链滴

记录一下即将重构的项目 spring boot + restful

作者：lizhongyue248

原文链接：https://ld246.com/article/1550747899150

来源网站：链滴

真的很心累，说实话自己真的不想重构，因为自己真的很喜欢 spring data jpa，他的简洁方便再加上 dk 1.8 的特性，真的不忍心将他从我项目中剥离，但是他的多对多问题真的给我带来了太多的烦恼，己能力不足以解决这些问题，一路下来，磕磕碰碰，最终却还是不得不放弃它，又爱又恨。这篇博客记录一下到目前为止自己不太满意的的一个项目吧，他在刚才已经被mybatis完全替换，为他保留一分支。

项目地址：XIAOMING

分支地址：XIAOMING-JPA

能带给你什么

1. spring data jpa 无限查询的一些解决办法
2. spring data redis 新的配置方式
3. spring boot restful 公共部分抽取

技术选型

- 核心框架：spring boot
- 持久层：spring data jpa
- 数据库：mysql
- 安全：spring security oauth2
- 加密：jwt
- 缓存：redis

带来的烦恼

1. mysql 多对多的无限查询问题
2. redis 序列化问题

mysql 多对多的无限查询问题

就像所描述那样，sys_user 表里面有 roles 字段存放所有权限， sys_role 里面有 users 字段。如下

```
public class SysUser extends BaseEntity implements UserDetails {

    // ...

    /**
     * 当前用户的权限
     */
    @ManyToMany(fetch = FetchType.EAGER)
    @JsonIgnoreProperties(value = "users")
    @JoinTable(name = "sys_user_role",
            joinColumns = {@JoinColumn(name = "user_id", nullable = false)},
            inverseJoinColumns = {@JoinColumn(name = "role_id", nullable = false)})
    private List<SysRole> roles;

    // ...
}

public class SysRole extends BaseEntity {

    // ...

    /**
     * 当前角色的菜单
     */
    @JsonIgnoreProperties(value = "roles")
    @ManyToMany(cascade = CascadeType.MERGE, fetch = FetchType.EAGER)
    @JoinTable(name = "sys_permission_role", joinColumns = @JoinColumn(name = "role_id"),
            inverseJoinColumns = @JoinColumn(name = "permission_id"))
    private List<SysPermission> permissions = new ArrayList<>();

    /**
     * 当前角色对应的用户
     * 双向映射造成数据重复查询死循环问题
     */
    @ManyToMany(mappedBy = "roles")
    private List<SysUser> users = new ArrayList<>();

}

public class SysPermission extends BaseEntity {
    // ...

    /**
     * 菜单角色
     * 双向映射造成数据重复查询死循环问题
     */
```

```
    @ManyToMany(mappedBy = "permissions")
    private List<SysRole> roles = new ArrayList<>();
}
```

存放拥有当前角色的所有用户，然后带来的结果是，他们两一直互相无限查询，打印无数 sql 语句最后**栈溢出。**尝试过很多解决办法，大概有如下几种：

- @JsonIgnore 注解，但是在数据库查询出来的时候会忽略掉此字段，所以不可行。
- @JsonIgnoreProperties 注解，奇怪的是时而有效时而无效。
- @Proxy(lazy = false) 注解，无效
- fetch = FetchType.EAGER 属性，需要在配置文件中添加如下配置才有小，不然要产生一个 什么 bg 异常

```
jpa:
  properties:
    hibernate:
      enable_lazy_load_no_trans: true
```

但是会带来N+1 问题，查询效率有所降低，不过小项目无所谓=0=

- @ToString(exclude = {"users", "permissions"}) 同时需要生成的 tostring 方法忽略掉这些字段不然在使用时会报 LazyInitializationException ... no session 错误。

redis 序列化问题

我缓存选择的是 redis 缓存，而在将他存入的时候遇到了一个 spring data jpa 分页查询无法序列化问题，因为他没有默认的无参构造，因而我的分页查询无法使用 redis 缓存。为啥不自己写一个？懒==

redis 的 CacheManager 网上搜到的方式大多不管用，我的方式如下：

```
    @Bean
    @Override
    public CacheManager cacheManager() {
        // 配置在这里配置
        RedisCacheConfiguration redisCacheConfiguration = RedisCacheConfiguration.defaultCa
heConfig()
                .entryTtl(Duration.ofHours(12)) // 过期时间
                .prefixKeysWith(applicationProperties.getName()) // 缓存前缀
                .serializeKeysWith(RedisSerializationContext.SerializationPair.fromSerializer(keySerial
zer())) // 序列化键
                .serializeValuesWith(RedisSerializationContext.SerializationPair.fromSerializer(valueS
rializer())) // 序列化值
                .disableCachingNullValues();
        // 创建缓存管理器
        return RedisCacheManager
                .builder(RedisCacheWriter.nonLockingRedisCacheWriter(connectionFactory))
                .cacheDefaults(redisCacheConfiguration)
                .transactionAware()
                .build();
    }
```

自己也写了 gson 和 FastJson 的序列化，有兴趣的可以看看 github 项目的 RedisConfig

然而真正让我放弃 spring data jpa 的原因，其实是因为在我前几天修改后，尝试查询，第一次查询功并存入redis，然后再次查询他就报序列化错误，我尝试解决了三天，实在找不到解决的办法了，在 stackoverflow 发起提问但是依旧没有办法解决，所以只有完全放弃 spring data jpa 换成 mybais试试了。

公共部分抽取

对于一个 restful 风格的项目，他的 controller、service、repository 层都是有公共的部分的，如果抽取，需要写很多重复的代码，作为一个合(zhuang)格(bi)的 JAVA 程序员，肯定是不容许他的存在更何况还会带来一处修改处处修改的尴尬，所以对他进行了公共部分抽取。

BaseEntity

提取实体类的公共字段

```
package cn.echocow.xiaoming.base;

import com.fasterxml.jackson.databind.annotation.JsonDeserialize;
import com.fasterxml.jackson.databind.annotation.JsonSerialize;
import com.fasterxml.jackson.datatype.jsr310.deser.LocalDateTimeDeserializer;
import com.fasterxml.jackson.datatype.jsr310.ser.LocalDateTimeSerializer;
import lombok.Getter;
import lombok.Setter;
import org.springframework.data.annotation.CreatedBy;
import org.springframework.data.annotation.CreatedDate;
import org.springframework.data.annotation.LastModifiedBy;
import org.springframework.data.annotation.LastModifiedDate;
import org.springframework.data.jpa.domain.support.AuditingEntityListener;

import javax.persistence.*;
import java.io.Serializable;
import java.time.LocalDateTime;

/**
 * 实体类基类
 *
 * @author Echo
 * @version 1.0
 * @date 2019-02-02 22:03
 */
@Getter
@Setter
@MappedSuperclass // 重点
@EntityListeners(AuditingEntityListener.class) // 自动填充创建、修改时间和创建、修改用户
public abstract class BaseEntity implements Serializable {

    /**
     * id 主键
     */
    @Id
    @Column(name = "id", nullable = false)
```

```
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    /**
     * 排序
     */
    @OrderColumn(name = "sort")
    private Integer sort;

    /**
     * 创建时间
     */
    @JsonDeserialize(using = LocalDateTimeDeserializer.class)
    @JsonSerialize(using = LocalDateTimeSerializer.class)
    @CreatedDate
    @Column(name = "create_time", nullable = false, columnDefinition = "datetime not null de
ault now() comment '创建时间'")
    private LocalDateTime createTime;

    /**
     * 创建用户
     */
    @CreatedBy
    @Column(name = "create_user")
    private String createUser;

    /**
     * 修改时间
     */
    @JsonDeserialize(using = LocalDateTimeDeserializer.class)
    @JsonSerialize(using = LocalDateTimeSerializer.class)
    @LastModifiedDate
    @Column(name = "modify_time", nullable = false, columnDefinition = "datetime not null d
fault now() comment '修改时间'")
    private LocalDateTime modifyTime;

    /**
     * 修改用户
     */
    @LastModifiedBy
    @Column(name = "modify_user")
    private String modifyUser;

    /**
     * 备注
     */
    @Column(name = "remark")
    private String remark;

}
```

BaseRepository

公共的仓库基类，一般适用于对公共字段的条件查询等。

```
package cn.echocow.xiaoming.base;

import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.data.jpa.repository.JpaSpecificationExecutor;
import org.springframework.data.jpa.repository.Modifying;
import org.springframework.data.jpa.repository.Query;
import org.springframework.data.repository.NoRepositoryBean;
import org.springframework.data.repository.query.Param;
import org.springframework.transaction.annotation.Transactional;

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

/**
 * 资源仓库基类
 *
 * @author Echo
 * @version 1.0
 * @date 2019-02-02 20:25
 */
@NoRepositoryBean
public interface BaseRepository<T, ID> extends JpaRepository<T, ID>, JpaSpecificationExecut
r<T> {

    /**
     * 批量删除
     *
     * @param ids ids
     */
    @Modifying
    @Transactional(rollbackFor = Exception.class)
    @Query("delete from #{#entityName} e where e.id in (:ids)")
    void deleteBatch(@Param("ids")List<Long> ids);
}
```

BaseService

service 公共接口

```
package cn.echocow.xiaoming.base;

import org.springframework.data.domain.Page;
import org.springframework.data.domain.Pageable;
import java.io.Serializable;
import java.util.List;

/**
 * 服务接口基类
 *
 * @author Echo
 * @version 1.0
```

```
 * @date 2019-02-02 20:25
 */
public interface BaseService<T extends BaseEntity, ID extends Serializable, R extends BaseRep
sitory<T, ID>> {

    /**
     * 通过 id 更新实体
     *
     * @param id     id
     * @param entity 实体
     * @return 更新后的实体
     */
    T update(ID id, T entity);

    /**
     * 保存实体
     *
     * @param entity 实体对象
     * @return 保存后的实体
     */
    T save(T entity);

    /**
     * 通过 id 查询
     *
     * @param id id
     * @return 实体
     */
    T findById(ID id);

    /**
     * 通过 id 删除
     *
     * @param id id
     */
    void deleteById(ID id);

    /**
     * 判断是否存在指定 id 对象
     *
     * @param id id
     * @return 结果
     */
    boolean exists(ID id);

    /**
     * 查询所有
     *
     * @return 集合
     */
    List<T> findAll();

    /**
     * 分页查询
```

```
     *
     * @param pageable 分页
     * @return 结果
     */
    Page<T> findAll(Pageable pageable);

    /**
     * 批量删除
     *
     * @param ids id 集合
     */
    void deleteBatch(List<Long> ids);
}
```

BaseServiceImpl

```
package cn.echocow.xiaoming.base.impl;

import cn.echocow.xiaoming.utils.CustomBeanUtils;
import cn.echocow.xiaoming.base.BaseEntity;
import cn.echocow.xiaoming.base.BaseRepository;
import cn.echocow.xiaoming.base.BaseService;
import cn.echocow.xiaoming.exception.ResourceNoFoundException;
import org.springframework.beans.BeanUtils;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.cache.annotation.CacheConfig;
import org.springframework.cache.annotation.CacheEvict;
import org.springframework.cache.annotation.CachePut;
import org.springframework.cache.annotation.Cacheable;
import org.springframework.data.domain.Page;
import org.springframework.data.domain.Pageable;
import org.springframework.util.Assert;

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

/**
 * 服务实现基类
 *
 * @author Echo
 * @version 1.0
 * @date 2019-02-02 20:26
 */
@CacheConfig(cacheNames = {"baseService"}, keyGenerator = "cacheKeyGenerator")
public abstract class BaseServiceImpl<T extends BaseEntity, ID extends Serializable, R extends
BaseRepository<T, ID>> implements BaseService<T, ID, R> {

    @Autowired
    protected R baseRepository;

    @Override
    @CachePut
    public T update(ID id, T entity) {
```

```
        T exist = baseRepository.findById(id).orElseThrow(() ->
                new ResourceNoFoundException(String.format("the resource by id %s not found!",
d)));
        BeanUtils.copyProperties(entity, exist, CustomBeanUtils.getNullPropertyNames(entity));
        return exist;
    }

    @Override
    @CacheEvict
    public T save(T entity) {
        return baseRepository.save(entity);
    }

    @Override
    @Cacheable
    public T findById(ID id) {
        return baseRepository.findById(id).orElseThrow(() ->
                new ResourceNoFoundException(String.format("the resource by id %s not found!",
d))
        );
    }

    @Override
    @CacheEvict(allEntries = true)
    public void deleteById(ID id) {
        if (!baseRepository.existsById(id)) {
            throw new ResourceNoFoundException(String.format("the resource by id %s not foun
!", id));
        }
        baseRepository.deleteById(id);
    }

    @Override
    @Cacheable
    public boolean exists(ID id) {
        return baseRepository.existsById(id);
    }

    @Override
    @Cacheable
    public List<T> findAll() {
        return baseRepository.findAll();
    }

    @Override
    public Page<T> findAll(Pageable pageable) {
        return baseRepository.findAll(pageable);
    }

    @Override
    @CacheEvict(allEntries = true)
    public void deleteBatch(List<Long> ids) {
        Assert.notNull(ids, "ids can not is null!");
        baseRepository.deleteBatch(ids);
```

```
    }
}
```

BaseController

最重要的，restful 风格基类 controller

RestResource 是对单个资源的封装，使用 spring boot hateoas 生成对应的 hateoas

RestResources 是对多个资源集合的封装，使用 spring boot hateoas 生成对应的 hateoas

```
package cn.echocow.xiaoming.base;

import cn.echocow.xiaoming.exception.InvalidRequestException;
import cn.echocow.xiaoming.resource.ApplicationResource;
import cn.echocow.xiaoming.resource.PageSimple;
import cn.echocow.xiaoming.resource.RestResource;
import cn.echocow.xiaoming.resource.RestResources;
import cn.echocow.xiaoming.resource.annotation.PageResult;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.data.domain.Page;
import org.springframework.data.domain.PageRequest;
import org.springframework.hateoas.Resources;
import org.springframework.http.HttpEntity;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.validation.BindingResult;
import org.springframework.web.bind.annotation.*;

import javax.validation.Valid;
import java.util.List;
import java.util.stream.Collectors;

/**
 * @author Echo
 * @version 1.0
 * @date 2019-02-03 21:43
 */
public abstract class BaseController<T extends BaseEntity, S extends BaseService> {

    @Autowired
    private S baseService;

    /**
     * 获取控制器，通过反射添加 rest hateoas
     *
     * @return 控制器
     */
    public abstract Class getControllerClass();

    /**
     * 保存一个资源
     * POST    /{entity}
```

```
     *
     * @param entity        实体
     * @param bindingResult 检验结果
     * @return http 响应
     */
    @PostMapping
    public HttpEntity<?> saveResource(@Valid @RequestBody T entity, BindingResult binding
esult) {
        if (bindingResult.hasErrors()) {
            throw new InvalidRequestException("Invalid parameter", bindingResult);
        }
        entity.setId(null);
        return new ResponseEntity<>(new RestResource<>(baseService.save(entity), getControll
rClass()), HttpStatus.CREATED);
    }

    /**
     * 删除指定 id 的资源
     * DELETE   /sysUsers/{id}
     *
     * @param id 资源 id
     * @return http 响应
     */
    @DeleteMapping("/{id}")
    public HttpEntity<?> deleteResource(@PathVariable Long id) {
        baseService.deleteById(id);
        return new ResponseEntity<>(new ApplicationResource(), HttpStatus.NO_CONTENT);
    }

    /**
     * 更新一个资源，提供当前资源的所有信息
     * PUT    /{entity}/{id}
     *
     * @param id            资源 id
     * @param entity        更新后的资源
     * @param bindingResult 参数校验
     * @return http 响应
     */
    @PutMapping("/{id}")
    public HttpEntity<?> putResource(@PathVariable Long id, @Valid @RequestBody T entity,
BindingResult bindingResult) {
        if (bindingResult.hasErrors()) {
            throw new InvalidRequestException("Invalid parameter", bindingResult);
        }
        return patchResource(id, entity);
    }

    /**
     * 更新一个资源，提供当前资源的部分信息
     * PATCH    /{entity}/{id}
     *
     * @param id     资源 id
     * @param entity 更新后的资源
     * @return http 响应
```

```
     */
    @PatchMapping("/{id}")
    public HttpEntity<?> patchResource(@PathVariable Long id, @RequestBody T entity) {
        return ResponseEntity.ok(new RestResource<>(baseService.update(id, entity), getControl
erClass()));
    }

    /**
     * 获取指定 id 的资源
     * Get    /{entity}/{id}
     *
     * @param id 资源 id
     * @return http 响应
     */
    @GetMapping("/{id}")
    public HttpEntity<?> getResource(@PathVariable Long id) {
        return ResponseEntity.ok(new RestResource<>(baseService.findById(id), getControllerCl
ss()));
    }

    /**
     * 获取所有资源/分页，可以直接使用 Pageable 来接受，忘记改了=-=
     *
     * @param page 页码
     * @param size 大小
     * @return http 响应
     */
    @GetMapping
    @PageResult
    public HttpEntity<?> getAllOrPagesResources(
            @RequestParam(required = false) Integer page,
            @RequestParam(required = false) Integer size) {
        if (page == null || size == null || page <= 0 || size <= 0) {
            List<T> all = baseService.findAll();
            return ResponseEntity.ok(new Resources<>(all.stream()
                    .map(entity -> new RestResource<>(entity, getControllerClass()))
                    .collect(Collectors.toList())));
        }
        Page<T> result = baseService.findAll(PageRequest.of(--page, size));
        RestResources<RestResource> resources = new RestResources<>(result.stream()
                .map(entity -> new RestResource<>(entity, getControllerClass()))
                .collect(Collectors.toList()));
        resources.setPage(new PageSimple(result.getSize(), result.getNumber() + 1, result.getTot
lElements(),
                result.getTotalPages(), result.hasPrevious(), result.hasNext()));
        return ResponseEntity.ok(resources);
    }

}
```

然后基础的 rest 风格就完成了。

为什么不用 spring data rest

他不能用缓存！！！他不能用缓存！！！他不能用缓存！！！我找了一段时间的资料，都没找到，难。

为了加 hateoas 真的累死我了，到后面还不满意，分页的 hateoas 我用 aop 进行的单独封装，通过加注解进行拦截再次封装，不过尝试了很多很多办法，最后只能使用字符串拼接。

```
package cn.echocow.xiaoming.aop;

import cn.echocow.xiaoming.resource.RestResources;
import cn.echocow.xiaoming.resource.annotation.PageResult;
import cn.echocow.xiaoming.resource.PageSimple;
import org.aspectj.lang.JoinPoint;
import org.aspectj.lang.annotation.*;
import org.aspectj.lang.reflect.MethodSignature;
import org.springframework.hateoas.Link;
import org.springframework.hateoas.mvc.ControllerLinkBuilder;
import org.springframework.http.ResponseEntity;
import org.springframework.stereotype.Component;

/**
 * 对于分页的集合，进行添加分页信息
 *
 * @author Echo
 * @version 1.0
 * @date 2019-02-02 15:55
 */
@Component
@Aspect
public class RestResultAop {

    @Pointcut("@annotation(cn.echocow.xiaoming.resource.annotation.PageResult)")
    public void pageResult() { }

    @AfterReturning(value = "pageResult()", returning = "result")
    public void doAfterReturningAdvice1(JoinPoint joinPoint, Object result) {
        MethodSignature signature = (MethodSignature) joinPoint.getSignature();
        PageResult annotation = signature.getMethod().getAnnotation(PageResult.class);
        if (annotation == null) {
            return;
        }
        try {
            ResponseEntity res = (ResponseEntity) result;
            if (!res.hasBody()) {
                return;
            }
            if (!(res.getBody() instanceof RestResources)) {
                return;
            }
            RestResources resources = (RestResources) res.getBody();
            if (resources == null) {
                return;
            }
            PageSimple pageInfo = resources.getPage();
            if (pageInfo == null) {
```

```
                return;
            }
            Integer size = pageInfo.getSize();
            Integer page = pageInfo.getNumber();
            // 尝试多次，只能手动封装
            String uri = ControllerLinkBuilder.linkTo(joinPoint.getTarget().getClass()).toString();
            resources.add(new Link(uri + "?page=" + page + "&size=" + size).withSelfRel());
            if (pageInfo.hasPrevioud()) {
                resources.add(new Link(uri + "?page=" + (page - 1) + "&size=" + size).withRel(Link.
EL_PREVIOUS));
            }
            if (pageInfo.hasNext()) {
                resources.add(new Link(uri + "?page=" + (page + 1) + "&size=" + size).withRel(Link
REL_NEXT));
            }
            resources.add(new Link(uri + "?page=" + 1 + "&size=" + size).withRel(Link.REL_FIRST))

            resources.add(new Link(uri + "?page=" + pageInfo.getTotalPages() + "&size=" + size)
withRel(Link.REL_LAST));
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

}
```

感受

因为前面说的一个 redis 的问题不得不放弃，自己还是太菜了。这是最后一个项目，完成了他，自己要开始考研道路，估计基本不会再去写项目了。不想留下遗憾，他的结构也是我比较满意的，不过是话，不太喜欢分层架构，跟喜欢一捅到底的架构，不过也希望这个项目不会让自己失望，加油！