什么是nosql？

nosql指的是not only sql,即非关系型数据库，以MongoDB为例，它所采取的存储方式是一种类似JSON格式的数据，通过健-值的形式进行数据的存储，而非如MySql之类的关系型数据库那样，以表的方式来构建数据之间的关系集合。

传统的关系型数据库存在着几个天生的限制，比如扩展困难，成本高，在数据达到一定规模后，表之间的逻辑关系非常复杂，因此会导致其读写速度的下滑非常严重，在面临海量数据的时候，其表现并不是让人十分满意。

在这个基础上，出现了nosql，相对于关系型数据库，其更关注于高并发读写和海量数据的存储，由于其存储方式（以MongoDB为例）是键-值的形式，结构更加灵活，它的存储是动态的，因而在水平扩展上十分的方便，适合于现在普遍的分布式架构。

**分布式与集群**

小饭店原来只有一个厨师，切菜洗菜备料炒菜全干。后来客人多了，厨房一个厨师忙不过来，又请了个厨师，两个厨师都能炒一样的菜，这两个厨师的关系是集群。为了让厨师专心炒菜，把菜做到极致，又请了个配菜师负责切菜，备菜，备料，厨师和配菜师的关系是分布式，一个配菜师也忙不过来了，又请了个配菜师，两个配菜师关系是集群。

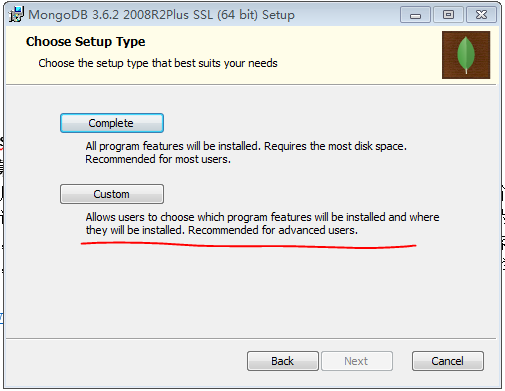
说白了，分布式指的是从业务中分离出子业务，部署到其他服务器上面，而集群，则是相同业务分布在不同的服务器上，实现负载均衡。

MongoDB通过副本集与分片功能，使得其很容易的实现分布式与集群两方面的扩展。

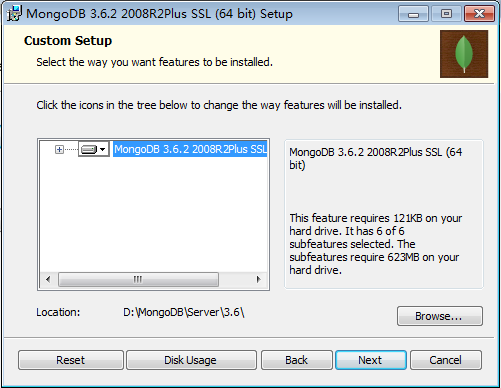
**Windows上的MongoDB安装**

下载地址(据说3.6版在windows上有bug，所以我下载了3.4版本)：  
<https://www.mongodb.com/download-center>

1.下载msi好后，双击打开，选择“Custom”



2.选择安装位置，下一步安装即可。



3.选择合适的位置，新建mongodb文件夹，在mongodb文件夹里新进data文件夹，作为mongodb的数据存储位置（名称自己定），同时在mongodb文件夹下建立log文件夹,再在log下创建mongo.log文件，用于存储log日志。

4.通过cmd进入mongodb安装的bin目录下，执行（若在path环境变量中，加入mongodb的bin路径，则可以在任意位置直接使用mongod，不需要再进入bin目录）

**mongod --port 27017 --dbpath “e:/mongodb/data” --logpath “e:/mongod/log/mongo.log”**

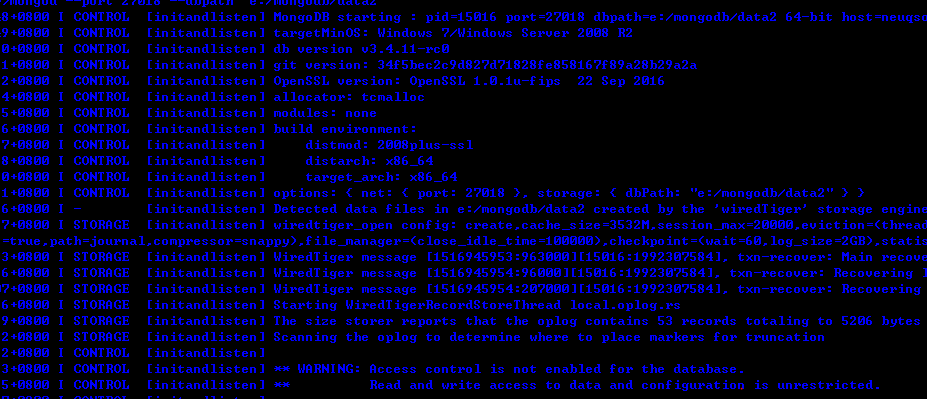
port指定端口，dbpath指定数据存储位置，logpath指定日志记录位置

图1 正常启动

如图1为正常启动服务，此时登录127.0.0.1:27017（在这里我用的是27018端口），显示图2，则启动成功

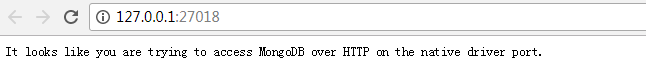


图2

5.也可以通过建立配置文件mongo.conf，内容如下

1. dbpath=e:\mongodb\data #数据库路径
2. logpath=e:\mongodb\log\mongo.log #日志输出文件路径
3. logappend=true #错误日志采用追加模式
4. journal=true #启用日志文件，默认启用
5. quiet=true #这个选项可以过滤掉一些无用的日志信息，若需要调试使用请设置为false
6. port=27017 #端口号 默认为27017

之后，启动可以采用mongod --config “d:/mongodb/server/3.4/mongo.conf” 的方式使用配置文件来启动

6. 每次都通过这种方式启动很麻烦，可以将其作为服务来管理

1. mongod --config "d:/mongodb/server/3.4/mongo.conf" --install --serviceName "MongoDB"

如上建立MongoDB服务后，可以直接通过 net start MongoDB来启动服务，也可以在services.msc中管理启动和关闭。

1. 启动后，就可以在cmd中，使用mongo.exe来连接服务

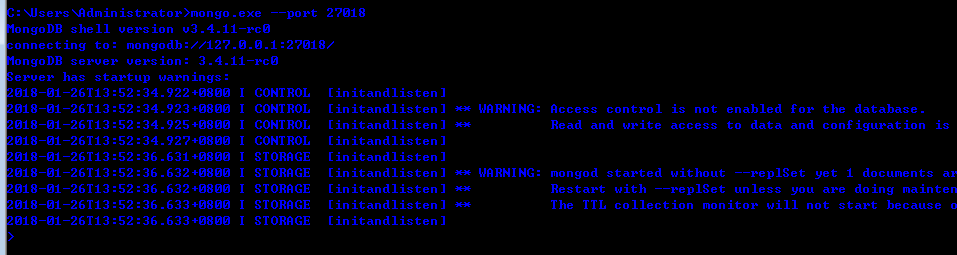


图3

1. 至此，mongodb安装完成。

**linux下安装mongodb**

***第一种（不推荐）***

1. 从官网选择，下载linux版本的tgz文件
2. 使用cd命令进入下载的tgz的文件夹
3. 使用tar –zxvf tgz文件全名进行解压到当前目录
4. 为了方便，可以在opt文件夹下使用sudo mkdir mongodb创建一个mongodb文件夹，同理再在mongodb下创建data文件夹和logs文件夹，然后使用sudo touch mongodb.log创建log文件用来记录日志
5. 再使用sudo mv –r 解压的文件夹名 ，将解压的安装包下的文件全部移动到mongodb中（这样做的目的是因为原来的解压文件夹名字太复杂， 不好操作，所以移动到自己创建的mongodb文件夹中来操作）
6. 使用sudo chmod –R 777 /opt/mongodb/给mongodb文件夹赋予所有可操作权限（不进行这一步授权，会导致log文件无法访问）
7. 进入/opt/mongodb/bin/文件夹下，执行sudo ./mongod --dbpath data路径 --logpath mongodb.log文件路径 --logappend(日志追加，否则默认是重写) --fork（开启守护进程）,启动服务器
8. 同样进入bin，使用sudo ./mongo 连接服务器后，即可访问数据库
9. 创建多个端口服务参考windows
10. 可以使用sudo vim /opt/mongodb/mongo.conf创建配置文件，配置文件内容参考windows，然后使用sudo ./mongod --config mongo.conf位置来通过配置文件启动 (ubuntu下是vi)

***第二种（官方文档建议，推荐）***

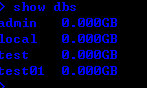
1. 使用sudo apt-key adv --keyserver hkp://keyserver.ubuntu.com:80 --recv 2930ADAE8CAF5059EE73BB4B58712A2291FA4AD5导入mongodb公钥
2. 使用echo "deb [ arch=amd64 ] http://repo.mongodb.com/apt/ubuntu trusty/mongodb-enterprise/3.6 multiverse" | sudo tee /etc/apt/sources.list.d/mongodb-enterprise.list创建list文件（此处是ubuntu14.04版本，其余版本可以去官方文档上查看）
3. sudo apt-get update更新一下ubuntu的安装包
4. 使用sudo apt-get install -y mongodb-enterprise=3.6.2 mongodb-enterprise-server=3.6.2 mongodb-enterprise-shell=3.6.2 mongodb-enterprise-mongos=3.6.2 mongodb-enterprise-tools=3.6.2（此处是3.6.2版本，可以自行更换版本号）进行安装文件的下载，一路回车即可
5. 下载完成后，即可直接在任意位置使用sudo service mongod start进行默认服务器的启动（默认端口为27017）,可以使用sudo service mongod restart重启，使用sudo service mongod stop停止服务
6. 在任意位置使用sudo mongo进行服务器连接，连接成功后即可进行操作
7. 第二种安装方式，自动进行了data、log、conf的创建，其位置分别在 /var/lib/mongodb /var/log/mongodb/mongod.log以及/etc/mongod.conf，可以通过修改conf文件来进行相关配置的修改，第二种安装方式自动创建了开机自启动和添加mongod服务，当然同时也可以使用第一种的启动方式。
8. 卸载（1）sudo apt-get purge mongodb-org\* （2）sudo rm -r /var/log/mongodb （3）sudo rm -r /var/lib/mongodb
9. 卸载之前首先停止服务

**创建与删除数据库**

use DATABASE

用于选择数据库，若数据库不存在，则会自动创建数据库，但是如果新建数据库里没有内容，则切换其他数据库后，该后数据库会被删除

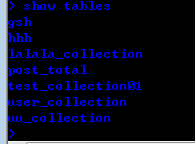
可以使用 show dbs来查看目前存在的所有数据库



在当前数据库下使用db.dropDatabase()命令，可以删除该数据库

**创建与删除collection集合**

mongodb可以直接在数据库下用 db.collection\_name.insert()来往collection\_name集合中插入数据，若该集合不存在，则会自动创建该集合。

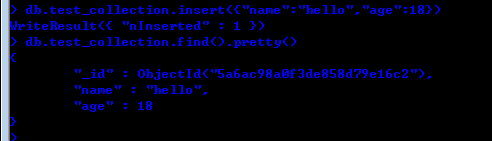
可以使用show tables 来查看所选数据库下的存在的集合

可以使用db.collection\_name.drop()来删除集合。

**插入文档（Document）**

db.test\_collection.insert({“name”:”hello”,”age”:18})

执行该命令，会创建test\_collection集合并插入一条文档，内容是name:hello,gae:18

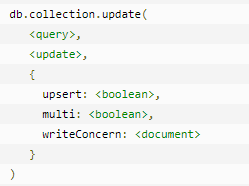


如图，插入成功会有WriteResult({“nInserted”:1})，使用db.test\_collection.find().pretty()进行查看目前集合中的所有信息，pretty()用于格式化结果。

db.test\_collection.insertMany([{“name”:”s1”,”age”:21},{“name”:”s2”,”age”:23}])

如上为插入多条。

**更新文档（Document）**



其中，query为查询条件，update为要更新的内容，upsert表示，如果更新的内容不存在，是否进行新的插入，true为插入，false为不插入；multi表示是否把符合条件的文档全部更新，若为false，则只更新第一条符合条件的信息，若为true则全部更新；writeConcern为抛出的异常等级。

db.test\_collection.update({“name”:”hello”},{$set:{“age”:30}})

如上，为查询出名称为hello的文档并将该条信息的age修改为30

3.2版本之后，可以使用 updateOne()和updateMany()来进行单条或者多条的更新

save()方法可以根据传入的文档来替换文档,若指定的\_id不存在，则会插入一条新信息，若存在，则更新已经存在的信息，并且这种替换是整个对象替换，如原来有status字段，但在下面更新的时候，没写status，那么更新后的文档信息的status将会消失。

db.test\_collection.save({\_id:ObjectId(“xxxxxxxxx”),“name”:”hello”,”age”:25})

如上，把name为hello的文档age替换为25

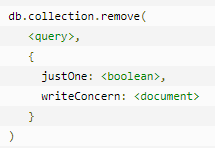
添加字段(multi:1代表新字段类型为double，2是string，以此类推)：

db.collection.update({条件},{$set:{新字段:值}},{multi:1})

删除字段：

db.collection.update({条件},{$unset:{字段:值}},{multi:1})

**删除文档（Document）**



query为条件，justOne为是否删除一条记录，writeConcern为异常等级

可以使用remove()方法，如

db.test\_collection.remove({“name”:”hello”},{justOne:false})

删除所有name为hello的文档

db.test\_collection.remove({})为删除集合下的所有信息

但新版目前一般使用deleteOne()和deleteMany()来进行删除，而不再使用remove()

**查询文档（Document）**

db.test\_collection.find(query,projection)

query为查询条件，projection为指定返回信息所要显示的健，1为显示，0为不显示，默认为显示

db.test\_collection.find({“name”:”hello”},{\_id:0})

如上，查询所有name为hello的信息，且不显示\_id字段



此时\_id字段没有显示

db.test\_collection.find({“name”:”hello”,”age”:{$lt:19}})

表示查询出name为helle并且年龄小于19的信息

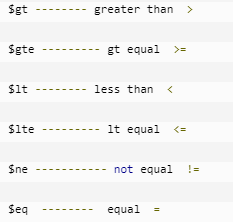
db.test\_collection.find({$or:[{“name”:”hello”},{“age”:{$lt:19}}]})

表示查询出name为hello或者age小于19的信息

db.test\_collection.find({“name”:”hello”,$or:[{“age”:{$lt:19}},{”address”:”beijing”}]})

表示查询出name为hello 并且（age小于19或地址为beijing）的信息

***条件操作符：***



***$type为类型操作符***:

db.test\_collection.find({“age”:{$type:2}})

表示查询出age字段类型为string的信息（1代表Double，2代表String，3…）

***limit():***

db.test\_collection.find().limit(2)

表示查询出前两条信息

***skip():***

db.test.collection.find().skip(2)

表示跳过前两条，从第三条开始显示

***sort():***

db.test\_collection.find().sort({“age”:1})

表示结果按age进行升序排列（-1为降序）

当limit() skip() sort()同时出现时，执行顺序是sort()skip()limit()

**索引（Index）**

***普通索引***：

db.test\_collection.createIndex({“name”:1})

表示在name上建立升序索引（-1为降序）

***复合索引：***

db.test\_collection.createIndex({“name”:1,”age”:-1})

表示在name上建立升序索引，age为降序索引的复合索引，单独使用第二个age作为查询条件时，没有使用索引，单独使用name或者，name与age同时使用时，都会使用复合索引(字段顺序可以颠倒)

***过期索引：***

db.test\_collection.createIndex({“time”:1},{ expireAfterSeconds:30})

在time上建立过期索引，30秒后自动删除

存储在过期索引字段的值必须是指定的时间类型，必须是ISODate或者ISODate数组，不能使用时间戳，否则不能自动删除。

过期索引不能是复合索引

***全文索引：***

建立全文索引db.posts.createIndex({key:"text"})，text为固定，key为健

使用全文索引，查询出key中包含runoob或者aaa的数据,这个包含是全包含，比如 可以查询出来key为 “runoob hfjkahj”这种，但是查询不出来“runoobsdsds”这种，search字段里的空格代表或

db.posts.find({$text:{$search:"runoob aa"}})

***地理索引：***

分为2种：2D平面地理位置索引 和 2D sphere 2D球面地里位置索引 2者的区别在于计算距离时使用的计算方式不同（平面距离还是球面距离）

db.collection.createIndex({key:”2d”})

db.collection.createIndex({key: '2dsphere'})

2D地理位置索引的取值范围以及表示方法 经纬度[经度,纬度]   
经纬度取值范围 经度[-180,180] 纬度[-90,90]

db.collection.insert({w:[180,90]})

使用$near 查询距离某个点最近的点 ,默认返回最近的100个点

db.collection.find({w:{$near:[x,y]}})

可以使用$maxDistance:x 限制返回的最远距离

db.collection.find({w:{$near:[x,y],$maxDistance:z}})

使用$geoWithin 查询某个形状内的点

查询矩形中的点

db.collection.find({w:{$geoWithin:{$box:[[0,0],[3,3]]}}})

查询圆中的点

db.collection.find({w:{$geoWithin:{$center:[[0,0],5]}}})

查询多边形中的点

db.collection.find({w:{$geoWithin:{$polygon:[[0,0],[0,1],[2,5],[6,1]]}}})

***索引的稀疏性***：

稀疏索引(或者称间隙索引)就是只包含有索引字段的文档的条目，即使索引字段包含一个空值。也就是说间隙索引可以跳过那些索引键不存在的文档。因为他并非包含所有的文档，因此称为稀疏索引。与之相对的非稀疏索引或者说普通索引则包含所有的文档以及为那些不包含索引的字段存储null值。

***创建间隙索引示例：***

db.addresses.createIndex( { "key": 1 }, { sparse: true } )

这个示例，哪些不包含key的键(列)的文档将不会被索引

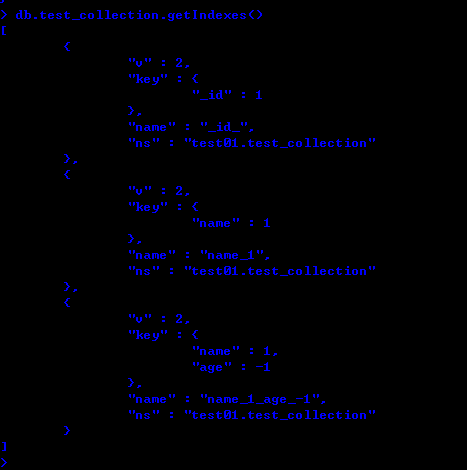
排序时稀疏索引将会失效，但可以使用hint(key:1)来强制使用,

稀疏索引主要使用的情况是：当建立的索引，如db.scores.createIndex( { key: 1 } , {unique: true} )指定了unique唯一性之后，不存在key字段的文档在整个集合中只能有一条，当开启了稀疏性之后，db.scores.createIndex( { key: 1 } , { sparse: true, unique: true } )，那么不存在key字段的文档将不应用索引，即可以有任意条不存在key字段的文档。

删除索引：

db.collection.dropIndex(“索引名”)

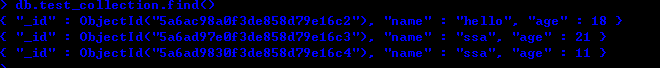
可以使用db.test\_collection.getIndexes()来查看索引

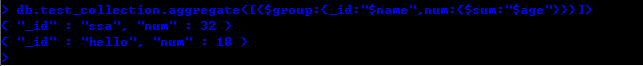


**聚合（aggregate）**

db.test\_collection.aggregate([{$group:{\_id:”$name”,num:{$sum:”$age”}}}])

表示把文档按照name分组后再分别求出每组的age之和显示,当\_id为空的时候，表示不进行分组





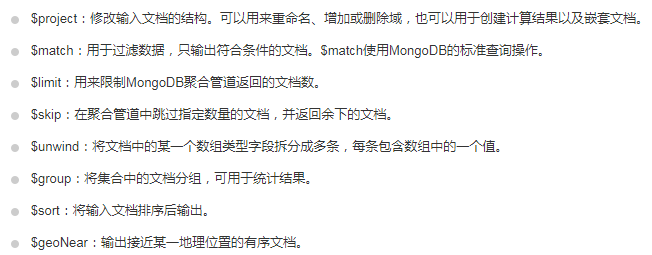


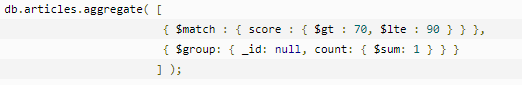
**管道**

管道在Unix和Linux中一般用于将当前命令的输出结果作为下一个命令的参数。

MongoDB的聚合管道将MongoDB文档在一个管道处理完毕后将结果传递给下一个管道处理。管道操作是可以重复的。

表达式：处理输入文档并输出。表达式是无状态的，只能用于计算当前聚合管道的文档，不能处理其它的文档。

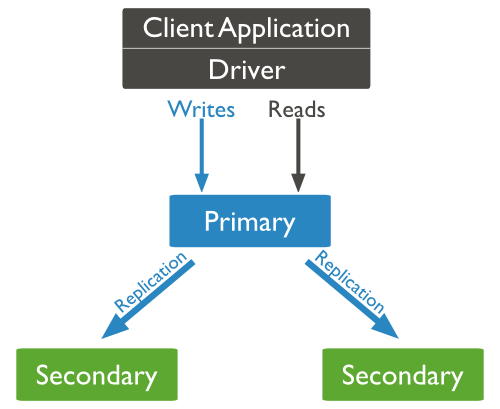




如图，表示先用$match找到分数大于70小于等于90的记录，然后把记录传递给$group管道，进行处理。

**副本集**

如下为mongodb副本集的结构示意图



mongodb的副本集，是指将数据同步到多个服务器，mongodb的复制至少需要两个节点，一个主节点，多个从节点，主要原理是：主节点会记录它的所有操作，记录在oplog中，副本集中的其他从节点会定期读取这些操作并在自身的数据副本中执行这些操作，从而保证主从节点数据的同步，

在mongodb中，当主节点失效时，其他子节点中会出现一个节点自动晋升为新的主节点，从而保证正常运转。

副本集建立过程：

1. 关闭正在运行的mongodb服务
2. 使用 mongod --port 端口 --dbpath 数据库位置 --replSet “副本集名称，同一个副本集名称要一致(关键)”来启动服务
3. 也可以通过配置文件的方式启动，在配置文件中，需要加入配置项

#master：默认为false，当设置为true，则配置当前实例作为主实例。

master = true

#slave：默认为false，当设置为true，则配置当前实例作为从实例。

slave = true

#source：默认为空，格式为：<host><:port>。用于从实例的复制：设置从的时候指定该选项会让从复制指定主的实例。

source = 127.0.0.1:30001

#only：默认为空，用于从选项，指定一个数据库进行复制。

only = abc          #只同步abc集合（库）

#slavedelay：设置从库同步主库的延迟时间，用于从设置，默认为0。

slavedelay = 60     #延迟60s同步主数据

#autoresync：默认为false，用于从设置。是否自动重新同步。设置为true，如果落后主超过10秒，会强制从自动重新同步。如果oplogSize太小，此设置可能有问题。如果OPLOG大小不足以存储主的变化状态和从的状态变化之间的差异，这种情况下强制重新同步是不必要的。当设置autoresync选项设置为false，10分钟内从不会进行大于1次的自动重新同步。

autoresync = false

1. mongod --port 27017 --dbpath “e:/mongodb/data” --replSet “rs0”

mongod --port 27018 --dbpath “e:/mongodb/data2” --replSet “rs0”

mongod --port 27019 --dbpath “e:/mongodb/data3” --replSet “rs0”

如上启动了三个mongodb服务，通过端口号不同，用来模拟三个服务器，

在第一个服务器的连接上，执行

rs.initiate({

\_id:”rs0”,

members:[

{\_id:0,host:”localhost:27017”},

{\_id:1,host:”localhost:27018”}

]

})

这样，就进行了副本集的初始化，因为是在27017上执行的，所以此时的主节点为27017，可以使用rs.isMaster()来查看节点信息。

1. 若要新增节点，则可以使用 rs.add(“host\_name:27019”)进行添加，注意，添加操作要在主节点上进行
2. 至此，副本集rs0就建立完毕，此时的所有节点的数据信息时一致的，注意子节点没有读写的权限,需要在字节点上使用rs.slaveOk()来开启读权限，只有在主节点失效后，才会有一个子节点变为主节点，并拥有读写的权限。可以使用

rs.isMaster()来查看当前节点是不是主节点，节点数至少为3个。

1. 可以通过rs.remove(“host\_name:端口”)来进行删除节点
2. 在linux上启动的时候，如果启动服务失败，可能是没有文件操作权限的问题 需要使用 sudo chmod –R 777 进行文件的授权后再启动，启动服务的时候使用sudo命令
3. 可以通过设置副本集的各个节点的priority，值越大，当主节点失效后，成为主节点的几率越高，可以使用如下命令进行修改

cfg = rs.conf()

cfg.members[0].priority = 0.5

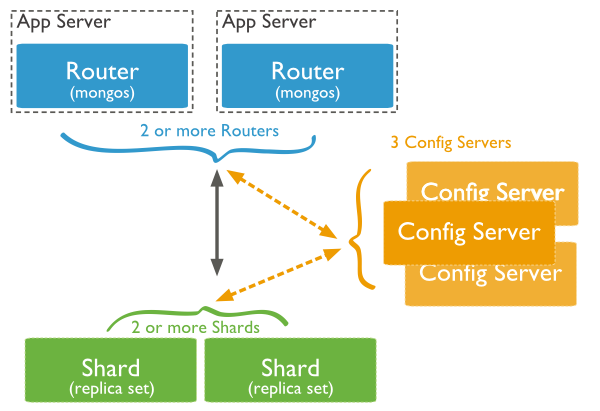
cfg.members[1].priority = 0.5

cfg.members[2].priority = 1

rs.reconfig(cfg)

**分片**

通过分片功能，可以把一个数据库，或者一个集合，分布到多个服务器上进行存储，通过负载均衡，使得每个数据库中存储大概相同数量的信息（这些信息不相同，相当于把同一个数据库拆分成不同的部分来存），如下图所示，每个shard代表着一个真正存储信息的数据集（每个shard也可以使用副本集来进行集群），config记录着数据和对应的片的关系，通过router来进行统一访问。



步骤：

1. 配置分片副本集rs1

启动27030和27031服务

mongod --port 27030 --dbpath=/data/db3 --logpath=/data/log/rs1-1.log --logappend --fork --shardsvr --replSet=rs1

mongod --port 27031 --dbpath=/data/db4 --logpath=/data/log/rs1-2.log --logappend --fork --shardsvr --replSet=rs1

在27030连接下初始化副本集rs1，此时27030为主节点

rs.initiate({\_id: 'rs1', members: [{\_id: 0, host: 'localhost:27030'}, {\_id: 1, host: 'localhost:27031'}]})

1. 创建config的副本集conf

启动27100和27101服务，作为config Server

mongod --port 27100 --dbpath=/data/conf1 --logpath=/data/log/conf-1.log --logappend --fork --configsvr --replSet=conf &

mongod --port 27101 --dbpath=/data/conf2 --logpath=/data/log/conf-2.log --logappend --fork --configsvr --replSet=conf &

在27100连接下初始化副本集conf，此时27100为主节点

rs.initiate({\_id: 'conf', members: [{\_id: 0, host: 'localhost:27100'}, {\_id: 1, host: 'localhost:27101'}]})

1. 创建路由

mongos --port 40000 --configdb conf/127.0.0.1:27100,127.0.0.1:27101 --fork --logpath=/data/log/route.log (在自己机器上测试的时候不要用localshot，用127.0.0.1,用localhost的话可能会报错)

1. 设置分片（此处的rs0副本集是在副本集模块那一部分就创建好的）

mongo localhost:40000

use admin

db.runCommand({ addshard: 'rs0/localhost:27020,localhost:27021'})

db.runCommand({ addshard: 'rs1/localhost:27030,localhost:27031'})

将test数据库进行分片负载均衡

db.runCommand({ enablesharding: 'test'})

将test数据库下的user集合进行分片

db.runCommand({ shardcollection: 'test.user', key: {name: 1}})

1. 可以通过sh.status()进行分片信息的查看
2. 3.4版本之后，shard可以使用副本集，也可以不使用，但config必须使用副本集的形式，shard服务启动的时候需要 --shardsvr，否则会出现在路由上没有办法进行相关增删改查的操作的情况，config服务器启动的时候，需要使用 --configsvr,否则mongos连接时会出现连接失败。

**备份**

mongodump –h dbhost –d dbname –o directory

将dbhost上的dbname数据库备份至directory位置上（在linux上操作时，要保证拥有对目的文件夹的写入权限，使用sudo chmod命令）

**Java简单应用**

<dependency>

<groupId>org.mongodb</groupId>

<artifactId>mongo-java-driver</artifactId>

<version>3.4.3</version>

</dependency>

首先添加依赖，然后

//建立连接

MongoClient mongoClient = **new** MongoClient("localhost",27017);

MongoDatabase database = mongoClient.getDatabase("test01");

//新增集合

database.createCollection("lalala\_collection");

MongoCollection<Document> collection = database.getCollection("user\_collection");

//新增文档

Document document = **new** Document("name","gsh").append("age", 18).append("status", "ok");

List<Document> list = **new** ArrayList<Document>();

list.add(document);

collection.insertMany(list);

//更新文档

collection.updateMany(Filters.*eq*("name","gsh"),

**new** Document("$set",**new** Document("age",20).append("status", "aha")));

//删除文档

collection.deleteOne(Filters.eq("name","ssd"));

//查询所有

FindIterable<Document> result = collection.find();

MongoCursor<Document> cursor = result.iterator();

**while**(cursor.hasNext()){

System.***out***.println(cursor.next().toJson());

}

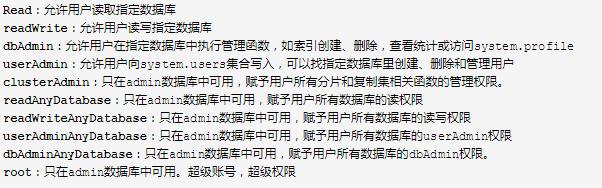
**权限开启**

1. 首先使用use admin进入到admin库中
2. 通过db.createUser({user:”用户名”,pwd:”密

码”,roles:[{role:”userAdminAnyDatabase”,db:”admin”}]})

1. 在配置文件中开启权限验证（auth=true，cfg文件则是security:authorization:enabled,要有换行，形式跟springboot的yml形式一样）
2. 重启mongodb服务
3. 进入admin库中，使用db.auth(’用户名’,’密码’)进行权限验证，成功验证后，会返回结果1
4. 进入test01库（自己创建的数据库）,同样使用createUser()方法创建一个普通用户，在授权角色时，可以使用read(只读),readWrite(读写)等给创建的用户授权，创建完成后，同样使用db.auth()命令进行授权，就可以根据所拥有的权限的不同，对数据库进行不同的操作。（如果你在admin下给test01库创建了一个普通用户，则对test01库授权的时候应该是在admin进行auth()操作，即在哪创建的用户，就在哪授权，此时如果在test01库下执行auth()会报错）

常用的role角色：



同一个用户可以拥有多个权限角色，也可以在多个库上建立相同的用户名的用户

可以使用show users来查询当前库下的用户信息，可以在admin库下使用

db.system.users.find()查看所有的用户信息

可以使用db.updateUser(“用户名”,{roles:[{role:”角色”,db:”test01”}]})来对用户的权限进行更新（覆盖掉原来的权限）

可以使用db.grantRolesToUser(“用户名”,[{role:”角色”,db:”test01”}])来进行追加权限

可以使用db.revokeRolesFromUser(“用户名”,[{role:”角色”,db:”test01”}])来收回权限

可以使用db.getUser(“用户名”)来查询某个用户的详细信息

可以使用db.dropUser(“用户名”)来删除用户，若返回true，则删除成功

***自定义role:***

内置角色只能控制User在DB级别上执行的操作，管理员可以创建自定义角色，控制用户在集合级别（Collection-Level）上执行的操作，即，控制User在当前DB的特定集合上执行特定的操作。

1. 在创建角色时，必须明确Role的四个特性：

* Scope：角色作用的范围，创建在Admin中的角色，能够在其他DB中使用；在其他DB中创建的角色，只能在当前DB中使用；
* Resource：角色控制的资源，表示授予在该资源上执行特定操作的权限；
* Privilege Actions：定义了User能够在资源上执行的操作，系统定义Action是：Privilege Actions；
* Inherit：角色能够继承其他角色权限

2.在admin 数据库中创建的角色，Scope是全局的，能够在admin，其他DB和集群中使用，并且能够继承其他DB的Role；而在非admin中创建的角色，Scope是当前数据库，只能在当前DB中使用，只能继承当前数据库的角色。

3. MongoDB的权限包由：资源（Resource）和操作（Action）两部分组成，Privilege Actions 定义User能够在资源上执行的操作，例如：MongoDB在文档级别（Document-Level）上执行的读写操作（Query and Write Actions）列表是：

* find
* insert
* remove
* update

4.创建一个自定义角色

db.createRole(

{

role: "new\_role",

privileges: [

{ resource: { cluster: true }, actions: [ "addShard" ] },

{ resource: { db: "config", collection: "" }, actions: [ "find", "update", "insert", "remove" ] },

{ resource: { db: "users", collection: "usersCollection" }, actions: [ "update", "insert", "remove" ] },

{ resource: { db: "", collection: "" }, actions: [ "find" ] }

],

roles: [

{ role: "read", db: "admin" }

]

}

)

5.在roles数组中，指定被继承的role，即，新建的new\_role从roles数组中继承权限：

* 如果被继承的role在当前DB中，定义的格式是：roles:["role"]；
* 如果被继承的role不在当前DB中，需要使用doc，指定该role所在的DB，定义的格式是：roles:[{role:"role\_name", db:"db\_name"}]；

db.createRole() ：Creates a role and specifies its privileges.

* db.updateRole() ：Updates a user-defined role.
* db.dropRole() ：Deletes a user-defined role.
* db.dropAllRoles() ：Deletes all user-defined roles associated with a database.
* db.grantPrivilegesToRole() ：Assigns privileges to a user-defined role.
* db.revokePrivilegesFromRole() ：Removes the specified privileges from a user-defined role.
* db.grantRolesToRole() ：Specifies roles from which a user-defined role inherits privileges.
* db.revokeRolesFromRole() ：Removes inherited roles from a role.
* db.getRole() ：Returns information for the specified role.
* db.getRoles() ：Returns information for all the user-defined roles in a database.

**赋予myrole3查询test01下book集合的权限：**

db.grantPrivilegesToRole("myrole3",[{resource:{db:"test01",collection:"book"},actions:["find"]}])

**收回myrole3查询test01下book集合的权限：**

db.revokePrivilegesFromRole("myrole3",[{resource:{db:"test01",collection:"book"},actions:["find"]}])

**收回myrole3继承的test01下的read角色：**

db.revokeRolesFromRole("myrole3",[{role:"read",db:"test01"}])

当创建角色的时候，指定的resource是数据库级别（即collcetion：“”），那么在收回权限的时候，不可以单独收回某个集合的权限，只能收回整个数据库的权限，若在授权时指定了collection，才可以针对某个集合进行权限收回。

**副本集开启权限验证(linux)：**

1. 创建一个密码文件(名字、内容、存放位置均自己决定) ，网上说使用openssl rand –base64来生成编码文件，但是这样会出现长度超出的错误，密码文件的内容在3.6版本中(其他版本未验证)，长度范围是6-1024。

2. 将密码文件通过scp命令复制到其他副本集的节点服务器上

scp xxx@host地址:源文件存放位置 本地存放位置,

3.将密码文件的权限修改为600（必须为600）

4. 先将主节点的副本集功能和安全验证功能在配置文件中关掉，然后重新启动服务器，按上面章节的方法，建立用户，成功后关闭服务器（这时可以再次打开安全验证先试一下用户名密码是否正确，我就出现了创建用户的时候打错了用户名密码，然后还没记住。。。）

5. 在没有打开安全验证的情况下，进行副本集的建立与配置，参考上面对应章节

6. 之后关闭所有服务器，开启权限验证，权限验证以yml格式的配置文件为例，需要在security下加上 keyFile: 密码文件位置

7. 按顺序依次启动节点，开启完成