

#创建数据库 use databasename

#删除数据库，db.dropDataba

#创建 tables/collections

db.creatCollection(name,para)

#删除 tables/collections

show tables

db.tables.drop()

#1,链接

mongo username:password@hostname/database

#2,show dbs

show dbs

#4,switch db

use db'name

#5,create new db

use db'name

#6,delete db(pyaglo)

use paylgo

db.dropDatabase()

#7,delete collection.name(site)

db.site.drop()

#8,show tables

show tables

#9,db.collection\_name.insert(document)

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

insertOne 文档就是字典

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

insertMany就是list

#10,update col ,title,

第一个参数是全部插入，第二个参数是全部更新

更新第一条记录：

db.col.update( { "count" : { $gt : 1 } } , { $set : { "test2" : "OK"} } );

全部更新：

db.col.update( { "count" : { $gt : 3 } } , { $set : { "test2" : "OK"} },false,true );

只添加第一条：

db.col.update( { "count" : { $gt : 4 } } , { $set : { "test5" : "OK"} },true,false );

全部添加加进去:

db.col.update( { "count" : { $gt : 5 } } , { $set : { "test5" : "OK"} },true,true );

#11,show dbdate

db.col.find().pretty()

#12, save

db.col.save({

"\_id" : ObjectId("56064f89ade2f21f36b03136"),

"title" : "MongoDB",

"description" : "MongoDB 是一个 Nosql 数据库",

"by" : "Runoob",

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

"tags" : [

"mongodb",

"NoSQL"

],

"likes" : 110

})

#13,delete doc

## delete one

db.COLLECTION\_NAME.remove(DELETION\_CRITERIA,1)

## delete all

db.COLLECTION\_NAME.remove(DELETION\_CRITERIA)

#14, find

db.collection.find(query, projection)

##

'''

等于 {<key>:<value>} db.col.find({"by":"菜鸟教程"}).pretty() where by = '菜鸟教程'

小于 {<key>:{$lt:<value>}} db.col.find({"likes":{$lt:50}}).pretty() where likes < 50

小于或等于 {<key>:{$lte:<value>}} db.col.find({"likes":{$lte:50}}).pretty() where likes <= 50

大于 {<key>:{$gt:<value>}} db.col.find({"likes":{$gt:50}}).pretty() where likes > 50

大于或等于 {<key>:{$gte:<value>}} db.col.find({"likes":{$gte:50}}).pretty() where likes >= 50

不等于 {<key>:{$ne:<value>}} db.col.find({"likes":{$ne:50}}).pretty() where likes != 50

'''

## and

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

## 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()

#15, delete collection.doc

db.col.remove({})

#16,type

Double 1

String 2

Object 3

Array 4

Binary data 5

Undefined 6 已废弃。

Object id 7

Boolean 8

Date 9

Null 10

Regular Expression 11

JavaScript 13

Symbol 14

JavaScript 15

32-bit integer 16

Timestamp 17

64-bit integer 18

Min key 255 Query with -1.

Max key 127

#17 limit or skip

##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。

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

##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。

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

#18, sort

db.COLLECTION\_NAME.find().sort({'key':'value'})

##19,MongoDB使用 ensureIndex() 方法来创建索引

db.COLLECTION\_NAME.ensureIndex({KEY:1})

#20,MongoDB中聚合的方法使用aggregate()

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

#22 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>

}

)

* **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。

**实例**

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

})

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

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

**更新**

**更多实例**

只更新第一条记录：

db.col.update( { "count" : { $gt : 1 } } , { $set : { "test2" : "OK"} } );

全部更新：

db.col.update( { "count" : { $gt : 3 } } , { $set : { "test2" : "OK"} },false,true );

只添加第一条：

db.col.update( { "count" : { $gt : 4 } } , { $set : { "test5" : "OK"} },true,false );

全部添加加进去:

db.col.update( { "count" : { $gt : 5 } } , { $set : { "test5" : "OK"} },true,true );

全部更新：

db.col.update( { "count" : { $gt : 15 } } , { $inc : { "count" : 1} },false,true );

只更新第一条记录：

db.col.update( { "count" : { $gt : 10 } } , { $inc : { "count" : 1} },false,false );

删除文档：

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" } )

查询

| **操作** | **格式** | **范例** | **RDBMS中的类似语句** |
| --- | --- | --- | --- |
| 等于 | {<key>:<value>} | db.col.find({"by":"菜鸟教程"}).pretty() | where by = '菜鸟教程' |
| 小于 | {<key>:{$lt:<value>}} | db.col.find({"likes":{$lt:50}}).pretty() | where likes < 50 |
| 小于或等于 | {<key>:{$lte:<value>}} | db.col.find({"likes":{$lte:50}}).pretty() | where likes <= 50 |
| 大于 | {<key>:{$gt:<value>}} | db.col.find({"likes":{$gt:50}}).pretty() | where likes > 50 |
| 大于或等于 | {<key>:{$gte:<value>}} | db.col.find({"likes":{$gte:50}}).pretty() | where likes >= 50 |
| 不等于 | {<key>:{$ne:<value>}} | db.col.find({"likes":{$ne:50}}).pretty() | where likes != 50 |

**AND**

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

**OR**

**db.col.find(**

**{**

**$or: [**

**{key1: value1}, {key2:value2}**

**]**

**}**

**).pretty()**

**db.col.find({$or:[{"by":"菜鸟教程"},{"title": "MongoDB 教程"}]}).pretty()**

**AND 和OR**

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

**db.col.find({likes:{$gte:'110'},$or:[{by:'xsj'},{title:'Mong'}]}).pretty()**

**projection**

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

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）

只能全1或全0，除了在inclusion模式时可以指定\_id为0

**MongoDB $type 操作符**

|  |  |  |
| --- | --- | --- |
| **Double** | **1** |  |
| **String** | **2** |  |
| **Object** | **3** |  |
| **Array** | **4** |  |
| **Binary data** | **5** |  |
| **Undefined** | **6** | **已废弃。** |
| **Object id** | **7** |  |
| **Boolean** | **8** |  |
| **Date** | **9** |  |
| **Null** | **10** |  |
| **Regular Expression** | **11** |  |
| **JavaScript** | **13** |  |
| **Symbol** | **14** |  |
| **JavaScript (with scope)** | **15** |  |
| **32-bit integer** | **16** |  |
| **Timestamp** | **17** |  |
| **64-bit integer** | **18** |  |
| **Min key** | **255** | **Query with -1.** |
| **Max key** | **127** |  |

**MongoDB Limit与Skip方法**

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

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

Mongodb sort()

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

**ensureIndex() 方法**

ensureIndex() 接收可选参数，可选参数列表如下：

|  |  |  |
| --- | --- | --- |
| **Parameter** | **Type** | **Description** |
| background | Boolean | 建索引过程会阻塞其它数据库操作，background可指定以后台方式创建索引，即增加 "background" 可选参数。 "background" 默认值为**false**。 |
| unique | Boolean | 建立的索引是否唯一。指定为true创建唯一索引。默认值为**false**. |
| name | string | 索引的名称。如果未指定，MongoDB的通过连接索引的字段名和排序顺序生成一个索引名称。 |
| dropDups | Boolean | 在建立唯一索引时是否删除重复记录,指定 true 创建唯一索引。默认值为 **false**. |
| sparse | Boolean | 对文档中不存在的字段数据不启用索引；这个参数需要特别注意，如果设置为true的话，在索引字段中不会查询出不包含对应字段的文档.。默认值为 **false**. |
| expireAfterSeconds | integer | 指定一个以秒为单位的数值，完成 TTL设定，设定集合的生存时间。 |
| v | index version | 索引的版本号。默认的索引版本取决于mongod创建索引时运行的版本。 |
| weights | document | 索引权重值，数值在 1 到 99,999 之间，表示该索引相对于其他索引字段的得分权重。 |
| default\_language | string | 对于文本索引，该参数决定了停用词及词干和词器的规则的列表。 默认为英语 |
| language\_override | string | 对于文本索引，该参数指定了包含在文档中的字段名，语言覆盖默认的language，默认值为 language. |

下表展示了一些聚合的表达式:

|  |  |  |
| --- | --- | --- |
| **表达式** | **描述** | **实例** |
| $sum | 计算总和。 | db.mycol.aggregate([{$group : {\_id : "$by\_user", num\_tutorial : {$sum : "$likes"}}}]) |
| $avg | 计算平均值 | db.mycol.aggregate([{$group : {\_id : "$by\_user", num\_tutorial : {$avg : "$likes"}}}]) |
| $min | 获取集合中所有文档对应值得最小值。 | db.mycol.aggregate([{$group : {\_id : "$by\_user", num\_tutorial : {$min : "$likes"}}}]) |
| $max | 获取集合中所有文档对应值得最大值。 | db.mycol.aggregate([{$group : {\_id : "$by\_user", num\_tutorial : {$max : "$likes"}}}]) |
| $push | 在结果文档中插入值到一个数组中。 | db.mycol.aggregate([{$group : {\_id : "$by\_user", url : {$push: "$url"}}}]) |
| $addToSet | 在结果文档中插入值到一个数组中，但不创建副本。 | db.mycol.aggregate([{$group : {\_id : "$by\_user", url : {$addToSet : "$url"}}}]) |
| $first | 根据资源文档的排序获取第一个文档数据。 | db.mycol.aggregate([{$group : {\_id : "$by\_user", first\_url : {$first : "$url"}}}]) |
| $last | 根据资源文档的排序获取最后一个文档数据 | db.mycol.aggregate([{$group : {\_id : "$by\_user", last\_url : {$last : "$url"}}}]) |

Aggregate

Match

**管道的概念**

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。

mongodump -h dbhost -d dbname -o dbdirectory

mongodump 命令可选参数列表如下所示：

|  |  |  |
| --- | --- | --- |
| **语法** | **描述** | **实例** |
| mongodump --host HOST\_NAME --port PORT\_NUMBER | 该命令将备份所有MongoDB数据 | mongodump --host runoob.com --port 27017 |
| mongodump --dbpath DB\_PATH --out BACKUP\_DIRECTORY |  | mongodump --dbpath /data/db/ --out /data/backup/ |
| mongodump --collection COLLECTION --db DB\_NAME | 该命令将备份指定数据库的集合。 | mongodump --collection mycol --db test |

* **--host <:port>, -h <:port>：**

MongoDB所在服务器地址，默认为： localhost:27017

* **--db , -d ：**

需要恢复的数据库实例，例如：test，当然这个名称也可以和备份时候的不一样，比如test2

* **--drop：**

恢复的时候，先删除当前数据，然后恢复备份的数据。就是说，恢复后，备份后添加修改的数据都会被删除，慎用哦！

* **<path>：**

mongorestore 最后的一个参数，设置备份数据所在位置，例如：c:\data\dump\test。

你不能同时指定 <path> 和 --dir 选项，--dir也可以设置备份目录。

* **--dir：**

指定备份的目录

你不能同时指定 <path> 和 --dir 选项。

Mongostat。它会间隔固定时间获取mongodb的当前运行状态，

mongotop查看哪些大量的时间花费在读取和写入数据