MongoDB使用初探

> MongoDB 是一种NoSQL 数据库，本文主要介绍MongoDB里面的一些基本概念和操作、以及可视化工具的安装和使用。

> 作者：奔放的辣条妹 阅读时间大约 20min+

1 MongoDB 简介

MongoDB 是一种NoSQL 数据库，存储的数据对象由键值对组成。MongoDB 所有存储在集合中的数据都是 BSON 格式。BSON 是一种类似 JSON 的二进制形式的存储格式，是 Binary JSON 的简称。如下所示：

```
{
    "_id" : ObjectId("5c89f787ca6e4e3ac1ecabkk"),
    "_plat" : "test_plat0",
    "update_time" : ISODate("2019-06-03T15:00:42.142Z"),
    "create_time" : ISODate("2019-03-14T14:41:11.217Z"),
    "creator" : "test_user",
    "admin" : [
        "admin1",
        "admin2"
    ],
    "ops" : [
        "ops1"
    ],
    "labels" : {
        "department" : "departmentA",
        "main_class" : "mainClassA"
    }
}
```

复制代码

下面对照关系型数据库介绍一些 MongoDB 里面的基本概念：

关系数据库术语	MongoDB术语	说明
database	database	数据库
table	collection	数据库表/集合
row	document	记录行/文档
column	field	数据字段/域
index	index	索引
primary key	primary key	主键，Mongodb自动将_id字段设置为主键

通过以下对比可以更理解 MongoDB：

id	姓名	年龄	性别
1	张三	23	男
2	李四	21	男

上述关系型数据在 MongoDB 中的数据形式为：

```
{
    "_id" : ObjectId("5c89f787ca6e4e3ac1ehhb23"),
    "姓名" : "张三",
    "年龄" : 23,
    "性别" : "男"
}
{
    "_id" : ObjectId("5c89f787ca6e4e3ac1ehhb24"),
    "姓名" : "李四",
    "年龄" : 21,
    "性别" : "男"
}
复制代码
```

2 MongoDB 基本操作

本节主要介绍通过命令行操作 MongoDB，以 MongoDB 安装在 CentOS 上为例进行说明。

2.1 db、集合等基本操作

DB 的查看、创建、删除，集合的查看、创建、删除等操作方式如下：

```
[root] mongo           #命令行输入mongo进入MongoDB命令交互模式
> show dbs             #列出已有db
> use my_db            #如果my_db存在，则切换到my_db，如果不存在，则创建之
> db                   #显示当前db
> show dbs             #发现列表里面没有my_db，因为此时db里面没有实际数据或者集合哦
> db.createCollection("my_col")  #创建集合my_col
> db.my_col_new.insert({"name":"测试一下"})  #往集合my_col_new里面插入一条数据，如果集合不存在，会自动创建
> show collections     #列出改db下面所有的集合
> show tables          #功能跟show collections是一样的哦
> db.my_col.drop()     #删除集合my_col
> db.dropDatabase()    #删除当前数据库，执行之前用db命令确认一下当前数据库是不是你要删除的这个哦
复制代码
```

2.2 数据插入

插入数据有4种方法：insert、insertOne、insertMany、save，下面通过例子详细介绍。

```
> db.my_col.insert({"name":"xiaoming"})  #insert可以插入一条数据
> db.my_col.insert([{"name":"xiaoming"},{"name":"test_user"}])  #insert也可以插入多条数据
> db.my_col.insertOne({"name":"xiaoming"})  #insertOne只能插入一条数据
> db.my_col.insertMany([{"name":"xiaoming"}])  #insertMany可以插入一条或多条数据，但是必须以列表(list)的形式组织数据
> db.my_col.save([{"name":"xiaoming"},{"name":"test_user"}])  #如果不指定_id，save的功能与insert一样
> db.my_col.save({"_id":ObjectId("5d07461141623d5db6cd4d43"),"name":"xiaoming"})
  #如果指定_id，mongodb就不为该条记录自动生成_id了，只有save可以指定_id，insert、insertOne、insertMany都不可以
复制代码
```

2.3 数据修改

修改数据有2种方法：update、save，下面详细介绍。

2.3.1 update

首先，看一下 update 的语法格式，请格外注意一些可选参数的值，这将直接影响你的修改结果：

```
db.collection.update(
   <query>,   #update的查询条件，类似sql update语句where后面的部分
   <update>,  #update的对象和一些更新的操作符等，也可以理解为sql update语句set后面的
   {
     upsert: <boolean>,  #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: <boolean>,   #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: <document>  #可选，抛出异常的级别
   }
)
复制代码
```

假设有这样一张学生成绩表：

```
> db.my_col.insert([{"name":"xiaoming","class":"c++", "score":60},
{"name":"xiaoming","class":"python", "score":95}])
> db.my_col.find().pretty()
{
        "_id" : ObjectId("5d0751ef41623d5db6cd4d44"),
        "name" : "xiaoming",
        "class" : "c++",
        "score" : 60
}
{
        "_id" : ObjectId("5d0751ef41623d5db6cd4d46"),
        "name" : "xiaoming",
        "class" : "python",
        "score" : 95
}
复制代码
```

xiaoming 同学发现老师把她的 c++ 课程分数录错了，需要修改为75分：

```
> db.my_col.update({"_id":ObjectId("5d0751ef41623d5db6cd4d44")},{$set:
{"score":75}})
复制代码
```

老师发现把 xiaoming 同学的名字录错了，需要全部修改过来：

```
> db.my_col.update({"name":"xiaoming"},{$set:{"name":"xming"}}) #这样是不对的，只会
修改一条记录
> db.my_col.update({"name":"xiaoming"},{$set:{"name":"xming"}},{multi:true})  #
这样才对
复制代码
```

将 xming 的 java 课程分数改为95分，如果找不到，就插入一条记录

```
> db.my_col.update({"name":"xming", "class": "java"},{$set:{"score": 95}},true)
复制代码
```

2.3.2 save

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>,   #文档数据
   {
     writeConcern: <document> #可选，抛出的异常级别
   }
)
复制代码
```

还以上面那张学生成绩表为例：

```
> db.my_col.save({
         "_id" : ObjectId("5d0751ef41623d5db6cd4d44"), #指定_id，新的文档会将旧的文
档覆盖
         "name" : "xming",
         "class" : "c++",
         "score" : 80
})
复制代码
```

2.4 数据删除

数据删除可以使用 deleteOne、deleteMany、remove，下面详细介绍。

2.4.1 deleteOne 和 deleteMany

使用方法如下：

```
> db.my_col.deleteOne({"name":"xming"})  #删除xming的一条成绩记录
> db.my_col.deleteMany({"name":"xming"}) #删除xming的所有成绩记录
> db.my_col.deleteMany({})  #删除成绩表里面的所有内容
复制代码
```

2.4.2 remove

首先还是来看语法格式：

```
db.collection.remove(
   <query>,    #可选，查询条件
   {
     justOne: <boolean>,  #可选，设置为true或者1，表示只删除一个文档，设置为false，表示删除所有匹配的文档，默认为false
     writeConcern: <document> #可选，抛出异常的级别
   }
)
复制代码
```

删除 xming 的所有成绩记录：

```
> db.col.remove({"name":"xming"})
> db.repairDatabase()  #remove方法并不会真正释放空间，需要继续执行 db.repairDatabase()来回收磁盘空间
> db.runCommand({ repairDatabase: 1 }) #与上一句等效，仍以执行一句即可
复制代码
```

ps：remove 现在已经过时了现在官方推荐使用 deleteOne 和 deleteMany 方法。

2.5 数据查询

数据查询的方法有 findOne 和 find，二者参数等用法一样，但是 findOne 只返回一条匹配的数据，find 返回全部的匹配数据，下面主要介绍 find 的用法。

2.5.1 条件操作符

操作	sql查询写法	mongo查询写法
等于	select * from my_col where score = 75;	db.my_col.find({"score": 75}).pretty()
小于	select * from my_col where score < 75;	db.my_col.find({"score": {$lt: 75}}).pretty()
小于等于	select * from my_col where score <= 75;	db.my_col.find({"score": {$lte: 75}}).pretty()
大于	select * from my_col where score > 75;	db.my_col.find({"score": {$gt: 75}}).pretty()
大于等于	select * from my_col where score >= 75;	db.my_col.find({"score": {$gte: 75}}).pretty()
不等于	select * from my_col where score != 75;	db.my_col.find({"score": {$ne: 75}}).pretty()

ps：pretty 能让查询结果以格式化的 json 形式打印出来，便于查看

2.5.2 排序、limit 与 skip

以分数从高到低显示学生的 c++ 课程成绩，只显示第10名到第20名的学生：

```
> db.my_col.find({"class": "c++"}).sort({"score":
-1}).skip(9).limit(11).pretty()
#sort：1为升序，-1为降序，默认升序
#limit：显示多少条数据
#skip：跳过多少条数据
复制代码
```

2.5.3 复合条件查询 and、or

and：find 方法可以传入多个键值对，每个键值对以逗号隔开，即常规 SQL 的 AND 条件

查询 xiaoming 同学的 c++ 课程成绩：

```
> db.my_col.find({"name": "xiaoming", "class": "c++"}).pretty()
复制代码
```

查询分数在75到85分之间的成绩记录：

```
> db.my_col.find({"score": {$gt: 75, $lt: 85}}).pretty()
复制代码
```

or：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()
复制代码
```

查询 xiaoming 或 zhangsan 的课程成绩：

```
> db.my_col.find({$or: [{"name": "xiaoming"}, {"name": "zhangsan"}]}).pretty()
复制代码
```

and + or 复合查询：

查询 xiaoming 的 c++ 或者 python 课程的成绩：

```
> db.my_col.find({"name": "xiaoming", $or: [{"class": "c++"}, {"class":
"python"}]}).pretty()
复制代码
```

2.5.4 包含in、不包含nin、全部all

查询 xiaoming、zhangsan 和 lisa 的成绩：

```
> db.my_col.find({"name": {$in: ["xiaoming","zhangsan","lisa"]}}).pretty()
复制代码
```

查询除了 xiaoming、zhangsan 和 lisa 之外，其他人的成绩：

```
> db.my_col.find({"name": {$nin: ["xiaoming","zhangsan","lisa"]}}).pretty()
复制代码
```

in和nin比较好理解，跟sql的用法类似，all类似于in，不同的地方是，in只需要满足列表中的一个值即可，而all需要满足列表中的全部值。比如，有下面这样一张课程表，表示每个学生修的课程：

```
> db.course.find().pretty()
{
        "_id" : ObjectId("5d084f1541623d5db6cd4d4c"),
        "name" : "xiaoming",
        "course" : [
                "c++",
                "python",
                "java"
        ]
}
{
        "_id" : ObjectId("5d084f1c41623d5db6cd4d4d"),
        "name" : "lisa",
        "course" : [
                "c++",
                "python",
                "java"
        ]
}
{
        "_id" : ObjectId("5d084f4a41623d5db6cd4d4e"),
        "name" : "tom",
        "course" : [
                "c++",
                "python"
        ]
}
复制代码
```

需要找出修了 c++ 和 java课程的学生：

```
> db.course.find({"course": {$all: ["c++", "java"]}}).pretty() # 用 all 操作符，表示需要满足 c++ 和 java 两项
复制代码
```

2.5.5 判断字段是否存在 exists

比如，有下面一张表，表示学生信息：

```
> db.stu_info.find().pretty()
{
        "_id" : ObjectId("5d08519a41623d5db6cd4d4f"),
        "name" : "xiaoming",
        "tel" : "138xxxxxxxx"
}
{
        "_id" : ObjectId("5d08531641623d5db6cd4d50"),
```

```
        "name" : "lisa"
}
{
        "_id" : ObjectId("5d08542e41623d5db6cd4d51"),
        "name" : "tom",
        "tel" : null
}
复制代码
```

需要找出没有 tel 字段的学生：

```
> db.stu_info.find({"tel": {$exists: false}}).pretty() #字段不存在就用false，存在就用true
复制代码
```

2.5.6 空值处理 null

以上面的学生信息表为例，找出 tel 为空值的学生：

```
> db.stu_info.find({"tel": null}).pretty()
{
        "_id" : ObjectId("5d08531641623d5db6cd4d50"),
        "name" : "lisa"
}
{
        "_id" : ObjectId("5d08542e41623d5db6cd4d51"),
        "name" : "tom",
        "tel" : null
}
复制代码
```

这时候把 tel 字段不存在和 tel 值为 null 的情况都查出来了！如果只想找 tel 值为 null 的情况：

```
> db.stu_info.find({"tel": {$in:[null], $exists:true}}).pretty()
复制代码
```

2.5.7 取模运算 mod

比如，查找学生成绩取模10 等于0 的数据（即100、90、80...等等）：

```
> db.my_col.find({"score": {$mod: [10, 0]}}).pretty()
复制代码
```

2.5.8 正则匹配 regex

查询学生名字以a开头的学生成绩：

```
> db.my_col.find({"name": {$regex: /^a.*/}})
复制代码
```

2.5.9 获取查询结果条数 count

获取学生成绩记录的条数：

```
> db.my_col.find().count()
复制代码
```

当使用 limit 方法限制返回的记录数时，默认情况下 count 方法仍然返回全部记录条数。如果希望返回限制之后的记录数量，要使用 count(true) 或者 count(非0)：

```
> db.my_col.find().count()
4
> db.my_col.find().limit(1).count()
4
> db.my_col.find().limit(1).count(true)
1
复制代码
```

2.5.10 distinct

查询课程成绩表中所有学生的名单：

```
> db.my_col.distinct("name")
复制代码
```

2.6 聚合aggregate

在MongoDB中，使用聚合框架可以对集合中的文档进行变换和组合，完成一些复杂的查询操作。聚合框架通过多个构件来创建一个管道(pipeline)，用于对一连串的文档进行处理。这些构件包括但不限于：

操作符	意义
$match	筛选
$project	投射，选择想要的字段或对字段进行重命名
$group	分组
$unwind	拆分
$sort	排序
$limit	限制查询条数
$skip	跳过一些条数

当需要使用多个操作符来完成文档的聚合时，我们可以传入一个数组条件，这也是aggregate的常见用法：

```
> db.my_col.aggregate([
        {$match: {"name": "xiaoming"}},   #查找 xiaoming 同学的课程成绩
        {$project: {"_id":  0}},   #不需要_id字段
        {$sort: {"score":  -1, "class": 1}},  #按分数降序排序；同样分数的，按课程名字升序排序
        {$skip: 1},    #跳过一条数据
        {$limit: 1}    #只显示一条数据
])
复制代码
```

$match$用于对文档集合进行筛选，之后就可以在筛选得到的文档子集上做聚合。”match"可以使用所有常规的查询操作符("gt”、”lt"、"in”等)。通常，在实际使用中应该尽可能将”match"放在管道的前面位置。这样做有两个好处：一是可以快速将不需要的文档过滤掉，以减少管道的工作量；二是如果在投射和分组之前执行"$match"，查询可以使用索引。

$project可以从子文档中提取字段，可以重命名字段。例如，查找学生课程成绩，不显示 _id 字段，显示姓名、课程、成绩字段，同时将 name 字段重命名为 student_name：

```
> db.my_col.aggregate([
        {$project: {"_id": 0, "student_name": "$name", "core": 1, "class": 1}}
])
复制代码
```

$sort$、skip、$limit 的用法比较好理解，就不多做说明。

$group 类似于 sql 中的 group by，主要用于数据处理，比如，计算每个学生的总课程成绩：

```
> db.my_col.aggregate([
        {$group: {_id: "$name",  total: {$sum: "$score"}}}
])
复制代码
```

sum可以替换成操作符avg、min、max，分别表示求平均成绩、最低成绩、最高成绩。

$unwind 可以将数组中的每一个值拆分为单独的文档，比如有下面一条记录，记录了一篇博客以及下面的评论：

```
> db.blog.findOne().pretty()
{
   "_id":ObjectId("5359f6f6ec7452081a7873d7"),
   "title":"这是一篇博客",
   "auth":"xiaoming",
   "comments":[
      {
          "author":"lisa",
          "date":ISODate("2019-01-01T17:52:04.148Z"),
          "text":"Nice post"
      },
      {
          "author":"tom",
          "date":ISODate("2019-01-01T17:52:04.148Z"),
          "text":"I agree"
      }
   ]
}
```

复制代码

现在要找到 lisa 的评论，可以先使用 $unwind 将每条评论拆分为一个独立的文档，然后再进行 match 查询：

```
> db.blog.aggregate({"$unwind":"$comments"})
{
   "results":
       {
          "_id":ObjectId("5359f6f6ec7452081a7873d7"),
          "title":"这是一篇博客",
          "author":"xiaoming",
          "comments":{
               "author":"lisa",
               "date":ISODate("2019-01-01T17:52:04.148Z"),
               "text":"Nice post"
          }
       },
       {
          "_id":ObjectId("5359f6f6ec7452081a7873d7"),
          "title":"这是一篇博客",
          "author":"xiaoming",
          "comments":{
               "author":"tom",
               "date":ISODate("2019-01-01T17:52:04.148Z"),
               "text":"I agree"
          }
       }
}

> db.blog.aggregate([
        {"$unwind":"$comments"},
        {"$match":{"comments.author":"lisa"}}
])
```

复制代码

2.7 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。

创建索引的基本语法如下：

```
db.collection.createIndex(
    {key1: option1, key2: option2},    #key为要创建索引的字段，option为创建索引的方式：1 为升序，-1 为降序，可以对多个字段创建索引，称为复合索引
    {
        background: <boolean>,  #可选，建索引过程会阻塞其它数据库操作，background 设置为 true 可指定以后台方式创建索引，默认值为 false
        unique: <boolean> #可选，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> #可选，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
        sparse: <boolean> #可选，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
    }
)
复制代码
```

对学生成绩表创建索引：

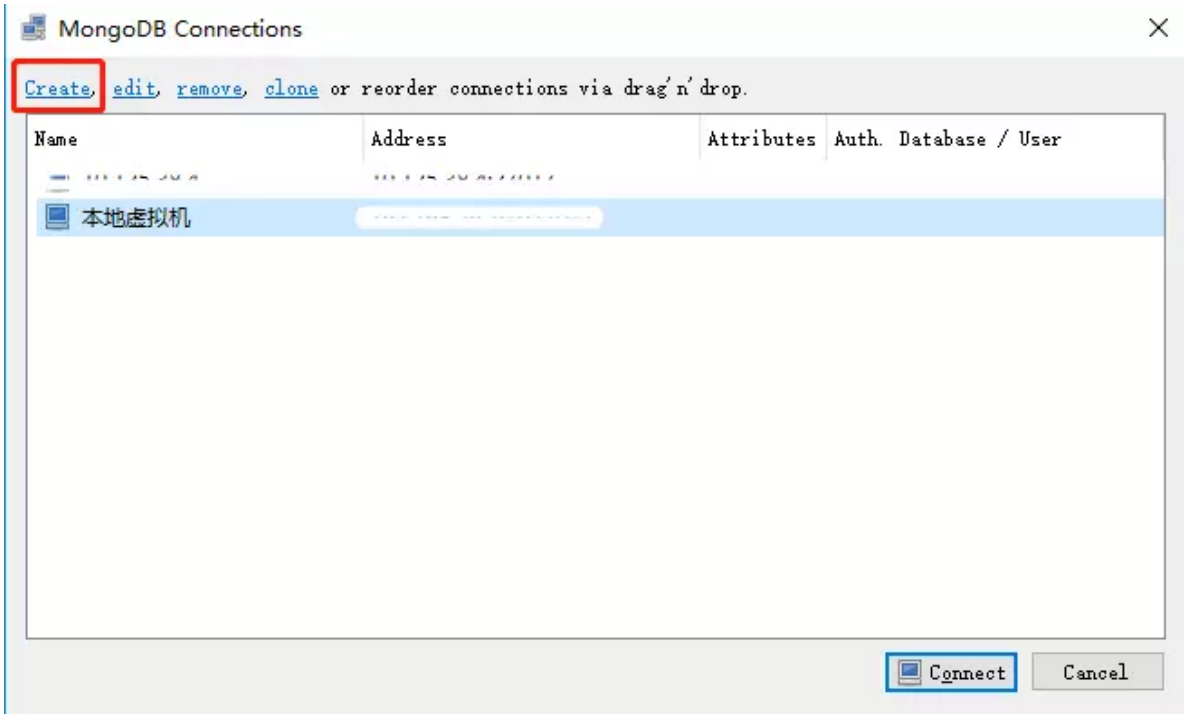
```
> db.my_col.createIndex({"score": 1}, {background: true})  #在后台创建
> db.my_col.getIndexes()   #查看集合索引
> db.my_col.totalIndexSize()  #查看集合索引大小
> db.my_col.dropIndex("索引名称")  #删除集合指定索引
> db.my_col.dropIndexes()  #删除集合所有索引
复制代码
```

3 可视化工具

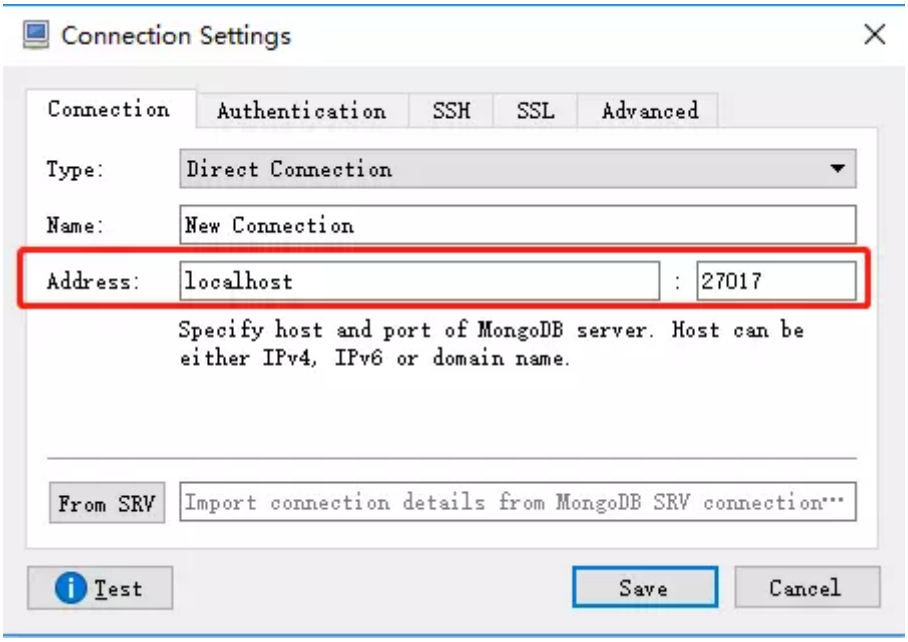
MongoDB有一款跨平台的可视化工具Robo 3T，非常简洁易用。

下载安装地址：robomongo.org/download

下载安装成功之后，点击 file → connect，弹出以下小窗口，然后点击create，新建配置：

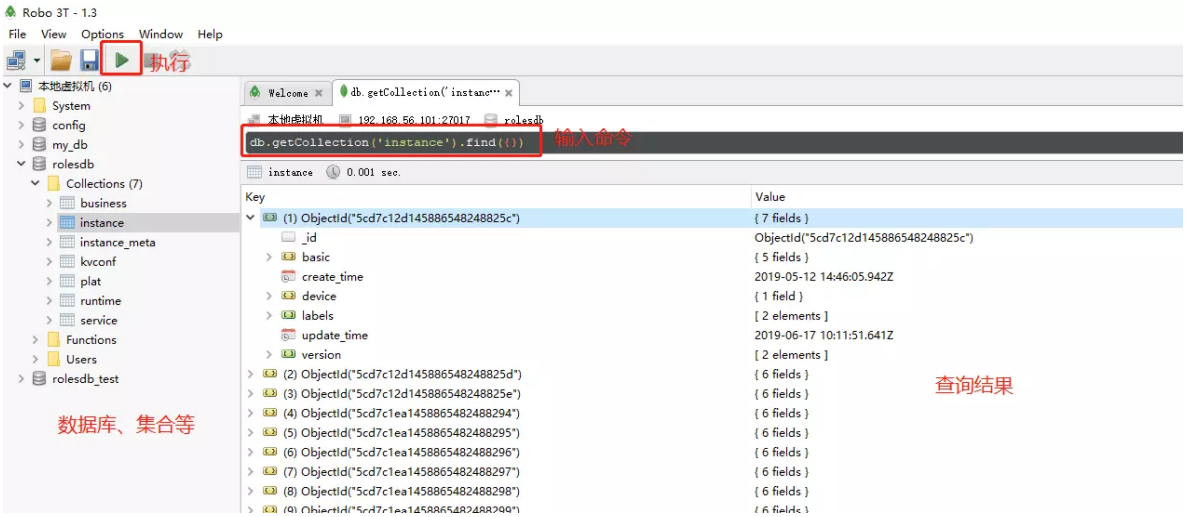


填入正确的地址和端口，保存即可。

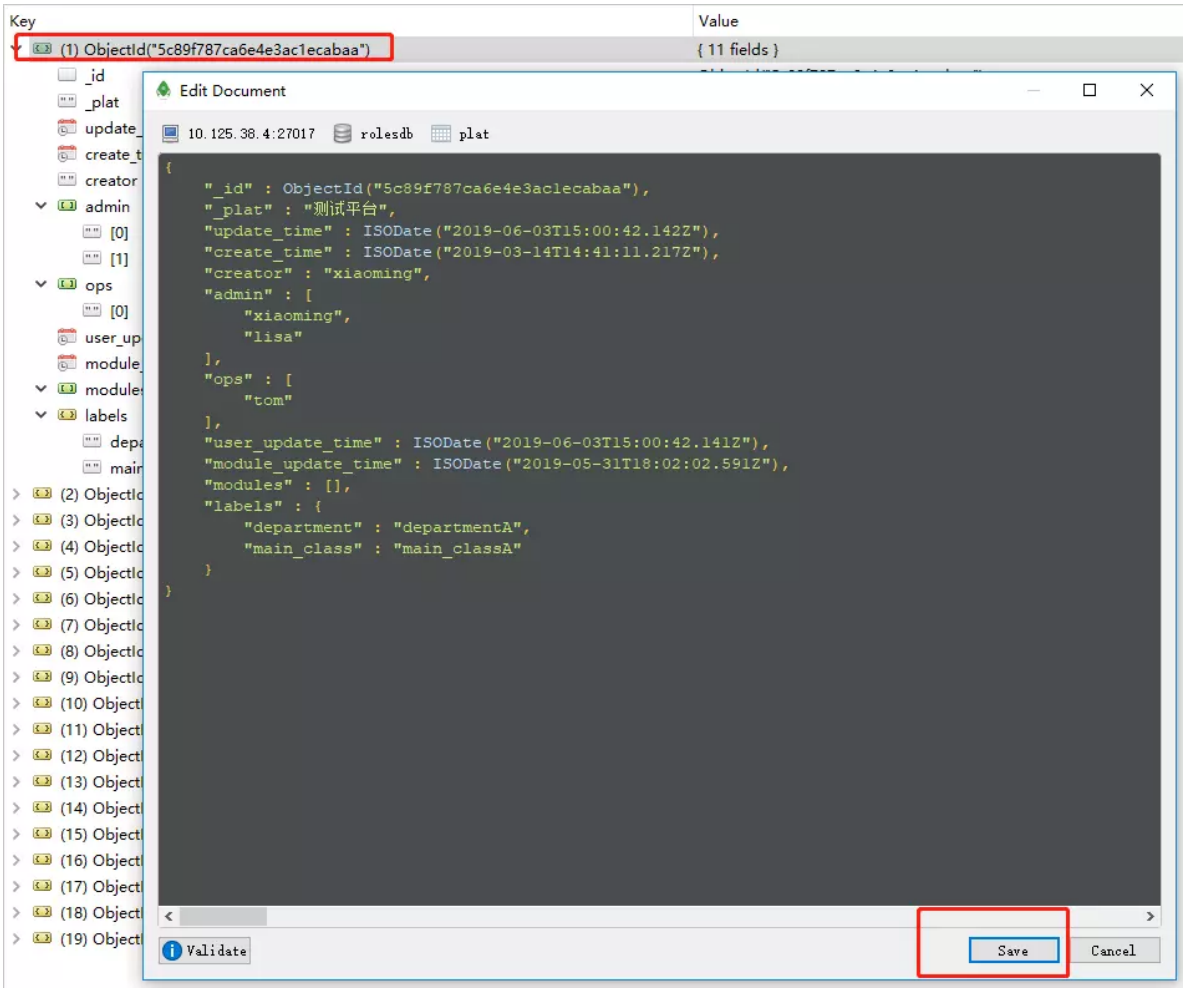


按照上述教程安装之后，如果你连不上所配置的MongoDB，排除网络原因，可能是因为DB服务配置的监听地址是127.0.0.1，需要改成0.0.0.0哦。

双击配置好的连接，即可进入交互界面，如下图所示，左边是数据库和集合等信息，右边是查询结果。可以在命令行输入查询命令然后点击执行，进行过滤等操作。



还可以直接右键对文档进行查看、编辑等操作，非常方便。



4 参考文档

MongoDB菜鸟教程：www.runoob.com/mongodb/mon...

MongoDB高级查询：cw.hubwiz.com/card/c/543b...

MongoDB聚合：cw.hubwiz.com/card/c/5481...

MongoDB官方文档：docs.mongodb.com/manual/intr...

Robo 3T 使用教程：www.cnblogs.com/tugenhua070...> MongoDB 是一种NoSQL 数据库，本文主要介绍MongoDB里面的一些基本概念和操作、以及可视化工具的安装和使用。