1、MongoDB入门

1.1、MongoDB简介

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

MongoDB是一个介于关系数据库和非关系数据库之间的产品，是非关系数据库当中功能最丰富，最像关系数据库 的，它支持的数据结构非常松散，是类似json的bson格式，因此可以存储比较复杂的数据类型。

MongoDB最大的特点是它支持的查询语言非常强大，其语法有点类似于面向对象的查询语言，几乎可以实现类似 关系数据库单表查询的绝大部分功能，而且还支持对数据建立索引。

官网:https://www.mongodb.com

1.2、通过docker安装MongoDB

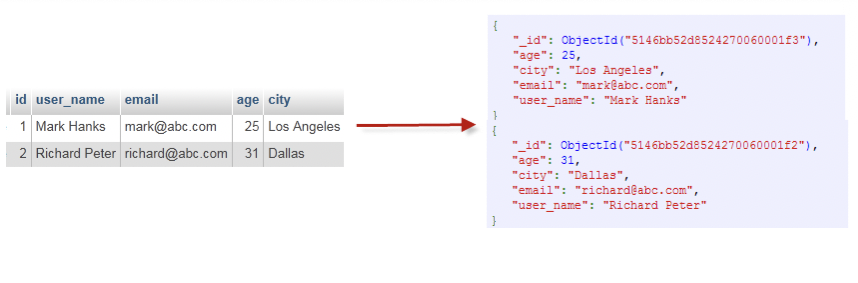
|  |
| --- |
| #拉取镜像 docker pull mongo:4.0.3  #创建容器 docker create --name mongodb -p 27017:27017 -v /data/mongodb:/data/db mongo:4.0.3  #启动容器 docker start mongodb  #进入容器 docker exec -it mongodb /bin/bash  #使用MongoDB客户端进行操作 mongo > show dbs #查询所有的数据库 admin 0.000GB config 0.000GB local 0.000GB |

1.3、MongoDB基本操作

1.3.1、基本概念

为了更好的理解，下面与SQL中的概念进行对比:

|  |  |  |
| --- | --- | --- |
| SQL术语/概念 | MongoDB术语/概念 | 解释/说明 |
| database | database | 数据库 |
| table | collection | 数据库表/集合 |
| row | document | 数据记录行/文档 |
| column | field | 数据字段/域 |
| index | index | 索引 |
| table joins |  | 表连接,MongoDB不支持 |
| primary key | primary key | 主键,MongoDB自动将\_id字段设置为主键 |



1.3.2、数据库以及表的操作

|  |
| --- |
| #查看所有的数据库 > show dbs admin 0.000GB config 0.000GB local 0.000GB  #通过use关键字切换数据库 > use admin switched to db admin  #创建数据库 #说明:在MongoDB中，数据库是自动创建的，通过use切换到新数据库中，进行插入数据即可自动创建数据库 > use testdb switched to db testdb > show dbs #并没有创建数据库 admin 0.000GB config 0.000GB local 0.000GB > db.user.insert({id:1,name:'zhangsan'}) #插入数据 WriteResult({ "nInserted" : 1 }) > show dbs  admin 0.000GB config 0.000GB local 0.000GB testdb 0.000GB #数据库自动创建  #查看表 > show tables user > show collections user >  #删除集合(表) > db.user.drop() true #如果成功删除选定集合，则 drop() 方法返回 true，否则返回 false。  #删除数据库 > use testdb #先切换到要删除的数据中 switched to db testdb > db.dropDatabase() #删除数据库 { "dropped" : "testdb", "ok" : 1 } > show dbs admin 0.000GB config 0.000GB local 0.000GB |

1.3.3、新增数据

在MongoDB中，存储的文档结构是一种类似于json的结构，称之为bson(全称为:Binary JSON)。

|  |
| --- |
| #插入数据  #语法:db.COLLECTION\_NAME.insert(document) > db.user.insert({id:1,username:'zhangsan',age:20}) WriteResult({ "nInserted" : 1 }) > db.user.save({id:2,username:'lisi',age:25}) WriteResult({ "nInserted" : 1 }) > db.user.find() #查询数据 { "\_id" : ObjectId("5c08c0024b318926e0c1f6dc"), "id" : 1, "username" : "zhangsan", "age" : 20 } { "\_id" : ObjectId("5c08c0134b318926e0c1f6dd"), "id" : 2, "username" : "lisi", "age" : 25 } |

1.3.4、更新数据

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objNew,true为插入，默认是 false，不插入。 multi : 可选，mongodb 默认是false,只更新找到的第一条记录，如果这个参数为true,就把按条件查出来多条 记录全部更新。 writeConcern :可选，抛出异常的级别。

|  |
| --- |
| > db.user.find()  { "\_id" : ObjectId("5c08c0024b318926e0c1f6dc"), "id" : 1, "username" : "zhangsan",  "age" : 20 }  { "\_id" : ObjectId("5c08c0134b318926e0c1f6dd"), "id" : 2, "username" : "lisi",  "age" : 25 }  > db.user.update({id:1},{$set:{age:22}}) #更新数据 WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })  > db.user.find()  { "\_id" : ObjectId("5c08c0024b318926e0c1f6dc"), "id" : 1, "username" : "zhangsan",  "age" : 22 }  { "\_id" : ObjectId("5c08c0134b318926e0c1f6dd"), "id" : 2, "username" : "lisi",  "age" : 25 }  #注意:如果这样写，会删除掉其他的字段 > db.user.update({id:1},{age:25}) WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 }) > db.user.find() { "\_id" : ObjectId("5c08c0024b318926e0c1f6dc"), "age" : 25 } { "\_id" : ObjectId("5c08c0134b318926e0c1f6dd"), "id" : 2, "username" : "lisi", "age" : 25 }  #更新不存在的字段，会新增字段 > db.user.update({id:2},{$set:{sex:1}}) #更新数据 > db.user.find() { "\_id" : ObjectId("5c08c0024b318926e0c1f6dc"), "age" : 25 } { "\_id" : ObjectId("5c08c0134b318926e0c1f6dd"), "id" : 2, "username" : "lisi", "age" : 25, "sex" : 1 }  #更新不存在的数据，默认不会新增数据  > db.user.update({id:3},{$set:{sex:1}})  WriteResult({ "nMatched" : 0, "nUpserted" : 0, "nModified" : 0 })  > db.user.find()  { "\_id" : ObjectId("5c08c0024b318926e0c1f6dc"), "age" : 25 }  { "\_id" : ObjectId("5c08c0134b318926e0c1f6dd"), "id" : 2, "username" : "lisi",  "age" : 25, "sex" : 1 }  #如果设置第一个参数为true，就是新增数据 > db.user.update({id:3},{$set:{sex:1}},true) WriteResult({  "nMatched" : 0,  "nUpserted" : 1,  "nModified" : 0,  "\_id" : ObjectId("5c08cb281418d073246bc642")  })  > db.user.find()  { "\_id" : ObjectId("5c08c0024b318926e0c1f6dc"), "age" : 25 }  { "\_id" : ObjectId("5c08c0134b318926e0c1f6dd"), "id" : 2, "username" : "lisi",  "age" : 25, "sex" : 1 }  { "\_id" : ObjectId("5c08cb281418d073246bc642"), "id" : 3, "sex" : 1 } |

1.3.5、删除数据

通过remove()方法进行删除数据，语法如下:

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

参数说明:

query :(可选)删除的文档的条件。 justOne : (可选)如果设为 true 或 1，则只删除一个文档，如果不设置该参数，或使用默认值 false，则删 除所有匹配条件的文档。 writeConcern :(可选)抛出异常的级别。

实例:

|  |
| --- |
| > db.user.remove({age:25}) WriteResult({ "nRemoved" : 2 }) #删除了2条数据  #插入4条测试数据 db.user.insert({id:1,username:'zhangsan',age:20}) db.user.insert({id:2,username:'lisi',age:21}) db.user.insert({id:3,username:'wangwu',age:22}) db.user.insert({id:4,username:'zhaoliu',age:22})  > db.user.remove({age:22},true) WriteResult({ "nRemoved" : 1 }) #删除了1条数据  #删除所有数据 > db.user.remove({})  #说明:为了简化操作，官方推荐使用deleteOne()与deleteMany()进行删除数据操作。 db.user.deleteOne({id:1}) db.user.deleteMany({}) #删除所有数据 |

1.3.6、查询数据

MongoDB 查询数据的语法格式如下:

|  |
| --- |
| db.user.find([query],[fields]) |

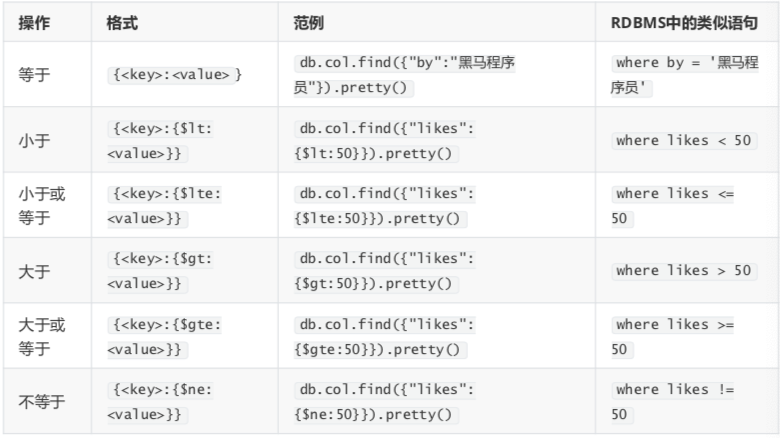
query :可选，使用查询操作符指定查询条件 fields :可选，使用投影操作符指定返回的键。查询时返回文档中所有键值， 只需省略该参数即可(默认省 略)。

如果你需要以易读的方式来读取数据，可以使用 pretty() 方法，语法格式如下:

|  |
| --- |
| >db.col.find().pretty() |

pretty() 方法以格式化的方式来显示所有文档。

条件查询:



实例:

|  |
| --- |
| #插入测试数据 db.user.insert({id:1,username:'zhangsan',age:20}) db.user.insert({id:2,username:'lisi',age:21}) db.user.insert({id:3,username:'wangwu',age:22}) db.user.insert({id:4,username:'zhaoliu',age:22})  db.user.find() #查询全部数据 db.user.find({},{id:1,username:1}) #只查询id与username字段 db.user.find().count() #查询数据条数 db.user.find({id:1}) #查询id为1的数据 db.user.find({age:{$lte:21}}) #查询小于等于21的数据 db.user.find({age:{$lte:21}, id:{$gte:2}}) #and查询，age小于等于21并且id大于等于2 db.user.find({$or:[{id:1},{id:2}]}) #查询id=1 or id=2  #分页查询:Skip()跳过几条，limit()查询条数 db.user.find().limit(2).skip(1) #跳过1条数据，查询2条数据  db.user.find().sort({id:-1}) #按照age倒序排序，-1为倒序，1为正序 |

1.4、索引

索引通常能够极大的提高查询的效率，如果没有索引，MongoDB在读取数据时必须扫描集合中的每个文件并选取

那些符合查询条件的记录。

这种扫描全集合的查询效率是非常低的，特别在处理大量的数据时，查询可以要花费几十秒甚至几分钟，这对网站

的性能是非常致命的。

索引是特殊的数据结构，索引存储在一个易于遍历读取的数据集合中，索引是对数据库表中一列或多列的值进行排

序的一种结构

|  |
| --- |
| #查看索引 > db.user.getIndexes() [  { "v" : 2,  } ]  "key" : {  "\_id" : 1  },  "name" : "\_id\_",  "ns" : "testdb.user"  #说明:1表示升序创建索引，-1表示降序创建索引。  #创建索引 > db.user.createIndex({'age':1}) {  "createdCollectionAutomatically" : false,  "numIndexesBefore" : 1,  "numIndexesAfter" : 2,  "ok" : 1  } |

|  |
| --- |
| #删除索引  db.user.dropIndex("age\_1")  #或者，删除除了\_id之外的索引  db.user.dropIndexes() |

|  |
| --- |
| #创建联合索引  db.user.createIndex({'age':1, 'id':-1}) |

|  |
| --- |
| #查看索引大小，单位:字节  db.user.totalIndexSize() |

1.5、执行计划

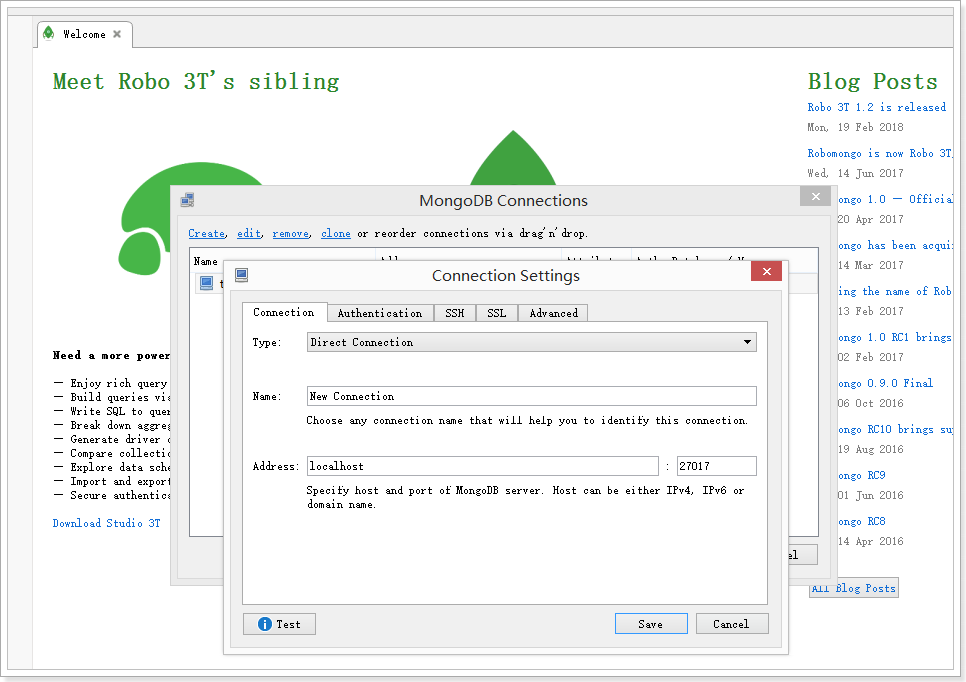
MongoDB 查询分析可以确保我们建议的索引是否有效，是查询语句性能分析的重要工具。

|  |
| --- |
| #插入1000条数据  for(var i=1;i<1000;i++)db.user.insert({id:100+i,username:'name\_'+i,age:10+i})  #查看执行计划  > db.user.find({age:{$gt:100},id:{$lt:200}}).explain() {  "queryPlanner" : {  "plannerVersion" : 1,  "namespace" : "testdb.user",  "indexFilterSet" : false,  "parsedQuery" : {  "$and" : [ {  "id" : {  "$lt" : 200  } },  { "age" : {  } ]  }, "winningPlan" : { #最佳执行计划  "stage" : "FETCH", #查询方式，常见的有COLLSCAN/全表扫描、IXSCAN/索引扫描、 FETCH/根据索引去检索文档、SHARD\_MERGE/合并分片结果、IDHACK/针对\_id进行查询  "inputStage" : {  "stage" : "IXSCAN",  "keyPattern" : {  "age" : 1,  "id" : -1 },  "indexName" : "age\_1\_id\_-1",  "isMultiKey" : false,  "multiKeyPaths" : {  "age" : [ ],  "id" : [ ] },  "isUnique" : false,  "isSparse" : false,  "isPartial" : false,  "indexVersion" : 2,  "direction" : "forward",  "indexBounds" : {  "age" : [  "(100.0, inf.0]"  ], "id" : [  "(200.0, -inf.0]"  ] }  "rejectedPlans" : [ ]  },  "serverInfo" : {  "host" : "c493d5ff750a",  "port" : 27017,  "version" : "4.0.3",  "gitVersion" : "7ea530946fa7880364d88c8d8b6026bbc9ffa48c"  },  "ok" : 1 } |

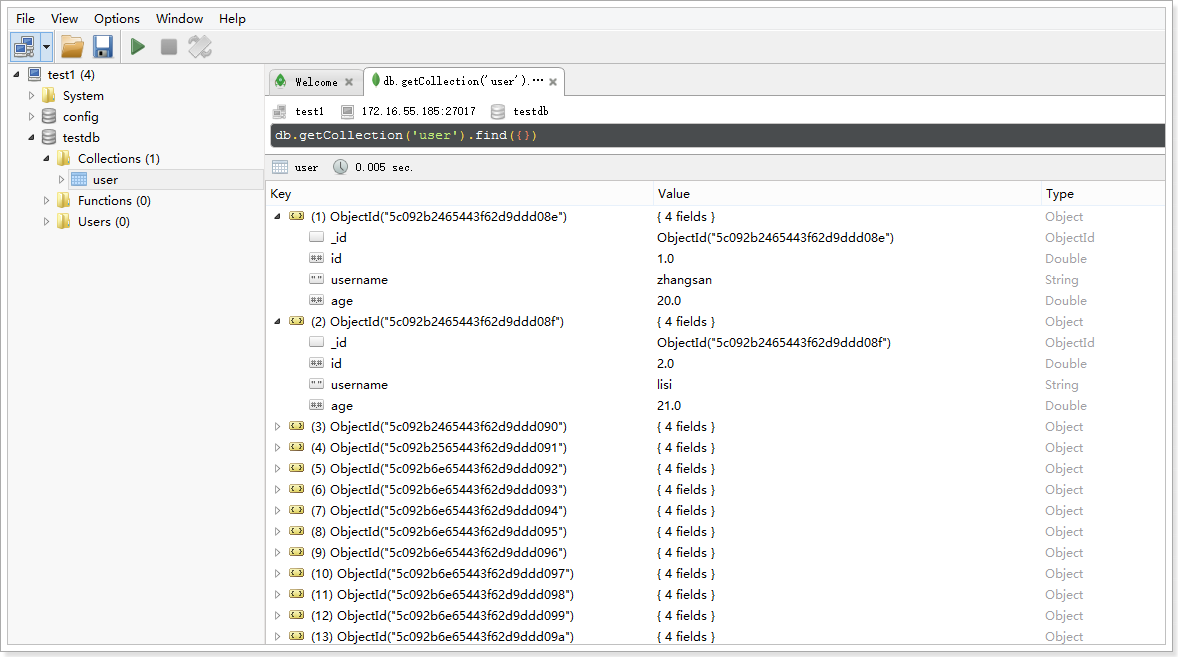
|  |
| --- |
| #测试没有使用索引  > db.user.find({username:'zhangsan'}).explain() {  "queryPlanner" : {  "plannerVersion" : 1,  "namespace" : "testdb.user",  "indexFilterSet" : false,  "parsedQuery" : {  "username" : {  "$eq" : "zhangsan"  } },  "winningPlan" : { "stage" : "COLLSCAN", #全表扫描 "filter" : {  "username" : {  "$eq" : "zhangsan"  } },  "direction" : "forward"  },  "rejectedPlans" : [ ]  },  "serverInfo" : {  "host" : "c493d5ff750a",  "port" : 27017,  "version" : "4.0.3",  "gitVersion" : "7ea530946fa7880364d88c8d8b6026bbc9ffa48c"  },  "ok" : 1 } |

1.6、UI客户端工具

Robo 3T是MongoDB的客户端工具，我们可以使用它来操作MongoDB。



查看数据:



2、通过JavaApi操作MongoDB

2.1、创建it-mongodb工程

pom.xml

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8"?>  <project xmlns="http://maven.apache.org/POM/4.0.0"  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0  http://maven.apache.org/xsd/maven-4.0.0.xsd">  <modelVersion>4.0.0</modelVersion>  <groupId>cn.itcast.mongodb</groupId>  <artifactId>itcast-mongodb</artifactId>  <version>1.0-SNAPSHOT</version>  <dependencies>  <dependency>  <groupId>org.mongodb</groupId>  <artifactId>mongodb-driver-sync</artifactId>  <version>3.9.1</version>  </dependency>  <dependency>  <groupId>junit</groupId>  <artifactId>junit</artifactId>  <version>4.12</version>  <scope>test</scope>  </dependency>  <dependency>  <groupId>org.projectlombok</groupId>  <artifactId>lombok</artifactId>  <version>1.18.4</version>  </dependency>  </dependencies>  <build>  <plugins>  <!-- java编译插件 --> <plugin>  <groupId>org.apache.maven.plugins</groupId>  <artifactId>maven-compiler-plugin</artifactId>  <version>3.2</version>  <configuration>  <source>1.8</source>  <target>1.8</target>  <encoding>UTF-8</encoding>  </configuration>  </plugin>  </plugins>  </build>  </project> |

2.2、编写Demo

该demo中演示了，如何连接到MongoDB，如何选择数据库、表，进行查询的操作。

|  |
| --- |
| package cn.itcast.mongodb;  import com.mongodb.client.\*;  import org.bson.Document;  import java.util.function.Consumer;  public class MongoDBDemo {  public static void main(String[] args) { // 建立连接  MongoClient mongoClient =  MongoClients.create("mongodb://172.16.55.185:27017");  // 选择数据库 MongoDatabase mongoDatabase = mongoClient.getDatabase("testdb");  // 选择表  MongoCollection<Document> userCollection =  mongoDatabase.getCollection("user");  // 查询数据  userCollection.find().limit(10).forEach((Consumer<? super Document>)  document -> {  System.out.println(document.toJson());  });  // 查询数据 userCollection.find().limit(10).forEach(new Consumer<Document>() {  @Override  public void accept(Document document) {  System.out.println(document.toJson());  }); // 关闭连接  mongoClient.close();  }  } |

2.3、CURD操作

|  |
| --- |
| package cn.itcast.mongodb;  import com.mongodb.client.MongoClient;  import com.mongodb.client.MongoClients;  import com.mongodb.client.MongoCollection;  import com.mongodb.client.MongoDatabase;  import com.mongodb.client.model.Projections;  import com.mongodb.client.model.Sorts;  import com.mongodb.client.model.Updates;  import com.mongodb.client.result.DeleteResult;  import com.mongodb.client.result.UpdateResult;  import org.bson.Document;  import org.junit.Before;  import org.junit.Test;  import java.util.function.Consumer;  import static com.mongodb.client.model.Filters.\*;  public class TestCRUD {  private MongoCollection<Document> mongoCollection;  @Before  public void init() { // 建立连接  MongoClient mongoClient =  MongoClients.create("mongodb://172.16.55.185:27017");  // 选择数据库 MongoDatabase mongoDatabase = mongoClient.getDatabase("testdb");  // 选择表  this.mongoCollection = mongoDatabase.getCollection("user");  }  // 查询age<=50并且id>=100的用户信息，并且按照id倒序排序，只返回id，age字段，不返回\_id字段 @Test public void testQuery() {  this.mongoCollection.find(  and(  lte("age", 50),  gte("id", 100)  ) )  .sort(Sorts.descending("id"))  .projection(  Projections.fields(  Projections.include("id","age"),  Projections.excludeId()  ) )  .forEach((Consumer<? super Document>) document -> {  System.out.println(document.toJson());  }); ;  }  @Test  public void testInsert(){  Document document = new Document("id",10001)  .append("name", "张三")  .append("age", 30); this.mongoCollection.insertOne(document); System.out.println("插入数据成功!");  this.mongoCollection.find(eq("id", 10001))  .forEach((Consumer<? super Document>) doc->{  }); }  @Test  System.out.println(doc.toJson());  public void testUpdate(){  UpdateResult updateResult = this.mongoCollection  .updateOne(eq("id", 10001), Updates.set("age", 25));  System.out.println(updateResult);  this.mongoCollection.find(eq("id", 10001))  .forEach((Consumer<? super Document>) doc->{  }  @Test  System.out.println(doc.toJson());  });  public void testDelete(){  DeleteResult deleteResult = this.mongoCollection.deleteOne(eq("id",  10001));  System.out.println(deleteResult);  }  } |

2.4、面向对象操作

前面对MongoDB的操作都是基于Document对象操作，操作略显繁琐，下面我们通过面向对象的方式进行操作。

创建Person、Address对象:

|  |
| --- |
| package cn.itcast.mongodb;  import lombok.AllArgsConstructor;  import lombok.Data;  import lombok.NoArgsConstructor;  import org.bson.types.ObjectId;  @Data  @AllArgsConstructor  @NoArgsConstructor  public class Person {  private ObjectId id;  private String name;  private int age;  private Address address;  } |

|  |
| --- |
| package cn.itcast.mongodb;  import lombok.AllArgsConstructor;  import lombok.Data;  import lombok.NoArgsConstructor;  @Data  @AllArgsConstructor  @NoArgsConstructor  public class Address {  private String street;  private String city;  private String zip;  } |

编写测试用例:

|  |
| --- |
| package cn.itcast.mongodb;  import com.mongodb.MongoClientSettings;  import com.mongodb.client.MongoClient;  import com.mongodb.client.MongoClients;  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;  import org.bson.codecs.configuration.CodecRegistries;  import org.bson.codecs.configuration.CodecRegistry;  import org.bson.codecs.pojo.PojoCodecProvider;  import org.bson.types.ObjectId;  import org.junit.Before;  import org.junit.Test;  import java.util.Arrays;  import java.util.List;  import java.util.function.Consumer;  public class TestPerson {  MongoCollection<Person> personCollection;  @Before  public void init() {  //定义对象的解码注册器 CodecRegistry pojoCodecRegistry = CodecRegistries.  fromRegistries(MongoClientSettings.getDefaultCodecRegistry(),  CodecRegistries.fromProviders(PojoCodecProvider.builder().automatic(true).build())  );  }  @Test  // 建立连接 MongoClient mongoClient =  MongoClients.create("mongodb://172.16.55.185:27017");  // 选择数据库 并且 注册解码器 MongoDatabase mongoDatabase = mongoClient.getDatabase("testdb")  .withCodecRegistry(pojoCodecRegistry);  // 选择表 this.personCollection = mongoDatabase  .getCollection("person", Person.class);  public void testInsert() { Person person = new Person(ObjectId.get(), "张三", 20,  new Address("人民路", "上海市", "666666")); this.personCollection.insertOne(person); System.out.println("插入数据成功");  }  @Test  public void testInserts() { List<Person> personList = Arrays.asList(new Person(ObjectId.get(), "张三",  20, new Address("人民路", "上海市", "666666")), new Person(ObjectId.get(), "李四", 21,  new Address("北京西路", "上海 new Address("南京东路", "上海 new Address("陕西南路", "上海 new Address("南京西路", "上海  市", "666666")), 市", "666666")), 市", "666666")),  new Person(ObjectId.get(), "王五", 22,  new Person(ObjectId.get(), "赵六", 23,  new Person(ObjectId.get(), "孙七", 24, 市", "666666")));  this.personCollection.insertMany(personList);  System.out.println("插入数据成功"); }  @Test  public void testQuery() {  this.personCollection.find(Filters.eq("name",  }  @Test  "张三")) .forEach((Consumer<? super Person>) person -> {  System.out.println(person);  });  public void testUpdate() {  UpdateResult updateResult =  this.personCollection.updateMany(Filters.eq("name", "张三"), Updates.set("age", 22));  System.out.println(updateResult);  }  @Test  public void testDelete() {  DeleteResult deleteResult =  this.personCollection.deleteOne(Filters.eq("name", "张三")); System.out.println(deleteResult);  } } |

3、SpringBoot整合MongoDB

spring-data对MongoDB做了支持，使用spring-data-mongodb可以简化MongoDB的操作。 地址:https://spring.io/projects/spring-data-mongodb

|  |
| --- |
| <parent>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-parent</artifactId>  <version>2.1.0.RELEASE</version>  </parent>  <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-data-mongodb</artifactId>  </dependency>  <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-test</artifactId>  <scope>test</scope>  </dependency> |

第二步，编写application.properties配置文件

|  |
| --- |
| # Spring boot application  spring.application.name = itcast-mongodb  spring.data.mongodb.uri=mongodb://172.16.55.185:27017/testdb |

第三步，编写PersonDao

|  |
| --- |
| package cn.itcast.mongodb.dao;  import cn.itcast.mongodb.Person;  import com.mongodb.client.result.DeleteResult;  import com.mongodb.client.result.UpdateResult;  import org.springframework.beans.factory.annotation.Autowired;  import org.springframework.data.mongodb.core.MongoTemplate;  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.stereotype.Component;  import java.util.List;  @Component  public class PersonDao {  @Autowired  private MongoTemplate mongoTemplate;  public void savePerson(Person person) {  this.mongoTemplate.save(person);  }  public List<Person> queryPersonListByName(String name) {  Query query = Query.query(Criteria.where("name").is(name));  return this.mongoTemplate.find(query, Person.class);  }  public List<Person> queryPersonListByName(Integer page, Integer rows) {  Query query = new Query().limit(rows).skip((page - 1) \* rows);  return this.mongoTemplate.find(query, Person.class);  }  public UpdateResult update(Person person) {  Query query = Query.query(Criteria.where("id").is(person.getId()));  Update update = Update.update("age", person.getAge());  return this.mongoTemplate.updateFirst(query, update, Person.class);  }  public DeleteResult deleteById(String id) {  Query query = Query.query(Criteria.where("id").is(id));  return this.mongoTemplate.remove(query, Person.class);  } } |

第四步，编写启动类

|  |
| --- |
| package cn.itcast.mongodb;  import org.springframework.boot.SpringApplication;  import org.springframework.boot.autoconfigure.SpringBootApplication;  @SpringBootApplication  public class MongoApplication {  public static void main(String[] args) {  SpringApplication.run(MongoApplication.class, args);  } } |

第五步，编写单元测试

|  |
| --- |
| package cn.itcast.mongodb;  import cn.itcast.mongodb.dao.PersonDao;  import org.bson.types.ObjectId;  import org.junit.Test;  import org.junit.runner.RunWith;  import org.springframework.beans.factory.annotation.Autowired;  import org.springframework.boot.test.context.SpringBootTest;  import org.springframework.test.context.junit4.SpringRunner;  import java.util.List;  @RunWith(SpringRunner.class)  @SpringBootTest  public class TestPersonDao {  @Autowired  private PersonDao personDao;  @Test  public void testSave() { Person person = new Person(ObjectId.get(), "张三", 20,  new Address("人民路", "上海市", "666666")); this.personDao.savePerson(person);  }  @Test  public void testQuery() { List<Person> personList = this.personDao.queryPersonListByName("张三"); for (Person person : personList) {  System.out.println(person);  }  }  @Test  public void testQuery2() {  List<Person> personList = this.personDao.queryPersonListByName(2, 2);  for (Person person : personList) {  System.out.println(person);  }  }  @Test  public void testUpdate() {  Person person = new Person();  person.setId(new ObjectId("5c0956ce235e192520086736"));  person.setAge(30);  this.personDao.update(person);  }  @Test  public void testDelete() {  this.personDao.deleteById("5c09ca05235e192d8887a389");  }  } |