MVC
Modèle-Vue-Contrôleur
Exemple d'application

0.1 Rappels

La plateforme Spring MVC repose sur un DispatcherServlet qui gère toutes les requêtes. En rappel, nous avons vu lors du cours JSP et servlet que les requêtes sont reçues sous la forme de requêtes HTTP envoyées par un client web. Les requêtes sont ensuite encapsulées dans l'objet HttpRequest et passées à la servlet.
Dans le cadre de la plateforme Spring MVC, les requêtes sont plutôt passées au DispatcherServlet. Le DispatcherServlet consulte ensuite un handler mapping [1] et désignera par la suite le contrôleur qui se chargera du traitement de la requête. La requête est traitée par le contrôleur et celui-ci retourne un objet de type ModelAndView : le ModelAndView : contient à la fois le Modèle et la Vue.

Il existe 3 types de handler mapping dans la plateforme Spring MVC :

1. BeanNameUrlHandlerMapping
2. ControllerClassNameHandlerMapping
3. SimpleUrlHandlerMapping example

0.2 Début de l'exemple d'application

Créez un projet web dynamique et configurez le fichier web.xml ainsi qu'il suit :

web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://java.sun.com/xml/ns/javaee"
xmlns:web="http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
id="WebApp_ID" version="2.5">
<display-name>SpringMVC</display-name>

<!-- The parameter tells about the location of configuration XML. Usually
all the data access beans and service layer beans are kept here. You can register
more than one XML here. -->
<context-param>
```

1. Composant logiciel qui affecte les requêtes aux contrôleurs

```
<param-name>contextConfigLocation</param-name>
<param-value>WEB-INF/applicationContext.xml</param-value>
</context-param>

<!-- The listener is reponsible for building the spring container. It looks for
all configuration XML as defined by parameter contextConfigLocation and also looks
for a configuration which is named as Dispatch Servlet name. In this case it will
be named as springapp-servlet.xml -->
<listener>
<listener-class>
org.springframework.web.context.ContextLoaderListener
</listener-class>
</listener>

<!-- Dispatcher Servlet which traps all the request targeted for Spring MVC -->
<servlet>
<servlet-name>springapp</servlet-name>
<servlet-class>
org.springframework.web.servlet.DispatcherServlet
</servlet-class>
<load-on-startup>1</load-on-startup>
</servlet>

<!-- Mapping for the request. It can be anything -->
<servlet-mapping>
<servlet-name>springapp</servlet-name>
<url-pattern>*.htm</url-pattern>
</servlet-mapping>
</web-app>
```

Librairies à inclure dans le projet

- jakarta-commons/commons-logging.jar
- mysql-connector-java-5.1.18
- j2ee/jstl.jar
- log4j/log4j-1.2.14.jar
- jakarta-taglibs/standard.jar
- dist/spring.jar
- dist/modules/spring-webmvc.jar

0.3 Couche d'accès aux données

Student.java

```
/**
 * Java bean which will be used to save and retrieve data.
 *
 */
public class Student {
protected String name;

//Getters and setters
public String getName() {
return name;
}

public void setName(String name) {
this.name = name;
}
}
```

L'interface DAO

```
public interface StudentDao {
public void saveStudent(Student student);
public List<Student> getAllStudents();
}
```

L'implementation du DAO

```
public class StudentJdbcDao implements StudentDao {

protected SimpleJdbcTemplate simpleJdbcTemplate;
public void setSimpleJdbcTemplate(SimpleJdbcTemplate simpleJdbcTemplate) {
this.simpleJdbcTemplate = simpleJdbcTemplate;
}

@Override
public void saveStudent(Student student) {
simpleJdbcTemplate.update("insert into STUDENT (name) values (?)",student.getName());

}

@Override
public List<Student> getAllStudents() {
return simpleJdbcTemplate.query
("Select name as Name from Student",
  new ParameterizedRowMapper<Student>(){
  public Student mapRow(ResultSet rs,int rowNum)
   throws SQLException {
  Student student = new Student();
  student.setName(rs.getString("Name"));
  return student;
  }
  }
);
}
}
```

Les classes DAO doivent être enregistrées en tant que bean dans Spring. Cela se fera dans le fichier de configuration applicationContext.xml. Ce fichier est est referencé dans le fichier de configuration web.xml à travers la balise contextConfigLocationCréez le premier fichier de configuration suivant :

applicationContext.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"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
  http://www.springframework.org/schema/beans/spring-beans-2.5.xsd">

<!-- The Dao class -->
<bean id="studentDao" class="com.oyejava.springmvc.StudentJdbcDao">
<property name="simpleJdbcTemplate" ref="jdbcTemplate" />
</bean>

<!-- Template class to access JDBC code -->
<bean id="jdbcTemplate"
class="org.springframework.jdbc.core.simple.SimpleJdbcTemplate">
<constructor-arg ref="dataSource" />
</bean>

<!-- Configuration for the data source -->
<bean id="dataSource" class="org.springframework.jdbc.datasource.DriverManagerDataSource"
     destroy-method="close">
   <property name="driverClassName" value="org.hsqldb.jdbcDriver" />
   <property name="url" value="jdbc:hsqldb:hsql://localhost" />
   <property name="username" value="sa" />
   <property name="password" value="" />
</bean>
</beans>
```

Créez le second fichier de configuration suivant ; ce second fichier est lui aussi référencé dans le fichier web.xml :

springapp-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"
xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-2.5.xsd">

<bean id="studentListController"
class="com.oyejava.springmvc.StudentListController">
<property name="studentDao" ref="studentDao" />
</bean>

    <!-- command class and command name are used to retrieve and set the
 value as name value pair in HttpRequest and Response. The form view
 tells that when the request comes for this Controller than which
 form to display in which user input can be taken. -->
<bean id="studentCreateController"
class="com.oyejava.springmvc.StudentCreateController">
<property name="studentDao" ref="studentDao" />
<property name="formView" value="createStudent" />
<property name="commandName" value="student" />
<property name="commandClass" value="com.oyejava.springmvc.Student" />
</bean>

<bean id="simpleUrlMapping"
class="org.springframework.web.servlet.handler.SimpleUrlHandlerMapping">
<property name="mappings">
<props>
<prop key="/studentList.htm">studentListController</prop>
<prop key="/createStudent.htm">studentCreateController</prop>
</props>
</property>
</bean>

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

</beans>
```

Dans le fichier web.xml :

1. le contexte créé par l'objet contextConfigLocationdefinit le contexte global de l'application ;
2. le contexte créé par le fichier springapp-servletdéfinit le context enfant et se refère aux beans déclaré dans le contexte général. Par exemple, le bean studentCreateController se refère à studentDao.Le bean studentDaoest déclaré dans le contexte général.

A l'intérieur de la classe du contrôleur :

```
/**
 * The controller class which is used to take the user input and
 * process the data at the backend
 */
public class StudentCreateController extends SimpleFormController {

protected StudentDao studentDao;

public void setStudentDao(StudentDao studentDao) {
this.studentDao = studentDao;
}

//The is object which is used to set the values when the form is
//displayed first time
protected Object formBackingObject(HttpServletRequest request)
throws Exception {
Student student = new Student();
student.setName("Default Name");
return student;
}
```

```
//This method is called when the form is submitted by the user.
//The command class is Student so Spring automatically parses the
//HttpRequest object, retrieves the name value pair out of it and
//sets the properties in the command object.
protected ModelAndView onsubmit(Object command) throws Exception {
   Student student = (Student) command;
   studentDao.saveStudent(student);
   return new ModelAndView("redirect:/studentList.htm");
}
}
```

Créez la classe contenant le contrôleur chargé de l'affichage de la liste des étudiants :

StudentDao

```
public class StudentListController extends AbstractController {

   protected StudentDao studentDao;
public void setStudentDao(StudentDao studentDao) {
this.studentDao = studentDao;
   }

   // This issues a request to database through data access layer and
   //gets the list of students. The list of students is put inside a
   //ModelAndView Object
    protected ModelAndView handleRequestInternal(HttpServletRequest arg0,
HttpServletResponse arg1) throws Exception {
     List<Student> studentList = studentDao.getAllStudents();

    //studentList - This is the logical view id and based on view resolve
    //will get converted into a physical view which can be a jsp file
    //students - This is the name of the parameter against which the list
    //          will be stored.
    //This will be used in jsp file to access the student list object.
    //studentList - studentList object.
  return new ModelAndView("studentList","students",studentList);
    }
}
```

Comment le dispatcheur de servlet DispatcherServlet sait-il exactement quel contrôleur doit être invoqué lors qu'une requête lui est transmise ? C'est le handler mapper qui s'en charge. Par exemple, dans le fichier springapp-servlet.xml le handler mapper portant le nom simpleUrlMapping est utilisé. La propriété du bean concerné indique quel type d'URL doit correspondre à un contrôleur donnée.

Ci-dessous, un exemple de configuration du controleur StudentListController

```
<bean id="/studentList.htm" class="com.oyejava.springmvc.StudentListController">
   <property name="studentDao" ref="studentDao" />
</bean>
```

0.4 Les vues

Les vues sont des fichiers de type .jsp

createStudent.jsp

```
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core"%>
<%@ taglib prefix="form" uri="http://www.springframework.org/tags/form"%>
<%@ taglib prefix="spring" uri="http://www.springframework.org/tags"%>
<body>
<form:form commandName="student">
<label for="name">Name:</label>
<form:input path="name" />
<input type="submit" value="Register" />
</form:form>
</body>
```

studentList.jsp

```
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<body>
<c:forEach items="${students}" var="student">
<tr>
<td>${student.name} <br/></td>
</tr>
</c:forEach>
</body>
```

0.5 L'intercepteur

LoggingInterceptor.java

```
public class LoggingInterceptor extends HandlerInterceptorAdapter {

private static Logger log = Logger.getLogger(LoggingInterceptor.class);

@Override
public boolean preHandle(HttpServletRequest request,
HttpServletResponse response, Object handler) throws Exception {
// TODO Auto-generated method stub
log.info("Entered for processing request");
return true;
}

@Override
public void postHandle(HttpServletRequest request,
HttpServletResponse response, Object handler,
ModelAndView modelAndView) throws Exception {
log.info("Exited for processing request");
}
}
```

Enregistrement de l'intercepteur dans le fichier springapp-servlet.xml

```
<bean id="loggingInterceptor" class="com.oyejava.springmvc.LoggingInterceptor" />

<bean id="simpleUrlMapping"
class="org.springframework.web.servlet.handler.SimpleUrlHandlerMapping">
<property name="interceptors">
<list>
<ref local="loggingInterceptor" />
</list>
</property>
<property name="mappings">
<props>
<prop key="/studentList.htm">
studentListController
</prop>
<prop key="/createStudent.htm">
studentCreateController
</prop>
</props>
</property>
</bean>
```

0.6 Internationalisation and localisation

Configuration de l'objet messageSource dans le fichier springapp-Servlet.xml

```
<bean id="messageSource"
class="org.springframework.context.support.ResourceBundleMessageSource">
   <property name="basename">
     <value>messages</value>
   </property>
</bean>
```

src/messages.properties

```
name=Name
save=Save
```

src/messages_de_DE.properties

```
Name=Name
save=Außer
```

0.7 validation

Il s'agit du mécanisme de validation des données saisies dans les vues :

StudentValidator.java

```
public class StudentValidator implements Validator {

public boolean supports(Class clazz) {
return clazz.equals(Student.class);
}

// Actual validate method
public void validate(Object obj, Errors errors) {
//command object is also available
Student student = (Student)obj;
//emptyField is resolved by looking into the properties file
//of locale. If it is not present than the fourth argument
//is used to display locale message
ValidationUtils.rejectIfEmptyOrWhitespace
     (errors, "name", "emptyField","Field empty");
}
}
```

Enregistrement du validator au niveau du contrôleur :

```
<bean id="studentCreateController" class="com.oyejava.springmvc.StudentCreateController">
   <property name="studentDao" ref="studentDao" />
   <property name="formView" value="createStudent" />
   <property name="commandName" value="student" />
   <property name="commandClass" value="com.oyejava.springmvc.Student" />
  <property name="validator">
<bean class="com.oyejava.springmvc.StudentValidator" />
  </property>
</bean>
```

createStudent.jsp

```
<body>
   <spring:hasBindErrors name="student">
<c:forEach var="error" items="${errors.allErrors}">
  <spring:message code="${error.code}" text="${error.defaultMessage}" />
</c:forEach>
    </spring:hasBindErrors>

<form:form commandName="student">
<label for="name"><spring:message code="name" /></label>
<form:input path="name" />
<input type="submit" value=<spring:message code="save"/> />
</form:form>
</body>
```