

目录

第一天 基础入门.....2

第二天 细说增删查改9

第三天 细说高级操作 15

第四天 索引操作..... 22

第五天 主从复制..... 30

第六天 分片技术.....41

第七天 运维技术.....46

第八天 驱动实践..... 57

第一天 基础入门

关于 mongodb 的好处，优点之类的这里就不说了，唯一要讲的一点就是 mongodb 中有三元素：数据库，集合，文档，其中“集合”

就是对应关系数据库中的“表”，“文档”对应“行”。

一： 下载

上 [MongoDB 官网](#)，我们发现 有 32bit 和 64bit，这个就要看你系统了，不过这里有两点注意：

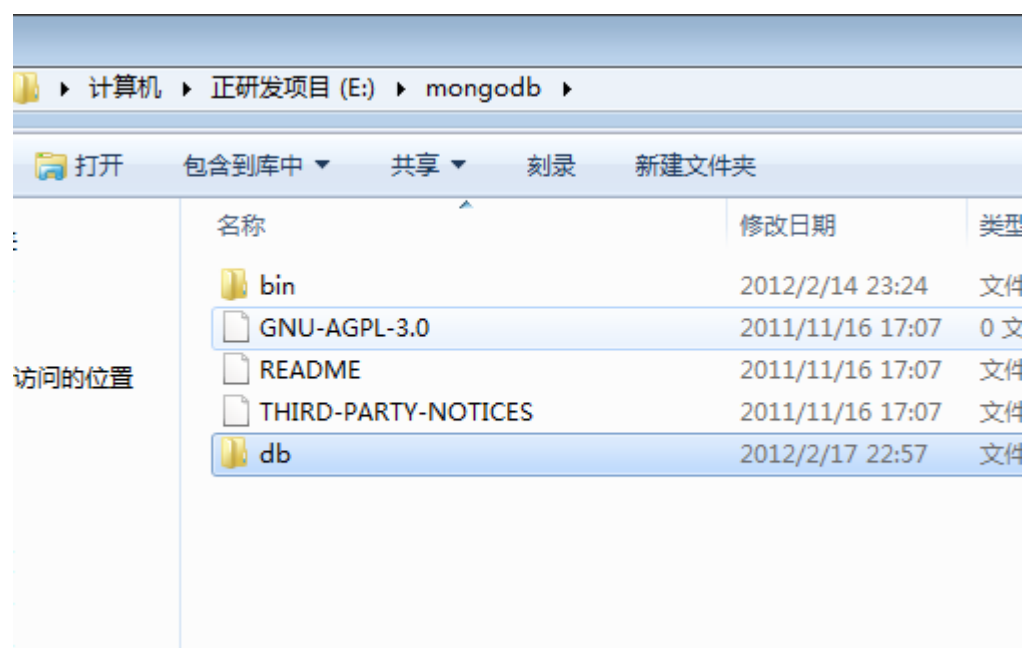
①：根据业界规则，偶数为“稳定版”（如：1.6.X，1.8.X），奇数为“开发版”（如：1.7.X，1.9.X），这两个版本的区别相信大家都知道吧。

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

我这里就下载“2.0.2 版本，32bit”，ok，下载之后我就放到“E 盘”，改下文件夹名字为“mongodb”。

二： 启动

①：启动之前，我们要给 mongodb 指定一个文件夹，这里取名为“db”，用来存放 mongodb 的数据。



②：微软徽标+R，输入 cmd，首先找到“mongodb”的路径，然后运行 mongod 开启命令，同时用--dbpath 指定数据存放地点为“db”文件夹。

```
管理员: C:\Windows\system32\cmd.exe - mongod --dbpath=E:\mongodb\db
Microsoft Windows [版本 6.1.7601]
版权所有 (c) 2009 Microsoft Corporation。保留所有权利。

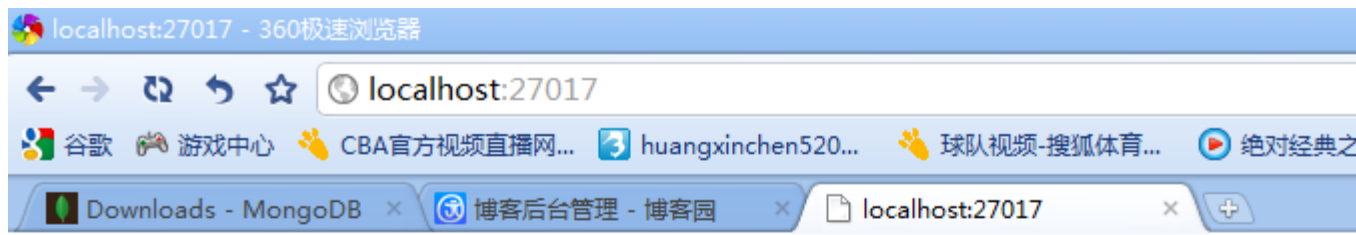
C:\Users\Administrator>E:

E:\>cd mongodb\bin

E:\mongodb\bin>mongod --dbpath=E:\mongodb\db
Fri Feb 17 23:03:29
Fri Feb 17 23:03:29 warning: 32-bit servers don't have journaling enabled by default
Fri Feb 17 23:03:29
Fri Feb 17 23:03:29 [initandlisten] MongoDB starting : pid=1520 port=27017 dbpath=E:
Fri Feb 17 23:03:29 [initandlisten]
Fri Feb 17 23:03:29 [initandlisten] ** NOTE: when using MongoDB 32 bit, you are limi
Fri Feb 17 23:03:29 [initandlisten] **      see http://blog.mongodb.org/post/137788
Fri Feb 17 23:03:29 [initandlisten] **      with --journal, the limit is lower
Fri Feb 17 23:03:29 [initandlisten]
Fri Feb 17 23:03:29 [initandlisten] db version v2.0.2, pdfile version 4.5
Fri Feb 17 23:03:29 [initandlisten] git version: 514b122d308928517f5841888ceaa4246a7
Fri Feb 17 23:03:29 [initandlisten] build info: windows (5, 1, 2600, 2, 'Service Pac
Fri Feb 17 23:03:29 [initandlisten] options: { dbpath: "E:\mongodb\db" }
Fri Feb 17 23:03:29 [websvr] admin web console waiting for connections on port 28017
Fri Feb 17 23:03:29 [initandlisten] waiting for connections on port 27017
Fri Feb 17 23:04:29 [clientcursormon] mem (MB) res:22 virt:81 mapped:0
```

③: 最后要看下是否开启成功, 从图中的信息中获知, mongodb 采用 27017 端口, 那么我们就在浏览器里面键入“<http://localhost:27017/>”,

打开后, mongodb 告诉我们在 27017 上 Add 1000 可以用 http 模式查看 mongodb 的管理信息。



You are trying to access MongoDB on the native driver port. For http diagnostic acco



三：基本操作

由于是开篇，就大概的说下基本的“增删查改”，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，默认连接的是“test”数据库。

```
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
> _
```

<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。

```
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。

```
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
>
```


第二天 细说增删查改

看过上一篇，相信大家都会知道如何开启 mongodb 了，这篇就细说下其中的增删查改，首先当我们用上篇同样的方式打开 mongodb，突然

傻眼了，擦，竟然开启不了，仔细观察“划线区域”的信息，发现 db 文件夹下有一个类似的“lock file”阻止了 mongodb 的开启，接下来我们要做的就

是干掉它，之后，开启成功，关于 mongodb 的管理方式将在后续文章分享。

```
管理员: C:\Windows\system32\cmd.exe
C:\>E:

E:\>cd mongodb\bin

E:\mongodb\bin>mongod --dbpath=E:\Mongodb\db
Sat Feb 18 22:06:00
Sat Feb 18 22:06:00 warning: 32-bit servers don't have journaling enabled by default
Sat Feb 18 22:06:00
Sat Feb 18 22:06:00 [initandlisten] MongoDB starting : pid=2872 port=27017 dbpath=E:
Sat Feb 18 22:06:00 [initandlisten]
Sat Feb 18 22:06:00 [initandlisten] ** NOTE: when using MongoDB 32 bit, you are limi
Sat Feb 18 22:06:00 [initandlisten] **      see http://blog.mongodb.org/post/137788
Sat Feb 18 22:06:00 [initandlisten] **      with --journal, the limit is lower
Sat Feb 18 22:06:00 [initandlisten]
Sat Feb 18 22:06:00 [initandlisten] db version v2.0.2, pdfile version 4.5
Sat Feb 18 22:06:00 [initandlisten] git version: 514b122d308928517f5841888ceaa4246a7
Sat Feb 18 22:06:00 [initandlisten] build info: windows (5, 1, 2600, 2, 'Service Pac
Sat Feb 18 22:06:00 [initandlisten] options: { dbpath: "E:\Mongodb\db" }
*****
Unclean shutdown detected.
Please visit http://dochub.mongodb.org/core/repair for recovery instructions.
*****
Sat Feb 18 22:06:00 [initandlisten] exception in initAndListen: 12596 old lock file,
Sat Feb 18 22:06:00 dbexit:
Sat Feb 18 22:06:00 [initandlisten] shutdown: going to close listening sockets...
Sat Feb 18 22:06:00 [initandlisten] shutdown: going to flush diaglog...
Sat Feb 18 22:06:00 [initandlisten] shutdown: going to close sockets...
Sat Feb 18 22:06:00 [initandlisten] shutdown: waiting for fs preallocator...
Sat Feb 18 22:06:00 [initandlisten] shutdown: closing all files...
Sat Feb 18 22:06:00 [initandlisten] closeAllFiles() finished
```

一： Insert 操作

上一篇也说过,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":{$gt:22}>>)
{ "_id" : ObjectId("4f3fc5b333c814a22045929a"), "name" : "joe", "password" : "12345", "age" : 25, "address" : "12345" }
>
> /* find age<22 */
... db.user.find(<<"age":{$lt:22}>>)
{ "_id" : ObjectId("4f3fc51633c814a220459299"), "name" : "jack", "password" : "12345", "age" : 20, "address" : "12345" }
>
> /* find age!=22 */
... db.user.find(<<"age":{$ne:22}>>)
{ "_id" : ObjectId("4f3fc51633c814a220459299"), "name" : "jack", "password" : "12345", "age" : 20, "address" : "12345" }
{ "_id" : ObjectId("4f3fc5b333c814a22045929a"), "name" : "joe", "password" : "12345", "age" : 25, "address" : "12345" }
>
> /* find age==20 */
... db.user.find(<<"age":20>>)
{ "_id" : ObjectId("4f3fc51633c814a220459299"), "name" : "jack", "password" : "12345", "age" : 20, "address" : "12345" }
>
=
```

<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" : "12345" }
>
> /* find province='anhui' || province='guangdong' */
... db.user.find(<<$or:[{"address.province":"anhui"}, {"address.province":"guangdong"}]>>)
{ "_id" : ObjectId("4f3fc51633c814a220459299"), "name" : "jack", "password" : "12345", "age" : 20, "address" : "12345" }
>
> /* find province='anhui' || province='guangdong' */
...
... /* */
... /* find province in ["anhui","guangdong"] */
... db.user.find(<<"address.province":{$in:["anhui","guangdong"]}>>)
{ "_id" : ObjectId("4f3fc51633c814a220459299"), "name" : "jack", "password" : "12345", "age" : 20, "address" : "12345" }
>
> /* find province not in ["anhui","guangdong"] */
... db.user.find(<<"address.province":{$nin:["anhui","guangdong"]}>>)
{ "_id" : ObjectId("4f3fc5b333c814a22045929a"), "name" : "joe", "password" : "12345", "age" : 25, "address" : "12345" }
>
```

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

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

<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。

四: Remove 操作

这个操作在上一篇简单的说过，这里就不赘述了。

第三天 细说高级操作

今天跟大家分享一下 mongodb 中比较好玩的知识，主要包括：聚合，游标。

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`，`value` 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：

```
管理员: C:\Windows\system32\cmd.exe - mongo
>
> 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“collection”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开销。

```

> 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 }
>

```

第四天 索引操作

这两天项目改版，时间比较紧，博客也就没跟得上，还望大家见谅。

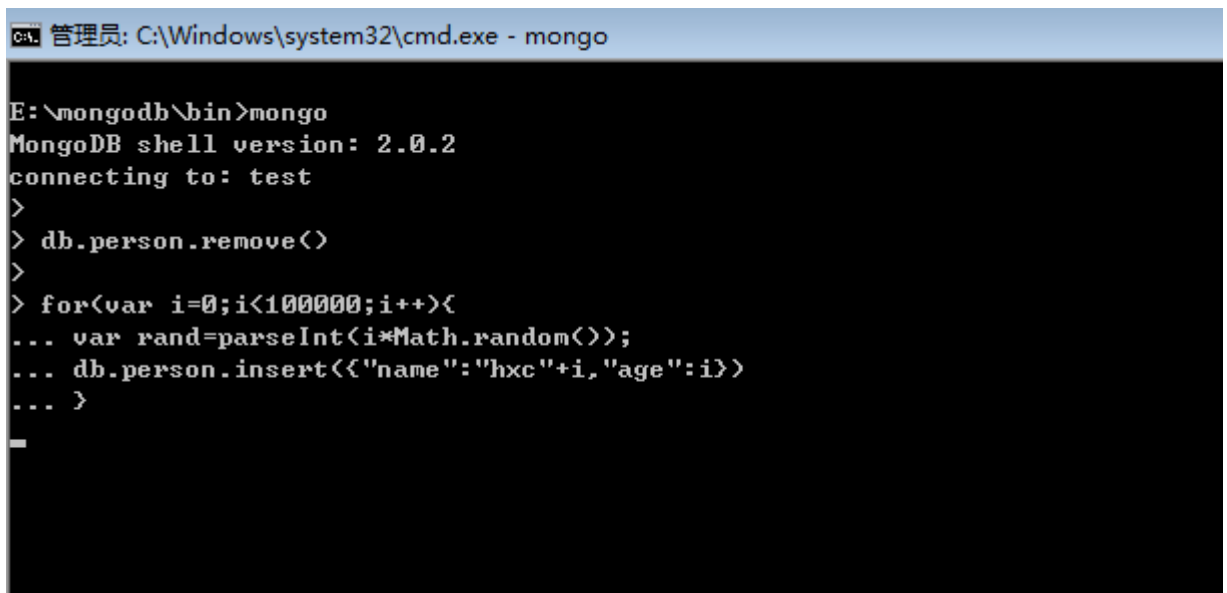
好，今天分享下 **mongodb** 中关于索引的基本操作，我们日常做开发都避免不了要对程序进行性能优化，而程序的操作无非就是 **CURD**，通常我们

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，处理不好就会被别人唾弃，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_"
  }
]
>
```

第五天 主从复制

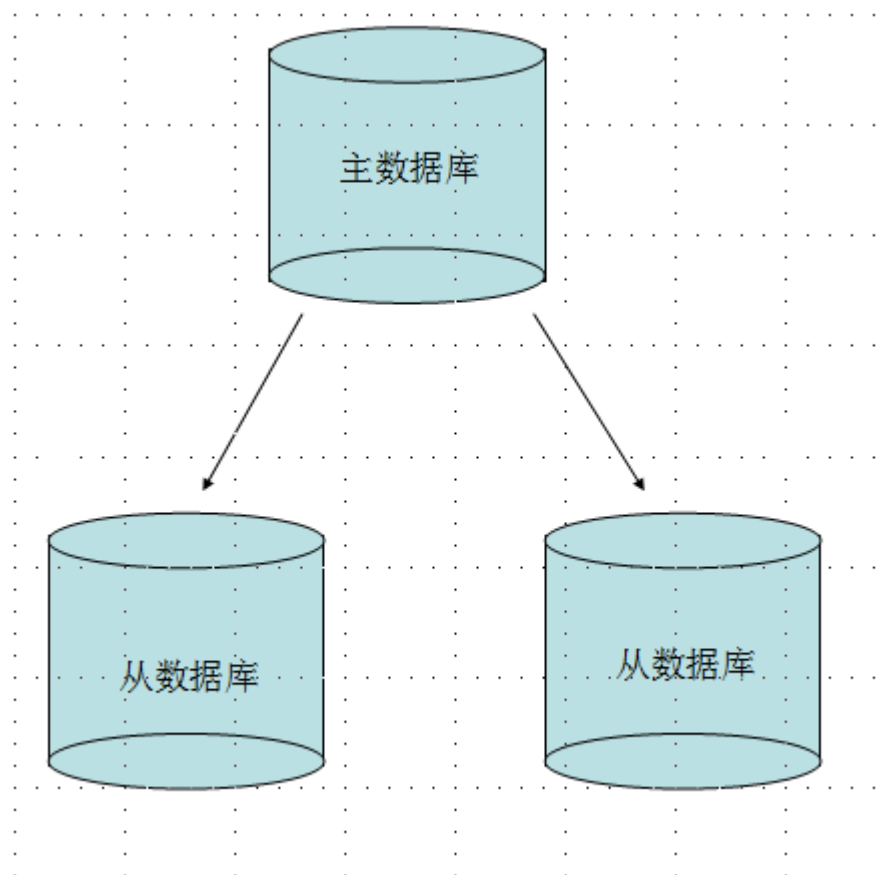
从这一篇开始我们主要讨论 **mongodb** 的部署技术。

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。

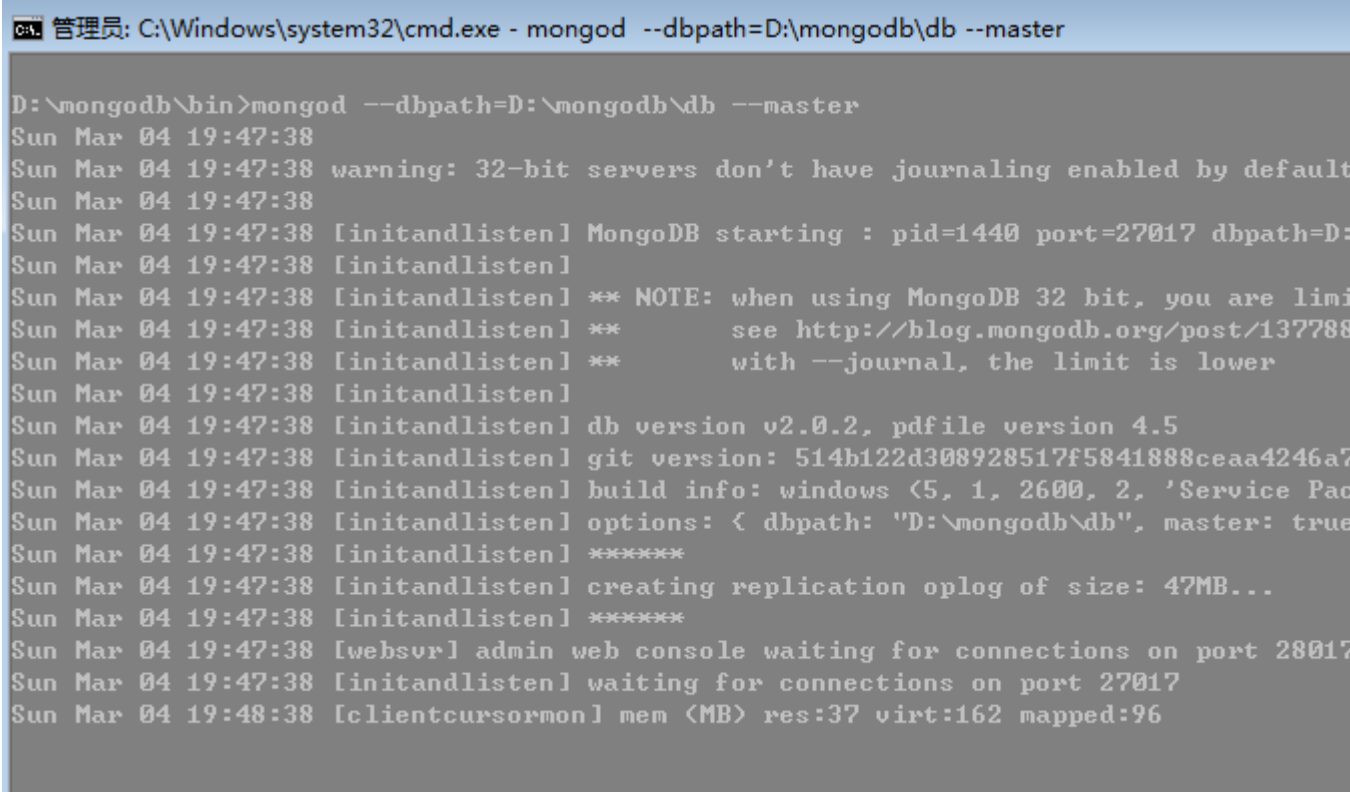
3：下面我们就一一实践

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。



```
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
Sun Mar 04 19:47:38
Sun Mar 04 19:47:38 [initandlisten] MongoDB starting : pid=1440 port=27017 dbpath=D:
Sun Mar 04 19:47:38 [initandlisten]
Sun Mar 04 19:47:38 [initandlisten] ** NOTE: when using MongoDB 32 bit, you are limi
Sun Mar 04 19:47:38 [initandlisten] ** see http://blog.mongodb.org/post/137788
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: 514b122d308928517f5841888ceaa4246a7
Sun Mar 04 19:47:38 [initandlisten] build info: windows (5, 1, 2600, 2, 'Service Pac
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
Sun Mar 04 19:56:53
Sun Mar 04 19:56:53 [initandlisten] MongoDB starting : pid=3348 port=8888 dbpath=E:\mongodb\db
Sun Mar 04 19:56:53 [initandlisten]
Sun Mar 04 19:56:53 [initandlisten] ** NOTE: when using MongoDB 32 bit, you are limited to 16MB of data
Sun Mar 04 19:56:53 [initandlisten] ** see http://blog.mongodb.org/post/137788
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: 514b122d308928517f5841888ceaa4246a7
Sun Mar 04 19:56:53 [initandlisten] build info: windows (5, 1, 2600, 2, 'Service Pack 3')
Sun Mar 04 19:56:53 [initandlisten] options: { dbpath: "E:\mongodb\db", port: 8888,
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 9888
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
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
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
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 mongod 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 mongod 的运行程序,cmd 窗口好多啊,大家不要搞乱了。

```

F:\mongodb\bin>mongo --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
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=UON
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 [websvr] 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("4f5368868f59fab66264355"), "host" : "127.0.0.1:27017" } 同步前
> db.sources.find()
{ "_id" : ObjectId("4f5368868f59fab66264355"), "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("4f5362f4d3bb14ea592bc7d"), "name" : "hxc", "age" : 23 }
{ "_id" : ObjectId("4f5362fbd3bb14ea592bc7e"), "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。

```
C:\Windows\system32\cmd.exe - mongod --dbpath=D:\mongodb\db --port 2222 --replSet shopex/127.0.0.1
D:\mongodb\bin>mongod --dbpath=D:\mongodb\db --port 2222 --replSet shopex/127.0.0.1
Mon Mar 05 21:08:43
Mon Mar 05 21:08:43 warning: 32-bit servers don't have journaling enabled by default
Mon Mar 05 21:08:43
Mon Mar 05 21:08:43 [initandlisten] MongoDB starting : pid=5144 port=2222 dbpath=D:
Mon Mar 05 21:08:43 [initandlisten]
Mon Mar 05 21:08:43 [initandlisten] ** NOTE: when using MongoDB 32 bit, you are limi
Mon Mar 05 21:08:43 [initandlisten] **      see http://blog.mongodb.org/post/137788
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: 514b122d308928517f5841888ceaa4246a7
Mon Mar 05 21:08:43 [initandlisten] build info: windows (5, 1, 2600, 2, 'Service Pack 3')
Mon Mar 05 21:08:43 [initandlisten] options: { dbpath: "D:\mongodb\db", port: 2222, replSet: "shopex/127.0.0.1" }
```

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 mongodb 程序打开。

```

C:\Windows\system32\cmd.exe - mongod --dbpath=E:\mongodb\db --port 3333 --replSet shopex/127.0.0.1
E:\mongodb\bin>mongod --dbpath=E:\mongodb\db --port 3333 --replSet shopex/127.0.0.1
Mon Mar 05 21:13:16
Mon Mar 05 21:13:16 warning: 32-bit servers don't have journaling enabled by default
Mon Mar 05 21:13:16
Mon Mar 05 21:13:16 [initandlisten] MongoDB starting : pid=4644 port=3333 dbpath=E:
Mon Mar 05 21:13:16 [initandlisten]
Mon Mar 05 21:13:16 [initandlisten] ** NOTE: when using MongoDB 32 bit, you are limi
Mon Mar 05 21:13:16 [initandlisten] **      see http://blog.mongodb.org/post/137788
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: 514b122d308928517f5841888ceaa4246a7
Mon Mar 05 21:13:16 [initandlisten] build info: windows (5, 1, 2600, 2, 'Service Pack 3')
Mon Mar 05 21:13:16 [initandlisten] options: { dbpath: "E:\mongodb\db", port: 3333,
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 sel
Mon Mar 05 21:13:17 [rsStart] replSet info you may need to run replSetInitiate -- rs
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/127.0.0.1:2222
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 g
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 ST
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 RE
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/127.0.0.1:3333
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
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 集群中的任何一个服务器端口，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
E:\mongodb\bin>mongod --dbpath=F:\mongodb\db --port 4444 --replSet shopex/127.0.0.1
Mon Mar 05 21:26:42
Mon Mar 05 21:26:42 warning: 32-bit servers don't have journaling enabled by default
Mon Mar 05 21:26:42
Mon Mar 05 21:26:42 [initandlisten] MongoDB starting : pid=4216 port=4444 dbpath=F:
Mon Mar 05 21:26:42 [initandlisten]
Mon Mar 05 21:26:42 [initandlisten] ** NOTE: when using MongoDB 32 bit, you are limi
Mon Mar 05 21:26:42 [initandlisten] **      see http://blog.mongodb.org/post/137788
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, pdf file version 4.5
Mon Mar 05 21:26:42 [initandlisten] git version: 514b122d308928517f5841888ceaa4246a
Mon Mar 05 21:26:42 [initandlisten] build info: windows (5, 1, 2600, 2, 'Service Pack 3')
Mon Mar 05 21:26:42 [initandlisten] options: { dbpath: "F:\mongodb\db", port: 4444,
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 configu
Mon Mar 05 21:26:42 [rsStart] { "_id": "shopex", version: 1, members: [ { "_id": 1, host
Mon Mar 05 21:26:42 [rsStart] replSet info Couldn't load config yet. Sleeping 20sec
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 configu
Mon Mar 05 21:27:02 [rsStart] { "_id": "shopex", version: 1, members: [ { "_id": 1, host
Mon Mar 05 21:27:02 [rsStart] replSet info Couldn't load config yet. Sleeping 20sec

```

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。

```
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 [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 [slaveTrack]
Mon Mar 05 21:32:50 [ctrlCTerm]
Mon Mar 05 21:32:50 [ctrlCTerm]
Mon Mar 05 21:32:50 dbexit: ready
```

D:\mongodb\bin>

管理员: C:\Windows\system32\cmd.exe - mongod --dbpath=E:\mon

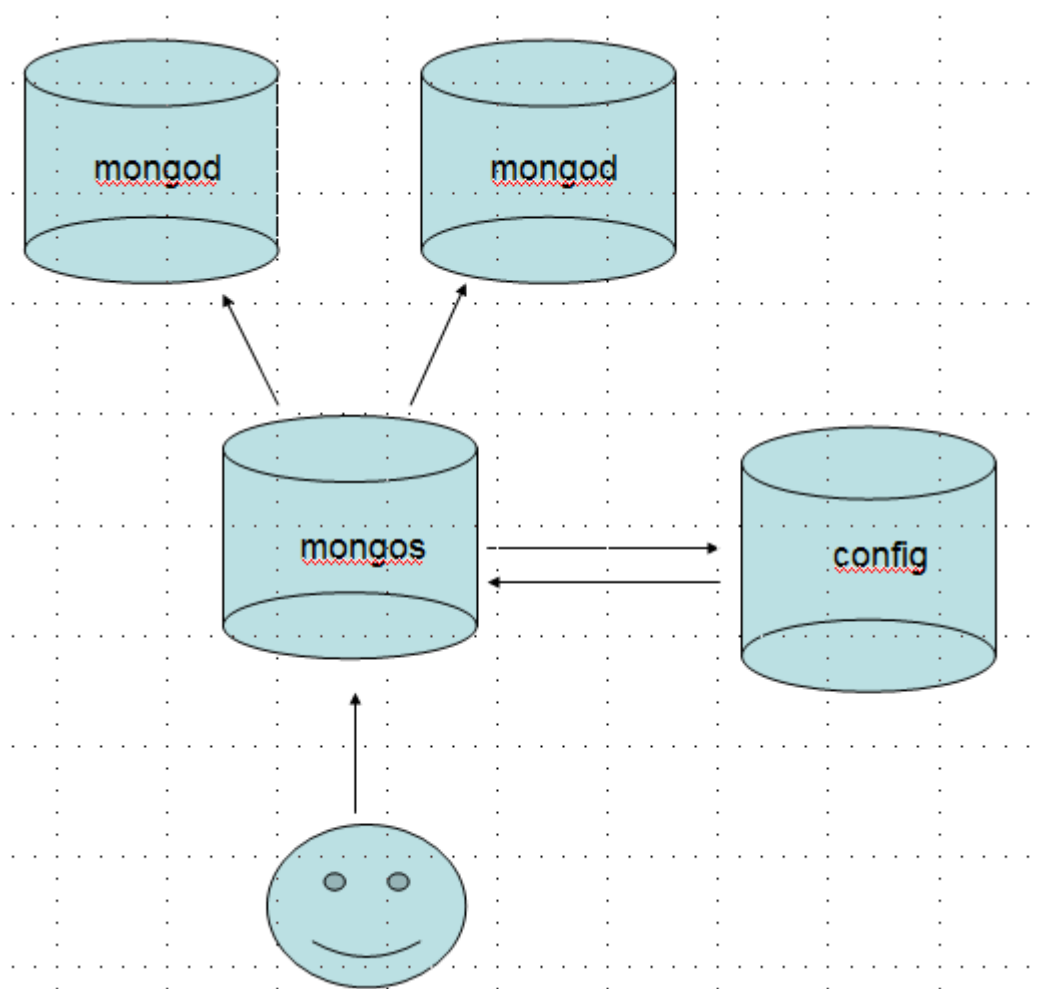
```
Mon Mar 05 21:30:41 [initandlisten] connection accepted
Mon Mar 05 21:30:43 [conn31] end connection 127.0.0.1
Mon Mar 05 21:30:43 [initandlisten] connection accepted
Mon Mar 05 21:31:11 [conn32] end connection 127.0.0.1
Mon Mar 05 21:31:11 [initandlisten] connection accepted
Mon Mar 05 21:31:13 [conn33] end connection 127.0.0.1
Mon Mar 05 21:31:13 [initandlisten] connection accepted
Mon Mar 05 21:31:41 [conn34] end connection 127.0.0.1
Mon Mar 05 21:31:41 [initandlisten] connection accepted
Mon Mar 05 21:31:43 [conn35] end connection 127.0.0.1
Mon Mar 05 21:31:43 [initandlisten] connection accepted
Mon Mar 05 21:32:11 [conn36] end connection 127.0.0.1
Mon Mar 05 21:32:11 [initandlisten] connection accepted
Mon Mar 05 21:32:13 [conn37] end connection 127.0.0.1
Mon Mar 05 21:32:13 [initandlisten] connection accepted
Mon Mar 05 21:32:41 [conn38] end connection 127.0.0.1
Mon Mar 05 21:32:41 [initandlisten] connection accepted
Mon Mar 05 21:32:43 [conn39] end connection 127.0.0.1
Mon Mar 05 21:32:43 [initandlisten] connection accepted
Mon Mar 05 21:32:49 [rsSync] replSet syncThread: 102
Mon Mar 05 21:32:49 [conn40] end connection 127.0.0.1
Mon Mar 05 21:32:49 [rsHealthPoll] replSet info 127.
Mon Mar 05 21:32:49 [rsHealthPoll] replSet member 12
Mon Mar 05 21:32:49 [rsMgr] not electing self, 127.0
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
Mon Mar 05 21:33:13 [initandlisten] connection accepted
Mon Mar 05 21:33:17 [clientcursormon] mem <MB> res:3
```


第六天 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。

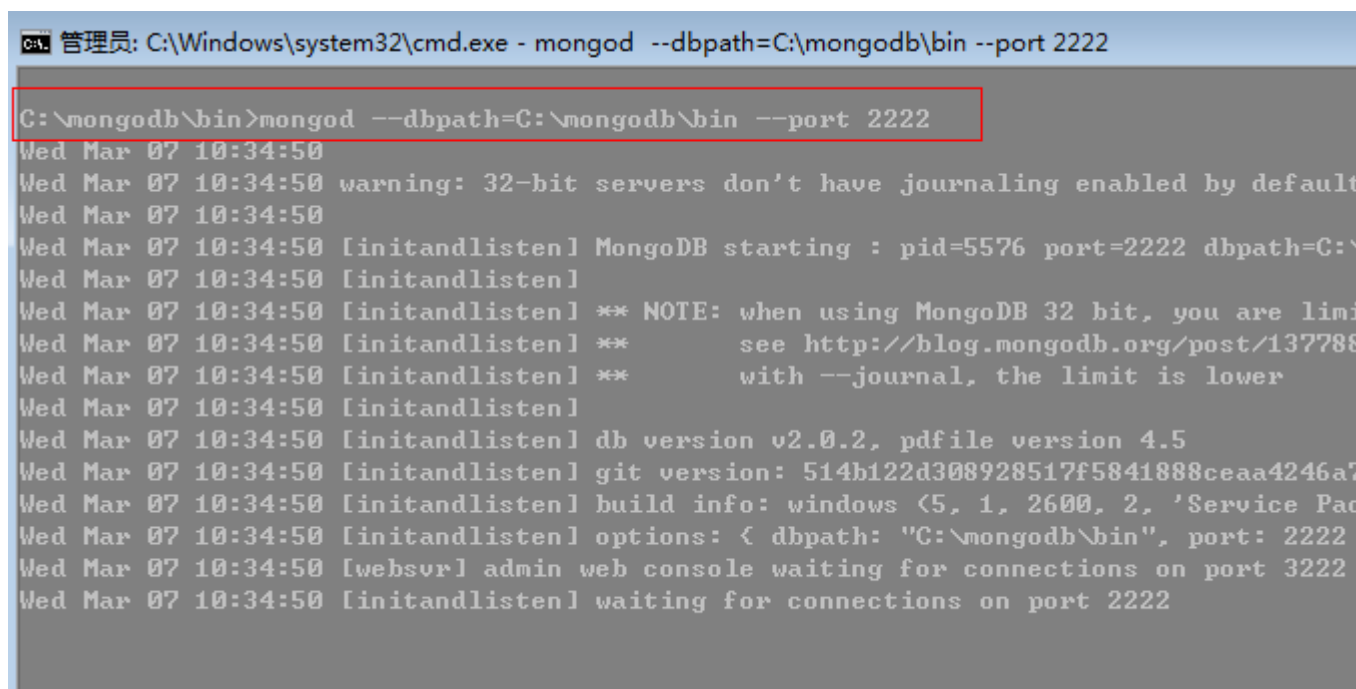
mongod: 一个普通的数据库实例，如果不分片的话，我们会直接连上 **mongod**。

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。



```
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
Wed Mar 07 10:34:50 warning: 32-bit servers don't have journaling enabled by default
Wed Mar 07 10:34:50
Wed Mar 07 10:34:50 [initandlisten] MongoDB starting : pid=5576 port=2222 dbpath=C:\
Wed Mar 07 10:34:50 [initandlisten]
Wed Mar 07 10:34:50 [initandlisten] ** NOTE: when using MongoDB 32 bit, you are limi
Wed Mar 07 10:34:50 [initandlisten] **      see http://blog.mongodb.org/post/137788
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: 514b122d308928517f5841888ceaa4246a7
Wed Mar 07 10:34:50 [initandlisten] build info: windows (5, 1, 2600, 2, 'Service Pack 3')
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 **mongodb**，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 02.0.2, pdfile version 4.5 starting <--help f
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_LI
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: VONXCEVF0IT7JDJ:3333 started at Mar 07 1
Wed Mar 07 10:40:54 [Balancer] created new distributed lock for balancer on 127.0.0.1:2222
Wed Mar 07 10:40:54 [Balancer] creating WriteBackListener for: 127.0.0.1:2222 server
```

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 mongodb，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
Wed Mar 07 10:47:06
Wed Mar 07 10:47:06 [initandlisten] MongoDB starting : pid=2140 port=4444 dbpath=E:
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
Wed Mar 07 10:48:10
Wed Mar 07 10:48:10 [initandlisten] MongoDB starting : pid=5072 port=5555 dbpath=F:
Wed Mar 07 10:48:10 [initandlisten]
Wed Mar 07 10:48:10 [initandlisten] ** NOTE: when using MongoDB 32 bit, you are limi
Wed Mar 07 10:48:10 [initandlisten] ** see http://blog.mongodb.org/post/137788
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()。

```
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。

```
管理员: 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 : sh
      { "name" : "jack0" } --> { "name" : "jack234813" } on : sh
      { "name" : "jack234813" } --> { "name" : "jack9999" } on : sh
      { "name" : "jack9999" } --> { "name" : { $maxKey : 1 } } on : sh
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。

第七天 运维技术

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，

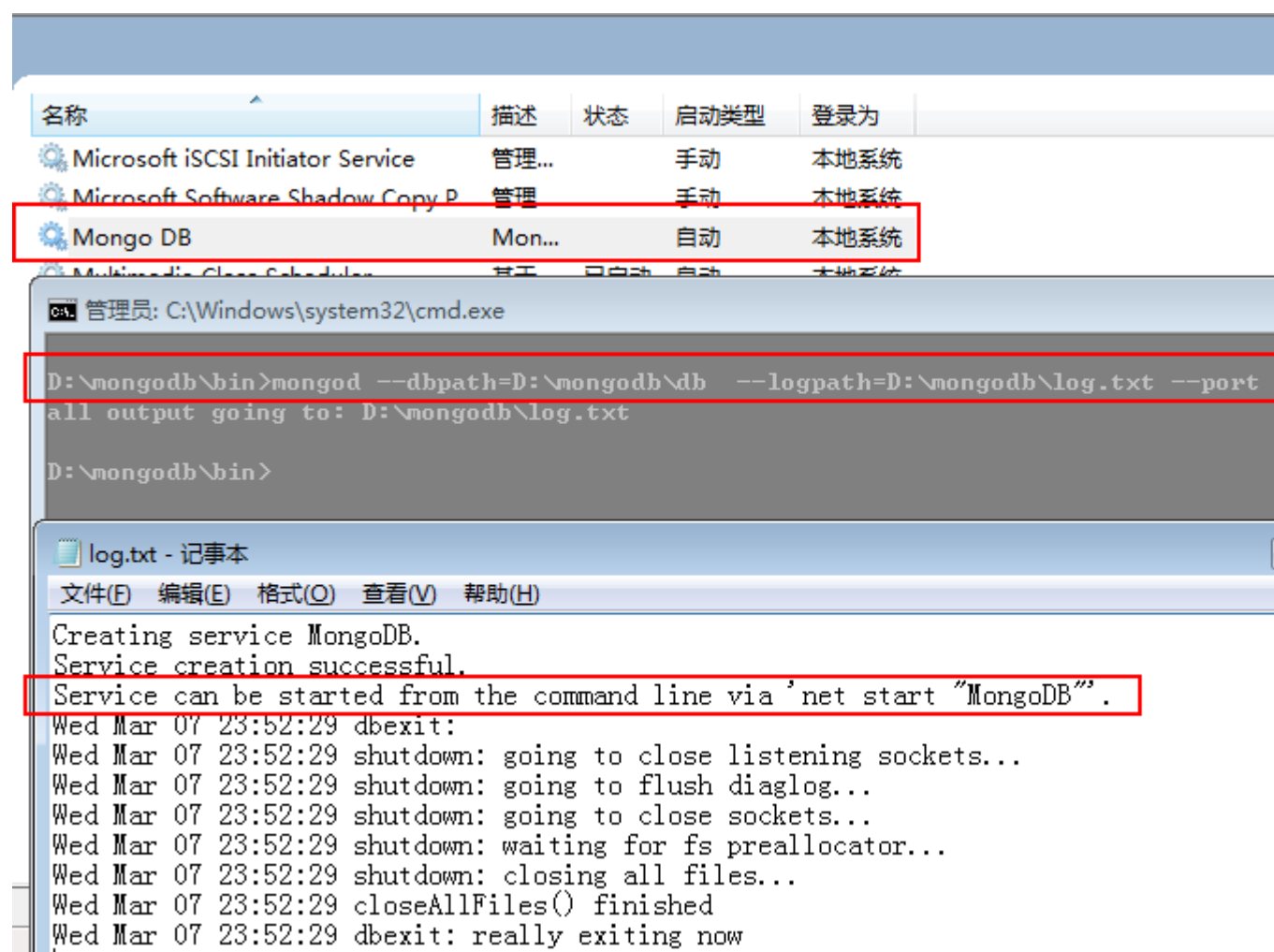
下面我们就一点一点的讲解。

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`:

管理员: C:\Windows\system32\cmd.exe

D:\mongodb\bin>mongod --help

Thu Mar 08 00:00:18

Thu Mar 08 00:00:18 *** NOTE: when using MongoDB 32 bit, you are limited to about 2 g

Thu Mar 08 00:00:18 ***

see <http://blog.mongodb.org/post/137788967/32-bit-limit>

Thu Mar 08 00:00:18 ***

with --journal, the limit is lower

Thu Mar 08 00:00:18

Allowed options:

General options:

-h [--help]

show this usage information

--version

show version information

-f [--config] arg

configuration file specifying additional options

-v [--verbose]

be more verbose (include multiple times for more verbosity e.g. -vvvvv)

--quiet

quieter output

--port arg

specify port number

--bind_ip arg

comma separated list of ip addresses to listen on
- all local ips by default

--maxConns arg

max number of simultaneous connections

--objcheck

inspect client data for validity on receipt

--logpath arg

log file to send write to instead of stdout - has
to be a file, not directory

--logappend

append to logpath instead of over-writing

--pidfilepath arg

full path to pidfile (if not set, no pidfile is
created)

--keyFile arg

private key for cluster authentication (only for
replica sets)

--auth

run with security

--cpu

periodically show cpu and iowait utilization

--dbpath arg

directory for datafiles

--diaglog arg

0=off 1=W 2=R 3=both 7=W+some reads

--directoryperdb

each database will be stored in a separate
directory

--journal

enable journaling

--journalOptions arg

journal diagnostic options

--journalCommitInterval arg

how often to group/batch commit (ms)

mongo:

```
D:\mongodb\bin>mongo --help
MongoDB shell version: 2.0.2
usage: mongo [options] [db address] [file names (ending in .js)]
db address can be:
    foo                foo database on local machine
    192.169.0.5/foo     foo database on 192.168.0.5 machine
    192.169.0.5:9999/foo foo database on 192.168.0.5 machine on port 9999
options:
    --shell            run the shell after executing files
    --nodb             don't connect to mongod on startup - no 'db address'
                      arg expected
    --norc             will not run the ".mongorc.js" file on start up
    --quiet            be less chatty
    --port arg         port to connect to
    --host arg         server to connect to
    --eval arg         evaluate javascript
    -u [ --username ] arg username for authentication
    -p [ --password ] arg password for authentication
    -h [ --help ]     show this usage information
    --version          show version information
    --verbose          increase verbosity
    --ipv6             enable IPv6 support (disabled by default)

file names: a list of files to run. files have to end in .js and will exit after un
```

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。

```
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.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,
    "available" : 100,
    "total" : 101
  }
}
```

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 **mongodstat**

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。

```
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  locl
      0      0      0      0      0      1      0      0m   53m   21m    0
      0      0      0      0      0      1      0      0m   53m   21m    0
      0      0      0      0      0      1      0      0m   53m   21m   12
      0      0      0      0      0      1      0      0m   53m   21m    4
      0      0      0      0      0      1      0      0m   53m   21m    0
      0      0      0      0      0      1      0      0m   53m   21m    0
      0      0      0      0      0      1      0      0m   53m   21m    0
      0      0      0      0      0      1      0      0m   53m   21m    0
      0      0      0      0      0      1      0      0m   53m   21m    0
insert  query  update  delete  getmore  command  flushes  mapped  vsize  res  faults  locl
      0      0      0      0      0      1      0      0m   53m   21m    0
      0      0      0      0      0      1      0      0m   53m   21m    0
      0      0      0      0      0      1      0      0m   53m   21m    0
      0      0      0      0      0      1      0      0m   53m   21m    0
      0      0      0      0      0      1      0      0m   53m   21m    0
      0      0      0      0      0      1      0      0m   53m   21m    0
      0      0      0      0      0      1      0      0m   53m   21m    0
      0      0      0      0      0      1      0      0m   53m   21m    0
      0      0      0      0      0      1      0      0m   53m   21m    0
insert  query  update  delete  getmore  command  flushes  mapped  vsize  res  faults  locl
      0      0      0      0      0      1      0      0m   53m   21m    0
      0      0      0      0      0      1      0      0m   53m   21m    0
      0      0      0      0      0      1      0      0m   53m   21m    0
      0      0      0      0      0      1      1      0m   53m   21m    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
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: {
  "$err" : "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 }
> db.system.users.remove()
> 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()
> 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()`

第八天 驱动实践

作为系列的最后一篇，得要说说 C# 驱动对 mongodb 的操作，目前驱动有两种：官方驱动和 **samus** 驱动，不过我个人还是喜欢后者，

因为提供了丰富的 **linq** 操作，相当方便。

官方驱动: <https://github.com/mongodb/mongo-csharp-driver/downloads>。下载后，还提供了一个酷似 msdn 的帮助文档。

samus 驱动: <https://github.com/samus/mongodb-csharp/downloads>。

下面就具体看看 **samus** 驱动, <https://github.com/samus/mongodb-csharp/blob/master/examples/Simple/Main.cs> 上面提供了一个简单的 **demo**，大体上看看我们就知道怎么玩了。

一： 实践

1: 我们建立一个 **Person** 实体，**MongoAlias** 特性表示取别名，这里的 **ID** 值将会覆盖掉数据库自动生成的 **_id**。

```
1 #region 数据实体
2     /// <summary>
3     /// 数据实体
4     /// </summary>
5     public class Person
6     {
7         [MongoAlias("_id")]
8         public string ID { get; set; }
9
10        public string Name { get; set; }
11
12        public int Age { get; set; }
13
14        public DateTime CreateTime { get; set; }
15    }
16 #endregion
```

2: 初始化一些变量

```

1      string connectionString = string.Empty;
2
3      string databaseName = string.Empty;
4
5      string collectionName = string.Empty;
6
7      static MongoDBHelper<T> mongodb;
8
9      #region 初始化操作
10     /// <summary>
11     /// 初始化操作
12     /// </summary>
13     public MongoDBHelper()
14     {
15         connectionString = "Server=127.0.0.1:2222";
16         databaseName = "shopex";
17         collectionName = "person";
18     }
19     #endregion

```

3: 为了方便 T 的继承类使用 linq 功能，我们还需要映射一下。

```

1 #region 实现 linq 查询的映射配置
2     /// <summary>
3     /// 实现 linq 查询的映射配置
4     /// </summary>
5     public MongoConfiguration configuration
6     {
7         get
8         {
9             var config = new MongoConfigurationBuilder();
10
11             config.Mapping(mapping =>
12             {
13                 mapping.DefaultProfile(profile =>
14                 {
15                     profile.SubClassesAre(t =>
16                     t.IsSubclassOf(typeof(T)));
17                 });
18                 mapping.Map<T>();
19                 mapping.Map<T>();
20             });

```

```

20
21         config.ConnectionString(connectionString);
22
23         return config.BuildConfiguration();
24     }
25 }
26 #endregion

```

4: 下面是一些基本的 CURD 的代码，跟写 EF 代码很类似，写起来好舒服。

```

1  #region 插入操作
2      /// <summary>
3  /// 插入操作
4  /// </summary>
5  /// <param name="person"></param>
6  /// <returns></returns>
7      public void Insert(T t)
8      {
9          using (Mongo mongo = new Mongo(configuration))
10         {
11             try
12             {
13                 mongo.Connect();
14
15                 var db = mongo.GetDatabase(databaseName);
16
17                 var collection =
db.GetCollection<T>(collectionName);
18
19                 collection.Insert(t, true);
20
21                 mongo.Disconnect();
22
23             }
24             catch (Exception)
25             {
26                 mongo.Disconnect();
27                 throw;
28             }
29         }
30     }
31 #endregion

```

```

32
33         #region 更新操作
34         /// <summary>
35         /// 更新操作
36         /// </summary>
37         /// <param name="person"></param>
38         /// <returns></returns>
39         public void Update(T t, Expression<Func<T, bool>> func)
40         {
41             using (Mongo mongo = new Mongo(configuration))
42             {
43                 try
44                 {
45                     mongo.Connect();
46
47                     var db = mongo.GetDatabase(databaseName);
48
49                     var collection =
db.GetCollection<T>(collectionName);
50
51                     collection.Update<T>(t, func, true);
52
53                     mongo.Disconnect();
54
55                 }
56                 catch (Exception)
57                 {
58                     mongo.Disconnect();
59                     throw;
60                 }
61             }
62         }
63         #endregion
64
65         #region 获取集合
66         /// <summary>
67         /// 获取集合
68         /// </summary>
69         /// <param name="person"></param>
70         /// <returns></returns>
71         public List<T> List(int pageIndex, int pageSize,
Expression<Func<T, bool>> func, out int pageCount)
72         {
73             pageCount = 0;

```

```

74
75         using (Mongo mongo = new Mongo(configuration))
76         {
77             try
78             {
79                 mongo.Connect();
80
81                 var db = mongo.GetDatabase(databaseName);
82
83                 var collection =
db.GetCollection<T>(collectionName);
84
85                 pageCount =
Convert.ToInt32(collection.Count());
86
87                 var personList =
collection.Linq().Where(func).Skip(pageSize * (pageIndex - 1))
88
89                                     .Take(pageSize)
90
91 .Select(i => i).ToList();
92
93         mongo.Disconnect();
94
95         return personList;
96     }
97     catch (Exception)
98     {
99         mongo.Disconnect();
100        throw;
101    }
102 }
103 #endregion
104 #region 读取单条记录
105     /// <summary>
106     /// 读取单条记录
107     /// </summary>
108     /// <param name="person"></param>
109     /// <returns></returns>
110     public T Single(Expression<Func<T, bool>> func)
111     {
112         using (Mongo mongo = new Mongo(configuration))

```

```

113         {
114             try
115             {
116                 mongo.Connect();
117
118                 var db = mongo.GetDatabase(databaseName);
119
120                 var collection =
db.GetCollection<T>(collectionName);
121
122                 var single =
collection.Linq().FirstOrDefault(func);
123
124                 mongo.Disconnect();
125
126                 return single;
127
128             }
129             catch (Exception)
130             {
131                 mongo.Disconnect();
132                 throw;
133             }
134         }
135     }
136     #endregion
137
138     #region 删除操作
139     /// <summary>
140     /// 删除操作
141     /// </summary>
142     /// <param name="person"></param>
143     /// <returns></returns>
144     public void Delete(Expression<Func<T, bool>> func)
145     {
146         using (Mongo mongo = new Mongo(configuration))
147         {
148             try
149             {
150                 mongo.Connect();
151
152                 var db = mongo.GetDatabase(databaseName);
153
154                 var collection =

```

```

db.GetCollection<T>(collectionName);
155
156             //这个地方要注意，一定要加上 T 参数，否则会当
作 object 类型处理
157 //导致删除失败
158             collection.Remove<T>(func);
159
160             mongo.Disconnect();
161
162         }
163         catch (Exception)
164         {
165             mongo.Disconnect();
166             throw;
167         }
168     }
169 }
170 #endregion

```

5. 好，我们开一下 2222 端口，由于前前篇我已经把这个 mongodb 做成了服务，现在就直接连过去了，并做一下对 Name 的索引。

```

D:\mongodb\bin>mongo 127.0.0.1:2222/shopex
MongoDB shell version: 2.0.2
connecting to: 127.0.0.1:2222/shopex
> db.person.ensureIndex<<"Name":1>>
>

```

6. 一切准备妥当，我们做下基本的操作，比如这里我添加一千条数据，注意我开启的是安全模式，如果插入不成功，将会抛出异常。

<1> Add:

```

1 static void Main(string[] args)
2     {
3         MongoDBHelper<Person> helper = new
MongoDBHelper<Person>();
4
5         //插入 1000 条数据
6         for (int i = 0; i < 1000; i++)
7         {
8             helper.Insert(new Person()

```

```

9          {
10              ID = Guid.NewGuid().ToString(),
11              Name = "jack" + i,
12              Age = i,
13              CreateTime = DateTime.Now
14          });
15      }
16
17      Console.WriteLine("插入成功");
18
19      Console.Read();
20  }

```

```

C:\Windows\system32\cmd.exe - mongo 127.0.0.1:2222/shopex
>
D:\mongodb\bin>mongo 127.0.0.1:2222/shopex
MongoDB shell version: 2.0.2
connecting to: 127.0.0.1:2222/shopex
> db.person.find()
{ "_id" : "d161ad87-c41c-47e1-87ca-f39903c10fd4", "Name" : "jack941", "Age" : 941,
{ "_id" : "8150dd8d-d49f-44d3-8f87-fc14388bf0fa", "Name" : "jack942", "Age" : 942,
{ "_id" : "a4efe750-2b77-44af-a6d9-42f965221cf5", "Name" : "jack943", "Age" : 943,
{ "_id" : "369fc26e-ec7f-4ffd-a54c-7d36fa3089b8", "Name" : "jack944", "Age" : 944,
{ "_id" : "bd8600b2-5a8c-4693-a954-d42a6ad1dba2", "Name" : "jack945", "Age" : 945,
{ "_id" : "879c209c-3637-41b4-b0a9-ddb7a2da7e54", "Name" : "jack946", "Age" : 946,
{ "_id" : "6ebd35df-fc74-43be-8d49-cf84192969de", "Name" : "jack947", "Age" : 947,
{ "_id" : "a30d1179-863e-42d4-be81-dc01a3ec0f51", "Name" : "jack948", "Age" : 948,
{ "_id" : "0b2ec845-65e6-4be8-99af-66172fb8323f", "Name" : "jack949", "Age" : 949,
{ "_id" : "15240fe1-1347-43b6-833e-10c73de96d60", "Name" : "jack950", "Age" : 950,
{ "_id" : "f4b63d96-164d-4682-98f9-b53bc0190d5c", "Name" : "jack951", "Age" : 951,
{ "_id" : "0d012c0f-09b7-44cf-8c3e-532d019bd6a7", "Name" : "jack952", "Age" : 952,
{ "_id" : "b4bf712e-6162-44cb-8bb9-fc8c7b9cb847", "Name" : "jack953", "Age" : 953,
{ "_id" : "116c571b-23ab-4b74-bc03-cac1386598a9", "Name" : "jack954", "Age" : 954,
{ "_id" : "e02be598-5fe6-47ec-9f14-89760b016eae", "Name" : "jack955", "Age" : 955,
{ "_id" : "6d2b068b-bfaa-416d-8609-e1dd7008834c", "Name" : "jack956", "Age" : 956,
{ "_id" : "c41b0519-c53c-4e98-89a7-261a69301752", "Name" : "jack957", "Age" : 957,
{ "_id" : "61201556-9fd5-480f-9b4a-2f2883a1be02", "Name" : "jack958", "Age" : 958,
{ "_id" : "ce089b06-2409-4e87-a4a5-48d68e67b78f", "Name" : "jack959", "Age" : 959,
{ "_id" : "46d00b4c-16ae-4efe-a47a-28c83faa5177", "Name" : "jack960", "Age" : 960,
has more
> db.person.count()
1000
> db.person.find(<"Age":999>)
{ "_id" : "9265b2e5-4673-496a-9357-f747446a8b53", "Name" : "jack999", "Age" : 999,
> db.person.find(<"Age":0>)
{ "_id" : "04843ccc-1f8c-42cb-8049-2d347db1092f", "Name" : "jack0", "Age" : 0, "Cre
>

```

乍一看显示的数据以为有问题，为什么没有出现 jack0 或者 jack999，不过 find 的一下后心情舒坦了。

<2> update: 这里就把 jack941 的名字改掉"mary"

```
1 static void Main(string[] args)
2     {
3         MongoDBHelper<Person> helper = new
MongoDBHelper<Person>();
4
5         //修改 jack941 改成 mary
6         var single = helper.Single(i => i.Name == "jack941");
7         single.Name = "mary";
8         helper.Update(single, i => i.ID == single.ID);
9
10        Console.WriteLine("修改成功");
11        Console.Read();
12    }
```

```
> db.person.find()
< { "_id" : "d161ad87-c41c-47e1-87ca-f39903c10fd4", "Name" : "mary", "Age" : 941, "Cre
< { "_id" : "8150dd8d-d49f-44d3-8f87-fc14388bf0fa", "Name" : "jack942", "Age" : 942, "
< { "_id" : "a4efe750-2b77-44af-a6d9-42f965221cf5", "Name" : "jack943", "Age" : 943, "
< { "_id" : "369fc26e-ec7f-4ffd-a54c-7d36fa3089b8", "Name" : "jack944", "Age" : 944, "
< { "_id" : "bd8600b2-5a8c-4693-a954-d42a6ad1dba2", "Name" : "jack945", "Age" : 945, "
< { "_id" : "879c209c-3637-41b4-b0a9-ddb7a2da7e54", "Name" : "jack946", "Age" : 946, "
< { "_id" : "6ebd35df-fc74-43be-8d49-cf84192969de", "Name" : "jack947", "Age" : 947, "
< { "_id" : "a30d1179-863e-42d4-be81-dc01a3ec0f51", "Name" : "jack948", "Age" : 948, "
< { "_id" : "0b2ec845-65e6-4be8-99af-66172fb8323f", "Name" : "jack949", "Age" : 949, "
< { "_id" : "15240fe1-1347-43b6-833e-10c73de96d60", "Name" : "jack950", "Age" : 950, "
< { "_id" : "f4b63d96-164d-4682-98f9-b53bc0190d5c", "Name" : "jack951", "Age" : 951, "
< { "_id" : "0d012c0f-09b7-44cf-8c3e-532d019bd6a7", "Name" : "jack952", "Age" : 952, "
< { "_id" : "b4bf712e-6162-44cb-8bb9-fc8c7b9cb847", "Name" : "jack953", "Age" : 953, "
< { "_id" : "116c571b-23ab-4b74-bc03-cac1386598a9", "Name" : "jack954", "Age" : 954, "
< { "_id" : "e02be598-5fe6-47ec-9f14-89760b016eae", "Name" : "jack955", "Age" : 955, "
< { "_id" : "6d2b068b-bfaa-416d-8609-e1dd7008834c", "Name" : "jack956", "Age" : 956, "
< { "_id" : "c41b0519-c53c-4e98-89a7-261a69301752", "Name" : "jack957", "Age" : 957, "
< { "_id" : "61201556-9fd5-480f-9b4a-2f2883a1be02", "Name" : "jack958", "Age" : 958, "
< { "_id" : "ce089b06-2409-4e87-a4a5-48d68e67b78f", "Name" : "jack959", "Age" : 959, "
< { "_id" : "46d00b4c-16ae-4efe-a47a-28c83faa5177", "Name" : "jack960", "Age" : 960, "
has more
>
```

<3>Delete: 删除 mary 这条记录

```
1 static void Main(string[] args)
2     {
3         MongoDBHelper<Person> helper = new
MongoDBHelper<Person>();
```

```

4
5      //删除 mary 这个记录
6      helper.Delete(i => i.Name == "mary");
7
8      Console.WriteLine("删除成功");
9      Console.Read();
10     }

```

```

> db.person.find<>
< "_id" : "8150dd8d-d49f-44d3-8f87-fc14388bf0fa", "Name" : "jack942", "Age" : 942, "
< "_id" : "a4efe750-2b77-44af-a6d9-42f965221cf5", "Name" : "jack943", "Age" : 943, "
< "_id" : "369fc26e-ec7f-4ffd-a54c-7d36fa3089b8", "Name" : "jack944", "Age" : 944, "
< "_id" : "bd8600b2-5a8c-4693-a954-d42a6ad1dba2", "Name" : "jack945", "Age" : 945, "
< "_id" : "879c209c-3637-41b4-b0a9-ddb7a2da7e54", "Name" : "jack946", "Age" : 946, "
< "_id" : "6ebd35df-fc74-43be-8d49-cf84192969de", "Name" : "jack947", "Age" : 947, "
< "_id" : "a30d1179-863e-42d4-be81-dc01a3ec0f51", "Name" : "jack948", "Age" : 948, "
< "_id" : "0b2ec845-65e6-4be8-99af-66172fb8323f", "Name" : "jack949", "Age" : 949, "
< "_id" : "15240fe1-1347-43b6-833e-10c73de96d60", "Name" : "jack950", "Age" : 950, "
< "_id" : "f4b63d96-164d-4682-98f9-b53bc0190d5c", "Name" : "jack951", "Age" : 951, "
< "_id" : "0d012c0f-09b7-44cf-8c3e-532d019bd6a7", "Name" : "jack952", "Age" : 952, "
< "_id" : "b4bf712e-6162-44cb-8bb9-fc8c7b9cb847", "Name" : "jack953", "Age" : 953, "
< "_id" : "116c571b-23ab-4b74-bc03-cac1386598a9", "Name" : "jack954", "Age" : 954, "
< "_id" : "e02be598-5fe6-47ec-9f14-89760b016eae", "Name" : "jack955", "Age" : 955, "
< "_id" : "6d2b068b-bfaa-416d-8609-e1dd7008834c", "Name" : "jack956", "Age" : 956, "
< "_id" : "c41b0519-c53c-4e98-89a7-261a69301752", "Name" : "jack957", "Age" : 957, "
< "_id" : "61201556-9fd5-480f-9b4a-2f2883a1be02", "Name" : "jack958", "Age" : 958, "
< "_id" : "ce089b06-2409-4e87-a4a5-48d68e67b78f", "Name" : "jack959", "Age" : 959, "
< "_id" : "46d00b4c-16ae-4efe-a47a-28c83faa5177", "Name" : "jack960", "Age" : 960, "
< "_id" : "5438ab65-9758-49a6-872f-ea7e75278df3", "Name" : "jack961", "Age" : 961, "
has more
>

```

<4> list 操作： 这里我获取一下名字里面带 9 的人数列表

```

1      static void Main(string[] args)
2      {
3          MongoDBHelper<Person> helper = new
MongoDbHelper<Person>();
4
5          int pagecount;
6
7          //获取名字里面带 9 的人数
8          var list = helper.List(1, 20, i =>
i.Name.Contains("9"), out pagecount);
9
10         Console.Read();
11     }

```

```
public class Program
{
    static void Main(string[] args)
    {
        MongoDBHelper<Person> helper = new MongoDBHelper<Person>();

        int pagecount;

        //获取名字里面带9的人数
        var list = helper.List(1, 20, i => i.Name.Contains("9"), out pagecount);

        Console.Read();
    }
}
```

监视 1

名称
list
[0]
[1]
[2]
[3]
[4]
[5]
[6]
[7]
[8]
[9]
[10]

监视 2

名称	值	类型
pagecount	999	int

总的运行代码

[View Code](#)

wow，趁着 3 天的休假，不断的努力终于把这个系列写完了，很感谢一直关注此系列的朋友。

