**spring data jpa使用笔记**

**一、Spring data JPA简介**

Spring data JPA是Spring在ORM框架，以及JPA规范的基础上，封装的一套JPA应用框架，并提供了一整套的数据访问层解决方案。

**二、Spring data JPA的功能**

Spring data JPA提供给用户使用的，主要有以下几个接口：

.Repository：仅仅是一个标识，表明任何继承它的均为仓库接口类，方便Spring自动扫描识别   
 .CrudRepository：继承Repository，实现了一组CRUD相关的方法   
 .PagingAndSortingRepository：继承CrudRepository，实现了一组分页排序相关的方法   
 .JpaRepository：继承PagingAndSortingRepository，实现一组JPA规范相关的方法   
 .JpaSpecificationExecutor：比较特殊，不属于Repository体系，实现一组JPA Criteria查询相关的方法。

**二、使用方法**

**1.定义一个IbaseDao，来继承PagingAndSortingRepository和JpaSpecificationExecutor，泛型T,PK分别是实体类的类型和主键的类型；**

@NoRepositoryBean

**public** **interface** IBaseDAO<T, PK **extends** Serializable>

**extends** PagingAndSortingRepository<T, PK>, JpaSpecificationExecutor<T> {

T **findByUuid**(PK id);

}

**2.根据模块，每个模块定义一个dao，继承IBaseDAO，同时传入此模块的类型；**

**public** **interface** AccidentInfoDao **extends** IBaseDAO<AccidentInfoEntity, String> {

}

**3.在该模块的service里，持有此dao，可以进行操作**

CrudRepository接口的方法：

1. <S **extends** T> S save(S entity);//保存
2. <S **extends** T> Iterable<S> save(Iterable<S> entities);//批量保存
3. T findOne(ID id);//根据id查询一个对象
4. **boolean** exists(ID id);//判断对象是否存在
5. Iterable<T> findAll();//查询所有的对象
6. Iterable<T> findAll(Iterable<ID> ids);//根据id列表查询所有的对象
7. **long** count();//计算对象的总个数
8. **void** delete(ID id);//根据id删除
9. **void** delete(T entity);//删除对象
10. **void** delete(Iterable<? **extends** T> entities);//批量删除
11. **void** deleteAll();//删除所有

**4.附带条件的分页排序查询**

**1）封装查询请求对象**

**public** **class** QueryRequest<T> {

**private** PageRequest pageRequest;

**private** Sort sort;

**private** Specification<T> specification;

……//省略get、set方法

}

**2）获取到Sort**

前端传递了一个参数orderby，格式如下"uuid@desc"，后端获取到后，进行解析

List<Sort.Order> orders = **new** ArrayList<Sort.Order>();

Sort.Order order = **new** Sort.Order(dir, property);

orders.add(order);

Sort sort = **new** Sort(orders);

3.获取分页信息（针对jquery datatable）

**int** pageIndex = 0;

**int** pageSize = RequestUtil.*getIntParameter*(request, *PARAM\_NAME\_PAGESIZE*, ***DEFAULT\_PARAM\_PAGESIZE***);

**if** (pageSize > 0) {

**int** startIdx = RequestUtil.*getIntParameter*(request, *PARAM\_NAME\_STARTINDEX*, 0);

pageIndex = startIdx / pageSize;

//第几页开始，每页的大小，sort

PageRequest pr = **new** PageRequest(pageIndex, pageSize, sort);

qr.setPageRequest(pr);

}

**4.设置查询条件**

qr.setSpecification(DynamicSpecifications.*buildSpecification*(searchMap, persistentClass)); qr.setSort(sort);

至此，查询请求对象封装完毕；开始查询，然后封装model返回给前端

**5.封装返回的model**

**public** **class** ActionResultModel<T> {

// recordsFiltered is the total number of records in the data set after filtering

**private** **long** recordsFiltered = 0;

// total number

**private** **long** recordsTotal = 0;

**private** **long** total = 0;

**private** **long** totalPages = 1;

**private** **long** pageNumber = 1;

**private** List<T> records;

**private** **boolean** success;

**private** String msg;

｝

**6.返回model**

@SuppressWarnings({ "unchecked", "rawtypes" })

**protected** ActionResultModel<T> **execQuery**(QueryRequest<T> qr, IService service) {

ActionResultModel<T> arm = **new** ActionResultModel<T>();

**try** {

**if** (qr.getPageRequest() != **null**) {

Page<T> data = service.findAll(qr.getSpecification(), qr.getPageRequest());

arm.setRecords(data.getContent());

arm.setTotal(data.getTotalElements());

arm.setTotalPages(data.getTotalPages());

arm.setPageNumber(data.getNumber());

} **else** {

List<T> list = service.findAll(qr.getSpecification(), qr.getSort());

arm.setRecords(list);

arm.setTotal(list.size());

arm.setTotalPages(1);

}

arm.setSuccess(**true**);

} **catch** (ServiceException e) {

arm.setSuccess(**false**);

arm.setMsg(e.getMessage());

e.printStackTrace();

}

**return** arm;

}

Spring包的方法WebUtils.getParametersStartingWith（request，String）

举个例子,比如页面上有   
<input type="text" name="p\_name" value="aileen">  
<input type="text" name="p\_age" value="12">  
<input type="text" name="p\_school" value="育红小学">  
  
提交表单后,后台:  
Map map = WebUtils.getParametersStartingWith(request, "p\_");  
得到的map 键是 (search)p\_ 后边的内容, 值是: value的内容   
这个页面提交的内容得到的map内容应该是:{name=aileen,age=12,school=育红小学}

Map filter=WebUtils.getParametersStartingWith(request, "p\_");方法

解释：参数 HttpServletRequest request

 java.long.String "p\_"

        返回类型：java.util.Map

对J2ee熟悉的朋友都知道在 request中有一个美好的方法叫做 request.getParameterMap() 返回一个Map 类型当然这个 Map 中就 包括了页面上传进来的所有的 Parameters 但是, 这  个   request.getParameterMap() 返回的 Map中还包括有其他的一些东西，所以我们并不能直接迭代就得到我们全部需要的，因为还会得到我们不需要的，所以。当我们使用WebUtils.getParametersStartingWith(request, "p\_"); 这个方法之后，便可以根据页面上传的值来取，这时很多人疑惑了。为什么呢，那么我们开始看第二个参数 我传了一个"p\_" 这是为什么呢。因为我的页面上所有的参数名都加了这个 "p\_" 前缀。那么WebUtils.getParametersStartingWith(request, "p\_");方法就根据前缀来取值并保存为一个Map.

**1.**@MappedSuperclass注解只能标准在类上：@Target({[**Java**](http://lib.csdn.net/base/java).lang.annotation.ElementType.TYPE})

**2.**标注为@MappedSuperclass的类将不是一个完整的实体类，他将不会映射到[**数据库**](http://lib.csdn.net/base/mysql)表，但是他的属性都将映射到其子类的数据库字段中。

**3.**标注为@MappedSuperclass的类不能再标注@Entity或@Table注解，无需实现序列化接口。

附录查询条件的解析

package com.yy.frame.specifications;

import java.util.Collection;

import java.util.HashMap;

import java.util.Iterator;

import java.util.List;

import java.util.Map;

import javax.persistence.criteria.CriteriaBuilder;

import javax.persistence.criteria.CriteriaBuilder.In;

import javax.persistence.criteria.CriteriaQuery;

import javax.persistence.criteria.Expression;

import javax.persistence.criteria.Path;

import javax.persistence.criteria.Predicate;

import javax.persistence.criteria.Root;

import javax.persistence.criteria.Subquery;

import org.apache.commons.lang3.StringUtils;

import org.springframework.data.jpa.domain.Specification;

import com.google.common.collect.Lists;

import com.yy.common.utils.CollectionsUtil;

import com.yy.common.utils.DateUtil;

import com.yy.modules.sys.org.OrgEntity;

public class DynamicSpecifications {

// 定义传输过来的字段，如果inOrgIds查询不为空，那么就是查询下级

private static final String ORGIDS = "org.nodepath2";

/\*\*

\* 创建动态查询条件组合.

\*/

public static <T> Specification<T> buildSpecification(Map<String, Object> searchParams, final Class<T> clazz) {

if (searchParams == null || searchParams.isEmpty())

return null;

Map<String, SearchFilter> filters = SearchFilter.parse(searchParams);

Specification<T> spec = bySearchFilter(filters.values(), clazz);

return spec;

}

public static <T> Specification<T> bySearchFilter(final Collection<SearchFilter> filters, final Class<T> clazz) {

return new Specification<T>() {

@Override

public Predicate toPredicate(Root<T> root, CriteriaQuery<?> query, CriteriaBuilder builder) {

boolean isorg = false;

String orgNodepath = "";

if (CollectionsUtil.isNotEmpty(filters)) {

List<Predicate> predicates = Lists.newArrayList();

for (SearchFilter filter : filters) {

// nested path translate, 如Task的名为"user.name"的filedName, 转换为Task.user.name属性

String[] names = StringUtils.split(filter.fieldName, ".");

if (ORGIDS.equals(names[0])) {

isorg = true;

orgNodepath = filter.value.toString();

continue;

}

Path expression = root.get(names[0]);

for (int i = 1; i < names.length; i++) {

expression = expression.get(names[i]);

}

Object value = replaceSpace(filter.value);

// date convert

if ("java.util.Date".equals(expression.getJavaType().getName())) {

value = DateUtil.strToDateOrDateTime(value.toString());

}

// logic operator

switch (filter.operator) {

case EQ:

predicates.add(builder.equal(expression, value));

break;

case LIKE:

builder.like((Expression<String>) expression, "%" + value + "%");

predicates.add(builder.like(expression, "%" + value + "%"));

break;

case LLIKE:

builder.like((Expression<String>) expression, "%" + value + "");

predicates.add(builder.like(expression, "%" + value + ""));

break;

case RLIKE:

builder.like((Expression<String>) expression, "" + value + "%");

predicates.add(builder.like(expression, "" + value + "%"));

break;

case BW:

predicates.add(builder.like(expression, value + "%"));

break;

case EW:

predicates.add(builder.like(expression, "%" + value));

break;

case GT:

predicates.add(builder.greaterThan(expression, (Comparable) value));

break;

case LT:

predicates.add(builder.lessThan(expression, (Comparable) value));

break;

case GTE:

predicates.add(builder.greaterThanOrEqualTo(expression, (Comparable) value));

break;

case LTE:

predicates.add(builder.lessThanOrEqualTo(expression, (Comparable) value));

break;

case NOTNULL:

predicates.add(builder.isNotNull(expression));

break;

case NULL:

predicates.add(builder.isNull(expression));

break;

case NE:

predicates.add(builder.notEqual(expression, value));

break;

case IN:

In in = builder.in(expression);

String[] values = ((String) value).split(",");

for (String val : values) {

in.value(replaceSpace(val));

}

predicates.add(in);

break;

case INFORINT:

In inforint = builder.in(expression);

String[] intvalues = ((String) value).split(",");

for (String val : intvalues) {

inforint.value(Integer.parseInt((replaceSpace(val))));

}

predicates.add(inforint);

break;

case NOTIN:

In in2 = builder.in(expression);

String[] values2 = ((String) value).split(",");

for (String val : values2) {

in2.value(replaceSpace(val));

}

predicates.add(builder.not(in2));

break;

case NOTINFORNOTIN:

In notinfornotint = builder.in(expression);

String[] notintvalues = ((String) value).split(",");

for (String val : notintvalues) {

notinfornotint.value(Integer.parseInt((replaceSpace(val))));

}

predicates.add(notinfornotint);

break;

case CONTAIN:

String[] values4 = ((String) value).split(",");

for (String val : values4) {

builder.like((Expression<String>) expression, "%" + val + "%");

predicates.add(builder.like(expression, "%" + val + "%"));

}

break;

case OR:

// by linjq

// or的话前端用 search\_OR\_随意字段=LIKE\_字段1#value1,EQ\_字段2#value2

// 随意字段必须为本实体的某个字段，实际上是不查这个随意字段的。只用“=号”后面的值进行or拼接

// 例子：search\_OR\_acStatus=EQ\_acStatus#yfj,EQ\_mobilePhone#15586948173

String[] values3 = ((String) value).split(",");

Map<String, Object> searchParams = new HashMap<String, Object>();

for (String val : values3) {

String[] valArr = val.split("#");

if (valArr.length != 2) {

continue;

}

String valArr1 =valArr[1];

if("NOTIN".equals(valArr[0].substring(0,"NOTIN".length())) ||

(valArr[0].length()>"NOTINFORNOTIN".length()&&"NOTINFORNOTIN".equals(valArr[0].substring(0,"NOTINFORNOTIN".length())))

|| "IN".equals(valArr[0].substring(0,"IN".length()))

|| (valArr[0].length()>"INFORINT".length()&&"INFORINT".equals(valArr[0].substring(0,"INFORINT".length())))

){

valArr1 = valArr1.replace(";", ",");

}

searchParams.put(valArr[0],valArr1);

}

Map<String, SearchFilter> filters = SearchFilter.parse(searchParams);

Predicate[] preArr = null;

if (!filters.isEmpty()) {

preArr = new Predicate[filters.size()];

int cnt = 0;

Iterator<String> it = filters.keySet().iterator();

while (it.hasNext()) {

String key = it.next();

if("NOTIN".equals(key.substring(0,5))){

key = key.replace(";", ",");

}

if("IN".equals(key.substring(0,2))){

key = key.replace(";", ",");

}

preArr[cnt] = getPath(root, filters.get(key), builder);

cnt++;

}

predicates.add(builder.or(preArr));

}

break;

}

}

if (predicates.size() > 0) {

// 加入只能查看下级组织

if (isorg) {

builder.and(predicates.toArray(new Predicate[predicates.size()]));

CriteriaQuery<Object> criteriaQuery = builder.createQuery();

Subquery<OrgEntity> subquery = criteriaQuery.subquery(OrgEntity.class);

Root subfromquery = subquery.from(OrgEntity.class);

Root fromquery = subquery.from(clazz);

subquery.select(subfromquery.get("uuid")); // field to map with main-query

// Expression<String> expression2 = (Expression<String>) "123";

// subquery.where(builder.equal(root.get("orgid"), subfromquery.get("uuid")));

Path expressionOrg = root.get("nodepath");

subquery.where(builder.like((Expression<String>) expressionOrg, orgNodepath + "%"));

return builder.and(builder.exists(subquery));

} else {

// 将所有条件用 and 联合起来

return builder.and(predicates.toArray(new Predicate[predicates.size()]));

}

}

}

// return query.getRestriction();

return builder.conjunction();

}

};

}

private static String replaceSpace(Object value) {

if (value == null) {

return null;

}

return StringUtils.trim(value.toString());

}

@SuppressWarnings({ "rawtypes", "incomplete-switch", "unchecked" })

private static <T> Predicate getPath(Root<T> root, SearchFilter filter, CriteriaBuilder builder) {

Predicate p = null;

// nested path translate, 如Task的名为"user.name"的filedName, 转换为Task.user.name属性

String[] names = StringUtils.split(filter.fieldName, ".");

Path expression = root.get(names[0]);

for (int i = 1; i < names.length; i++) {

expression = expression.get(names[i]);

}

Object value = filter.value;

switch (filter.operator) {

case EQ:

p = builder.equal(expression, value);

break;

case LIKE:

p = builder.like((Expression<String>) expression, "%" + value + "%");

break;

case BW:

p = builder.like(expression, value + "%");

break;

case EW:

p = builder.like(expression, "%" + value);

break;

case GT:

p = builder.greaterThan(expression, (Comparable) value);

break;

case LT:

p = builder.lessThan(expression, (Comparable) value);

break;

case GTE:

p = builder.greaterThanOrEqualTo(expression, (Comparable) value);

break;

case LTE:

p = builder.lessThanOrEqualTo(expression, (Comparable) value);

break;

case NOTNULL:

p = builder.isNotNull(expression);

break;

case NULL:

p = builder.isNull(expression);

break;

case NE:

p = builder.notEqual(expression, value);

break;

case IN:

In in = builder.in(expression);

String[] values = ((String) value).split(",");

for (String val : values) {

in.value(val);

}

p = in;

break;

case INFORINT:

In inforint = builder.in(expression);

String[] intvalues = ((String) value).split(",");

for (String val : intvalues) {

inforint.value(Integer.parseInt((replaceSpace(val))));

}

p = inforint;

break;

case NOTIN:

In in2 = builder.in(expression);

String[] values2 = ((String) value).split(",");

for (String val : values2) {

in2.value(val);

}

p = builder.not(in2);

break;

case NOTINFORNOTIN:

In notinfornotint = builder.in(expression);

String[] notintvalues = ((String) value).split(",");

for (String val : notintvalues) {

notinfornotint.value(Integer.parseInt((replaceSpace(val))));

}

p = builder.not(notinfornotint);

break;

}

return p;

}

}