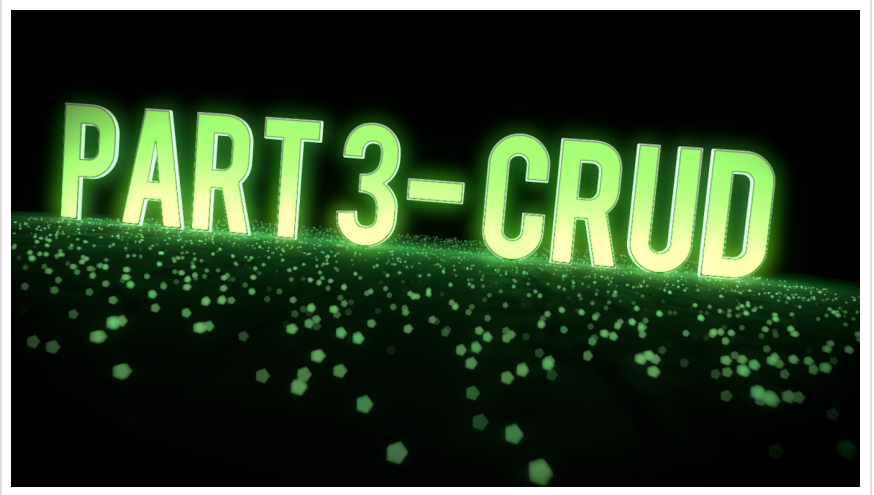
« Part 2 – Securing Web Application GlassFish V3 – JAAS (Authentication and Authorization)
Part 4 – Lazy Loading and Pagination with JSF 2.1 / Primefaces 3.4.2 DataTable and Form validation »

Part 3 – Basic Implementation of CRUD

2

21 Sep 2012 | Java EE

In this post, we will create Java Beans to encapsulate Business Logic of our web application. We will start with creating a service class for persisting objects. After that, we will create a backing bean that is going to be used in the web layer. (it will be explained in the next post)



The following shows a basic implementation of a generic Data Access Service class for CRUD operations. The web container is going to be located within the same JVM as our EJB container so we won't need any remote interface for DataAccessService class. DataAccessService class going to be stateless as we won't need any any conversational state and DataAccessService instances will be shared by multiple clients. Entity manager instance is injected with @PersistenceContext to manipulate User Entity (Cread, Read, Update, and Delete). In the example, static Named Queries used because the performance cost of processing dynamic queries higher than processing static named queries.

Abstract Data Access Service class

```
1  /**
2   * Implementation of the generic Data Access Service
3   * All CRUD (create, read, update, delete) basic data access operat
4   * persistent object are performed in this class.
5   * @author Emre Simtay <emre@simtay.com>
6   **/
7  public abstract class DataAccessService<T> {
8
9  @PersistenceContext
10  private EntityManager em;
11
12 public DataAccessService() {
13  }
14
15  private Class<T> type;
16
17 /**
```

Blogroll

› Arquillian testing tool for Java EE
› PrimeFaces JSF Component Suite

Recent Posts

› Glassfish4 and JSF 2.2 ViewScope support (JSF 2.2, CDI, JPA 2 (ORM) / EclipseLink, JAAS , MySQL, example CRUD web application)
› My Book Review – Instant PrimeFaces Starter- Packt Publishing
› Reviewing Instant PrimeFaces Starter- Packt Publishing
› My Book Review – PrimeFaces Cookbook by Oleg Varaksin and Mert Çalışkan
› Reviewing PrimeFaces Cookbook – Packt Publishing – Oleg Varaksin, Mert Çalışkan

Recent Comments

› simtay on Part 4 – Lazy Loading and Pagination with JSF 2.1 / Primefaces 3.4.2 DataTable and Form validation
› simtay on Simple CRUD Web Application with JSF 2.1, PrimeFaces 3.5, Maven and JPA
› Mohamed on Part 4 – Lazy Loading and Pagination with JSF 2.1 / Primefaces 3.4.2 DataTable and Form validation
› dukgo on Simple CRUD Web Application with JSF 2.1, PrimeFaces 3.5, Maven and JPA
› simtay on Simple CRUD Web Application with JSF 2.1, PrimeFaces 3.5, Maven and JPA

Archives

› November 2013 › August 2013
› April 2013 › March 2013
› January 2013 › September 2012

Categories

```
18   * Default constructor
19   *
20   * @param type entity class
21   */
22   public DataAccessService(Class<T> type) {
23   this.type = type;
24   }
25
26   /**
27   * Stores an instance of the entity class in the database
28   * @param T Object
29   * @return
30   */
31   public T create(T t) {
32   this.em.persist(t);
33   this.em.flush();
34   this.em.refresh(t);
35   return t;
36   }
37
38  /**
39   * Retrieves an entity instance that was previously persisted to th
40   * @param T Object
41   * @param id
42   * @return
43   */
44   public T find(Object id) {
45   return this.em.find(this.type, id);
46   }
47
48  /**
49   * Removes the record that is associated with the entity instance
50   * @param type
51   * @param id
52   */
53   public void delete(Object id) {
54   Object ref = this.em.getReference(this.type, id);
55   this.em.remove(ref);
56   }
57
58  /**
59   * Removes the number of entries from a table
60   * @param <T>
61   * @param items
62   * @return
63   */
64   public boolean deleteItems(T[] items) {
65   for (T item : items) {
66   if( item instanceof User){
67   User user = (User)item;
68   if(user.getId() == 1){
69   continue;
70   }
71   }
72   em.remove(em.merge(item));
73   }
74   return true;
75   }
76
77  /**
78   * Updates the entity instance
79   * @param <T>
80   * @param t
81   * @return the object that is updated
82   */
83   public T update(T item) {
84   if( item instanceof User){
85   User user = (User)item;
86   if(user.getId() == 1){
87   return item;
88   }
89   }
90   return (T) this.em.merge(item);
91
92   }
93
94  /**
95   * Returns the number of records that meet the criteria
96   * @param namedQueryName
97   * @return List
98   */
99   public List findWithNamedQuery(String namedQueryName) {
100  return this.em.createNamedQuery(namedQueryName).getResultList();
101  }
102
103 /**
104  * Returns the number of records that meet the criteria
105  * @param namedQueryName
106  * @param parameters
```

› Java EE

Meta

› Log in
› Entries RSS
› Comments RSS
› WordPress.org

```
 * @return List
 */
 public List findWithNamedQuery(String namedQueryName, Map paramete
 return findWithNamedQuery(namedQueryName, parameters, 0);
 }

/**
 * Returns the number of records with result limit
 * @param queryName
 * @param resultLimit
 * @return List
 */
 public List findWithNamedQuery(String queryName, int resultLimit)
 return this.em.createNamedQuery(queryName).
 setMaxResults(resultLimit).
 getResultList();
 }

/**
 * Returns the number of records that meet the criteria
 * @param <T>
 * @param sql
 * @param type
 * @return List
 */
 public List<T> findByNativeQuery(String sql) {
 return this.em.createNativeQuery(sql, type).getResultList();
 }

/**
 * Returns the number of total records
 * @param namedQueryName
 * @return int
 */
 public int countTotalRecord(String namedQueryName) {
 Query query = em.createNamedQuery(namedQueryName);
 Number result = (Number) query.getSingleResult();
 return result.intValue();
 }

/**
 * Returns the number of records that meet the criteria with parame
 * result limit
 * @param namedQueryName
 * @param parameters
 * @param resultLimit
 * @return List
 */
 public List findWithNamedQuery(String namedQueryName, Map paramete
 Set<Map.Entry<String, Object>> rawParameters = parameters.entrySet
 Query query = this.em.createNamedQuery(namedQueryName);
 if (resultLimit > 0) {
 query.setMaxResults(resultLimit);
 }
 for (Map.Entry<String, Object> entry : rawParameters) {
 query.setParameter(entry.getKey(), entry.getValue());
 }
 return query.getResultList();
 }

 /**
 * Returns the number of records that will be used with lazy loadin
 * @param namedQueryName
 * @param start
 * @param end
 * @return List
 */
 public List findWithNamedQuery(String namedQueryName, int start, i
 Query query = this.em.createNamedQuery(namedQueryName);
 query.setMaxResults(end - start);
 query.setFirstResult(start);
 return query.getResultList();
 }
}
```

User service class that extends DataAccessService class

```
@Stateless
public class UserService extends DataAccessService<User>{

    public UserService(){
        super(User.class);
    }

    /**
     * Returns new user
     * @return User
     */
```

```
    public User newUser(){
        return new User();
    }

}
```

User Entity Class after adding NamedQueries.

```
@Entity
@Table(name="users")
@NamedQueries({@NamedQuery(name = "User.populateUsers", query = "SELE
@NamedQuery(name = "User.countUsersTotal", query = "SELECT COUNT(u) F
public class User extends BaseEntity implements Serializable {
    public final static String ALL = "User.populateUsers";
    public final static String TOTAL = "User.countUsersTotal";
........
}
```

User controller class using DataAccessService

```
@Named
@SessionScoped
public class UserController implements Serializable {
 private @Inject transient Logger logger;
 private @Inject UserService das;
 // Selected users that will be removed
 private User[] selectedUsers;
 // Lazy loading user list
 private LazyDataModel<User> lazyModel;
 // Creating new user
 private User newUser = new User();
 // Selected user that will be updated
 private User selectedUser = new User();

 // Available role list
 private List<Role> roleList;

/**
 * Default constructor
 */
 public UserController() {

}

/**
 * Initializing Data Access Service for LazyUserDataModel class
 * role list for UserContoller class
 */
 @PostConstruct
 public void init(){
 logger.log(Level.INFO, "UserController is initializing");
 lazyModel = new LazyUserDataModel(das);
 roleList = das.findWithNamedQuery(Role.ALL);
 }

/**
 * Create, Update and Delete operations
 */
 public void doCreateUser() {
 das.create(newUser);
 }

 /**
 *
 * @param actionEvent
 */
 public void doUpdateUser(ActionEvent actionEvent){
 das.update(selectedUser);
 }

 /**
 *
 * @param actionEvent
 */
 public void doDeleteUsers(ActionEvent actionEvent){
 das.deleteItems(selectedUsers);
 }

 /**
 * Getters, Setters
 * @return
 */

public User getSelectedUser() {
 return selectedUser;
 }

/**
```

```
69   *
70   * @param selectedUser
71   */
72   public void setSelectedUser(User selectedUser) {
73   this.selectedUser = selectedUser;
74   }
75
76   /**
77   *
78   * @return
79   */
80   public User[] getSelectedUsers() {
81   return selectedUsers;
82   }
83
84   /**
85   *
86   * @param selectedUsers
87   */
88   public void setSelectedUsers(User[] selectedUsers) {
89   this.selectedUsers = selectedUsers;
90   }
91
92  /**
93   *
94   * @return
95   */
96   public User getNewUser() {
97   return newUser;
98   }
99
100 /**
101  *
102  * @param newUser
103  */
104  public void setNewUser(User newUser) {
105  this.newUser = newUser;
106  }
107
108  /**
109  *
110  * @return LazyDataModel
111  */
112  public LazyDataModel<User> getLazyModel() {
113  return lazyModel;
114  }
115
116 /**
117  *
118  * @return List<Role>
119  */
120  public List<Role> getRoleList() {
121  return roleList;
122  }
123
124 /**
125  *
126  * @param roleList
127  */
128  public void setRoleList(List<Role> roleList) {
129  this.roleList = roleList;
130  }
131 }
```

Download the source code here
Please try live demo (Username: Admin, Password:1234)

Happy programmers day!!!

2 thoughts on "Part 3 – Basic Implementation of CRUD"

Reply

Jun 19, 2013 4:04 am

ersanmaz

Thanks a lot for your post. I have a question about deleteItems and update methods in DataAccessService class. You used these methods only for User class. But, i want to rewrite them to manipulate several classes. How should i change them to do that?

Reply

Jul 5, 2013 2:17 pm

simtay

Yes, you can create generic services!

Leave a Reply

Author (required)

Email (will not be published)(required)

Website

Just to prove you are a human, please answer the following math challenge either out loud or typing into the text box

× eight = sixty four

b i link b-quote code close tags

Post Comment

« Part 2 – Securing Web Application GlassFish V3 – JAAS (Authentication and Authorization)

Part 4 – Lazy Loading and Pagination with JSF 2.1 / Primefaces 3.4.2 DataTable and Form validation »