Spring Integration

For simple implementation of DAO class in JPA base, Spring provides JPA Templates such as JDBC templates and hibernate templates. However, in JPA, how to directly use Method of Entity Manager (**plain JPA**) is also guided. We'll explain how to set and use two methods. Spring JPA

Basic Configuration

In order to use JPA under Spring, persistence.xml and ApplicationContext file configuration is required.

persistence.xml configuration

```
<persistence-unit name="HSQLMUnit" transaction-type="RESOURCE_LOCAL">
    // implement Hibernate
    <provider>org.hibernate.ejb.HibernatePersistence</provider>

    // Entity Class List
    <class>egovframework.sample.model.bidirection.User</class>
    <class>egovframework.sample.model.bidirection.Role</class>
    <class>egovframework.sample.model.bidirection.Department</class>
    <exclude-unlisted-classes/>

    <properties>
        // HSQL setup for other setup for each DBMS.
        <property name="hibernate.dialect" value="org.hibernate.dialect.HSQLDialect"/>
        <property name="hibernate.show_sql" value="true"/>
        <property name="hibernate.format_sql" value="true"/>
        <property name="hibernate.hbm2ddl.auto" value="create-drop"/>
    </properties>
</persistence-unit>
```

Above Entity Class List and following <exclude-unlisted-classes/> sets to recognize the listing of entity class in project as entity, and dialect setting is separate setting per DBMS. In above example, HSQL setting.

Application Context Configuration

```
    // 1.Transation Manager configuration
    <bean id="transactionManager" class="org.springframework.orm.jpa.JpaTransactionManager">
        <property name="entityManagerFactory" ref="entityManagerFactory"/>
    </bean>

    // 2.Entity Manager Factory configuration
    <bean id="entityManagerFactory"
class="org.springframework.orm.jpa.LocalContainerEntityManagerFactoryBean">
        <property name="persistenceUnitName" value="HSQLMUnit"/>
        <property name="persistenceXmlLocation" value="classpath:META-
INF/persistHSQLMemDB.xml"/>
        <property name="dataSource" ref="dataSource"/>
    </bean>

     // 3.DataSource configuration
    <bean id="dataSource" class="org.apache.commons.dbcp.BasicDataSource" destroy-
method="close">
        <property name="driverClassName" value="net.sf.log4jdbc.DriverSpy"/>
        <property name="url" value="jdbc:log4jdbc:hsqldb:mem:testdb"/>
        <property name="username" value="sa"/>
        <property name="password" value=""/>
        <property name="defaultAutoCommit" value="false"/>
    </bean>

    // 4.JPA Annotation configuration
    <bean class="org.springframework.orm.jpa.support.PersistenceAnnotationBeanPostProcessor"/>

    // 5.Annotation configuration
    <context:component-scan base-package="egovframework"/>

    // 6.Annotation based Transaction activation setup
    <tx:annotation-driven />
```

The above example shows 1.Transation Manager setting, 2.Entity Manager Factory setting, 3.DataSource setting, 4.JPA Annotation use setting, 5.Annotation use setting, 6.Annotation based Transaction activation setting. No. 1~4 is the setting for JPA. When writing, the part requiring changes are 2,3,5   and No. 2 is persistence.xml file location and persistenceUnitName setting, No. 3 is DataSource setting for DBMS connection and No. 5 is package setting.

Using JPATemplate

The JpaDaoSupport can be inherited in Spring and call entity methods through get JpaTemplate().

DAO Class Source

```
public class UserDAO extends JpaDaoSupport {
    // Designate Entity Manager Factory set in Application Context to set EntityManagerFactory of
parent.
    @Resource(name="entityManagerFactory")
    public void setEMF(EntityManagerFactory entityManagerFactory) {
        super.setEntityManagerFactory(entityManagerFactory);
    }

    // Input by getTemplate()
    public void createUser(User user) throws Exception {
        this.getJpaTemplate().persist(user);
    }

    // Inquiry by getTemplate()
    public User findUser(String userId) throws Exception {
        return (User) this.getJpaTemplate().find(User.class, userId);
    }

    // query by getTemplate() .. find method supported
    public List findUserListAll() throws Exception {
        return this.getJpaTemplate().find("FROM User user ORDER BY user.userName");
    }

    // Deletion by getTemplate()
    public void removeUser(User user) throws Exception {
        this.getJpaTemplate().remove(this.getJpaTemplate().getReference(User.class,
user.getUserId()));
    }

    // Updating by getTemplate()
    public void updateUser(User user) throws Exception {
        this.getJpaTemplate().merge(user);
    }
}
```

Above example shows that the function is implemented through this.getJpaTemplate().method() after inheriting JpaDaoSupport.

Entity Class Source

```
@Entity
public class User implements Serializable {
    private static final long serialVersionUID = -8077677670915867738L;

    @Id
    @Column(name = "USER_ID", length=10)
    private String userId;

    @Column(name = "USER_NAME", length=20)
    private String userName;

    @Column(length=20)
    private String password;

    ...
}
```

Above example is the part of User Entity Class source used in DAO class.

Using Plain JPA

The entity method from entity manager defined in JPA. Can minimize dependence on spring under Spring environment by working through Entity Manager.

DAO Class Source

```
public class RoleDAO {
    // It is possible to define in 4. JPA Annotation Configuration in Application Context. Entity manager
is designated based on annotation.
    @PersistenceContext
    private EntityManager em;

    // Input through EntityManager
    public void createRole(Role role) throws Exception {
        em.persist(role);
    }

    // Inquire through EntityManager
    public Role findRole(String roleId) throws Exception {
        return (Role) em.find(Role.class, roleId);
    }

    // Query through EntityManager
    public List findRoleListAll() throws Exception {
        Query query = em.createQuery("FROM Role role ORDER BY role.roleName");
        return   query.getResultList();
    }

    // Delete through EntityManager
    public void removeRole(Role role) throws Exception {
        em.remove(em.getReference(Role.class, role.getRoleId()));
    }

    // Update through EntityManager
    public void updateRole(Role role) throws Exception {
        em.merge(role);
    }
}
```

The above example shows that the function was implemented using method of Entity Manager

Entity Class Source

```
@Entity
public class Role implements Serializable {

    private static final long serialVersionUID = 1042037005623082102L;

    @Id
    @Column(name = "ROLE_ID", length=10)
    private String roleId;

    @Column(name = "ROLE_NAME", length=20)
    private String roleName;

    @Column(name = "DESC" , length=50)
    private String desc;
    ...
}
```

The above example is the part of Role Entity Class source used in DAO class