Spring – MVC – Hibernate

# Prepare the workspace

For a project which uses spring and hibernate you’ve first to prepare a workspace. I use usually eclipse to create a maven project and provide a pom.xml with the needed dependencies. In this case:

For the spring framework:

|  |
| --- |
| <!-- Spring MVC framework -->  <dependency>  <groupId>org.springframework</groupId>  <artifactId>spring-webmvc</artifactId>  <version>3.0.1.RELEASE</version>  </dependency>  <!-- Spring MVC framework -->  <dependency>  <groupId>org.springframework</groupId>  <artifactId>spring-orm</artifactId>  <version>3.0.1.RELEASE</version>  </dependency>  <!-- Spring MVC framework -->  <dependency>  <groupId>org.springframework</groupId>  <artifactId>spring-test</artifactId>  <version>3.0.1.RELEASE</version>  </dependency> |

For the log4j framework:

|  |
| --- |
| <!-- Log4J -->  <dependency>  <groupId>log4j</groupId>  <artifactId>log4j</artifactId>  <version>1.2.15</version>  <exclusions>  <exclusion>  <groupId>com.sun.jmx</groupId>  <artifactId>jmxri</artifactId>  </exclusion>  <exclusion>  <groupId>com.sun.jdmk</groupId>  <artifactId>jmxtools</artifactId>  </exclusion>  <exclusion>  <groupId>javax.jms</groupId>  <artifactId>jms</artifactId>  </exclusion>  </exclusions>  </dependency> |

|  |
| --- |
| <dependency>  <groupId>org.slf4j</groupId>  <artifactId>slf4j-log4j12</artifactId>  <version>1.5.6</version>  <type>jar</type>  </dependency> |

Test- and mock-framework:

|  |
| --- |
| <dependency>  <groupId>org.mockito</groupId>  <artifactId>mockito-all</artifactId>  <version>1.9.5</version>  </dependency>  <dependency>  <groupId>cglib</groupId>  <artifactId>cglib</artifactId>  <version>2.2</version>  <type>jar</type>  <scope>compile</scope>  </dependency>  <dependency>  <groupId>junit</groupId>  <artifactId>junit</artifactId>  <version>4.11</version>  <scope>test</scope>  </dependency> |

Servlet engine:

|  |
| --- |
| <!-- JSTL -->  <dependency>  <groupId>javax.servlet</groupId>  <artifactId>jstl</artifactId>  <version>1.1.2</version>  </dependency>  <dependency>  <groupId>taglibs</groupId>  <artifactId>standard</artifactId>  <version>1.1.2</version>  </dependency>  <!-- for compile only, your container should have this -->  <dependency>  <groupId>javax.servlet</groupId>  <artifactId>servlet-api</artifactId>  <version>2.5</version>  <scope>provided</scope>  </dependency> |

MySQL and Hibernate:

|  |
| --- |
| <dependency>  <groupId>mysql</groupId>  <artifactId>mysql-connector-java</artifactId>  <version>5.1.16</version>  <scope>provided</scope>  </dependency>  <dependency>  <groupId>org.hibernate</groupId>  <artifactId>hibernate-entitymanager</artifactId>  <version>3.4.0.GA</version>  <scope>compile</scope>  </dependency> |

# Tomcat configuration

Unpack the tomcat. To get access via http://localhost:8080/manager/html to the tomcat manager site you’ve to configure a user with manager-gui role. It is good practice to give the administrator admin rights as well.

File: **<tomcathome>/conf/tomcat-users.xml**

|  |
| --- |
| user username="admin" password="<password>" roles="admin,manager-gui"/> |

## Deployment

The easiest way to deploy is to drop the war-file into the <tomcathome>/webapps directory. During development it is a better approach to create a context pointing to the target directory of your web app in development. The deployment directory is <tomcathome>/conf/Catalina/localhost. The structure of the deployment descriptor is like this:

File: **<tomcathome>/conf/Catalina/localhost/mycontext.xml**

|  |
| --- |
| <Context  docBase="<your path>/target/SpringMVC"  path="/mycontext  reloadable="true"  /> |

## Data Source

To configure a data source in tomcat a context has to be configured in <tomcathome>/conf/context.xml. In the context tag you’ve to define a resource:

File**: <tomcathome>/conf/context.xml**

|  |
| --- |
| <Resource name="jdbc/springDataSource"  auth="Container"  type="javax.sql.DataSource"  driverClassName="com.mysql.jdbc.Driver"  url="jdbc:mysql://localhost:3306/neusta"  username="<uername>"  password="<password>"  removeAbandoned="true"  removeAbandonedTimeout="90"  logAbandoned="true"  maxActive="20"  maxIdle="10"  maxWait="-1"/> |

The data base connection is now under the control of the tomcat database pool. In the above example a database connection for mysql is configured. You’ve to place the jdbc-driver into the <tomcathome>/lib directory.

Now you’ve to tell your application to work with the jdbc datasource. This is made in your applicationcontext.xml of the spring framework

|  |
| --- |
| <bean id="myDataSource"   class="org.springframework.jndi.JndiObjectFactoryBean">  <property name="jndiName" value="java:comp/env/jdbc/springDataSource" />  </bean>  <bean id="entityManagerFactory"  class="org.springframework.orm.jpa.LocalContainerEntityManagerFactoryBean">  <property name="persistenceUnitName" value="SpringMVC" />  <property name="dataSource" ref="myDataSource" />  <property name="jpaVendorAdapter">  <bean  class="org.springframework.orm.jpa.vendor.HibernateJpaVendorAdapter">  <property name="showSql" value="false" />  <property name="generateDdl" value="true" />  <property name="databasePlatform"  value="org.hibernate.dialect.MySQL5InnoDBDialect" />  </bean>  </property>  </bean>  <bean id="mySessionFactory"  class="org.springframework.orm.hibernate3.LocalSessionFactoryBean">  <property name="dataSource" ref="myDataSource" />  <property name="hibernateProperties">  <value> org.hibernate.dialect.MySQL5InnoDBDialect </value>  </property>  </bean>  <bean id="transactionManager"   class="org.springframework.orm.jpa.JpaTransactionManager" />  <tx:annotation-driven /> |

Secondly you’ve to configure your persistence.xml in the META\_INF folder:

File: **META-INF/persistence.xml**

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8"?>  <persistence version="1.0" xmlns=<http://java.sun.com/xml/ns/persistence>  xmlns:xsi=<http://www.w3.org/2001/XMLSchema-instance>  xsi:schemaLocation="http://java.sun.com/xml/ns/persistence   http://java.sun.com/xml/ns/persistence/persistence\_1\_0.xsd">  <persistence-unit name="SpringMVC">  <property name="persistenceUnitManager">  <bean  class="org.springframework.orm.jpa.persistenceunit.  DefaultPersistenceUnitManager">  <property name="defaultDataSource" ref="myDataSource" />  </bean>  </property>  </persistence-unit>  </persistence> |

## Data Source for testing

For test purpose maybe you want to test without tomcat. For this you’ve to configure an own application context definition and persistence.xml only for tests. It is good practice to provide a separate schema for testing.

File: **META-INF/applicationcontext.xml**

|  |
| --- |
| <bean id="dataSource"  class="org.springframework.jdbc.datasource.DriverManagerDataSource">  <property name="username" value="<username>" />  <property name="password" value="<password>" />  <property name="driverClassName" value="com.mysql.jdbc.Driver" />  <property name="url" value="jdbc:mysql://localhost:3306/neustatest" />  </bean>  <bean id="entityManagerFactory"  class="org.springframework.orm.jpa.LocalContainerEntityManagerFactoryBean">  <property name="persistenceUnitName" value="SpringMVC" />  <property name="jpaVendorAdapter">  <bean   class="org.springframework.orm.jpa.vendor.HibernateJpaVendorAdapter">  <property name="showSql" value="true" />  <property name="generateDdl" value="true" />  <property name="databasePlatform"   value="org.hibernate.dialect.MySQL5InnoDBDialect" />  </bean>  </property>  </bean>  <bean id="transactionManager"   class="org.springframework.orm.jpa.JpaTransactionManager" />  <tx:annotation-driven /> |

And of course you need another persistence definition.

File: **META-INF/persistence.xml**

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8"?>  <persistence version="1.0" xmlns="http://java.sun.com/xml/ns/persistence" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://java.sun.com/xml/ns/persistence http://java.sun.com/xml/ns/persistence/persistence\_1\_0.xsd">  <persistence-unit name="SpringMVC">  </persistence-unit>  </persistence> |

# Spring MVC Configuration

# Connect to velocity

Usually in the Spring MVC world you need to connect you rendering engine with the framework. This is done by resolver. A JSP resolver e.g. looks like this:

|  |
| --- |
| <bean id="viewResolver" class="org.springframework.web.servlet.view.InternalResourceViewResolver">  <property name="prefix">  <value>/WEB-INF/pages/</value>  </property>  <property name="suffix">  <value>.jsp</value>  </property>  </bean> --> |

In your dispatcher-servlet configuration for velocity you need to configure an own view-resolver and a configuration for velocity. The name of the xml file is configured in the web.xml the servlet configuration file.

File: **mvc-dispatcher-servlet.xml**

|  |
| --- |
| <?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-2.5.xsd  http://www.springframework.org/schema/context  http://www.springframework.org/schema/context/spring-context-2.5.xsd">  <context:annotation-config />  <context:component-scan base-package="de.neusta.common.\*" />  <bean id="velocityConfig"  class="org.springframework.web.servlet.view.velocity.VelocityConfigurer">  <property name="resourceLoaderPath">  <value>/WEB-INF/pages/</value>  </property>  </bean>  <bean id="viewResolver"  class="org.springframework.web.servlet.view.velocity.VelocityViewResolver">  </bean>  </beans> |