

## MongoDB 简介

MongoDB 是一个基于分布式文件存储的数据库。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。

MongoDB 是一个介于关系数据库和非关系数据库之间的产品，是非关系数据库当中功能最丰富，最像关系数据库的。他支持的数据结构非常松散，是类似 json 的 bson 格式，因此可以存储比较复杂的数据类型。Mongo 最大的特点是他支持的查询语言非常强大，其语法有点类似于面向对象的查询语言，几乎可以实现类似关系数据库单表查询的绝大部分功能，而且还支持对数据建立索引。

它的特点是高性能、易部署、易使用，存储数据非常方便。主要功能特性有：

- \*面向集合存储，易存储对象类型的数据。
- \*模式自由。
- \*支持动态查询。
- \*支持完全索引，包含内部对象。
- \*支持查询。
- \*支持复制和故障恢复。
- \*使用高效的二进制数据存储，包括大型对象（如视频等）。
- \*自动处理碎片，以支持云计算层次的扩展性。
- \*支持 RUBY, PYTHON, JAVA, C++, PHP, C# 等多种语言。
- \*文件存储格式为 BSON（一种 JSON 的扩展）。
- \*可通过网络访问。

## 在 WIN7 下安装运行 mongodb

### 1、下载

下载地址：<http://www.mongodb.org/downloads>

①：根据业界规则，偶数为“稳定版”（如：1.6.X, 1.8.X），奇数为“开发版”（如：1.7.X, 1.9.X），这两个版本的区别相信大家都知道吧。

②：32bit 的 mongod 最大只能存放 2G 的数据，64bit 就没有限制。

## 2、安装

安装非常简单，解压就行了，我解压后，放在 D:/MongoDB 目录下。

为了命令行的方便，可以把 D:/MongoDB/bin 加到系统环境变量的 path 中了。

下载 Windows 64-bit 版本并解压缩，程序文件都在 bin 目录中，其它两个目录分别是 C++调用的是的头文件和库文件。bin 目录中包含如下几个程序：

1. mongo.exe，命令行客户端工具。
2. mongod.exe，数据库服务程序。
3. mongodump.exe，数据库备份程序。
4. mongoexport.exe，数据导出工具。
5. mongofiles.exe，GridFS 工具。
6. mongoimport.exe，数据导入工具。
7. mongorestore.exe，数据库恢复工具。
8. mongos.exe，貌似是性能检测工具。

## 3、运行

先创建一个 D:/MongoDB/data 文件夹，然后运行下面命令

```
D:\>mongod --dbpath D:/MongoDB/data
```

## 4、安装 Windows 服务

每次运行 mongod --dbpath D:/MongoDB/data 命令行来启动 MongoDB 实在是不方便，把它作为 Windows 服务，这样就方便多了。

创建一个日志文件夹，D:/MongoDB/logs，然后运行下面命令

```
D:\MongoDB\bin>mongod --logpath D:\MongoDB\logs\MongoDB.log  
--logappend --dbpath D:\MongoDB\data --serviceName MongoDB --install
```

**注意：**这条命令要到 MongoDB 的 bin 目录下运行，刚开始的时候，我就直接在 D:\下运行，结果服务的可执行目录为【"D:\mongod"

--logpath "D:\MongoDB\logs\MongoDB.log" --logappend --dbpath "D:\MongoDB\data" --directoryperdb --service 】，肯定是不对的。

该命令行指定了日志文件：D:\MongoDB\logs\MongoDB.log，日志是以追加的方式输出的；

数据文件目录：D:\MongoDB\data，并且参数--directoryperdb 说明每个 DB 都会新建一个目录；

Windows 服务的名称：MongoDB；

以上的三个参数都是可以根据自己的情况而定的，可以通过 `mongod --help` 查看更多的参数。

最后是安装参数：--install，与之相对的是--remove

**启动 MongoDB:** `net start MongoDB`

**停止 MongoDB:** `net stop MongoDB`

**删除 MongoDB:** `sc delete MongoDB`

### **mongodb 安装常见问题**

1. 启动服务器出现 1067 错误，删除 data 目录下的 mongod.lock 重启就可以了
2. 启动时提示缺少 vcruntime140.dll，需要安装下 mongodb\bin 目录下的 vcredist\_x64.exe
3. 提示拒绝访问，需要找到 cmd.exe 以管理员身份运行，执行上述命令
4. 安装 vcredist\_x64.exe 失败，提示“ 0x80070424 指定服务未安装”需要更新系统补丁包 Windows6.1-KB2999226-x64.cab

下载后，执行下面命令

**`DISM /Online /Add-Package /PackagePath:C:\ Windows6.1-KB2999226-x64.cab`**

### **mongodb 命令的基本操作**

我们再开一个 cmd，输入 mongo 命令打开 shell，其实这个 shell 就是 mongodb 的客户端，同时也是一个 js 的编译器

```
C:\Windows\system32\cmd.exe - mongo
Microsoft Windows [版本 6.1.7601]
版权所有 (c) 2009 Microsoft Corporation。保留所有权利。

C:\Users\Administrator>E:

E:\>cd mongodb\bin

E:\mongodb\bin>mongo
MongoDB shell version: 2.0.2
connecting to: test
> _
```

在 MongoDB 的 cmd 窗口中，执行如下命令：

- |                              |                            |
|------------------------------|----------------------------|
| (1) show dbs                 | 查看当前有哪些数据库                 |
| (2) use yourdb               | 使用某个数据库                    |
| (3) show collections         | 查看本数据库中有哪些 collection（表）   |
| (4) db.yourCollection.find() | 查看 yourCollection 这张表存储的数据 |

#### <1> insert 操作

好，数据库有了，下一步就是集合，这里就取集合名为“person”，要注意的就是文档是一个 json 的扩展（Bson）形式。

```
C:\Windows\system32\cmd.exe - mongo

E:\mongodb\bin>mongo
MongoDB shell version: 2.0.2
connecting to: test
> db.person.insert(<<"name":"jack","age":20>>)
> db.person.insert(<<"name":"joe","age":25>>)
>
```

#### <2> find 操作

我们将数据插入后，肯定是要 find 出来，不然插了也白插，这里要注意两点：

- ① “\_id”：这个字段是数据库默认给我们加的 GUID，目的就是保证数据的唯一性。
- ② 严格的按照 Bson 的形式书写文档，不过也没关系，错误提示还是很强大的。

```
管理员: C:\Windows\system32\cmd.exe - mongo

E:\mongodb\bin>mongo
MongoDB shell version: 2.0.2
connecting to: test
> db.person.insert(<<"name":"jack","age":20>>)
> db.person.insert(<<"name":"joe","age":25>>)
> db.person.find()
{ "_id" : ObjectId("4f3e77e41a15c6974206a629"), "name" : "jack", "age" : 20 }
{ "_id" : ObjectId("4f3e77f81a15c6974206a62a"), "name" : "joe", "age" : 25 }
>
> db.person.find(<<"name":joe>>)
Fri Feb 17 23:55:54 ReferenceError: joe is not defined (shell):1
> db.person.find(<<"name":"joe">>)
{ "_id" : ObjectId("4f3e77f81a15c6974206a62a"), "name" : "joe", "age" : 25 }
>
=
```

### <3> update 操作

update 方法的第一个参数为“查找的条件”，第二个参数为“更新的值”，学过 C#，相信还是很好理解的。

```
管理员: C:\Windows\system32\cmd.exe - mongo

connecting to: test
> db.person.insert(<<"name":"jack","age":20>>)
> db.person.insert(<<"name":"joe","age":25>>)
> db.person.find()
{ "_id" : ObjectId("4f3e77e41a15c6974206a629"), "name" : "jack", "age" : 20 }
{ "_id" : ObjectId("4f3e77f81a15c6974206a62a"), "name" : "joe", "age" : 25 }
>
> db.person.find(<<"name":joe>>)
Fri Feb 17 23:55:54 ReferenceError: joe is not defined (shell):1
> db.person.find(<<"name":"joe">>)
{ "_id" : ObjectId("4f3e77f81a15c6974206a62a"), "name" : "joe", "age" : 25 }
>
> db.person.update(<<"name":"joe">>,<<"name":"joe","age":30>>)
> db.person.find(<<"name":"joe">>)
{ "_id" : ObjectId("4f3e77f81a15c6974206a62a"), "name" : "joe", "age" : 30 }
>
=
```

### <4> remove 操作

remove 中如果不带参数将删除所有数据，呵呵，很危险的操作，在 mongodb 中是一个不可撤回的操作，三思而后行。

```
>
> db.person.update(<{"name":"joe"},<{"name":"joe","age":30}>)
> db.person.find(<{"name":"joe"}>)
{ "_id" : ObjectId<"4f3e77f81a15c6974206a62a">, "name" : "joe", "age" : 30 }
>
>
> db.person.remove(<{"name":"joe"}>)
> db.person.find()
{ "_id" : ObjectId<"4f3e77e41a15c6974206a629">, "name" : "jack", "age" : 20 }
>
> db.person.remove()
> db.person.find()
> db.person.count()
0
>
```

## 扩展 mongodb 基本操作

### 一： Insert 操作

前面也说过,文档是采用“K-V”格式存储的，如果大家对 JSON 比较熟悉的话，我相信学 mongodb 是手到擒来，我们知道 JSON 里面 Value

可能是“字符串”，可能是“数组”，又有可能是内嵌的一个 JSON 对象，相同的方式也适合于 BSON。

常见的插入操作也就两种形式存在：“单条插入”和“批量插入”。

#### ① 单条插入

先前也说了，mongo 命令打开的是一个 javascript shell。所以 js 的语法在这里面都行得通，看起来是不是很牛 X。

```
管理员: C:\Windows\system32\cmd.exe - mongo

E:\mongodb\bin>mongo
MongoDB shell version: 2.0.2
connecting to: test
> var single={"name":"jack","password":"12345","age":20,
...         "address":{"province":"anhui","city":"hefei"},
...         "favourite":["apple","banana"]}
>
> db.user.insert(single)
>
> single.name="joe"
joe
> single.age=25
25
> single.address={"province":"jiangsu","city":"nanjing"}
{ "province" : "jiangsu", "city" : "nanjing" }
> single.favourite=["money","mm"]
[ "money", "mm" ]
>
> db.user.insert(single)
>
> db.user.find(<
< "_id" : ObjectId<"4f3fc51633c814a220459299">, "name" : "jack", "password" : "12345"
< "_id" : ObjectId<"4f3fc5b333c814a22045929a">, "name" : "joe", "password" : "12345"
>
```

## ② 批量插入

这玩意跟“单条插入”的差异相信大家应该知道，由于 mongodb 中没有提供给 shell 的“批量插入方法”，没关系，各个语言的 driver 都打通

了跟 mongodb 内部的批量插入方法，因为该方法是不可或缺的，如果大家非要模拟下批量插入的话，可以自己写了 for 循环，里面就是 insert。

## 二：Find 操作

日常开发中，我们玩查询，玩的最多的也就是二类：

①： >, >=, <, <=, !=, =。

②： And, OR, In, NotIn

这些操作在 mongodb 里面都封装好了，下面就一一介绍：

<1>"\$gt", "\$gte", "\$lt", "\$lte", "\$ne", "没有特殊关键字"，这些跟上面是一一对应的，举几个例子。

```

>
> /* find age>22 */
... db.user.find(<<"age":{$gt:22}>>)
{ "_id" : ObjectId("4f3fc5b33c814a22045929a"), "name" : "joe", "password" : "12345", "age" : 25, "address" : "123456789"}
>
> /* find age<22 */
... db.user.find(<<"age":{$lt:22}>>)
{ "_id" : ObjectId("4f3fc51633c814a220459299"), "name" : "jack", "password" : "12345", "age" : 20, "address" : "123456789"}
>
> /* find age!=22 */
... db.user.find(<<"age":{$ne:22}>>)
{ "_id" : ObjectId("4f3fc51633c814a220459299"), "name" : "jack", "password" : "12345", "age" : 20, "address" : "123456789"}
{ "_id" : ObjectId("4f3fc5b33c814a22045929a"), "name" : "joe", "password" : "12345", "age" : 25, "address" : "123456789"}
>
> /* find age==20 */
... db.user.find(<<"age":20>>)
{ "_id" : ObjectId("4f3fc51633c814a220459299"), "name" : "jack", "password" : "12345", "age" : 20, "address" : "123456789"}
=

```

<2> "无关键字", "\$or", "\$in", "\$nin" 同样我也是举几个例子

```

>
> /* find name='jack' && province='anhui' */
... db.user.find(<<"name":"jack","address.province":"anhui">>)
{ "_id" : ObjectId("4f3fc51633c814a220459299"), "name" : "jack", "password" : "12345", "age" : 20, "address" : "123456789"}
>
> /* find province='anhui' || province='guangdong' */
... db.user.find(<<{$or:[{"address.province":"anhui"}, {"address.province":"guangdong"}]}>>)
{ "_id" : ObjectId("4f3fc51633c814a220459299"), "name" : "jack", "password" : "12345", "age" : 20, "address" : "123456789"}
>
> /* find province='anhui' || province='guangdong' */
...
... /* */
... /* find province in ["anhui","guangdong"] */
... db.user.find(<<"address.province":{$in:["anhui","guangdong"]}>>)
{ "_id" : ObjectId("4f3fc51633c814a220459299"), "name" : "jack", "password" : "12345", "age" : 20, "address" : "123456789"}
>
> /* find province not in ["anhui","guangdong"] */
... db.user.find(<<"address.province":{$nin:["anhui","guangdong"]}>>)
{ "_id" : ObjectId("4f3fc5b33c814a22045929a"), "name" : "joe", "password" : "12345", "age" : 25, "address" : "123456789"}
>

```

<3> 在 mongodb 中还有一个特殊的匹配，那就是“正则表达式”，这玩意威力很强的。



```

>
>
> /* find name startwith 'j' and endwith 'e' */
... db.user.find(<<"name":/^j/, "name":/e$/>>)
< {"_id" : ObjectId<"4f3fc5b333c814a22045929a">, "name" : "joe", "password" : "12345"}
>

```

<4> 有时查询很复杂,很蛋疼,不过没关系,mongodb 给我们祭出了大招,它就是\$where,为什么这么说,是因为\$where 中的 value

就是我们非常熟悉,非常热爱的 js 来助我们一马平川。

```

> /* find name='jack' */
... db.user.find(<<$where:function() { return this.name=='jack' }>>)
< {"_id" : ObjectId<"4f3fc51633c814a220459299">, "name" : "jack", "password" : "12345"}
>

```

### 三: Update 操作

更新操作无非也就两种,整体更新和局部更新,使用场合相信大家也清楚。

#### <1> 整体更新

不知道大家可还记得,我在上一篇使用 update 的时候,其实那种 update 是属于整体更新。

```

>
> db.user.find()
< {"_id" : ObjectId<"4f3fc51633c814a220459299">, "name" : "jack", "password" : "12345"}
< {"_id" : ObjectId<"4f3fc5b333c814a22045929a">, "name" : "joe", "password" : "12345"}
>
> /* update jack's age=30 */
> var model=db.user.findOne(<<"name":"jack">>)
> model.age=30
30
> db.user.update(<<"name":"jack">>,model)
> db.user.find()
< {"_id" : ObjectId<"4f3fc51633c814a220459299">, "name" : "jack", "password" : "12345"}
< {"_id" : ObjectId<"4f3fc5b333c814a22045929a">, "name" : "joe", "password" : "12345"}
>

```

#### <2> 局部更新

有时候我们仅仅需要更新一个字段，而不是整体更新，那么我们该如何做呢？easy 的问题，mongodb 中已经给我们提供了两个

修改器：\$inc 和 \$set。

### ① \$inc 修改器

\$inc 也就是 increase 的缩写，学过 sql server 的同学应该很熟悉，比如我们做一个在线用户状态记录，每次修改会在原有的基础上

自增 \$inc 指定的值，如果“文档”中没有此 key，则会创建 key，下面的例子一看就懂。

```
> db.user.find()
< {"_id" : ObjectId("4f3fc51633c814a220459299"), "name" : "jack", "password" : "12345", "age" : 30, "a
< {"_id" : ObjectId("4f3fc5b333c814a22045929a"), "name" : "joe", "password" : "12345", "age" : 25, "a
>
> /* add jack age to 60 */
... db.user.update(<"name": "jack">, <$inc: {"age": 30}>)
> db.user.find()
< {"_id" : ObjectId("4f3fc51633c814a220459299"), "name" : "jack", "password" : "12345", "age" : 60, "a
< {"_id" : ObjectId("4f3fc5b333c814a22045929a"), "name" : "joe", "password" : "12345", "age" : 25, "a
>
```

### ② \$set 修改器

啥也不说了，直接上代码

```
> db.user.find()
< {"_id" : ObjectId("4f3fc51633c814a220459299"), "name" : "jack", "password" : "12345", "age" : 30, "a
< {"_id" : ObjectId("4f3fc5b333c814a22045929a"), "name" : "joe", "password" : "12345", "age" : 25, "a
>
> /* update jack age=10 */
... db.user.update(<"name": "jack">, <$set: {"age": 10}>)
> db.user.find()
< {"_id" : ObjectId("4f3fc51633c814a220459299"), "name" : "jack", "password" : "12345", "age" : 10, "a
< {"_id" : ObjectId("4f3fc5b333c814a22045929a"), "name" : "joe", "password" : "12345", "age" : 25, "a
>
```

### <3> upsert 操作

这个可是 mongodb 创造出来的“词”，大家还记得 update 方法的第一次参数是“查询条件”吗？，那么这个 upsert 操作就是说：如果我

没有查到，我就在数据库里面新增一条，其实这样也有好处，就是避免了我在数据库里面判断是 update 还是 add 操作，使用起来很简单

将 update 的第三个参数设为 true 即可。

```

> db.user.find()
{ "_id" : ObjectId("4f3fc51633c814a220459299"), "name" : "jack", "password" : "12345" }
{ "_id" : ObjectId("4f3fc5b333c814a22045929a"), "name" : "joe", "password" : "12345" }
>
>
> db.user.update(<{"name":"jackson"},<{$inc:<{"age":1}>>,<true>)
> db.user.find()
{ "_id" : ObjectId("4f3fc51633c814a220459299"), "name" : "jack", "password" : "12345" }
{ "_id" : ObjectId("4f3fc5b333c814a22045929a"), "name" : "joe", "password" : "12345" }
{ "_id" : ObjectId("4f3fe3cb7af5bb1f9bbae0cb"), "age" : 1, "name" : "jackson" }
>
>

```

#### <4> 批量更新

在 mongodb 中如果匹配多条，默认的情况下只更新第一条，那么如果我们有需求必须批量更新，那么在 mongodb 中实现也是很简单

的，在 update 的第四个参数中设为 true 即可。例子就不举了。

##### 一：聚合

常见的聚合操作跟 sql server 一样，有：count, distinct, group, mapReduce。

#### <1> count

count 是最简单，最容易，也是最常用的聚合工具，它的使用跟我们 C# 里面的 count 使用简直一模一样。

```

> db.person.remove(<{"address":"beijing"}>)
>
> db.person.find()
{ "_id" : ObjectId("4f40a02c4b272b37b4d76f3f"), "name" : "jack", "age" : 20 }
{ "_id" : ObjectId("4f40a0374b272b37b4d76f40"), "name" : "jackson", "age" : 22 }
{ "_id" : ObjectId("4f40a03e4b272b37b4d76f41"), "name" : "joe", "age" : 26 }
{ "_id" : ObjectId("4f40a0454b272b37b4d76f42"), "name" : "mary", "age" : 20 }
{ "_id" : ObjectId("4f40a08d4b272b37b4d76f43"), "name" : "Alice", "age" : 22 }
{ "_id" : ObjectId("4f40a0ca4b272b37b4d76f44"), "name" : "Maria", "age" : 22 }
>
>
> db.person.count()
6
> db.person.count(<{"age":20}>)
2
>
>

```

#### <2> distinct

这个操作相信大家也是非常熟悉的，指定了谁，谁就不能重复，直接上图。

```

> db.person.find()
{ "_id" : ObjectId("4f40a02c4b272b37b4d76f3f"), "name" : "jack", "age" : 20 }
{ "_id" : ObjectId("4f40a0374b272b37b4d76f40"), "name" : "jackson", "age" : 22 }
{ "_id" : ObjectId("4f40a03e4b272b37b4d76f41"), "name" : "joe", "age" : 26 }
{ "_id" : ObjectId("4f40a0454b272b37b4d76f42"), "name" : "mary", "age" : 20 }
{ "_id" : ObjectId("4f40a08d4b272b37b4d76f43"), "name" : "Alice", "age" : 22 }
{ "_id" : ObjectId("4f40a0ca4b272b37b4d76f44"), "name" : "Maria", "age" : 22 }
>
> db.person.distinct("age")
[ 20, 22, 26 ]
>

```

### <3> group

在 mongodb 里面做 group 操作有点小复杂，不过大家对 sql server 里面的 group 比较熟悉的话还是一眼

能看的明白的，其实 group 操作本质上形成了一种“k-v”模型，就像 C#中的 Dictionary，好，有了这种思维，

我们来看看如何使用 group。

下面举的例子就是按照 age 进行 group 操作，value 为对应 age 的姓名。下面对这些参数介绍一下：

**key:** 这个就是分组的 key，我们这里是对年龄分组。

**initial:** 每组都分享一个“初始化函数”，特别注意：是每一组，比如这个的 age=20 的 value 的 list 分享一个

initial 函数，age=22 同样也分享一个 initial 函数。

**\$reduce:** 这个函数的第一个参数是当前的文档对象，第二个参数是上一次 function 操作的累计对象，第一次

为 initial 中的{"perosn": []}。有多少个文档，\$reduce 就会调用多少次。

```

> db.person.find()
< "_id" : ObjectId<"4f40a02c4b272b37b4d76f3f">, "name" : "jack", "age" : 20 >
< "_id" : ObjectId<"4f40a0374b272b37b4d76f40">, "name" : "jackson", "age" : 22 >
< "_id" : ObjectId<"4f40a03e4b272b37b4d76f41">, "name" : "joe", "age" : 26 >
< "_id" : ObjectId<"4f40a0454b272b37b4d76f42">, "name" : "mary", "age" : 20 >
< "_id" : ObjectId<"4f40a08d4b272b37b4d76f43">, "name" : "Alice", "age" : 22 >
< "_id" : ObjectId<"4f40a0ca4b272b37b4d76f44">, "name" : "Maria", "age" : 22 >
>
> db.person.group(<
... "key":<"age":true>,
... "initial":<"person":[]>,
... "$reduce":function(cur,prev)<
...     prev.person.push(cur.name);
... >
... >>
[
    <
        "age" : 20,
        "person" : [
            "jack",
            "mary"
        ]
    >,
    <
        "age" : 22,
        "person" : [
            "jackson",
            "Alice",
            "Maria"
        ]
    >,
    <
        "age" : 26,
        "person" : [
            "joe"
        ]
    >
]
>

```

看到上面的结果，是不是有点感觉，我们通过 **age** 查看到了相应的 **name** 人员，不过有时我们可能有如下的要求：

- ①：想过滤掉 **age>25** 一些人员。
- ②：有时 **person** 数组里面的人员太多，我想加上一个 **count** 属性标明一下。

针对上面的需求，在 **group** 里面还是很好办到的，因为 **group** 有这么两个可选参数：**condition** 和 **finalize**。

**condition**：这个就是过滤条件。

**finalize**：这是个函数，每一组文档执行完后，多会触发此方法，那么在每组集合里面加上 **count** 也就是它的活了。

```

> db.person.find()
{ "_id" : ObjectId<"4f40a02c4b272b37b4d76f3f">, "name" : "jack", "age" : 20 }
{ "_id" : ObjectId<"4f40a0374b272b37b4d76f40">, "name" : "jackson", "age" : 22 }
{ "_id" : ObjectId<"4f40a03e4b272b37b4d76f41">, "name" : "joe", "age" : 26 }
{ "_id" : ObjectId<"4f40a0454b272b37b4d76f42">, "name" : "mary", "age" : 20 }
{ "_id" : ObjectId<"4f40a08d4b272b37b4d76f43">, "name" : "Alice", "age" : 22 }
{ "_id" : ObjectId<"4f40a0ca4b272b37b4d76f44">, "name" : "Maria", "age" : 22 }
> db.person.group(<
... "key":{"age":true},
... "initial":{"person":[ ]},
... "reduce":function(doc,out){
...     out.person.push(doc.name);
... },
... "finalize":function(out){
...     out.count=out.person.length;
... },
... "condition":{"age":{"$lt:25}}
... })
[
  {
    "age" : 20,
    "person" : [
      "jack",
      "mary"
    ],
    "count" : 2
  },
  {
    "age" : 22,
    "person" : [
      "jackson",
      "Alice",
      "Maria"
    ],
    "count" : 3
  }
]
>

```

#### <4> mapReduce

这玩意算是聚合函数中最复杂的了，不过复杂也好，越复杂就越灵活。

mapReduce 其实是一种编程模型，用在分布式计算中，其中有一个“map”函数，一个“reduce”函数。

##### ① map:

这个称为映射函数，里面会调用 emit(key,value)，集合会按照你指定的 key 进行映射分组。

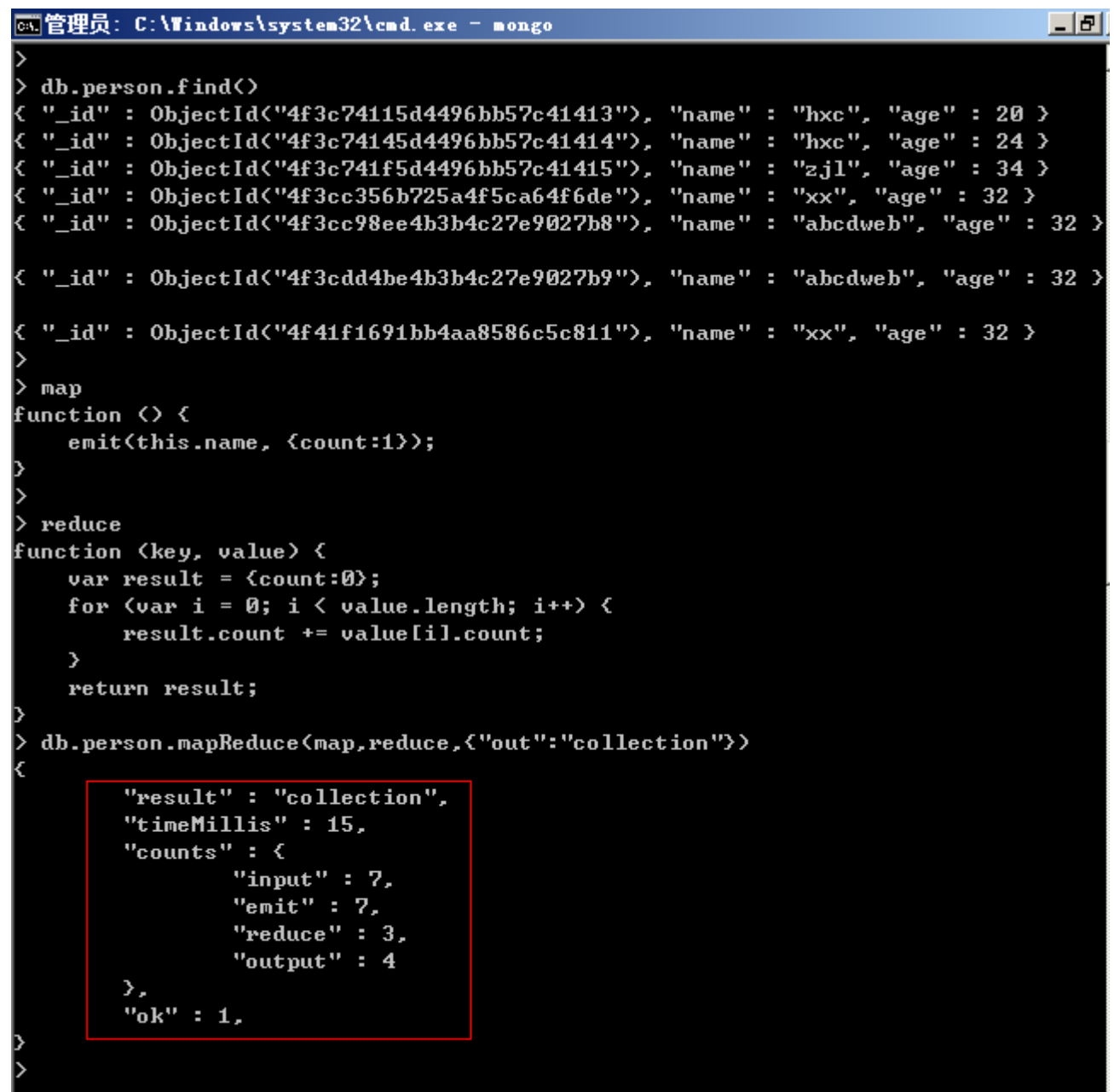
##### ② reduce:

这个称为简化函数，会对 map 分组后的数据进行分组简化，注意：在 reduce(key,value)中的 key 就是

emit 中的 key，vlaue 为 emit 分组后的 emit(value)的集合，这里也就是很多 {"count":1}的数组。

### ③ mapReduce:

这个就是最后执行的函数了，参数为 map，reduce 和一些可选参数。具体看图可知：



```
>
> db.person.find()
< "_id" : ObjectId<"4f3c74115d4496bb57c41413">, "name" : "hxc", "age" : 20 >
< "_id" : ObjectId<"4f3c74145d4496bb57c41414">, "name" : "hxc", "age" : 24 >
< "_id" : ObjectId<"4f3c741f5d4496bb57c41415">, "name" : "zjl", "age" : 34 >
< "_id" : ObjectId<"4f3cc356b725a4f5ca64f6de">, "name" : "xx", "age" : 32 >
< "_id" : ObjectId<"4f3cc98ee4b3b4c27e9027b8">, "name" : "abcdweb", "age" : 32 >
< "_id" : ObjectId<"4f3cdd4be4b3b4c27e9027b9">, "name" : "abcdweb", "age" : 32 >
< "_id" : ObjectId<"4f41f1691bb4aa8586c5c811">, "name" : "xx", "age" : 32 >
>
> map
function <> {
    emit<this.name, {count:1}>;
}
>
> reduce
function <key, value> {
    var result = {count:0};
    for <var i = 0; i < value.length; i++> {
        result.count += value[i].count;
    }
    return result;
}
> db.person.mapReduce<map,reduce,<"out":"collection">>
<
  "result" : "collection",
  "timeMillis" : 15,
  "counts" : {
    "input" : 7,
    "emit" : 7,
    "reduce" : 3,
    "output" : 4
  },
  "ok" : 1,
>
>
```

从图中我们可以看到如下信息：

result: "存放的集合名";

input:传入文档的个数。

emit: 此函数被调用的次数。

reduce: 此函数被调用的次数。

output:最后返回文档的个数。

最后我们看一下“collecton”集合里面按姓名分组的情况。

```
>
> db.collection.find()
{ "_id" : "abcdweb", "value" : { "count" : 2 } }
{ "_id" : "hxc", "value" : { "count" : 2 } }
{ "_id" : "xx", "value" : { "count" : 2 } }
{ "_id" : "zjl", "value" : { "count" : 1 } }
>
```

## 二：游标

mongodb 里面的游标有点类似我们说的 C#里面延迟执行，比如：

```
var list=db.person.find();
```

针对这样的操作，list 其实并没有获取到 person 中的文档，而是申明一个“查询结构”，等我们需要的时候通过

for 或者 next()一次性加载过来，然后让游标逐行读取，当我们枚举完了之后，游标销毁，之后我们在通过 list 获取时，

发现没有数据返回了。

```
>
> var list=db.person.find();
>
> list.forEach(function(x){
... print(x.name);
... })
hxc
hxc
zjl
xx
abcdweb
abcdweb
xx
> list
>
```

当然我们的“查询构造”还可以搞的复杂点，比如分页，排序都可以加进去。

```
var single=db.person.find().sort({"name",1}).skip(2).limit(2);
```



那么这样的“查询构造”可以在我们需要执行的时候执行，大大提高了不必要的花销。

```
> var single=db.person.find().sort(<{"name":1}>).skip(2).limit(3)
> single
< "_id" : ObjectId("4f3c74115d4496bb57c41413"), "name" : "hxc", "age" : 20 >
< "_id" : ObjectId("4f3c74145d4496bb57c41414"), "name" : "hxc", "age" : 24 >
< "_id" : ObjectId("4f3cc356b725a4f5ca64f6de"), "name" : "xx", "age" : 32 >
> ■
```

## Mongodb 索引应用

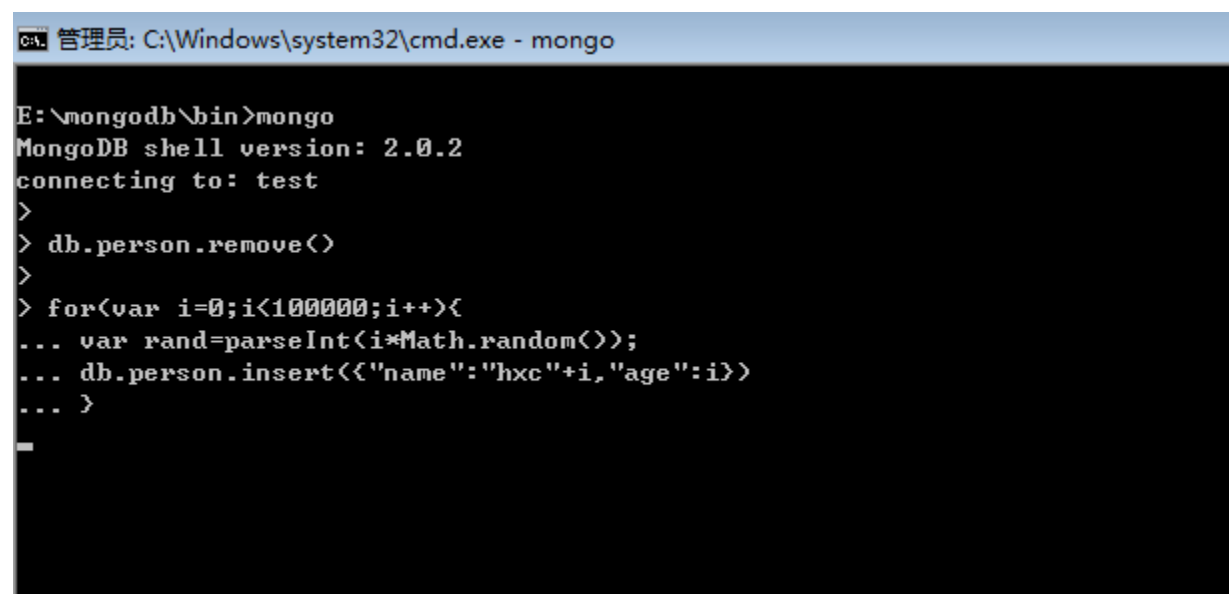
好，今天分享下 mongodb 中关于索引的基本操作，我们日常做开发都避免不了要对程序进行性能优化，而程序的操作无非就是 CURD，通常我们

又会花费 50%的时间在 R 上面，因为 Read 操作对用户来说是非常敏感的，处理不好就会被人唾弃，呵呵。

从算法上来说有 5 种经典的查找，具体的可以参见我的算法速成系列，这其中就包括我们今天所说的“索引查找”，如果大家对 sqlserver 比较了解

的话，相信索引查找能给我们带来什么样的性能提升吧。

我们首先插入 10w 数据，上图说话：



```
管理员: C:\Windows\system32\cmd.exe - mongo
E:\mongodb\bin>mongo
MongoDB shell version: 2.0.2
connecting to: test
>
> db.person.remove()
>
> for(var i=0;i<100000;i++){
... var rand=parseInt(i*Math.random());
... db.person.insert(<{"name":"hxc"+i,"age":i}>)
... }
■
```

### 一：性能分析函数（explain）

好了，数据已经插入成功，既然我们要做分析，肯定要有分析的工具，幸好 mongodb 中给我们提供了一个关键字叫做“explain”，那么怎么用呢？

还是看图，注意，这里的 name 字段没有建立任何索引，这里我就查询一个“name10000”的姓名。

```

> db.person.find(<{'name':'hxc'+10000}>)
{ "_id" : ObjectId<"4f4cee61c31a64c0b29bab31">, "name" : "hxc10000", "age" : 10000 }
>
> db.person.find(<{'name':'hxc'+10000}>).explain()
{
  "cursor" : "BasicCursor",
  "nscanned" : 100000,
  "nscannedObjects" : 100000,
  "n" : 1,
  "millis" : 114,
  "nYields" : 0,
  "nChunkSkips" : 0,
  "isMultiKey" : false,
  "indexOnly" : false,
  "indexBounds" : {
    }
  }
}
>

```

仔细看红色区域，有几个我们关心的 key。

**cursor:** 这里出现的是"BasicCursor",什么意思呢，就是说这里的查找采用的是“表扫描”，也就是顺序查找，很悲催啊。

**nscanned:** 这里是 10w，也就是说数据库浏览了 10w 个文档，很恐怖吧，这样玩的话让人受不了啊。

**n:** 这里是 1，也就是最终返回了 1 个文档。

**millis:** 这个就是我们最最最....关心的东西，总共耗时 114 毫秒。

## 二：建立索引（ensureIndex）

在 10w 条这么简单的集合中查找一个文档要 114 毫秒有一点点让人不能接收，好，那么我们该如何优化呢？mongodb 中给

我们带来了索引查找，看看能不能让我们的查询一飞冲天.....

```

> db.person.ensureIndex(<{"name":1}>)
> db.person.find(<{"name":"hxc"+10000}>).explain()
{
  "cursor" : "BtreeCursor name_1",
  "nscanned" : 1,
  "nscannedObjects" : 1,
  "n" : 1,
  "millis" : 1,
  "nYields" : 0,
  "nChunkSkips" : 0,
  "isMultiKey" : false,
  "indexOnly" : false,
  "indexBounds" : {
    "name" : [
      [
        "hxc10000",
        "hxc10000"
      ]
    ]
  }
}

```

这里我们使用了 `ensureIndex` 在 `name` 上建立了索引。“1”：表示按照 `name` 进行升序，“-1”：表示按照 `name` 进行降序。

我的神啊，再来看看这些敏感信息。

**cursor:** 这里出现的是“`BtreeCursor`”，这么牛 X，mongodb 采用 B 树的结构来存放索引，索引名为后面的“`name_1`”。

**nscanned:** 我擦，数据库只浏览了一个文档就 OK 了。

**n:** 直接定位返回。

**millis:** 看看这个时间真的不敢相信，秒秒杀。

通过这个例子相信大家对于索引也有了感官方面的认识了吧。

### 三：唯一索引

和 `sqlserver` 一样都可以建立唯一索引，重复的键值自然就不能插入，在 mongodb 中的使用方法是：

`db.person.ensureIndex({"name":1}, {"unique":true})`。

```

>
> db.person.remove({})
> db.person.ensureIndex({name:1},{unique:true})
> db.person.insert({name:"hxc","age":20})
> db.person.insert({name:"hxc","age":22})
E11000 duplicate key error index: test.person.$name_1 dup key: { : "hxc" }
>

```

#### 四：组合索引

有时候我们的查询不是单条件的，可能是多条件，比如查找出生在'1989-3-2'名字叫'jack'的同学，那么我们可以建立“姓名”和“生日”

的联合索引来加速查询。

```

>
> db.person.insert({name:"hxc","birthday":"1989-2-2"})
> db.person.insert({name:"jack","birthday":"1989-3-2"})
> db.person.insert({name:"joe","birthday":"1989-2-22"})
> db.person.insert({name:"mary","birthday":"1989-3-12"})
> db.person.insert({name:"jr","birthday":"1989-3-2"})
>
>
> db.person.ensureIndex({name:1,"birthday":1})
> db.person.ensureIndex({birthday:1,"name":1})
>
>

```

看到上图，大家或者也知道 name 跟 birthday 的不同，建立的索引也不同，升序和降序的顺序不同都会产生不同的索引，

那么我们可以用 `getindexes` 来查看下 person 集合中到底生成了那些索引。

```

> db.person.getIndexes()
[
  {
    "v" : 1,
    "key" : {
      "_id" : 1
    },
    "ns" : "test.person",
    "name" : "_id_"
  },
  {
    "v" : 1,
    "key" : {
      "name" : 1
    },
    "unique" : true,
    "ns" : "test.person",
    "name" : "name_1"
  },
  {
    "v" : 1,
    "key" : {
      "name" : 1,
      "birthday" : 1
    },
    "ns" : "test.person",
    "name" : "name_1_birthday_1"
  },
  {
    "v" : 1,
    "key" : {
      "birthday" : 1,
      "name" : 1
    },
    "ns" : "test.person",
    "name" : "birthday_1_name_1"
  }
]
>

```

此时我们肯定很好奇，到底查询优化器会使用哪个查询作为操作，呵呵，还是看看效果图：

```

> db.person.find(<<"birthday":"1989-3-2","name":"jack">>).explain()
{
  "cursor" : "BtreeCursor name_1_birthday_1",
  "nscanned" : 1,
  "nscannedObjects" : 1,
  "n" : 1,
  "millis" : 1,
  "nYields" : 0,
  "nChunkSkips" : 0,
  "isMultiKey" : false,
  "indexOnly" : false,
  "indexBounds" : {
    "name" : [
      [
        "jack",
        "jack"
      ]
    ],
    "birthday" : [
      [
        "1989-3-2",
        "1989-3-2"
      ]
    ]
  }
}
>

```

看完上图我们要相信查询优化器，它给我们做出的选择往往是最优的，因为我们做查询时，查询优化器会使用我们建立的这些索引来创建查询方案，

如果某一个先执行完则其他查询方案被 **close** 掉，这种方案会被 **mongodb** 保存起来，当然如果非要用自己指定的查询方案，这也是

可以的，在 **mongodb** 中给我们提供了 **hint** 方法让我们可以暴力执行。

```

> db.person.find(<<"birthday":"1989-3-2","name":"jack">>).hint(<<"birthday":1,"name":1>
<
  "cursor" : "BtreeCursor birthday_1_name_1",
  "nscanned" : 1,
  "nscannedObjects" : 1,
  "n" : 1,
  "millis" : 1,
  "nYields" : 0,
  "nChunkSkips" : 0,
  "isMultiKey" : false,
  "indexOnly" : false,
  "indexBounds" : {
    "birthday" : [
      [
        "1989-3-2",
        "1989-3-2"
      ]
    ],
    "name" : [
      [
        "jack",
        "jack"
      ]
    ]
  }
}

```

## 五： 删除索引

可能随着业务需求的变化，原先建立的索引可能没有存在的必要了，可能有的人想说没必要就没必要呗，但是请记住，索引会降低 CUD 这三

种操作的性能，因为这玩意需要实时维护，所以啥问题都要综合考虑一下，这里就把刚才建立的索引清空掉来演示一下:dropIndexes 的使用。

```

> db.person.dropIndexes<"name_1">
{
  "nIndexesWas" : 4,
  "msg" : "non-_id indexes dropped for collection",
  "ok" : 1
}
> db.person.dropIndexes<"name_1_birthday_1">
{
  "nIndexesWas" : 1,
  "msg" : "non-_id indexes dropped for collection",
  "ok" : 1
}
> db.person.dropIndexes<"birthday_1_name_1">
{
  "nIndexesWas" : 1,
  "msg" : "non-_id indexes dropped for collection",
  "ok" : 1
}
> db.person.getIndexes<>
[
  {
    "v" : 1,
    "key" : {
      "_id" : 1
    },
    "ns" : "test.person",
    "name" : "_id_"
  }
]
>

```

## Mongodb 主从复制

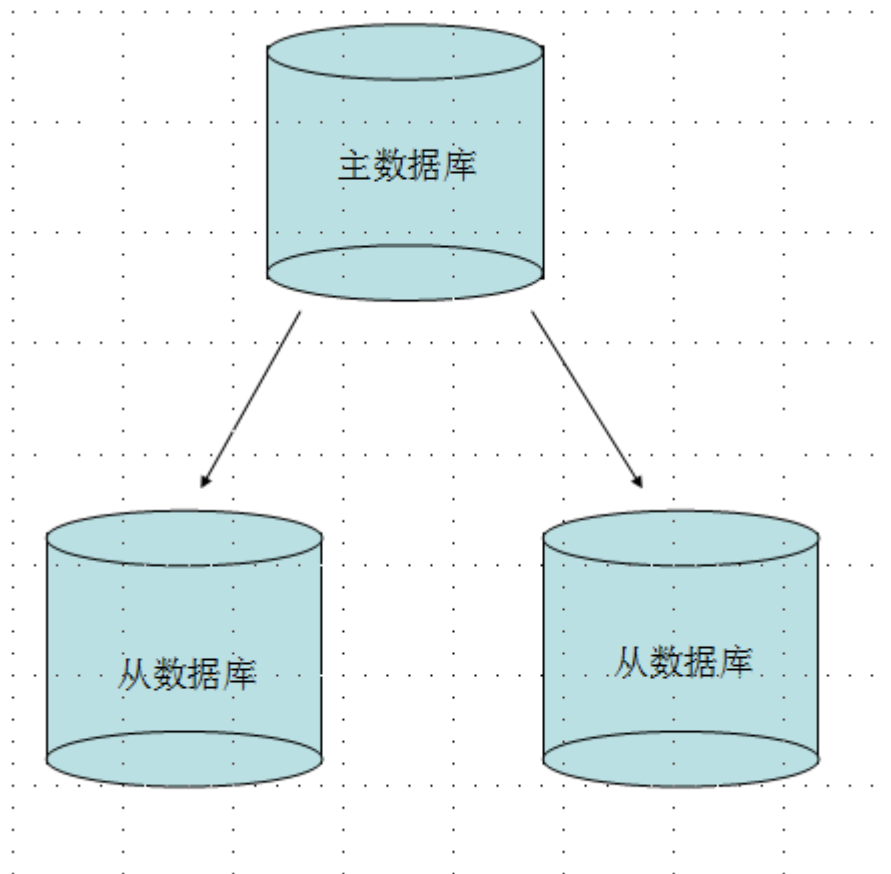
我们知道 sql server 能够做到读写分离，双机热备份和集群部署，当然 mongodb 也能做到，实际应用中我们不希望数据库采用单点部署，

如果碰到数据库宕机或者被毁灭性破坏那是多么的糟糕。

### 一：主从复制

#### 1： 首先看看模型图





2: 从上面的图形中我们可以分析出这种架构有如下的好处:

- <1> 数据备份。
- <2> 数据恢复。
- <3> 读写分离。

3: 下面我们就一一实践

实际应用中我们肯定是多服务器部署, 限于自己懒的装虚拟机, 就在一台机器上实践了。

第一步: 我们把 `mongodb` 文件夹放在 D 盘和 E 盘, 模拟放在多服务器上。

第二步: 启动 D 盘上的 `mongodb`, 把该数据库指定为主数据库, 其实命令很简单: `>mongodb --dbpath='XXX' --master,`

端口还是默认的 27017.

```
管理员: C:\Windows\system32\cmd.exe - mongod --dbpath=D:\mongodb\db --master

D:\mongodb\bin>mongod --dbpath=D:\mongodb\db --master
Sun Mar 04 19:47:38
Sun Mar 04 19:47:38 warning: 32-bit servers don't have journaling enabled by default. Please use --journal
Sun Mar 04 19:47:38 [initandlisten] MongoDB starting : pid=1440 port=27017 dbpath=D:\mongodb\db master
Sun Mar 04 19:47:38 [initandlisten]
Sun Mar 04 19:47:38 [initandlisten] ** NOTE: when using MongoDB 32 bit, you are limited to about 2 g
Sun Mar 04 19:47:38 [initandlisten] ** see http://blog.mongodb.org/post/137788967/32-bit-limit
Sun Mar 04 19:47:38 [initandlisten] ** with --journal, the limit is lower
Sun Mar 04 19:47:38 [initandlisten]
Sun Mar 04 19:47:38 [initandlisten] db version v2.0.2, pdfile version 4.5
Sun Mar 04 19:47:38 [initandlisten] git version: 514b122d308928517f5841888ceaa4246a7f18e3
Sun Mar 04 19:47:38 [initandlisten] build info: windows <5, 1, 2600, 2, 'Service Pack 3'> BOOST_LIB_
Sun Mar 04 19:47:38 [initandlisten] options: { dbpath: "D:\mongodb\db", master: true }
Sun Mar 04 19:47:38 [initandlisten] *****
Sun Mar 04 19:47:38 [initandlisten] creating replication oplog of size: 47MB...
Sun Mar 04 19:47:38 [initandlisten] *****
Sun Mar 04 19:47:38 [websvr] admin web console waiting for connections on port 28017
Sun Mar 04 19:47:38 [initandlisten] waiting for connections on port 27017
Sun Mar 04 19:48:38 [clientcursormon] mem <MB> res:37 virt:162 mapped:96
```

第三步：同样的方式启动 E 盘上的 mongodb，指定该数据库为从属数据库，命令也很简单，当然我们要换一个端口，比如：8888。

source 表示主数据库的地址。

```
>mongod --dbpath=xxxx --port=8888 --slave
--source=127.0.0.1:27017
```

```
管理员: C:\Windows\system32\cmd.exe - mongod --dbpath=E:\mongodb\db --port 8888 --slave --source=127.0.0.1:27017

E:\mongodb\bin>mongod --dbpath=E:\mongodb\db --port 8888 --slave --source=127.0.0.1:27017
Sun Mar 04 19:56:53
Sun Mar 04 19:56:53 warning: 32-bit servers don't have journaling enabled by default. Please use --journal
Sun Mar 04 19:56:53 [initandlisten] MongoDB starting : pid=3348 port=8888 dbpath=E:\mongodb\db slave=1 32-b
Sun Mar 04 19:56:53 [initandlisten]
Sun Mar 04 19:56:53 [initandlisten] ** NOTE: when using MongoDB 32 bit, you are limited to about 2 gigabyte
Sun Mar 04 19:56:53 [initandlisten] ** see http://blog.mongodb.org/post/137788967/32-bit-limitations
Sun Mar 04 19:56:53 [initandlisten] ** with --journal, the limit is lower
Sun Mar 04 19:56:53 [initandlisten]
Sun Mar 04 19:56:53 [initandlisten] db version v2.0.2, pdfile version 4.5
Sun Mar 04 19:56:53 [initandlisten] git version: 514b122d308928517f5841888ceaa4246a7f18e3
Sun Mar 04 19:56:53 [initandlisten] build info: windows <5, 1, 2600, 2, 'Service Pack 3'> BOOST_LIB_VERSION
Sun Mar 04 19:56:53 [initandlisten] options: { dbpath: "E:\mongodb\db", port: 8888, slave: true, source: "1
Sun Mar 04 19:56:53 [initandlisten] waiting for connections on port 8888
Sun Mar 04 19:56:53 [websvr] admin web console waiting for connections on port 8888
Sun Mar 04 19:56:54 [replslave] build index local.sources <_id: 1>
Sun Mar 04 19:56:54 [replslave] build index done 0 records 0.049 secs
Sun Mar 04 19:56:54 [replslave] repl: from host:127.0.0.1:27017
Sun Mar 04 19:56:54 [replslave] build index local.me <_id: 1>
Sun Mar 04 19:56:54 [replslave] build index done 0 records 0.007 secs
Sun Mar 04 19:56:58 [replslave] repl: applied 0 operations
Sun Mar 04 19:56:58 [replslave] repl: end sync_pullOpLog syncedTo: Mar 04 19:56:52 4f535884:1
Sun Mar 04 19:56:58 [replslave] repl: sleep 2 sec before next pass
Sun Mar 04 19:57:00 [replslave] repl: from host:127.0.0.1:27017
Sun Mar 04 19:57:02 [replslave] repl: from host:127.0.0.1:27017
Sun Mar 04 19:57:06 [replslave] repl: applied 1 operations
Sun Mar 04 19:57:06 [replslave] repl: end sync_pullOpLog syncedTo: Mar 04 19:57:02 4f53588e:1
Sun Mar 04 19:57:06 [replslave] repl: from host:127.0.0.1:27017
Sun Mar 04 19:57:08 [replslave] repl: from host:127.0.0.1:27017
Sun Mar 04 19:57:10 [replslave] repl: from host:127.0.0.1:27017
Sun Mar 04 19:57:16 [replslave] repl: applied 1 operations
Sun Mar 04 19:57:16 [replslave] repl: end sync_pullOpLog syncedTo: Mar 04 19:57:12 4f535898:1
Sun Mar 04 19:57:16 [replslave] repl: from host:127.0.0.1:27017
Sun Mar 04 19:57:18 [replslave] repl: from host:127.0.0.1:27017
```

第四步：从图中的红色区域我们发现了一条：“applied 1 operations”这样的语句，并且发生的时间相隔 10s，也就说明从属数据库每 10s

就向主数据库同步数据，同步依据也就是寻找主数据库的“OpLog”日志，可以在图中红色区域内发现“sync\_pullOpLog”字样。

接下来我们要做的就是测试，惊讶的发现数据已经同步更新，爽啊。

```
C:\Windows\system32\cmd.exe - mongo 127.0.0.1:27017

D:\mongodb\bin>mongo 127.0.0.1:27017
MongoDB shell version: 2.0.2
connecting to: 127.0.0.1:27017/test
> db.person.insert(<{"name":"hxc","age":23}>)
> db.person.insert(<{"name":"jack","age":25}>)
> db.person.find(<>)
< "_id" : ObjectId("4f5362f4d3bb114ea592bc7d"), "name" : "hxc", "age" : 23 >
< "_id" : ObjectId("4f5362fbd3bb114ea592bc7e"), "name" : "jack", "age" : 25 >
>

C:\Windows\system32\cmd.exe - mongo 127.0.0.1:8888

Microsoft Windows [版本 6.1.7601]
版权所有 (c) 2009 Microsoft Corporation。保留所有权利。

C:\Users\Administrator>cd E:\mongodb\bin
C:\Users\Administrator>E:

E:\mongodb\bin>mongo 127.0.0.1:8888
MongoDB shell version: 2.0.2
connecting to: 127.0.0.1:8888/test
> db.person.find(<>)
< "_id" : ObjectId("4f5362f4d3bb114ea592bc7d"), "name" : "hxc", "age" : 23 >
< "_id" : ObjectId("4f5362fbd3bb114ea592bc7e"), "name" : "jack", "age" : 25 >
>
```

4: 如果我还想增加一台从属数据库，但是我不想在启动时就指定，而是后期指定，那么mongodb可否做的到呢？答案肯定是可以的。

我们的主或者从属数据库中都有一个叫做 local 的集合，主要是用于存放内部复制信息。

好，那么我们就试一下，我在 F 盘再拷贝一份 mongodb 的运行程序，cmd 窗口好多啊，大家不要搞乱了。

```
F:\mongodb\bin>mongod --dbpath=F:\mongodb\db --port 5555 --slave
Sun Mar 04 20:59:04
Sun Mar 04 20:59:04 warning: 32-bit servers don't have journaling enabled by default. Please use --journal if you want durability.
Sun Mar 04 20:59:04
Sun Mar 04 20:59:04 [initandlisten] MongoDB starting : pid=1380 port=5555 dbpath=F:\mongodb\db slave=1 32-bit host=UONXCEVF0IT7JDJ
Sun Mar 04 20:59:04 [initandlisten]
Sun Mar 04 20:59:04 [initandlisten] ** NOTE: when using MongoDB 32 bit, you are limited to about 2 gigabytes of data
Sun Mar 04 20:59:04 [initandlisten] ** see http://blog.mongodb.org/post/137788967/32-bit-limitations
Sun Mar 04 20:59:04 [initandlisten] ** with --journal, the limit is lower
Sun Mar 04 20:59:04 [initandlisten]
Sun Mar 04 20:59:04 [initandlisten] db version v2.0.2, pdfile version 4.5
Sun Mar 04 20:59:04 [initandlisten] git version: 514b122d308928517f5841888ceaa4246a7f18e3
Sun Mar 04 20:59:04 [initandlisten] build info: windows (5, 1, 2600, 2, 'Service Pack 3') BOOST_LIB_VERSION=1_42
Sun Mar 04 20:59:04 [initandlisten] options: < dbpath: "F:\mongodb\db", port: 5555, slave: true >
Sun Mar 04 20:59:04 [websocket] admin web console waiting for connections on port 6555
Sun Mar 04 20:59:04 [initandlisten] waiting for connections on port 5555
Sun Mar 04 20:59:05 [replslave] no source given, add a master to local.sources to start replication
Sun Mar 04 20:59:05 [replslave] repl: sleep 20 sec before next pass
Sun Mar 04 20:59:25 [replslave] no source given, add a master to local.sources to start replication
Sun Mar 04 20:59:25 [replslave] repl: sleep 20 sec before next pass
Sun Mar 04 20:59:45 [replslave] no source given, add a master to local.sources to start replication
Sun Mar 04 20:59:45 [replslave] repl: sleep 20 sec before next pass
Sun Mar 04 21:00:04 [clientcursormon] mem (MB) res:21 virt:66 mapped:0
Sun Mar 04 21:00:05 [replslave] no source given, add a master to local.sources to start replication
Sun Mar 04 21:00:05 [replslave] repl: sleep 20 sec before next pass
```

看上面的 log，提示没有主数据库，没关系，某一天我们良心发现，给他后期补贴一下，哈哈，再开一个 cmd 窗口，语句也就是

在 sources 中 add 一个 host 地址，最后发现数据也同步到 127.0.0.1:5555 这台从属数据库中....

```
管理员: C:\Windows\system32\cmd.exe - mongo 127.0.0.1:5555
Microsoft Windows [版本 6.1.7601]
版权所有 (c) 2009 Microsoft Corporation。保留所有权利。

C:\Users\Administrator>cd F:\mongodb\bin

C:\Users\Administrator>F:

F:\mongodb\bin>mongo 127.0.0.1:5555
MongoDB shell version: 2.0.2
connecting to: 127.0.0.1:5555/test

> use local
switched to db local
> db.sources.insert(<{"host":"127.0.0.1:27017"}>)
> db.sources.find()
{ "_id" : ObjectId<"4f5368868f59fa666264355">, "host" : "127.0.0.1:27017" } 同步前
> db.sources.find()
{ "_id" : ObjectId<"4f5368868f59fa666264355">, "host" : "127.0.0.1:27017", "source" : "main", "syncedTo" : { "t" : 1330866343000, "i" : 1 } } 同步后
>
> use test
switched to db test
> db.person.find()
{ "_id" : ObjectId<"4f5362f4d3bb114ea592bc7d">, "name" : "hxc", "age" : 23 }
{ "_id" : ObjectId<"4f5362f4d3bb114ea592bc7e">, "name" : "jack", "age" : 25 }
>
```

## 5: 读写分离

这种手段在大一点的架构中都有实现，在 mongodb 中其实很简单，在默认的情况下，从属数据库不支持数据的读取，但是没关系，

在驱动中给我们提供了一个叫做"slaveOkay"来让我们可以显示的读取从属数据库来减轻主数据库的性能压力，这里就不演示了。

## 二：副本集

这个也是很牛 X 的主从集群，不过跟上面的集群还是有两点区别的。

<1>： 该集群没有特定的主数据库。

<2>： 如果哪个主数据库宕机了，集群中就会推选出一个从属数据库作为主数据库顶上，这就具备了自动故障恢复功能，很牛 X 的啊。

好，我们现在就来试一下，首先把所有的 cmd 窗口关掉重新来，清掉 db 下的所有文件。

第一步： 既然我们要建立集群，就得取个集群名字，这里就取我们的公司名 shopex，--replSet 表示让服务器知道 shopex 下还有其他数据库，

这里就把 D 盘里面的 mongodb 程序打开，端口为 2222。指定端口为 3333 是 shopex 集群下的另一个数据库服务器。

```
管理员: C:\Windows\system32\cmd.exe - mongod --dbpath=D:\mongodb\db --port 2222 --replSet shopex/127.0.0.1:3333
D:\mongodb\bin>mongod --dbpath=D:\mongodb\db --port 2222 --replSet shopex/127.0.0.1:3333
Mon Mar 05 21:08:43
Mon Mar 05 21:08:43 warning: 32-bit servers don't have journaling enabled by default. Please use --journal
Mon Mar 05 21:08:43
Mon Mar 05 21:08:43 [initandlisten] MongoDB starting : pid=5144 port=2222 dbpath=D:\mongodb\db 32-bit hos
Mon Mar 05 21:08:43 [initandlisten]
Mon Mar 05 21:08:43 [initandlisten] ** NOTE: when using MongoDB 32 bit, you are limited to about 2 gigaby
Mon Mar 05 21:08:43 [initandlisten] ** see http://blog.mongodb.org/post/137788967/32-bit-limitation
Mon Mar 05 21:08:43 [initandlisten] ** with --journal, the limit is lower
Mon Mar 05 21:08:43 [initandlisten]
Mon Mar 05 21:08:43 [initandlisten] db version v2.0.2, pdfile version 4.5
Mon Mar 05 21:08:43 [initandlisten] git version: 514b122d308928517f5841888ceaa4246a7f18e3
Mon Mar 05 21:08:43 [initandlisten] build info: windows (5, 1, 2600, 2, 'Service Pack 3') BOOST_LIB_VERSI
Mon Mar 05 21:08:43 [initandlisten] options: { dbpath: "D:\mongodb\db", port: 2222, replSet: "shopex/127.0.0.1:3333"
```

第二步：既然上面说 3333 是另一个数据库服务器，不要急，现在就来开，这里把 E 盘的 mongod 程序打开。

```
管理员: C:\Windows\system32\cmd.exe - mongod --dbpath=E:\mongodb\db --port 3333 --replSet shopex/127.0.0.1:2222
E:\mongodb\bin>mongod --dbpath=E:\mongodb\db --port 3333 --replSet shopex/127.0.0.1:2222
Mon Mar 05 21:13:16
Mon Mar 05 21:13:16 warning: 32-bit servers don't have journaling enabled by default. Please use --journal
Mon Mar 05 21:13:16
Mon Mar 05 21:13:16 [initandlisten] MongoDB starting : pid=4644 port=3333 dbpath=E:\mongodb\db 32-bit
Mon Mar 05 21:13:16 [initandlisten]
Mon Mar 05 21:13:16 [initandlisten] ** NOTE: when using MongoDB 32 bit, you are limited to about 2 g
Mon Mar 05 21:13:16 [initandlisten] ** see http://blog.mongodb.org/post/137788967/32-bit-limitation
Mon Mar 05 21:13:16 [initandlisten] ** with --journal, the limit is lower
Mon Mar 05 21:13:16 [initandlisten]
Mon Mar 05 21:13:16 [initandlisten] db version v2.0.2, pdfile version 4.5
Mon Mar 05 21:13:16 [initandlisten] git version: 514b122d308928517f5841888ceaa4246a7f18e3
Mon Mar 05 21:13:16 [initandlisten] build info: windows (5, 1, 2600, 2, 'Service Pack 3') BOOST_LIB_VERSI
Mon Mar 05 21:13:16 [initandlisten] options: { dbpath: "E:\mongodb\db", port: 3333, replSet: "shopex/127.0.0.1:2222"
Mon Mar 05 21:13:17 [initandlisten] waiting for connections on port 3333
Mon Mar 05 21:13:17 [initandlisten] connection accepted from 127.0.0.1:50941 #1
Mon Mar 05 21:13:17 [conn1] end connection 127.0.0.1:50941
Mon Mar 05 21:13:17 [initandlisten] connection accepted from 127.0.0.1:50942 #2
Mon Mar 05 21:13:17 [rsStart] trying to contact 127.0.0.1:2222
Mon Mar 05 21:13:17 [rsStart] replSet can't get local.system.replset config from self or any seed (
Mon Mar 05 21:13:17 [rsStart] replSet info you may need to run replSetInitiate -- rs.initiate() in
Mon Mar 05 21:13:17 [websvr] admin web console waiting for connections on port 4333
Mon Mar 05 21:13:22 [initandlisten] connection accepted from 127.0.0.1:50944 #3
Mon Mar 05 21:13:27 [rsStart] trying to contact 127.0.0.1:2222
```

第三步：ok，看看上面的日志红色区域，似乎我们还没有做完，是的，log 信息告诉我们要初始化一下“副本集”，既然日志这么说，那我也就

这么做，随便连接一下哪个服务器都行，不过一定要进入 admin 集合。

```

C:\Windows\system32\cmd.exe - mongo 127.0.0.1:2222/admin
>
D:\mongodb\bin>mongo 127.0.0.1:2222/admin
MongoDB shell version: 2.0.2
connecting to: 127.0.0.1:2222/admin
> db.runCommand(<<"replSetInitiate":{
...   "_id":"shopex",
...   "members":[
...     <
...     "_id":1,
...     "host":"127.0.0.1:2222"
...   ],
...   <
...   "_id":2,
...   "host":"127.0.0.1:3333"
... ]>>>)
{
  "info" : "Config now saved locally.  Should come online in about a minute.",
  "ok" : 1
}

```

第四步：开启成功后，我们要看看谁才能成为主数据库服务器，可以看到端口为 2222 的已经成为主数据库服务器。

```

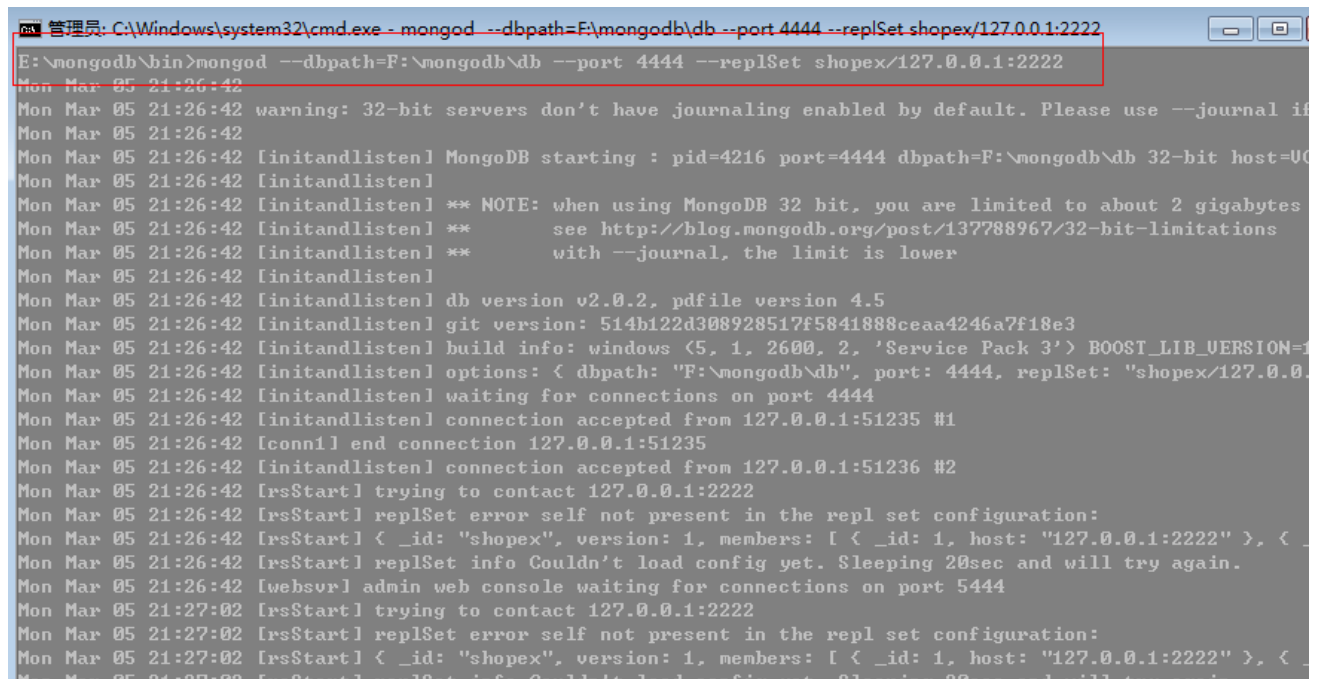
C:\Windows\system32\cmd.exe - mongod --dbpath=D:\mongodb\db --port 2222 --replSet shopex/127.0.0.1:3333
Mon Mar 05 21:17:43 [rsStart] trying to contact 127.0.0.1:3333
Mon Mar 05 21:17:43 [rsStart] replSet STARTUP2
Mon Mar 05 21:17:43 [rsMgr] replSet total number of votes is even - add arbiter or give one mem
Mon Mar 05 21:17:43 [rsHealthPoll] replSet member 127.0.0.1:3333 is up
Mon Mar 05 21:17:43 [rsSync] replSet SECONDARY
Mon Mar 05 21:17:43 [rsMgr] replSet info electSelf 1
Mon Mar 05 21:17:43 [rsMgr] replSet couldn't elect self, only received 1 votes
Mon Mar 05 21:17:43 [clientcursormon] mem <MB> res:37 virt:165 mapped:96
Mon Mar 05 21:17:49 [rsHealthPoll] replSet member 127.0.0.1:3333 is now in state STARTUP2
Mon Mar 05 21:17:49 [rsMgr] not electing self, 127.0.0.1:3333 would veto
Mon Mar 05 21:17:55 [rsMgr] replSet info electSelf 1
Mon Mar 05 21:17:55 [rsMgr] replSet PRIMARY
Mon Mar 05 21:17:57 [rsHealthPoll] replSet member 127.0.0.1:3333 is now in state RECOVERING
Mon Mar 05 21:18:03 [initandlisten] connection accepted from 127.0.0.1:51034 #7
Mon Mar 05 21:18:03 [initandlisten] connection accepted from 127.0.0.1:51035 #8

C:\Windows\system32\cmd.exe - mongod --dbpath=E:\mongodb\db --port 3333 --replSet shopex/127.0.0.1:2222
Mon Mar 05 21:18:03 [rsSync] build index local.me < _id: 1 >
Mon Mar 05 21:18:03 [rsSync] build index done 0 records 0.046 secs
Mon Mar 05 21:18:03 [rsSync] replSet initial sync drop all databases
Mon Mar 05 21:18:03 [rsSync] dropAllDatabasesExceptLocal 1
Mon Mar 05 21:18:03 [rsSync] replSet initial sync clone all databases
Mon Mar 05 21:18:03 [rsSync] replSet initial sync query minValid
Mon Mar 05 21:18:03 [rsSync] replSet initial oplog application from 127.0.0.1:2222 starting at
Mon Mar 05 21:18:05 [rsSync] replSet initial sync finishing up
Mon Mar 05 21:18:06 [rsSync] replSet set minValid=4f54bcf5:1
Mon Mar 05 21:18:06 [rsSync] build index local.replset.minvalid < _id: 1 >
Mon Mar 05 21:18:06 [rsSync] build index done 0 records 0.007 secs
Mon Mar 05 21:18:06 [rsSync] replSet initial sync done
Mon Mar 05 21:18:07 [rsSync] replSet syncing to: 127.0.0.1:2222
Mon Mar 05 21:18:07 [rsSync] replSet SECONDARY
Mon Mar 05 21:18:11 [conn3] end connection 127.0.0.1:50944
Mon Mar 05 21:18:11 [initandlisten] connection accepted from 127.0.0.1:51037 #5
Mon Mar 05 21:18:17 [clientcursormon] mem <MB> res:37 virt:164 mapped:96
Mon Mar 05 21:18:41 [conn5] end connection 127.0.0.1:51037

```

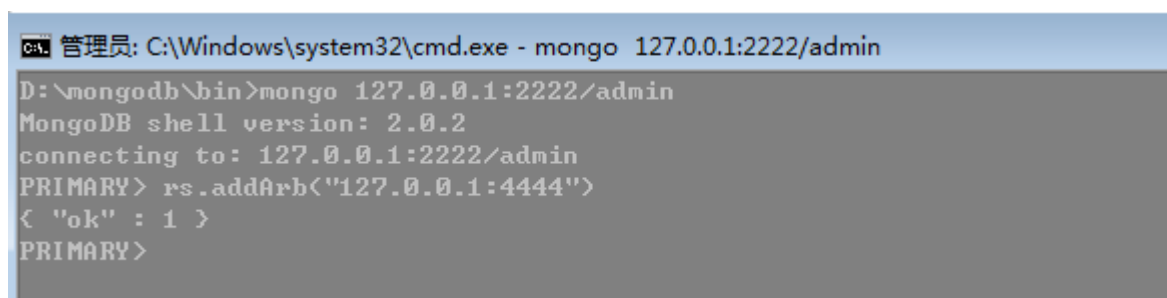
第五步：我们知道 **sql server** 里面有一个叫做仲裁服务器，那么 **mongodb** 中也是有的，跟 **sql server** 一样，仲裁只参与投票选举，这里我们

把 **F** 盘的 **mongodb** 作为仲裁服务器，然后指定 **shopex** 集群中的任一个服务器端口，这里就指定 **2222**。



```
管理员: C:\Windows\system32\cmd.exe - mongod --dbpath=F:\mongodb\db --port 4444 --replSet shopex/127.0.0.1:2222
E:\mongodb\bin>mongod --dbpath=F:\mongodb\db --port 4444 --replSet shopex/127.0.0.1:2222
Mon Mar 05 21:26:42 warning: 32-bit servers don't have journaling enabled by default. Please use --journal if
Mon Mar 05 21:26:42 [initandlisten] MongoDB starting : pid=4216 port=4444 dbpath=F:\mongodb\db 32-bit host=UO
Mon Mar 05 21:26:42 [initandlisten]
Mon Mar 05 21:26:42 [initandlisten] ** NOTE: when using MongoDB 32 bit, you are limited to about 2 gigabytes
Mon Mar 05 21:26:42 [initandlisten] ** see http://blog.mongodb.org/post/137788967/32-bit-limitations
Mon Mar 05 21:26:42 [initandlisten] ** with --journal, the limit is lower
Mon Mar 05 21:26:42 [initandlisten]
Mon Mar 05 21:26:42 [initandlisten] db version v2.0.2, pdfile version 4.5
Mon Mar 05 21:26:42 [initandlisten] git version: 514b122d308928517f5841888ceaa4246a7f18e3
Mon Mar 05 21:26:42 [initandlisten] build info: windows (5, 1, 2600, 2, 'Service Pack 3') BOOST_LIB_VERSION=1
Mon Mar 05 21:26:42 [initandlisten] options: { dbpath: "F:\mongodb\db", port: 4444, replSet: "shopex/127.0.0.
Mon Mar 05 21:26:42 [initandlisten] waiting for connections on port 4444
Mon Mar 05 21:26:42 [initandlisten] connection accepted from 127.0.0.1:51235 #1
Mon Mar 05 21:26:42 [conn1] end connection 127.0.0.1:51235
Mon Mar 05 21:26:42 [initandlisten] connection accepted from 127.0.0.1:51236 #2
Mon Mar 05 21:26:42 [rsStart] trying to contact 127.0.0.1:2222
Mon Mar 05 21:26:42 [rsStart] replSet error self not present in the repl set configuration:
Mon Mar 05 21:26:42 [rsStart] { _id: "shopex", version: 1, members: [ { _id: 1, host: "127.0.0.1:2222" }, {
Mon Mar 05 21:26:42 [rsStart] replSet info Couldn't load config yet. Sleeping 20sec and will try again.
Mon Mar 05 21:26:42 [websvr] admin web console waiting for connections on port 5444
Mon Mar 05 21:27:02 [rsStart] trying to contact 127.0.0.1:2222
Mon Mar 05 21:27:02 [rsStart] replSet error self not present in the repl set configuration:
Mon Mar 05 21:27:02 [rsStart] { _id: "shopex", version: 1, members: [ { _id: 1, host: "127.0.0.1:2222" }, {
Mon Mar 05 21:27:02 [rsStart] replSet info Couldn't load config yet. Sleeping 20sec and will try again.
```

然后我们在 **admin** 集合中使用 **rs.addArb()**追加即可。



```
管理员: C:\Windows\system32\cmd.exe - mongo 127.0.0.1:2222/admin
D:\mongodb\bin>mongo 127.0.0.1:2222/admin
MongoDB shell version: 2.0.2
connecting to: 127.0.0.1:2222/admin
PRIMARY> rs.addArb("127.0.0.1:4444")
< "ok" : 1 >
PRIMARY>
```

追加好了之后，我们使用 **rs.status()**来查看下集群中的服务器状态，图中我们可以清楚的看到谁是主，还是从，还是仲裁。

```
管理员: C:\Windows\system32\cmd.exe - mongo 127.0.0.1:2222/admin
PRIMARY> rs.status()
{
  "set" : "shopex",
  "date" : ISODate("2012-03-05T13:30:40Z"),
  "myState" : 1,
  "members" : [
    {
      "_id" : 1,
      "name" : "127.0.0.1:2222",
      "health" : 1,
      "state" : 1,
      "stateStr" : "PRIMARY",
      "optime" : {
        "t" : 1330954176000,
        "i" : 1
      },
      "optimeDate" : ISODate("2012-03-05T13:29:36Z"),
      "self" : true
    },
    {
      "_id" : 2,
      "name" : "127.0.0.1:3333",
      "health" : 1,
      "state" : 2,
      "stateStr" : "SECONDARY",
      "uptime" : 777,
      "optime" : {
        "t" : 1330954176000,
        "i" : 1
      },
      "optimeDate" : ISODate("2012-03-05T13:29:36Z"),
      "lastHeartbeat" : ISODate("2012-03-05T13:30:39Z"),
      "pingMs" : 0
    },
    {
      "_id" : 3,
      "name" : "127.0.0.1:4444",
      "health" : 1,
      "state" : 7,
      "stateStr" : "ARBITER",
      "uptime" : 64,
      "optime" : {
        "t" : 0,

```

不是说该集群有自动故障恢复吗？那么我们就可以来试一下，在 2222 端口的 cmd 服务器按 Ctrl+C 来 KO 掉该服务器，立马我们发现

在 3333 端口的从属服务器即可顶上，最后大家也可以再次使用 rs.status()来看下集群中服务器的状态。



```
管理员: C:\Windows\system32\cmd.exe
Mon Mar 05 21:32:13 [conn42] error
Mon Mar 05 21:32:13 [initandlisten]
Mon Mar 05 21:32:17 [conn43] error
Mon Mar 05 21:32:17 [initandlisten]
Mon Mar 05 21:32:43 [conn44] error
Mon Mar 05 21:32:43 [initandlisten]
Mon Mar 05 21:32:43 [clientcursor]
Mon Mar 05 21:32:47 [conn45] error
Mon Mar 05 21:32:47 [initandlisten]
Mon Mar 05 21:32:49 Ctrl-C signal
Mon Mar 05 21:32:49 got kill on
Mon Mar 05 21:32:49 [ctrlCTerm]
Mon Mar 05 21:32:49 dbexit:
Mon Mar 05 21:32:49 [ctrlCTerm]
Mon Mar 05 21:32:49 [ctrlCTerm]
Mon Mar 05 21:32:49 [ctrlCTerm]
Mon Mar 05 21:32:49 [ctrlCTerm]
Mon Mar 05 21:32:49 [ctrlCTerm]
Mon Mar 05 21:32:49 [conn2] end
Mon Mar 05 21:32:49 [conn46] error
Mon Mar 05 21:32:49 [conn47] error
Mon Mar 05 21:32:49 [conn33] error
Mon Mar 05 21:32:49 [slaveTrac]
Mon Mar 05 21:32:50 [ctrlCTerm]
Mon Mar 05 21:32:50 [ctrlCTerm]
Mon Mar 05 21:32:50 dbexit: rea
D:\mongodb\bin>

管理员: C:\Windows\system32\cmd.exe - mongod --dbpath=E:\mongodb\db --port 3333 --replSet shop
Mon Mar 05 21:30:41 [initandlisten] connection accepted from 127.0.0.1:51299 #3
Mon Mar 05 21:30:43 [conn31] end connection 127.0.0.1:51292
Mon Mar 05 21:30:43 [initandlisten] connection accepted from 127.0.0.1:51302 #3
Mon Mar 05 21:31:11 [conn32] end connection 127.0.0.1:51299
Mon Mar 05 21:31:11 [initandlisten] connection accepted from 127.0.0.1:51307 #3
Mon Mar 05 21:31:13 [conn33] end connection 127.0.0.1:51302
Mon Mar 05 21:31:13 [initandlisten] connection accepted from 127.0.0.1:51308 #3
Mon Mar 05 21:31:41 [conn34] end connection 127.0.0.1:51307
Mon Mar 05 21:31:41 [initandlisten] connection accepted from 127.0.0.1:51317 #3
Mon Mar 05 21:31:43 [conn35] end connection 127.0.0.1:51308
Mon Mar 05 21:31:43 [initandlisten] connection accepted from 127.0.0.1:51318 #3
Mon Mar 05 21:32:11 [conn36] end connection 127.0.0.1:51317
Mon Mar 05 21:32:11 [initandlisten] connection accepted from 127.0.0.1:51332 #3
Mon Mar 05 21:32:13 [conn37] end connection 127.0.0.1:51318
Mon Mar 05 21:32:13 [initandlisten] connection accepted from 127.0.0.1:51333 #3
Mon Mar 05 21:32:41 [conn38] end connection 127.0.0.1:51332
Mon Mar 05 21:32:41 [initandlisten] connection accepted from 127.0.0.1:51340 #4
Mon Mar 05 21:32:43 [conn39] end connection 127.0.0.1:51333
Mon Mar 05 21:32:43 [initandlisten] connection accepted from 127.0.0.1:51341 #4
Mon Mar 05 21:32:49 [rsSync] replSet syncThread: 10278 dbclient error communic
Mon Mar 05 21:32:49 [conn40] end connection 127.0.0.1:51340
Mon Mar 05 21:32:49 [rsHealthPoll] replSet info 127.0.0.1:2222 is down (or slow
Mon Mar 05 21:32:49 [rsHealthPoll] replSet member 127.0.0.1:2222 is now in stat
Mon Mar 05 21:32:49 [rsMgr] not electing self, 127.0.0.1:4444 would veto
Mon Mar 05 21:32:55 [rsMgr] replSet info electSelf 2
Mon Mar 05 21:32:55 [rsMgr] replSet PRIMARY
Mon Mar 05 21:33:13 [conn41] end connection 127.0.0.1:51341
Mon Mar 05 21:33:13 [initandlisten] connection accepted from 127.0.0.1:51364 #4
Mon Mar 05 21:33:17 [clientcursormon] mem <MB> res:37 virt:165 mapped:96
```

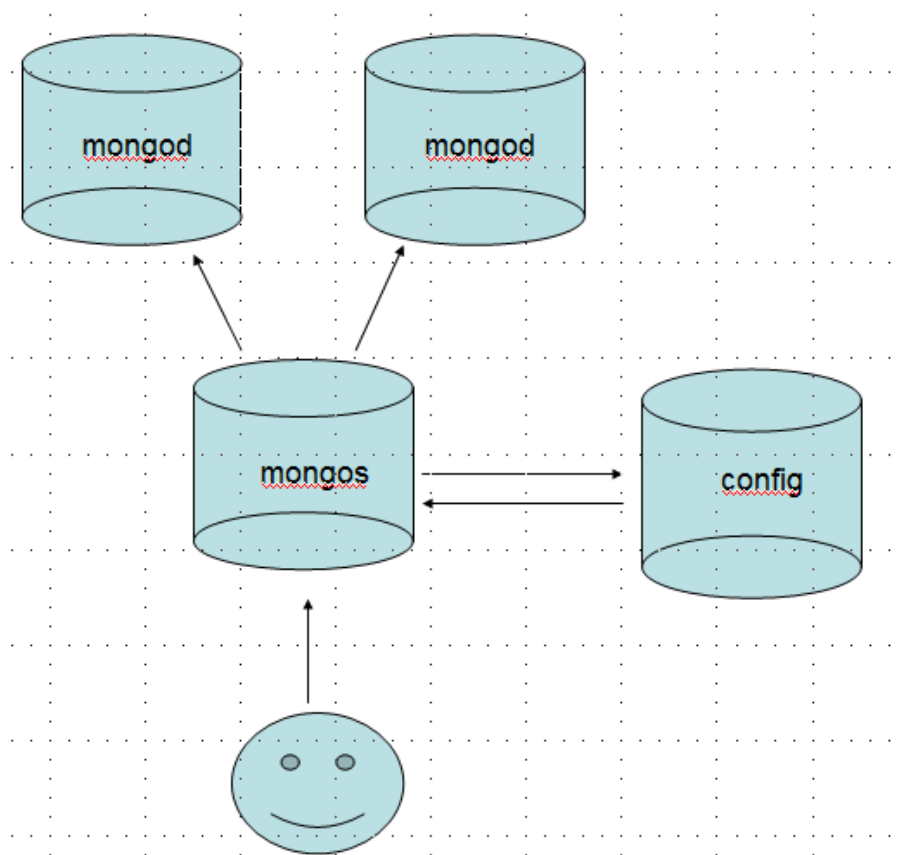
## Mongodb 分片技术

在 mongodb 里面存在另一种集群，就是分片技术，跟 sql server 的表分区类似，我们知道当数据量达到 T 级别的时候，我们的磁盘，内存

就吃不消了，针对这样的场景我们该如何应对。

### 一：分片

mongodb 采用将集合进行拆分，然后将拆分的数据均摊到几个片上的一种解决方案。



下面我对这张图解释一下：

人脸： 代表客户端，客户端肯定说，你数据库分片不分片跟我没关系，我叫你干啥就干啥，没什么好商量的。

mongos： 首先我们要了解“片键”的概念，也就是说拆分集合的依据是什么？按照什么键值进行拆分集合....

好了，mongos 就是一个路由服务器，它会根据管理员设置的“片键”将数据分摊到自己管理的 mongod 集群，数据

和片的对应关系以及相应的配置信息保存在"config 服务器"上。

mongod： 一个普通的数据库实例，如果不分片的话，我们会直接连上 mongod。

## 二： 实战

首先我们准备 4 个 mongodb 程序，我这里是均摊在 C，D，E，F 盘上，当然你也可以做多个文件夹的形式。

### 1： 开启 config 服务器

先前也说了，mongos 要把 mongod 之间的配置放到 config 服务器里面，理所当然首先开启它，我这里就建立 2222 端口。

```
管理员: C:\Windows\system32\cmd.exe - mongod --dbpath=C:\mongodb\bin --port 2222

C:\mongodb\bin>mongod --dbpath=C:\mongodb\bin --port 2222
Wed Mar 07 10:34:50 warning: 32-bit servers don't have journaling enabled by default. Please use
Wed Mar 07 10:34:50 [initandlisten] MongoDB starting : pid=5576 port=2222 dbpath=C:\mongodb\bin
Wed Mar 07 10:34:50 [initandlisten]
Wed Mar 07 10:34:50 [initandlisten] ** NOTE: when using MongoDB 32 bit, you are limited to about
Wed Mar 07 10:34:50 [initandlisten] ** see http://blog.mongodb.org/post/137788967/32-bit-1
Wed Mar 07 10:34:50 [initandlisten] ** with --journal, the limit is lower
Wed Mar 07 10:34:50 [initandlisten]
Wed Mar 07 10:34:50 [initandlisten] db version v2.0.2, pdfile version 4.5
Wed Mar 07 10:34:50 [initandlisten] git version: 514b122d308928517f5841888ceaa4246a7f18e3
Wed Mar 07 10:34:50 [initandlisten] build info: windows (5, 1, 2600, 2, 'Service Pack 3') BOOST_
Wed Mar 07 10:34:50 [initandlisten] options: { dbpath: "C:\mongodb\bin", port: 2222 }
Wed Mar 07 10:34:50 [websvr] admin web console waiting for connections on port 3222
Wed Mar 07 10:34:50 [initandlisten] waiting for connections on port 2222
```

## 2: 开启 mongos 服务器

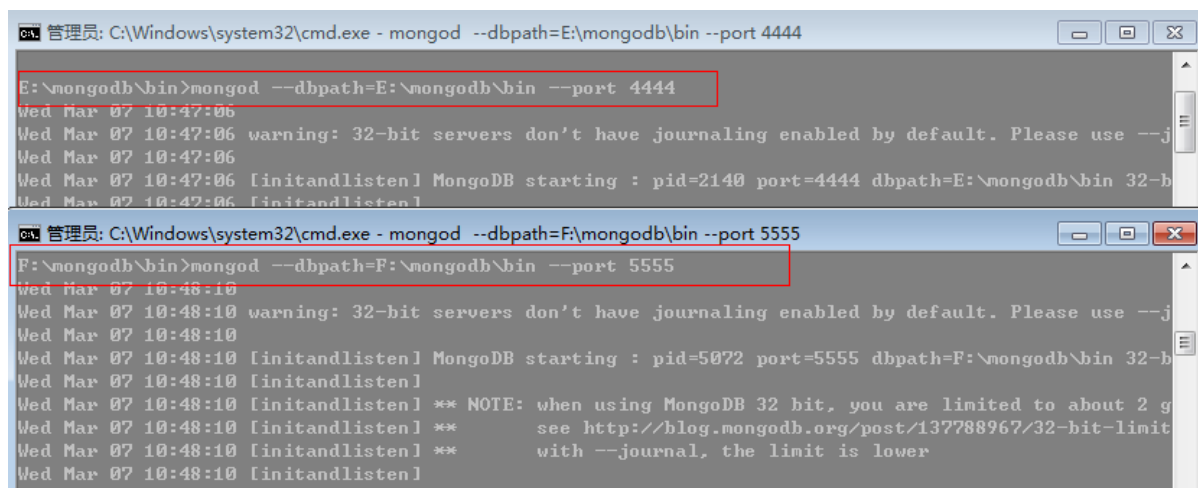
这里要注意的是我们开启的是 mongos，不是 mongod，同时指定下 config 服务器，这里我就开启 D 盘上的 mongod，端口 3333。

```
管理员: C:\Windows\system32\cmd.exe - mongos --port 3333 --configdb=127.0.0.1:2222

D:\mongodb\bin>mongos --port 3333 --configdb=127.0.0.1:2222
Wed Mar 07 10:40:54 mongos db version v2.0.2, pdfile version 4.5 starting (--help for usage)
Wed Mar 07 10:40:54 git version: 514b122d308928517f5841888ceaa4246a7f18e3
Wed Mar 07 10:40:54 build info: windows (5, 1, 2600, 2, 'Service Pack 3') BOOST_LIB_VERSION=1_42
Wed Mar 07 10:40:54 [mongosMain] waiting for connections on port 3333
Wed Mar 07 10:40:54 [websvr] admin web console waiting for connections on port 4333
Wed Mar 07 10:40:54 [Balancer] about to contact config servers and shards
Wed Mar 07 10:40:54 [Balancer] config servers and shards contacted successfully
Wed Mar 07 10:40:54 [Balancer] balancer id: VONXCEVF0IT7JDDJ:3333 started at Mar 07 10:40:54
Wed Mar 07 10:40:54 [Balancer] created new distributed lock for balancer on 127.0.0.1:2222 < lock timeout :
Wed Mar 07 10:40:54 [Balancer] creating WriteBackListener for: 127.0.0.1:2222 serverID: 4f56cab69325c125a66
```

## 3: 启动 mongod 服务器

对分片来说，也就是要添加片了，这里开启 E，F 盘的 mongod，端口为：4444，5555。



```
管理员: C:\Windows\system32\cmd.exe - mongod --dbpath=E:\mongodb\bin --port 4444
E:\mongodb\bin>mongod --dbpath=E:\mongodb\bin --port 4444
Wed Mar 07 10:47:06
Wed Mar 07 10:47:06 warning: 32-bit servers don't have journaling enabled by default. Please use --j
Wed Mar 07 10:47:06
Wed Mar 07 10:47:06 [initandlisten] MongoDB starting : pid=2140 port=4444 dbpath=E:\mongodb\bin 32-b
Wed Mar 07 10:47:06 [initandlisten]

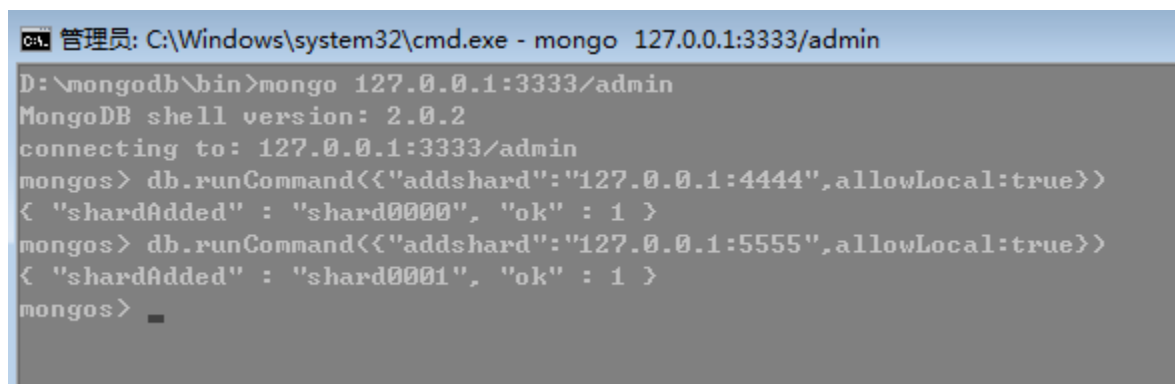
管理员: C:\Windows\system32\cmd.exe - mongod --dbpath=F:\mongodb\bin --port 5555
F:\mongodb\bin>mongod --dbpath=F:\mongodb\bin --port 5555
Wed Mar 07 10:48:10
Wed Mar 07 10:48:10 warning: 32-bit servers don't have journaling enabled by default. Please use --j
Wed Mar 07 10:48:10
Wed Mar 07 10:48:10 [initandlisten] MongoDB starting : pid=5072 port=5555 dbpath=F:\mongodb\bin 32-b
Wed Mar 07 10:48:10 [initandlisten]
Wed Mar 07 10:48:10 [initandlisten] ** NOTE: when using MongoDB 32 bit, you are limited to about 2 g
Wed Mar 07 10:48:10 [initandlisten] ** see http://blog.mongodb.org/post/137788967/32-bit-limit
Wed Mar 07 10:48:10 [initandlisten] ** with --journal, the limit is lower
Wed Mar 07 10:48:10 [initandlisten]
```

#### 4: 服务配置

哈哈，是不是很高兴，还差最后一点配置我们就可以大功告成。

<1> 先前图中也可以看到，我们 client 直接跟 mongos 打交道，也就说明我们要连接 mongos 服务器，然后将 4444, 5555 的 mongod

交给 mongos,添加分片也就是 addshard()。



```
管理员: C:\Windows\system32\cmd.exe - mongo 127.0.0.1:3333/admin
D:\mongodb\bin>mongo 127.0.0.1:3333/admin
MongoDB shell version: 2.0.2
connecting to: 127.0.0.1:3333/admin
mongos> db.runCommand(<{"addshard":"127.0.0.1:4444",allowLocal:true}>)
< "shardAdded" : "shard0000", "ok" : 1 >
mongos> db.runCommand(<{"addshard":"127.0.0.1:5555",allowLocal:true}>)
< "shardAdded" : "shard0001", "ok" : 1 >
mongos> _
```

这里要注意的是，在 addshard 中，我们也可以添加副本集，这样能达到更高的稳定性。

<2>片已经集群了，但是 mongos 不知道该如何切分数据，也就是我们先前所说的片键，在 mongodb 中设置片键要做两步

①: 开启数据库分片功能，命令很简单 enablesharding(),这里我就开启 test 数据库。

②: 指定集合中分片的片键，这里我就指定为 person.name 字段。

```
C:\Windows\system32\cmd.exe - mongo 127.0.0.1:3333/admin
D:\mongodb\bin>mongo 127.0.0.1:3333/admin
MongoDB shell version: 2.0.2
connecting to: 127.0.0.1:3333/admin
mongos> db.runCommand(<<"addshard":"127.0.0.1:4444",allowLocal:true>>)
{ "shardAdded" : "shard0000", "ok" : 1 }
mongos> db.runCommand(<<"addshard":"127.0.0.1:5555",allowLocal:true>>)
{ "shardAdded" : "shard0001", "ok" : 1 }
mongos> db.runCommand(<<"enablesharding":"test">>)
{ "ok" : 1 }
mongos> db.runCommand(<<"shardcollection":"test.person","key":{ "name":1 }>>)
{ "collectionsharded" : "test.person", "ok" : 1 }
mongos>
```

## 5: 查看效果

好了，至此我们的分片操作全部结束，接下来我们通过 `mongos` 向 `mongodb` 插入 10w 记录，然后通过 `printShardingStatus` 命令

查看 `mongodb` 的数据分片情况。

```
C:\Windows\system32\cmd.exe - mongo 127.0.0.1:3333/admin
mongos> use test
switched to db test
mongos> for(var i=0;i<100000;i++){
... db.person.insert(<<"name":"jack"+i,"age":i>>)
... }
mongos> db.printShardingStatus()
--- Sharding Status ---
  sharding version: { "_id" : 1, "version" : 3 }
  shards:
    { "_id" : "shard0000", "host" : "127.0.0.1:4444" }
    { "_id" : "shard0001", "host" : "127.0.0.1:5555" }
  databases:
    { "_id" : "admin", "partitioned" : false, "primary" : "config" }
    { "_id" : "test", "partitioned" : true, "primary" : "shard0000" }
    test.person chunks:
      shard0000          3
      shard0001          1
    { "name" : { $minKey : 1 } } --> { "name" : "jack0" } on : shard0000 { "t
    { "name" : "jack0" } --> { "name" : "jack234813" } on : shard0000 { "t
    { "name" : "jack234813" } --> { "name" : "jack9999" } on : shard0000 { "t
    { "name" : "jack9999" } --> { "name" : { $maxKey : 1 } } on : shard0001 {
mongos>
```

这里主要看三点信息：

- ① **shards:** 我们清楚的看到已经别分为两个片了，`shard0000` 和 `shard0001`。
- ② **databases:** 这里有个 `partitioned` 字段表示是否分区，这里清楚的看到 `test` 已经分区。
- ③ **chunks:** 这个很有意思，我们发现集合被砍成四段：

无穷小 —— jack0, jack0 ——jack234813,  
jack234813——jack9999, jack9999——无穷大。

分区情况为：3: 1，从后面的 on shardXXXX 也能看得出。

## Mongodb 运维管理

这一篇我们以管理员的视角来看 mongodb，作为一名管理员，我们经常接触到的主要有 4 个方面：

1. 安装部署
2. 状态监控
3. 安全认证
4. 备份和恢复，

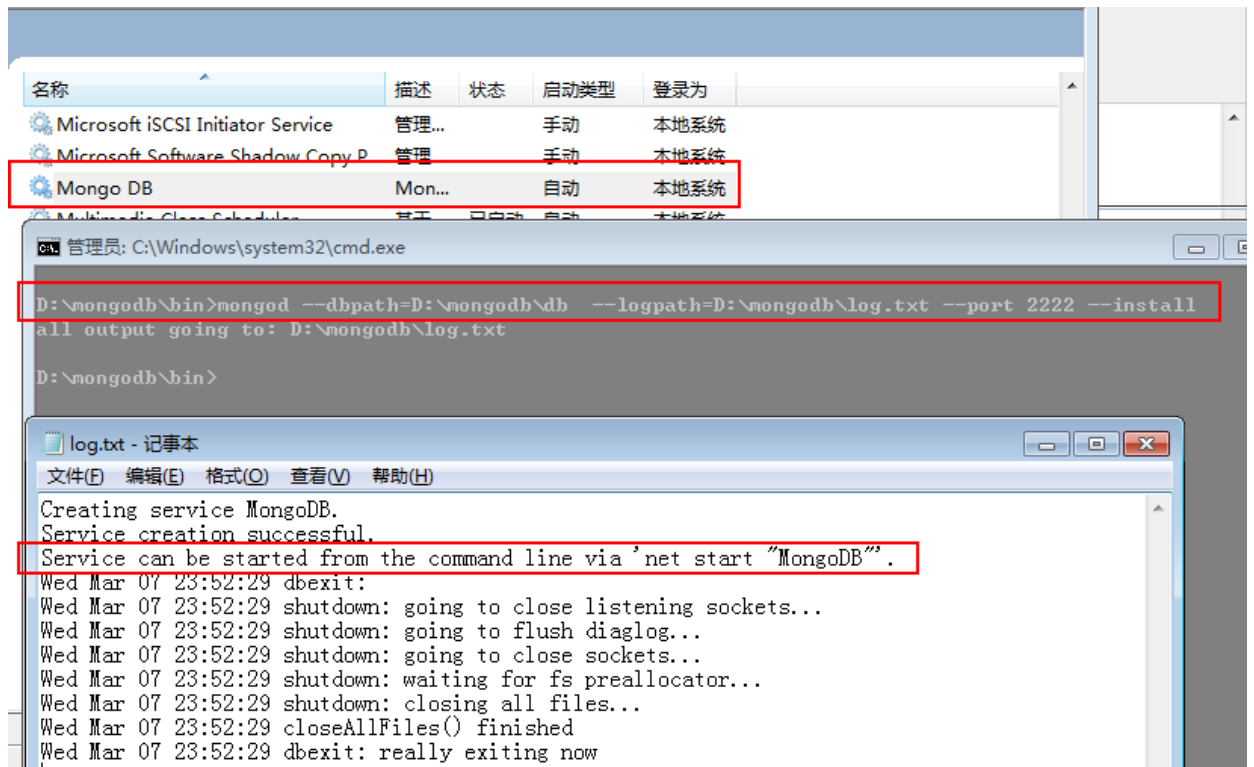
下面我们就一点一点的讲解。

### 一：安装部署

我之前的文章都是采用 console 程序来承载，不过在生产环境中这并不是最佳实践，谁也不愿意在机器重启后满地找牙似找 mongodb，

在 mongodb 里面提供了一个叫做“服务寄宿”的模式，我想如果大家对 wcf 比较熟悉的话很容易听懂。好了，我们实践一下，这里我开一下 D 盘

里面的 mongodb。



这里要注意的有两点：

<1> logpath: 当我们使用服务寄宿的时候，用眼睛都能想明白肯定不会用 console 来承载日志信息了。

<2> install: 开启安装服务寄宿，很 happy 啊，把管理员的手工操作降低到最小，感谢 mongodb。

好了，console 程序叫我看 log 日志，那我就看看，发现 mongodb 已经提示我们如何开启 mongodb，接着我照做就是了。

```
C:\Windows\system32\cmd.exe - mongo 127.0.0.1:2222/admin

D:\mongodb\bin>net start MongoDB

Mongo DB 服务已经启动成功。

D:\mongodb\bin>mongo 127.0.0.1:2222/admin
MongoDB shell version: 2.0.2
connecting to: 127.0.0.1:2222/admin
>
```

名称	描述	状态	启动类型	登录为
Microsoft iSCSI Initiator Service	管理从这台计算机...		手动	本地系统
Microsoft Software Shadow Copy P...	管理卷影复制服务...		手动	本地系统
Mongo DB	Mongo DB Server	已启动	自动	本地系统
Multimedia Class Scheduler	基于系统范围内的...	已启动	自动	本地系统

还要提醒大家一点的就是，这些命令参数很多很复杂也就很容易忘，不过没关系，数据库给我们提供了一个 **help** 方法，我们可以

拿 **mongod** 和 **mongo** 说事。

**mongod:**





## 二：状态监控

监控可以让我们实时的了解数据库的健康状况以及性能调优，在 **mongodb** 里面给我们提供了三种方式。

### 1: http 监视器

这个我在先前的文章中也提到了，这里就不赘述了。

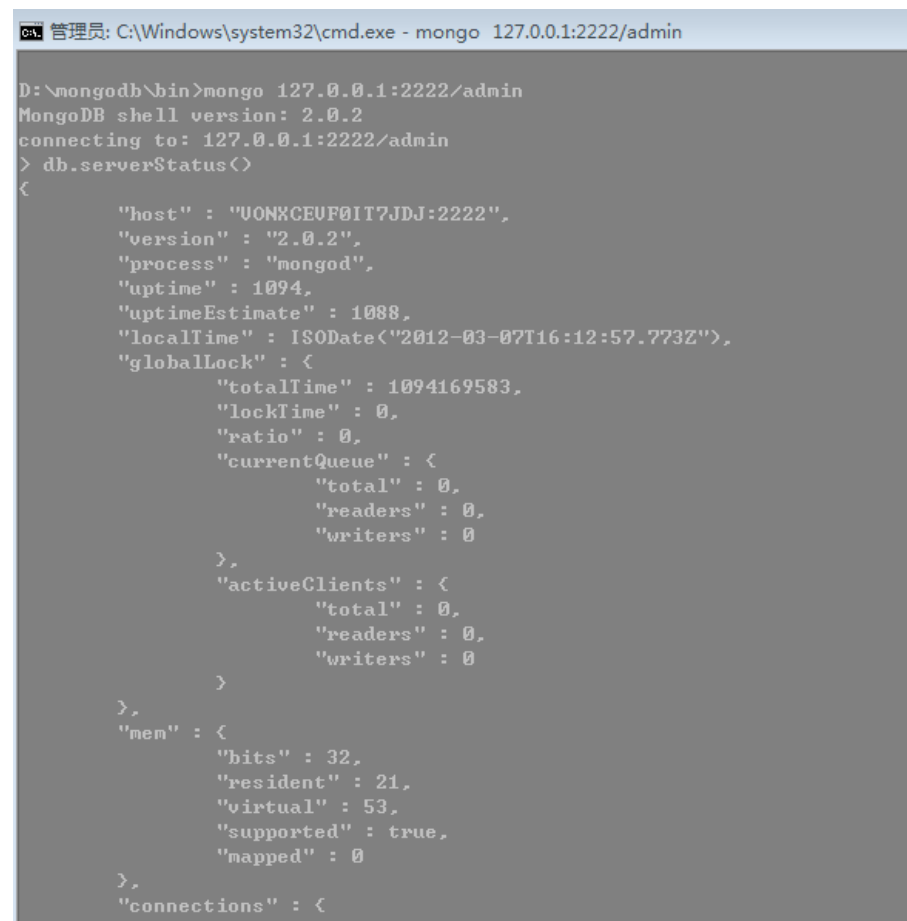
### 2: serverStatus()

这个函数可以获取到 **mongodb** 的服务器统计信息，其中包括：全局锁，索引，用户操作行为等等这些统计信息，对管理员来说非常

重要，具体的参数含义可以参考园友：

<http://www.cnblogs.com/xuegang/archive/2011/10/13/2210339.html>

这里还是截个图混个眼熟。



```
管理员: C:\Windows\system32\cmd.exe - mongo 127.0.0.1:2222/admin

D:\mongodb\bin>mongo 127.0.0.1:2222/admin
MongoDB shell version: 2.0.2
connecting to: 127.0.0.1:2222/admin
> db.serverStatus()
{
  "host" : "UONXCEUF0IT7JDJ:2222",
  "version" : "2.0.2",
  "process" : "mongod",
  "uptime" : 1094,
  "uptimeEstimate" : 1088,
  "localTime" : ISODate("2012-03-07T16:12:57.773Z"),
  "globalLock" : {
    "totalTime" : 1094169583,
    "lockTime" : 0,
    "ratio" : 0,
    "currentQueue" : {
      "total" : 0,
      "readers" : 0,
      "writers" : 0
    },
    "activeClients" : {
      "total" : 0,
      "readers" : 0,
      "writers" : 0
    }
  },
  "mem" : {
    "bits" : 32,
    "resident" : 21,
    "virtual" : 53,
    "supported" : true,
    "mapped" : 0
  },
  "connections" : {
    "current" : 1
```

### 3: mongostat

前面那些统计信息再牛 X，那也是静态统计，不能让我观看实时数据变化，还好，**mongodb** 里面提供了这里要说的 **mongostat** 监视器，这玩意会每秒刷新，在实际生产环境中大有用处，还是截张图，很有意思，是不是感觉大军压境了。

```
管理员: C:\Windows\system32\cmd.exe - mongostat --port 2222
>
>
D:\mongodb\bin>mongostat --port 2222
connected to: 127.0.0.1:2222
insert  query update delete getmore command flushes mapped vsize  res faults locked % i
  0      0      0      0      0      1      0      0m  53m  21m    0      0
  0      0      0      0      0      1      0      0m  53m  21m    0      0
  0      0      0      0      0      1      0      0m  53m  21m   12      0
  0      0      0      0      0      1      0      0m  53m  21m    4      0
  0      0      0      0      0      1      0      0m  53m  21m    0      0
  0      0      0      0      0      1      0      0m  53m  21m    0      0
  0      0      0      0      0      1      0      0m  53m  21m    0      0
  0      0      0      0      0      1      0      0m  53m  21m    0      0
  0      0      0      0      0      1      0      0m  53m  21m    0      0
  0      0      0      0      0      1      0      0m  53m  21m    0      0
insert  query update delete getmore command flushes mapped vsize  res faults locked % i
  0      0      0      0      0      1      0      0m  53m  21m    0      0
  0      0      0      0      0      1      0      0m  53m  21m    0      0
  0      0      0      0      0      1      0      0m  53m  21m    0      0
  0      0      0      0      0      1      0      0m  53m  21m    0      0
  0      0      0      0      0      1      0      0m  53m  21m    0      0
  0      0      0      0      0      1      0      0m  53m  21m    0      0
  0      0      0      0      0      1      0      0m  53m  21m    0      0
  0      0      0      0      0      1      0      0m  53m  21m    0      0
  0      0      0      0      0      1      0      0m  53m  21m    0      0
  0      0      0      0      0      1      0      0m  53m  21m    0      0
insert  query update delete getmore command flushes mapped vsize  res faults locked % i
  0      0      0      0      0      1      0      0m  53m  21m    0      0
  0      0      0      0      0      1      0      0m  53m  21m    0      0
  0      0      0      0      0      1      0      0m  53m  21m    0      0
  0      0      0      0      0      1      1      0m  53m  21m    0      0
```

三：安全认证

作为数据库软件，我们肯定不想谁都可以访问，为了确保数据的安全，**mongodb** 也会像其他的数据库软件一样可以采用用户

验证的方法，那么该怎么做呢？其实很简单，**mongodb** 提供了 **addUser** 方法，还有一个注意点就是如果在 **admin** 数据库中添加

将会被视为“超级管理员”。

```
C:\ 管理员: C:\Windows\system32\cmd.exe - mongo 127.0.0.1:2222/admin
D:\mongodb\bin>mongo 127.0.0.1:2222/admin
MongoDB shell version: 2.0.2
connecting to: 127.0.0.1:2222/admin
> db.addUser("admin","admin")
{ "n" : 0, "connectionId" : 6, "err" : null, "ok" : 1 }
{
  "user" : "admin",
  "readOnly" : false,
  "pwd" : "7c67ef13bbd4cae106d959320af3f704",
  "_id" : ObjectId("4f578bff26dc40003a635a84")
}
> use test
switched to db test
> db.addUser("jack","jack",true)
{ "n" : 0, "connectionId" : 6, "err" : null, "ok" : 1 }
{
  "user" : "jack",
  "readOnly" : true,
  "pwd" : "71cb3d6555d289fc5d2f14904d6d448c",
  "_id" : ObjectId("4f578c0f26dc40003a635a85")
}
>
> _
```

上面的 admin 用户将会被视为超级管理员，“jack”用户追加的第三个参数表示是否是“只读用户”，好了，该添加的我们都添加了，

我们第一次登录时不是采用验证模式，现在我们使用--reinstall 重启服务并以--auth 验证模式登录。

```
C:\ 管理员: C:\Windows\system32\cmd.exe
D:\mongodb\bin>mongod --dbpath=D:\mongodb\db --logpath=D:\mongodb\log.txt --port 2222 --auth --reinstall
all output going to: D:\mongodb\log.txt
D:\mongodb\bin>net start MongoDB
Mongo DB 服务正在启动。
Mongo DB 服务已经启动成功。
D:\mongodb\bin>
```

好了，我们进入 test 集合翻翻数据看看情况，我们发现 jack 用户始终都是没有写入的权限，不管是授权或者未授权。

```
C:\Windows\system32\cmd.exe - mongo 127.0.0.1:2222/test
D:\mongodb\bin>mongo 127.0.0.1:2222/test
MongoDB shell version: 2.0.2
connecting to: 127.0.0.1:2222/test
> db.person.find()
error: {
  "errmsg" : "unauthorized db:test lock type:-1 client:127.0.0.1",
  "code" : 10057
}
> db.person.insert(<{"name":"hxc"}>)
unauthorized
>
> db.auth("jack","jack")
1
>
> db.person.find()
>
> db.person.insert(<{"name":"hxc"}>)
unauthorized
>
_
```

#### 四：备份和恢复

这玩意的重要性我想都不需要我来说了吧，这玩意要是搞不好会死人的，mongodb 里面常用的手段有 3 种。

##### 1： 直接 copy

这个算是最简单的了，不过要注意一点，在服务器运行的情况下直接 copy 是很有风险的，可能 copy 出来时，数据已经遭到

破坏，唯一能保证的就是要暂时关闭下服务器，copy 完后重开。

##### 2: mongodump 和 mongorestore

这个是 mongo 给我们提供的内置工具，很好用，能保证在不关闭服务器的情况下 copy 数据。

为了操作方便，我们先删除授权用户。

```
管理员: C:\Windows\system32\cmd.exe - mongo 127.0.0.1:2222/admin
D:\mongodb\bin>mongo 127.0.0.1:2222/admin
MongoDB shell version: 2.0.2
connecting to: 127.0.0.1:2222/admin
> db.auth("admin","admin")
1
> db.system.users.find()
{ "_id" : ObjectId("4f578bff26dc40003a635a84"), "user" : "admin", "readOnly" : false, "pwd" : "7c67e" }
> db.system.users.remove()
1
> db.system.users.find()
[]
> use test
switched to db test
> db.auth("jack","jack")
1
> db.auth("admin","admin")
0
> db.system.user.remove()
cannot delete from system namespace
> db.system.users.remove()
1
> db.system.users.find()
[]
>
```

好了，我们转入正题，这里我先在 D 盘建立一个 backup 文件夹用于存放 test 数据库。

```
管理员: C:\Windows\system32\cmd.exe
D:\mongodb\bin>mongodump --port 2222 -d test -o D:\mongodb\backup
connected to: 127.0.0.1:2222
DATABASE: test to D:/mongodb/backup/test
    test.system.users to D:/mongodb/backup/test/system.users.bson
        0 objects
    test.system.indexes to D:/mongodb/backup/test/system.indexes.bson
        1 objects
D:\mongodb\bin>
```

计算机 > 开发软件 (D:) > mongodb > backup > test

名称	修改日期	类型	大小
system.indexes.bson	2012/3/8 0:56	BSON 文件	1 KB
system.users.bson	2012/3/8 0:56	BSON 文件	0 KB

快看，数据已经备份过来了，太爽了，现在我们用 mongorestore 恢复过去，记住啊，它是不用关闭机器的。

```
管理员: C:\Windows\system32\cmd.exe

D:\mongodb\bin>mongorestore --port 2222 -d test --drop D:\mongodb\backup\test
connected to: 127.0.0.1:2222
Thu Mar 08 00:59:01 D:/mongodb/backup/test/system.users.bson
Thu Mar 08 00:59:01 going into namespace [test.system.users]
Thu Mar 08 00:59:01 file D:/mongodb/backup/test/system.users.bson empty, skipping
Thu Mar 08 00:59:01 D:/mongodb/backup/test/system.indexes.bson
Thu Mar 08 00:59:01 going into namespace [test.system.indexes]
Thu Mar 08 00:59:01 dropping
Thu Mar 08 00:59:01 < key: { _id: 1 }, ns: "test.system.users", name: "_id_" >
1 objects found
```

提一点的就是 **drop** 选项, 这里是说我将 **test** 数据恢复之前先删除原有数据库里面的数据, 同样大家可以通过 **help** 查看。

### 3: 主从复制

这个我在上上篇有所介绍, 这里也不赘述了。

其实上面的 1, 2 两点都不能保证获取数据的实时性, 因为我们在备份的时候可能还有数据灌在内存中不出来, 那么我们

想说能不能把数据暴力的刷到硬盘上, 当然是可以的, **mongodb** 给我们提供了 **fsync+lock** 机制就能满足我们提的需求。

**fsync+lock** 首先会把缓冲区数据暴力刷入硬盘, 然后给数据库一个写入锁, 其他实例的写入操作全部被阻塞, 直到 **fsync**

**+lock** 释放锁为止。

这里就不测试了。

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

释放锁: `db.$cmd.unlock.findOne()`

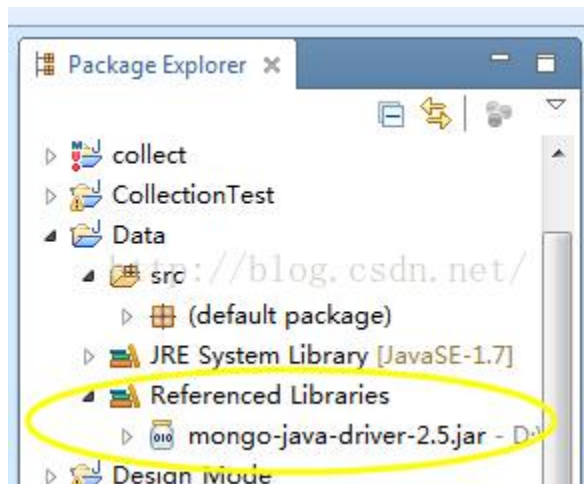
## Java 操作 mongodb

### HelloWorld 程序

学习任何程序的第一步, 都是编写 HelloWorld 程序, 我们也不例外, 看下如何通过 [Java](#) 编写一个 HelloWorld 的程序。

首先, 要通过 [Java](#) 操作 [MongoDB](#), 必须先下载 MongoDB 的 Java 驱动程序 `mongo-java-driver-2.13.2.jar`。

新建一个 Java 工程, 将下载的驱动 jar 包放在库文件路径下, 如图所示:



1、 建立 SimpleTest.java，完成简单的 mongoDB 数据库操作

```
Mongo mongo = new Mongo();
```

这样就创建了一个 MongoDB 的数据库连接对象，它默认连接到当前机器的 localhost 地址，端口是 27017。

```
DB db = mongo.getDB("test");
```

这样就获得了一个 test 的数据库，如果 mongoDB 中没有创建这个数据库也是可以正常运行的。如果你读过上一篇文章就知道，mongoDB 可以在没有创建这个数据库的情况下，完成数据的添加操作。当添加的时候，没有这个库，mongoDB 会自动创建当前数据库。

得到了 db，下一步我们要获取一个“聚集集合 DBCollection”，通过 db 对象的 getCollection 方法来完成。

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

这样就获得了一个 DBCollection，它相当于我们数据库的“表”。

查询所有数据

```
DBCursor cur = users.find();  
while (cur.hasNext()) {  
    System.out.println(cur.next());  
}
```

完整源码

```
package com.hoo.test;  
  
import java.net.UnknownHostException;  
import com.mongodb.DB;  
import com.mongodb.DBCollection;
```



```

import com.mongodb.DBCursor;
import com.mongodb.Mongo;
import com.mongodb.MongoException;
import com.mongodb.util.JSON;

/**
 * <b>function:</b>MongoDB 简单示例
 */
public class SimpleTest {

    public static void main(String[] args) throws UnknownHostException,
MongoException {
        Mongo mg = new Mongo();
        //查询所有的 Database
        for (String name : mg.getDatabaseNames()) {
            System.out.println("dbName: " + name);
        }

        DB db = mg.getDB("test");
        //查询所有的聚集集合
        for (String name : db.getCollectionNames()) {
            System.out.println("collectionName: " + name);
        }

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

        //查询所有的数据
        DBCursor cur = users.find();
        while (cur.hasNext()) {
            System.out.println(cur.next());
        }
        System.out.println(cur.count());
        System.out.println(cur.getCursorId());
        System.out.println(JSON.serialize(cur));
    }
}

```

2、完成 CRUD 操作，首先建立一个 MongoDB4CRUDTest.java，基本测试代码如下：

```

package com.hoo.test;

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

```

```

import org.bson.types.ObjectId;
import org.junit.After;
import org.junit.Before;
import org.junit.Test;
import com.mongodb.BasicDBObject;
import com.mongodb.Bytes;
import com.mongodb.DB;
import com.mongodb.DBCollection;
import com.mongodb.DBCursor;
import com.mongodb.DBObject;
import com.mongodb.Mongo;
import com.mongodb.MongoException;
import com.mongodb.QueryOperators;
import com.mongodb.util.JSON;

/**
 * <b>function:</b>实现 MongoDB 的 CRUD 操作
 */
public class MongoDBCRUDTest {

    //private Mongo mg = null;
    MongoClient mg = null;
    private DB db;
    private DBCollection users;

    @Before
    public void init() {
        try {
            //mg = new Mongo(); //过时
            //mg = new Mongo("localhost", 27017); //过时
            mg = new MongoClient("localhost", 27017); //最新用法
        } catch (UnknownHostException e) {
            e.printStackTrace();
        } catch (MongoException e) {
            e.printStackTrace();
        }
        //获取 temp DB; 如果默认没有创建, mongodb 会自动创建
        db = mg.getDB("temp");
        //获取 users DBCollection; 如果默认没有创建, mongodb 会自动创建
        users = db.getCollection("users");
    }

    @After
    public void destory() {

```

```

        if (mg != null)
            mg.close();
        mg = null;
        db = null;
        users = null;
        System.gc();
    }

    public void print(Object o) {
        System.out.println(o);
    }
}

```

### 3、 添加操作

在添加操作之前，我们需要写个查询方法，来查询所有的数据。代码如下：

```

/**
 * <b>function:</b> 查询所有数据
 * @author hoojo
 * @createDate 2011-6-2 下午 03:22:40
 */
private void queryAll() {
    print("查询 users 的所有数据: ");
    //db 游标
    DBCursor cur = users.find();
    while (cur.hasNext()) {
        print(cur.next());
    }
}

@Test
public void add() {
    //先查询所有数据
    queryAll();
    print("count: " + users.count());

    DBObject user = new BasicDBObject();
    user.put("name", "hoojo");
    user.put("age", 24);
    //users.save(user) 保存, getN() 获取影响行数
    //print(users.save(user).getN());

    //扩展字段, 随意添加字段, 不影响现有数据
}

```

```

user.put("sex", "男");
print(users.save(user).getN());

//添加多条数据, 传递 Array 对象
print(users.insert(user, new BasicDBObject("name", "tom")).getN());

//添加 List 集合
List<DBObject> list = new ArrayList<DBObject>();
list.add(user);
DBObject user2 = new BasicDBObject("name", "lucy");
user.put("age", 22);
list.add(user2);
//添加 List 集合
print(users.insert(list).getN());

//查询下数据, 看看是否添加成功
print("count: " + users.count());
queryAll();
}

```

#### 4、 删除数据

```

@Test
public void remove() {
    queryAll();
    print("删除 id = 4de73f7acd812d61b4626a77: " + users.remove(new
BasicDBObject("_id", new
ObjectId("4de73f7acd812d61b4626a77"))).getN());
    print("remove age >= 24: " + users.remove(new BasicDBObject("age",
new BasicDBObject("$gte", 24))).getN());
}

```

#### 5、 修改数据

```

@Test
public void modify() {
    print("修改: " + users.update(new BasicDBObject("_id", new
ObjectId("4dde25d06be7c53ffbd70906")), new BasicDBObject("age",
99)).getN());
    print("修改: " + users.update(
        new BasicDBObject("_id", new
ObjectId("4dde2b06feb038463ff09042")),
        new BasicDBObject("age", 121),

```

```

        true, //如果数据库不存在, 是否添加
        false //多条修改
    ).getN());

print("修改: " + users.update(
    new BasicDBObject("name", "haha"),
    new BasicDBObject("name", "dingding"),
    true, //如果数据库不存在, 是否添加
    true //false 只修改第一天, true 如果有多条就不修改
).getN());

//当数据库不存在就不修改、不添加数据, 当多条数据就不修改
//print("修改多条: " + coll.updateMulti(new BasicDBObject("_id", new
ObjectId("4dde23616be7c19df07db42c")), new BasicDBObject("name",
"199")));
}

```

## 6、 查询数据

```

@Test
public void query() {
    //查询所有
    //queryAll();

    //查询 id = 4de73f7acd812d61b4626a77
    print("find id = 4de73f7acd812d61b4626a77: " + users.find(new
BasicDBObject("_id", new
ObjectId("4de73f7acd812d61b4626a77"))).toArray());

    //查询 age = 24
    print("find age = 24: " + users.find(new BasicDBObject("age",
24)).toArray());

    //查询 age >= 24
    print("find age >= 24: " + users.find(new BasicDBObject("age", new
BasicDBObject("$gte", 24))).toArray());
    print("find age <= 24: " + users.find(new BasicDBObject("age", new
BasicDBObject("$lte", 24))).toArray());

    print("查询 age!=25: " + users.find(new BasicDBObject("age", new
BasicDBObject("$ne", 25))).toArray());
    print("查询 age in 25/26/27: " + users.find(new BasicDBObject("age",
new BasicDBObject(QueryOperators.IN, new int[] { 25, 26,
27 }))).toArray());
}

```

```

        print("查询 age not in 25/26/27: " + users.find(new BasicDBObject("age",
new BasicDBObject(QueryOperators.NIN, new int[] { 25, 26,
27 }))).toArray());

        print("查询 age exists 排序:" + users.find(new BasicDBObject("age", new
BasicDBObject(QueryOperators.EXISTS, true))).toArray());

        print("只查询 age 属性: " + users.find(null, new BasicDBObject("age",
true)).toArray());

        print("只查属性: " + users.find(null, new BasicDBObject("age", true),
0, 2).toArray());

        print("只查属性: " + users.find(null, new BasicDBObject("age", true),
0, 2, Bytes.QUERYOPTION_NOTIMEOUT).toArray());

        //只查询一条数据, 多条去第一条
        print("findOne: " + users.findOne());
        print("findOne: " + users.findOne(new BasicDBObject("age", 26)));
        print("findOne: " + users.findOne(new BasicDBObject("age", 26), new
BasicDBObject("name", true)));

        //查询修改、删除
        print("findAndRemove 查询 age=25 的数据, 并且删除: " +
users.findAndRemove(new BasicDBObject("age", 25)));

        //查询 age=26 的数据, 并且修改 name 的值为 Abc
        print("findAndModify: " + users.findAndModify(new
BasicDBObject("age", 26), new BasicDBObject("name", "Abc")));
        print("findAndModify: " + users.findAndModify(
            new BasicDBObject("age", 28), //查询 age=28 的数据
            new BasicDBObject("name", true), //查询 name 属性
            new BasicDBObject("age", true), //按照 age 排序
            false, //是否删除, true 表示删除
            new BasicDBObject("name", "Abc"), //修改的值, 将 name 修改成 Abc
            true,
            true));

        queryAll();
    }
}

```

mongoDB 不支持联合查询、子查询, 这需要我们自己在程序中完成。将查询的结果集在 Java 查询中进行需要的过滤即可。

## 7、其他操作

```
public void testOthers() {
```

```

DBObject user = new BasicDBObject();
user.put("name", "hoojo");
user.put("age", 24);

//JSON 对象转换
print("serialize: " + JSON.serialize(user));
//反序列化
print("parse: " + JSON.parse("{ \"name\" : \"hoojo\" , \"age\" : 24}"));

print("判断 temp Collection 是否存在: " + db.collectionExists("temp"));

//如果不存在就创建
if (!db.collectionExists("temp")) {
    DBObject options = new BasicDBObject();
    options.put("size", 20);
    options.put("capped", 20);
    options.put("max", 20);
    print(db.createCollection("account", options));
}

//设置 db 为只读
db.setReadOnly(true);

//只读不能写入数据
db.getCollection("test").save(user);
}

```

## Mongodb 中如何插入数据

下面,讲解下如何使用 4 种方式,将 JSON 数据插入到 Mongodb 中去。首先我们准备 JSON 格式的数据,如下:

```

{
  "database": "mkyongDB",
  "table": "hosting",
  "detail": {
    {
      records : 99,
      index : "vps_index1",
      active : "true"
    }
  }
}

```

我们希望用不同的方式，通过 JAVA 代码向 Mongoddb 插入以上格式的 JSON 数据

第一种方法，是使用 BasicDBObject，方法如下代码所示：

```
BasicDBObject document = new BasicDBObject();
document.put("database", "mkyongDB");
document.put("table", "hosting");
BasicDBObject documentDetail = new BasicDBObject();
documentDetail.put("records", "99");
documentDetail.put("index", "vps_index1");
documentDetail.put("active", "true");
document.put("detail", documentDetail);
collection.insert(document);
```

第二种方法是使用 BasicDBObjectBuilder 对象，如下代码所示：

```
BasicDBObjectBuilder documentBuilder = BasicDBObjectBuilder.start()
.add("database", "mkyongDB")
.add("table", "hosting");
BasicDBObjectBuilder documentBuilderDetail = BasicDBObjectBuilder.start()
.add("records", "99")
.add("index", "vps_index1")
.add("active", "true");
documentBuilder.add("detail", documentBuilderDetail.get());
collection.insert(documentBuilder.get());
```

第三种方法是使用 Map 对象，代码如下：

```
Map documentMap = new HashMap();
documentMap.put("database", "mkyongDB");
documentMap.put("table", "hosting");
Map documentMapDetail = new HashMap();
documentMapDetail.put("records", "99");
documentMapDetail.put("index", "vps_index1");
documentMapDetail.put("active", "true");
documentMap.put("detail", documentMapDetail);
collection.insert(new BasicDBObject(documentMap));
```

第四种方法，也就是最简单的，即直接插入 JSON 格式数据

```
String json = '{"database' : 'mkyongDB','table' : 'hosting','+
'detail' : {'records' : 99, 'index' : 'vps_index1', 'active' : 'true'}}';
DBObject dbObject =(DBObject)JSON.parse(json);
```



```
collection.insert(dbObject);
```

这里使用了 JSON 的 parse 方法, 将解析后的 JSON 字符串转变为 DBObject 对象后再直接插入到 collection 中去。

完整的代码如下所示:

```
package com.mkyong.core;
import java.net.UnknownHostException;
import java.util.HashMap;
import java.util.Map;
import com.mongodb.BasicDBObject;
import com.mongodb.BasicDBObjectBuilder;
import com.mongodb.DB;
import com.mongodb.DBCollection;
import com.mongodb.DBCursor;
import com.mongodb.DBObject;
import com.mongodb.Mongo;
import com.mongodb.MongoException;
import com.mongodb.util.JSON;
/**
 * Java MongoDB : Insert a Document
 *
 */
public class InsertDocumentApp {
    public static void main(String[] args) {
        try {
            Mongo mongo = new Mongo("localhost", 27017);
            DB db = mongo.getDB("yourdb");
            // get a single collection
            DBCollection collection = db.getCollection("dummyColl");
            // BasicDBObject example
            System.out.println("BasicDBObject example...");
            BasicDBObject document = new BasicDBObject();
            document.put("database", "mkyongDB");
            document.put("table", "hosting");
            BasicDBObject documentDetail = new BasicDBObject();
            documentDetail.put("records", "99");
            documentDetail.put("index", "vps_index1");
            documentDetail.put("active", "true");
            document.put("detail", documentDetail);
```

```

collection.insert(document);
DBCursor cursorDoc = collection.find();
while(cursorDoc.hasNext()){
    System.out.println(cursorDoc.next());
}
collection.remove(new BasicDBObject());
// BasicDBObjectBuilder example
System.out.println("BasicDBObjectBuilder example...");
BasicDBObjectBuilder documentBuilder = BasicDBObjectBuilder.start()
.add("database", "mkyongDB")
.add("table", "hosting");
BasicDBObjectBuilder documentBuilderDetail = BasicDBObjectBuilder.start()
.add("records", "99")
.add("index", "vps_index1")
.add("active", "true");
documentBuilder.add("detail", documentBuilderDetail.get());
collection.insert(documentBuilder.get());
DBCursor cursorDocBuilder = collection.find();
while(cursorDocBuilder.hasNext()){
    System.out.println(cursorDocBuilder.next());
}
collection.remove(new BasicDBObject());
// Map example
System.out.println("Map example...");
Map documentMap =new HashMap();
documentMap.put("database", "mkyongDB");
documentMap.put("table", "hosting");
Map documentMapDetail =new HashMap();
documentMapDetail.put("records", "99");
documentMapDetail.put("index", "vps_index1");
documentMapDetail.put("active", "true");
documentMap.put("detail", documentMapDetail);
collection.insert(new BasicDBObject(documentMap));
DBCursor cursorDocMap = collection.find();
while(cursorDocMap.hasNext()){
    System.out.println(cursorDocMap.next());
}
collection.remove(new BasicDBObject());
// JSON parse example
System.out.println("JSON parse example...");

```

```
String json="{ 'database' : 'mkyongDB','table' : 'hosting','+
'detail' : { 'records' : 99, 'index' : 'vps_index1', 'active' : 'true' } }";
DBObject dbObject =(DBObject)JSON.parse(json);
collection.insert(dbObject);
DBCursor cursorDocJSON = collection.find();
while(cursorDocJSON.hasNext()){
System.out.println(cursorDocJSON.next());
}
collection.remove(new BasicDBObject());
}catch(UnknownHostException e){
e.printStackTrace();
}catch(MongoException e){
e.printStackTrace();
}
}
}
```

## 更新 Document

假设如下的 JSON 格式的数据已经保存到 Mongoddb 中去了，现在要更新相关的数据。

```
{ "_id" : { "$oid" : "x" } , "hosting" : "hostA" , "type" : "vps" , "clients" : 1000}
{ "_id" : { "$oid" : "x" } , "hosting" : "hostB" , "type" : "dedicated server" , "clients" : 100}
{ "_id" : { "$oid" : "x" } , "hosting" : "hostC" , "type" : "vps" , "clients" : 900}
```

假设现在要将 **hosting** 中值为 **hostB** 的进行更新，则可以使用如下的方法：

```
BasicDBObject newDocument =new BasicDBObject();
newDocument.put("hosting", "hostB");
newDocument.put("type", "shared host");
newDocument.put("clients", 111);
collection.update(new BasicDBObject().append("hosting", "hostB"), newDocument);
```

可以看到，这里依然使用了 **BasicDBObject** 对象，并为其赋值了新的值后，然后使用 **collection** 的 **update** 方法，即可更新该对象。

更新后的输出如下：

```
{ "_id" : { "$oid" : "x" } , "hosting" : "hostA" , "type" : "vps" , "clients" : 1000}
{ "_id" : { "$oid" : "x" } , "hosting" : "hostB" , "type" : "shared host" , "clients" : 111}
{ "_id" : { "$oid" : "x" } , "hosting" : "hostC" , "type" : "vps" , "clients" : 900}
```

另外, 还可以使用 mongodb 中的 \$inc 修饰符号去对某个值进行更新, 比如, 要将 hosting 值为 hostB 的 document 的 clients 的值得更新为 199(即  $100+99=199$ ), 可以这样:

```
BasicDBObject newDocument = new BasicDBObject().append("$inc",
new BasicDBObject().append("clients", 99));
collection.update(new BasicDBObject().append("hosting", "hostB"), newDocument);
```

则输出如下:

```
{"_id": {"$oid": "x"}, "hosting": "hostA", "type": "vps", "clients": 1000}
{"_id": {"$oid": "x"}, "hosting": "hostB", "type": "dedicated server", "clients": 199}
{"_id": {"$oid": "x"}, "hosting": "hostC", "type": "vps", "clients": 900}
```

接下来, 讲解 \$set 修饰符的使用。比如要把 hosting 中值为 hostA 的 document 中的 type 的值进行修改, 则可以如下实现:

```
BasicDBObject newDocument3 = new BasicDBObject().append("$set",
new BasicDBObject().append("type", "dedicated server"));
collection.update(new BasicDBObject().append("hosting", "hostA"), newDocument3);
```

则输出如下, 把 type 的值从 vps 改为 dedicated server:

```
{"_id": {"$oid": "x"}, "hosting": "hostB", "type": "dedicated server", "clients": 100}
{"_id": {"$oid": "x"}, "hosting": "hostC", "type": "vps", "clients": 900}
{"_id": {"$oid": "x"}, "hosting": "hostA", "clients": 1000, "type": "dedicated server"}
```

要注意的是, 如果不使用 \$set 的修饰符, 而只是如下代码:

```
BasicDBObject newDocument3 = new BasicDBObject().append("type", "dedicated server");
collection.update(new BasicDBObject().append("hosting", "hostA"), newDocument3);
```

则会将所有的三个 document 的 type 类型都改为 dedicated server 了, 因此要使用 \$set 以更新特定的 document 的特定的值。

如果要更新多个 document 中相同的值, 可以使用 \$multi, 比如, 要把所有 vps 为 type 的 document, 将它们的 clients 的值更新为 888, 可以如下实现:

```
BasicDBObject updateQuery = new BasicDBObject().append("$set",
new BasicDBObject().append("clients", "888"));
collection.update(new BasicDBObject().append("type", "vps"), updateQuery, false, true);
```

输出如下:

```
{"_id": {"$oid": "x"}, "hosting": "hostA", "clients": "888", "type": "vps"}
{"_id": {"$oid": "x"}, "hosting": "hostB", "type": "dedicated server", "clients": 100}
{"_id": {"$oid": "x"}, "hosting": "hostC", "clients": "888", "type": "vps"}
```

最后，还是给出更新 document 的完整例子：

```
package com.liao;
import java.net.UnknownHostException;
import com.mongodb.BasicDBObject;
import com.mongodb.DB;
import com.mongodb.DBCollection;
import com.mongodb.DBCursor;
import com.mongodb.Mongo;
import com.mongodb.MongoException;

publicclass UpdateDocumentApp {
    publicstaticvoid printAllDocuments(DBCollection collection){
        DBCursor cursor = collection.find();
        while (cursor.hasNext()) {
            System.out.println(cursor.next());
        }
    }

    publicstaticvoid removeAllDocuments(DBCollection collection){
        collection.remove(new BasicDBObject());
    }

    publicstaticvoid insertDummyDocuments(DBCollection collection){
        BasicDBObject document = new BasicDBObject();
        document.put("hosting", "hostA");
        document.put("type", "vps");
        document.put("clients", 1000);
        BasicDBObject document2 = new BasicDBObject();
        document2.put("hosting", "hostB");
        document2.put("type", "dedicated server");
        document2.put("clients", 100);
        BasicDBObject document3 = new BasicDBObject();
        document3.put("hosting", "hostC");
        document3.put("type", "vps");
        document3.put("clients", 900);
        collection.insert(document);
        collection.insert(document2);
        collection.insert(document3);
    }

    publicstaticvoid main(String[] args) {
        try {
```

```

Mongo mongo = new Mongo("localhost", 27017);
DB db = mongo.getDB("yourdb");
DBCollection collection = db.getCollection("dummyColl");
System.out.println("Testing 1...");
insertDummyDocuments(collection);
//find hosting = hostB, and update it with new document
BasicDBObject newDocument = new BasicDBObject();
newDocument.put("hosting", "hostB");
newDocument.put("type", "shared host");
newDocument.put("clients", 111);
collection.update(new BasicDBObject().append("hosting", "hostB"), newDocument);
printAllDocuments(collection);
removeAllDocuments(collection);
System.out.println("Testing 2...");
insertDummyDocuments(collection);
BasicDBObject newDocument2 = new BasicDBObject().append("$inc",
new BasicDBObject().append("clients", 99));
collection.update(new BasicDBObject().append("hosting", "hostB"), newDocument2);
printAllDocuments(collection);
removeAllDocuments(collection);
System.out.println("Testing 3...");
insertDummyDocuments(collection);
BasicDBObject newDocument3 = new BasicDBObject().append("$set",
new BasicDBObject().append("type", "dedicated server"));
collection.update(new BasicDBObject().append("hosting", "hostA"), newDocument3);
printAllDocuments(collection);
removeAllDocuments(collection);
System.out.println("Testing 4...");
insertDummyDocuments(collection);
BasicDBObject updateQuery = new BasicDBObject().append("$set",
new BasicDBObject().append("clients", "888"));
collection.update(
new BasicDBObject().append("type", "vps"), updateQuery, false, true);
printAllDocuments(collection);
removeAllDocuments(collection);
System.out.println("Done");
} catch (UnknownHostException e) {
e.printStackTrace();
} catch (MongoException e) {
e.printStackTrace();
}

```

```
}  
}  
}
```

## 查询 Document

下面学习如何查询 document,先用下面的代码往数据库中插入 1-10 数字:

```
for(int i=1; i <=10; i++){  
collection.insert(new BasicDBObject().append("number", i));  
  
}
```

接下来,看下如下的例子:

### 1) 获得数据库中的第一个 document:

```
DBObject doc = collection.findOne();  
System.out.println(doc);
```

输出为:

```
{"_id" : {"$oid" : "4dc7f7b7bd0fb9a86c6c80bd"} , "number" : 1}
```

### 2) 获得 document 的集合

```
DBCursor cursor = collection.find();  
while(cursor.hasNext()){  
System.out.println(cursor.next());  
}
```

这里,使用 collection.find()方法,获得当前数据库中所有的 documents 对象集合  
然后通过对 DBCursor 对象集合的遍历,即可输出当前所有 documents。输出如下:

```
{"_id" : {"$oid" : "4dc7f7b7bd0fb9a86c6c80bd"} , "number" : 1}  
//.....中间部分省略,为 2 到 9 的输出  
{"_id" : {"$oid" : "4dc7f7b7bd0fb9a86c6c80c6"} , "number" : 10}
```

### 3) 获取指定的 document

比如要获得 number=5 的 document 对象内容,可以使用 collection 的 find 方法即可,  
如下:

```
BasicDBObject query =new BasicDBObject();  
query.put("number", 5);  
DBCursor cursor = collection.find(query);  
while(cursor.hasNext()){
```

```
System.out.println(cursor.next());
}
```

即输出：

```
{"_id": {"$oid": "4dc7f7b7bd0fb9a86c6c80c1"}, "number": 5}
```

#### 4) 使用 in 操作符号

在 mongodb 中，也可以使用 in 操作符，比如要获得 number=9 和 number=10 的 document 对象，可以如下操作：

```
BasicDBObject query = new BasicDBObject();
List list = new ArrayList();
list.add(9);
list.add(10);
query.put("number", new BasicDBObject("$in", list));
DBCursor cursor = collection.find(query);
while(cursor.hasNext()){
    System.out.println(cursor.next());
}
```

这里使用了一个 List，并将 list 传入到 BasicDBObject 的构造函数中，并使用了 in 操作符号，输出如下：

```
{"_id": {"$oid": "4dc7f7b7bd0fb9a86c6c80c5"}, "number": 9}
{"_id": {"$oid": "4dc7f7b7bd0fb9a86c6c80c6"}, "number": 10}
```

#### 5) 使用 >, < 等比较符号

在 mongodb 中，也可以使用比如 >, < 等数量比较符号，比如要输出 number > 5 的 document 集合，则使用 “\$gt” 即可，同理，小于关系则使用 \$lt，例子如下：

```
BasicDBObject query = new BasicDBObject();
query.put("number", new BasicDBObject("$gt", 5));
DBCursor cursor = collection.find(query);
while(cursor.hasNext()){
    System.out.println(cursor.next());
}
```

输出如下：

```
{"_id": {"$oid": "4dc7f7b7bd0fb9a86c6c80c2"}, "number": 6}
{"_id": {"$oid": "4dc7f7b7bd0fb9a86c6c80c3"}, "number": 7}
{"_id": {"$oid": "4dc7f7b7bd0fb9a86c6c80c4"}, "number": 8}
{"_id": {"$oid": "4dc7f7b7bd0fb9a86c6c80c5"}, "number": 9}
{"_id": {"$oid": "4dc7f7b7bd0fb9a86c6c80c6"}, "number": 10}
```



也可以多个比较符号一起使用，比如要输出 `number>5` 和 `number<8` 的 document,则如下：

```
BasicDBObject query =new BasicDBObject();
query.put("number", new BasicDBObject("$gt", 5).append("$lt", 8));
DBCursor cursor = collection.find(query);
while(cursor.hasNext()){
System.out.println(cursor.next());
}
```

同样，如果是不等于的关系的话，可以使用 `$ne` 操作符，如下：

```
BasicDBObject query5 =new BasicDBObject();
query5.put("number", new BasicDBObject("$ne", 8));
DBCursor cursor6 = collection.find(query5);
while(cursor6.hasNext()){
System.out.println(cursor6.next());
}
```

以上输出 `number=8` 之外的所有 document。

## 删除 document

下面我们学习如何删除 document，依然以上面的已插入的 1-10 的 documents 集合为例说明：

### 1) 删除第一个 document

```
DBObject doc = collection.findOne();
collection.remove(doc);
```

### 2) 删除指定的 document

比如删除 `number=2` 的 document,如下方法：

```
BasicDBObject document =new BasicDBObject();
document.put("number", 2);
collection.remove(document);
```

要注意的是，如下的方法将只会删除 `number=3` 的 document。

```
BasicDBObject document =new BasicDBObject();
document.put("number", 2);
document.put("number", 3);
collection.remove(document);
```

### 3) 使用 in 操作符号指定删除 document

下面的例子将同时删除 `number=4` 和 `number=5` 的 document，使用的是 `in` 操作符

```
BasicDBObject query2 =new BasicDBObject();
List list =new ArrayList();
list.add(4);
list.add(5);
query2.put("number", new BasicDBObject("$in", list));
collection.remove(query2);
```

#### 4) 使用 “\$gt” 删除大于某个值的 document

```
BasicDBObject query =new BasicDBObject();
query.put("number", new BasicDBObject("$gt", 9));
collection.remove(query);
```

以上会删除 number=10 的 document。

#### 5) 删除所有的 document

```
DBCursor cursor = collection.find();
while(cursor.hasNext()){
collection.remove(cursor.next());
}
```

### 保存图片到 Mongodb

下面将讲解如何使用 Java MongoDB GridFS API 去保存图片等二进制文件到 Mongodb，关于 Java MongoDB GridFS API 的详细论述，请参考 <http://www.mongodb.org/display/DOCS/GridFS+Specification>

#### 1) 保存图片

代码段如下：

```
String newFileName = "mkyong-java-image";
File imageFile =newFile("c:\\JavaWebHosting.png");
GridFS gfsPhoto =new GridFS(db, "photo");
GridFSInputFile gfsFile = gfsPhoto.createFile(imageFile);
gfsFile.setFilename(newFileName);
gfsFile.save();
```

这里，将 c 盘下的 JavaWebHosting.png 保存到 mongodb 中去，并命名为 mkyong-java-image。

#### 2) 读取图片信息

代码段如下

```
String newFileName = "mkyong-java-image";
GridFS gfsPhoto =new GridFS(db, "photo");
GridFSDBFile imageForOutput = gfsPhoto.findOne(newFileName);
```

```
System.out.println(imageForOutput);
```

将会输出 JSON 格式的结果;

```
{
  "_id" :
  {
    "$oid" : "4dc9511a14a7d017fee35746"
  } ,
  "chunkSize" : 262144 ,
  "length" : 22672 ,
  "md5" : "1462a6cfa27669af1d8d21c2d7dd1f8b" ,
  "filename" : "mkyong-java-image" ,
  "contentType" : null ,
  "uploadDate" :
  {
    "$date" : "2011-05-10T14:52:10Z"
  } ,
  "aliases" : null
}
```

可以看到，输出的是文件的属性相关信息。

### 3) 输出已保存的所有图片

下面代码段，输出所有保存在 photo 命名空间下的图片信息：

```
GridFS gfsPhoto =new GridFS(db, "photo");
DBCursor cursor = gfsPhoto.getFileList();
while(cursor.hasNext()){
    System.out.println(cursor.next());
}
```

### 4) 从数据库中读取一张图片并另存

下面的代码段，从数据库中读取一张图片并另存为另外一张图片到磁盘中

```
String newFileName = "mkyong-java-image";
GridFS gfsPhoto =new GridFS(db, "photo");
GridFSDBFile imageForOutput = gfsPhoto.findOne(newFileName);
imageForOutput.writeTo("c:\\JavaWebHostingNew.png");
```

### 5) 删除图片

```
String newFileName = "mkyong-java-image";
GridFS gfsPhoto =new GridFS(db, "photo");
gfsPhoto.remove(gfsPhoto.findOne(newFileName));
```

## 如何将 JSON 数据格式转化为 DBObject 格式

在 mongodb 中，可以使用 com.mongodb.util.JSON 类，将 JSON 格式的字符串转变为 DBObject 对象。MongoDB for JAVA 驱动中提供了用于向数据库中存储普通对象的接口 DBObject，当一个文档从 MongoDB 中取出时，它会自动把文档转换成 DBObject 接口类型，要将它实例化为需要的对象。比如：

```
{
  'name' : 'mkyong',
  'age' : 30
}
```

这样的 JSON 格式字符串，转换方法为：

```
DBObject dbObject =(DBObject) JSON.parse("{\"name':'mkyong', 'age':30}");
```

完整的代码如下：

```
package com.mkyong.core;
import java.net.UnknownHostException;
import com.mongodb.DB;
import com.mongodb.DBCollection;
import com.mongodb.DBCursor;
import com.mongodb.DBObject;
import com.mongodb.Mongo;
import com.mongodb.MongoException;
import com.mongodb.util.JSON;
/**
 * Java MongoDB : Convert JSON data to DBObject
 */
public class App {
    public static void main(String[] args){
        try{
            Mongo mongo = new Mongo("localhost", 27017);
            DB db = mongo.getDB("yourdb");
            DBCollection collection = db.getCollection("dummyColl");
            DBObject dbObject =(DBObject) JSON
                .parse("{\"name':'mkyong', 'age':30}");
            collection.insert(dbObject);
            DBCursor cursorDoc = collection.find();
            while(cursorDoc.hasNext()){
                System.out.println(cursorDoc.next());
            }
        }
    }
}
```

```
System.out.println("Done");
}catch(UnknownHostException e){
e.printStackTrace();
}catch(MongoException e){
e.printStackTrace();
}
}
}
```

则输出为:

```
{"_id": {"$oid": "4dc9ebb5237f275c2fe4959f"}, "name": "mkyong", "age": 30}
Done
```

可以看到，将 JSON 格式的数据类型直接转换为 mongodb 中的文档类型并输出。

参考资料:

[https://www.tutorialspoint.com/mongodb/mongodb\\_java.htm](https://www.tutorialspoint.com/mongodb/mongodb_java.htm)