# [MongoDB文档、集合、数据库的概念](https://www.cnblogs.com/wenxudong/articles/6232000.html)

**基本概念**

文档是MongoDB中数据的基本单元，是MongoDB的核心概念，很类似关系数据库中的行（记录） 文档就是一行一行数据的概念

集合可以被看作没有模式的表（表） 集合就是表（collections）

MongoDB每个实例都可容纳多个独立数据库，每个数据库都有自己的集合和权限（数据库）

# MongoDB 创建数据库

### 语法

MongoDB 创建数据库的语法格式如下：

use DATABASE\_NAME

如果数据库不存在，则创建数据库，否则切换到指定数据库。

### 实例

以下实例我们创建了数据库 runoob:

> use runoob

switched to db runoob

> db

runoob

>

如果你想查看所有数据库，可以使用 **show dbs** 命令：

> show dbs

admin 0.000GB

local 0.000GB

>

可以看到，我们刚创建的数据库 runoob 并不在数据库的列表中， 要显示它，我们需要向 runoob 数据库插入一些数据。

> db.runoob.insert({"name":"xxx"})

WriteResult({ "nInserted" : 1 })

> show dbs

local 0.078GB

runoob 0.078GB

test 0.078GB

>

MongoDB 中默认的数据库为 test，如果你没有创建新的数据库，集合将存放在 test 数据库中。

# MongoDB 创建集合

本章节我们为大家介绍如何使用 MongoDB 来创建集合。

MongoDB 中使用 **createCollection()** 方法来创建集合。

语法格式：

db.createCollection(name, options)

参数说明：

* name: 要创建的集合名称
* options: 可选参数, 指定有关内存大小及索引的选项

options 可以是如下参数：

| **字段** | **类型** | **描述** |
| --- | --- | --- |
| capped | 布尔 | （可选）如果为 true，则创建固定集合。固定集合是指有着固定大小的集合，当达到最大值时，它会自动覆盖最早的文档。 **当该值为 true 时，必须指定 size 参数。** |
| autoIndexId | 布尔 | （可选）如为 true，自动在 \_id 字段创建索引。默认为 false。 |
| size | 数值 | （可选）为固定集合指定一个最大值（以字节计）。 **如果 capped 为 true，也需要指定该字段。** |
| max | 数值 | （可选）指定固定集合中包含文档的最大数量。 |

在插入文档时，MongoDB 首先检查固定集合的 size 字段，然后检查 max 字段。

### 实例

在 test 数据库中创建 runoob 集合：

> use test

switched to db test

> db.createCollection("runoob")

{ "ok" : 1 }

>

如果要查看已有集合，可以使用 show collections 命令：

> show collections

runoob

system.indexes

在 MongoDB 中，你不需要创建集合。当你插入一些文档时，MongoDB 会自动创建集合。

> db.mycol2.insert({"name" : "菜鸟教程"})

> show collections

mycol2

...

# MongoDB 插入文档

## 插入文档

MongoDB 使用 insert() 或 save() 方法向集合中插入文档，语法如下：

db.COLLECTION\_NAME.insert(document)

### 实例

以下文档可以存储在 MongoDB 的 runoob 数据库 的 col 集合中：

>db.col.insert({title: 'MongoDB 教程',

description: 'MongoDB 是一个 Nosql 数据库',

by: '菜鸟教程',

url: 'http://www.runoob.com',

tags: ['mongodb', 'database', 'NoSQL'],

likes: 100

})

3.2 版本后还有以下几种语法可用于插入文档:

* db.collection.insertOne():向指定集合中插入一条文档数据
* db.collection.insertMany():向指定集合中插入多条文档数据

# 插入单条数据

> var document = db.collection.insertOne({"a": 3})

> document

{

"acknowledged" : true,

"insertedId" : ObjectId("571a218011a82a1d94c02333")

}

# 插入多条数据

> var res = db.collection.insertMany([{"b": 3}, {'c': 4}]) //封装成数组

> res

{

"acknowledged" : true,

"insertedIds" : [

ObjectId("571a22a911a82a1d94c02337"),

ObjectId("571a22a911a82a1d94c02338")

]

}

# update()方法

update() 方法用于更新已存在的文档。语法格式如下：

db.collection.update(

<query>,

<update>,

{

upsert: <boolean>,

multi: <boolean>,

writeConcern: <document>

}

)

**参数说明：**

* **query**: update的查询条件，类似sql update查询内where后面的。
* **update**: update的对象和一些更新的操作符（如$,$inc...）等，也可以理解为sql update查询内set后面的
* **upsert**: 可选，这个参数的意思是，如果不存在update的记录，是否插入objNew,true为插入，默认是false，不插入。
* **multi**: 可选，mongodb 默认是false,只更新找到的第一条记录，如果这个参数为true,就把按条件查出来多条记录全部更新。
* **writeConcern**:可选，抛出异常的级别。

对应的例子如下:

db.col.update({'title':'MongoDB 教程'},{$set:{'title':'MongoDB'}},{multi:true})

# save ()方法

save() 方法通过传入的文档来替换已有文档。语法格式如下：

db.collection.save(

<document>,

{

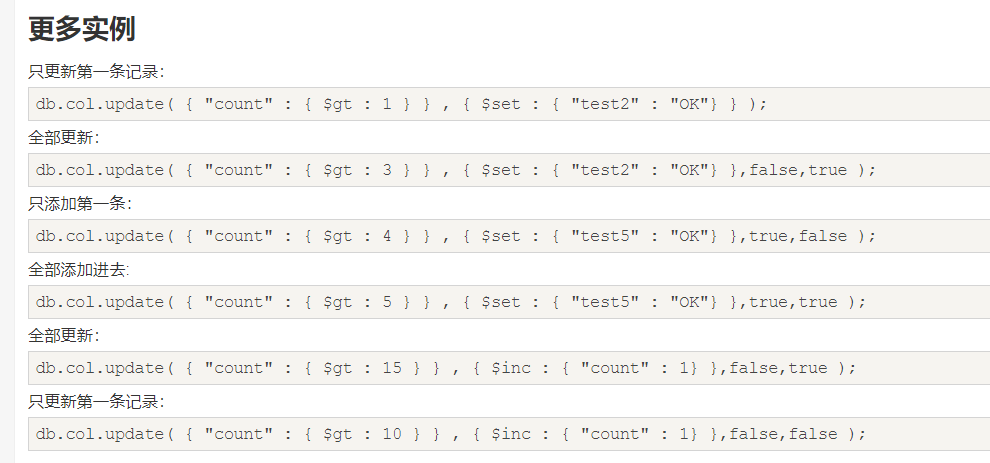
writeConcern: <document>

}

)

**参数说明：**

* **document**: 文档数据。
* **writeConcern**:可选，抛出异常的级别。



如上信息updata额外的操作。

重要补充:

在3.2版本开始，MongoDB提供以下更新集合文档的方法：

* db.collection.updateOne() 向指定集合更新单个文档
* db.collection.updateMany() 向指定集合更新多个文档

首先我们在test集合里插入测试数据

use test

db.test\_collection.insert( [

{"name":"abc","age":"25","status":"zxc"},

{"name":"dec","age":"19","status":"qwe"},

{"name":"asd","age":"30","status":"nmn"},

] )

更新单个文档

> db.test\_collection.updateOne({"name":"abc"},{$set:{"age":"28"}}) //对指定的name更新指定的数值 类似SQL: set age = 28 where name = “abc”

{ "acknowledged" : true, "matchedCount" : 1, "modifiedCount" : 1 }

> db.test\_collection.find()

{ "\_id" : ObjectId("59c8ba673b92ae498a5716af"), "name" : "abc", "age" : "28", "status" : "zxc" }

{ "\_id" : ObjectId("59c8ba673b92ae498a5716b0"), "name" : "dec", "age" : "19", "status" : "qwe" }

{ "\_id" : ObjectId("59c8ba673b92ae498a5716b1"), "name" : "asd", "age" : "30", "status" : "nmn" }

>

更新多个文档

> db.test\_collection.updateMany({"age":{$gt:"10"}},{$set:{"status":"xyz"}})

类似SQL:set status = “xyz” where age > 10

{ "acknowledged" : true, "matchedCount" : 3, "modifiedCount" : 3 }

> db.test\_collection.find()

{ "\_id" : ObjectId("59c8ba673b92ae498a5716af"), "name" : "abc", "age" : "28", "status" : "xyz" }

{ "\_id" : ObjectId("59c8ba673b92ae498a5716b0"), "name" : "dec", "age" : "19", "status" : "xyz" }

{ "\_id" : ObjectId("59c8ba673b92ae498a5716b1"), "name" : "asd", "age" : "30", "status" : "xyz" }

# remove()方法

remove() 方法已经过时了，现在官方推荐使用 deleteOne() 和 deleteMany() 方法。

如删除集合下全部文档：

db.inventory.deleteMany({})

删除 status 等于 A 的全部文档：

db.inventory.deleteMany({ status : "A" })

删除 status 等于 D 的一个文档：

db.inventory.deleteOne( { status: "D" } )

果然remove()方法已经过时了  
  
deleteMany()是把能匹配到这个字段的对象全找到并且全删了  
  
deleteOne()是吧匹配到的单个对象删了，至于顺序。。。有待考证

# find () 方法 MongoDB 查询文档

MongoDB 查询文档使用 find() 方法。

find() 方法以非结构化的方式来显示所有文档。

### 语法

MongoDB 查询数据的语法格式如下：

db.collection.find(query, projection) //第一个{}里面就是条件，第二个是过滤选项

* **query** ：可选，使用查询操作符指定查询条件
* **projection** ：可选，使用投影操作符指定返回的键。查询时返回文档中所有键值， 只需省略该参数即可（默认省略）。

如果你需要以易读的方式来读取数据，可以使用 pretty() 方法，语法格式如下：

>db.col.find().pretty()

pretty() 方法以格式化的方式来显示所有文档。

工具表:



## find():AND 条件

MongoDB 的 find() 方法可以传入多个键(key)，每个键(key)以逗号隔开，即常规 SQL 的 AND 条件。

语法格式如下：

>db.col.find({key1:value1, key2:value2}).pretty()

列如:

db.col.find({"by":"菜鸟教程", "title":"MongoDB 教程"}).pretty()

类似Sql:

**WHERE by='菜鸟教程' AND title='MongoDB 教程'**

## find():OR 条件 (要用中括号括起来)

MongoDB OR 条件语句使用了关键字 **$or**,语法格式如下：

>db.col.find(

{

$or: [

     {key1: value1}, {key2:value2}

]

}

).pretty()

## find():AND 和 OR 联合使用

以下实例演示了 AND 和 OR 联合使用，类似常规 SQL 语句为：

**'where likes>50 AND (by = '菜鸟教程' OR title = 'MongoDB 教程')'**

>db.col.find

({"likes": {$gt:50}, $or: [{"by": "菜鸟教程"},{"title": "MongoDB 教程"}]}).pretty()

注意or用了中括号[ ]

## find():projection 参数的使用方法

**1.db.collection.find(query, projection)**

若不指定 projection，则默认返回所有键，指定 projection 格式如下，有两种模式

db.collection.find(query, {title: 1, by: 1}) // inclusion模式 指定返回的键，不返回其他键

db.collection.find(query, {title: 0, by: 0}) // exclusion模式 指定不返回的键,返回其他键

\_id 键默认返回，需要主动指定 \_id:0 才会隐藏

两种模式不可混用（因为这样的话无法推断其他键是否应返回）

db.collection.find(query, {title: 1, by: 0}) // 错误

只能全1或全0，除了在inclusion模式时可以指定\_id为0，就是\_id是可以设置为0的不并入考虑范围

db.collection.find(query, {\_id:0, title: 1, by: 1}) // 正确

理解的话就是选择查询的是需要返回的还是不需要返回的内容。

这个上面就是又查title又查by

除了 find() 方法之外，还有一个 findOne() 方法，它只返回一个文档。是一样的

2.

若不想指定查询条件参数 **query** 可以 用 **{}** 代替，但是需要指定 **projection** 参数：

querydb.collection.find({}, {title: 1})

3.

如果是 qty 大于 50 小于 80 不能这样写：

db.posts.find( { qty: { $gt: 50 }, qty: { $lt: 80 } } )

应该这样：

db.posts.find( { qty: { $gt: 50 ,$lt: 80}} )

# MongoDB 条件操作符

## 1.MongoDB (>) 大于操作符 - $gt

如果你想获取 "col" 集合中 "likes" 大于 100 的数据，你可以使用以下命令：

db.col.find({likes : {$gt : 100}})

类似于SQL语句：

Select \* from col where likes > 100;

## 2.MongoDB（>=）大于等于操作符 - $gte

如果你想获取"col"集合中 "likes" 大于等于 100 的数据，你可以使用以下命令：

db.col.find({likes : {$gte : 100}})

类似于SQL语句：

Select \* from col where likes >=100;

## 3.MongoDB (<) 小于操作符 - $lt

如果你想获取"col"集合中 "likes" 小于 150 的数据，你可以使用以下命令：

db.col.find({likes : {$lt : 150}})

类似于SQL语句：

Select \* from col where likes < 150;

## 4.MongoDB (<=) 小于操作符 - $lte

如果你想获取"col"集合中 "likes" 小于等于 150 的数据，你可以使用以下命令：

db.col.find({likes : {$lte : 150}})

类似于SQL语句：

Select \* from col where likes <= 150;

## 5 .MongoDB 使用 (<) 和 (>) 查询 - $lt 和 $gt

如果你想获取"col"集合中 "likes" 大于100，小于 200 的数据，你可以使用以下命令：

db.col.find({likes : {$lt :200, $gt : 100}})

类似于SQL语句：

Select \* from col where likes>100 AND likes<200;

总结:

$gt -------- greater than >

$gte --------- gt equal >=

$lt -------- less than <

$lte --------- lt equal <=

$ne ----------- not equal !=

$eq -------- equal =

## 6.模糊查询

查询 title 包含"教"字的文档：

db.col.find({title:/教/})

查询 title 字段以"教"字开头的文档：

db.col.find({title:/^教/})

查询 titl e字段以"教"字结尾的文档：

db.col.find({title:/教$/})

## 7.type查询: MongoDB 操作符 - $type 实例

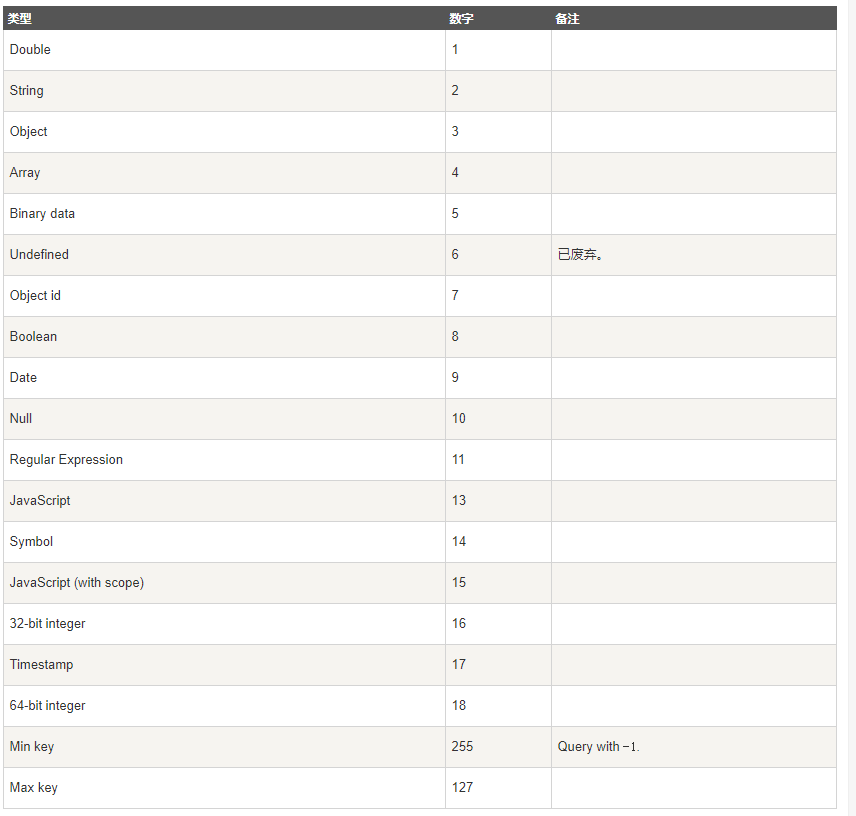
如果想获取 "col" 集合中 title 为 String 的数据，你可以使用以下命令：

db.col.find({"title" : {$type : 2}})

或

db.col.find({"title" : {$type : 'string'}})

可以支持的type类型:



# MongoDB Limit与Skip方法

如果你需要在MongoDB中读取指定数量的数据记录，可以使用MongoDB的Limit方法，limit()方法接受一个数字参数，该参数指定从MongoDB中读取的记录条数。

### MongoDB Limit语法

limit()方法基本语法如下所示：

>db.COLLECTION\_NAME.find().limit(NUMBER)

### 实例

以下实例为显示查询文档中的两条记录：

> db.col.find({},{"title":1,\_id:0}).limit(2)

{ "title" : "PHP 教程" }

{ "title" : "Java 教程" }

补充说明：

* 第一个 {} 放 where 条件，为空表示返回集合中所有文档。
* 第二个 {} 指定那些列显示和不显示 （0表示不显示 1表示显示)。
* 这个上面{“title”:1,\_id:0}就是表示只查title，不显示默认的\_id.

## MongoDB Skip() 方法

我们除了可以使用limit()方法来读取指定数量的数据外，还可以使用skip()方法来跳过指定数量的数据，skip方法同样接受一个数字参数作为跳过的记录条数。

### 语法

skip() 方法脚本语法格式如下：

>db.COLLECTION\_NAME.find().limit(NUMBER).skip(NUMBER)

### 实例

以下实例只会显示第二条文档数据

>db.col.find({},{"title":1,\_id:0}).limit(1).skip(1)

{ "title" : "Java 教程" }

>

理解起来应该蛮简单，就是一个函数链式调用。

# MongoDB 实现分页的方法limit()+skip()

想要读取从 10 条记录后 100 条记录，相当于 sql 中limit (10,100)。

> db.COLLECTION\_NAME.find().skip(10).limit(100)

以上实例在集合中跳过前面 10 条返回 100 条数据。

skip 和 limit 结合就能实现分页。

1.补充说明:

可以联系邮箱 **libaoyuan\_87@126.com**，互相学习。

## 补充说明：

skip和limit方法只适合小数据量分页，如果是百万级效率就会非常低，因为skip方法是一条条数据数过去的，建议使用where\_limit

在查看了一些资料之后，发现所有的资料都是这样说的：

不要轻易使用Skip来做查询，否则数据量大了就会导致性能急剧下降，这是因为Skip是一条一条的数过来的，多了自然就慢了。

这么说Skip就要避免使用了，那么如何避免呢？首先来回顾SQL分页的后一种时间戳分页方案，这种利用字段的有序性质，利用查询来取数据的方式，可以直接避免掉了大量的数数。也就是说，如果能附带上这样的条件那查询效率就会提高，事实上是这样的么？我们来验证一下：

这里我们假设查询第100001条数据，这条数据的Amount值是：2399927，我们来写两条语句分别如下：

b.test.sort({"amount":1}).skip(100000).limit(10) //183ms

db.test.find({amount:{$gt:2399927}}).sort({"amount":1}).limit(10) //53ms

结果已经附带到注释了，很明显后者的性能是前者的三分之一，差距是非常大的。也印证了Skip效率差的理论。

# MongoDB sort() 排序方法

在 MongoDB 中使用 sort() 方法对数据进行排序，sort() 方法可以通过参数指定排序的字段，并使用 1 和 -1 来指定排序的方式，其中 1 为升序排列，而 -1 是用于降序排列。

## 语法

sort()方法基本语法如下所示：

>db.COLLECTION\_NAME.find().sort({KEY:1})

## 实例

实例 以下实例演示了 col 集合中的数据按字段 likes 的降序排列：

>db.col.find({},{"title":1,\_id:0}).sort({"likes":-1})

{ "title" : "PHP 教程" }

{ "title" : "Java 教程" }

{ "title" : "MongoDB 教程" }

>

# sort()->skip()->limit()顺序

skip(), limilt(), sort()三个放在一起执行的时候，执行的顺序是先 sort(), 然后是 skip()，最后是显示的 limit()。

# MongoDB 索引

索引通常能够极大的提高查询的效率，如果没有索引，MongoDB在读取数据时必须扫描集合中的每个文件并选取那些符合查询条件的记录。

这种扫描全集合的查询效率是非常低的，特别在处理大量的数据时，查询可以要花费几十秒甚至几分钟，这对网站的性能是非常致命的。

索引是特殊的数据结构，索引存储在一个易于遍历读取的数据集合中，索引是对数据库表中一列或多列的值进行排序的一种结构

### 语法

createIndex()方法基本语法格式如下所示：

>db.collection.createIndex(keys, options)

语法中 Key 值为你要创建的索引字段，1 为指定按升序创建索引，如果你想按降序来创建索引指定为 -1 即可。

### 实例

>db.col.createIndex({"title":1})

>

createIndex() 方法中你也可以设置使用多个字段创建索引（关系型数据库中称作复合索引）。

>db.col.createIndex({"title":1,"description":-1})

>



### 补充说明

1、查看集合索引

db.col.getIndexes()

2、查看集合索引大小

db.col.totalIndexSize()

3、删除集合所有索引

db.col.dropIndexes()

4、删除集合指定索引

db.col.dropIndex("索引名称")

# MongoDB 聚合（count(\*)）与管道

MongoDB中聚合(aggregate)主要用于处理数据(诸如统计平均值,求和等)，并返回计算后的数据结果。有点类似sql语句中的 count(\*)。

## aggregate() 方法

MongoDB中聚合的方法使用aggregate()。

### 语法

aggregate() 方法的基本语法格式如下所示：

>db.COLLECTION\_NAME.aggregate(AGGREGATE\_OPERATION)

通过以上集合计算每个作者所写的文章数，使用aggregate()计算结果如下：

> db.mycol.aggregate([{$group : {\_id : "$by\_user", num\_tutorial : {$sum : 1}}}])

{

"result" : [

{

"\_id" : "runoob.com",

"num\_tutorial" : 2

},

{

"\_id" : "Neo4j",

"num\_tutorial" : 1

}

],

"ok" : 1

}

>

以上实例类似sql语句：

select by\_user, count(\*) from mycol group by by\_user

### 高级函数（sum,max…）



## 关于管道

管道在Unix和Linux中一般用于将当前命令的输出结果作为下一个命令的参数。

MongoDB的聚合管道将MongoDB文档在一个管道处理完毕后将结果传递给下一个管道处理。管道操作是可以重复的。

表达式：处理输入文档并输出。表达式是无状态的，只能用于计算当前聚合管道的文档，不能处理其它的文档。

这里我们介绍一下聚合框架中常用的几个操作：

* $project：修改输入文档的结构。可以用来重命名、增加或删除域，也可以用于创建计算结果以及嵌套文档。
* $match：用于过滤数据，只输出符合条件的文档。$match使用MongoDB的标准查询操作。
* $limit：用来限制MongoDB聚合管道返回的文档数。
* $skip：在聚合管道中跳过指定数量的文档，并返回余下的文档。
* $unwind：将文档中的某一个数组类型字段拆分成多条，每条包含数组中的一个值。
* $group：将集合中的文档分组，可用于统计结果。
* $sort：将输入文档排序后输出。
* $geoNear：输出接近某一地理位置的有序文档。

### 管道操作符实例

1、$project实例

db.article.aggregate(

{ $project : {

title : 1 ,

author : 1 ,

}}

);

这样的话结果中就只还有\_id,tilte和author三个字段了，默认情况下\_id字段是被包含的，如果要想不包含\_id话可以这样:

db.article.aggregate(

{ $project : {

\_id : 0 ,

title : 1 ,

author : 1

}});

2.$match实例

db.articles.aggregate( [

{ $match : { score : { $gt : 70, $lte : 90 } } },

{ $group: { \_id: null, count: { $sum: 1 } } }

] );

$match用于获取分数大于70小于或等于90记录，然后将符合条件的记录送到下一阶段$group管道操作符进行处理。

3.$skip实例

db.article.aggregate(

{ $skip : 5 });

经过$skip管道操作符处理后，前五个文档被"过滤"掉。

# MongoDB高级查询

1. db.mycol.aggregate([{$group : {\_id : "$by\_user", num\_tutorial : {$sum : 1}}}])

以上实例类似sql语句：

select by\_user as \_id, count(\*) as num\_tutorial from mycol group by by\_user

1. tatoo

  c0d\*\*\*@163.com

db.articles.aggregate({

$project : {

title: 2,

by\_user: 1,

}

})

这样子也是可行的。也就是说非0也可以进行表示显示该字段,负数也可以表示显示该字段。

1. zoujq

按日、按月、按年、按周、按小时、按分钟聚合操作如下：

db.getCollection('m\_msg\_tb').aggregate(

[

{$match:{m\_id:10001,mark\_time:{$gt:new Date(2017,8,0)}}},

{$group: {

\_id: {$dayOfMonth:'$mark\_time'},

pv: {$sum: 1}

}

},

{$sort: {"\_id": 1}}

])

时间关键字如下：

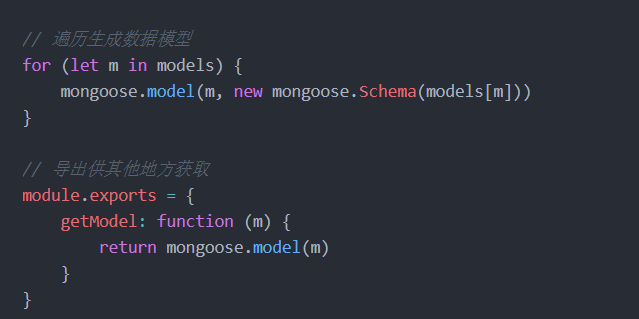
* + $dayOfYear: 返回该日期是这一年的第几天（全年 366 天）。
  + $dayOfMonth: 返回该日期是这一个月的第几天（1到31）。
  + $dayOfWeek: 返回的是这个周的星期几（1：星期日，7：星期六）。
  + $year: 返回该日期的年份部分。
  + $month： 返回该日期的月份部分（ 1 到 12）。
  + $week： 返回该日期是所在年的第几个星期（ 0 到 53）。
  + $hour： 返回该日期的小时部分。
  + $minute: 返回该日期的分钟部分。
  + $second: 返回该日期的秒部分（以0到59之间的数字形式返回日期的第二部分，但可以是60来计算闰秒）。
  + $millisecond：返回该日期的毫秒部分（ 0 到 999）。
  + $dateToString： { $dateToString: { format: , date: } }。

# 实操截图！！！！！

## 模型层jsdbc



承接上面没截完整的。

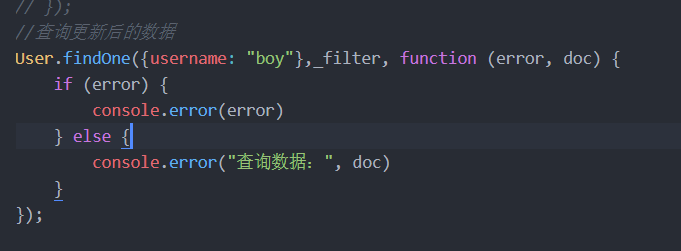


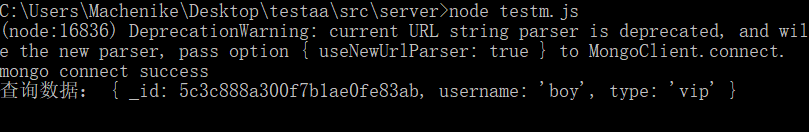
## 调用层



### filter操作（projection）

加了filter之后：



成功过滤了。

### 查询删除操作



其余的都差不多了，就是调用的函数不同而已，

Mongooes的操作就是在基本的查询语句后面多了一个回调函数而已，没什么新鲜的。

要实现查询的话去看看收藏的那个文档就ok

：<https://blog.csdn.net/u012810020/article/details/54582051>

### 更新查询操作

