# JsonCreator使用说明

## 原理介绍

实体转换JSON系统中最核心的功能,定义的表达式基本就是为了实体转换为JSON而设计的。怎么样去实现实体转JSON呢？这就涉及到JSON实质的问题，JSON是对象还是字符串？

## 使用说明

|  |
| --- |
| package com.kuiren.common.service;  import java.util.Collection;  import java.util.List;  import java.util.Map;  import org.hibernate.Criteria;  import org.hibernate.Session;  import org.springframework.jdbc.core.JdbcTemplate;  import com.kuiren.common.BasicService;  import com.kuiren.common.dao.GenericDao;  import com.kuiren.common.dao.JdbcDao;  import com.kuiren.common.page.Page;  import com.kuiren.common.vo.KeyValue;  import com.kuiren.common.vo.KeyValuePair;  import com.kuiren.common.vo.PropertyQuery;  /\*\*  \* 版权归作者所有  \*  \* 业务层接口基类  \*  \* @author 彭仁夔  \*  \* @param <T>  \* 实体Model  \* @param <PK>  \* 主键类型，如Integer  \*/  public interface BaseService<T, PK> {  /\*\*  \* 根据指定的分页参数及实体查询参数  \*  \* @param entityPage  \* 分页参数，  \* @param entity  \* 查询参数  \* @return  \*/  public List<T> search(Page<T> entityPage, T entity);  /\*\*  \* 根据实体查询参数取得满足所有的条件的数据  \*  \* @param entity  \* @return  \*/  public List<T> search(T entity);  /\*\*  \* 保存（插入）参数指定的实体  \*  \* @param entity  \* 新创建的实体  \* @return 主键  \*/  public PK save(T entity);  /\*\*  \* 保存或更新数据: 对象中如果有主键就修改, 如果没有主键就保存. 对于修改的实体，一般先通过find(id)函数找到原有  \* 实体，之后把修改的部分属性更新到原有实体上，然后进行保存  \*  \* @param entity  \* 实体  \*/  public void saveOrUpdate(T entity);  /\*\*  \* 保存（插入）或修改参数指定的实体(集合）  \*  \* @param entities实体  \* (集合）  \*/  public void saveAll(List<T> entities);  /\*\*  \* merge 和saveOrUpdate不一样，当session中某持久化对象有id相同的两个纪录时，  \* 必须用merge，merge会在保存之前来合并记录，不然会报错。 合并记录后的动作和 saveOrUpdate一样  \*  \* @param entity  \* 实体  \*/  public void merge(T entity);  /\*\*  \* 更新实体  \*  \* @param entity  \* 实体  \*/  public void update(T entity);  /\*\*  \* 把指定的属性字段s更新到满足条件的的数据库数据中去  \*  \* @param list  \* 更新的字段列表 KeyValuePair中只要指定key和value即可  \* @param where  \* 条件列表 KeyValuePair中只要指定key和value即可  \*/  public void updateByProperty(List<KeyValuePair> list,  List<KeyValuePair> where);  /\*\*  \* @see {@link BaseService#updateByProperty}简化，只能指定一个查询条件  \* @param list  \* 更新的字段列表 KeyValuePair中只要指定key和value即可  \* @param wherename  \* 查询字段名  \* @param wherevalue  \* 查询字段值  \*/  public void updateByProperty(List<KeyValuePair> list, String wherename,  Object wherevalue);  /\*\*  \* @see {@link BaseService#updateByProperty}简化，只能指定一个查询条件 和一个更新字段  \* @param name更新字段名  \* @param value更新字段值  \* @param wherename  \* 查询字段名  \* @param wherevalue查询字段值  \*/  public void updateByProperty(String name, Object value, String wherename,  Object wherevalue);  /\*\*  \* 更新实体(集合）  \*  \* @param entities  \*/  public void updateAll(List<T> entities);  /\*\*  \* 删除实体  \*  \* @param entity  \* 实体  \*/  public void delete(T entity);  /\*\*  \* 通过Id删除实体  \*  \* @param id  \* 实体id  \*/  public void deleteById(PK id);  /\*\*  \* 根据指定的属性名及值删除满足条件的所有实体  \*  \* @param name  \* @param value  \*/  public void deleteByProperty(String name, Object value);  /\*\*  \* 删除实体(集合）  \*  \* @param entities  \*/  public void deleteAll(List<T> entities);  /\*\*  \* 根据Id载入实体,通过findByPK代替  \*  \* @param id  \* @return 实体  \*/  @Deprecated  public T find(final PK id);  /\*\*  \* 根据Id载入实体  \*  \* @param id  \* @return 实体  \*/  public T findByPK(final PK id);  /\*\*  \* 载入所有实体  \*  \* @return 实体  \*/  public List<T> findAll();  /\*\*  \* 载入所有实体进行分页  \*  \* @param page  \* @return  \*/  public Page<T> findAll(Page<T> page);  /\*\*  \* 通过hql和指定参数来查询实体（集合），通过findByHql代替  \*  \* @param hql  \* @param values  \* @return 实体（集合）  \*/  @Deprecated  public List<T> find(String hql, Object... values);  /\*\*  \* 通过hql和指定参数来查询实体（集合）  \*  \* @param hql  \* @param values  \* @return 实体（集合）  \*/  public List<T> findByHql(String hql, Object... values);  /\*\*  \* 通过hql和指定参数来查询实体  \*  \* @param hql  \* @param values  \* @return 实体（集合）  \*/  public T findUniqueByHql(String hql, Object... values);  /\*\*  \* 根据指定属性及值查询实体（集合）  \*  \* @param propertyName  \* 属性名  \* @param value  \* 值  \* @return 实体（集合）  \*/  public List<T> findByProperty(String propertyName, Object value);  /\*\*  \* 根据其sql的语句查找到对象转换为对象list，其语句类似如下 String sql =  \* " select {a.\*} from t\_ast\_asset\_type a "; List<TAssetType> olist =  \* (List<TAssetType>) getSession() .createSQLQuery(sql).addEntity("a",  \* TAssetType.class).list();  \*  \* @author 彭仁夔于2015年10月8日上午9:36:45创建  \* @param sql  \* 查询的sql  \* @param clz  \* 转换的类 例中为 TAssetType.class  \* @param alias  \* 别名 例中别名为a  \* @return  \*  \*/  public <T1> List<T1> findEnitiesBySql(String sql, Class<T1> clz,  String alias);  /\*\*  \* 根据指定属性及值查询实体  \*  \* @param propertyName  \* @param value  \* @return 满足条件的第一个记录  \*/  public T findUniqueByProperty(String propertyName, Object value);  /\*\*  \* 根据分页、hql,参数查找 实体  \*  \* @param page  \* @param hql  \* @param values  \* @return  \*/  public Page<T> find(Page<T> page, String hql, Object... values);  /\*\*  \* 取得hibernate的session  \*  \* @return Session  \*/  public Session getSession();  /\*\*  \* 取得Spring Jdbc的Template  \*  \* @return Template  \*/  public JdbcTemplate getJdbcTemplate();  /\*\*  \* 根据指定的名称及对应的值查询实体列表  \*  \* @param names  \* 当前实体的属性名称列表  \* @param objs  \* 属性名称对应的值列表  \* @param kValues  \* order列表，采用KeyValue进行封装，  \* 其key为order的name，其value为desc或asc，默认采用asx  \* @return  \*/  public List<T> findByProperty(String[] names, Object[] objs,  List<KeyValue> kValues);  /\*\*  \* 与@see{@link BaseService#findByProperty}相类似，多了一个top参数， 用来指定取得前面多少条  \*  \* @param names  \* @param objs  \* @param kValues  \* @param top  \* @return  \*/  List<T> findByProperty(String[] names, Object[] objs,  List<KeyValue> kValues, int top);  /\*\*  \* 根据参数取得实体列表 \* \* newsService.findByPropertyQuery(  \* "top:desc,createTime:desc,newsId:desc", 7, new PropertyQuery( "newsType",  \* "=", ntNewsType), new PropertyQuery( "state", "=", 2));  \*  \* @param kvs  \* order排序（采用id:desc,name:asc)的格式  \* @param top  \* 前面top条  \* @param propertyQueries  \* 查询条件列表 {@link PropertyQuery} ，支持 like,<,>，=四种常用关系，其中=为默认  \* @return  \*/  List<T> findByPropertyQuery(String kvs, Integer top,  PropertyQuery... propertyQueries);  /\*\*  \* @see @link {@link BaseService#findByPropertyQuery}  \* @param kvs  \* @param propertyQueries  \* @return  \*/  List<T> findByPropertyQuery(String kvs, PropertyQuery... propertyQueries);  List<T> findByPropertyQueries(PropertyQuery... propertyQueries);  /\*\*  \*  \* \* newsService.findByPropertyQuery(  \* "top:desc,createTime:desc,newsId:desc", new PropertyQuery( "newsType",  \* "=", ntNewsType), new PropertyQuery( "state", "=", 2));  \*  \* @see @link {@link BaseService#findByPropertyQuery},只取得前面一条记录  \* @param kvs  \* @param propertyQueries  \* @return  \*/  T findUniqueByPropertyQuery(String kvs, PropertyQuery... propertyQueries);  /\*\*  \* 多个属性找到唯一的记录  \*  \* @author:彭仁夔 于2014年11月6日下午3:48:24创建  \* @param propertyQueries  \* @return  \*/  T findUniqueByPropertyQueries(PropertyQuery... propertyQueries);  /\*\*  \* @see {@link JdbcTemplate#queryForInt}  \* @param sql  \* @return  \*/  int queryForInt(String sql);  /\*\*  \* @see {@link JdbcTemplate#queryForMap}  \* @param sql  \* @return  \*/  Map queryForMap(String sql);  /\*\*  \* @see {@link JdbcTemplate#queryForObject}  \* @param sql  \* @return  \*/  Object queryForObject(String sql, Class clz);  void deleteByProperties(KeyValuePair... kvs);  /\*\*  \* 取得Hibernte的entitydao,一般不需要使用  \*  \* @return  \*/  GenericDao<T, PK> getEntityDao();  /\*\*  \* 取得JdbcTemplate的jdbcDal，一般不需要使用  \*  \* @return  \*/  JdbcDao<T, PK> getJdbcDao();  void updateEmit(T entity, boolean emptyStrEmit);  void updateEmit(T entity, boolean emptyStrEmit, boolean idnulladd);  List<KeyValue> getEnums(String methodName);  String findPropsByDotSep(String displayname, String name, String dotSepStr);  /\*\*  \* <pre>  \* @see {@link BasicService#findByIds},  \* 与其不同的是可以指定查询的字段名  \* </pre>  \*  \* @param name  \* 当前需要查询的字段名  \* @param dotSepStr  \* 查询字段的值，采用，分隔  \* @return  \*/  List<T> findByDotSep(String name, String dotSepStr);  /\*\*  \* @see {@link BasicService#findByDotSep}, 与之不同是需要自己去指定集合，而不是，分隔的字符串  \* @param propname  \* 当前需要查询的字段名  \* @param collection  \* 查询的集合  \* @return  \*/  List<T> findByArr(String propname, Collection collection);  /\*\*  \* @see BaseService#findByPropertyQuery(String, PropertyQuery...)  \* ，少了第一个参数，用于order  \* @author:彭仁夔 于2014年10月23日上午11:20:22创建  \* @param propertyQueries  \* @return  \*/  List<T> findByPropertyQuery(PropertyQuery... propertyQueries);  /\*\*  \* 采用hibernate运行sql语句  \*  \* @author:彭仁夔 于2014年11月21日下午12:01:38创建  \* @param sql  \*/  void updateSql(String sql);  /\*\*  \* 取得当前的实体的类  \*  \* @author 彭仁夔于2015年12月24日下午8:15:27创建  \* @return  \*  \*/  public Class<?> getObjClz();  /\*\*  \* 从指定节点的某行开始，找到所有子节点<br>  \* // --包括本行<br>  \* // select \* from XT\_COMMON\_TYPE<br>  \* // start with id = '402881ff5397c87c015397ebd7820001'<br>  \* // connect by pid= prior id ;<br>  \* // --不包括本行<br>  \* // select \* from XT\_COMMON\_TYPE<br>  \* // start with pid = '402881ff5397c87c015397ebd7820001'<br>  \* // connect by pid= prior id ;<br>  \*  \* @param tableName  \* 表名  \* @param id  \* id字段的名称 一般为id ,数据库中的字段  \* @param pid  \* 父节点的名称，一般为pid,parent\_id，数据库中的字段  \* @param clz  \* 转换为类型  \* @param containSelf  \* 是否包括自身行（查询的起始的节点值）  \* @param pidvalue  \* 查询的起始的节点值  \* @return 彭仁夔 于 2016年3月24日 上午11:44:12 创建  \*/  <T1> List<T1> findNestedEnities(String tableName, String id, String pid,  Class<T1> clz, boolean containSelf, String pidvalue);  /\*\*  \* 根据指定的实体class取得hibernate 的Criteria  \* @param c  \* @return 彭仁夔 于 2016年4月2日 上午9:07:25 创建  \*/  Criteria getCriteria(Class c);  Criteria getCriteria();  } |