**JSF 2 + Spring 3 Integration Example**

http://0.gravatar.com/avatar/622c70d2908e68ecc070ca6754245bb2?s=40&d=http%3A%2F%2Fwww.mkyong.com%2Fwp-content%2Fthemes%2Fmkyongnew%2Fimages%2Favatar-guest.jpg%3Fs%3D40&r=G

Posted on December 5, 2010 ,     Last modified : September 17, 2012

By [mkyong](http://www.mkyong.com/author/mkyong/)

In this tutorial, we will show you how to integrate JSF 2.0 with Spring 3 using :

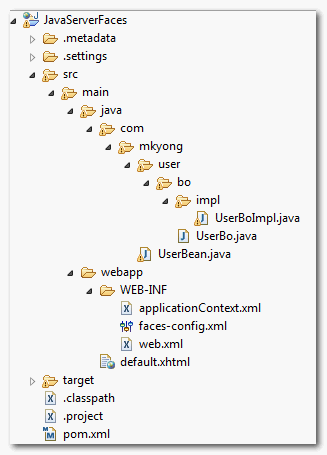
1. JSF XML faces-config.xml
2. Spring annotations
3. JSR-330 standard injection

Tools and technologies used :

1. JSF 2.1.13
2. Spring 3.1.2.RELEASE
3. Maven 3
4. Eclipse 4.2
5. Tomcat 6 or 7

**1. Directory Structure**

A standard Maven project for demonstration.



**2. Project Dependencies**

Declares JSF 2, Spring 3, JSR-330 inject, and Tomcat’s dependencies.

pom.xml

**<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.org/maven-v4\_0\_0.xsd"**>**

**<modelVersion>**4.0.0**</modelVersion>**

**<groupId>**com.mkyong.common**</groupId>**

**<artifactId>**JavaServerFaces**</artifactId>**

**<packaging>**war**</packaging>**

**<version>**1.0-SNAPSHOT**</version>**

**<name>**JavaServerFaces Maven Webapp**</name>**

**<url>**http://maven.apache.org**</url>**

**<dependencies>**

*<!-- Spring framework -->*

**<dependency>**

**<groupId>**org.springframework**</groupId>**

**<artifactId>**spring-core**</artifactId>**

**<version>**3.1.2.RELEASE**</version>**

**</dependency>**

**<dependency>**

**<groupId>**org.springframework**</groupId>**

**<artifactId>**spring-web**</artifactId>**

**<version>**3.1.2.RELEASE**</version>**

**</dependency>**

*<!-- JSR-330 -->*

**<dependency>**

**<groupId>**javax.inject**</groupId>**

**<artifactId>**javax.inject**</artifactId>**

**<version>**1**</version>**

**</dependency>**

*<!-- JSF -->*

**<dependency>**

**<groupId>**com.sun.faces**</groupId>**

**<artifactId>**jsf-api**</artifactId>**

**<version>**2.1.13**</version>**

**</dependency>**

**<dependency>**

**<groupId>**com.sun.faces**</groupId>**

**<artifactId>**jsf-impl**</artifactId>**

**<version>**2.1.13**</version>**

**</dependency>**

**<dependency>**

**<groupId>**javax.servlet**</groupId>**

**<artifactId>**jstl**</artifactId>**

**<version>**1.2**</version>**

**</dependency>**

**<dependency>**

**<groupId>**javax.servlet**</groupId>**

**<artifactId>**servlet-api**</artifactId>**

**<version>**2.5**</version>**

**</dependency>**

**<dependency>**

**<groupId>**javax.servlet.jsp**</groupId>**

**<artifactId>**jsp-api**</artifactId>**

**<version>**2.1**</version>**

**</dependency>**

*<!-- EL -->*

**<dependency>**

**<groupId>**org.glassfish.web**</groupId>**

**<artifactId>**el-impl**</artifactId>**

**<version>**2.2**</version>**

**</dependency>**

*<!-- Tomcat 6 need this -->*

**<dependency>**

**<groupId>**com.sun.el**</groupId>**

**<artifactId>**el-ri**</artifactId>**

**<version>**1.0**</version>**

**</dependency>**

**</dependencies>**

**<build>**

**<finalName>**JavaServerFaces**</finalName>**

**<plugins>**

**<plugin>**

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

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

**<version>**2.3.1**</version>**

**<configuration>**

**<source>**1.6**</source>**

**<target>**1.6**</target>**

**</configuration>**

**</plugin>**

**</plugins>**

**</build>**

**</project>**

**3. JSF 2 + Spring Integration**

Spring’s bean in Spring Ioc context, and JSF’s managed bean in JSF Ioc context, how to make both working together? The solution is defined Spring’s SpringBeanFacesELResolver in faces-config.xml. Check this [official Spring guide](http://static.springsource.org/spring/docs/3.1.x/spring-framework-reference/html/web-integration.html#jsf-springbeanfaceselresolver).

faces-config.xml

**<?xml** version="1.0" encoding="UTF-8"**?>**

**<faces-config** xmlns="http://java.sun.com/xml/ns/javaee"

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

xsi:schemaLocation="http://java.sun.com/xml/ns/javaee

http://java.sun.com/xml/ns/javaee/web-facesconfig\_2\_1.xsd"

version="2.1"**>**

**<application>**

**<el-resolver>**

org.springframework.web.jsf.el.SpringBeanFacesELResolver

**</el-resolver>**

**</application>**

**</faces-config>**

See following 3 examples to inject Spring’s bean in JSF managed bean.

**3.1. XML Schema Example**

Many developers still prefer to use XML to manage beans. With SpringBeanFacesELResolver, just uses EL ${userBo} to inject Spring’s bean into JSF’s managed bean.

UserBo.java

**package** com.mkyong.user.bo;

**public** **interface** UserBo{

**public** String getMessage();

}

UserBoImpl.java

**package** com.mkyong.user.bo.impl;

**import** com.mkyong.user.bo.UserBo;

**public** **class** UserBoImpl **implements** UserBo{

**public** String getMessage() {

**return** "JSF 2 + Spring Integration";

}

}

UserBean.java – JSF backing bean

**package** com.mkyong;

**import** java.io.Serializable;

**import** com.mkyong.user.bo.UserBo;

**public** **class** UserBean{

*//later inject in faces-config.xml*

UserBo userBo;

**public** **void** setUserBo(UserBo userBo) {

**this**.userBo = userBo;

}

**public** String printMsgFromSpring() {

**return** userBo.getMessage();

}

}

applicationContext.xml – Declares userBo bean

**<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="userBo" class="com.mkyong.user.bo.impl.UserBoImpl"**></bean>**

**</beans>**

faces-config.xml – Declares managed bean and inject userBo

**<?xml** version="1.0" encoding="UTF-8"**?>**

**<faces-config**

xmlns="http://java.sun.com/xml/ns/javaee"

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

xsi:schemaLocation="http://java.sun.com/xml/ns/javaee

http://java.sun.com/xml/ns/javaee/web-facesconfig\_2\_0.xsd"

version="2.0"**>**

**<managed-bean>**

**<managed-bean-name>**user**</managed-bean-name>**

**<managed-bean-class>**com.mkyong.UserBean**</managed-bean-class>**

**<managed-bean-scope>**session**</managed-bean-scope>**

**<managed-property>**

**<property-name>**userBo**</property-name>**

**<value>**#{userBo}**</value>**

**</managed-property>**

**</managed-bean>**

**</faces-config>**

**3.2. Spring Annotations – Auto Scan**

This example is using Spring annotations. Injects like a normal bean with @ManagedBean, @Autowired and @Component, it just works as expected.

UserBoImpl.java

**package** com.mkyong.user.bo.impl;

**import** org.springframework.stereotype.Service;

**import** com.mkyong.user.bo.UserBo;

@Service

**public** **class** UserBoImpl **implements** UserBo{

**public** String getMessage() {

**return** "JSF 2 + Spring Integration";

}

}

UserBean.java

**package** com.mkyong;

**import** javax.faces.bean.ManagedBean;

**import** javax.faces.bean.SessionScoped;

**import** org.springframework.beans.factory.annotation.Autowired;

**import** org.springframework.stereotype.Component;

**import** com.mkyong.user.bo.UserBo;

@Component

@ManagedBean

@SessionScoped

**public** **class** UserBean{

@Autowired

UserBo userBo;

**public** **void** setUserBo(UserBo userBo) {

**this**.userBo = userBo;

}

**public** String printMsgFromSpring() {

**return** userBo.getMessage();

}

}

applicationContext.xml – Enable the component auto scan

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

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

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

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

http://www.springframework.org/schema/beans/spring-beans-3.1.xsd

http://www.springframework.org/schema/context

http://www.springframework.org/schema/context/spring-context-3.1.xsd"**>**

**<context:component-scan** base-package="com.mkyong" **/>**

**</beans>**

**Mixed use of both JSF and Spring annotations** are working fine, but it look weird and duplicated – @Component and@ManagedBean together. Actually, you can just uses a single @Component, see following new version, it’s pure Spring, and it works!

UserBean.java

**package** com.mkyong;

**import** org.springframework.beans.factory.annotation.Autowired;

**import** org.springframework.context.annotation.Scope;

**import** org.springframework.stereotype.Component;

**import** com.mkyong.user.bo.UserBo;

@Component

@Scope("session")

**public** **class** UserBean{

@Autowired

UserBo userBo;

**public** **void** setUserBo(UserBo userBo) {

**this**.userBo = userBo;

}

**public** String printMsgFromSpring() {

**return** userBo.getMessage();

}

}

**3.3. JSR-330 Annotation**

Since Spring 3.0, [Spring offer supports for JSR-330 injection standard](http://www.mkyong.com/spring3/spring-3-and-jsr-330-inject-and-named-example/). Now, you can uses @Inject to replace for@Autowired and @Named for @Component. This is recommended to solution, follow JSR-330 standard make the application more portable to other environments, and it works fine in Spring framework.

UserBoImpl.java

**package** com.mkyong.user.bo.impl;

**import** javax.inject.Named;

**import** com.mkyong.user.bo.UserBo;

@Named

**public** **class** UserBoImpl **implements** UserBo{

**public** String getMessage() {

**return** "JSF 2 + Spring Integration";

}

}

UserBean.java

**package** com.mkyong;

**import** javax.inject.Inject;

**import** javax.inject.Named;

**import** org.springframework.context.annotation.Scope;

**import** com.mkyong.user.bo.UserBo;

@Named

@Scope("session") *//need this, JSR-330 in Spring context is singleton by default*

**public** **class** UserBean {

@Inject

UserBo userBo;

**public** **void** setUserBo(UserBo userBo) {

**this**.userBo = userBo;

}

**public** String printMsgFromSpring() {

**return** userBo.getMessage();

}

}

applicationContext.xml – Need component auto scan also

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

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

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

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

http://www.springframework.org/schema/beans/spring-beans-3.1.xsd

http://www.springframework.org/schema/context

http://www.springframework.org/schema/context/spring-context-3.1.xsd"**>**

**<context:component-scan** base-package="com.mkyong" **/>**

**</beans>**

**4. Demo**

Example in **3.1**, **3.2** and **3.3** are doing exactly the thing – Inject userBo into JSF bean, just different implementation. Now, create a simple JSF page to show the the result.

default.xhtml

<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"

"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<**html** xmlns="http://www.w3.org/1999/xhtml"

xmlns:h="http://java.sun.com/jsf/html"

>

<h:body>

<**h1**>JSF 2.0 + Spring Example</**h1**>

#{userBean.printMsgFromSpring()}

</h:body>

</**html**>

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>**JavaServerFaces**</display-name>**

*<!-- Add Support for Spring -->*

**<listener>**

**<listener-class>**

org.springframework.web.context.ContextLoaderListener

**</listener-class>**

**</listener>**

**<listener>**

**<listener-class>**

org.springframework.web.context.request.RequestContextListener

**</listener-class>**

**</listener>**

*<!-- Change to "Production" when you are ready to deploy -->*

**<context-param>**

**<param-name>**javax.faces.PROJECT\_STAGE**</param-name>**

**<param-value>**Development**</param-value>**

**</context-param>**

*<!-- Welcome page -->*

**<welcome-file-list>**

**<welcome-file>**default.jsf**</welcome-file>**

**</welcome-file-list>**

*<!-- JSF Mapping -->*

**<servlet>**

**<servlet-name>**facesServlet**</servlet-name>**

**<servlet-class>**javax.faces.webapp.FacesServlet**</servlet-class>**

**<load-on-startup>**1**</load-on-startup>**

**</servlet>**

**<servlet-mapping>**

**<servlet-name>**facesServlet**</servlet-name>**

**<url-pattern>**\*.jsf**</url-pattern>**

**</servlet-mapping>**

**<servlet-mapping>**

**<servlet-name>**facesServlet**</servlet-name>**

**<url-pattern>**\*.xhtml**</url-pattern>**

**</servlet-mapping>**

**</web-app>**

Done, see output : *http://localhost:8080/JavaServerFaces/default.jsf*