SPRING JDBC, SPRING MVC

Spring MVC and JDBC CRUD Example

SOUMITRA 9 COMMENTS

Introduction

This tutorial Spring MVC and JDBC CRUD example shows how MVC(Model, View, Controller) works in Spring 4.x. In this tutorial you will also find how JDBC works with Spring MVC. We will also see how annotation like @Autowired works in Spring MVC and JDBC CRUD example. You will also see how datasource is configured in Spring. In this example we will see how to integrate Spring 4, MySQL with Maven 3.

Related Posts:

- Spring MVC and Spring JDBC Example
- Spring MVC and JDBC CRUD with zero XML

Prerequisites

The following configurations are required in order to run the application

Eclipse Kepler
JDK 1.8
Tomcat 8
Have maven 3 installed and configured
Spring 4 dependencies in pom.xml

Example with Source Code

For Spring MVC and JDBC CRUD example, we will create maven based web project in Eclipse.

If you already have an idea on how to create a maven project in Eclipse will be great otherwise I will tell you here how to create a maven project in Eclipse.

Creating Project

Create a maven based web project in Eclipse. Below steps show how to create maven based web application:

Go to File -> New -> Other. On popup window under Maven select Maven Project. Then click on Next. Select the workspace location – either default or browse the location. Click on Next. Now in next window select the row as highlighted from the below list of archtypes and click on Next button.

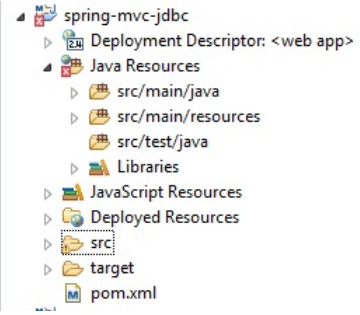
maven-arctype-webapp

Now enter the required fields (Group Id, Artifact Id) as shown below

Group Id : com.roytuts
Artifact Id : spring-mvc-jdbc

Project Structure

The created project looks like below image.



Updating pom.xml

Modify the pom.xml file as shown below. As it is a Spring mvc project so we have added Spring web dependency. We have added MySQL dependency as we are working with MySQL database. We have added jstl and jsp dependencies for jsp pages.

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.roytuts</groupId>
  <artifactId>spring-mvc-jdbc</artifactId>
  <packaging>war</packaging>
  <version>0.0.1-SNAPSHOT</version>
  <url>http://maven.apache.org</url>
  <properties>
    <java.version>1.8</java.version>
    <spring.version>4.1.6.RELEASE</spring.version>
    <mysqlconnector.version>5.1.34</mysqlconnector.version>
  </properties>
  <dependencies>
    <!-- Spring -->
    <dependency>
      <groupId>org.springframework</groupId>
      <artifactId>spring-context</artifactId>
      <version>${spring.version}</version>
    </dependency>
    <dependency>
      <groupId>org.springframework</groupId>
      <artifactId>spring-web</artifactId>
      <version>${spring.version}</version>
    </dependency>
    <dependency>
      <groupId>org.springframework</groupId>
      <artifactId>spring-webmvc</artifactId>
      <version>${spring.version}</version>
    </dependency>
    <dependency>
      <groupId>org.springframework</groupId>
      <artifactId>spring-jdbc</artifactId>
      <version>${spring.version}</version>
    </dependency>
    <dependency>
      <groupId>javax.servlet</groupId>
      <artifactId>javax.servlet-api</artifactId>
      <version>3.1.0</version>
      <scope>provided</scope>
```

```
    </dependency>
    <!-- jstl -->
    <dependency>
      <groupId>javax.servlet</groupId>
      <artifactId>jstl</artifactId>
      <version>1.2</version>
    </dependency>
    <!-- mysql java connector -->
    <dependency>
      <groupId>mysql</groupId>
      <artifactId>mysql-connector-java</artifactId>
      <version>${mysqlconnector.version}</version>
    </dependency>
  </dependencies>
  <build>
    <finalName>spring-mvc-jdbc</finalName>
    <plugins>
      <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-compiler-plugin</artifactId>
        <configuration>
          <source>${java.version}</source>
          <target>${java.version}</target>
        </configuration>
      </plugin>
    </plugins>
  </build>
</project>
```

Configuring Buildpath

If you see JRE System Library[J2SE-1.5] then change the version by below process.

Do right-click on the project and go to Build -> Configure build path, under Libraries tab click on JRE System Library[J2SE-1.5], click on Edit button and select the appropriate jdk 1.8 from the next window. Click on Finish then Ok.

Change also the Compiler compliance level as 1.8 from Java -> Compiler.

Deployment Descriptor – web.xml

Now when the build process finished in Eclipse then modify the web.xml file with below source code.

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="3.1"
  xmlns="http://xmlns.jcp.org/xml/ns/javaee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
  http://xmlns.jcp.org/xml/ns/javaee/web-app_3_1.xsd">
  <!-- dispatcher servlet acts as a front controller for each request/respo
  <servlet>
    <servlet-name>spring-mvc-jdbc</servlet-name>
    <servlet-class>org.springframework.web.servlet.DispatcherServlet</servl
    <!-- load Spring controllers while dispatcher servlet loads -->
    <init-param>
      <param-name>contextConfigLocation</param-name>
      <param-value>classpath:controllers.xml</param-value>
    </init-param>
    <load-on-startup>1</load-on-startup>
```

```
    </servlet>
    <servlet-mapping>
      <servlet-name>spring-mvc-jdbc</servlet-name>
      <url-pattern>/</url-pattern>
    </servlet-mapping>
  </web-app>
```

In the above deployment descriptor, we load the controllers during Dispatcher servlet startup.

Creating Spring Configurations

Create *spring-config.xml* file under *src/main/resources* directory with the below source code. We have declared the annotation support configuration for both transaction and other stereotype like @Service, @Repository etc. We have defined beans for data source, transaction, JDBC template and other custom beans in the below Spring config.

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:p="http://www.springframework.org/schema/p"
  xmlns:context="http://www.springframework.org/schema/context"
  xmlns:tx="http://www.springframework.org/schema/tx"
  xsi:schemaLocation="http://www.springframework.org/schema/beans http://ww
            http://www.springframework.org/schema/context http://www.spring
            http://www.springframework.org/schema/tx http://www.springframe
  <!-- Support annotation -->
  <context:annotation-config />
  <!-- support annotation transaction -->
  <tx:annotation-driven
    transaction-manager="txManager" />
  <!-- declare datasource -->
  <bean id="dataSource" class="org.springframework.jdbc.datasource.DriverMa
    <property name="driverClassName" value="com.mysql.jdbc.Driver" />
    <property name="url" value="jdbc:mysql://localhost:3306/cdcol" />
    <property name="username" value="root" />
    <property name="password" value="" />
  </bean>
  <bean id="txManager"
    class="org.springframework.jdbc.datasource.DataSourceTransactionManager
    <property name="dataSource" ref="dataSource" />
  </bean>
  <!-- spring jdbc template -->
  <bean id="jdbcTemplate"
    class="org.springframework.jdbc.core.JdbcTemplate">
    <property name="dataSource" ref="dataSource"></property>
  </bean>
  <!-- service -->
  <bean id="userDetailService"
    class="com.roytuts.spring.mvc.jdbc.service.impl.UserDetailServiceImpl"
  <!-- dao -->
  <bean id="userDetailDao"
    class="com.roytuts.spring.mvc.jdbc.dao.impl.UserDetailDaoImpl" />
</beans>
```

Create *controllers.xml* file under classpath directory *src/main/resources*. Through this config file we import other Spring config files, we scan all the Spring controller classes and also we define view resolver in order to use jsp pages as presentation layer.

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:p="http://www.springframework.org/schema/p"
  xmlns:context="http://www.springframework.org/schema/context"
  xsi:schemaLocation="
        http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans.xsd
        http://www.springframework.org/schema/context
        http://www.springframework.org/schema/context/spring-context.xsd">
  <import resource="classpath:spring-config.xml" />
  <!-- Scan the package where Spring Controllers are placed -->
  <context:component-scan
    base-package="com.roytuts.spring.mvc.jdbc.controller" />
  <!-- Resolves logical String-based view names to actual View types -->
  <bean id="viewResolver"
    class="org.springframework.web.servlet.view.InternalResourceViewResolve
    <property name="viewClass"
      value="org.springframework.web.servlet.view.JstlView" />
    <!-- Where pages are kept -->
    <property name="prefix" value="/pages/" />
    <!-- What is the page extension -->
    <property name="suffix" value=".jsp" />
  </bean>
</beans>
```

Creating Table

Create MySQL table *user_detail* to store user information.

```
CREATE TABLE `user_detail` (
  `id` int(10) NOT NULL AUTO_INCREMENT,
  `first_name` varchar(20) NOT NULL,
  `last_name` varchar(15) NOT NULL,
  `email` varchar(100) NOT NULL,
  `dob` varchar(16) NOT NULL,
  PRIMARY KEY (`id`)
);
```

Dump some data into the table in order to test the application.

```
insert  into `user_detail`(`id`,`first_name`,`last_name`,`email`,`dob`) val
```

Creating Model Class

Now create POJO class and mapper class which will map Java object to database table *user_detail*.

```
package com.roytuts.spring.mvc.jdbc.model;
public class UserDetail {
  private int id;
  private String firstName;
  private String lastName;
  private String email;
  private String dob;
```

```
  public UserDetail() {
  }
  public UserDetail(int id, String firstName, String lastName, String email
    this.id = id;
    this.firstName = firstName;
    this.lastName = lastName;
    this.email = email;
    this.dob = dob;
  }
  public int getId() {
    return id;
  }
  public void setId(int id) {
    this.id = id;
  }
  public String getFirstName() {
    return firstName;
  }
  public void setFirstName(String firstName) {
    this.firstName = firstName;
  }
  public String getLastName() {
    return lastName;
  }
  public void setLastName(String lastName) {
    this.lastName = lastName;
  }
  public String getEmail() {
    return email;
  }
  public void setEmail(String email) {
    this.email = email;
  }
  public String getDob() {
    return dob;
  }
  public void setDob(String dob) {
    this.dob = dob;
  }
}
```

Creating RowMapper

Mapper class implements Spring's parameterized Rowmapper to provide mapping between database table and Java class.

```
package com.roytuts.spring.mvc.jdbc.rowmapper;
import java.sql.ResultSet;
import java.sql.SQLException;
import org.springframework.jdbc.core.RowMapper;
import com.roytuts.spring.mvc.jdbc.model.UserDetail;
public class UserDetailRowMapper implements RowMapper<UserDetail> {
  public UserDetail mapRow(ResultSet rs, int row) throws SQLException {
    UserDetail userDetail = new UserDetail();
    userDetail.setId(rs.getInt("id"));
    userDetail.setFirstName(rs.getString("first_name"));
```

```
      userDetail.setLastName(rs.getString("last_name"));
      userDetail.setEmail(rs.getString("email"));
      userDetail.setDob(rs.getString("dob"));
      return userDetail;
    }
  }
```

Creating DAO

Create DAO interface for querying database table. This interface is implemented by several client classes in order to provide their own implementations. It is always recommended to write code to interface rather than to class to provide loose coupling between components through dependency injection.

```
package com.roytuts.spring.mvc.jdbc.dao;
import java.util.List;
import com.roytuts.spring.mvc.jdbc.model.UserDetail;
public interface UserDetailDao {
  public UserDetail getUserDetail(int id);
  public List<UserDetail> getAllUserDetail();
  public int addUserDetail(UserDetail userDetail);
  public int updateUserDetail(UserDetail userDetail);
  public int deleteUserDetail(int id);
}
```

Create the corresponding DAO implementation class. Here in the below class we have applied transaction while modifying or writing data to database otherwise data may not be in consistent state in the database. You may also apply transaction while reading data.

```
package com.roytuts.spring.mvc.jdbc.dao.impl;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.jdbc.core.JdbcTemplate;
import org.springframework.jdbc.core.simple.SimpleJdbcInsert;
import org.springframework.transaction.annotation.Transactional;
import com.roytuts.spring.mvc.jdbc.dao.UserDetailDao;
import com.roytuts.spring.mvc.jdbc.model.UserDetail;
import com.roytuts.spring.mvc.jdbc.rowmapper.UserDetailRowMapper;
public class UserDetailDaoImpl implements UserDetailDao {
  @Autowired
  private JdbcTemplate jdbcTemplate;
  @Transactional
  public UserDetail getUserDetail(int id) {
    UserDetail userDetail = (UserDetail) jdbcTemplate.queryForObject("selec
        new Object[] { id }, new UserDetailRowMapper());
    return userDetail;
  }
  @Transactional
  public List<UserDetail> getAllUserDetail() {
    List<UserDetail> userDetail = (List<UserDetail>) jdbcTemplate.query("se
        new UserDetailRowMapper());
    return userDetail;
  }
  @Transactional
  public int addUserDetail(UserDetail userDetail) {
```

```
    SimpleJdbcInsert simpleJdbcInsert = new SimpleJdbcInsert(jdbcTemplate);
    simpleJdbcInsert.withTableName("user_detail").usingGeneratedKeyColumns(
    Map<String, Object> parameters = new HashMap<String, Object>(4);
    parameters.put("first_name", userDetail.getFirstName());
    parameters.put("last_name", userDetail.getLastName());
    parameters.put("email", userDetail.getEmail());
    parameters.put("dob", userDetail.getDob());
    Number insertedId = simpleJdbcInsert.executeAndReturnKey(parameters);
    return insertedId.intValue();
  }
  @Transactional
  public int updateUserDetail(UserDetail userDetail) {
    String sql = "update user_detail set first_name = ?, last_name = ?, ema
    int resp = jdbcTemplate.update(sql, new Object[] { userDetail.getFirstN
        userDetail.getEmail(), userDetail.getDob(), userDetail.getId() });
    return resp;
  }
  @Transactional
  public int deleteUserDetail(int id) {
    int resp = jdbcTemplate.update("delete from user_detail where id = ?",
    return resp;
  }
}
```

Creating Service Class

Create the service interface for processing logic or business logic.

```
package com.roytuts.spring.mvc.jdbc.service;
import java.util.List;
import com.roytuts.spring.mvc.jdbc.model.UserDetail;
public interface UserDetailService {
  public UserDetail getUserDetail(int id);
  public List<UserDetail> getAllUserDetail();
  public int addUserDetail(UserDetail userDetail);
  public int updateUserDetail(UserDetail userDetail);
  public int deleteUserDetail(int id);
}
```

Create the corresponding service implementation class. This service class communicates with DAO layer and gets data and finally applies business processing logic on those data and sends to the controller layer which then pass to the presentation layer for displaying on view.

```
package com.roytuts.spring.mvc.jdbc.service.impl;
import java.util.List;
import org.springframework.beans.factory.annotation.Autowired;
import com.roytuts.spring.mvc.jdbc.dao.UserDetailDao;
import com.roytuts.spring.mvc.jdbc.model.UserDetail;
import com.roytuts.spring.mvc.jdbc.service.UserDetailService;
public class UserDetailServiceImpl implements UserDetailService {
  @Autowired
  private UserDetailDao userDetailDao;
  public UserDetail getUserDetail(int id) {
    return userDetailDao.getUserDetail(id);
  }
  public List<UserDetail> getAllUserDetail() {
```

```
    return userDetailDao.getAllUserDetail();
  }
  @Override
  public int addUserDetail(UserDetail userDetail) {
    return userDetailDao.addUserDetail(userDetail);
  }
  @Override
  public int updateUserDetail(UserDetail userDetail) {
    return userDetailDao.updateUserDetail(userDetail);
  }
  @Override
  public int deleteUserDetail(int id) {
    return userDetailDao.deleteUserDetail(id);
  }
  public UserDetailDao getUserDetailDao() {
    return userDetailDao;
  }
}
```

Creating Controller Class

Create Spring controller class which will handle user request and response.

```
package com.roytuts.spring.mvc.jdbc.controller;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.ui.ModelMap;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.bind.annotation.RequestParam;
import com.roytuts.spring.mvc.jdbc.model.UserDetail;
import com.roytuts.spring.mvc.jdbc.service.UserDetailService;
@Controller
@RequestMapping("/")
public class UserDetailController {
  @Autowired
  private UserDetailService userDetailService;
  @RequestMapping(value = "user/{id}", method = RequestMethod.GET)
  public String getUserDetail(@PathVariable int id, ModelMap userModel) {
    userModel.addAttribute("userDetail", userDetailService.getUserDetail(id
    return "user";
  }
  @RequestMapping(value = "users", method = RequestMethod.GET)
  public String getUsersDetails(ModelMap userModel) {
    userModel.addAttribute("userDetail", userDetailService.getAllUserDetail
    return "users";
  }
  @RequestMapping(value = "addUser")
  public String addPage() {
    return "add";
  }
  @RequestMapping(value = "add/user", method = RequestMethod.POST)
  public String addUser(@RequestParam(value = "fname", required = true) Str
      @RequestParam(value = "lname", required = true) String lname,
      @RequestParam(value = "email", required = true) String email,
```

```
      @RequestParam(value = "dob", required = true) String dob, ModelMap us
    UserDetail userDetail = new UserDetail();
    userDetail.setFirstName(fname);
    userDetail.setLastName(lname);
    userDetail.setEmail(email);
    userDetail.setDob(dob);
    int resp = userDetailService.addUserDetail(userDetail);
    if (resp > 0) {
      userModel.addAttribute("msg", "User with id : " + resp + " added succ
      userModel.addAttribute("userDetail", userDetailService.getAllUserDeta
      return "users";
    } else {
      userModel.addAttribute("msg", "User addition failed.");
      return "add";
    }
  }
  @RequestMapping(value = "delete/user/{id}", method = RequestMethod.GET)
  public String deleteUser(@PathVariable("id") int id, ModelMap userModel)
    int resp = userDetailService.deleteUserDetail(id);
    userModel.addAttribute("userDetail", userDetailService.getAllUserDetail
    if (resp > 0) {
      userModel.addAttribute("msg", "User with id : " + id + " deleted succ
    } else {
      userModel.addAttribute("msg", "User with id : " + id + " deletion fai
    }
    return "users";
  }
  @RequestMapping(value = "update/user/{id}", method = RequestMethod.GET)
  public String updatePage(@PathVariable("id") int id, ModelMap userModel)
    userModel.addAttribute("id", id);
    userModel.addAttribute("userDetail", userDetailService.getUserDetail(id
    return "update";
  }
  @RequestMapping(value = "update/user", method = RequestMethod.POST)
  public String updateUser(@RequestParam int id, @RequestParam(value = "fna
      @RequestParam(value = "lname", required = true) String lname, @Reques
      @RequestParam("dob") String dob, ModelMap userModel) {
    UserDetail userDetail = new UserDetail();
    userDetail.setId(id);
    userDetail.setFirstName(fname);
    userDetail.setLastName(lname);
    userDetail.setEmail(email);
    userDetail.setDob(dob);
    int resp = userDetailService.updateUserDetail(userDetail);
    userModel.addAttribute("id", id);
    if (resp > 0) {
      userModel.addAttribute("msg", "User with id : " + id + " updated succ
      userModel.addAttribute("userDetail", userDetailService.getAllUserDeta
      return "users";
    } else {
      userModel.addAttribute("msg", "User with id : " + id + " updation fai
      userModel.addAttribute("userDetail", userDetailService.getUserDetail(
      return "update";
    }
  }
}
```

Creating Views

Create a directory called *pages* under webapp directory for putting created jsp views. Notice how we have configured this view directory in view resolver in the above spring config.

Displaying Users

Now create *user.jsp* file under webapp/pages directory for displaying single user details.

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
  pageEncoding="ISO-8859-1"%>
<html>
<title>Spring MVC and JDBC CRUD Example</title>
<body>
  <h2>Spring MVC and JDBC CRUD Example</h2>
  Id : ${userDetail.id}
  <br /> First Name : ${userDetail.firstName}
  <br /> Last Name : ${userDetail.lastName}
  <br /> Email : ${userDetail.email}
  <br /> DOB : ${userDetail.dob}
</body>
</html>
```

Create users.jsp file under webapp/pages directory for displaying all users details:

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
  pageEncoding="ISO-8859-1"%>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
<html>
<title>Spring MVC and JDBC CRUD Example</title>
<body>
  <h2>Spring MVC and JDBC CRUD Example</h2>
  <c:if test="${not empty msg}">
        ${msg}
    </c:if>
  <c:choose>
    <c:when test="${userDetail != null}">
      <h3>List of Users</h3>
      <table cellpadding="5" cellspacing="5">
        <thead>
          <tr>
            <th>ID</th>
            <th>First Name</th>
            <th>Last Name</th>
            <th>Email</th>
            <th>DOB</th>
            <th>Actions</th>
          </tr>
        </thead>
        <tbody>
          <c:forEach var="user" items="${userDetail}">
            <tr>
              <td>${user.id}</td>
              <td>${user.firstName}</td>
              <td>${user.lastName}</td>
              <td>${user.email}</td>
              <td>${user.dob}</td>
```

```
              <td><a
                href="<%=request.getContextPath()%>/update/user/${user.id}"
                  <a
                href="<%=request.getContextPath()%>/delete/user/${user.id}"
                onclick="return confirm('Do you really want to delete?')">D
            </tr>
          </c:forEach>
        </tbody>
      </table>
    </c:when>
    <c:otherwise>
        No User found in the DB!
        </c:otherwise>
  </c:choose>
</body>
</html>
```

Create add.jsp file under webapp/pages directory for adding new user.

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
  pageEncoding="ISO-8859-1"%>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
<html>
<title>Spring MVC and JDBC CRUD Example</title>
<body>
  <h2>Spring MVC and JDBC CRUD Example</h2>
  <c:if test="${not empty msg}">
        ${msg}
    </c:if>
  <h3>Add User</h3>
  <form method="POST" name="add_user"
    action="<%=request.getContextPath()%>/add/user">
    Name: <input name="fname" value="${firstName}" type="text" /> <br />
    <br /> Last Name: <input name="lname" value="${lastName}" type="text" />
    <br /> <br /> Email: <input name="email" value="${email}"
      type="text" /><br /> <br /> DOB: <input name="dob" value="${dob}"
      type="text" /><br /> <br /> <input value="Add User" type="submit" />
  </form>
</body>
</html>
```

Create update.jsp file under webapp/pages directory for updating new user.

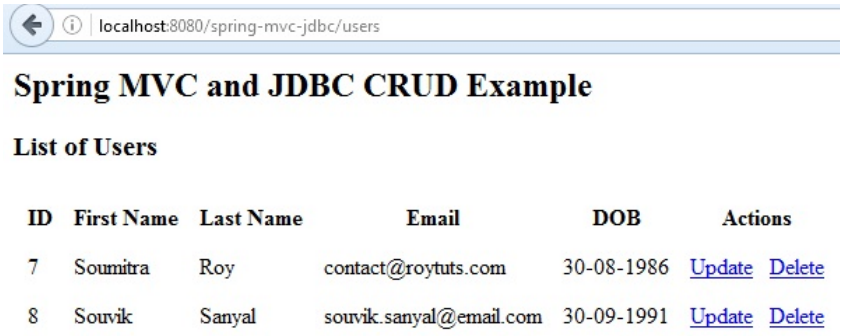
```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
  pageEncoding="ISO-8859-1"%>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
<html>
<title>Spring MVC and JDBC CRUD Example</title>
<body>
  <h2>Spring MVC and JDBC CRUD Example</h2>
  <c:if test="${not empty msg}">
        ${msg}
    </c:if>
  <h3>Update User</h3>
  <form method="POST" name="update_user"
```

```
      action="<%=request.getContextPath()%>/update/user">
      <input hidden="hidden" name="id" value="${id}" type="text" /> First
      Name: <input name="fname" value="${userDetail.firstName}" type="text" /
      <br />
      <br /> Last Name: <input name="lname" value="${userDetail.lastName}"
        type="text" /> <br /> <br /> Email: <input name="email"
        value="${userDetail.email}" type="text" /><br />
      <br /> DOB: <input name="dob" value="${userDetail.dob}" type="text" /><
      <br /> <input value="Update User" type="submit" />
    </form>
  </body>
  </html>
```

Now we have finished writing code for Spring MVC and JDBC CRUD example. So let's test the application.

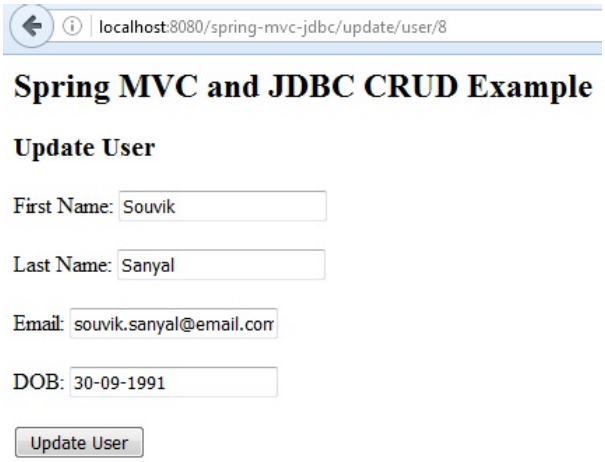
Testing the Application

Now run the application on Tomcat server 8 and when the application successfully deployed onto the server, please hit the URL *http://localhost:8080/spring-mvc-jdbc/users* , you will below output in the browser.

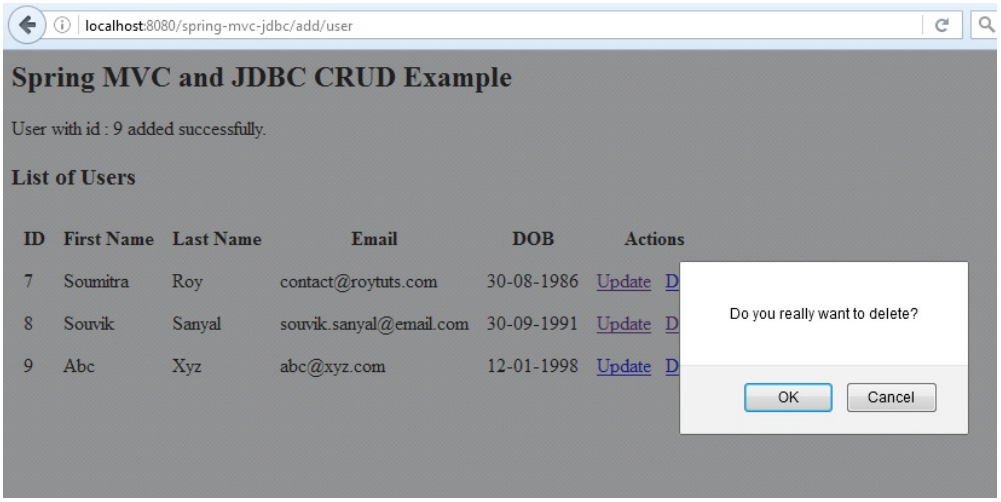


When you hit the URL http://localhost:8080/spring-mvc-jdbc/user in the browser, you will see the below output.

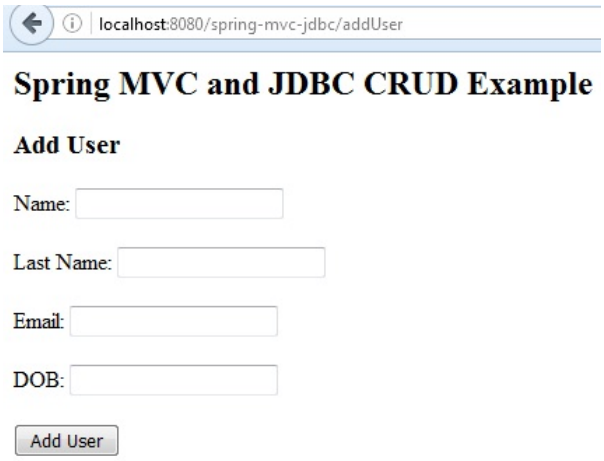
When you click on the update link on page http://localhost:8080/spring-mvc-jdbc/users in the browser, you will see the below output



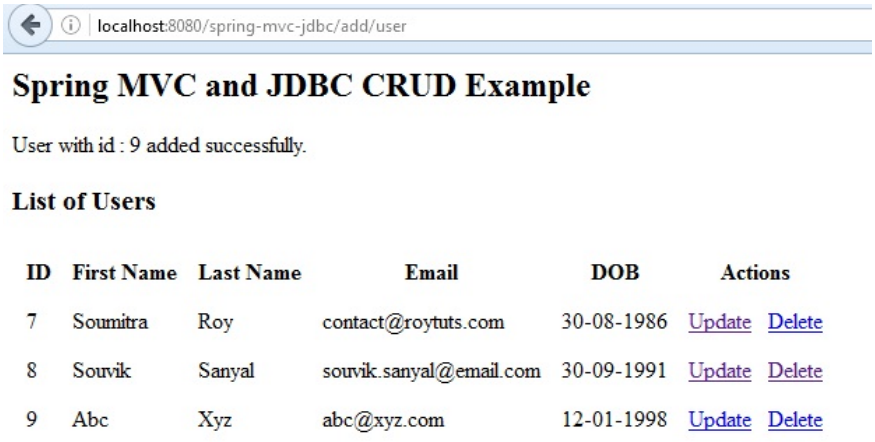
When you click on the delete link on page http://localhost:8080/spring-mvc-jdbc/users in the browser, you will see the below output



When you hit the URL http://localhost:8080/spring-mvc-jdbc/addUser in the browser, you will see the below output



After addition of user the page navigates automatically to the all users page



The newly added user information is inserted into database

id	first_name	last_name	email	dob
7	Soumitra	Roy	contact@roytuts.com	30-08-1986
8	Souvik	Sanyal	souvik.sanyal@email.com	30-09-1991
9	Abc	Xyz	abc@xyz.com	12-01-1998
(Auto)	(NULL)	(NULL)	(NULL)	(NULL)

Hope you will be able to apply similar concept to your own project after completing this Spring MVC and JDBC CRUD example.

Source Code

Download Source Code

Thanks for reading.

Tagged Spring Annotation, Spring JdbcTemplate, Transaction

← foreach loop example in Mule ESB

JPA CRUD Example →

9 thoughts on "Spring MVC and JDBC CRUD Example"

Bijoy karmakar says:
9th May, 2019 at 6:28 am

I am getting error :
SEVERE: Servlet [spring-mvc-jdbc] in web application [/spring-mvc-jdbc] threw load() exception
java.lang.ClassNotFoundException: org.springframework.web.servlet.DispatcherServlet
at org.apache.catalina.loader.WebappClassLoaderBase.loadClass(WebappClassLoaderBase.java:1274)
at org.apache.catalina.loader.WebappClassLoaderBase.loadClass(WebappClassLoaderBase.java:1108)
at org.apache.catalina.core.DefaultInstanceManager.loadClass(DefaultInstanceManager.java:520)
at
org.apache.catalina.core.DefaultInstanceManager.loadClassMaybePrivileged(DefaultInstanceManager.java:501)
at org.apache.catalina.core.DefaultInstanceManager.newInstance(DefaultInstanceManager.java:118)
at org.apache.catalina.core.StandardWrapper.loadServlet(StandardWrapper.java:1061)
at org.apache.catalina.core.StandardWrapper.load(StandardWrapper.java:1000)

at org.apache.catalina.core.StandardContext.loadOnStartup(StandardContext.java:4901)
at org.apache.catalina.core.StandardContext.startInternal(StandardContext.java:5211)
at org.apache.catalina.util.LifecycleBase.start(LifecycleBase.java:152)
at org.apache.catalina.core.ContainerBase$StartChild.call(ContainerBase.java:1403)
at org.apache.catalina.core.ContainerBase$StartChild.call(ContainerBase.java:1393)
at java.util.concurrent.FutureTask.run(FutureTask.java:266)
at java.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.java:1149)
at java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:624)
at java.lang.Thread.run(Thread.java:748)

Reply

Soumitra Roy Sarkar says:
10th May, 2019 at 9:54 am

May be you are missing spring-web dependency.

Reply

UMA says:
29th Jun, 2018 at 2:56 am

what is the need of the row mapper class in the project if we not write the row mapper class can any other alternative way to configure the crud operation using springjdbc. can u plz clarified me.
can u mention some of spinet of code
thanks a lot....

Reply

Soumitra Roy Sarkar says:
30th Jun, 2018 at 3:30 pm

if you do not want to use row mapper, then also it is possible. Please read doc https://docs.spring.io/spring/docs/4.0.x/spring-framework-reference/html/jdbc.html

Reply

Priyanka says:
16th May, 2018 at 1:56 pm

Hi ,
Nice example can you share source code plz.
Thanks in advance
Priyanka

Reply

Soumitra Roy Sarkar says:
17th May, 2018 at 2:04 am

Unfortunately for this tutorial I don't have source code due to system crash.

Reply

trang says:
29th Mar, 2018 at 9:44 am

What is this error?
SEVERE: Servlet.service() for servlet [spring-mvc-jdbc] in context with path [/spring-mvc-jdbc] threw exception [Request processing failed; nested exception is org.springframework.jdbc.UncategorizedSQLException: StatementCallback; uncategorized SQLException for SQL [select * from user_details]; SQL state [S0022]; error code [0]; Column 'first_name' not found.; nested exception is java.sql.SQLException: Column 'first_name' not found.] with root cause
java.sql.SQLException: Column 'first_name' not found.

Reply

Soumitra Roy Sarkar says:
30th Mar, 2018 at 4:53 am

please make sure table has the column name “first_name”

Reply

Alex says:
5th Jan, 2018 at 2:43 pm

Thank you very much for the example! The code works fine for me. :)

Reply

Leave a Reply

Your email address will not be published. Required fields are marked *

Comment

Name *

Email *