4/25/2016

Jose Antonio Hernandez Vazquez

Web App Template

Contents

[STS Project Configuration 2](#_Toc449473200)

[Maven Configuration 3](#_Toc449473201)

[Spring MVC Configuration 4](#_Toc449473202)

[DB Connection 6](#_Toc449473203)

[CRUD Operations with JDBC 8](#_Toc449473204)

[Getting all data 8](#_Toc449473205)

[Getting specific data 8](#_Toc449473206)

[Update data 8](#_Toc449473207)

[Save data 8](#_Toc449473208)

[Delete data 8](#_Toc449473209)

[RowMapper 9](#_Toc449473210)

[CRUD Operations with JPA 10](#_Toc449473211)

# STS Project Configuration

* File → New → Dynamic Web Project.
  1. Choose Project name → Next.
  2. Add folders:
     1. src/main/java
     2. src/main/resources
     3. src/test/java
     4. src/test/resources, → Next
  3. Choose content directory name: webapp.
     1. Check generate web.xml, → Finish.

Note: Create an index.jsp under webapp to test is working.

# Maven Configuration

* Right click → configure → Convert to maven project.
* Add maven-compiler-plugin inside POM.xml file, in build tags:

<plugin>

<groupId>org.apache.maven.plugins</groupId>

<artifactId>maven-compiler-plugin</artifactId>

<version>3.2</version>

<configuration>

<source>${java.version}</source>

<target>${java.version}</target>

<encoding>${project.build.sourceEncoding}</encoding>

</configuration>

</plugin>

* Add maven-war-plugin

<plugin>

<groupId>org.apache.maven.plugins</groupId>

<artifactId>maven-war-plugin</artifactId>

<version>2.6</version>

</plugin>

Note:

* 1. Plugins goes inside build/plugins tags.
  2. Add properties tag to easiest manage

<properties>

<project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>

<java.version>1.7</java.version>

<jackson-json.version>2.7.3</jackson-json.version>

<log4j.version>1.2.17</log4j.version>

<spring.version>3.2.15.RELEASE</spring.version>

</properties>

# Spring MVC Configuration

* Add Spring MVC dependency to POM.xml, version 3.2.15

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-webmvc</artifactId>

<version>${spring.version}</version>

</dependency>

* Add Log4J dependency, version 1.2.17

<dependency>

<groupId>log4j</groupId>

<artifactId>log4j</artifactId>

<version>${log4j.version}</version>

</dependency>

* Add Jackson dependencies, version 2.7.3

<dependency>

<groupId>com.fasterxml.jackson.core</groupId>

<artifactId>jackson-annotations</artifactId>

<version>${jackson-json.version}</version>

</dependency>

<dependency>

<groupId>com.fasterxml.jackson.core</groupId>

<artifactId>jackson-core</artifactId>

<version>${jackson-json.version}</version>

</dependency>

<dependency>

<groupId>com.fasterxml.jackson.core</groupId>

<artifactId>jackson-databind</artifactId>

<version>${jackson-json.version}</version>

</dependency>

* Create webmvc-config.xml in src/main/WEB-INF/spring, it can be created by File → New → Spring Bean Configuration File, and choosing namespace: beans, context, mvc, p
  1. Add <context:component-scan base-package=”…” />
  2. Add <mvc:annotation-driven />
  3. Add bean for InternalResourceViewResolver:

<bean id=*"internalResourceViewResolver"* class=*"org.springframework.web.servlet.view.InternalResourceViewResolver"*

p:prefix=*"/WEB-INF/views/"* p:suffix=*".jsp"*/>

* Add DispatcherServlet definition and mapping to web.xml, and add init parameter called “contextConfigLocation” pointing webmvc-config.xml:

<servlet>

<servlet-name>spring-mvc</servlet-name>

<servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>

<init-param>

<param-name>contextConfigLocation</param-name>

<param-value>WEB-INF/spring/webmvc-config.xml</param-value>

</init-param>

</servlet>

<servlet-mapping>

<servlet-name>spring-mvc</servlet-name>

<url-pattern>/</url-pattern>

</servlet-mapping>

* Create controller class, use annotations:
  1. @Controller
  2. @RequestMapping(value=”…”)
  3. @RequestMapping(value=”…”, method = RequestMethod.GET)
  4. @RequestMapping(value=”…”, method = RequestMethod.GET, produces = MediaType.APLICATION\_JSON\_VALUE)
  5. @ResponseBody
* Create views.

# DB Connection

* Add dependencies for:
  + spring-orm, v 3.2.15
  + hibernate-core, v 4.3.11.Final
  + hibernate-validator, v 5.0.3.Final
  + hibernate-jpa-2.1-api, v 1.0.0.Final
  + hibernate-entitymanager, v 4.3.11.Final
  + mysql-connector-java, v 5.0.3.Final
  + Optionally, add guava, v 19.0
* Create src/main/resources/META-INF/persistence.xml file
  + Must to contain the persistence-unit with name and transaction-type set to RESOURCE\_LOCAL
  + Provider
  + Class: List of classes to be handled by entityManager
  + Properties: for provider

<?xml version="1.0" encoding="UTF-8" standalone="no"?>

<**persistence** xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xmlns="http://java.sun.com/xml/ns/persistence" version="2.0"

xsi:schemaLocation="http://java.sun.com/xml/ns/persistence http://java.sun.com/xml/ns/persistence/persistence\_2\_0.xsd">

<**persistence-unit** name="persistenceUnit" transaction-type="**RESOURCE\_LOCAL**">

<provider>org.hibernate.jpa.HibernatePersistenceProvider</provider>

<class>jahv.selfstudy.model.EmployeeEntity</class>

<exclude-unlisted-classes>true</exclude-unlisted-classes>

<properties>

<property name="hibernate.dialect" value="org.hibernate.dialect.MySQL5Dialect"/>

<property name="hibernate.hbm2ddl.auto" value="validate" />

<property name="hibernate.show\_sql" value="true" />

</properties>

</persistence-unit>

</persistence>

* Create src/main/resources/META-INF/spring/jdbc-applicationContext.xml
  + It must contain the xmlns for beans, XMLSchema-instance, tx

<?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:tx="http://www.springframework.org/schema/tx"

xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd

http://www.springframework.org/schema/tx http://www.springframework.org/schema/tx/spring-tx-3.2.xsd">

* + Add the <tx:annotation-driven />, for Transactional annotation
  + Add bean for entityManagerFactory

<bean id="entityManagerFactory" class="org.springframework.orm.jpa.LocalContainerEntityManagerFactoryBean">

<property name="persistenceUnitName" value="persistenceUnit" />

<property name="dataSource" ref="dataSource" />

<property name="jpaDialect">

<bean class="org.springframework.orm.jpa.vendor.HibernateJpaDialect" />

</property>

</bean>

* + Add bean for transactionManager

<bean id="transactionManager" class="org.springframework.orm.jpa.JpaTransactionManager">

<property name="entityManagerFactory" ref="entityManagerFactory" />

</bean>

* + Add bean for DataSource

<bean id="dataSource" class="org.springframework.jdbc.datasource.DriverManagerDataSource">

<property name="driverClassName" value="com.mysql.jdbc.Driver" />

<property name="url" value="jdbc:mysql://localhost:3306/webapptemplate" />

<property name="username" value="root" />

<property name="password" value="" />

</bean>

* + Add bean for JDBC Template (If needed)

<bean id="jdbcTemplate" class="org.springframework.jdbc.core.JdbcTemplate">

<property name="dataSource" ref="dataSource" />

</bean>

* Add jdbc-applicationContext.xml to spring loading through ContextLoaderListener in web.xml

<listener>

<listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>

</listener>

<context-param>

<param-name>contextConfigLocation</param-name>

<param-value>classpath\*:META-INF/spring/jdbc-applicationContext.xml</param-value>

</context-param>

# CRUD Operations with JDBC

* Create DTO class
* Create repository class and inject the jdbcTemplate bean defined in jdbc-applicationContex.xml

@Repository

public class EmployeeRepositoryJDBC {

@Autowired

private JdbcTemplate jdbcTemplate;

## Getting all data

**public** List<EmployeeEntity> getEmployees() {

**final** String sql = "SELECT \* FROM EMPLOYEES";

List<EmployeeEntity> employees = jdbcTemplate.query(sql, **new** EmployeeRowMapper());

**return** employees;

}

## Getting specific data

**public** EmployeeEntity getEmployee(**final** Long id) {

**final** String sql = "SELECT \* FROM EMPLOYEES WHERE ID = ?";

**final** Object[] args = **new** Object[] { id };

**return** jdbcTemplate.queryForObject(sql, args, **new** EmployeeRowMapper());

}

## Update data

**public** **void** updateEmployee(**final** Long id) {

**final** String sql = "UPDATE EMPLOYEES SET NOTES = ? WHERE ID = ?";

**final** Object[] args = **new** Object[] { **new** Date().toString(), id };

jdbcTemplate.update(sql, args);

}

## Save data

**public** **void** saveEmployee(**final** EmployeeEntity employee) {

**final** String sql = "INSERT INTO EMPLOYEES(first\_name, last\_name, department) VALUES (?,?,?)";

**final** Object[] args = **new** Object[] { employee.getFirstName(), employee.getLastName(),

employee.getDepartment() };

jdbcTemplate.update(sql, args);

}

## Delete data

**public** **void** deleteEmployee(**final** Long id) {

**final** String sql = "DELETE FROM EMPLOYEES WHERE ID = ?";

**final** Object[] args = **new** Object[] { id };

jdbcTemplate.update(sql, args);

}

* Create service and controller

## RowMapper

**public** **class** EmployeeRowMapper **implements** RowMapper<EmployeeEntity> {

@Override

**public** EmployeeEntity mapRow(ResultSet rs, **int** rowNum) **throws** SQLException {

EmployeeEntity e = **new** EmployeeEntity();

e.setId(rs.getLong(1));

e.setFirstName(rs.getString(2));

e.setLastName(rs.getString(3));

e.setDepartment(rs.getString(4));

e.setNotes(rs.getString(5));

**return** e;

}

}

# CRUD Operations with JPA

* Add dependency for spring-aspects, v 3.2.15.RELEASE
* Add dependency for spring-data-jpa, v 1.6.5.RELEASE
* Create/Mapping entity class

@Entity

@Table(name = "employees")

**public** **class** EmployeeEntity **implements** Serializable {

**private** **static** **final** **long** ***serialVersionUID*** = 1L;

@Id

@GeneratedValue(strategy = GenerationType.***IDENTITY***)

@Column(columnDefinition = "INT")

**private** Long id;

@Column(name = "first\_name", nullable = **false**)

**private** String firstName;

* Create repository interface, when extending JpaRepository we have default crud operations

@Repository

**public** **interface** EmployeeRepositoryJPA **extends** JpaRepository<EmployeeEntity, Long> {

}

* Add jpa support in jdbc-applicationContext.xml

<beans xmlns="http://www.springframework.org/schema/beans"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xmlns:tx="http://www.springframework.org/schema/tx"

xmlns:jpa="http://www.springframework.org/schema/data/jpa"

xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd

http://www.springframework.org/schema/tx http://www.springframework.org/schema/tx/spring-tx-3.2.xsd

http://www.springframework.org/schema/data/jpa http://www.springframework.org/schema/data/jpa/spring-jpa.xsd">

* Update tx:annotation in jdbc-applicationContext.xml

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

* Define the repositories package in jdbc-applicationContext.xml

<jpa:repositories base-package="jahv.selfstudy.repository"

entity-manager-factory-ref="entityManagerFactory" transaction-manager-ref="transactionManager" />

* Create service. Annotate service methods as @Transactional. Some Service methods:
  + employeeRepositoryJPA.findAll();
  + employeeRepositoryJPA.findOne(id);
  + employeeRepositoryJPA.save(employee); //For insertions and updations if the entity is managed by entity manager
  + employeeRepositoryJPA.delete(employee);
* Create controller

# Integration Testing for Repositories

* Add dependencies for:
  + spring-test, v 3.2.15.RELEASE
  + junit, v 4.12
* Add corresponding it for class(same package, same name, ending with IT) under src/test/it

@ContextConfiguration(locations = "classpath\*:META-INF/spring/jdbc-applicationContext.xml")

@TransactionConfiguration(defaultRollback = **true**)

**public** **class** EmployeeRepositoryJPAIT **extends** AbstractTransactionalJUnit4SpringContextTests {

@Autowired

**private** EmployeeRepositoryJPA employeeRepositoryJPA;

@Test

**public** **void** testFindAll() {

**final** List<EmployeeEntity> employees = employeeRepositoryJPA.findAll();

Assert.*assertFalse*(employees.isEmpty());

}