link: null
title: 珠峰架构师成长计划
description: Windows官方安装指南
keywords: null
author: null
date: null
publisher: 珠峰架构师成长计划
stats: paragraph=296 sentences=268, words=1342

1. 什么是MongoDB

- MongoDB是一个基于分布式文件存储的开源数据库系统
- MongoDB 将数据存储为一个文档，数据结构由键值(key=>value)对组成。MongoDB 文档类似于 JSON 对象。字段值可以包含其他文档，数组及文档数组。

2. MongoDB安装

2.1 windows安装

Windows官方安装指南 (https://www.mongodb.org/downloads)

- mongodb32位安装版 链接: https://pan.baidu.com/s/1SHJ1vre_CQOE3u-W0zniqQ (https://pan.baidu.com/s/1SHJ1vre_CQOE3u-W0zniqQ) 密码: chan
- MongoDB64位绿色版 链接: https://pan.baidu.com/s/1EkAB2SrcU1mfMfff_WDxtA (https://pan.baidu.com/s/1EkAB2SrcU1mfMfff_WDxtA) 密码: w913
- mongo客户端 链接: https://pan.baidu.com/s/1YFxLZ-55D-WFR8os2fXN0A (https://pan.baidu.com/s/1YFxLZ-55D-WFR8os2fXN0A) 密码: 61qd

2.2 mac安装

Mac官方安装指南 (https://docs.mongodb.com/manual/tutorial/install-mongodb-on-os-x/)

1. 先安装homebrew

```
http://brew.sh/
```

2. 使用brew安装mongodb

```
brew install mongodb
```

3. 再安装可视化工具 Robomongo (https://robomongo.org/)

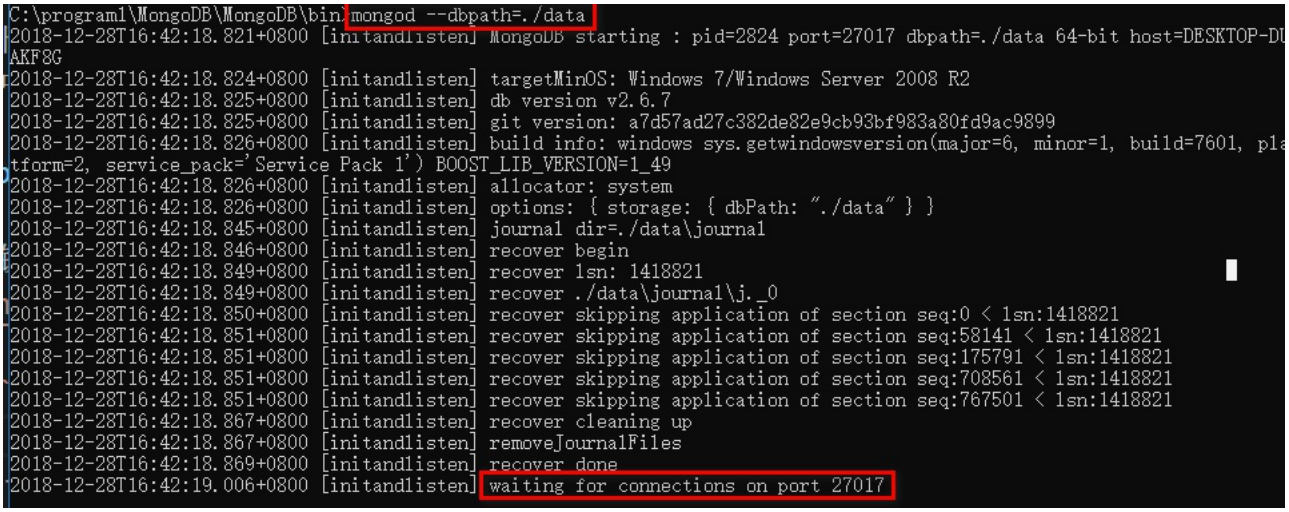
3. mongodb启动与连接

3.1 windows启动服务器端

1. 找到mongodb安装目录,一般是 `C:\Program Files\MongoDB 2.6 Standard\bin`
2. 按下Shift+鼠标右键,选择在此处打开命令窗口
3. 在除C盘外的盘符新建一个空目录,如 `D:\Mongodb\data`
4. 在命令行中输入 `mongod --dbpath=&#x521A;&#x521B;&#x5EFA;&#x7684;&#x7A7A;&#x76EE;&#x5F55;`,如

```
mongod --dbpath=D:\Mongodb\data
```

5. 再按回车键



- 如果出现 `waiting for connections on port 27017`就表示 **启动成功**,已经在**27017**端口上监听了客户端的请求
- 注意： `--dbpath`后的值表示数据库文件的存储路径,而且后面的路径必须事先创建好，必须已经 **存在**，否则服务开启失败
- 注意：这个命令窗体绝对 **不能关**,关闭这个窗口就相当于停止了 `mongodb`服务

4. 添加到window服务

- 以管理员身份运行命令
- logfile是一个不存在的文件名，而非目录名

```
mongod.exe --logpath C:\program1\MongoDB\bin\log\logfile --logappend --dbpath C:\program1\MongoDB\bin\data --serviceName MongoDB --install
```

4. MongoDB基本概念

- **数据库** MongoDB的单个实例可以容纳 **多个** 独立的数据库，比如一个学生管理系统就可以对应一个数据库实例
- **集合** 数据库是由集合组成的,一个集合用来表示一个 **实体**,如学生集合
- **文档** 集合是由文档组成的，一个文档表示一条 **记录**,比如一位同学张三就是一个文档

Mongodb	mysql
文档(document) (单个文档最大16M)	记录(row)
集合(collection)	表(table)
数据库(database) （32位系统上，一个数据库的文件大小不能超过2G）	数据库(database)

5. 数据库操作

5.1 使用数据库

语法

```
use database_name
```

- database_name 代表数据库的名字
- 注：如果此数据库存在，则切换到此数据库下,如果此数据库还不存在也可以切过来,但是并不能立刻创建数据库

切换到 school数据库下

```
use school
```

5.2 查看所有数据库

语法

```
show dbs
```

- 备注: 我们刚创建的数据库 school如果不在列表内， 要显示它，我们需要向 school数据库插入一些数据

```
db.students.insert({name:'zfpx',age:1});
```

5.3 查看当前使用的数据库

语法

```
db
```

- 注：db代表的是当前数据库 也就是school这个数据库

5.4 删除数据库

语法

```
db.dropDatabase()
```

6. 集合操作

6.1 查看集合帮助

语法

```
db.students.help();
```

6.2 查看数据库下的集合

```
show collections
```

6.3 创建集合

6.3.1 创建一个空集合

```
db.createCollection(collection_Name)
```

- collection_Name集合的名称

6.3.2 创建集合并插入一个文档

- collection_Name集合的名称
- document要插入的文档

```
db.collection_Name.insert(document)
```

7. 插入文档

7.1 insert

```
db.collection_name.insert(document);
`
```

- collection_name 集合的名字
- document 插入的文档

> 每当插入一条新文档的时候mongodb会自动为此文档生成一个 _id属性,_id一定是唯一的，用来唯一标识一个文档 _id也可以直接指定，但如果数据库中此集合下已经有此_id的话插入会失败

```
db.students.insert({_id:1,name:'zfpx',age:1});
WriteResult({ "nInserted" : 1 })
db.students.insert({_id:1,name:'zfpx',age:1});
```

7.2 save

```
db.collection_name.save(document)
```

- collection_name 集合的名字
- document 插入的文档

> 注：如果不指定 _id 字段 save() 方法类似于 insert() 方法。如果指定 _id 字段，则会更新该 _id 的数据。

```
> db.students.save({_id:1,name:'zfpx',age:1});
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 0 })
> db.students.save({_id:1,name:'zfpx',age:100});
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
```

8. 更新文档

8.1 语法

```
db.collection.update(
   ,
   ,
   {
     upsert: ,
     multi:
   }
)
```

8.2 参数

- query 查询条件,指定要更新符合哪些条件的文档
- update 更新后的对象或指定一些更新的操作符
    - $set直接指定更新后的值
    - $inc在原基础上累加
- upsert 可选，这个参数的意思是，如果不存在符合条件的记录时是否插入updateObj. 默认是false,不插入。
- multi 可选，mongodb 默认只更新找到的第一条记录，如果这个参数为true,就更新所有符合条件的记录。

8.3 upsert

将 students集合中数据中 name是zfpx2的值修改为zfpx22

```
> db.students.insert({_id:1,name:'zfpx1'});
WriteResult({ "nInserted" : 1 })
> db.students.update({_id:2},{name:'zfpx2'},{upsert:true});
WriteResult({ "nMatched" : 0, "nUpserted" : 1, "nModified" : 0, "_id" : 2 })
```

8.4 multi

- 如果有多条name是zfpx2的数据只更新一条,如果想全部更新需要指定 {multi:true}的参数

```
db.students.update({name:'zfpx2'},{$set:{age:10}},{multi:true});
WriteResult({ "nMatched" : 2, "nUpserted" : 0, "nModified" : 2 });
```

9. 更新操作符

9.1 $set

直接指定更新后的值

```
db.c3.update({name:'zfpx2'},{$set:{age:10}},{multi:true});
```

9.2 $inc

在原基础上累加

```
db.c3.update({name:'zfpx2'},{$inc:{age:1}},{multi:true});
```

9.3 $unset

删除指定的键

```
db.c3.update({name:'zfpx2'},{$unset:{age:1}},{multi:true});
```

9.4 $push

向数组中添加元素

```
var result = db.student.update({name:'张三'},{
    $push:{"hobbys":"smoking"}
});
```

9.5 $ne

$ne类似于MYSQL的 not in 或者 not exists

```
db.student.update({name:'zfpx1',hobbys:{$ne:'smoking'}},{$push:{"hobbys":"smoking"}});
```

9.5 $addToSet

向集合中添加元素

```
db.student.update({name:'zfpx1'},{$addToSet:{"hobbys":"smoking"}});
```

9.6 $pull

- 向集合中删除元素

```
db.student.update({name:'zfpx1'},{$pull:{"hobbys":"smoking"}});
```

9.7 $each

把数组中的元素逐个添加到集合中

```
var hobbys = ["A",'B'];
db.student.update({name:'zfpx1'},{$addToSet:{hobbys:{$each:hobbys}}});
```

9.8 $pop

从数组中移除指定的索引中对应的元素

```
db.student.update({name:'zfpx1'},{$pop:{hobbys:1}});
```

9.9 修改指定索引元素

```
db.c3.update({name:'zfpx1'},{$set:{"hobbys.0":"smoking2"}});
```

10. 文档的删除

remove方法是用来移除集合中的数据

10.1 语法

```
db.collection.remove(
   ,
   {
     justOne:
   }
)
```

10.2 参数

- query：（可选）删除的文档的条件。
- justOne：（可选）如果设为 true 或 1，则只删除匹配到的多个文档中的第一个

10.3 实例

删除worker集合里name是zfpx2的所有文档数据

```
> db.students.remove({name:'zfpx2'});
WriteResult({ "nRemoved" : 2 })
```

即使匹配多条也只删除一条

```
> db.students.remove({name:"zfpx2"},{justOne:true})
WriteResult({ "nRemoved" : 1 })
```

11. 查询文档

11.1 find

语法

```
db.collection_name.find()
```

参数

- collection_name 集合的名字

实例 查询students下所有的文档

```
db.students.find()
```

11.2 查询指定列

语法

```
db.collection_name.find({queryWhere},{key:1,key:1})
```

参数列表

- collection_name 集合的名字
- queryWhere 参阅查询条件操作符
- key 指定要返回的列
- 1 表示要显示

实例 只返回显示age列

```
> db.students.find({},{age:1});
```

11.3 findOne

查询匹配结果的第一条数据 **语法**

```
db.collection_name.findOne()
```

实例

```
db.students.findOne()
```

11.4 $in

查询字段在某个范围内

```
db.student.find({age:{$in:[30,100]}},{name:1,age:1});
```

11.5 $nin

查询字段不在某个范围内

```
db.student.find({age:{$nin:[30,100]}},{name:1,age:1});
```

11.6 $not

对特定条件取反

```
db.student.find({age:{$not:{$gte:20,$lte:30}}});
```

11.7 array

对数组的查询

```
$slice:["$array", [startIndex, ] length ] （startIndex可以省略，默认从0开始）
"friends" : [ "A", "B" ] }   "friends" : [ "C", "D" ]
db.stu.find({},{friends:{$slice:[0,3]}});  "friends" : [ "A", "B", "C" ]
```

11.8 where

```
db.student.find({$where:"this.age>30"},{name:1,age:1});
```

11.9 cursor

- 游标不是查询结果，而是查询的一个返回资源或者接口，通过这个接口，可以逐条读取数据

```
var result = db.student.find();
```

12. 条件操作符

条件操作符用于比较两个表达式并从mongoDB集合中获取数据

12.1 大于操作符

语法

```
db.collectoin_name.find({<key>:{$gt:<value>}})
</value></key>
```

参数

- collectoin_name 集合名称
- key 字段
- value 值

查询 age 大于 30的数据

```
db.students.find({age:{$gt:30}})
```

12.2 大于等于操作符

语法

```
db.collectoin_name.find({:{$gte:<value>}})
```

参数

- collectoin_name 集合名称
- key 字段
- value 值

查询age 3大于等于30 的数据

```
db.students.find({age: {$gte: 30}})
```

12.3 小于操作符

语法

```
db.collectoin_name.find( {:{$lt:<value>}})
```

参数

- collectoin_name集合名称
- key 字段
- value 值

实例

```
db.students.find({age: {$lt: 30}}) 查询age 小于30的数据
```

12.4 小于等于操作符

语法

```
db.collectoin_name.find({<key>:{$lte:<value>}})
</value></key>
```

参数

- collectoin_name集合名词
- key字段
- value值

查询age 小于等于30的数据

```
db.students.find({age: {$lte: 30}})
```

12.5 同时使用 $gte和$lte

语法

```
db.collectoin_name.find({:{$gte:},:{$lte:}})
```

参数

- collectoin_name 集合名称
- key 字段
- value 值

实例 查询age 大于等于 30 并且 age 小于等于 50 的数据

```
db.students.find({age: {$gte: 30, $lte: 50}})
```

12.6 等于

语法

```
db.collectoin_name.find({:,:})
```

参数

- collectoin_name集合名词
- key字段
- value值

查询age = 30的数据

```
db.students.find({"age": 30})`
```

12.7 使用 _id进行查询

语法

```
db.collectoin_name.find({"_id" : ObjectId("value")})
```

参数

- value _id的值

实例 查询_id是 562af23062d5a57609133974 数据

```
> db.students.find({_id:ObjectId("5adb666ecd738e9771638985")});
{ "_id" : ObjectId("5adb666ecd738e9771638985"), "name" : "zzzz" }
```

12.8 查询结果集的条数

语法

```
db.collectoin_name.find().count()
```

参数

- collectoin_name 集合名称

实例

```
db.students.find().count()
```

12.9 正则匹配

语法

```
db.collection.find({key:/value/})
```

参数

- collectoin_name 集合名称
- key 字段
- value 值

实例 查询name里包含zhang的数据

```
db.students.find({name:/value/})
```

查询某个字段的值当中是否以另一个值开头

```
db.students.find({name:/^zhang/})
```

13. 与和或

13.1 and

find方法可以传入多个键(key)，每个键(key)以逗号隔开

语法

```
db.collection_name.find({key1:value1, key2:value2})
```

实例 查询name是zfpx并且age是1的数据

```
db.students.find({name:'zfpx',age:1})
```

13.2 or

语法

```
db.collection_name.find(
   {
      $or: [
   {key1: value1}, {key2:value2}
      ]
   }
)
```

实例 查询age = 30 或者 age = 50 的数据

```
db.students.find({$or:[{age:30},{age:50}]})
```

13.3 and和or联用

语法

```
db.collection_name.find(
   {
     key1:value1,
     key2:value2,
     $or: [
   {key1: value1},
   {key2:value2}
     ]
   }
)
```

实例 查询 name是zfpx 并且 age是30 或者 age是 50 的数据

```
db.students.find({name:'zfpx',$or:[{age:30},{age:50}]})
```

14. 分页查询

14.1 limit

读取指定数量的数据记录 **语法**

```
db.collectoin_name.find().limit(number)
```

参数

- collectoin_name集合

- number读取的条数

实例 查询前3条数据

```
db.students.find().limit(3)
```

14.2 skip

跳过指定数量的数据，skip方法同样接受一个数字参数作为跳过的记录条数 **语法**

```
db.collectoin_name.find().skip(number)
```

参数

- collectoin_name集合
- number跳过的条数

实例 查询3条以后的数据

```
db.students.find().skip(3)
```

14.3 skip+limit

通常用这种方式来实现分页功能 **语法**

```
db.collectoin_name.find().skip(skipNum).limit(limitNum)
```

参数

- collectoin_name 集合名称
- skipNum 跳过的条数
- limitNum 限制返回的条数

实例 查询在4-6之间的数据

```
db.students.find().skip(3).limit(3);
```

14.4 sort排序

sort()方法可以通过参数指定排序的字段，并使用 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。 **语法**

```
db.collectoin_name.find().sort({key:1})
db.collectoin_name.find().sort({key:-1})
```

参数

- collectoin_name集合
- key表示字段

实例 查询出并升序排序 {age:1} age表示按那个字段排序 1表示升序

```
db.students.find().sort({age:1})
```

15. 执行脚本

```
var username = 'zfpx';
var password = '123456';
var user = { "username": username, "password": password };
var db = connect('students');
var result = db.users.insert(user);
print('write ' + result);
```

```
var start = Date.now();
var db = connect('students');
for (var i = 0; i < 1000; i++) {
    db.users.insert({ "username": "zfpx" + i });
}
var cost = Date.now() - start;
print('cost ' + cost + ' ms');
```

```
var start = Date.now();
var db = connect('students');
var users = [];
for (var i = 0; i < 1000; i++) {
    users.push({ "username": "zfpx" + i });
}
db.users.insert(users);
var cost = Date.now() - start;
print('cost ' + cost + ' ms');
```

在命令行中执行

```
script>mongo 1.js
MongoDB shell version: 2.6.7
connecting to: test
connecting to: students
write WriteResult({ "nInserted" : 1 })
```

16. 备份与导出

```
mongodump
    -- host 127.0.0.1
    -- port 27017
    -- out D:/databack/backup
    -- collection mycollection
    -- db test
    -- username
    -- password

mongorestore
--host
--port
--username
--password
```

17. 权限

17.1 创建用户

- 使用 use admin进入我们的admin库
- 使用 db.createUser方法来创建集合

```
db.createUser({
    user:'zfpx',
    pwd:'123456',
    customData:{
        name:'zhufengpeixun',
        email:'zhufengpeixun@126.com',
        age:9
    },
    roles:[
        {
          role:'readWrite',
          db:'school'
        },
        'read'
    ]
});
```

17.2 查询用户

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

17.3 删除用户

```
db.system.users.remove({user:'zfpx'});
```

17.4 启动数据库权限检查

```
mongod --auth
mongo  -u zfpx -p 123456 127.0.0.1:27017/admin
```

17.5 鉴权

```
use admin;
db.auth('zfpx','zfpx');
```

- 正确返回1，如果错误返回0

18. 索引

18.1 准备数据

```
var db = connect('school');
var users = [];
for(var i=0;i<20;i++){
    users.push({_id:i,name:'zfpx'+i});
}
print(users.length);
db.users.insert(users);
```

18.2 打印出查询时间

```
var startTime = Date.now();
var  db = connect('school');
var  records=db.users.find({name:"zfpx100"});
records.forEach(function(item){printjson(item)});
print(Date.now() - startTime);
```

18.3 建立索引

```
db.users.ensureIndex({name:1});
```

19. 附录

19.1 ObjectId构成

之前我们使用MySQL等关系型数据库时，主键都是设置成自增的。但在分布式环境下，这种方法就不可行了，会产生冲突。为此，MongoDB采用了一个称之为ObjectId的类型来做主键。ObjectId是一个12字节的BSON 类型字符串。按照字节顺序，一次代表：

- 4字节：UNIX时间戳
- 3字节：表示运行MongoDB的机器
- 2字节：表示生成此_id的进程
- 3字节：由一个随机数开始的计数器生成的值

19.2 Mongodb启动命令mongod参数说明

选项 含义 --port 指定服务端口号，默认端口27017 --logpath 指定MongoDB日志文件，注意是指定文件不是目录 --logappend 使用追加的方式写日志 --dbpath 指定数据库路径 --directoryperdb 设置每个数据库将被保存在一个单独的目录

19.3 集合命令

- db.students.help();
- DBCollection help
- db.students.find().help() - show DBCursor help 显示游标帮助
- db.students.count() 显示条数
- db.students.copyTo(newColl) - duplicates collection by copying all docum ents to newColl; no indexes are copied. 把一个旧集合拷贝到一个新的集合，不拷贝索引
- db.students.convertToCapped(maxBytes) - calls {convertToCapped:'students', size:maxBytes}} command
- db.students.dataSize() 数据大小
- db.students.distinct(key) - e.g. db.students.distinct('x') 统计唯一的key的数量
- db.students.drop() drop the collection ，删除集合
- db.students.dropIndex(index) - e.g. db.students.dropIndex("indexName") 删除索引 or db.students.dropIndex({ "indexKey" : 1 })
- db.students.dropIndexes() 删除 所有的索引
- db.students.ensureIndex(keypattern[,options]) - options is an object wit h these possible fields: name, unique, dropDups 添加索引
- db.students.reIndex()
- db.students.find([query],[fields]) - query is an optional query filter. fields is optional set of fields to return. 查找文档

```
    e.g. db.students.find( {x:
```

77} , {name:1, x:1})

- db.students.find(...).count() 数量

- db.students.find(...).limit(n) 限制返回的条数
- db.students.find(...).skip(n) 设置跳过的条数
- db.students.find(...).sort(...) 排序
- db.students.findOne([query]) 查找一条
- db.students.findAndModify({ update : ... , remove : bool [, query: {}, sort: {}, 'new': false] }) 查找并且修改 更新后的值，是否删除，查询条件 排序 是否返回新值
- db.students.getDB() get DB object associated with collection 获得DB
- db.students.getPlanCache() get query plan cache associated with collecti on
- db.students.getIndexes() 获取索引
- db.students.group({ key : ..., initial: ..., reduce : ...[, cond: ...] }) 分组统计
- db.students.insert(obj) 插入文档
- db.students.mapReduce(mapFunction , reduceFunction ,) 统计
- db.students.aggregate([pipeline],) - performs an agg regation on a collection; returns a cursor 聚合
- db.students.remove(query) 删除
- db.students.renameCollection(newName ,) renames the colle ction. 重命名集合
- db.students.runCommand(name ,) runs a db command with the gi ven name where the first param is the collection name
- db.students.save(obj) 保存对象
- db.students.stats() 统计信息
- db.students.storageSize() - includes free space allocated to this collec tion
- db.students.totalIndexSize() - size in bytes of all the indexes
- db.students.totalSize() - storage allocated for all data and indexes
- db.students.update(query, object[, upsert_bool, multi_bool]) - instead o f two flags, you can pass an object with fields: upsert, multi 更新
- db.students.validate( ) - SLOW
- db.students.getShardVersion() - only for use with sharding
- db.students.getShardDistribution() - prints statistics about data distri bution in the cluster
- db.students.getSplitKeysForChunks( ) - calculates split p oints over all chunks and returns splitter function
- db.students.getWriteConcern() - returns the write concern used for any o perations on this collection, inherited from server/db if set
- db.students.setWriteConcern( ) - sets the write conc ern for writes to the collection
- db.students.unsetWriteConcern( ) - unsets the write concern for writes to the collection

20. 用户和权限

20.1 角色

20.1.1 数据库用户角色

针对每一个数据库进行控制。

- read :提供了读取所有非系统集合，以及系统集合中的system.indexes, system.js, system.namespaces
- readWrite: 包含了所有read权限，以及修改所有非系统集合的和系统集合中的system.js的权限

20.1.2 数据库管理角色

每一个数据库包含了下面的数据库管理角色。

- dbOwner：该数据库的所有者，具有该数据库的全部权限。
- dbAdmin：一些数据库对象的管理操作，但是没有数据库的读写权限。（参考：http://docs.mongodb.org/manual/reference/built-in-roles/#dbAdmin） (http://docs.mongodb.org/manual/reference/built-in-roles/#dbAdmin）)
- userAdmin：为当前用户创建、修改用户和角色。拥有userAdmin权限的用户可以将该数据库的任意权限赋予任意的用户。

20.1.3 集群管理权限

- admin数据库包含了下面的角色，用户管理整个系统，而非单个数据库。这些权限包含了复制集和共享集群的管理函数。
- clusterAdmin：提供了最大的集群管理功能。相当于clusterManager, clusterMonitor, and hostManager和dropDatabase的权限组合。
- clusterManager：提供了集群和复制集管理和监控操作。拥有该权限的用户可以操作config和local数据库（即分片和复制功能）
- clusterMonitor：仅仅监控集群和复制集。
- hostManager：提供了监控和管理服务器的权限，包括shutdown节点，logrotate, repairDatabase等。 备份恢复权限：admin数据库中包含了备份恢复数据的角色。包括backup、restore等等。

20.1.4 所有数据库角色

- admin数据库提供了一个mongod实例中所有数据库的权限角色：
- readAnyDatabase：具有read每一个数据库权限。但是不包括应用到集群中的数据库。
- readWriteAnyDatabase：具有readWrite每一个数据库权限。但是不包括应用到集群中的数据库。
- userAdminAnyDatabase：具有userAdmin每一个数据库权限，但是不包括应用到集群中的数据库。
- dbAdminAnyDatabase：提供了dbAdmin每一个数据库权限，但是不包括应用到集群中的数据库。

20.1.5 超级管理员权限

- root: dbadmin到admin数据库、useradmin到admin数据库以及UserAdminAnyDatabase。但它不具有备份恢复、直接操作system.*集合的权限，但是拥有root权限的超级用户可以自己给自己赋予这些权限。

20.1.6 备份恢复角色

backup、restore；

20.1.7 内部角色

__system

20.2 权限配置

- 数据库用户角色 read readWrite
- 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
- 所有数据库角色 readAnyDatabase readWriteAnyDatabase userAdminAnyDatabase dbAdminAnyDatabase
- 超级用户角色 **root**
- mongod.conf (/etc/mongod.conf) 配置文件

20.2.1 常见命令

```
ps -ef | grep mongo
/usr/bin/mongod -f /etc/mongod.conf
systemctl restart mongod.service
```

20.2.2 创建超级管理员

```
show users
```

20.2.3 创建用户

```
use admin
db.createUser({
  user:'admin',
  pwd:'123456',
  roles:[{role:'root',db:'admin'}]
});
```

20.2.4 管理员登录

```
mongo admin -u admin -p 123456
```

20.2.5 给每个库创创建一个自己的管理员

```
use question;
db.createUser({
  user:'questionadmin',
  pwd:'123456',
  roles:[{role:'dbOwner',db:'question'}]
});
```

20.2.6 删除用户

```
db.dropUser("admin")
db.dropAllUser();
```

20.2.7 更新用户

```
db.updateUser('admin',{pwd:'password'})
```

20.2.8 密码认证

```
db.auth('admin','password');
```