

## 原 Struts2 + Spring + Hibernate 通用 Service 和 DAO

2013年10月11日 19:32:06 阅读数：6943

我在 Struts2 + Spring + Hibernate 项目开发中总结出了一个Service 和 DAO ,可以用于处理任何的pojo(bean)。使用这两个Service 和 DAO 可以极大地提高开发的效率，不必再分别针对不同的pojo编写对应的Service 和 DAO。内容如下：

DAO：

接口：BaseDao.java

```
[java] 1. package dao;
2.
3. import java.util.List;
4.
5.
6. /**
7.  * @author 雷霄骅
8.  * 对Object的DAO操作
9.  * 提供了通用的一些方法
10. */
11.
12. public interface BaseDao {
13.     public void save(Object object);
14.     public void delete(Object object);
15.     public void update(Object object);
16.     public Object ReadSingle(String targetName,String propertyName,Object value);
17.     public List<Object> ReadByProperty(String targetName,String propertyName,Object value);
18.     public List<Object> ReadAll(String targetName);
19.     public List<Object> ReadAllByOrder(String targetName,String propertyName,String order);
20.     public Object get(int id);
21.     public List<Object> ReadByPropertyList(String targetName,List<String> propertyName, List<Object> value);
22.     public Integer ReadCount(String targetName);
23.     public List<Object> ReadLimitedByOrder(String targetName, String propertyName,int num, String order);
24. }
```

实现：BaseDaoImpl.java

```
[java] 1. package dao;
2.
3. import java.sql.SQLException;
4. import java.util.List;
5.
6. import org.hibernate.HibernateException;
7. import org.hibernate.Query;
8. import org.hibernate.Session;
9. import org.springframework.orm.hibernate3.HibernateCallback;
10. import org.springframework.orm.hibernate3.support.HibernateDaoSupport;
11.
12.
13.
14.
15. /**
16.  * @author 雷霄骅
17.  * HibernateTemplate提供非常多的常用方法来完成基本的操作，比如通常的增加、删除、修改、查询等操作，
18.  * Spring 2.0更增加对命名SQL查询的支持，也增加对分页的支持。大部分情况下，就可完成大多数DAO对象的CRUD操作。
19.  */
20. public class BaseDaoImpl extends HibernateDaoSupport implements BaseDao{
21.
22.     @Override
23.     public void save(Object object) {
24.         getHibernateTemplate().save(object);
25.         //System.out.println("save "+object.toString());
26.     }
27.
28.     @Override
29.     public void delete(Object object) {
30.         getHibernateTemplate().delete(object);
31.     }
32.
33.     @Override
34.     public void update(Object object) {
35.         getHibernateTemplate().update(object);
36.         //改用saveOrUpdate, 在评价的时候，第一次创建的时候Save，其他时候Update
37.         //getHibernateTemplate().saveOrUpdate(object);
38.         //System.out.println("update "+object.toString());
39.     }
40.
41.     @SuppressWarnings("unchecked")
42.     @Override
43.     public Object ReadSingle(final String targetName,final String propertyName, final Object value) {
44.         // TODO Auto-generated method stub
```

```

44. // TODO Auto-generated method stub
45. return (Object) getHibernateTemplate().execute(new HibernateCallback() {
46.     /*doInHibernate()。session的创建和销毁，一切都在程序内部完成。*/
47.     public Object doInHibernate(Session session)
48.         throws HibernateException, SQLException {
49.         String hql = "from "+targetName+" as "+targetName+" where "+targetName+"." + propertyName + "=:value";
50.         Query query = session.createQuery(hql);
51.         query.setParameter("value", value);
52.         return query.uniqueResult();
53.     }
54. });
55. }
56.
57. @SuppressWarnings("unchecked")
58. @Override
59. public List<Object> ReadAll(String targetName) {
60.     // TODO Auto-generated method stub
61.     String hql="from "+targetName;
62.     return getHibernateTemplate().find(hql);
63. }
64.
65. @SuppressWarnings("unchecked")
66. @Override
67. public List<Object> ReadByProperty(final String targetName, final String propertyName,
68.     final Object value) {
69.     // TODO Auto-generated method stub
70.     return (List<Object>) getHibernateTemplate().execute(new HibernateCallback() {
71.         /*doInHibernate()。session的创建和销毁，一切都在程序内部完成。*/
72.         public Object doInHibernate(Session session)
73.             throws HibernateException, SQLException {
74.             String hql = "from "+targetName+" as "+targetName+" where "+targetName+"." + propertyName + "=:value";
75.             Query query = session.createQuery(hql);
76.             query.setParameter("value", value);
77.             return query.list();
78.         }
79.     });
80. }
81. //比ReadByProperty简单很多
82. @Override
83. public Object get(int id) {
84.     // TODO Auto-generated method stub
85.     return getHibernateTemplate().get(Object.class, id);
86. }
87.
88. @Override
89. public List<Object> ReadByPropertyList(final String targetName,
90.     final List<String> propertyName, final List<Object> value) {
91.     // TODO Auto-generated method stub
92.     return (List<Object>) getHibernateTemplate().execute(new HibernateCallback() {
93.         /*doInHibernate()。session的创建和销毁，一切都在程序内部完成。*/
94.         public Object doInHibernate(Session session) throws HibernateException, SQLException {
95.             String hql = "from "+targetName+" as "+targetName;
96.
97.             //-----
98.             for(int i=0;i<propertyName.size();i++){
99.                 String propertynameTemp= propertyName.get(i);
100.                 Object propertyvalueTemp= value.get(i);
101.                 if(propertynameTemp!=null){
102.                     if(i==0){
103.                         hql=hql+" where "+targetName+"." + propertynameTemp + "=" + propertyvalueTemp + " ";
104.                     }else{
105.                         hql=hql+" and "+targetName+"." +propertynameTemp + "=" + propertyvalueTemp + " ";
106.                     }
107.                 }
108.             }
109.             //-----
110.             Query query = session.createQuery(hql);
111.             //当返回的数据不是一条的时候，不用uniqueResult(), 而用list()
112.             return query.list();
113.         }
114.     });
115. }
116. //这里在Hibernate2.0之前版本list.get(0)返回的是Integer类型。
117. //但是在Hibernate3.0以后版本list.get(0)返回的是Long类型。
118. //所以在这里不可以由Long型强转成Integer类型。
119. //Integer属于不可更改类型，而且Long和Integer没有任何继承关系，当然不能这样转换。
120. @Override
121. public Integer ReadCount(final String targetName) {
122.     // TODO Auto-generated method stub
123.     return (Integer) getHibernateTemplate().execute(new HibernateCallback() {
124.         /*doInHibernate()。session的创建和销毁，一切都在程序内部完成。*/
125.         public Object doInHibernate(Session session)
126.             throws HibernateException, SQLException {
127.             String hql = "select count(*) from "+targetName;
128.             //System.out.println(hql);
129.             //注:java.lang.Number是Integer,Long的父类。
130.             return ((Number)session.createQuery(hql).iterate().next()).intValue();
131.         }
132.     });
133. }
134.
135. @Override

```

```

136.     public List<Object> ReadLimitedByOrder(final String targetName,
137.         final String propertyName, final int num, final String order) {
138.         // TODO Auto-generated method stub
139.         return (List<Object>) getHibernateTemplate().execute(new HibernateCallback() {
140.             /*doInHibernate()。session的创建和销毁，一切都在程序内部完成。*/
141.             public Object doInHibernate(Session session) throws HibernateException, SQLException {
142.                 String hql ="from "+targetName+" as "+targetName+ " order by "+targetName+"." + propertyName+ " " + order;
143.                 Query query = session.createQuery(hql);
144.                 query.setMaxResults(num);
145.                 //当返回的数据不是一条的时候，不用uniqueResult(), 而用list()
146.                 return query.list();
147.             }
148.         });
149.     }
150.
151.     @Override
152.     public List<Object> ReadAllByOrder(String targetName, String propertyName,
153.         String order) {
154.         // TODO Auto-generated method stub
155.         String hql ="from "+targetName+" as "+targetName+ " order by "+targetName+"." + propertyName+ " " + order;
156.         return getHibernateTemplate().find(hql);
157.     }
158.
159.
160.
161. }

```

## Service :

接口：BaseService.java

```

1. package service;
2.
3. import java.util.List;
4.
5. /**
6.  * @author 雷霄骅
7.  * 对Object的Service
8.  * 提供了一些通用的方法
9.  */
10. public interface BaseService {
11.     public void save(Object object);
12.     public void update(Object object);
13.     public void delete(Object object);
14.     public Object ReadByID(String targetName,int id);
15.     @SuppressWarnings("rawtypes")
16.     public List ReadAll(String targetName);
17.     public List ReadAllByOrder(String targetName,String propertyName,String order);
18.     @SuppressWarnings("rawtypes")
19.     public List ReadByProperty(String targetName,String propertyName,Object propertyValue);
20.     public List ReadByPropertyList(String targetName,List<String> propertyName,List<Object> propertyValue);
21.     public List ReadLimitedByOrder(String targetName,String propertyName,int num,String order);
22.     public Object ReadSingle(String targetName,String propertyName,Object propertyValue);
23.     public int ReadCount(String targetName);
24.
25. }

```

实现：BaseServiceImpl.java

```

1. package service;
2.
3. import java.util.ArrayList;
4. import java.util.List;
5.
6. import dao.BaseDao;
7. /**
8.  * @author 雷霄骅
9.  * 对Object的Service
10.  * 提供了一些通用的方法
11.  */
12. public class BaseServiceImpl implements BaseService {
13.
14.     private BaseDao baseDao;
15.     @Override
16.     public void save(Object object) {
17.         // TODO Auto-generated method stub
18.         baseDao.save(object);
19.     }
20.
21.     @Override
22.     public void update(Object object) {
23.         // TODO Auto-generated method stub
24.         baseDao.update(object);
25.     }

```

```

26.
27.     @Override
28.     public void delete(Object object) {
29.         // TODO Auto-generated method stub
30.         baseDao.delete(object);
31.     }
32.
33.     @Override
34.     public Object ReadByID(String targetName,int id) {
35.         // TODO Auto-generated method stub
36.         return baseDao.ReadSingle(targetName,"id", id);
37.     }
38.
39.     @SuppressWarnings("rawtypes")
40.     @Override
41.     public List ReadAll(String targetName) {
42.         // TODO Auto-generated method stub
43.         return baseDao.ReadAll(targetName);
44.     }
45.
46.     @SuppressWarnings("rawtypes")
47.     @Override
48.     public List ReadAllByOrder(String targetName,String propertyName,String order) {
49.         // TODO Auto-generated method stub
50.         return baseDao.ReadAllByOrder(targetName,propertyName,order);
51.     }
52.
53.     public BaseDao getBaseDao() {
54.         return baseDao;
55.     }
56.
57.     public void setBaseDao(BaseDao baseDao) {
58.         this.baseDao = baseDao;
59.     }
60.
61.     @Override
62.     public List ReadByProperty(String targetName, String propertyName,
63.         Object propertyValue) {
64.         // TODO Auto-generated method stub
65.         return baseDao.ReadByProperty(targetName, propertyName, propertyValue);
66.     }
67.
68.     @Override
69.     public Object ReadSingle(String targetName, String propertyName,
70.         Object propertyValue) {
71.         // TODO Auto-generated method stub
72.         return baseDao.ReadSingle(targetName, propertyName, propertyValue);
73.     }
74.
75.     @Override
76.     public int ReadCount(String targetName) {
77.         // TODO Auto-generated method stub
78.         return baseDao.ReadCount(targetName);
79.     }
80.
81.     @Override
82.     public List ReadLimitedByOrder(String targetName, String propertyName,
83.         int num, String order) {
84.         return baseDao.ReadLimitedByOrder(targetName,propertyName,num,order);
85.     }
86.
87.     @Override
88.     public List ReadByPropertyList(String targetName,
89.         List<String> propertyName, List<Object> propertyValue) {
90.         // TODO Auto-generated method stub
91.         return baseDao.ReadByPropertyList(targetName,propertyName,propertyValue);
92.     }
93.
94. }

```

这样，在Action层调用方法的时候，可以直接调用BaseService相应的方法完成操作。

举一个例子：

有这么一个名字叫Blog的pojo：

```

1. package bean;
2.
3. import java.sql.Timestamp;
4. import javax.persistence.Column;
5. import javax.persistence.Entity;
6. import javax.persistence.FetchType;
7. import javax.persistence.GeneratedValue;
8. import static javax.persistence.GenerationType.IDENTITY;
9. import javax.persistence.Id;
10. import javax.persistence.JoinColumn;
11. import javax.persistence.ManyToOne;

```

```

12. import javax.persistence.Table;
13.
14. /**
15.  * Blog entity. @author MyEclipse Persistence Tools
16.  */
17. @Entity
18. @Table(name = "blog", catalog = "vqe")
19. public class Blog implements java.io.Serializable {
20.
21.     // Fields
22.
23.     private Integer id;
24.     private Admin admin;
25.     private String title;
26.     private Timestamp modifytime;
27.     private String content;
28.
29.     // Constructors
30.
31.     /** default constructor */
32.     public Blog() {
33.     }
34.
35.     /** full constructor */
36.     public Blog(Admin admin, String title, Timestamp modifytime, String content) {
37.         this.admin = admin;
38.         this.title = title;
39.         this.modifytime = modifytime;
40.         this.content = content;
41.     }
42.
43.     // Property accessors
44.     @Id
45.     @GeneratedValue(strategy = IDENTITY)
46.     @Column(name = "id", unique = true, nullable = false)
47.     public Integer getId() {
48.         return this.id;
49.     }
50.
51.     public void setId(Integer id) {
52.         this.id = id;
53.     }
54.
55.     @ManyToOne(fetch = FetchType.EAGER)
56.     @JoinColumn(name = "adminid")
57.     public Admin getAdmin() {
58.         return this.admin;
59.     }
60.
61.     public void setAdmin(Admin admin) {
62.         this.admin = admin;
63.     }
64.
65.     @Column(name = "title", length = 200)
66.     public String getTitle() {
67.         return this.title;
68.     }
69.
70.     public void setTitle(String title) {
71.         this.title = title;
72.     }
73.
74.     @Column(name = "modifytime", length = 19)
75.     public Timestamp getModifytime() {
76.         return this.modifytime;
77.     }
78.
79.     public void setModifytime(Timestamp modifytime) {
80.         this.modifytime = modifytime;
81.     }
82.
83.     @Column(name = "content", length = 10000)
84.     public String getContent() {
85.         return this.content;
86.     }
87.
88.     public void setContent(String content) {
89.         this.content = content;
90.     }
91.
92. }

```

该类代表博客的一篇文章。

在Action层只需调用BaeService对应的方法就能完成相应的操作。换句话说，只要把pojo的类的名字当一个字符串传递给ReadByID这种的函数，就可以实现相应的功能

。

```

1. //根据ID读取：
2. Blog blog=(Blog) baseService.ReadByID("Blog", blogid);
3. //添加：
4. baseService.save(blog);
5. //修改：
6. baseService.update(blog);
7. //删除：
8. baseService.delete(blog);
9. //读取所有（根据时间降序）
10. List<Blog> resultblog=baseService.ReadAllByOrder("Blog","modifytime","desc");
11. //读取num条（根据时间降序）
12. List<Blog> resultblog=baseService.ReadLimitedByOrder("Blog","modifytime",num,"desc");

```

完整的实现Blog（博客）的增删改查的Action示例：

```

1. package action;
2.
3.
4. import java.sql.Timestamp;
5. import java.util.Date;
6. import java.util.List;
7. import java.util.Map;
8.
9. import service.BaseService;
10.
11.
12. import bean.Admin;
13. import bean.Blog;
14.
15. import com.opensymphony.xwork2.ActionContext;
16. import com.opensymphony.xwork2.ActionSupport;
17. /**
18.  * @author 雷霄骅
19.  * Action
20.  */
21. public class BlogAct extends ActionSupport {
22.     private int blogid;
23.     private int num;
24.     private Blog blog;
25.     private List<Blog> resultblog;
26.     private BaseService baseService;
27.
28.     public int getBlogid() {
29.         return blogid;
30.     }
31.
32.     public void setBlogid(int blogid) {
33.         this.blogid = blogid;
34.     }
35.
36.     public Blog getBlog() {
37.         return blog;
38.     }
39.
40.     public void setBlog(Blog blog) {
41.         this.blog = blog;
42.     }
43.
44.     public BaseService getBaseService() {
45.         return baseService;
46.     }
47.
48.     public void setBaseService(BaseService baseService) {
49.         this.baseService = baseService;
50.     }
51.
52.     public List<Blog> getResultblog() {
53.         return resultblog;
54.     }
55.
56.     public void setResultblog(List<Blog> resultblog) {
57.         this.resultblog = resultblog;
58.     }
59.
60.     public int getNum() {
61.         return num;
62.     }
63.
64.     public void setNum(int num) {
65.         this.num = num;
66.     }
67.
68.     public String Add(){
69.         try{
70.             //-----
71.             ActionContext context = ActionContext.getContext();
72.             Map sessionMap = context.getSessionMap();

```

```

72.         Map sessionMap = context.getSession();
73.         Admin admin=(Admin)sessionMap.get("admin");
74.         //-----
75.         blog.setModifytime( new Timestamp(new Date().getTime()));
76.         blog.setAdmin(admin);
77.         baseService.save(blog);
78.         return SUCCESS;
79.     }
80.     catch(Exception ex){
81.         ex.printStackTrace();
82.         return ERROR;
83.     }
84. }
85.
86. public String Delete(){
87.     try{
88.         blog=(Blog) baseService.ReadByID("Blog", blogid);
89.         baseService.delete(blog);
90.         return SUCCESS;
91.     }
92.     catch(Exception ex){
93.         ex.printStackTrace();
94.         return ERROR;
95.     }
96. }
97.
98. public String Read(){
99.     try{
100.         blog=(Blog) baseService.ReadByID("Blog", blogid);
101.         return SUCCESS;
102.     }
103.     catch(Exception ex){
104.         ex.printStackTrace();
105.         return ERROR;
106.     }
107. }
108.
109. public String Update(){
110.     try{
111.         //-----
112.         ActionContext context = ActionContext.getContext();
113.         Map sessionMap = context.getSession();
114.         Admin admin=(Admin)sessionMap.get("admin");
115.         //-----
116.         blog.setModifytime( new Timestamp(new Date().getTime()));
117.         blog.setAdmin(admin);
118.         baseService.update(blog);
119.         return SUCCESS;
120.     }
121.     catch(Exception ex){
122.         ex.printStackTrace();
123.         return ERROR;
124.     }
125. }
126.
127. public String ReadAll(){
128.     try{
129.         resultblog=baseService.ReadAllByOrder("Blog","modifytime","desc");
130.         return SUCCESS;
131.     }
132.     catch(Exception ex){
133.         ex.printStackTrace();
134.         return ERROR;
135.     }
136. }
137.
138. public String ReadLimitedByOrder(){
139.     try{
140.         resultblog=baseService.ReadLimitedByOrder("Blog","modifytime",num,"desc");
141.         return SUCCESS;
142.     }
143.     catch(Exception ex){
144.         ex.printStackTrace();
145.         return ERROR;
146.     }
147. }
148. }

```

此PDF由spygg生成,请尊重原作者版权!!!

我的邮箱:liushidc@163.com