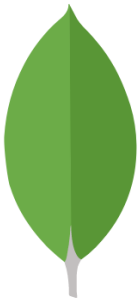
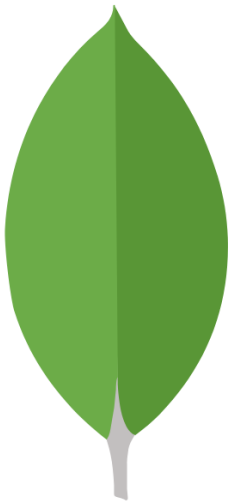


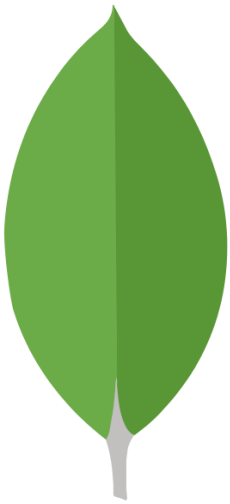
mongoDB

MONGODB
零基础入门手册

| 一灰 著 |

微信公众号：一灰灰BLOG
微信公众号：楼仔





MongoDB零基础入门

MongoDB 是一个基于分布式文件存储的数据库，本片文档主要面向0基础的小伙伴，如何迅速的学会MongoDB的基本知识点以及掌握基础的CURD，实现业务支撑

第一卷：MongoDB原生知识点

环境安装与初始化

> MongoDB 是一个基于分布式文件存储的数据库。由 C++ 语言编写。旨在为 WEB 应用提供可扩展的高性能数据存储解决方案。

MongoDB 是一个介于关系数据库和非关系数据库之间的产品，是非关系数据库当中功能最丰富，最像关系数据库的。

本篇为mongodb系列教程第一篇，环境安装与连接设置

1. docker安装

首先介绍最简单的安装方式，docker安装，请先保证docker环境存在（没有安装的推荐查看: Centos安装docker与使用说明）

安装命令如下:

```
# 下载镜像
docker pull mongo

# 加载并运行镜像
docker run --name mongo -p 27017:27017 -d mongo --auth
# 进入容器
docker exec -it mongo /bin/bash
```

2. centos安装

直接借助yum进行安装，命令如下

```
# 查看支持的mongo库
yum list | grep mongo

yum install -y mongodb.x86_64 mongodb-server.x86_64
```

3. 用户配置

直接通过mongodb提供的终端命令进行设置，

```
# 为mongo创建登录用户和密码
mongo

use admin
db.createUser({user:"root",pwd:"root",roles:[{role:'root',db:'admin'}]})
exit
```

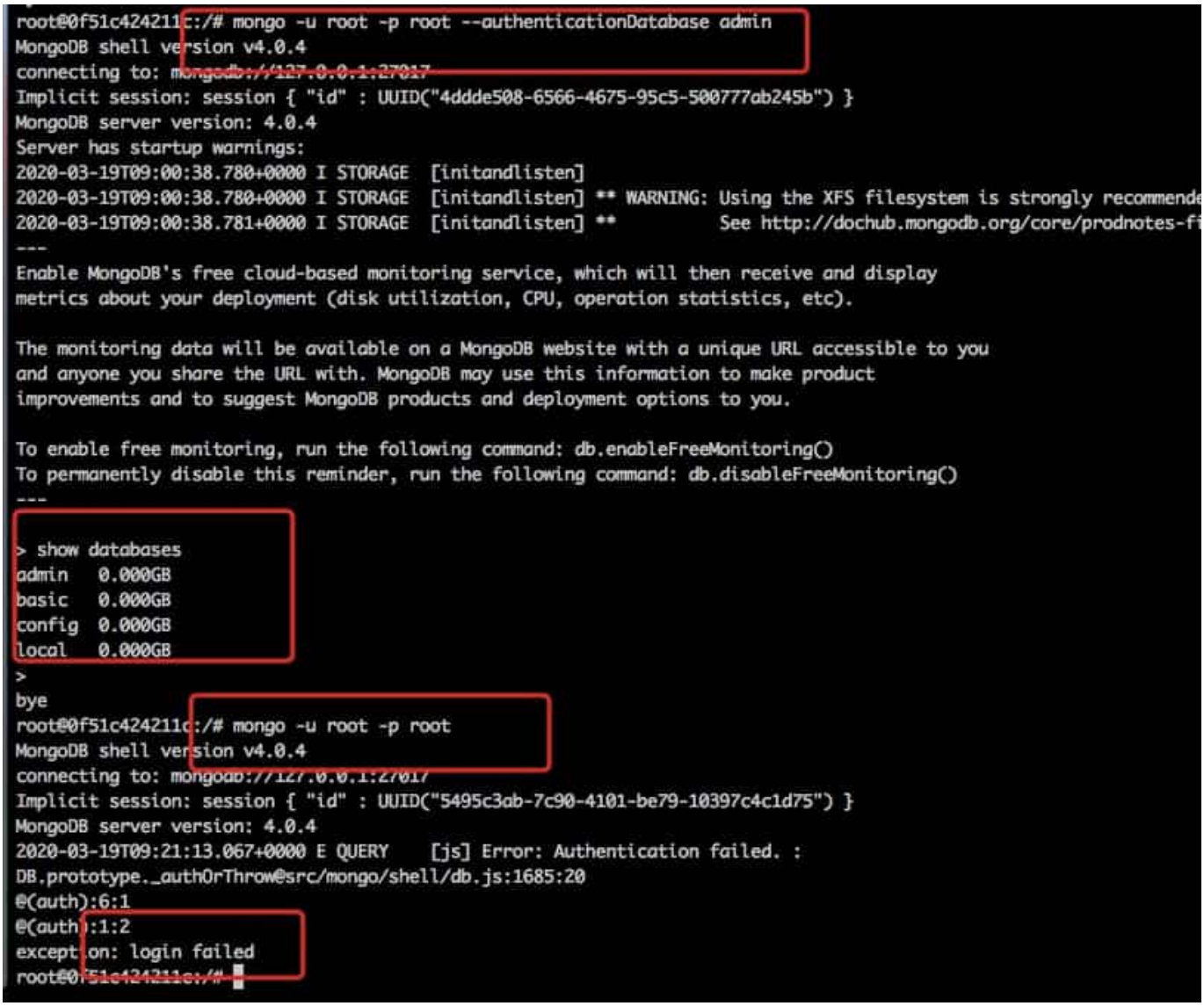
4. 终端控制台

mongodb集成了终端控制台，通过`mongo`进入；

但是当我们设置了登录认证时，有下面两种使用姿势

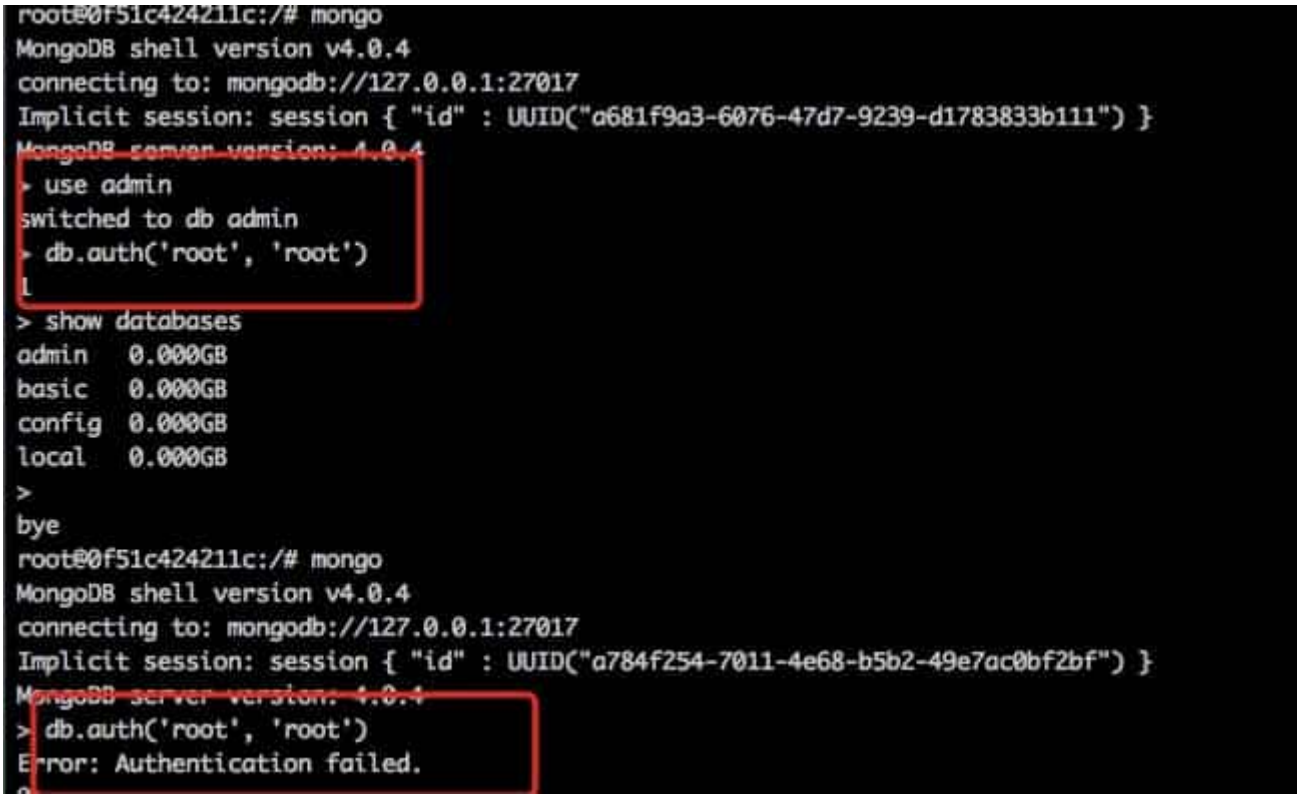
case1

```
# 直接指定用户名密码，注意--authenticationDatabase admin 必须得有
mongo -u root -p root --authenticationDatabase admin
```



case2

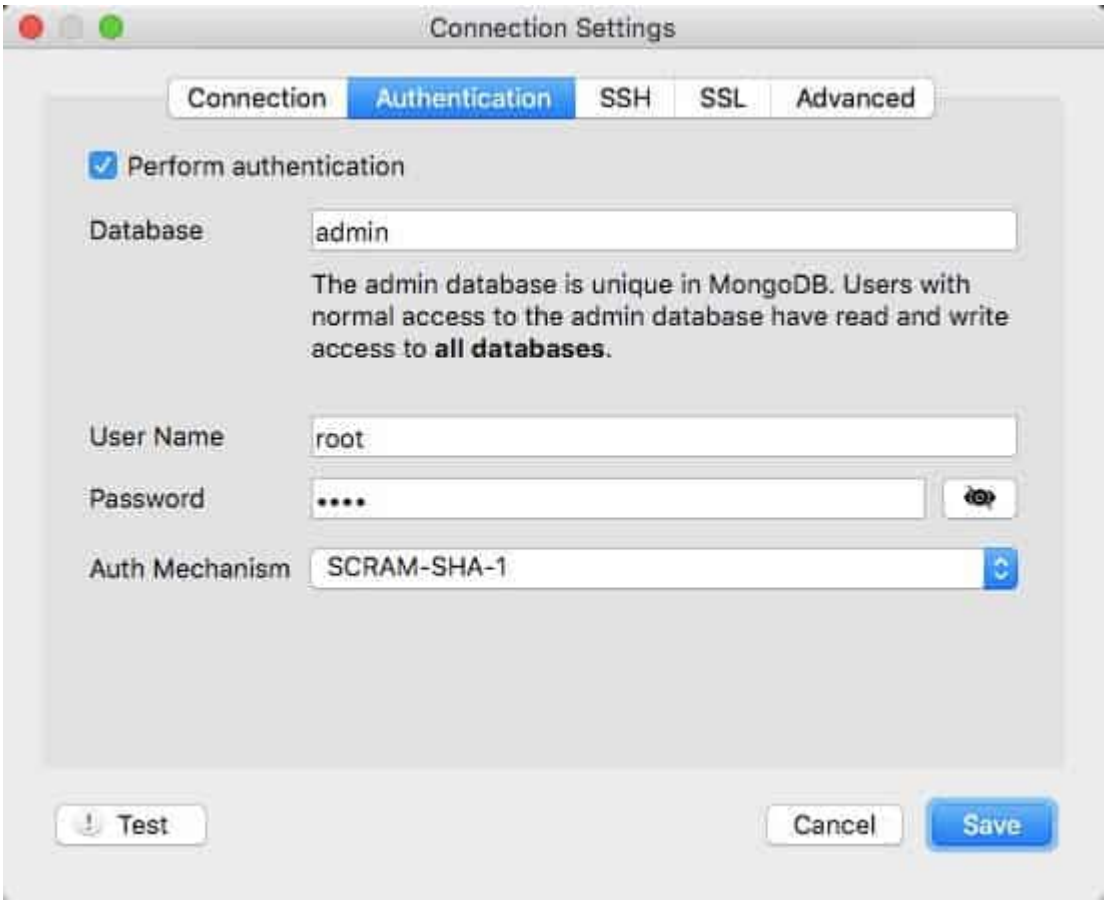
```
mongo

# 下一行不可少
use admin
db.auth('root', 'root')
```



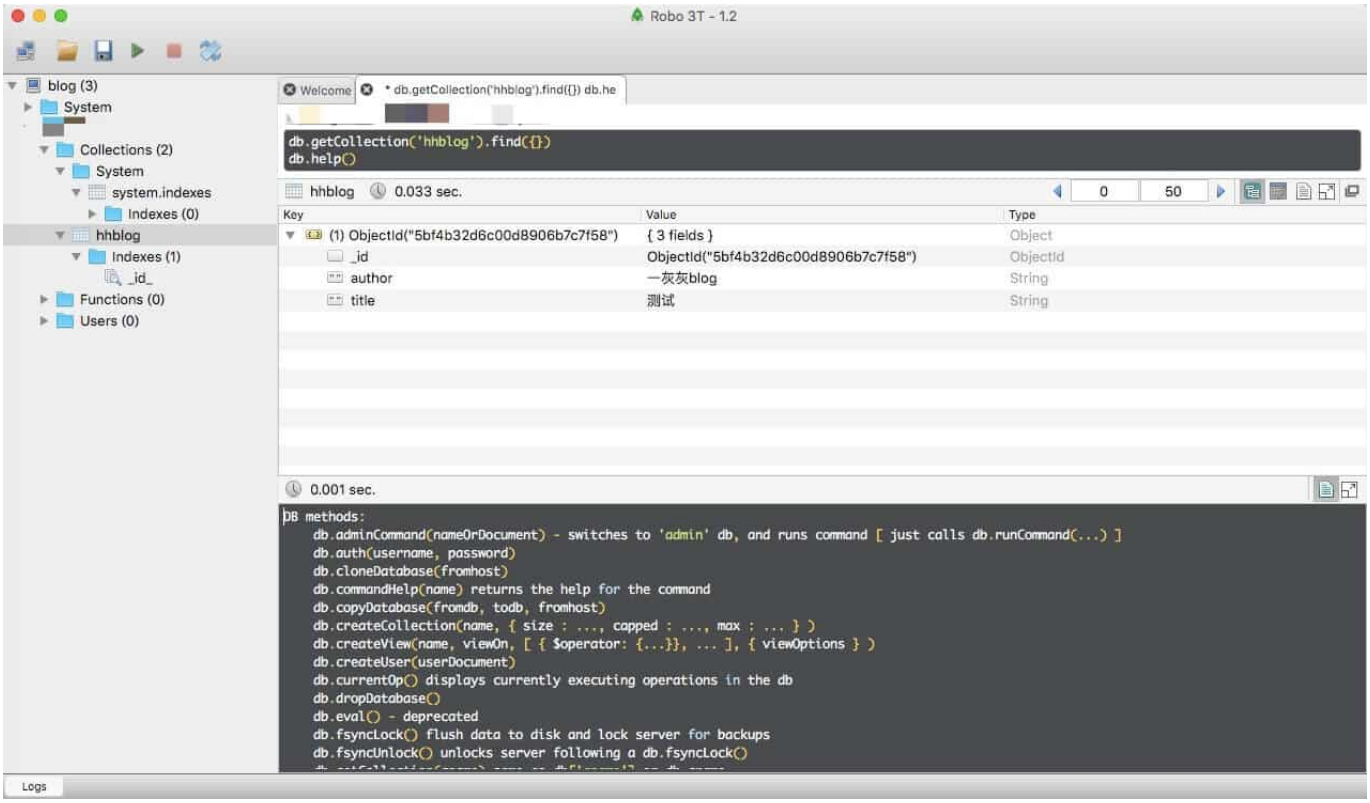
5. 可视化操作工具

终端虽好，使用起来终究不太顺手，可视化工具推荐使用`ROBO 3T`操作mongodb，官网下载地址: https://robomongo.org/

然后配置mongodb连接信息（支持ssh验证方式哦），下面是一个简单的配置



然后就可以通过它来操作mongodb了



基本概念

mongodb和我们通常使用的关系型数据库如mysql，在一些基本概念上有相同之处，但也有一些区别，在进行mongodb的语言介绍之前，有必要先了解一些基础概念

本文将对比sql对一些基础概念进行解释说明

> MongoDB 概念解析

在sql中，会区分database, table, row, column, index, primaryId；在mongodb中也有对应的概念

sql	mongodb	说明
database	db	数据库
table	collection	表/集合
row	document	行/文档
column	field	字段
index	index	索引
primaryId	_id	主键
lock	lock	锁

下面对以上基本概念进行简单说明，详情的后续博文会补上

1. 数据库

数据库可以理解为collection的聚集体，每个mongodb实例可以有多个database，每个database可以有多个collection

常见的几个命令如下：

```
# 显示所有db
show dbs

# 选中某个db
use db_name

# 显示当前选中的db
db

# 删除
db.dropDatabase()
```

2. 集合

document的集合，与table最大的区别是它的结构不是固定的，不需要事先定义字段、类型

首次新增document时，集合被创建；

3. document

文档，也就是具体的数据；bson结构，kv方式

最大的特点是不要求所有的document的结构一致，相同的field的数据类型可以不一致

4. index

索引，同样是用来提高查询效率，避免全盘扫描

5. lock

支持读写锁，document加读锁时，其他读操作ok，写操作禁止；加写锁时，其他读写操作禁止

6. 事务

> MongoDB 4.0 事务实现解析

版本`>= 4.0`，支持事务，支持多文档ACID，后续详细说明

连接

后续的所有文章的基础，都是需要先连上mongodb，然后才能执行各种命令操作；

本文将介绍一下如何连接一个已经启动的mongodb服务器

1. 连接语法

标准URI连接语法：

```
mongodb://[username:password@]host1[:port1][,host2[:port2],...[,hostN[:portN]]]
[/[database][?options]]
```

- `mongodb://` 固定前缀
- `username:password@`：如果开启了用户登录验证，需要指定用户名密码
- `host1:port1`：mongodb服务器的ip/域名 + 端口(不填时，默认为27017)
- `database`：如果指定`username:password@`，连接并验证登陆指定数据库。若不指定，默认打开 test 数据库
- `?options`：是连接选项。如果不使用/database，则前面需要加上

2. 实例

直接连接方式如下，注意这种方式会保留用户名和密码，会有一定的安全风险

连接目标服务器

```
# 连接本地mongodb
mongo mongodb://root:root@127.0.0.1:27017/admin
```

连接多台服务器

```
mongo mongodb://root:root@127.0.0.1:27017,127.0.0.1:27018/admin
```

连接 replica set 三台服务器, 写入操作应用在主服务器 并且分布查询到从服务器

```
mongo mongodb://host1,host2,host3/?slaveOk=true
```

基本工具介绍

mongodb服务器安装完毕之后，提供了一些配套的操作工具，接下来我们有必要认识一下它们，并了解基本用法

0. mongod

启动mongodb实例的主要命令，常见的使用姿势如下

```
mongod --dbpath=/data/mongodb/data --logpath=/data/mongodb/logs --logappend --auth  --
port=27017 --fork
```

1. mongo 命令行使用

mongodb安装完毕之后，会自带一个终端命令行工具，通过它可以连接mongodb，并执行相关命令

a. 连接

介绍三种连接mongodb的姿势

case1

```
mongo --host 目标主机 --port 端口号 -u 用户名 -p 密码 --authenticationDatabase admin
```

case2

```
mongo mongodb://root:root@127.0.0.1:27017/admin
```

case3

上面两种姿势虽然简单，但是用户名密码有暴露的风险，推荐使用下面这种方式

```
mongo --host 目标主机 --port 端口号

use admin
db.auth('用户名', '密码')
```

b. 操作

连接上mongodb服务器之后，就可以执行mongo命令，查看数据库，管理文档，比如下面给几个常见的操作

```
# 查看所有database
show dbs

# 选择数据库(不存在时，创建)
use basic

# 显示所有集合
show collections

# 查看文档
db.demo.findOne({})
```

2. mongoimport/mongoexport

用于导入导出数据，如

将库`database`中的集合`collection`导出到json文件`out.json`

```
bin/mongoexport -h localhost:27107 -u user -p pwd -d database -c collection -o
out.json
```

从json文件导入到目标集合`new_collection`

```
bin/mongoimport -h localhost:27107 -u user -p pwd -d database -c new_collection
./out.json
```

3. mongodump/mongorestore

使用mongodump命令来备份MongoDB数据, 将数据库`basic`的所有集合备份到目录`/tmp/outDir`下

```
mongodump -d basic -u root -p root --authenticationDatabase admin -o /tmp/outDir
```

使用mongorestore恢复，如下

```
# --drop 表示先删除当前数据，然后再恢复，可以不指定
mongorestore -u root -p root --authenticationDatabase admin --drop /tmp/outDir
```

4. mongostate

mongostat是mongodb自带的状态检测工具，在命令行下使用。它会间隔固定时间获取mongodb的当前运行状态，并输出。如果你发现数据库突然变慢或者有其他问题的话，你第一手的操作就考虑采用mongostat来查看mongo的状态。

```
mongostat -u root -p root --authenticationDatabase admin
```

```
root@0f51c424211c:/# mongostat -u root -p root --authenticationDatabase admin
insert query update delete getmore command dirty used flushes vsize   res qrw arw net_in net_out conn             time
    *0    *0     *0     *0       0    12|0  0.0% 0.0%       0 1.09G 76.0M 0|0 1|0  1.44k   65.8k    2 Mar 20 03:45:10.145
    *0    *0     *0     *0       0    11|0  0.0% 0.0%       0 1.09G 76.0M 0|0 1|0  1.39k   63.5k    2 Mar 20 03:45:11.149
    *0    *0     *0     *0       0    11|0  0.0% 0.0%       0 1.09G 76.0M 0|0 1|0  1.39k   63.7k    2 Mar 20 03:45:12.149
    *0    *0     *0     *0       0    12|0  0.0% 0.0%       0 1.09G 76.0M 0|0 1|0  1.45k   66.2k    2 Mar 20 03:45:13.111
    *0    *0     *0     *0       0    11|0  0.0% 0.0%       0 1.09G 76.0M 0|0 1|0  1.39k   63.7k    2 Mar 20 03:45:14.111
    *0    *0     *0     *0       0    11|0  0.0% 0.0%       0 1.09G 76.0M 0|0 1|0  1.39k   63.7k    2 Mar 20 03:45:15.112
    *0    *0     *0     *0       0    12|0  0.0% 0.0%       0 1.09G 76.0M 0|0 1|0  1.40k   64.0k    2 Mar 20 03:45:16.108
    *0    *0     *0     *0       0    12|0  0.0% 0.0%       0 1.09G 76.0M 0|0 1|0  1.39k   63.8k    2 Mar 20 03:45:17.106
    *0    *0     *0     *0       0    11|0  0.0% 0.0%       0 1.09G 76.0M 0|0 1|0  1.38k   63.4k    2 Mar 20 03:45:18.112
    *0    *0     *0     *0       0    11|0  0.0% 0.0%       0 1.09G 76.0M 0|0 1|0  1.38k   62.9k    2 Mar 20 03:45:19.124
```

5. mongotop

mongotop提供每个集合的水平的统计数据，默认每s输出一次

```
root@0f51c424211c:/# mongotop -u root -p root --authenticationDatabase admin
2020-03-20T03:46:20.506+0000    connected to: 127.0.0.1

                    ns    total    read    write    2020-03-20T03:46:21Z
     admin.system.roles      0ms     0ms      0ms
     admin.system.users      0ms     0ms      0ms
   admin.system.version      0ms     0ms      0ms
             basic.col       0ms     0ms      0ms
            basic.demo       0ms     0ms      0ms
       basic.system.js       0ms     0ms      0ms
     basic.system.users      0ms     0ms      0ms
config.system.sessions       0ms     0ms      0ms
      local.startup_log      0ms     0ms      0ms
   local.system.replset      0ms     0ms      0ms

                    ns    total    read    write    2020-03-20T03:46:22Z
     admin.system.roles      0ms     0ms      0ms
     admin.system.users      0ms     0ms      0ms
```

数据库 Database

我们通常把mongodb叫文档型数据库，mysql叫关系型数据库，influxdb叫时序数据库，如果熟悉这三个的话，会发现他们都有一个`database`，它是`collection/table/measurement`的上一级，可以简单的把它理解为更高层级的集合，方便统一管理/权限划分/业务拆分

下面简单介绍一下database的基础操作

1. 创建数据库

当数据库不存在时，通过`use + 数据库`命令可以用来创建数据库；当数据库存在时，表示选中

```
use dbname
```

2. 查看数据库

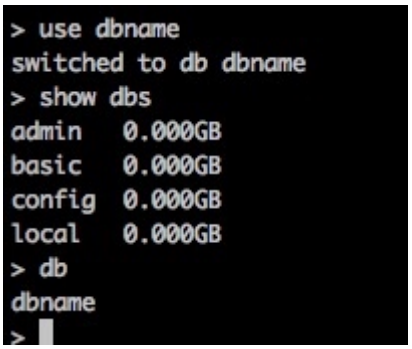
通过 `db` 查看当前的数据库

通过 `show dbs` 查看当前的数据库列表

请注意，新创建一个数据库时，直接使用`show dbs`命令，并不会显示出来，如下

为了显示这个数据库，需要插入一个文档

```
db.dbname.insert({"name": "一灰灰blog", "age": 18})
```



3. 删除数据库

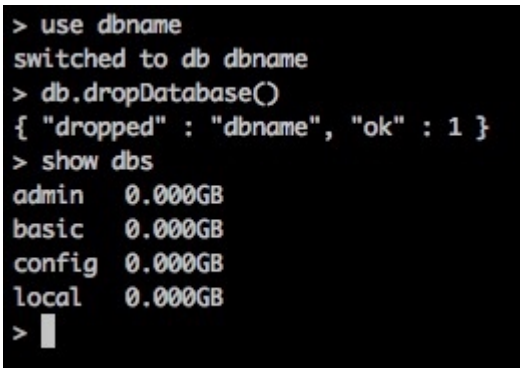
对于数据库而言，任何删除命令都需要慎重处理，一不小心就得跑路了。。。

命令如下: `db.dropDatabase()`

实例说明:

一般来说我们需要删除时，两步走

```
# 选中db
use dbname
# 执行删除命令
db.dropDatabase()
```



4. 潜规则

需要注意，有三个数据库属于预留的，有特殊的作用，不能新建同名的数据

- admin: 将一个用户添加到这个数据库，这个用户自动继承所有数据库的权限; 一些特定的服务器端命令也只能从这个数据库运行，比如列出所有的数据库或者关闭服务器。
- local: 这个数据永远不会被复制，可以用来存储限于本地单台服务器的任意集合
- config: 当Mongo用于分片设置时，config数据库在内部使用，用于保存分片的相关信息。

命名规则：

- 不能是空字符串
- 不能含有特殊字符（如 ` `, `.`, `$`, `\`, `/`, `\0`
- 小写
- 最多64字节

集合 Collection

集合，相当于关系型数据库中的table，在mongodb中，集合的概念非常贴切，属于文档(Document)的集合

其最大的特点是：

- 没有固定的结构

1. 创建集合

创建命令如: `db.createCollection(name, options)`

重点看一下参数`options`的可选项

- capped: true，表示创建固定大小的集合，需要指定size；超过数量之后，覆盖最早的文档

- size: 固定集合时配套使用，KB为单位
- autoIndexId: 自动为 `_id` 添加索引，默认true
- max: 固定集合时，文档的最大数量

一个简单的实例

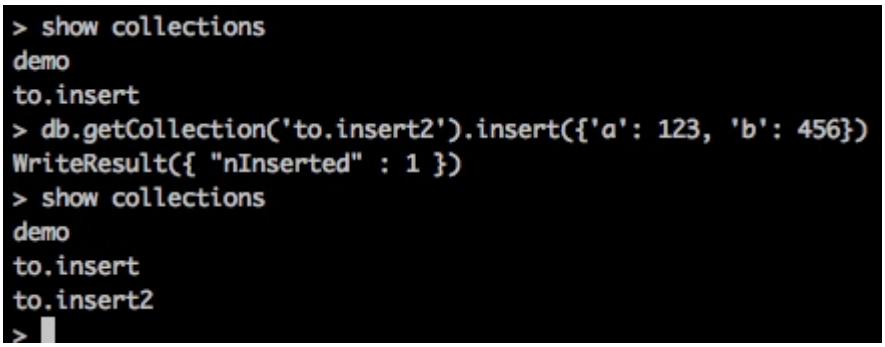
```
# 创建一个名为 to.insert 的集合
db.createCollection('to.insert')
```

除此之外，新插入一个文档时，集合若不存在，也会创建对应的集合，如

```
# 不推荐在集合名中包含点号，如果没有点号时，可以通过 db.test_collection.insert({'a': 1})来插入数据，更简单
db.getCollection('to.insert2').insert({'a': 123, 'b': 456})
```

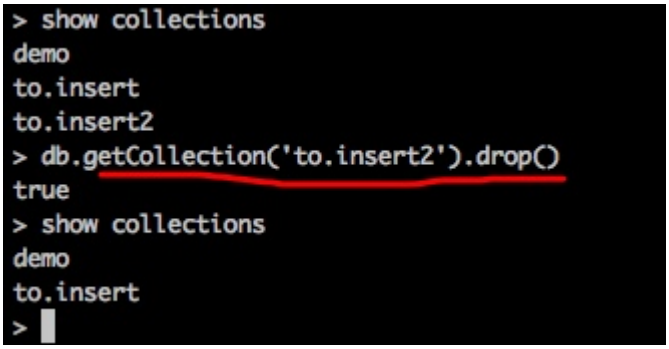
2. 查看集合

通过 `show collections` 查看数据库下的集合列表



3. 删除集合

通过命令 `db.col.drop()` 来删除



4. 命名规则

- 不能全是空白字符
- 不应包含特殊字符
- 不要以`system.`开头

文档 Document 插入姿势

文档相当于关系数据库中数据行，也是我们最关心的数据本身；以BSON格式存储（和json区别不大）

我们通常所说业务开发者的CURD四大技能，在mongodb中，就是针对Document而言，接下来我们先看一下文档的新增使用姿势

1. 基本语法

插入语法：`db.collection.insert()`

因为集合不要求定义数据结构，所以插入的文档格式理论上可以完全不一样，可以拥有完全不同的数据结构，相同的字段拥有不同的数据类型

2. 实例演示

下面给出几个实例进行说明

基本数据类型插入

```
# 插入两个数据，注意age的数据类型不一样哦
db.doc_demo.insert({'name': 'yihui', 'age': 18})
db.doc_demo.insert({'address': 'China', 'age': 18.8})
```

数组类型插入

```
db.doc_demo.insert({'name': 'yihui', 'skill': ['java', 'python', 'php', 'js']})
```

Object类型插入

```
db.doc_demo.insert({'name': 'yihui', 'site': {'blog':'https://blog.hhui.top',
'spring': 'https://spring.hhui.top'}})
```

```
> db.doc_demo.find({})
{ "_id" : ObjectId("5e786582b0d677183afba746"), "name" : "yihui", "age" : 18 }
{ "_id" : ObjectId("5e78659ab0d677183afba747"), "address" : "China", "age" : 18.8 }
{ "_id" : ObjectId("5e786622b0d677183afba748"), "name" : "yihui", "skill" : [ "java", "python", "php", "js" ] }
{ "_id" : ObjectId("5e786680b0d677183afba749"), "name" : "yihui", "site" : { "blog" : "https://blog.hhui.top", "spring" : "https://spring.hhui.top" } }
>
```

3. 数据类型

mongodb支持的基本数据类型，除了我们常见的string,int,float,boolean之外，还有一些其他的;

数据类型	说明
String	字符串， UTF8编码
Integer	整型，32/64位
Boolean	布尔
Double	浮点
Min/Max keys	将一个值与 BSON（二进制的 JSON）元素的最低值和最高值相对比
Array	数组
Timestamp	时间戳，记录文档修改或添加的具体时间
Object	内嵌文档
Null	创建空值
Symbol	符号。该数据类型基本上等同于字符串类型，但不同的是，它一般用于采用特殊符号类型的语言。
Date	日期，用 UNIX 时间格式来存储当前日期或时间。你可以指定自己的日期时间：创建 Date 对象，传入年月日信息。
ObjectID	对象ID
Binary Data	二进制
code	代码类型。用于在文档中存储 JavaScript 代码。
Regular expression	正则表达式类型。用于存储正则表达式。

ObjectId 类似唯一主键，可以很快的去生成和排序，包含 12 bytes，含义是：

- 前 4 个字节表示创建 unix 时间戳,格林尼治时间 UTC 时间，比北京时间晚了 8 个小时
- 接下来的 3 个字节是机器标识码
- 紧接的两个字节由进程 id 组成 PID
- 最后三个字节是随机数

文档 Document 删除姿势

前面一篇介绍了插入文档的使用姿势，这一篇则主要介绍删除的使用case

1. 基本语法

```
db.collection.remove(
   <query>,
    {
     justOne: <boolean>,
     writeConcern: <document>
   }
)
```

第一个为需要删除的匹配条件；第二个表示是否只删除一个，默认是false，删除所有满足条件的文档

注意

- 当query为空时，表示删除所有文档，高危操作，谨慎执行

2. 实例演示

借用给我们上一篇插入的文档来进行演示，当前存在的文档为

```
> db.doc_demo.find({})
{ "_id" : ObjectId("5e786582b0d677183afba746"), "name" : "yihui", "age" : 18 }
{ "_id" : ObjectId("5e78659ab0d677183afba747"), "address" : "China", "age" : 18.8 }
{ "_id" : ObjectId("5e786622b0d677183afba748"), "name" : "yihui", "skill" : [ "java",
"python", "php", "js" ] }
{ "_id" : ObjectId("5e786680b0d677183afba749"), "name" : "yihui", "site" : { "blog" :
"https://blog.hhui.top", "spring" : "https://spring.hhui.top" } }
```

根据id进行删除

```
db.doc_demo.remove({"_id": ObjectId("5e786582b0d677183afba746")})
```

根据name删除第一个满足条件的记录

```
db.doc_demo.remove({"name":"yihui"}, {justOne: true})
```

再次查看剩下的内容如下：

```
> db.doc_demo.find({})
{ "_id" : ObjectId("5e78659ab0d677183afba747"), "address" : "China", "age" : 18.8 }
{ "_id" : ObjectId("5e786680b0d677183afba749"), "name" : "yihui", "site" : { "blog" : "https://blog.hhui.top", "spring" : "https://spring.hhui.top" } }
>
```

文档 Document 更新姿势

本篇介绍update/save两种方法提供的更新姿势

1. update

用于更新已经存在的文档，语法如下

```
db.collection.update(
   <query>,
   <update>,
   {
     upsert: <boolean>,
     multi: <boolean>,
     writeConcern: <document>
   }
)
```

- query: 查询条件
- update: 更新语句
- upsert: （可选）true, 不存在update的记录时插入；默认是false，不插入
- multi: (可选) true，表示更新所有满足条件的记录；默认false，只更新第一条
- writeConcern: (可选)，抛出异常的级别

插入两条用于测试的数据

```
db.doc_demo.insert({'name': '一灰灰', 'age': 19, 'skill': ['java', 'python', 'sql']})
db.doc_demo.insert({'name': '一灰灰blog', 'age': 20, 'skill': ['web', 'shell', 'js']})
```

下面给出几个更新的实例

更新age

```
# 将name为"一灰灰"的文档age + 1
db.doc_demo.update({'name':'一灰灰'}, {$inc: {'age': 1}})
# 修改name
db.doc_demo.update({'name':'一灰灰'}, {$set: {'name': '一灰灰Blog'}})
```

```
> db.doc_demo.update({'name':'一灰灰'}, {$inc: {'age': 1}})
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
> db.doc_demo.find({})
{ "_id" : ObjectId("5e7b5ac10172dc950171c488"), "name" : "一灰灰", "age" : 20, "skill" : [ "java", "python", "sql" ] }
{ "_id" : ObjectId("5e7b5ac40172dc950171c489"), "name" : "一灰灰blog", "age" : 20, "skill" : [ "web", "shell", "js" ] }
> db.doc_demo.update({'name':'一灰灰'}, {$set: {'name': '一灰灰Blog'}})
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
> db.doc_demo.find({})
{ "_id" : ObjectId("5e7b5ac10172dc950171c488"), "name" : "一灰灰Blog", "age" : 20, "skill" : [ "java", "python", "sql" ] }
{ "_id" : ObjectId("5e7b5ac40172dc950171c489"), "name" : "一灰灰blog", "age" : 20, "skill" : [ "web", "shell", "js" ] }
```

更新所有age为20的文档，新增一个tag成员

```
db.doc_demo.update({'age': 20}, {$set: {'tag': 1}}, {multi:true})
```

```
> db.doc_demo.update({'age': 20}, {$set: {'tag': 1}}, {multi:true})
WriteResult({ "nMatched" : 2, "nUpserted" : 0, "nModified" : 2 })
> db.doc_demo.find({})
{ "_id" : ObjectId("5e7b5ac10172dc950171c488"), "name" : "一灰灰Blog", "age" : 20, "skill" : [ "java", "python", "sql" ], "tag" : 1 }
{ "_id" : ObjectId("5e7b5ac40172dc950171c489"), "name" : "一灰灰blog", "age" : 20, "skill" : [ "web", "shell", "js" ], "tag" : 1 }
```

更新一个不存在的文档

```
db.doc_demo.update({'name': '一灰灰'}, {$set: {'age': 18, 'sex': 'man'}}, {upsert:
true})
```

```
> db.doc_demo.update({'name': '一灰灰'}, {$set: {'age': 18, 'sex': 'man'}}, {upsert: true})
WriteResult({
        "nMatched" : 0,
        "nUpserted" : 1,
        "nModified" : 0,
        "_id" : ObjectId("5e7b5bb085a742842d2e23fc")
})
> db.doc_demo.find({})
{ "_id" : ObjectId("5e7b5ac10172dc950171c488"), "name" : "一灰灰Blog", "age" : 20, "skill" : [ "java", "python", "sql" ], "tag" : 1 }
{ "_id" : ObjectId("5e7b5ac40172dc950171c489"), "name" : "一灰灰blog", "age" : 20, "skill" : [ "web", "shell", "js" ], "tag" : 1 }
{ "_id" : ObjectId("5e7b5bb085a742842d2e23fc"), "name" : "一灰灰", "age" : 18, "sex" : "man" }
```

2. save

save最大的特点是覆盖，用新的文档完全覆盖旧的文档；而update，则是更新指定的field

语法如下：

```
db.collection.save(
   <document>,
   {
     writeConcern: <document>
   }
)
```

举例如下

```
db.doc_demo.save({'name': '一灰灰', 'age': 22, 'hobby': ['reading', 'walking']})
```

```
> db.doc_demo.save({'name': '一灰灰', 'age': 22, 'hobby': ['reading', 'walking']})
WriteResult({ "nInserted" : 1 })
> db.doc_demo.find({})
{ "_id" : ObjectId("5e7b5ac10172dc950171c488"), "name" : "一灰灰Blog", "age" : 20, "skill" : [ "java", "python", "sql" ], "tag" : 1 }
{ "_id" : ObjectId("5e7b5ac40172dc950171c489"), "name" : "一灰灰blog", "age" : 20, "skill" : [ "web", "shell", "js" ], "tag" : 1 }
{ "_id" : ObjectId("5e7b5bb085a742842d2e23fc"), "name" : "一灰灰", "age" : 18, "sex" : "man" }
{ "_id" : ObjectId("5e7b5c2e0172dc950171c48a"), "name" : "一灰灰", "age" : 22, "hobby" : [ "reading", "walking" ] }
>
```

那么问题来了，**怎样判定是新增一条记录，还是覆盖已经存在的记录呢?**

- 有唯一键来判定
- 即：如果save的文档中，某个field有唯一性要求，那么当数据库中存在这个field文档文档时，执行覆盖操作；否则执行插入

举例如下, 指定ObjectId

```
db.doc_demo.save({ "_id" : ObjectId("5e7b5c2e0172dc950171c48a"), "name" : "一灰灰New",
"age" : 18, "hobby" : [ "play game" ] })
```

```
> db.doc_demo.save({ "_id" : ObjectId("5e7b5c2e0172dc950171c48a"), "name" : "一灰灰New", "age" : 18, "hobby" : [ "play game" ] })
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
> db.doc_demo.find({})
{ "_id" : ObjectId("5e7b5ac10172dc950171c488"), "name" : "一灰灰Blog", "age" : 20, "skill" : [ "java", "python", "sql" ], "tag" : 1 }
{ "_id" : ObjectId("5e7b5ac40172dc950171c489"), "name" : "一灰灰blog", "age" : 20, "skill" : [ "web", "shell", "js" ], "tag" : 1 }
{ "_id" : ObjectId("5e7b5bb085a742842d2e23fc"), "name" : "一灰灰", "age" : 18, "sex" : "man" }
{ "_id" : ObjectId("5e7b5c2e0172dc950171c48a"), "name" : "一灰灰New", "age" : 18, "hobby" : [ "play game" ] }
```

文档 Document 查询基础篇

MongoDb文档查询，主要借助find方法来完成，在实际的业务开发中，为了满足各种复杂的业务场景，查询的姿势也是各种各样，本篇则主要介绍基本的使用姿势，不涉及到聚合、排序、分页相关内容

1. 查询语法

查询语法定义比较简单，复杂的是查询条件的组合；语法定义如下

```
db.collection.find(query, projection)
```

- query: 查询条件，如果不填，则表示查询所有文档
- projection: 查询需要返回的field，如果不填则返回所有的数据

此外为了mongo-cli的返回结果更加友好，可以在最后添加`.pretty()`，使输出更友好

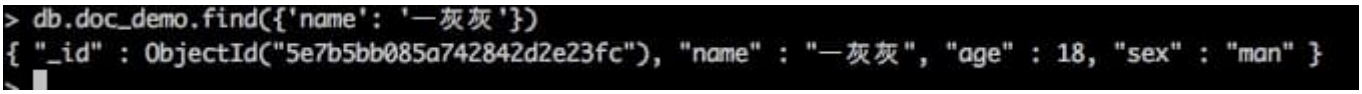
2. 查询所有

```
db.doc_demo.find()
```

```
> db.doc_demo.find()
{ "_id" : ObjectId("5e7b5ac10172dc950171c488"), "name" : "一灰灰Blog", "age" : 20, "skill" : [ "java", "python", "sql" ], "tag" : 1 }
{ "_id" : ObjectId("5e7b5ac40172dc950171c489"), "name" : "一灰灰blog", "age" : 20, "skill" : [ "web", "shell", "js" ], "tag" : 1 }
{ "_id" : ObjectId("5e7b5bb085a742842d2e23fc"), "name" : "一灰灰", "age" : 18, "sex" : "man" }
{ "_id" : ObjectId("5e7b5c2e0172dc950171c48a"), "name" : "一灰灰New", "age" : 18, "hobby" : [ "play game" ] }
>
```

3. 根据条件精准查询

```
db.doc_demo.find({'name': '一灰灰'})
```



4. 数字比较查询

对于数字类型的field，可以借助符号 `$gt` (>), `$get` (>=), `$lt` (<), `$lte` (<=), `$ne` (!=) 来表示具体的操作

```
#查询age>18的文档
db.doc_demo.find({'age': {$gt: 18}})

# 查询age<20的文档
db.doc_demo.find({'age': {$lt: 20}})
```

5. 模糊查询

在mysql中有一个like用于模糊查询，在mongodb中，同样支持基于正则的模糊查询

```
# 查询name以灰灰结尾的文档
db.doc_demo.find({'name': /灰灰$/})
# 查询name中包含 lo 字符的文档
db.doc_demo.find({'name': /lo/})
# 查询name中包含l, g字符的文档
db.doc_demo.find({'name': /l.g/})
# 查询name以一灰灰开头的文档
db.doc_demo.find({'name': /^一灰灰/})
```

```
> db.doc_demo.find({'name': /灰灰$/})
{ "_id" : ObjectId("5e7b5bb085a742842d2e23fc"), "name" : "一灰灰", "age" : 18, "sex" : "man" }
> db.doc_demo.find({'name': /lo/})
{ "_id" : ObjectId("5e7b5ac10172dc950171c488"), "name" : "一灰灰Blog", "age" : 20, "skill" : [ "java", "python", "sql" ], "tag" : 1 }
{ "_id" : ObjectId("5e7b5ac40172dc950171c489"), "name" : "一灰灰blog", "age" : 20, "skill" : [ "web", "shell", "js" ], "tag" : 1 }
> db.doc_demo.find({'name': /l.g/})
{ "_id" : ObjectId("5e7b5ac10172dc950171c488"), "name" : "一灰灰Blog", "age" : 20, "skill" : [ "java", "python", "sql" ], "tag" : 1 }
{ "_id" : ObjectId("5e7b5ac40172dc950171c489"), "name" : "一灰灰blog", "age" : 20, "skill" : [ "web", "shell", "js" ], "tag" : 1 }
> db.doc_demo.find({'name': /^一灰灰/})
{ "_id" : ObjectId("5e7b5ac10172dc950171c488"), "name" : "一灰灰Blog", "age" : 20, "skill" : [ "java", "python", "sql" ], "tag" : 1 }
{ "_id" : ObjectId("5e7b5ac40172dc950171c489"), "name" : "一灰灰blog", "age" : 20, "skill" : [ "web", "shell", "js" ], "tag" : 1 }
{ "_id" : ObjectId("5e7b5bb085a742842d2e23fc"), "name" : "一灰灰", "age" : 18, "sex" : "man" }
{ "_id" : ObjectId("5e7b5c2e0172dc950171c48a"), "name" : "一灰灰New", "age" : 18, "hobby" : [ "play game" ] }
>
```

6. and条件

多个查询条件需要满足时，并不需要什么特殊的操作，只需要在查询bson中，加上多个条件即可

```
# 查询age > 18, 且name为 一灰灰blog的文档
db.doc_demo.find({'age': {$gt: 18}, 'name':'一灰灰blog'})
```

```
> db.doc_demo.find({'age': {$gt: 18}, 'name':'一灰灰blog'})
{ "_id" : ObjectId("5e7b5ac40172dc950171c489"), "name" : "一灰灰blog", "age" : 20, "skill" : [ "web", "shell", "js" ], "tag" : 1 }
>
```

7. or条件

和and不需要额外的操作不同，or条件需要借助 `$or` 来实现，语法如下

```
db.collection.find({$or: [{queyr1, query2}]})
```

实例如下：

```
# 查询age > 18, 且name为 一灰灰blog的文档 或 age < 20 且name为一灰灰的文档
db.doc_demo.find({$or: [{'age': {$gt: 18}, 'name':'一灰灰blog'}, {'age': {$lt: 20},
'name': '一灰灰'}]})
```

```
> db.doc_demo.find({$or: [{'age': {$gt: 18}, 'name':'一灰灰blog'}, {'age': {$lt: 20}, 'name': '一灰灰'}]})
{ "_id" : ObjectId("5e7b5ac40172dc950171c489"), "name" : "一灰灰blog", "age" : 20, "skill" : [ "web", "shell", "js" ], "tag" : 1 }
{ "_id" : ObjectId("5e7b5bb085a742842d2e23fc"), "name" : "一灰灰", "age" : 18, "sex" : "man" }
>
```

8. 限制返回成员

有些时候我们只需要获取文档中的部分成员，可以在第二个参数中进行指定，规则如下

- `成员名: 1`： 表示这个成员需要返回

- `成员名: 0`： 表示这个成员不返回

```
# 表示返回的结果中，除了_id之外，其他的正常返回
db.doc_demo.find({}, {'_id': 0})

# 表示返回的结果中，除了_id之外，就只要name和age
db.doc_demo.find({}, {'name': 1, 'age': 1})
```

```
> db.doc_demo.find({}, {'_id': 0})
{ "name" : "一灰灰Blog", "age" : 20, "skill" : [ "java", "python", "sql" ], "tag" : 1 }
{ "name" : "一灰灰blog", "age" : 20, "skill" : [ "web", "shell", "js" ], "tag" : 1 }
{ "name" : "一灰灰", "age" : 18, "sex" : "man" }
{ "name" : "一灰灰New", "age" : 18, "hobby" : [ "play game" ] }
> db.doc_demo.find({}, {'name': 1, 'age': 1})
{ "_id" : ObjectId("5e7b5ac10172dc950171c488"), "name" : "一灰灰Blog", "age" : 20 }
{ "_id" : ObjectId("5e7b5ac40172dc950171c489"), "name" : "一灰灰blog", "age" : 20 }
{ "_id" : ObjectId("5e7b5bb085a742842d2e23fc"), "name" : "一灰灰", "age" : 18 }
{ "_id" : ObjectId("5e7b5c2e0172dc950171c48a"), "name" : "一灰灰New", "age" : 18 }
```

请注意，一般在使用了 `成员名: 1` 来指定返回field时，会自动返回 `_id`，如果不需要，请显示加上 `_id: 0`

9. field类型查询

根据field的成员类型来作为查询条件，一般有两种方式，这里只介绍更优雅的，语法如下

```
{field: {$type: '类型'}}
```

举例说明

```
db.doc_demo.find({'skill': {$type: 'array'}})
```

```
> db.doc_demo.find({'skill': {$type: 'array'}})
{ "_id" : ObjectId("5e7b5ac10172dc950171c488"), "name" : "一灰灰Blog", "age" : 20, "skill" : [ "java", "python", "sql" ], "tag" : 1 }
{ "_id" : ObjectId("5e7b5ac40172dc950171c489"), "name" : "一灰灰blog", "age" : 20, "skill" : [ "web", "shell", "js" ], "tag" : 1 }
```

10. 存在查询

mongodb的一个特点就是集合的结构不固定，所以某个成员可能存在也可能不存在，所以当我们的查询条件中需要加一个是否存在的判断时，可以如下

```
# 查询tag存在的文档
db.doc_demo.find({'tag': {$exists:true}})
# 查询tag不存在的文档
db.doc_demo.find({'tag': null})
```

```
> db.doc_demo.find({'tag': {$exists:true}})
{ "_id" : ObjectId("5e7b5ac10172dc950171c488"), "name" : "一灰灰Blog", "age" : 20, "skill" : [ "java", "python", "sql" ], "tag" : 1 }
{ "_id" : ObjectId("5e7b5ac40172dc950171c489"), "name" : "一灰灰blog", "age" : 20, "skill" : [ "web", "shell", "js" ], "tag" : 1 }
> db.doc_demo.find({'tag': null})
{ "_id" : ObjectId("5e7b5bb085a742842d2e23fc"), "name" : "一灰灰", "age" : 18, "sex" : "man" }
{ "_id" : ObjectId("5e7b5c2e0172dc950171c48a"), "name" : "一灰灰New", "age" : 18, "hobby" : [ "play game" ] }
>
```

文档 Document 查询高级篇

上一篇的mongodb查询，主要介绍的是一些基本操作，当然有基本就高阶操作；

本文将带来更多的查询姿势

- 排序
- 分页
- 聚合

1. 排序

在mongodb中，使用sort方法进行排序，语法如下

```
db.collection.find().sort({key: 1})
```

请注意，sort内部是一个对象，key为field，value为1或者-1，其中1表示升序，-1表示降序

实例说明，根据age进行排序

```
db.doc_demo.find().sort({'age': 1})
```

输出如下:

```
> db.doc_demo.find().sort({'age': 1})
{ "_id" : ObjectId("5e7b5bb085a742842d2e23fc"), "name" : "一灰灰", "age" : 18, "sex" : "man" }
{ "_id" : ObjectId("5e7b5c2e0172dc950171c48a"), "name" : "一灰灰New", "age" : 18, "hobby" : [ "play game" ] }
{ "_id" : ObjectId("5e7b5ac10172dc950171c488"), "name" : "一灰灰Blog", "age" : 20, "skill" : [ "java", "python", "sql" ], "tag" : 1 }
{ "_id" : ObjectId("5e7b5ac40172dc950171c489"), "name" : "一灰灰blog", "age" : 20, "skill" : [ "web", "shell", "js" ], "tag" : 1 }
>
```

上面的演示属于常规的操作，但是针对mongodb的特点，自然会有一些疑问

q1: 如果某个文档没有包含这个field，排序是怎样的?

```
db.doc_demo.find().sort({'tag': 1})
```

从输出来看，升序时，不包含这个field的文档，在最前面；降序时，不包含这个field的文档，在最后面

q2: 支持多个field排序吗?

原则上一般不建议多个field的排序（比较影响性能），但对于数据库而言，你得支持吧

```
# 在开始之前，先改一下tag，让文档不完全一致
db.doc_demo.update({"_id": ObjectId("5e7b5ac10172dc950171c488")}, {$set: {'tag': 2}})
db.doc_demo.update({"_id": ObjectId("5e7b5bb085a742842d2e23fc")}, {$set: {'tag': 2}})

# 先根据age进行升序排，当age相同的，根据tag降序排
db.doc_demo.find().sort({'age': 1, 'tag': -1})
# 先根据tag进行升序排，tag相同的，根据age升序排
db.doc_demo.find().sort({'tag': 1, 'age': 1})
```

请注意上的输出，在涉及到多个field排序时，优先根据第一个进行排序，当文档的field相同时，再根据后面的进行排序

2. 分页

当文档很多时，我们不可能把所有的文档一次返回，所以就有了常见的分页，在sql中我们一般使用`limit offset`来实现分页，在mongodb中也差不多

limit()

限制返回的文档数

```
db.doc_demo.find().limit(2)
```

skip()

使用limit进行返回条数限制，使用skip进行分页，表示跳过前面的n条数据

```
# 跳过第一条数据，返回两条； 相当于返回第2、3条数据
db.doc_demo.find().limit(2).skip(1)
```

3. 聚合

使用`aggregate()`来实现聚合，用于处理求和、平均值，最大值，分组等

数据准备:

```
{ "_id" : ObjectId("5e7b5ac10172dc950171c488"), "name" : "一灰灰blog", "age" : "19",
"skill" : [ "java", "python", "sql" ], "tag" : 2 }
{ "_id" : ObjectId("5e7b5ac40172dc950171c489"), "name" : "一灰灰blog", "age" : 20,
"skill" : [ "web", "shell", "js" ], "tag" : 1 }
{ "_id" : ObjectId("5e7b5bb085a742842d2e23fc"), "name" : "一灰灰", "age" : 18, "sex" :
"man", "tag" : 2 }
{ "_id" : ObjectId("5e7b5c2e0172dc950171c48a"), "name" : "一灰灰", "age" : 18, "hobby"
: [ "play game" ] }
```

分组查询

根据name进行分组统计

```
# 根据name进行分组，统计文档数量
# 相当于sql中的  select name, count(1) from doc_demo group by name
db.doc_demo.aggregate([{$group: {_id: "$name", size: {$sum: 1}}}])
```

```
> db.doc_demo.aggregate([{$group: {_id: "$name", size: {$sum: 1}}}])
{ "_id" : "一灰灰", "size" : 2 }
{ "_id" : "一灰灰blog", "size" : 2 }
```

请注意，分组的条件中

- `_id`：表示根据哪个字段进行分组
- `size: {}`：表示聚合条件指定，将结果输出到名为size的field中
- `filed`名前加`$`进行指定

当前mongodb支持的聚合表达式包括:

表达式	说明	举例说明
sum	求和	`db.doc_demo.aggregate([{$group: {_id: "$name", size: {$sum: '$age'}}}])`
avg	平均值	`db.doc_demo.aggregate([{$group: {_id: "$name", size: {$avg: '$age'}}}])`
min	取最小	`db.doc_demo.aggregate([{$group: {_id: "$name", age: {$min: '$age'}}}])`
max	取最大	`db.doc_demo.aggregate([{$group: {_id: "$name", age: {$max: '$age'}}}])`
push	结果插入到一个数组中	`db.doc_demo.aggregate([{$group: {_id: "$name", age: {$push: '$age'}}}])`
addToSet	结果插入集合，过滤重复	`db.doc_demo.aggregate([{$group: {_id: "$name", age: {$addToSet: '$age'}}}])`
first	第一个	`db.doc_demo.aggregate([{$group: {_id: "$name", age: {$first: '$age'}}}])`
last	最后一个	`db.doc_demo.aggregate([{$group: {_id: "$name", age: {$last: '$age'}}}])`

```
> db.doc_demo.aggregate([{$group: {_id: "$name", size: {$sum: '$age'}}}])
{ "_id" : "一灰灰", "size" : 36 }
{ "_id" : "一灰灰blog", "size" : 39 }
> db.doc_demo.aggregate([{$group: {_id: "$name", size: {$avg: '$age'}}}])
{ "_id" : "一灰灰", "size" : 18 }
{ "_id" : "一灰灰blog", "size" : 19.5 }
> db.doc_demo.aggregate([{$group: {_id: "$name", age: {$min: '$age'}}}])
{ "_id" : "一灰灰", "age" : 18 }
{ "_id" : "一灰灰blog", "age" : 19 }
> db.doc_demo.aggregate([{$group: {_id: "$name", age: {$max: '$age'}}}])
{ "_id" : "一灰灰", "age" : 18 }
{ "_id" : "一灰灰blog", "age" : 20 }
> db.doc_demo.aggregate([{$group: {_id: "$name", age: {$push: '$age'}}}])
{ "_id" : "一灰灰", "age" : [ 18, 18 ] }
{ "_id" : "一灰灰blog", "age" : [ 19, 20 ] }
> db.doc_demo.aggregate([{$group: {_id: "$name", age: {$addToSet: '$age'}}}])
{ "_id" : "一灰灰", "age" : [ 18 ] }
{ "_id" : "一灰灰blog", "age" : [ 20, 19 ] }
> db.doc_demo.aggregate([{$group: {_id: "$name", age: {$first: '$age'}}}])
{ "_id" : "一灰灰", "age" : 18 }
{ "_id" : "一灰灰blog", "age" : 19 }
> db.doc_demo.aggregate([{$group: {_id: "$name", age: {$last: '$age'}}}])
{ "_id" : "一灰灰", "age" : 18 }
{ "_id" : "一灰灰blog", "age" : 20 }
>
```

上面虽然介绍了分组支持的一些表达式，但是没有查询条件，难道只能针对所有的文档进行分组统计么？

分组过滤

借助`$match`来实现过滤统计，如下

```
db.doc_demo.aggregate([
  {$match: {'tag': {$gt: 1}}},
  {$group: {_id: '$name', age: {$sum: 1}}}
])
```

```
> db.doc_demo.find()
{ "_id" : ObjectId("5e7b5ac10172dc950171c488"), "name" : "一灰灰blog", "age" : 19, "skill" : [ "java", "python", "sql" ], "tag" : 2 }
{ "_id" : ObjectId("5e7b5ac40172dc950171c489"), "name" : "一灰灰blog", "age" : 20, "skill" : [ "web", "shell", "js" ], "tag" : 1 }
{ "_id" : ObjectId("5e7b5bb085a742842d2e23fc"), "name" : "一灰灰", "age" : 18, "sex" : "man", "tag" : 2 }
{ "_id" : ObjectId("5e7b5c2e0172dc950171c48a"), "name" : "一灰灰", "age" : 18, "hobby" : [ "play game" ] }
> db.doc_demo.aggregate([{$match: {'tag': {$gt: 1}}}, {$group: {_id: '$name', age: {$sum: 1}}}])
{ "_id" : "一灰灰", "age" : 1 }
{ "_id" : "一灰灰blog", "age" : 1 }
>
```

请注意，$match的语法规则和find的查询条件一样，会将满足条件的数据传递给后面的分组计算

这种方式和liux中的管道特别相似，aggregate方法的参数数组中，前面的执行完毕之后，将结果传递给后面的继续执行，除了`$match`和`$group`之外，还有一些其他的操作

操作	说明
$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。

文档 Document 查询非典型篇

前面介绍的查询可以说是常见的典型case，但是mongodb中有两个比价特殊的数据类型，数组 + 对象，自然的也会有一些非典型的查询case，下面主要针对这两种数据类型的查询姿势，给出实例讲解

1. 数组

首先准备一些供数组操作的文档如下

```
{ "_id" : ObjectId("5e7b5ac10172dc950171c488"), "name" : "一灰灰blog", "age" : 19,
"skill" : [ "java", "python", "sql" ], "tag" : 2 }
{ "_id" : ObjectId("5e7b5ac40172dc950171c489"), "name" : "一灰灰blog", "age" : 20,
"skill" : [ "web", "shell", "js" ], "tag" : 1 }
{ "_id" : ObjectId("5e7b5bb085a742842d2e23fc"), "name" : "一灰灰", "age" : 18, "sex" :
"man", "tag" : 2 }
{ "_id" : ObjectId("5e7b5c2e0172dc950171c48a"), "name" : "一灰灰", "age" : 18, "hobby"
: [ "play game" ] }
{ "_id" : ObjectId("5e7c5627f020f58f5323e52d"), "name" : "一灰灰2", "age" : 22, "skill"
: [ "android", "ios" ] }
```

长度查询

根据数组长度进行查询，借助 `$size` 来统计数组长度

```
# 查询数组长度为3的文档
db.doc_demo.find({'skill': {$size: 3}})
```

长度范围查询

请注意，不支持长度的比较查询，如下，会报语法错误

```
db.doc_demo.find({'skill:{$size: {$gt: 2}}})
```

```
> db.doc_demo.find({'skill:{$size: {$gt: 2}}})
2020-03-26T07:16:03.382+0000 E QUERY    [js] SyntaxError: unterminated string literal @(shell):1:18
>
```

要实现范围查询，可以借助 `$where` 来实现(`$where` 比较强大，后面单独说明)

```
# 请注意判空需要有
db.doc_demo.find({$where:'this.skill !=null && this.skill.length>2'})
```

```
> db.doc_demo.find({$where: 'this.skill !=null && this.skill.length>2'})
{ "_id" : ObjectId("5e7b5ac10172dc950171c488"), "name" : "一灰灰blog", "age" : 19, "skill" : [ "java", "python", "sql" ], "tag" : 2 }
{ "_id" : ObjectId("5e7b5ac40172dc950171c489"), "name" : "一灰灰blog", "age" : 20, "skill" : [ "web", "shell", "js" ], "tag" : 1 }
>
```

数组内容查询

根据数组内容进行查询，常见的有两种方式，一个是直接根据数组定位比较如

```
# 查询skill数组中，第一个元素为java的文档
db.doc_demo.find({'skill.0': 'java'})
```

上面这种实用性可能并不大，另外一个常见的case就是查询数组中包含某个元素的文档，这时可以借助 `$elemMatch` 来实现

```
# 查询skill数组中包含 java 元素的文档
db.doc_demo.find({'skill': {$elemMatch: {$eq: 'java'}}})
```

```
> db.doc_demo.find({'skill.0': 'java'})
{ "_id" : ObjectId("5e7b5ac10172dc950171c488"), "name" : "一灰灰blog", "age" : 19, "skill" : [ "java", "python", "sql" ], "tag" : 2 }
> db.doc_demo.find({'skill': {$elemMatch: {$eq: 'java'}}})
{ "_id" : ObjectId("5e7b5ac10172dc950171c488"), "name" : "一灰灰blog", "age" : 19, "skill" : [ "java", "python", "sql" ], "tag" : 2 }
```

说明，当数组的元素是Object类型时，还可以用右边这种姿势：`db.doc_demo.find({'skill': {$elemMatch: {'subField': 'xxx'}}})`

2. Object

因为mongodb支持内嵌文档，所以根据内嵌文档进行查询的场景也是不少的

首先准备三个用于后续查询测试的文档

```
{ "_id" : ObjectId("5e7c5a61f020f58f5323e52e"), "name" : "一灰灰", "doc" : { "title" : "简单的标题", "content" : "简单的内容", "tag" : [ "java", "后端" ] } }
{ "_id" : ObjectId("5e7c5a8af020f58f5323e52f"), "name" : "一灰灰", "doc" : { "title" : "哈哈", "content" : "嘻嘻哈哈", "tag" : [ "随笔" ], "draft" : true } }
{ "_id" : ObjectId("5e7c5ae7f020f58f5323e530"), "name" : "一灰灰", "doc" : { "title" : "22", "content" : "3333", "tag" : [ "随笔" ], "draft" : false, "visit" : 10 } }
```

根据内嵌文档字段查询

查询姿势和field查询相似，只是需要注意一下key的语法为: `field.subField`, 实例如下

```
db.doc_demo.find({'doc.title': '22'})
```

```
> db.doc_demo.find({'doc.title': '22'})
{ "_id" : ObjectId("5e7c5ae7f020f58f5323e530"), "name" : "一灰灰", "doc" : { "title" : "22", "content" : "3333", "tag" : [ "随笔" ], "draft" : false, "visit" : 10 } }
```

存在性查询

查询嵌入文档包含某个field的case，和普通的查询姿势也一样

```
db.doc_demo.find({'doc.visit': {$exists: true}})
```

排序

根据Object的成员进行排序，操作姿势也基本一样

```
db.doc_demo.find({'doc': {$exists: true}}).sort({'doc.visit': -1})
```

```
> db.doc_demo.find({'doc': {$exists: true}}).sort({'doc.visit': -1})
{ "_id" : ObjectId("5e7c5ae7f020f58f5323e530"), "name" : "一灰灰", "doc" : { "title" : "22", "content" : "3333", "tag" : [ "随笔" ], "draft" : false, "visit" : 10 } }
{ "_id" : ObjectId("5e7c5a61f020f58f5323e52e"), "name" : "一灰灰", "doc" : { "title" : "简单的标题", "content" : "简单的内容", "tag" : [ "java", "后端" ] } }
{ "_id" : ObjectId("5e7c5a8af020f58f5323e52f"), "name" : "一灰灰", "doc" : { "title" : "哈哈", "content" : "嘻嘻哈哈", "tag" : [ "随笔" ], "draft" : true } }
```

文档更新删除之非典型篇

前面介绍document的新增、删除、更新都处于相对常见和基础的说明，但是考虑到mongodb非结构化的特点，它的一些特性是我们的mysql不会遇到的，本文将针对这些特殊场景给出示例说明

- 在现有文档中，增加一个field
- 删除文档中的某个field
- 重命名文档的field
- 在文档的数组orObject中，添加/删除/更新数据

1. 增加field

我们知道修改文档的命令格式如下

```
db.collection.update(
   <query>,
   <update>,
   {
     upsert: <boolean>,
     multi: <boolean>,
     writeConcern: <document>
   }
)
```

当我们更新一个文档中，不存在的field，会怎样

```
# 插入一条数据，然后设置一个不存在的field
db.doc_demo.insert({    "author" : "一灰灰blog",    "title" : "测试"})
db.doc_demo.update({'author': '一灰灰blog'}, {$set: {'skill': ['java', 'db']}})
```

```
> db.doc_demo.insert({    "author" : "一灰灰blog",    "title" : "测试"})
WriteResult({ "nInserted" : 1 })
> db.doc_demo.update({'author': '一灰灰blog'}, {$set: {'skill': ['java', 'db']}}) db.doc_demo.update({'author': '一灰灰blog'}, {$set: {'skill': ['java', 'db']}})
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
> db.doc_demo.find()
{ "_id" : ObjectId("5e8f30b45d1625cd9287ceae"), "author" : "一灰灰blog", "title" : "测试", "skill" : [ "java", "db" ] }
```

2. 重命名field

同样是借助update方法，但是我们用到的关键字为 `$rename`

```
db.doc_demo.update({'author': '一灰灰blog'}, {$rename: {'skill': 'like'}})
```

```
> db.doc_demo.find()
{ "_id" : ObjectId("5e8f30b45d1625cd9287ceae"), "author" : "一灰灰blog", "title" : "测试", "skill" : [ "java", "db" ] }
> db.doc_demo.update({'author': '一灰灰blog'}, {$rename: {'skill': 'like'}})
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
> db.doc_demo.find()
{ "_id" : ObjectId("5e8f30b45d1625cd9287ceae"), "author" : "一灰灰blog", "title" : "测试", "like" : [ "java", "db" ] }
```

请注意，当文档中不存在这个field，则不会有任何影响

3. 删除field

既然`$set`可以新增一个不存在的field，那么是不是就可以用`$unset`来删除一个已存在的field呢

```
db.doc_demo.update({'author': '一灰灰blog'}, {$unset: {'title': 1}})
```

```
> db.doc_demo.find()
{ "_id" : ObjectId("5e8f30b45d1625cd9287ceae"), "author" : "一灰灰blog", "title" : "测试", "like" : [ "java", "db" ] }
> db.doc_demo.update({'author': '一灰灰blog'}, {$unset: {'title': 1}})
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
> db.doc_demo.find()
{ "_id" : ObjectId("5e8f30b45d1625cd9287ceae"), "author" : "一灰灰blog", "like" : [ "java", "db" ] }
```

4. 数组元素修改

> 数组元素的修改删除增加，可以参考官方教程: MongoDB update-array Method

如果我们希望直接修改数组中的某个元素，可以借助之前查询的case

```
# 修改数组中第0个元素
db.doc_demo.update({'author': '一灰灰blog'}, {$set: {'like.0': 'spring'}})
# 如果查询条件中，包含了数组内容的过滤，则可以用`$`来代替具体的数组下标，如
db.doc_demo.update({'author': '一灰灰blog', 'like': {$eq: 'db'}}, {$set: {'like.$':
'mysql'}})
```

```
> db.doc_demo.find()
{ "_id" : ObjectId("5e8f30b45d1625cd9287ceae"), "author" : "一灰灰blog", "like" : [ "java", "db" ] }
> db.doc_demo.update({'author': '一灰灰blog'}, {$set: {'like.0': 'spring'}})
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
> db.doc_demo.find()
{ "_id" : ObjectId("5e8f30b45d1625cd9287ceae"), "author" : "一灰灰blog", "like" : [ "spring", "db" ] }
> db.doc_demo.update({'author': '一灰灰blog', 'like': {$eq: 'db'}}, {$set: {'like.$': 'mysql'}})
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
> db.doc_demo.find()
{ "_id" : ObjectId("5e8f30b45d1625cd9287ceae"), "author" : "一灰灰blog", "like" : [ "spring", "mysql" ] }
>
```

请注意，使用`$`占位符的前途是，前面的查询条件可以限定数组元素

5. 数组元素新增

元素添加支持两种方式，一是`addToSet`，一是`push`

`$addToSet`

- 确保没有重复的项添加到数组集合，对于已经存在的重复元素不受影响；
- 不能保证添加时元素的顺序
- 如果值是数组，则作为一个元素添加进去
- 可以通过 `$each` 实现添加多个元素到数组中

```
# 不存在时，则添加，存在则忽略
db.doc_demo.update({'author': '一灰灰blog'}, {$addToSet: {'like': 'redis'}})
# 借助 $each 实现批量添加
db.doc_demo.update({'author': '一灰灰blog'}, {$addToSet: {'like': {$each: ['mongodb',
'es']}}})
```

`$push`

- 如果被更新的文档该数组不存在，那么$push将添加数组字段和值
- 如果字段不是数组，失败
- 如果值是数组，那么整个数组作为一个单个元素添加到数组

```
# 不存在时，创建一个数组
db.doc_demo.update({'author': '一灰灰blog'}, {$push: {'skill': 'a'}})
# 存在时，添加到数组
db.doc_demo.update({'author': '一灰灰blog'}, {$push: {'skill': 'a'}})
# 批量添加
db.doc_demo.update({'author': '一灰灰blog'}, {$push: {'skill': {$each: ['b', 'c']}}})
```

```
> db.doc_demo.update({'author': '一灰灰blog'}, {$push: {'skill': 'a'}})
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
> db.doc_demo.update({'author': '一灰灰blog'}, {$push: {'skill': 'a'}})
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
> db.doc_demo.update({'author': '一灰灰blog'}, {$push: {'skill': {$each: ['b', 'c']}}})
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
> db.doc_demo.find()
{ "_id" : ObjectId("5e8f30b45d1625cd9287ceae"), "author" : "一灰灰blog", "like" : [ "spring", "mysql", "redis", "mongodb", "es" ], "skill" : [ "a", "a", "b", "c" ] }
```

6. 数组元素删除

`$pop` 删除第一个or最后一个

```
# 删除最后一个
db.doc_demo.update({'author': '一灰灰blog'}, {$pop: {'skill': 1}})
# 删除第一个
db.doc_demo.update({'author': '一灰灰blog'}, {$pop: {'skill': -1}})
```

```
> db.doc_demo.find()
{ "_id" : ObjectId("5e8f30b45d1625cd9287ceae"), "author" : "一灰灰blog", "like" : [ "spring", "mysql", "redis", "mongodb", "es" ], "skill" : [ "a", "a", "b", "c" ] }
> db.doc_demo.update({'author': '一灰灰blog'}, {$pop: {'skill': 1}})
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
> db.doc_demo.find()
{ "_id" : ObjectId("5e8f30b45d1625cd9287ceae"), "author" : "一灰灰blog", "like" : [ "spring", "mysql", "redis", "mongodb", "es" ], "skill" : [ "a", "a", "b" ] }
> db.doc_demo.update({'author': '一灰灰blog'}, {$pop: {'skill': -1}})
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
> db.doc_demo.find()
{ "_id" : ObjectId("5e8f30b45d1625cd9287ceae"), "author" : "一灰灰blog", "like" : [ "spring", "mysql", "redis", "mongodb", "es" ], "skill" : [ "a", "b" ] }
```

`$pull` 删除满足条件的数组元素

```
# 将数组中添加几个元素
db.doc_demo.update({'author': '一灰灰blog'}, {$push: {'skill': {$each: ['a', 'b',
'c']}}})
# 删除指定的元素
db.doc_demo.update({'author': '一灰灰blog'}, {$pull: {'skill': 'b'}})
# 删除多个指定的元素
db.doc_demo.update({'author': '一灰灰blog'}, {$pull: {'skill': {$in: ['a', 'c']}}})
```

```
> db.doc_demo.find()
{ "_id" : ObjectId("5e8f30b45d1625cd9287ceae"), "author" : "一灰灰blog", "like" : [ "spring", "mysql", "redis", "mongodb", "es" ], "skill" : [ "a", "b", "a", "b", "c" ] }
> db.doc_demo.update({'author': '一灰灰blog'}, {$pull: {'skill': 'b'}})
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
> db.doc_demo.find()
{ "_id" : ObjectId("5e8f30b45d1625cd9287ceae"), "author" : "一灰灰blog", "like" : [ "spring", "mysql", "redis", "mongodb", "es" ], "skill" : [ "a", "a", "c" ] }
> db.doc_demo.update({'author': '一灰灰blog'}, {$pull: {'skill': {$in: ['a', 'c']}}})
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
> db.doc_demo.find()
{ "_id" : ObjectId("5e8f30b45d1625cd9287ceae"), "author" : "一灰灰blog", "like" : [ "spring", "mysql", "redis", "mongodb", "es" ], "skill" : [ ] }
```

注意，`$pull`后面跟上的可以理解为限定条件，查询教程篇的一些操作也是支持的（如比较查询等）

7. 内嵌文档操作

对于内嵌文档的操作，实际上普通的field的操作姿势没有什么区别，只是对于key加了一个`xx.xx`的限定而已

```
# 删除测试数据
db.doc_demo.remove({})
# 初始话一条演示文档
db.doc_demo.insert({'author': '一灰灰blog',})
# 不存在内嵌文档，则新增
db.doc_demo.update({}, {$set: {'t': {'a': 1, 'b': 2}}})
# 修改子field
db.doc_demo.update({}, {$set: {'t.a': 10}})
# 新增子field
db.doc_demo.update({}, {$set: {'t.c': 'c'}})
# 删除子field
db.doc_demo.update({}, {$unset: {'t.c': 1}})
# 重命名
db.doc_demo.update({}, {$rename: {'t.b': 't.dd'}})
```

```
> db.doc_demo.remove({})
WriteResult({ "nRemoved" : 1 })
> db.doc_demo.insert({'author': '一灰灰blog',})
WriteResult({ "nInserted" : 1 })
> db.doc_demo.find()
{ "_id" : ObjectId("5e8fc7eab448ddf1fe27d097"), "author" : "一灰灰blog" }
> db.doc_demo.update({}, {$set: {'t': {'a': 1, 'b': 2}}})
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
> db.doc_demo.find()
{ "_id" : ObjectId("5e8fc7eab448ddf1fe27d097"), "author" : "一灰灰blog", "t" : { "a" : 1, "b" : 2 } }
> db.doc_demo.update({}, {$set: {'t.a': 10}})
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
> db.doc_demo.find()
{ "_id" : ObjectId("5e8fc7eab448ddf1fe27d097"), "author" : "一灰灰blog", "t" : { "a" : 10, "b" : 2 } }
> db.doc_demo.update({}, {$set: {'t.c': 'c'}})
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
> db.doc_demo.find()
{ "_id" : ObjectId("5e8fc7eab448ddf1fe27d097"), "author" : "一灰灰blog", "t" : { "a" : 10, "b" : 2, "c" : "c" } }
> db.doc_demo.update({}, {$unset: {'t.c': 1}})
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
> db.doc_demo.find()
{ "_id" : ObjectId("5e8fc7eab448ddf1fe27d097"), "author" : "一灰灰blog", "t" : { "a" : 10, "b" : 2 } }
> db.doc_demo.update({}, {$rename: {'t.b': 't.dd'}})
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
> db.doc_demo.find()
{ "_id" : ObjectId("5e8fc7eab448ddf1fe27d097"), "author" : "一灰灰blog", "t" : { "a" : 10, "dd" : 2 } }
>
```

索引

索引一般用来提高查询效率，避免全集合搜索，那么在mongodb中，支持索引么？如果支持，如何定义索引，如何使用索引，如何确定一个sql是否走索引？

1. 创建索引

语法定义:

```
db.collection.createIndex(keys, options)
```

请注意，在3.0之前的版本中，也可以使用`ensureIndex`来创建索引

参数说明:

- keys：kv结构，key为fieldName, value为1 表示升序创建索引；-1 表示降序创建索引；支持多字段索引
- options：可选参数

常见参数说明如下表:

参数名	说明
`background`	true，则后台方式创建索引，不阻塞其他操作；默认为false
`unique`	true，则表示唯一约束索引，比如`_id`就有唯一约束；默认为false
`name`	索引名，不指定时，根据field + 方向生成索引名
`sparse`	true, 则不包含这个字段的不创建索引，且索引查询时查不到不包含这个字段的文档；默认false
`expireAfterSeconds`	设置文档在集合的生存时间，s为单位
`v`	版本号
`weight`	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`	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`	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

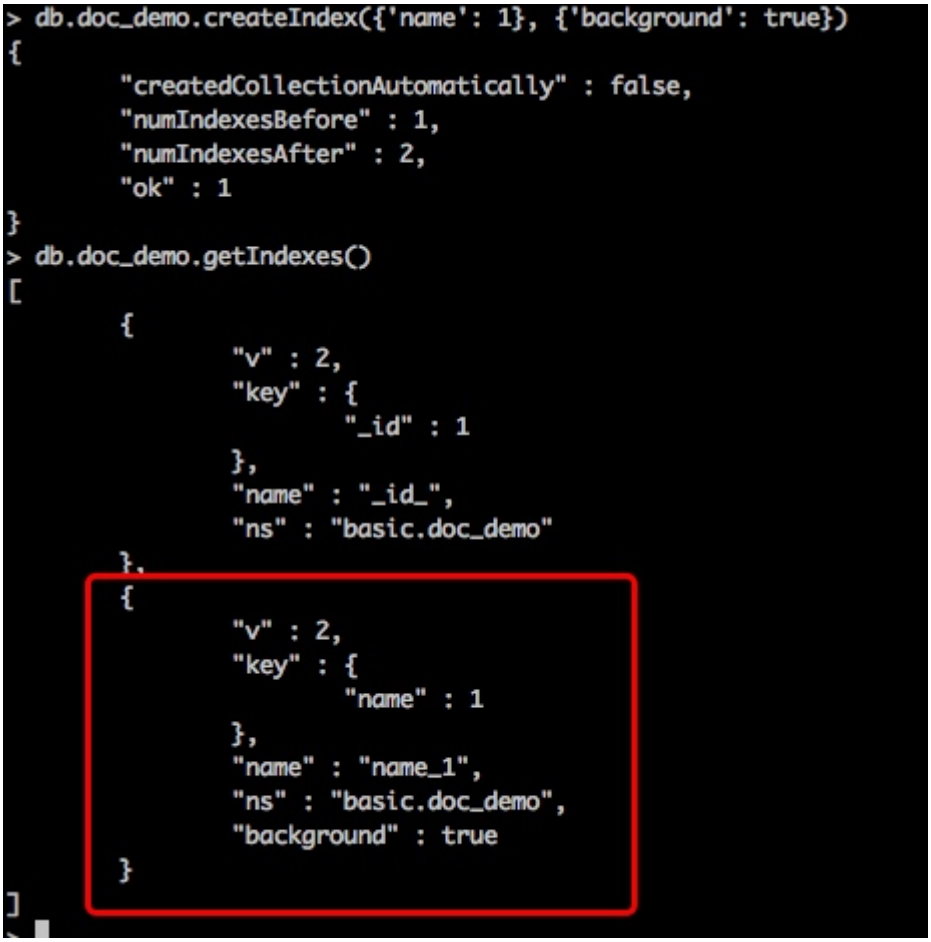
实例如下:

```
db.doc_demo.createIndex({'name': 1}, {'background': true})
```

2. 索引查询

查看一个集合定义了哪些索引，借助`getIndexes()`方法即可，如

```
db.doc_demo.getIndexes()
```



3. 索引分析

虽然我们创建了索引，但是我们的查询语句却并不一定会走索引，在mysql中我们知道有一个`explain`语句来分析索引情况，在mongodb中也存在类似的方法

集合数据如下

```
{ "_id" : ObjectId("5e7b5ac10172dc950171c488"), "name" : "一灰灰blog", "age" : 19,
"skill" : [ "java", "python", "sql" ], "tag" : 2 }
{ "_id" : ObjectId("5e7b5ac40172dc950171c489"), "name" : "一灰灰blog", "age" : 20,
"skill" : [ "web", "shell", "js" ], "tag" : 1 }
{ "_id" : ObjectId("5e7b5bb085a742842d2e23fc"), "name" : "一灰灰", "age" : 18, "sex" :
"man", "tag" : 2 }
{ "_id" : ObjectId("5e7b5c2e0172dc950171c48a"), "name" : "一灰灰", "age" : 18, "hobby"
: [ "play game" ] }
{ "_id" : ObjectId("5e7c5627f020f58f5323e52d"), "name" : "一灰灰2", "age" : 22, "skill"
: [ "android", "ios" ] }
{ "_id" : ObjectId("5e7c5a61f020f58f5323e52e"), "name" : "一灰灰", "doc" : { "title" :
"简单的标题", "content" : "简单的内容", "tag" : [ "java", "后端" ] } }
{ "_id" : ObjectId("5e7c5a8af020f58f5323e52f"), "name" : "一灰灰", "doc" : { "title" :
"哈哈", "content" : "嘻嘻哈哈", "tag" : [ "随笔" ], "draft" : true } }
{ "_id" : ObjectId("5e7c5ae7f020f58f5323e530"), "name" : "一灰灰", "doc" : { "title" :
"22", "content" : "3333", "tag" : [ "随笔" ], "draft" : false, "visit" : 10 } }
```

当前集合上除了默认的`_id`索引之外，针对`name`也创建了升序索引

如需要判断一个查询语句的情况，可以在后面加上`explain()`方法，如下

```
db.doc_demo.find({'name': '一灰灰'}).explain()
```

输出如下

```
{
    "queryPlanner" : {
        "plannerVersion" : 1,
        "namespace" : "basic.doc_demo",
        "indexFilterSet" : false,
        "parsedQuery" : {
            "name" : {
                "$eq" : "一灰灰"
            }
        },
        "winningPlan" : {
            "stage" : "FETCH",
            "inputStage" : {
                "stage" : "IXSCAN",
                "keyPattern" : {
                    "name" : 1
                },
                "indexName" : "name_1",
                "isMultiKey" : false,
                "multiKeyPaths" : {
                    "name" : [ ]
                },
                "isUnique" : false,
                "isSparse" : false,
                "isPartial" : false,
                "indexVersion" : 2,
                "direction" : "forward",
                "indexBounds" : {
                    "name" : [
                        "[\"一灰灰\", \"一灰灰\"]"
                    ]
                }
            }
        },
        "rejectedPlans" : [ ]
    },
    "serverInfo" : {
        "host" : "0f51c424211c",
        "port" : 27017,
        "version" : "4.0.4",
        "gitVersion" : "f288a3bdf201007f3693c58e140056adf8b04839"
    },
    "ok" : 1
}
```


关于是否走索引，主要看stage，通常会有以下几种状态

stage	描述
COLLSCAN	全表扫描
IXSCAN	扫描索引
FETCH	根据索引去检索指定document
SHARD_MERGE	将各个分片返回数据进行merge
SORT	表明在内存中进行了排序
LIMIT	使用limit限制返回数
SKIP	使用skip进行跳过
IDHACK	针对_id进行查询
SHARDING_FILTER	通过mongos对分片数据进行查询
COUNT	利用db.coll.explain().count()之类进行count运算
COUNTSCAN	count不使用Index进行count时的stage返回
COUNT_SCAN	count使用了Index进行count时的stage返回
SUBPLA	未使用到索引的$or查询的stage返回
TEXT	使用全文索引进行查询时候的stage返回
PROJECTION	限定返回字段时候stage的返回

上面的具体查询，对应的stage组合是`Fetch+ixscan`，也就是说会根据索引查询

虽然mongodb会根据查询来选择索引，但并不能保证都能选到最优的索引；这种时候我们可以通过`hint`来强制指定索引，举例如下

```
db.doc_demo.find({'age': 18, 'name':'一灰灰'}).hint({'name': 1}).explain()
```

4. 删除索引

一般有下面两种删除方式，全量删除和指定索引删除

```
# 全量删除
db.collection.dropIndexes()
# 指定删除
db.collection.dropIndex(索引名)
```

请注意，指定索引名删除时，如果不确定索引名是啥，可以通过`getIndexes()`来查看

5. 文档自动删除

在创建索引的时候，其中有一个参数比较有意思，有必要单独拿出来说明一下，`expireAfterSeconds` 设置文档的生存时间

使用它有几个潜规则：

- 索引字段为Date类型
- 单字段索引，不支持混合索引
- 非立即执行

```
# 插入一条文档，请注意这个时间，因为时区原因相对于北京时间，少8小时
db.doc_demo.insert({'name': 'yihui', 'log': '操作了啥啥啥', 'createDate': new
Date('Mar27, 2020 2:54:00')})

# 创建索引
db.doc_demo.createIndex({'createDate': 1}, {expireAfterSeconds: 60})
```

然后过一段时间（并不一定10:55分的时候会删除）再去查询，会发现插入的文档被删除了

利用这种特性，在mongodb中存一些需要定时删除的数据，相比较我们常用的mysql而言，还是有很大优势的

6. 覆盖索引

覆盖索引的概念有些类似mysql中的不回表查询的case，直接查询索引，就可以返回所需要的字段了

比如在前面的case中，我只查询name字段，可以走覆盖索引；但是返回除了name，还有`_id`，那么就不能了

```
# 覆盖索引
db.doc_demo.find({'name': '一灰灰'}, {'name': 1, '_id': 0})
# 非覆盖索引
db.doc_demo.find({'name': '一灰灰'}, {'name': 1})
```

注意：所有索引字段是一个数组时，不能使用覆盖索引

第二卷：Spring整合MongoDB

文档数据库SpringBoot的系列教程，彻底扫平阻碍我们CURD的路障，适用于对Mongodb不了解或了解不够的小伙伴

基本环境搭建与使用

SpringBoot结合mongodb进行业务开发，也属于比较基本的需求了，本文为mongo系列的基本篇，主要就是环境搭建、工程的配置设置相关

1. 环境搭建

正式开始之前，第一步就是需要安装Mongo的环境了，因为环境的安装和我们spring的主题没有太大的关系，因此我们选择最简单的使用姿势：直接用docker来安装mongo来使用

下面的安装过程都是mac环境，其他操作系统可以直接安装mongodb，移步相关教程

1.1 docker 安装

可以直接到官网进行下载安装，但是对系统版本有要求，所以需要使用`Docker ToolBox`，实际试过之后，感觉不太好用，实际上是将docker安装到虚拟机中了，下面直接使用`brew`命令进行安装

安装命令

```
brew cask install docker
```

执行完毕之后，会多一个应用名为`docker`， 双击运行，输入密码等即可

1.2 mongo 安装使用

直接使用官方的mongo镜像即可，然后绑定端口映射，就可以在宿主机中使用mongo

```
# 下载镜像
docker pull mongo
# 加载并运行镜像
docker run --name mongo -p 27017:27017 -d mongo --auth
# 进入容器
docker exec -it d9132f1e8b26 /bin/bash
# 为mongo创建登录用户和密码
mongo
use admin
db.createUser({user:"root",pwd:"root",roles:[{role:'root',db:'admin'}]})
exit
```

上面完毕之后，可以在宿主机进行连接测试，判断是否安装成功

2. SpringBoot工程配置

2.1 pom依赖

整个框架选择的是spring-boot，所有spring这一套相关的pom配置少不了，我们主要需要注意的包就是`spring-boot-starter-data-mongodb`

```
<parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>2.0.4.RELEASE</version>
    <relativePath/> <!-- lookup parent from repository -->
</parent>

<properties>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    <project.reporting.outputEncoding>UTF-8</project.reporting.outputEncoding>
    <spring-cloud.version>Finchley.RELEASE</spring-cloud.version>
    <java.version>1.8</java.version>
</properties>

<dependencies>
   <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-data-mongodb</artifactId>
    </dependency>
    <dependency>
        <groupId>com.alibaba</groupId>
        <artifactId>fastjson</artifactId>
        <version>1.2.45</version>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter</artifactId>
    </dependency>

    <dependency>
        <groupId>org.projectlombok</groupId>
        <artifactId>lombok</artifactId>
        <optional>true</optional>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-test</artifactId>
        <scope>test</scope>
    </dependency>
</dependencies>

<build>
    <pluginManagement>
        <plugins>
            <plugin>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-maven-plugin</artifactId>
            </plugin>
```

```
        </plugins>
    </pluginManagement>
</build>

<repositories>
    <repository>
        <id>spring-milestones</id>
        <name>Spring Milestones</name>
        <url>https://repo.spring.io/milestone</url>
        <snapshots>
            <enabled>false</enabled>
        </snapshots>
    </repository>
</repositories>
```

2.2 配置文件

配置文件如下，主要就是连接mongo的url

```
spring.data.mongodb.uri=mongodb://root:root@localhost:27017/basic?
authSource=admin&authMechanism=SCRAM-SHA-1
```

通过上面的实例，也知道格式如下:

`mongodb://用户名:密码@host:port/dbNmae?参数`

- 当没有用户名和密码时，可以省略掉中间的 `root:root@`；
- 当需要认证时，请格外注意
  - mongodb新版的验证方式改成了`SCRAM-SHA-1`，所以参数中一定一定一定得加上
    - `?authSource=admin&authMechanism=SCRAM-SHA-1`
  - 如果将mongodb的验证方式改成了`MONGODB-CR`,则上面的可以不需要

2.3 测试使用

写一个简单的测试类，看下mongodb是否连接成功，是否可以正常操作

```java
@Slf4j
@Component
public class MongoTemplateHelper {

    @Getter
    @Setter
    private MongoTemplate mongoTemplate;

    public MongoTemplateHelper(MongoTemplate mongoTemplate) {
        this.mongoTemplate = mongoTemplate;
    }



    /**
     * 保存记录
     *
     * @param params
     * @param collectionName
     */
    public void saveRecord(Map<String, Object> params, String collectionName) {
        mongoTemplate.save(params, collectionName);
    }

    /**
     * 精确查询方式
     *
     * @param query
     * @param collectionName
     */
    public void queryRecord(Map<String, Object> query, String collectionName) {
        Criteria criteria = null;
        for (Map.Entry<String, Object> entry : query.entrySet()) {
            if (criteria == null) {
                criteria = Criteria.where(entry.getKey()).is(entry.getValue());
            } else {
                criteria.and(entry.getKey()).is(entry.getValue());
            }
        }

        Query q = new Query(criteria);
        Map result = mongoTemplate.findOne(q, Map.class, collectionName);
        log.info("{}", result);
    }
}
```

上面提供了两个方法，新增和查询，简单的使用姿势如

```
@SpringBootApplication
public class Application {

    private static final String COLLECTION_NAME = "personal_info";


    public Application(MongoTemplateHelper mongoTemplateHelper) {
        Map<String, Object> records = new HashMap<>(4);
        records.put("name", "小灰灰Blog");
        records.put("github", "https://github.com/liuyueyi");
        records.put("time", LocalDateTime.now());

        mongoTemplateHelper.saveRecord(records, COLLECTION_NAME);

        Map<String, Object> query = new HashMap<>(4);
        query.put("name", "小灰灰Blog");
        mongoTemplateHelper.queryRecord(query, COLLECTION_NAME);
    }

    public static void main(String[] args) {
        SpringApplication.run(Application.class);
    }

}
```

然后开始执行，查看输出，结果演示如下

2.4 说明

最后针对认证的问题，需要额外提一句，开始测试的时候，使用的配置如下

```
spring.data.mongodb.username=root
spring.data.mongodb.password=root
spring.data.mongodb.authentication-database=basic
spring.data.mongodb.host=localhost
spring.data.mongodb.port=27017
```

然而因为mongo采用的是SHA-1加密方式，所以始终验证不通过；然后查了一下，各种让改mongo的验证版本，改回去用CR的方式；但明显这种并不是一种好的解决方式，既然新的版本选择了新的加密方式，总有他的理由，所以应该改的还是spring的使用姿势；目前还没找到匹配上面这种配置方式的解决方案；

本文选择的是用url的方式指定加密方式来解决这个问题，当然研究下后面这种方式内部实现，应该就能知道前面的可以怎么解决，这点记下来，后续再开坑填

3. 其他

0. 项目

- 工程：spring-boot-demo
- module: 110-mongo-basic

查询基本使用姿势

学习一个新的数据库，一般怎么下手呢？基本的CURD没跑了，当可以熟练的增、删、改、查一个数据库时，可以说对这个数据库算是入门了，如果需要更进一步的话，就需要了解下数据库的特性，比如索引、事物、锁、分布式支持等

本篇博文为mongodb的入门篇，将介绍一下基本的查询操作，在Spring中可以怎么玩

1. 基本使用

1.0. 环境准备

在正式开始之前，先准备好环境，搭建好工程，对于这一步的详细信息，可以参考博文: 181213-SpringBoot高级篇MongoDB之基本环境搭建与使用

接下来，在一个集合中，准备一下数据如下，我们的基本查询范围就是这些数据

 data

1.1. 根据字段进行查询

最常见的查询场景，比如我们根据查询`user=一灰灰blog`的数据，这里主要会使用`Query` + `Criteria` 来完成

```
@Component
public class MongoReadWrapper {
    private static final String COLLECTION_NAME = "demo";

    @Autowired
    private MongoTemplate mongoTemplate;

    /**
     * 指定field查询
     */
    public void specialFieldQuery() {
        Query query = new Query(Criteria.where("user").is("一灰灰blog"));
        // 查询一条满足条件的数据
        Map result = mongoTemplate.findOne(query, Map.class, COLLECTION_NAME);
        System.out.println("query: " + query + " | specialFieldQueryOne: " + result);

        // 满足所有条件的数据
        List<Map> ans = mongoTemplate.find(query, Map.class, COLLECTION_NAME);
        System.out.println("query: " + query + " | specialFieldQueryAll: " + ans);
    }
}
```

上面是一个实际的case，从中可以知道一般的查询方式为:

- `Criteria.where(xxx).is(xxx)`来指定具体的查询条件
- 封装Query对象 `new Query(criteria)`
- 借助`mongoTemplate`执行查询 `mongoTemplate.findOne(query, resultType, collectionName)`

其中findOne表示只获取一条满足条件的数据；find则会将所有满足条件的返回；上面执行之后，输出结果如

```
query: Query: { "user" : "一灰灰blog" }, Fields: { }, Sort: { } | specialFieldQueryOne:
{_id=5c2368b258f984a4fda63cee, user=一灰灰blog, desc=帅气逼人的码农界老秀}
query: Query: { "user" : "一灰灰blog" }, Fields: { }, Sort: { } | specialFieldQueryAll:
[{_id=5c2368b258f984a4fda63cee, user=一灰灰blog, desc=帅气逼人的码农界老秀},
{_id=5c3afaf4e3ac8e8d2d39238a, user=一灰灰blog, desc=帅气逼人的码农界老秀3},
{_id=5c3afb1ce3ac8e8d2d39238d, user=一灰灰blog, desc=帅气逼人的码农界老秀6, age=18.0},
{_id=5c3b0031e3ac8e8d2d39238e, user=一灰灰blog, desc=帅气逼人的码农界老秀6, age=20.0},
{_id=5c3b003ee3ac8e8d2d39238f, user=一灰灰blog, desc=帅气逼人的码农界老秀6, sign=hello
world}]
```

1.2. and多条件查询

前面是只有一个条件满足，现在如果是要求同时满足多个条件，则利用`org.springframework.data.mongodb.core.query.Criteria#and`来斜街多个查询条件

```java
/**
 * 多个查询条件同时满足
 */
public void andQuery() {
    Query query = new Query(Criteria.where("user").is("一灰灰blog").and("age").is(18));
    Map result = mongoTemplate.findOne(query, Map.class, COLLECTION_NAME);
    System.out.println("query: " + query + " | andQuery: " + result);
}
```

输出结果如下

```
query: Query: { "user" : "一灰灰blog", "age" : 18 }, Fields: { }, Sort: { } | andQuery:
{_id=5c3afb1ce3ac8e8d2d39238d, user=一灰灰blog, desc=帅气逼人的码农界老秀6, age=18.0}
```

1.3. or或查询

and对应的就是or，多个条件中只要一个满足即可，这个与and的使用有些区别, 借助`org.springframework.data.mongodb.core.query.Criteria#orOperator`来实现，传参为多个`Criteria`对象，其中每一个表示一种查询条件

```
/**
 * 或查询
 */
public void orQuery() {
    // 等同于 db.getCollection('demo').find({"user": "一灰灰blog", $or: [{ "age": 18},
{ "sign": {$exists: true}}]})
    Query query = new Query(Criteria.where("user").is("一灰灰blog")
            .orOperator(Criteria.where("age").is(18),
Criteria.where("sign").exists(true)));
    List<Map> result = mongoTemplate.find(query, Map.class, COLLECTION_NAME);
    System.out.println("query: " + query + " | orQuery: " + result);

    // 单独的or查询
    // 等同于Query: { "$or" : [{ "age" : 18 }, { "sign" : { "$exists" : true } }] },
Fields: { }, Sort: { }
    query = new Query(new Criteria().orOperator(Criteria.where("age").is(18),
Criteria.where("sign").exists(true)));
    result = mongoTemplate.find(query, Map.class, COLLECTION_NAME);
    System.out.println("query: " + query + " | orQuery: " + result);
}
```

执行后输出结果为

```
query: Query: { "user" : "一灰灰blog", "$or" : [{ "age" : 18 }, { "sign" : { "$exists"
: true } }] }, Fields: { }, Sort: { } | orQuery: [{_id=5c3afb1ce3ac8e8d2d39238d, user=
一灰灰blog, desc=帅气逼人的码农界老秀6, age=18.0}, {_id=5c3b003ee3ac8e8d2d39238f, user=
一灰灰blog, desc=帅气逼人的码农界老秀6, sign=hello world}]
query: Query: { "$or" : [{ "age" : 18 }, { "sign" : { "$exists" : true } }] }, Fields:
{ }, Sort: { } | orQuery: [{_id=5c3afb1ce3ac8e8d2d39238d, user=一灰灰blog, desc=帅气逼
人的码农界老秀6, age=18.0}, {_id=5c3b003ee3ac8e8d2d39238f, user=一灰灰blog, desc=帅气逼
人的码农界老秀6, sign=hello world}, {_id=5c3b0538e3ac8e8d2d392390, user=二灰灰blog,
desc=帅气逼人的码农界老秀6, sign=hello world}]
```

1.4. in查询

标准的in查询case

```
/**
 * in查询
 */
public void inQuery() {
    // 相当于:
    Query query = new Query(Criteria.where("age").in(Arrays.asList(18, 20, 30)));
    List<Map> result = mongoTemplate.find(query, Map.class, COLLECTION_NAME);
    System.out.println("query: " + query + " | inQuery: " + result);
}
```

输出

```
query: Query: { "age" : { "$in" : [18, 20, 30] } }, Fields: { }, Sort: { } | inQuery:
[{_id=5c3afb1ce3ac8e8d2d39238d, user=一灰灰blog, desc=帅气逼人的码农界老秀6, age=18.0},
{_id=5c3b0031e3ac8e8d2d39238e, user=一灰灰blog, desc=帅气逼人的码农界老秀6, age=20.0}]
```

1.5. 数值比较

数值的比较大小，主要使用的是 `get`，`gt`，`lt`，`let`

```
/**
 * 数字类型，比较查询 >
 */
public void compareBigQuery() {
    // age > 18
    Query query = new Query(Criteria.where("age").gt(18));
    List<Map> result = mongoTemplate.find(query, Map.class, COLLECTION_NAME);
    System.out.println("query: " + query + " | compareBigQuery: " + result);

    // age >= 18
    query = new Query(Criteria.where("age").gte(18));
    result = mongoTemplate.find(query, Map.class, COLLECTION_NAME);
    System.out.println("query: " + query + " | compareBigQuery: " + result);
}

/**
 * 数字类型，比较查询 <
 */
public void compareSmallQuery() {
    // age < 20
    Query query = new Query(Criteria.where("age").lt(20));
    List<Map> result = mongoTemplate.find(query, Map.class, COLLECTION_NAME);
    System.out.println("query: " + query + " | compareSmallQuery: " + result);

    // age <= 20
    query = new Query(Criteria.where("age").lte(20));
    result = mongoTemplate.find(query, Map.class, COLLECTION_NAME);
    System.out.println("query: " + query + " | compareSmallQuery: " + result);
}
```

输出

```
query: Query: { "age" : { "$gt" : 18 } }, Fields: { }, Sort: { } | compareBigQuery:
[{_id=5c3b0031e3ac8e8d2d39238e, user=一灰灰blog, desc=帅气逼人的码农界老秀6, age=20.0}]
query: Query: { "age" : { "$gte" : 18 } }, Fields: { }, Sort: { } | compareBigQuery:
[{_id=5c3afb1ce3ac8e8d2d39238d, user=一灰灰blog, desc=帅气逼人的码农界老秀6, age=18.0},
{_id=5c3b0031e3ac8e8d2d39238e, user=一灰灰blog, desc=帅气逼人的码农界老秀6, age=20.0}]
query: Query: { "age" : { "$lt" : 20 } }, Fields: { }, Sort: { } | compareSmallQuery:
[{_id=5c3afb1ce3ac8e8d2d39238d, user=一灰灰blog, desc=帅气逼人的码农界老秀6, age=18.0}]
query: Query: { "age" : { "$lte" : 20 } }, Fields: { }, Sort: { } | compareSmallQuery:
[{_id=5c3afb1ce3ac8e8d2d39238d, user=一灰灰blog, desc=帅气逼人的码农界老秀6, age=18.0},
{_id=5c3b0031e3ac8e8d2d39238e, user=一灰灰blog, desc=帅气逼人的码农界老秀6, age=20.0}]
```

1.6. 正则查询

牛逼高大上的功能

```
/**
 * 正则查询
 */
public void regexQuery() {
    Query query = new Query(Criteria.where("user").regex("^一灰灰blog"));
    List<Map> result = mongoTemplate.find(query, Map.class, COLLECTION_NAME);
    System.out.println("query: " + query + " | regexQuery: " + result);
}
```

输出

```
query: Query: { "user" : { "$regex" : "^一灰灰blog", "$options" : "" } }, Fields: { }, Sort: { } | regexQuery: [{_id=5c2368b258f984a4fda63cee, user=一灰灰blog, desc=帅气逼人的码农界老秀}, {_id=5c3afacde3ac8e8d2d392389, user=一灰灰blog2, desc=帅气逼人的码农界老秀2}, {_id=5c3afaf4e3ac8e8d2d39238a, user=一灰灰blog, desc=帅气逼人的码农界老秀3}, {_id=5c3afafbe3ac8e8d2d39238b, user=一灰灰blog4, desc=帅气逼人的码农界老秀4}, {_id=5c3afb0ae3ac8e8d2d39238c, user=一灰灰blog5, desc=帅气逼人的码农界老秀5}, {_id=5c3afb1ce3ac8e8d2d39238d, user=一灰灰blog, desc=帅气逼人的码农界老秀6, age=18.0}, {_id=5c3b0031e3ac8e8d2d39238e, user=一灰灰blog, desc=帅气逼人的码农界老秀6, age=20.0}, {_id=5c3b003ee3ac8e8d2d39238f, user=一灰灰blog, desc=帅气逼人的码农界老秀6, sign=hello world}]
```

1.7. 查询总数

统计常用，这个主要利用的是`mongoTemplate.count`方法

```
/**
 * 查询总数
 */
public void countQuery() {
    Query query = new Query(Criteria.where("user").is("一灰灰blog"));
    long cnt = mongoTemplate.count(query, COLLECTION_NAME);
    System.out.println("query: " + query + " | cnt " + cnt);
}
```

输出

```
query: Query: { "user" : "一灰灰blog" }, Fields: { }, Sort: { } | cnt 5
```

1.8. 分组查询

这个对应的是mysql中的group查询，但是在mongodb中，更多的是通过聚合查询，可以完成很多类似的操作，下面借助聚合，来看一下分组计算总数怎么玩

```
/*
 * 分组查询
 */
public void groupQuery() {
    // 根据用户名进行分组统计，每个用户名对应的数量
    // aggregate([ { "$group" : { "_id" : "user" , "userCount" : { "$sum" : 1}}}] )
    Aggregation aggregation =
Aggregation.newAggregation(Aggregation.group("user").count().as("userCount"));
    AggregationResults<Map> ans = mongoTemplate.aggregate(aggregation,
COLLECTION_NAME, Map.class);
    System.out.println("query: " + aggregation + " | groupQuery " +
ans.getMappedResults());
}
```

注意下，这里用`Aggregation`而不是前面的`Query`和`Criteria`，输出如下

```
query: { "aggregate" : "__collection__", "pipeline" : [{ "$group" : { "_id" : "$user",
"userCount" : { "$sum" : 1 } } }] } | groupQuery [{_id=一灰灰blog, userCount=5}, {_id=
一灰灰blog2, userCount=1}, {_id=一灰灰blog4, userCount=1}, {_id=二灰灰blog,
userCount=1}, {_id=一灰灰blog5, userCount=1}]
```

1.9. 排序

sort，比较常见的了，在mongodb中有个有意思的地方在于某个字段，document中并不一定存在，这是会怎样呢?

```
/**
 * 排序查询
 */
public void sortQuery() {
    // sort查询条件，需要用with来衔接
    Query query = Query.query(Criteria.where("user").is("一灰灰
blog")).with(Sort.by("age"));
    List<Map> result = mongoTemplate.find(query, Map.class, COLLECTION_NAME);
    System.out.println("query: " + query + " | sortQuery " + result);
}
```

输出结果如下，对于没有这个字段的document也被查出来了

```
query: Query: { "user" : "一灰灰blog" }, Fields: { }, Sort: { "age" : 1 } | sortQuery
[{_id=5c2368b258f984a4fda63cee, user=一灰灰blog, desc=帅气逼人的码农界老秀},
{_id=5c3afaf4e3ac8e8d2d39238a, user=一灰灰blog, desc=帅气逼人的码农界老秀3},
{_id=5c3b003ee3ac8e8d2d39238f, user=一灰灰blog, desc=帅气逼人的码农界老秀6, sign=hello
world}, {_id=5c3afb1ce3ac8e8d2d39238d, user=一灰灰blog, desc=帅气逼人的码农界老秀6,
age=18.0}, {_id=5c3b0031e3ac8e8d2d39238e, user=一灰灰blog, desc=帅气逼人的码农界老秀6,
age=20.0}]
```

1.10. 分页

数据量多的时候，分页查询比较常见，用得多就是limit和skip了

```
/**
 * 分页查询
 */
public void pageQuery() {
    // limit限定查询2条
    Query query = Query.query(Criteria.where("user").is("一灰灰
blog")).with(Sort.by("age")).limit(2);
    List<Map> result = mongoTemplate.find(query, Map.class, COLLECTION_NAME);
    System.out.println("query: " + query + " | limitPageQuery " + result);


    // skip()方法来跳过指定数量的数据
    query = Query.query(Criteria.where("user").is("一灰灰
blog")).with(Sort.by("age")).skip(2);
    result = mongoTemplate.find(query, Map.class, COLLECTION_NAME);
    System.out.println("query: " + query + " | skipPageQuery " + result);
}
```

输出结果表明，limit用来限制查询多少条数据，skip则表示跳过前面多少条数据

```
query: Query: { "user" : "一灰灰blog" }, Fields: { }, Sort: { "age" : 1 } |
limitPageQuery [{_id=5c2368b258f984a4fda63cee, user=一灰灰blog, desc=帅气逼人的码农界老
秀}, {_id=5c3afaf4e3ac8e8d2d39238a, user=一灰灰blog, desc=帅气逼人的码农界老秀3}]
query: Query: { "user" : "一灰灰blog" }, Fields: { }, Sort: { "age" : 1 } |
skipPageQuery [{_id=5c3b003ee3ac8e8d2d39238f, user=一灰灰blog, desc=帅气逼人的码农界老秀
6, sign=hello world}, {_id=5c3afb1ce3ac8e8d2d39238d, user=一灰灰blog, desc=帅气逼人的码
农界老秀6, age=18.0}, {_id=5c3b0031e3ac8e8d2d39238e, user=一灰灰blog, desc=帅气逼人的码
农界老秀6, age=20.0}]
```

1.11. 小结

上面给出的一些常见的查询姿势，当然并不全面，比如我们如果需要查询document中的部分字段怎么办？比如document内部结果比较复杂，有内嵌的对象或者数组时，嵌套查询可以怎么玩？索引什么的又可以怎么利用起来，从而优化查询效率？如何通过传说中自动生成的`_id`来获取文档创建的时间戳？

先留着这些疑问，后面再补上

2. 其他

2.0. 项目

- 工程：spring-boot-demo
- module: mongo-template
- 相关博文: 181213-SpringBoot高级篇MongoDB之基本环境搭建与使用

新增文档使用姿势

本篇博文为mongodb的curd中一篇，前面介绍简单的查询使用，这一篇重点则放在插入数据;

1. 基本使用

首先是准备好基本环境，可以参考博文

- 181213-SpringBoot高级篇MongoDB之基本环境搭建与使用
- 190113-SpringBoot高级篇MongoDB之查询基本使用姿势

1.1. 新增一条数据

MongoDB一个基本数据称为document，和mysql不一样，没有强制约束哪些字段，可以随意的插入，下面是一个简单的插入演示

```
private static final String COLLECTION_NAME = "demo";

@Autowired
private MongoTemplate mongoTemplate;

/**
 * 新增一条记录
 */
public void insert() {
    JSONObject object = new JSONObject();
    object.put("name", "一灰灰blog");
    object.put("desc", "欢迎关注一灰灰Blog");
    object.put("age", 28);

    // 插入一条document
    mongoTemplate.insert(object, COLLECTION_NAME);


    JSONObject ans = mongoTemplate
            .findOne(new Query(Criteria.where("name").is("一灰灰
blog").and("age").is(28)), JSONObject.class,
                    COLLECTION_NAME);
    System.out.println(ans);
}
```

使用的关键地方为一行: `mongoTemplate.insert(object, COLLECTION_NAME);`

- 第一个参数为待插入的document
- 第二个参数为collection name （相当于mysql的table)

执行后输出结果为如下

```
{"name":"一灰灰blog","_id":
{"counter":12472353,"date":1548333180000,"machineIdentifier":14006254,"processIdentifi
er":17244,"time":1548333180000,"timeSecond":1548333180,"timestamp":1548333180},"age":2
8,"desc":"欢迎关注一灰灰Blog"}
```

1.2. 批量插入

一次插入多条记录，传集合进去即可

```
/**
 * 批量插入
 */
public void insertMany() {
    List<Map<String, Object>> records = new ArrayList<>();
    for (int i = 0; i < 3; i++) {
        Map<String, Object> record = new HashMap<>(4);
        record.put("wechart", "一灰灰blog");
        record.put("blog", Arrays.asList("http://spring.hhui.top",
"http://blog.hhui.top"));
        record.put("nums", 210);
        record.put("t_id", i);
        records.add(record);
    }

    // 批量插入文档
    mongoTemplate.insert(records, COLLECTION_NAME);

    // 查询插入的内容
    List<Map> result =
            mongoTemplate.find(new Query(Criteria.where("wechart").is("一灰灰blog")),
Map.class, COLLECTION_NAME);
    System.out.println("Query Insert Records: " + result);
}
```

返回结果如下:

```
Query Insert Records: [{t_id=0, wechart=一灰灰blog, _id=5c49b07cd5b7ee435cbe5022, blog=
[http://spring.hhui.top, http://blog.hhui.top], nums=210}, {t_id=1, wechart=一灰灰blog,
_id=5c49b07cd5b7ee435cbe5023, blog=[http://spring.hhui.top, http://blog.hhui.top],
nums=210}, {t_id=2, wechart=一灰灰blog, _id=5c49b07cd5b7ee435cbe5024, blog=
[http://spring.hhui.top, http://blog.hhui.top], nums=210}]
```

1.3. upsert，不存在才插入

我们希望在插入之前，判断数据是否存在，如果不存在则插入；如果存在则更新；此时就可以采用upsert来使用，一般三个参数

```
mongoTemplate.upsert(Query query, Update update, String collectionName)
```

第一个为查询条件，第二个为需要更新的字段，最后一个指定对应的collection，一个简单的实例如下

```
/**
 * 数据不存在，通过 upsert 新插入一条数据
 *
 * set 表示修改key对应的value
 * addToSet 表示在数组中新增一条
 */
public void upsertNoMatch() {
    // addToSet 表示将数据塞入document的一个数组成员中
    UpdateResult upResult = mongoTemplate.upsert(new
Query(Criteria.where("name").is("一灰灰blog").and("age").is(100)),
            new Update().set("age", 120).addToSet("add", "额外增加"), COLLECTION_NAME);
    System.out.println("nomatch upsert return: " + upResult);

    List<JSONObject> re = mongoTemplate
            .find(new Query(Criteria.where("name").is("一灰灰
blog").and("age").is(120)), JSONObject.class,
                    COLLECTION_NAME);
    System.out.println("after upsert return should not be null: " + re);
    System.out.println("----------------------------------------");
}
```

输出结果如下:

```
nomatch upsert return: AcknowledgedUpdateResult{matchedCount=0, modifiedCount=0,
upsertedId=BsonObjectId{value=5c49b07ce6652f7e1add1ea2}}
after upsert return should not be null: [{"add":["额外增加"],"name":"一灰灰blog","_id":
{"counter":14491298,"date":1548333180000,"machineIdentifier":15099183,"processIdentifi
er":32282,"time":1548333180000,"timeSecond":1548333180,"timestamp":1548333180},"age":1
20}]
----------------------------------------
```

1.4. upsert，存在则更新

前面的demo是演示不存在，那么存在数据呢?

```
/**
 * 只有一条数据匹配，upsert 即表示更新
 */
public void upsertOneMatch() {
    // 数据存在，使用更新
    UpdateResult result = mongoTemplate.upsert(new Query(Criteria.where("name").is("一灰灰blog").and("age").is(120)),
            new Update().set("age", 100), COLLECTION_NAME);
    System.out.println("one match upsert return: " + result);

    List<JSONObject> ans = mongoTemplate
            .find(new Query(Criteria.where("name").is("一灰灰blog").and("age").is(100)), JSONObject.class,
                    COLLECTION_NAME);
    System.out.println("after update return should be one: " + ans);
    System.out.println("----------------------------------------");
}
```

输出结果如下，注意下面的输出数据的 `_id`，正视前面插入的那条数据，两个数据唯一的不同，就是age被修改了

```
one match upsert return: AcknowledgedUpdateResult{matchedCount=1, modifiedCount=1, upsertedId=null}
after update return should be null: [{"add":["额外增加"],"name":"一灰灰blog","_id":{"counter":14491298,"date":1548333180000,"machineIdentifier":15099183,"processIdentifier":32282,"time":1548333180000,"timeSecond":1548333180,"timestamp":1548333180},"age":100}]
```

1.5. upsert，多条满足时

如果query条件命中多条数据，怎么办？会修改几条数据呢？

```
/**
 * 两条数据匹配时，upsert 将只会更新一条数据
 */
public void upsertTwoMatch() {
    // 多条数据满足条件时，只会修改一条数据
    System.out.println("----------------------------------------");
    List<JSONObject> re = mongoTemplate
            .find(new Query(Criteria.where("name").is("一灰灰
blog").and("age").in(Arrays.asList(28, 100))),
                    JSONObject.class, COLLECTION_NAME);
    System.out.println("original record: " + re);

    UpdateResult result = mongoTemplate
            .upsert(new Query(Criteria.where("name").is("一灰灰
blog").and("age").in(Arrays.asList(28, 100))),
                    new Update().set("age", 120), COLLECTION_NAME);
    System.out.println("two match upsert return: " + result);

    re = mongoTemplate.find(new Query(Criteria.where("name").is("一灰灰
blog").and("age").is(120)), JSONObject.class,
            COLLECTION_NAME);
    System.out.println("after upsert return size should be 1: " + re);
    System.out.println("----------------------------------------");
}
```

根据实际输出进行查看，发现只有一条数据被修改；另外一条保持不变，结果如下

```
----------------------------------------
original record: [{"name":"一灰灰blog","_id":
{"counter":12472353,"date":1548333180000,"machineIdentifier":14006254,"processIdentifi
er":17244,"time":1548333180000,"timeSecond":1548333180,"timestamp":1548333180},"age":2
8,"desc":"欢迎关注一灰灰Blog"}, {"add":["额外增加"],"name":"一灰灰blog","_id":
{"counter":14491298,"date":1548333180000,"machineIdentifier":15099183,"processIdentifi
er":32282,"time":1548333180000,"timeSecond":1548333180,"timestamp":1548333180},"age":1
00}]
two match upsert return: AcknowledgedUpdateResult{matchedCount=1, modifiedCount=1,
upsertedId=null}
after upsert return size should be 1: [{"name":"一灰灰blog","_id":
{"counter":12472353,"date":1548333180000,"machineIdentifier":14006254,"processIdentifi
er":17244,"time":1548333180000,"timeSecond":1548333180,"timestamp":1548333180},"age":1
20,"desc":"欢迎关注一灰灰Blog"}]
----------------------------------------
```

2. 其他

2.0. 项目

- 工程：spring-boot-demo
- module: mongo-template

相关博文

- 181213-SpringBoot高级篇MongoDB之基本环境搭建与使用
- 190113-SpringBoot高级篇MongoDB之查询基本使用姿势

修改基本使用姿势

本篇依然是MongoDB curd中的一篇，主要介绍document的更新，主要内容如下

- 常见类型成员的修改
- 数组类型成员的增删改
- document类型成员的增删改

1. 基本使用

首先是准备好基本环境，可以参考博文

- 181213-SpringBoot高级篇MongoDB之基本环境搭建与使用

- 190113-SpringBoot高级篇MongoDB之查询基本使用姿势

在开始之前，先封装一个输出方法，用于打印修改后的record对象

```
private void queryAndPrint(Query query, String tag) {
    System.out.println("------------ after " + tag + " age ------------");
    Map record = mongoTemplate.findOne(query, Map.class, COLLECTION_NAME);
    System.out.println("query records: " + record);
    System.out.println("------------  end " + tag + " age -------------\n");
}
```

1.1. 基本类型修改

mongodb支持我们常见的各种基本类型，而MongoTemplate也封装了不少对应的修改方法，最基础的修改，主要是借助`Update`来实现

常见的使用姿势如:

a. 基本使用姿势

```
public void basicUpdate() {
    /*
     * {
     *     "_id" : ObjectId("5c49b07ce6652f7e1add1ea2"),
     *     "age" : 100,
     *     "name" : "一灰灰blog",
     *     "desc" : "Java Developer",
     *     "add" : [
     *         "额外增加"
     *     ],
     *     "date" : ISODate("2019-01-28T08:00:08.373Z"),
     *     "doc" : {
     *         "key" : "小目标",
     *         "value" : "升职加薪，迎娶白富美"
     *     }
     * }
     */

    // 1. 直接修改值的内容
    Query query = new Query(Criteria.where("_id").is("5c49b07ce6652f7e1add1ea2"));

    Update update = new Update().set("desc", "Java & Python Developer");
    mongoTemplate.updateFirst(query, update, COLLECTION_NAME);
    queryAndPrint(query, "set");
}
```

输出结果为：

```
------------- after set age -------------
query records: {_id=5c49b07ce6652f7e1add1ea2, age=100, name=一灰灰blog, desc=Java &
Python Developer, add=[额外增加], date=Mon Jan 28 16:00:08 CST 2019, doc={key=小目标,
value=升职加薪，迎娶白富美}}
-------------  end set age --------------
```

b. 数字增加/减少

数字类型修改，使用 `org.springframework.data.mongodb.core.query.Update#inc`

```
// 数字修改，实现添加or减少
Update numUp = new Update().inc("age", 20L);
mongoTemplate.updateFirst(query, numUp, COLLECTION_NAME);
queryAndPrint(query, "inc");
```

输出结果为:

```
------------- after inc age -------------
query records: {_id=5c49b07ce6652f7e1add1ea2, age=120, name=一灰灰blog, desc=Java &
Python Developer, add=[额外增加], date=Mon Jan 28 16:00:08 CST 2019, doc={key=小目标,
value=升职加薪，迎娶白富美}}
-------------  end inc age --------------
```

c. 数字比较修改

数字简单比较之后修改，如 `org.springframework.data.mongodb.core.query.Update#max`

```
// 数字比较修改
Update cmpUp = new Update().max("age", 88);
mongoTemplate.updateFirst(query, cmpUp, COLLECTION_NAME);
queryAndPrint(query, "cmp");
```

输出结果

```
------------- after cmp age -------------
query records: {_id=5c49b07ce6652f7e1add1ea2, age=120, name=一灰灰blog, desc=Java &
Python Developer, add=[额外增加], date=Mon Jan 28 16:00:08 CST 2019, doc={key=小目标,
value=升职加薪，迎娶白富美}}
-------------  end cmp age --------------
```

d. 乘法

乘法运算, 主要使用 `org.springframework.data.mongodb.core.query.Update#multiply`

```
// 乘法
Update mulUp = new Update().multiply("age", 3);
mongoTemplate.updateFirst(query, mulUp, COLLECTION_NAME);
queryAndPrint(query, "multiply");
```

输出结果

```
------------- after multiply age -------------
query records: {_id=5c49b07ce6652f7e1add1ea2, age=360.0, name=一灰灰blog, desc=Java &
Python Developer, add=[额外增加], date=Mon Jan 28 16:00:08 CST 2019, doc={key=小目标,
value=升职加薪，迎娶白富美}}
-------------  end multiply age --------------
```

e. 日期修改

日期修改, 如 `org.springframework.data.mongodb.core.query.Update#currentDate`

```
// 日期修改
Update dateUp = new Update().currentDate("date");
mongoTemplate.updateFirst(query, dateUp, COLLECTION_NAME);
queryAndPrint(query, "date");
```

输出结果

```
------------- after date age -------------
query records: {_id=5c49b07ce6652f7e1add1ea2, age=360.0, name=一灰灰blog, desc=Java &
Python Developer, add=[额外增加], date=Mon Feb 18 19:34:56 CST 2019, doc={key=小目标,
value=升职加薪，迎娶白富美}}
-------------  end date age --------------
```

1.2. field修改

不同于mysql的列表是固定的,mongodb的field可以增加、删除和重命名，下面分别看下三种case如何使用

```
/**
 * 修改字段名，新增字段，删除字段
 */
public void fieldUpdate() {
    /**
     * {
     *      "_id" : ObjectId("5c6a7ada10ffc647d301dd62"),
     *      "age" : 28.0,
     *      "name" : "一灰灰blog",
     *      "desc" : "Java Developer",
     *      "add" : [
     *          "额外增加"
     *      ],
     *      "date" : ISODate("2019-01-28T08:00:08.373Z"),
     *      "doc" : {
     *          "key" : "小目标",
     *          "value" : "升职加薪，迎娶白富美"
     *      }
     * }
     */
    Query query = new Query(Criteria.where("_id").is("5c6a7ada10ffc647d301dd62"));
    renameFiled(query);

    addField(query);
    delField(query);
}
```

a. 重命名

利用`org.springframework.data.mongodb.core.query.Update#rename`来实现重命名，需要注意的是，当修改的docuemnt没有这个成员时，相当于没有任务操作

```
private void renameFiled(Query query) {
    Update update = new Update().rename("desc", "skill");
    mongoTemplate.updateFirst(query, update, COLLECTION_NAME);

    queryAndPrint(query, "rename");

    // 如果字段不存在，相当于没有更新
    update = new Update().rename("desc", "s-skill");
    mongoTemplate.updateFirst(query, update, COLLECTION_NAME);
    queryAndPrint(query, "rename Not exists!");
}
```

输出结果如下，后面一个语句相当于没有执行

```
------------- after rename age -------------
query records: {_id=5c6a7ada10ffc647d301dd62, age=28.0, name=一灰灰blog, add=[额外增
加], date=Mon Jan 28 16:00:08 CST 2019, doc={key=小目标, value=升职加薪，迎娶白富美},
skill=Java Developer}
-------------  end rename age --------------

------------- after rename Not exists! age -------------
query records: {_id=5c6a7ada10ffc647d301dd62, age=28.0, name=一灰灰blog, add=[额外增
加], date=Mon Jan 28 16:00:08 CST 2019, doc={key=小目标, value=升职加薪，迎娶白富美},
skill=Java Developer}
-------------  end rename Not exists! age --------------
```

b. 新增成员

新增也是直接利用的`Update#set`方法，当存在时，修改；不存在时，添加

- 另外提一下`setOnInsert`，如果要更新的文档存在那么$setOnInsert操作符不做任何处理；

```
private void addField(Query query) {
    // 新增一个字段
    // 直接使用set即可
    Update update = new Update().set("new-skill", "Python");
    mongoTemplate.updateFirst(query, update, COLLECTION_NAME);

    queryAndPrint(query, "addField");

    // 当更新一个不存在的文档时，可以使用setOnInsert
    // 如果要更新的文档存在那么$setOnInsert操作符不做任何处理；
}
```

输出结果如下：

```
------------- after addField age -------------
query records: {_id=5c6a7ada10ffc647d301dd62, age=28.0, name=一灰灰blog, add=[额外增
加], date=Mon Jan 28 16:00:08 CST 2019, doc={key=小目标, value=升职加薪，迎娶白富美},
skill=Java Developer, new-skill=Python}
-------------  end addField age --------------
```

c. 删除成员

删除document中的某个成员，借助`org.springframework.data.mongodb.core.query.Update#unset`， 正好与添加对上

```
private void delField(Query query) {
    // 删除字段，如果不存在，则不操作
    Update update = new Update().unset("new-skill");
    mongoTemplate.updateFirst(query, update, COLLECTION_NAME);

    queryAndPrint(query, "delField");
}
```

输出结果如下

```
------------- after delField age -------------
query records: {_id=5c6a7ada10ffc647d301dd62, age=28.0, name=一灰灰blog, add=[额外增
加], date=Mon Jan 28 16:00:08 CST 2019, doc={key=小目标, value=升职加薪，迎娶白富美},
skill=Java Developer}
-------------  end delField age --------------
```

1.3. 数组操作

在MongoDB的document中，有两个有意思的类型，一个是数组，一个是document（即可以嵌套），这里则主要介绍下如何操作数组中的成员

```
/**
 * 更新文档中字段为数组成员的值
 */
public void updateInnerArray() {
    /**
     * {
     *     "_id" : ObjectId("5c6a7ada10ffc647d301dd62"),
     *     "age" : 28.0,
     *     "name" : "一灰灰blog",
     *     "skill" : "Java Developer",
     *     "add" : [
     *         "额外增加"
     *     ],
     *     "date" : ISODate("2019-01-28T08:00:08.373Z"),
     *     "doc" : {
     *         "key" : "小目标",
     *         "value" : "升职加薪，迎娶白富美"
     *     }
     * }
     */
    Query query = new Query(Criteria.where("_id").is("5c6a7ada10ffc647d301dd62"));
    this.addData2Array(query);
    this.batchAddData2Array(query);
    this.delArrayData(query);
    this.updateArrayData(query);
}
```

a. 添加到数组中

在数组中新增一个数据，提供了两种方式，一个是`org.springframework.data.mongodb.core.query.Update#addToSet(java.lang.String, java.lang.Object)`，一个是`org.springframework.data.mongodb.core.query.Update#push(java.lang.String, java.lang.Object)`；两个的区别在于前者不能插入重复数据，后者可以

```
private void addData2Array(Query query) {
    // 新加一个元素到数组，如果已经存在，则不会加入
    String insert = "新添加>>" + System.currentTimeMillis();
    Update update = new Update().addToSet("add", insert);
    mongoTemplate.updateFirst(query, update, COLLECTION_NAME);
    queryAndPrint(query, "add2List");

    // push 新增元素，允许出现重复的数据
    update = new Update().push("add", 10);
    mongoTemplate.updateFirst(query, update, COLLECTION_NAME);
    queryAndPrint(query, "push2List");
}
```

输出结果

```
------------- after add2List age -------------
query records: {_id=5c6a7ada10ffc647d301dd62, age=28.0, name=一灰灰blog, add=[额外增加,
新添加>>1550489696892], date=Mon Jan 28 16:00:08 CST 2019, doc={key=小目标, value=升职
加薪，迎娶白富美}, skill=Java Developer}
-------------  end add2List age --------------

------------- after push2List age -------------
query records: {_id=5c6a7ada10ffc647d301dd62, age=28.0, name=一灰灰blog, add=[额外增加,
新添加>>1550489696892, 10], date=Mon Jan 28 16:00:08 CST 2019, doc={key=小目标, value=
升职加薪，迎娶白富美}, skill=Java Developer}
-------------  end push2List age --------------
```

b. 批量添加

一次添加多个，借助`addToSet`的`each`来实现

```
private void batchAddData2Array(Query query) {
    // 批量插入数据到数组中，注意不会将重复的数据丢入mongo数组中
    Update update = new Update().addToSet("add").each("2", "2", "3");
    mongoTemplate.updateFirst(query, update, COLLECTION_NAME);
    queryAndPrint(query, "batchAdd2List");
}
```

输出结果:

```
------------- after batchAdd2List age -------------
query records: {_id=5c6a7ada10ffc647d301dd62, age=28.0, name=一灰灰blog, add=[额外增加,
新添加>>1550489696892, 10, 2, 3], date=Mon Jan 28 16:00:08 CST 2019, doc={key=小目标,
value=升职加薪，迎娶白富美}, skill=Java Developer}
-------------  end batchAdd2List age --------------
```

c. 删除

借助pull来精确删除某个值

```
private void delArrayData(Query query) {
    // 删除数组中元素
    Update update = new Update().pull("add", "2");
    mongoTemplate.updateFirst(query, update, COLLECTION_NAME);
    queryAndPrint(query, "delArrayData");
}
```

输出如下，注意对比，`2`没有了

```
------------- after delArrayData age -------------
query records: {_id=5c6a7ada10ffc647d301dd62, age=28.0, name=一灰灰blog, add=[额外增加,
新添加>>1550489696892, 10, 3], date=Mon Jan 28 16:00:08 CST 2019, doc={key=小目标,
value=升职加薪，迎娶白富美}, skill=Java Developer}
-------------  end delArrayData age --------------
```

d. 修改

修改，首先的问题是要定位，确定删除数组中某个下标的元素，这里借助了一个有意思的站位

- 定位删除的数组元素方法：`arrayKey.index`
  - `arrayKey`是数组在docment中的名
  - `index`表示要删除的索引

一个实例如下

```
private void updateArrayData(Query query) {
    // 使用set，field.index 来更新数组中的值
    // 更新数组中的元素，如果元素存在，则直接更新；如果数组个数小于待更新的索引位置，则前面补null
    Update update = new Update().set("add.1", "updateField");
    mongoTemplate.updateFirst(query, update, COLLECTION_NAME);
    queryAndPrint(query, "updateListData");

    update = new Update().set("add.10", "nullBefore");
    mongoTemplate.updateFirst(query, update, COLLECTION_NAME);
    queryAndPrint(query, "updateListData");
}
```

输出结果，注意后面的，如果数组个数小于待更新的索引位置，则前面补null

```
------------- after updateListData age -------------
query records: {_id=5c6a7ada10ffc647d301dd62, age=28.0, name=一灰灰blog, add=[额外增加, updateField, 10, 3], date=Mon Jan 28 16:00:08 CST 2019, doc={key=小目标, value=升职加薪，迎娶白富美}, skill=Java Developer}
-------------  end updateListData age --------------

------------- after updateListData age -------------
query records: {_id=5c6a7ada10ffc647d301dd62, age=28.0, name=一灰灰blog, add=[额外增加, updateField, 10, 3, null, null, null, null, null, null, nullBefore], date=Mon Jan 28 16:00:08 CST 2019, doc={key=小目标, value=升职加薪，迎娶白富美}, skill=Java Developer}
-------------  end updateListData age --------------
```

1.4. document操作

内嵌文档，可以所是MongoDB的一个特色了，我们则来看下如何进行操作

```
/**
 * 更新文档中字段为document类型的值
 */
public void updateInnerDoc() {
    /**
     * {
     *     "_id" : ObjectId("5c6a956b10ffc647d301dd63"),
     *     "age" : 18.0,
     *     "name" : "一灰灰blog",
     *     "date" : ISODate("2019-02-28T08:00:08.373Z"),
     *     "doc" : {
     *         "key" : "小目标",
     *         "value" : "升职加薪，迎娶白富美"
     *     },
     *     "skill" : "Java Developer"
     * }
     */

    Query query = new Query(Criteria.where("_id").is("5c6a956b10ffc647d301dd63"));
    this.addFieldToDoc(query);
    this.updateFieldOfDoc(query);
    this.delFieldOfDoc(query);
}
```

a. 添加

借助前面的站位思想，就很好实现了，定位元素的方式采用

- docName.fieldName
  - docName 为内嵌文档在docunent中的fieldName
  - fieldName 为内嵌文档内部需要修改的fieldName

```
private void addFieldToDoc(Query query) {
    // 内嵌doc新增field
    Update update = new Update().set("doc.title", "好好学习，天天向上!");
    mongoTemplate.updateFirst(query, update, COLLECTION_NAME);
    queryAndPrint(query, "addFieldToDoc");
}
```

输出如下

```
------------- after addFieldToDoc age -------------
query records: {_id=5c6a956b10ffc647d301dd63, age=18.0, name=一灰灰blog, date=Thu Feb
28 16:00:08 CST 2019, doc={key=小目标, value=升职加薪，迎娶白富美, title=好好学习，天天向
上!}, skill=Java Developer}
-------------  end addFieldToDoc age --------------
```

c. 修改

```
private void updateFieldOfDoc(Query query) {
    // 内嵌doc修改field
    Update update = new Update().set("doc.title", "新的标题：一灰灰Blog!");
    mongoTemplate.updateFirst(query, update, COLLECTION_NAME);
    queryAndPrint(query, "updateFieldOfDoc");
}
```

输出如下

```
------------- after updateFieldOfDoc age -------------
query records: {_id=5c6a956b10ffc647d301dd63, age=18.0, name=一灰灰blog, date=Thu Feb
28 16:00:08 CST 2019, doc={key=小目标, value=升职加薪，迎娶白富美, title=新的标题：一灰灰
Blog!}, skill=Java Developer}
-------------  end updateFieldOfDoc age --------------
```

d. 删除

```
private void delFieldOfDoc(Query query) {
    // 删除内嵌doc中的field
    Update update = new Update().unset("doc.title");
    mongoTemplate.updateFirst(query, update, COLLECTION_NAME);
    queryAndPrint(query, "delFieldOfDoc");
}
```

输出如下

```
------------- after delFieldOfDoc age -------------
query records: {_id=5c6a956b10ffc647d301dd63, age=18.0, name=一灰灰blog, date=Thu Feb
28 16:00:08 CST 2019, doc={key=小目标, value=升职加薪，迎娶白富美}, skill=Java
Developer}
-------------  end delFieldOfDoc age --------------
```

2. 其他

2.0. 项目

- 工程：spring-boot-demo
- 子module：111-mongo-template
