**【MongoDB】**

# 【主要内容】

1. MongoDB的简介
2. MongoDB的安装
3. MongoDB的可视化工具Robo3t的安装
4. MongoDB的文档数据模型
5. MongoClientCURD的操作
6. MongoTemplateCURD的操作
7. Mongo复杂查询的操作
8. MongoDB数据安全配置

# 【学习目标】

|  |  |
| --- | --- |
| 知识点 | 要求 |
| MongoDB的安装 | 掌握 |
| MongoDB的可视化工具Robo3t的安装 | 掌握 |
| MongoDB的文档数据模型 | 掌握 |
| MongoClientCURD的操作 | 掌握 |
| MongoTemplateCURD的操作 | 掌握 |
| Mongo复杂查询的操作 | 掌握 |
| MongoDB数据安全配置 | 了解 |

# MongoDB的简介

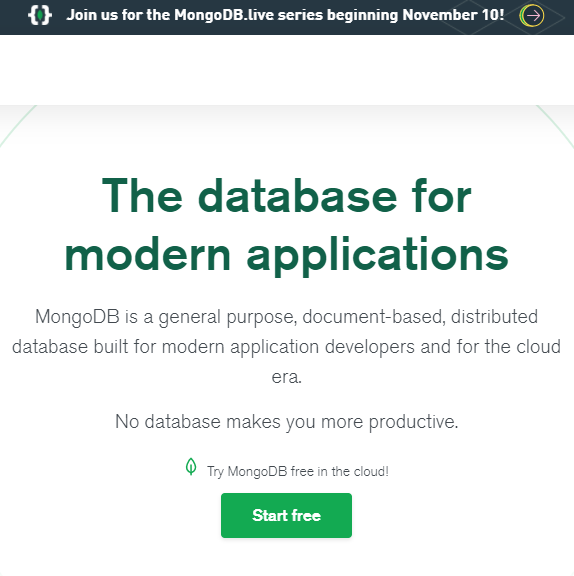
MongoDB是一个**基于分布式文件存储的数据库**。由C++语言编写。旨在为WEB应用提供可扩展的高性能数据存储解决方案。

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。Mongo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。

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。我们也常说他是**Nosql**

## MongoDB官网

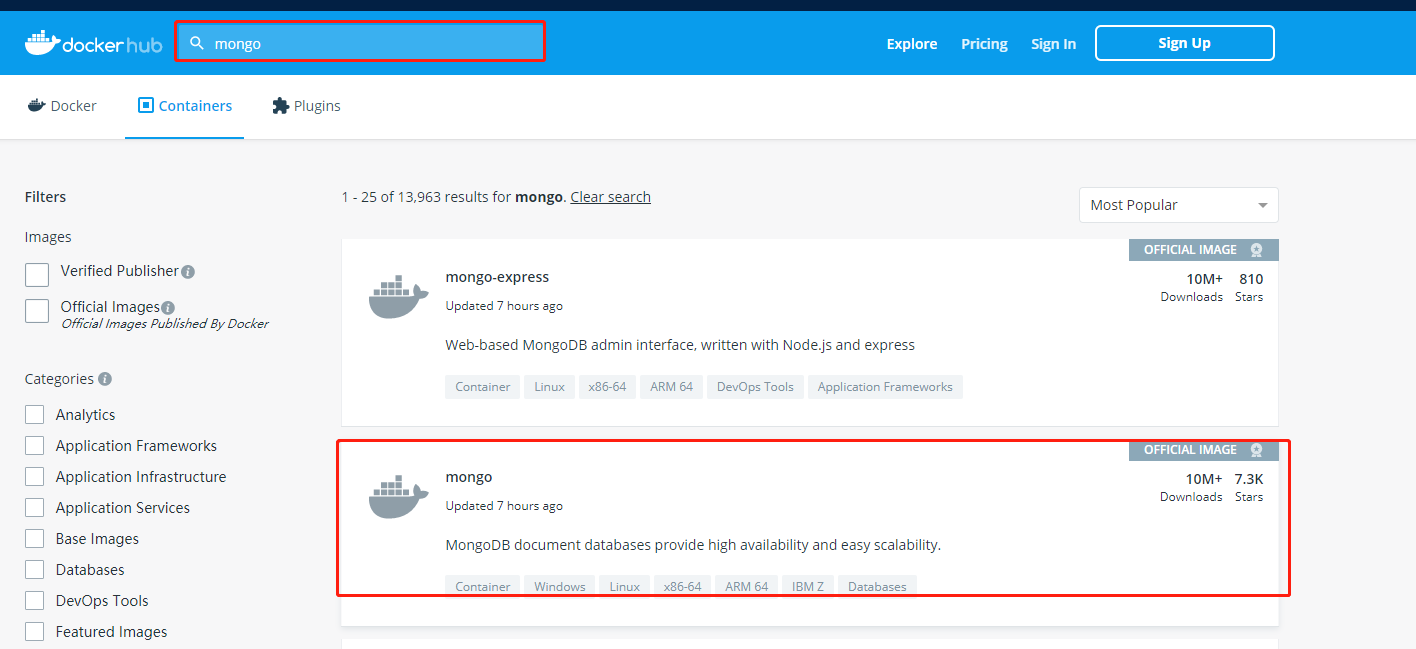
地址：<https://www.mongodb.com/>



# MongoDB的安装

## 使用docker安装MongoDB（推荐）

我们在docke hub上搜索mongo



使用docker命令下载并且运行Mongo,端口为27017，如果是ECS服务器，记住开放此端口

|  |
| --- |
| **docker run --name mgb -p 27017:27017  -d mongo** |

至此MongoDB镜像下载运行完成

## 使用安装包安装MongoDB

### 下载MongoDB的Linux安装包

先安装wget命令

|  |
| --- |
| yum -y install wget |

下载压缩文件

|  |
| --- |
| wget <https://fastdl.mongodb.org/linux/mongodb-linux-x86_64-amazon-4.4.1.tgz> |

### 在Linux上创建一个文件夹

|  |
| --- |
| mkdir /usr/local/mongo |

### 解压安装包到指定文件夹下

|  |
| --- |
| tar -zxvf mongodb-linux-x86\_64-amazon-4.4.1.tgz -C /usr/local/mongo/ |

### 进入解压后的文件夹，创建一个数据文件夹

|  |
| --- |
| cd /usr/local/mongo/mongodb-linux-x86\_64-amazon-4.4.1/ |

创建一个数据文件夹，不然启动会报错

|  |
| --- |
| mkdir -p ./data/db |

### 启动MongoDB

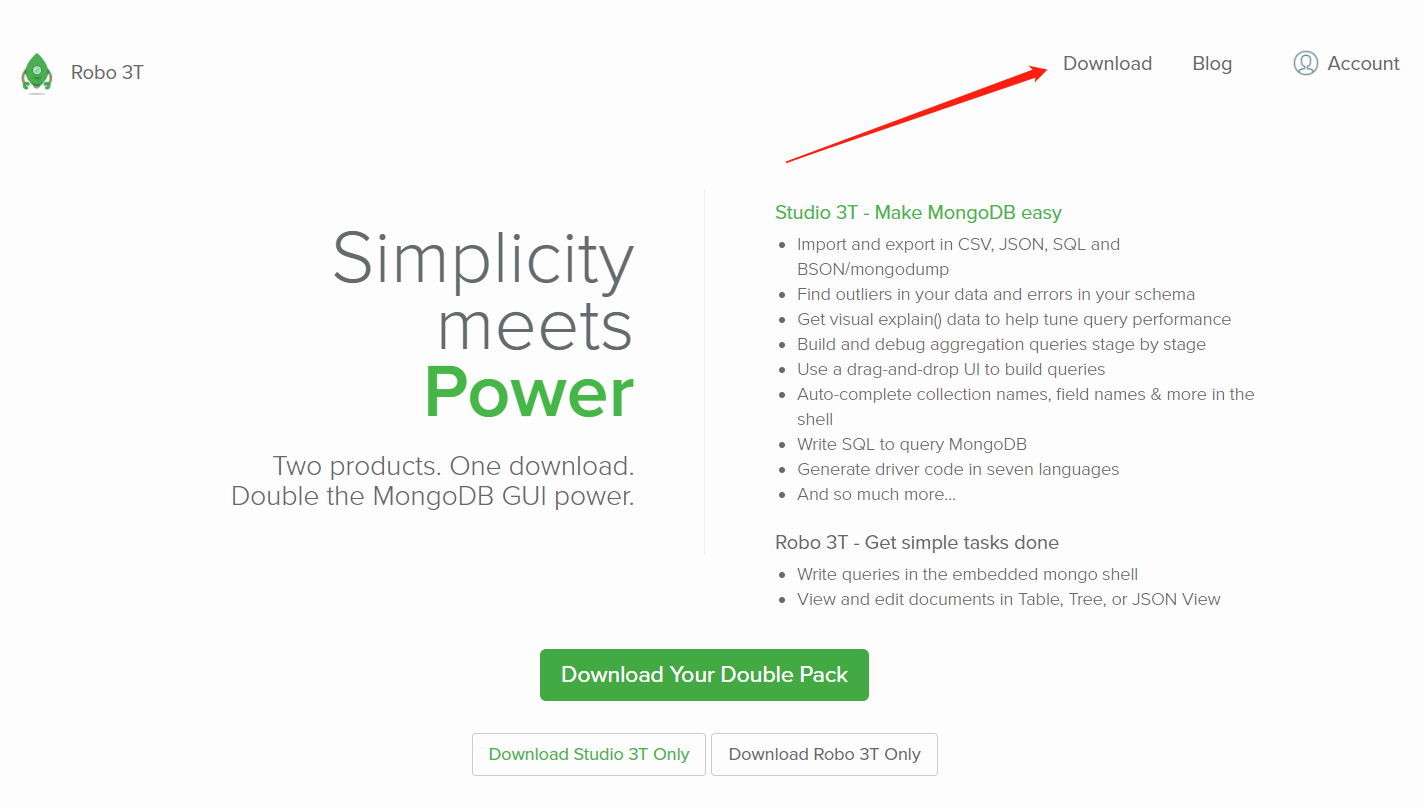
进入bin目录下 执行启动命令 使用nohup 后台运行 --dbpath指定数据库文件夹，--bind\_ip指定局域网访问

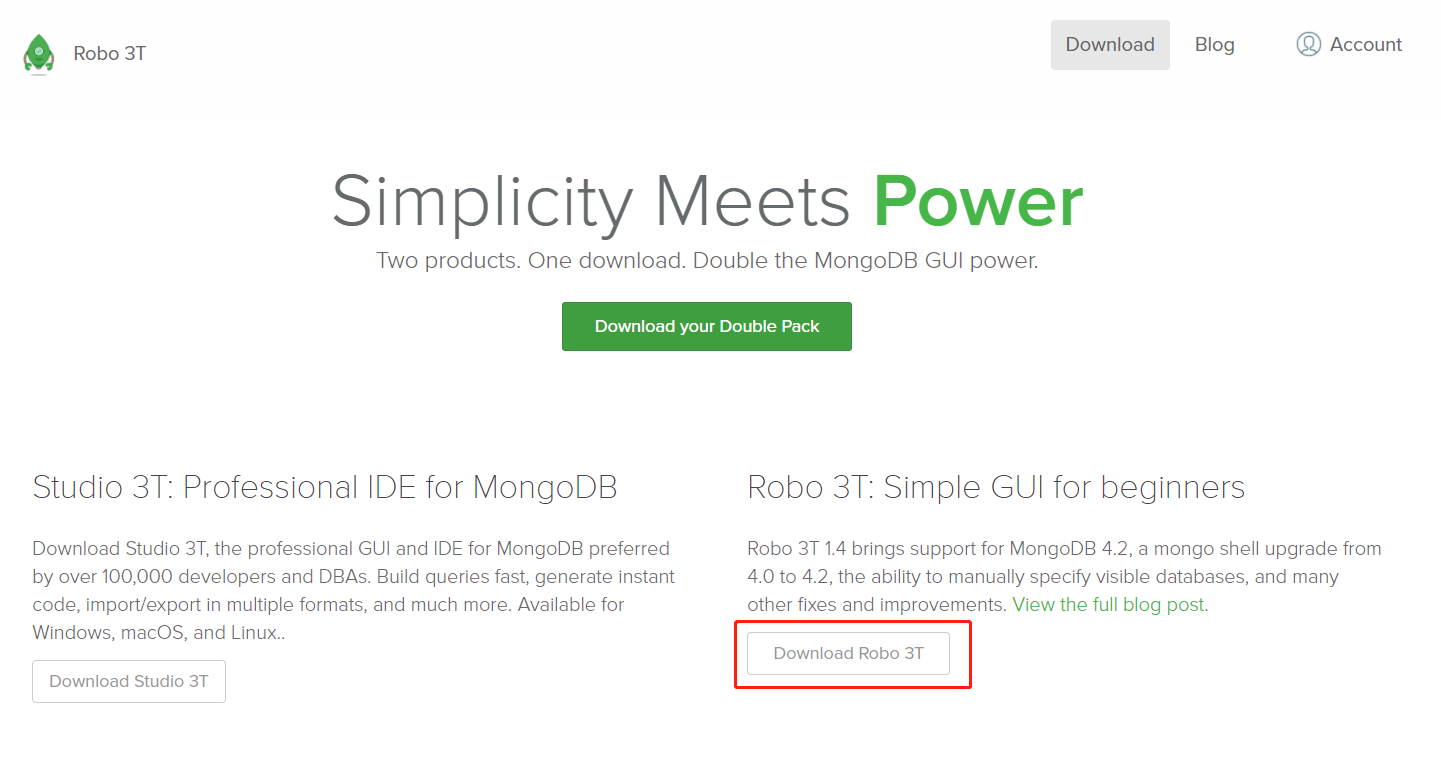
|  |
| --- |
| nohup ./mongod --dbpath=/usr/local/mongo/mongodb-linux-x86\_64-amazon-4.4.1/data/db/ --bind\_ip=0.0.0.0 & |

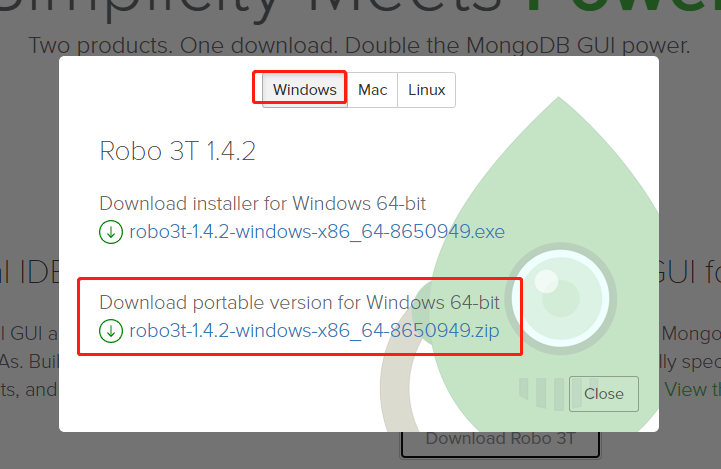
# 可视化工具Robo3t安装

## 下载Robo3t

官网： <https://robomongo.org/>







下载链接：

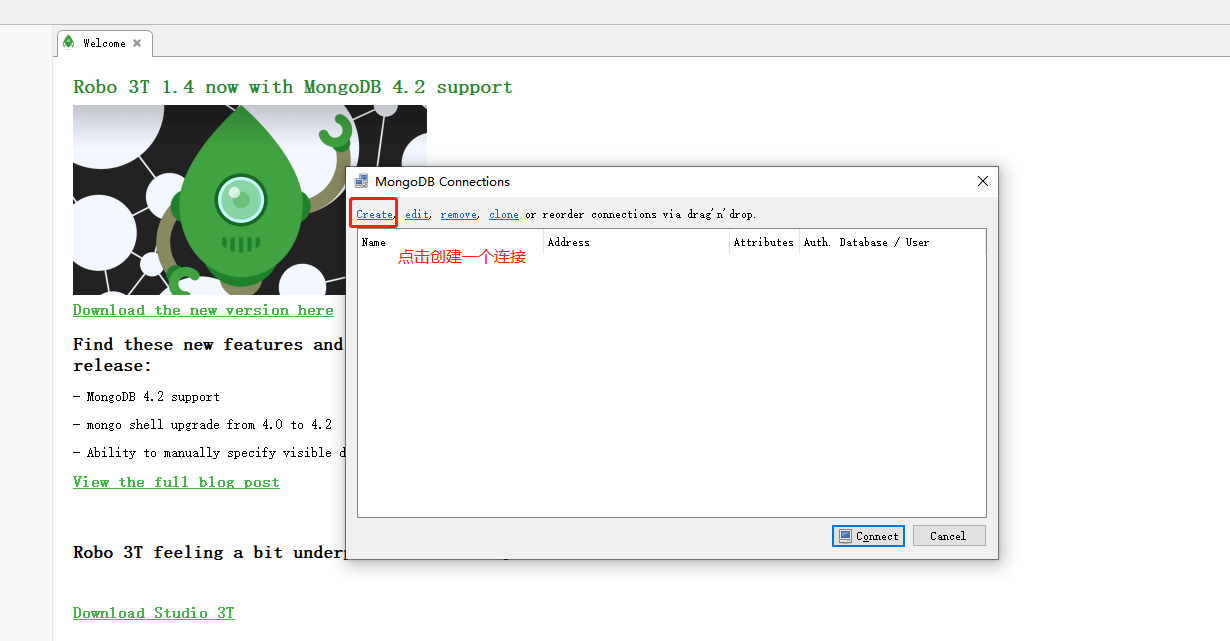
<https://download.studio3t.com/robomongo/windows/robo3t-1.4.2-windows-x86_64-8650949.zip>

## 安装Robo3t

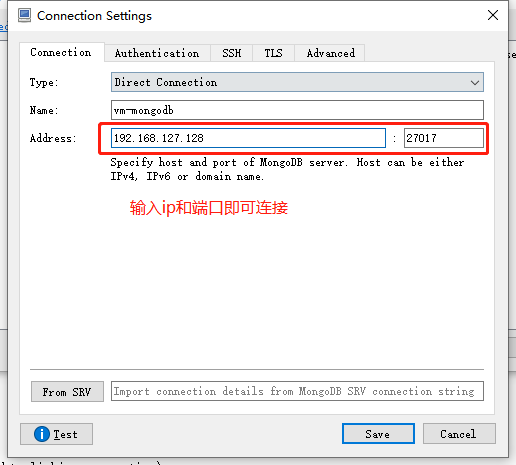
简单安装即可，一直下一步

## 使用Robo3t连接MongoDB

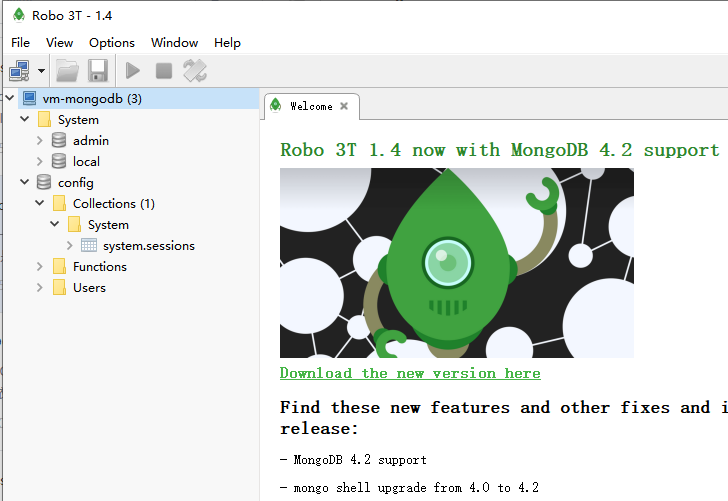
打开Robo3t可视化工具，创建一个连接



输入安装的主机地址和端口号，并且给连接一个名称，即可连接成功

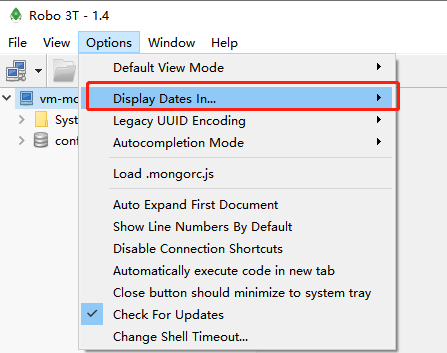


测试连接



## 注意要点Robo3t修改时区

在Options-->Dispaly Dates in... -->选择Local TimeZone



# MongoDB名词解释（重点）

MonoDB也可以算是我们第一个接触到的**文档型数据库**，我们和常用的关系型数据库mysql进行名词对比，因为mysql我们非常熟悉

|  |  |
| --- | --- |
| MongoDB | Mysql |
| 数据库（Database） | 数据库（Database） |
| 集合（collection） | 表（table） |
| 文档（document） | 行数据（row） |
| 字段（field） | 列（column） |
| 索引（index） | 索引（index） |
| 主键（\_id\_）MongoDB自动将\_id字段设为主键 | 主键（primary key） |

# 使用linux命令操作MongoDB（了解）

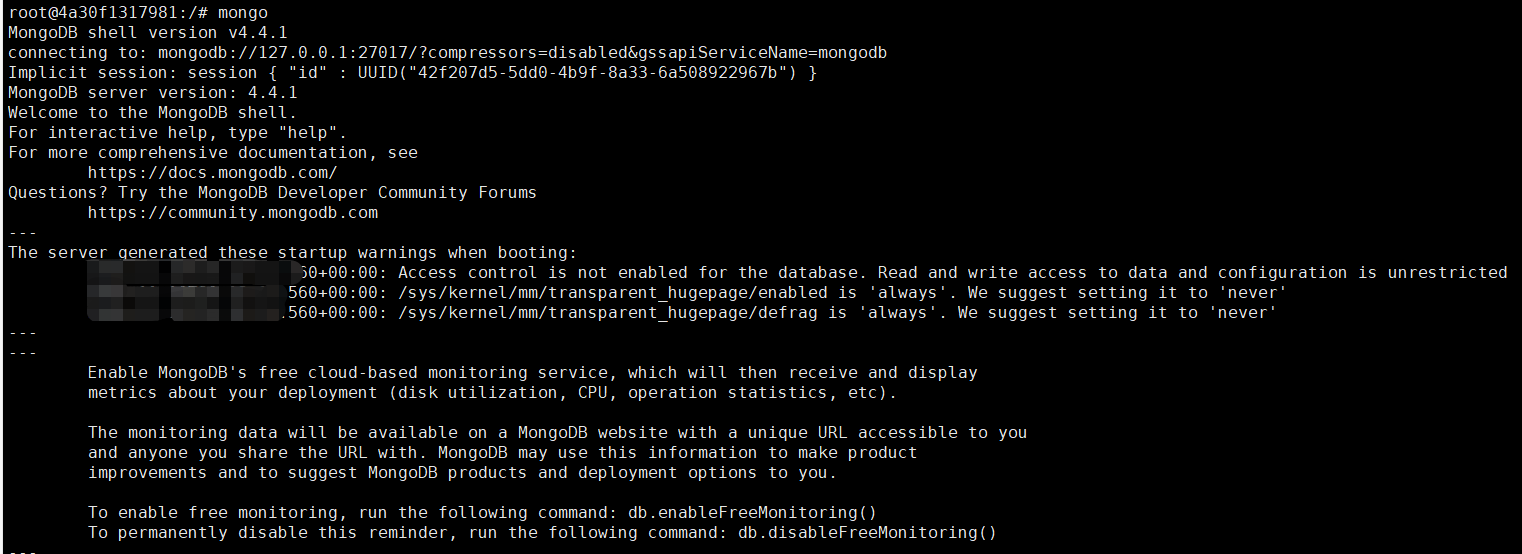
## Linux上链接MongoDB服务

因为我这里使用docker镜像运行模式，所以我先进入容器，然后在操作Mongo

|  |
| --- |
| docker exec -it mgb bash |

进入容器后，执行mongo命令

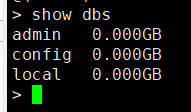
|  |
| --- |
| mongo |



## 使用命令操作数据库

### 查看数据库

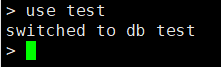
|  |
| --- |
| show dbs |



### 创建数据库

use 数据库名称

|  |
| --- |
| use test |



### 删除数据库

db.dropDatabase()

|  |
| --- |
| db.dropDatabase(); |

## 使用命令操作集合

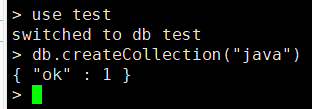
### 创建一个集合

先创建一个数据库test

|  |
| --- |
| use test |

再创建一个集合java

|  |
| --- |
| db.createCollection(“java”) |



### 查询集合

|  |
| --- |
| show collections |



### 删除集合

db.集合名称.drop()

|  |
| --- |
| db.java.drop() |



## 使用命令操作文档

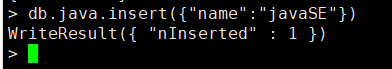
### 新增一个文档

在新增文档之前，我们需要先创建集合，因为上面我们把集合删了

|  |
| --- |
| db.createCollection("java") |

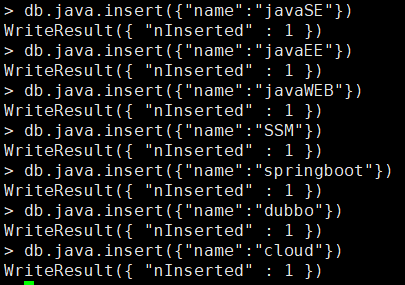
在集合中插入一条数据 **db.集合名称.insert({这里面是文档数据，json格式})**

|  |
| --- |
| db.java.insert({"name":"javaSE"}) |



### 新增多个文档

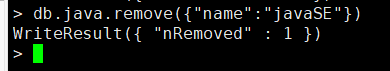
我们在java这个集合中新增多个文档 方便我们测试删除和查询，更新使用命令操作比较麻烦，我们后面直接使用java来操作



### 删除一个文档

db.集合名称.remove({传入匹配条件的json格式数据})

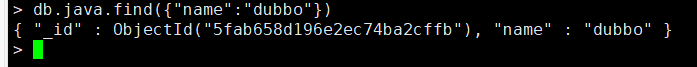
|  |
| --- |
| db.java.remove({"name":"javaSE"}) |



### 查询一个文档

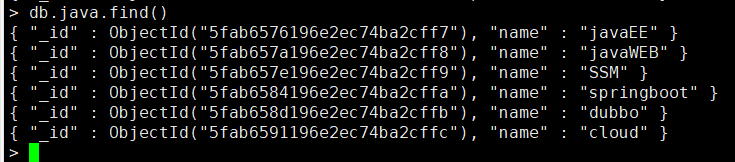
db.集合名称.find({传入匹配条件的json格式数据})

|  |
| --- |
| db.java.find({"name":"dubbo"}) |



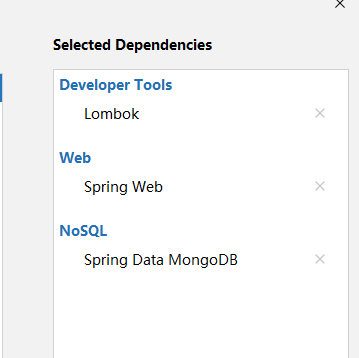
### 查询所有文档

|  |
| --- |
| db.java.find() |



# 使用SpringBoot操作MongoDB（重点）

## 创建项目选择依赖



## 修改application.yml文件

|  |
| --- |
| server:  port: 8081 spring:  data:  mongodb:  host: 192.168.127.128  port: 27017  database: sxt #指定数据库  auto-index-creation: true #开启索引集合自动创建，默认是关闭的，项目一起动就会根据实体类创建集合索引 |

## 创建一个实体类User，用来对应MongoDB的集合

|  |
| --- |
| package com.sxt.domain;  import lombok.Data; import lombok.NoArgsConstructor; import org.springframework.data.annotation.Id; import org.springframework.data.mongodb.core.index.Indexed; import org.springframework.data.mongodb.core.mapping.Document; import org.springframework.data.mongodb.core.mapping.Field; import org.springframework.data.mongodb.core.mapping.FieldType;  import java.util.Date; import java.util.List;  /\*\*  \* @Author 武汉尚学堂  \* @Document(value = "user") 标记此类被MongoDB创建成集合,名称为user  \*/ @Data @NoArgsConstructor @Document(value = "user") public class User {   /\*\*  \* 索引id  \*  \* @Id 标记为主键  \*/  @Id  private String id;   /\*\*  \* @Field 标记为MongoDB的属性字段，也可以不添加  \* @Indexed 给该字段新建索引，能提高查询速度  \*/  @Field(targetType = FieldType.*STRING*)  @Indexed  private String name;   /\*\*  \* 可以指定别名  \*/  @Field(value = "birth\_day", targetType = FieldType.*DATE\_TIME*)  private Date birth;   /\*\*  \* 这里注意 类型需要指定泛型中的类型 不要使用FieldType.ARRAY  \*/  @Field(targetType = FieldType.*STRING*)  private List<String> hobby;   @Field(targetType = FieldType.*STRING*)  private String address;   @Field(targetType = FieldType.*INT32*)  private Integer age;   @Field(targetType = FieldType.*BOOLEAN*)  private Boolean man;   @Field(targetType = FieldType.*DOUBLE*)  private Double price;  } |

## MongoDB注解说明

**@Document(value = "user")**

标注Java的类名对应的MongoDB的集合名称

**@Id、@MongoId**

使用@Id或@MongoId（低版本依赖可能需升级才有）映射MongoDB的\_id字段，若没有使用这两个注解，则字段中名称为id的字段将自动映射到\_id上。使用这两个注解可标注的字段类型为String、ObjectId（org.bson.types.ObjectId）

**@Field**

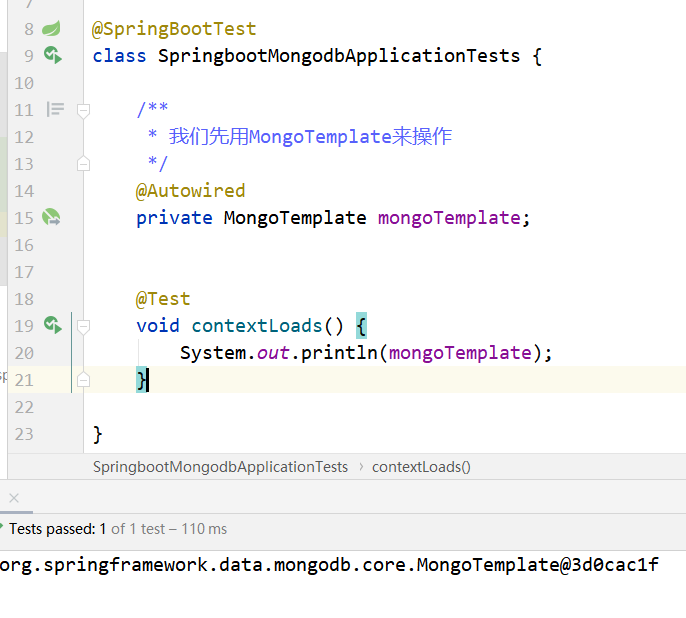
使用@Field字段，将Java类中字段名称与MongoDB集合中字段名称不一致的字段映射成一致的。如Java中使用了驼峰命名的startTs字段，想要映射成MongoDB中的start\_ts字段就可以使用@Field完成，也可以指定字段类型

**@Transient**

使用@Transient标注该字段不持久化至数据库中。

## 使用测试类测试集合的创建和删除

注入MongoTemplate，我们使用MongoTemplate来操作集合（一般用它来操作复杂的查询，简单的crud我们后面会使用继承仓库的模式）



### 新建一个集合

只要执行上面的单元测试，我们去查看Robo3t，会发现数据库和集合已经创建好了，因为我们在配置文件配置了auto-index-creation: true，当项目一起动就自动创建了

我们现在把这个配置关掉，然后我们自己手动创建（真实的情况建议开启此配置）

|  |
| --- |
| /\*\*  \* 测试集合的新建  \*  \* @throws Exception  \*/ @Test public void testCollectionCreate() throws Exception {  // 根据实体类创建一个集合  mongoTemplate.createCollection(User.class); } |

### 删除一个集合

|  |
| --- |
| /\*\*  \* 测试集合的删除  \*  \* @throws Exception  \*/ @Test public void testCollectionDel() throws Exception {  // 根据实体类删除这个集合  mongoTemplate.dropCollection(User.class); } |

## 我们使用继承仓库的方式来操作（简单的crud）

### 新建一个UserDao，继承MongoDB的仓库

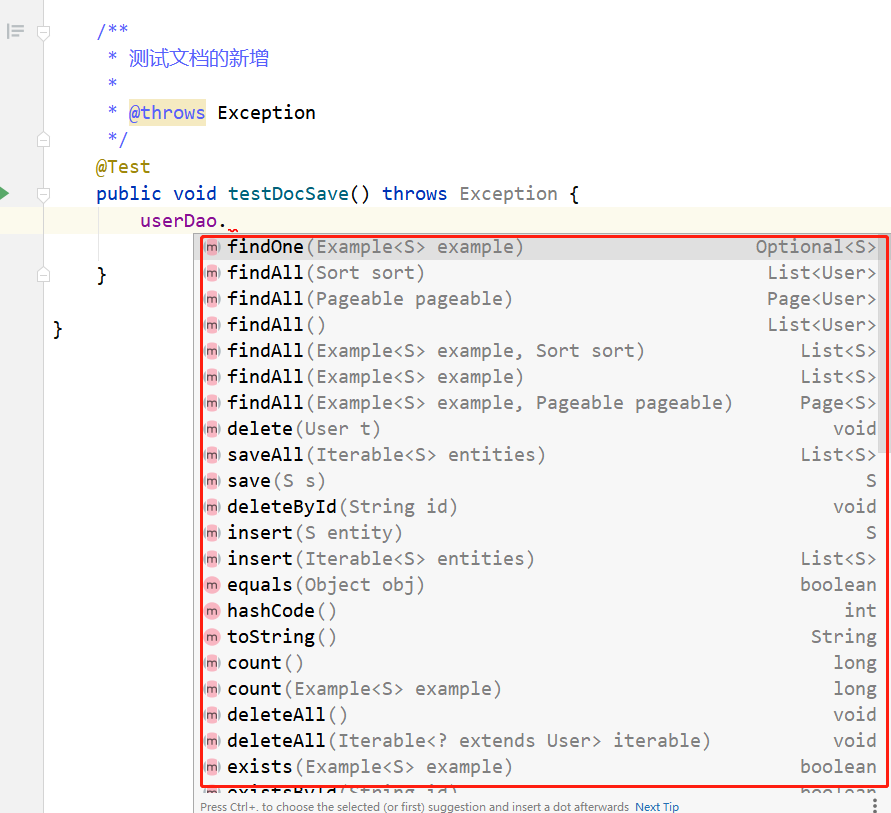
|  |
| --- |
| package com.sxt.dao;  import com.sxt.domain.User; import org.springframework.data.mongodb.repository.MongoRepository; import org.springframework.stereotype.Repository;   /\*\*  \* @Author 武汉尚学堂  \* 创建自己的接口，去继承MongoDB的仓库，从而继承了很多方法  \*/ @Repository public interface UserDao extends MongoRepository<User, String> {  } |

### 在测试类中注入UserDao

|  |
| --- |
| /\*\*  \* 我们使用userDao来进行简单crud操作  \*/ @Autowired private UserDao userDao; |

### 使用UserDao新增文档数据

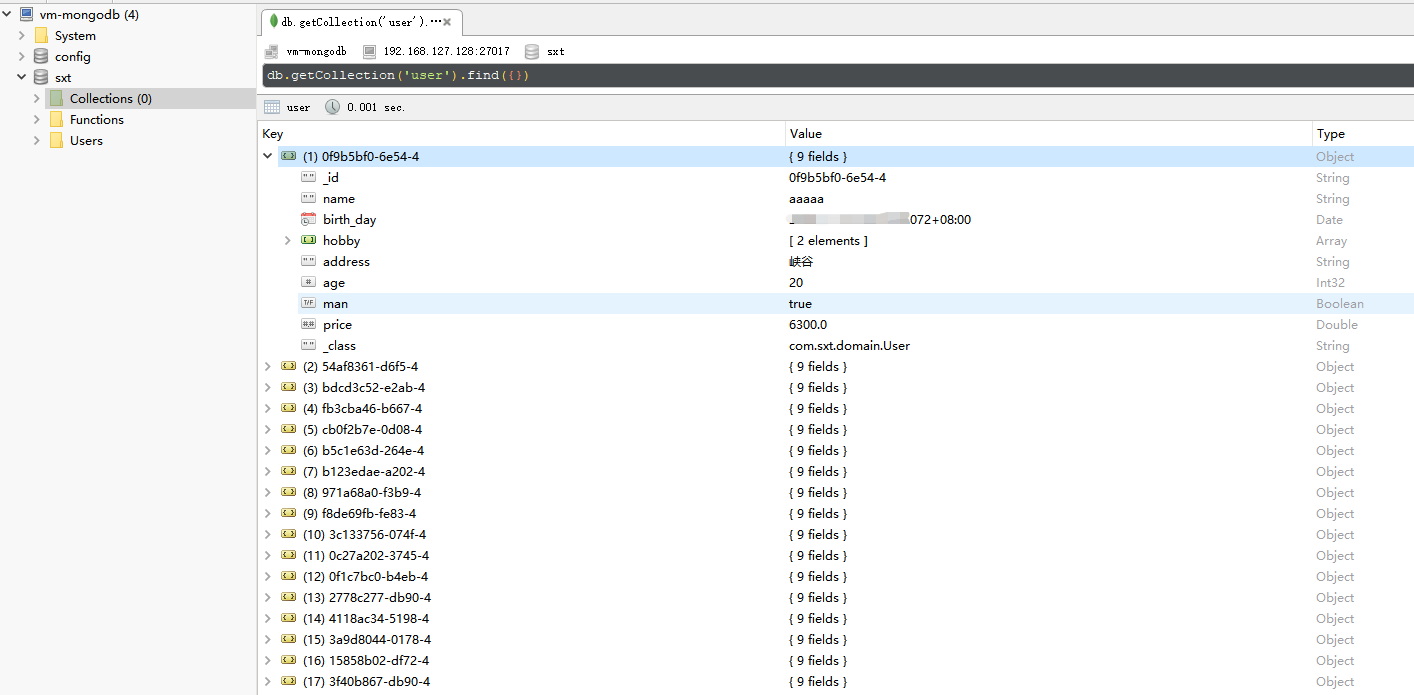
我们可以看到userDao里面有很多方法了，用起来很爽



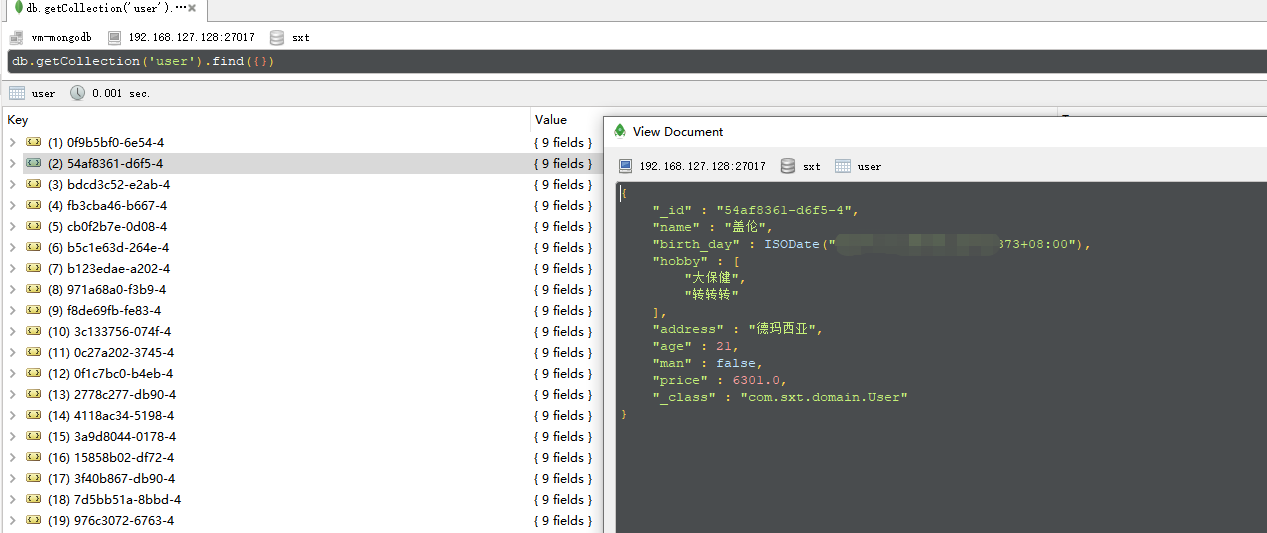
我们先新增100条数据，方便我们后面做修改删除和查询测试

|  |
| --- |
| /\*\*  \* 测试文档的新增  \*  \* @throws Exception  \*/ @Test public void testDocSave() throws Exception {  ArrayList<User> users = new ArrayList<>(200);  for (int i = 0; i < 100; i++) {  users.add(new User(  UUID.*randomUUID*().toString().substring(0, 15),  i % 2 == 0 ? "亚索" : "盖伦",  new Date(),  i % 2 == 0 ? Arrays.*asList*("吹风", "哈萨克") : Arrays.*asList*("大保健", "转转转"),  i % 2 == 0 ? "峡谷" : "德玛西亚",  20 + i,  i % 2 == 0,  (double) (6300 + i)  ));  // 睡0.3s，方便后期做区间查询  Thread.*sleep*(300);  }  userDao.saveAll(users);  System.*out*.println("插入数据成功"); } |

我们去Robo3t查看一下新增的数据



右键单击一条数据，查看json类型的文档，方便我们阅览

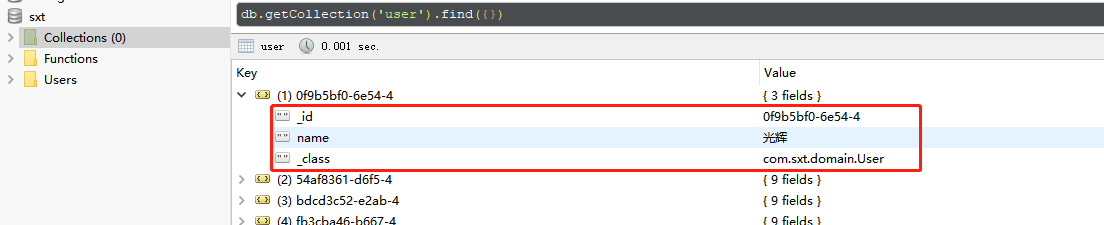


### 使用UserDao修改一条数据（不安全的修改）

userDao.save()来进行修改，根据id修改，注意是不安全的修改，会把其他属性设置为null

|  |
| --- |
| /\*\*  \* 测试修改数据（不安全的修改）  \*  \* @throws Exception  \*/ @Test public void testUpdate() throws Exception {  // 使用userDao的save方法，根据id来修改，注意是不安全的修改  User user = new User();  user.setId("0f9b5bf0-6e54-4");  user.setName("光辉");  userDao.save(user); } |

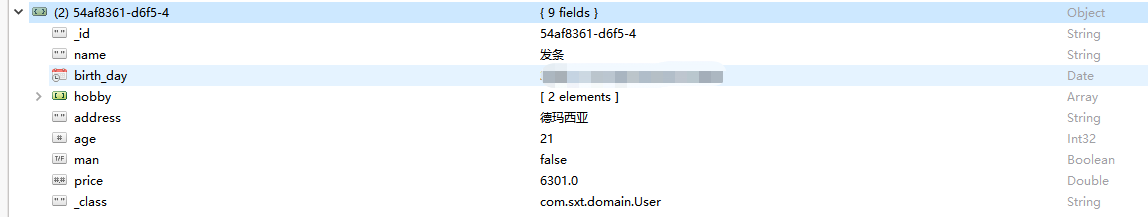
查看MongoDB中的数据，发现其他属性不见了



### 使用UserDao修改一条数据（安全的修改）

|  |
| --- |
| /\*\*  \* 安全的修改  \* 先使用query创建查询条件，然后创建更新的字段  \*  \* @throws Exception  \*/ @Test public void testUpdate2() throws Exception {  Query query = new Query();  query.addCriteria(Criteria.*where*("id").is("54af8361-d6f5-4"));  Update update = Update.*update*("name", "发条");  mongoTemplate.updateFirst(query, update, User.class); } |

查看MongoDB中的数据



### 使用UserDao删除一条数据

|  |
| --- |
| /\*\*  \* 测试删除一条文档  \*  \* @throws Exception  \*/ @Test public void testDel() throws Exception {  userDao.deleteById("0f9b5bf0-6e54-4"); } |

### 使用UserDao查询数据

|  |
| --- |
| /\*\*  \* 测试userDao查询数据  \*  \* @throws Exception  \*/ @Test public void testUserDaoFind() throws Exception {  // 根据id查询一条数据  Optional<User> optionalUser = userDao.findById("54af8361-d6f5-4");  User user = optionalUser.get();  System.*out*.println(user);  System.*out*.println("---------------------------------------");  // 查询所有  List<User> users = userDao.findAll();  System.*out*.println(users);  System.*out*.println("---------------------------------------");  // 单条件查询 根据名字查询数据  User user1 = new User();  user1.setName("发条");  Optional<User> optional = userDao.findOne(Example.*of*(user1));  User user2 = optional.get();  System.*out*.println(user2); } |

## 使用MongoTemplate进行复杂查询操作

### 多条件查询

|  |
| --- |
| /\*\*  \* 测试复杂查询之 多条件查询  \*  \* @throws Exception  \*/ @Test public void testDiff() throws Exception {  // 使用query对象。对查询条件进行组装  Query query = new Query();  // 使用Criteria创建查询对象 where,is,lt,gt等  query.addCriteria(Criteria.*where*("name").is("亚索").and("age").lt("38"));  // 执行查询  List<User> users = mongoTemplate.find(query, User.class);  System.*out*.println(users); } |

### 分页查询

|  |
| --- |
| /\*\*  \* 测试复杂查询之分页查询  \*  \* @throws Exception  \*/ @Test public void testDiff2() throws Exception {  Query query = new Query();  // 查询价格大于6385的  query.addCriteria(Criteria.*where*("price").gte(6385.00));  // 分页查询，注意从0开始  query.with(PageRequest.*of*(0,  20,  Sort.Direction.*DESC*, // 排序方式  "price") //排序字段  );  List<User> users = mongoTemplate.find(query, User.class);  System.*out*.println(users); } |

### 排序和Skip和limit查询

|  |
| --- |
| /\*\*  \* 复杂查询之skip和limit的使用  \*  \* @throws Exception  \*/ @Test public void testDiff3() throws Exception {  Query query = new Query();  // 根据时间倒序  query.with(Sort.*by*(Sort.Direction.*DESC*, "birth"));  // 跳过前三个 往后取十个  query.skip(3).limit(10);  List<User> users = mongoTemplate.find(query, User.class);  System.*out*.println(users); } |

## 在继承仓库的地方可以实现动态添加查询（了解）

注意要先写返回值，然后写find，这样才能提示





测试查询

