链滴

SpringBoot 通用 MongoDao 封装

作者：HusenHuang
原文链接：https://ld246.com/article/1568889847552
来源网站：链滴

MongoDao

```
package com.yoyo.framework.mongo;

import com.mongodb.client.result.DeleteResult;
import com.mongodb.client.result.UpdateResult;
import com.yoyo.framework.api.RTPaging;
import com.yoyo.framework.json.JSONUtils;
import com.yoyo.framework.reflect.ReflectUtil;
import org.bson.Document;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.data.annotation.Id;
import org.springframework.data.domain.Sort;
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.Repository;
import org.springframework.util.Assert;

import java.lang.reflect.ParameterizedType;
import java.util.List;
import java.util.Objects;
import java.util.Optional;

/***
 @Author:MrHuang
 @Date: 2019/9/4 16:19
 @DESC: TODO
 @VERSION: 1.0
 ***/
@Repository
public class MongoDao<K,V> {

    @Autowired
    private MongoTemplate mongoTemplate;

    /**
     * 新增
     * @param v
     * @return
     */
    public V insert(V v) {
        return mongoTemplate.insert(v);
    }

    /**
     * 查询
     * @param id
     * @return
     */
    public V findById(K id) {
        Class<V> vClass = (Class<V>)((ParameterizedType)getClass().getGenericSuperclass()).ge
```

```
ActualTypeArguments()[1];
        return mongoTemplate.findById(id, vClass);
    }

    /**
     * 根据ID批量查询
     * @param ids
     * @return
     */
    public List<V> findByIds(List<K> ids) {
        Class<V> vClass = (Class<V>)((ParameterizedType)getClass().getGenericSuperclass()).ge
ActualTypeArguments()[1];
        String idFieldName = ReflectUtil.getFieldName(vClass, Id.class);
        Assert.notNull(idFieldName, "@Id not find");
        return mongoTemplate.find(Query.query(Criteria.where(idFieldName).in(ids)), vClass);
    }

    /**
     * 强制更新
     * @param v
     * @return
     */
    public UpdateResult updateById(V v) {
        Update update = Update.fromDocument(Document.parse(JSONUtils.object2Json(v)));
        ReflectUtil.FieldNameValue id = ReflectUtil.getFieldNameValue(v, Id.class);
        Assert.notNull(id, "@Id not find");
        return mongoTemplate.updateFirst(Query.query(Criteria.where(id.getFieldName()).is(id.g
tFieldValue())), update, v.getClass());
    }

    /**
     * 乐观锁更新
     * @param v
     * @return
     */
    public UpdateResult updateByIdWithVersion(V v) {
        ReflectUtil.FieldNameValue id = ReflectUtil.getFieldNameValue(v, Id.class);
        Assert.notNull(id, "@Id not find");
        ReflectUtil.FieldNameValue version = ReflectUtil.getFieldNameValue(v, MongoVersion.cl
ss);
        Assert.notNull(version, "@MongoVersion not find");
        Object fieldValue = Optional.ofNullable(version.getFieldValue()).orElse("0");
        Criteria criteria = Criteria.where(id.getFieldName()).is(id.getFieldValue()).and(version.getF
eldName()).is(fieldValue);
        // 版本号+1
        ReflectUtil.setFieldValue(v, version.getFieldName(), (Integer.parseInt(fieldValue.toString())
+ 1) + "");
        Update update = Update.fromDocument(Document.parse(JSONUtils.object2Json(v)));
        update.set("_class", v.getClass().getName());
        return mongoTemplate.updateFirst(Query.query(criteria), update, v.getClass());
    }

    /**
     * 物理删除
```

```
     * @param id
     * @return
     */
    public DeleteResult deleteById(K id) {
        Class<V> vClass = (Class<V>)((ParameterizedType)getClass().getGenericSuperclass()).ge
ActualTypeArguments()[1];
        String idFieldName = ReflectUtil.getFieldName(vClass, Id.class);
        Assert.notNull(idFieldName, "@Id not find");
        return mongoTemplate.remove(Query.query(Criteria.where(idFieldName).is(id)), vClass);
    }


    /**
     * 根据查询条件查找列表
     * @param criteria
     * @return
     */
    public List<V> find(Criteria criteria) {
        Class<V> vClass = (Class<V>)((ParameterizedType)getClass().getGenericSuperclass()).ge
ActualTypeArguments()[1];
        return mongoTemplate.find(Query.query(criteria), vClass);
    }

    /**
     * 根据查询条件查找列表
     * @param criteria
     * @return
     */
    public List<V> find(Criteria criteria, Sort sort, Long skip, Integer limit) {
        Class<V> vClass = (Class<V>)((ParameterizedType)getClass().getGenericSuperclass()).ge
ActualTypeArguments()[1];
        Query query = Query.query(criteria);
        if (Objects.nonNull(sort)) {
            query.with(sort);
        }
        if (Objects.nonNull(skip)) {
            query.skip(skip);
        }
        if (Objects.nonNull(limit)) {
            query.limit(limit);
        }
        return mongoTemplate.find(query, vClass);
    }

    /**
     * 根据查询条件查找条数
     * @param criteria
     * @return
     */
    public long count(Criteria criteria) {
        Class<V> vClass = (Class<V>)((ParameterizedType)getClass().getGenericSuperclass()).ge
ActualTypeArguments()[1];
        return mongoTemplate.count(Query.query(criteria), vClass);
    }
```

```
    /**
     * 根据查询条件分页查询
     * @param criteria 查询条件
     * @param sort 排序条件
     * @param pageNow 查找的页数
     * @param pageSize 每页显示大小
     */
    public RTPaging<V> paging(Criteria criteria, Sort sort, long pageNow, int pageSize) {
        long totalRecord = this.count(criteria);
        long totalPage = RTPaging.getTotalPage(totalRecord, pageSize);
        long skip = RTPaging.getSkip(pageNow, pageSize);
        List<V> recond = this.find(criteria, sort, skip, pageSize);
        return new RTPaging<V>().setPageNow(pageNow).setPageSize(pageSize)
                .setTotalRecord(totalRecord).setTotalPage(totalPage)
                .setRecord(recond);
    }


    /**
     * 更新第一条匹配到的
     * @param criteria
     * @param update
     * @return
     */
    public UpdateResult updateFirst(Criteria criteria, Update update) {
        Class<V> vClass = (Class<V>)((ParameterizedType)getClass().getGenericSuperclass()).ge
ActualTypeArguments()[1];
        return mongoTemplate.updateFirst(Query.query(criteria), update, vClass);
    }

    /**
     * 更新所有匹配到的
     * @param criteria
     * @param update
     * @return
     */
    public UpdateResult updateMulti(Criteria criteria, Update update) {
        Class<V> vClass = (Class<V>)((ParameterizedType)getClass().getGenericSuperclass()).ge
ActualTypeArguments()[1];
        return mongoTemplate.updateMulti(Query.query(criteria), update, vClass);
    }
}
```

ReflectUtil

```
package com.yoyo.framework.reflect;

import lombok.Data;
import lombok.experimental.Accessors;
import lombok.extern.slf4j.Slf4j;
import org.apache.commons.lang3.reflect.FieldUtils;
```

```
import org.springframework.util.Assert;
import org.springframework.util.CollectionUtils;

import java.io.Serializable;
import java.lang.annotation.Annotation;
import java.lang.reflect.Field;
import java.util.List;
import java.util.Objects;
import java.util.Optional;

/***
 @Author:MrHuang
 @Date: 2019/9/4 16:42
 @DESC: TODO
 @VERSION: 1.0
 ***/
@Slf4j
public class ReflectUtil {

    /**
     * 根据注解获取Field
     * @param clazz
     * @param annotationCls
     * @return
     */
    public static Field getField(Class<?> clazz,  Class<? extends Annotation> annotationCls) {
        List<Field> list = FieldUtils.getFieldsListWithAnnotation(clazz, annotationCls);
        if (!CollectionUtils.isEmpty(list)) {
            return list.get(0);
        }
        return null;
    }

    /**
     * 根据注解获取FieldValue
     * @param object
     * @param annotationCls
     * @return
     */
    public static Object getFieldValue(Object object, Class<? extends Annotation> annotationC
s)  {
        return Optional.ofNullable(getFieldNameValue(object, annotationCls)).map(FieldNameVa
ue::getFieldValue).orElse(null);
    }

    /**
     * 根据注解获取FieldName
     * @param object
     * @param annotationCls
     * @return
     */
    public static String getFieldName(Object object, Class<? extends Annotation> annotationC
s) {
        return Optional.ofNullable(getFieldNameValue(object, annotationCls)).map(FieldNameVa
```

```
ue::getFieldName).orElse(null);
    }

    public static String getFieldName(Class objectClass, Class<? extends Annotation> annotati
nCls) {
        Field field = ReflectUtil.getField(objectClass, annotationCls);
        return Objects.nonNull(field) ? field.getName() : null;
    }

    /**
     * 根据注解获取FieldNameValue
     * @param object
     * @param annotationCls
     * @return
     */
    public static FieldNameValue getFieldNameValue(Object object, Class<? extends Annotatio
> annotationCls) {
        if (Objects.isNull(object) || Objects.isNull(annotationCls)) {
            return null;
        }
        Field field = ReflectUtil.getField(object.getClass(), annotationCls);
        if (Objects.nonNull(field)) {
            field.setAccessible(true);
            try {
                return new FieldNameValue().setFieldName(field.getName()).setFieldValue(field.get
object));
            } catch (IllegalAccessException e) {
                log.error("ReflectUtil getFieldNameValue error", e);
            }
        }
        return null;
    }

    /**
     * 根据注解获取FieldNameValue
     * @param object
     * @param fieldName
     * @param newFieldValue
     * @return
     */
    public static void setFieldValue(Object object, String fieldName, Object newFieldValue) {
        Field field = FieldUtils.getField(object.getClass(), fieldName, true);
        try {
            field.set(object, newFieldValue);
        } catch (IllegalAccessException e) {
            log.error("ReflectUtil setFieldValue error", e);
        }
    }

    @Data
    @Accessors(chain = true)
    public static class FieldNameValue implements Serializable {
        private String fieldName;
```

```
        private Object fieldValue;
    }
}
```

RoleDTO

```
package com.yoyo.authority.role.pojo.dto;

import com.yoyo.framework.mongo.MongoVersion;
import lombok.Data;
import lombok.experimental.Accessors;
import org.springframework.data.annotation.Id;
import org.springframework.data.mongodb.core.mapping.Document;
import org.springframework.data.mongodb.core.mapping.Field;

import java.io.Serializable;
import java.time.LocalDateTime;
import java.util.List;

/***
 @Author:MrHuang
 @Date: 2019/9/4 10:48
 @DESC: TODO
 @VERSION: 1.0
 ***/
@Data
@Accessors(chain = true)
@Document(collection = "t_role")
public class RoleDTO implements Serializable {

    @Id
    private String rid;

    @Field
    private String name;

    @Field
    private String remark;

    @Field
    private Integer roleStatus;

    @Field
    private List<String> bindMenuId;

    @Field
    private String createTime;

    @Field
    private String updateTime;

    @MongoVersion
    private String version;
```

```
}
```

MongoVersion

```
package com.yoyo.framework.mongo;

import java.lang.annotation.ElementType;
import java.lang.annotation.Retention;
import java.lang.annotation.RetentionPolicy;
import java.lang.annotation.Target;

/***
 @Author:MrHuang
 @Date: 2019/9/5 15:02
 @DESC: TODO 乐观锁版本
 @VERSION: 1.0
 ***/
@Target({ElementType.FIELD})
@Retention(RetentionPolicy.RUNTIME)
public @interface MongoVersion {
}
```

RoleDao

```
package com.yoyo.authority.role.dao;

import com.yoyo.authority.role.pojo.dto.RoleDTO;
import com.yoyo.framework.mongo.MongoDao;
import org.springframework.stereotype.Repository;

/***
 @Author:MrHuang
 @Date: 2019/9/5 16:54
 @DESC: TODO
 @VERSION: 1.0
 ***/
@Repository
public class RoleDao extends MongoDao<String, RoleDTO> {
}
```