Jair Rillo Junior's blog

How to create custom JSF Facelets component

July 9th, 2009 | Tags: Component, Facelets, Java, JSF

One of the most interesting thing in JSF is the capability to create custom components and use them anywhere. With facelets the creating is even easier. In this topic, I'll show you how to create a component and use it in your JSF project.

Creating the project

First of all, let's create a simple project that will contain our component. We're going to use Maven to do that (if you aren't familiar with Maven, check it out its official site, as well as this blog for more information). Let's use the quickstart archetype to create the project. Simply type the following:

```
mvn archetype:create \
  -DarchetypeGroupId=org.apache.maven.archetypes \
  -DgroupId=com.jairrillo.component \
  -DartifactId=MyJSFComponent
```

A directory **MyJSFComponent** is gonna be created.

Adding required JAR files

After the project has been created, we need to add the required JAR files. Fortunately it is an easy task with Maven. We simply need to change the pom.xml file. Below following a complete file.
pom.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-ins
  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.jairrillo</groupId>
    <artifactId>MyJSFComponent</artifactId>
    <packaging>jar</packaging>
```

```
<version>0.1</version>
<name>MyJSFComponent</name>
<url>http://maven.apache.org</url>
<!-- Using Java version 1.6 -->
<build>
    <plugins>
        <plugin>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-compiler-plugin</artifactId>
            <version>2.0.2</version>
            <configuration>
                <source>1.6</source>
                <target>1.6</target>
            </configuration>
        </plugin>
    </plugins>
</build>
<!-- SUN and JBoss' repositories -->
<pluginRepositories>
    <pluginRepository>
        <id>maven.java.net</id>
        <name>Java.net Maven2 Repository</name>
        <url>http://download.java.net/maven/2</url>
    </pluginRepository>
    <pluginRepository>
        <id>repository.jboss.com</id>
        <url>http://repository.jboss.com/maven2/</url>
    </pluginRepository>
</pluginRepositories>

<dependencies>
    <!-- JSF -->
    <dependency>
        <groupId>javax.faces</groupId>
        <artifactId>jsf-api</artifactId>
        <version>1.2_12</version>
    </dependency>
    <dependency>
        <groupId>javax.faces</groupId>
        <artifactId>jsf-impl</artifactId>
        <version>1.2_12</version>
    </dependency>
    <!-- JUnit -->
    <dependency>
        <groupId>junit</groupId>
        <artifactId>junit</artifactId>
        <version>4.6</version>
        <scope>test</scope>
    </dependency>
    <!-- Servlet and JSP APIs -->
    <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>
```

```
    </dependencies>
</project>
```

Basically, in the POM.xml we setup the SUN's and JBoss' repositories, the JARs from JSF, Servlet and JSP APIs. Those files are required for our project.

Creating the Component itself

There will be only one Java class in our example. It's gonna be a new error message component that will display the messages within a border. (It's a dummy example. This behavior can be achieved without creating a new component, but if you understand the process, you can create other useful components).

Below following the class. Create it into the component package. Like below:

```
package com.jairrillo.component;

import java.io.IOException;
import java.util.Iterator;

import javax.faces.application.FacesMessage;
import javax.faces.component.UIComponentBase;
import javax.faces.context.FacesContext;
import javax.faces.context.ResponseWriter;

/**
 * @author Jair Rillo Junior
 */
public class ErrorMessageComponent extends UIComponentBase {

    public static final String COMPONENT_TYPE = "com.jairrillo.component.errorMessage";
    public static final String RENDER_TYPE = "com.jairrillo.component.ErrorMessageComponent";

    /**
     * constructor for the component.
     */
    public ErrorMessageComponent() {
        super();
        setRendererType(null);
    }

    @Override
    public String getFamily() {
        return COMPONENT_TYPE;
    }

        @SuppressWarnings("unchecked")
        @Override
    public void encodeBegin(FacesContext context) throws IOException {
        Iterator iter = context.getMessages();
        //check if there are messages
        if (iter.hasNext()) {
                StringBuffer messages = new StringBuffer();
            ResponseWriter writer = context.getResponseWriter();
            //going through all messages
            while (iter.hasNext()) {
                FacesMessage msg = (FacesMessage) iter.next();
                messages.append(msg.getSummary() + "<br />");
            }
```

```
            //print out the result
            writer.write("<div style='border: 1px solid black; width: 100%;'>");
            writer.write(messages.toString());
            writer.write("</div>");
        }
    }
    @Override
    public void encodeEnd(FacesContext context) throws IOException {
        super.encodeEnd(context);
    }
}
```

As you can see above, the **encodeBegin** method has the major implementation.

Registering the component

Besides the Java class (component) we need to register it in the faces-config.xml and taglib files. To do that, first you must create a resource directory. Its path is **/src/main/resources**. Also, create a META-INF folder within it. Put the files below within META-INF folder.

faces-config.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<faces-config version="1.2" xmlns="http://java.sun.com/xml/ns/javaee" xmlns:xsi="http://www.w3.org
    <component>
        <component-type>com.jairrillo.component.errorMessage</component-type>
        <component-class>com.jairrillo.component.ErrorMessageComponent</component-class>
    </component>
</faces-config>
```

error_message-component.taglib.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE facelet-taglib PUBLIC
"-//Sun Microsystems, Inc.//DTD Facelet Taglib 1.0//EN"
"http://java.sun.com/dtd/facelet-taglib_1_0.dtd">

<facelet-taglib>
    <namespace>http://my-own-component</namespace>
    <tag>
        <tag-name>errorMessage</tag-name>
        <component>
            <component-type>com.jairrillo.component.errorMessage</component-type>
            <renderer-type>com.jairrillo.component.ErrorMessageComponent</renderer-type>
        </component>
    </tag>
</facelet-taglib>
```

The configuration is over. Now simply package the component in a JAR file. In the console, type: **mvn package** and then the MyJSFComponent.jar is gonna be created into target directory.

Using it in a JSF/Facelets project

Now you can use the component in any JSF/Facelets project. Simply import the jar file (copying into WEB-INF/lib directory) and use the namespace (http://my-own-component) and taglib () into the xhtml file. It's done!!!

Conclusion

Creating new component in JSF/Facelets is really simple. You can create many custom components and use them in varies projects. It is a great asset.

You can download the source code here. If you have any question or comment, fell free to let the message below.
I hope this topic be useful for anyone.

Leave a comment | Trackback
« Sharing data between PortletSession and HttpSession Bem Vindo! Welcome! »

1. Jesus
   February 21st, 2012 at 06:33
   #1

   U also have to register the new taglib in web.xml like this:
   …

   javax.faces.FACELETS_LIBRARIES
   /WEB-INF/HtmlCustomComponent.taglib.xml

   …

Name (required)
E-Mail (will not be published) (required)
Website

Subscribe to comments feed
Submit Comment

- **Jair Rillo Junior**

Brazilian guy, IT Specialist, Linux and Mac User. Work with Java/JEE and IBM Products, such as: WebSphere and DB2. Like studying Ruby, Android and IOS. Also, I like playing tennis, however I am not good enough. Write a post in this blog once a year. Follow me on twitter if you understand portuguese: @jairrillo.

- **Categories**

- Application Server (17)
  - Glassfish (8)
  - JBoss (3)
  - Tomcat (2)
  - Websphere (6)
- Games (1)
  - Cocos2d-X (1)
- IBM (1)
- IDE (16)
  - Eclipse (6)
  - Emacs (1)
  - Netbeans (4)
  - RAD (4)
  - Vim (2)
- Java (31)
  - EJB3 (9)
  - Hibernate (3)
  - JPA (4)
  - JSF (6)
  - Maven (4)
  - Spring (1)
  - Struts 2 (1)
  - WebService (1)
- Linux (9)
- Mobile (3)
  - Android (1)
  - IOS (2)
- Pessoal (4)
- Ruby (13)
  - JRuby (2)
  - Rails (6)
- Testing (6)
  - JUnit (3)
  - RSpec (2)
- Web (1)
  - JQuery (1)

- **Blogs**

  - Blog Caelum
  - Carlos Vilela’s blog
  - Débito Técnico
  - James Gosling’s Weblog
  - Leonardo’s blog
  - Martin Fowler’s Bliki
  - Philip’s Blog
  - Rodrigo Urubatan’s Blog
  - Sérgio Taborda’s Weblog
  - Visão Ágil

- **Ruby**

- Akita’s blog

- **Websites**

  - GUJ
  - IBM developerWorks
  - InfoQ
  - OnJava
  - The Server Side

TOP

- Log in