mongodb简介

MongoDB（来自于英文单词“Humongous”，中文含义为“庞大”）是可以应用于各种规模的企业、各个行业以及各类应用程序的开源数据库。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是一个高性能，开源，无模式的文档型数据库，是当前NoSql数据库中比较热门的一种。

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。他支持的数据结构非常松散，是类似json的bjson格式，因此可以存储比较复杂的数据类型。Mongo最大的特点是他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。

传统的关系数据库一般由数据库（database）、表（table）、记录（record）三个层次概念组成，MongoDB是由数据库（database）、集合（collection）、文档对象（document）三个层次组成。MongoDB对于关系型数据库里的表，但是集合中没有列、行和关系概念，这体现了模式自由的特点。

MongoDB中的一条记录就是一个文档，是一个数据结构，由字段和值对组成。MongoDB文档与JSON对象类似。字段的值有可能包括其它文档、数组以及文档数组。MongoDB支持OS X、Linux及Windows等操作系统，并提供了Python，PHP，Ruby，Java及C++语言的驱动程序，社区中也提供了对Erlang及.NET等平台的驱动程序。

MySQL的适合对大量或者无固定格式的数据进行存储，比如：日志、缓存等。对事物支持较弱，不适用复杂的多文档（多表）的级联查询。文中演示mongodb版本为3.4。

mongodb的增删改查

spring Boot对各种流行的数据源都进行了封装，当然也包括了mongodb,下面给大家介绍如何在spring boot中使用mongodb：

1、pom包配置

pom包里面添加spring-boot-starter-data-mongodb包引用

?

1

2

3

4

5

6

<dependencies>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-data-mongodb</artifactId>

</dependency>

</dependencies>

2、在application.properties中添加配置

?

1

spring.data.mongodb.uri=mongodb://name:pass@localhost:27017/test

多个IP集群可以采用以下配置：

?

1

spring.data.mongodb.uri=mongodb://user:pwd@ip1:port1,ip2:port2/database

2、创建数据实体

?

1

2

3

4

5

6

7

public class UserEntity implements Serializable {

private static final long serialVersionUID = -3258839839160856613L;

private Long id;

private String userName;

private String passWord;

//getter、setter省略

}

3、创建实体dao的增删改查操作

dao层实现了UserEntity对象的增删改查

?

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

23

24

25

26

27

28

29

30

31

32

33

34

35

36

37

38

39

40

41

42

43

44

45

46

@Component

public class UserDaoImpl implements UserDao {

@Autowired

private MongoTemplate mongoTemplate;

/\*\*

\* 创建对象

\* @param user

\*/

@Override

public void saveUser(UserEntity user) {

mongoTemplate.save(user);

}

/\*\*

\* 根据用户名查询对象

\* @param userName

\* @return

\*/

@Override

public UserEntity findUserByUserName(String userName) {

Query query=new Query(Criteria.where("userName").is(userName));

UserEntity user = mongoTemplate.findOne(query , UserEntity.class);

return user;

}

/\*\*

\* 更新对象

\* @param user

\*/

@Override

public void updateUser(UserEntity user) {

Query query=new Query(Criteria.where("id").is(user.getId()));

Update update= new Update().set("userName", user.getUserName()).set("passWord", user.getPassWord());

//更新查询返回结果集的第一条

mongoTemplate.updateFirst(query,update,UserEntity.class);

//更新查询返回结果集的所有

// mongoTemplate.updateMulti(query,update,UserEntity.class);

}

/\*\*

\* 删除对象

\* @param id

\*/

@Override

public void deleteUserById(Long id) {

Query query=new Query(Criteria.where("id").is(id));

mongoTemplate.remove(query,UserEntity.class);

}

}

4、开发对应的测试方法

?

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

23

24

25

26

27

28

29

30

31

@RunWith(SpringRunner.class)

@SpringBootTest

public class UserDaoTest {

@Autowired

private UserDao userDao;

@Test

public void testSaveUser() throws Exception {

UserEntity user=new UserEntity();

user.setId(2l);

user.setUserName("小明");

user.setPassWord("fffooo123");

userDao.saveUser(user);

}

@Test

public void findUserByUserName(){

UserEntity user= userDao.findUserByUserName("小明");

System.out.println("user is "+user);

}

@Test

public void updateUser(){

UserEntity user=new UserEntity();

user.setId(2l);

user.setUserName("天空");

user.setPassWord("fffxxxx");

userDao.updateUser(user);

}

@Test

public void deleteUserById(){

userDao.deleteUserById(1l);

}

}

5、查看验证结果

可以使用工具mongoVUE工具来连接后直接图形化展示查看，也可以登录服务器用命令来查看

1.登录mongos

?

1

bin/mongo -host localhost -port 20000

2、切换到test库

?

1

use test

3、查询userEntity集合数据

?

1

db.userEntity.find()

根据3查询的结果来观察测试用例的执行是否正确。

到此springboot对应mongodb的增删改查功能已经全部实现。

多数据源mongodb的使用

在多mongodb数据源的情况下，我们换种更优雅的方式来实现

1、pom包配置

添加lombok和spring-boot-autoconfigure包引用

?

1

2

3

4

5

6

7

8

9

10

11

12

13

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-data-mongodb</artifactId>

</dependency>

<dependency>

<groupId>org.projectlombok</groupId>

<artifactId>lombok</artifactId>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-autoconfigure</artifactId>

<version>RELEASE</version>

</dependency>

Lombok - 是一个可以通过简单的注解形式来帮助我们简化消除一些必须有但显得很臃肿的Java代码的工具，通过使用对应的注解，可以在编译源码的时候生成对应的方法。简单试了以下这个工具还挺好玩的，加上注解我们就不用手动写 getter\setter、构建方式类似的代码了。

?

1

spring-boot-autoconfigure - 就是spring boot的自动化配置

2、配置文件使用YAML的形式添加两条数据源，如下：

?

1

2

3

4

5

6

7

8

9

mongodb:

primary:

host: 192.168.9.60

port: 20000

database: test

secondary:

host: 192.168.9.60

port: 20000

database: test1

3、配置两个库的数据源

封装读取以mongodb开头的两个配置文件

?

1

2

3

4

5

6

@Data

@ConfigurationProperties(prefix = "mongodb")

public class MultipleMongoProperties {

private MongoProperties primary = new MongoProperties();

private MongoProperties secondary = new MongoProperties();

}

配置不同包路径下使用不同的数据源

第一个库的封装

?

1

2

3

4

5

6

@Configuration

@EnableMongoRepositories(basePackages = "com.neo.model.repository.primary",

mongoTemplateRef = PrimaryMongoConfig.MONGO\_TEMPLATE)

public class PrimaryMongoConfig {

protected static final String MONGO\_TEMPLATE = "primaryMongoTemplate";

}

第二个库的封装

?

1

2

3

4

5

6

@Configuration

@EnableMongoRepositories(basePackages = "com.neo.model.repository.secondary",

mongoTemplateRef = SecondaryMongoConfig.MONGO\_TEMPLATE)

public class SecondaryMongoConfig {

protected static final String MONGO\_TEMPLATE = "secondaryMongoTemplate";

}

读取对应的配置信息并且构造对应的MongoTemplate

?

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

23

24

25

26

@Configuration

public class MultipleMongoConfig {

@Autowired

private MultipleMongoProperties mongoProperties;

@Primary

@Bean(name = PrimaryMongoConfig.MONGO\_TEMPLATE)

public MongoTemplate primaryMongoTemplate() throws Exception {

return new MongoTemplate(primaryFactory(this.mongoProperties.getPrimary()));

}

@Bean

@Qualifier(SecondaryMongoConfig.MONGO\_TEMPLATE)

public MongoTemplate secondaryMongoTemplate() throws Exception {

return new MongoTemplate(secondaryFactory(this.mongoProperties.getSecondary()));

}

@Bean

@Primary

public MongoDbFactory primaryFactory(MongoProperties mongo) throws Exception {

return new SimpleMongoDbFactory(new MongoClient(mongo.getHost(), mongo.getPort()),

mongo.getDatabase());

}

@Bean

public MongoDbFactory secondaryFactory(MongoProperties mongo) throws Exception {

return new SimpleMongoDbFactory(new MongoClient(mongo.getHost(), mongo.getPort()),

mongo.getDatabase());

}

}

两个库的配置信息已经完成。

4、创建两个库分别对应的对象和Repository

借助lombok来构建对象

?

1

2

3

4

5

6

7

8

9

10

11

12

13

14

@Data

@AllArgsConstructor

@NoArgsConstructor

@Document(collection = "first\_mongo")

public class PrimaryMongoObject {

@Id

private String id;

private String value;

@Override

public String toString() {

return "PrimaryMongoObject{" + "id='" + id + '\'' + ", value='" + value + '\''

+ '}';

}

}

对应的Repository

?

1

2

public interface PrimaryRepository extends MongoRepository<PrimaryMongoObject, String> {

}

继承了 MongoRepository 会默认实现很多基本的增删改查，省了很多自己写dao层的代码

Secondary和上面的代码类似就不贴出来了

5、最后测试

?

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

23

24

25

26

27

28

29

@RunWith(SpringRunner.class)

@SpringBootTest

public class MuliDatabaseTest {

@Autowired

private PrimaryRepository primaryRepository;

@Autowired

private SecondaryRepository secondaryRepository;

@Test

public void TestSave() {

System.out.println("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

System.out.println("测试开始");

System.out.println("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

this.primaryRepository

.save(new PrimaryMongoObject(null, "第一个库的对象"));

this.secondaryRepository

.save(new SecondaryMongoObject(null, "第二个库的对象"));

List<PrimaryMongoObject> primaries = this.primaryRepository.findAll();

for (PrimaryMongoObject primary : primaries) {

System.out.println(primary.toString());

}

List<SecondaryMongoObject> secondaries = this.secondaryRepository.findAll();

for (SecondaryMongoObject secondary : secondaries) {

System.out.println(secondary.toString());

}

System.out.println("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

System.out.println("测试完成");

System.out.println("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

}

}

到此，mongodb多数据源的使用已经完成。

package mongodbProject1;

import java.util.List;

import mg.pojo.User;

import mg.pojo.UserList;

import mg.service.UserService;

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

import org.springframework.data.mongodb.core.MongoTemplate;

import org.springframework.data.mongodb.core.query.Criteria;

import org.springframework.data.mongodb.core.query.CriteriaDefinition;

import org.springframework.data.mongodb.core.query.Query;

import com.mongodb.CommandResult;

import com.mongodb.DBObject;

public class Test {

static ApplicationContext context=null;

static MongoTemplate mongoTemplate=null;

static{

context=new ClassPathXmlApplicationContext("applicationContext.xml");

mongoTemplate=context.getBean(MongoTemplate.class);

}

/\*\*

\* 查询UserName中等于123的

\* where(String n) is(String s)

\*/

@org.junit.Test

public void TestFind(){

Query query=Query.query(

Criteria.where("UserName").is("123"));// is相当于sql语句中的=

DBObject obj=query.getFieldsObject();

try{

List<User>userlist=mongoTemplate.find(query, User.class);

System.out.println(userlist);

}catch(Exception e){e.printStackTrace();}

}

/\*\*

\* all()方法是相当于and一样，功能是查询所有某个类型是数组或列表的字段中包含有"00"与"lzh"的记录具体详见

\* http://docs.mongodb.org/manual/reference/operator/query/all/

\* 测试数据：

\* {

Password: "xyz",

UserName: [ "school", "book", "bag", "headphone", "appliance" ],

}

\*/

@org.junit.Test

public void testAll(){

Query query=Query.query(Criteria.where("UserName").all("00","lzh"));

try{

List<UserList>userlist=mongoTemplate.find(query, UserList.class);System.out.println(userlist);

}catch(Exception e){e.printStackTrace();}

}

/\*\*

\* elemMatch()方法使用，其数据库格式如下

\* 查询的是对象数组下对象属性是否匹配相应的值

\* 数据格式如下：

\* db.inventory.find( {

qty: { $all: [

{ "$elemMatch" : { size: "M", num: { $gt: 50} } },

{ "$elemMatch" : { num : 100, color: "green" } }

] }

} )

\*/

@org.junit.Test

public void testelemMatch(){

Criteria c=new Criteria();

Query qm=new Query();

qm.addCriteria(c.elemMatch(Criteria.where("UserName").is("lzh1").and("Password").is(100)));//括号里的字符串是数据字段名称

DBObject s=qm.getQueryObject();//转换成DBObject为了更方便获取得到字符串命令

String n=s.toString();

Query query=Query.query(Criteria.where("user").all(s));

try{

List<UserList>userlist=mongoTemplate.find(query, UserList.class);System.out.println("list大小"+userlist.size()+"\n"+userlist);

}catch(Exception e){e.printStackTrace();}

}

/\*\*

\* and操作，相当于sql语句中的and

\*/

@org.junit.Test

public void testAnd(){

Query query=Query.query(Criteria.where("UserName").is("00").and("Password").is("123"));

try{

List<User>userlist=mongoTemplate.find(query, User.class);System.out.println("list大小"+userlist.size()+"\n"+userlist);

}catch(Exception e){e.printStackTrace();}

}

/\*\*

\* 该方法是使用regex()（正则表达式）方法以及or（或）操作查询数据

\* 相当于db.user.find({ "UserName" : "00", "$or" : [{ "Password" : /lz/ }] });

\*/

@org.junit.Test

public void testor(){

try{

Criteria c=Criteria.where("Password").regex("lz");//这里的正则表达式是/lzh/

Query query=Query.query(Criteria.where("UserName").is("00").orOperator(c));

List<User>userlist=mongoTemplate.find(query, User.class);

System.out.println("list大小"+userlist.size()+"\n"+userlist);

}catch(Exception e){e.printStackTrace();}

}

/\*\*使用正则表达式查询

\* Criteria.where("Password").regex(re, options);其中re,option 都是字符串,

\* option可以选值为：i,m,x,s i表示不区分大小写，m表示能使用^以及$等正则表达式来识别数据库中使用\n换行的每一行开始字符以及字符。

\* x

\* 具体原文介绍http://docs.mongodb.org/manual/reference/operator/query/regex/

\*/

@org.junit.Test

public void testRegex(){

try{

Criteria c=Criteria.where("Password").regex("lz","i");//这里的正则表达式是/lzh/

Query query=Query.query(c);

List<User>userlist=mongoTemplate.find(query, User.class);

System.out.println("list大小"+userlist.size()+"\n"+userlist);

}catch(Exception e){e.printStackTrace();}

}

}

package com.weixinplatform.wxwebserver.collection.template;

import java.util.List;

import java.util.Set;

/\*\*

\* Created by jihb on 15/5/6.

\*/

public class ViewTemplate {

private String id;

private String name;//模本名称

private String desc;//描述

private String type;//模本类型

//模板界面

List<Template> template;

public static class Template{

private String id;

private String view;//视图名称

private String image;//图片路径

public String getId() {

return id;

}

public void setId(String id) {

this.id = id;

}

public String getView() {

return view;

}

public void setView(String view) {

this.view = view;

}

public String getImage() {

return image;

}

public void setImage(String image) {

this.image = image;

}

}

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 String getDesc() {

return desc;

}

public void setDesc(String desc) {

this.desc = desc;

}

public String getType() {

return type;

}

public void setType(String type) {

this.type = type;

}

public List<Template> getTemplate() {

return template;

}

public void setTemplate(List<Template> template) {

this.template = template;

}

}

mongodb数据库中的数据如下：

{

"\_id": ObjectId("5550622dfc194b340a1feb5f"),

"name": "hsaflkjhdf",

"desc": "dfsafsdafsafad",

"type": "微企模板",

"template": [

{

"\_id": ObjectId("5550d3280a706d025d8eb5d6"),

"view": "aaaaaaaaaaa",

"image": "http://127.0.0.1:7012/staticdata/newsdata/template/images/redare/5550622dfc194b340a1feb5f/143136029221214278739681.jpg"

}

]

}

删除template子元素中的一条记录：

Update update = new Update();

update.pull("template",new BasicDBObject("id",templateId));

Query query = Query.query(Criteria.where("id").is(viewTemplateId));

return mongodbTemplate.updateFirst(query,update,ViewTemplate.class);

修改template子元素的某一条满足条件的记录：

Update update =new Update();

update.set("template.$.view", view);

update.set("template.$.image", imagePath);

Query query = Query.query(new Criteria().andOperator(Criteria.where("id").is(viewTemplateId),Criteria.where("template").elemMatch(Criteria.where("id").is(templateId))));

return mongoTemplate.updateFirst(query, update, ViewTemplate.class);

删除MongoDB数组的某一条

Query query =new Query（Criteria.where("uuid").is(uuid)）;

//若多个查询条件 可以拼接 .addCriteria()

ProjectNotice notice=mongoTemplate.findOne(query,ProjectNotice.class); //表字段映射类

notice.getNotice().forEach(t->{ //对应数组

Update update=new Update();

update.pull("notice",t); //t对应的是集合的实体

mongoTemplate.updateFirst(query,update,ProjectNotice.class); //执行指定条件的删除

})

针对mongotemplate的查询写法,可能会跟其他orm框架里面的查询写法混淆,特地记录一下.   
一. 常用查询:

1. 查询一条数据:(多用于保存时判断db中是否已有当前数据,这里 is 精确匹配,模糊匹配 使用 regex…)
2. public PageUrl getByUrl(String url) {
3. return findOne(new Query(Criteria.where("url").is(url)),PageUrl.class);
4. }

* 1
* 2
* 3

1. 查询多条数据:linkUrl.id 属于分级查询
2. public List<PageUrl> getPageUrlsByUrl(int begin, int end,String linkUrlid) {
3. Query query = new Query();
4. query.addCriteria(Criteria.where("linkUrl.id").is(linkUrlid));
5. return find(query.limit(end - begin).skip(begin), PageUrl.class);
6. }

* 1
* 2
* 3
* 4
* 5

1. 模糊查询:
2. public long getProcessLandLogsCount(List<Condition> conditions)
3. {
4. Query query = new Query();
5. if (conditions != null && conditions.size() > 0) {
6. for (Condition condition : conditions) {
7. query.addCriteria(Criteria.where(condition.getKey()).regex(".\*?\\" +condition.getValue().toString()+ ".\*"));
8. }
9. }
10. return count(query, ProcessLandLog.class);
11. }

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10

最下面，我在代码亲自实践过的模糊查询，只支持字段属性是字符串的查询，你要是查字段属性是int的模糊查询，还真没辙。

1. gte: 大于等于,lte小于等于…注意查询的时候各个字段的类型要和mongodb中数据类型一致
2. public List<ProcessLandLog> getProcessLandLogs(int begin,int end,List<Condition> conditions,String orderField,Direction direction)
3. {
4. Query query = new Query();
5. if (conditions != null && conditions.size() > 0) {
6. for (Condition condition : conditions) {
7. if(condition.getKey().equals("time")){
8. query.addCriteria(Criteria.where("time").gte(condition.getValue())); *//gte: 大于等于*
9. }else if(condition.getKey().equals("insertTime")){
10. query.addCriteria(Criteria.where("insertTime").gte(condition.getValue()));
11. }else{
12. query.addCriteria(Criteria.where(condition.getKey()).is(condition.getValue()));
13. }
14. }
15. }
16. return find(query.limit(end - begin).skip(begin).with(new Sort(new Sort.Order(direction, orderField))), ProcessLandLog.class);
17. }
19. public List<DpsLand> getDpsLandsByTime(int begin, int end, Date beginDate,Date endDate) {
20. return find(new Query(Criteria.where("updateTime").gte(beginDate).lte(endDate)).limit(end - begin).skip(begin),
21. DpsLand.class);
22. }

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13
* 14
* 15
* 16
* 17
* 18
* 19
* 20
* 21

查询字段不存在的数据

1. public List<GoodsDetail> getGoodsDetails2(int begin, int end) {
2. Query query = new Query();
3. query.addCriteria(Criteria.where("goodsSummary").not());
4. return find(query.limit(end - begin).skip(begin),GoodsDetail.class);
5. }

* 1
* 2
* 3
* 4
* 5

查询字段不为空的数据

Criteria.where("key1").ne("").ne(null)

* 1

查询或语句：a || b

1. Criteria criteria = new Criteria();
2. criteria.orOperator(Criteria.where("key1").is("0"),Criteria.where("key1").is(null));

* 1
* 2

查询且语句：a && b这里写代码片   
Criteria criteria = new Criteria();   
criteria.and(“key1”).is(false);   
criteria.and(“key2”).is(type);   
Query query = new Query(criteria);   
long totalCount = this.mongoTemplate.count(query, Xxx.class);

查询一个属性的子属性，例如：查下面数据的key2.keyA的语句

1. var s = {
2. key1: value1,
3. key2: {
4. keyA: valueA,
5. keyB: valueB
6. }
7. };
9. @Query("{'key2.keyA':?0}")
10. List<Asset> findAllBykeyA(String keyA);

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10

1. 查询数量:
2. public long getPageInfosCount(List<Condition> conditions) {
3. Query query = new Query();
4. if (conditions != null && conditions.size() > 0) {
5. for (Condition condition : conditions) {
6. query.addCriteria(Criteria.where(condition.getKey()).is(condition.getValue()));
7. }
8. }
9. return count(query, PageInfo.class);
10. }

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9

1. 更新一条数据的一个字段:
2. public WriteResult updateTime(PageUrl pageUrl) {
3. String id = pageUrl.getId();
4. return updateFirst(new Query(Criteria.where("id").is(id)),Update.update("updateTime", pageUrl.getUpdateTime()), PageUrl.class);
5. }

* 1
* 2
* 3
* 4

1. 更新一条数据的多个字段:
2. *//调用更新*
3. private void updateProcessLandLog(ProcessLandLog processLandLog,
4. int crawlResult) {
5. List<String> fields = new ArrayList<String>();
6. List<Object> values = new ArrayList<Object>();
7. fields.add("state");
8. fields.add("result");
9. fields.add("time");
10. values.add("1");
11. values.add(crawlResult);
12. values.add(Calendar.getInstance().getTime());
13. processLandLogReposity.updateProcessLandLog(processLandLog, fields,
14. values);
15. }
16. *//更新*
17. public void updateProcessLandLog(ProcessLandLog land, List<String> fields,List<Object> values) {
18. Update update = new Update();
19. int size = fields.size();
20. for(int i = 0 ; i < size; i++){
21. String field = fields.get(i);
22. Object value = values.get(i);
23. update.set(field, value);
24. }
25. updateFirst(new Query(Criteria.where("id").is(land.getId())), update,ProcessLandLog.class);
26. }

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13
* 14
* 15
* 16
* 17
* 18
* 19
* 20
* 21
* 22
* 23
* 24
* 25

1. 删除数据:
2. public void deleteObject(Class<T> clazz,String id) {
3. remove(new Query(Criteria.where("id").is(id)),clazz);
4. }

* 1
* 2
* 3

9.保存数据:

1. *//插入一条数据*
2. public void saveObject(Object obj) {
3. insert(obj);
4. }
6. *//插入多条数据*
7. public void saveObjects(List<T> objects) {
8. for(T t:objects){
9. insert(t);
10. }

！