MongoDB 数据查询

李焕贞

河北师范大学软件学院

本章大纲

- find函数介绍及使用
- 查询操作符介绍及使用
- 内嵌文档与数组查询
- MongoDB的游标
- 模糊查询（操作符$regex 正则表达式）
- findAndModify函数的使用

find函数介绍及使用

MongoDB数据查询使用find函数，其功能与SQL中的select函数相同，可提供与关系型数据库类似的许多功能，包含映射、排序等。

关系型数据库	MongoDB
SELECT * FROM student WHERE Sage > 25 AND Sage <= 50	db.student.find ({ Sage:{ $gt:25, $lte:50 } })
SELECT student.Sno, Sname, Cno, Grade FROM student LEFT OUT JOIN SC ON (student.Sno = SC.Sno)	db.student.find({ scores: { $elemMatch: { $gt: 95, $lt: 99 } } })

find函数介绍及使用

db.集合名.find(query,fields,limit,skip)

主要有四个参数

1、query 指明查询条件，相当于SQL中的where语句

例如：db.student.find({name:”joe”,age:{$lt:22}})

2、fields 用于字段映射，指定是否返回该字段，0代表不返回，1代表返回，语法格式：{field:0}或{field:1}

例如：db.student.find({“age”:{$lt:22}},{“_id”:0，”name”:1})

```
db.student.find({"age":{$lt:22}},{"age":0,"name":1})
db.student.find({"age":{$lt:22}},{"_id":1,"name":0})
```

find函数介绍及使用

db.集合名.find(query,fields,limit,skip)

主要有四个参数

3、limit：限制查询结果集的文档数量，指定查询返回结果数量的上限

例如：db.student.find({name:”joe”},{“name”:1,”age”:1},5})

4、skip：跳过一定数据量的结果，设置第一条返回文档的偏移量

例如：db.student.find({name:”joe”},{“name”:1,”age”:1},5,20})

返回值为cursor对象

例如：var cursor=db.student.find({name:”joe”})

find函数介绍及使用

使用find函数时需要注意以下问题：

1、MongoDB不支持多集合间的连接查询，find函数一次查询只能针对一个集合

2、find参数为空或者查询条件为空文档时，会返回集合中所以的文档

例如：db.student.find()或db.student.find({})

3、除了将limit和skip作为find函数的参数外，还可以单独使用limit和skip函数来修饰查询结果

例如：db.student.find({age:{$lt:22}}).limit(5).skip(10)

find函数介绍及使用

使用find函数时需要注意以下问题：

4、返回的查询结果集默认是无序的，如果需要对结果进行排序，可以使用sort函数。1表示升序，-1为降序

例如：db.student.find().sort({name:1,age:-1})

5、db.collection.findOne()只会返回第一条数据

6、当查询的集合文档数量很大时，为了加快数据的查询速度可以创建索引

7、除了使用find函数实现基本查询之外，MongoDB还提供了聚合框架，用于复杂查询

本章大纲

- find函数介绍及使用
- 查询操作符介绍及使用
- 内嵌文档与数组查询
- MongoDB的游标
- 模糊查询
- findAndModify函数的使用

查询操作符介绍及使用

MongoDB拥有强大的数据查询功能，这主要靠其提供的查询操作符（Query Operators)来实现，它包括**比较查询操作符、逻辑查询操作符、元素查询操作符、数组查询操作符**等。

查询操作符的使用十分灵活，使用它们能够构造出复杂的查询条件，可以满足大部分的数据查询功能。

查询操作符介绍及使用

比较查询操作符（Comparison Query Operators），用于大小值比较以及包含与不包含关系的判断。

比较操作符	作 用	参 数
$eq / $ne	等于/ 不等于	{ <field>: { $eq: <value> } }
$gt / $gte	大于 / 大于等于	{ <field>: { $gt: <value> } }
$lt / $lte	小于 / 小于等于	{ <field>: { $lt: <value> } }
$in / $nin	包含 / 不包含	{ field: { $in: [<value1>, <value2>, ... <valueN>] } }

查询操作符介绍及使用

逻辑查询操作符（Logical Query Operators），可连接多个查询条件，用于逻辑与、或、非以及取反操作。

逻辑操作符	作 用	参 数
$and	与	{ $and: [{ <expression1> }, { <expression2> } , ... , { <expressionN> }] }
$or	或	{ $or: [{ <expression1> }, { <expression2> }, ... , { <expressionN> }] }
$nor	非	{ $nor: [{ <expression1> }, { <expression2> }, ... { <expressionN> }] }
$not	取反	{ field: { $not: { <operator-expression> } } }

查询操作符介绍及使用

元素查询操作符（Element Query Operators），查询文档中字段的属性，包括字段是否存在以及字段的数据类型。

元素操作符	作 用	参 数
$exists	字段是否存在	{ field: { $exists: <boolean> } }
$type	选择字段值为指定BSON数据类型编号的文档	{ field: { $type: <BSON type> } }

查询操作符介绍及使用

$where操作符功能强大且灵活，它可以将JavaScript表达式的字符串或JavaScript函数作为查询语句的一部分。在JavaScript表达式和函数中，可以使用this或obj来引用当前操作的文档。

例如：db.collection.find({ $where: "this.credits== this.debits" })

JavaScript表达式或函数返回值为true时，才会返回当前的文档。

查询时，$where操作符**不能使用索引**，每个文档需要从BSON对象转换成JavaScript对象后，才可以通过$where表达式来运行。因此，它比常规查询要慢很多，一般情况下，要避免使用$where查询。

本章大纲

- find函数介绍及使用
- 查询操作符介绍及使用
- 内嵌文档与数组查询
- MongoDB的游标
- 模糊查询
- findAndModify函数的使用

内嵌文档与数组查询

内嵌文档查询包括两种情况

- 查询整个内嵌文档

  当内嵌文档键值对的数量以及键值对的顺序都相同时，才会匹配

- 查询文档的某个字段

  需要使用. 号操作符

例如：db.student.find({“address.city”:”Beijing”})

内嵌文档与数组查询

MonogDB中对数组类型的查询主要包括以下几种情况:

- 查询整个数组　要求元素内容和元素顺序必须完全相同
- 查询数组中的元素(普通元素)，包含两种情况:

1、与位置无关，查询数组中含有某个值的元素，

例如:db.student.find({score: 80})

2、与位置有关，按照指定的数组索引查询数组元素的值(使用点号操作符)

例如:db.student.find({'scores.2': 95})

内嵌文档与数组查询

数组元素是文档时，包含两种情况：

1、与位置无关，查询数组中满足条件的子文档元素

例如：db.student.find({“score.成绩”: 80})

2、与位置有关，按照指定的数组索引查询数组子文档

例如：db.student.find({“scores.2.成绩”: 95})

内嵌文档与数组查询

针对数组MongoDB提供了数组操作符，可以实现非常复杂的数组操作，用于条件查询和数组元素的映射。

条件查询	作用	元素映射	作用
$all	查询参数数组中包含指定元素的文档	$	返回第一个匹配的数组元素
$elemMatch	多条件查询	$elemMatch	按条件返回数组元素的子集
$size	查询指定长度的数组	$slice	返回连续数组元素的子集

本章大纲

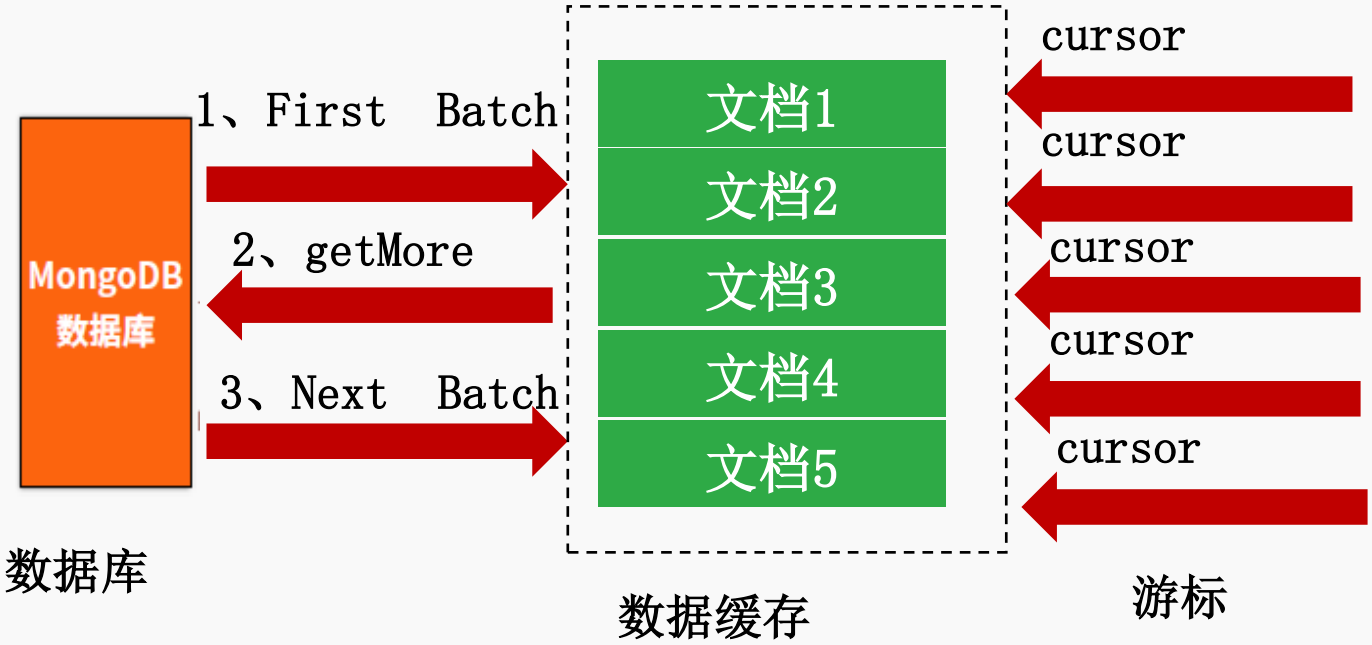
- find函数介绍及使用
- 查询操作符介绍及使用
- 内嵌文档与数组查询
- MongoDB的游标
- 模糊查询
- findAndModify函数的使用

MongoDB的游标

- 查询结果返回过程简介
- MongoDB游标介绍及使用
- 游标的生命周期
- 游标快照介绍及使用

MongoDB的游标

用find函数进行查询时，MongoDB并不是一次返回结果集中的所有文档，而是以多条文档的形式分批来返回查询结果，返回的文档会缓存到内存中。



MongoDB的游标

这种批量返回结果的好处是，一方面，它可以减少客户端对服务器连接次数，从而减轻服务器的查询负担；另一方面，当查询结果集很大时，批量返回结果的方式可以减少客户端的等待时间，提高数据的处理效率。

MongoDB的游标

在MongoDB中，提供了类似关系型数据的游标cursor对象，使用它来遍历结果集中的数据。

MongoDB手册对游标的定义为：”A pointer to the result set of a query .Client can iterate through a cursor to retrieve results.”

可以从两方面看待游标：客户端的游标以及客户端游标表示的数据库游标。

MongoDB的游标

针对游标，MongoDB提供了许多函数，主要包括：

名称	作用	名称	作用
hasNext	判断是否还有更多的文档	sort	对查询结果进行排序
next	用来获取下一条文档	objsLeftInBatch	查看当前批次剩余的未被迭代的文档数量
toArray	将查询结果放到数组中	addOption	为游标设置辅助选项，修改游标的默认行为
count	获取结果集中总的文档数量	hint	为查询强制使用指定索引
limit	限制结果返回数量	explain	用于获取查询执行过程报告
skip	跳过指定数目的文档	snapshot	对查询结果使用快照

MongoDB的游标

使用游标时，需要注意下面几个问题：

1、当调用find函数时，shell并不立即查询数据库，而是**等真正开始获取结果**时才发送查询请求

```
var cursor = db.student.find().sort({age:1}).limit(2).skip(10);
var cursor = db.student.find().limit(2).sort({age:1}).skip(10);
var cursor = db.student.find().skip(10).limit(2).sort({age:1});
```

2、游标对象的每个方法几乎都返回游标对象本身，这样就可以方便的进行链式函数调用

MongoDB的游标

使用游标时，需要注意下面几个问题：

3、MongoDB Shell中使用游标输出文档包含两种情况：

- 自动迭代 如果不将find函数返回的游标赋值给一个局部变量进行保存的话，默认情况下游标会自动迭代20次
- 手动迭代 将find函数返回的游标赋值给一个局部变量，然后使用游标对象提供的函数进行手动迭代

4、使用清空后的游标，进行迭代输出时，显示内容为空

MongoDB的游标

一个游标从创建到被销毁的整个过程存在的时间，称为游标的生命周期，它包括游标的创建、使用以及销毁三个阶段。

当客户端使用find函数向服务器端发起一次查询请求时，会在服务器端创建一个游标，然后就可以使用游标函数来操作查询结果。

下面三种情况会让游标销毁:

- 游标遍历完成后，或者客户端主动发送终止消息
- 客户端保存的游标变量不在作用域内
- 在服务器端10分钟内未对游标进行操作

MongoDB的游标

关于游标的生命周期，需要说明：

- 有时希望游标存在时间长一些，避免这种“超时销毁”的行为，可以使用addOption函数为游标添加noTimeout的选项：

```
var mycurosr =
 db.collection.find().addOption(DBQuery.Option.noTimeout);
```

MongoDB的游标

使用serverStatus()函数可以查看当前系统的游标状态

db.serverStatus().metrics.cursor{

"timedOut":<number>,

"open":{

"noTimeout":<number>,

"pinned":<number>,

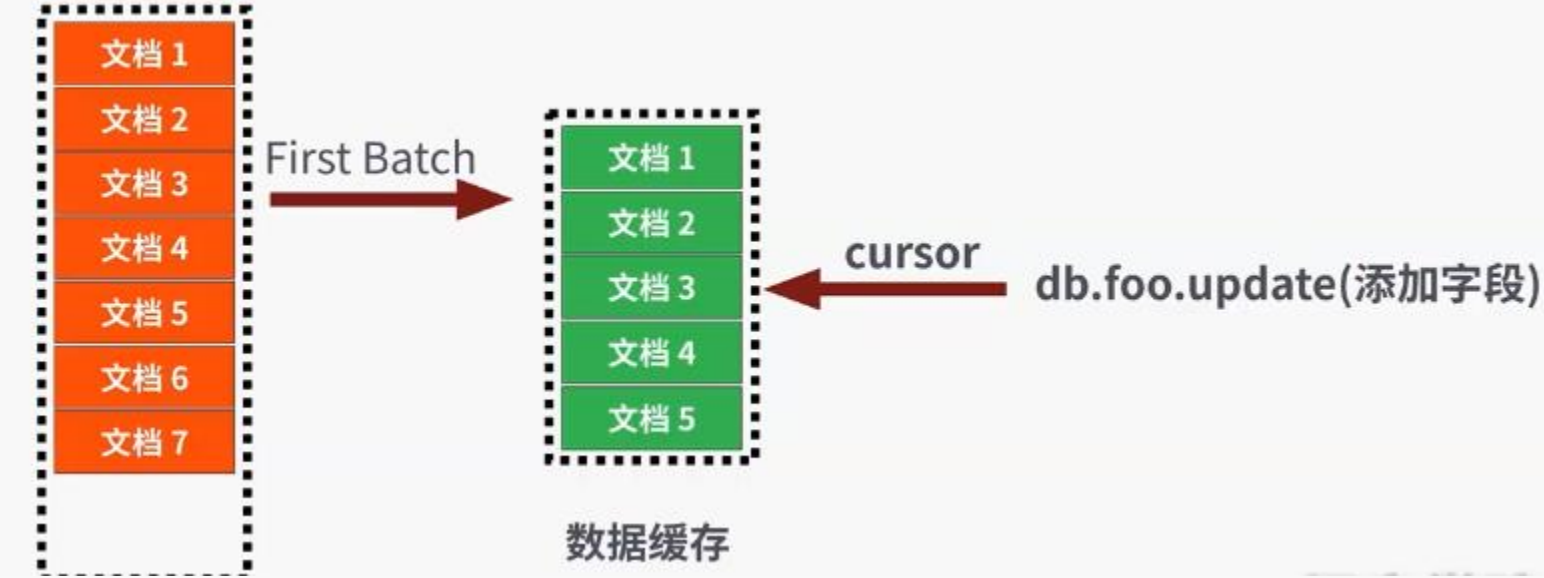
"total":<number>

}}

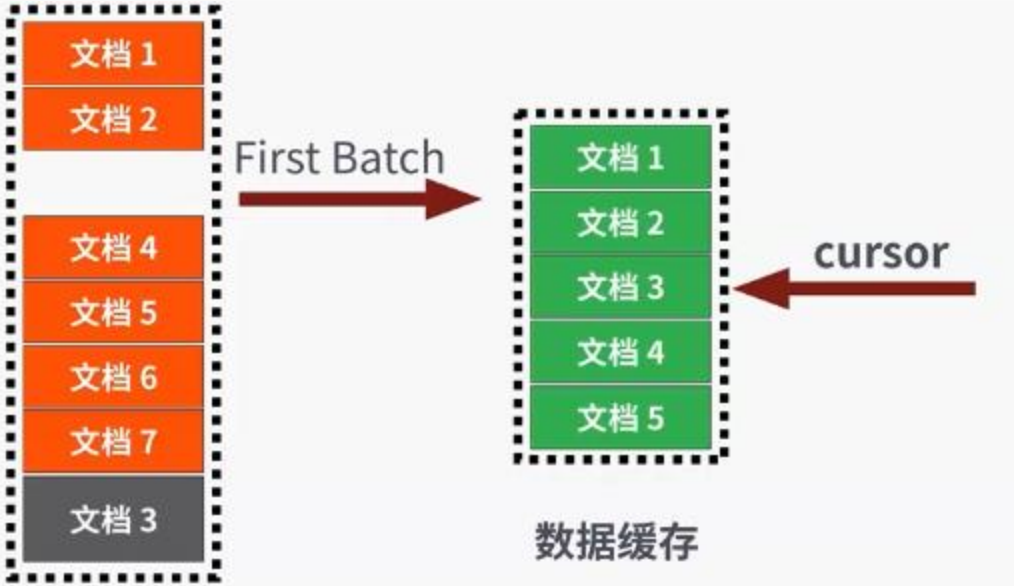
```
> db.serverStatus().metrics.cursor
{
        "timedOut" : NumberLong(0),
        "open" : {
                "noTimeout" : NumberLong(0),
                "pinned" : NumberLong(0),
                "total" : NumberLong(0)
        }
}
>
```

MongoDB的游标

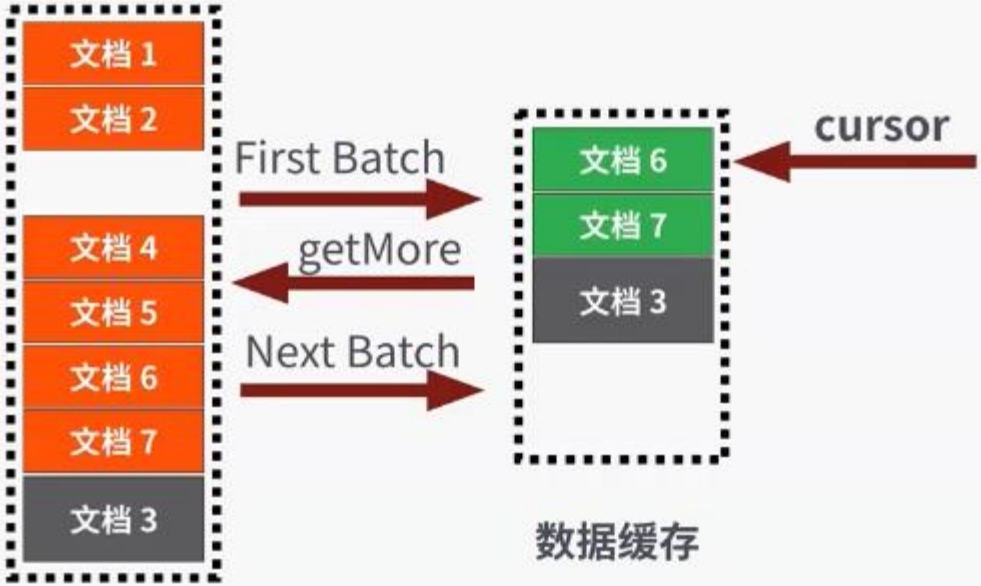
MongoDB的游标在整个生命周期中不具有隔离性（isolated），当查询结果集很大且在查询结果集上执行数据更新操作时，可能会多次返回同一个文档。



MongoDB的游标


文档 1
文档 2
文档 4
文档 5
文档 6
文档 7
文档 3
First Batch
文档 1
文档 2
文档 3
文档 4
文档 5
cursor
数据缓存

MongoDB的游标


文档 1
文档 2
文档 4
文档 5
文档 6
文档 7
文档 3
First Batch
getMore
Next Batch
cursor
文档 6
文档 7
文档 3
数据缓存

MongoDB的游标

游标可能会返回那些由于体积变大而被移动到集合末尾的文档。解决这一问题的方法是对查询进行快照。其语法如下：

db.collection.find().**snapshot()**

使用快照后，查询就会在_id索引上来遍历执行，这样就可以保证每个文档只会被返回一次，从而保证获取结果的一致性。但快照会使得查询速度变慢，使用需谨慎。

本章大纲

- find函数介绍及使用
- 查询操作符介绍及使用
- 内嵌文档与数组查询
- MongoDB的游标
- 模糊查询
- findAndModify函数的使用

模糊查询

MongoDB查询条件可以使用正则表达式，从而实现模糊查询的功能。模糊查询可以使用$regex操作符或直接使用正则表达式对象。

MySQL 数据库	MongoDB
SELECT * FROM student WHERE Sname LIKE '%joe%';	db.student.find ({ Sname: { $regex:/joe/ } })
SELECT * FROM student WHERE Sname REGEXP 'joe';	db.student.find ({ Sname:/joe/ })

模糊查询

MongoDB使用$regex操作符来设置匹配字符串的正则表达式，使用PCRE(Pert Compatible Regular Expression)作为正则表达式语言。

- regex操作符

  {<field>:{$regex:/pattern/，$options:'<options>'}}

  {<field>:{$regex:'pattern'，$options:'<options>'}}

  {<field>:{$regex:/pattern/<options>}}

- 正则表达式对象　　{<field>: /pattern/<options>}

模糊查询

$regex与正则表达式对象的区别:

- 在$in操作符中只能使用正则表达式对象

例如:{name:{$in:[/^joe/i,/^jack/}}

- 在使用隐式的$and操作符中，只能使用$regex

例如:{name:{$regex:/^jo/i, $nin:['john']}}

- 当option选项中包含x或s选项时，只能使用$regex

例如:{name:{$regex:/m.*line/,$options:"si"}}

模糊查询

$regex操作符中的option选项可以改变正则匹配的默认行为，它包括i，m，x以及s四个选项，其含义如下：

- i　忽略大小写　{<field>{$regex/pattern/i}}

  设置i选项后，模式中的字母会进行大小写不敏感匹配。

- m　多行匹配模式，{<field>{$regex/pattern/,$options:'m'}

  m选项会更改^和$元字符的默认行为，分别使用与行的开头和结尾匹配，而不是与输入字符串的开头和结尾匹配

模糊查询

- x 忽略非转义的空白字符 <field>:{$regex:/pattern/,$options:'m'}

设置x选项后，正则表达式中的非转义的空白字符将被忽略，同时#被解释为注释的开头注，**只能显式位于option选项中**

- s 单行匹配模式{<field>:{$regex:/pattern/,$options:'s'}

设置s选项后，会改变模式中的点号(.)元字符的默认行为，它会匹配所有字符，包括换行符(\n)，**只能显式位于option选项中**。

模糊查询

使用$regex操作符时，需要注意下面几个问题:

- i，m，x，s可以组合使用

  例如:{name:{$regex:/j*k/,$options:"si"}}

- 在设置索引的字段上进行正则匹配可以提高查询速度，而且当正则表达式使用的是前缀表达式时，查询速度会进一步提高

  例如:{name:{$regex: /^joe/}

本章大纲

- find函数介绍及使用
- 查询操作符介绍及使用
- 内嵌文档与数组查询
- MongoDB的游标
- 模糊查询
- findAndModify函数的使用

findAndModify函数的使用

findAndModify执行分为find和update两步，属于get-and-set式的操作，它的功能强大之处在于可以保证操作的原子性。

findAndModify对于操作查询以及执行其它需要取值和赋值风格的原子性操作是十分方便的，使用它可以实现一些简单的类事务操作。

findAndModify函数的使用

findAndModify函数有七个参数，每个参数含义如下表所示：

名称	作用	名称	作用
query: <document>	查询文档，用来指明要更新的条件	new: <boolean>	返回更新前的文档还是更新后的文档
sort: <document>	对查询结果排序，默认采用升序	fields: <document>	投影，设置返回文档需要显示的字段
remove: <boolean>	布尔类型，表示是否删除文档	upsert: <boolean>	与 update 函数的 upsert参数含义相同
update: <document>	对满足 query 条件的文档进行更新		

findAndModify函数的使用

findAndModify与update函数的比较:

- 默认情况下，两个操作都只能修改一个文档,update函数可以通过{multi:true}选项一次修改多条文档
- 都以原子的方式来更新修改文档
- 当多个文档满足query条件时:
  1、findAndModify使用sort选项，对结果排序，选择第一个文档
  2、update不能选择具体更新哪一个文档
- 返回值不同 findAndModify是具体文档update是WriteResult对象
- findAndModify无法指定writeConcern函数

findAndModify函数的使用

主键自动增长可以使向数据库添加数据时，不考虑主键的取值，记录插入后，数据库会自动为其分配一个值，确保绝对不会出现重复。

大部分的关系型数据库都提供了主键自增的功能，例如，MySQL数据库可以使用AUTO_INCREMENT来很容易的实现字段自增。

```
CREATE TABLE test(id INT AUTO_INCREMENT);
```

由于MongoDB是为分布式存储而设计，_id主键默认使用的是ObjectId类型的值，它比自增式主键更适合在分布式环境下使用，所以MongoDB默认不支持字段自增功能。

findAndModify函数的使用

实现自增主键的功能，需要完成三个操作:

- 取得counters集合的当前最大_id值
- 对counters集合的id值加1
- 修改counters集合的原_id值

为防止多个客户端同时修改_id值，需要保证上面的三个操作以原子的方式进行自增。

利用findAndModify函数的get-and-set的原子特性，来实现_id的自增。