Tapestry51SpringJpaIntegration

Goal

The end result should be a Tapestry 5 (5.1 in this case) application, making use of Spring with Spring Transactions, and JPA. This example will be an incredibly simplified phone book application.

Updating web.xml

To start we need to add our Spring XML files to the context-params of web.xml:

```
<context-param>
    <param-name>contextConfigLocation</param-name>
    <param-value>/WEB-INF/applicationContext.xml</param-value>
</context-param>
```

Then we need to update the Tapestry 5 filter to be org.apache.tapestry5.spring.TapestrySpringFilter

Finally we want to open our Entity Manager on page requests, so we add a filter (before the Tapestry filter) and filter-mapping:

```
<filter>
    <filter-name>JpaFilter</filter-name>
    <filter-class>org.springframework.orm.jpa.support.OpenEntityManagerInViewFilter</filter-class>
</filter>
<filter-mapping>
    <filter-name>JpaFilter</filter-name>
    <url-pattern>/*</url-pattern>
</filter-mapping>
```

Spring Beans XML File

We would like to use JPA exception translation, annotations, and transactions. We also need a Dao for our Person object.

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:context="http://www.springframework.org/schema/context"
       xmlns:jee="http://www.springframework.org/schema/jee"
       xmlns:tx="http://www.springframework.org/schema/tx"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema
/beans/spring-beans-2.5.xsd
                           http://www.springframework.org/schema/context http://www.springframework.org/schema
/context/spring-context-2.5.xsd
                           http://www.springframework.org/schema/jee http://www.springframework.org/schema/jee
/spring-jee-2.5.xsd
                           http://www.springframework.org/schema/tx http://www.springframework.org/schema/tx
/spring-tx-2.5.xsd">
    <bean class="org.springframework.dao.annotation.PersistenceExceptionTranslationPostProcessor" />
    <context:annotation-config />

    <bean id="entityManagerFactory" class="org.springframework.orm.jpa.LocalEntityManagerFactoryBean">
        <property name="persistenceUnitName" value="tapestryUnit" />
    </bean>
    <!-- Switch to this in JEE -->
    <!-- <jee:jndi-lookup id="entityManagerFactory" jndi-name="persistence/tapestryUnit" /> -->

    <bean id="transactionManager" class="org.springframework.orm.jpa.JpaTransactionManager">
        <property name="entityManagerFactory" ref="entityManagerFactory" />
    </bean>

    <tx:annotation-driven transaction-manager="transactionManager" />

    <bean id="personDao" class="com.company.myproject.dao.impl.PersonDaoImpl" />
</beans>
```

persistence.xml

Our persistence.xml is the same as any other. In this case we are not deploying to a JEE container, but just a web container, so we've defined the connection properties inside persistence.xml. We typically would use connection pooling, but for simplicity we have not. Note we will use Hibernate (though we don't have to!), and DB2.

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence xmlns="http://java.sun.com/xml/ns/persistence" version="1.0">
    <persistence-unit name="tapestryUnit" transaction-type="RESOURCE_LOCAL">
        <provider>org.hibernate.ejb.HibernatePersistence</provider>
        <class>com.company.myproject.entities.Person</class>
        <exclude-unlisted-classes />

        <properties>
            <property name="hibernate.dialect" value="org.hibernate.dialect.DB2Dialect" />
            <property name="hibernate.connection.driver_class" value="com.ibm.db2.jcc.DB2Driver" />
            <property name="hibernate.connection.username" value="tapestry" />
            <property name="hibernate.connection.password" value="tapestry" />
            <property name="hibernate.connection.url" value="jdbc:db2://dbserver:50000/TESTS:
currentSchema=PHNPRJ;" />
            <property name="hibernate.show_sql" value="true" />
        </properties>
    </persistence-unit>
</persistence>
```

The DAO

Our DAO is just like any other JPA with Spring annotations DAO.

The entity the DAO will control is:

```
@Entity
@Table(name = "PERSON")
public class Person implements Serializable {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "ID")
    private Long id;

    @Version
    @Column(name = "VER")
    private Integer version;

    @Column(name = "FIRSTNAME", length = 15, nullable = false)
    private String firstName;

    @Column(name = "LASTNAME", length = 15, nullable = false)
    private String lastName;

    @Column(name = "PHONENR", length = 10)
    private String phoneNumber;

    // ... SNIPPED GETTERS/SETTERS ...
```

Note that as I'm using an identity column, the database must be created properly. hbm2ddl does not do this. Use this DDL:

```
CREATE TABLE PHNPRJ.PERSON (
    ID INT NOT NULL GENERATED ALWAYS AS IDENTITY,
    VER INT NOT NULL DEFAULT 0,
    FIRSTNAME VARCHAR(15) NOT NULL DEFAULT '',
    LASTNAME VARCHAR(15) NOT NULL DEFAULT '',
    PHONENR VARCHAR(10) DEFAULT '',
    CONSTRAINT PERSON_PK PRIMARY KEY (ID)
);
```

The DAO interface:

```
public interface PersonDao {
    public List<Person> getPeople();
    public Person read(Long id);
    public Person save(Person person);
    public void delete(Long id);
    public void delete(Person person);
}
```

The DAO:

```
@Repository
@Transactional(readOnly = true, propagation = Propagation.SUPPORTS)
public class PersonDaoImpl implements PersonDao {
    @PersistenceContext
    private EntityManager entityManager;

    public List<Person> getPeople() {
        return entityManager.createQuery("from Person p order by p.lastName, p.firstName, p.phoneNumber").
getResultList();
    }

    public Person read(Long id) {
        return (Person)entityManager.createQuery("from Person p where p.id = :id").setParameter("id", id).
getSingleResult();
    }

    @Transactional(readOnly = false, propagation = Propagation.REQUIRED)
    public Person save(Person person) {
        if (person.getId() == null || person.getId() == 0) {
            entityManager.persist(person);
        } else {
            person = entityManager.merge(person);
        }
        return person;
    }

    @Transactional(readOnly = false, propagation = Propagation.REQUIRED)
    public void delete(Long id) {
        entityManager.remove(read(id));
    }

    @Transactional(readOnly = false, propagation = Propagation.REQUIRED)
    public void delete(Person person) {
        entityManager.remove(person);
    }

    public void setEntityManager(EntityManager entityManager) {
        this.entityManager = entityManager;
    }
}
```

The Rest

Everything else is standard Tapestry. To get our DAO in a T5 page:

```
@Inject
private PersonDao personDao
```

We do not need anything special in our AppModule.

Note do not include the tapestry-hibernate module as we are not using hibernate.cfg.xml.