.2 分片设计思想

分片为应对高吞吐量与大数据量提供了方法。使用分片减少了每个分片需要处理的请求数，因此，通过水平扩展，集群可以提高自己的存储容量和吞吐量。举例来说，当插入一条数据时，应用只需要访问存储这条数据的分片.使用分片减少了每个分片存储的数据。

MongoDB 最多可以支持1024个分片节点

例如，如果数据库1tb的数据集，并有4个分片，然后每个分片可能仅持有256 GB的数据。如果有40个分片，那么每个切分可能只有25GB的数据。



5.1.1.3 分片机制提供了如下三种优势

保证集群总是可读写

MongoDB通过多种途径来确保集群的可用性和可靠性。将MongoDB的分片和复制功能结合使用，在确保数据分片到多台服务器的同时，也确保了每分数据都有相应的备份，这样就可以确保有服务器换掉时，其他的从库可以立即接替坏掉的部分继续工作。

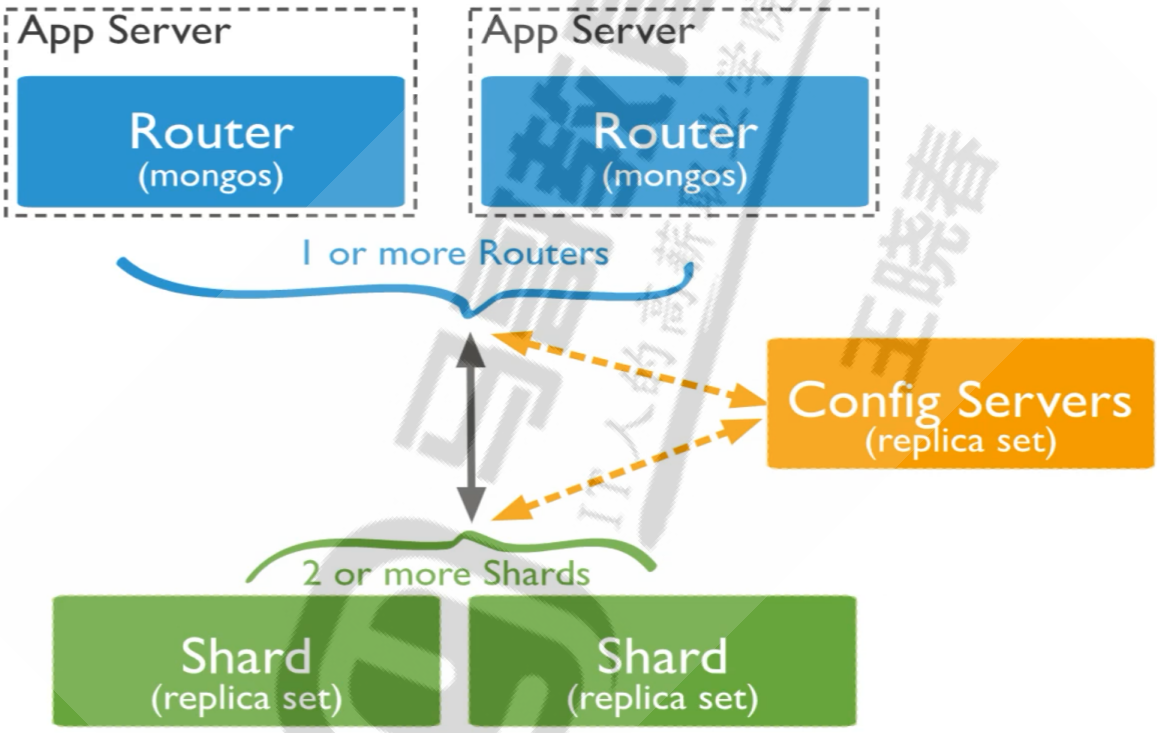
使集群易于扩展

当系统需要更多的空间和资源的时候，MongoDB使我们可以按需方便的扩充系统容量。

对集群进行抽象，让集群“不可见”

MongoDB自带了一个叫做mongos的专有路由进程。mongos就是掌握统一路口的路由器，其会将客户端发来的请求准确无误的路由到集群中的一个或者一组服务器上，同时会把接收到的响应拼装起来发回到客户端。

5.1.1.4 分片集群架构



分片集群的结构

组件	说明
Mongos	提供集群统一的对外应用访问入口，转发应用请求,选择合适的数据节点进行读写,合并多个数据节点的返回数据,应用的所有操作均通过mongos执行。数据路由，和客户端打交道的模块。mongos本身没有任何数据，它也不知道该怎么处理数据，用于寻找config server,是无状态节点,一般至少有2个mongos节点。
Config Server	存储集群所有节点、分片数据路由信息。所有存、取数据的方式，所有shard节点的信息，分片功能的一些配置信息。可以理解为真实数据的元数据。默认需要配置3个Config Server节点。
Shard	真正的数据存储位置。以复制集为单位,横向扩展,最大1024个分片,分片之间数据不重复,所有分片在一起才能可以完整的工作,以chunk为单位存数据。

Mongos本身并不持久化数据，Sharded cluster所有的元数据都会存储到 Config Server，而用户的数据分散存储到各个shard。Mongos启动后，会从配置服务器加载元数据，开始提供服务，将用户的请求正确路由到对应的shard。

Mongos 的路由功能

当数据写入时，MongoDB Cluster根据分片键设计写入数据。

当外部语句发起数据查询时，MongoDB根据数据分布自动路由至指定节点返回数据。

三种角色的资源配置

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。

注意: 实际测试是最好的检验，来看你的资源配置是否完备。

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，基于以下原因则开始考虑扩展

- 扩展需要新的资源，申请新资源需要时间,数据量大的话,可能会长达几天时间
- 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。

5.1.1.5 分片集群的相关概念

以下各种概念由小到大

- 片键shard key:文档中的一个字段
- 文档doc:包含shard key的一行数据
- 块Chunk :包含n个文档,默认64M
- 分片Shard:包含n个chunk,一个分片对应一个复制集
- 集群Cluster:包含n个分片

5.1.2 集群中数据分布

5.1.2.1 Chunk 介绍

在一个shard server内部，MongoDB还是会把数据分为chunks，每个chunk代表这个shard server内部一部分数据。chunk的产生，会有以下两个用途：

Splitting：当一个chunk的大小超过配置中的chunk size时，MongoDB的后台进程会把这个chunk切分成更小的chunk，从而避免chunk过大的情况

Balancing：在MongoDB中，balancer是一个后台进程，负责chunk的迁移，从而均衡各个shard server的负载，系统初始1个chunk，chunk size默认值64M,生产库上选择适合业务的chunk size是最好的。ongoDB会自动拆分和迁移chunks。

分片集群的数据分布（shard节点）

（1）使用chunk来存储数据

（2）进群搭建完成之后，默认开启一个chunk，大小是64M

（3）存储需求超过64M，chunk会进行分裂，如果单位时间存储需求很大，设置更大的chunk

（4）chunk会被自动均衡迁移

5.1.2.2 chunksize 的设置

适合业务的chunksize是最好的

chunk的分裂和迁移非常消耗IO资源

chunk分裂的时机：在插入和更新，读数据不会分裂

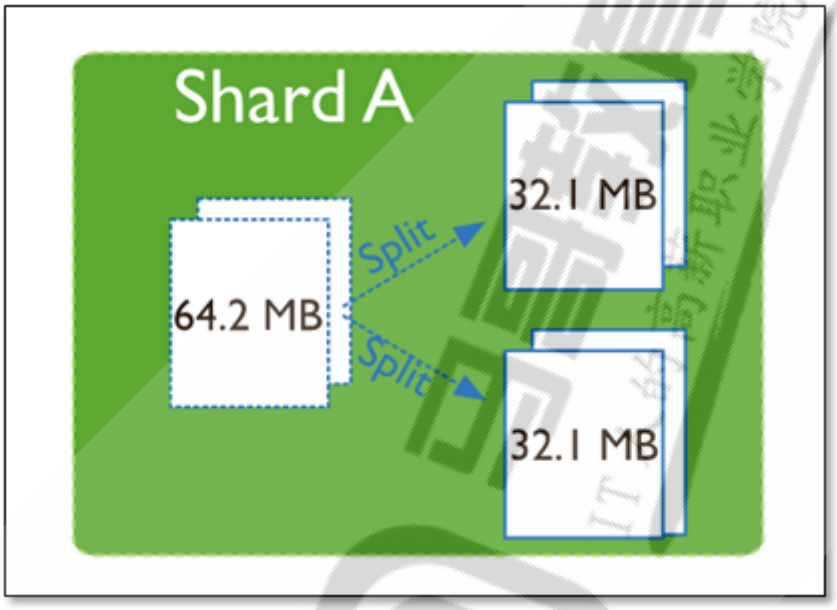
chunksize的大小设置：

小的chunksize：数据均衡是迁移速度快，数据分布更均匀。数据分裂频繁，路由节点消耗更多资源。

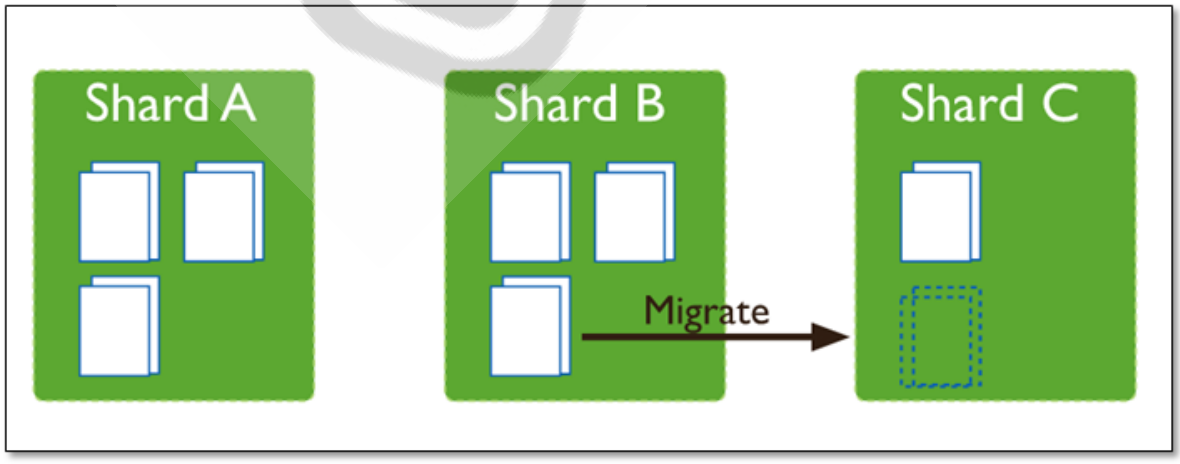
大的chunksize：数据分裂少。数据块移动集中消耗IO资源。通常100-200M

5.1.2.3 chunk 分裂及迁移

随着数据的增长，其中的数据大小超过了配置的 chunk size，默认是64M，则这个chunk就会分裂成两个。数据的增长会让chunk分裂得越来越多。



这时候，各个shard 上的chunk数量就会不平衡。mongos中的一个组件balancer 就会执行自动平衡。把chunk从chunk数量最多的shard节点挪动到数量最少的节点。



chunkSize 对分裂及迁移的影响

MongoDB 默认的 chunksize 为64MB，如无特殊需求，建议保持默认值；chunksize 会直接影响到 chunk 分裂、迁移的行为。

chunksize 越小，chunk 分裂及迁移越多，数据分布越均衡；反之，chunksize 越大，chunk 分裂及迁移会更少，但可能导致数据分布不均。

chunksize 太小，容易出现 jumbo chunk（即shardKey 的某个取值出现频率很高，这些文档只能放到一个 chunk 里，无法再分裂）而无法迁移

chunksize 越大，则可能出现 chunk 内文档数太多（chunk 内文档数不能超过 250000 ）而无法迁移。

chunk 自动分裂只会在数据写入时触发，所以如果将 chunksize 改小，系统需要一定的时间来将 chunk 分裂到指定的大小。

chunk 只会分裂，不会合并，所以即使将 chunksize 改大，现有的 chunk 数量不会减少，但 chunk 大小会随着写入不断增长，直到达到目标大小。

5.1.3 数据分片

5.1.3.1 分片键 shard key

MongoDB中数据的分片是以集合为基本单位的，集合中的数据通过片键（Shard key）被分成多部分。其实片键就是在集合中选一个键，用该键的值作为数据拆分的依据。

所以一个好的片键对分片至关重要。片键必须是一个索引，通过sh.shardCollection加会自动创建索引（前提是此集合不存在的情况下）。一个自增的片键对写入和数据均匀分布就不是很好，因为自增的片键总会在一个分片上写入，后续达到某个阀值可能会写到别的分片。但是按照片键查询会非常高效。

随机片键对数据的均匀分布效果很好。注意尽量避免在多个分片上进行查询。在所有分片上查询，mongos会对结果进行归并排序。

对集合进行分片时，需要选择一个片键，片键是每条记录都必须包含的，且建立了索引的单个字段或复合字段，MongoDB按照片键将数据划分到不同的数据块中，并将数据块均衡地分布到所有分片中。

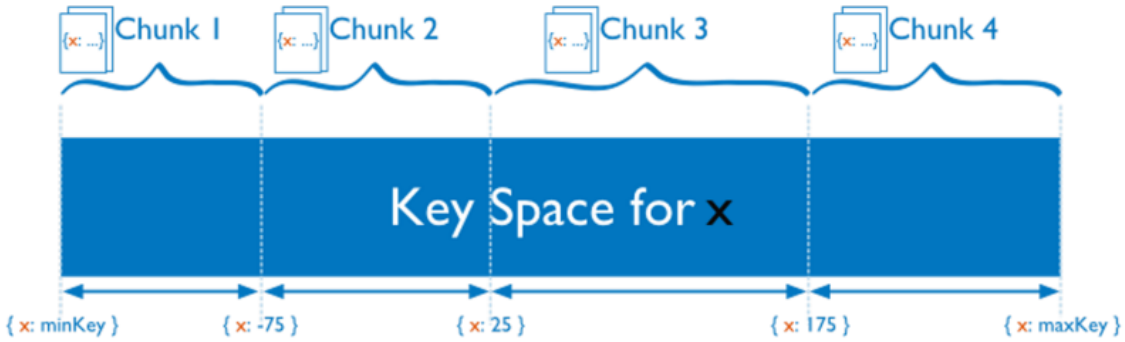
为了按照片键划分数据块，MongoDB使用基于范围的分片方式或者基于哈希的分片方式。

注意:

- 分片键是不可变。
- 分片键必须有索引。
- 分片键大小限制512bytes。
- 分片键用于路由查询。
- 不接受已进行collection级分片的collection上插入无分片键的文档,也不支持空值插入

5.1.3.2 以范围为基础的分片Sharded Cluster

Sharded Cluster支持将单个集合的数据分散存储在多shard上，用户可以指定根据集合内文档的某个字段即shard key来进行范围分片（range sharding）



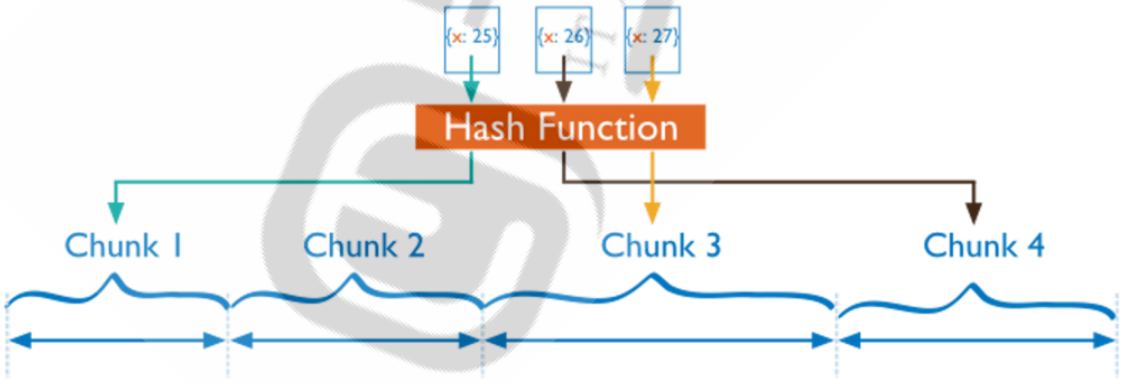
Pros	Cons
片键范围查询性能好	数据分布可能不均匀
优化读	容易有热点

对于基于范围的分片，MongoDB按照片键的范围把数据分成不同部分。

假设有一个数字的片键:想象一个从负无穷到正无穷的直线，每一个片键的值都在直线上画了一个点。MongoDB把这条直线划分为更短的不重叠的片段，并称之为数据块，每个数据块包含了片键在一定范围内的数据。在使用片键做范围划分的系统中，拥有"相近"片键的文档很可能存储在同一个数据块中，因此也会存储在同一个分片中。

5.1.3.3 基于哈希的分片

分片过程中利用哈希索引作为分片的单个键，且哈希分片的片键只能使用一个字段，而基于哈希片键最大的好处就是保证数据在各个节点分布基本均匀。



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	

对于基于哈希的分片，MongoDB计算一个字段的哈希值，并用这个哈希值来创建数据块。在使用基于哈希分片的系统中，拥有"相近"片键的文档很可能不会存储在同一个数据块中，因此数据的分离性更好一些。

Hash分片与范围分片互补，能将文档随机的分散到各个chunk，充分的扩展写能力，弥补了范围分片的不足，但不能高效的服务范围查询，所有的范围查询要分发到后端所有的Shard才能找出满足条件的文档。

5.1.3.4 分片键选择建议

1、递增的sharding key

数据文件挪动小。（优势）

因为数据文件递增，所以会把insert的写IO永久放在最后一片上，造成最后一片的写热点。同时，随着最后一片的数据量增大，将不断的发生迁移至之前的片上。

2、随机的sharding key

数据分布均匀，insert的写IO均匀分布在多个片上。（优势）

大量的随机IO，磁盘不堪重荷。

3、混合型key

大方向随机递增，小范围随机分布。

为了防止出现大量的chunk均衡迁移，可能造成的IO压力。我们需要设置合理分片使用策略（片键的选择、分片算法（range、hash））

选择合适的分片键建议

- 选择基数大的片键,如:以年龄18-60作为片键,最多43个chunk

  如果基数小的片键,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

  太大的块，会导致水平扩展时移动块会非常困难
- 片键取值的分布应尽可能均匀,如果使用分布不均匀的片键,会导致以下

  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
- 对主要查询使用片键做为条件查询,可以直接定位具体分片,而非所有分片响应,提高效率
- 主要从基数,写分布,定向查询三个方面考虑是否是合适的分片键

比如: 一个email系统中使用片键:{user_id:1,time:1}, 三个方面基数大,写分布,定向查询都不错

5.1.3.5 合理的分片

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。

范例: 合理分片

```
A=所需存储总量/单服务器可挂载容量
如:8TB/2TB = 4
B=工作集大小/(单服务器物理内存容量*0.6) #0.6表示mongodb默认使用内存的60%
如:400GB / (256G*0.6) = 3
C=并发量总数/（单服务器并发量*0.7) #0.7表示单机变成分片有额外开销
如:30000/(9000*0.7)= 6

#结果:
分片数量=max(A,B,C)=6
```

5.2 部署分片集群

官方说明

```
https://docs.mongodb.com/manual/tutorial/deploy-shard-cluster/
```

5.2.1 分片集群环境规划

以下案例准备10个实例,使用端口号:38017-38026

```
(1)mongos节点:
38017

(2)config server节点: 38018-38020
3台构成的复制集（1主两从，不支持arbiter)
38018-38020（复制集名字configsvr)

(3)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)
```

5.2.2 shard 节点配置过程

5.2.2.1 shard 节点的目录创建

```
#在shard节点创建三个目录
mkdir -p /mongodb/{38021..38026}/{conf,data,log}
```

5.2.2.2 shard 节点的配置文件

```
#第一组复制集搭建:21-23 (1主1从1Arb)
cat > /mongodb/38021/conf/mongodb.conf<<EOF
systemLog:
  destination: file
  path: /mongodb/38021/log/mongodb.log
  logAppend: true
storage:
  dbPath: /mongodb/38021/data
net:
  bindIp: 0.0.0.0
  port: 38021
replication :
  replSetName: shard1
sharding:
```

```
  clusterRole: shardsvr
processManagement:
  fork: true
EOF

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


#第二组节点:24-26(1主1从1Arb)
cat > /mongodb/38024/conf/mongodb.conf <<EOF
systemLog:
  destination: file
  path: /mongodb/38024/log/mongodb.log
  logAppend: true
storage:
  dbPath: /mongodb/38024/data
net:
  bindIp : 0.0.0.0
  port: 38024
replication:
  replSetName: shard2
sharding:
  clusterRole: shardsvr
processManagement:
  fork: true
EOF

sed 's#38024#38025#g' /mongodb/38024/conf/mongodb.conf  >
/mongodb/38025/conf/mongodb.conf
sed 's#38024#38026#g' /mongodb/38024/conf/mongodb.conf  >
/mongodb/38026/conf/mongodb.conf
```

5.2.2.3 启动所有shard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: 'shard1', members: [
                 {_id: 0, host: '10.0.0.100:38021'},
                 {_id: 1, host: '10.0.0.100:38022'},
                 {_id: 2, host: '10.0.0.100:38023', "arbiterOnly": true}]
         }

rs.initiate (config)

mongo --port 38024
```

```
use admin

config = { _id: 'shard2', members: [
                 {_id: 0, host: '10.0.0.100:38024'} ,
                 {_id: 1, host: '10.0.0.100:38025'},
                 {_id: 2, host: '10.0.0.100:38026', "arbiterOnly": true} ]
         }
rs.initiate (config)
```

5.2.3 config节点配置

5.2.3.1 config节点的目录创建

```
mkdir -p /mongodb/{38018..38020}/{conf,data,log}
```

5.2.3.2 config节点的配置文件

```
cat > /mongodb/38018/conf/mongodb.conf <<EOF
systemLog:
  destination: file
  path: /mongodb/38018/log/mongodb.conf
  logAppend: true
storage:
  dbPath: /mongodb/38018/data
net:
  bindIp: 0.0.0.0
  port: 38018              #如果不指定此行,config节点默认使用27019
replication:
  replSetName: config
sharding:
  clusterRole: configsvr  #指定角色
processManagement:
  fork: true
EOF


sed 's#38018#38019#g' /mongodb/38018/conf/mongodb.conf >
/mongodb/38019/conf/mongodb.conf
sed 's#38018#38020#g' /mongodb/38018/conf/mongodb.conf >
/mongodb/38020/conf/mongodb.conf
```

5.2.3.3 启动节点并配置config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: '10.0.0.100:38018' },
                  {_id: 1, host: '10.0.0.100:38019'} ,
                  {_id: 2, host: '10.0.0.100:38020'}]
        }
```

```
rs.initiate (config)

#注意:
mongodb 3.4之前 config server 可以是一个独立节点，官方建议复制集
mongodb 3.4之后，要求config server 必须为replica set，但是不支持arbiter
```

5.2.4 mongos 节点配置

5.2.4.1 mongos 节点的目录创建

```
#注意mongos节点无数据目录
mkdir -p /mongodb/38017/{conf,log}
```

5.2.4.2 mongos 节点的配置文件

```
cat > /mongodb/38017/conf/mongos.conf <<EOF
systemLog:
  destination: file
  path: /mongodb/38017/log/mongos.log
  logAppend: true
net:
  bindIp: 0.0.0.0
  port: 38017                #不加此行,默认监听27017
sharding:
  configDB: config/10.0.0.100:38018,10.0.0.100:38019,10.0.0.100:38020
processManagement:
  fork: true
EOF
```

5.2.4.3 启动 mongos

注意: 使用专门的mongos程序,不是mongod服务程序

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

5.2.4.4 分片集群添加节点

片集群添加节点,连接到其中一个mongos (10.0.0.100) ， 做以下配置

```
1)连接到mongos的admin数据库
# su - mongod
$mongo 10.0.0.100:38017/admin

2)添加分片
mongos>sh.addShard("shard1/10.0.0.100:38021,10.0.0.100:38022,10.0.0.100:38023")
mongos>sh.addShard("shard2/10.0.0.100:38024,10.0.0.100:38025,10.0.0.100:38026")
#或者下面
mongos>db.runCommand({addshard:"shard1/10.0.0.100:38021,10.0.0.100:38022,10.0.0.
100:38023", name : "shard1"}
mongos>db.runCommand({addshard:"shard2/10.0.0.100:38024,10.0.0.100:38025,10.0.0.
100:38026", name : "shard2")

3)列出分片
mongos>db.adminCommand({listShards: 1})
```

```
#或者
use admin
mongos>db.runCommand({listshards: 1})

4)整体状态查看
mongos>sh.status()
```

5.2.5 配置分片规则

5.2.5.1 Range 分片配置及测试

```
1)连接到mongos的admin数据库激活数据库分片功能
mongo --port 38017 admin
mongos>sh.enableSharding("数据库名称")
#或者
mongos>db.runCommand ({ enablesharding: "数据库名称" })

#示例:对test库启用分片功能
mongos>sh.enableSharding("test")
#或者
mongos>db.runCommand ({enablesharding: "test" })

2)指定分片键对集合分片
#创建索引
mongos>use test
mongos>db.vast.createIndex({id: 1}) #针对id从小到大创建索引

#开启range分片
mongos>use admin
mongos>sh.shardCollection("test.vast", {id: 1})
#或者
mongos>db.runCommand({shardcollection: "test.vast",key: {id: 1}})

#如果直接对_id进行分片,无需创建索引
mongos>sh.shardCollection("test.vast", {_id: 1})


3)插入大量数据集合分片验证
mongos> use test
mongos> for (i=0;i<1000000 ;i++){ db.vast.insert ( { "id" :i, "name " : "wang" ,
"age" : 20,"date" : new Date()})}
mongos> db.vast.stats ()

4)分片结果测试shard1
mongo --port 38021
db.vast.count()
```

5.2.5.2 Hash分片配置及测试

范例： 对magedu库下的vast大表进行hash创建哈希索引

```
1）连接到mongos的admin数据库对于magedu数据库开启分片功能
mongo --port 38017 admin
mongos>sh.enableSharding("magedu")
mongos>db.runCommand ({ enablesharding : "magedu"})
```

```
2）对于magedu库下的vast表建立hash索引
use magedu
mongos> db.vast.createIndex( { id: "hashed"} )

3）在vast表的id列开启hash分片
mongos>use admin
mongos>sh.shardCollection ( "magedu.vast", { id:"hashed"})
#或者
mongos>db.runCommand({shardcollection: "test.vast",key: {id: "hashed"}})

4）插入足够的大量数据才会分片
mongos>use magedu
mongos>for(i=0;i<100000;i++){db.vast.insert ( { "id": i, "name" : "wang" , "age"
:20 , "date": new Date()})}

5)hash分片结果测试
分别统计shard节点的行数
mongo --port 38021
use magedu
db.vast.count()
mongo --port 38024
use magedu
db.vast.count()
```

5.3 分片集群的查询及管理

范例: 查看分片信息

```
#判断是否是shard集群
mongo --port 38017 admin
mongos>db.runCommand({isdbgrid: 1})

#列出所有分片信息
mongos>db.runCommand({listshards:1})

#列出开启分片的数据库,列出所有数据库分片情况
mongos>use config
mongos>db.databases.find({"partitioned": true})
或者:
mongos>db.databases.find()
{ "_id" : "test", "primary" : "shard1", "partitioned" : true, "version" : {
"uuid" : UUID("ba30276f-0067-4fe9-8259-64d8ba9bd64d"), "timestamp" :
Timestamp(1637473659, 2), "lastMod" : 1 } }
{ "_id" : "magedu", "primary" : "shard2", "partitioned" : true, "version" : {
"uuid" : UUID("e20bad67-986c-4b80-ad3d-3a6b71e57f57"), "timestamp" :
Timestamp(1637475186, 219), "lastMod" : 1 } }


#查看分片的片键
mongos>db.collections.find().pretty()

#查看分片的详细信息,涵盖上面命令内容的汇总命令
mongos>sh.status()

#查看chunk大小
mongos>use config
mongos>db.settings.find()
```

```
#修改clunk大小为4M
https://docs.mongodb.com/manual/tutorial/modify-chunk-size-in-sharded-cluster/
mongos>db.settings.insertOne( { _id:"chunksize", value: 4 } )
或者
mongos>db.settings.save({_id:'chunksize',values:4})
mongos>db.settings.find()

#手动预先分片,可以在添加数据时会自动预先分配到chunk,避免后续的chunk移动,减少网络IO
#指定划分chunk的范围区间,id=20000,40000,60000,80000为划分chunk的分界点
mongos>sh.splitAt('test.vast',{id:20000})
mongos>sh.splitAt('test.vast',{id:40000})
mongos>sh.splitAt('test.vast',{id:60000})
mongos>sh.splitAt('test.vast',{id:80000})
```

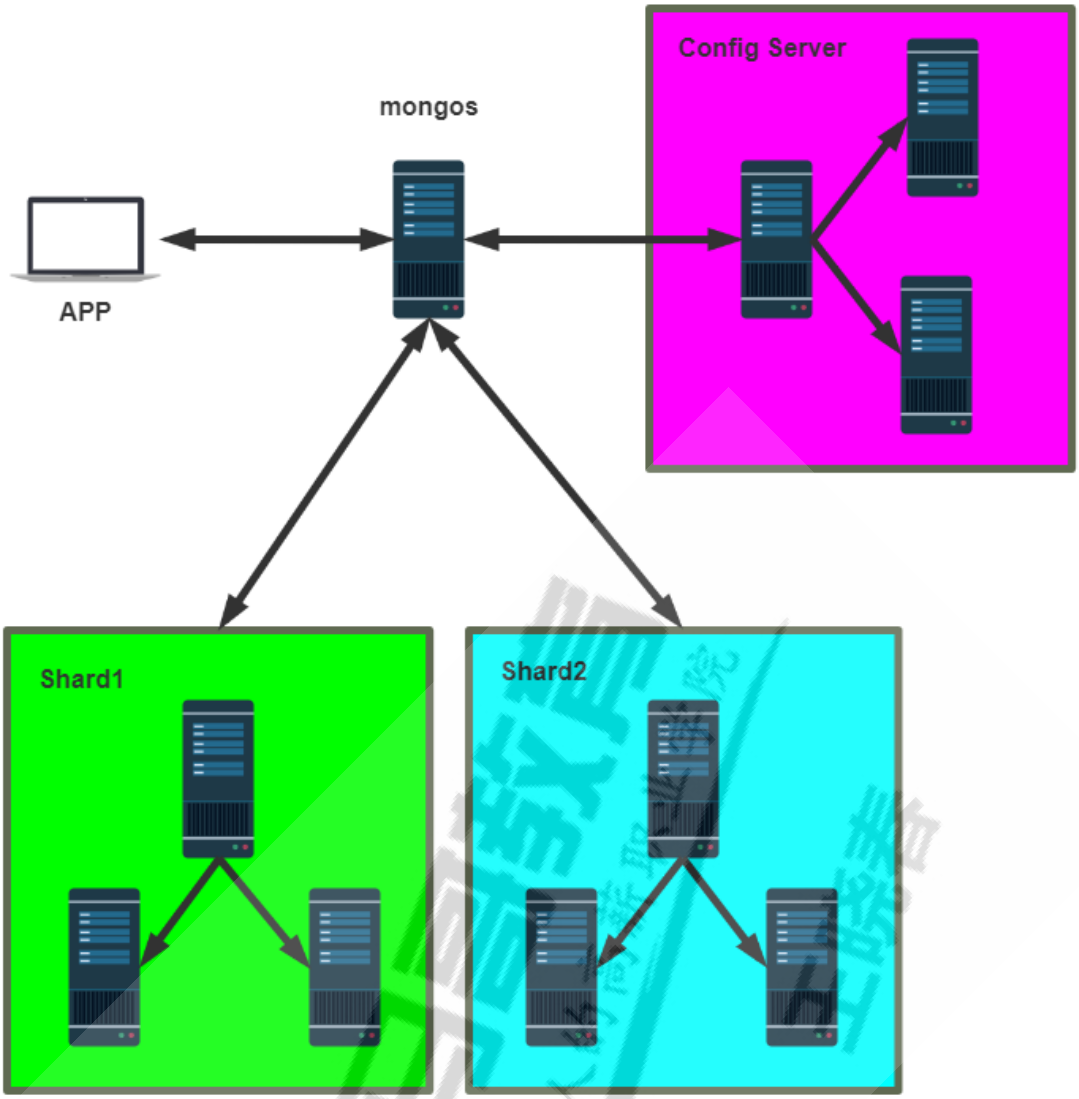
范例: 删除分片

```
#删除分片节点（谨慎)
1)确认balancer是否在工作
mongos>sh.getBalancerState()

2)删除shard2节点(谨慎),此步时间会花很多时间可能会长达几天,选择空闲时间
mongos>use admin
mongos>db.runCommand({removeShard: "shard2"})

#注意:删除操作一定会立即触发blancer
```

范例: 添加分片节点

```
#搭建复制集shard3
过程略

#添加分片
mongos>sh.addShard("shard3/10.0.0.100:38027,10.0.0.100:38028,10.0.0.100:38029")
#或者
mongos>db.runCommand({addshard:
"shard3/10.0.0.100:38027,10.0.0.100:38028,10.0.0.100:38029" ,name : "shard3"} )
```

5.4 实战案例: 多服务器实现MongoDB分片集群



5.4.1 环境规划

```
准备十台主机,角色分配如下
shard1
10.0.0.8:27017
10.0.0.18:27017
10.0.0.28:27017
shard2
10.0.0.38:27017
10.0.0.48:27017
10.0.0.58:27017
config server
10.0.0.68:27019
10.0.0.78:27019
10.0.0.88:27019
mongos
10.0.0.98:27017
```

5.4.2 实现过程

5.4.2.1 配置 shard1

```
#在shard1的三个节点上执行下面操作
[root@rocky8 ~]#cat > /mongodb/conf/mongo.conf <<EOF
systemLog:
  destination: file
  path: "/mongodb/log/mongodb.log"
  logAppend: true
storage:
  dbPath: "/mongodb/data/"
  journal:
    enabled: true
processManagement:
  fork: true
net:
  port: 27017
  bindIp: 0.0.0.0
replication :
  oplogSizeMB: 2048
  replSetName: shard1
sharding:
  clusterRole: shardsvr
EOF

[root@rocky8 ~]#systemctl restart mongod.service

#在10.0.0.8节点执行下面操作
[root@rocky8 ~]#mongo
use admin
config = { _id: 'shard1', members: [
               {_id: 0, host: '10.0.0.8:27017'},
               {_id: 1, host: '10.0.0.18:27017'},
               {_id: 2, host: '10.0.0.28:27017'}]
         }

rs.initiate (config)
```

5.4.2.2 配置 shard2

```
#在shard2的三个节点上执行下面操作
[root@rocky8 ~]#cat > /mongodb/conf/mongo.conf <<EOF
systemLog:
  destination: file
  path: "/mongodb/log/mongodb.log"
  logAppend: true
storage:
  dbPath: "/mongodb/data/"
  journal:
    enabled: true
processManagement:
  fork: true
net:
  port: 27017
  bindIp: 0.0.0.0
```

```
replication :
  oplogSizeMB: 2048
  replSetName: shard2
sharding:
  clusterRole: shardsvr
EOF

[root@rocky8 ~]#systemctl restart mongod.service

#在10.0.0.38节点执行下面操作
[root@rocky8 ~]#mongo
use admin
config = { _id: 'shard2', members: [
                 {_id: 0, host: '10.0.0.38:27017'},
                 {_id: 1, host: '10.0.0.48:27017'},
                 {_id: 2, host: '10.0.0.58:27017'}]
         }

rs.initiate (config)
```

5.4.2.3 配置 config server

```
#在config的三个节点上执行下面操作
[root@rocky8 ~]#cat > /mongodb/conf/mongo.conf <<EOF
systemLog:
  destination: file
  path: "/mongodb/log/mongodb.log"
  logAppend: true
storage:
  dbPath: "/mongodb/data/"
  journal:
    enabled: true
processManagement:
  fork: true
net:
  bindIp: 0.0.0.0
replication:
  oplogSizeMB: 2048
  replSetName: config
sharding:
  clusterRole: configsvr
EOF

[root@rocky8 ~]#systemctl restart mongod.service

#在10.0.0.68节点执行下面操作,注意:config server的端口默认是27019
[root@rocky8 ~]#mongo --port 27019
use admin
config = { _id: 'config', members: [
                  {_id: 0, host: '10.0.0.68:27019' },
                  {_id: 1, host: '10.0.0.78:27019'} ,
                  {_id: 2, host: '10.0.0.88:27019'}]
         }


rs.initiate (config)
```

5.4.2.4 配置 mongos

```
#配置mongs节点
#在mongos节点10.0.0.98节点执行下面操作
[root@rocky8 ~]#cat > /mongodb/conf/mongo.conf <<EOF
systemLog:
  destination: file
  path: "/mongodb/log/mongodb.log"
sharding:
  configDB: config/10.0.0.68:27019,10.0.0.78:27019,10.0.0.88:27019
net:
  bindIp: 0.0.0.0
processManagement:
  fork: true
EOF

#先停止原mongod服务
[root@rocky8 ~]#systemctl stop mongod

#启动mongs,注意不是mongod
[root@rocky8 ~]#mongos -f /mongodb/conf/mongo.conf

[root@rocky8 ~]#ss -ntlp|grep mongos
LISTEN 0      128          0.0.0.0:27017      0.0.0.0:*    users:
(("mongos",pid=1928,fd=13))

#添加分片
[root@rocky8 ~]#mongo admin
mongos>sh.addShard("shard1/10.0.0.8:27017")
mongos>sh.addShard("shard2/10.0.0.38:27017")
#或者
mongos>sh.addShard("shard1/10.0.0.8:27017,10.0.0.18:27017,10.0.0.28:27017")
mongos>sh.addShard("shard2/10.0.0.38:27017,10.0.0.48:27017,10.0.0.58:27017")

#查看分片状态
mongos> sh.status()
--- Sharding Status ---
  sharding version: {
    "_id" : 1,
    "minCompatibleVersion" : 5,
    "currentVersion" : 6,
    "clusterId" : ObjectId("61951Oec03c01fb9bc4a22aa")
  }
  shards:
        {  "_id" : "shard1",  "host" : "shard1/10.0.0.18:27017,10.0.0.8:27017",
 "state" : 1,  "topologyTime" : Timestamp(1637160656, 5) }
        {  "_id" : "shard2",  "host" : "shard2/10.0.0.38:27017,10.0.0.48:27017",
 "state" : 1,  "topologyTime" : Timestamp(1637160679, 5) }
  active mongoses:
        "5.0.3" : 1
  autosplit:
        Currently enabled: yes
  balancer:
        Currently enabled: yes
        Currently running: no
        Failed balancer rounds in last 5 attempts: 0
        Migration results for the last 24 hours:
```

```
                No recent migrations
  databases:
        {  "_id" : "config",  "primary" : "config",  "partitioned" : true }
mongos> db.runCommand({listshards: 1})
{
    "shards" : [
        {
            "_id" : "shard1",
            "host" : "shard1/10.0.0.18:27017,10.0.0.8:27017",
            "state" : 1,
            "topologyTime" : Timestamp(1637160656, 5)
        },
        {
            "_id" : "shard2",
            "host" : "shard2/10.0.0.38:27017,10.0.0.48:27017",
            "state" : 1,
            "topologyTime" : Timestamp(1637160679, 5)
        }
    ],
    "ok" : 1,
    "$clusterTime" : {
        "clusterTime" : Timestamp(1637160731, 1),
        "signature" : {
            "hash" : BinData(0,"AAAAAAAAAAAAAAAAAAAAAAAAAAA="),
            "keyId" : NumberLong(0)
        }
    },
    "operationTime" : Timestamp(1637160731, 1)
}
```

5.4.2.5 配置分片规则

5.4.2.5.1 启用 Range 分片规则

```
#启用数据库的分片
mongos> sh.enableSharding("test")

#创建索引
mongos>use test
mongos>db.vast.createIndex({id: 1}) #针对id字段从小到大创建索引

#开启range分片规则
mongos>sh.shardCollection("test.vast", {id: 1})
#或者
mongos>db.runCommand({shardcollection: "test.vast",key: {id: 1}})

#如果是对_id进行range分片,无需创建索引
mongos>sh.shardCollection("test.vast", {_id: 1})
#或者
mongos>db.runCommand({shardcollection: "test.vast",key: {_id: 1}})

#插入足够的大量数据才会分片
mongos> use test
mongos> for (i=0;i<1000000 ;i++){ db.vast.insert ( { "id" :i, "name" : "wang" ,
"age" : 20,"date" : new Date()})}
```

5.4.2.5.2 启用 Hash 分片规则

```
#启用数据库的分片
mongos> sh.enableSharding("magedu")

#指定在magedu库的vast表的id列上创建hash索引
mongos> use magedu
mongos> db.vast.createIndex({id: "hashed"} )

#指定表分片基于hash规则
mongos> sh.shardCollection("magedu.vast", {id: 'hashed'})
#或者
mongos>db.runCommand({shardcollection: "magedu.vast",key: {id: 'hashed'}})

#插入足够的大量数据才会分片
mongos> use magedu
mongos> for (i=0;i<1000000 ;i++){ db.vast.insert ( { "id" :i, "name" : "wang" ,
"age" : 20,"date" : new Date()})}

#查看分片
mongos> db.adminCommand({listShards: 1})
mongos> db.runCommand({listshards: 1})
```

5.4.2.6 验证分片集群

```
#查看分片
mongos> db.adminCommand({listShards: 1})
mongos> db.runCommand ({listshards: 1})
mongos> sh.status()

mongos> use config
mongos> db.databases.find()

#分别在shard1和shard2上验证数据数量
use test
db.vast.count()

use magedu
db.vast.count()
```

5.5 balance 操作

5.5.1 介绍

mongos的一个重要功能balancer，自动巡查所有shard节点上的chunk的情况，自动做chunk迁移,即实现balance功能

balancer工作时机

- 自动运行，当检测系统不繁忙的时候做迁移
- 在做节点删除的时候，立即开始迁移工作
- balancer可以指定只在预设定的时间窗口内运行

有时候需要关闭和开启balancer

比如:备份的时候进行balance会导致数据不一致,所以需要错开备份时间段

范例: 停止和开启balancer

```
mongos> sh.stopBalancer()
mongos> sh.startBalancer()
```

5.5.2 自定义自动平衡进行的时间段

```
#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
mongo --port 38017 admin
use config
sh.setBalancerState(true)
sh.getBalancerState()


#设置3:00到5:00进行balance
db.settings.update({id: "balancer"},{$set: {activeWindow:{start: "3:00",stop :
"5:00"}}}, true)

sh.getBalancerWindow()
sh.status()

db.settings.find()
{ "_id" : "chunksize", "value" : 4 }
{ "_id" : ObjectId("619a0c8d251b017dc44bfde2"), "id" : "balancer",
"activeWindow" : { "start" : "3:00", "stop" : "5:00" } }
{ "_id" : "balancer", "mode" : "full", "stopped" : false }
{ "_id" : "autosplit", "enabled" : true }
```

5.5.3 集合的balance管理

```
#关闭某个集合的balancer
sh.disableBalancing("test.vast")

#打开某个集合的balancer
sh.enableBalancing ("test.vast")

#确定某个集合的balance是开启或者关闭
db.getSiblingDB("config").collections.findOne({_id : "magedu.vast"}).noBalance
```

6 MongoDB 备份恢复

虽然有复制集的高可用性保证，数据丢失的可能很低，但仍然需要备份

备份的目的:

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

6.1 备份恢复工具介绍

6.1.1 mongoexport/mongoimport说明

mongoexport/mongoimport可以实现逻辑备份,类似于mysqldump,可以导出json或csv格式文件

应用场景总结

- 异构平台迁移,比如:MySQL和MongoDB
- 同平台跨大版本的MongoDB数据导出导入: 比如mongodb 2 ----> mongodb 3

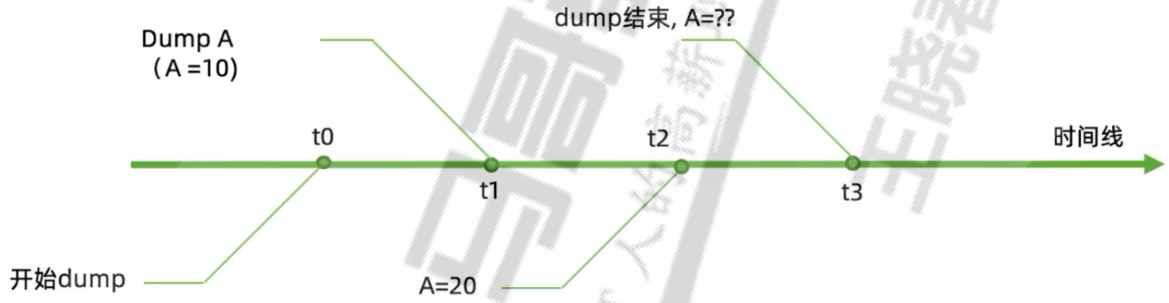
6.1.2 mongodump/mongorestore说明

物理备份,日常备份恢复时使用．导出的二进制文件

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。

使用mongodump备份最灵活，但速度上也是最慢的

mongodump出来的数据不能表示某个时间点，只是某个时间段



6.1.3 mongoexport/mongoimport与mongodump/mongorestore的对比

mongoexport/mongoimport导入/导出的是JSON格式

而mongodump/mongorestore导入/导出的是BSON格式。

JSON可读性强但体积较大，BSON则是二进制文件，体积小但对人类几乎没有可读性。

在一些mongodb版本之间，BSON格式可能会随版本不同而有所不同，所以不同版本之间用mongodump/mongorestore可能不会成功，具体要看版本之间的兼容性。

当无法使用BSON进行跨版本的数据迁移的时候，使用JSON格式即mongoexport/mongoimport是一个可选项。

跨版本的mongodump/mongorestore并不推荐，实在要做请先检查文档看两个版本是否兼容

JSON虽然具有较好的跨版本通用性，但其只保留了数据部分，不保留索引，账户等其他基础信息。

6.1.4 备份工具安装

版本说明

```
https://docs.mongodb.com/database-tools/mongoexport/#mongodb-binary-bin.mongoexport

Versioning
Starting with MongoDB 4.4, mongoexport is now released separately from the
MongoDB Server and uses its own versioning, with an initial version of 100.0.0.
Previously, mongoexport was released alongside the MongoDB Server and used
matching versioning.

For documentation on the MongoDB 4.2 or earlier versions of mongoexport,
reference the MongoDB Server Documentation for that version of the tool:
```

下载链接

```
https://www.mongodb.com/try/download/database-tools
```

6.2 mongoexport/mongoimport

6.2.1 导出工具mongoexport

此工具只能针对单表,不支持整库的表导出

官方帮助

```
https://docs.mongodb.com/database-tools/mongoexport/
```

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
--type=csv或 --csv  #指定导出为csv格式，默认json格式
--authenticationDatabase <验证库> #指定验证库
```

范例:

```
#单表备份至json格式
mongoexport -uroot -p123456 --port 27017 --authenticationDatabase admin -d
magedu -c vast -o /mongodb/vast.json
mongoexport -d test -c log -f uid,name -q='{"uid":3}' -o vast.json

#单表备份至csv格式，则需要使用--type=csv参数
mongoexport -uroot -p123456 --port 27017 --authenticationDatabase admin -d test
-c vast --type=csv -f uid,name,age,date -o /mongodb/vast.csv
```

6.2.2 导入工具mongoimport

官方帮助

```
https://docs.mongodb.com/database-tools/mongoimport/
```

范例:

```
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
--type=csv #指定导入csv格式文件，默认json格式
-j，--numInsertionworkers=<number>  #同时插入的数量,默认为
--headerline  #指明第一行是列名，不需要导入
```

范例: 数据恢复

```
#导入json格式表数据到vast1表中
mongoimport -uroot -p123456 --port 27017 --authenticationDatabase admin -d
magedu -c vast1 /mongodb/vast.json

#导入标准的csv格式(文件第一行有列名字)的文件到vast2表中
mongoimport-uroot -p123456 --port 27017 --authenticationDatabase admin -d magedu
-c vast2 --type=csv --headerline --file /mongodb/vast.csv

#导入非标准的csv格式(第一行没有列名字)的文件到vast3表中
mongoimport -uroot -p123456 --port 27017 --authenticationDatabase admin -d
magedu -c vast3 --type=csv -f id,name, age,date --file/mongodb/vast.csv
```

6.2.3 异构平台迁移案例

6.2.3.1 将MySQL数据迁移至MongoDB

范例：方法1

```
#将hellodb数据库下students表进行导出，导入到mongodb
1) mysql开启安全路径
vim /etc/my.cnf
[mysqld]
secure-file-priv=/tmp

vim /lib/systemd/system/mysqld.service
#PrivateTmp=true 将此行注释,否则会在/tmp/生成如下类似的临时的子目录
/tmp/systemd-private-4e55a87321e741b7b98959db343effa5-mysqld.service-
JsMtwj/tmp/students_info.csv

#重启数据库生效
systemctl restart mysqld

2）导出mysql的students表数据
```

```
#默认导出为Tab分隔的csv文件
mysql> select * from hellodb.students into outfile '/tmp/students.csv';
#指定导出分隔符为,的csv文件
mysql> select * from hellodb.students into outfile '/tmp/students.csv' fields
terminated by ',';

3)导出列名
#如果列很多,可以使用infomation_schema.columns导出列名
select column_name from information_schema.columns where table_schema='hellodb'
and table_name='students';
#group_concat函数可以多行合并成一行显示
select group_concat(column_name) from information_schema.columns where
table_schema='hellodb' and table_name='students';

4)在mongodb中导入备份
#方法1
将第3步查出的列名加入到前面导出的文件students.csv的第一行
mongoimport   -uroot -p123456 --port 27017 --authenticationDatabase admin -d
test -c students --type=csv --headerline --file /tmp/students.csv
#方法2:使用选项 -f 指定列名
mongoimport   -uroot -p123456 --port 27017 --authenticationDatabase admin -d
test -c students --type=csv -f StuID,Name,Age,Gender,ClassID,TeacherID --file
/tmp/students.csv
```

范例：方法2

```
1) mysql开启安全路径
vim /etc/my.cnf
[mysqld]
secure-file-priv=/data/mysql

vim /lib/systemd/system/mysqld.service
#PrivateTmp=true 将此行注释,否则会在/tmp/生成临时的子目录

#重启数据库生效
systemctl restart mysqld

mkdir -p /data/mysql
chown mysql.mysql /data/mysql

2)导出students表到目录中，生成以Tab分割的文本文件
mysqldump hellodb students  -T /data/mysql
ls /data/mysql/
students.sql  students.txt

3)将tab替换为,
sed -i.bak 's/\t/,/g' /data/mysql/students.txt

4)导入mongodb
mongoimport -d test -c students -f stuid,name,age,gender,classid,teacherid --
type=csv /data/mysql/students.txt
```

6.2.3.2 将MongoDB数据迁移至MySQL

```
[root@rocky8 ~]#head /tmp/students_info.csv
1,Shi Zhongyu,22,M,2,3
2,Shi Potian,22,M,1,7
3,Xie Yanke,53,M,2,16
4,Ding Dian,32,M,4,4
5,Yu Yutong,26,M,3,1
6,Shi Qing,46,M,5,\N
7,Xi Ren,19,F,3,\N
8,Lin Daiyu,17,F,7,\N
9,Ren Yingying,20,F,6,\N
10,Yue Lingshan,19,F,3,\N

mysql> use hellodb
Database changed
mysql> create table students_info like students;
mysql> load data infile '/tmp/students_info.csv' into table students_info fields
terminated by ',' optionally enclosed by '"';
```

6.3 mongodump和mongorestore

分片集备份大致与复制集原理相同，不过存在以下差异:

- 应分别为每个片和config备份，如果有3个分片，即需要分别进行3个分片和config共4个备份
- 分片集备份不仅要考虑一个分片内的一致性问题，还要考虑分片间的一致性问题，因此每个片要能够恢复到同一个时间点
- 如果在备份期间发生均衡的数据迁移，会造成数据丢失，所以备份前需要停止balancer

6.3.1 mongodump

官方帮助

```
https://docs.mongodb.com/database-tools/mongodump/
```

mongodump用法如下:

```
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= #并行导出的数量,默认为4
--oplog  #备份的同时备份oplog,此选项是replica set或者master/ slave模式专用
```

规律:

- 导出的文件放在以 database命名的目录下
- 每个表导出2个文件,分别是bson结构的数据文件和json格式的索引信息
- 如果不声明表名。导出所有的表

6.3.2 mongorestore

官方帮助

```
https://docs.mongodb.com/database-tools/mongorestore/
```

mongorestore与mongoimport参数类似

mongorestore用法如下:

```
mongorestore --help
-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
--authenticationDatabase  #验证数据的名称
--gzip  #备份时压缩
--oplogReplay  #恢复完数据文件后再重放oplog。默认重放dump/oplog.bson
--oplogFile=<filename>  #指定需要重放的oplog文件位置
--oplogLimit=<seconds>[:ordinal]  #重放oplog 时截止到指定时间点
--drop  #恢复的时候把之前的集合drop掉
```

6.3.3 备份和恢复范例

范例：mongodump

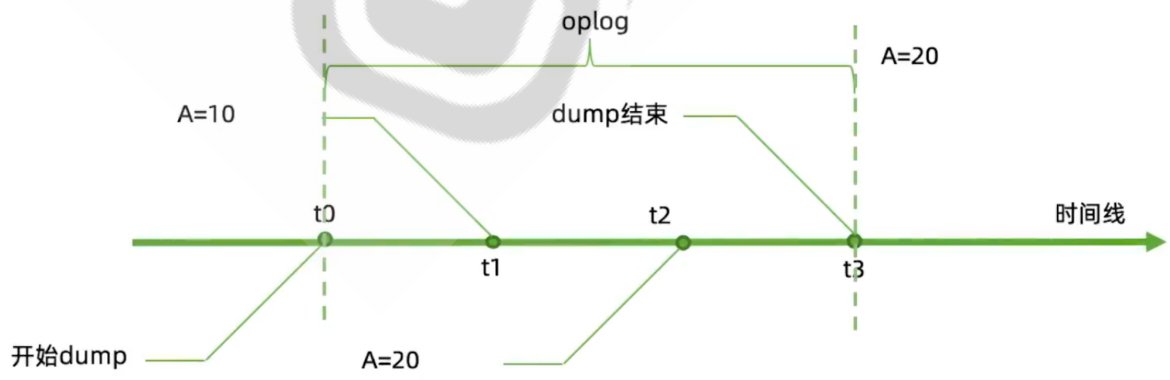
```
#全库备份
#注意:完全备份不会备份config和local库,但会备份admin库和test库及用户库
mkdir -p /backup/full
mongodump -uroot -p123456 --port 27017 --authenticationDatabase admin -o
/backup/full
mongodump -uroot -p123456 --port 27017 --authenticationDatabase admin -o
/backup/full-`date +%F`

#备份指定库test
mkdir -p /backup/test_backup
mongodump -uroot -p123456 --port 27017 --authenticationDatabase admin -d test -o
/backup/test_backup

#备份magedu库下的vast集合
mkdir -p /backup/vast
mongodump-uroot -p123456 --port 27017 --authenticationDatabase admin -d magedu -
c vast -o /backup/vast

#完全备份并压缩备份
mkdir -p /backup/full_gzip
mongodump -uroot -p123456 --port 27017 --authenticationDatabase admin -o
/backup/full_gzip --gzip
```

范例: mongorestore

```
#恢复完全备份
mongorestore -uroot -p123456 --port 27017 --authenticationDatabase admin
/backup/full

#从完全备份的压缩备份中的test库恢复至test1库
mongorestore -uroot -p123456 --port 27017 --authenticationDatabase admin -d
test1 /backup/full_gzip/test --gzip

#从完全备份中的vast表恢复到test库中的test库的表vast1中
mongorestore-uroot -p123456 --port 27017 --authenticationDatabase admin -d test
-c vast1 --gzip /backup/full_gzip/test/vast.bson.gz

#默认不允许覆盖恢复会报错,可以使用选项--drop实现覆盖恢复,表示恢复的时候先之前的集合drop掉(危
险),再恢复
mongorestore -uroot -p123456 --port 27017 --authenticationDatabase admin -d
magedu --drop /backup/magedu
```

6.3.4 利用 oplog 恢复数据

```
#关于oplog说明: 官方:https://docs.mongodb.com/manual/core/replica-set-oplog/
类似于MySQL中的binlog,记录了MongoDB数据的更新,只支持复制集,不支持单节点
在replica set中oplog是一个定容集合(capped collection,固定大小)，保存在local库的
db.oplog.rs
它的默认大小是磁盘空间的5%（可以通过--oplogSizeMB参数修改).
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单位时间内的更新次数不同而
变化。
至少要预留MongoDB的1个全备的周期大小。
默认情况下，MongoDB 不设置最小 oplog 保留期，并自动从最旧的条目开始截断 oplog，以保持配置的最
大 oplog 大小
oplog有一个非常重要的特性--幂等性（idempotent）。即对一个数据集合，使用oplog中记录的操作重放
时，无论被重放多少次，其结果会是一样的。比如，如果oplog中记录的是一个插入操作，并不会因为你重放
了两次，数据库中就得到两条相同的记录。这是一个很重要的特性.
```



范例： 利用oplog幂等性恢复数据的一致

```
#注意：此操作不要在分片集群中执行
#单机默认没有oplog，需要实现复制集才能开启oplog
[root@ubuntu1804 ~]#cat /mongodb/conf/mongo.conf
systemLog:
    destination: file
    path: "/mongodb/log/mongodb.log"
```

```
    logAppend: true
storage:
  dbPath: "/mongodb/data/"
  journal:
    enabled: true
processManagement:
  fork: true
net:
  port: 27017
  bindIp: 0.0.0.0
replication:
  replSetName: myrepl

[root@ubuntu1804 ~]#systemctl restart mongod

#开启复制集
[root@ubuntu1804 ~]#mongo
>rs.initiate()
>rs.initiate({ _id: "myrepl", members: [{_id:0,host:"10.0.0.8:27017"}]})

#查看复制集和oplog信息
myrepl:PRIMARY> rs.printReplicationInfo()
configured oplog size:   4622.462158203125MB    #集合大小
log length start to end: 245022secs (68.06hrs)  #oplog 存在时长
oplog first event time:  Wed Nov 20 2019 17:52:05 GMT+0800 (CST)
oplog last event time:   Wed Nov 20 2019 17:52:05 GMT+0800 (CST)
now:                     Wed Nov 20 2019 17:52:15 GMT+0800 (CST)

#插入大量数据，同时执行后面的备份操作
myrepl:PRIMARY> use test
myrepl:PRIMARY> for (i=1;i<1000000 ;i++){ db.magedu.insert ( { "id" :i, "name" :
"wang" , "age" : 20,"date" : new Date()})}

#oplog只支持完全备份
[root@ubuntu1804 ~]#mkdir /backup/

#使用--oplog选项可以备份同时也备份oplog实现热备份,会记录备份过程中的数据变化。并以oplog.bson
保存下来
[root@ubuntu1804 ~]#mongodump  --oplog -o /backup/
[root@ubuntu1804 ~]#ls /backup/
admin  config  oplog.bson  test

#查看oplog日志
[root@ubuntu1804 ~]#bsondump /backup/oplog.bson | head -n1
{"op":"i","ns":"test.magedu","ui":{"$binary":
{"base64":"hQWhnxxZRT6RqqCJKMi32g==","subType":"04"}},"o":{"_id":
{"$oid":"6195b8594414c155d4825e15"},"id":{"$numberDouble":"5891.0"},"name
":"wang","age":{"$numberDouble":"20.0"},"date":{"$date":
{"$numberLong":"1637202009082"}}},"ts":{"$timestamp":
{"t":1637202009,"i":60}},"t":{"$numberLong":"1"},"v":{"$numberLong":"2"},"wall":
{"$date":{"$numberLong":"1637202009082"}}}
2021-11-18T11:11:00.247+0800    5 objects found
2021-11-18T11:11:00.248+0800    write /dev/stdout: broken pipe

[root@ubuntu1804 ~]#mongo
#查看所有操作的oplog
myrepl:PRIMARY>use local
myrepl:PRIMARY>db.oplog.rs.find().pretty()
```

```
#查看创建操作的oplog,op的i表示insert,u表示update,d表示delete,c表示ddl,n表示note
myrepl:PRIMARY>db.oplog.rs.find({op: "c"}).pretty()

#备份完成后删除表magedu
[root@ubuntu1804 ~]#mongo
myrepl:PRIMARY>use test
myrepl:PRIMARY>db.magedu.drop()

#不使用oplog恢复，查看数据数量
[root@ubuntu1804 ~]#mongorestore --drop  /backup
[root@ubuntu1804 ~]#mongo
>my_repl:PRIMARY> db.magedu.count()

#使用oplog恢复，查看数据数量，和上面比较
[root@ubuntu1804 ~]#mongorestore --drop --oplogReplay  /backup
[root@ubuntu1804 ~]#mongo
>myrepl:PRIMARY> db.magedu.count()
```

范例: 利用oplog恢复误删除的表

```
故障说明:
每天3点全备，oplog恢复窗口为48小时
上午10点test库的vast业务表被误删除。
恢复过程:
1、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test.vast被误删除时间点的oplog
5、恢复到测试库
6、将误删除表导出，恢复到生产库
```

故障还原过程

```
1)完全备份
mongo
myrepl:PRIMARY> use test
myrepl:PRIMARY> for (i=0;i<100 ;i++){ db.vast.insert ( { "id" :i, "name" :
"wang" , "age" : 20,"date" : new Date()})}

#执行完全备份
mkdir -p /backup/full
mongodump  --oplog -o /backup/full
ls /backup/full/
admin  config  oplog.bson  test


2)模拟数据变化
mongo
myrepl:PRIMARY> db.test1.insert({id:1})
myrepl:PRIMARY> db.test2.insert({name:"wang"})
myrepl:PRIMARY> show tables
test1
test2
vast

3)模拟10:00误删除test库中的vast表
```

```
myrepl:PRIMARY> db.vast.drop()
myrepl:PRIMARY> show tables
test1
test2

#开始恢复,先停应用后,执行下面恢复过程
4)备份当前oplog.rs表即备份最新oplog
mongodump  -d local -c oplog.rs  -o /backup/oplog
tree /backup/oplog/
/backup/oplog/
└── local
    ├── oplog.rs.bson
    └── oplog.rs.metadata.json

1 directory, 2 files

5)将备份下来的oplog的bson文件,覆盖完全备份的oplog.bson文件,用来还原
cp /backup/oplog/local/oplog.rs.bson /backup/full/oplog.bson

6)查看local库的oplog.rs表获取误删除vast表的时间戳
myrepl:PRIMARY> use local
myrepl:PRIMARY> db.oplog.rs.find({op:"c"}).pretty()
#查看到误删除vast表的记录如下,记下ts信息
.....
{
    "op" : "c",
    "ns" : "test.$cmd",
    "ui" : UUID("db7e9d49-d965-42ad-9134-863d723cbbd6"),
    "o" : {
        "drop" : "vast"
    },
    "o2" : {
        "numRecords" : 99
    },
    "ts" : Timestamp(1574251087, 1), #此行需要记录下来
    "t" : NumberLong(1),
    "v" : NumberLong(2),
    "wall" : ISODate("2019-11-20T11:58:07.223Z")
}

7)利用oplog恢复备份至误删除时间点
mongorestore --oplogReplay --oplogLimit "1574251087:1" --drop /backup/full/

8)验证是否恢复
myrepl:PRIMARY> use test
myrepl:PRIMARY> show tables;
test1
test2
vast
myrepl:PRIMARY> db.vast.count()
100
```

马哥教育
IT人的高薪职业学院
王晓春