**Spring MVC Hibernate MySQL Integration CRUD Example Tutorial**

<https://www.jianshu.com/p/0a8027c48757>

2016.08.26 09:52:36字数 983阅读 355

# Spring MVC Hibernate MySQL Integration CRUD Example Tutorial

We learned how to integrate Spring and Hibernate in our last tutorial. Today we will move forward and integrate Spring MVC and Hibernate frameworks in a web application CRUD example.

Our final project structure looks like below image, we will look into each of the componentsone by one.

Alt text

Spring-MVC-Hibernate-Example

Note that I am using Spring 4.0.3.Release and Hibernate 4.3.5.Final versions for our example, same program is also compatible for Spring 4 and Hibernate 3, however you need to make small changes in spring bean configuration file discussed in the last tutorial.

### Maven Dependencies

Let’s look at all the maven dependencies are required for hibernate and spring mvc framework integration.

pom.xml

Some of the dependencies above are included by STS (Spring Tool Suite) when I create Spring MVC project. Important dependencies above are **spring-context, spring-webmvc, spring-tx, hibernate-core, hibernate-entitymanager and spring-orm.** I am using Apache Commons DBCP for connection pooling, but in real life situations, most probably you have connection pooling done by the container and all we need is to provide the JNDI reference details to use.

NOTE: I noticed that some of the readers are getting database connection issues. Notice that in my pom.xml, there is no database driver. That works for me because I have MySQL driver in tomcat lib directory and some DataSource connections configured with it. For any database connection related issues, either put the database driver in container lib or include that in pom.xml dependencies.

### Deployment Descriptor

We need to plugin spring framework in our web application, that is done by configuring Spring framework **DispatcherServlet** as front controller. Our web.xml file looks like below.

web.xml

Most of the part is boiler plate code, most important part is the **spring context file location** where we will configure our **spring beans** and **services**. If you want, you can change them according to your project requirements.

### Hibernate Entity Bean

We are using **JPA annotations** in our entity bean class, however we can also have a simple java bean and mapping details in the xml file. In that case, we need to provide mapping file details while configuring **Hibernate SessionFactory** in spring bean configurations.

Person.java

Our entity bean maps to PERSON table in MySQL database, notice that I have not annotated “name” and “country” fields with @Column annotation because they are of same name. Below SQL script shows the table details.

person.sql

CREATE TABLE `Person` (

`id` int(11) unsigned NOT NULL AUTO\_INCREMENT,

`name` varchar(20) NOT NULL DEFAULT '',

`country` varchar(20) DEFAULT NULL,

PRIMARY KEY (`id`)

) ENGINE=InnoDB AUTO\_INCREMENT=1 DEFAULT CHARSET=utf8;

### Hibernate DAO Implementation

We will create PersonDAO **interface** to declare the methods that we will use in our project. Next, we will provide **hibernate specific implementation** for it.

PersonDAO.java

Hibernate specific DAO implementation looks like below.

Notice that I am not using Hibernate Transaction, that is because it will be taken care by Spring framework.

### Spring Service Classes

Here are our **service classes** that are using Hibernate DAO classes to work with Person objects.

Notice that spring declarative transaction management is applied by using @Transactional annotation.

### Spring Controller Class

Our DAO and Service classes are ready, it’s time to write our controller class that will take care of client requests and use service classes to perform database specific operations and then return the view pages

Notice that I am using @Controller annotation, so that Spring framework will treat it as a Controller class to handleclient requests. Also I am using @Autowired and @Qualifier annotations for injecting PersonService, we could have done it in the spring context xml file too.

Recommended Read: Spring Bean Autowiring

### Spring Bean Configuration

Our services are ready, all we need is to wire them through spring bean configurations. Our root-context.xml file is empty, so we will look only into servlet-context.xml file.

dataSource bean is defined for org.apache.commons.dbcp.BasicDataSource class for basic connection pooling.

org.springframework.orm.hibernate4.LocalSessionFactoryBean bean is used for Hibernate 4 SessionFactory. For Hibernate 3, you will find similar classes as org.springframework.orm.hibernate3.LocalSessionFactoryBean and org.springframework.orm.hibernate3.AnnotationSessionFactoryBean.

One important point is that when we are depending on Spring framework for Hibernate Session management, we should not define hibernate.current\_session\_context\_class property, otherwise you will get a lot of session transaction related issues.

personDAO and personService beans are self understood.

transactionManager bean definition for org.springframework.orm.hibernate4.HibernateTransactionManager is required for Spring ORM to support hibernate session transaction management. For Hibernate 3, you will find similar class as org.springframework.orm.hibernate3.HibernateTransactionManager. Spring uses AOP for transaction management, you can now relate it with @Transactional annotation.

Recommended Read: Spring AOP and Spring Transaction Management

### View Page

Our last part of application is the view page, notice the attributes added to Model in Controller handler methods, we will use them to create our view page. We will also use JSTL tags, spring core and spring form tags