Spring Data MongoDB 常用 CRUD 形式总结

作者：Arthur
原文链接：https://ld246.com/article/1509882133082
来源网站：链滴

作者：Arthur 联系方式：ArthurHappy@qq.com 博客：blog.arthurpapa.cn

Spring Data操作MongoDB做CRUD的时候一般会使用两种方式。本文就这两种方式做个简单的对介绍，并对我们在业务中经常使用到的查询做一些举例。

其中使用到的User类如下

User.java 如下

```
import org.springframework.data.annotation.*;
import org.springframework.data.mongodb.core.index.Indexed;
import org.springframework.data.mongodb.core.mapping.Document;

@Document(collection = "User")
public class User(){
    @Id
    private String _id;
    private String name;
    private Integer age;

    // 只在本文最后一部分使用
    // @CreatedDate
    // @Indexed
    // private DateTime createdAt;
    // @LastModifiedDate
    // @Indexed
    // private DateTime updatedAt;
}
```

MongoTemplate 和MongoRepository 使用介绍

1. 直接使用 MongoTemplate 形式。

首先注入MongoTemplate

```
@Autowired
MongoTemplate mongoTemplate;
```

1. insert

```
// insert用来插入一条数据
User user = new User();
user.setName("Jon");
mongoTemplate.insert(user, "user");
```

2. save

```
// save的形式在首次插入的时候和insert效果一样
User user = new User();
user.setName("Albert");
mongoTemplate.save(user, "user");
```

3. save - update

```
// save 还可以用来更新一条数据
user = mongoTemplate.findOne(
Query.query(Criteria.where("name").is("Jack")), User.class);
user.setName("Jim");
mongoTemplate.save(user, "user");
```

4. updateFirst

假设我们现在有数据如下：

```
[
    {
        "_id" : ObjectId("55b5ffa5511fee0e45ed614b"),
        "_class" : "org.baeldung.model.User",
        "name" : "Alex"
    },
    {
        "_id" : ObjectId("55b5ffa5511fee0e45ed614c"),
        "_class" : "org.baeldung.model.User",
        "name" : "Alex"
    }
]
```

我们使用updateFirst 更新数据

```
Query query = new Query();
query.addCriteria(Criteria.where("name").is("Alex"));
Update update = new Update();
update.set("name", "James");
mongoTemplate.updateFirst(query, update, User.class);
```

更新后的数据，我们发现只更新了第一条数据

```
[
    {
        "_id" : ObjectId("55b5ffa5511fee0e45ed614b"),
        "_class" : "org.baeldung.model.User",
        "name" : "James"
    },
    {
        "_id" : ObjectId("55b5ffa5511fee0e45ed614c"),
        "_class" : "org.baeldung.model.User",
        "name" : "Alex"
    }
]
```

5. updateMulti

和上一条updateFirst对应的，如果要全部更新，使用updateMuti，会将上一条例子中所有的Alex更为James

```
Query query = new Query();
query.addCriteria(Criteria.where("name").is("Alex"));
Update update = new Update();
update.set("name", "James");
mongoTemplate.updateMulti(query, update, User.class);
```

6. findAndModify

这个方法其实和上一条类似，只是在更新后会把更新后的内容返回。

```
Query query = new Query();
query.addCriteria(Criteria.where("name").is("Markus"));
Update update = new Update();
update.set("name", "Nick");
User user = mongoTemplate.findAndModify(query, update, User.class);
```

7. upsert

查找并更新，如果无法查到，则结合query和update的信息创建一条新的数据并插入。

```
Query query = new Query();
query.addCriteria(Criteria.where("name").is("Markus"));
Update update = new Update();
update.set("name", "Nick");
mongoTemplate.upsert(query, update, User.class);
```

8. remove

删除

```
mongoTemplate.remove(user, "user");
```

2. 使用 MongoRepository 形式。

第一步，创建某个collection的repository：

```
@Repository
public interface UserRepository  extends MongoRepository<User, String> {

}
```

不同于mongoTemplate，repository 只需写方法名就能指定做的操作。我们一般只需要对删除和查做一些自定义。下面就增删改查做一些介绍。

1. 插入和更新

MongoRepository中已经有两个方法，saveandinsert。和上面的mongoTemplate一样，insert用插入，save兼具插入和更新的功能。以下是MongoRepository中自带的saveandinsert方法。

```
<S extends T> List<S> save(Iterable<S> entites);
<S extends T> S insert(S entity);
<S extends T> List<S> insert(Iterable<S> entities);
```

使用用例

```
// 1. 注入
@Autowired
UserRepository userRepository;
// 2. 准备数据
User user = new User("MIKE");
User user2 = new User("ARHTUR");
List<User> users = Arrays.asList(user, user2);
// 3. insert 插入
userRepository.insert(user);
userRepository.insert(users);
// 4. save 更新
User user4save = userRepository.findUserByNameIs("ARHTUR");
userRepository.save(user4save);
```

2. 查找

查找可以使用函数名作为查询的条件，几乎所有的简单查询都能用这种方式实现，这种方式也是最简的方式。

```
// 根据name查找 只返回一条
User findUserByNameIs(String name);
// 根据name查找 返回匹配的所有数据
List<User> findUsersByNameIs(String name);
// 根据name查找，in匹配
List<User> findUsersByNameIn(List<String> name);
// 根据名字等于给定名字和年龄大于给定年龄来作为筛选条件
List<User> findUsersByNameIsAndAgeGreaterThan(String name,Integer age)
```

类似的还有很多，可以根据ide的提示来写，基本上符合我们正常的思维模式。

3. 删除

删除和查询基本类似。

```
// 删除和name匹配的一条数据
Integer removeUserByNameIs(String name);
// 删除所有和name匹配的数据
Integer removeUsersByNameIs(String name);
// 根据name查找，删除in匹配的数据
Integer removeUsersByNameIn(List<String> name);
// 根据名字等于给定名字和年龄大于给定年龄来作为筛选条件s删除
Integer removeUsersByNameIsAndAgeGreaterThan(String name,Integer age)
```

两种方式的对比

MongoTemplate在写法上比MongoRepository 更复杂一些，但是带来更多的灵活性。对于复杂的询操作，我们一般使用MongoTemplate，对于一些简单的查询我们会使用MongoRepository 。可这么理解，MongoRepository 只是作为一种对于简单查询的简便操作，而MongoTemplate才是我在做一些复杂查询时的首选。

常见业务问题做法

我们在业务中经常会有这样的情况：判断一个条件是否为空，如果为空就不使用该查询条件作为约束

另外我们也会经常使用到分页功能。

举例：

数据：

```
[
    {
        "_id" : ObjectId("55b5ffa5511fee0e45ed614b"),
        "_class" : "org.baeldung.model.User",
        "name" : "James",
        "age"  : 12
    },
    {
        "_id" : ObjectId("55b5ffa5511fee0e45ed614c"),
        "_class" : "org.baeldung.model.User",
        "name" : "Alex",
        "age"  : 13
    },
    {
        "_id" : ObjectId("55bsafa5511fee0e45ed6asf"),
        "_class" : "org.baeldung.model.User",
        "name" : "Marray",
        "age"  : 15
    }
]
```

查询要求：能根据给定的name list和age list查询，给定的如果为空则不作为查询条件。另外给出时范围作为条件之一，并提供分页功能。 其中User为文章开头的User.java。

实现：

```
// 请确保names和ages不为null或者自行添加非null判断
public List<User> findUsers(List names,List ages,Date from,Date to,Integer offset,Integer limit)

    Criteria criteria = new Criteria();
    // 非空判断
    if(!names.isEmpty())
        criteria = criteria.and("name").in(names);
    if(!ages.isEmpty())
        criteria = criteria.and("age").in(ages);
    // 时间范围
   criteria.and("createdAt").gte(from).lte(to);
   // 分页条件
   Pageable pageableRequest = new PageRequest(offset, limit);
   // 查询
   return mongoTemplate.find(new Query().addCriteria(criteria).with(pageableRequest), User.cl
ss);
}
```