# mongoDB索引篇

上文中对索引页开了一个简单的小头，这篇教程就来详细叙述一下mongoDB的索引

## 索引的作用，含义和特点

索引是对数据库表中一列或多列的值进行排序的一种结构，使用索引可快速访问数据库表中的特定信息。

数据库索引好比是一本书前面的目录，能加快数据库的查询速度。

建立索引的目的是加快对表中记录的查找或排序。为表设置索引要付出代价的：一是增加了数据库的存储空间，二是在插入和修改数据时要花费较多的时间(因为索引也要随之变动)。数据库索引就是为了提高表的搜索效率而对某些字段中的值建立的目录 。

## 索引的建立

在增删改查篇的末尾其实提及了建立单个索引的语句

db.collectionname.ensureIndex({字段名:1/-1}) //其中1为升序索引，-1为降序索引

## 索引的分类

1：\_id索引

2：单键索引

3：多键索引

4：复合索引

5：过期索引

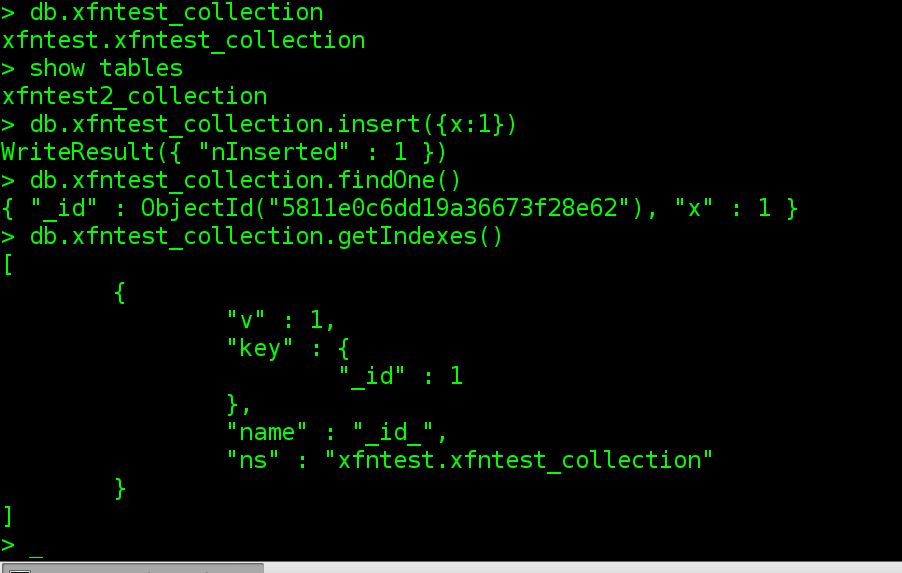
6：全文索引

7：地理位置索引

### \_id索引

\_id索引：绝大多数集合默认建立的索引，对于每个插入的数据，MongoDB都会生成一条唯一的\_id字段。

只要我建立一个空表，插入一条数据，\_id字段（索引）就会自动建立



如上图

建立了xfntest\_collection的表，然后插入一条json型数据{x:1}

然后查看表的索引，发现key已经变成了\_id字段

### 单键索引

1.单键索引是最普通的索引

2.与\_id索引不同，单键索引不会自动创建

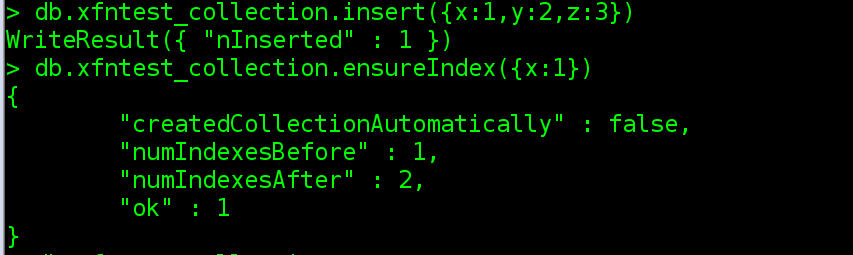
单键索引的建立在上方已经提及

db.collectionname.ensureIndex({字段名:1/-1}) //其中1为升序索引，-1为降序索引

索引可以重复创建，若创建已经存在的索引，则会直接返回成功。

比如我数据库中的数据形式类似于{x:1,y:2,z:3}，我想为x字段建立一个升序索引

db.collectionname.ensureIndex({x:1})





此时出现了两个key(即两个索引)，一个是\_id字段，另外一个就是x字段

### 多键索引

多键索引与单键索引创建形式相同，区别在于字段的值。

1）单键索引：值为一个单一的值，如字符串，数字或日期。

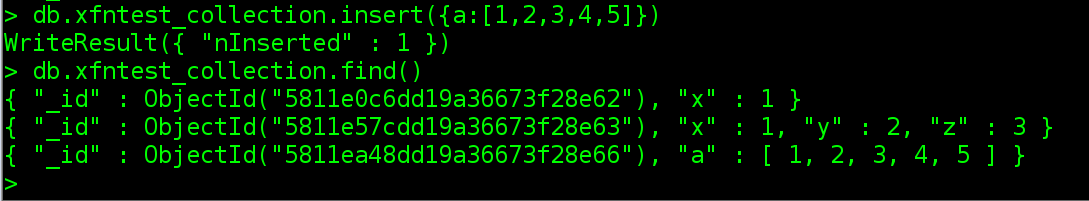
2）多键索引：值具有多个记录，如数组。

先插入一条数组型数据吧

db.collectionname.insert({a:[1,2,3]})

此时已经添加了一个多键索引(插入的同时已经建好索引)

自行来测试db.collectionname.insert({a:[1,2,3]})，然后getIndexes()一下吧，同样没有key为a的索引，但是其实a的索引已经建立，也就是说，如果mongoDB中如果插入数组类型的多键数据，索引是自动建立的，证据就是我们能很方便的对插入的多键类型进行快速的查找和排序，详见<http://blog.csdn.net/xinghebuluo/article/details/7221642>



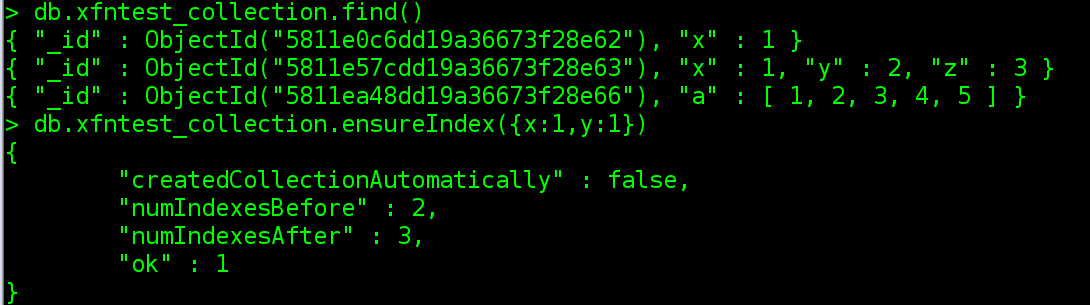
由于多键索引不太好表示，所以此时getIndexes()是找不到多键索引的，但是它确实存在

### 复合索引

有时我们查询的条件不只一个，比如我想查找x为1且y为2的数据

此时需要建立符合索引，其实就是在建立单键索引的基础上添加别的字段再标明一下升序或者降序就可以了

db.collectionname.ensureIndex({字段名a:1/-1, 字段名b:1/-1,…})



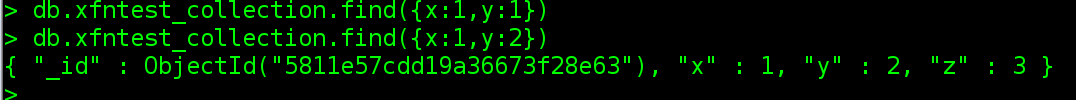
比如我现在想建立关于x和y两个字段的升序索引

db.xfntest\_collection.ensureIndex({x:1,y:1})

简单查看一下索引发现都有了

查询一下x为1且y为1的数据（没有）

查询一下x为1且y为2的数据（发现了）



### 过期索引

定义：是指在一段时间后会过期的索引

1、索引过期后，相应的数据会被删除

2、适合存储一些在一段时间之后会失效的数据，比如用户的登录信息，存储的日志

3、建立方法

db.collection.ensureIndex({字段名:1},{expireAfterSeconds:10}) //创建过期索引，time-字段，expireAfterSeconds在多少秒后过期，单位：秒（注意expireAfterSeconds遵循驼峰法则，不要拼错，否则不会删除）

a.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，除标准时间之外的过期索引都不能被自动删除，比如指定time:22

比如使用new Date()创建的时间类型就是ISODate

b.如果指定了ISODate数组，则按照快到期的时间进行删除

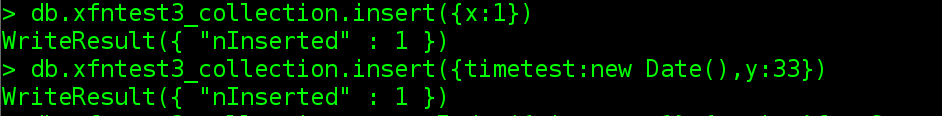
c.过期索引不能是复合索引，因为不能指定两个字段均为时间字段，否则容易冲突

d.删除时间是不精确的

删除过程是由后台程序每60s跑一次，而且删除也需要一些时间，所以存在误差

如果设置时间很小，则会在60秒内才可以删除，而不是在规定时间内删除

也就是说expireAfterseconds指定小于60的值意义不大，因为最少也要60s之后执行

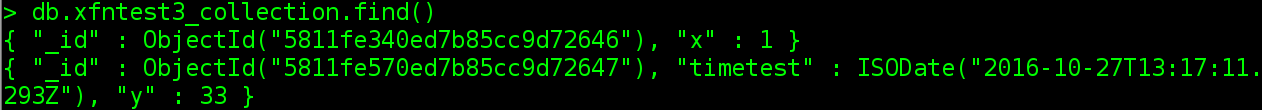


添加两条数据，一条普通，一条名字为我们想设置的过期索引



过期索引建立完成了

此时看一下数据

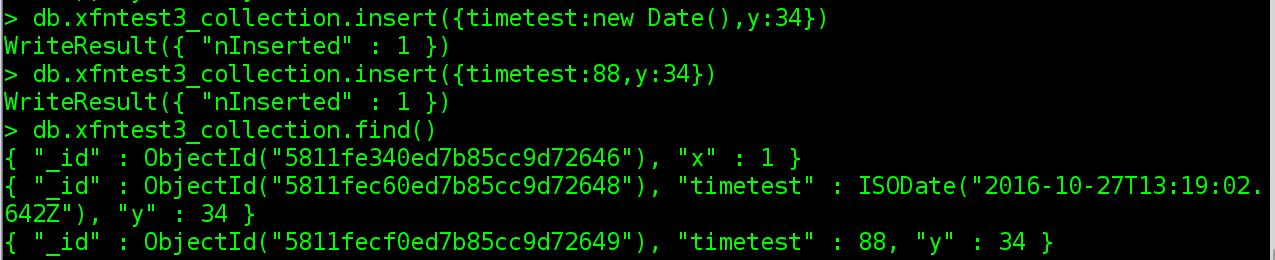


一条普通数据和一条包含过期索引时间为13:17:11的数据

过了1分钟

再插入两条数据包含过期索引，但是timetest值一个为ISODate一个为普通数字

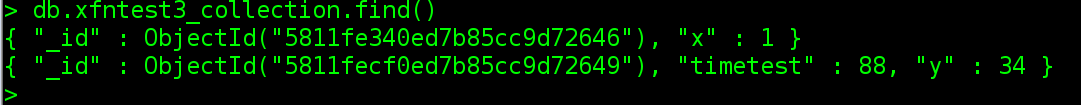
再看一下数据



包含过期索引时间为13:17:11的数据被自动删除

添加了包含过期索引时间为13:19:02的数据和为88的数据

又过了1分钟



包含过期索引时间为13:19:02的数据被删除，包含88的普通数据还在

说明两点

1. 在过期索引建立后，无论数据是在过期索引建立前还是建立后插入，只要过期时间到了就会被删除
2. 想要数据过期被删除，那么插入数据的过期索引字段值就一定要是ISODate或者ISODate数组的形式

### 全文索引

定义：对字符串与字符串数组创建全文可搜索的索引

不使用全文索引：查找困难，效率低下，需要正则匹配，逐条扫描。

使用全文索引：简单查询即可查询需要的结果

db.collectionname.ensureIndex( { key: "text" } )

db.collectionname.ensureIndex( { key1: "text", key2: "text" } )

db.collectionname.ensureIndex( { "$\*\*": "text" } )

1、db.imooc\_collection.ensuerIndex({key:"text"})

为key字段建全文索引，text为参数。

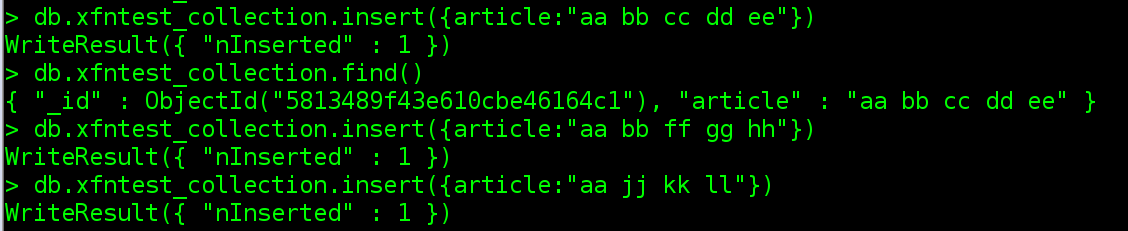
2、db.imooc\_collection.ensuerIndex({key\_1:"text",key\_2:"text"})

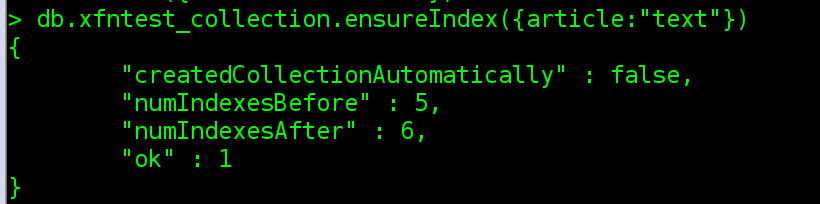
为key\_1,key\_2字段建全文索引，text为参数。

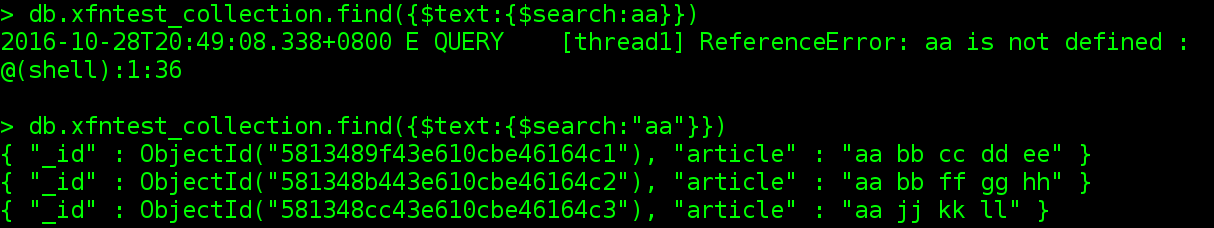
3、db.imooc\_collection.ensuerIndex({"$\*\*":"text"})

不指定字段，创建集合下所有信息的全文索引。

实际操作一下吧

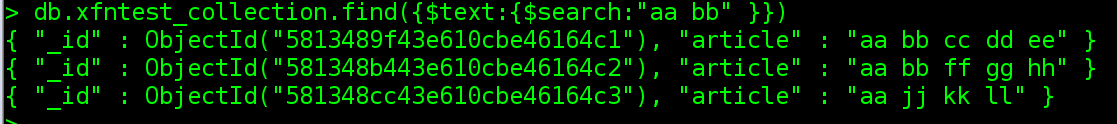
首先我们插入3条包含article的数据，注意在值处加双引号

为artIcle字段建立全文索引



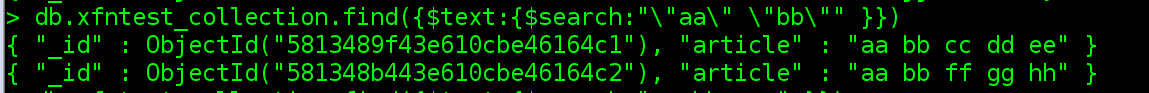
查询包含“aa”的数据，注意双引号别丢了

3条都出现了

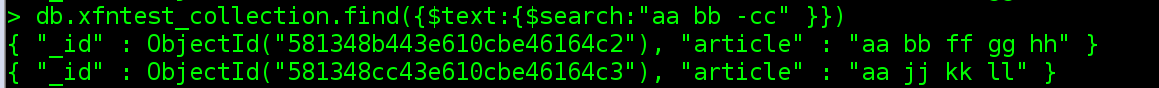


查包含aa或包含bb的数据就是这样，还是3条都出现了

嗯，我还想查包含aa且包含bb的数据呢



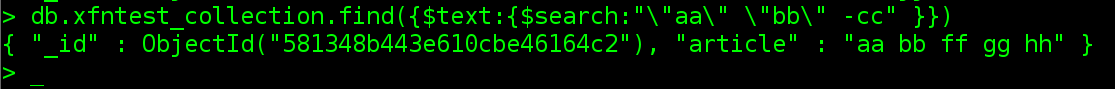
就是在查询的条件上继续添加双引号，但是因为和前面的双引号可能歧义闭合，我们在内部的双引号前面加上反斜杠\，进行转义



若是不想包含某项的数据出现，前面添加减号-就行

如上图即为：包含aa或包含bb但是不想包含cc的数据

那么我还是想要包含aa且包含bb，但是不包含cc的数据呢



老规矩：且的话，前面加\”，进行限定和转义

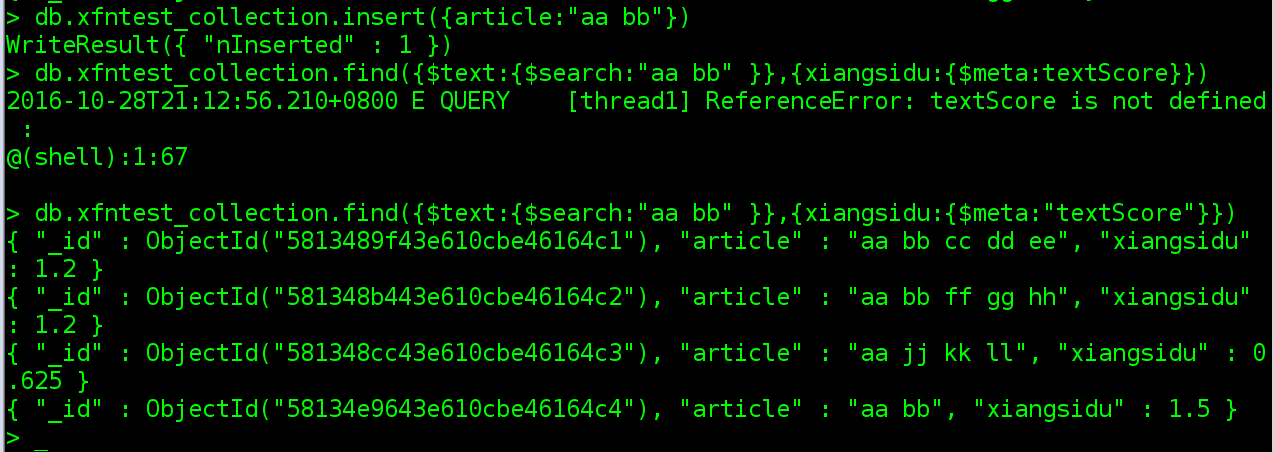
全文索引的相似度

创建全文索引后，在find({$text:{$search:”xx yy zz”}})后，添加

{score:{$meta:”textScore”}}，就可以显示查询出的每一条数据和查询条件的相似度，相似度数值越高越相似

其实是添加了一个叫score的字段，用于保存相似度，名字可以自己自定义

为了方便比较，添加了一个只含“aa bb”的数据



一开始出错是因为双引号没加，记得添加

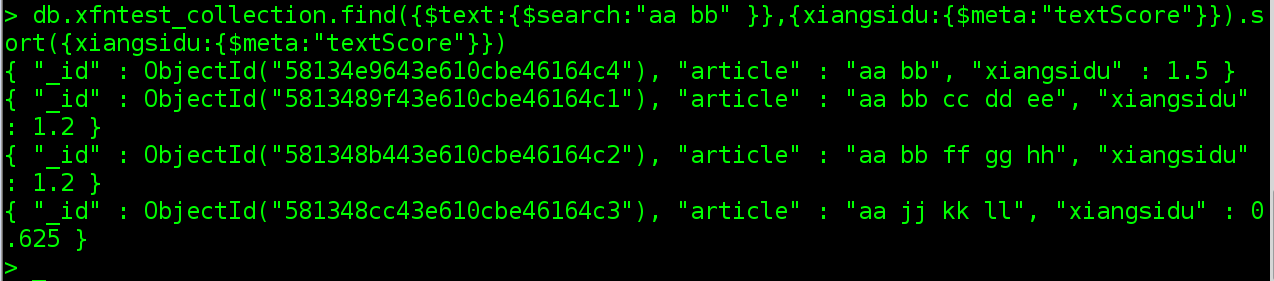
查询含aa或bb的数据

最后结果显示，添加了一个叫相似度的字段，其中只含“aa bb”的字段相似度最高

我能对相似度结果排序嘛？从高到低

可以，调用.sort()方法，参数和相似度字段相同

.sort({score:{$meta:”textScore”}})



全文索引的使用限制

每次查询，只能制定一个$text查询

$text查询不能出现在$nor查询中

查询中如何包含了$text，hint不在起作用（hint用于自行指定使用的索引，常用于测试索引的性能）

目前，MongoDB全文索引还不支持中文(2.6.3版本)

### 地理位置索引

概念：将一些点的位置存储在MongoDB中，创建索引后，可以按照位置来查找其他点

子分类：

2d索引：用于存储和查找平面上的点

2dsphere索引：用于存储和查找球面上的点

查找方法：

1、查找据李某个点一定距离内的点

2、查找包含在某区域内的点

#### 2d索引

db.collection.ensureIndex({字段名:”2d”})

位置表示方式：经纬度【经度，维度】

取值范围：经度【-180,180】纬度【-90,90】

例如：db.collection.insert({w:[180,90]})

查询方式：

（1）$near查询：查询距离某个点最近的点,默认返回100个最近的点

//db.collection.find({w:{$near:[1,1]}})

//db.collection.find({w:{$near:[1,1]，$maxDistance:10}}) //maxDistance为最大距离限制

（2）$geoWithin查询：查询某个形状内的点

--》形状的表示

a、$box:矩形，使用

{$box:[[<x1>,<y1>],[<x2>,<y2>]]}//第一个坐标代表左边界(左上)，第二个右边界(右下)

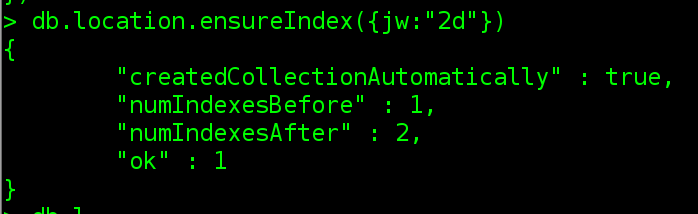
b、$center:圆形

{$center:[[<x1>,<y1>],r]}//r为圆的半径

c、$polygon:多边形

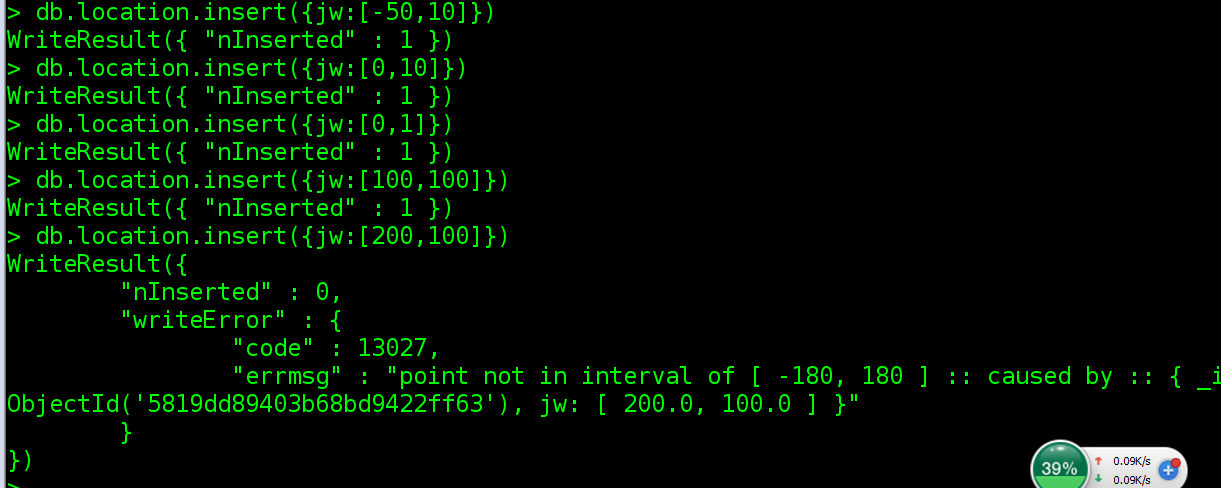
{$polygon:[[<x1>,<y1>],[<x2>,<y2>],[<x3>,<y3>]]}

简单看一下



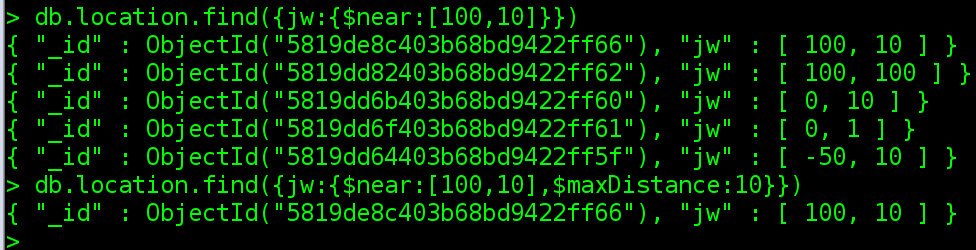
我们在jw字段建立了2d索引

插入一下数据



经度超过180，报错了

但是纬度超过90了居然没报错？（我们在实际使用时一定要注意，纬度不能超出[-90,90]否则会有麻烦，看了一下，纬度的范围也是[-180,180]，不知道是不是教程错了）



使用$near查询最近的点

查询某区域中的点

$geoWithin查询：查询某个形状内的点

--》形状的表示

a、$box:矩形，使用

{$box:[[<x1>,<y1>],[<x2>,<y2>]]}//第一个坐标代表左边界，第二个右边界

db.collection.find({jw:{$geowithin:{$box:[[1,1],[2,3]]}}})

b、$center:圆形

{$center:[[<x1>,<y1>],r]}//r为圆的半径

c、$polygon:多边形

{$polygon:[[<x1>,<y1>],[<x2>,<y2>],[<x3>,<y3>]]}

2D索引中的 geoNear查询

geoNear使用runCommand命令进行使用，常用使用如下：

db.runCommand({

getNear:<collection>,//查询的表名

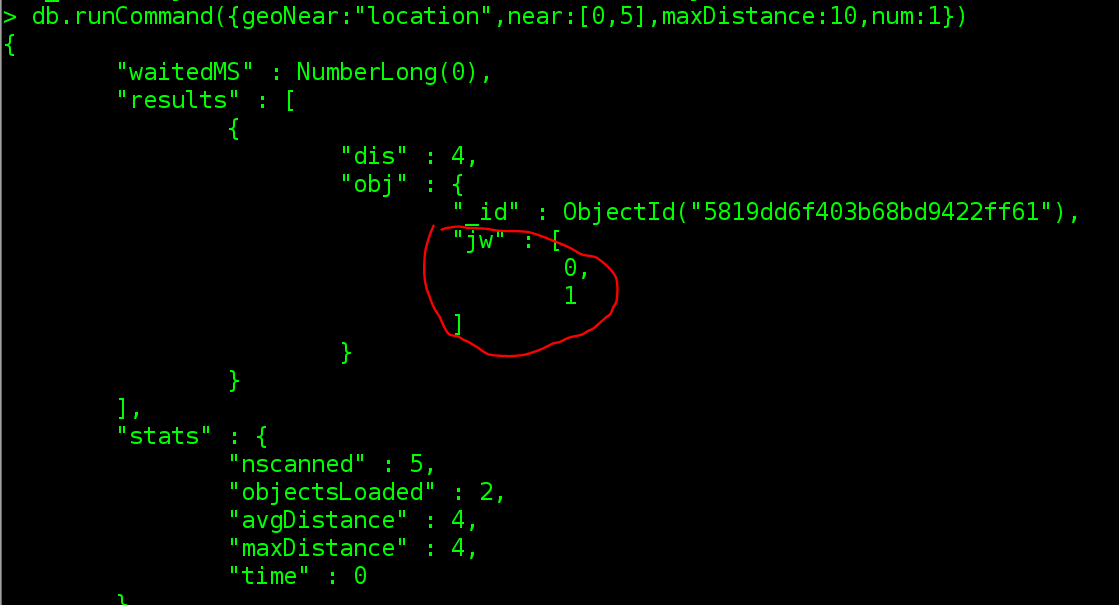
near:[x,y],//查询的坐标点

minDistance:(对2d索引无效，对2dsphere索引有效)

maxDistance://最大距离

num:2//返回的数量

})



图为查到的最近点

#### 2Dsphere索引

2Dsphere：球面地理位置索引

创建方法:db.collection.ensureIndex({w: '2dsphere'})

2Dsphere位置表示方式：

GeoJSON：描述一个点，一条直线，多边形等形状。

格式：

{type:'', coordinates:[list]}

GeoJSON查询可支持多边形交叉点等，支持MaxDistance 和 MinDistance

## 索引小结篇之索引属性（重要属性简介）

### name指定

创建索引的格式：

db.collection.ensureIndex({indexValue},{indexProperty})

其中：indexProperty比较重要的有

1：名字

db.collection.ensureIndex({indexValue},{name:})

MongoDB会自动的创建，规则是key\_1 或者 key\_-1 1或者-1代表排序方向，一般影响不大，长度一般有限制125字节

为了见名知意我们可以自己来命名

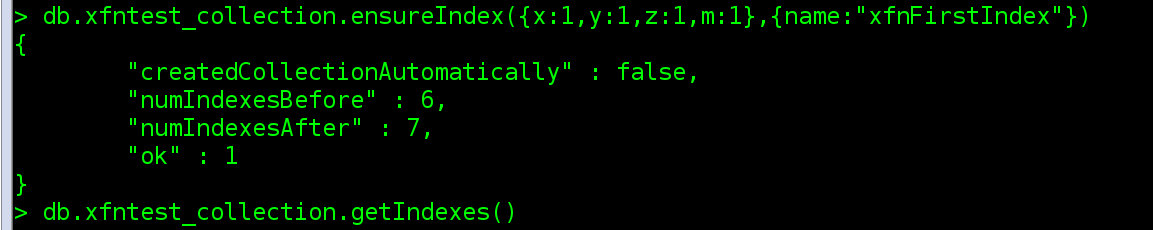
db.collection.ensureIndex({x:1,y:1,z:1,m:1},{name:"normal\_index"})

删除索引 db.collection.dropIndex(indexName)

2：唯一性

3：稀疏性

4：是否定时删除



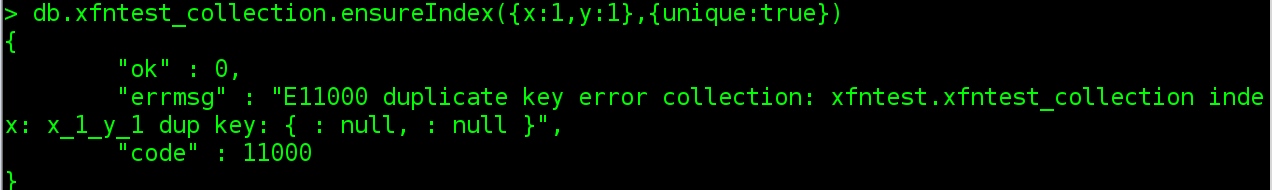
简而言之，就是自己给自己建立的复合索引起一个名字，既方便自己下次使用的时候能理解含义，也方便删除的时候只需要这个索引的名字就够了

### 创建唯一索引unique

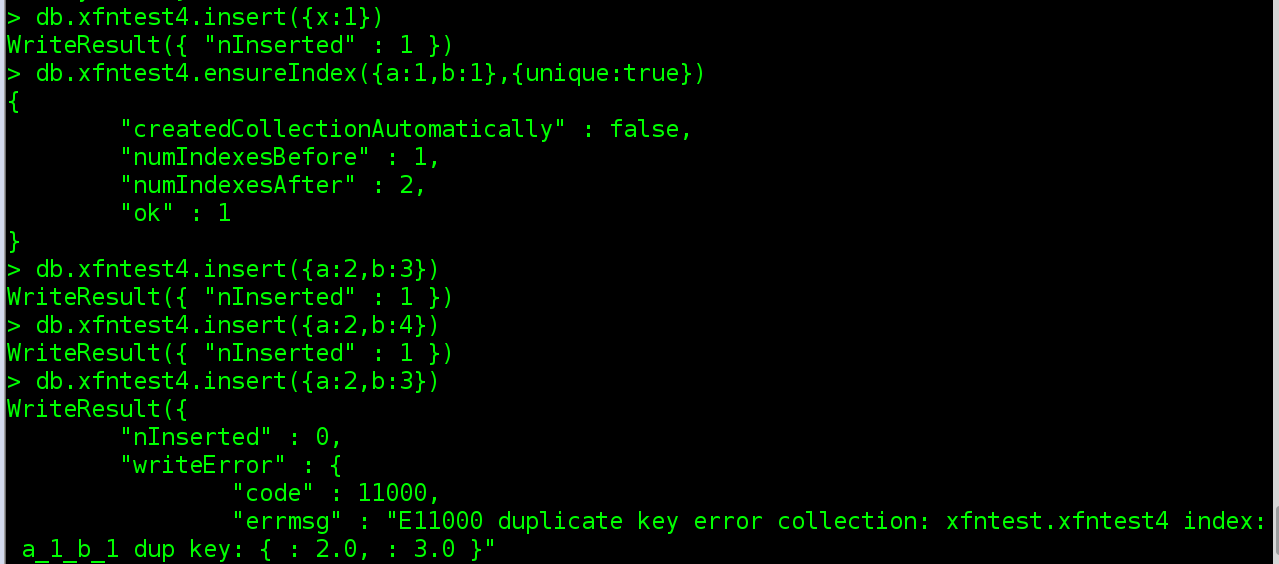
有时候我们想实现：插入某字段数据时，若该数据已经存在则不重复插入的功能，此时我们需要在相应字段的索引，添加unique属性为true

db.collection.ensureIndex({m:1,n:1},{unique:true})

关于这个属性实践的时候遇到了一个问题，尝试了一下没解决



后来我换了一个新的collection，插入了一条数据，然后插入unique索引，实践成功



插入重复数据会报

### sqarse稀疏性

>db.collection.ensureIndex({},{sparse:true/false}) #指定索引是否稀疏

MongoDB索引默认是不稀疏的。

稀疏性的不同代表了MongoDB在处理索引中存在但是文档中不存在的字段的两种不同的方法。

例如，我们为一个collection的x字段指定了索引，但这个collection中可以插入如{y:1,z:1}这种不存在x字段的数据，如果索引为不稀疏的，mongodb依然会为这个数据（{y:1,z:1}）创建索引，如果在创建索引时指定为稀疏索引，那么就可以避免这件事情发生了。

（这个相当于，我给一本书以包含x的词条建立目录，

不指定稀疏性的话：不包含x，但是包含y,z等其他字母的字段我也写进目录里面。

指定稀疏性：就不把这些不含x的词条写进目录了）

>db.imooc\_2.insert({"m":1})

>db.imooc\_2.insert({"n":1})

通过$exists可以判断字段是否存在，如

>db.imooc\_2.find({m:{$exists:true}}) #筛选出有m字段的文档

给这个文档的m字段创建一个稀疏索引：

>db.imooc\_2.ensureIndex({m:1},{sparse:true})

第二条文档不存在m字段，所以不会创建这个索引

如果使用稀疏索引查找不存在稀疏索引字段的文档，mongodb则不会使用这个索引查找

例如：

>db.imooc\_2.find({m:{$exists:false}}) #可以查到数据,因为find()搜索所有数据

但如果我们通过hint强制使用索引，就不会查到数据了

>db.imooc\_2.find({m:{$exists:false}}).hint("m\_1") #使用名字为m\_1的索引，强制查询不存在m字段的数据

查不出数据，因为m\_1索引里，根本找不到n

（强制指定索引查询相当于：我只在包含x的目录中查，如果稀疏性设为true,则其中不可能查到不包含x,反而包含其他字母的词条）

## 索引构建情况分析

创建索引的好处：加快索引查询。

创建索引的坏处：增加磁盘消耗，降低写入性能。

评判索引的4种方法：

1.mongostat 工具

2.profile集合

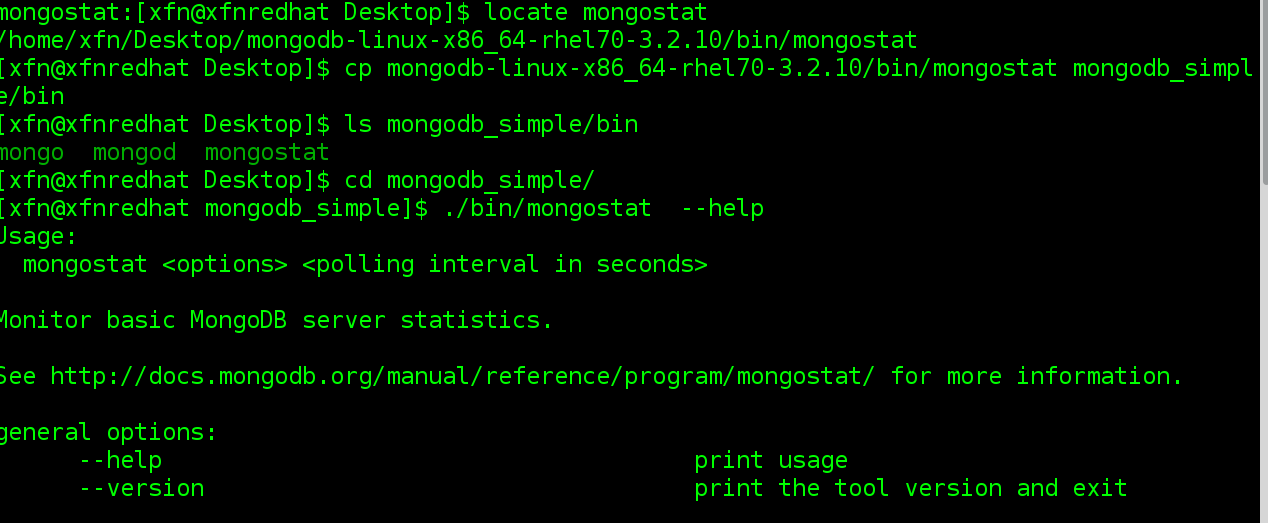
3.日志分析

4.使用explain查询分析器分析

### mongostat篇

首先找到mongostat文件（使用locate命令），将其复制到mongodb\_simple的bin文件夹下

然后简单看一下帮助文件，了解一下大致含义



使用方法：mongostat -h ip:port -u -p

字段说明：

返回的采样数据采用百分比

索引情况：idx miss 索引未命中率

输出字段：

inserts --当前的插入数量（单位：秒）

query --当前的查询数量（单位：秒）

update --当前更新的数量（单位：秒）

delete --当前的删除数量（单位：秒）

getmore --当前的迭代返回数量（单位：秒）

command --执行命令的数量

flushes --刷盘时间（单位：秒）

mapped --mmap 大小

vsize --磁盘空间大小

res --常驻内存大小

faults --内存换页时间（单位：秒）

locked --锁的使用情况

idx miss --未命中索引率

qr|qw --读|写队列

ar|aw --活跃的客户端连接数量

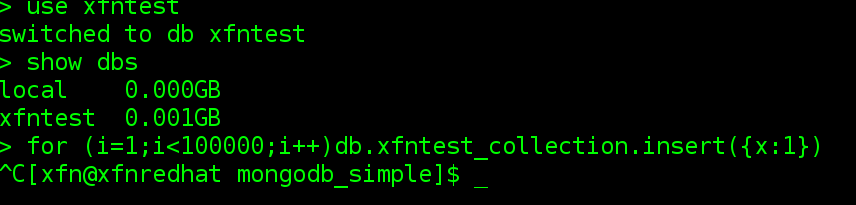
netIn --网卡输入流量

netOup --网卡输出流量

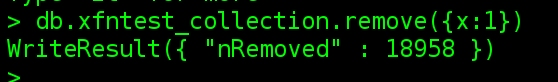
conn --当前连接到mongodb的连接数量

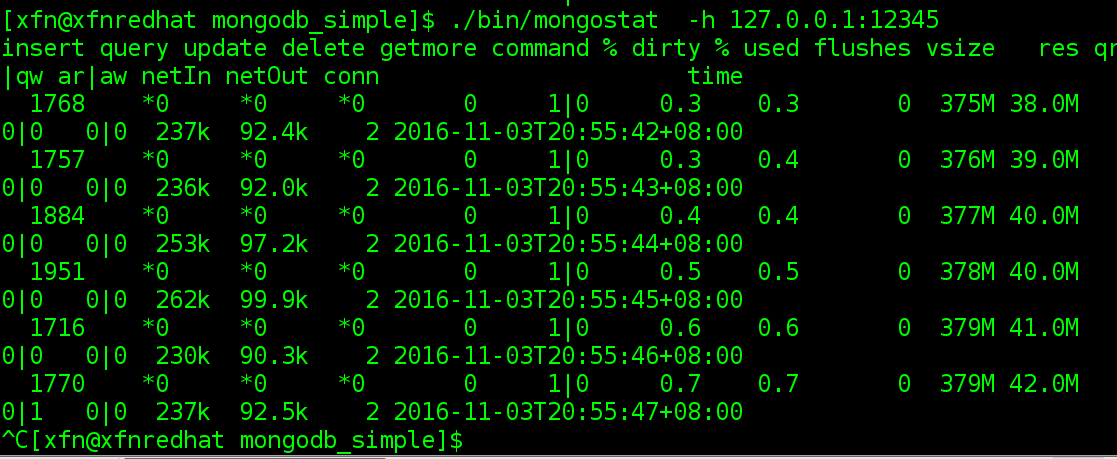


大致如图，后面有些字段没有显示



原本应该是{x:i},但是为了方便删除，写成了{x:1}





可以在插入大量数据时，使用mongostat查看当前mongostat性能

性能分析工具mongostat -h ip地址:端口

查询结果中需要注意的表列值

qr/qw 表示读队列和写队列值，较高时数据库存在性能问题

idx miss 表示查询时索引命中情况，较高时影响查询效率

### profile集合

查看当前数据库的profile状态

db.getProfilingStatus()

{ "was" : 0, "slowms" : 100 }

查看当前数据库的记录级别

db.getProfilingLevel()

0|1|2

设置当前数据库的profile记录级别

db.setProfilingLevel(0|1|2)

was --profile记录级别，0关闭，1记录所有慢查询，2记录所有操作

slowms --慢查询阀值

查看profile文件

db.system.profile.find()

{ "op" : "query",--操作类型

"ns" : "imooc.system.profile", --查询的命名空间，;databasename.collectionname'

"query" : { "query" : { }, --查询条件

"orderby" : { "$natural" : -1 } }, --约束条件

"ntoreturn" : 1, --返回数据条目

"ntoskip" : 0, --跳过的条目

"nscanned" : 1, --扫描的数目含索引

"nscannedObjects" : 1, --扫描的数据数目

"keyUpdates" : 0, --

"numYield" : 0, --其他情况

"lockStats" : { --锁状态

"timeLockedMicros" : { --锁占用时间（毫秒）

"r" : NumberLong(82), --读锁

"w" : NumberLong(0) --写锁

},

"timeAcquiringMicros" : {

"r" : NumberLong(2), "w" : NumberLong(2)

}

},

"nreturned" : 1,

"responseLength" : 651, --返回长度

"millis" : 0, --查询时间

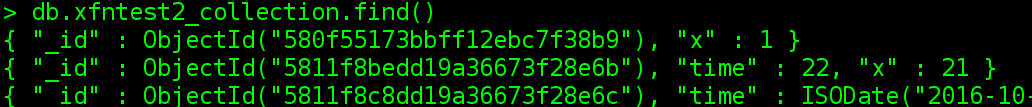
### 日志查看

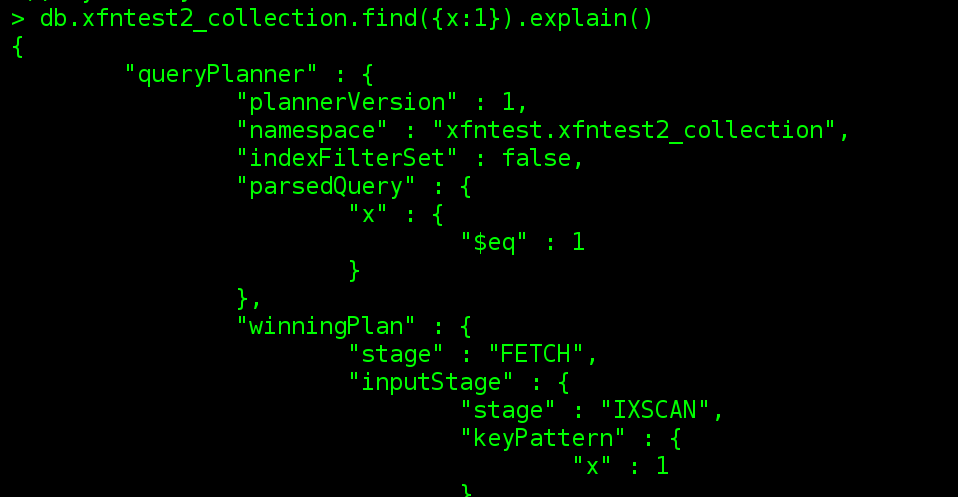
关于日志，首先要去conf/mongod.conf下添加参数

berbose = vvvvv //日志记录级别

日志和profile类似也会记录系统相应的使用情况，并且能够配置日志的记录情况，verbose这个参数就是是控制日志记录的情况的一个参数，位数越多表示记录的越详细

### 关于explain()





在执行MongoDB的命令时如果加上explain()这个参数，就会显示出对应的命令的执行详细信息，供我们来分析对应的命令的执行情况，就像profile功能一样。

db.colltction.find({x:1}).explain()

{

"cursor" : "BasicCursor", --使用的游标

"isMultiKey" : false,

"n" : 1,

"nscannedObjects" : 100000, --扫描的数据量

"nscanned" : 100000, --包含索引的扫描量

"nscannedObjectsAllPlans" : 100000,

"nscannedAllPlans" : 100000,

"scanAndOrder" : false,

"indexOnly" : false,

"nYields" : 781,

"nChunkSkips" : 0,

"millis" : 25, --查询消耗时间（毫秒）

"server" : "XXX",

"filterSet" : false

}

//版本不同，explain显示的结果也不同