<http://blog.csdn.net/u012706811/article/details/53218083>

## 待解决问题

带条件的快速分页 Spectial ，结合代码 刷 BLOG3文章

http://blog.csdn.net/u012706811/article/details/53218102

删除之前不要find一遍，提高效率

事务控制

视图查询

解决方案 参考这个Demo

https://github.com/irek907/spring\_data\_jpa\_xxx

## 资料：

<https://github.com/fandro/spring-data-jpa-mybatis> 一个现成的实现

<https://www.ibm.com/developerworks/cn/opensource/os-cn-spring-jpa/index.html> 没办法，刷JPA的 中文官网DEMO 说明了

https://www.petrikainulainen.net/programming/spring-framework/spring-data-jpa-tutorial-introduction/ 英文教程

## 框架特点

1. 实体类集成 BaseController<T>，一行代码解决一个通用的功能，新增 修改 删除(可以批量) ID查询，简洁优雅
2. 嵌入mybatis，可以轻松实现 复杂sql查询
3. JPA+mybatis PageHelper 基本可以实现 跨数据库(分页问题解决，特殊函数就没办法了)
4. 支持根据实体类属性生成表字段功能，简化开发。而且封装了 新增修改功能，修改字段的时候只需要维护实体类属性就可以了。
5. SpringDataJpa 支持按照方法名查询，也支持一般的分页查询，实在不想写，也可以拼sql实现 复杂查询，特别是在联级操作的时候，而不依赖mybatis

@Query(value="SELECT \* from users where id in(?1)",nativeQuery=true)  
List<Users> findByIdInQuery(List<Integer> idList);

1. 支持自定义SQL的修改、删除，所以基本上 JPA可以完成所有的 DML操作，不需要借助 Mybatis

@Modifying   
@Query(value="delete from users where id in(?1)",nativeQuery=true)   
void deleteUser(List<Integer> idList);

## 心得笔记

201705016

成功 引入 mybatis + pageHelper

开始测试 切换到 oracle

20170518

其实应该这个系统设置初始目标是一个好用的系统，而不是即好用又强大的系统。

现在无法通过 BaseDao<T> 来实现通用方法，但是可以通过entityManager，实现单独的事务控制。

也就是说 初步实现一套封装好的，简单的增删改查通用方法。复杂的联级业务还是要自己单独在自己的service里面去实现。因为entityManager操作的方法是不受Spring的申明式实务控制，但是dao方法是受实务控制的。

Mybatis+PageHelper的引入

如果复杂查询不多的话，可以封装在一个mybatisService 的类里面，根据返回类不同，返回不同的类，也可以定义多个 resultMap。

### 遇到的坑

写BaseDao extends 直接报错，不能这样关联，哎，可惜了，只能用 entityManager 来封装通用方法。

public interface BaseDao extends JpaRepository<T, Serializable>{

## 对比Hibernate

1. Hibernate支持的功能，JPA都支持
2. 轻松实现Query接口，手写HQL或者本地SQL代码，简化代码，不需要重写Query 的JAVA代码
3. 允许通过 方法名来查询且不需要去实现，灵活而简洁
4. 提供了分页 排序，且带条件的查询
5. 支持扩展 Q\_EQ Q\_LIKE

## 注意事项

### 1）BaseController的事务控制

BaseController 主要是为了实现 通用简单的方法，新增 修改 删除，因为entituMannager 不受transactional的事务控制，所以只能手动控制，所以无法参杂其他的方法进行联级的事务控制。

### 2）稍微复杂的查询

直接用mybatisService查询，其实很方便的。特别是在不需要 字段名和属性名转换的话，非常方便，直接resultType=“Users”

再结合MybatisGenerator，生成 resultMap，666

## 通用方法 BaseController

### Merge

新增或者修改

public void merge(T entity){  
 EntityManager entityManager = entityManagerFactory.createEntityManager();  
 EntityTransaction tran=entityManager.getTransaction();  
 tran.begin();  
 entityManager.merge(entity);  
 tran.commit();  
}

### delete(String ids)

支持批量删除，传入 带 ， 分隔符的ID

public int delete(String ids){  
 if(StringUtils.*isEmpty*(ids.trim())){  
 return 0;  
 }  
 EntityManager entityManager = entityManagerFactory.createEntityManager();  
 EntityTransaction tran=entityManager.getTransaction();  
 tran.begin();  
 String[] idArr=ids.split(",");  
 for(String id:idArr){  
 T entity=(T) entityManager.find(getType(), Integer.*parseInt*(id));  
 entityManager.remove(entity);  
 }  
 tran.commit();   
 return idArr.length;  
}

### findById（String id）

public T findById(String id){  
 EntityManager entityManager = entityManagerFactory.createEntityManager();  
 return entityManager.find(getType(), Integer.*parseInt*(id));  
}

## API

### 批量新增

List<User> users = new ArrayList<>();

users.add(user);

//保存多个

userRepository.save(users);

### 批量删除

userRepository.deleteInBatch(users);

### findAll

List<T> findAll();

List<T> findAll(Sort sort);

List<T> findAll(Iterable<ID> ids);

Iterable<T> findAll(Sort sort);

Page<T> findAll(Pageable pageable);

## 分页

### 根据查询条件带分页带排序

public void getMethodName(){  
 String niceName="jingli";  
 Page<Users> page=userDao.findByNiceName(niceName,new PageRequest(0, 5,Sort.Direction.*DESC*, "age"));  
}

Page<Users> findByNiceName(String niceName, Pageable pageable);

### Order 普通拼接

public Page<Users> findAllUserPage(){  
 Sort sort2=new Sort(Sort.Direction.*DESC*, "age","niceName");  
 return userDao.findAll(new PageRequest(0, 3, sort2));  
}

### 综合 order ASC DESC

// order by age desc,niceName asc   
//notice the fieldName is the Entity's attribute name instead of the column name  
public Page<Users> findAllUserPage(){  
 Sort sort = new Sort(Sort.Direction.DESC, "age")  
 .and(new Sort(Sort.Direction.ASC, "niceName"));  
 return userDao.findAll(new PageRequest(0, 3, sort));  
}

## 查询

### 支持本地SQL查询，nativeQuery=true

@Query(value="select nice\_name from users where id=?1",nativeQuery=true)  
String findByOriginSqlQuery(int id);  
  
@Query("SELECT niceName from users where id=?1")  
String findNiceNameByQueryJPA(int id);

### 2.解析方法名创建查询 find+全局修饰+By+实体的属性名称+限定词+连接词+ …(其它实体属性)+OrderBy+排序属性+排序方向

### In 查询

@Query(value="SELECT \* from users where id in(?1)",nativeQuery=true)  
List<Users> findByIdInQuery(List<Integer> idList);

### 自定义查询且封装成 实体类 或者Map

注意事项：本地查询的 话 要 as 成对应的属性名字，例如 user\_name as userName, 但是 a.password 则不需要转义 会自动赋值给 password属性

@Resource  
EntityManagerFactory entityManagerFactory;  
public void test2() {  
 String sql = "select b.name as deptName ,a.id,a.nice\_name as niceName,a.user\_name as userName,a.age from users a join dept b on a.dept\_id=b.id";  
 EntityManager entityManager = entityManagerFactory.createEntityManager();  
 javax.persistence.Query query = entityManager.createNativeQuery(sql);  
 Query queryTest = query.unwrap(org.hibernate.SQLQuery.class).setResultTransformer(Transformers.*ALIAS\_TO\_ENTITY\_MAP*);  
 List<Map> list = queryTest.list();  
  
 Query queryTest2 = query.unwrap(org.hibernate.SQLQuery.class).setResultTransformer(Transformers.*aliasToBean*(Users.class));  
 List<Users> list2 = queryTest2.list();  
}

## 删除

### 带事务的批量删除

@Modifying   
@Query(value="delete from users where id in(?1)",nativeQuery=true)   
void deleteUser(List<Integer> idList);

@Transactional  
public void test(){  
 List<Integer> idList=new ArrayList<Integer>();  
 idList.add(2);  
 idList.add(3);  
 userDao.deleteUser(idList);  
}

## 主键UUID自定义生成策略

轻松实现跨数据库了，否在 oracle Sequence mysql indentity还容易出错

如果直接插入数据库

Oracle:

select lower(sys\_guid()) from dual;

Mysql

SELECT REPLACE(UUID(),'-','')

@Id  
@GeneratedValue(generator = "idGenerator")  
@GenericGenerator(name = "idGenerator", strategy = "com.vanceinfo.javaserial.common.PrimaryKeyGeneratorCustom")  
@Column(length=32)   
 private String id;

public class PrimaryKeyGeneratorCustom extends AbstractUUIDGenerator implements Configurable {  
 public Serializable generate(SessionImplementor session, Object obj) {  
 String uuid=UUID.*randomUUID*().toString().trim().replaceAll("-", "");  
 return uuid;  
 }  
 @Override  
 public void configure(Properties properties) throws HibernateException {  
 // *TODO Administrator Auto-generated method stu* }  
}