[8天学通MongoDB——第一天 基础入门](http://www.cnblogs.com/huangxincheng/archive/2012/02/18/2356595.html)

   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](http://www.mongodb.org/downloads) ，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。

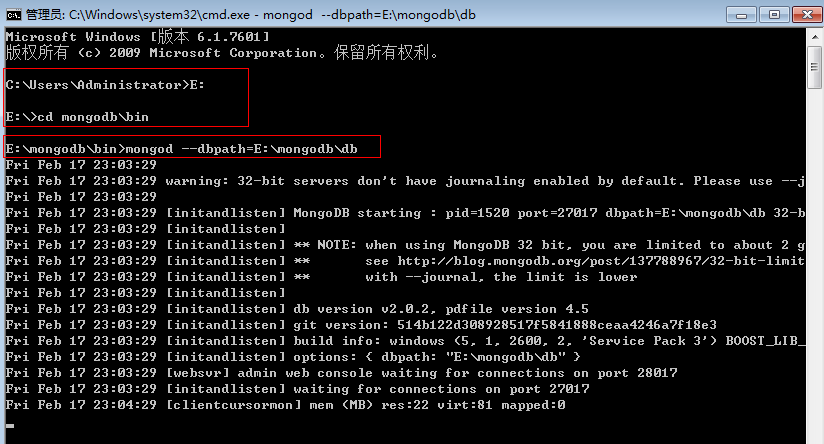
     我这里就下载"2.0.2版本，32bit“，ok，下载之后我就放到”E盘“，改下文件夹名字为”mongodb“。

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。

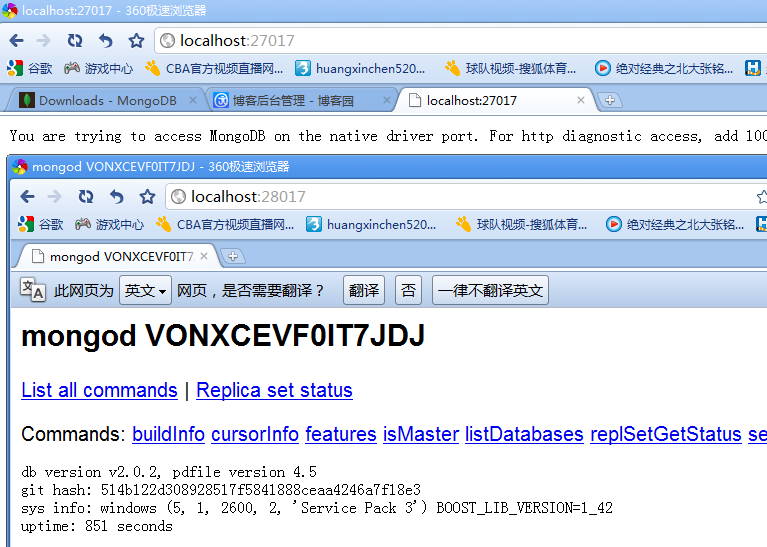


     ②：微软徽标+R，输入cmd，首先找到“mongodb”的路径，然后运行mongod开启命令，同时用--dbpath指定数据存放地点为“db”文件夹。



      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/>”，

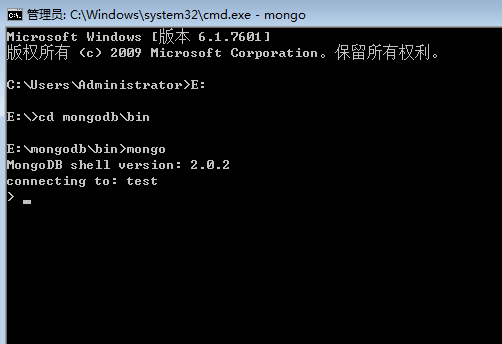
             打开后，mongodb告诉我们在27017上Add 1000可以用http模式查看mongodb的管理信息。



三：基本操作

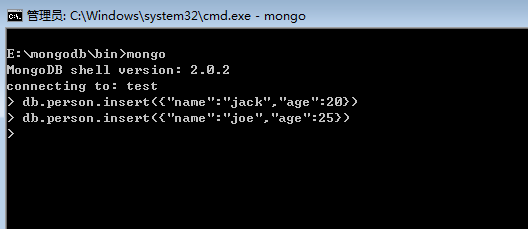
     由于是开篇，就大概的说下基本的“增删查改“，我们再开一个cmd，输入mongo命令打开shell，其实这个shell就是mongodb的客户端，

同时也是一个js的编译器，默认连接的是“test”数据库。



    <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。

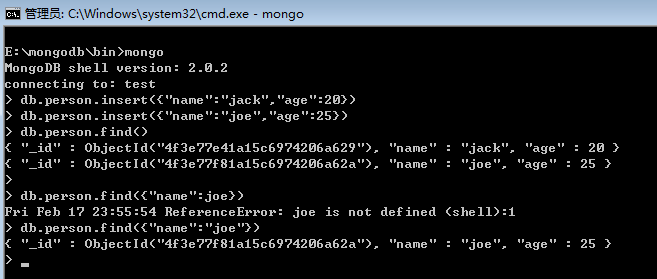


  <2> find 操作

       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：

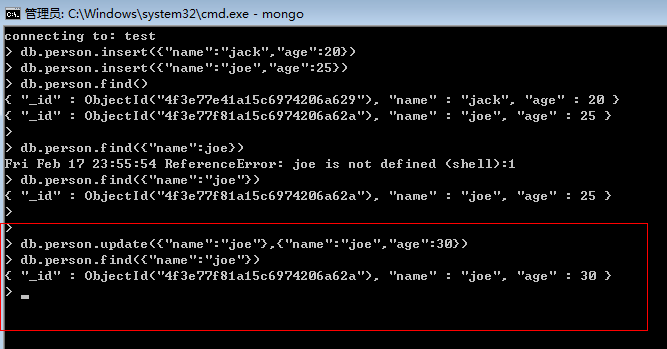
           ① “\_id"： 这个字段是数据库默认给我们加的GUID，目的就是保证数据的唯一性。

           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。



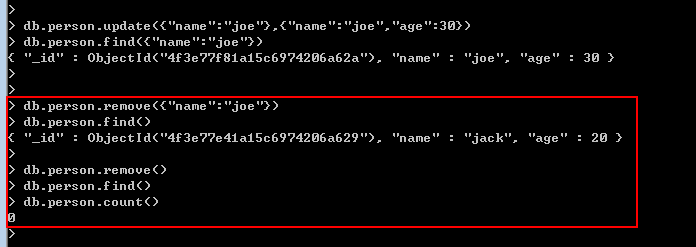
 <3> update操作

      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。



<4> remove操作

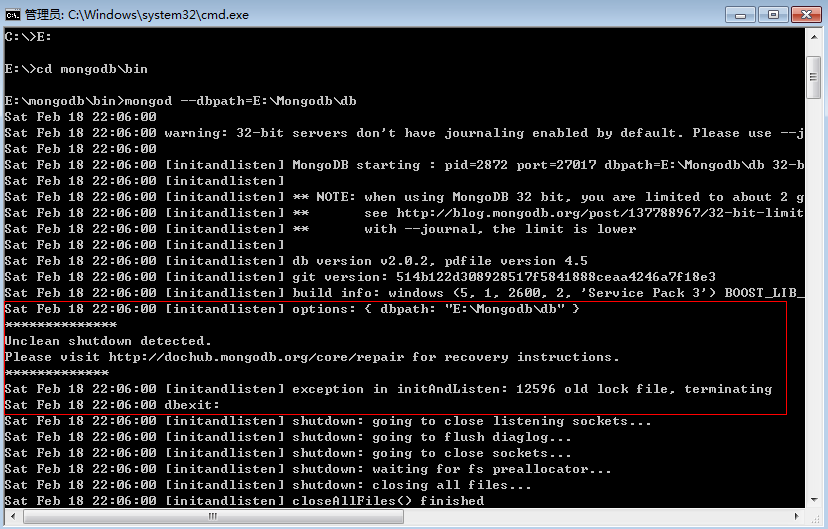
      remove中如果不带参数将删除所有数据，呵呵，很危险的操作，在mongodb中是一个不可撤回的操作，三思而后行。



  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，首先当我们用上一篇同样的方式打开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。



一： Insert操作

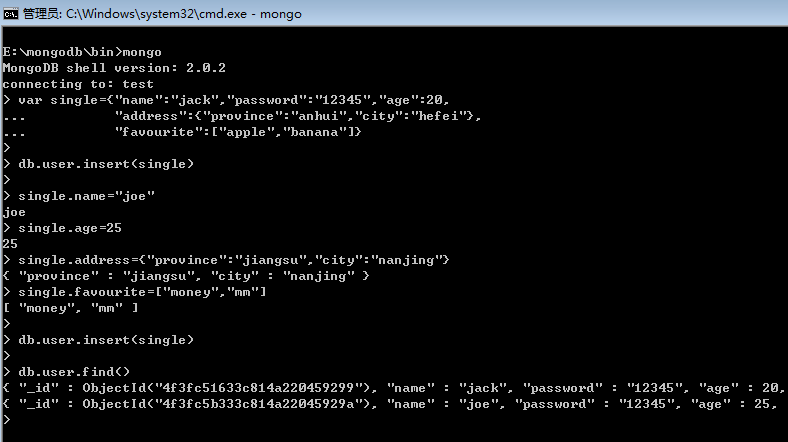
     上一篇也说过,文档是采用“K-V”格式存储的，如果大家对JSON比较熟悉的话，我相信学mongodb是手到擒来，我们知道JSON里面Value

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。



   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

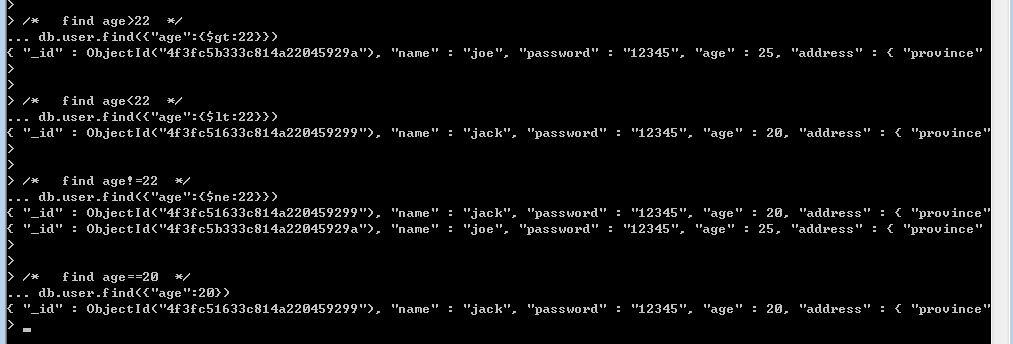
     日常开发中，我们玩查询，玩的最多的也就是二类：

     ①： >, >=, <, <=, !=, =。

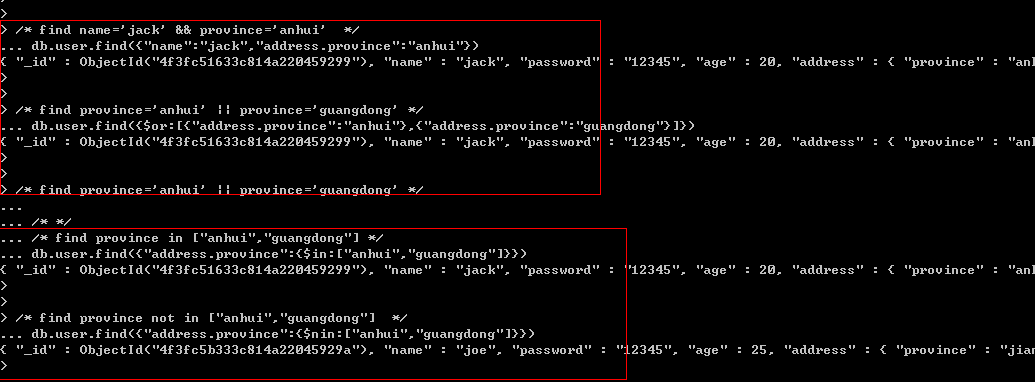
     ②：And，OR，In，NotIn

这些操作在mongodb里面都封装好了，下面就一一介绍：

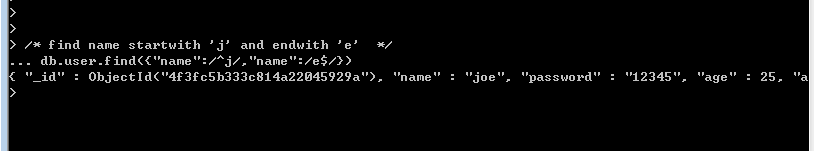
    <1>"$gt", "$gte", "$lt", "$lte", "$ne", "没有特殊关键字"，这些跟上面是一一对应的，举几个例子。



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

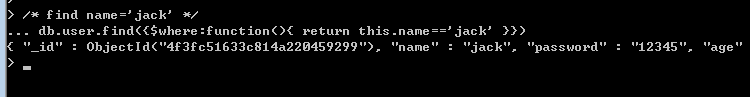


<3> 在mongodb中还有一个特殊的匹配，那就是“正则表达式”，这玩意威力很强的。



<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。

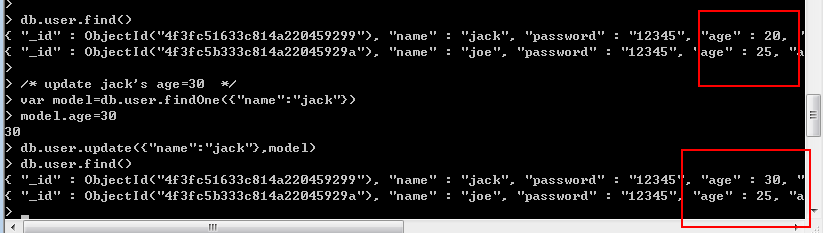


三：Update操作

      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。



  <2> 局部更新

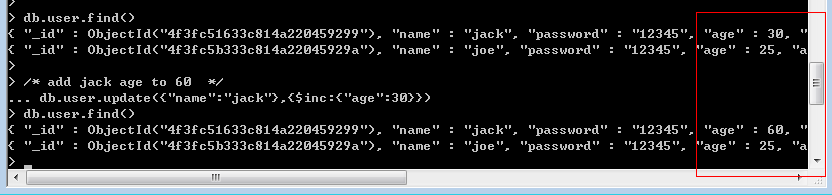
        有时候我们仅仅需要更新一个字段，而不是整体更新，那么我们该如何做呢？easy的问题，mongodb中已经给我们提供了两个

   修改器： $inc 和 $set。

   ①  $inc修改器

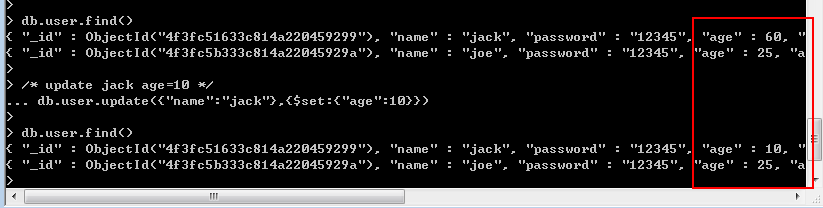
       $inc也就是increase的缩写，学过sql server 的同学应该很熟悉，比如我们做一个在线用户状态记录，每次修改会在原有的基础上

    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。



 ② $set修改器

      啥也不说了，直接上代码

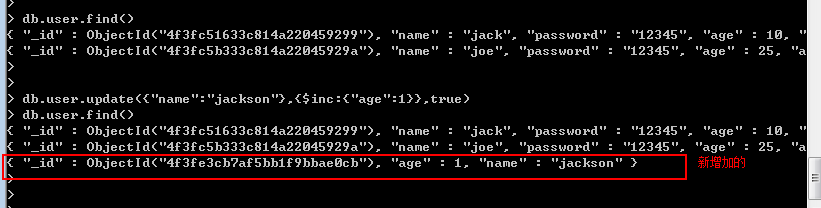


 <3> upsert操作

     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。



 <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。

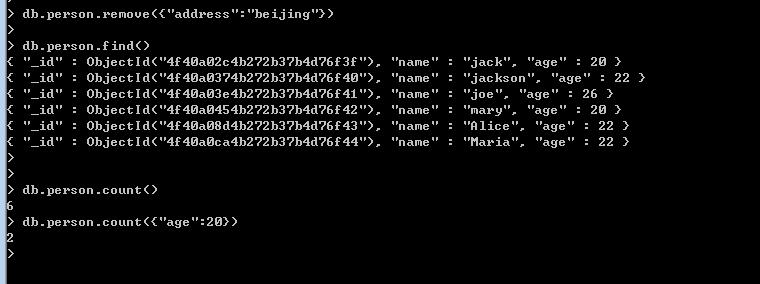
今天跟大家分享一下mongodb中比较好玩的知识，主要包括：聚合，游标。

一： 聚合

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

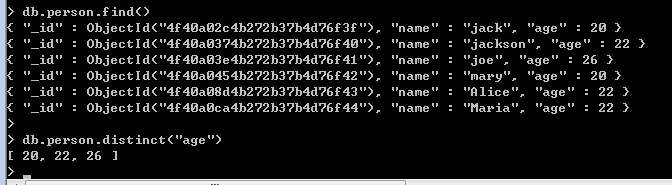
<1> count

        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。



<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。



<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。



   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。



<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。

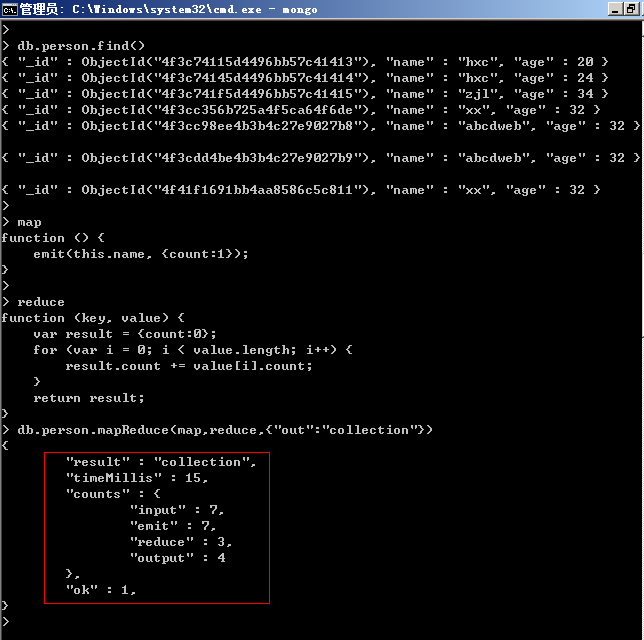
   ② reduce：

         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：



从图中我们可以看到如下信息：

       result: "存放的集合名“；

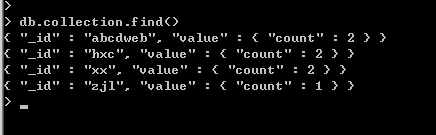
       input:传入文档的个数。

       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。



二：游标

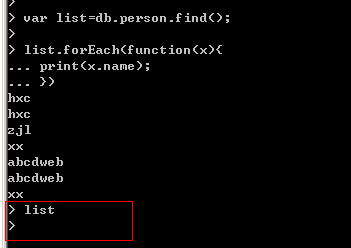
    mongodb里面的游标有点类似我们说的C#里面延迟执行，比如：

      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，

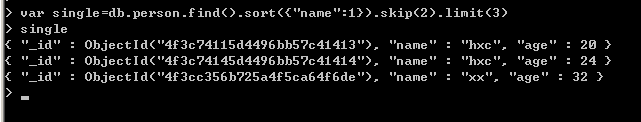
发现没有数据返回了。



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。



 这些天项目改版，时间比较紧，博客也就没跟得上，还望大家见谅。

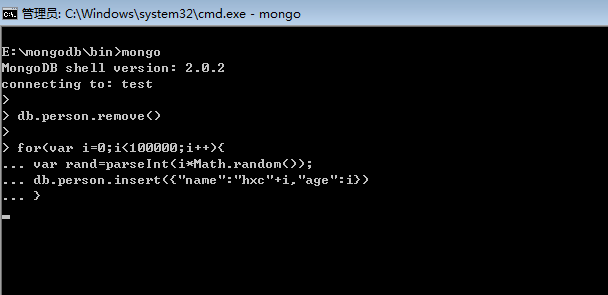
    好，今天分享下mongodb中关于索引的基本操作，我们日常做开发都避免不了要对程序进行性能优化，而程序的操作无非就是CURD，通常我们

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。

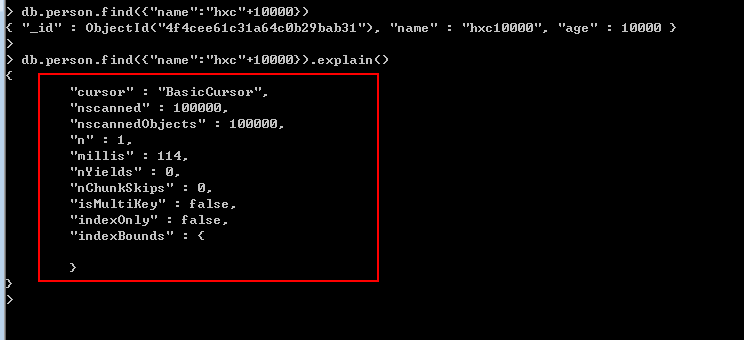
   我们首先插入10w数据，上图说话：



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。



 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。

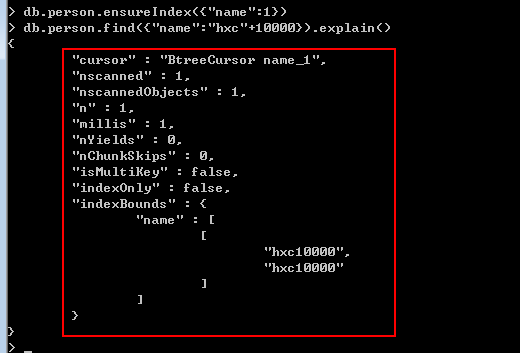
   n:             这里是1，也就是最终返回了1个文档。

   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.....



  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。

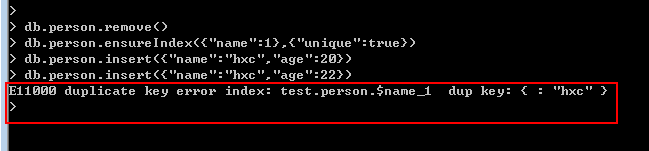
   millis:        看看这个时间真的不敢相信，秒秒杀。

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：

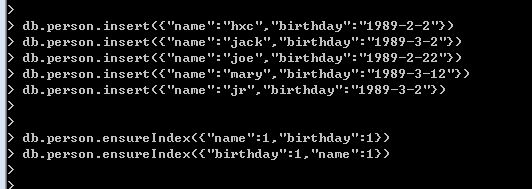
db.person.ensureIndex({"name":1},{"unique":true})。



四：组合索引

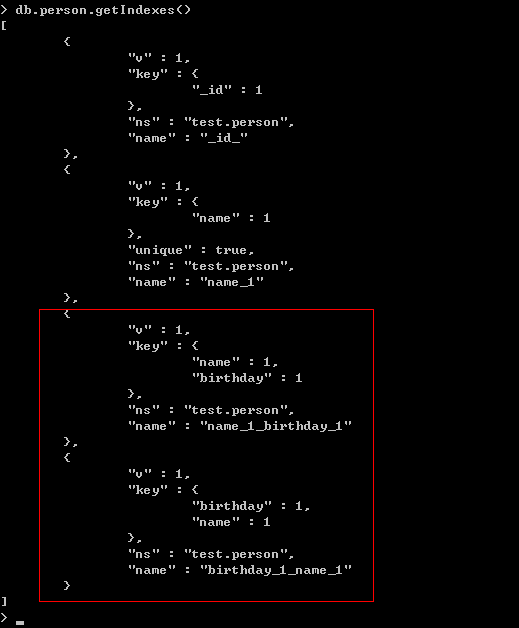
     有时候我们的查询不是单条件的，可能是多条件，比如查找出生在‘1989-3-2’名字叫‘jack’的同学，那么我们可以建立“姓名”和"生日“

的联合索引来加速查询。



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。



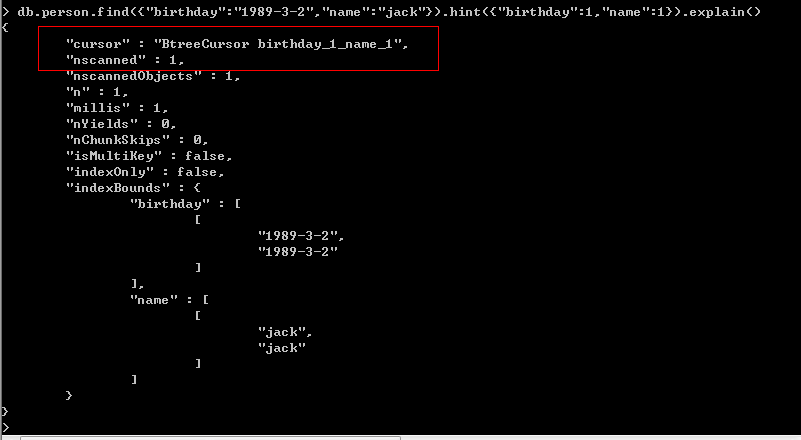
此时我们肯定很好奇，到底查询优化器会使用哪个查询作为操作，呵呵，还是看看效果图：



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

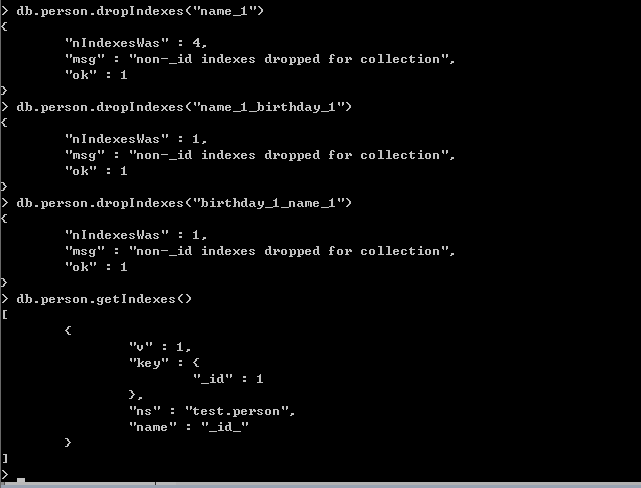
可以的，在mongodb中给我们提供了hint方法让我们可以暴力执行。



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。



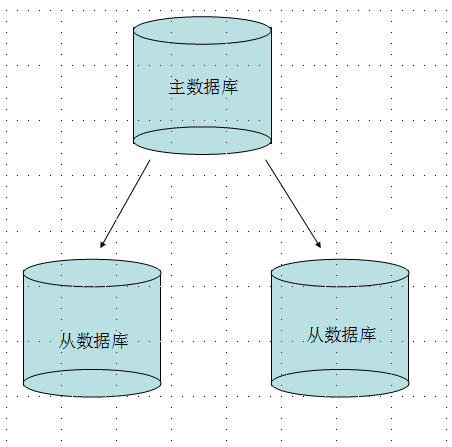
从这一篇开始我们主要讨论mongodb的部署技术。

    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。

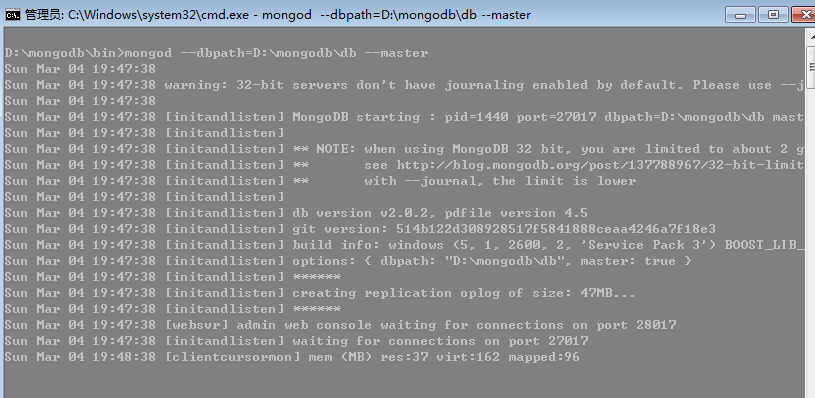
3：下面我们就一一实践

     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，

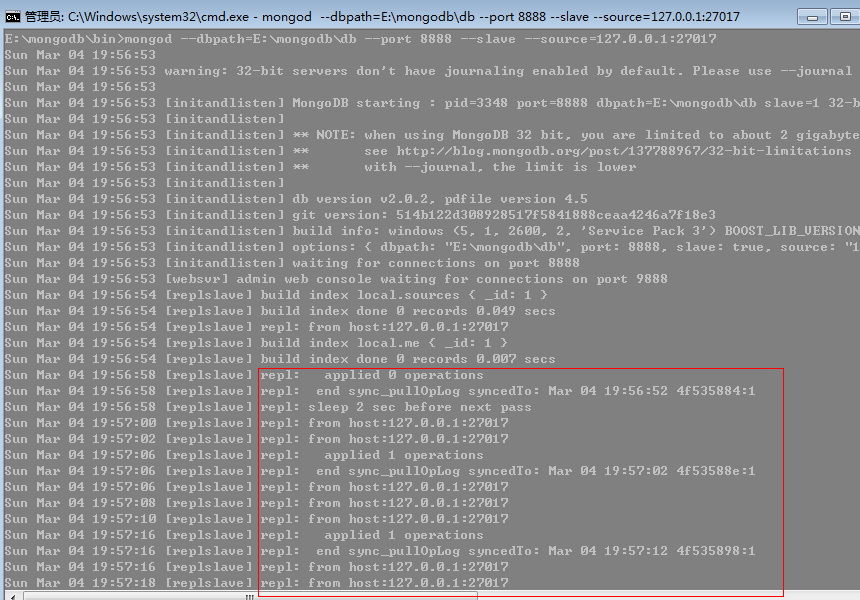
           端口还是默认的27017.



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。

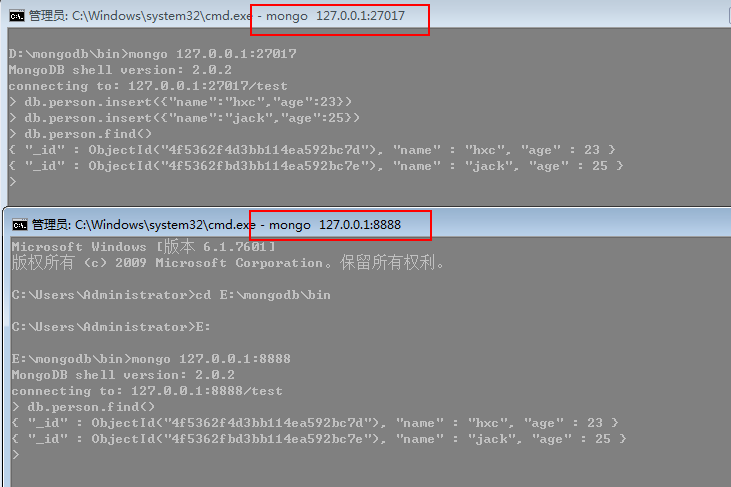
           >mongod --dbpath=xxxx --port=8888 --slave --source=127.0.0.1:27017



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。

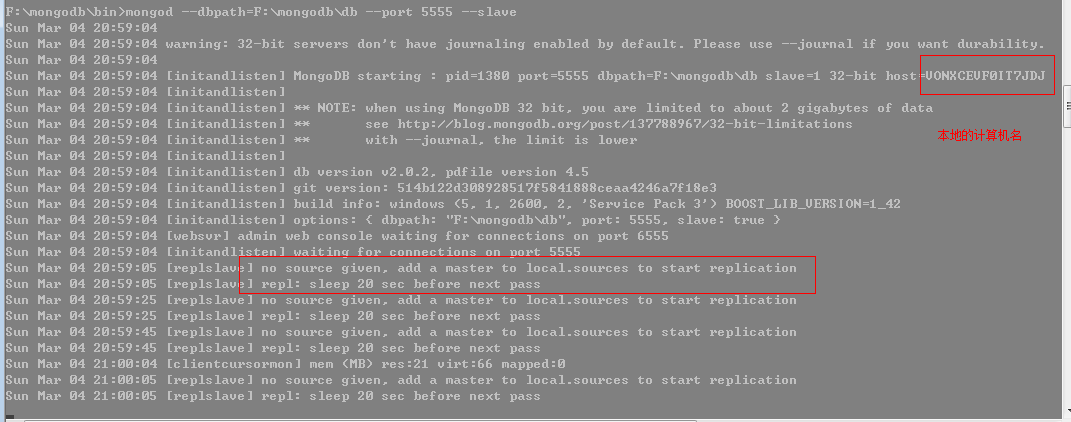
           接下来我们要做的就是测试，惊讶的发现数据已经同步更新，爽啊。



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。

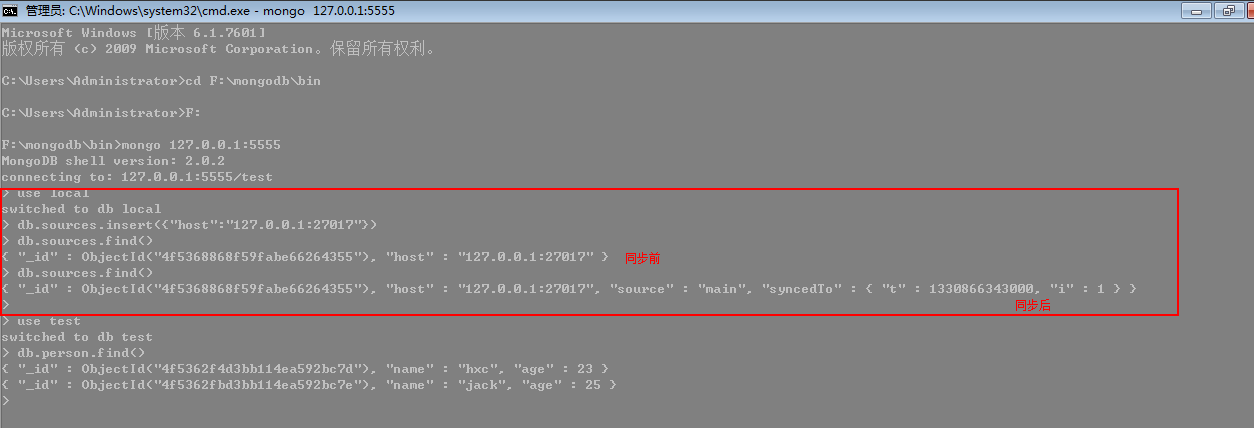
      我们的主或者从属数据库中都有一个叫做local的集合，主要是用于存放内部复制信息。

      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。



    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....



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。

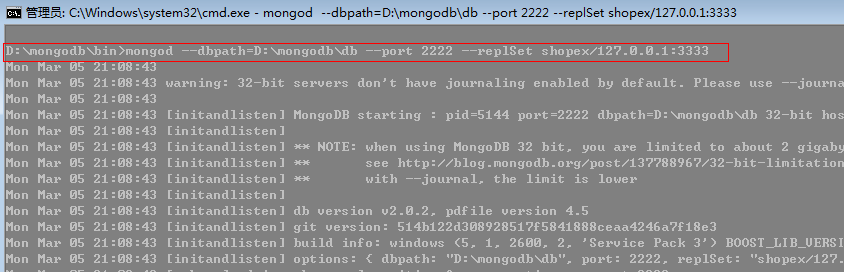
      <1>:  该集群没有特定的主数据库。

      <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。

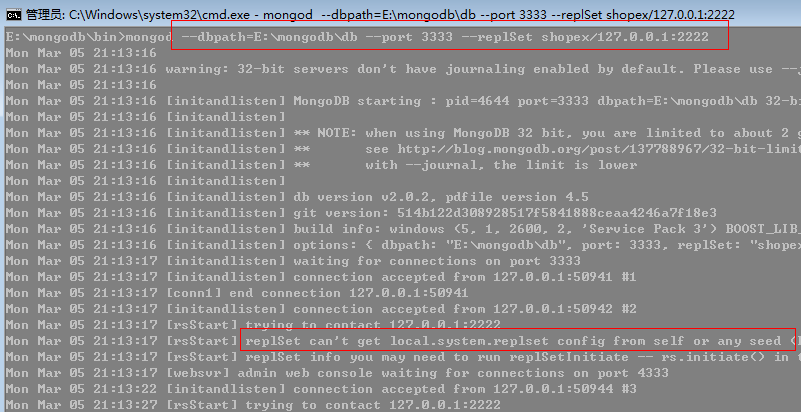
                 好，我们现在就来试一下，首先把所有的cmd窗口关掉重新来，清掉db下的所有文件。

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。

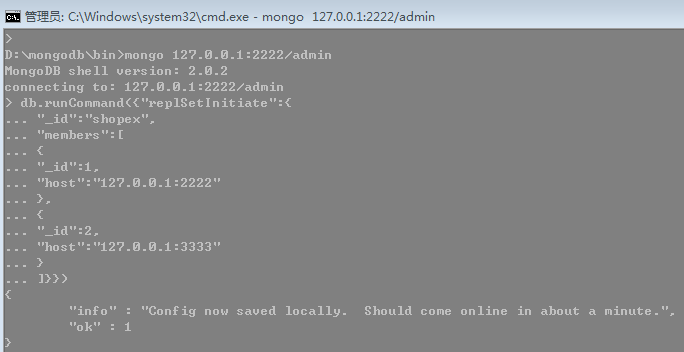


第二步:  既然上面说3333是另一个数据库服务器，不要急，现在就来开，这里把E盘的mongodb程序打开。

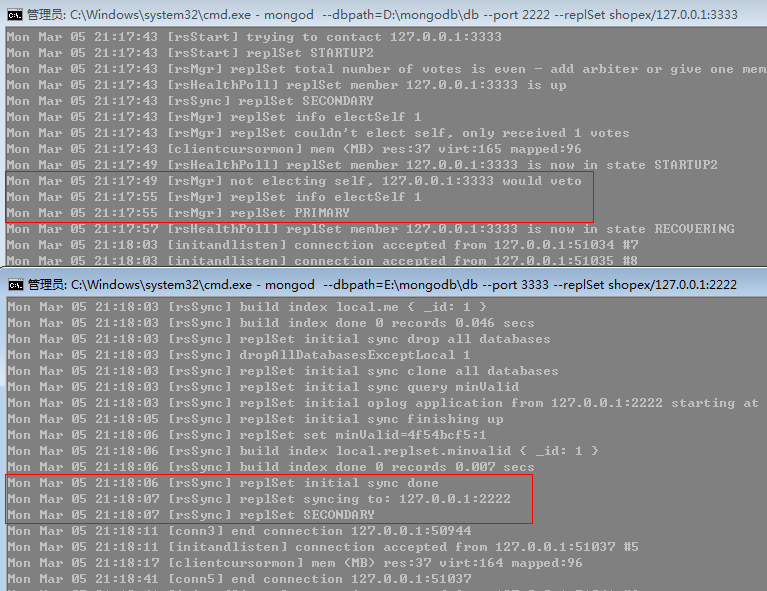


第三步：  ok，看看上面的日志红色区域，似乎我们还没有做完，是的，log信息告诉我们要初始化一下“副本集“，既然日志这么说，那我也就

             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。

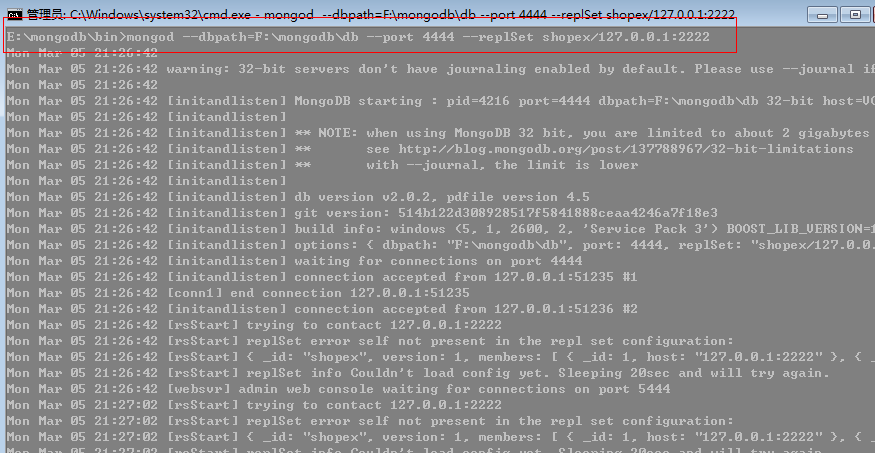


第四步： 开启成功后，我们要看看谁才能成为主数据库服务器，可以看到端口为2222的已经成为主数据库服务器。

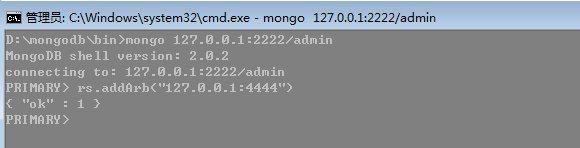


第五步：我们知道sql server里面有一个叫做仲裁服务器，那么mongodb中也是有的，跟sql server一样，仲裁只参与投票选举，这里我们

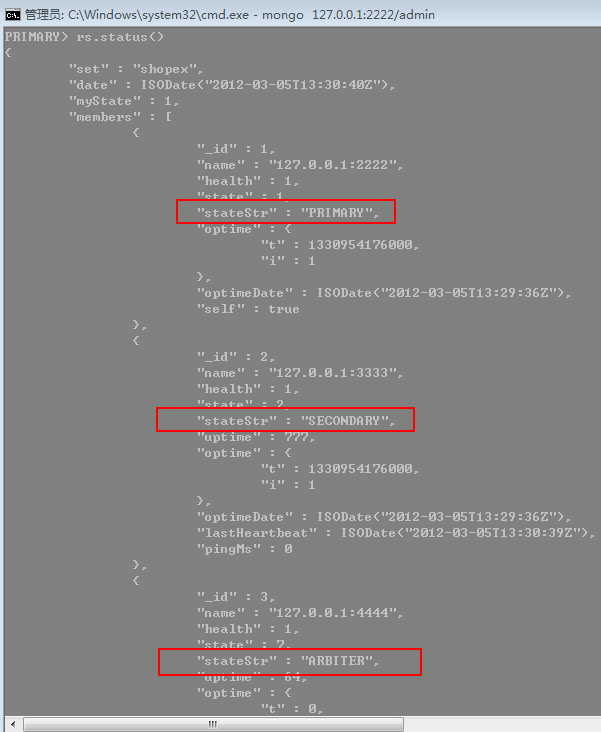
           把F盘的mongodb作为仲裁服务器，然后指定shopex集群中的任一个服务器端口，这里就指定2222。



然后我们在admin集合中使用rs.addArb()追加即可。

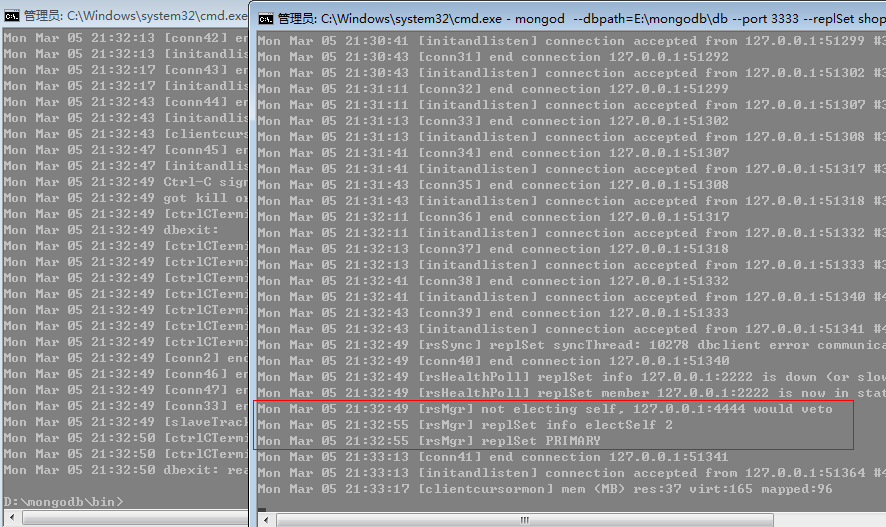


追加好了之后，我们使用rs.status()来查看下集群中的服务器状态，图中我们可以清楚的看到谁是主，还是从，还是仲裁。



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。

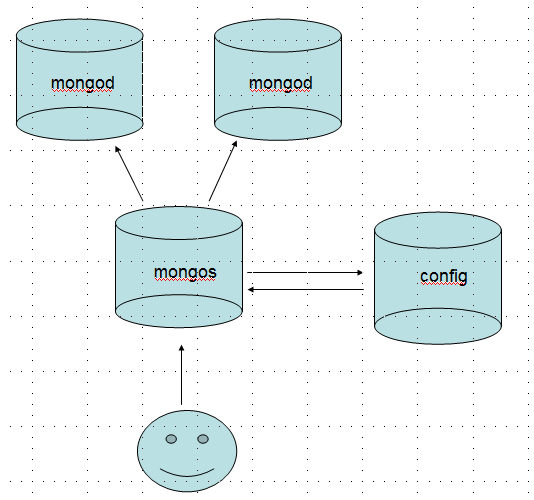


 在mongodb里面存在另一种集群，就是分片技术，跟sql server的表分区类似，我们知道当数据量达到T级别的时候，我们的磁盘，内存

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。

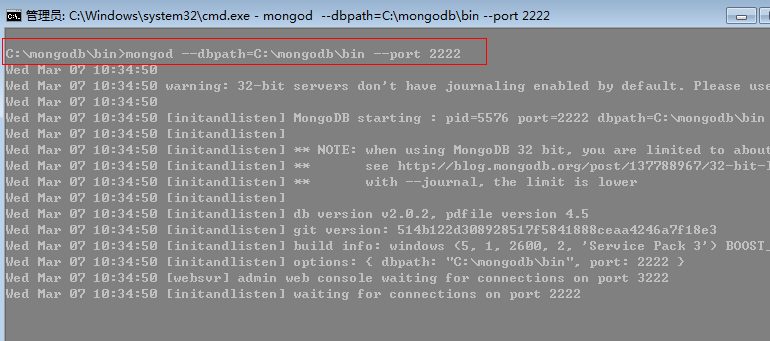
    mongod:   一个普通的数据库实例，如果不分片的话，我们会直接连上mongod。

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。

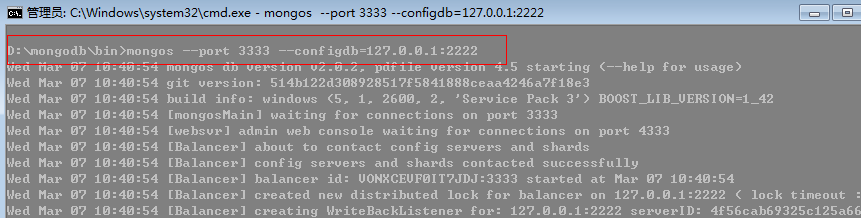
1：开启config服务器

      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。



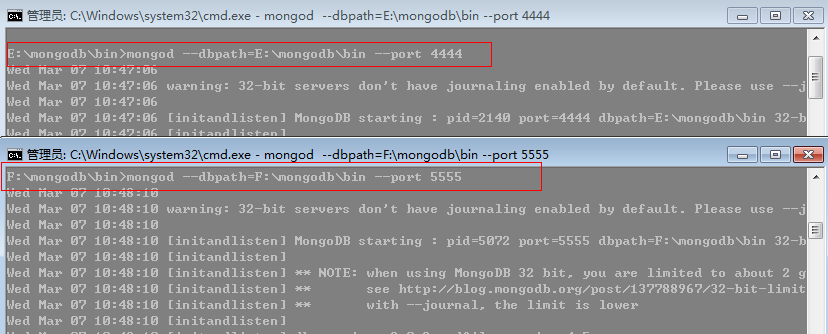
2: 开启mongos服务器

    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。



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。

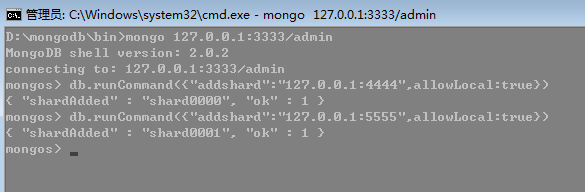


4： 服务配置

    哈哈，是不是很兴奋，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()。

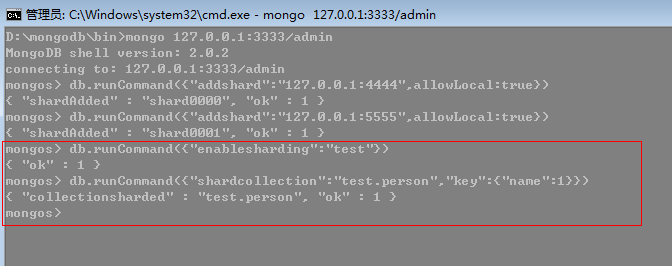


这里要注意的是，在addshard中，我们也可以添加副本集，这样能达到更高的稳定性。

<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。

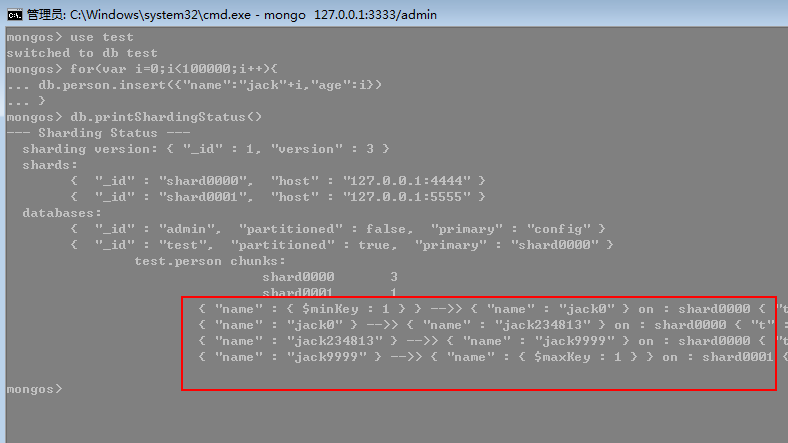
        ②：指定集合中分片的片键，这里我就指定为person.name字段。



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。



 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。

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，

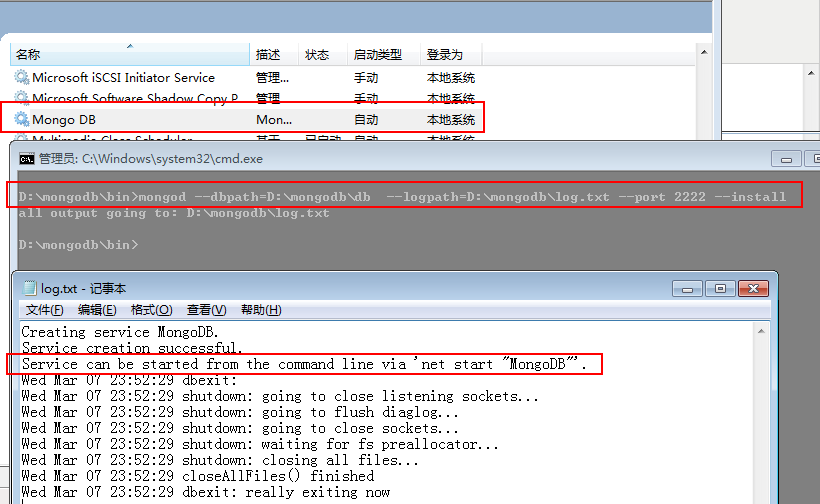
下面我们就一点一点的讲解。

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。

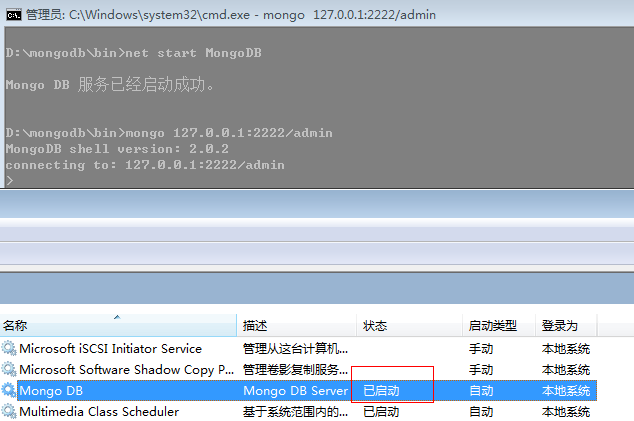


这里要注意的有两点：

   <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。

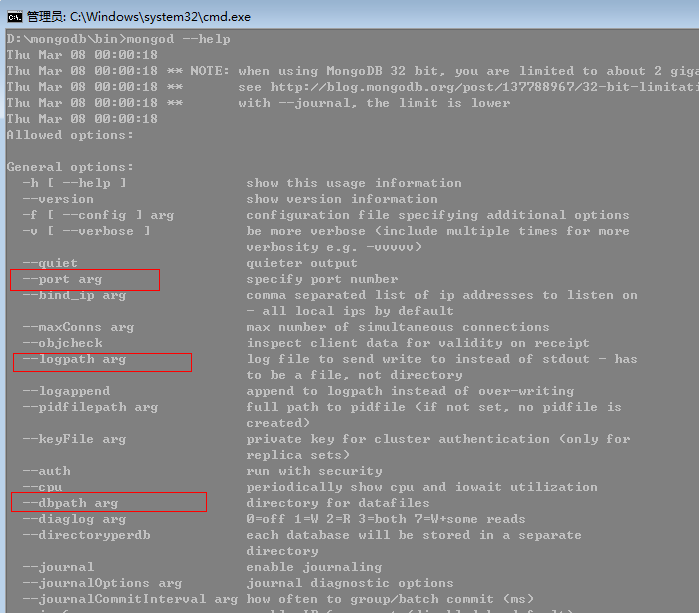
好了，console程序叫我看log日志，那我就看看，发现mongodb已经提示我们如何开启mongodb，接着我照做就是了。



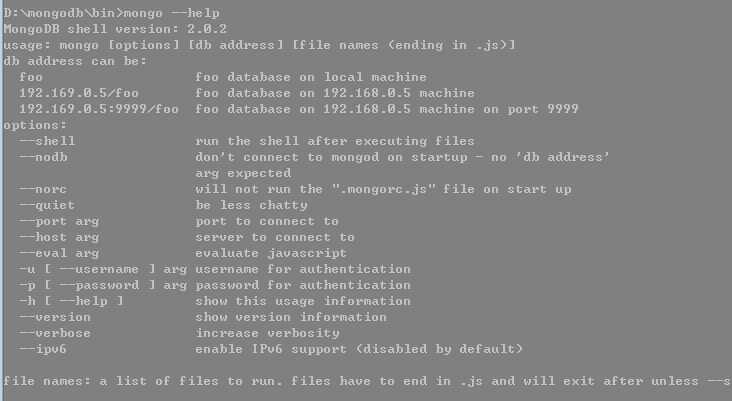
还要提醒大家一点的就是，这些命令参数很多很复杂也就很容易忘，不过没关系，数据库给我们提供了一个help方法，我们可以

拿mongod和mongo说事。

mongod：



mongo：



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

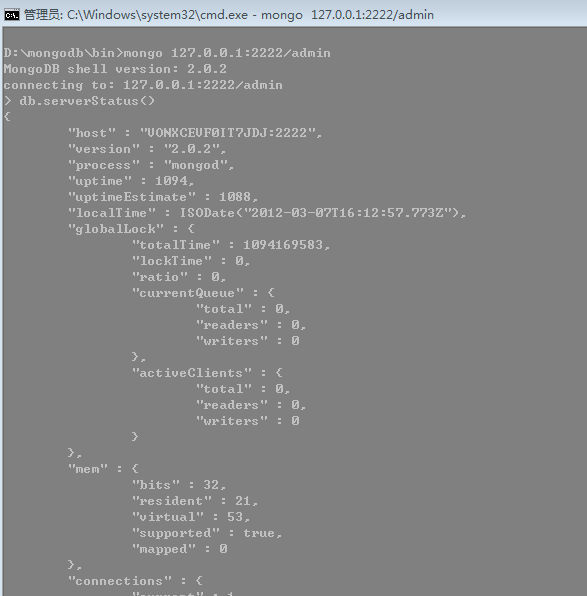
     这个我在先前的文章中也提到了，这里就不赘述了。

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>

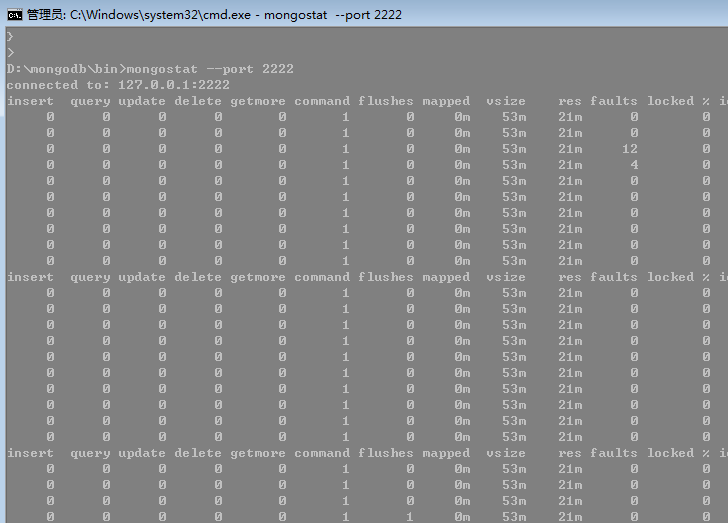
    这里还是截个图混个眼熟。



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。

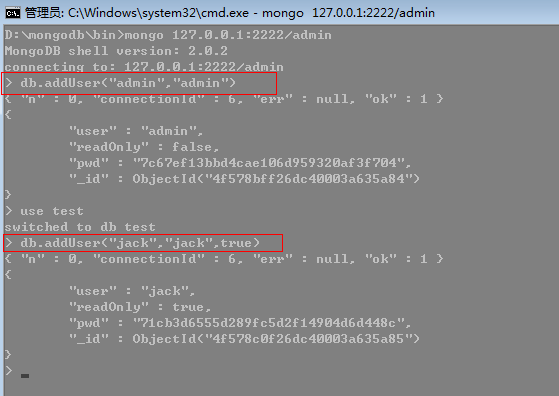


三: 安全认证

     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

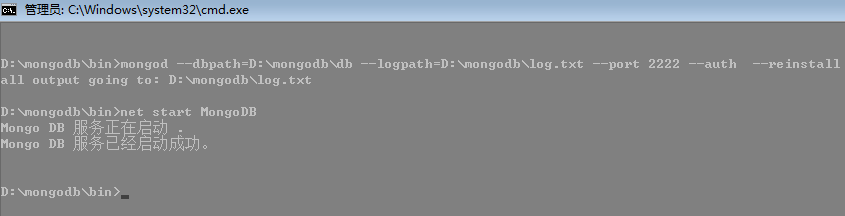
验证的方法，那么该怎么做呢？其实很简单，mongodb提供了addUser方法，还有一个注意点就是如果在admin数据库中添加

将会被视为“超级管理员”。

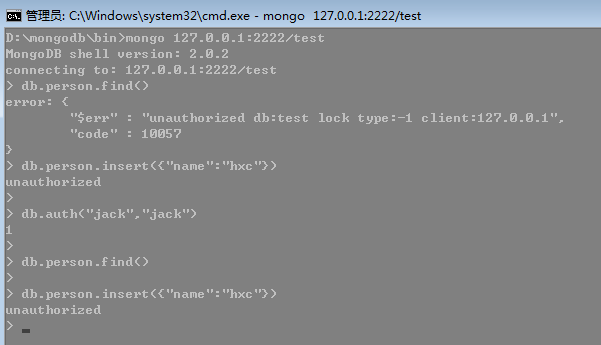


上面的admin用户将会被视为超级管理员，“jack”用户追加的第三个参数表示是否是“只读用户”，好了，该添加的我们都添加了，

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。



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。



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

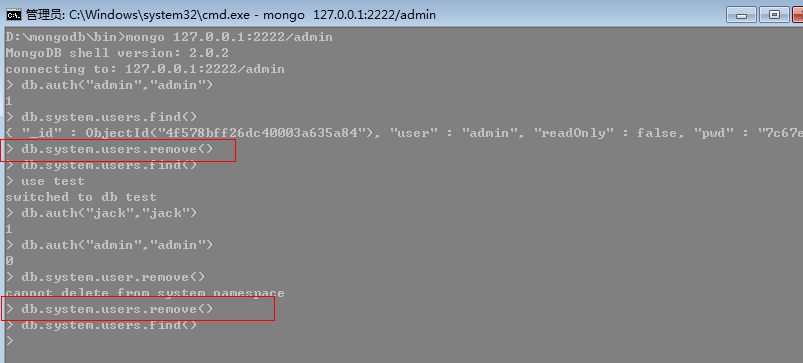
       这个算是最简单的了，不过要注意一点，在服务器运行的情况下直接copy是很有风险的，可能copy出来时，数据已经遭到

        破坏，唯一能保证的就是要暂时关闭下服务器，copy完后重开。

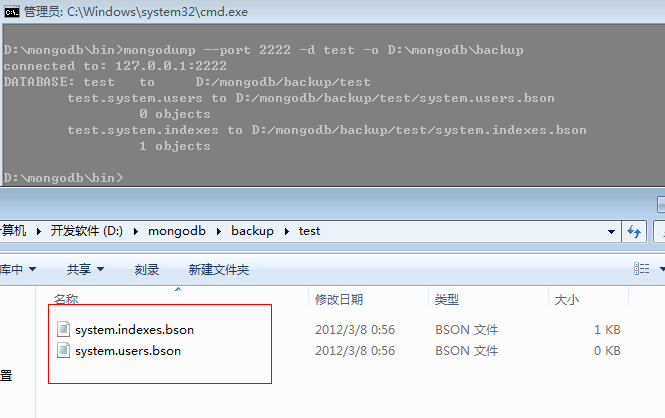
2：mongodump和mongorestore

      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。

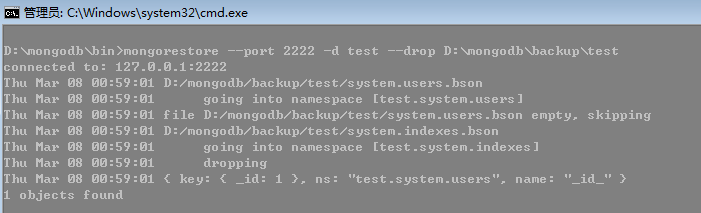
为了操作方便，我们先删除授权用户。



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。



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。



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()

  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。

[复制代码](javascript:void(0);)

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

[复制代码](javascript:void(0);)

2：初始化一些变量

[复制代码](javascript:void(0);)

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

[复制代码](javascript:void(0);)

3：为了方便T的继承类使用linq功能，我们还需要映射一下。

[复制代码](javascript:void(0);)

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 => t.IsSubclassOf(typeof(T)));  
16 });  
17 mapping.Map<T>();  
18 mapping.Map<T>();  
19 });  
20   
21 config.ConnectionString(connectionString);  
22   
23 return config.BuildConfiguration();  
24 }  
25 }  
26 #endregion

[复制代码](javascript:void(0);)

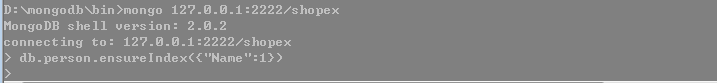
4：下面是一些基本的CURD的代码，跟写EF代码很类似，写起来好舒服。

[复制代码](javascript:void(0);)

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 .Take(pageSize).Select(i => i).ToList();  
 89   
 90 mongo.Disconnect();  
 91   
 92 return personList;  
 93   
 94 }  
 95 catch (Exception)  
 96 {  
 97 mongo.Disconnect();  
 98 throw;  
 99 }  
100 }  
101 }  
102 #endregion  
103   
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

[复制代码](javascript:void(0);)

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。



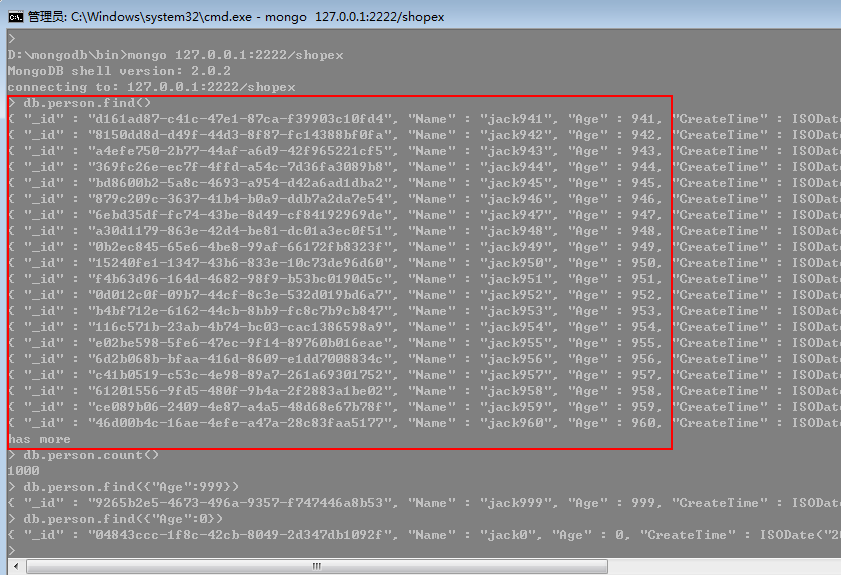
6. 一切准备妥当，我们做下基本的操作，比如这里我添加一千条数据，注意我开启的是安全模式，如果插入不成功，将会抛出异常。

 <1> Add:

[复制代码](javascript:void(0);)

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 }

[复制代码](javascript:void(0);)



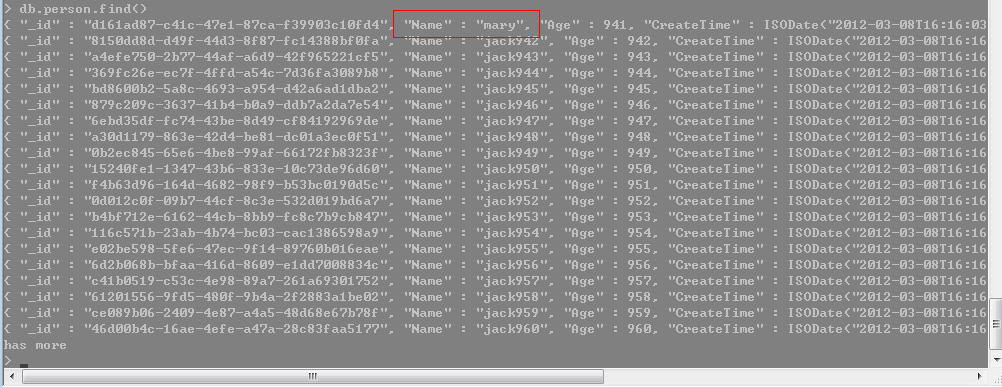
乍一看显示的数据以为有问题，为什么没有出现jack0或者jack999，不过find的一下后心情舒坦了。

<2> update:   这里就把jack941的名字改掉“mary”

[复制代码](javascript:void(0);)

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 }

[复制代码](javascript:void(0);)

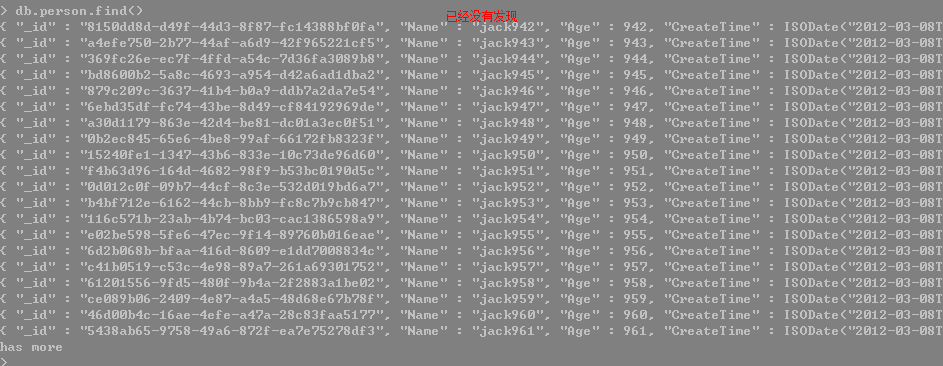


<3>Delete：  删除mary这条记录

[复制代码](javascript:void(0);)

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 }

[复制代码](javascript:void(0);)

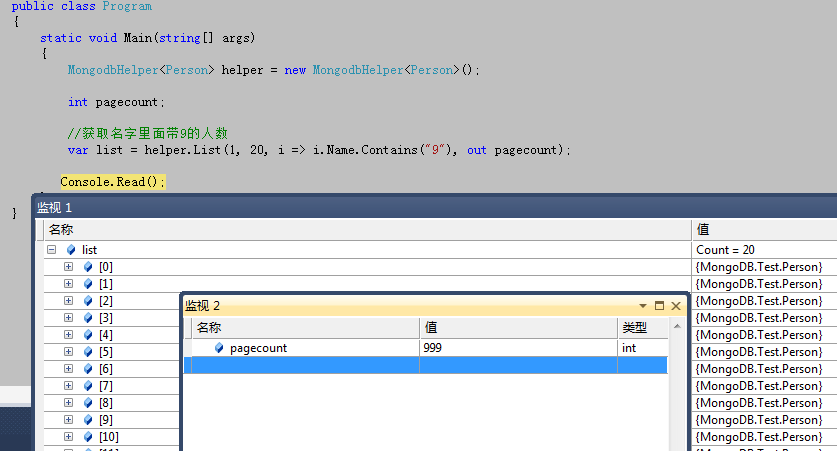


<4> list操作： 这里我获取一下名字里面带9的人数列表

[复制代码](javascript:void(0);)

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 }

[复制代码](javascript:void(0);)



总的运行代码

http://images.cnblogs.com/OutliningIndicators/ExpandedBlockStart.gifView Code

[复制代码](javascript:void(0);)

1 using System;  
 2 using System.Collections.Generic;  
 3 using System.Linq;  
 4 using System.Text;  
 5 using System.Configuration;  
 6 using System.Linq.Expressions;  
 7   
 8 using MongoDB.Configuration;  
 9 using MongoDB.Linq;  
 10 using MongoDB.Attributes;  
 11   
 12   
 13 namespace MongoDB.Test  
 14 {  
 15 public class MongodbHelper<T> where T : class  
 16 {  
 17 string connectionString = string.Empty;  
 18   
 19 string databaseName = string.Empty;  
 20   
 21 string collectionName = string.Empty;  
 22   
 23 static MongodbHelper<T> mongodb;  
 24   
 25 #region 初始化操作  
 26 /// <summary>  
 27 /// 初始化操作  
 28 /// </summary>  
 29 public MongodbHelper()  
 30 {  
 31 connectionString = "Server=127.0.0.1:2222";  
 32 databaseName = "shopex";  
 33 collectionName = "person";  
 34 }  
 35 #endregion  
 36   
 37 #region 实现linq查询的映射配置  
 38 /// <summary>  
 39 /// 实现linq查询的映射配置  
 40 /// </summary>  
 41 public MongoConfiguration configuration  
 42 {  
 43 get  
 44 {  
 45 var config = new MongoConfigurationBuilder();  
 46   
 47 config.Mapping(mapping =>  
 48 {  
 49 mapping.DefaultProfile(profile =>  
 50 {  
 51 profile.SubClassesAre(t => t.IsSubclassOf(typeof(T)));  
 52 });  
 53 mapping.Map<T>();  
 54 mapping.Map<T>();  
 55 });  
 56   
 57 config.ConnectionString(connectionString);  
 58   
 59 return config.BuildConfiguration();  
 60 }  
 61 }  
 62 #endregion  
 63   
 64 #region 插入操作  
 65 /// <summary>  
 66 /// 插入操作  
 67 /// </summary>  
 68 /// <param name="person"></param>  
 69 /// <returns></returns>  
 70 public void Insert(T t)  
 71 {  
 72 using (Mongo mongo = new Mongo(configuration))  
 73 {  
 74 try  
 75 {  
 76 mongo.Connect();  
 77   
 78 var db = mongo.GetDatabase(databaseName);  
 79   
 80 var collection = db.GetCollection<T>(collectionName);  
 81   
 82 collection.Insert(t, true);  
 83   
 84 mongo.Disconnect();  
 85   
 86 }  
 87 catch (Exception)  
 88 {  
 89 mongo.Disconnect();  
 90 throw;  
 91 }  
 92 }  
 93 }  
 94 #endregion  
 95   
 96 #region 更新操作  
 97 /// <summary>  
 98 /// 更新操作  
 99 /// </summary>  
100 /// <param name="person"></param>  
101 /// <returns></returns>  
102 public void Update(T t, Expression<Func<T, bool>> func)  
103 {  
104 using (Mongo mongo = new Mongo(configuration))  
105 {  
106 try  
107 {  
108 mongo.Connect();  
109   
110 var db = mongo.GetDatabase(databaseName);  
111   
112 var collection = db.GetCollection<T>(collectionName);  
113   
114 collection.Update<T>(t, func, true);  
115   
116 mongo.Disconnect();  
117   
118 }  
119 catch (Exception)  
120 {  
121 mongo.Disconnect();  
122 throw;  
123 }  
124 }  
125 }  
126 #endregion  
127   
128 #region 获取集合  
129 /// <summary>  
130 ///获取集合  
131 /// </summary>  
132 /// <param name="person"></param>  
133 /// <returns></returns>  
134 public List<T> List(int pageIndex, int pageSize, Expression<Func<T, bool>> func, out int pageCount)  
135 {  
136 pageCount = 0;  
137   
138 using (Mongo mongo = new Mongo(configuration))  
139 {  
140 try  
141 {  
142 mongo.Connect();  
143   
144 var db = mongo.GetDatabase(databaseName);  
145   
146 var collection = db.GetCollection<T>(collectionName);  
147   
148 pageCount = Convert.ToInt32(collection.Count());  
149   
150 var personList = collection.Linq().Where(func).Skip(pageSize \* (pageIndex - 1))  
151 .Take(pageSize).Select(i => i).ToList();  
152   
153 mongo.Disconnect();  
154   
155 return personList;  
156   
157 }  
158 catch (Exception)  
159 {  
160 mongo.Disconnect();  
161 throw;  
162 }  
163 }  
164 }  
165 #endregion  
166   
167 #region 读取单条记录  
168 /// <summary>  
169 ///读取单条记录  
170 /// </summary>  
171 /// <param name="person"></param>  
172 /// <returns></returns>  
173 public T Single(Expression<Func<T, bool>> func)  
174 {  
175 using (Mongo mongo = new Mongo(configuration))  
176 {  
177 try  
178 {  
179 mongo.Connect();  
180   
181 var db = mongo.GetDatabase(databaseName);  
182   
183 var collection = db.GetCollection<T>(collectionName);  
184   
185 var single = collection.Linq().FirstOrDefault(func);  
186   
187 mongo.Disconnect();  
188   
189 return single;  
190   
191 }  
192 catch (Exception)  
193 {  
194 mongo.Disconnect();  
195 throw;  
196 }  
197 }  
198 }  
199 #endregion  
200   
201 #region 删除操作  
202 /// <summary>  
203 /// 删除操作  
204 /// </summary>  
205 /// <param name="person"></param>  
206 /// <returns></returns>  
207 public void Delete(Expression<Func<T, bool>> func)  
208 {  
209 using (Mongo mongo = new Mongo(configuration))  
210 {  
211 try  
212 {  
213 mongo.Connect();  
214   
215 var db = mongo.GetDatabase(databaseName);  
216   
217 var collection = db.GetCollection<T>(collectionName);  
218   
219 //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  
220 //导致删除失败  
221 collection.Remove<T>(func);  
222   
223 mongo.Disconnect();  
224   
225 }  
226 catch (Exception)  
227 {  
228 mongo.Disconnect();  
229 throw;  
230 }  
231 }  
232 }  
233 #endregion  
234 }  
235   
236 #region 数据实体  
237 /// <summary>  
238 /// 数据实体  
239 /// </summary>  
240 public class Person  
241 {  
242 [MongoAlias("\_id")]  
243 public string ID { get; set; }  
244   
245 public string Name { get; set; }  
246   
247 public int Age { get; set; }  
248   
249 public DateTime CreateTime { get; set; }  
250 }  
251 #endregion  
252 }

[复制代码](javascript:void(0);)

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。http://pic002.cnblogs.com/images/2012/214741/2012030900330960.gif