# 一 MongoDB的简介

## 1 MongoDB简介

MongoDB 是一个基于分布式、文件存储的NoSQL数据库。由C++语言编写，旨在为 WEB 应用提供可扩展的高性能数据存储解决方案。特点如下：

模式自由 : 可以把不同结构的文档存储在同一个数据库里

面向集合的存储：适合存储 JSON风格文件的形式

完整的索引支持：对任何属性可索引

复制和高可用性：支持服务器之间的数据复制，支持主-从模式及服务器之间的相 互复制，复制的主要目的是提供冗余及自动故障转移

自动分片： 支持云级别的伸缩性。自动分片功能支持水平的数据库集群，可 动态添加额外的机器

丰富的查询： 支持丰富的查询表达方式，查询指令使用JSON形式的标记，可 轻易查询文档中的内嵌的对象及数组

快速就地更新： 查询优化器会分析查询表达式，并生成一个高效的查询计划

高效存储方式： 支持二进制数据及大型对象（如照片或图片）

## 2 MongoDB术语

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、文档数组。

三元素：数据库，集合，文档。

集合： 类似于关系数据库中的表，储存多个文档，结构不固定，如可以存储如下文档 在一个集合中。

{'name':'huangrong','age':18}

{'book':'shuihuzhuan','heros':'108'}

文档： 对应着关系数据库中的行，即一个对象，由键值对构成，是json的扩展Bson 形式 {'name':'guojing','gender':'男'}

SQL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

column field 数据字段/域

index index 索引

table joins 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

## 3 数据类型

Object ID： 每个文档都有的唯一文档标识：\_id，默认Mongo会为每个文档 提供一个独特的\_id，类型为objectID，也可以自己设置\_id插入文档。

String： 字符串，最常用，必须是有效的UTF-8

Boolean： 存储一个布尔值，true或false

Integer： 整数可以是32位或64位，这取决于服务器

Double： 存储浮点值

Arrays： 数组或列表，多个值存储到一个键

Object： 用于嵌入式的文档，即一个值为一个文档

Null： 存储Null值

Timestamp： 时间戳

Date： 存储当前日期或时间的UNIX时间格式

注意：objectID是一个12字节的十六进制数，前4个字节为当前时间戳，接下来3个字节的机器ID，接下来的2个字节中MongoDB的服务进程id，最后3个字节是简单的增量值。

# 二 MongoDB 基本使用

## 0 开启并链接服务

开启服务端：**mongod --dbpath c:\data #data是 数据文件存放文件夹名**

客户端连接：mongo #一定要在新的cmd中输入

导入数据： mongoimport

## 1 数据库操作

show dbs 查看所有数据库列表

use 数据库名 使用/创建数据库。如果是创建，此时并未创建，插入数据后才算。

db.studetn.insert({"name":"lisi"});

student是一个未创建的集合名，系统会自动创建该集合。

show collections 查询当前所有的集合。

db 查看当前数据库名称

db.stats() 查看当前数据库信息

exit 终端退出连接，或ctrl+c

db.dropDatabase(); 删除当前所在的数据库，如果数据库不存在，则什么也不做

## 2 集合操作

db.createCollection(name, options)

name 集合名

options 文档，参数是可选的，所以只需要到指定的集合名称。

db.createCollection("stu") //不限制集合大小

db.createCollection("sub", { capped : true, size : 10 } ) //限制集合大小

参数capped： 默认值为false表示不设置上限，值为true表示设置上限

参数size： 当capped值为true时，需要指定此参数，表示上限大小，当文档达 到上限时，会将之前的数据覆盖，单位为字节。

show collections 查看当前数据库的集合

db.集合名称.drop() 删除

## 3 CRUD

### 3.1 插入数据 insert

db.集合名.insert(document)

插入文档时，如果不指定\_id参数，MongoDB会为文档分配一个唯一的ObjectId

例1： db.stu.insert({name:'gj',gender:1})

例2： s1={\_id:'20160101',name:'hr'}

s1.gender=0

db.stu.insert(s1)

插入数据，随着数据的插入，相应的数据库、集合也对应创建成功。

我们不可能一条一条的insert，如果希望用sublime在外部写好数据库的形式，然后导入数据库：

mongoimport --db test --collection restautants --drop --file primier -dataset.json

**-db test 想往哪个数据库里面导入**

**--collection restaurants 想往哪个集合中导入**

**--drop 把原有集合清空**

**--file primer-dataset.json 哪个文件**

**这样，我们就能用sublime创建一个json文件，然后用mongoimport命令导入，这样学习数据库非常方便。**

### 3.2 删除数据 remove

db.集合名称.remove(<query>,{justOne: <boolean>})

参数query:可选，删除的文档的条件

参数justOne:可选，如果设为true或1，则只删除一条，默认false，表示删除多条

例1： 只删除匹配到的第一条

db.stu.remove({gender:0},{justOne:true})

例2： 全部删除

db.stu.remove({})

### 3.3 修改数据 update

db.集合名称.update(<query>,<update>,{multi: <boolean>})

参数query:查询条件，类似sql语句update中where部分

参数update:更新操作符，类似sql语句update中set部分

参数multi:可选，默认是false，表示只更新找到的第一条记录，值为true表示把满 足条件的文档全部更新

例1： 全文档更新

db.stu.update({name:'hr'},{name:'mnc'})

例2： 指定属性更新，通过操作符$set

db.stu.insert({name:'hr',gender:0})

db.stu.update({name:'hr'},{$set:{name:'hys'}})

例3： 修改多条匹配到的数据

db.stu.update({},{$set:{gender:0}},{multi:true})

### 3.4 保存数据 save

db.集合名称.save(document)

如果文档的\_id已经存在则修改，如果文档的\_id不存在则添加

db.stu.save({\_id:'20160102','name':'yk',gender:1})

db.stu.save({\_id:'20160102','name':'wyk'})

# 三 查询find

## 1 查询语法

列出所有文档--find中没有参数 db.student.find()

精确匹配： db.student.find({'name':'zs'});

db.集合名称.find({条件},{要返回的键}) 条件为空，则查询全部，要返回的键是可选的，

Mongo默认返回id，如果不需要id，那么 在返回的键里书写 \_id=0，需要的键值为1

db.集合名称.findOne({条件文档}) 查询单个符合要求的文档

db.集合名称.find({条件文档}).pretty() 结果格式化

## 2 比较运算符

比较运算符：等于，默认是等于判断，没有运算符

小于 $lt

小于或等于 $lte

大于 $gt

大于或等于 $gte

不等于 $ne

例1：查询名称等于'gj'的学生

db.stu.find({name:'gj'})

例2：查询年龄大于或等于18，且小于25的学生

db.stu.find({age:{$gte:18,$lt:30}})

## 3 逻辑云算符

逻辑运算符：查询时可以有多个条件，多个条件之间需要通过逻辑运算符连接

逻辑与：默认是逻辑与的关系

例3：查询年龄大于或等于18，并且性别为1的学生

db.stu.find({age:{$gte:18},gender:1})

逻辑或：使用$or

例4：查询年龄大于18，或性别为0的学生

db.stu.find({$or:[{age:{$gt:18}},{gender:1}]})

and和or一起使用

例5：查询年龄大于18或性别为0的学生，并且学生的姓名为gj

db.stu.find({$or:[{age:{$gte:18}},{gender:1}],name:'gj'})

## 4 范围运算符

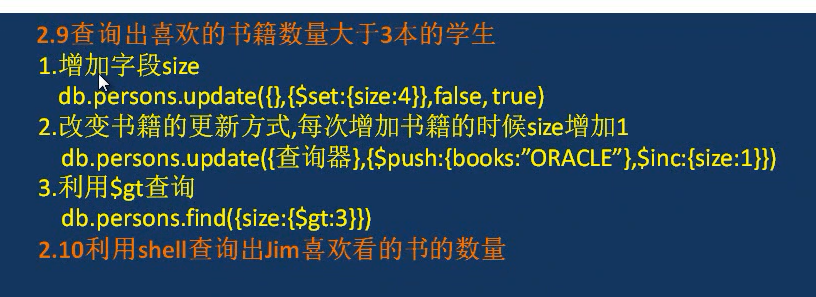
使用"$in"判断是否在某个范围内

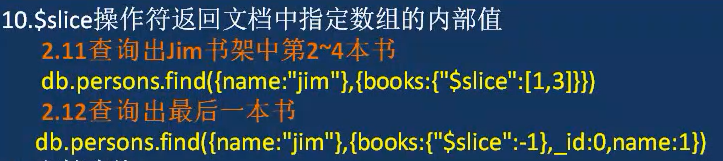
例6：查询年龄为18、28的学生

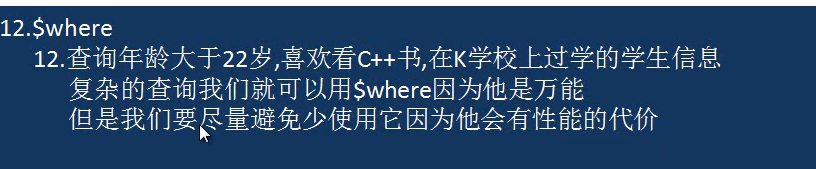
db.stu.find({age:{$in:[18,28]}})

$nin 不在某个范围

$not 取反操作，可以在任何场景使用，$in 和 $nin只能在集合中使用







## 5 支持正则表达式

使用//或$regex编写正则表达式

例7：查询姓黄的学生

db.stu.find({name:/^黄/})

db.stu.find({name:{$regex:'^黄'}}})

## 6 自定义查询

使用$where后面写一个函数，返回满足条件的数据

例7：查询年龄大于30的学生

db.stu.find({$where:function(){return this.age>20}})

## 7 分页 limit与skip

分页：有一个参数叫做page=3，生成的JSON不一样。就是我们想寻找所有的新闻，但是是位于第3页的新闻。那么有两种做法：

1） 错误的做法： 将所有的result都读取到数组，然后进行数据操作，进行分页；

2） 正确的做法： 在数据库中，只读取这么多内容。

mongodb提供了两个简便的函数：imit() skip()

db.student.find().limit(4) 表示读取4条内容

db.student.find().limit(4).skip(5) skip表示略过条数

加入，第一页是page=0。每页10条，所以当前页的查询语句

db.student.find({}).limit(10).skip(page\*10)

查询quan全部数据总数：db.student.stats().count

db.集合名称.find().limit(NUMBER) 用于读取指定数量的文档

参数NUMBER表示要获取文档数，如果没有指定参数则显示集合中的所有文档

db.集合名称.find().skip(NUMBER) 用于跳过指定数量的文档

参数NUMBER表示跳过的记录条数，默认值为0

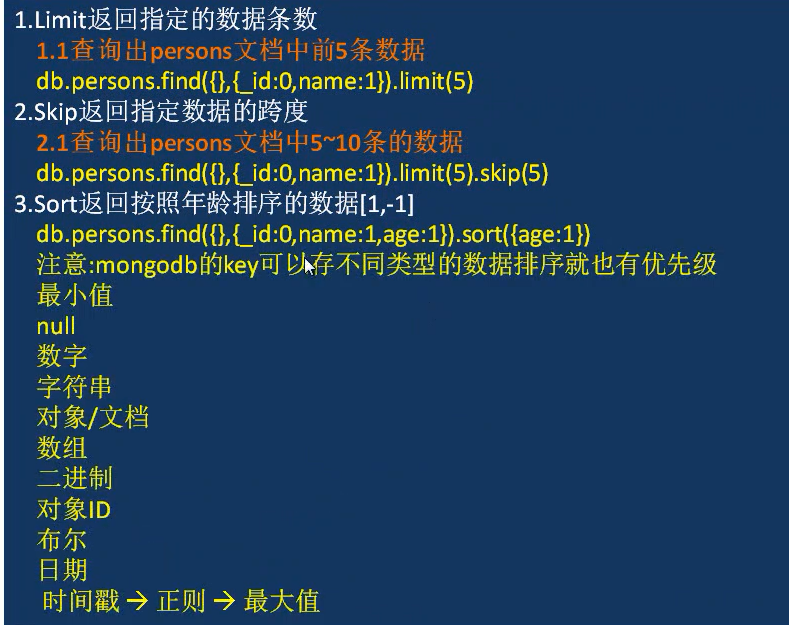
方法limit()和skip()可以一起使用，不分先后顺序

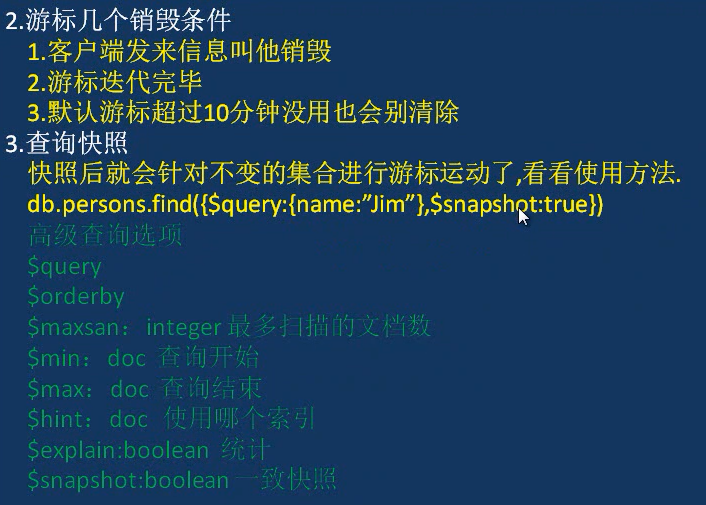
创建数据集

for(i=0;i<15;i++){db.t1.insert({\_id:i})}

查询第5至8条数据

db.stu.find().limit(4).skip(5)或db.stu.find().skip(5).limit(4)





## 8 投影

在查询到的返回结果中，只选择必要的字段，而不是选择一个文档的整个字段，如：一个文档有5个字段，需要显示只有3个，投影其中3个字段即可。

db.集合名称.find({},{字段名称:1,...})

参数为字段与值，值为1表示显示，值为0不显示

对于需要显示的字段，设置为1即可，不设置即为不显示

特殊：对于\_id列默认是显示的，如果不显示需要明确设置为0

例1： db.stu.find({},{name:1,gender:1})

例2： db.stu.find({},{\_id:0,name:1,gender:1})

## 9 排序

db.集合名称.find().sort({字段:1,...})

参数1为升序排列

参数-1为降序排列

例1： db.stu.find().sort({gender:-1,age:1}) 根据性别降序，再根据年龄升序

## 10 统计个数

db.集合名称.find({条件}).count() db.集合名称.count({条件})

例1： db.stu.find({gender:1}).count() 统计男生人数

例2： db.stu.count({age:{$gt:20},gender:1}) 统计年龄大于20的男生人数

## 11 消除重复

db.集合名称.distinct('去重字段',{条件})

例1: db.stu.distinct('gender',{age:{$gt:18}}) 查找年龄大于18的性别（去重）

# 四 高级操作

## 1 聚合aggregate 与表达式

聚合(aggregate)主要用于计算数据，类似sql中的sum()、avg()

db.集合名称.aggregate([{管道:{表达式}}])

在mongodb中，文档处理完毕后，通过类似Linux的管道进行下一次处理。

常用管道

$group：将集合中的文档分组，可用于统计结果

$match：过滤数据，只输出符合条件的文档

$project：修改输入文档的结构，如重命名、增加、删除字段、创建计算结果

$sort：将输入文档排序后输出

$limit：限制聚合管道返回的文档数

$skip：跳过指定数量的文档，并返回余下的文档

$unwind：将数组类型的字段进行拆分

处理输入文档并输出：表达式:'$列名' 常用表达式如下：

$sum：计算总和，$sum:1同count表示计数

$avg：计算平均值

$min：获取最小值

$max：获取最大值

$push：在结果文档中插入值到一个数组中

$first：根据资源文档的排序获取第一个文档数据

$last：根据资源文档的排序获取最后一个文档数据

db.mycol.aggregate([{$group : {\_id : "$by\_user", num\_tutorial : {$sum : 1}}}])

类似sql语句： select by\_user, count(\*) from mycol group by by\_user

## 2 $group

将集合中的文档分组，可用于统计结果

\_id表示分组的依据，使用某个字段的格式为'$字段'

例1：统计男生、女生的总人数

db.stu.aggregate([

{$group:

{

\_id:'$gender',

counter:{$sum:1}

}

}

])

group by null：将集合中所有文档分为一组

例2：求学生总人数、平均年龄

db.stu.aggregate([

{$group:

{

\_id:null,

counter:{$sum:1},

avgAge:{$avg:'$age'}

}

}

])

## 3 透视数据

例3：统计学生性别及学生姓名

db.stu.aggregate([

{$group:

{

\_id:'$gender',

name:{$push:'$name'}

}

}

])

使用$$ROOT可以将文档内容加入到结果集的数组中，代码如下

db.stu.aggregate([

{$group:

{

\_id:'$gender',

name:{$push:'$$ROOT'}

}

}

])

$match

用于过滤数据，只输出符合条件的文档

使用MongoDB的标准查询操作

例1：查询年龄大于20的学生

db.stu.aggregate([

{$match:{age:{$gt:20}}}

])

例2：查询年龄大于20的男生、女生人数

db.stu.aggregate([

{$match:{age:{$gt:20}}},

{$group:{\_id:'$gender',counter:{$sum:1}}}

])

$project

修改输入文档的结构，如重命名、增加、删除字段、创建计算结果

例1：查询学生的姓名、年龄

db.stu.aggregate([

{$project:{\_id:0,name:1,age:1}}

])

例2：查询男生、女生人数，输出人数

db.stu.aggregate([

{$group:{\_id:'$gender',counter:{$sum:1}}},

{$project:{\_id:0,counter:1}}

])

$sort

将输入文档排序后输出

例1：查询学生信息，按年龄升序

b.stu.aggregate([{$sort:{age:1}}])

例2：查询男生、女生人数，按人数降序

db.stu.aggregate([

{$group:{\_id:'$gender',counter:{$sum:1}}},

{$sort:{counter:-1}}

])

$limit

限制聚合管道返回的文档数

例1：查询2条学生信息

db.stu.aggregate([{$limit:2}])

$skip

跳过指定数量的文档，并返回余下的文档

例2：查询从第3条开始的学生信息

db.stu.aggregate([{$skip:2}])

例3：统计男生、女生人数，按人数升序，取第二条数据

db.stu.aggregate([

{$group:{\_id:'$gender',counter:{$sum:1}}},

{$sort:{counter:1}},

{$skip:1},

{$limit:1}

])

注意顺序：先写skip，再写limit

$unwind

将文档中的某一个数组类型字段拆分成多条，每条包含数组中的一个值

语法1

对某字段值进行拆分

db.集合名称.aggregate([{$unwind:'$字段名称'}])

构造数据

db.t2.insert({\_id:1,item:'t-shirt',size:['S','M','L']})

查询

db.t2.aggregate([{$unwind:'$size'}])

语法2

对某字段值进行拆分

处理空数组、非数组、无字段、null情况

db.inventory.aggregate([{

$unwind:{

path:'$字段名称',

preserveNullAndEmptyArrays:<boolean>#防止数据丢失

}

}])

构造数据

db.t3.insert([

{ "\_id" : 1, "item" : "a", "size": [ "S", "M", "L"] },

{ "\_id" : 2, "item" : "b", "size" : [ ] },

{ "\_id" : 3, "item" : "c", "size": "M" },

{ "\_id" : 4, "item" : "d" },

{ "\_id" : 5, "item" : "e", "size" : null }

])

使用语法1查询

db.t3.aggregate([{$unwind:'$size'}])

查看查询结果，发现对于空数组、无字段、null的文档，都被丢弃了

问：如何能不丢弃呢？

答：使用语法2查询

db.t3.aggregate([{$unwind:{path:'$sizes',preserveNullAndEmptyArrays:true}}])

# 五 索引

### 1 索引的简单使用

使用索引可以提升查询速度，我们先创建这样一份多数据文档：

for(i=0;i<10000;i++){db.Users.insert(name:'test'+i,age:i)}

使用性能分析explain：

db.Users.find({name:'test9999'}).explain('executionStats')

此时输出的语句中包含了explain结果，其中TimeMillis表示整体查询时间（毫秒）

使用索引

步骤一：建立索引，1只是代表升序，-1代表降序

db.Users.ensureIndex({name:1})

步骤二：对索引属性进行查询，发现查询时间明显少于刚才直接查询的结果

db.Users.find({name:'test9999'}).explain('executionStats')

## 2 索引命令

建立唯一索引，实现唯一约束功能：

db.Users.ensureIndex({'name':1},{'unique':true})

联合索引，对多个属性建立一个索引，按照find()出现的顺序

db.Users.ensureIndex({name:1,age:1})

查看所有索引

db.Users.getIndexes()

删除索引

db.Users.dropIndexes('索引名')

# 六 安全性与超级管理员

为了更安全的访问mongodb，需要访问者提供用户名和密码，于是需要在mongodb中创建用户，mongo采用了角色-用户-数据库的安全管理方式。

常用系统角色如下：

root： 只在admin数据库中可用，超级账号，超级权限

read： 允许用户读取指定数据库

readWrite： 允许用户读写指定数据库

第一步：创建超级管理用户，超级管理员的作用主要是 为公司不同项目的数据库分配用户。

use admin

db.createUser({

user:'admin',

pwd:'123',

roles:[{role:'root',db:'admin'}]

})

第二步：修改配置文件，启用安全认证

sudo vi /etc/mongod.conf

security:

authorization: enabled

注意：keys and values之间一定要加空格, 否则解析会报错

第三步：重启再连接

重启服务 sudo service mongod stop

sudo service mongod start

终端连接 mongo -u 'admin' -p '123' --authenticationDatabase 'admin'

此时需要这样验证登录才能正常使用

第四步：普通用户管理，使用超级管理员登录，然后进入用户管理操作

查看当前数据库的用户

use test1

show users

创建普通用户

db.createUser({

user:'t1',

pwd:'123',

roles:[{role:'readWrite',db:'test1'}]

})

终端连接

mongo -u t1 -p 123 --authenticationDatabase test1

切换数据库，执行命令查看效果

修改用户：可以修改pwd、roles属性

db.updateUser('t1',{pwd:'456'})

# 七 复制与备份

## 1 复制

复制提供了数据的冗余备份，并在多个服务器上存储数据副本，提高了数据的可用性，并可以保证数据的安全性。复制还允许从硬件故障和服务中断中恢复数据。

复制至少需要两个节点A、B...

A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；

节点常见的搭配方式为：一主一从、一主多从；

主节点记录在其上的所有操作，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；

设置复制节点，接下来的操作需要打开多个终端窗口，而且可能会连接多台ubuntu主机，会显得有些乱，建议在xshell中实现：

step1: 创建数据库目录t1、t2

mkdir t1

mkdir t2

step2: 使用如下格式启动mongod，注意副本集replSet的名称是一致的

mongod --bind\_ip 192.168.196.128 --port 27017 --dbpath ~/Desktop/t1 --replSet rs0

mongod --bind\_ip 192.168.196.128 --port 27018 --dbpath ~/Desktop/t2 --replSet rs0

step3: 连接主服务器，此处设置192.168.196.128:27017为主服务器

mongo --host 192.168.196.128 --port 27017

step4:初始化

rs.initiate()

step5:查看当前状态

rs.status()

step6:添加复本集

rs.add('192.168.196.128:27018')

step7:连接第二个mongo服务

mongo --host 192.168.196.128 --port 27018

step8:向主服务器中插入数据

use test1

for(i=0;i<10;i++){db.t1.insert({\_id:i})}

db.t1.find()

step9:在从服务器中插查询，说明：如果在从服务器上进行读操作，需要设置rs.slaveOk()

rs.slaveOk()

db.t1.find()

删除从节点 rs.remove('192.168.196.128:27018')

关闭主服务器后，再重新启动，会发现原来的从服务器变为了从服务器，新启动的服务器（原来的从服务器）变为了从服务器。

## 2 备份

mongodump -h dbhost -d dbname -o dbdirectory

-h：服务器地址，也可以指定端口号

-d：需要备份的数据库名称

-o：备份的数据存放位置，此目录中存放着备份出来的数据

例1

sudo mkdir test1bak

sudo mongodump -h 192.168.196.128:27017 -d test1 -o ~/Desktop/test1bak

## 3 恢复

mongorestore -h dbhost -d dbname --dir dbdirectory

-h：服务器地址

-d：需要恢复的数据库实例

--dir：备份数据所在位置

例2

mongorestore -h 192.168.196.128:27017 -d test2 --dir ~/Desktop/test1bak/test1