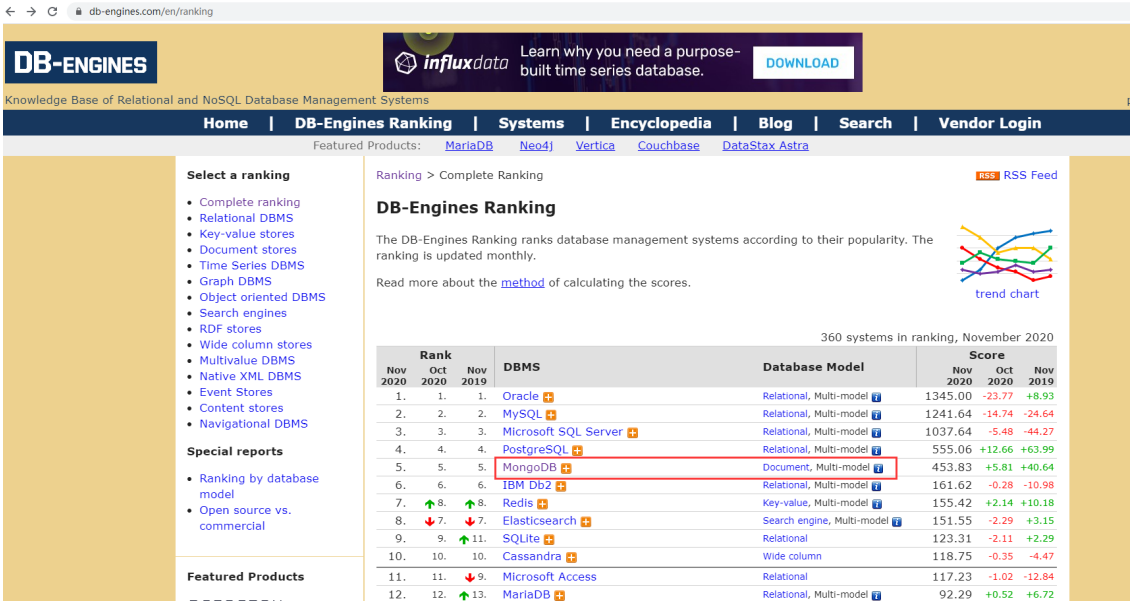
1.MongoDB 基本概念详解
2.MongoDB 快速实战
3.MongoDB 核心操作与原理详解

Mongo 是 hu**mongo**us 的中间部分，在英文里是“巨大无比”的意思。所以 MongoDB 可以翻译成“巨大无比的数据库”，更优雅的叫法是“海量数据库”。**Mongodb**是一款**非关系型**数据库，说到非关系型数据库，区别于关系型数据库最显著的特征就是没有SQL语句，数据没有固定的数据类型，关系数据库的所使用的SQL语句自从 IBM 发明出来以后，已经有 40 多年的历史了，但是时至今日，开发程序员一般不太喜欢这个东西，因为它的基本理念和程序员编程的想法不一致。后来所谓的 NoSQL 风，指的就是那些不用 SQL 作为查询语言的数据存储系统，而文档数据库 MongoDB 正是 NoSQL 的代表。

看一下当下 数据库的排名就会发现，目前排在Mongodb数据库前面的无一例外是老牌的关系型数据库，而在No SQL序列中，Mongodb排名第一，且有上升的趋势

以下来自 https://www.db-engines.com 的数据

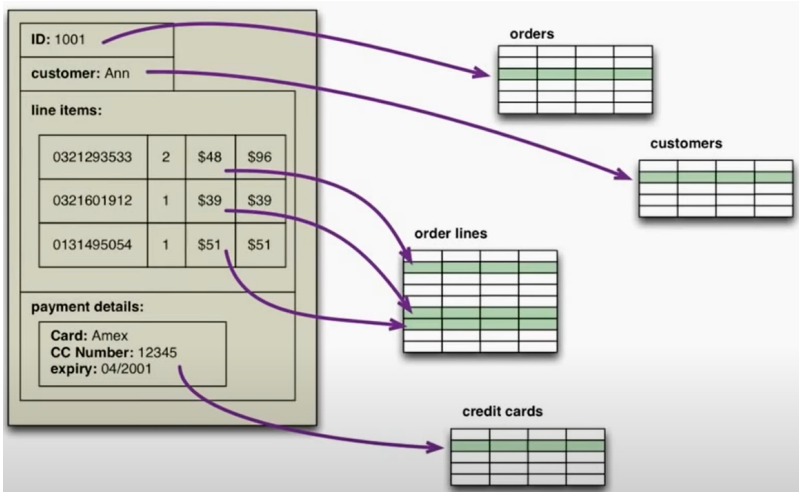


我们在正式进入Mongodb的学习之前，先来了解一下，MongoDB都有哪些特点，为什么要引入MongoDB以及MongoDB和关系型数据库的差异?

1. MongoDB中的记录是一个文档，它是由字段和值对组成的数据结构。MongoDB文档类似于JSON对象。字段的值可以包括其他文档，数组和文档数组。MongoDB数据模型和你的对象在内存中的表现形式一样，一目了然的对象模型。

```
{
  name: "sue",                          <-- field: value
  age: 26,                              <-- field: value
  status: "A",                          <-- field: value
  groups: [ "news", "sports" ]          <-- field: value
}
```

关系型数据库设计（第三范式）：



2.同一个集合中可以包含不同字段（类型）的文档对象：同一个集合的字段可能不同
3.线上修改数据模式，修改时应用与数据库都无须下线

关系型数据库和文档型数据库主要概念对应

	关系型数据库	**文档型数据库**
模型实体	表	集合
模型属性	列	字段
模型关系	表关联	内嵌数组，引用字段关联

MongoDB安装：

1. 获取安装包

```
wget https://fastdl.mongodb.org/linux/mongodb-linux-x86_64-rhel70-4.4.2.tgz
```

2. 进行解压

```
tar -xvzf mongodb-linux-x86_64-rhel70-4.4.2.tgz
```

3. 添加到系统执行路径下面(~/.bashrc)

```
export PATH=$PATH:<你机器MongoDB bin目录，如：/usr/local/mongodb/mongodb-linux-x86_64-rhel70-4.4.2/bin>
```

执行 source ~/.bashrc

4. 创建数据目录

```
mkdir -p /data/db # 这个路径是MongoDB默认的数据存放路径
```

5. 启动MongoDB服务

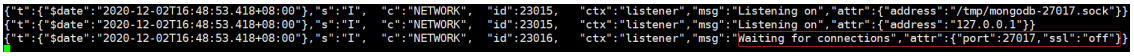
```
mongod # 如果你不希望使用的默认数据目录可以通过 添加 --dbpath 参数指定路径
```

或者从后台启动

mongod --logpath /data/db/logpath/output --fork

需要指定 --logpath , 或者--syslog

出现如下图所示提示则说明服务已经启动成功



https://docs.mongodb.com/guides/server/install/

客户端使用(mongo shell, 用来操作MongoDB的javascript客户端界面):

1. 连接服务

```
mongo --host <HOSTNAME> --port <PORT>
# 如果在本机使用的都是默认参数，也可以直接忽略所有参数
```

2. 设置密码

```
use admin # 设置密码需要切换到admin库

db.createUser(
 {
 user: "gj",
 pwd: "gj123",
```

```
  roles: [ "root" ]
  }
)
show users # 查看所有用户信息
```

```
> show users
{
        "_id" : "admin.gj",
        "userId" : UUID("5fe90658-1172-4871-a26d-a2ae09d2bb0c"),
        "user" : "gj",
        "db" : "admin",
        "roles" : [
                {
                        "role" : "root",
                        "db" : "admin"
                }
        ],
        "mechanisms" : [
                "SCRAM-SHA-1",
                "SCRAM-SHA-256"
        ]
}
```

3. 停服务

```
db.shutdownServer() # 停掉服务
```

4. exit 退出 mongo

5. 以授权模式启动

```
mongod --auth
```

6. 授权方式连接

```
mongo -u gj
```

7. 连上之后就可以进行操作：

连上之后先来看看都有哪些操作

```
> help
        db.help()                    help on db methods
        db.mycoll.help()             help on collection methods
        sh.help()                    sharding helpers
        rs.help()                    replica set helpers
        help admin                   administrative help
        help connect                 connecting to a db help
        help keys                    key shortcuts
        help misc                    misc things to know
        help mr                      mapreduce

        show dbs                     show database names
        show collections             show collections in current database
        show users                   show users in current database
        show profile                 show most recent system.profile entries with time >= 1ms
        show logs                    show the accessible logger names
        show log [name]              prints out the last segment of log in memory, 'global' is default
        use <db_name>                set current database
        db.mycoll.find()             list objects in collection mycoll
        db.mycoll.find( { a : 1 } )  list objects in mycoll where a == 1
        it                           result of the last line evaluated; use to further iterate
        DBQuery.shellBatchSize = x   set default number of items to display on shell
        exit                         quit the mongo shell
```

连接进来之后，就是一个命令行的窗体， 这也是JavaScript 语言的运行环境，所以可以在上面用 javascript 进行脚本编写，执行，操作，管理数据库。

```
> var name="guo";
> print(name)
guo
>
> function doSth(obj1, obj2 ){  print( obj1+ obj2)
> doSth(name," jia");
guo jia
```

安全说明：

MongoDB基于安全性考虑，默认安装后只会绑定本地回环 IP 127.0.0.1, 可以通过启动服务时，指定绑定的IP 如 只允许通过 IP：192.168.109.200 访问，

```
mongod --bind_ip 192.168.109.200
```

这时登录需要通过

```
mongo -host 192.168.109.200 -u gj
```

通过命令访问 MongoDB

https://docs.mongodb.com/manual/tutorial/getting-started/

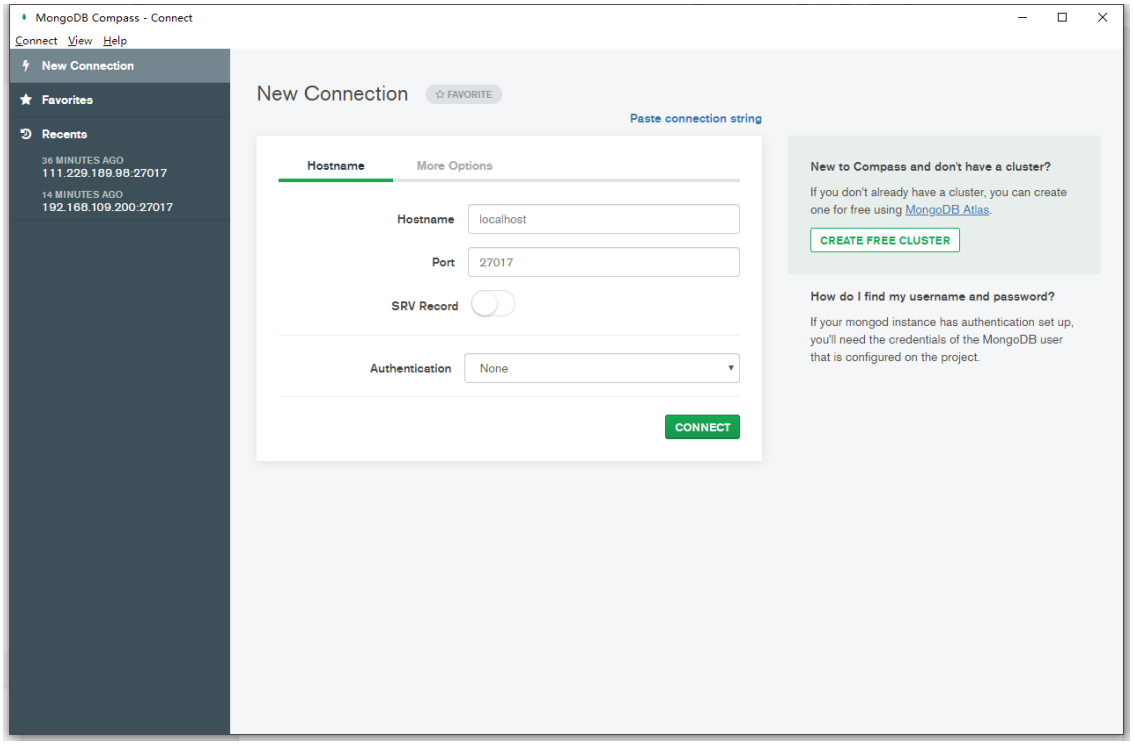
```
输入 db # 显示当前所在的数据库

```

```
use example # 切换数据库
```

UI客户端访问：

https://docs.mongodb.com/compass/master/install



基本操作：添加数据

```
db.集合.insertOne(<JSON对象>) // 添加单个文档
```

```
db.集合.insertMany([{<JSON对象1>},{<JSON对象2>}]) // 批量添加文档
db.集合.insert() // 添加单个文档
```

开始创建文档

```
db.collection.insertOne(
doc ,
{
 writeConcern: 安全级别 // 可选字段
}
)
```

writeConcern 定义了本次文档创建操作的安全写级别简单来说， 安全写级别用来判断一次数据库写入操作是否成功，安全写级别越高，丢失数据的风险就越低，然而写入操作的延迟也可能更高。

writeConcern 决定一个写操作落到多少个节点上才算成功。 writeConcern的取值包括
0： 发起写操作，不关心是否成功
1- 集群中最大数据节点数： 写操作需要被复制到指定节点数才算成功
majority: 写操作需要被复制到大多数节点上才算成功
发起写操作的程序将阻塞到写操作到达指定的节点数为止

```
db.emp.insertOne(
{
 name:"zhangsan",
 age:20,
 sex:"m"}
);
```

```
> db.emp.insertOne({name:"zhangsan",age:20,sex:"m"});
{
        "acknowledged" : true,
        "insertedId" : ObjectId("5fe0ef13ac05741b758b3ced")
}
> db.emp.find();
{ "_id" : ObjectId("5fe0ef13ac05741b758b3ced"), "name" : "zhangsan", "age" : 20, "sex" : "m" }
>
```

插入文档时，如果没有显示指定主键，MongoDB将默认创建一个主键，字段固定为_id,ObjectId() 可以快速生成的12字节id 作为主键，**ObjectId** 前四个字节代表了主键生成的时间，精确到秒。主键ID在客户端驱动生成，一定程度上代表了顺序性，但不保证顺序性， 可以通过ObjectId("id值").getTimestamp() 获取创建时间。

如：
ObjectId("5fe0ef13ac05741b758b3ced").getTimestamp();

```
> ObjectId("5fe0ef13ac05741b758b3ced").getTimestamp();
ISODate("2020-12-21T18:53:07Z")
```

创建多个文档

```
db.collection.insertMany(
[ {doc } , {doc }, ....],
 {
 writeConcern: doc,
 ordered: true/false
 }
)
```

ordered：觉得是否按顺序进行写入

顺序写入时，一旦遇到错误，便会退出，剩余的文档无论正确与否，都不会写入

乱序写入，则只要文档可以正确写入就会正确写入，不管前面的文档是否是错误的文档

MongoDB以集合（collection）的形式组织数据，collection 相当于关系型数据库中的表，如果collection不存在，当你对不存在的collection进行操作时，将会自动创建一个collection

如下：

将会创建一个 inventory 集合，并且插入 5 个文档

```
db.inventory.insertMany([
 { item: "journal", qty: 25, status: "A", size: { h: 14, w: 21, uom: "cm" }, tags: [ "blank", "red" ] },
 { item: "notebook", qty: 50, status: "A", size: { h: 8.5, w: 11, uom: "in" }, tags: [ "red", "blank" ] },
 { item: "paper", qty: 10, status: "D", size: { h: 8.5, w: 11, uom: "in" }, tags: [ "red", "blank", "plain" ] },
 { item: "planner", qty: 0, status: "D", size: { h: 22.85, w: 30, uom: "cm" }, tags: [ "blank", "red" ] },
 { item: "postcard", qty: 45, status: "A", size: { h: 10, w: 15.25, uom: "cm" }, tags: [ "blue" ] }
]);
```

上述操作返回一个包含确认指示符的文档和一个包含每个成功插入文档的_id的数组

整个文档查询：

db.inventory.find({})　　查询所有的文档

db.inventory.find({}).pretty()　　返回格式化后的文档

条件查询：

1. 精准等值查询

db.inventory.find({ status: "D" });

db.inventory.find({ qty: 0 });

2. 多条件查询

db.inventory.find({ qty: 0, status: "D" });

3. 嵌套对象精准查询

db.inventory.find({ "size.uom": "in" });

4. 返回指定字段

db.inventory.find({ }, { item: 1, status: 1 });

默认会返回_id 字段， 同样可以通过指定 _id:0 ,不返回_id 字段

5. 条件查询 and

db.inventory.find({$and:[{"qty":"0"},{"status":"A"}]}).pretty();

6. 条件查询 or

db.inventory.find({$or:[{"qty":"0"},{"status":"A"}]}).pretty();

Mongo查询条件和SQL查询对照表

SQL	MQL
a<>1 或者 a!=1	{ a : {$ne: 1}}
a>1	{ a: {$gt:1}}
a>=1	{ a: {$gte:1}}
a<1	{ a: {$lt:1}}
a<=1	{ a: { $lte:1}}
in	{ a: { $in:[x, y, z]}}
not in	{ a: { $nin:[x, y, z]}}
a is null	{ a: { $exists: false }}

insertOne, inertMany, insert 的区别

insertOne, 和 insertMany命令不支持 explain命令

insert支持 explain命令

复合主键

可以使用文档作为复合主键

```
db.demeDoc.insert(
{
 _id: { product_name: 1, product_type: 2},
supplierId:" 001",
create_Time: new Date()
}
)
```

注意复合主键，字段顺序换了，会当做不同的对象被创建，即使内容完全一致

逻辑操作符匹配

$not : 匹配筛选条件不成立的文档

$and : 匹配多个筛选条件同时满足的文档

$or : 匹配至少一个筛选条件成立的文档

$nor : 匹配多个筛选条件全部不满足的文档

构造一组数据:

```
db.members.insertMany([
{
  nickName:"曹操",
  points:1000
},
{
  nickName:"刘备",
  points:500
}
]);
```

```
$not
```

用法:

```
{ field:  { $not : { operator-expression} }}
```

积分不小于100 的

```
db.members.find({points: { $not: { $lt: 100}}} );
```

$not 也会筛选出并不包含查询字段的文档

```
$and
```

用法

```
{ $and : [ condition expression1 , condition expression2 ..... ]}
```

昵称等于曹操， 积分大于 1000 的文档

```
db.members.find({$and : [ {nickName:{ $eq : "曹操"}}, {points:{ $gt:1000}}]});
```

当作用在不同的字段上时 可以省略 $and

```
db.members.find({nickName:{ $eq : "曹操"}, points:{ $gt:1000}});
```

当作用在同一个字段上面时可以简化为

```
db.members.find({points:{ $gte:1000, $lte:2000}});
```

```
$or
```

用法

```
{ $or :{ condition1, condition2, condition3,... }}
```

```
db.members.find(
{$or : [
 {nickName:{ $eq : "刘备"}},
 {points:{ $gt:1000}}]}
);
```

如果都是等值查询的话， $or 和 $in 结果是一样的

字段匹配

$exists：匹配包含查询字段的文档

```
{ field : {$exists: <boolean>} }
```

文档游标

```
cursor.count( applySkipLimit)
```

```
> db.movies.find().skip(1).count();
4
> db.movies.find().skip(1).limit(1).count();
4
> db.movies.find().skip(1).limit(1).count(true);
1
```

默认情况下， 这里的count不会考虑 skip 和 limit的效果，如果希望考虑 limit 和 skip ， 需要设置为 true。 分布式环境下，count 不保证数据的绝对正确

```
cursor.sort( <doc>)
```

这里的<doc> 定义了排序的要求

```
{ field: ordering}
```

1 表示由小到大， -1 表示逆向排序

当同时应用 sort， skip， limit 时 ，应用的顺序为 sort， skip， limit

文档投影： 可以有选择性的返回数据

```
db.collection.find( 查询条件, 投影设置)
```

投影设置：{ field: < 1 ： 1 表示需要返回， 0： 表示不需要返回 ， 只能为 0，或者1 ， 非主键字段，不能同时混选0 或 1>}

```
db.members.find({},{_id:0 ,nickName:1, points:1})
```

```
db.members.find({},{_id:0 ,nickName:1, points:0})
```

```
> db.members.find({},{_id:0 ,nickName:1, points:0})
Error: error: {
        "ok" : 0,
        "errmsg" : "Cannot do exclusion on field points in inclusion projection",
        "code" : 31254,
        "codeName" : "Location31254"
}
```

可以使用 $slice 返回数组中的部分元素

如， 先添加一个数组元素的文档

```
db.members.insertOne(
{
 _id: {uid:3,accountType: "qq"},
```

```
 nickName:"张飞",
 points:1200,
 address:[
 {address:"xxx",post_no:0000},
 {address:"yyyyy",post_no:0002}
 ]}
);
```

返回数组的第一个元素

```
db.members.find(
{},
{_id:0,
 nickName:1,
 points:1,
 address:
 {$slice:1}
});
```

返回倒数第一个

```
db.members.find(
{},
{_id:0,
 nickName:1,
 points:1,
 address:{$slice:-1}}
 );
```

slice: 值
1： 数组第一个元素
-1：最后一个元素
-2：最后两个元素
slice[1,2] : skip, limit 对应的关系

还可以使用 elementMatch 进行数组元素进行匹配
添加一组数据

```
db.members.insertOne(
{
 _id: {uid:4,accountType: "qq"},
 nickName:"张三",
```

```
 points:1200,
 tag:["student","00","IT"]}
);
```

查询tag数组中第一个匹配"00" 的元素

```
db.members.find(
 {},
 {_id:0,
 nickName:1,
 points:1,
 tag: { $elemMatch: {$eq: "00" } }
});
```

$elemMatch 和 $ 操作符可以返回数组字段中满足条件的第一个元素

更新操作

updateOne/updateMany 方法要求更新条件部分必须具有以下之一，否则将报错

$set　给符合条件的文档新增一个字段，有该字段则修改其值

$unset 给符合条件的文档，删除一个字段

$push： 增加一个对象到数组底部

$pop：从数组底部删除一个对象

$pull：如果匹配指定的值，从数组中删除相应的对象

$pullAll：如果匹配任意的值，从数据中删除相应的对象

$addToSet：如果不存在则增加一个值到数组

更新文档:

单条插入数据， 插入两跳

```
db.userInfo.insert([
{ name:"zhansan",
 tag:["90","Programmer","PhotoGrapher"]
},
{ name:"lisi",
 tag:["90","Accountant","PhotoGrapher"]
}]);
```

将tag 中有90 的文档，增加一个字段： flag: 1

```
db.userInfo.updateMany(
{tag:"90"},
{$set:{flag:1}}
);
```

只修改一个则用

```
db.userInfo.updateOne(
 {tag:"90"},
 {$set:{flag:2}}
 );
```

基于上面这两条数据，可以来查询一下数组中的元素

userInfo 中，会计和程序员的文档

```
db.userInfo.find(
{$or:
  [
   {tag:"Accountant"},
   {tag:"Programmer"}
  ]
});
```

userInfo 中，90后的文档

```
db.userInfo.find({tag:"90"});
```

更新文档

```
db.collection.update( <query>,<update>,<options>)
```

<query> 定义了更新时的筛选条件
<update> 文档提供了更新内容
<options> 声明了一些更新操作的参数

更新文档操作只会作用在第一个匹配的文档上
如果<update> 不包含任何更新操作符，则会直接使用update 文档替换集合中符合文档筛选条件的文档

更新特定字段

db.collection.update(<query>,<update>,<options>)
<query> 定义了更新时的筛选条件
<update> 文档提供了更新内容
<options> 声明了一些更新操作的参数

如果<update>只包含更新操作符，db.collection.update() 将会使用update更新集合中符合<query>筛选条件的文档中的特定字段。

默认只会更新第一个匹配的值，可以通过设置 options {multi: true} 设置匹配多个文档并更新

```
db.doc.update(
 {name:"zhangsan"},
 {$set:{ flag: 1 }},
 {multi:true}
);
```

更新操作符

$set 更新或新增字段

$unset删除字段

$rename 重命名字段

$inc 加减字段值

$mul 相乘字段值

$min 采用最小值

$max 次用最大值

删除文档

```
db.collection.remove(<query>,<options>)
```

默认情况下，会删除所有满足条件的文档， 可以设定参数 { justOne:true},只会删除满足添加的第一条文档

删除集合

```
db.collection.drop( { writeConcern:<doc>})
```

<doc> 定义了本次删除集合操作的安全写级别

这个指令不但删除集合内的所有文档，且删除集合的索引

db.collection.remove 只会删除所有的文档，直接使用remve删除所有文档效率比较低，可以使用drop 删除集合，才重新创建集合以及索引。

查询数组中的对象

加两行数据，文档中存在数组，且数组中你的元素为对象

```
db.userInfo.insertMany([
{ name:"wangwu",
 tag: ["90","accountant","PhotoGrapher"],
```

```
 address:[{province:"hunan",city:"changsha"},{province:"hubei",city:"wuhan"}]},
{ name:"zhaoliu",
 tag: ["90","accountant","PhotoGrapher"],
 address:[{province:"hunan",city:"changsha"},{province:"hubei",city:"huanggang"}]}
]);
```

删除文档 remove

```
db.userInfo.remove({ 条件查询});
```

文档：VIP-01 MongoDB快速入门.note

链接：http://note.youdao.com/noteshare?id=5e038498891617c552667b853742fdc1&sub=6D05DF89C0BE4783A3031FFA6713F683