Mindtree

Welcome to possible

Spring and JPA Integration

Mindtree

© Mindtree limited 2012

Objectives

- Understand Spring and JPA integration
- Understand how to apply declarative transaction

No More Spring Templates

- From Spring 3.1 version, the **JpaTemplate** and the corresponding **JpaDaoSupport** have been **deprecated** in favour of using the native Java Persistence API.
- Also, both of these classes are only relevant for JPA 1
- As a consequence, it is now best practice to **use the Java Persistence API directly** instead of the **JpaTemplate**, which will effectively decouple the DAO layer implementation from Spring entirely.

LocalContainerEntityManagerFactoryBean

- FactoryBean that creates a JPA EntityManagerFactory according to JPA's standard container bootstrap contract.
- This is the most powerful way to set up a shared JPA EntityManagerFactory in a Spring application context.
- The EntityManagerFactory can then be passed to JPA-based DAOs via dependency injection.
- Note that switching to a JNDI lookup or to a LocalEntityManagerFactoryBean definition is just a matter of configuration!

The pom.xml file dependencies to be added are listed

- Properties specifying hibernate and Spring versions

```
<properties>
    <hibernate.version>4.3.5.Final</hibernate.version>
    <spring-framework.version>4.1.4.RELEASE</spring-framework.version>
    <logback.version>1.0.13</logback.version>
    <slf4j.version>1.7.5</slf4j.version>
    <commons-lang3.version>3.3.2</commons-lang3.version>
</properties>
```

The pom.xml file dependencies to be added are listed

- JPA persistence provider configuration

```
<!-- Hibernate -->
<dependency>
    <groupId>org.hibernate</groupId>
    <artifactId>hibernate-core</artifactId>
    <version>${hibernate.version}</version>
</dependency>
<dependency>
    <groupId>org.hibernate</groupId>
    <artifactId>hibernate-entitymanager</artifactId>
    <version>${hibernate.version}</version>
</dependency>

<!-- for proxy -->
<dependency>
    <groupId>org.javassist</groupId>
    <artifactId>javassist</artifactId>
    <version>3.15.0-GA</version>
</dependency>
```

The pom.xml file dependencies to be added are listed

- Spring context and Transaction configuration

```
<!-- Spring context module -->
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-context</artifactId>
    <version>${spring-framework.version}</version>
</dependency>
<!--  Spring Transaction module -->
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-tx</artifactId>
    <version>${spring-framework.version}</version>
</dependency>
<!--  Spring ORM module -->
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-orm</artifactId>
    <version>${spring-framework.version}</version>
</dependency>
```

The pom.xml file dependencies to be added are listed

- Database drivers and loggers

```
<dependency>
    <groupId>mysql</groupId>
    <artifactId>mysql-connector-java</artifactId>
    <version>5.1.34</version>
    <!-- <scope>runtime</scope> -->
</dependency>
<!-- Logging with SLF4J & LogBack -->
<dependency>
    <groupId>org.slf4j</groupId>
    <artifactId>slf4j-api</artifactId>
    <version>${slf4j.version}</version>
    <scope>compile</scope>
</dependency>
<dependency>
    <groupId>ch.qos.logback</groupId>
    <artifactId>logback-classic</artifactId>
    <version>${logback.version}</version>
    <scope>runtime</scope>
</dependency>
```

Spring JPA integration Configuration

- Create persistence-mysql.properties file in resources

persistence-mysql.properties

```
# jdbc.X
jdbc.driverClassName=com.mysql.jdbc.Driver
jdbc.url=jdbc:mysql://localhost:3306/sample_db?createDatabaseIfNotExist=true
jdbc.user=root
jdbc.pass=Welcome123

# hibernate.X
hibernate.dialect=org.hibernate.dialect.MySQL5Dialect
hibernate.show_sql=true
hibernate.hbm2ddl.auto=update
```

Spring JPA integration Configuration

- The applicationContext.xml

```
<!--  enable annotation configuration -->
<context:annotation-config />

<!-- Specify the base package for the beans to be scanned  -->
<context:component-scan base-package="com.mindtree.kalinga" />

<!-- properties file configuration -->
<context:property-placeholder location="classpath:persistence-mysql.properties" />

<!--  configure datasource -->
<bean id="dataSource"
    class="org.springframework.jdbc.datasource.DriverManagerDataSource">
    <!--  values from properties file -->
    <property name="driverClassName" value="${jdbc.driverClassName}" />
    <property name="url" value="${jdbc.url}" />
    <property name="username" value="${jdbc.user}" />
    <property name="password" value="${jdbc.pass}" />
</bean>
```

Spring JPA integration Configuration

- The applicationContext.xml contd.

```
<!--  Configure EntityManagerFactcory   -->
<bean id="myEmf"
    class="org.springframework.orm.jpa.LocalContainerEntityManagerFactoryBean">
    <property name="dataSource" ref="dataSource" />
    <!--  specify the package where entity classes reside  -->
    <property name="packagesToScan" value="com.mindtree.kalinga.entity" />
    <!-- specify the Persistence Provider -->
    <property name="jpaVendorAdapter">
        <bean class="org.springframework.orm.jpa.vendor.HibernateJpaVendorAdapter" />
    </property>
    <!--  additional properties of Persistence provider supported from JPA 2 -->
    <property name="jpaProperties">
        <props>
            <prop key="hibernate.hbm2ddl.auto">${hibernate.hbm2ddl.auto}</prop>
            <prop key="hibernate.dialect">${hibernate.dialect}</prop>
        </props>
    </property>
</bean>

<!--  Specify the TransactionManger used for declarative transaction -->
<bean id="transactionManager" class="org.springframework.orm.jpa.JpaTransactionManager">
    <property name="entityManagerFactory" ref="myEmf" />
</bean>
```

DAO layer interface

```java
/**
 *
 * @author Banu Prakash
 * interface for PRODUCT CRUD operations
 */
public interface ProductDao {
    /**
     * Method to return all products present in the database
     * @return all products
     */
    List<Product> getProducts();
    /**
     * Fetch Product based on product id
     * @param id id of the product
     * @return product whose id is "id"
     */
    Product getProduct(int id);
    /**
     * persist a product to database
     * @param product product to be persisted
     */
    void addProduct(Product product);
}
```

Dao Layer implementation

```
@Repository
public class ProductDaoJpaImpl implements ProductDao {
    /*
     * inject persistence context
     */
    @PersistenceContext
    private EntityManager manager;

    @Override
    public List<Product> getProducts() {
        TypedQuery<Product> query =
                manager.createQuery("select p from Product p",Product.class);
        return query.getResultList();
    }

    @Override
    public void addProduct(Product product) {
        manager.persist(product);
    }

    @Override
    public Product getProduct(int id) {
        return manager.find(Product.class, id);
    }
}
```

Service Layer

- @EnableTransactionManagement /* Enables Spring's annotation-driven transaction management capability */
- @Transactional /* declaratively control transaction boundaries on CDI managed beans */

```
@Service
@EnableTransactionManagement
public class OrderServiceImpl implements OrderService {

    @Autowired
    private ProductDao productDao;

    @Override
    public List<Product> getProducts() {
        return productDao.getProducts();
    }

    @Transactional
    @Override
    public void addProduct(Product product) {
        productDao.addProduct(product);
    }

    @Override
    public Product getProduct(int id) {
        return productDao.getProduct(id);
    }

}
```

Controller class

- There is no change in how you inject Service layer into Spring MVC controller classes.

Mindtree

Welcome to possible

Contact Person

Contact_ contact@mindtree.com

+Country code-Phone

www.mindtree.com

© Mindtree limited 2012