Spring Generic DAO and Generic Service Implementation

For most developers, writing almost the same code for every DAO in an application has become indespensible. To solve this problem and to rescue developers from copying and pasting same codes into every dao classes, we will use **DAO pattern**.

In this tutorial, I will show you how to create dao pattern in a spring application. The foundation of using a Generic DAO is the CRUD operations that you can perform on each entity.

Generic Dao Implementation

Model Class

Hibernate ORM framework uses model classes for database tables. Therefore we need to create a model class as follows:

```
@Entity
public class Admin {
     @Id
     @GeneratedValue(strategy= GenerationType.AUTO)
     private int adminId;

     @Column(length = 20)
     private String username;

     @Column(length = 20)
     private String password;

     @Column(length = 60)

     private String name;

     @Column(length = 60)

     private String surname;

     @Column(length = 254)
     private String email;

//Getter and Setters omitted
//......
}
```

GenericDao Interface

GenericDao interface contains common methods used by multiple dao classes. Also, we are using Java generic concept in this interface. E and K letters represent **Model class type and primary key type**, respectively.

```
public interface GenericDao<E,K> {
     public void add(E entity) ;
     public void saveOrUpdate(E entity) ;
     public void update(E entity) ;
     public void remove(E entity);
     public E find(K key);
     public List<E> getAll() ;
}
```

GenericDaoImpl Class

```
@SuppressWarnings("unchecked")
@Repository
public abstract class GenericDaoImpl<E, K extends Serializable>
          implements GenericDao<E, K> {
     @Autowired
     private SessionFactory sessionFactory;
```

```java

    protected Class<? extends E> daoType;

    /**
    * By defining this class as abstract, we prevent Spring from creating
    * instance of this class If not defined as abstract,
    * getClass().getGenericSuperClass() would return Object. There would be
    * exception because Object class does not hava constructor with parameters.
    */
    public GenericDaoImpl() {
        Type t = getClass().getGenericSuperclass();
        ParameterizedType pt = (ParameterizedType) t;
        daoType = (Class) pt.getActualTypeArguments()[0];
    }

    protected Session currentSession() {
        return sessionFactory.getCurrentSession();
    }

    @Override
    public void add(E entity) {
        currentSession().save(entity);
    }

    @Override
    public void saveOrUpdate(E entity) {
        currentSession().saveOrUpdate(entity);
    }

    @Override
    public void update(E entity) {
        currentSession().saveOrUpdate(entity);
    }

    @Override
    public void remove(E entity) {
        currentSession().delete(entity);
    }

    @Override
    public E find(K key) {
        return (E) currentSession().get(daoType, key);
    }

    @Override
    public List<E> getAll() {
        return currentSession().createCriteria(daoType).list();
    }
}
```

This class is marked as **abstract**, so it can be used only through specific entity DAO implementations. The most important part of this class is its constructor implementation. This is where all the magic will happen. In order to keep things as clean as possible and to avoid AOP and other techniques, we use this hack: we get the actual parametrized type of a concrete class, store it in a Class variable, and use it in EntityManager's methods. Because this class is marked as **abstract**, `getClass().getGenericSuperclass();` returns `GenericDaoImpl<Admin, Integer>` in this example.

Custom Concrete DAOs

Writing concrete DAOs is very easy. Just follow the below steps:
1. Create entity interface that extends **GenericDao.**
2. Create a concrete implementation of the entity interface that extends **GenericDaoImpl** class.

AdminDao Interface

In our example, the entity class is Admin, so we create **AdminDao** interface:

```java
public interface AdminDao extends GenericDao<Admin, Integer>{
     public boolean removeAdmin(Integer id);
     public boolean isAdminRegistered(String userName, String password);
     public Admin getAdmin(String username);
}
```

AdminDaoImpl Class

AdminDaoImpl class is annotated with @Repository annotation because dao classes are generally annotated as **Repository.** Also note that this class extends **GenericDaoImpl** and implements **AdminDao.**

```java
@Repository
public class AdminDaoImpl extends GenericDaoImpl<Admin, Integer>
                         implements AdminDao {
     @Override
     public boolean removeAdmin(Integer id) {
         Query employeeTaskQuery = currentSession().createQuery(
                 "from Admin u where :id");
         employeeTaskQuery.setParameter("id", id);
         return employeeTaskQuery.executeUpdate() > 0;
     }

     @Override
     public boolean isAdminRegistered(String userName, String password) {
         /*You can use any character instead of 'A'. If a record is found,
         only single character, in this example 'A', will return from database
         */
         Query employeeTaskQuery = currentSession().createQuery(
                 "select 'A' from Admin u where username=:username and password=:password");
         employeeTaskQuery.setParameter("username", userName);
         employeeTaskQuery.setParameter("password", password);
         return employeeTaskQuery.list().size() > 0;
     }

     @Override
     public Admin getAdmin(String username) {
         Query query = currentSession().createQuery(
                 "from Admin " +
                         "where username=:username");
         query.setParameter("username", username);
         return (Admin) query.uniqueResult();

     }
}
```

Generic Service Implementation

Generic service implementation save us from repetitive service class codes like generic dao. Therefore, creating generic service implementation looks like the generic dao implementation.

GenericService Interface

```java
public interface GenericService<E, K> {
     public void saveOrUpdate(E entity);
     public List<E> getAll();
     public E get(K id);
     public void add(E entity);
     public void update(E entity);
     public void remove(E entity);
}
```

GenericServiceImpl Class

```java
@Service
public abstract class GenericServiceImpl<E, K>
        implements GenericService<E, K> {

    private GenericDao<E, K> genericDao;

    public GenericServiceImpl(GenericDao<E,K> genericDao) {
        this.genericDao=genericDao;
    }

    public GenericServiceImpl() {
    }

    @Override
    @Transactional(propagation = Propagation.REQUIRED)
    public void saveOrUpdate(E entity) {
        genericDao.saveOrUpdate(entity);
    }

    @Override
    @Transactional(propagation = Propagation.REQUIRED, readOnly = true)
    public List<E> getAll() {
        return genericDao.getAll();
    }

    @Override
    @Transactional(propagation = Propagation.REQUIRED, readOnly = true)
    public E get(K id) {
        return genericDao.find(id);
    }

    @Override
    @Transactional(propagation = Propagation.REQUIRED)
    public void add(E entity) {
        genericDao.add(entity);
    }

    @Override
    @Transactional(propagation = Propagation.REQUIRED)
    public void update(E entity) {
        genericDao.update(entity);
    }

    @Override
    @Transactional(propagation = Propagation.REQUIRED)
    public void remove(E entity) {
        genericDao.remove(entity);
    }
}
```

AdminService Interface

```java
public interface AdminService extends GenericService<Admin,Integer>{
    public boolean removeAdmin(Integer id);
    public boolean isAdminRegistered(String userName, String password);
    public Admin getAdmin(String userName);
}
```

AdminServiceImpl Class

```java
@Service
public class AdminServiceImpl extends GenericServiceImpl<Admin, Integer>
        implements AdminService {

    private AdminDao adminDao;
```

```
        public AdminServiceImpl(){

        }
        @Autowired
        public AdminServiceImpl(
                @Qualifier("adminDaoImpl") GenericDao<Admin, Integer> genericDao) {
            super(genericDao);
            this.adminDao= (AdminDao) genericDao;
        }

        @Override
        @Transactional(propagation = Propagation.REQUIRED)
        public boolean removeAdmin(Integer id) {
            return adminDao.removeAdmin(id);
        }

        @Override
        @Transactional(propagation = Propagation.REQUIRED, readOnly = true)
        public boolean isAdminRegistered(String userName, String password) {
            return adminDao.isAdminRegistered(userName, password);
        }

        @Override
        @Transactional(propagation = Propagation.REQUIRED, readOnly = true)
        public Admin getAdmin(String userName) {
            return adminDao.getAdmin(userName);
        }
}
```

Usage

After all steps successfully implemented, now we can use **AdminService** in a controller:

AdminController Class

```
@Controller
public class AdminController {

        @Autowired(required = true)
        private AdminService adminService;
        @RequestMapping(value = "/index", method = RequestMethod.GET)
        public String viewLogin(Model model) {
            model.addAttribute("admin", new Admin());
            return "/index"
        }
        @RequestMapping(value = "/index", method = RequestMethod.POST)
        public String login(@ModelAttribute("admin") Admin admin, Model model) {
            if (admin.getUsername() != null) {
                Admin registeredAdmin = adminService.getAdmin(admin.getUsername());
                if(registeredAdmin!=null){
                    model.addAttribute("message", "Welcome "+admin.getUsername());
                    model.addAttribute("messageType","information");
                }else{
                    model.addAttribute("message", "User not found");
                    model.addAttribute("messageType","warning");
                }
            } else {
                model.addAttribute("message", "User not found");
                model.addAttribute("messageType","warning");
            }
            return "/index"
        }
}
```

Result

Generic DAO and Service implementation combine common database operations. Therefore we should use DAO design pattern in every enterprise application.