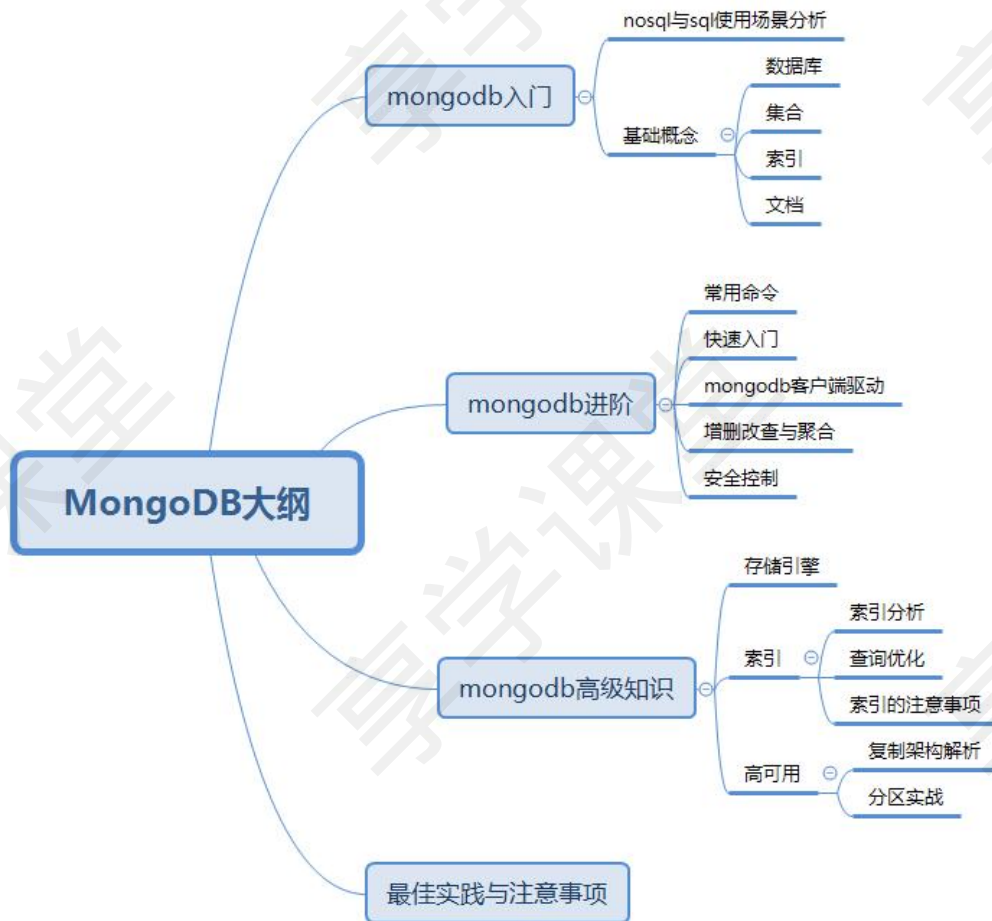


# 1. MongoDB 综述

## 1.1. 课程概述



## 1.2. 什么是 Nosql

NoSQL: Not Only SQL,本质也是一种数据库的技术，相对于传统数据库技术，它不会遵循一些约束，比如：sql 标准、ACID 属性，表结构等。

### Nosql 优点

- 满足对数据库的高并发读写
- 对海量数据的高效存储和访问
- 对数据库高扩展性和高可用性

- 灵活的数据结构，满足数据结构不固定的场景

#### Nosql 缺点

- 一般不支持事务
- 实现复杂 SQL 查询比较复杂
- 运维人员数据维护门槛较高
- 目前不是主流的数据库技术

### 1.2.1. NoSql 分类

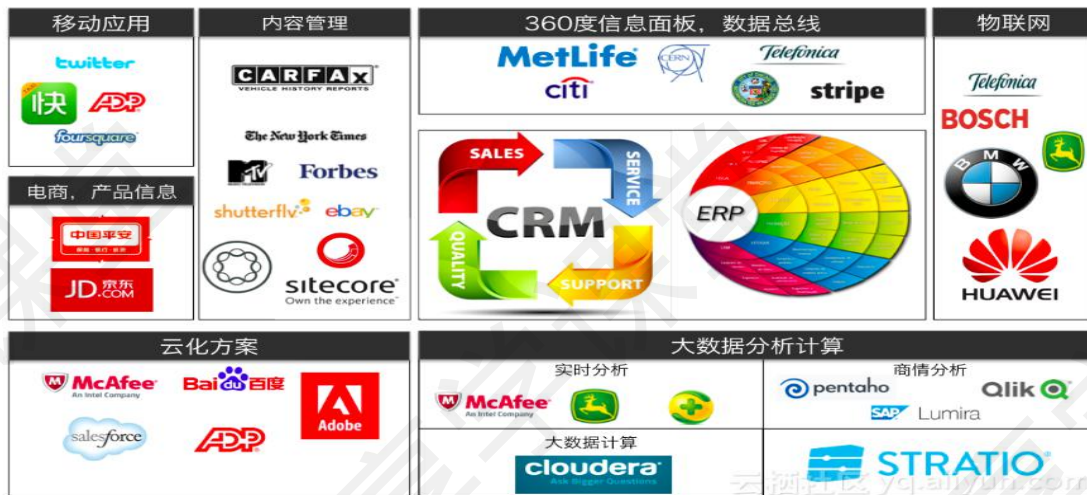
序号	类型	应用场景	典型产品
1	Key-value存储	缓存，处理高并发数据访问	Redis memcached
2	列式数据库	分布式文件系统	Cassandra Hbase
3	文档型数据库	Web应用，并发能力较强，表结构可变	mongoDB
4	图结构数据库	社交网络，推荐系统，关注构建图谱	infoGrid Neo4J

### 1.2.2. 数据库流程度排行

<https://db-engines.com/en/ranking>



### 1.2.3. 谁在使用 MongoDB



## 1.3. MongoDB 概念入门

### 1.3.1. 什么是 MongoDB

MongoDB: 是一个数据库 ,高性能、无模式、文档性，目前 nosql 中最热门的数据库，开源产品，基于 c++开发。是 nosql 数据库中功能最丰富，最像关系数据库的。

#### 特性

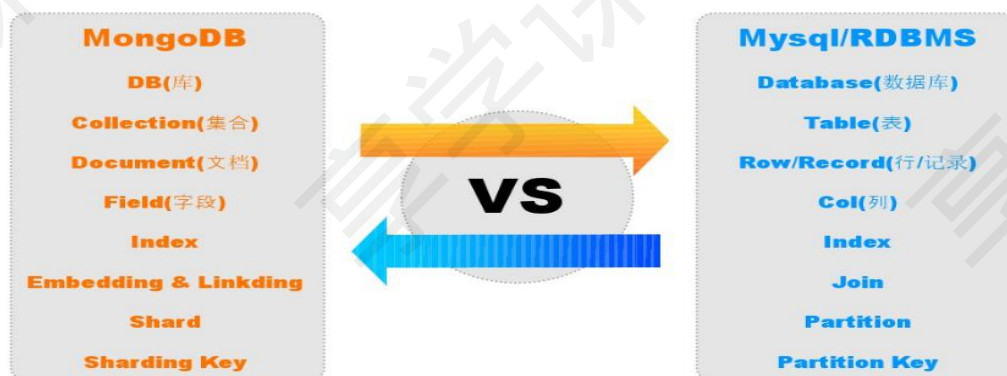
- 面向集合文档的存储：适合存储 Bson (json 的扩展) 形式的数据；
- 格式自由，数据格式不固定，生产环境下修改结构都可以不影响程序运行；
- 强大的查询语句，面向对象的查询语言，基本覆盖 sql 语言所有能力；
- 完整的索引支持，支持查询计划；
- 支持复制和自动故障转移；
- 支持二进制数据及大型对象（文件）的高效存储；

- 使用分片集群提升系统扩展性;
- 使用内存映射存储引擎, 把磁盘的 IO 操作转换为内存的操作;

### 1.3.2. MongoDB 基本概念



### 1.3.3. MongoDB 概念与 RDMS 概念对比



### 1.3.4. 应不应该用 MongoDB?

并没有某个业务场景必须要使用 MongoDB 才能解决, 但使用 MongoDB 通常能让你以更低成本解决问题 (包括学习、开发、运维等成本)

应用特征	Yes / No
应用不需要事务及复杂 join 支持	必须 Yes
新应用，需求会变，数据模型无法确定，想快速迭代开发	?
应用需要2000-3000以上的读写QPS（更高也可以）	?
应用需要TB甚至 PB 级别数据存储	?
应用发展迅速，需要能快速水平扩展	?
应用要求存储的数据不丢失	?
应用需要99.999%高可用	?
应用需要大量的地理位置查询、文本查询	?

如果上述有 1 个 Yes，可以考虑 MongoDB，2 个及以上的 Yes，选择 MongoDB 绝不会后悔！

### 1.3.5. MongoDB 使用场景

MongoDB 的应用已经渗透到各个领域，比如游戏、物流、电商、内容管理、社交、物联网、视频直播等，以下是几个实际的应用案例：

- 游戏场景，使用 MongoDB 存储游戏用户信息，用户的装备、积分等直接以内嵌文档的形式存储，方便查询、更新
- 物流场景，使用 MongoDB 存储订单信息，订单状态在运送过程中会不断更新，以 MongoDB 内嵌数组的形式来存储，一次查询就能将订单所有的变更读取出来。
- 社交场景，使用 MongoDB 存储用户信息，以及用户发表的朋友圈信息，通过地理位置索引实现附近的人、地点等功能
- 物联网场景，使用 MongoDB 存储所有接入的智能设备信息，以及设备汇报的日志信息，并对这些信息进行多维度的分析
- 视频直播，使用 MongoDB 存储用户信息、礼物信息等
- .....

### 1.3.6. 不使用 MongoDB 的场景

- 高度事务性系统：例如银行、财务等系统。MongoDB 对事物的支持较弱；
- 传统的商业智能应用：特定问题的数据分析，多数据实体关联，涉及到复杂的、高度优化的查询方式；
- 使用 sql 方便的时候；数据结构相对固定，使用 sql 进行查询统计更加便利的时候；



## 2. MongoDB 应用与开发

### 2.1. MongoDB 安装

- 官网下载安装介质：<https://www.mongodb.com/download-center>，选择适当的版本，这里以 linux 版本 mongodb-linux-x86\_64-4.0.4 为例；

[https://www.mongodb.org/dl/linux/x86\\_64](https://www.mongodb.org/dl/linux/x86_64)

```
tar zxvf mongodb-linux-x86_64-4.0.4.tgz
mv mongodb-linux-x86_64-4.0.4 mongodb
mkdir -p mongodb/{data/db,log,conf}
vi mongodb/conf/mgdb.conf
```

<https://docs.mongodb.com/v2.4/reference/configuration-options/>

```
dbpath=/soft/mongodb/data/db #数据文件存放目录
logpath=/soft/mongodb/log/mongodb.log #日志文件存放目录
port=27017 #端口，默认 27017，可以自定义
logappend=true #开启日志追加添加日志
fork=true #以守护程序的方式启用，即在后台运行
bind_ip=0.0.0.0 #本地监听 IP，0.0.0.0 表示本地所有 IP
auth=false #是否需要验证权限登录(用户名和密码)
```

修改环境变量

```
vi /etc/profile
export MONGODB_HOME=/soft/mongodb
export PATH=$PATH:$MONGODB_HOME/bin
source /etc/profile
```

配置开机启动

```
vi /usr/lib/systemd/system/mongodb.service
```

```
[Unit]
Description=mongodb
After=network.target remote-fs.target nss-lookup.target

[Service]
Type=forking
RuntimeDirectory=mongodb
PIDFile=/soft/mongodb/data/db/mongod.lock
ExecStart=/soft/mongodb/bin/mongod --config /soft/mongodb/conf/mgdb.conf
```

```
ExecStop=/soft/mongodb/bin/mongod --shutdown --config /soft/mongodb/conf/mgdb.conf
PrivateTmp=true

[Install]
WantedBy=multi-user.target
```

```
systemctl daemon-reload
systemctl start mongodb
systemctl enable mongodb
```

```
启动 mongodb
service mongodb stop
service mongodb start
```

<https://docs.mongodb.com/v4.0/reference/configuration-options/#storage.e.dbPath>

```
storage:
  dbPath: "/soft/mongodb/data/db"
systemLog:
  destination: file
  path: "/soft/mongodb/log/mongodb.log"
net:
  bindIp: 0.0.0.0
  port: 27017
processManagement:
  fork: true
setParameter:
  enableLocalhostAuthBypass: false
```

## 2.2. 快速入门

### 2.2.1. 目标

- 直观感受 mongoDB 的魅力
- mongo 开发入门（原生、spring）
- 开发框架版本选择
- mongoDB 数据类型全解析
- 对 nosql 的理念有初步的认识

执行命令

mongo

### 2.2.2. 数据结构介绍

```
{
  "_id" : ObjectId("59f938235d93fc4af8a37114"),
  "username" : "lison",
  "country" : "in11digo",
  "address" : {
    "aCode" : "邮编",
    "add" : "d11pff"
  },
  "favorites" : {
    "movies" : ["杀破狼 2","1dushe","雷神 1"],
    "cites" : ["1sh","1cs","1zz"]
  },
  "age" : 18,
  "salary" : NumberDecimal("2.099"),
  "lenght" : 1.79
}
```

### 2.2.3. 需求描述

- 新增 5 人

- 查询

查询喜欢的城市包含东莞和东京的 user

```
select * from users where favorites.cites has "东莞"、"东京"
```

查询国籍为英国或者美国，名字中包含 s 的 user

```
select * from users where username like '%s%' and (country= English or country= USA)
```

- 修改

把 lison 的年龄修改为 6 岁

```
update users set age=6 where username = lison'
```

喜欢的城市包含东莞的人，给他喜欢的电影加入"小电影 2""小电影 3"

```
update users set favorites.movies add "小电影 2 ", "小电影 3" where favorites.cites has
"东莞"
```

- 删除

删除名字为 lison 的 user

```
delete from users where username = 'lison'
```



删除年龄大于 8 小于 25 的 user

```
delete from users where age >8 and age <25
```

- 事务操作

Lison 和 james 要完成一次事务操作，james 转账 0.5 给 lison

```
update users set lenght= lenght-0.5 where username = 'james'
```

```
update users set lenght= lenght+0.5 where username = 'lison'
```

## 2.2.4. 使用 MongoDB 脚本实现

### 2.2.4.1. 新增 5 人

```
db.users.drop();
var user1 = {
  "username": "lison",
  "country": "china",
  "address": {
    "aCode": "411000",
    "add": "长沙"
  },
  "favorites": {
    "movies": ["杀破狼 2", "战狼", "雷神 1"],
    "cites": ["长沙", "深圳", "上海"]
  },
  "age": 18,
  "salary": NumberDecimal("18889.09"),
  "lenght": 1.79
};
var user2 = {
  "username": "james",
  "country": "English",
  "address": {
    "aCode": "311000",
    "add": "地址"
  },
  "favorites": {
```

```
        "movies": ["复仇者联盟","战狼","雷神 1"],
        "cites": ["西安","东京","上海"]
    },
    "age": 24,
    "salary": NumberDecimal("7889.09"),
    "lenght": 1.35
};
var user3 = {
    "username": "deer",
    "country": "japan",
    "address": {
        "aCode": "411000",
        "add": "长沙"
    },
    "favorites": {
        "movies": ["肉蒲团","一路向西","倩女幽魂"],
        "cites": ["东莞","深圳","东京"]
    },
    "age": 22,
    "salary": NumberDecimal("6666.66"),
    "lenght": 1.85
};
var user4 =
{
    "username": "mark",
    "country": "USA",
    "address": {
        "aCode": "411000",
        "add": "长沙"
    },
    "favorites": {
        "movies": ["蜘蛛侠","钢铁侠","蝙蝠侠"],
        "cites": ["青岛","东莞","上海"]
    },
    "age": 20,
    "salary": NumberDecimal("6398.22"),
    "lenght": 1.77
};
var user5 =
{
    "username": "peter",
    "country": "UK",
    "address": {
```

```

        "aCode" : "411000",
        "add" : "TEST"
    },
    "favorites" : {
        "movies" : ["蜘蛛侠","钢铁侠","蝙蝠侠"],
        "cites" : ["青岛","东莞","上海"]
    },
    "salary":NumberDecimal("1969.88")
};

db.users.insert(user1);
db.users.insert(user2);
db.users.insert(user3);
db.users.insert(user4);
db.users.insert(user5);

```

#### 2.2.4.2. 查询

查询喜欢的城市包含东莞和东京的 user

```

select * from users  where favorites.cites has "东莞"、"东京"
db.users.find({ "favorites.cites" : { "$all" : [ "东莞" , "东京"]}}).pretty()

```

查询国籍为英国或者美国，名字中包含 s 的 user

```

select * from users  where username like '%s%' and (country= English or country= USA)
db.users.find({ "$and" : [ { "username" : { "$regex" : ".*s.*"} } , { "$or" : [ { "country" :
"English"} , { "country" : "USA"} ] } ]}).pretty()

```

//思考 查询姓名是 deer 或者 james 的文档

#### 2.2.4.3. 修改

把 lison 的年龄修改为 6 岁

```

update  users  set age=6 where username = lison'
db.users.updateMany({ "username" : "lison"},{ "$set" : { "age" : 6}})

```

//思考，又过了一年，lison 年龄又涨了一岁

喜欢的城市包含东莞的人，给他喜欢的电影加入"小电影 2""小电影 3"

```

update users  set favorites.movies add "小电影 2 ", "小电影 3" where favorites.cites  has
"东莞"
db.users.updateMany({ "favorites.cites" : " 东 莞 "}, { "$addToSet" : { "favorites.movies" :
{"$each" : [ "小电影 2 " , "小电影 3"]}}},true)

```

#### 2.2.4.4. 删除

删除名字为 lison 的 user

```
delete from users where username = 'lison'  
db.users.deleteMany({ "username" : "lison" })
```

删除年龄大于 8 小于 25 的 user

```
delete from users where age >8 and age <25  
db.users.deleteMany({ "$and" : [ {"age" : {"$gt" : 8}}, {"age" : {"$lt" : 25}} ] })
```

#### 2.2.4.5. 事务操作

- 事务操作

Lison 和 james 要完成一次事务操作，james 转账 1 给 lison

begin

```
update users set lenght= lenght-1 where username = 'james'
```

```
update users set lenght= lenght+1 where username = 'lison'
```

commit

```
db.users.find({"username": {"$in":["lison", "james"]}}).pretty();
```

```
s = db.getMongo().startSession()  
s.startTransaction()  
  
db.users.update({"username" : "james"}, {"$inc": {"lenght": -1}})  
db.users.update({"username" : "lison"}, {"$inc": {"lenght": 1}})  
  
s.commitTransaction()  
s.abortTransaction()
```

注：以上操作是错误的方式，事务操作一定要在集群的环境下才可以，方式如下

```
usersCollection .find({"username": {"$in":["lison", "james"]}}).pretty();
```

```
s = db.getMongo().startSession();  
  
s.startTransaction()  
usersCollection = s.getDatabase("lison").users
```

```
usersCollection.update({"username" : "james"}, {"$inc": {"length": -1}})
usersCollection.update({"username" : "lison"}, {"$inc": {"length": 1}})

s.commitTransaction()
s.abortTransaction()
```

## 2.2.5. Java 客户端

### 2.2.5.1. 原始客户端

#### 2.2.5.1.1. 引入 pom 文件

```
<dependencies>
  <dependency>
    <groupId>org.mongodb</groupId>
    <artifactId>mongo-java-driver</artifactId>
    <version>3.11.2</version>
  </dependency>
  <dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
    <version>4.12</version>
  </dependency>
</dependencies>
```

#### 2.2.5.1.2. Document 方式

```
package cn.enjoy.mg;

import java.math.BigDecimal;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import java.util.function.Consumer;

import org.bson.Document;
```

```
import org.bson.conversions.Bson;
import org.junit.Before;
import org.junit.Test;

import com.mongodb.MongoClient;
import com.mongodb.client.FindIterable;
import com.mongodb.client.MongoCollection;
import com.mongodb.client.MongoDatabase;
import com.mongodb.client.result.DeleteResult;
import com.mongodb.client.result.UpdateResult;

import static com.mongodb.client.model.Updates.*;
import static com.mongodb.client.model.Filters.*;

//原生 java 驱动 document 的操作方式
public class QuickStartJavaDocTest {

    //数据库
    private MongoDatabase db;

    //文档集合
    private MongoCollection<Document> doc;

    //连接客户端（内置连接池）
    private MongoClient client;

    @Before
    public void init() {
        client = new MongoClient("192.168.244.123", 27017);
        db = client.getDatabase("lison");
        doc = db.getCollection("users");
    }

    @Test
    public void insertDemo() {
        Document doc1 = new Document();
        doc1.append("username", "cang");
        doc1.append("country", "USA");
        doc1.append("age", 20);
    }
}
```



```
doc1.append("lenght", 1.77f);
doc1.append("salary", new BigDecimal("6565.22")); //存金额，使用 bigdecimal 这个数据类型
```

```
//添加“address”子文档
```

```
Map<String, String> address1 = new HashMap<String, String>();
address1.put("aCode", "0000");
address1.put("add", "xxx000");
doc1.append("address", address1);
```

```
//添加“favorites”子文档，其中两个属性是数组
```

```
Map<String, Object> favorites1 = new HashMap<String, Object>();
favorites1.put("movies", Arrays.asList("aa", "bb"));
favorites1.put("cites", Arrays.asList("东莞", "东京"));
doc1.append("favorites", favorites1);
```

```
Document doc2 = new Document();
doc2.append("username", "Chen");
doc2.append("country", "China");
doc2.append("age", 30);
doc2.append("lenght", 1.77f);
doc2.append("salary", new BigDecimal("8888.22"));
Map<String, String> address2 = new HashMap<>();
address2.put("aCode", "411000");
address2.put("add", "我的地址 2");
doc2.append("address", address2);
Map<String, Object> favorites2 = new HashMap<>();
favorites2.put("movies", Arrays.asList("东游记", "一路向东"));
favorites2.put("cites", Arrays.asList("珠海", "东京"));
doc2.append("favorites", favorites2);
```

```
//使用 insertMany 插入多条数据
```

```
doc.insertMany(Arrays.asList(doc1, doc2));
```

```
}
```

```
@Test
```

```
public void testFind() {
```

```
    final List<Document> ret = new ArrayList<>();
```

```
    //block 接口专门用于处理查询出来的数据
```

```
    Consumer<Document> printDocument = new Consumer<Document>() {
```

```
        @Override
```

```
        public void accept(Document document) {
```

```
            System.out.println(document);
```

```

        ret.add(document);
    }
};
//select * from users  where favorites.cites has "东莞"、"东京"
//db.users.find({ "favorites.cites" : { "$all" : [ "东莞", "东京"]}})
Bson all = all("favorites.cites", Arrays.asList("东莞", "东京")); //定义数据过滤器，喜欢的城市中要包含"东莞"、"东京"
FindIterable<Document> find = doc.find(all);

find.forEach(printDocument);

System.out.println("----->" + String.valueOf(ret.size()));
ret.removeAll(ret);

//select * from users  where username like '%s%' and (contry= English or contry =
USA)
// db.users.find({ "$and" : [ { "username" : { "$regex" : ".*c.*"}}, { "$or" : [ { "country" : "English"}, { "country" : "USA"}] } ] })

String regexStr = ".*c.*";
Bson regex = regex("username", regexStr); //定义数据过滤器，username like '%s%'
Bson or = or(eq("country", "English"), eq("country", "USA")); //定义数据过滤器，(contry= English or contry = USA)
Bson and = and(regex, or);
FindIterable<Document> find2 = doc.find(and);
find2.forEach(printDocument);
System.out.println("----->" + String.valueOf(ret.size()));

}

@Test
public void testUpdate() {
    //update  users  set age=6 where username = 'lison'
    // db.users.updateMany({ "username" : "lison"}, { "$set" : { "age" : 6}}, true)

    Bson eq = eq("username", "cang"); //定义数据过滤器，username = 'cang'
    Bson set = set("age", 8); //更新的字段.来自于 Updates 包的静态导入
    UpdateResult updateMany = doc.updateMany(eq, set);
    System.out.println("----->" +
String.valueOf(updateMany.getModifiedCount()); //打印受影响的行数

    //update users  set favorites.movies add "小电影 2 ", "小电影 3" where favorites.cites

```

```

has "东莞"
        //db.users.updateMany({ "favorites.cites" : " 东 莞 "}, { "$addToSet" :
{ "favorites.movies" : { "$each" : [ "小电影 2 ", "小电影 3"] } }},true)

        Bson eq2 = eq("favorites.cites", "东莞");//定义数据过滤器，favorites.cites  has "东莞"
"

        Bson addEachToSet = addEachToSet("favorites.movies", Arrays.asList("小电影 2 ", "小
电影 3"));//更新的字段来自于 Updates 包的静态导入
        UpdateResult updateMany2 = doc.updateMany(eq2, addEachToSet);
        System.out.println("----->"+
String.valueOf(updateMany2.getModifiedCount());
    }

    @Test
    public void testDelete() {

        //delete from users where username = 'lison'
        //db.users.deleteMany({ "username" : "lison" })
        Bson eq = eq("username", "lison");//定义数据过滤器， username='lison'
        DeleteResult deleteMany = doc.deleteMany(eq);
        System.out.println("----->"+
String.valueOf(deleteMany.getDeletedCount()));//打印受影响的行数

        //delete from users where age >8 and age <25
        //db.users.deleteMany({ "$and" : [ {"age" : {"$gt": 8}} , {"age" : {"$lt" : 25}} ] })

        Bson gt = gt("age", 8);//定义数据过滤器， age > 8，所有过滤器的定义来自于 Filter
这个包的静态方法，需要频繁使用所以静态导入
        // Bson gt = Filter.gt("age",8);

        Bson lt = lt("age", 25);//定义数据过滤器， age < 25
        Bson and = and(gt, lt);//定义数据过滤器，将条件用 and 拼接
        DeleteResult deleteMany2 = doc.deleteMany(and);
        System.out.println("----->"+
String.valueOf(deleteMany2.getDeletedCount()));//打印受影响的行数
    }

    @Test
    public void testTransaction() {
        // begin
        // update  users  set lenght= lenght-1  where username = 'james'
        // update  users  set lenght= lenght+1  where username = 'lison'
        // commit

```

```

        ClientSession clientSession = client.startSession();
        clientSession.startTransaction();
        Bson eq = eq("username", "james");
        Bson inc = inc("length", -1);
        doc.updateOne(clientSession,eq,inc);

        Bson eq2 = eq("username", "lison");
        Bson inc2 = inc("length", 1);

        doc.updateOne(clientSession,eq2,inc2);

        clientSession.commitTransaction();
        // clientSession.abortTransaction();

    }
}

```

### 2.2.5.1.3. POJO 方式

新增 Favorites

```

package cn.enjoy.entity;

import java.util.List;

public class Favorites {
    private List<String> movies;
    private List<String> cites;
    public List<String> getMovies() {
        return movies;
    }
    public void setMovies(List<String> movies) {
        this.movies = movies;
    }
    public List<String> getCites() {
        return cites;
    }
    public void setCites(List<String> cites) {
        this.cites = cites;
    }
}

```

```
@Override
public String toString() {
    return "Favorites [movies=" + movies + ", cites=" + cites + "]";
}
}
```

新增 Address

```
package cn.enjoy.entity;

public class Address {

    private String aCode;
    private String add;
    public String getaCode() {
        return aCode;
    }
    public void setaCode(String aCode) {
        this.aCode = aCode;
    }
    public String getAdd() {
        return add;
    }
    public void setAdd(String add) {
        this.add = add;
    }
    @Override
    public String toString() {
        return "Address [aCode=" + aCode + ", add=" + add + "]";
    }
}
```

新增 User

```
package cn.enjoy.entity;

import java.math.BigDecimal;

import org.bson.types.ObjectId;

public class User {

    private ObjectId id;
```

```
private String username;

private String country;

private Address address;

private Favorites favorites;

private int age;

private BigDecimal salary;

private float lenght;

public String getUsername() {
    return username;
}

public void setUsername(String username) {
    this.username = username;
}

public String getCountry() {
    return country;
}

public void setCountry(String country) {
    this.country = country;
}

public Address getAddress() {
    return address;
}

public void setAddress(Address address) {
    this.address = address;
}

public Favorites getFavorites() {
    return favorites;
}

public void setFavorites(Favorites favorites) {
    this.favorites = favorites;
}

public Objectid getId() {
    return id;
}

public void setId(Objectid id) {
```



```

        this.id = id;
    }
    public int getAge() {
        return age;
    }
    public void setAge(int age) {
        this.age = age;
    }
    public BigDecimal getSalary() {
        return salary;
    }
    public void setSalary(BigDecimal salary) {
        this.salary = salary;
    }
    public float getLenght() {
        return lenght;
    }
    public void setLenght(float lenght) {
        this.lenght = lenght;
    }
}

@Override
public String toString() {
    return "User [id=" + id + ", username=" + username + ", country="
        + country + ", address=" + address + ", favorites=" + favorites
        + ", age=" + age + ", salary=" + salary + ", lenght=" + lenght + "];"
}

}

```

```

package cn.enjoy.mg;

import static com.mongodb.client.model.Updates.*;
import static com.mongodb.client.model.Filters.*;

import java.math.BigDecimal;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;
import java.util.function.Consumer;

```

```
import org.bson.Document;
import org.bson.codecs.configuration.CodecRegistries;
import org.bson.codecs.configuration.CodecRegistry;
import org.bson.codecs.pojo.PojoCodecProvider;
import org.bson.conversions.Bson;
import org.junit.Before;
import org.junit.Test;

import cn.enjoy.entity.Address;
import cn.enjoy.entity.Favorites;
import cn.enjoy.entity.User;
import com.mongodb.MongoClient;
import com.mongodb.MongoClientOptions;
import com.mongodb.ServerAddress;
import com.mongodb.client.FindIterable;
import com.mongodb.client.MongoCollection;
import com.mongodb.client.MongoDatabase;
import com.mongodb.client.model.Filters;
import com.mongodb.client.model.Updates;
import com.mongodb.client.result.DeleteResult;
import com.mongodb.client.result.UpdateResult;

//原生 java 驱动 Pojo 的操作方式
public class QuickStartJavaPojoTest {

    private MongoDBDatabase db;

    private MongoCollection<User> doc;

    private MongoClient client;

    @Before
    public void init(){
        //编解码器的 list
        List<CodecRegistry> codecResgistes = new ArrayList<>();
        //list 加入默认的编解码器集合
        codecResgistes.add(MongoClient.getDefaultCodecRegistry());
        //生成一个 pojo 的编解码器
        CodecRegistry pojoCodecRegistry = CodecRegistries.
            fromProviders(PojoCodecProvider.builder().automatic(true).build());
```

```

//list 加入 pojo 的编解码器
codecResgistes.add(pojoCodecRegistry);
//通过编解码器的 list 生成编解码器注册中心
CodecRegistry registry = CodecRegistries.fromRegistries(codecResgistes);

//把编解码器注册中心放入 MongoClientOptions
//MongoClientOptions 相当于连接池的配置信息
MongoClientOptions build = MongoClientOptions.builder().
    codecRegistry(registry).build();

ServerAddress serverAddress = new ServerAddress("192.168.244.123", 27017);

client = new MongoClient(serverAddress, build);
db =client.getDatabase("lison");
doc = db.getCollection("users",User.class);
}

```

```

@Test
public void insertDemo(){
    User user = new User();
    user.setUsername("cang");
    user.setCountry("USA");
    user.setAge(20);
    user.setLenght(1.77f);
    user.setSalary(new BigDecimal("6265.22"));

    //添加“address”子文档
    Address address1 = new Address();
    address1.setaCode("411222");
    address1.setAdd("sdfsdf");
    user.setAddress(address1);

    //添加“favorites”子文档，其中两个属性是数组
    Favorites favorites1 = new Favorites();
    favorites1.setCites(Arrays.asList("东莞","东京"));
    favorites1.setMovies(Arrays.asList("西游记","一路向西"));
    user.setFavorites(favorites1);

    User user1 = new User();
    user1.setUsername("chen");
    user1.setCountry("China");
}

```

```

        user1.setAge(30);
        user1.setLenght(1.77f);
        user1.setSalary(new BigDecimal("6885.22"));
        Address address2 = new Address();
        address2.setaCode("411000");
        address2.setAdd("我的地址 2");
        user1.setAddress(address2);
        Favorites favorites2 = new Favorites();
        favorites2.setCites(Arrays.asList("珠海","东京"));
        favorites2.setMovies(Arrays.asList("东游记","一路向东"));
        user1.setFavorites(favorites2);

        //使用 insertMany 插入多条数据
        doc.insertMany(Arrays.asList(user,user1));

    }

    @Test
    public void testFind(){

        final List<User> ret = new ArrayList<>();
        Consumer<User> printDocument = new Consumer<User>() {
            @Override
            public void accept(User t) {
                System.out.println(t.toString());
                ret.add(t);
            }
        };

        //select * from users  where favorites.cites has "东莞"、"东京"
        //db.users.find({ "favorites.cites" : { "$all" : [ "东莞", "东京"]}})
        Bson all = all("favorites.cites", Arrays.asList("东莞","东京")); //定义数据过滤器，喜欢的城市中要包含"东莞"、"东京"
        FindIterable<User> find = doc.find(all);
        find.forEach(printDocument);
        System.out.println("----->" + String.valueOf(ret.size()));
        ret.removeAll(ret);

        //select * from users  where username like '%s%' and (contry= English or contry =
        USA)
        // db.users.find({ "$and" : [ { "username" : { "$regex" : ".*c.*"} }, { "$or" : [ { "country" :

```

```

"English"}, { "country" : "USA"}}}})
    String regexStr = ".*c.*";
    Bson regex = regex("username", regexStr);//定义数据过滤器, username like '%s%'
    Bson or = or(eq("country","English"),eq("country","USA"));//定义数据过滤器, (contry=
English or contry = USA)
    FindIterable<User> find2 = doc.find(and(regex,or));
    find2.forEach(printDocument);
    System.out.println("----->" + String.valueOf(ret.size()));

}

@Test
public void testUpdate(){
    //update  users  set age=6 where username = 'lison'
    //db.users.updateMany({ "username" : "lison"}, { "$set" : { "age" : 6}},true)
    Bson eq = eq("username", "lison");//定义数据过滤器, username = 'lison'
    Bson set = set("age", 8);//更新的字段.来自于 Updates 包的静态导入
    UpdateResult updateMany = doc.updateMany(eq, set);

    System.out.println("----->" + String.valueOf(updateMany.getModifiedCount()));//打
    印受影响的行数

    //update users  set favorites.movies add "小电影 2 ", "小电影 3" where favorites.cites
    has "东莞"
    //db.users.updateMany({ "favorites.cites" : "东莞"}, { "$addToSet" :
    { "favorites.movies" : { "$each" : [ "小电影 2 ", "小电影 3"]}}},true)
    Bson eq2 = eq("favorites.cites", "东莞");//定义数据过滤器, favorites.cites  has "东莞
    "

    Bson addEachToSet = addEachToSet("favorites.movies", Arrays.asList( "小电影 2 ", "小
    电影 3"));//更新的字段.来自于 Updates 包的静态导入
    UpdateResult updateMany2 = doc.updateMany(eq2, addEachToSet);

    System.out.println("----->" + String.valueOf(updateMany2.getModifiedCount()));
}

@Test
public void testDelete(){

    //delete from users where username = 'lison'
    //db.users.deleteMany({ "username" : "lison" })
    Bson eq = eq("username", "lison");//定义数据过滤器, username='lison'
    DeleteResult deleteMany = doc.deleteMany(eq);
    System.out.println("----->" + String.valueOf(deleteMany.getDeletedCount()));//

```

打印受影响的行数

```
//delete from users where age >8 and age <25
//db.users.deleteMany({"$and" : [ {"age" : {"$gt": 8}}, {"age" : {"$lt" : 25}}]})
Bson gt = gt("age",8);//定义数据过滤器，age > 8，所有过滤器的定义来自于 Filter
这个包的静态方法，需要频繁使用所以静态导入

Bson lt = lt("age",25);//定义数据过滤器，age < 25
Bson and = and(gt,lt);//定义数据过滤器，将条件用 and 拼接
DeleteResult deleteMany2 = doc.deleteMany(and);

System.out.println("----->"+String.valueOf(deleteMany2.getDeletedCount()));//打印受影响的行数
}
```

com.mongodb.MongoClient

```
public class MongoClient extends Mongo implements Closeable {
    public static CodecRegistry getDefaultCodecRegistry() {
        return MongoClientSettings.getDefaultCodecRegistry();
    }
}
```

## 2.2.5.2. Spring-data-mongodb 客户端

### 2.2.5.2.1. 引入 Spring 等 jar

```
<dependency>
    <groupId>org.springframework.data</groupId>
    <artifactId>spring-data-mongodb</artifactId>
    <version>2.2.1.RELEASE</version>
</dependency>

<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-context</artifactId>
    <version>5.2.1.RELEASE</version>
</dependency>
```



```

<dependency>
  <groupId>org.springframework</groupId>
  <artifactId>spring-core</artifactId>
  <version>5.2.1.RELEASE</version>
</dependency>
<dependency>
  <groupId>org.springframework</groupId>
  <artifactId>spring-context-support</artifactId>
  <version>5.2.1.RELEASE</version>
</dependency>
<dependency>
  <groupId>org.springframework</groupId>
  <artifactId>spring-beans</artifactId>
  <version>5.2.1.RELEASE</version>
</dependency>
<dependency>
  <groupId>org.springframework</groupId>
  <artifactId>spring-aop</artifactId>
  <version>5.2.1.RELEASE</version>
</dependency>

<dependency>
  <groupId>org.springframework</groupId>
  <artifactId>spring-test</artifactId>
  <version>5.2.1.RELEASE</version>
</dependency>

```

#### 2.2.5.2.2. 新增 applicationContext.xml

在 resources 目录下新增 spring 配置文件

```

<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:context="http://www.springframework.org/schema/context"
  xmlns:mongo="http://www.springframework.org/schema/data/mongo"
  xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans.xsd
    http://www.springframework.org/schema/context

```

```
http://www.springframework.org/schema/context/spring-context.xsd
http://www.springframework.org/schema/data/mongo

http://www.springframework.org/schema/data/mongo/spring-mongo.xsd">

<context:component-scan base-package="cn.enjoy">

</context:component-scan>

<!-- mongodb 连接池配置 -->
<mongo:mongo-client id="mongo" host="192.168.244.123" port="27017">
    <mongo:client-options
        write-concern="ACKNOWLEDGED"
        threads-allowed-to-block-for-connection-multiplier="5"
        max-wait-time="1200"
        connect-timeout="1000"/>
</mongo:mongo-client>

<!-- mongodb 数据库工厂配置 -->
<mongo:db-factory dbname="lison" mongo-ref="mongo" />

<!-- mongodb 模板配置 -->
<bean                                id="anotherMongoTemplate"
class="org.springframework.data.mongodb.core.MongoTemplate">
    <constructor-arg name="mongoDbFactory" ref="mongoDbFactory" />
</bean>

</beans>
```

### 2.2.5.2.3. 修改实体类

```
package cn.enjoy.entity;

import java.math.BigDecimal;

import org.bson.types.ObjectId;
import org.springframework.data.mongodb.core.mapping.Document;

@Document(collection="users")
```

```
public class User {  
  
    private Objectid id;  
  
    private String username;  
  
    private String country;  
  
    private Address address;  
  
    private Favorites favorites;  
  
    private int age;  
  
    private BigDecimal salary;  
  
    private float lenght;  
  
    public String getUsername() {  
        return username;  
    }  
    public void setUsername(String username) {  
        this.username = username;  
    }  
    public String getCountry() {  
        return country;  
    }  
    public void setCountry(String country) {  
        this.country = country;  
    }  
    public Address getAddress() {  
        return address;  
    }  
    public void setAddress(Address address) {  
        this.address = address;  
    }  
    public Favorites getFavorites() {  
        return favorites;  
    }  
    public void setFavorites(Favorites favorites) {  
        this.favorites = favorites;  
    }  
    public Objectid getId() {  
        return id;  
    }  
}
```

```

    }
    public void setId(ObjectId id) {
        this.id = id;
    }
    public int getAge() {
        return age;
    }
    public void setAge(int age) {
        this.age = age;
    }
    public BigDecimal getSalary() {
        return salary;
    }
    public void setSalary(BigDecimal salary) {
        this.salary = salary;
    }
    public float getLenght() {
        return lenght;
    }
    public void setLenght(float lenght) {
        this.lenght = lenght;
    }
}

@Override
public String toString() {
    return "User [id=" + id + ", username=" + username + ", country="
        + country + ", address=" + address + ", favorites=" + favorites
        + ", age=" + age + ", salary=" + salary + ", lenght=" + lenght + "];"
}
}

```

#### 2.2.5.2.4. 新增单元测试

```

package cn.enjoy.mg;

import static org.springframework.data.mongodb.core.query.Criteria.where;
import static org.springframework.data.mongodb.core.query.Query.query;
import static org.springframework.data.mongodb.core.query.Update.update;

```

```
import java.math.BigDecimal;
import java.util.Arrays;
import java.util.List;

import javax.annotation.Resource;

import cn.enjoy.entity.Address;
import cn.enjoy.entity.Favorites;
import cn.enjoy.entity.User;
import com.mongodb.client.result.DeleteResult;
import com.mongodb.client.result.UpdateResult;
import org.junit.Test;
import org.junit.runner.RunWith;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.data.mongodb.core.MongoOperations;
import org.springframework.data.mongodb.core.query.Criteria;
import org.springframework.data.mongodb.core.query.Query;
import org.springframework.data.mongodb.core.query.Update;
import org.springframework.test.context.ContextConfiguration;
import org.springframework.test.context.junit4.SpringJUnit4ClassRunner;

//spring Pojo 的操作方式
@RunWith(SpringJUnit4ClassRunner.class)
@ContextConfiguration("classpath:applicationContext.xml")
public class QuickStartSpringPojoTest {

    @Resource
    private MongoOperations tempelate;

    @Test
    public void insertDemo(){
        User user = new User();
        user.setUsername("cang");
        user.setCountry("USA");
        user.setAge(20);
        user.setLenght(1.77f);
        user.setSalary(new BigDecimal("6265.22"));

        //添加“address”子文档
        Address address1 = new Address();
```

```

address1.setaCode("411222");
address1.setAdd("sdfsdf");
user.setAddress(address1);

//添加“favorites”子文档，其中两个属性是数组
Favorites favorites1 = new Favorites();
favorites1.setCites(Arrays.asList("东莞","东京"));
favorites1.setMovies(Arrays.asList("西游记","一路向西"));
user.setFavorites(favorites1);

User user1 = new User();
user1.setUsername("chen");
user1.setCountry("China");
user1.setAge(30);
user1.setLenght(1.77f);
user1.setSalary(new BigDecimal("6885.22"));
Address address2 = new Address();
address2.setaCode("411000");
address2.setAdd("我的地址 2");
user1.setAddress(address2);
Favorites favorites2 = new Favorites();
favorites2.setCites(Arrays.asList("珠海","东京"));
favorites2.setMovies(Arrays.asList("东游记","一路向东"));
user1.setFavorites(favorites2);

templete.insertAll(Arrays.asList(user,user1));
}

@Test
public void testFind(){

//select * from users  where favorites.cites has "东莞"、"东京"
//db.users.find({ "favorites.cites" : { "$all" : [ "东莞" , "东京"]}})
Criteria all = where("favorites.cites").all(Arrays.asList("东莞","东京"));
List<User> find = templete.find(query(all), User.class);
System.out.println(find.size());
for (User user : find) {
    System.out.println(user.toString());
}

//select * from users  where username like '%s%' and (contry= English or contry =
USA)

```



```

        // db.users.find({ "$and" : [ { "username" : { "$regex" : ".*s.*" } }, { "$or" : [ { "country" :
"English" }, { "country" : "USA" } ] } ] })
        String regexStr = ".*c.*";
        //username like '%s%'
        Criteria regex = where("username").regex(regexStr);
        //contry= EngLish
        Criteria or1 = where("country").is("English");
        //contry= USA
        Criteria or2 = where("country").is("USA");

        Criteria or = new Criteria().orOperator(or1,or2);

        Query query = query(new Criteria().andOperator(regex,or));

        List<User> find2 = tempelate.find(query, User.class);

        System.out.println(find2.size());
        for (User user : find2) {
            System.out.println(user.toString());
        }
    }

    @Test
    public void testUpdate(){
        //update  users  set age=6 where username = 'lison'
        //db.users.updateMany({ "username" : "lison"},{ "$set" : { "age" : 6}},true)
        Query query = query(where("username").is("lison"));
        Update update = update("age", 6);
        UpdateResult updateFirst = tempelate.updateMulti(query, update, User.class);
        System.out.println(updateFirst.getModifiedCount());

        //update users  set favorites.movies add "小电影 2 ", "小电影 3" where favorites.cites
has "东莞"
        //db.users.updateMany({ "favorites.cites" : " 东 莞 "}, { "$addToSet" :
{ "favorites.movies" : { "$each" : [ "小电影 2 ", "小电影 3" ] } } },true)
        query = query(where("favorites.cites").is("东莞"));
        update = new Update().addToSet("favorites.movies").each("小电影 2 ", "小电影 3");
        UpdateResult updateMulti = tempelate.updateMulti(query, update, User.class);
        System.out.println("----->" + updateMulti.getModifiedCount());
    }

    @Test
    public void testDelete(){

```

```

//delete from users where username = 'lison'
//db.users.deleteMany({ "username" : "lison" } )
Query query = query(where("username").is("lison"));
DeleteResult remove = template.remove(query, User.class);
System.out.println("----->" + remove.getDeletedCount());

//delete from users where age >8 and age <25
//db.users.deleteMany({"$and" : [ {"age" : {"$gt": 8}}, {"age" : {"$lt" : 25}}]})
query = query(new Criteria().andOperator(where("age").gt(8), where("age").lt(25)));
DeleteResult remove2 = template.remove(query, User.class);
System.out.println("----->" + remove2.getDeletedCount());
}
}

```

## 2.2.5.2.5. 事务测试

### 2.2.5.2.5.1. 修改 applicationContext.xml

```

<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:context="http://www.springframework.org/schema/context"
       xmlns:mongo="http://www.springframework.org/schema/data/mongo"
       xmlns:tx="http://www.springframework.org/schema/tx"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
                           http://www.springframework.org/schema/beans/spring-beans.xsd
                           http://www.springframework.org/schema/context
                           http://www.springframework.org/schema/context/spring-context.xsd
                           http://www.springframework.org/schema/tx
                           http://www.springframework.org/schema/tx/spring-tx.xsd
                           http://www.springframework.org/schema/data/mongo
                           http://www.springframework.org/schema/data/mongo/spring-mongo.xsd">

    <context:component-scan base-package="cn.enjoy">

    </context:component-scan>

    <!-- mongodb 连接池配置 -->

```

```

<!--<mongo:mongo-client      id="mongo"      host="192.168.244.123"      port="27017"
credentials="lison:lison@lison">-->
<mongo:mongo-client id="mongo" host="192.168.244.123" port="27017">
  <mongo:client-options
    write-concern="ACKNOWLEDGED"
    threads-allowed-to-block-for-connection-multiplier="5"
    max-wait-time="1200"
    connect-timeout="1000"/>
</mongo:mongo-client>

<!-- mongodb 数据库工厂配置 -->
<mongo:db-factory  dbname="lison" mongo-ref="mongo" />

<tx:annotation-driven transaction-manager="transactionManager"/>

<bean                                     id="transactionManager"
class="org.springframework.data.mongodb.MongoTransactionManager">
  <property name="dbFactory" ref="mongoDbFactory"/>
</bean>

<!-- mongodb 模板配置 -->
<bean                                     id="anotherMongoTemplate"
class="org.springframework.data.mongodb.core.MongoTemplate">
  <constructor-arg name="mongoDbFactory" ref="mongoDbFactory" />
</bean>

</beans>

```

#### 2.2.5.2.5.2. 新增 UserService

```

package cn.enjoy.service;

public interface UserService {

    void doTransaction();

}

```

#### 2.2.5.2.5.3. 新增实现类 UserServiceImpl

```

package cn.enjoy.service.impl;

```

```

import cn.enjoy.entity.User;
import cn.enjoy.service.UserService;
import com.mongodb.MongoClient;
import org.springframework.data.mongodb.core.MongoOperations;
import org.springframework.data.mongodb.core.query.Query;
import org.springframework.data.mongodb.core.query.Update;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;

import javax.annotation.Resource;

import static org.springframework.data.mongodb.core.query.Criteria.where;
import static org.springframework.data.mongodb.core.query.Query.query;

@Service
public class UserServiceImpl implements UserService{

    @Resource
    private MongoOperations tempelate;

    @Override
    @Transactional
    public void doTransaction() {
        Query query = query(where("username").is("lison"));
        Update update = new Update().inc("lenght",1);
        tempelate.updateMulti(query,update, User.class);

        query = query(where("username").is("james"));
        update = new Update().inc("lenght",-1);
        tempelate.updateMulti(query,update, User.class);
    }
}

```

#### 2.2.5.2.5.4. 修改 QuickStartSpringPojoTest

增加 spring 事务单元测试

```

@Test
public void doTransaction() {
    userService.doTransaction();
}

```

### 2.2.5.3. 日志显示

如果需要显示日志

```
<!-- 日志相关依赖 -->
<dependency>
    <groupId>org.slf4j</groupId>
    <artifactId>slf4j-api</artifactId>
    <version>1.7.10</version>
</dependency>
<dependency>
    <groupId>ch.qos.logback</groupId>
    <artifactId>logback-classic</artifactId>
    <version>1.1.2</version>
</dependency>
<dependency>
    <groupId>ch.qos.logback</groupId>
    <artifactId>logback-core</artifactId>
    <version>1.1.2</version>
</dependency>
```

在 resource 目录下新增 logback.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!--
scan: 当此属性设置为 true 时，配置文件如果发生改变，将会被重新加载，默认值为 true。
scanPeriod: 设置监测配置文件是否有修改的时间间隔，如果没有给出时间单位，默认单位是毫秒当 scan 为 true 时，此属性生效。默认的时间间隔为 1 分钟。
debug: 当此属性设置为 true 时，将打印出 logback 内部日志信息，实时查看 logback 运行状态。默认值为 false。
-->
<configuration scan="false" scanPeriod="60 seconds" debug="false">
    <!-- 定义日志的根目录 -->
    <!-- <property name="LOG_HOME" value="/app/log" /> -->
    <!-- 定义日志文件名称 -->
    <property name="appName" value="netty"></property>
    <!-- ch.qos.logback.core.ConsoleAppender 表示控制台输出 -->
    <appender name="stdout" class="ch.qos.logback.core.ConsoleAppender">
        <Encoding>UTF-8</Encoding>
    <!--
        日志输出格式: %d 表示日期时间，%thread 表示线程名，%-5level: 级别从左显示 5 个字符宽度
        %logger{50} 表示 logger 名字最长 50 个字符，否则按照句点分割。 %msg: 日志消息，%n 是换行符
    -->
```

```

-->
<encoder>
  <pattern>%d{yyyy-MM-dd HH:mm:ss.SSS} [%thread] %-5level %logger{50}
- %msg%n</pattern>
</encoder>
</appender>

<!-- 滚动记录文件，先将日志记录到指定文件，当符合某个条件时，将日志记录到其他
文件 -->
<appender name="appLogAppender"
class="ch.qos.logback.core.rolling.RollingFileAppender">
  <Encoding>UTF-8</Encoding>
  <!-- 指定日志文件的名称 -->
  <file>cache-demo2.log</file>
  <!--
  当发生滚动时，决定 RollingFileAppender 的行为，涉及文件移动和重命名
  TimeBasedRollingPolicy：最常用的滚动策略，它根据时间来制定滚动策略，既负责
滚动也负责出发滚动。
  -->
  <rollingPolicy class="ch.qos.logback.core.rolling.TimeBasedRollingPolicy">
    <!--
    滚动时产生的文件的存放位置及文件名称 %d{yyyy-MM-dd}：按天进行日志滚
动

    %i：当文件大小超过 maxFileSize 时，按照 i 进行文件滚动
    -->
    <fileNamePattern>${appName}-%d{yyyy-MM-dd}-%i.log</fileNamePattern>
    <!--
    可选节点，控制保留的归档文件的最大数量，超出数量就删除旧文件。假设设
置每天滚动，
    且 maxHistory 是 365，则只保存最近 365 天的文件，删除之前的旧文件。注意，
删除旧文件是，
    那些为了归档而创建的目录也会被删除。
    -->
    <MaxHistory>365</MaxHistory>
    <!--
    当日志文件超过 maxFileSize 指定的大小是，根据上面提到的%i 进行日志文件
滚动 注意此处配置 SizeBasedTriggeringPolicy 是无法实现按文件大小进行滚动的，必须配置
timeBasedFileNamingAndTriggeringPolicy
    -->
    <timeBasedFileNamingAndTriggeringPolicy
class="ch.qos.logback.core.rolling.SizeAndTimeBasedFNATP">
      <maxFileSize>100MB</maxFileSize>
    </timeBasedFileNamingAndTriggeringPolicy>
  </rollingPolicy>

```

```

<!--
    日志输出格式: %d 表示日期时间, %thread 表示线程名, %-5level: 级别从左显示
    5 个字符宽度 %logger{50} 表示 logger 名字最长 50 个字符, 否则按照句点分割。 %msg:
    日志消息, %n 是换行符
-->
<encoder>
    <pattern>%d{yyyy-MM-dd HH:mm:ss.SSS} - [ %thread ] - [ %-5level ]
    [ %logger{50} : %line ] - %msg%n</pattern>
</encoder>
</appender>

<!--
    logger 主要用于存放日志对象, 也可以定义日志类型、级别
    name: 表示匹配的 logger 类型前缀, 也就是包的前半部分
    level: 要记录的日志级别, 包括 TRACE < DEBUG < INFO < WARN < ERROR
    additivity: 作用在于 children-logger 是否使用 rootLogger 配置的 appender 进行输出,
    false: 表示只用当前 logger 的 appender-ref, true: 表示当前 logger 的 appender-ref 和 rootLogger
    的 appender-ref 都有效
-->
<!--
    <logger name="edu.hyh" level="info" additivity="true">
        <appender-ref ref="appLogAppender" />
    </logger> -->

<!--
    root 与 logger 是父子关系, 没有特别定义则默认为 root, 任何一个类只会和一个 logger
    对应,
    要么是定义的 logger, 要么是 root, 判断的关键在于找到这个 logger, 然后判断这个 logger
    的 appender 和 level。
-->

<logger name="org.springframework.beans.factory.support" level="info" additivity="true">

</logger>

<root level="debug">
    <appender-ref ref="stdout" />
    <appender-ref ref="appLogAppender" />
</root>
</configuration>

```

## 2.2.6. 类型转换器

在 mongodb 3.4 版本里面新增了个数据类型 Decimal128

但在前面操作的时候发现 User 里面的 salary 依然还是字符串

```
{
  "favorites" : {
    "movies" : [
      "东游记",
      "一路向东"
    ],
    "cites" : [
      "珠海",
      "东京"
    ]
  },
  "age" : 30,
  "salary" : "6885.22",
  "height" : 1.7699999809265137,
  "_class" : "cn.enjoy.entity.User"
}
```

这种情况需要使用到类型转换器

### 2.2.6.1. 新增 BigDecimalToDecimal128Converter

```
package cn.enjoy.convert;

import java.math.BigDecimal;

import org.bson.types.Decimal128;
import org.springframework.core.convert.converter.Converter;

public class BigDecimalToDecimal128Converter implements Converter<BigDecimal, Decimal128>
{
    @Override
    public Decimal128 convert(BigDecimal source) {
        return new Decimal128(source);
    }
}
```

### 2.2.6.2. 新增 Decimal128ToBigDecimalConverter

```
package cn.enjoy.convert;

import java.math.BigDecimal;

import org.bson.types.Decimal128;
import org.springframework.core.convert.converter.Converter;
```



```

public class Decimal128ToBigDecimalConverter implements Converter<Decimal128, BigDecimal>
{
    @Override
    public BigDecimal convert(Decimal128 source) {
        return source.bigDecimalValue();
    }
}

```

### 2.2.6.3. 修改 applicationContext.xml

```

<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:context="http://www.springframework.org/schema/context"
    xmlns:mongo="http://www.springframework.org/schema/data/mongo"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans.xsd
        http://www.springframework.org/schema/context
            http://www.springframework.org/schema/context/spring-context.xsd
        http://www.springframework.org/schema/data/mongo
            http://www.springframework.org/schema/data/mongo/spring-mongo.xsd">

    <context:component-scan base-package="cn.enjoy">

    </context:component-scan>

    <!-- mongodb 连接池配置 -->
    <mongo:mongo-client id="mongo" host="192.168.244.123" port="27017">
        <mongo:client-options
            write-concern="ACKNOWLEDGED"
            threads-allowed-to-block-for-connection-multiplier="5"
            max-wait-time="1200"
            connect-timeout="1000"/>
    </mongo:mongo-client>

    <!-- mongodb 数据库工厂配置 -->

```

```

<mongo:db-factory dbname="lison" mongo-ref="mongo" />

<mongo:mapping-converter base-package="cn.enjoy.convert">
  <mongo:custom-converters>
    <mongo:converter>
      <bean class="cn.enjoy.convert.BigDecimalToDecimal128Converter"/>
    </mongo:converter>
    <mongo:converter>
      <bean class="cn.enjoy.convert.Decimal128ToBigDecimalConverter"/>
    </mongo:converter>
  </mongo:custom-converters>
</mongo:mapping-converter>

<!-- mongodb 模板配置 -->
<bean
                                id="anotherMongoTemplate"
class="org.springframework.data.mongodb.core.MongoTemplate">
  <constructor-arg name="mongoDbFactory" ref="mongoDbFactory" />
  <constructor-arg name="mongoConverter" ref="mappingConverter"/>
</bean>
</beans>

```

#### 2.2.6.4. 测试

```

{
  "_id" : ObjectId("5dc65d12be776a41fbdb065f"),
  "username" : "chen",
  "country" : "china",
  "address" : {
    "aCode" : "411000",
    "add" : "我的地址2"
  },
  "favorites" : {
    "movies" : [
      "东游记",
      "一路向东"
    ],
    "cites" : [
      "珠海",
      "东京"
    ]
  },
  "age" : 30,
  "salary" : NumberDecimal("6885.22"),
  "length" : 1.7699999809265137,
  "_class" : "cn.enjoy.entity.User"
}

```

#### 2.2.7. 开发框架版本选择

## 2.2.8. java 驱动与 mongoDB 兼容性

<https://docs.mongodb.com/ecosystem/drivers/java/>

Java Driver Version	MongoDB 4.2	MongoDB 4.0	MongoDB 3.6	MongoDB 3.4	MongoDB 3.2	MongoDB 3.0	MongoDB 2.6
Version 3.11	✓	✓	✓	✓	✓	✓	✓
Version 3.10		✓	✓	✓	✓	✓	✓
Version 3.9		✓	✓	✓	✓	✓	✓
Version 3.8		✓	✓	✓	✓	✓	✓
Version 3.7			✓	✓	✓	✓	✓
Version 3.6			✓	✓	✓	✓	✓
Version 3.5				✓	✓	✓	✓
Version 3.4				✓	✓	✓	✓
Version 3.3					✓	✓	✓
Version 3.2					✓	✓	✓

可见 mongodb 具备强大的向下兼容性

## 2.2.9. java 驱动与 jdk 的兼容性

Java Driver Version	Java 5	Java 6	Java 7	Java 8	Java 11 (+)
Version 3.11		✓	✓	✓	✓
Version 3.10		✓	✓	✓	✓
Version 3.9		✓	✓	✓	✓
Version 3.8		✓	✓	✓	✓
Version 3.4		✓	✓	✓	✓
Version 3.3		✓	✓	✓	✓
Version 3.2		✓	✓	✓	✓
Version 3.1		✓	✓	✓	✓
Version 3.0		✓	✓	✓	✓

## 2.2.10. spring data mongo 与 java mongo 驱动兼容性

spring mongodb 版本	spring版本支持	jdk版本支持	mongodb server支持	默认的mongodb java驱动版本
Spring Data MongoDB 1.x	4.3.13.RELEASE 以上	jdk 1.6以上	2.6版本以上, 3.4以下	2.14.3
Spring Data MongoDB 2.x	5.0.2.RELEASE 以上	jdk 1.8以上	2.6版本以上, 3.6	3.5.0

## 2.2.11. mongoDB 数据类型

数据类型	示例	说明
null	{"key":null}	null表示空值或者不存在该字段
布尔	{"key","true"} {"key","false"}	布尔类型表示真或者假
32位整数	{"key":8}	存储32位整数, 但在shell界面显示会被自动转成64位浮点数
64位整数	{"key":{"floatApprox":8}}	存储64位整数, floatApprox意思是使用64位浮点数近似表示一个64位整数
64位浮点数	{"key":8.21}	存储64位整数, shell客户端显示的数字都是这种类型:
字符串	{"key":"value"} {"key":"8"}	UTF-8格式
对象ID	{"key":ObjectId()}	12字节的唯一ID
日期	{"key":new Date()}	
代码	{"key":function() {}}	
二进制数据		主要存储文件
未定义	{"key":undefined}	值没有定义, null和undefined是不同的
数组	{"key":[16,15,17]}	集合或者列表
内嵌文档	{"user":{"name":"lison"}}	子对象
Decimal128	{"price":NumberDecimal("2.099")}	3.4版本新增的数据类型, 无精度问题

## 2.3. 查询

```
db.users.drop();
var user1 = {
    "username" : "lison",
    "country" : "china",
    "address" : {
        "aCode" : "411000",
        "add" : "长沙"
    },
    "favorites" : {
        "movies" : ["杀破狼 2","战狼","雷神 1"],
        "cites" : ["长沙","深圳","上海"]
    },
}
```

```
        "age" : 18,
        "salary":NumberDecimal("18889.09"),
        "lenght" :1.79
    };

    var user2 = {
        "username" : "james",
        "country" : "English",
        "address" : {
            "aCode" : "311000",
            "add" : "地址"
        },
        "favorites" : {
            "movies" : ["复仇者联盟","战狼","雷神 1"],
            "cites" : ["西安","东京","上海"]
        },
        "age" : 24,
        "salary":NumberDecimal("7889.09"),
        "lenght" :1.35
    };

    var user3 = {
        "username" : "deer",
        "country" : "japan",
        "address" : {
            "aCode" : "411000",
            "add" : "长沙"
        },
        "favorites" : {
            "movies" : ["肉蒲团","一路向西","倩女幽魂"],
            "cites" : ["东莞","深圳","东京"]
        },
        "age" : 22,
        "salary":NumberDecimal("6666.66"),
        "lenght" :1.85
    };

    var user4 =
    {
        "username" : "mark",
        "country" : "USA",
        "address" : {
            "aCode" : "411000",
            "add" : "长沙"
        },
        "favorites" : {
```

```

        "movies": ["蜘蛛侠", "钢铁侠", "蝙蝠侠"],
        "cites": ["青岛", "东莞", "上海"]
    },
    "age": 20,
    "salary": NumberDecimal("6398.22"),
    "lenght": 1.77
};

var user5 =
{
    "username": "peter",
    "country": "UK",
    "address": {
        "aCode": "411000",
        "add": "TEST"
    },
    "favorites": {
        "movies": ["蜘蛛侠", "钢铁侠", "蝙蝠侠"],
        "cites": ["青岛", "东莞", "上海"]
    },
    "salary": NumberDecimal("1969.88")
};

db.users.insert(user1);
db.users.insert(user2);
db.users.insert(user3);
db.users.insert(user4);
db.users.insert(user5);

```

### 2.3.1. 查询概要

MongoDB 查询数据的语法格式如下：

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

- **query**：可选，使用查询操作符指定查询条件
- **projection**：可选，使用投影操作符指定返回的键。查询时返回文档中所有键值， 只需省略

该参数即可（默认省略）。

注意：0 表示字段排除，非 0 表示字段选择并排除其他字段，所有字段必须设置同样的值；

- 需要以易读的方式来读取数据，可以使用 `pretty()` 方法；

举例子：`db.users.find({"$and":[{"username":"lison"}, {"age":18}], {"username":0, "age":0})`

### 2.3.2. 查询选择器

运算符类型	运算符	描述
范围	\$eq	等于
	\$lt	小于
	\$gt	大于
	\$lte	小于等于
	\$gte	大于等于
	\$in	判断元素是否在指定的集合范围里
	\$all	判断数组中是否包含某几个元素,无关顺序
布尔运算	\$nin	判断元素是否不在指定的集合范围里
	\$ne	不等于, 不匹配参数条件
	\$not	不匹配结果
	\$or	有一个条件成立则匹配
	\$nor	所有条件都不匹配
	\$and	所有条件都必须匹配
其他	\$exists	判断元素是否存在
	.	子文档匹配
	\$regex	正则表达式匹配

### 2.3.3. 查询选择器实战

(1)client 指定端口和 ip 连接 mongodb  
./mongo localhost:27022

(2)in 选择器示例:

```
db.users.find({"username":{"$in":["lison", "mark", "james"]}}).pretty()
```

查询姓名为 lison、mark 和 james 这个范围的人

(3)exists 选择器示例:

```
db.users.find({"lenght":{"$exists":true}}).pretty()
```

判断文档有没有关心的字段

(4)not 选择器示例:

```
db.users.find({"lenght":{"$not":{"$gte":1.77}}}).pretty()
```

查询高度小于 1.77 或者没有身高的人

not 语句 会把不包含查询语句字段的文档 也检索出来

```
db.users.find({"lenght":{"$lt":1.77}}).pretty()
```

```
db.users.find({"$or":[{"lenght":{"$lt":1.77}},{"lenght":{"$exists":false}]}).pretty()
```

### 2.3.4. 查询选择

- 映射

字段选择并排除其他字段: `db.users.find({},{'username':1})`

`db.users.find({},{'username':1,'age':1})`

字段排除: `db.users.find({},{'username':0})`

- 排序

`sort()`: `db.users.find().sort({'username':1}).pretty()`

1: 升序    -1: 降序

- 跳过和限制

`skip(n)`: 跳过 n 条数据

`limit(n)`: 限制 n 条数据

e.g: `db.users.find().sort({'username':1}).limit(2).skip(2)`

- 查询唯一值

`distinct()`: 查询指定字段的唯一值, e.g: `db.users.distinct("username")`

## 2.3.5. 字符串数组选择查询

- 数组单元素查询

`db.users.find({'favorites.movies':"蜘蛛侠"})`

查询数组中包含"蜘蛛侠"

- 数组精确查找

`db.users.find({'favorites.movies':['杀破狼 2','战狼','雷神 1']},{'favorites.movies':1})`

查询数组等于["杀破狼 2", "战狼", "雷神 1"]的文档, 严格按照数量、顺序;

- 数组多元素查询

`db.users.find({'favorites.movies':{'$all':['雷神 1','战狼']}},{'favorites.movies':1})`

查询数组包含["雷神 1", "战狼"]的文档, 跟顺序无关, 跟数量有关

`db.users.find({'favorites.movies':{'$in':['雷神 1','战狼']}},{'favorites.movies':1})`

查询数组包含["雷神 1", "战狼"]中任意一个的文档, 跟顺序无关, 跟数量无关

- 索引查询

`db.users.find({'favorites.movies.0':"杀破狼 2"},{'favorites.movies':1})`

查询数组中第一个为"杀破狼 2"的文档

- 返回数组子集

`db.users.find({},{'favorites.movies':{'$slice':[1,2]},'favorites':1})`

`$slice` 可以取两个元素数组, 分别表示跳过和限制的条数;

对比 `db.users.find({},{'favorites':1})`



### 2.3.6. 对象数组选择查询

```
db.users.drop();
var user1 = {
  "username" : "lison",
  "country" : "china",
  "address" : {
    "aCode" : "411000",
    "add" : "长沙"
  },
  "favorites" : {
    "movies" : ["妇联 4","杀破狼 2","战狼","雷神 1","神奇动物在哪里"],
    "cites" : ["长沙","深圳","上海"]
  },
  "age" : 18,
  "salary":NumberDecimal("18889.09"),
  "lenght" :1.79,
  "comments" : [
    {
      "author" : "lison1",
      "content" : "lison 评论 1",
      "commentTime" : ISODate("2017-01-06T00:00:00")
    },
    {
      "author" : "lison2",
      "content" : "lison 评论 2",
      "commentTime" : ISODate("2017-02-06T00:00:00")
    },
    {
      "author" : "lison3",
      "content" : "lison 评论 3",
      "commentTime" : ISODate("2017-03-06T00:00:00")
    },
    {
      "author" : "lison4",
      "content" : "lison 评论 4",
      "commentTime" : ISODate("2017-04-06T00:00:00")
    },
    {
      "author" : "lison5",
      "content" : "lison 是苍老师的小迷弟",
      "commentTime" : ISODate("2017-05-06T00:00:00")
    }
  ],
}
```

```

        {
            "author" : "lison6",
            "content" : "lison 评论 6",
            "commentTime" : ISODate("2017-06-06T00:00:00")
        },
        {
            "author" : "lison7",
            "content" : "lison 评论 7",
            "commentTime" : ISODate("2017-07-06T00:00:00")
        },
        {
            "author" : "lison8",
            "content" : "lison 评论 8",
            "commentTime" : ISODate("2017-08-06T00:00:00")
        },
        {
            "author" : "lison9",
            "content" : "lison 评论 9",
            "commentTime" : ISODate("2017-09-06T00:00:00")
        }
    ]
};

var user2 = {
    "username" : "james",
    "country" : "English",
    "address" : {
        "aCode" : "311000",
        "add" : "地址"
    },
    "favorites" : {
        "movies" : ["复仇者联盟","战狼","雷神 1"],
        "cites" : ["西安","东京","上海"]
    },
    "age" : 24,
    "salary":NumberDecimal("7889.09"),
    "lenght" :1.35,
    "comments" : [
        {
            "author" : "lison1",
            "content" : "lison 评论 1",
            "commentTime" : ISODate("2017-10-06T00:00:00")
        },
        {

```

```

        "author" : "lison6",
        "content" : "lison 评论 6",
        "commentTime" : ISODate("2017-11-06T05:26:18")
    },
    {
        "author" : "lison12",
        "content" : "lison 评论 12",
        "commentTime" : ISODate("2017-11-06T00:00:00")
    }
]
};
var user3={
    "username": "deer",
    "country": "japan",
    "address": {
        "aCode" : "411000",
        "add" : "长沙"
    },
    "favorites" : {
        "movies" : ["肉蒲团", "一路向西", "倩女幽魂"],
        "cites" : ["东莞", "深圳", "东京"]
    },
    "age" : 22,
    "salary": NumberDecimal("6666.66"),
    "lenght" : 1.85,
    "comments" : [
        {
            "author" : "lison1",
            "content" : "lison 评论 1",
            "commentTime" : ISODate("2017-10-06T00:00:00")
        },
        {
            "author" : "lison22",
            "content" : "lison 评论 6",
            "commentTime" : ISODate("2017-11-06T00:00:00")
        },
        {
            "author" : "lison16",
            "content" : "lison 评论 12",
            "commentTime" : ISODate("2017-11-06T00:00:00")
        }
    ]
};
var user4 =

```

```

{
  "username" : "mark",
  "country" : "USA",
  "address" : {
    "aCode" : "411000",
    "add" : "长沙"
  },
  "favorites" : {
    "movies" : ["蜘蛛侠","钢铁侠","蝙蝠侠"],
    "cites" : ["青岛","东莞","上海"]
  },
  "age" : 20,
  "salary":NumberDecimal("6398.22"),
  "lenght" :1.77
};

var user5 =
{
  "username" : "peter",
  "country" : "UK",
  "address" : {
    "aCode" : "411000",
    "add" : "TEST"
  },
  "favorites" : {
    "movies" : ["蜘蛛侠","钢铁侠","蝙蝠侠"],
    "cites" : ["青岛","东莞","上海"]
  },
  "salary":NumberDecimal("1969.88")
};

db.users.insert(user1);
db.users.insert(user2);
db.users.insert(user3);
db.users.insert(user4);
db.users.insert(user5);

```

- 单元素查询

```

db.users.find({"comments":{"
    "author" : "lison6",
    "content" : "lison 评论 6","commentTime" :
ISODate("2017-06-06T00:00:00Z")}}})

```

备注：对象数组精确查找

- 查找 lison1 或者 lison12 评论过的 user （\$in 查找符）

```
db.users.find({"comments.author":{"$in":["lison1","lison12"]}}).pretty()
```

备注：跟数量无关，跟顺序无关；

- 查找 lison1 和 lison12 都评论过的 user

```
db.users.find({"comments.author":{"$all":["lison12","lison1"]}}).pretty()
```

备注：跟数量有关，跟顺序无关；

- 查找 lison5 评语为包含"苍老师"关键字的 user （\$elemMatch 查找符）

```
db.users.find({"comments":{"$elemMatch":{"author": "lison5",  
                                           "content" :  
                                           { "$regex" : ".*苍老师.*" }}}) .pretty()
```

备注：数组中对象数据要符合查询对象里面所有的字段，\$全元素匹配，和顺序无关；

## 2.3.7. Java 客户端解析

### 2.3.7.1. 原生客户端

- MongoClient → MongoDBDatabase → MongoClient

- ✓ MongoClient 被设计成线程安全、可以被多线程共享的。通常访问数据库集群的应用只需要一个实例
- ✓ 如果需要使用 pojo 对象读写，需要将 PojoCodecProvider 注入到 client 中

- 查询和更新的 API 类

- ✓ 查询器：com.mongodb.client.model.Filters
- ✓ 更新器：com.mongodb.client.model.Updates
- ✓ 投影器：com.mongodb.client.model.Projections

```
package cn.enjoy.mg;  
  
import static com.mongodb.client.model.Filters.*;  
import static com.mongodb.client.model.Projections.*;  
import static com.mongodb.client.model.Sorts.*;  
import static com.mongodb.client.model.Aggregates.*;  
  
import java.text.ParseException;  
import java.text.SimpleDateFormat;  
import java.time.LocalDateTime;  
import java.time.ZoneId;
```

```
import java.time.ZonedDateTime;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.Date;
import java.util.List;

import javax.annotation.Resource;

import org.bson.BSON;
import org.bson.BsonDocument;
import org.bson.Document;
import org.bson.codecs.BsonTypeClassMap;
import org.bson.codecs.DocumentCodec;
import org.bson.codecs.configuration.CodecRegistries;
import org.bson.codecs.configuration.CodecRegistry;
import org.bson.codecs.pojo.PojoCodecProvider;
import org.bson.conversions.Bson;
import org.junit.Before;
import org.junit.Test;
import org.junit.runner.RunWith;
import org.junit.runner.manipulation.Filter;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.data.mongodb.core.MongoOperations;
import org.springframework.test.context.ContextConfiguration;
import org.springframework.test.context.junit4.SpringJUnit4ClassRunner;

import com.mongodb.Block;
import com.mongodb.MongoClient;
import com.mongodb.MongoClientOptions;
import com.mongodb.ServerAddress;
import com.mongodb.WriteConcern;
import com.mongodb.client.AggregateIterable;
import com.mongodb.client.FindIterable;
import com.mongodb.client.MongoCollection;
import com.mongodb.client.MongoDatabase;
import com.mongodb.client.model.Accumulators;
import com.mongodb.client.model.Filters;
import com.mongodb.client.model.Projections;
import com.mongodb.client.model.PushOptions;
import com.mongodb.client.model.Updates;
import com.mongodb.client.result.UpdateResult;
import com.mongodb.operation.OrderBy;
```

```

@RunWith(SpringJUnit4ClassRunner.class)
@ContextConfiguration("classpath:applicationContext.xml")
public class JavaQueryTest {

    private static final Logger logger = LoggerFactory
        .getLogger(JavaQueryTest.class);

    private MongoDBDatabase db;

    private MongoCollection<Document> collection;

    private MongoCollection<Document> orderCollection;

    @Resource(name="mongo")
    private MongoClient client;

    @Before
    public void init() {
        db = client.getDatabase("lison");
        collection = db.getCollection("users");
        orderCollection = db.getCollection("ordersTest");
    }

    // -----操作符使用实例-----

    // db.users.find({"username":{"$in":["lison", "mark", "james"]}}).pretty()
    // 查询姓名为 lison、mark 和 james 这个范围的人
    @Test
    public void testInOper() {
        Bson in = in("username", "lison", "mark", "james");
        FindIterable<Document> find = collection.find(in);
        printOperation(find);
    }

    // db.users.find({"lenght":{"$exists":true}}).pretty()
    // 判断文档有没有关心的字段
    @Test
    public void testExistsOper() {
        Bson exists = exists("lenght", true);
        FindIterable<Document> find = collection.find(exists);
        printOperation(find);
    }

    // db.users.find().sort({"username":1}).limit(1).skip(2)

```

```

// 测试 sort, limit, skip
@Test
public void testSLSOper() {
    Document sort = new Document("username", 1);
    FindIterable<Document> find = collection.find().sort(sort).limit(1).skip(2);
    printOperation(find);
}

// db.users.find({"lenght":{"$not":{"$gte":1.77}}}).pretty()
// 查询高度小于 1.77 或者没有身高的人
// not 语句 会把不包含查询语句字段的文档 也检索出来

@Test
public void testNotOper() {
    Bson gte = gte("lenght", 1.77);
    Bson not = not(gte);
    FindIterable<Document> find = collection.find(not);
    printOperation(find);
}

// -----字符串数组查询实例-----

// db.users.find({"favorites.movies":"蜘蛛侠"})
// 查询数组中包含"蜘蛛侠"
@Test
public void testArray1() {
    Bson eq = eq("favorites.movies", "蜘蛛侠");
    FindIterable<Document> find = collection.find(eq);
    printOperation(find);
}

// db.users.find({"favorites.movies":["妇联 4","杀破狼 2","战狼","雷神 1","神奇动物在哪里"]}, {"favorites.movies":1})
// 查询数组等于[“杀破狼 2”,“战狼”,“雷神 1”]的文档，严格按照数量、顺序；

@Test
public void testArray2() {
    Bson eq = eq("favorites.movies", Arrays.asList("妇联 4","杀破狼 2","战狼","雷神 1","神奇动物在哪里"));
    FindIterable<Document> find = collection.find(eq);
    printOperation(find);
}

```



```

//数组多元素查询
@Test
public void testArray3() {

    // db.users.find({"favorites.movies":{"$all":["雷神 1", "战狼"]}},{"favorites.movies":1})
    // 查询数组包含["雷神 1", "战狼"]的文档，跟顺序无关
    Bson all = all("favorites.movies", Arrays.asList("雷神 1", "战狼"));
    FindIterable<Document> find = collection.find(all);
    printOperation(find);
// db.users.find({"favorites.movies":{"$in":["雷神 1", "战狼"]}},{"favorites.movies":1})
// 查询数组包含["雷神 1", "战狼"]中任意一个的文档，跟顺序无关，跟数量无关
    Bson in = in("favorites.movies", Arrays.asList("雷神 1", "战狼"));
    find = collection.find(in);
    printOperation(find);
}

// // db.users.find({"favorites.movies.0":"妇联 4"},"favorites.movies":1})
// 查询数组中第一个为"妇联 4"的文档

@Test
public void testArray4() {
    Bson eq = eq("favorites.movies.0", "妇联 4");
    FindIterable<Document> find = collection.find(eq);
    printOperation(find);
}

// db.users.find({}, {"favorites.movies":{"$slice":[1,2]}, "favorites":1})
// $slice 可以取两个元素数组,分别表示跳过和限制的条数;

@Test
public void testArray5() {
    Bson slice = slice("favorites.movies", 1, 2);
    Bson include = include("favorites");
    Bson projection = fields(slice, include);
    FindIterable<Document> find = collection.find().projection(projection);
    printOperation(find);
}

// -----对象数组查询实例-----

//db.users.find({"comments":{"author":"lison6","content":"lison
6","commentTime":ISODate("2017-06-06T00:00:00Z")}})
//备注：对象数组精确查找

```

```

@Test
public void testObjArray1() throws ParseException {
    SimpleDateFormat formatter = new SimpleDateFormat("yyyy-MM-dd hh:mm:ss");
    Date commentDate = formatter.parse("2017-06-06 08:00:00");

    Document comment = new Document().append("author", "lison6")
        .append("content", "lison 评论 6")
        .append("commentTime", commentDate);

    Bson eq = eq("comments", comment);
    FindIterable<Document> find = collection.find(eq);
    printOperation(find); }

//数组多元素查询
@Test
public void testObjArray2() {

//    查找 lison1 或者 lison12 评论过的 user （$in 查找符）
//    db.users.find({"comments.author":{"$in":["lison1","lison12"]}}).pretty()
//    备注：跟数量无关，跟顺序无关；

    Bson in = in("comments.author", Arrays.asList("lison1","lison12"));
    FindIterable<Document> find = collection.find(in);
    printOperation(find);

//    查找 lison1 和 lison12 都评论过的 user
//    db.users.find({"comments.author":{"$all":["lison12","lison1"]}}).pretty()
//    备注：跟数量有关，跟顺序无关；

    Bson all = all("comments.author", Arrays.asList("lison12","lison1"));
    find = collection.find(all);
    printOperation(find);
}

//查找 lison5 评语为包含“苍老师”关键字的 user （$elemMatch 查找符）
//    db.users.find({"comments":{"$elemMatch":{"author" : "lison5", "content" : { "$regex" : ".*苍老师.*"}}}})
//备注：数组中对象数据要符合查询对象里面所有的字段，$全元素匹配，和顺序无关；

@Test
public void testObjArray3() throws ParseException {

```

```

        Bson eq = eq("author","lison5");
        Bson regex = regex("content", ".*苍老师.*");
        Bson elemMatch = Filters.elemMatch("comments", and(eq,regex));
        FindIterable<Document> find = collection.find(elemMatch);
        printOperation(find);
    }

    // dbRef 测试
    // dbref 其实就是关联关系的信息载体，本身并不会去关联数据
    @Test
    public void dbRefTest() {
        FindIterable<Document> find = collection.find(eq("username", "lison"));
        printOperation(find);
    }

    private Block<Document> getBlock(final List<Document> ret) {
        Block<Document> printBlock = new Block<Document>() {
            @Override
            public void apply(Document t) {
                System.out.println("-----");
                CodecRegistry codecRegistry =
                    CodecRegistries.fromRegistries(MongoClient.getDefaultCodecRegistry());
                final DocumentCodec codec = new DocumentCodec(codecRegistry, new
                    BsonTypeClassMap());
                System.out.println(t.toJson(codec));
                System.out.println("-----");
                ret.add(t);
            }
        };
        return printBlock;
    }

    //打印查询出来的数据和查询的数据量
    private void printOperation( FindIterable<Document> find) {
        final List<Document> ret = new ArrayList<Document>();
        Block<Document> printBlock = getBlock(ret);
        find.forEach(printBlock);
        System.out.println(ret.size());
        ret.removeAll(ret);
    }

```

```
private void printOperation(List<Document> ret, Block<Document> printBlock,
    AggregateIterable<Document> aggregate) {
    aggregate.forEach(printBlock);
    System.out.println(ret.size());
    ret.removeAll(ret);
}
```

@Test

// 测试 elemMatch 操作符，数组中对象数据要符合查询对象里面所有的字段

// 查找 lison5 评语为“lison 是苍老师的小迷弟”的人

// db.users.find({"comments":{"\$elemMatch":{"author" : "lison5","content" :

// "lison 是苍老师的小迷弟"}}}) .pretty()

```
public void testElemMatch() {
```

```

        Document filter = new Document().append("author", "lison5").append(
            "content", "lison 是苍老师的小迷弟");
        Bson elemMatch = Filters.elemMatch("comments", filter);

        FindIterable<Document> find = collection.find(elemMatch);

        printOperation(find);

    }

    /**
     * db.users.updateOne({"username":"lison"}, {"$push": { "comments": {
     * $each: [{ "author": "james", "content": "lison 是个好老师! ", "commentTime":
     * ISODate("2018-01-06T04:26:18.354Z") } ], $sort: {"commentTime":-1} }}});
     */
    @Test
    // 新增评论时，使用$sort 运算符进行排序，插入评论后，再按照评论时间降序排序
    public void demoStep1() {
        Bson filter = eq("username", "lison");
        Document comment = new Document().append("author", "cang")
            .append("content", "lison 是我的粉丝")
            .append("commentTime", new Date());

        // $sort: {"commentTime":-1}
        Document sortDoc = new Document().append("commentTime", -1);
        PushOptions sortDocument = new PushOptions().sortDocument(sortDoc);
        // $each
        Bson pushEach = Updates.pushEach("comments", Arrays.asList(comment),
            sortDocument);

        UpdateResult updateOne = collection.updateOne(filter, pushEach);
        System.out.println(updateOne.getModifiedCount());
    }

    @Test
    // 查看人员时加载最新的三条评论；
    // db.users.find({"username":"lison"}, {"comments":{"$slice":[0,3]}}).pretty()
    public void demoStep2() {
        FindIterable<Document> find = collection.find(eq("username", "lison"))
            .projection(slice("comments", 0, 3));

        printOperation(find);
    }

    @Test
    // 点击评论的下一页按钮，新加载三条评论

```

```

// db.users.find({"username":"lison"},{"comments":{"$slice":[3,3]},"$id":1}).pretty();
public void demoStep3() {
    // {"username":"lison"}
    Bson filter = eq("username", "lison");
    // "$slice":[3,3]
    Bson slice = slice("comments", 3, 3);
    // "$id":1
    Bson includeID = include("id");

    // {"comments":{"$slice":[3,3]},"$id":1}
    Bson projection = fields(slice, includeID);

    FindIterable<Document> find = collection.find(filter).projection(
        projection);
    printOperation(find);
}

@Test
/**
 * db.users.aggregate([{"$match":{"username":"lison"},
                        {"$unwind":"$comments"},
                        {"$sort":{"comments.commentTime":-1}},
                        {"$project":{"comments":1}},
                        {"$skip":6},
                        {"$limit":3}}]
 */
// 如果有多种排序需求怎么处理,使用聚合
public void demoStep4() {
    final List<Document> ret = new ArrayList<Document>();
    Block<Document> printBlock = getBlock(ret);
    List<Bson> aggregates = new ArrayList<Bson>();

    aggregates.add(match(eq("username", "lison")));
    aggregates.add(unwind("$comments"));
    aggregates.add(sort(orderBy(ascending("comments.commentTime"))));
    aggregates.add(project(fields(include("comments"))));
    aggregates.add(skip(0));
    aggregates.add(limit(3));

    AggregateIterable<Document> aggregate = collection
        .aggregate(aggregates);

    printOperation(ret, printBlock, aggregate);
}

```

```

@Test
public void aggregationTest() {
    Block<Document> printBlock = new Block<Document>() {
        @Override
        public void apply(Document t) {
            logger.info("-----");
            System.out.println(t.toJson());
            logger.info("-----");
        }
    };

    // 定义数据的处理类
    // final List<Document> ret = new ArrayList<Document>();
    // //
    // Document filter = new Document().append("useCode", "tony");
    //
    // FindIterable<Document> find = orderCollection.find(filter);
    //
    // printOperation(ret, printBlock, find);

    // db.ordersTest.aggregate([{"$group":{_id:"$useCode",count: { $sum:
    // "$price" } } }])

    List<Bson> aggregates = new ArrayList<Bson>();
    aggregates.add(group("$useCode", Accumulators.sum("sum", "$price")));
    AggregateIterable<Document> aggregate = orderCollection
        .aggregate(aggregates);
    aggregate.forEach(printBlock);
}
}

```

### 2.3.7.2. Spring mongodb 解析

#### 2.3.7.2.1. 修改 User 实体类

```

package cn.enjoy.entity;

import java.math.BigDecimal;

```

```
import java.util.List;

import org.bson.types.ObjectId;
import org.springframework.data.mongodb.core.mapping.DBRef;
import org.springframework.data.mongodb.core.mapping.Document;

@Document(collection="users")
public class User {

    private ObjectId id;

    private String username;

    private String country;

    private Address address;

    private Favorites favorites;

    private int age;

    private BigDecimal salary;

    private float lenght;

    private List<Comment> comments;

    public String getUsername() {
        return username;
    }
    public void setUsername(String username) {
        this.username = username;
    }
    public String getCountry() {
        return country;
    }
    public void setCountry(String country) {
        this.country = country;
    }
    public Address getAddress() {
        return address;
    }
    public void setAddress(Address address) {
```



```
        this.address = address;
    }
    public Favorites getFavorites() {
        return favorites;
    }
    public void setFavorites(Favorites favorites) {
        this.favorites = favorites;
    }
    public Objectid getId() {
        return id;
    }
    public void setId(Objectid id) {
        this.id = id;
    }
    public int getAge() {
        return age;
    }
    public void setAge(int age) {
        this.age = age;
    }
    public BigDecimal getSalary() {
        return salary;
    }
    public void setSalary(BigDecimal salary) {
        this.salary = salary;
    }
    public float getLenght() {
        return lenght;
    }
    public void setLenght(float lenght) {
        this.lenght = lenght;
    }
}

public List<Comment> getComments() {
    return comments;
}
public void setComments(List<Comment> comments) {
    this.comments = comments;
}
@Override
public String toString() {
    return "User [id=" + id + ", username=" + username + ", country="
```

```
        + country + ", address=" + address + ", favorites=" + favorites  
        + ", age=" + age + ", salary=" + salary + ", lenght=" + lenght  
        + ", comments=" + comments + "];  
    }  
}
```

#### 2.3.7.2.2. 新增 **Comment** 实体类

```
package cn.enjoy.entity;  
  
import java.util.Date;  
import org.springframework.data.mongodb.core.mapping.Document;  
  
public class Comment {  
    private String author;  
  
    private String content;  
  
    private Date commentTime;  
  
    public String getAuthor() {  
        return author;  
    }  
  
    public void setAuthor(String author) {  
        this.author = author;  
    }  
  
    public Date getCommentTime() {  
        return commentTime;  
    }  
  
    public void setCommentTime(Date commentTime) {  
        this.commentTime = commentTime;  
    }  
  
    public String getContent() {  
        return content;  
    }  
}
```

```
    public void setContent(String content) {  
        this.content = content;  
    }  
  
    @Override  
    public String toString() {  
        return "Comment [author=\"" + author + "\", commentTime=\"" + commentTime  
            + "\", content=\"" + content + "\"]";  
    }  
}
```

### 2.3.7.2.3. Spring 查询测试类

```
package cn.enjoy.mg;  
  
import static org.springframework.data.mongodb.core.aggregation.Aggregation.*;  
import static org.springframework.data.mongodb.core.query.Criteria.*;  
import static org.springframework.data.mongodb.core.query.Query.*;  
  
import java.text.ParseException;  
import java.text.SimpleDateFormat;  
import java.util.ArrayList;  
import java.util.Arrays;  
import java.util.Date;  
import java.util.List;  
  
import javax.annotation.Resource;  
  
import cn.enjoy.entity.Comment;  
import cn.enjoy.entity.User;  
import com.mongodb.client.result.UpdateResult;  
import org.bson.Document;  
import org.bson.conversions.Bson;  
import org.junit.Test;  
import org.junit.runner.RunWith;  
import org.slf4j.Logger;  
import org.slf4j.LoggerFactory;  
import org.springframework.data.domain.Sort;  
import org.springframework.data.domain.Sort.Direction;
```

```

import org.springframework.data.mongodb.core.MongoOperations;
import org.springframework.data.mongodb.core.aggregation.Aggregation;
import org.springframework.data.mongodb.core.aggregation.AggregationResults;
import org.springframework.data.mongodb.core.query.Criteria;
import org.springframework.data.mongodb.core.query.Query;
import org.springframework.data.mongodb.core.query.Update;
import org.springframework.data.mongodb.core.query.Update.PushOperatorBuilder;
import org.springframework.test.context.ContextConfiguration;
import org.springframework.test.context.junit4.SpringJUnit4ClassRunner;


import com.mongodb.Block;
import com.mongodb.WriteResult;
import com.mongodb.client.FindIterable;
import com.mongodb.client.model.Filters;

@RunWith(SpringJUnit4ClassRunner.class)
@ContextConfiguration("classpath:applicationContext.xml")
public class SpringQueryTest {

    private static final Logger logger = LoggerFactory
        .getLogger(SpringQueryTest.class);

    @Resource
    private MongoOperations templete;

    // -----操作符使用实例-----

    // db.users.find({"username":{"$in":["lison", "mark", "james"]}}).pretty()
    // 查询姓名为 lison、mark 和 james 这个范围的人
    @Test
    public void testInOper() {
        Query query = query(where("username").in("lison", "mark", "james"));
        List<User> find = templete.find(query, User.class);
        printUsers(find);

    }

    // db.users.find({"lenght":{"$exists":true}}).pretty()

```

```

// 判断文档有没有关心的字段
@Test
public void testExistsOper() {

    Query query = query(where("lenght").exists(true));
    List<User> find = tempelate.find(query, User.class);
    printUsers(find);

}

// db.users.find().sort({"username":1}).limit(1).skip(2)
// 测试 sort, limit, skip
@Test
public void testSLSOper() {

    //Query query = query(where(null)).with(new Sort(new Sort.Order(Direction.ASC,
"username"))).limit(1).skip(2);
    Query query = query(where(null)).with(Sort.by(Direction.ASC,
"username")).limit(1).skip(2);
    List<User> find = tempelate.find(query, User.class);
    printUsers(find);

}

// db.users.find({"lenght":{"$not":{"$gte":1.77}}}).pretty()
// 查询高度小于 1.77 或者没有身高的人
// not 语句 会把不包含查询语句字段的文档 也检索出来

@Test
public void testNotOper() {
    Query query = query(where("lenght").not().gte(1.77));
    List<User> find = tempelate.find(query, User.class);
    printUsers(find);

}

// -----字符串数组查询实例-----

// db.users.find({"favorites.movies":"蜘蛛侠"})
// 查询数组中包含"蜘蛛侠"
@Test
public void testArray1() {
    Query query = query(where("favorites.movies").is("蜘蛛侠"));

```

```

        List<User> find = tempelate.find(query, User.class);
        printUsers(find);
    }

    // db.users.find({"favorites.movies":["妇联 4","杀破狼 2","战狼","雷神 1","神奇动物在哪里"],{"favorites.movies":1})
    // 查询数组等于[“杀破狼 2”，“战狼”，“雷神 1”]的文档，严格按照数量、顺序；

    @Test
    public void testArray2() {
        Query query = query(where("favorites.movies").is(Arrays.asList("妇联 4","杀破狼 2","战狼","雷神 1","神奇动物在哪里")));
        List<User> find = tempelate.find(query, User.class);
        printUsers(find);
    }

    //数组多元素查询
    @Test
    public void testArray3() {
        // db.users.find({"favorites.movies":{"$all":["雷神 1","战狼"]}},{"favorites.movies":1})
        // 查询数组包含[“雷神 1”，“战狼”]的文档，跟顺序无关

        Query query = query(where("favorites.movies").all(Arrays.asList("雷神 1","战狼")));
        List<User> find = tempelate.find(query, User.class);
        printUsers(find);

        // db.users.find({"favorites.movies":{"$in":["雷神 1","战狼"]}},{"favorites.movies":1})
        // 查询数组包含[“雷神 1”，“战狼”]中任意一个的文档，跟顺序无关，跟数量无关
        query = query(where("favorites.movies").in(Arrays.asList("雷神 1","战狼")));
        find = tempelate.find(query, User.class);
        printUsers(find);
    }

    // // db.users.find({"favorites.movies.0":"妇联 4"},{"favorites.movies":1})
    // 查询数组中第一个为“妇联 4”的文档

    @Test
    public void testArray4() {
        Query query = query(where("favorites.movies.0").is("妇联 4"));
        List<User> find = tempelate.find(query, User.class);
        printUsers(find);
    }

```

```
// db.users.find({},{"favorites.movies":{"$slice":[1,2]},"favorites":1})
```

```
// $slice 可以取两个元素数组,分别表示跳过和限制的条数;
```

```
@Test
```

```
public void testArray5() {
```

```
    Query query = query(where(null));
```

```
    query.fields().include("favorites").slice("favorites.movies", 1, 2);
```

```
    List<User> find = tempelate.find(query, User.class);
```

```
    printUsers(find);
```

```
}
```

```
// -----对象数组查询实例-----
```

```
//db.users.find({"comments":{"author":"lison6","content":"lison  
6","commentTime":ISODate("2017-06-06T00:00:00Z")}})
```

```
//备注：对象数组精确查找
```

```
//坑：居然和属性定义的顺序有关
```

```
@Test
```

```
public void testObjArray1() throws ParseException {
```

```
    SimpleDateFormat formatter = new SimpleDateFormat("yyyy-MM-dd hh:mm:ss");
```

```
    Date commentDate = formatter.parse("2017-06-06 08:00:00");
```

```
    Comment comment = new Comment();
```

```
    comment.setAuthor("lison6");
```

```
    comment.setCommentTime(commentDate);
```

```
    comment.setContent("lison 评论 6");
```

```
    Query query = query(where("comments").is(comment));
```

```
    List<User> find = tempelate.find(query, User.class);
```

```
    printUsers(find);
```

```
}
```

```
//数组多元素查询
```

```
@Test
```

```
public void testObjArray2() {
```

```
// 查找 lison1 或者 lison12 评论过的 user （$in 查找符）
```

```
// db.users.find({"comments.author":{"$in":["lison1","lison12"]}}).pretty()
```

```
// 备注：跟数量无关，跟顺序无关；
```

```

Query query = query(where("comments.author").in(Arrays.asList("lison1","lison12")));
List<User> find = tempelate.find(query, User.class);
printUsers(find);

// 查找 lison1 和 lison12 都评论过的 user
// db.users.find({"comments.author":{"$all":["lison12","lison1"]}}).pretty()
// 备注：跟数量有关，跟顺序无关；

query = query(where("comments.author").all(Arrays.asList("lison1","lison12")));
find = tempelate.find(query, User.class);
printUsers(find);
}

@Test
//(1)注意相关的实体 bean 要加上注解@document， @dbRef
//(2)spring 对 dbRef 进行了封装，发起了两次查询请求
public void dbRefTest(){
    System.out.println("-----");
    List<User> users = tempelate.findAll(User.class);
    System.out.println("-----");
    System.out.println(users);
//    System.out.println(users.get(0).getComments());
}

private void printUsers(List<User> find) {
    for (User user : find) {
        System.out.println(user);
    }
    System.out.println(find.size());
}

//-----

//查找 lison5 评语为包含“苍老师”关键字的 user（$elemMatch 查找符）

```



```

// db.users.find({"comments":{"$elemMatch":{"author" : "lison5", "content" : { "$regex" : ".*
苍老师.*"}}}})
//备注：数组中对象数据要符合查询对象里面所有的字段，$全元素匹配，和顺序无关；

@Test
public void testObjArray3() throws ParseException {
//    and(where("author").is("lison5"),where("content").regex(".*苍老师.*")))
Criteria              andOperator              =              new
Criteria().andOperator(where("author").is("lison5"),where("content").regex(".*苍老师.*"));
Query query = query(where("comments").elemMatch(andOperator));
List<User> find = tempelate.find(query, User.class);
printUsers(find);
}

@Test
// db.users.find({"comments":{"$elemMatch":{"author" : "lison5","content" :
// "lison 是苍老师的小迷弟"}}}) .pretty()
public void testElemMatch() {
Query              query              =
query(where("comments").elemMatch(where("author").is("lison5").and("content").is("lison 是苍
老师的小迷弟")));
List<User> find = tempelate.find(query, User.class);
System.out.println(find.size());
}

/**
 *      db.users.updateOne({"username":"lison"},{
          {"$push": {
              "comments": {
                  $each: [{
                      "author" : "james",
                      "content" : "lison 是个好老师! ",
                      "commentTime"
                      :
                      ISODate("2018-01-06T04:26:18.354Z")
                  }
              ],
              $sort: {"commentTime":-1}
          }
      }
  });
 */
@Test
// 新增评论时，使用$sort 运算符进行排序，插入评论后，再按照评论时间降序排序
public void demoStep1() {

```

```

Query query = query(where("username").is("lison"));
Comment comment = new Comment();
comment.setAuthor("cang");
comment.setCommentTime(new Date());
comment.setContent("lison 是我的粉丝");

Update update = new Update();
PushOperatorBuilder pob = update.push("comments");
pob.each(comment);
pob.sort(Sort.by(Direction.DESC, "commentTime"));

System.out.println("-----");
UpdateResult updateFirst = templete.updateFirst(query, update, User.class);
System.out.println("-----");
System.out.println(updateFirst.getModifiedCount());
}

@Test
// 查看人员时加载最新的三条评论:
// db.users.find({"username":"lison"}, {"comments":{"$slice":[0,3]}}).pretty()
public void demoStep2() {
    // {"username":"lison"}
    Query query = query(where("username").is("lison"));
    // {"comments":{"$slice":[0,3]}}
    query.fields().include("comments").slice("comments", 0, 3);
    System.out.println("-----");
    List<User> find = templete.find(query, User.class);
    System.out.println("-----");
    System.out.println(find);
}

@Test
// 点击评论的下一页按钮，新加载三条评论
// db.users.find({"username":"lison"}, {"comments":{"$slice":[3,3]}, "$id":1}).pretty();
public void demoStep3() {
    Query query = query(where("username").is("lison"));
    query.fields().include("comments").slice("comments", 3, 3)
        .include("id");
    System.out.println("-----");
    List<User> find = templete.find(query, User.class);
    System.out.println("-----");
    System.out.println(find);
}

```

```

/**
 * db.users.aggregate([{"$match":{"username":"lison"}},
                        {"$unwind":"$comments"},
                        {"$sort":{"comments.commentTime":-1}},
                        {"$project":{"comments":1}},
                        {"$skip":6},
                        {"$limit":3}])

 */
// 如果有多种排序需求怎么处理,使用聚合
@Test
public void demoStep4() {
    Aggregation aggs = newAggregation(
        match(where("username").is("lison")),
        unwind("comments"),
        sort(Direction.ASC, "comments.commentTime"),
        project("comments"),
        skip(6),
        limit(3));
    System.out.println("-----");
    AggregationResults<Object> aggregate = template.aggregate(aggs, "users",
Object.class);
    System.out.println("-----");
    List<Object> mappedResults = aggregate.getMappedResults();
    System.out.println(mappedResults.size());
}
}

```

### 2.3.8. Mongodb 连接池配置

参数名	默认值	说明
writeConcern	ACKNOWLEDGED	写入安全机制，是一种客户端设置，用于控制写入安全的级别： ACKNOWLEDGED 默认选项，数据写入到Primary就向客户端发送确认 0 Unacknowledged 对客户端的写入不需要发送任何确认，适用于性能要求高，但不关注正确性的场景； 1 W1 数据写入后，会等待集群中1台发送确认 2 W2 数据写入后，会等待集群中两台发送确认 3 W3 数据写入后，会等待集群中3台发送确认 JOURNALED 确保所有数据提交到 journal file MAJORITY 等待集群中大多数服务器提交后确认；
codecRegistry	MongoClient.getDefaultCodecRegistry()	编解码类，实现Codec接口
minConnectionsPerHost		最小连接数，connections-per-host
connectionsPerHost	100	最大连接数
threadsAllowedToBlockForConnectionMultiplier	5	此参数跟connectionsPerHost的乘积为一个线程变为可用的最大阻塞数，超过此乘积数之后的所有线程将及时获取一个异常
maxWaitTime	1000 * 60 * 2	一个线程等待链接可用的最大等待毫秒数，0表示不等待
maxConnectionIdleTime	0	设置池连接的最大空闲时间，0表示没有限制
maxConnectionLifeTime	0	设置池连接的最大使用时间，0表示没有限制
connectTimeout	1000*10	连接超时时间
alwaysUseMBeans	false	是否打开JMX监控

参数名	默认值	说明
heartbeatFrequency	10000	设置心跳频率。这是驱动程序尝试确定群集中每个服务器的当前状态的频率。
minHeartbeatFrequency	500	设置最低心跳频率。如果驱动程序必须经常重新检查服务器的可用性，那么至少要等上一次检查以避免浪费。
heartbeatConnectTimeout	20000	心跳检测连接超时时间
heartbeatSocketTimeout	20000	心跳检测Socket超时时间

### 2.3.9. 数据模式设计

#### 2.3.9.1. mongoDB 的数据结构

```
{
  "_id" : ObjectId("59f938235d93fc4af8a37114"),
  "username" : "lison",
  "country" : "in11digo",
  "address" : {
    "aCode" : "邮编",
    "add" : "d11pff"
  },
  "favorites" : {
    "movies" : ["杀破狼2","1dushe","雷神1"],
    "cites" : ["1sh","1cs","1zz"]
  },
  "age" : 18 ,
  "salary" : NumberDecimal("2.099"),
  "lenght" : 1.79
}
```

### 2.3.9.2. MySql 等数据库

User表	
字段	类型
Id	Nvarchar
Username	Nvarchar
.....	.....

favorites表	
字段	类型
Id	Nvarchar
Type	Nvarchar
.....	.....

### 2.3.9.3. nosql 在数据模式设计上的优势

- 读写效率高-在 IO 性能上有先天独厚的优势；
- 可扩展能力强,不需要考虑关联，数据分区分库，水平扩展就比较简单；
- 动态模式，不要求每个文档都具有完全相同的结构。对很多异构数据场景支持非常好；
- 模型自然-文档模型最接近于我们熟悉的对象模型；

### 2.3.9.4. mongoDB 能不能实现关联查询？

先准备测试数据

```
var comments1 = {
  "_id": "xxoo1",
  "lists":
  [
    {
      "author" : "lison1",
```

```
        "content" : "lison 评论 1",
        "commentTime" : ISODate("2017-12-06T04:26:18")
    },
    {
        "author" : "lison2",
        "content" : "lison 评论 2",
        "commentTime" : ISODate("2017-12-06T04:26:18")
    },
    {
        "author" : "lison3",
        "content" : "lison 评论 3",
        "commentTime" : ISODate("2017-12-06T04:26:18")
    },
    {
        "author" : "lison4",
        "content" : "lison 评论 4",
        "commentTime" : ISODate("2017-12-06T04:26:18")
    },
    {
        "author" : "lison5",
        "content" : "lison 评论 5",
        "commentTime" : ISODate("2017-12-06T04:26:18")
    },
    {
        "author" : "lison6",
        "content" : "lison 评论 6",
        "commentTime" : ISODate("2017-12-06T04:26:18")
    },
    {
        "author" : "lison7",
        "content" : "lison 评论 7",
        "commentTime" : ISODate("2017-12-06T04:26:18")
    },
    {
        "author" : "lison8",
        "content" : "lison 评论 8",
        "commentTime" : ISODate("2017-12-06T04:26:18")
    },
    {
        "author" : "lison9",
        "content" : "lison 评论 9",
        "commentTime" : ISODate("2017-12-06T04:26:18")
    }
}
```

```
};

var comments2 = {
  "_id": "xxoo2",
  "lists":
  [
    {
      "author" : "james1",
      "content" : "james 评论 1",
      "commentTime" : ISODate("2017-12-06T04:26:18")
    },
    {
      "author" : "james2",
      "content" : "james 评论 2",
      "commentTime" : ISODate("2017-12-06T04:26:18")
    },
    {
      "author" : "james3",
      "content" : "james 评论 3",
      "commentTime" : ISODate("2017-12-06T04:26:18")
    },
    {
      "author" : "james4",
      "content" : "james 评论 4",
      "commentTime" : ISODate("2017-12-06T04:26:18")
    },
    {
      "author" : "james5",
      "content" : "james 评论 5",
      "commentTime" : ISODate("2017-12-06T04:26:18")
    },
    {
      "author" : "james6",
      "content" : "james 评论 6",
      "commentTime" : ISODate("2017-12-06T04:26:18")
    },
    {
      "author" : "james7",
      "content" : "james 评论 7",
      "commentTime" : ISODate("2017-12-06T04:26:18")
    },
    {
      "author" : "james8",
      "content" : "james 评论 8",
```

```

        "commentTime" : ISODate("2017-12-06T04:26:18")
    },
    {
        "author" : "james9",
        "content" : "james 评论 9",
        "commentTime" : ISODate("2017-12-06T04:26:18")
    }
]
};
db.comments.drop();
db.comments.insert(comments1);
db.comments.insert(comments2);

db.users.drop();
var user1 = {
    "username" : "lison",
    "country" : "china",
    "address" : {
        "aCode" : "411000",
        "add" : "长沙"
    },
    "favorites" : {
        "movies" : ["杀破狼 2","战狼","雷神 1"],
        "cites" : ["长沙","深圳","上海"]
    },
    "age" : 18,
    "salary":NumberDecimal("18889.09"),
    "lenght" :1.79,
    "comments":{
        "$ref" : "comments",
        "$id" : "xxoo1",
        "$db" : "lison"
    }
};
var user2 = {
    "username" : "james",
    "country" : "English",
    "address" : {
        "aCode" : "311000",
        "add" : "地址"
    },
    "favorites" : {
        "movies" : ["复仇者联盟","战狼","雷神 1"],
        "cites" : ["西安","东京","上海"]
    }
};

```



```
    },
    "age" : 24,
    "salary":NumberDecimal("7889.09"),
    "lenght" :1.35
};
var user3={
    "username" : "deer",
    "country" : "japan",
    "address" : {
        "aCode" : "411000",
        "add" : "长沙"
    },
    "favorites" : {
        "movies" : ["肉蒲团","一路向西","倩女幽魂"],
        "cites" : ["东莞","深圳","东京"]
    },
    "age" : 22,
    "salary":NumberDecimal("6666.66"),
    "lenght" :1.85
};
var user4 =
{
    "username" : "mark",
    "country" : "USA",
    "address" : {
        "aCode" : "411000",
        "add" : "长沙"
    },
    "favorites" : {
        "movies" : ["蜘蛛侠","钢铁侠","蝙蝠侠"],
        "cites" : ["青岛","东莞","上海"]
    },
    "age" : 20,
    "salary":NumberDecimal("6398.22"),
    "lenght" :1.77
};
var user5 =
{
    "username" : "peter",
    "country" : "UK",
    "address" : {
        "aCode" : "411000",
        "add" : "TEST"
```

```

    },
    "favorites" : {
        "movies" : ["蜘蛛侠","钢铁侠","蝙蝠侠"],
        "cites" : ["青岛","东莞","上海"]
    },
    "salary":NumberDecimal("1969.88")
};

db.users.insert(user1);
db.users.insert(user2);
db.users.insert(user3);
db.users.insert(user4);
db.users.insert(user5);

```

- 先考虑内嵌， 直接按照你的对象模型来设计你的数据模型。如果你的对象模型数量不多， 关系不是很复杂， 直接一种对象对应一个集合就可以了
- 单个 bson 文档最大不能超过 16M； 当文档超过 16M 的时候， 就应该考虑使用引用（DBRef）了， 在主表里存储一个 id 值， 指向另一个表中的 id 值。

DBRef 语法： { "\$ref" : <value>, "\$id" : <value>, "\$db" : <value> }

\$ref: 引用文档所在的集合的名称；

\$id: 所在集合的\_id 字段值；

\$db: 可选， 集合所在的数据库实例；

#### 2.3.9.4.1. 注意：

Tips: DBRef 只是关联信息的数据载体， 本身并不会去关联数据；

#### 2.3.9.4.2. 使用 dbref 脚本示例：

```

var lison = db.users.findOne({"username":"lison"});
var dbref = lison.comments;
db[dbref.$ref].findOne({"_id":dbref.$id})

```

#### 2.3.9.4.3. JAVA 客户端解析

JavaQueryTest.dbRefTest

```
@Test
public void dbRefTest() {
    FindIterable<Document> find = collection.find(eq( fieldName: "username", value: "lison"));
    printOperation(find);
}
```

est dbRefTest

1 test passed - 57ms

: details.

#### 2.3.9.4.4.1. 新增实体类 Comments

```
package cn.enjoy.entity;

import java.util.List;

import org.springframework.data.mongodb.core.mapping.Document;

@Document(collection="comments")
public class Comments {

    private List<Comment> lists;

    public List<Comment> getLists() {
        return lists;
    }

    public void setLists(List<Comment> lists) {
        this.lists = lists;
    }

    @Override
    public String toString() {
        return "Comments{" +
            "lists=" + lists +
            '}';
    }
}
```

```
}
```

#### 2.3.9.4.4.2. 修改 Users 实体类

```
package cn.enjoy.entity;

import java.math.BigDecimal;
import java.util.List;

import org.bson.types.ObjectId;
import org.springframework.data.mongodb.core.mapping.DBRef;
import org.springframework.data.mongodb.core.mapping.Document;

@Document(collection="users")
public class User {

    private ObjectId id;

    private String username;

    private String country;

    private Address address;

    private Favorites favorites;

    private int age;

    private BigDecimal salary;

    private float lenght;
    //private List<Comment> comments;
    private Comments comments;

    public Comments getComments() {
        return comments;
    }

    public void setComments(Comments comments) {
        this.comments = comments;
    }
}
```

```
public String getUsername() {  
    return username;  
}  
public void setUsername(String username) {  
    this.username = username;  
}  
public String getCountry() {  
    return country;  
}  
public void setCountry(String country) {  
    this.country = country;  
}  
public Address getAddress() {  
    return address;  
}  
public void setAddress(Address address) {  
    this.address = address;  
}  
public Favorites getFavorites() {  
    return favorites;  
}  
public void setFavorites(Favorites favorites) {  
    this.favorites = favorites;  
}  
public Objectid getId() {  
    return id;  
}  
public void setId(Objectid id) {  
    this.id = id;  
}  
public int getAge() {  
    return age;  
}  
public void setAge(int age) {  
    this.age = age;  
}  
public BigDecimal getSalary() {  
    return salary;  
}  
public void setSalary(BigDecimal salary) {  
    this.salary = salary;  
}  
public float getLenght() {
```

```

        return lenght;
    }

    public void setLenght(float lenght) {
        this.lenght = lenght;
    }

    // public List<Comment> getComments() {
    //     return comments;
    // }

    // public void setComments(List<Comment> comments) {
    //     this.comments = comments;
    // }

    @Override
    public String toString() {
        return "User [id=" + id + ", username=" + username + ", country="
            + country + ", address=" + address + ", favorites=" + favorites
            + ", age=" + age + ", salary=" + salary + ", lenght=" + lenght
            + ", comments=" + comments + "]\n";
    }
}

```

### 2.3.9.4.4.3. 测试

cn.enjoy.mg.SpringQueryTest#dbRefTest



```

it=1.79, comments=Comments(list=[Comment [author=lison1, commentTime=Wed Dec 06 12:26:18 CST 2017, content=lison评论1], Comment [author=lison2, commentTime=Wed Dec 06 12:26:18 CST 2017, content=lison评论2], Comment [auth

```

## 2.3.10. 聚合的理解

聚合框架就是定义一个管道，管道里的每一步都为下一步输出数据（类似于 JDK8 的 Stream API，既流式编程）



### 2.3.10.1. 常用的管道操作

**\$project**: 投影，指定输出文档中的字段；

**\$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 聚管道返回的文档数。

**\$skip**: 在聚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。

### 2.3.10.2. \$group 操作符

- **\$group**: 可以分组的数据执行如下的表达式计算：

**\$sum**: 计算总和。

**\$avg**: 计算平均值。

**\$min**: 根据分组，获取集合中所有文档对应值得最小值。

**\$max**: 根据分组，获取集合中所有文档对应值得最大值。

### 2.3.10.3. 聚合训练

#### 2.3.10.3.1. 新建实体 Order

```
package cn.enjoy.entity;

import java.math.BigDecimal;
import java.util.Date;

import org.bson.types.ObjectId;
```

```
import org.springframework.data.annotation.Id;
import org.springframework.data.mongodb.core.mapping.Document;

@Document(collection = "orders")
public class Order {

    @Id
    private String id;

    private String orderCode;

    private String useCode;

    private Date orderTime;

    private BigDecimal price;

    private String[] Auditors;

    public String getOrderCode() {
        return orderCode;
    }

    public void setOrderCode(String orderCode) {
        this.orderCode = orderCode;
    }

    public Date getOrderTime() {
        return orderTime;
    }

    public void setOrderTime(Date orderTime) {
        this.orderTime = orderTime;
    }

    public BigDecimal getPrice() {
        return price;
    }

    public void setPrice(BigDecimal price) {
        this.price = price;
    }

    public String getId() {
```



```

        return id;
    }

    public void setId(String id) {
        this.id = id;
    }

    public String getUseCode() {
        return useCode;
    }

    public void setUseCode(String useCode) {
        this.useCode = useCode;
    }

    public String[] getAuditors() {
        return Auditors;
    }

    public void setAuditors(String[] auditors) {
        Auditors = auditors;
    }
}

```

### 2.3.10.3.2. 产生测试数据

```

package cn.enjoy.mg;

import java.math.BigDecimal;
import java.text.SimpleDateFormat;
import java.util.Date;

import org.junit.Test;

public class RondonDateTest {

    @Test
    public void testRondonDate() {
        for(int i=0;i<=10000;i++){
            //            Date date = randomDate("2015-01-01","2017-10-31");

```

```

//                                System.out.println(new SimpleDateFormat("yyyy.MM.dd
HH:mm:ss").format(date));
        BigDecimal test = randomBigDecimal(10000,1);
        System.out.println(test.toString());
    }
}

/**
 * 获取随机日期
 * @param beginDate 起始日期，格式为： yyyy-MM-dd
 * @param endDate 结束日期，格式为： yyyy-MM-dd
 * @return
 */
public static Date randomDate(String beginDate,String endDate){
    try {
        SimpleDateFormat format = new SimpleDateFormat("yyyy-MM-dd");
        Date start = format.parse(beginDate);
        Date end = format.parse(endDate);

        if(start.getTime() >= end.getTime()){
            return null;
        }

        long date = random(start.getTime(),end.getTime());

        return new Date(date);
    } catch (Exception e) {
        e.printStackTrace();
    }
    return null;
}

private static long random(long begin,long end){
    long rtn = begin + (long)(Math.random() * (end - begin));
    if(rtn == begin || rtn == end){
        return random(begin,end);
    }
    return rtn;
}

public static BigDecimal randomBigDecimal(float max,float min) {
//    float Max = 10000, Min = 1.0f;
    BigDecimal db = new BigDecimal(Math.random() * (max - min) + min);

```

```

        return db.setScale(2, BigDecimal.ROUND_HALF_UP);// 保留 30 位小数并四舍五
入
    }
}

```

使用 GenarateOrdersTest 产生 100000 条测试数据，代码如下

```

package cn.enjoy.mg;

import static org.springframework.data.mongodb.core.query.Criteria.where;
import static org.springframework.data.mongodb.core.query.Query.query;
import static org.springframework.data.mongodb.core.query.Update.update;

import java.util.ArrayList;
import java.util.HashSet;
import java.util.List;
import java.util.Random;
import java.util.UUID;

import javax.annotation.Resource;
import cn.enjoy.entity.Order;
import org.junit.Test;
import org.junit.runner.RunWith;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.data.mongodb.core.MongoOperations;
import org.springframework.test.context.ContextConfiguration;
import org.springframework.test.context.junit4.SpringJUnit4ClassRunner;

@RunWith(SpringJUnit4ClassRunner.class)
@ContextConfiguration("classpath:applicationContext.xml")
public class GenarateOrdersTest {

    private static final Logger logger = LoggerFactory.getLogger(GenarateOrdersTest.class);

    @Resource
    private MongoOperations tempelate;

    //随机生成 orderTest 数据
    @Test
    public void batchInsertOrder() {
        String[] userCodes = new String[] { "james", "AV", "allen", "six",
            "peter", "mark", "king", "zero", "lance", "deer", "lison" };
        String[] auditors = new String[]

```

```

{ "auditor1","auditor2","auditor3","auditor4","auditor5"};
    List<Order> list = new ArrayList<Order>();
    Random rand = new Random();
    for (int i = 0; i < 100000; i++) {
        Order order = new Order();
        int num = rand.nextInt(11);
        order.setUseCode(userCodes[num]);
        order.setOrderCode(UUID.randomUUID().toString());
        order.setOrderTime(RandomDateTest.randomDate("2015-01-01","2017-10-31"));
        order.setPrice(RandomDateTest.randomBigDecimal(10000, 1));
        int length = rand.nextInt(5)+1;
        String[] temp = new String[length];
        for (int j = 0; j < temp.length; j++) {
            temp[j] = getFromArrays(temp,auditors,rand);
        }
        order.setAuditors(temp);
        list.add(order);
    }
    template.insertAll(list);
}

private String getFromArrays(String[] temp, String[] auditors, Random rand) {
    String ret = null;
    boolean test = true;
    while (test) {
        ret = auditors[rand.nextInt(5)];
        int i =0;
        for (String _temp : temp) {
            i++;
            if(ret.equals(_temp)){
                break;
            }
        }
        if(i==temp.length){
            test=false;
        }
    }
    return ret;
}
}

```

### 2.3.10.3.3. 训练 1

查询 2015 年 4 月 3 号之前，每个用户每个月消费的总金额，并按用户名进行排序：

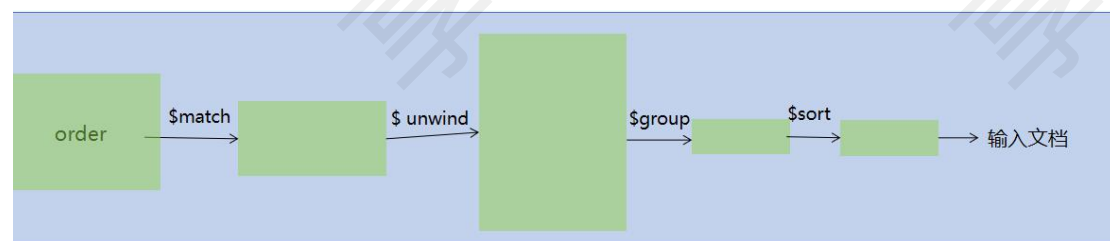
```
db.orders.aggregate([
  {"$match":{"orderTime":{"$lt":new Date("2015-04-03T16:00:00.000Z")}}},
  {"$group":{"_id":{"useCode":"$useCode","month":{"$month":"$orderTime"},"total":{"$sum":"$price"}}},
  {"$sort":{"_id":1}}
])
```



### 2.3.10.3.4. 训练 2

查询 2015 年 4 月 3 号之前，每个审核员分别审批的订单总金额，按审核员名称进行排序：

```
db.orders.aggregate([{"$match":{"orderTime":{"$lt":new Date("2015-04-03T16:00:00.000Z")}}},
  {"$unwind":"$Auditors"},
  {"$group":{"_id":{"Auditors":"$Auditors"},"total":{"$sum":"$price"}}},
  {"$sort":{"_id":1}}])
```



### 2.3.10.3.5. Java 代码

cn.enjoy.mg.JavaQueryTest#aggreationTest

```
/**
 * db.orders.aggregate([
 *   {"$match":{"orderTime":{"$lt":new Date("2015-04-03T16:00:00.000Z")}}},
 *   {"$group":{"_id":{"useCode":"$useCode","month":{"$month":"$orderTime"},"total":{"$sum":"$price"}}},
 *   {"$sort":{"_id":1}}
 * ])
```

```

price"}},
    {"$sort":{"_id":1}}
})
*/
@Test
public void aggregationTest1() throws Exception {
    Block<Document> printBlock = new Block<Document>() {
        @Override
        public void apply(Document t) {
            logger.info("-----");
            System.out.println(t.toJson());
            logger.info("-----");
        }
    };

    SimpleDateFormat formatter = new SimpleDateFormat("yyyy-MM-dd hh:mm:ss");
    Date commentDate = formatter.parse("2015-04-03 08:00:00");

    DBObject groupFiled = new BasicDBObject();
    groupFiled.put("useCode", "$useCode");
    groupFiled.put("month", eq("$month", "$orderTime"));

    List<Bson> aggregates = new ArrayList<Bson>();
    aggregates.add(match(it("orderTime", commentDate)));
    aggregates.add(group(groupFiled, Accumulators.sum("sum", "$price")));
    aggregates.add(sort(eq("_id", 1)));
    Aggregatable<Document> aggregate = orderCollection
        .aggregate(aggregates);
    aggregate.forEach(printBlock);
}

/**
 *
 * db.orders.aggregate([{"$match":{"orderTime": {"$lt": new
Date("2015-04-03T16:00:00.000Z")}},
    {"$unwind":"$Auditors"},
    {"$group":{"_id":{"Auditors":"$Auditors"},"total":{"$sum":"$price"}},
    {"$sort":{"_id":1}}]
*/
@Test
public void aggregationTest2() throws Exception {
    Block<Document> printBlock = new Block<Document>() {

```

```

@Override
public void apply(Document t) {
    logger.info("-----");
    System.out.println(t.toJson());
    logger.info("-----");
}
};

SimpleDateFormat formatter = new SimpleDateFormat("yyyy-MM-dd hh:mm:ss");
Date commentDate = formatter.parse("2015-04-03 08:00:00");
List<Bson> aggregates = new ArrayList<Bson>();
aggregates.add(match(lt("orderTime",commentDate)));
aggregates.add(unwind("$Auditors"));
aggregates.add(group("$Auditors", Accumulators.sum("sum", "$price")));
aggregates.add(sort(eq("_id",1)));
AggregateIterable<Document> aggregate = orderCollection
    .aggregate(aggregates);
aggregate.forEach(printBlock);
}

```

### 2.3.10.3.6. Spring Data 代码

cn.enjoy.mg.SpringQueryTest#aggretrionTest

```

@Test
public void aggretrionTest1() throws Exception {
    SimpleDateFormat formatter = new SimpleDateFormat("yyyy-MM-dd hh:mm:ss");
    Date commentDate = formatter.parse("2015-04-04 00:00:00");
    Aggregation aggs = newAggregation(
        match(where("orderTime").lt(commentDate)),

        project("useCode","price","orderTime").and(DateOperators.DateToString.dateOf("orderTime").toString("%m")).as("month"),
        group("useCode","month").sum("price").as("total"),
        sort(Sort.by(Direction.ASC, "_id"))
    );

    AggregationResults<Object> aggregate = template.aggregate(aggs, "orders", Object.class);
    List<Object> mappedResults = aggregate.getMappedResults();
    System.out.println(mappedResults);
}

```

```

    }

    /**
     *
     * db.orders.aggregate([{"$match":{"orderTime" : { "$lt" : new
Date("2015-04-03T16:00:00.000Z")}}},
    {"$unwind":"$Auditors"},
    {"$group":{"_id":{"Auditors":"$Auditors"},"total":{"$sum":"$price"}}},
    {"$sort":{"_id":1}}])
    */
    @Test
    public void aggregationTest2() throws Exception {
        SimpleDateFormat formatter = new SimpleDateFormat("yyyy-MM-dd hh:mm:ss");
        Date commentDate = formatter.parse("2015-04-04 00:00:00");
        Aggregation aggs = newAggregation(
            match(where("orderTime").lt(commentDate)),
            unwind("Auditors"),
            group("Auditors").sum("price").as("total"),
            sort(Sort.by(Direction.ASC,"_id"))
        );

        AggregationResults<Object> aggregate = template.aggregate(aggs, "orders",
Object.class);
        List<Object> mappedResults = aggregate.getMappedResults();
        System.out.println(mappedResults);
    }
}

```

## 2.4. 更新

### 2.4.1. 新增操作

insertOne: 插入单个文档

insertMany: 插入多个文档

如果数据库和集合不存在，insert 操作将自动创建；

对于插入的数据，mongoDB 自动生成 ObjectId 作为\_id 字段（物理主键）



## 2.4.2. 删除操作

`deleteOne(query)`: 删除单个文档

`deleteMany(query)`: 删除多个文档

删除操作是不会删除索引的，就算你把数据全部删除；

## 2.4.3. 修改

### 2.4.3.1. 更新的方法

#### 2.4.3.1.1. 替换更新

```
db.users.update({"username":"lison"},"country":"USA")
```

#### 2.4.3.1.2. 操作符更新（推荐使用）

- 性能更好
- 原子性操作

```
db.users.update({"username":"james"},"$set":{"country":"USA"})
```

### 2.4.3.2. 修改语法

`update()` 方法用于更新已存在的文档。语法格式如下：

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

参数说明：

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

### 2.4.3.2.1. 示例

```
db.users.update({"username":"cang"},"$set":{"age":18},"upsert":true))
```

- 数据不存在，记录将被插入
- 与插入操作相比，upsert 插入的结果返回了\_id 字段

### 2.4.3.3. 更新选择器

类型	运算符	描述
操作符	\$inc	指定值加n
	\$set	更新指定字段
	\$unset	将指定字段删除
	\$rename	更新字段名称
数组操作符	\$	定位到某一个元素
	\$push	添加值到数组中
	\$addToSet	添加值到数组中，有重复则不处理
	\$pop	删除数组第一个或者最后一个
	\$pull	从数组中删除匹配查询条件的值
	\$pullAll	从数组中删除多个值
数组运算修饰符	\$each	与\$push和\$addToSet等一起使用来操作多个值
	\$slice	与\$push和\$each一起使用来操作用来缩小更新后数组的大小
	\$sort	与\$push、\$each和\$slice一起使用来对数组进行排序

#### 2.4.3.3.1. 删除字段示例

```
db.users.updateMany({"username":"lison"},"$unset":{"country":"","age":""}))
```

#### 2.4.3.3.2. 更新字段名称示例

```
db.users.updateMany({"username":"lison"},"$rename":{"lenght":"height",  
"username":"name"}))
```

#### 2.4.3.3.3. \$each 作用示例

```
db.users.updateMany({ "username" : "james"}, { "$addToSet" : { "favorites.movies" : [ "小电影 2", "小电影 3"]}})  
db.users.updateMany({ "username" : "james"}, { "$addToSet" : { "favorites.movies" : { "$each" : [ "小电影 2", "小电影 3"]}}})
```

#### 2.4.3.3.4. 删除字符串数组中元素示例

```
db.users.updateMany({ "username" : "james"}, { "$pull" : { "favorites.movies" : [ "小电影 2 ", "小电影 3"] } })
```

```
db.users.updateMany({ "username" : "james"}, { "$pullAll" : { "favorites.movies" : [ "小电影 2 ", "小电影 3"] } })
```

```
db.users.updateMany({ "username" : "james"}, { "$pull" : { "favorites.movies" : { $in: [ "小电影 2 ", "小电影 3"] } } })
```

#### 2.4.3.3.5. 向对象数组中插入元素

给 **james** 老师增加一条评论（**\$push**,默认放在数组最后）

```
db.users.updateOne({ "username": "james"}, { "$push": { "comments": { "author": "lison23", "content": "ydddyytttt", "commentTime": ISODate("2019-01-06T00:00:00") } } })
```

给 **james** 老师批量新增两条评论（**\$push**,**\$each**）

```
db.users.updateOne({ "username": "james"}, { "$push": { "comments":
```

```
{ "$each": [ { "author": "lison22", "content": "yyyytttt", "commentTime": ISODate("2019-07-06T00:00:00") },
```

```
{ "author": "lison23", "content": "ydddyytttt", "commentTime": ISODate("2019-06-06T00:00:00") } ] } })
```

给 **james** 老师批量新增两条评论并对数组进行排序（**\$push**,**\$each**,**\$sort**）

```
db.users.updateOne({ "username": "james"}, { "$push": { "comments":
```

```
{ "$each": [ { "author": "lison22", "content": "yyyytttt", "commentTime": ISODate("2019-04-06T00:00:00") },
```

```
{ "author": "lison23", "content": "ydddyytttt", "commentTime": ISODate("2019-05-06T00:00:00") } ], $sort: { "commentTime": -1 } } })
```

#### 2.4.3.3.6. 删除对象数组中元素示

删除 lison22 对 james 的所有评论（批量删除）

```
db.users.update({"username":"james"},
                {"$pull":{"comments":{"author":"lison22"}}})
```

删除 lison5 对 lison 评语为"lison 是苍老师的小迷弟"的评论

```
db.users.update({"username":"lison"},
                {"$pull":{"comments":{"author":"lison5",
                                     "content":"lison
```

是苍老师的小迷弟"}}})

#### 2.4.3.3.7. 更新对象数组中元素，\$符号示例

```
db.users.updateMany({"username":"james","comments.author":"lison23"},
                    {"$set":{"comments.$.content":"xxoo",
                             "comments.$.author":"lison10" }})
```

含义：精确修改某人某一条精确的评论，如果有多个符合条件的数据，则修改第一条数据。无法批量修改数组元素，也无法对数组元素做批量更新

### 2.4.4. 更新的注意点

- mongodb 的更新都是原子的，mongodb 所有的写操作都是有锁的。mongoDB 2.2 之前锁级别为实例级别，mongoDB 2.2 到 3.2 之前的版本锁级别为数据库级别，mongoDB 3.2 以后，WiredTiger 的锁级别是文档级别；
- findAndModify 命令：在同一往返过程中原子更新文档并返回它；

#### 2.4.4.1. findAndModify 命令示例

- 常规的 update 的方法不能返回更新后的数据

```
db.fam.update({"name":"morris1"}, {"$inc":{"age":1}})
```

- 使用 findandModify 方法在修改数据同时返回更新前的数据或更新后的数据

```
db.fam.findAndModify({query:{name:'morris1'},
                     update:{$inc:{age:1}},
                     'new':true});
```

<https://docs.mongodb.com/manual/reference/method/db.collection.findAndModify/>

#### 2.4.4.1.1. 测试脚本

```
db.fam.drop()

var doc1 = {
  _id : 1,
  name : 'morris1',
  age : 18};

db.fam.insert(doc1);

var doc2 = {
  _id : 2,
  name : 'morris2',
  age : 18};

db.fam.insert(doc2);

var doc3 = {
  _id : 3,
  name : 'morris1',
  age : 18};

db.fam.insert(doc3);
```

```
var ret1  = db.fam.update({"name":"morris1"}, {"$inc":{"age":1}})
printjson(ret1);

var ret2 = db.fam.findAndModify({
  query:{name:'morris1'},

  update:{$inc:{age:1}},

                                     'new':true
});
```

## 2.4.5. JAVA 客户端实现

```
package cn.enjoy.mg;

import static com.mongodb.client.model.Filters.*;
import static com.mongodb.client.model.Projections.*;
import static com.mongodb.client.model.Sorts.*;
import static com.mongodb.client.model.Aggregates.*;
import static com.mongodb.client.model.Updates.*;

import java.text.ParseException;
import java.text.SimpleDateFormat;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.Date;
import java.util.List;

import javax.annotation.Resource;

import org.bson.BSON;
import org.bson.BsonDocument;
import org.bson.Document;
import org.bson.codecs.configuration.CodecRegistries;
import org.bson.codecs.configuration.CodecRegistry;
import org.bson.codecs.pojo.PojoCodecProvider;
import org.bson.conversions.Bson;
import org.bson.types.ObjectId;
import org.junit.Before;
import org.junit.Test;
import org.junit.runner.RunWith;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.test.context.ContextConfiguration;
import org.springframework.test.context.junit4.SpringJUnit4ClassRunner;

import com.mongodb.Block;
import com.mongodb.MongoClient;
import com.mongodb.MongoClientOptions;
import com.mongodb.ServerAddress;
import com.mongodb.WriteConcern;
import com.mongodb.client.AggregateIterable;
import com.mongodb.client.FindIterable;
```

```

import com.mongodb.client.MongoCollection;
import com.mongodb.client.MongoDatabase;
import com.mongodb.client.model.Filters;
import com.mongodb.client.model.FindOneAndUpdateOptions;
import com.mongodb.client.model.Projections;
import com.mongodb.client.model.PushOptions;
import com.mongodb.client.model.ReturnDocument;
import com.mongodb.client.model.UpdateOptions;
import com.mongodb.client.model.Updates;
import com.mongodb.client.result.UpdateResult;
import com.mongodb.operation.OrderBy;

@RunWith(SpringJUnit4ClassRunner.class)
@ContextConfiguration("classpath:applicationContext.xml")
public class JavaUpdateObjArray {

    private static final Logger logger = LoggerFactory.getLogger(JavaUpdateObjArray.class);

    private MongoDatabase db;

    private MongoCollection<Document> collection;

    @Resource(name="mongo")
    private MongoClient client;

    @Before
    public void init(){
        db = client.getDatabase("lison");
        collection=db.getCollection("users");
    }

    //-----upsert demo-----

    //测试 upsert
    //db.users.update({"username":"cang"},"$set":{"age":18}},{"upsert":true})
    @Test
    public void upsertTest(){
        Bson filter = eq("username","cang");
        Bson update = set("age", 18);
    }

```

```

        UpdateOptions upsert = new UpdateOptions().upsert(true);
        UpdateResult updateOne = collection.updateOne(filter, update, upsert);
        System.out.println(updateOne.getModifiedCount());
        System.out.println(updateOne.getUpsertedId());
    }

    //测试 unset,删除字段示例
    //db.users.updateMany({"username":"lison"}, {"$unset":{"country":"","age":""}})
    @Test
    public void unsetTest(){
        Bson filter = eq("username", "lison");
        Bson country = unset("country");
        Bson age = unset("age");
        Bson update = combine(country, age);
        UpdateResult updateOne = collection.updateMany(filter, update);
        System.out.println(updateOne.getModifiedCount());
        System.out.println(updateOne.getUpsertedId());
    }

    //测试 rename,更新字段名称示例
    //db.users.updateMany({"username":"lison"}, {"$rename":{"length":"height",
"username":"name"}})

    @Test
    public void renameTest(){
        Bson filter = eq("username", "lison");
        Bson rename1 = rename("length", "height");
        Bson rename2 = rename("username", "name");
        Bson update = combine(rename1, rename2);
        UpdateResult updateOne = collection.updateMany(filter, update);
        System.out.println(updateOne.getModifiedCount());
        System.out.println(updateOne.getUpsertedId());
    }

    //测试 pull pullAll,删除字符串数组中元素示例
    //    db.users.updateMany( {"username" : "james"}, { "$pull" : { "favorites.movies" : [ "小电影 2
", "小电影 3"]}})
    //    db.users.updateMany( {"username" : "james"}, { "$pullAll" : { "favorites.movies" : [ "小电
影 2 ", "小电影 3"]}})

```



```

@Test
public void pullAllTest(){
    Bson filter = eq("username","james");
    Bson pull = pull("favorites.movies", Arrays.asList("小电影 2 ", "小电影 3"));
    UpdateResult updateOne = collection.updateMany(filter, pull);
    System.out.println(updateOne.getModifiedCount());
    System.out.println(updateOne.getUpsertedId());

    Bson pullAll = pullAll("favorites.movies", Arrays.asList("小电影 2 ", "小电影 3"));
    updateOne = collection.updateMany(filter, pullAll);
    System.out.println(updateOne.getModifiedCount());
    System.out.println(updateOne.getUpsertedId());
}

//-----insert demo-----

//给 james 老师增加一条评论（$push）
//db.users.updateOne({"username":"james"},
//                    {"$push":{"comments":{"author":"lison23",
//                    "content":"ydddytyttt",
//
"@Test
public void addOneComment(){
    Document comment = new Document().append("author", "lison23")
                                     .append("content", "ydddytyttt")
                                     .append("commentTime",
getDate("2019-01-06"));
    Bson filter = eq("username","james");
    Bson update = push("comments",comment);
    UpdateResult updateOne = collection.updateOne(filter, update);
    System.out.println(updateOne.getModifiedCount());
}

```

```

// 给 james 老师批量新增两条评论 ($push,$each)
// db.users.updateOne({"username":"james"},
//     {"$push":{"comments":
//
// {"$each":[{"author":"lison22","content":"yyyytttt","commentTime":ISODate("2019-02-06T00:00:
00")}},
//
// {"author":"lison23","content":"ydddytyttt","commentTime":ISODate("2019-03-06T00:00:00")}}]}
// })

@Test
public void addManyComment(){
    Document comment1 = new Document().append("author", "lison33")
        .append("content", "lison33lison33")
        .append("commentTime",
getDate("2019-02-06"));
    Document comment2 = new Document().append("author", "lison44")
        .append("content", "lison44lison44")
        .append("commentTime",
getDate("2019-03-06"));

    Bson filter = eq("username","james");
    Bson pushEach = pushEach("comments",Arrays.asList(comment1,comment2));
    UpdateResult updateOne = collection.updateOne(filter, pushEach);
    System.out.println(updateOne.getModifiedCount());

}

// 给 james 老师批量新增两条评论并对数组进行排序 ($push,$eachm,$sort)
// db.users.updateOne({"username":"james"},
//     {"$push":{"comments":
//
// {"$eachm":[{"author":"lison22","content":"yyyytttt","commentTime":ISODate("2019-04-06T00:00:
00")}},
//
// {"author":"lison23","content":"ydddytyttt","commentTime":ISODate("2019-05-06T00:00:00")}}],
//     $sort: {"commentTime":-1} } } })

@Test
public void addManySortComment(){
    Document comment1 = new Document().append("author", "lison00")
        .append("content", "lison00lison00")
        .append("commentTime",

```

```

getDate("2019-04-06"));
    Document comment2 = new Document().append("author", "lison01")
                                        .append("content", "lison01lison01")
                                        .append("commentTime",
getDate("2019-05-06"));

    Bson filter = eq("username","james");

    Document sortDoc = new Document().append("commentTime", -1);
    PushOptions pushOption = new PushOptions().sortDocument(sortDoc);

    Bson                                pushEach
pushEach("comments",Arrays.asList(comment1,comment2),pushOption);

    UpdateResult updateOne = collection.updateOne(filter, pushEach);
    System.out.println(updateOne.getModifiedCount());

}

//-----delete demo-----

// 删除 lison1 对 james 的所有评论 （批量删除）
// db.users.update({"username": "james"},
//                  {"$pull":{"comments":{"author":"lison33"}}})

@Test
public void deleteByAuthorComment(){
    Document comment = new Document().append("author", "lison33");
    Bson filter = eq("username","james");
    Bson update = pull("comments",comment);
    UpdateResult updateOne = collection.updateOne(filter, update);
    System.out.println(updateOne.getModifiedCount());
}

// 删除 lison5 对 lison 评语为 “lison 是苍老师的小迷弟” 的评论（精确删除）
// db.users.update({"username":"lison"},
//                  {"$pull":{"comments":{"author":"lison5",
//                  "content":"lison 是苍老师的小迷弟"}}})

@Test
public void deleteByAuthorContentComment(){
    Document comment = new Document().append("author", "lison5")
                                        .append("content", "lison 是苍老师的小迷弟
");

```

```

        Bson filter = eq("username","lison");
        Bson update = pull("comments",comment);
        UpdateResult updateOne = collection.updateOne(filter, update);
        System.out.println(updateOne.getModifiedCount());
    }

    //-----update demo-----
    //    db.users.updateMany({"username":"james","comments.author":"lison01"},
    //        {"$set":{"comments.$.content":"xxoo",
    //            "comments.$.author":"lison10" }})
    //    含义：精确修改某人某一条精确的评论，如果有多个符合条件的数据，则修改最后
    //    一条数据。无法批量修改数组元素
    @Test
    public void updateOneComment(){
        Bson filter = and(eq("username","james"),eq("comments.author","lison01"));
        Bson updateContent = set("comments.$.content","xxoo");
        Bson updateAuthor = set("comments.$.author","lison10");
        Bson update = combine(updateContent,updateAuthor);
        UpdateResult updateOne = collection.updateOne(filter, update);
        System.out.println(updateOne.getModifiedCount());
    }

    //-----findandModify
    //demo-----
    //使用 findandModify 方法在修改数据同时返回更新前的数据或更新后的数据
    //db.fam.findAndModify({query:{name:'morris1'},
    //    update:{$inc:{age:1}},
    //    'new':true});

    @Test
    public void findAndModifyTest(){
        Bson filter = eq("name","morris1");
        Bson update = inc("age",1);
        //    //实例化 findAndModify 的配置选项
        FindOneAndUpdateOptions fauo = new FindOneAndUpdateOptions();
        //    //配置"new":true
        fauo.returnDocument(ReturnDocument.AFTER);//
        MongoCollection<Document> numCollection = db.getCollection("fam");
        Document ret = numCollection.findOneAndUpdate(filter, update,fauo);
        System.out.println(ret.toJson());
    }
}

```

```
private Date getDate(String string) {  
    SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd");  
  
    Date parse=null;  
    try {  
        parse = sdf.parse(string);  
    } catch (ParseException e) {  
        e.printStackTrace();  
    }  
    return parse;  
}  
}
```

## 2.4.6. Spring Data 实现

### 2.4.6.1. 把 User 实体修改回来

```
public List<Comment> getComments() {  
    return comments;  
}  
  
public void setComments(List<Comment> comments) {  
    this.comments = comments;  
}  
  
private List<Comment> comments;
```

### 2.4.6.2. 新增实体 Doc

```
package cn.enjoy.entity;  
  
import org.springframework.data.mongodb.core.mapping.Document;  
  
@Document(collection="fam")  
public class Doc {  
  
    private String id;  
  
    private String name;
```

```
private int age;

public String getId() {
    return id;
}

public void setId(String id) {
    this.id = id;
}

public String getName() {
    return name;
}

public void setName(String name) {
    this.name = name;
}

public int getAge() {
    return age;
}

public void setAge(int age) {
    this.age = age;
}

@Override
public String toString() {
    return "Doc [id=" + id + ", name=" + name + ", age=" + age + "];"
}

}
```

#### 2.4.6.3. 单元测试

```
package cn.enjoy.mg;
```

```

import static org.springframework.data.mongodb.core.query.Criteria.where;
import static org.springframework.data.mongodb.core.query.Query.query;
import static org.springframework.data.mongodb.core.query.Update.update;

import java.text.ParseException;
import java.text.SimpleDateFormat;
import java.util.Arrays;
import java.util.Date;

import javax.annotation.Resource;

import com.mongodb.client.result.UpdateResult;
import org.bson.types.ObjectId;
import org.junit.Test;
import org.junit.runner.RunWith;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.data.domain.Sort;
import org.springframework.data.domain.Sort.Direction;
import org.springframework.data.mongodb.core.FindAndModifyOptions;
import org.springframework.data.mongodb.core.MongoOperations;
import org.springframework.data.mongodb.core.query.Criteria;
import org.springframework.data.mongodb.core.query.Query;
import org.springframework.data.mongodb.core.query.Update;
import org.springframework.data.mongodb.core.query.Update.PushOperatorBuilder;
import org.springframework.test.context.ContextConfiguration;
import org.springframework.test.context.junit4.SpringJUnit4ClassRunner;

import cn.enjoy.entity.Comment;
import cn.enjoy.entity.Doc;
import cn.enjoy.entity.User;
import com.mongodb.WriteResult;

@RunWith(SpringJUnit4ClassRunner.class)
@ContextConfiguration("classpath:applicationContext.xml")
public class SpringUpdateObjArray {

    private static final Logger logger = LoggerFactory.getLogger(SpringUpdateObjArray.class);

    @Resource
    private MongoOperations tempelate;

```

```

//-----upsert demo-----

//测试 upsert
//db.users.update({"username":"cang"},"$set":{"age":18},"upsert":true})
@Test
public void upsertTest(){
    Query query = query(Criteria.where("username").is("cang"));
    Update set = new Update().set("age", 18);
    UpdateResult upsert = tempelate.upsert(query, set, User.class);
    System.out.println(upsert.getModifiedCount());
    System.out.println(upsert.getUpsertedId());
}

//测试 unset,删除字段示例
//db.users.updateMany({"username":"lison"},"$unset":{"country":"","age":""})
@Test
public void unsetTest(){
    Query query = query(Criteria.where("username").is("lison"));
    Update unset = new Update().unset("country").unset("age");

    UpdateResult upsert = tempelate.updateMulti(query, unset, User.class);
    System.out.println(upsert.getModifiedCount());
}

//测试 rename,更新字段名称示例
//db.users.updateMany({"username":"lison"},"$rename":{"lenght":"height",
"username":"name"}})

@Test
public void renameTest(){
    Query query = query(Criteria.where("username").is("lison"));
    Update rename = new Update().rename("lenght", "height").rename("username",
"name");
    UpdateResult upsert = tempelate.updateMulti(query, rename, User.class);
    System.out.println(upsert.getModifiedCount());
}

//测试 pull pullAll,删除字符串数组中元素示例

```



```

//      db.users.updateMany({ "username" : "james"}, { "$pull" : { "favorites.movies" : [ "小电影 2
", "小电影 3"]}})
//      db.users.updateMany({ "username" : "james"}, { "$pullAll" : { "favorites.movies" : [ "小电
影 2 ", "小电影 3"]}})

@Test
public void pullAllTest(){

    Query query = query(Criteria.where("username").is("james"));
    Update pull = new Update().pull("favorites.movies", Arrays.asList("小电影 2 ", "小电影
3"));

    UpdateResult upsert = tempelate.updateMulti(query, pull, User.class);
    System.out.println(upsert.getModifiedCount());

    query = query(Criteria.where("username").is("james"));
    Update pullAll = new Update().pullAll("favorites.movies", new String[]{"小电影 2 ", "小
电影 3"});
    upsert = tempelate.updateMulti(query, pullAll, User.class);
    System.out.println(upsert.getModifiedCount());
}

//-----insert demo-----

//给 james 老师增加一条评论 ($push)
//db.users.updateOne({"username":"james"},
//      {"$push":{"comments":{"author":"lison23",
//      "content":"ydddytyttt",
//
"commentTime":ISODate("2019-01-06T00:00:00")}}})
@Test
public void addOneComment(){
    Query query = query(Criteria.where("username").is("james"));
    Comment comment = new Comment();
    comment.setAuthor("lison23");
    comment.setContent("ydddytyttt");
    comment.setCommentTime(getDate("2019-01-06"));
    Update push = new Update().push("comments", comment);

```

```

        UpdateResult updateFirst = tempelate.updateFirst(query, push, User.class);
        System.out.println(updateFirst.getModifiedCount());
    }

    // 给 james 老师批量新增两条评论 ($push,$each)
    // db.users.updateOne({"username":"james"},
    //     {"$push":{"comments":
    //         {"$each":[{"author":"lison22","content":"yyyytttt","commentTime":ISODate("2019-02-06T00:00:
    //         00")},
    //         {"author":"lison23","content":"ydddyyytttt","commentTime":ISODate("2019-03-06T00:00:00")}}]}
    // )

    @Test
    public void addManyComment(){
        Query query = query(Criteria.where("username").is("james"));
        Comment comment1 = new Comment();
        comment1.setAuthor("lison55");
        comment1.setContent("lison55lison55");
        comment1.setCommentTime(getDate("2019-02-06"));
        Comment comment2 = new Comment();
        comment2.setAuthor("lison66");
        comment2.setContent("lison66lison66");
        comment2.setCommentTime(getDate("2019-03-06"));
        //Update      push      =      new      Update().pushAll("comments",      new
Comment[] {comment1,comment2});
        //Update      push      =      new      Update().push("comments",      new
Comment[] {comment1,comment2});
        Update      push      =      new      Update().push("comments").each(new
Comment[] {comment1,comment2});
        UpdateResult updateFirst = tempelate.updateFirst(query, push, User.class);
        System.out.println(updateFirst.getModifiedCount());
    }

    // 给 james 老师批量新增两条评论并对数组进行排序 ($push,$eachm,$sort)
    // db.users.updateOne({"username":"james"},
    //     {"$push": {"comments":
    //         {"$each": [ {"author":"lison22","content":"yyyytttt","commentTime":ISODate("2019-04-06T00:00:
    //         00")},
    //         {"author":"lison23","content":"ydddyyytttt","commentTime":ISODate("2019-05-06T00:00:00")} ],
    //         $sort: {"commentTime":-1} } } })

```

```

@Test
public void addManySortComment(){
    Query query = query(Criteria.where("username").is("james"));
    Comment comment1 = new Comment();
    comment1.setAuthor("lison77");
    comment1.setContent("lison55lison55");
    comment1.setCommentTime(getDate("2019-04-06"));
    Comment comment2 = new Comment();
    comment2.setAuthor("lison88");
    comment2.setContent("lison66lison66");
    comment2.setCommentTime(getDate("2019-05-06"));

    Update update = new Update();
    PushOperatorBuilder pob = update.push("comments");
    pob.each(comment1,comment2);
    pob.sort(Sort.by(Direction.DESC, "commentTime"));

    System.out.println("-----");
    UpdateResult updateFirst = tempelate.updateFirst(query, update,User.class);
    System.out.println(updateFirst.getModifiedCount());
}

//-----delete demo-----

// 删除 lison1 对 james 的所有评论 （批量删除）
// db.users.update({"username":"james"},
// {"$pull":{"comments":{"author":"lison23"}}})

@Test
public void deleteByAuthorComment(){
    Query query = query(Criteria.where("username").is("james"));

    Comment comment1 = new Comment();
    comment1.setAuthor("lison55");

/*
    BasicDBObject comment1 = new BasicDBObject ();
    comment1.put("author","lison23");*/

    Update pull = new Update().pull("comments",comment1);
    UpdateResult updateFirst = tempelate.updateFirst(query, pull, User.class);
    System.out.println(updateFirst.getModifiedCount());
}

```

```

// 删除 lison5 对 lison 评语为“lison 是苍老师的小迷弟”的评论（精确删除）
// db.users.update({"username":"lison"},
//     {"$pull":{"comments":{"author":"lison5",
//         "content":"lison 是苍老师的小迷弟"}}})

@Test
public void deleteByAuthorContentComment(){
    Query query = query(Criteria.where("username").is("lison"));
    Comment comment1 = new Comment();
    comment1.setAuthor("lison5");
    comment1.setContent("lison 是苍老师的小迷弟");
    Update pull = new Update().pull("comments",comment1);
    UpdateResult updateFirst = tempelate.updateFirst(query, pull, User.class);
    System.out.println(updateFirst.getModifiedCount());
}

//-----update demo-----
// db.users.updateMany({"username":"james","comments.author":"lison1"},
//     {"$set":{"comments.$.content":"xxoo",
//         "comments.$.author":"lison10" }})
// 含义：精确修改某人某一条精确的评论，如果有多个符合条件的数据，则修改最后
// 一条数据。无法批量修改数组元素
@Test
public void updateOneComment(){
    Query query
    query(where("username").is("lison").and("comments.author").is("lison4"));
    Update update
    update("comments.$.content","xxoo").set("comments.$.author","lison11");
    UpdateResult updateFirst = tempelate.updateFirst(query, update, User.class);
    System.out.println(updateFirst.getModifiedCount());
}

//-----findandModify
demo-----

//使用 findandModify 方法在修改数据同时返回更新前的数据或更新后的数据
//db.fam.findAndModify({query:{name:'morris1'},
//    update:{$inc:{age:1}},
//    'new':true});

@Test

```

```
public void findAndModifyTest(){
    Query query = query(where("name").is("morris1"));
    Update update = new Update().inc("age", 1);
    FindAndModifyOptions famo = FindAndModifyOptions.options().returnNew(true);

    Doc doc = tempelate.findAndModify(query, update,famo, Doc.class);
    System.out.println(doc.toString());
}

private Date getDate(String string) {
    SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd");

    Date parse=null;
    try {
        parse = sdf.parse(string);
    } catch (ParseException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }
    return parse;
}
}
```

## 2.4.7. 查询实战演练

### 2.4.7.1. 需求描述

- A. 查看一个人的信息，打开页面只显示三条评论
- B. 点击评论的下一页按钮，新加载三条评论
- C. 默认按照评论时间降序，但是也可以选择按照姓名排序

### 2.4.7.2. 难点

- A. 数组中数据的排序问题？
- B. 数组中的数据怎么按照指定的方式进行排序？
- C. 每次仅仅加载三条评论信息（可以包含 id 字段）？

### 2.4.7.3. 提示

- A. 添加数据时注意排序
- B. 查询的时候投影是有技巧的
- C. 排序考虑聚合？

### 2.4.7.4. 答案

#### 2.4.7.4.1. 脚本

(1) 考虑到默认顺序，所以新增评论时，使用\$sort 运算符按照评论时间降序排序；

```
db.users.updateOne({"username":"lison"},
  {
    "$push": {
      "comments": {
        $each: [
          {
            "author": "james",
            "content": "lison 是个好老师!",
            "commentTime"
          }
        ],
        $sort: {"commentTime":-1}
      }
    }
  }
);
```

注意：\$sort 操作符必须和\$each 配合使用

(2) 由于评论已经按照时间降序排序，所以查看人员时直接加载最新的三条评论；

```
db.users.find({"username":"lison"},{"comments":{"$slice":[0,3]}}).pretty()
```

(3) 点击评论的下一页按钮，新加载后三条评论（注意：仅仅加载评论的数据，人员信息不加载）

```
db.users.find({"username":"lison"},{"comments":{"$slice":[3,3]},"$id":1}).pretty();
```

(4) 如果有多种排序需求怎么处理? 使用聚合

```
db.users.aggregate([{"$match":{"username":"lison"},
{"$unwind":"$comments"},
{"$sort":{"comments.commentTime":1}},
{"$project":{"comments":1}},
{"$skip":0},
{"$limit":3}])
```

#### 2.4.7.4.2. 代码

cn.enjoy.mg.JavaQueryTest#demoStep1

```
@Test
// 新增评论时, 使用$sort 运算符进行排序, 插入评论后, 再按照评论时间降序排序
public void demoStep1() {
    Bson filter = eq("username", "lison");
    Document comment = new Document().append("author", "cang")
        .append("content", "lison 是我的粉丝")
        .append("commentTime", new Date());
    // $sort: {"commentTime":-1}
    Document sortDoc = new Document().append("commentTime", -1);
    PushOptions sortDocument = new PushOptions().sortDocument(sortDoc);
    // $each
    Bson pushEach = Updates.pushEach("comments", Arrays.asList(comment),
        sortDocument);

    UpdateResult updateOne = collection.updateOne(filter, pushEach);
    System.out.println(updateOne.getModifiedCount());
}

@Test
// 查看人员时加载最新的三条评论:
// db.users.find({"username":"lison"},{"comments":{"$slice":[0,3]}}).pretty()
public void demoStep2() {
    FindIterable<Document> find = collection.find(eq("username", "lison"))
        .projection(slice("comments", 0, 3));
    printOperation(find);
}

@Test
```

```

// 点击评论的下一页按钮，新加载三条评论
// db.users.find({"username":"lison"},{"comments":{"$slice":[3,3]}, "$id":1}).pretty();
public void demoStep3() {
    // {"username":"lison"}
    Bson filter = eq("username", "lison");
    // "$slice":[3,3]
    Bson slice = slice("comments", 3, 3);
    // "$id":1
    Bson includeID = include("id");

    // {"comments":{"$slice":[3,3]}, "$id":1}
    Bson projection = fields(slice, includeID);

    FindIterable<Document> find = collection.find(filter).projection(
        projection);
    printOperation(find);
}

@Test
/**
 * db.users.aggregate([{"$match":{"username":"lison"}},
                        {"$unwind":"$comments"},
                        {"$sort":{"comments.commentTime":-1}},
                        {"$project":{"comments":1}},
                        {"$skip":6},
                        {"$limit":3}])
 */
// 如果有多种排序需求怎么处理,使用聚合
public void demoStep4() {
    final List<Document> ret = new ArrayList<Document>();
    Block<Document> printBlock = getBlock(ret);
    List<Bson> aggregates = new ArrayList<Bson>();

    aggregates.add(match(eq("username", "lison")));
    aggregates.add(unwind("$comments"));
    aggregates.add(sort(orderBy(ascending("comments.commentTime"))));
    aggregates.add(project(fields(include("comments"))));
    aggregates.add(skip(0));
    aggregates.add(limit(3));

    AggregateIterable<Document> aggregate = collection
        .aggregate(aggregates);

    printOperation(ret, printBlock, aggregate);
}

```



```
}
```

## 2.5. 其他命令

### 2.5.1. 其他常用命令

show dbs : 显示数据库列表

show collections : 显示集合列表

db : 显示当前数据库

db.stats() : 显示数据库信息

db.serverStatus() : 查看服务器状态

db.dropDatabase(): 删除数据库

db.help(), db.collection.help(): 内置帮助, 显示各种方法的说明;

db.users.find().size(): 获取查询集合的数量;

db.users.drop(): 删除集合;

### 2.5.2. MongoDB 怎么优雅关机?

在生产环境, 不要用 kill -9 关掉 mongod 的进程, 很可能造成 mongodb 的数据丢失;  
优雅的关机:

- 第一种方式

use admin

db.shutdownServer()

- 第二种方式

mongod --shutdown -f mongod.conf (service mongodb start)

mongod --shutdown -f /soft/mongodb/conf/mgdb.conf --auth

### 2.5.3. 数据管理命令

- 数据备份 mongodump

mongodump -p 27022 -d lison -o /soft/backup

-p :端口; -d :备份的数据库名称 ; -o:指定备份的路径

其本质为: 执行查询, 然后写入文件;

- 数据恢复 mongorestore

mongorestore -p 27022 -d lison /soft/backup/lison --drop

--drop 已存在 lison 库则删除原数据库, 去掉--drop 则是合并

- 数据导出 **mongoexport**（针对集合）

```
mongoexport -p 27022 -d lison -c users -f id,username,age,salary --type=json -o /soft/backup/users.json
```

-c :指定导出的集合； -f :要导出的字段； --type: 导出的文件格式类型[csv,json]

- 数据导入 **mongoimport**（针对集合）

```
mongoimport -p 27022 -d lison -c users /soft/backup/users.json --upsert
```

```
mongoimport -p 27022 -d lison -c users /soft/backup/users.json --upsert
```

--upsert 表示更新现有数据，如果不适用—upsert,则导入时已经存在的文档会报 id 重复，数据不再插入，也可以使用—drop 删除原有数据

## 2.6. 安全

### 2.6.1. Role-Based Access Control 基于角色的控制

角色类型	类型说明	角色名称	说明
数据库一般角色 (Database User Roles)	每个数据库都包含的一般角色；	read	提供读取所有非系统集合和部分系统集合的数据的能力，系统集合包括：system.indexes、system.js和system.namespaces集合。
		readWrite	提供read角色的所有权限以及修改所有非系统集合和system.js集合上的数据的能力。
数据库管理角色 (Database Administration Roles)	每个数据库都包含的数据库管理角色；	dbAdmin	提供执行管理任务的能力，如与模式相关的任务、索引、收集统计信息。此角色不授予用户和角色管理的权限。
		userAdmin	提供在当前数据库上创建和修改角色和用户的能力。
		dbOwner	提供对数据库执行任何管理操作的能力。此角色结合了readWrite、dbAdmin和userAdmin角色授予的权限。
集群管理角色 (Cluster Administration Roles)	在admin数据库创建，用于管理整个数据库集群系统而不是特定数据库的角色。这些角色包括但不限于副本集和分片集群管理功能。	clusterManager	在集群上提供管理和监视操作。具有此角色的用户可以分别访问在分片和复制中使用的config和local数据库。
		clusterMonitor	为监控工具（如MongoDB Cloud Manager和Ops Manager监控代理）提供只读访问权限。
		hostManager	提供监视和管理服务器的能力。
		clusterAdmin	提供权限最高的集群管理访问。此角色结合了由clusterManager、clusterMonitor和hostManager角色授予的权限。此外，该角色还提供了dropDatabase操作。

角色类型	类型说明	角色名称	说明
备份和恢复角色 (Backup and Restoration Roles)	在admin数据库创建，用于专门的备份和恢复的角色	backup	提供备份数据所需的权限。此角色提供足够的权限来使用MongoDB Cloud Manager备份代理、Ops Manager备份代理或使用mongodump。
		restore	提供使用mongorestore恢复数据所需的权限
全数据库角色 (All-Database Roles)	在admin数据库创建，适用于除mongod实例中的local和config之外的所有数据库：	readAnyDatabase	提供与读取相同的只读权限，除了适用于群集中除本地和配置数据库以外的所有权限。该角色还提供了整个集群上的listDatabases操作。
		readWriteAnyDatabase	提供与readWrite相同的读取和写入权限，除了它适用于群集中除本地和配置数据库以外的所有数据。该角色还提供了整个集群上的listDatabases操作。
		userAdminAnyDatabase	提供与userAdmin相同的用户管理操作访问权限，除了适用于群集中除本地数据库和配置数据库外的所有数据。
		dbAdminAnyDatabase	提供与dbAdmin相同的数据库管理操作访问权限，除了它适用于除集群中的本地数据库和配置数据库以外的所有数据库管理操作。该角色还提供了整个集群上的listDatabases操作。
超级角色 (Superuser Roles)	所有资源的完整权限	root	提供对readWriteAnyDatabase、dbAdminAnyDatabase、userAdminAnyDatabase、clusterAdmin、还原和备份相结合的操作和所有资源的访问。

## 2.6.2. 客户端授权

### 2.6.2.1. shell 脚本创建用

```
db.createUser({'user':'boss', 'pwd':'boss', 'roles':[{'role':'userAdminAnyDatabase', 'db':'admin'}]})
db.createUser({'user':'lison', 'pwd':'lison', 'roles':[{'role':'readWrite', 'db':'lison'}]})
```

Tips:

服务器启动需要加上 auth 参数连接服务器才需要验证

如：mongod -f /soft/mongodb/conf/mgdb.conf --auth

切换到数据库上，才能给当前数据库创建用户；

### 2.6.2.2. MongoDB 权限初始化过程

1.启动 mongod

2.数据库增加安全模式后，初始化一个“userAdminAnyDatabase”非常重要

通过客户端连接，使用 admin 数据库，执行如下脚本：

```
db.createUser({'user':'boss', 'pwd':'boss', 'roles':[{'role':'userAdminAnyDatabase', 'db':'admin'}]})
```

3.使用刚创建成功的用户登录：db.auth("boss","boss");

4.切换到 lison 数据库（use lison），创建读写权限用户：

```
db.createUser({'user':'lison', 'pwd':'lison', 'roles':[{'role':'readWrite', 'db':'lison'}]})
```

5.使用读写权限用户 lison 登录，db.auth("lison","lison")，登录后测试;

ps:也可以以非 auth 模式启动，然后创建用户后，用 auth 模式启动

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

### 2.6.2.3. Java 客户端安全认证

MongoCredential 类包括每个受支持的身份验证机制的静态工厂方法。

```
public static MongoCredential createCredential(final String userName,  
                                              final String database,  
                                              final char[] password)
```

```
package cn.enjoy.mg;  
  
import static com.mongodb.client.model.Filters.*;  
import static com.mongodb.client.model.Projections.*;  
import static com.mongodb.client.model.Sorts.*;  
import static com.mongodb.client.model.Aggregates.*;  
  
import java.text.ParseException;  
import java.text.SimpleDateFormat;  
import java.time.LocalDateTime;  
import java.time.ZoneId;  
import java.time.ZonedDateTime;  
import java.util.ArrayList;  
import java.util.Arrays;  
import java.util.Date;  
import java.util.List;  
  
import javax.annotation.Resource;  
  
import org.bson.BSON;  
import org.bson.BsonDocument;  
import org.bson.Document;  
import org.bson.codecs.configuration.CodecRegistries;  
import org.bson.codecs.configuration.CodecRegistry;  
import org.bson.codecs.pojo.PojoCodecProvider;  
import org.bson.conversions.Bson;
```

```
import org.junit.Before;
import org.junit.Test;
import org.junit.runner.RunWith;
import org.junit.runner.manipulation.Filter;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.data.mongodb.core.MongoOperations;
import org.springframework.test.context.ContextConfiguration;
import org.springframework.test.context.junit4.SpringJUnit4ClassRunner;

import com.mongodb.Block;
import com.mongodb.MongoClient;
import com.mongodb.MongoClientOptions;
import com.mongodb.MongoCredential;
import com.mongodb.ReadPreference;
import com.mongodb.ServerAddress;
import com.mongodb.WriteConcern;
import com.mongodb.client.AggregateIterable;
import com.mongodb.client.FindIterable;
import com.mongodb.client.MongoCollection;
import com.mongodb.client.MongoDatabase;
import com.mongodb.client.model.Accumulators;
import com.mongodb.client.model.Filters;
import com.mongodb.client.model.Projections;
import com.mongodb.client.model.PushOptions;
import com.mongodb.client.model.Updates;
import com.mongodb.client.result.UpdateResult;
import com.mongodb.operation.OrderBy;

public class JavaAuthTest {

    private static final Logger logger = LoggerFactory
        .getLogger(JavaAuthTest.class);

    private MongoDatabase db;

    private MongoCollection<Document> collection;

    private MongoClient client;

    @Before
    public void init() {
        MongoCredential createCredential =
```

```

        MongoCredential.createCredential("lison", "lison", "lison".toCharArray());
        MongoClientOptions mco = MongoClientOptions.builder()
            .writeConcern(WriteConcern.JOURNALED)
            .connectionsPerHost(100)
            .readPreference(ReadPreference.secondary())
            .threadsAllowedToBlockForConnectionMultiplier(5)
            .maxWaitTime(120000).connectTimeout(10000).build();
        List<ServerAddress> asList = Arrays.asList(
            new ServerAddress("192.168.244.123",27017));
        this.client = new MongoClient(asList, createCredential,mco);
        db = client.getDatabase("lison");
        collection = db.getCollection("users");
    }

    // -----操作符使用实例-----

    // db.users.find({"username":{"$in":["lison", "mark", "james"]}}).pretty()
    // 查询姓名为 lison、mark 和 james 这个范围的人
    @Test
    public void testInOper() {
        Bson in = in("username", "lison", "mark", "james");
        FindIterable<Document> find = collection.find(in);
        printOperation(find);
    }

    // -----

    //返回对象的处理器，打印每一行数据
    private Block<Document> getBlock(final List<Document> ret) {
        Block<Document> printBlock = new Block<Document>() {
            @Override
            public void apply(Document t) {
                logger.info("-----");
                logger.info(t.toJson());
                logger.info("-----");
                ret.add(t);
            }
        };
        return printBlock;
    }

    //打印查询出来的数据和查询的数据量
    private void printOperation( FindIterable<Document> find) {
        final List<Document> ret = new ArrayList<Document>();
    }

```

```
Block<Document> printBlock = getBlock(ret);
find.forEach(printBlock);
System.out.println(ret.size());
ret.removeAll(ret);
}
}
```

#### 2.6.2.4. spring 客户端安全认证

```
<mongo:mongo-client id="mongo" host="192.168.244.123" port="27017"
credentials="lison:lison@lison">
<!--<mongo:mongo-client id="mongo" host="192.168.244.123" port="27017">-->
  <mongo:client-options
    write-concern="ACKNOWLEDGED"
    threads-allowed-to-block-for-connection-multiplier="5"
    max-wait-time="1200"
    connect-timeout="1000"/>
</mongo:mongo-client>
```

## 3. MongoDB 高级进阶

### 3.1. 存储引擎

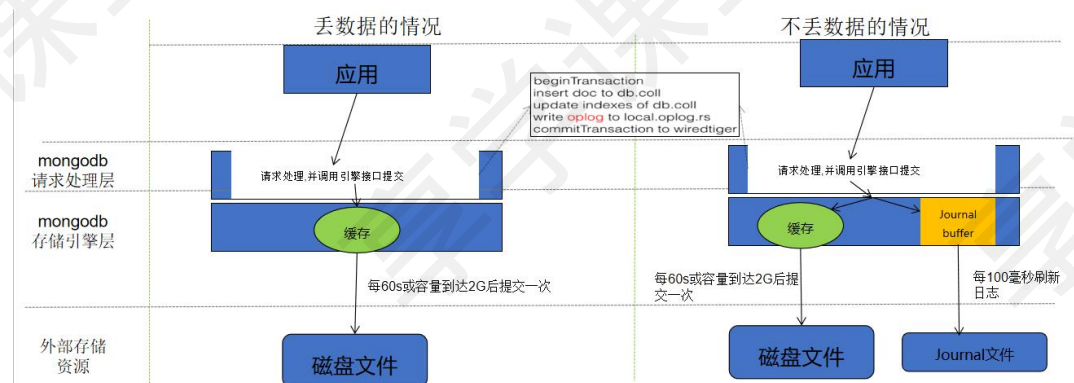
#### 3.1.1. wiredTiger

MongoDB 从 3.0 开始引入可插拔存储引擎的概念。目前主要有 MMAPV1、WiredTiger 存储引擎可供选择。在 3.2 版本之前 MMAPV1 是默认的存储引擎,其采用 linux 操作系统内存映射技术,但一直饱受诟病; 3.4 以上版本默认的存储引擎是 wiredTiger,相对于 MMAPV1 其有如下优势:

- 读写操作性能更好,WiredTiger 能更好的发挥多核系统的处理能力;
- MMAPV1 引擎使用表级锁,当某个单表上有并发的操作,吞吐将受到限制。WiredTiger 使用文档级锁,由此带来并发及吞吐的提高
- 相比 MMAPV1 存储索引时 WiredTiger 使用前缀压缩,更节省对内存空间的损耗;
- 提供压缩算法,可以大大降低对硬盘资源的消耗,节省约 60%以上的硬盘资源;

### 3.1.2. WT 写入的原理

mongodb 数据会丢失？你需要了解 WT 写入的原理



#### Journaling

类似于关系数据库中的事务日志。Journaling 能够使 MongoDB 数据库由于意外故障后快速恢复。MongoDB 2.4 版本后默认开启了 Journaling 日志功能, mongod 实例每次启动时都会检查 journal 日志文件看是否需要恢复。由于提交 journal 日志会产生写入阻塞, 所以它对写入的操作有性能影响, 但对于读没有影响。在生产环境中开启 Journaling 是很有必要的。

### 3.1.3. 写策略解析

```
//需要等待返回结果
db.users.updateMany({"username":"lison"}, {"$unset":{"country":"","age":""}}, {writeConcern:
{w:1, j: true, wtimeout: 5000 }})

//不需要等待返回结果
db.users.updateMany({"username":"lison"}, {"$unset":{"country":"","age":""}}, {writeConcern:
{w:0, j: true, wtimeout: 5000 }})
```

写策略配置: { w: <value>, j: <boolean>, wtimeout: <number> }

- w: 数据写入到 number 个节点才向用客户端确认
- {w: 0} 对客户端的写入不需要发送任何确认, 适用于性能要求高, 但不关注正确性的场景
- {w: 1} 默认的 writeConcern, 数据写入到 Primary 就向客户端发送确认
- {w: "majority"} 数据写入到副本集大多数成员后向客户端发送确认, 适用于对数据安全性要求比较高的场景, 该选项会降低写入性能



- **j**: 写入操作的 **journal** 持久化后才向客户端确认  
默认为{j: false}, 如果要求写入持久化了才向客户端确认, 则指定该选项为 true
- **wtimeout**: 写入超时时间, 仅 **w** 的值大于 1 时有效。  
当指定{w: }时, 数据需要成功写入 **number** 个节点才算成功, 如果写入过程中有节点故障, 可能导致这个条件一直不能满足, 从而一直不能向客户端发送确认结果, 针对这种情况, 客户端可设置 **wtimeout** 选项来指定超时时间, 当写入过程持续超过该时间仍未结束, 则认为写入失败。

### 3.1.4. Java 代码实现写策略

**Q1: 写策略配置相关的类是?**

答: com.mongodb.WriteConcern, 其中有如下几个常用写策略配置:

- UNACKNOWLEDGED: 不等待服务器返回或确认, 仅可以抛出网络异常;
- ACKNOWLEDGED: 默认配置, 等待服务器返回结果;
- JOURNALED: 等待服务器完成 **journal** 持久化之后返回;
- W1 : 等待集群中一台服务器返回结果;
- W2 : 等待集群中两台服务器返回结果;
- W3 : 等待集群中三台服务器返回结果;
- MAJORITY: 等待集群中多数服务器返回结果;

**Q2: Java 代码中如何加入写策略**

答: Java 客户端可以按两种方式来设置写策略:

- 在 MongoClient 初始化过程中使用 MongoClientOptions.writeConcern(writeConcern)来进行配置;

cn.enjoy.mg.QuickStartJavaPojoTest#init

```
MongoClientOptions build
MongoClientOptions.builder().writeConcern(WriteConcern.ACKNOWLEDGED).
    codecRegistry(registry).build();
```

- 在写操作过程中, 也可动态的指定写策略, mongodb 可以在三个层次来进行写策略的配置, 既 MongoClient、MongoDatabase、MongoCollection 这三个类都可以通过 WriteConcern 方法来设置写策略;

**Q3: Spring 中如何配置写策略**

在配置文件中配置, 如下图:

```

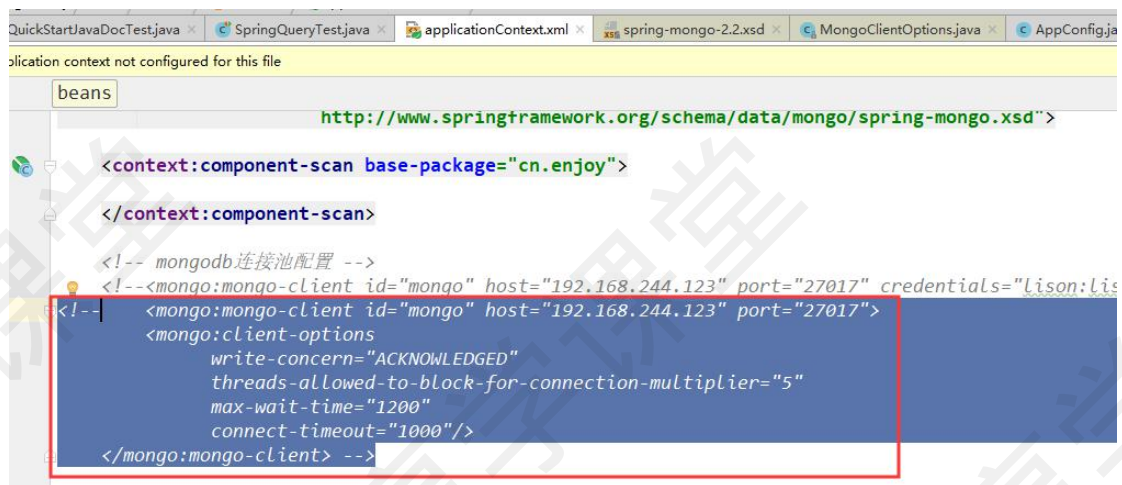
<!-- <mongo:mongo-client host="116.62.222.124" port="27022" credentials="lison:lison@lison"> --
<mongo:mongo-client host="116.62.222.124" port="27022" >
<!--replica-set="192.168.1.142:27017,192.168.1.142:27018,192.168.1.142:27017"-->
<mongo:client-options
  write-concern="JOURNALLED"
  connections-per-host="1"
  threads-allowed-to-block-for-connection-multiplier="5"
  max-wait-time="120000"
  connect-timeout="10000"/>
</mongo:mongo-client>

```

**Q4: WriteConcern 类的几个常用写策略配置满足不了项目的需求，怎么修改？**

答：通过配置类，在 Spring 容器中加入自己定制化的 MongoClient，代码如下：

### 3.1.4.1. 注释 applicationContext.xml



### 3.1.4.2. 增加配置类

```

package cn.enjoy.config;

import java.util.Arrays;
import java.util.List;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;

import com.mongodb.MongoClient;
import com.mongodb.MongoClientOptions;
import com.mongodb.MongoCredential;
import com.mongodb.ReadPreference;
import com.mongodb.ServerAddress;
import com.mongodb.WriteConcern;

```

```

@Configuration
public class AppConfig {

    /**
     * Use the standard Mongo driver API to create a com.mongodb.MongoClient instance.
     */
    @Bean(name="mongo")
    public MongoClient mongoClient() {

        //      MongoCredential createCredential =
        //          MongoCredential.createCredential("lison", "lison", "lison".toCharArray());

        WriteConcern wc = WriteConcern.W1.withJournal(true);
        MongoClientOptions mco = MongoClientOptions.builder()
            .writeConcern(wc)
            .connectionsPerHost(100)
            .readPreference(ReadPreference.secondary())
            .threadsAllowedToBlockForConnectionMultiplier(5)
            .maxWaitTime(120000).connectTimeout(10000).build();
        //      List<ServerAddress> asList = Arrays.asList(
        //          new ServerAddress("192.168.1.142", 27018),
        //          new ServerAddress("192.168.1.142", 27017),
        //          new ServerAddress("192.168.1.142", 27019));
        List<ServerAddress> asList = Arrays.asList(
            new ServerAddress("192.168.244.123", 27017));

        MongoClient client = new MongoClient(asList, mco);
        return client;
    }
}

```

### 3.1.5. 配置文件

```

storage:
  journal:
    enabled: true
  dbPath: /data/zhou/mongo1/
  ##是否一个库一个文件夹
  directoryPerDB: true
  ##数据引擎
  engine: wiredTiger
  ##WT 引擎配置

```

```
wiredTiger:
  engineConfig:
    ##WT 最大使用 cache（根据服务器实际情况调节）
    cacheSizeGB: 1
    ##是否将索引也按数据库名单独存储
    directoryForIndexes: true
    journalCompressor:none （默认 snappy）
  ##表压缩配置
  collectionConfig:
    blockCompressor: zlib (默认 snappy,还可选 none、zlib)
  ##索引配置
  indexConfig:
    prefixCompression: true
```

#### 压缩算法 Tips:

性能: none > snappy > zlib

压缩比: zlib > snappy > none

<https://docs.mongodb.com/v4.0/reference/configuration-options/#storage.wiredTiger.collectionConfig.blockCompressor>