DATABASE ACCESS WITH SPRING

By High School Technology Services
www.myhsts.org

Audience

- Those familiar with Java and its syntaxes(If you are new to Java, please go through the Java course at www.myhsts.org)
- Those interested in building web or console applications using Java

Brief Recap

- Review all the concepts that we have covered in the previous chapter including Value Injection, Constructor Injection and Qualifiers.
- Answer any outstanding questions.

Objective

- Overview of Spring Database Support
- Configuring a DataSource
- Using Spring with Hibernate
- Using Spring with JPA

Materials

- These Powerpoint Slides

Overview of Spring Database Support

- Spring Framework provides an excellent support for handling Database related operations. It simplifies the work of connecting to the database and many other operations which would reduce the extra work from developer's side.
- Spring JDBC does take care of opening the connection, preparing and executing the statement, setting up the loops to iterate through the results (if any), processing any exceptions, handling transactions and finally closing the connection, statement and resultset.
- With all the above operations performed by spring, developer needs to take care of defining connection parameters, specifying SQL statements, declaring parameters and providing the values to them and also doing work for any iteration.

Configuring a DataSource

- The DataSource should always be configured as a bean in the Spring IoC container. In the first case the bean is given to the service directly; in the second case it is given to the prepared template.

```
DriverManagerDataSource dataSource = new DriverManagerDataSource();
dataSource.setDriverClassName("org.hsqldb.jdbcDriver");
dataSource.setUrl("jdbc:hsqldb:hsql://localhost:");
dataSource.setUsername("sa");
dataSource.setPassword("");
```

- XML Configuration :

```
<bean id="dataSource" class="org.springframework.jdbc.datasource.DriverManagerDataSource">
    <property name="driverClassName" value="${jdbc.driverClassName}"/>
    <property name="url" value="${jdbc.url}"/>
    <property name="username" value="${jdbc.username}"/>
    <property name="password" value="${jdbc.password}"/>
</bean>

<context:property-placeholder location="jdbc.properties"/>
```

Using Spring with Hibernate

High Level Hibernate Overview

- Hibernate is an Object-Relational Mapping(ORM) solution for JAVA and it raised as an open source persistent framework created by Gavin King in 2001. It is a powerful, high performance Object-Relational Persistence and Query service for any Java Application.
- Hibernate takes care of mapping Java classes to database tables using XML files and without writing any line of code.
- Hibernate does not require an application server to operate.
- Hibernate manipulates Complex associations of objects of your database.
- Hibernate supports HSQL Database Engine, MySQL, Oracle and Informix Dynamic Server.
- Supported technologies include Maven, J2EE

Using Spring with Hibernate

SessionFactory configuration

- We can setup the sessionfactory bean as shown in the following display.

```
private static SessionFactory factory;
public static void main(String[] args) {
   try{
      factory = new Configuration().configure().buildSessionFactory();
   }catch (Throwable ex) {
      System.err.println("Failed to create sessionFactory object." + ex);
      throw new ExceptionInInitializerError(ex);
   }
```

Using Spring with Hibernate

LocalSessionFactoryBean configuration

- We can setup the LocalSessionFactory bean as shown in the following display.

```
@Bean
public LocalSessionFactoryBean sessionFactory() {
   LocalSessionFactoryBean sessionFactory = new LocalSessionFactoryBean();
   sessionFactory.setDataSource(restDataSource());
   sessionFactory.setPackagesToScan(
     new String[] { "org.hsts.spring.persistence.model" });
   sessionFactory.setHibernateProperties(hibernateProperties());

   return sessionFactory;
}
```

Using Spring with Hibernate

Contextual Sessions and Spring Hibernate

- Contextual Session is a mapping of a current Session to a user's Context.
- A Hibernate interface CurrentSessionContext is there to map a current session (e.g. SessionFactory.getCurrentSession()) to different contexts. This interface has 3 implementations:
- JTASessionContext: current sessions are tracked and scoped by a JTA transaction.
- ThreadLocalSessionContext: current sessions are tracked by thread of execution.
- ManagedSessionContext: current sessions are tracked by thread of execution. However, you are responsible to bind and unbind a Session instance with static methods on this class: it does not open, flush, or close a Session.

Using Spring with JPA

Managing the EntityManager(EM)

- EntityManager holds the properties of the entities in the application, and EntityManagerFactoryBean would store all the EntityManager in the code. We would create the EntityManagerFactory in the following way.
- We would setup the EntityManager in the repository layer.
- This EntityManager would be under @PersistenceContext.

```
@PersistenceContext
private EntityManager em;

public Reading findOneReading(String readingID) {
    TypedQuery<Reading> query = em.createNamedQuery("Reading.findOneById",Reading.class);
    query.setParameter("paramReadingID",readingID);

    List<Reading> listofreadings = query.getResultList();

    if(listofreadings.size()!=0 && listofreadings.size()==1)
    {
        return listofreadings.get(0);
    }
    else
        return  null;
}
```

Using Spring with JPA

LocalContainerEntityManagerFactoryBean and Container-managed EMs

- LocalContainerEntityManager could be used to manage all the EntityManagers.

```
@Bean
public LocalContainerEntityManagerFactoryBean emf()
{
    LocalContainerEntityManagerFactoryBean emf = new LocalContainerEntityManagerFactoryBean();
    emf.setDataSource(provideDataSource());
    emf.setJpaVendorAdapter(new HibernateJpaVendorAdapter());
    emf.setPackagesToScan("org.hsts.entity");

    Properties properties = new Properties();
    properties.put("hibernate.dialect", "org.hibernate.dialect.MySQL57Dialect");
    properties.put("hibernate.hbm2ddl.auto", "validate");
    properties.put("hibernate.show_sql", "true");
    emf.setJpaProperties(properties);
    return emf;

}

@Bean
public DataSource provideDataSource()
{
    DriverManagerDataSource ds = new DriverManagerDataSource();
    ds.setDriverClassName("com.mysql.cj.jdbc.Driver");
    ds.setUrl("jdbc:mysql://localhost:3306/cartracker_db?useSSL=false");
    ds.setUsername("root");
    ds.setPassword("root");
    return  ds;
}
```

Using Spring with JPA

JEE and JNDI Lookup of the EM

- The JNDIDataSource lookup can be done as displayed using JndiDataSourceLookup keyword.

```
@Bean
    public DataSource dataSource() {
        final JndiDataSourceLookup dsLookup = new JndiDataSourceLookup();
        dsLookup.setResourceRef(true);
        DataSource dataSource = dsLookup.getDataSource("jdbc/yourJdbcGoesHere");
        return dataSource;
    }
```

Using Spring with JPA

Configuration and Vendor Adaptors

- We can set the configuration and the vendor information in the java.util.Properties keyword.

```
Properties properties = new Properties();
properties.put("hibernate.dialect", "org.hibernate.dialect.MySQL57Dialect");
properties.put("hibernate.hbm2ddl.auto", "validate");
properties.put("hibernate.show_sql", "true");
emf.setJpaProperties(properties);
```

- We can set the properties later to the EntityManagerFactoryBean using the setter method setJpaProperties(properties).

```
emf.setJpaVendorAdapter(new HibernateJpaVendorAdapter());
```

Using Spring with JPA

Creating a JPA Repository/DAO Bean - @PersistenceUnit, @PersistenceContext

- We can create a JPA Repository as the following display,

```
public interface UserRepository extends Repository<User, Long> {

  List<User> findByEmailAddressAndLastname(String emailAddress, String lastname);
}
```

- Here the interface UserRepository extends Repository class which has the entity called ‘User’ and the id field type Long.
- We can use the @PersistenceContext to maintain the persistence of the EntityManager as displayed below.

```
@PersistenceContext
private EntityManager em;
```


Q&A

Next Chapter

- Overview of AOP
- Spring AOP Introduction
- Pointcut Expressions and Advice
- Marker Annotations (Rubber Stamp AOP)
- @AspectJ Based AOP Support
- Other Considerations

Assignment

- Describe about LocalSessionFactoryBean and why it is used in Spring.
- Explain what is contextual session.
- Brief about the functionalities provided by @PersistenceContext.