MongoDB

MongoDB介绍

业务应用场景

传统的关系型数据库（如MySQL），在数据操作的“三高”需求以及应对Web2.0的网站需求面前，显得力不从心。

解释：“三高”需求：

- High performance - 对数据库高并发读写的需求。
- Huge Storage - 对海量数据的高效率存储和访问的需求。
- High Scalability && High Availability- 对数据库的高可扩展性和高可用性的需求。

具体的应用场景如：

1）社交场景，使用 MongoDB 存储存储用户信息，以及用户发表的朋友圈信息，通过地理位置索引实现附近的人、地点等功能。

2）游戏场景，使用 MongoDB 存储游戏用户信息，用户的装备、积分等直接以内嵌文档的形式存储，方便查询、高效率存储和访问。

3）物流场景，使用 MongoDB 存储订单信息，订单状态在运送过程中会不断更新，以 MongoDB 内嵌数组的形式来存储，一次查询就能将 订单所有的变更读取出来。

4）物联网场景，使用 MongoDB 存储所有接入的智能设备信息，以及设备汇报的日志信息，并对这些信息进行多维度的分析。

5）视频直播，使用 MongoDB 存储用户信息、点赞互动信息等。

什么时候选择MongoDB?

在架构选型上，除了上述的三个特点外，如果你还犹豫是否要选择它？可以考虑以下的一些问题：

应用不需要事务及复杂 join 支持

新应用，需求会变，数据模型无法确定，想快速迭代开发

应用需要2000-3000以上的读写QPS（更高也可以） 应用需要TB甚至 PB 级别数据存储

应用发展迅速，需要能快速水平扩展

应用要求存储的数据不丢失

应用需要99.999%高可用

应用需要大量的地理位置查询、文本查询

如果上述有1个符合，可以考虑 MongoDB，2个及以上的符合，选择 MongoDB 绝不会后悔。

什么是MongoDB?

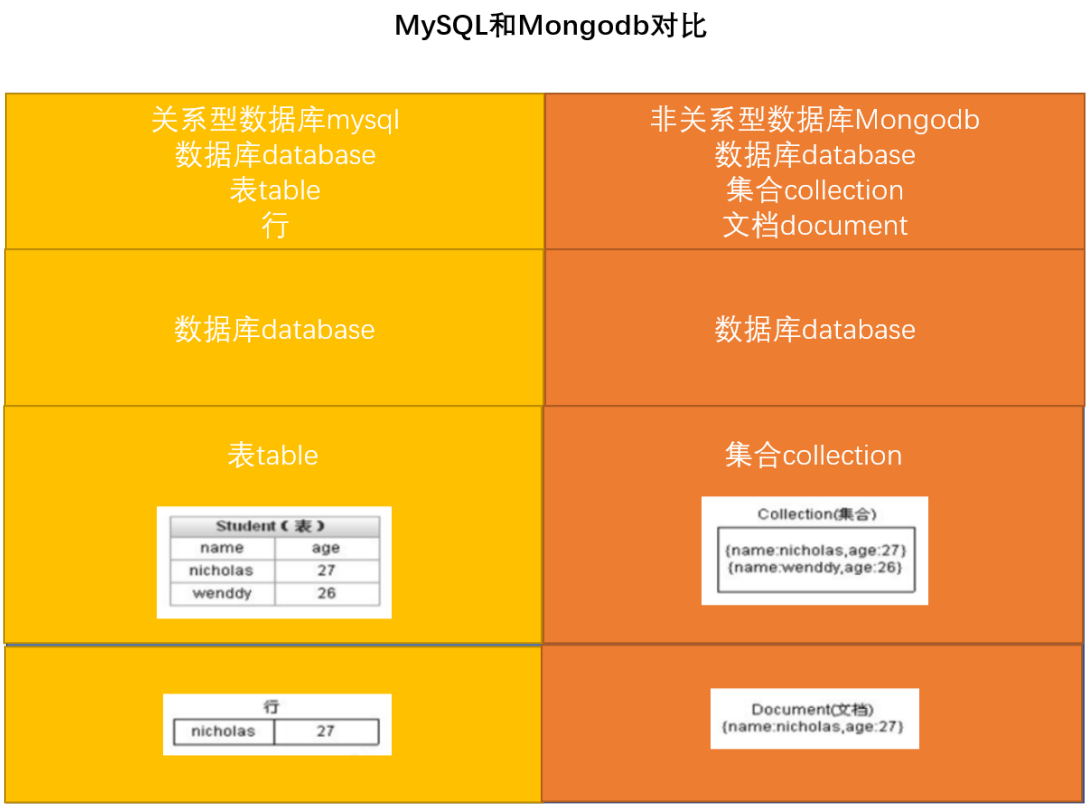
MongoDB是一个开源、高性能、无模式的文档型数据库，当初的设计就是用于简化开发和方便扩展，是NoSQL数据库产品中的一种。是最 像关系型数据库（MySQL）的非关系型数据库。

它支持的数据结构非常松散，是一种类似于 JSON 的 格式叫BSON，所以它既可以存储比较复杂的数据类型，又相当的灵活。

MongoDB中的记录是一个文档，它是一个由字段和值对（field:value）组成的数据结构。MongoDB文档类似于JSON对象，即一个文档认 为就是一个对象。字段的数据类型是字符型，它的值除了使用基本的一些类型外，还可以包括其他文档、普通数组和文档数组。

体系结构

MySQL和MongoDB的对比



MySQL术语/概念	MongoDB术语/概念	解释、说明
database	database	数据库
table	collection	数据库表/集合
row	document	数据记录行/文档
colnum	field	数据字段/域
index	index	索引
table join		表连接，MongoDB不支持
primary key	primary key	主键，MongoDB自动将_id字段设置为主键

数据类型

数据类型	描述	举例
字符串	utf8字符串都可以表示为字符串类型的数据	{"x":"foobar"}
对象id	对象id是文档的12字节的唯一ID	{"X":Objectid()}
布尔值	真或者假：true或者false	{"x":true}
数组	值的集合或者列表都可以表示成数组	{"x":["a","b","c"]}
整数	(Int32 Int64 你们就知道有个Int就行了,一般我们用Int32)	{"age":18}
null	表示空值或者未定义的对象	{"x":null}
undefined	文档中也可以使用未定义类型	{"x":undefined}

MongoDB特点

（1）高性能： MongoDB提供高性能的数据持久性。特别是, 对嵌入式数据模型的支持减少了数据库系统上的I/O活动。 索引支持更快的查询，并且可以包含来自嵌入式文档和数组的键。 （文本索引解决搜索的需求、TTL索引解决历史数据自动过期的需求、地 理位置索引可用于构建各种 O2O 应用）mmapv1、wiredtiger、mongorocks（rocksdb）、in-memory 等多引擎支持满足各种场景需求。Gridfs解决文件存储的需求。

（2）高可用性： MongoDB的复制工具称为副本集（replica set），它可提供自动故障转移和数据冗余。

（3）高扩展性： MongoDB提供了水平可扩展性作为其核心功能的一部分。 分片将数据分布在一组集群的机器上。（海量数据存储，服务能力水平扩展） 从3.4开始，MongoDB支持基于片键创建数据区域。在一个平衡的集群中，MongoDB将一个区域所覆盖的读写只定向到该区域内的那些 片。

（4）丰富的查询支持： MongoDB支持丰富的查询语言，支持读和写操作(CRUD)，比如数据聚合、文本搜索和地理空间查询等。

（5）其他特点：如无模式（动态模式）、灵活的文档模型

部署安装以及基本命令

二进制安装

下载地址：https://www.mongodb.com/try/download/community

- 准备软件包，上官网下载软件包

```
[root@server1 ~]# wget https://fastdl.mongodb.org/linux/mongodb-linux-x86_64-rhel70-4.4.4.tgz
```

- 关闭防火墙和selinux

```
[root@server1 ~]# systemctl stop firewalld
[root@server1 ~]# setenforce 0
```

- 关闭大叶内存机制

```
[root@server1 ~]# sudo mkdir /etc/tuned/virtual-guest-no-thp
[root@server1 ~]# vim /etc/tuned/virtual-guest-no-thp/tuned.conf
[main]
include=virtual-guest
[vm]
transparent_hugepages=never
[root@server1 ~]# sudo tuned-adm profile virtual-guest-no-thp
[root@server1 ~]# cat  /sys/kernel/mm/transparent_hugepage/enabled
always madvise [never]
[root@server1 ~]# cat /sys/kernel/mm/transparent_hugepage/defrag
[always] madvise never
参考文档：https://docs.mongodb.com/manual/tutorial/transparent-huge-pages/
```

- 准备用户

```
[root@server1 ~]# useradd mongod
[root@server1 ~]# passwd mongod
更改用户 mongod 的密码 。
新的 密码：
无效的密码： 密码是一个回文
重新输入新的 密码：
passwd：所有的身份验证令牌已经成功更新。
```

- 准备目录结构

```
[root@server1 ~]# mkdir -p /mongodb/{conf,log,data}
```

- 解压压缩包，并将bin目录复制到/mongodb/下

```
[root@server1 ~]# tar xvf mongodb-linux-x86_64-rhel70-4.4.4.tgz
[root@server1 ~]# cp -r mongodb-linux-x86_64-rhel70-4.4.4/bin /mongodb/
```

- 移动解压后的文件夹到指定的目录中

```
[root@server1 ~]# mkdir  /usr/local/mongodb
[root@server1 ~]# mv mongodb-linux-x86_64-rhel70-4.4.4/* /usr/local/mongodb/
```

- 设置目录结构权限

```
[root@server1 ~]# chown -R mongod:mongod /mongodb
```

- 设置新用户环境变量

```
[root@server1 ~]# su - mongod
[mongod@server1 ~]$ vim .bash_profile
export PATH=/mongodb/bin:$PATH
[mongod@server1 ~]$ source .bash_profile
```

- 启动mongodb

```
[mongod@server1 ~]$ mongod --dbpath=/mongodb/data --
logpath=/mongodb/log/mongodb.log --port=27017 --logappend --fork
```

- 登录mongodb

```
[mongod@localhost ~]$ mongo
```

- 关闭

```
[mongod@localhost ~]$ mongod --dbpath=/mongodb/data --
logpath=/mongodb/log/mongodb.log --port=27017 --logappend --shutdown
```

- 配置文件解释

```
YAML模式

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
   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

------------------以下是副本集与分片集群有关----------------------

replication:
 oplogSizeMB: <NUM>
 replSetName: "<REPSETNAME>"
 secondaryIndexPrefetch: "all"

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

---for mongos only
replication:
```

```
   localPingThresholdMs: <int>

sharding:
   configDB: <string>
---
```

##

- 使用配置文件启动服务

```
[root@server1 ~]# 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: 192.168.80.10,127.0.0.1
[mongod@server1 ~]$ mongod -f /mongodb/conf/mongo.conf
about to fork child process, waiting until server is ready for connections.
forked process: 1676
child process started successfully, parent exiting
[mongod@server1 ~]$ mongod -f /mongodb/conf/mongo.conf --shutdown
```

yum安装

- 准备yum仓库

```
[mngodb-org]
name=MongoDB Repository
baseurl=http://mirrors.aliyun.com/mongodb/yum/redhat/7Server/mongodb-org/4.0/x86_64/
gpgcheck=0
enabled=1
```

- 更新yum仓库

```
[root@localhost ~]# yum update
```

- 使用yum安装

```
[root@localhost ~]# yum install mongodb-org
```

数据库命令

数据库：articledb，专栏文章的评论

字段名称	字段含义	字段类型	备注
_id	ID	Objectid或string	主键
articleid	文章ID	string	
content	评论内容	string	
userid	评论人id	string	
nickname	评论人昵称	string	
createdatetime	评论的日期时间	date	
likenum	点赞数	int	
replynum	回复数	int	
state	状态	string	0：不可见 1：可见
parentid	上级id	string	如果为0表示为文章顶级评论

数据库操作

数据库名可以是满足以下条件的任意UTF-8字符串。

1. 不能是空字符串（""）。
2. 不得含有' '（空格）、.、$、/、\和\0 (空字符)。
3. 应全部小写。
4. 最多64字节。

- 选择和创建数据库

```
# 选择数据库，如果数据库不存在则自动创建
use 数据库名称
> use articledb
switched to db articledb
```

- 查看所有数据库

```
show dbs
show databases
> show dbs
admin   0.000GB
config  0.000GB
local   0.000GB
> show databases
admin   0.000GB
config  0.000GB
local   0.000GB
有一些数据库名是保留的，可以直接访问这些有特殊作用的数据库。
admin： 从权限的角度来看，这是"root"数据库。要是将一个用户添加到这个数据库，这个用户自动继承所有数据库的权限。一些特
定的服务器端命令也只能从这个数据库运行，比如列出所有的数据库或者关闭服务器。
local: 这个数据永远不会被复制，可以用来存储限于本地单台服务器的任意集合
config: 当Mongo用于分片设置时，config数据库在内部使用，用于保存分片的相关信息。
```

- 查看当前所在数据库

```
# 如果没有创建出数据库，默认的数据库为test
> db
articledb
```

- 删除当前所在数据库

```
> db.dropDatabase()
```

集合操作

集合的命名规范：

1. 集合名不能是空字符串""。
2. 集合名不能含有\0字符（空字符），这个字符表示集合名的结尾。
3. 集合名不能以"system."开头，这是为系统集合保留的前缀。
4. 用户创建的集合名字不能含有保留字符。有些驱动程序的确支持在集合名里面包含，这是因为某些系统生成的集合中包含该字符。除 非你要访问这种系统创建的集合，否则千万不要在名字里出现$。

- 创建一个名为 mycollection 的普通集合

```
> db.createCollection("mycollection")
```

- 查看当前所在库的所有集合

```
> show collections;
mycollection
> show tables;
mycollection
```

- 删除mycollection集合

```
删除成功返回结果为true，删除失败返回结果为false
> db.mycollection.drop()
true
```

文档操作

文档（document）的数据结构和 JSON 基本一样。

所有存储在集合中的数据都是 BSON 格式。

JSON示例：

```

{

"employees": [

{ "firstName":"Bill" , "lastName":"Gates" },

```

```
{ "firstName":"George" , "lastName":"Bush" },

{ "firstName":"Thomas" , "lastName":"Carter" }

]

}
```

- 单个文档插入，使用insert或者save方法插入文档

```
db.comment.insert({"articleid":"100000","content":"今天天气真好，阳光明媚","userid":"1001","nickname":"Rose","createdatetime":new Date(),"likenum":NumberInt(10),"state":null})
WriteResult({ "nInserted" : 1 })
解释说明
1. 如果插入数据的集合不存在，则会隐式地创建出此集合
2. mongo中的数字，默认情况下是double类型，如果要存整型，必须使用函数NumberInt(整型数字)
3. 插入当前日期使用 new Date()
4. 插入的数据没有指定 _id ，会自动生成主键值
5. 如果某字段没值，可以赋值为null，或不写该字段
```

- 文档内容相关
  - 文档中的键/值对是有序的。
  - 文档中的值不仅可以是在双引号里面的字符串，还可以是其他几种数据类型（甚至可以是整个嵌入的文档)。
  - MongoDB区分类型和大小写。
  - MongoDB的文档不能有重复的键。
  - 文档的键是字符串。除了少数例外情况，键可以使用任意UTF-8字符。
  - 键不能含有\0 (空字符)。这个字符用来表示键的结尾。
  - .和$有特别的意义，只有在特定环境下才能使用。
  - 以下划线"_"开头的键是保留的(不是严格要求的)。
- 批量插入多行数据

```
db.comment.insertMany([
    {"_id":"1","articleid":"100001","content":"我们不应该把清晨浪费在手机上，健康很重要，一杯温水幸福你我他。","userid":"1002","nickname":"相忘于江湖","createdatetime":new Date("2019-08-05T22:08:15.522Z"),"likenum":NumberInt(1000),"state":"1"},
    {"_id":"2","articleid":"100001","content":"我夏天空腹喝凉开水，冬天喝温开水","userid":"1005","nickname":"伊人憔悴","createdatetime":new Date("2019-08-05T23:58:51.485Z"),"likenum":NumberInt(888),"state":"1"},
    {"_id":"3","articleid":"100001","content":"我一直喝凉开水，冬天夏天都喝。","userid":"1004","nickname":"杰克船长","createdatetime":new Date("2019-08-06T01:05:06.321Z"),"likenum":NumberInt(666),"state":"1"},
    {"_id":"4","articleid":"100001","content":"研究表明，刚烧开的水千万不能喝，因为烫嘴。","userid":"1003","nickname":"凯撒","createdatetime":new Date("2019-08-06T11:01:02.521Z"),"likenum":NumberInt(3000),"state":"1"}
]);
提示
插入时指定了 _id ，则主键就是该值。
如果某条数据插入失败，将会终止插入，但已经插入成功的数据不会回滚掉。
```

- 查询所有数据

```
> db.comment.find()
> db.comment.find({})
```

- 查询指定文档

```
> db.comment.find({userid:'1003'})
```

- 转换为JSON格式输出

```
> db.comment.find().pretty()
```

- 投影查询

```
> db.comment.find({userid:"1003"},{userid:1,nickname:1})
> db.comment.find({},{userid:1,nickname:1})
> db.comment.find({},{userid:1,nickname:1,_id:0})
```

- 覆盖修改

```
> db.comment.update({_id:"1"},{userid:"10000"})
> db.comment.find({userid:"10000"})
{ "_id" : "1", "userid" : "10000" }
```

- 局部修改

```
为了只修改部分数据需要使用$set修改器
> db.comment.update({_id:"2"},{$set:{likenum:NumberInt(889)}})
```

- 批量修改

```
db.comment.update({userid:"1003"},{$set:{nickname:"凯撒大帝"}},{multi:true})
```

- 删除文档

```
# 删除全部文档
db.comment.remove({})
# 删除指定文档
db.comment.remove({_id:"1"})
```

文档的更多查询

- 统计所有记录数量

```
db.comment.count()
```

- 按条件统计记录

```
db.comment.count({userid:"1003"})
```

- 分页查询

```
# limit是限制行数，skip是查询的起始位置
>db.COLLECTION_NAME.find().limit(NUMBER).skip(NUMBER)
```

- 排序查询

```
使用 1 和 -1 来指定排序的方式，其中 1 为升序排列，而 -1 是用于降序排列。
db.COLLECTION_NAME.find().sort({KEY:1})
```

- 正则表达式查询

```
db.collection.find({field:/正则表达式/})
正则表达式采用的是js的语法
db.comment.find({content:/开水/})
```

- 比较查询

```
db.集合名称.find({ "field" : { $gt: value }}) // 大于: field > value
db.集合名称.find({ "field" : { $lt: value }}) // 小于: field < value
db.集合名称.find({ "field" : { $gte: value }}) // 大于等于: field >= value
db.集合名称.find({ "field" : { $lte: value }}) // 小于等于: field <= value
db.集合名称.find({ "field" : { $ne: value }}) // 不等于: field != value
db.comment.find({likenum:{$gt:NumberInt(700)}})
```

- 包含查询

```
# $in表示userid在其中的
db.comment.find({userid:{$in:["1003","1004"]}})
# $nin表示userid不在其中的
db.comment.find({userid:{$nin:["1003","1004"]}})
```

- 条件连接查询

```
$and:[ { },{ },{ } ]
$or:[ { },{ },{ } ]
db.comment.find({$and:[{likenum:{$gte:NumberInt(700)}},{likenum:
{$lt:NumberInt(2000)}}]})
```

命令小结

```
选择切换数据库：use articledb
插入数据：db.comment.insert({bson数据})
查询所有数据：db.comment.find();
条件查询数据：db.comment.find({条件})
查询符合条件的第一条记录：db.comment.findOne({条件})
查询符合条件的前几条记录：db.comment.find({条件}).limit(条数)
查询符合条件的跳过的记录：db.comment.find({条件}).skip(条数)
修改数据：db.comment.update({条件},{修改后的数据}) 或db.comment.update({条件},
{$set:{要修改部分的字段:数据})
修改数据并自增某字段值：db.comment.update({条件},{$inc:{自增的字段:步进值}})
删除数据：db.comment.remove({条件})
统计查询：db.comment.count({条件})
模糊查询：db.comment.find({字段名:/正则表达式/})
条件比较运算：db.comment.find({字段名:{$gt:值}})
```

```
包含查询：db.comment.find({字段名:{$in:[值1，值2]}})或db.comment.find({字段名:{$nin:[值1，值2]}})
条件连接查询：db.comment.find({$and:[{条件1},{条件2}]})或db.comment.find({$or:[{条件1},{条件2}]})
```

索引管理

概述

索引支持在MongoDB中高效地执行查询。如果没有索引，MongoDB必须执行全集合扫描，即扫描集合中的每个文档，以选择与查询语句 匹配的文档。这种扫描全集合的查询效率是非常低的，特别在处理大量的数据时，查询可以要花费几十秒甚至几分钟，这对网站的性能是非 常致命的。

如果查询存在适当的索引，MongoDB可以使用该索引限制必须检查的文档数。 索引是特殊的数据结构，它以易于遍历的形式存储集合数据集的一小部分。索引存储特定字段或一组字段的值，按字段值排序。索引项的排 序支持有效的相等匹配和基于范围的查询操作。此外，MongoDB还可以使用索引中的排序返回排序结果

索引类型

- 单字段索引
  - MongoDB支持在文档的单个字段上创建用户定义的升序/降序索引，称为单字段索引（Single Field Index）。
- 复合索引
  - MongoDB还支持多个字段的用户定义索引，即复合索引（Compound Index）。
- 其他索引
  - 地理空间索引（Geospatial Index）、文本索引（Text Indexes）、哈希索引（Hashed Indexes）。

索引的管理操作

- 查看索引

```
> db.comment.getIndexes()
[
    {
        "v" : 2,
        "key" : {
            "_id" : 1
        },
        "name" : "_id_",
        "ns" : "articledb.comment"
    }
]
```

- 创建单字段索引

```
> db.comment.createIndex({userid:1})
```

- 创建复合索引

```
> db.comment.createIndex({userid:1,nickname:-1})
```

- 删除所有索引

```
> db.comment.dropIndex({})
```

- 删除指定索引

```
> db.comment.dropIndex({userid:1})
```

副本集

简介

MongoDB中的副本集（Replica Set）是一组维护相同数据集的mongod服务。 副本集可提供冗余和高可用性，是所有生产部署的基础。

也可以说，副本集类似于有自动故障恢复功能的主从集群。通俗的讲就是用多台机器进行同一数据的异步同步，从而使多台机器拥有同一数据的多个副本，并且当主库宕掉时在不需要用户干预的情况下自动切换其他备份服务器做主库。而且还可以利用副本服务器做只读服务器，实现读写分离，提高负载。

（1）冗余和数据可用性

复制提供冗余并提高数据可用性。 通过在不同数据库服务器上提供多个数据副本，复制可提供一定级别的容错功能，以防止丢失单个数据库服务器。 在某些情况下，复制可以提供增加的读取性能，因为客户端可以将读取操作发送到不同的服务上， 在不 同数据中心维护数据副本可以增加分布式应用程序的数据位置和可用性。 您还可以为专用目的维护其他 副本，例如灾难恢复，报告或备份。

（2）MongoDB中的复制

副本集是一组维护相同数据集的mongod实例。 **副本集包含多个数据承载节点和可选的一个仲裁节点。**在承载数据的节点中，一个且仅一个成员被视为主节点，而其他节点被视为次要（从）节点。 主节点接收所有写操作。 副本集只能有一个主要能够确认具有{w：“most”}写入关注的写入; 虽然在某 些情况下，另一个mongod实例可能暂时认为自己也是主要的。主要记录其操作日志中的数据集的所有 更改，即oplog。

（3）主从复制和副本集区别

主从集群和副本集最大的区别就是副本集没有固定的“主节点”；整个集群会选出一个“主节点”，当其挂掉后，又在剩下的从节点中选中其他节点为“主节点”，副本集总有一个活跃点(主、primary)和一个或多个备份节点(从、secondary)。

副本集的三个角色

- 主要成员（Primary）：主要接收所有写操作。就是主节点。
- 副本成员（Replicate）：从主节点通过复制操作以维护相同的数据集，即备份数据，不可写操作，但可 以读操作（但需要配置）。是默认的一种从节点类型。
- 仲裁者（Arbiter）：不保留任何数据的副本，只具有投票选举作用。当然也可以将仲裁服务器维护为副 本集的一部分，即副本成员同时也可以是仲裁者。也是一种从节点类型。

关于仲裁者的额外说明：

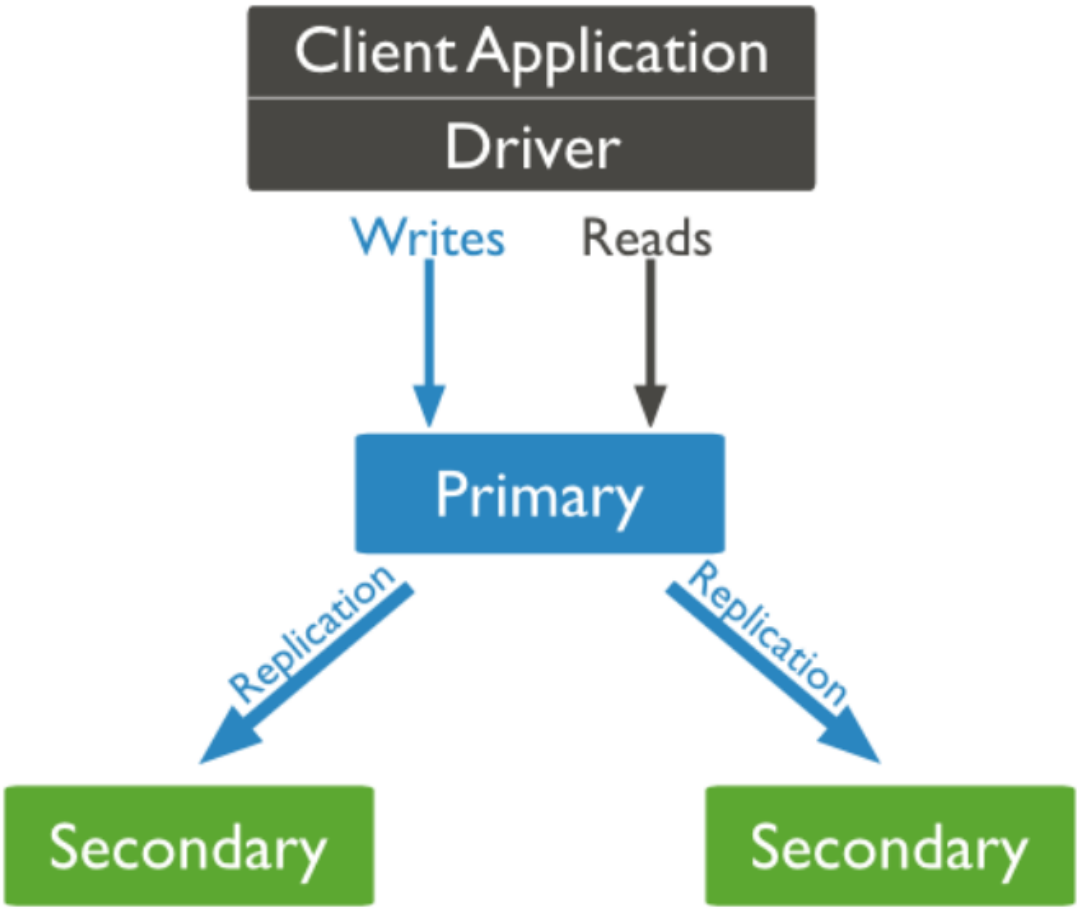
您可以将额外的mongod实例添加到副本集作为仲裁者。 仲裁者不维护数据集。 仲裁者的目的是通过响应其他副本集成员的心跳和选举请求来维护副本集中的仲裁。 因为它们不存储数据集，所以仲裁器可以是提供副本集仲裁功能的好方法，其资源成本比具有数据集的全功能副本集成员更便宜。

如果您的副本集具有偶数个成员，请添加仲裁者以获得主要选举中的“大多数”投票。 仲裁者不需要专用硬件。

仲裁者将永远是仲裁者，而主要人员可能会退出并成为次要人员，而次要人员可能成为选举期间的主要人员。

如果你的副本+主节点的个数是偶数，建议加一个仲裁者，形成奇数，容易满足大多数的投票。

如果你的副本+主节点的个数是奇数，可以不加仲裁者。



副本集搭建

- 准备三台节点

```
# 在三台节点的配置文件中都需要加上以下配置，并且replSetName的值三台节点必须一致
replication:
  oplogSizeMB: 2048
  replSetName: my_repl
```

- 分别启动三个实例
- 主节点初始化，并添加副本

```
> rs.initiate()
{
    "info2" : "no configuration specified. Using a default configuration for the set",
    "me" : "127.0.0.1:27017",
    "ok" : 1,
    "operationTime" : Timestamp(1602764647, 1),
    "$clusterTime" : {
        "clusterTime" : Timestamp(1602764647, 1),
        "signature" : {
            "hash" : BinData(0,"AAAAAAAAAAAAAAAAAAAAAAAAAAA="),
```

```
            "keyId" : NumberLong(0)
        }
    }
}
my_repl:SECONDARY>
my_repl:PRIMARY>
提示：
1）“ok”的值为1，说明创建成功。
2）命令行提示符发生变化，变成了一个从节点角色，此时默认不能读写。稍等片刻，回车，变成主节点。
```

- 查看副本节点配置

```
my_repl:PRIMARY> rs.conf()
{
    "_id" : "my_repl",
    "version" : 1,
    "protocolVersion" : NumberLong(1),
    "writeConcernMajorityJournalDefault" : true,
    "members" : [
        {
            "_id" : 0,
            "host" : "127.0.0.1:27017",
            "arbiterOnly" : false,
            "buildIndexes" : true,
            "hidden" : false,
            "priority" : 1,
            "tags" : {

            },
            "slaveDelay" : NumberLong(0),
            "votes" : 1
        }
    ],
    "settings" : {
        "chainingAllowed" : true,
        "heartbeatIntervalMillis" : 2000,
        "heartbeatTimeoutSecs" : 10,
        "electionTimeoutMillis" : 10000,
        "catchUpTimeoutMillis" : -1,
        "catchUpTakeoverDelayMillis" : 30000,
        "getLastErrorModes" : {

        },
        "getLastErrorDefaults" : {
            "w" : 1,
            "wtimeout" : 0
        },
        "replicaSetId" : ObjectId("5f883f6730b7d82e37e0dcca")
    }
}
1） "_id" : "my_repl" ：副本集的配置数据存储的主键值，默认就是副本集的名字
2） "members" ：副本集成员数组，此时只有一个： "host" : "127.0.0.1:27017" ，该成员不
是仲裁节点： "arbiterOnly" : false ，优先级（权重值）： "priority" : 1,
3） "settings" ：副本集的参数配置。
```

- 添加副本集

```
# 从节点包括仲裁节点是不需要做任何操作的，只需要正常启动即可，剩下的交给主节点去添加
# 添加副本节点
myrs:PRIMARY> rs.add("192.168.80.136:27017")
# 添加仲裁节点
myrs:PRIMARY> rs.addArb("192.168.80.137:27017")
```

- 设置从节点可读

```
rs.slaveOk()
```

分片集群

概念

分片（sharding）是一种**跨多台机器分布数据**的方法， MongoDB使用分片来支持具有非常大的数据集和高吞吐量操作的部署。

换句话说：分片(sharding)是指将数据拆分，将其分散存在不同的机器上的过程。有时也用分区(partitioning)来表示这个概念。将数据分散到不同的机器上，不需要功能强大的大型计算机就可以储存更多的数据，处理更多的负载。

具有大型数据集或高吞吐量应用程序的数据库系统可以会挑战单个服务器的容量。例如，高查询率会耗尽服务器的CPU容量。工作集大小大于系统的RAM会强调磁盘驱动器的I / O容量。

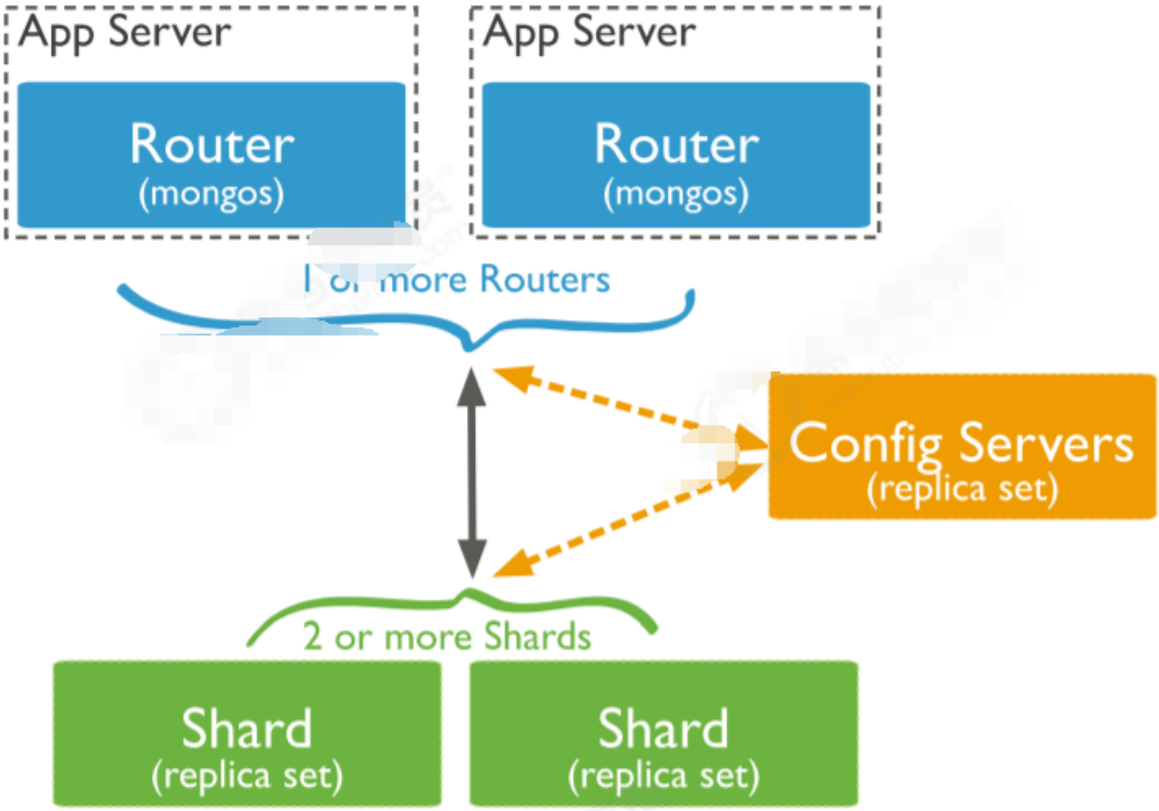
有两种解决系统增长的方法：垂直扩展和水平扩展。

垂直扩展意味着增加单个服务器的容量，例如使用更强大的CPU，添加更多RAM或增加存储空间量。可用技术的局限性可能会限制单个机器对于给定工作负载而言足够强大。此外，基于云的提供商基于可用的硬件配置具有硬性上限。结果，垂直缩放有实际的最大值。 水平扩展意味着划分系统数据集并加载多个服务器，添加其他服务器以根据需要增加容量。虽然单个机 器的总体速度或容量可能不高，但每台机器处理整个工作负载的子集，可能提供比单个高速大容量服务器更高的效率。扩展部署容量只需要根据需要添加额外的服务器，这可能比单个机器的高端硬件的总体 成本更低。权衡是基础架构和部署维护的复杂性增加。

MongoDB支持通过分片进行水平扩展。

分片集群包含的组件

- 分片（存储）：每个分片包含分片数据的子集。 每个分片都可以部署为副本集。
- mongos（路由）：mongos充当查询路由器，在客户端应用程序和分片集群之间提供接口。
- config servers（“调度”的配置）：配置服务器存储群集的元数据和配置设置。 从MongoDB 3.4开始，必须将配置服务器部署为副本集（CSRS）。




```
路由节点mongos
    * 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
    * 建议至少两个
配置节点mongod
    * 提供集群元数据存储
    * 分片数据分布的映射
数据节点mongod
    * 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以完整工作
```

集群部署

规划

```
10个实例：38017-38026
（1）configserver: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）
（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
```

分片节点准备

- 目录准备

```
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
```

- 第一组副本集搭建：端口号21-23，一主一从一监控

```
cat >  /mongodb/38021/conf/mongodb.conf  <<EOF
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: 192.168.80.10,127.0.0.1
  port: 38021
replication:
  oplogSizeMB: 2048
  replSetName: sh1
sharding:
  clusterRole: shardsvr
processManagement:
  fork: true
EOF
\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
```

- 第二组副本集搭建：端口号24-26，一主一从一监控

```
cat > /mongodb/38024/conf/mongodb.conf <<EOF
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: 192.168.80.10,127.0.0.1
  port: 38024
replication:
  oplogSizeMB: 2048
  replSetName: sh2
sharding:
  clusterRole: shardsvr
processManagement:
  fork: true
EOF

\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
```

- 启动所有节点，并搭建副本集

```
mongod -f  /mongodb/38021/conf/mongodb.conf --fork
mongod -f  /mongodb/38022/conf/mongodb.conf --fork
mongod -f  /mongodb/38023/conf/mongodb.conf --fork
mongod -f  /mongodb/38024/conf/mongodb.conf --fork
mongod -f  /mongodb/38025/conf/mongodb.conf --fork
mongod -f  /mongodb/38026/conf/mongodb.conf  --fork
ps -ef |grep mongod

mongo --port 38021
use  admin
config = {_id: 'sh1', members: [
                          {_id: 0, host: '192.168.80.10:38021'},
                          {_id: 1, host: '192.168.80.10:38022'},
                          {_id: 2, host:
'192.168.80.10:38023',"arbiterOnly":true}]
          }

rs.initiate(config)

 mongo --port 38024
 use admin
config = {_id: 'sh2', members: [
                          {_id: 0, host: '192.168.80.10:38024'},
                          {_id: 1, host: '192.168.80.10:38025'},
```

```
                            {_id: 2, host:
'192.168.80.10:38026',"arbiterOnly":true}]
          }

rs.initiate(config)
```

config节点配置

- 目录创建

```
mkdir -p /mongodb/38018/conf  /mongodb/38018/log  /mongodb/38018/data
mkdir -p /mongodb/38019/conf  /mongodb/38019/log  /mongodb/38019/data
mkdir -p /mongodb/38020/conf  /mongodb/38020/log  /mongodb/38020/data
```

- 配置文件准备

```
cat > /mongodb/38018/conf/mongodb.conf <<EOF
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: 192.168.80.10,127.0.0.1
  port: 38018
replication:
  oplogSizeMB: 2048
  replSetName: configReplSet
sharding:
  clusterRole: configsvr
processManagement:
  fork: true
EOF

\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
```

- 启动所有节点

```
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: '192.168.80.10:38018'},
                            {_id: 1, host: '192.168.80.10:38019'},
                            {_id: 2, host: '192.168.80.10:38020'}]
           }
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
```

mongos节点配置

- 目录创建

```
mkdir -p /mongodb/38017/conf  /mongodb/38017/log
```

- 准备配置文件

```
cat > /mongodb/38017/conf/mongos.conf <<EOF
systemLog:
  destination: file
  path: /mongodb/38017/log/mongos.log
  logAppend: true
net:
  bindIp: 192.168.80.10,127.0.0.1
  port: 38017
sharding:
  configDB:
configReplSet/192.168.80.10:38018,192.168.80.10:38019,192.168.80.10:38020
processManagement:
  fork: true
EOF
```

- 启动mongos

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

使用分片

- 分片集群添加节点

```
 连接到其中一个mongos（192.168.80.10），做以下配置
（1）连接到mongs的admin数据库
# su - mongod
$ mongo 192.168.80.10:38017/admin
（2）添加分片
db.runCommand( { addshard : "sh1/192.168.80.10:38021,192.168.80.10:38022,192.168.80.10:38023",name:"shard1"} )
db.runCommand( { addshard : "sh2/192.168.80.10:38024,192.168.80.10:38025,192.168.80.10:38026",name:"shard2"} )
（3）列出分片
mongos> db.runCommand( { listshards : 1 } )
（4）整体状态查看
mongos> sh.status();
```

- 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" } )
```

- 指定分片键对集合分片

```
### 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} } )
```

- 分片验证

```
admin> use test
test> for(i=1;i<10000000;i++){ db.vast.insert({"id":i,"name":"zhenjiang","age":70,"date":new Date()}); }
test> db.vast.stats()
shard1:
mongo --port 38021
db.vast.count();

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

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.
5. 从3.6 版本开始，不添加bindIp参数，默认不让远程登录，只能本地管理员登录。
```

- 用户创建语法

```
use 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 eagle -p 123 10.0.0.53/eagle
```

- 创建超级管理员

```
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')
```

- 配置文件开启用户认证

```
security:
  authorization: enabled
```

- 登录验证

```
mongo -uroot -proot123  admin
mongo -uroot -proot123  10.0.0.53/admin

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

- 查看用户

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

- 删除用户

```
删除用户
db.createUser({user: "app02",pwd: "app02",roles: [ { role: "readWrite" , db: "eagle" } ]})
mongo -uroot -proot123 10.0.0.53/admin
use eagle
db.dropUser("app02")
```

- 用户角色
  - 数据库用户角色：read、readWrite;
  - 所有数据库用户角色：readAnyDatabase、readWriteAnyDatabase、userAdminAnyDatabase、dbAdminAnyDatabase
  - 数据库管理角色：dbAdmin、dbOwner、userAdmin；
  - 集群管理角色：clusterAdmin、clusterManager、clusterMonitor、hostManager；
  - 备份恢复角色：backup、restore；
  - 超级用户角色：root
  - 内部角色：system