Spring Multiple Form Controller

Introduction

As you may already guessed, the goal of this feature is to provide a Spring controller capable of handling many WEB forms with different actions within the same WEB page or view.

The main problem with such feature is how to handle different form beans with different scopes (request, session, ...) and, thus, handling many forms validation within the same controller.

Objectives

1- To render multiple forms within the same WEB page when it's firstly requested.

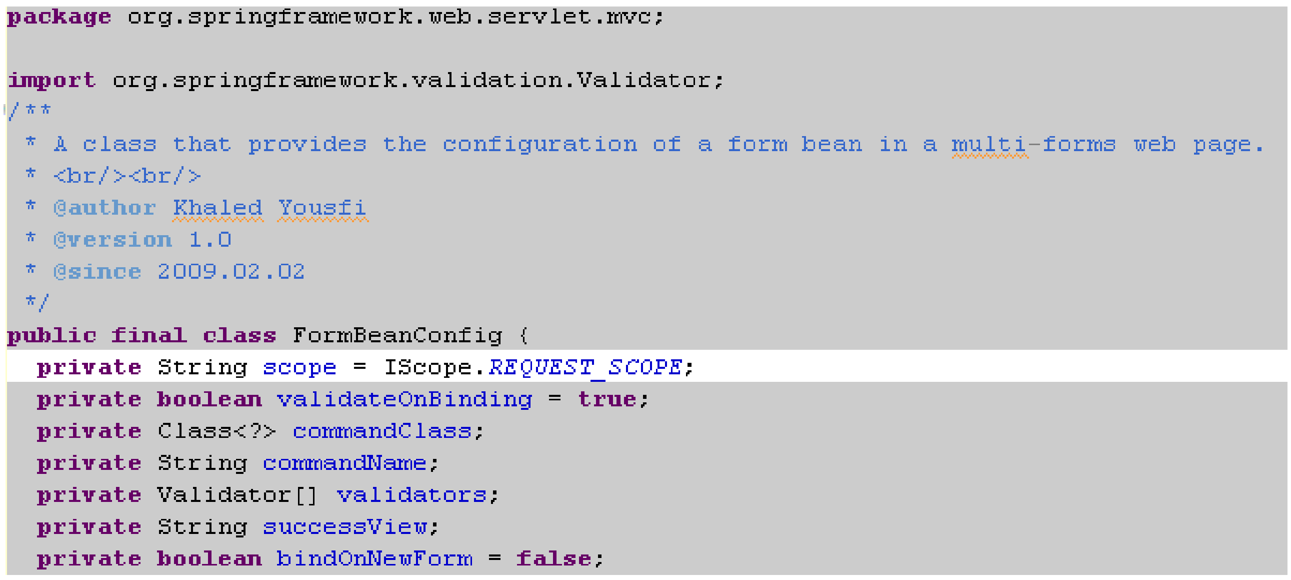
2- After a specific form submission, to bind the data to the appropriate form bean (a POJO) and then to validate it.

3- To re-render the multi-forms view in case of validation error.
=> Take into account the previous validation errors of all forms.

4- If the validation is successful, redirect to the appropriate success view.

Implementation

The main idea of this implementation is to have a controller that maintains a map of form bean configurations. The following class represents such bean config. :

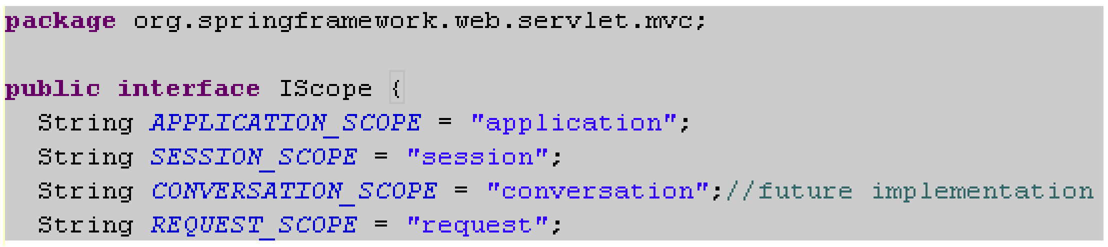


```
package org.springframework.web.servlet.mvc;

import org.springframework.validation.Validator;
/**
 * A class that provides the configuration of a form bean in a multi-forms web page.
 * <br/><br/>
 * @author Khaled Yousfi
 * @version 1.0
 * @since 2009.02.02
 */
public final class FormBeanConfig {
  private String scope = IScope.REQUEST_SCOPE;
  private boolean validateOnBinding = true;
  private Class<?> commandClass;
  private String commandName;
  private Validator[] validators;
  private String successView;
  private boolean bindOnNewForm = false;
```

The FormBeanConfig class attributes accessors (getters & setters) are omitted form more simplicity.

IScope is a simple interface that contains different strings constants denoting the different form bean scopes.



```java
package org.springframework.web.servlet.mvc;

public interface IScope {
  String APPLICATION_SCOPE = "application";
  String SESSION_SCOPE = "session";
  String CONVERSATION_SCOPE = "conversation";//future implementation
  String REQUEST_SCOPE = "request";
```

The controller needs all the form bean configurations in order to handle appropriately a submitted form or simply its multi-forms view.

This multi-form controller implementation maintains a map of key-value. The keys are the action names of the forms whereas the values are instances of the form bean configurations (singletons).

Please take a look at the smfc-servlet.xml from the demo application :

```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

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

    <bean id="messageSource"
class="org.springframework.context.support.ResourceBundleMessageSource">
        <property name="basenames">
            <list>
                <value>messages</value>
                <value>errors</value>
            </list>
         </property>
      </bean>

      <!-- CONTROLLERS -->
      <bean id="welcomeController"
class="com.yk.spring.smfc.web.WelcomeController" />

      <bean id="urlFilenameViewController"
class="org.springframework.web.servlet.mvc.UrlFilenameViewController"/>

      <bean id="myMultiformController"
class="com.yk.spring.smfc.web.MyMultiformController">
         <property name="formView" value="forms"/>
         <property name="actions">
             <map>
                 <entry key="/loginAction.htm" value-ref="loginFormConfig"/>
                 <entry key="/searchAction.htm" value-ref="searchFormConfig"/>
                 <entry key="/postMessageAction.htm" value-
ref="postMessageFormConfig"/>
               <entry key="/addAuthorAction.htm" value-
ref="addAuthorFormConfig"/>
             </map>
         </property>
      </bean>

    <!-- FORMS CONFIGURATIONS -->
    <bean id="loginFormConfig"
class="org.springframework.web.servlet.mvc.FormBeanConfig">
       <property name="scope" value="session"/>
       <property name="successView" value="redirect:loginActionSuccessView.htm"/
>
       <property name="commandClass" value="com.yk.spring.smfc.domain.Login"/>
       <property name="commandName" value="login"/>
       <property name="validator" ref="loginFormValidator"/>
    </bean>

    <bean id="searchFormConfig"
class="org.springframework.web.servlet.mvc.FormBeanConfig">
       <property name="scope" value="request"/>
       <property name="successView"
value="redirect:searchActionSuccessView.htm"/>
       <property name="commandClass" value="com.yk.spring.smfc.domain.Search"/>
       <property name="commandName" value="search"/>
       <property name="validator" ref="searchFormValidator"/>
    </bean>

    <bean id="postMessageFormConfig"
class="org.springframework.web.servlet.mvc.FormBeanConfig">
       <property name="scope" value="session"/>
       <property name="successView"
value="redirect:postMessageActionSuccessView.htm"/>
       <property name="commandClass" value="com.yk.spring.smfc.domain.Message"/>
       <property name="commandName" value="message"/>
       <property name="validator" ref="postMessageFormValidator"/>
    </bean>

    <bean id="addAuthorFormConfig"
class="org.springframework.web.servlet.mvc.FormBeanConfig">
       <property name="scope" value="session"/>
       <property name="successView"
value="redirect:addAuthorActionSuccessView.htm"/>
       <property name="commandClass" value="com.yk.spring.smfc.domain.Author"/>
       <property name="commandName" value="author"/>
       <property name="validator" ref="addAuthorFormValidator"/>
    </bean>

      <!-- URL MAPPINGS -->
      <bean
class="org.springframework.web.servlet.handler.SimpleUrlHandlerMapping">
            <property name="mappings">
                  <props>
                        <prop key="/welcome.htm">welcomeController</prop>
                        <prop key="/forms.htm">myMultiformController</prop>
                        <prop
key="/loginAction.htm">myMultiformController</prop>
                        <prop
key="/searchAction.htm">myMultiformController</prop>
                        <prop key="/addAuthorAction.htm">myMultiformController</
prop>
                        <prop
key="/postMessageAction.htm">myMultiformController</prop>

                        <prop
```

```
key="/loginActionSuccessView.htm">urlFilenameViewController</prop>
                        <prop
key="/searchActionSuccessView.htm">urlFilenameViewController</prop>
                        <prop
key="/postMessageActionSuccessView.htm">urlFilenameViewController</prop>
                        <prop
key="/addAuthorActionSuccessView.htm">urlFilenameViewController</prop>
                  </props>
            </property>
      </bean>
</beans>
```

How the controller works ?

When a multi-forms view is requested, the controller creates binders (validation) and saves the errors holder to the appropriate scope.

Here is the **saveErrors** method :

```
protected final void saveErrors(FormBeanConfig formBeanConfig,
HttpServletRequest request, BindException errors) {
    if (formBeanConfig != null) {
      if(logger.isDebugEnabled()){
        logger.debug("Bean Config is not null. Saving errors object...");
      }
      if (formBeanConfig.isRequestForm()) {
        request.setAttribute(BIND_ERRORS_SCOPE +
formBeanConfig.getCommandName(), errors);
        return;
      }
      if (formBeanConfig.isSessionForm()) {
        request.getSession(false).setAttribute(BIND_ERRORS_SCOPE +
formBeanConfig.getCommandName(), errors);
        return;
      }
      throw new IllegalStateException("Unknown form bean config scope :
"+formBeanConfig.getScope());//NotImplementedException ??
    }else{
      if(logger.isDebugEnabled()){
        logger.debug("Bean Config is null. No saving of the errors object.");
      }
    }
  }
```

When submitting a form, the controller should validate the submitted form data and save the eventual errors to the appropriate scope. If there is any error, this controller restores all the previous validation errors and re-render the input view page.

Here is the **restoreErrors** method :

```
  /**
   * Returns the validation errors of the form bean given its configuration bean
instance.
   * @param formBeanConfig
   * @param request
   * @return
   */
  private BindException restoreErrors(FormBeanConfig formBeanConfig,
HttpServletRequest request) {
    if(formBeanConfig.isRequestForm()){
      return restoreErrorsFromRequest(formBeanConfig.getCommandName(), request);
    }
    if(formBeanConfig.isSessionForm()){
      return restoreErrorsFromSession(formBeanConfig.getCommandName(), request);
    }
    throw new IllegalStateException("Unknown bean config scope :
"+formBeanConfig.getScope());//NotImplementedException ??
  }

  private BindException restoreErrorsFromSession(String commandName,
HttpServletRequest request) {
    return
(BindException)request.getSession(false).getAttribute(BIND_ERRORS_SCOPE+commandN
ame);
  }

  private BindException restoreErrorsFromRequest(String commandName,
HttpServletRequest request) {
    return (BindException)request.getAttribute(BIND_ERRORS_SCOPE+commandName);
  }
```

Conclusion

Even though it's not yet validated and integrated, this feature demonstrates how Spring framework is easy to handle and to use.

As a future feature **:o)**, we can imagine a spring tool that automatically generates a fully functional CRUD web application given a database table.

The multi-forms controller could help us to create a 'multi criteria search form' on the top of a view with a selectable items list on the view main part.

khaled.yousfi@gmail.com