

一、介绍

MongoDB 是一个高性能分布式文件存储数据库，通常采用官方的二进制包进行安装。

二、MongoDB 安装

a) 准备工作

- i. CentOS 32 位系统，建议采用多核 CPU，在多核并行编译时，物理内存不能少于 8G，配置下编译大概耗时 30 分钟。

- ii. 安装依赖的软件包

- 1. `yum install pcre-devel python-devel scons`

- iii. 从官方下载最新的源码包（选择对应版本的源码包）

- 1. <https://www.mongodb.com/download-center#community>
 - 2. `wget https://fastdl.mongodb.org/linux/mongodb-linux-i686-3.2.7.tgz`

b) 使用官网二进制包安装（需要对应版本，为了后续测试高级，请使用此安装方式）

```
wget -c https://fastdl.mongodb.org/linux/mongodb-linux-x86_64-3.2.8.tgz
```

-c 选项断点续传

```
tar -zxvf mongodb-linux-x86_64-3.2.8.tgz
```

```
mv mongodb-linux-x86_64-3.2.8 /web/mongodb //复制到指定目录即可
```

```
cd /web/mongodb
```

```
mkdir mongodb_db //创建数据库存放位置
```

```
/web/mongodb/bin/mongod --dbpath=/web/mongodb/mongodb_db/
```

指定数据库

在另一个终端打开 mongod

```
/web/mongodb/bin/mongo
```

出现如下，则说明 OK

```
2016-04-13T06:58:53.672-0400 I CONTROL [initandlisten] ** .....
```

MongoDB 服务加入随机启动

```
vi /etc/rc.local
```

使用 vi 编辑器打开配置文件，并在其中加入下面一行代码

```
/web/mongodb/bin/mongod -dbpath=/usr/local/mongodb/data/db --port 27017  
-logpath=/usr/local/mongodb/log --logappend
```

MongoDB 服务客户端加入环境变量

```
ln -s /web/mongodb/bin/mongo /usr/bin/mongo
```

执行 mongod 启动 MongoDB 服务器指定配置文件

```
/web/mongodb/bin/mongod --config mongodb.conf
```

二、使用官方 yum 安装（亲自操作成功，启动服务器指定参数时，需要到原生脚本指定）

a) 文档地址 <https://docs.mongodb.com/master/tutorial/install-mongodb-on-red-hat/>

b) 创建 mongodb-org-2.6.repo 文件

i. vim /etc/yum.repos.d/mongodb-org-2.6.repo

```
[mongodb-org-2.6]  
name=MongoDB 2.6 Repository  
baseurl=http://downloads-distro.mongodb.org/repo/redhat/os/x86_64/  
gpgcheck=0  
enabled=1
```

c) 执行安装命令

i. sudo yum install -y mongodb-org

报错：yum install mongodb-org-server

ii. 分别执行

1. yum install mongodb-org-2.6.12-1.x86_64

2. yum install mongodb-org-server
3. yum install mongodb-org-mongos
4. yum install mongodb-org-shell
5. yum install mongodb-org-tools

d) 检查是否安装

- i. rpm -qa |grep mongodb

e) 必要时关闭 SELINUX

- i. /etc/selinux/config SELINUX=disabled

f) 默认文件路径

- i. /etc/mongod.conf 配置文件
- ii. /var/lib/mongo MongoDB 实例存储其数据文件
- iii. /var/log/mongodb 日志文件

g) 管理 mongod

service mongod start| stop | restart chkconfig mongod on 开机自启动

h) 查看是否启动服务

netstat -lanp | grep "27017"

i) 启动客户端

Linux :在命令行执行 mongo 即可 windows : mongo 127.0.0.1:27017/admin

```
[root@localhost ~]# mongo
MongoDB shell version: 2.6.12
connecting to: test
>
```

j) yum 安装相关信息

####主要信息####

/usr/bin/mongo

/usr/bin/mongod

/usr/bin/mongodump

/var/log/mongodb/mongod.log

/tmp/mongodb-27017.sock

/etc/mongod.conf

/etc/rc.d/init.d/mongod

三、Windows 下载安装

- k) 下载地址 <https://www.mongodb.com/download-center#community>
- l) 安装注意
- i. 先在硬盘上创建两个目录用存放 mongoDB 数据和存放官网下载安装文件
 - ii. 配置环境变量方便在 CMD 启动服务（找到 \$path\bin\mongod.exe）
 - 1. mongod --dbpath D:\software\MongoDBDATA
 - iii. mongod 帮助文档 mongod --help
 - iv. mongodb 启动数据库服务配置文件（网上查阅）
 - v. 可视化工具建议使用 <https://robomongo.org/>

三、mongodb 和关系型数据库的对比图

对比项	mongodb	MySQL oracle
表	集合 list	二维表 table
表的一行数据	文档 document	一条记录 record
表字段	键 key	字段 field
字段值	值 value	值 value

主外键	无	PK,FK
灵活度扩展性	极高	差

- a) 关系数据的表的记录必须保证保证拥有每一个字段
- b) mongodb 的每一个 document 的 key 可以不一样 (每条记录 key 都可以不一样)
- c) 关系型数据查询使用 SQL
- d) mongodb 查询使用内置 find 函数->基于 BSON 的特点查询工具

四、基本 SHELL 命令

- e) 创建一个数据库
 - i. use [dbname] //此时你什么也没有处理就离开则这个空数据库会被删除
- f) 给指定数据库添加集合并且添加记录
 - i. db.[documentname].insert({name:"zhangsan"}); //自动创建一个文档 ID
- g) 查看所有数据库
 - i. show dbs //默认提供本地数据
- h) 查看数据库中的所有文档

```

i. show collections

> show dbs
admin    (empty)
foobar   0.078GB
local    0.078GB
> db.users.insert({name:"zhangsan"})
WriteResult({ "nInserted" : 1 })
> use foobar
> show collections
system.indexes //系统自动生产存储文档索引的文档
users
>

```

实例：文档 ID

```
> db.system.indexes.find()
{ "v" : 1, "key" : { "_id" : 1 }, "name" : "_id_", "ns" : "admin.users" }
> db.users.find()
{ "_id" : ObjectId("57603ea03f4a07a1fd16353e"), "name" : "zhangsan" }
```

i) 查询指定文档的数据（查第一条 findOne()）

```
> db.system.indexes.find()
{ "v" : 1, "key" : { "_id" : 1 }, "name" : "_id_", "ns" : "admin.users" }
> db.users.find()
{ "_id" : ObjectId("57603ea03f4a07a1fd16353e"), "name" : "zhangsan" }
```

// 本条记录系统会自动添加一个_id，值为对象类型 ObjectId

j) 更新文档数据

db.[documentName].update({查询条件}, {更新内容})

```
db.users.update({name: 'lisi'}, {$set: {name: 'changeName'}})
```

相当于：update users set name = 'changeName' where name= 'lisi';

```
> db.users.find()
{ "_id" : ObjectId("57603ea03f4a07a1fd16353e"), "name" : "zhangsan" }
{ "_id" : ObjectId("576040413f4a07a1fd16353f"), "name" : "lisi" }
> db.users.update({name: 'lisi'}, {$set: {name: 'changeName'}})
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
> db.users.find()
{ "_id" : ObjectId("57603ea03f4a07a1fd16353e"), "name" : "zhangsan" }
{ "_id" : ObjectId("576040413f4a07a1fd16353f"), "name" : "changeName" }
>
```

```
db.users.update({age: 25}, {$set: {name: 'changeName'}});
```

相当于：update users set name = 'changeName' where age = 25;

```
db.users.update({name: 'Lisi'}, {$inc: {age: 50}});
```

相当于：update users set age = age + 50 where name = 'Lisi';

```
db.users.update({name: 'Lisi'}, {$inc: {age: 50}, $set: {name: 'hoho'}});
```

相当于：update users set age = age + 50, name = 'hoho' where name = 'Lisi';

修改的整体思路：先查询再利用修改器 \$set 修改内容

k) 删除文档中的数据

```
> db.users.find()
{ "_id" : ObjectId("57603ea03f4a07a1fd16353e"), "name" : "zhangsan" }
{ "_id" : ObjectId("576040413f4a07a1fd16353f"), "name" : "changeName" }
```

```

> db.users.find()
{ "_id" : ObjectId("57603ea03f4a07a1fd16353e"), "name" : "zhangsan" }
{ "_id" : ObjectId("576040413f4a07a1fd16353f"), "name" : "changeName" }
> db.users.remove({name:'zhangsan'});
WriteResult({ "nRemoved" : 1 })
> db.users.find()
{ "_id" : ObjectId("576040413f4a07a1fd16353f"), "name" : "changeName" }
>

```

l) 添加文档数据

```

> db.users.insert({name:"lisi",age:25,sex:nan});
2016-06-15T01:51:38.095+0800 ReferenceError: nan is not defined
> db.users.insert({name:"lisi",age:25,sex:'nan'});
WriteResult({ "nInserted" : 1 })

```

m) 删除库中的集合（删除记录）

```

> db.users.drop();
true
> show collections
system.indexes

```

n) 删除当前使用数据库

```
db.dropDatabase()
```

o) Shell 的 help

i. 里面有所有的 shell 可以完成命令帮助

```
help //全局
```

```
db.help(); //数据库中
```

```
db.[document].help(); //文档中
```

```
db.[document].find().help(); //命令
```

```
rs.help();
```

p) Mongodb api 文档 <http://api.mongodb.com/js/2.1.2/index.html>

q) 数据库和集合命名规范

i. 不能是空字符串

- ii. 不得含有 ' 空格 , ¥ / \ \O () -
- iii. 应该全部小写
- iv. 最多 64 个字节
- v. 数据库名不能和现有系统同名
- r) 这样的集合名字也是合法的
- s) mongodb 的 shell 内置 javascript 引擎可以直接执行 JS 代码


```
function insert(object){
    db.getCollection('users').text.insert(object)
}
insert({name:"lishi"});
```
- t) mongodb 可以使用 eval
 - i. db.eval("return 'update'");
- u) BSON 是 JSON 的扩展，它先新增了诸如日期，浮点等 JSON 不支持的数据类型

BSON	
null	用于表示空或者不存在的字段
布尔	两个数值 true 和 false
32 位何 64 位整数	Shell 中不支持 需用到其他高级语言的驱动来完成,JS 不可使用.
64 位浮点	Shell 中使用的数字其实全是这种类型 {x:3.414}
UTF-8	其实就是字符串类型
对象 ID	内置默认 ID 对象 {_id:ObjectId()}
日期	{x:new Date()}
正则	{x:/uspcat/i}
Javascript 代码块	{x:function(){...}}
undefined	为定义类型注意他和 null 不是一个类型
数组	{gps:[20,56]}
内嵌文档	{x:{name:"uspcat"}}
二进制	任意字节的字符串 shell 中时无法使用的

- v) MongoVUE 安装和简单使用 (可视化工具)

四、插入文档 (Document)

```
> use users
switched to db users
```



```
> db.users.insert({_id:"001",name:"zhangsan"}) //手动指定 ID
```

```
WriteResult({ "nInserted" : 1 })
```

```
> db.users.find()
```

```
{ "_id" : "001", "name" : "zhangsan" }
```

a) 插入文档

```
db.[documentName].insert({})
```

b) 批量插入

shell 这样执行是错误的 db.[documentName].insert([{}, {}, {},])

shell 不支持批量插入（根据版本而定）

MongoDB shell version: 2.6.12

```
db.users.insert([ {name:"lis",age:24}, {name:"wangwu",age:25}, {name:"zhaoliu",age:34} ])
```

```
BulkWriteResult({
```

```
  "writeErrors" : [ ],
```

```
  "writeConcernErrors" : [ ],
```

```
  "nInserted" : 3,
```

```
  "nUpserted" : 0,
```

```
  "nMatched" : 0,
```

```
  "nModified" : 0,
```

```
  "nRemoved" : 0,
```

```
  "upserted" : [ ]
```

```
})
```

```
> db.users.find()
```

```
{ "_id" : ObjectId("57b33ceeefc749aeba8e5304"), "name" : "zhangsan", "age" : 23 }
```

```
{ "_id" : ObjectId("57b33da5efc749aeba8e5305"), "name" : "lis", "age" : 24 }
```

```
{ "_id" : ObjectId("57b33da5efc749aeba8e5306"), "name" : "wangwu", "age" : 25 }
```

```
{ "_id" : ObjectId("57b33da5efc749aeba8e5307"), "name" : "zhaoliu", "age" : 34 }
```

想完成批量插入可以用 mongo 的应用驱动或是 shell 的 for 循环

c) Save 操作

save 操作和 insert 操作区别在于当遇到_id 相同的情况下

save 完成保存操作 //相当于更新数据

insert 则会报错

```
// SAVE
```

```
> db.users.insert({_id:001,name:"lili",age:26})
```

```

WriteResult({ "nInserted" : 1 })
> db.users.save({_id:001,name:"liming",age:26})
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })

//INSERT 报错

> db.users.insert({_id:001,name:"liming",age:26})
WriteResult({
  "nInserted" : 0,
  "writeError" : {
    "code" : 11000,
    "errmsg" : "insertDocument :: caused by :: 11000 E11000 duplicate key error index:
users.users.$_id_ dup key: { : 1.0 }"
  }
})

```

五、删除文档 (Document)

- a) 删除列表中所有数据

```
db.[documentName].remove()
```

集合的本身和索引不会被删除

```
> db.users.remove({})
```

```
WriteResult({ "nRemoved" : 5 })
```

- b) 根据条件删除

```
db.[documentName].remove({})
```

删除集合 text 中 name 等于 uspcat 的纪录

```
db.text.remove({name:" uspcat" })
```

- c) 小技巧

如果你想清楚一个数据量十分庞大的集合

直接删除该集合并且重新建立索引的办法

比直接用 remove 的效率高很多

六、更新文档 (Document)

- a) 强硬的文档替换式更新操作

```
db.[documentName].update({查询器},{修改器})

> db.users.update({name:"lisi"},{name:"lisi"})
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
```

备注：当查询时 name 字段没有内容没有了，**强硬的更新会用新的文档代替老的文档**

b) 主键冲突的时候会报错并且停止更新操作

因为是强硬替换当替换的文档和已有文档 ID 冲突的时候，则系统会报错

c) insertOrUpdate 操作

目的:查询器查出来数据就执行更新操作,查不出来就添加操作

做法:db.[documentName].update({查询器},{修改器},true)

```
db.users.update({name:"xixixi"},{name:"wuwu"},true)
WriteResult({
  "nMatched" : 0,
  "nUpserted" : 1,
  "nModified" : 0,
  "_id" : ObjectId("57b34bfc2a55f37234fb993f")
})
```

d) 批量更新操作

默认情况当查询器查询出多条数据的时候默认就修改**第一条**数据

```
{ "_id" : ObjectId("57b34581efc749aeba8e5308"), "name" : "lisi" }
{ "_id" : ObjectId("57b34581efc749aeba8e5309"), "name" : "wangwu", "age" : 25 }
{ "_id" : ObjectId("57b34581efc749aeba8e530a"), "name" : "wangwu", "age" : 34 }
> db.users.update({name:"wangwu"},{name:"xixi"})
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
> db.users.find()
{ "_id" : ObjectId("57b34581efc749aeba8e5308"), "name" : "lisi" }
{ "_id" : ObjectId("57b34581efc749aeba8e5309"), "name" : "xixi" }
{ "_id" : ObjectId("57b34581efc749aeba8e530a"), "name" : "wangwu", "age" : 34 }
```

如何实现批量修改

db.[documentName].update({查询器},{修改器},false, true)

```
{ "_id" : ObjectId("57b34581efc749aeba8e5308"), "name" : "lisi" }
```

```

{ "_id" : ObjectId("57b34581efc749aeba8e5309"), "name" : "wangwu" }
{ "_id" : ObjectId("57b34581efc749aeba8e530a"), "name" : "wangwu", "age" : 34 }
> db.users.update({name:"wangwu"},{$set:{name:"xixi"}},false,true)
WriteResult({ "nMatched" : 2, "nUpserted" : 0, "nModified" : 2 })
> db.users.find()
{ "_id" : ObjectId("57b34581efc749aeba8e5308"), "name" : "lisi" }
{ "_id" : ObjectId("57b34581efc749aeba8e5309"), "name" : "xixi" }
{ "_id" : ObjectId("57b34581efc749aeba8e530a"), "name" : "xixi", "age" : 34 }

```

e) 使用修改器来完成局部更新操作

修改器名称	语法	案例
\$set	{ \$set: { field: value } }	{ \$set: { name: "uspcat" } }
它用来指定一个键值对,如果存在键就进行修改不存在则进行添加.		
\$inc	{ \$inc : { field : value } }	{ \$inc : { "count" : 1 } }
只是使用与数字类型,他可以为指定的键对应的数字类型的数值进行加减操作.		
\$unset	{ \$unset : { field : 1 } }	{ \$unset : { "name":1 } }
他的用法很简单,就是删除指定的键		
\$push	{ \$push : { field : value } }	{ \$push : { books:"JS" } }
1.如果指定的键是数组追加加新的数值 2.如果指定的键不是数组则中断当前操作 Cannot apply \$push/\$pushAll modifier to non-array 3.如果不存在指定的键则创建数组类型的键值对		
\$pushAll	{ \$pushAll : { field : array } }	{ \$push : { books:["EXTJS","JS"] } }
用法和\$push 相似他可以批量添加数组数据		
\$addToSet	{ \$addToSet: { field : value } }	{ \$addToSet: { books:"JS" } }
目标数组存在此项则不操作,不存在此项则加进去		

// 添加一个文档中空数组

```

> db.users.insert({_id:001,name:"zhangsan",age:25,books:[]})
WriteResult({ "nInserted" : 1 })
> db.users.update({_id:001},{ $push:{books:"php"}})

```

```
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
```

```
> db.users.find({_id:001})
{ "_id" : 1, "name" : "zhangsan", "age" : 25, "books" : [ "php", "java", "c++" ] }
> db.users.update({_id:001},{$unset:{books:1}})
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
> db.users.find({_id:001})
{ "_id" : 1, "name" : "zhangsan", "age" : 25 }
>
```

```
> db.users.update({_id:001},{$pushAll:{books:["php","mysql","c++","js"]}})
```

```
> db.users.find({_id:001})
{ "_id" : 1, "name" : "zhangsan", "age" : 25, "books" : [ "php", "mysql", "c++", "js" ] }
> db.users.update({_id:001},{$addToSet:{books:"node.js"}})
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
> db.users.find({_id:001})
{ "_id" : 1, "name" : "zhangsan", "age" : 25, "books" : [ "php", "mysql", "c++", "js",
"node.js" ] }
```

修改器名称	语法	案例
\$pop	{ \$pop: { field: value } }	{ \$pop: { name: 1 } } { \$pop: { name: -1 } }
从指定数组删除一个值 1 删除最后一个数值,-1 删除第一个数值		
\$pull	{ \$pull: { field : value } }	{ \$pull : { "book" : "JS" } }
删除一个被指定的数值		
\$pullAll	{ \$pullAll: { field : array } }	{ \$pullAll: { "name":["JS","JAVA"] } }
一次性删除多个指定的数值		
\$	{ \$push : { field : value } }	{ \$push : { books:"JS" }

1. 数组定位器,如果数组有多个数值我们只想对其中一部分进行操作我们就要用到定位器(\$)

例子:

例如有文档

```
{name:"YFC",age:27,books:[{type:'JS',name:"EXTJS4"},{type:"JS",name:"JQUERY"},{type:"DB",name:"MONGO DB"}]}
```

我们要把 type 等于 JS 的文档增加一个相同的作者 author 是 USPCAT

办法:

```
db.text.update({"books.type":"JS"},{$set:{"books.$author":"USPCAT"}})
```

```
{ "_id" : 1, "name" : "zhangsan", "age" : 25, "books" : [ "php", "mysql", "c++", "js", "node.js" ] }
```

```
> db.users.update({_id:001},{ $pop:{books:-1}})
```

```
{ "_id" : 1, "name" : "zhangsan", "age" : 25, "books" : [ "mysql", "c++", "js", "node.js" ] }
```

```
> db.users.update({_id:001},{ $pullAll:{books:["php","js"]}})
```

```
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
```

```
> db.users.find({_id:001})
```

```
{ "_id" : 1, "name" : "zhangsan", "age" : 25, "books" : [ "mysql", "c++", "node.js" ] }
```

```
>
```

***切记修改器是放到最外面,后面要学的查询器是放到内层的**

f) \$addToSet 与 \$each 结合完成批量**数组**更新

```
db.text.update({_id:1000},{ $addToSet:{books:{ $each:[ "JS" ," DB" ]}}})
```

\$each 会循环后面的数组把每一个数值进行 \$addToSet 操作

```
{ "_id" : 1, "name" : "zhangsan", "age" : 25, "books" : [ "mysql", "c++", "node.js", "php", "js", "php", "jss" ] }
```

```
> db.users.update({_id:001},{ $addToSet:{books:{ $each:["php","php5"]}}})
```

```
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
```

```
> db.users.find({_id:001})
```

```
{ "_id" : 1, "name" : "zhangsan", "age" : 25, "books" : [ "mysql", "c++", "node.js", "php", "js", "php", "jss", "php5" ] }
```

g) 存在分配与查询效率

当 document 被创建的时候 DB 为其分配内存和预留内存当修改操作不超过预留内存的时候则速度非常快反而超过了就要分配新的内存，则会消耗时间



h) runCommand 函数和 findAndModify 函数

runCommand 可以执行 mongoDB 中的特殊函数

findAndModify 就是特殊函数之一他的用于是返回 update 或 remove 后的文档

runCommand({ "findAndModify" : " processes" ,

query:{查询器},

sort{排序},

new:true

update:{更新器},

remove:true

}).value

// 实例

```
ps = db.runCommand({
  "findAndModify":"persons",
  "query":{"name":"text"},
  "update":{"$set":{"email":"1221"}},
  "new":true
}).value
do_something(ps)
```

// 详细介绍

findAndModify 的调用方式和普通的更新略有不同，还有点慢，这是因为它要等待数据库的响应。这对于操作查询以及执行其他需要取值和赋值风格的原子性操作来说是十分方便的。

findAndModify 命令中每个键对应的值如下所示。

findAndModify 字符串，集合名。

query 查询文档，用来检索文档的条件。

sort 排序结果的条件。

update 修改器文档，对所找到的文档执行的更新。

remove 布尔类型，表示是否删除文档。

new 布尔类型，表示返回的是更新前的文档还是更新后的文档。默认是更新前的文档。

"update"和"remove"必须有一个，也只能有一个。要是匹配不到文档，这个命令会返回一个错误。

这个命令有些限制。它一次只能处理一个文档，也不能执行 upsert 操作，只能更新已有文档。相比普通更新来说，findAndModify 速度要慢一些。大概耗时相当于一次查找，一次更新和一次 getLastError 顺序执行所需的时间。

```
db.runCommand("findAndModify":集合名,"query":{查询条件},"update":{修改器})
```

七、Find 详解

a) 指定返回的键 (select fields from tableName)

```
db.[documentName].find ({条件},{键指定})
```

数据准备数据


```
var persons = [{
  name:"jim",
  age:25,
  email:"75431457@qq.com",
  c:89,m:96,e:87,
  country:"USA",
  books:["JS","C++","EXTJS","MONGODB"]
},
{
  name:"tom",
  age:25,
  email:"214557457@qq.com",
  c:75,m:66,e:97,
  country:"USA",
  books:["PHP","JAVA","EXTJS","C++"]
},
{
  name:"lili",
  age:26,
  email:"344521457@qq.com",
  c:75,m:63,e:97,
  country:"USA",
  books:["JS","JAVA","C#","MONGODB"]
},
{
  name:"zhangsan",
  age:27,
  email:"2145567457@qq.com",
  c:89,m:86,e:67,
  country:"China",
  books:["JS","JAVA","EXTJS","MONGODB"]
},
{
  name:"lisi",
  age:26,
  email:"274521457@qq.com",
  c:53,m:96,e:83,
  country:"China",
  books:["JS","C#","PHP","MONGODB"]
},
{
  name:"wangwu",
  age:27,
  email:"65621457@qq.com",
```

```

        c:45,m:65,e:99,
        country:"China",
        books:["JS","JAVA","C++","MONGODB"]
    },
    {
        name:"zhaoliu",
        age:27,
        email:"214521457@qq.com",
        c:99,m:96,e:97,
        country:"China",
        books:["JS","JAVA","EXTJS","PHP"]
    },
    {
        name:"piaoyingjun",
        age:26,
        email:"piaoyingjun@uspcat.com",
        c:39,m:54,e:53,
        country:"Korea",
        books:["JS","C#","EXTJS","MONGODB"]
    },
    {
        name:"lizhenxian",
        age:27,
        email:"lizhenxian@uspcat.com",
        c:35,m:56,e:47,
        country:"Korea",
        books:["JS","JAVA","EXTJS","MONGODB"]
    },
    {
        name:"lixiaoli",
        age:21,
        email:"lixiaoli@uspcat.com",
        c:36,m:86,e:32,
        country:"Korea",
        books:["JS","JAVA","PHP","MONGODB"]
    },
    {
        name:"zhangsuying",
        age:22,
        email:"zhangsuying@uspcat.com",
        c:45,m:63,e:77,
        country:"Korea",
        books:["JS","JAVA","C#","MONGODB"]
    }
}

```

```

for(var i = 0;i<persons.length;i++){
    db.persons.insert(persons[i])
}
var persons = db.persons.find({name:"jim"})
while(persons.hasNext()){
    obj = persons.next();
    print(obj.books.length)
}

```

- i. 查询出所有数据的指定键(name ,age ,country) (true 1 , false 0)

```
db.persons.find({}, {name:1,age:1,country:1,_id:0 , country:1})
```

表示：查询记录条数显示字段

_id 关闭显示信息 country , name , age 显示字段信息

```
> db.persons.find({}, {name:true,age:true,country:true,_id:0,country:1})
```

- b) 查询条件

1. 比较操作符

比较操作符		
\$lt	<	{age:{\$gte:22,\$lte:27}}
\$lte	<=	
\$gt	>	
\$gte	>=	
\$ne	!=	{age:{\$ne:26}}

2.查询条件大于小于

- 2.1 查询出年龄在 25 到 27 岁之间的学生

```
db.persons.find({age: {$gte:25,$lte:27},{_id:0,age:1})
```

- 2.2 查询出所有不是韩国籍的学生的数学成绩

```
db.persons.find({country:{$ne:" Korea" }},{_id:0,m:1})
```

3.包含或不包含 (\$in 或 \$nin)

3.1 查询国籍是中国或美国的学生信息

```
db.persons.find({country:{$in:[ "USA" , "China" ]}})
```

3.2 查询国籍不是中国或美国的学生信息

```
db.persons.find({country:{$nin:[ "USA" , "China" ]}})
```

4.OR 查询 (\$or)

4.1 查询语文成绩大于 85 或者英语大于 90 的学生信息

```
db.persons.find({$or:[{c:{$gte:85}},{e:{$gte:90}]}},{_id:0,c:1,e:1})
```

5.Null

5.1 把中国国籍的学生上增加新的键 sex

```
db.person.update({country:" China" },{$set:{sex:" m" }},false,true)
```

5.2 查询出 sex 等于 null 的学生

```
db.persons.find({sex:{$in:[null]}},{country:1})
```

6.正则查询

2.1 查询出名字中存在“ li” 的学生的信息

```
db.persons.find({name:/li/i},{_id:0,name:1})
```

7.\$not 的使用

\$not 可以用到任何地方进行取反操作

2.1 查询出名字中不存在“ li” 的学生的信息

```
db.persons.find({name:{$not:/li/i}},{_id:0,name:1})
```

\$not 和 \$nin 的区别是 \$not 可以用在任何地方 \$nin 是用到集合上的

8.数组查询 \$all 和 index 应用

2.1 查询喜欢看 MONGODB 和 JS 的学生

```
db.persons.find({books:{$all:[ "MONGOBD" ," JS" ]}}, {books:1,_id:0})
```

2.2 查询第二本书是 JAVA 的学习信息

```
db.persons.find({ "books.1" : " JAVA" })
```

9.查询指定长度数组\$size 它不能与比较查询符一起使用(这是弊端)

9.1 查询出喜欢的书籍数量是 4 本的学生

```
db.persons.find({books:{$size:4}}, {_id:0,books:1})
```

9.2 查询出喜欢的书籍数量大于 3 本的学生

9.2.1.增加字段 size

```
db.persons.update({},{$set:{size:4}},false, true)
```

9.2.2.改变书籍的更新方式,每次增加书籍的时候 size 增加 1

```
db.persons.update({查询器},{ $push:{books:" ORACLE" }, $inc:{size:1}})
```

9.2.3.利用\$gt 查询

```
db.persons.find({size:{$gt:3}})
```

9.3 利用 shell 查询出 Jim 喜欢看的书的数量

```
var persons = db.persons.find({name:"jim"}) // 查询结果是游标
while(persons.hasNext()){
    obj = persons.next();
    print(obj.books.length)
}
```

课间小结

1.mongodb 是 NOSQL 数据库但是他在文档查询上还是很强大的

2.查询符基本是用到花括号里面的更新符基本是在外面

3.shell 是个彻彻底底的 JS 引擎,但是一些特殊的操作要靠他的各个驱动包

来完成(JAVA,NODE.JS)

10.\$slice 操作符返回文档中指定数组的内部值

2.11 查询出 Jim 书架中第 2~4 本书

```
db.persons.find({name:"jim"},{books:{"$slice":[1,3]}})
```

2.12 查询出最后一本书

```
db.persons.find({name:"jim"},{books:{"$slice":-1},_id:0,name:1})
```

11.文档查询

为 jim 添加学习简历文档（允许每条记录的存在结构不一样）

```
var jim = [{
  school:"K",
  score:"A"
},{
  school:"L",
  score:"B"
},{
  school:"J",
  score:"A+"
}]
db.persons.update({name:"jim"},{$set:{school:jim}})
```

2.13 查询出在 K 上过学的学生

1. 这个用绝对匹配可以完成,但是有些问题(找找问题?顺序?总要带着 score?)

```
db.persons.find({school:{school:"K",score:"A"}},{_id:0,school:1})
```

2.为了解决顺序的问题我可以用对象“.”的方式定位

```
db.persons.find({"school.score":"A","school.school":"K"},{_id:0,school:1})
```

3.这样也问题看例子:

```
db.persons.find({"school.score":"A","school.school":" J" },{_id:0,school:1})
```

同样能查出刚才那条数据,原因是 score 和 school 会去其他对象对比

4.正确做法单条条件组查询\$elemMatch

```
db.persons.find({school:{$elemMatch:{school:"K",score:"A"}}})
```

12.\$where

12.1 查询年龄大于 22 岁,喜欢看 C++书,在 K 学校上过学的学生信息

复杂的查询我们就可以用\$where 因为他是万能,但是我们要尽量避免少

使用它因为他会有性能的代价。

```

db.persons.find({"$where":function(){
    var books = this.books;
    var school = this.school;
    if(this.age > 22){
        var php = null;
        for ( var i = 0; i < books.length; i++) {
            if(books[i] == "C++){
                php = books[i];
                if(school){
                    for (var j = 0; j < school.length; j++) {
                        if(school[j].school == "K"){
                            return true;
                        }
                    }
                    break;
                }
            }
        }
    }
}})

```

八、分页与排序

- a) Limit 返回指定的数据条数

查询出 persons 文档中前 5 条数据

```
db.persons.find({}, {_id:0,name:1}).limit(5)
```

- b) Skip 返回指定数据的跨度

查询出 persons 文档中 5~10 条的数据

```
db.persons.find({}, {_id:0,name:1}).limit(5).skip(5)
```

- c) Sort 返回按照年龄排序的数据[1,-1]

```
db.persons.find({}, {_id:0,name:1,age:1}).sort({age:1})
```

注意:mongodb 的 key 可以存不同类型的数据排序就也有优先级

最小值

null

数字

字符串

对象/文档

数组

二进制

对象 ID
布尔
日期
时间戳 --> □ 正则 --> 最大值

d) Limit 和 Skip 完成分页

1. 三条数据位一页进行分页

第一页□ `db.persons.find({}, {_id:0,name:1}).limit(3).skip(0)`

第二页□ `db.persons.find({}, {_id:0,name:1}).limit(3).skip(3)`

2. skip 有性能问题,没有特殊情况下也可以换个思路,对文档进行重新解构设计

_id	name	Age	Date
001	Jim	25	2012-07-31:12:24:24
002	tom	34	2012-07-31:12:24:54
003	Lilli	21	2012-07-31:12:24:57
004	zhangsan	23	2012-07-31:12:25:24
005	wangwu	26	2012-07-31:12:27:26
006	zhaoliu	29	2012-07-31:12:30:24

每次查询操作的时候前后台传值全要把上次的最后一个文档的日期保存下来

`db.persons.find({date:{$gt:日期数值}}).limit(3)`

个人建议□ 应该把软件的重点放到便捷和精确查询上而不是分页的性能上

因为用户最多不会翻查过 2 页的

九、游标和其他知识

a) 游标

利用游标遍历查询数据

```
var persons = db.persons.find();
while(persons.hasNext()){
    obj = persons.next();
    print(obj.name)
}
```


_id	name	Age
001	Jim	25
002	tom	34
003	Lilli	21
004	zhangsan	23
005	wangwu	26
006	zhaoliu	29





光标到了底部就会释放资源
不能再读取了。

b) 游标几个销毁条件

- 1.客户端发来信息叫他销毁
- 2.游标迭代完毕
- 3.默认游标超过 10 分钟没用也会别清除

c) 查询快照

快照后就会针对不变的集合进行游标运动了,看看使用方法.

```
db.persons.find({$query:{name:" Jim" },$snapshot:true})
```

高级查询选项

\$query

\$orderby

\$maxscan : integer 最多扫描的文档数

\$min : doc 查询开始

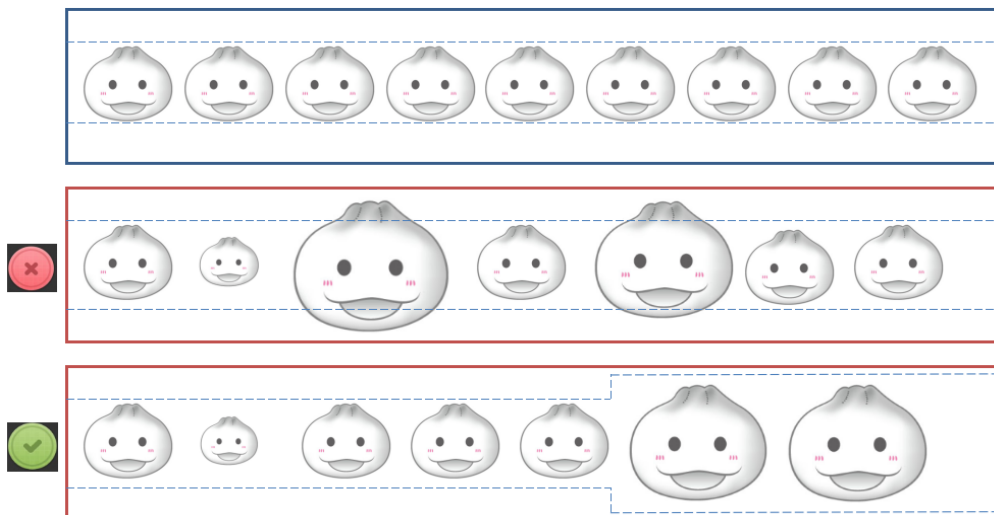
\$max : doc 查询结束

\$hint : doc 使用哪个索引

\$explain:boolean 统计

\$snapshot:boolean 一致快照

d) 为什么有的时候要用查询快照?看图



十、索引详解

a) 创建简单索引

数据准备，使用 shell 执行添加数据

```
use books    //选择
```

```
db.books.drop(); //删除
```

```
for(var i = 0 ; i<200000 ;i++){
    db.books.insert({number:i,name:i+"book"})
}
```

1.先检验一下查询性能

```
var start = new Date()
db.books.find({number:65871})
var end = new Date()
end - start
```

2.为 number 创建索引 （创建即可提高 10-100 倍）

```
db.books.ensureIndex({number:1})
```

3.再执行第一部的代码可以看出有数量级的性能提升

b) 索引使用需要注意的地方

1.创建索引的时候注意

1 是正序创建索引-1 是倒序创建索引

2.索引的创建在提高查询性能的同时会影响插入的性能

对于经常查询少插入的文档可以考虑用索引

3.符合索引要注意索引的先后顺序

4.每个键全建立索引不一定就能提高性能呢

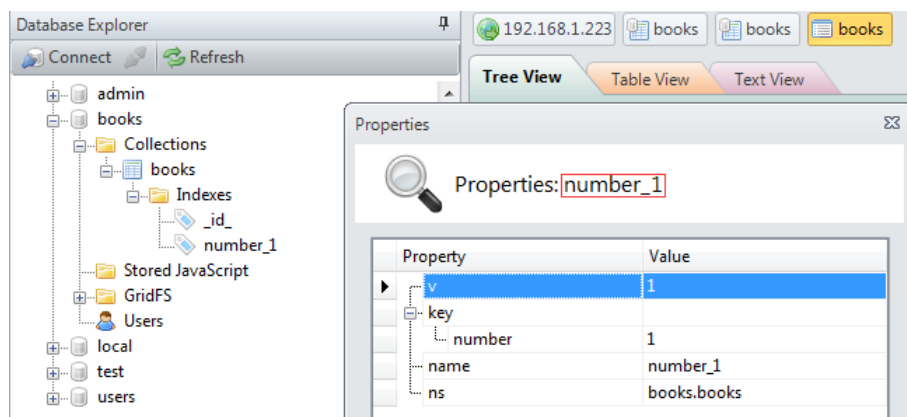
索引不是万能的

5.在做排序工作的时候如果是超大数据量也可以考虑加上索引

用来提高排序的性能

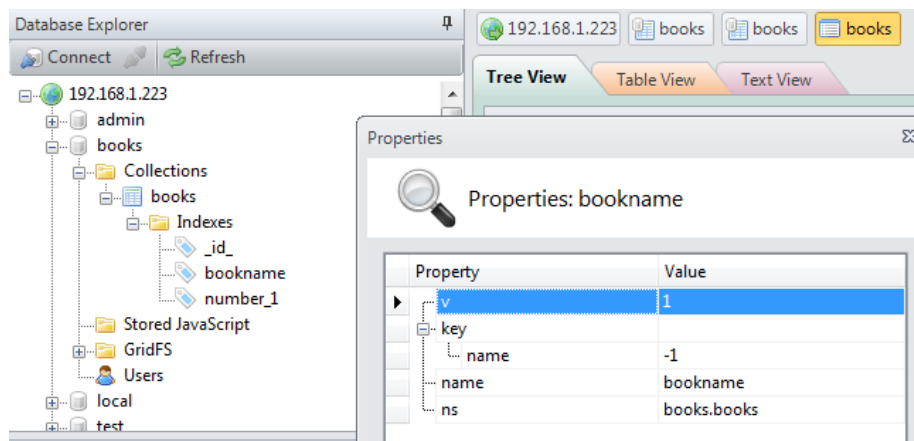
c) 索引的名称

3.1 用 VUE 查看索引名称



3.2 创建索引同时指定索引的名字

```
> db.books.ensureIndex({name:-1},{name:"bookname"})
{
  "createdCollectionAutomatically" : false,
  "numIndexesBefore" : 2,
  "numIndexesAfter" : 3,
  "ok" : 1
}
```



d) 唯一索引

1. 查看/显示集合的索引

`db.collectionName.getIndexes()`或则 `db.system.indexes.find()`

1 如何解决文档 books 不能插入重复的数值，所以建立唯一索引

```
> db.books.ensureIndex({number:-1},{unique:true})
{
  "createdCollectionAutomatically" : false,
  "numIndexesBefore" : 2,
  "numIndexesAfter" : 3,
  "ok" : 1
}
[
  {
    "v" : 1,
    "key" : {
      "_id" : 1
    },
    "name" : "_id_",      #索引名
    "ns" : "survey.data" #集合名
  },
  {
    "v" : 1,
    "unique" : true,      #唯一索引
    "key" : {
```

```

        "sid" : 1,
        "user" : 1
    },
    "name" : "sid_1_user_1",
    "ns" : "survey.data"
}
]

```

试验

```
db.books.insert({name:" 1book" })
```

e) 创建索引：db.collections.ensureIndex({...})

> db.books.ensureIndex({"account.name":1}) #内嵌文档上创建索引。

```

{
  "createdCollectionAutomatically" : false,
  "numIndexesBefore" : 3,
  "numIndexesAfter" : 4,
  "ok" : 1
}

```

> db.books.ensureIndex({"age":1},{ "name":"idx_name"}) #指定索引名称

```

{
  "createdCollectionAutomatically" : false,
  "numIndexesBefore" : 4,
  "numIndexesAfter" : 5,
  "ok" : 1
}

```

>

db.books.ensureIndex({"name":1,"age":1},{ "name":"idx_name_age","background":true

}) #后台创建复合索引

```

{
  "createdCollectionAutomatically" : false,
  "numIndexesBefore" : 5,
  "numIndexesAfter" : 6,
  "ok" : 1
}

```

```
>
db.books.ensureIndex({"name":1,"age":1},{ "name":"uk_name_age","background":true,
"unique":true}) #后台创建唯一索引
```

```
{
  "createdCollectionAutomatically" : false,
  "numIndexesBefore" : 1,
  "numIndexesAfter" : 2,
  "ok" : 1
}
```

```
>
db.books.ensureIndex({"name":1,"age":1},{ "unique":true,"dropDups":true,"name":"uk
_name_age"}) #删除重复数据创建唯一索引，dropDups 在 3.0 里废弃。
```

```
{
  "createdCollectionAutomatically" : false,
  "numIndexesBefore" : 1,
  "numIndexesAfter" : 2,
  "ok" : 1
}
```

f) 设置索引时间

```
db.books.ensureIndex({"name":1},{expireAfterSeconds:60}) #创建 TTL 索引，过期时
间 60 秒，即 60 秒时间生成的数据会被删除。
```

g) 删除索引：dropIndex

```
{
  "v" : 1,
  "key" : {
    "number" : 1
  },
  "name" : "number_1",
  "ns" : "books.books"
}
```

```
db.books.dropIndex({"number" : 1}) #删除索引，指定"key"
```

```
db.books.dropIndex("number_1") #删除索引，指定"name"
```

```
db.books.dropIndex("") #删除索引，删除集合的全部索引
```

h) 剔除重复值

1 如果建议唯一索引之前已经有重复数值如何处理

```
db.books.ensureIndex({name:-1},{unique:true,dropDups:true})
```

系统会自动删除重复的索引记录

i) 重建索引：索引出现损坏需要重建。reindex

```
db.books.reIndex() #执行
```

j) Hint

1 如何强制查询使用指定的索引呢?

```
db.books.find({name:"1book",number:1}).hint({name:-1})
```

指定索引必须是已经创建了的索引

k) Explain

1. 如何详细查看本次查询使用那个索引和查询数据的状态信息

```
db.books.find({name:"1book"}).explain()
```

```
> db.books.find({name:"1book"}).explain()
```

```
{
  "cursor" : "BasicCursor",
  "isMultiKey" : false,
  "n" : 2,
  "nscannedObjects" : 400001,
  "nscanned" : 400001,
  "nscannedObjectsAllPlans" : 400001,
  "nscannedAllPlans" : 400001,
  "scanAndOrder" : false,
  "indexOnly" : false,
  "nYields" : 3125,
  "nChunkSkips" : 0,
  "millis" : 187,
  "server" : "product:27017",
  "filterSet" : false
}
```

"cursor" : "BtreeCursor name_-1 " 使用索引

"nscanned" : 1 查到几个文档

"millis" : 0 查询时间 0 是很不错的性能

十一、索引管理

a) system.indexes

1. 在 shell 查看数据库已经建立的索引

```
db.system.indexes.find()
db.system.namespaces.find()
```

b) 后台执行

1. 执行创建索引的过程会暂时锁表问题如何解决?

为了不影响查询我们可以叫索引的创建过程在后台

```
db.books.ensureIndex({name:-1},{background:true})
```

c) 删除索引

1. 批量和精确删除索引

```
db.runCommand({dropIndexes : " books" , index:" name_-1" })
```

```
db.runCommand({dropIndexes : " books" , index:" *" })
```

十二、空间索引

a) mongoDB 提供强大的空间索引可以查询出一定范围的地理坐标.看例子



准备数据


```
var map = [{  
  "gis" : {  
    "x" : 185,  
    "y" : 150  
  }  
}, {  
  "gis" : {  
    "x" : 70,  
    "y" : 180  
  }  
}, {  
  "gis" : {  
    "x" : 75,  
    "y" : 180  
  }  
}, {  
  "gis" : {  
    "x" : 185,  
    "y" : 185  
  }  
}, {  
  "gis" : {  
    "x" : 65,  
    "y" : 185  
  }  
}, {  
  "gis" : {  
    "x" : 50,  
    "y" : 50  
  }  
}, {  
  "gis" : {  
    "x" : 50,  
    "y" : 50  
  }  
}, {  
  "gis" : {  
    "x" : 60,  
    "y" : 55  
  }  
}, {  
  "gis" : {  
    "x" : 65,  
    "y" : 80  
  }  
}
```

```

    }
  },{
    "gis": {
      "x": 55,
      "y": 80
    }
  },{
    "gis": {
      "x": 0,
      "y": 0
    }
  },{
    "gis": {
      "x": 0,
      "y": 200
    }
  },{
    "gis": {
      "x": 200,
      "y": 0
    }
  },{
    "gis": {
      "x": 200,
      "y": 200
    }
  }
}
for(var i = 0;i<map.length;i++){
  db.map.insert(map[i])
}

```

1.查询出距离点(70,180)最近的 3 个点

添加 2D 索引 最大 max 最小值 min

```

> db.map.ensureIndex({"gis":"2d"},{min:-1,max:201})
{
  "createdCollectionAutomatically" : false,
  "numIndexesBefore" : 1,
  "numIndexesAfter" : 2,
  "ok" : 1
}

```

默认会建立一个[-180,180]之间的 2D 索引

1.查询点(70,180)最近的 3 个点

```
b.map.find({"gis":{"$near":[70,180]}},{gis:1,_id:0}).limit(3)
```

2.查询以点(50,50)和点(190,190)为对角线的正方形中的所有的点

```
db.map.find({"gis":{"$within":{"$box":[[50,50],[190,190]]}}},{_id:0,gis:1})
```

3.查询出以圆心为(56,80)半径为 50 规则下的圆心面积中的点

```
db.map.find({"gis":{"$within":{"$center":[[56,80],50]}},{_id:0,gis:1})
```

```
> db.map.find({"gis":{"$within":{"$center":[[56,80],50]}},{_id:0,gis:1})
"gis" : { "x" : 55, "y" : 80 } }
"gis" : { "x" : 50, "y" : 50 } }
"gis" : { "x" : 50, "y" : 50 } }
"gis" : { "x" : 60, "y" : 55 } }
"gis" : { "x" : 65, "y" : 80 } }
> db.map.find({"gis":{"$within":{"$box":[[50,50],[190,190]]}}},{_id:0,gis:1})
{ "gis" : { "x" : 185, "y" : 150 } }
{ "gis" : { "x" : 75, "y" : 180 } }
{ "gis" : { "x" : 70, "y" : 180 } }
{ "gis" : { "x" : 65, "y" : 185 } }
{ "gis" : { "x" : 65, "y" : 80 } }
{ "gis" : { "x" : 55, "y" : 80 } }
{ "gis" : { "x" : 60, "y" : 55 } }
{ "gis" : { "x" : 50, "y" : 50 } }
{ "gis" : { "x" : 50, "y" : 50 } }
{ "gis" : { "x" : 185, "y" : 185 } }
>
```

十三、Count Distinct Group

数据准备：

```
var persons = [{
  name:"jim",
  age:25,
  email:"75431457@qq.com",
  c:89,m:96,e:87,
  country:"USA",
  books:["JS","C++","EXTJS","MONGODB"]
},
```

```
{
  name:"tom",
  age:25,
  email:"214557457@qq.com",
  c:75,m:66,e:97,
  country:"USA",
  books:["PHP","JAVA","EXTJS","C++"]
},
{
  name:"lili",
  age:26,
  email:"344521457@qq.com",
  c:75,m:63,e:97,
  country:"USA",
  books:["JS","JAVA","C#","MONGODB"]
},
{
  name:"zhangsan",
  age:27,
  email:"2145567457@qq.com",
  c:89,m:86,e:67,
  country:"China",
  books:["JS","JAVA","EXTJS","MONGODB"]
},
{
  name:"lisi",
  age:26,
  email:"274521457@qq.com",
  c:53,m:96,e:83,
  country:"China",
  books:["JS","C#","PHP","MONGODB"]
},
{
  name:"wangwu",
  age:27,
  email:"65621457@qq.com",
  c:45,m:65,e:99,
  country:"China",
  books:["JS","JAVA","C++","MONGODB"]
},
{
  name:"zhaoliu",
  age:27,
  email:"214521457@qq.com",
```

```

        c:99,m:96,e:97,
        country:"China",
        books:["JS","JAVA","EXTJS","PHP"]
    },
    {
        name:"piaoyingjun",
        age:26,
        email:"piaoyingjun@uspcat.com",
        c:39,m:54,e:53,
        country:"Korea",
        books:["JS","C#","EXTJS","MONGODB"]
    },
    {
        name:"lizhenxian",
        age:27,
        email:"lizhenxian@uspcat.com",
        c:35,m:56,e:47,
        country:"Korea",
        books:["JS","JAVA","EXTJS","MONGODB"]
    },
    {
        name:"lixiaoli",
        age:21,
        email:"lixiaoli@uspcat.com",
        c:36,m:86,e:32,
        country:"Korea",
        books:["JS","JAVA","PHP","MONGODB"]
    },
    {
        name:"zhangsuying",
        age:22,
        email:"zhangsuying@uspcat.com",
        c:45,m:63,e:77,
        country:"Korea",
        books:["JS","JAVA","C#","MONGODB"]
    }
}

for(var i = 0;i<persons.length;i++){
    db.persons.insert(persons[i])
}

var persons = db.persons.find({name:"jim"})
while(persons.hasNext()){
    obj = persons.next();
    print(obj.books.length)
}

```

1.Count

请查询 persons 中美国学生的人数.

```
db.persons.find({country:"USA"}).count()
```

2.Distinct

请查询出 persons 中一共有多少个国家分别是什么.

```
db.runCommand({distinct:"persons ", key:"country"}).values
```

3.Group

语法:

```
db.runCommand({group:{  
    ns:集合名字,  
  
    Key:分组的键对象,  
  
    Initial:初始化累加器,  
  
    $reduce:组分解器,  
  
    Condition:条件,  
  
    Finalize:组完成器  
}})
```

分组首先会按照 key 进行分组,每组的 每一个文档全要执行\$reduce 的方法,

他接收 2 个参数一个是组内本条记录,一个是累加器数据.

3.1 请查出 persons 中每个国家学生数学成绩最好的学生信息(必须在 90 以上)

```
db.runCommand({group:{  
    ns:"persons",  
  
    key:{"country":true}, // 针对分组的字段  
  
    initial:{m:0},  
  
    $reduce:function(doc,prev){ // doc 本组文档数据    prev 累加器  
  
        if(doc.m > prev.m){  
            prev.m = doc.m;  
        }  
    }  
}})
```

```

        prev.name = doc.name;
        prev.country = doc.country;
    }
},
condition:{m:{$gt:90}}
}))

```

3.2 在 3.1 要求基础之上吧没个人的信息链接起来写一个描述赋值到 m 上

```

db.runCommand({group:{
  ns:"persons",
  key:{"country":true},
  initial:{m:0},
  $reduce:function(doc,prev){
    if(doc.m > prev.m){
      prev.m = doc.m;
      prev.name = doc.name;
      prev.country = doc.country;
    }
  },
  finalize:function(prev){
    prev.m = prev.name+" Math scores "+prev.m
  },
  condition:{m:{$gt:90}}
}))

```

4.用函数格式化分组的键

4.1 如果集合中出现键 Counrty 和 counTry 同时存在那分组有点麻烦这要如何解决呢

```

db.persons.insert({
  name:"USPCAT",
  age:27,
  email:"2145567457@qq.com",
  c:89,m:100,e:67,
  counTry:"China",
  books:["JS","JAVA","EXTJS","MONGODB"]
})

```

```

db.runCommand({group:{
  ns:"persons",
  $keyf:function(doc){
    if(doc.counTry){
      return {country:doc.counTry}
    }else{
      return {country:doc.country}
    }
  }
})

```

```

    }
  },
  initial:{m:0},
  $reduce:function(doc,prev){
    if(doc.m > prev.m){
      prev.m = doc.m;
      prev.name = doc.name;
      if(doc.country){
        prev.country = doc.country;
      }else{
        prev.country = doc.counTry;
      }
    }
  },
  finalize:function(prev){
    prev.m = prev.name+" Math scores "+prev.m
  },
  condition:{m:{$gt:90}}
})

```

十四、数据库命令操作

1.命令执行器 runCommand

1.1 用命令执行完成一次删除表的操作

```
> db.runCommand({drop:"persons"})    //执行命令
```

```
{ "ns" : "persons.persons", "nIndexesWas" : 1, "ok" : 1 }    //结果
```

2.如何查询 mongoDB 为我们提供的命令

1.在 shell 中执行 db.listCommands()

2.访问网址 http://localhost:28017/_commands

必须加上 --rest 选项启动 默认端口 +1000

--rest 配置文件 rest = true 设置为 true 在 MongoDB 默认会开启一个 HTTP 协议提供

REST 的服务 (nohttpinterface = false) , 端口是 Server 端口加上 1000 , 即 28017 。

3.常用命令举例

3.1 查询服务器版本号和主机操作系统

```
db.runCommand({buildInfo:1})
```

3.2 查询执行集合的详细信息,大小,空间,索引等.....

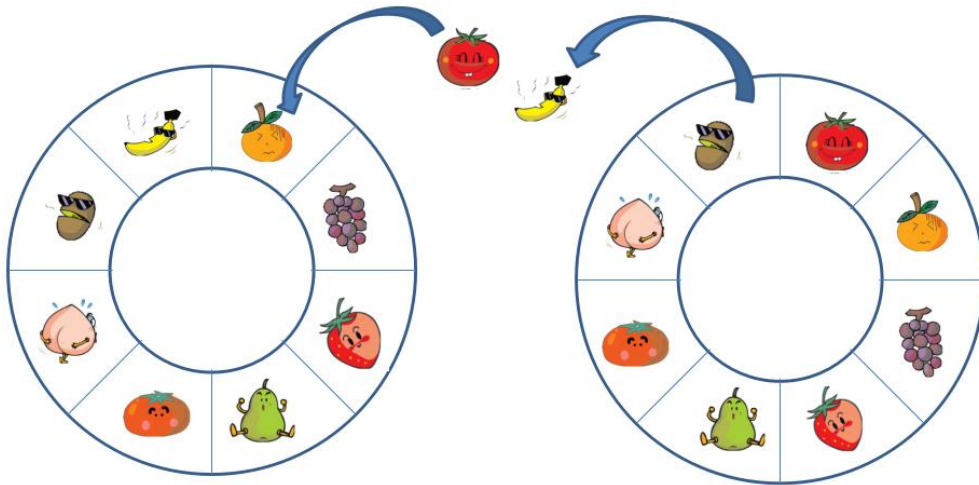
```
db.runCommand({collStats:"persons"})
```

3.3 查看操作本集合最后一次错误信息

```
db.runCommand({getLastError:"persons"})
```

十五、固定集合特性

1.固定集合概念



备注：长度固定，当有一个元素添加时，在末尾的元素将被抛出

2.固定特性

2.1 固定集合默认是没有索引的就算是_id 也是没有索引的

2.2 由于不需分配新的空间他的插入速度是非常快的

2.3 固定集合的顺是确定的导致查询速度是非常快的

2.4 最适合的是应用就是日志管理

3.创建固定集合

3.1 创建一个新的固定集合要求大小是 100 个字节,可以存储文档 10 个

```
db.createCollection("mycoll",{size:100,capped:true,max:10})
```

3.2 把一个普通集合转换成固定集合

```
db.runCommand({convertToCapped:" persons" ,size:100000})
```

4.反向排序,默认是插入顺序排序.

4.1 查询固定集合 mycoll 并且反向排序

```
db.mycoll.find().sort({$natural:-1})
```

5.尾部游标,可惜 shell 不支持 java 和 php 等驱动是支持的

5.1 尾部游标概念

这是个特殊的只能用到固定级和身上的游标,他在没有结果的时候

也不回自动销毁他是一直等待结果的到来


十六、GridFS 文件系统

1.概念

GridFS 是 mongoDB 自带的文件系统他用二进制的形式存储文件

大型文件系统的绝大多是特性 GridFS 全可以完成

2.利用的工具 （可视化工具建议使用: <https://robomongo.org/>）

 mongofiles.exe

3.使用 GridFS

3.1 查看 GridFS 的所有功能

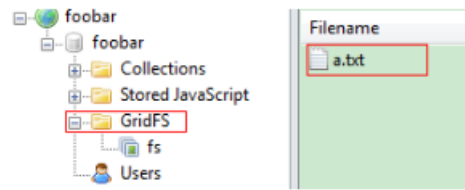
```
cmd □ mongofiles
```

3.2 上传一个文件

```
mongofiles -d foobar -l "E:\a.txt" put "a.txt "
```

3.3 查看 GridFS 的文件存储状态

利用 VUE 查看



集合查看

`db.fs.chunks.find()` 和 `db.fs.files.find()` 存储了文件系统的所有文件信息

3.4 查看文件内容

```
C:\Users\thinkpad>mongofiles -d foobar get "a.txt "
```

VUE 可以查看,shell 无法打开文件

3.5 查看所有文件

```
mongofiles -d foobar list
```

3.5 删除已经存在的文件 VUE 中操作

```
mongofiles -d foobar delete 'a.txt'
```

十七、服务器端脚本

1.Eval

1.1 服务器端运行 eval

```
db.eval("function(name){ return name}","uspcat")
```

2.Javascript 的存储

2.1 在服务上保存 js 变量活着函数共全局调用

1.把变量加载到特殊集合 system.js 中

```
db.system.js.insert({_id:name,value:" uspcat" })
```

2.调用

```
db.eval("return name;")
```

System.js 相当于 Oracle 中的存储过程,因为 value 不单单可以写变量

还可以写函数体也就是 javascript 代码

十八、MongoDB 启动配置详讲

1.启动项 mongod --help

--dbpath	指定数据库的目录,默认在 window 下是 c:\data\db\
--port	指定服务器监听的端口号码,默认是 27017
--fork	用守护进程的方式启动 mongoDB
--logpath	指定日志的输出路径,默认是控制台
--config	指定启动项用文件的路径
--auth	用安全认证方式启动数据库

1.1 利用 config 配置文件来启动数据库改变端口为 8888

mongodb.conf 文件

```
dbpath = D:\software\mongod\db  
port = 8888
```

启动文件

```
mongod.exe --config mongodb.conf
```

shell 文件

```
mongo 127.0.0.1:8888
```

2.停止 mongoDB 服务

1.1ctrl+c 组合键可以关闭数据库

1.2admin 数据库命令关闭数据

```
db.admin.shutdownServer()
```

十九、导出，导入，运行时备份

1.导出数据(中断其他操作)

打开 CMD

利用 mongoexport

- d 指明使用的库
- c 指明要导出的表
- o 指明要导出的文件名
- csv 制定导出的 csv 格式
- q 过滤导出
- type <json|csv|tsv>

1.1 把数据好 foobar 中的 persons 导出

```
mongoexport -d foobar -c persons -o D:/persons.json
```

1.2 导出其他主机数据库的文档

```
mongoexport --host 192.168.0.16 --port 37017
```

2.导入数据(中断其他操作)

API

<http://cn.docs.mongodb.org/manual/reference/mongoimport/>

2.1 到入 persons 文件

```
mongoimport --db foobar --collection persons --file d:/persons.json
```

3.运行时备份 mongodump

API

<http://cn.docs.mongodb.org/manual/reference/mongodump/>

1.1 导出 127.0.0.1 服务下的 27017 下的 foobar 数据库

```
mongodump --host 127.0.0.1:27017 -d foobar -o d:/foobar
```

4.运行时恢复 mongorestore

API

<http://cn.docs.mongodb.org/manual/reference/mongorestore/>

2.1 删除原本的数据库用刚才导出的数据库恢复

```
db.dropDatabase()
```

```
mongorestore --host 127.0.0.1:27017 -d foobar --directoryperdb d:/foobar/foobar
```

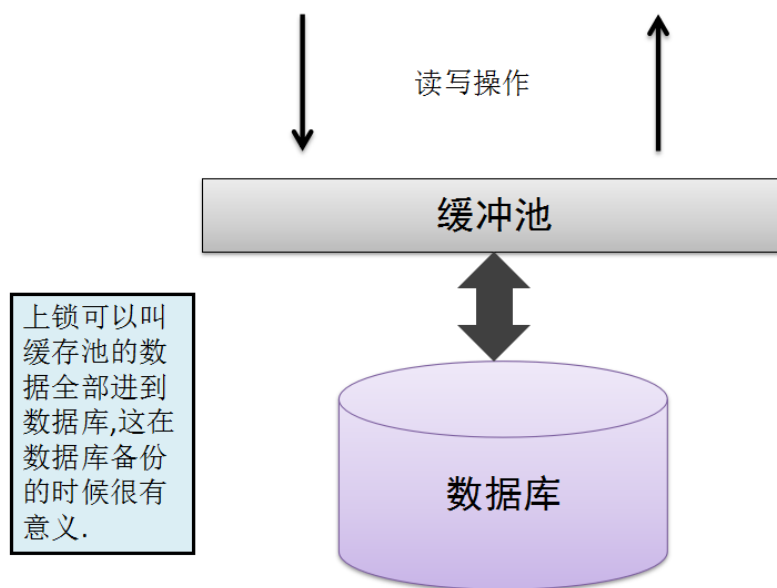
5.懒人备份

mongoDB 是文件数据库这其实就可以用拷贝文件的方式进行备份

二十、Fsync 锁，数据修复

1.Fsync 的使用

先来看看 mongoDB 的简单结构



2.上锁和解锁

上锁

```
db.runCommand({fsync:1,lock:1});
```

解锁

```
db.currentOp()
```

3.数据修复

当停电等不可逆转灾难来临的时候,由于 mongodb 的存储结构导致

会产生垃圾数据,在数据恢复以后这垃圾数据依然存在,这是数据库

提供一个自我修复的能力.使用起来很简单

```
db.repairDatabase()
```

二十一、用户管理，安全认证

1.添加一个用户

1.1 为 admin 添加 uspcat 用户和 foobar 数据库的 yunfengcheng 用户

```
use admin
```

```
db.addUser( "用户名" ," 密码" );
```

```
use foobar
```

```
db.addUser( "用户名" ," 密码" );
```

2.启用用户

```
db.auth( "名称" ," 密码" )
```

3.安全检查 --auth

非 foobar 是不能操作数据库的

```
> use foobar
switched to db foobar
> db.persons.find()
error: {
  "err" : "unauthorized db:foobar lock type:-1 client:127.0.0.1",
  "code" : 10057
}
```

启用该数据库中的用户才能访问

```
> db.auth<"yunfengcheng","123">
1
> db.persons.find()
{ "_id" : ObjectId<"501661020360ae34141abf69">, "age" :
++", "EXTJS", "MONGODB", "ORACLE" 1, "c" : 89, "country"
```

非 admin 数据库的用户不能使用数据库命令

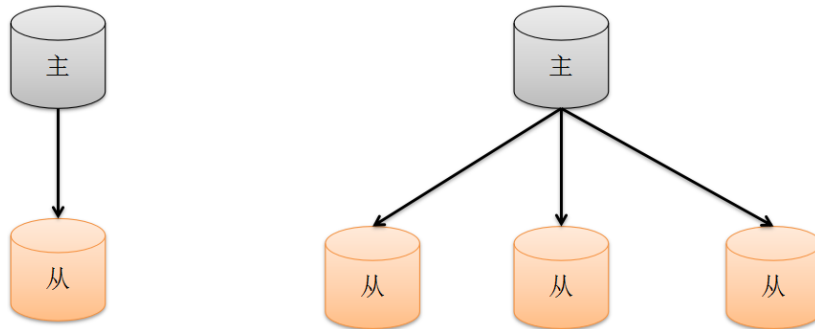
```
> use admin
switched to db admin
> db.auth<"uspcat","123">
1
> show dbs
admin    0.078125GB
foobar   0.078125GB
local    8.07421875GB
test     0.078125GB
```

4.用户删除操作

```
db.system.users.remove({user:"yunfengcheng"});
```

二十二、主从复制

1.主从复制是一个简单的数据库同步备份的集群技术.



1.1 在数据库集群中要明确的知道谁是主服务器,主服务器只有一台.

1.2 从服务器要知道自己的数据源也就是对于的主服务是谁.

1.3--master 用来确定主服务器,--slave 和 --source 来控制从服务器

2.主从复制集群案例

8888.conf

dbpath = D:\software\mongod\01\8888 主数据库地址

port = 8888 主数据库端口号

bind_ip = 127.0.0.1 主数据库所在服务器

master = true 确定我是主服务器

7777.conf

dbpath = D:\software\mongod\01\7777 从数据库地址

port = 7777 从数据库端口号

bind_ip = 127.0.0.1 从数据库所在服务器

source = 127.0.0.1:8888 确定我数据库端口

slave = true 确定自己是从服务器



Mongod 服务器

mongod --config 7777.conf

mongod --config 8888.conf

Mongo 客户端

mongo 127.0.0.1:7777

mongo 127.0.0.1:8888

3.主从复制的其他设置项

--only 从节点□ 指定复制某个数据库默认是复制全部数据库

--slavedelay 从节点□ 设置主数据库同步数据的延迟单位是秒)

--fastsync 从节点□ 以主数据库的节点快照为节点启动从数据库

--autoresync 从节点□ 如果不同步则从新同步数据库

--oplogSize 主节点□ 设置oplog 的大小(主节点操作记录存储到 local 的 oplog 中)

3.利用 shell 动态添加和删除从节点

```
> use local
switched to db local
> show collections
me
sources
system.indexes
> db.sources.find()
{ "_id" : ObjectId("50207daf8eefa30d7ca9ee33"), "host" : "127.0.0.1:8888", "source" : "main", "syncedTo" : { "t" : 1344309071000, "i" : 1 } }
>
```

不难看出从节点中关于主节点的信息全部存到 local 的 sources 的集合中

我们只要对集合进行操作就可以动态操作主从关系

挂接主节点:操作之前只留下从数据库服务

```
db.sources.insert({ "host" : " 127.0.0.1:8888" })
```

删除已经挂接的主节点:操作之前只留下从数据库服务

```
db.sources.remove({ "host" : " 127.0.0.1:8888" })
```

检查：从库查询，报错

```
> db.persons.find()
```

```
Error: error: { "ok" : 0, "errmsg" : "not master and slaveOk=false", "code" : 13435 }
```

问题：

对于 replica set 中的 secondary 节点默认是不可读的。在写多读少的应用中，使用

Replica Sets 来实现读写分离。通过在连接时指定或者在主库指定 `slaveOk`，由 Secondary 来分担读的压力，Primary 只承担写操作。

解放方案：

有两种方法实现从机的查询：

第一种方法：`db.getMongo().setSlaveOk();`

第二种方法：`rs.slaveOk();`

有一个缺点就是，下次再通过 mongo 进入实例的时候，查询仍然会报错，为此可

`vi ~/.mongorc.js`

增加一行 `rs.slaveOk()`

测试：

`/web/mongodb/bin/mongod --config /web/mongodb/etc/8888.conf` //主库

`/web/mongodb/bin/mongod --config /web/mongodb/etc/7777.conf` //从库

主从操作：

`use persons`

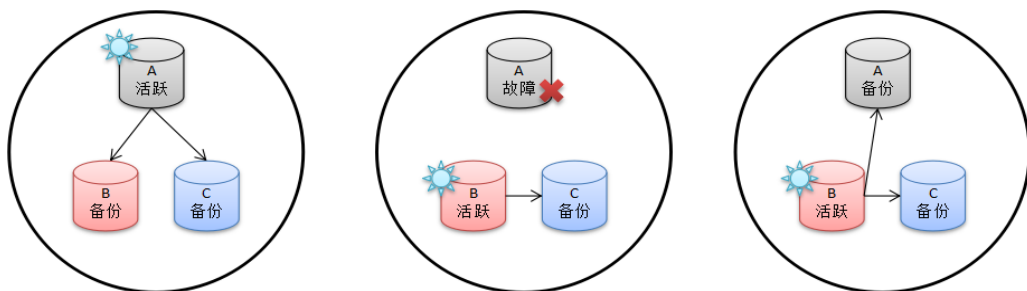
`db.persons.insert({name:"zhangsan"})`

从库操作：

`db.persons.find()` //报错请查阅上面解决方案

二十三、副本集

1.副本集概念



1.1 第一张图表明 A 是活跃的 B 和 C 是用于备份的

1.2 第二张图当 A 出现了故障,这时候集群根据权重算法推选出 B 为活跃的数据库

1.3 第三张图当 A 恢复后他自动又会变为备份数据库

2.副本集概念

配置文件

A.conf

dbpath = D:\software\MongoDBDATA\07\1111

port = 1111 #端口

bind_ip = 127.0.0.1 #服务地址

replSet = child/127.0.0.1:2222 #设定同伴

B.conf

dbpath = D:\software\MongoDBDATA\07\2222

port = 2222

bind_ip = 127.0.0.1

replSet = child/127.0.0.1:3333

C.conf

dbpath = D:\software\MongoDBDATA\07\3333

port = 3333

bind_ip = 127.0.0.1

replSet = child/127.0.0.1:1111

服务器

mongod --config A.conf

mongod --config B.conf

mongod --config C.conf

客户端

mongo 127.0.0.1:1111

mongo 127.0.0.1:2222

mongo 127.0.0.1:3333

3.初始化副本集（任意一个服务器进行初始化）

use admin

db.runCommand({"replSetInitiate":

{

"_id":'child',

"members":[{"

"_id":1,

```

        "host":"127.0.0.1:1111"
      },{
        "_id":2,
        "host":"127.0.0.1:2222"
      },{
        "_id":3,
        "host":"127.0.0.1:3333"
      }
    ]
  }
})

```

备注：在没有设置权重时，根据内置算法随机选取活跃数据库，**只有该数据库才能查询**

操作，活跃数据库才能对应操作

4.查看副本集状态

```
rs.status()
```

5.Shell 展示

A 活跃

```

child:PRIMARY> use persons
switched to db persons
child:PRIMARY> db.persons.insert({name:"zhangsan"})
WriteResult({ "nInserted" : 1 })
child:PRIMARY> db.persons.find()
{ "_id" : ObjectId("57ade14b03b29fad69e71991"), "name" : "zhangsan" }
child:PRIMARY>

```

B 备份

```

child:SECONDARY> use persons
switched to db persons
child:SECONDARY> db.persons.find()
{ "_id" : ObjectId("57ade14b03b29fad69e71991"), "name" : "zhangsan" }
child:SECONDARY>

```

C 备份

```

child:SECONDARY> use persons
switched to db persons
child:SECONDARY> db.persons.find()
{ "_id" : ObjectId("57ade14b03b29fad69e71991"), "name" : "zhangsan" }
child:SECONDARY>

```

测试：然后终止任何一个服务器，终止所有的客户端并重启查看

6.节点和初始化高级参数

standard 常规节点:参与投票有可能成为活跃节点

passive 副本节点:参与投票,但是不能成为活跃节点

arbiter 仲裁节点:只是参与投票不复制节点也不能成为活跃节点

7.高级参数

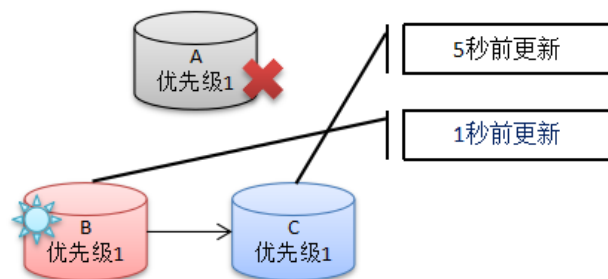
Priority 0 到 1000 之间 ,0 代表是副本节点 ,1 到 1000 是常规节点

arbiterOnly : true 仲裁节点

使用用法

```
members":[{"_id":1,
  "host":"127.0.0.1:1111 ",
  arbiterOnly : true
}]"
```

8.优先级相同时仲裁组建的规则



9.读写分离操作□ 扩展读

6.1 一般情况下作为副本的节点是不能进行数据库读操作的,但是在读取密集型的

系统中读写分离是十分必要的

```
C:\Users\thinkpad\Desktop\副本集>mongo 127.0.0.1:2222
MongoDB shell version: 2.0.6
connecting to: 127.0.0.1:2222/test
SECONDARY> use foobar
switched to db foobar
SECONDARY> db.persons.find(<)
error: < "$err" : "not master and slaveok=false", "code" : 13435 >
SECONDARY> _
```

备注：提示如图，根据在主从复制时查阅解决

<http://stackoverflow.com/questions/8990158/mongodb-replicates-and-error-err-not-master-and-slaveok-false-code>

6.2 设置读写分离

slaveOkay: true

很遗憾他在 shell 中无法掩饰,这个特性是被写到 MongoDB 的

驱动程序中的,在 java 和 node 等其他语言中可以完成

10.Oplog

他是被存储在本地数据库 local 中的,他的每一个文档保证这一个节点操作如果想故障

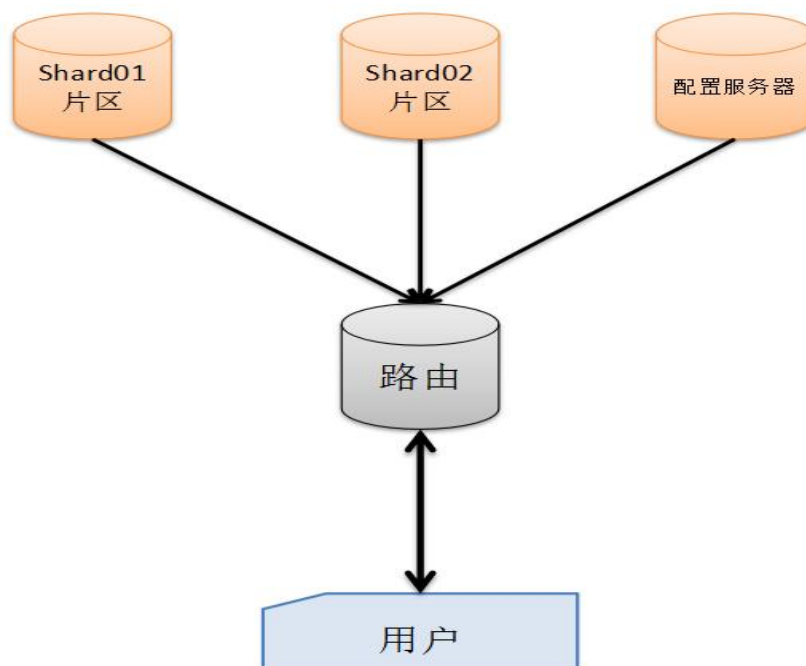
恢复可以更彻底 oplog 可已经尽量设置大一些用来保存更多的操作信息。

改变 oplog 大小

主库 --master --oplogSize size、

二十四、分片（解决大数据，分布存储）

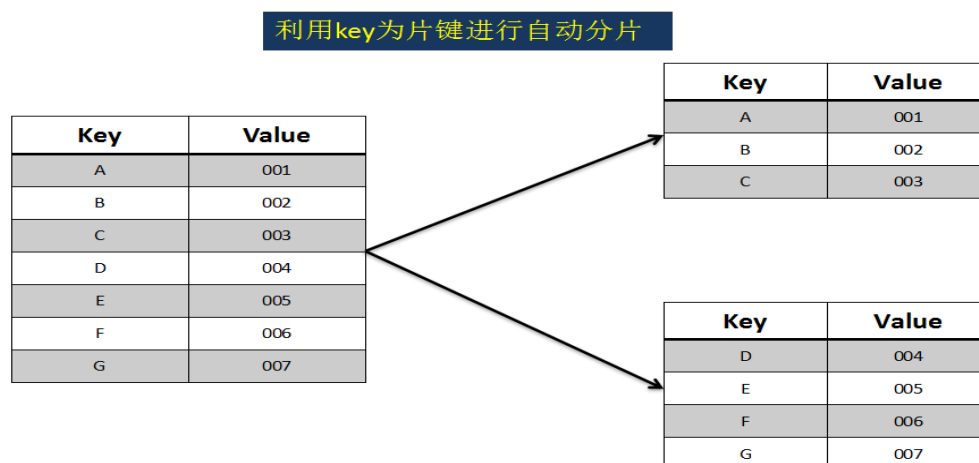
1.插入负载技术□ 分片架构图



备注：先读取配置服务器信息，然后根据配置识别数据，最后更加数据处理到对应信息到片区中。以内存为缓存区上锁才能存入数据库中。

2. 片键的概念和用处

看下面这个普通的集合和分片后的结果



3. 什么时候用到分片呢?

- 3.1 机器的磁盘空间不足
- 3.2 单个的 mongoDB 服务器已经不能满足大量的插入操作
- 3.3 想通过把大数据放到内存中来提高性能
- 3.4 比如微博等应用可以配置访问 IP 指定到不同的分片

4. 分片步骤

4.1 创建一个配置服务器

配置服务器

配置文件

```
dbpath=/web/mongodb/config_db  
port=2000  
bind_ip = 127.0.0.1
```

启动服务器

```
/web/mongodb/bin/mongod --config /web/mongodb/etc/config.conf
```

4.2 创建路由服务器,并且连接配置服务器

路由器是调用 mongos 命令

路由器 （路由服务器监听配置服务器）

```
/web/mongodb/bin/mongos --port 1000 --configdb 127.0.0.1:2000 #路由服务器
```

```
/web/mongodb/bin/mongo 127.0.0.1:1000/admin #客户端（先别执行）
```

4.3 添加 2 个分片数据库（8081 和 8082）

```
vim fp8081.conf
```

```
dbpath=/web/mongodb/8081 #配置文件
```

```
port=8081
```

```
bind_ip = 127.0.0.1
```

```
/web/mongodb/bin/mongod --config /web/mongodb/etc/fp8081.conf #服务器
```

```
/web/mongodb/bin/mongo 127.0.0.1:8081/admin #客户端（先别执行）
```

```
-----  
vim fp8082.conf
```

```
dbpath=/web/mongodb/8082 #配置文件
```

```
port=8082
```

```
bind_ip = 127.0.0.1
```

```
/web/mongodb/bin/mongod --config /web/mongodb/etc/fp8082.conf #服务器
```

```
/web/mongodb/bin/mongo 127.0.0.1:8082/admin #客户端（先别执行）
```

4.5 利用路由为集群添加分片(允许本地访问)

在配置路由服务器和配置服务器后为两个片区添加路由形成聚群效果

打开路由器客户端，执行命令（将分片服务器连接路由器）

```
/web/mongodb/bin/mongo 127.0.0.1:1000/admin
```

```
mongos> db.runCommand({addshard:"127.0.0.1:8081",allowLocal:true})
```

```
{ "shardAdded" : "shard0000", "ok" : 1 }
```

```
mongos> db.runCommand({addshard:"127.0.0.1:8082",allowLocal:true})
```

```
{ "shardAdded" : "shard0001", "ok" : 1 }
```

```
mongos>
```


切记之前不能使用任何数据库语句

```
[root@localhost ~]# /web/mongodb/bin/mongo 127.0.0.1:1000/admin
MongoDB shell version: 3.2.8
connecting to: 127.0.0.1:1000/admin
Server has startup warnings:
2016-08-12T23:31:53.554+0800 I CONTROL [main] ** WARNING: You are running
2016-08-12T23:31:53.554+0800 I CONTROL [main]
mongos> db.runCommand({addshard:"127.0.0.1:8081",allowLocal:true})
{ "shardAdded" : "shard0000", "ok" : 1 }
mongos> db.runCommand({addshard:"127.0.0.1:8082",allowLocal:true})
{ "shardAdded" : "shard0001", "ok" : 1 }
mongos>
```

4.6 打开数据分片功能,为数据库 foobar 打开分片功能

同样在路由器客户端执行

use admin

db.runCommand({"enablesharding":"foobar"})

```
mongos> use admin
switched to db admin
mongos> db.runCommand({"enablesharding":"foobar"})
{ "ok" : 1 }
mongos>
```

4.7 对集合进行分片

db.runCommand({"shardcollection":"foobar.bar","key":{"_id":1}})

利用 key 为片键进行自动分片（对数据 foobar 数据库中 bar 集合进行分片了）

```
mongos> use admin
switched to db admin
mongos> db.runCommand({"enablesharding":"foobar"})
{ "ok" : 1 }
mongos> db.runCommand({"shardcollection":"foobar.bar","key":{"_id":1}})
{ "collectionsharded" : "foobar.bar", "ok" : 1 }
mongos>
```

4.8 利用大数据量进行测试 (800000 条) --->数据量要足够大

```
function add(){
  var i = 0;
  for(;i<200000;i++){
    db.bar.insert({"age":i+10,"name":"jim"})
  }
}
```

```
function add2(){
  var i = 0;
  for(;i<200000;i++){
```

```

        db.bar.insert({"age":12,"name":"tom"+i})
    }
}

```

```

function add3(){
    var i = 0;
    for(;i<200000;i++){
        db.bar.insert({"age":12,"name":"lili"+i})
    }
}

```

```

mongos> show dbs
admin    0.008GB
config  0.001GB
foobar   0.000GB
mongos> use foobar
switched to db foobar
mongos> function add(){
...   var i = 0;
...   for(;i<10000;i++){
...       db.bar.insert({"age":i+10,"name":"jim"})
...   }
... }
mongos> add()
mongos> add()
mongos> function add2(){
...   var i = 0;
...   for(;i<200000;i++){
...       db.bar.insert({"age":12,"name":"tom"+i})
...   }
... }
mongos> add2()
mongos> db.bar.find()

```

备注：插入的过程需要 2~3 分钟时间，请耐心等待

```

mongos> add2()
mongos> use foobar
switched to db foobar
mongos> db.bar.find().count()
420001
mongos>

```

执行：/web/mongodb/bin/mongo 127.0.0.1:8081

```

> use foobar
switched to db foobar
> db.bar.find().count()
335690
>

```

执行：/web/mongodb/bin/mongo 127.0.0.1:8082

```
> use foobar
switched to db foobar
> db.bar.find().count()
84310
>
```

插入测试数据后，根据查看各个分片数据库，都是内置算法进行分布式存储的

5.查看配置库对于分片服务器的配置存储

如果在路由服务器执行 db.printShardingStatus()查看分片情况时。

报错如下：

"errmsg" : "Surprised to discover that 127.0.0.1:2000 does not believe it is a config server"

因为 db.printShardingStatus()命令必须在配置服务器上执行

执行：（配置化服务器）

```
/web/mongodb/bin/mongo 127.0.0.1:2000
db.printShardingStatus()
```

```
> db.printShardingStatus()
--- Sharding Status ---
  sharding version: {
    "_id" : 1,
    "minCompatibleVersion" : 5,
    "currentVersion" : 6,
    "clusterId" : ObjectId("57ade71d18f9930696aae3e8")
  }
  shards:
    { "_id" : "shard0000", "host" : "127.0.0.1:8081" }
    { "_id" : "shard0001", "host" : "127.0.0.1:8082" }
  active mongoses:
    "3.2.8" : 1
```

6.查看集群对 bar 的自动分片机制配置信息

在配置服务器上执行命令

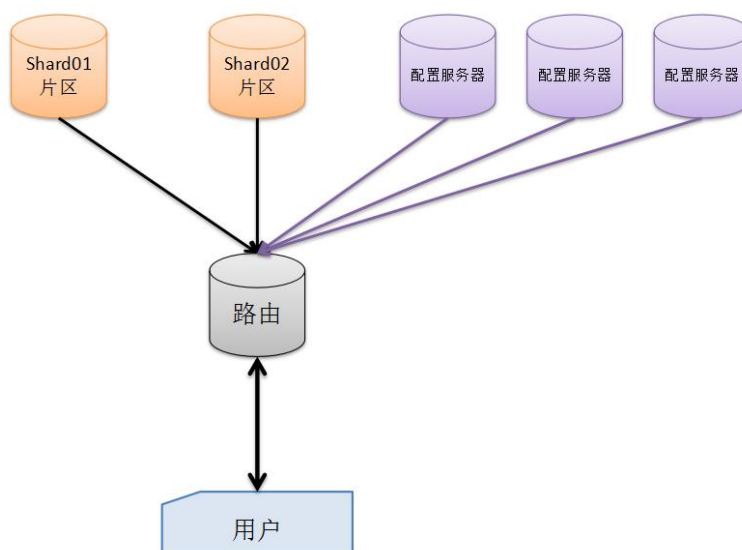
```

> show dbs
admin    0.008GB
config   0.001GB
local    0.000GB
> use cofig
switched to db cofig
> show dbs
admin    0.008GB
config   0.001GB
local    0.000GB
> use config
switched to db config
> show collections
changelog
chunks
collections
databases
lockpings
locks
mongos
settings
shards
tags
version
> db.shards.find()
{ "_id" : "shard0000", "host" : "127.0.0.1:8081" }
{ "_id" : "shard0001", "host" : "127.0.0.1:8082" }
>

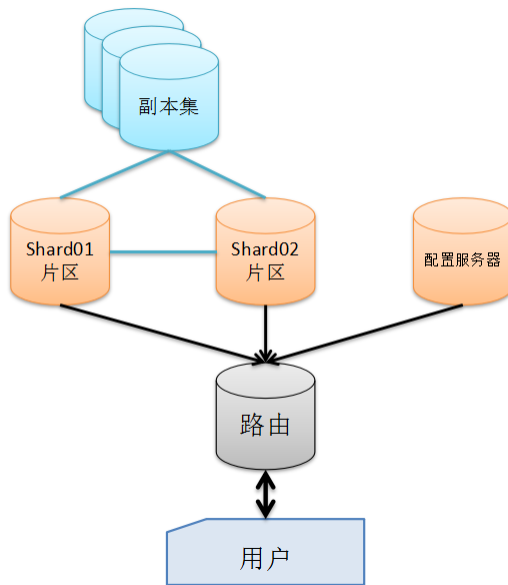
```

分片规则，按照日期和国际化

7. 保险起见配置服务器集群



8.分片与副本集一起使用



二十五、JAVA 操作 mongoDB

1. 查询所有数据库数据

DataBase.java

```
package com.mongodb.text;
import java.net.UnknownHostException;
import com.mongodb.DB;
import com.mongodb.DBCollection;
import com.mongodb.DBCursor;
import com.mongodbDBObject;
import com.mongodb.Mongo;
import com.mongodb.MongoException;
import com.mongodb.util.JSON;

public class DataBase {
    public static void main(String[] args)
        throws UnknownHostException, MongoException {

        //1.建立一个 Mongo 的数据库连接对象
        Mongo mg = new Mongo("127.0.0.1:27017");

        //查询所有的 Database
        for (String name : mg.getDatabaseNames()) {
            System.out.println("dbName: " + name);
        }
    }
}
```

```

//2.创建相关数据库的连接

DB db = mg.getDB("foobar");

//查询数据库所有的集合

for (String name : db.getCollectionNames()) {
    System.out.println("collectionName: " + name);
}

DBCollection users = db.getCollection("persons");

//查询所有的数据

DBCursor cur = users.find();
while (cur.hasNext()) {
DBObject object = cur.next();
System.out.println(object.get("name"));
}
System.out.println(cur.count());
System.out.println(cur.getCursorId());
System.out.println(JSON.serialize(cur));
}
}

```

2. MongoDB 数据库基本操作

```

package com.mongodb.text;

import java.net.UnknownHostException;
import java.util.List;

import org.bson.types.ObjectId;

import com.mongodb.BasicDBObject;
import com.mongodb.DB;
import com.mongodb.DBCollection;
import com.mongodb.DBCursor;
import com.mongodb.DBObject;
import com.mongodb.Mongo;
import com.mongodb.MongoException;

public class MongoDB {

    //1.建立一个 Mongo 的数据库连接对象

```

```

static Mongo connection = null;

//2.创建相关数据库的连接

static DB db = null;
public MongoDB(String dbName) throws UnknownHostException, MongoException{
    connection = new Mongo("127.0.0.1:27017");
    db = connection.getDB(dbName);
}
public static void main(String[] args) throws UnknownHostException, MongoException {

    //实例化

    MongoDB mongoDb = new MongoDB("foobar");
    /**

        * 1.创建一个名字叫 javadb 的数据库

        */
    //    mongoDb.createCollection("javadb");
    /**

        * 2.为集合 javadb 添加一条数据

        */
    //    DBObject dbs = new BasicDBObject();
    //    dbs.put("name", "uspcat.com");
    //    dbs.put("age", 2);
    //    List<String> books = new ArrayList<String>();
    //    books.add("EXTJS");
    //    books.add("MONGODB");
    //    dbs.put("books", books);
    //    mongoDb.insert(dbs, "javadb");
    /**

        * 3.批量插入数据

        */
    //    List<DBObject> dbObjects = new ArrayList<DBObject>();
    //    DBObject jim = new BasicDBObject("name", "jim");
    //    DBObject lisi = new BasicDBObject("name", "lisi");
    //    dbObjects.add(jim);
    //    dbObjects.add(lisi);
    //    mongoDb.insertBatch(dbObjects, "javadb");
    /**

        * 4.根据 ID 删除数据

        */
    //    mongoDb.deleteById("502870dab9c368bf5b151a04", "javadb");

```

```

/**
 * 5.根据条件删除数据
 */
//DBObject lisi = new BasicDBObject();
//lisi.put("name", "lisi");
//int count = mongoDb.deleteByDbs(lisi, "javadb");

//System.out.println("删除数据的条数是: "+count);

/**
 * 6.更新操作,为集合增加 email 属性
 */
//DBObject update = new BasicDBObject();
//update.put("$set",
//        new BasicDBObject("eamil", "uspcat@126.com"));
//mongoDb.update(new BasicDBObject(),
//        update, false, true, "javadb");

/**
 * 7.查询出 persons 集合中的 name 和 age
 */
//DBObject keys = new BasicDBObject();
//keys.put("_id", false);
//keys.put("name", true);
//keys.put("age", true);
//DBCursor cursor = mongoDb.find(null, keys, "persons");
//while (cursor.hasNext()) {
//    DBObject object = cursor.next();
//    System.out.println(object.get("name"));
//}

//7.查询出年龄大于 26 岁并且英语成绩小于 80 分

//DBObject ref = new BasicDBObject();
//ref.put("age", new BasicDBObject("$gte", 26));
//ref.put("e", new BasicDBObject("$lte", 80));
//DBCursor cursor = mongoDb.find(ref, null, "persons");
//while (cursor.hasNext()) {
//    DBObject object = cursor.next();
//    System.out.print(object.get("name")+"-->");
//    System.out.print(object.get("age")+"-->");
//    System.out.println(object.get("e"));
//}

/**

```


* 8.分页例子

```
*/
DBCursor cursor = mongoDb.find(null, null, 0, 3, "persons");
while (cursor.hasNext()) {
    DBObject object = cursor.next();
    System.out.print(object.get("name")+"-->");
    System.out.print(object.get("age")+"-->");
    System.out.println(object.get("e"));
}

//关闭连接对象
connection.close();
}
/**
```

* 穿件一个数据库集合

* @param collName 集合名称

* @param db 数据库实例

```
*/
public void createCollection(String collName){
    DBObject dbs = new BasicDBObject();
    db.createCollection("javadb", dbs);
}
/**
```

* 为相应的集合添加数据

* @param dbs

* @param collName

```
*/
public void insert(DBObject dbs,String collName){

    //1.得到集合
    DBCollection coll = db.getCollection(collName);

    //2.插入操作
    coll.insert(dbs);
}
/**
```

* 为集合批量插入数据

* @param dbses

```

    * @param collName
    */
    public void insertBatch(List<DBObject> dbses,String collName){

        //1.得到集合

        DBCollection coll = db.getCollection(collName);

        //2.插入操作

        coll.insert(dbses);
    }
    /**

    * 根据 id 删除数据

    * @param id
    * @param collName

    * @return 返回影响的数据条数

    */
    public int deleteById(String id,String collName){

        //1.得到集合

        DBCollection coll = db.getCollection(collName);
        DBObject dbs = new BasicDBObject("_id", new ObjectId(id));
        int count = coll.remove(dbs).getN();
        return count;
    }
    /**

    * 根据条件删除数据

    * @param id
    * @param collName

    * @return 返回影响的数据条数

    */
    public int deleteByDbs(DBObject dbs,String collName){

        //1.得到集合

        DBCollection coll = db.getCollection(collName);
        int count = coll.remove(dbs).getN();
        return count;
    }
    /**

    * 更新数据

```

```

* @param find 查询器

* @param update 更新器

* @param upsert 更新或插入

* @param multi 是否批量更新

* @param collName 集合名称

* @return 返回影响的数据条数

*/
public int update(DBObject find,
                  DBObject update,
                  boolean upsert,
                  boolean multi,
                  String collName){

    //1.得到集合

    DBCollection coll = db.getCollection(collName);
    int count = coll.update(find, update, upsert, multi).getN();
    return count;
}
/**

* 查询器(分页)

* @param ref
* @param keys
* @param start
* @param limit
* @return

*/
public DBCursor find(DBObject ref,
                     DBObject keys,
                     int start,
                     int limit,
                     String collName){
    DBCursor cur = find(ref, keys, collName);
    return cur.limit(limit).skip(start);
}
/**

* 查询器(不分页)

```

```
* @param ref
* @param keys
* @param start
* @param limit
* @param collName
* @return
*/
public DBCursor find(DBObject ref,
                    DBObject keys,
                    String collName){

    //1.得到集合

    DBCollection coll = db.getCollection(collName);
    DBCursor cur = coll.find(ref, keys);
    return cur;
}
}
```