FREE eBook

LEARNING jsf

#jsf

Table of Contents

About

Chapter 1: Getting started with jsf

Remarks

JavaServer Faces (JSF) is a model-view-presenter framework typically used to create HTML form based web applications. Using the standard components and render kit, stateful HTML views can be defined using Facelets or JSP tags and wired to model data and application logic via backing beans.

Versions

Version	Release Notes	Release Date
2.3	announcement by Arjan Tijms	2017-03-28
2.2		2013-05-21
2.1		2010-11-22
2.0		2009-07-01
1.2		2006-05-11
1.1		2004-05-27
1.0		2004-03-11

Examples

Installing JSF

Content has been moved back to the good 'ol JSF wiki page

Hello World

Content has been moved back to the good 'ol JSF wiki page

Minimum requirements

Content has been moved back to the good 'ol JSF wiki page

Read Getting started with jsf online: https://riptutorial.com/jsf/topic/916/getting-started-with-jsf

Chapter 2: Ajax Integration

Examples

Partially update the view

Makes an ajax request and updates only part of the view.

Bean.java

```
@ManagedBean
@ViewScoped
public class Bean {

    public Date getCurrentDate() {
        return new Date();
    }

}
```

sample.xhtml

```
<html xmlns="http://www.w3.org/1999/xhtml"
    xmlns:h="http://xmlns.jcp.org/jsf/html"
    xmlns:f="http://xmlns.jcp.org/jsf/core"
    xmlns:ui="http://xmlns.jcp.org/jsf/facelets">
<h:head />
<h:body>
    <h:form>
        <h:commandButton value="Execute ajax">
            <f:ajax render="output" />
        </h:commandButton>
        <p>
            <h:outputText id="output" value="Ajax date: #{bean.currentDate}" />
        </p>
        <p>
            <h:outputText id="output2" value="Non-Ajax date: #{bean.currentDate}" />
        </p>
    </h:form>
</h:body>
</html>
```

Send form parts and listen to the request

Makes a request only sending part of the form. The `text1` value is set, but not `text2`, as the listener states.

Bean.java

```
@ManagedBean
@ViewScoped
public class Bean {
    private String text1;

    private String text2;

    public String getText1() {
        return text1;
    }

    public void setText1(String text1) {
        this.text1 = text1;
    }

    public String getText2() {
        return text2;
    }

    public void setText2(String text2) {
        this.text2 = text2;
    }

    public void listener() {
        System.out.println("values: " + text1 + " " + text2);
    }

}
```

sample.xhtml

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml"
    xmlns:h="http://xmlns.jcp.org/jsf/html"
    xmlns:f="http://xmlns.jcp.org/jsf/core"
    xmlns:ui="http://xmlns.jcp.org/jsf/facelets">
<h:head />
<h:body>
    <h:form>
        <h:inputText id="my_input" value="#{bean.text1}" />
        <h:inputText value="#{bean.text2}" />
        <h:commandButton value="Execute ajax">
            <f:ajax execute="@this my_input" listener="#{bean.listener}" />
        </h:commandButton>
    </h:form>
</h:body>
</html>
```

Ajax on javascript event

The date is updated whenever user types on the input field:

Bean.java

```
@ManagedBean
@ViewScoped
public class Bean {

    public Date getCurrentDate(){
        return new Date();
    }

}
```

sample.xhtml

```
<html xmlns="http://www.w3.org/1999/xhtml"
    xmlns:h="http://xmlns.jcp.org/jsf/html"
    xmlns:f="http://xmlns.jcp.org/jsf/core"
    xmlns:ui="http://xmlns.jcp.org/jsf/facelets">
<h:head />
<h:body>
    <h:form>
        <h:inputText>
            <f:ajax event="keyup" render="output" />
        </h:inputText>
        <p>
            <h:outputText id="output" value="Ajax date: #{bean.currentDate}" />
        </p>
    </h:form>
</h:body>
</html>
```

Delay

Executes the requests with the specified delay in milliseconds, meaning that if any subsequent request happens after the previous has been queued, the first one will be skiped. The feature is available starting from JSF 2.2:

Bean.java

```
@ManagedBean
@ViewScoped
public class Bean {

    public Date getCurrentDate(){
        return new Date();
    }

}
```

sample.xhtml

```
<html xmlns="http://www.w3.org/1999/xhtml"
    xmlns:h="http://xmlns.jcp.org/jsf/html"
    xmlns:f="http://xmlns.jcp.org/jsf/core"
    xmlns:ui="http://xmlns.jcp.org/jsf/facelets">
<h:head />
<h:body>
    <h:form>
        <h:inputText>
            <f:ajax event="keyup" render="output" delay="2000" />
        </h:inputText>
        <p>
```

```
            <h:outputText id="output" value="Ajax date: #{bean.currentDate}" />
        </p>
    </h:form>
</h:body>
</html>
```

Read Ajax Integration online: https://riptutorial.com/jsf/topic/4916/ajax-integration

Chapter 3: Comments in JSF

Introduction

JSF as a markup language, supports comments of some parts of code, but we have be carefully, because if we use a normal HTML comment code like this: <!-- I want to comment the next button --> <!-- <h:commandButton value="Push" onclick="alert('Hello');" /> --> It's possible that it doesn't has commented anything. This is because JSF process this code as default, even if is commented between tags `<!--` and `-->`. There are two solutions to comment any JSF code

Syntax

- <ui:remove> JSF code that you want to comment </ui:remove>

Remarks

You can find more information at Oracle documentation:

- <ui:remove> at oracle.com
- SKIP_COMMENTS at facelets.java.net

Examples

Use tag

We need to use tag `<ui:remove>` and `</ui:remove>` between any JSF code that we want to comment it.

```
<ui:remove>
    <h:outputLabel value="Yeah, I'm really commented" />
</ui:remove>
```

Of course you need add this xmlns to your header html tag. Check this minimal full example:

```
<html xmlns="http://www.w3.org/1999/xhtml"
    xmlns:h="http://java.sun.com/jsf/html"
    xmlns:ui="http://java.sun.com/jsf/facelets">

    <ui:remove>
        <h:outputLabel value="Yeah, I'm really commented" />
    </ui:remove>

</html>
```

Configure facelets.SKIP_COMMENTS

You must to add to web.xml a configuration tag like this:

```
<context-param>
    <param-name>facelets.SKIP_COMMENTS</param-name>
    <param-value>true</param-value>
</context-param>
```

Now you can use normal HTML comments tag `<!--` and `-->`

```
<!--
    <h:outputLabel value="Yeah, I'm really commented" />
-->
```

Previuosly full example with facelets.SKIP_COMMENTS configured in web.xml will be:

```
<html xmlns="http://www.w3.org/1999/xhtml"
    xmlns:h="http://java.sun.com/jsf/html"
    xmlns:ui="http://java.sun.com/jsf/facelets">

    <!--
        <h:outputLabel value="Yeah, I'm really commented" />
    -->

</html>
```

Read Comments in JSF online: https://riptutorial.com/jsf/topic/8164/comments-in-jsf

Chapter 4: JSF Annotations

Remarks

I get many informations from this web sites:

- http://www.jmdoudoux.fr/java/dej/chap-annotations.html
- http://docs.oracle.com/javaee/6/tutorial/doc/girch.html

Examples

Introduction to annotations

Why annotations?

Generally we use annotation to facilitate the development and to make the code more clear and clean.

What are annotations?

Java 5 annotations provide standardization of metadata in a general goal. This metadata associated with Java features can be exploited in the compilation or execution.

Java was modified to allow the implementation of annotations:

- A dedicated syntax was added in Java to allow the definition and use of annotations.
- bytecode is enhanced to allow storage of annotations.

Where can annotations be used?

Annotations can be used with :

packages, classes, interfaces, constructors, methods, fields, parameters, variables or annotations themselves.

Categories of annotation

There are three categories of annotation:

- **Markers**: These annotations do not have an attribute

For example `@Deprecated`, `@Override` ...

- **Single value annotation**: these annotations have only one attribute

For example `@MyAnnotation ( "test")`

- **Full annotations**: these annotations have multiple attributes

For example `@MyAnnotation (arg1 = "test 3", arg2 = "test 2", arg3 = "test3")`

Like we see before you can create your own annotation

Managed bean scope annotation

Create managed bean

To create a manage bean you need the annotation `@ManagedBean`

for example:

```
@ManagedBean
public class Example {}
```

You need the package:

```
import javax.faces.bean.ManagedBean;
```

Managed bean Scope

We use annotations to define the scope in which the bean will be stored.

There are many scope of managed bean: `@NoneScoped`, `@RequestScoped`, `@ViewScoped`, `@SessionScoped`, `@ApplicationScoped`, ...

- Application (`@ApplicationScoped`): Application scope persists across all users' interactions with a web application.
- Session (`@SessionScoped`): Session scope persists across multiple HTTP requests in a web application.
- View (`@ViewScoped`): View scope persists during a user's interaction with a single page (view) of a web application.
- Request (`@RequestScoped`): Request scope persists during a single HTTP request in a web application.
- None (`@NoneScoped`): Indicates a scope is not defined for the application.
- Custom (`@CustomScoped`): A user-defined, nonstandard scope. Its value must be configured as a `java.util.Map`. Custom scopes are used infrequently.

Read JSF Annotations online: https://riptutorial.com/jsf/topic/7021/jsf-annotations

Chapter 5: JSF Templates

Remarks

JSF provides special tags to create common layout for a web application called **facelets** tags. These tags gives flexibility to manage common parts of a multiple pages at one place.

Namespaces:

```
xmlns:h="http://xmlns.jcp.org/jsf/html"
xmlns:ui="http://xmlns.jcp.org/jsf/facelets"
```

Examples

How to create a template

Setting up a template for one application

Create a file named `template.xhtml` under the `/WEB-INF` folder, that way the template files will be accessible only for the framework.

`/WEB-INF/template.xhtml`

```
<!DOCTYPE html>
<html lang="en"
    xmlns="http://www.w3.org/1999/xhtml"
    xmlns:h="http://xmlns.jcp.org/jsf/html"
    xmlns:ui="http://xmlns.jcp.org/jsf/facelets">

    <h:head>
        <title><ui:define name="title">Default title</ui:define></title>
    </h:head>

    <h:body>
        <!-- Some styles that we might include for our whole application -->
        <h:outputStylesheet name="css/template.css" />

        <!-- Shared content for the application, e.g. a header, this can be an include -->
        <div>
            Application Header
        </div>

        <!-- The content we want to define in our template client -->
        <div>
            <ui:insert name="content" />
        </div>
    </h:body>
</html>
```

This file will act as a template for the application. Now we'll define a specific view in our view directory.

`/home.xhtml`

```
<ui:composition template="/WEB-INF/template.xhtml"
    xmlns="http://www.w3.org/1999/xhtml"
    xmlns:ui="http://xmlns.jcp.org/jsf/facelets">

    <ui:define name="title">Home</ui:define>

    <ui:define name="content">
        Welcome to the application!
    </ui:define>
</ui:composition>
```

Have a look at the `template` attribute in this client view, this tells JSF to use the template we want. Then, using `<ui:define>` we define specific content to be inserted where it is told in the template. Accessing `/home.xhtml` in client will render the whole result.

Read JSF Templates online: https://riptutorial.com/jsf/topic/5033/jsf-templates

Chapter 6: The Flash Scope in JSF 2

Remarks

The Flash concept is taken from Ruby on Rails and provides a way to pass temporary objects between the user views generated by the faces lifecycle. As in Rails, anything one places in the flash will be exposed to the next view encountered by the same user session and then cleared out. It is important to note that “next view” may have the same view id as the previous view.

The JSF implementation must ensure the proper behaviour of the flash is preserved even in the case of a `<navigation-case>` that contains a `<redirect />`. The implementation must ensure the proper behavior of the flash is preserved even in the case of adjacent GET requests on the same session. This allows Faces applications to fully utilize the “Post/Redirect/Get” design pattern.

Examples

Demonstration of the Flash Scope usage

Bean1

```
@ManagedBean
@ViewScoped
public class Bean1 implements Serializable {

    /**
     * Just takes the given param, sets it into flash context and redirects to
     * page2
     *
     * @param inputValue
     * @return
     */
    public String goPage2(String inputValue) {
        FacesContext.getCurrentInstance().getExternalContext().getFlash()
                .put("param", inputValue);
        return "page2?faces-redirect=true";
    }

}
```

page1.xhtml

```
<html xmlns="http://www.w3.org/1999/xhtml"
    xmlns:h="http://java.sun.com/jsf/html"
    xmlns:ui="http://java.sun.com/jsf/facelets"
    xmlns:c="http://java.sun.com/jsp/jstl/core">
<h:head />
<h:body>

    <!-- Sets the first flash param at the action method -->
    <h:form>
        <h:inputText value="#{inputValue}" />
        <h:commandButton action="#{bean1.goPage2(inputValue)}"
            value="Go Page 2" />
    </h:form>

    <!-- Sets the second flash param -->
    <c:set target="#{flash}" property="param2" value="Myparam2" />

    <!-- Tries to retrieve both of the params.
    Note none of them is displayed at the first page hit.
    If page refreshed, the second param which has been already set, will be displayed -->
    <p>Param1: #{flash['param']}</p>
    <p>Param2: #{flash['param2']}</p>
</h:body>
</html>
```

Bean2

```
@ManagedBean
@ViewScoped
public class Bean2 implements Serializable {

    public String getParam() {
        /**
         * Takes the parameter from the flash context
         */
        return (String) FacesContext.getCurrentInstance().getExternalContext()
                .getFlash().get("param");
    }

}
```

page2.xhtml

```
<html xmlns="http://www.w3.org/1999/xhtml"
    xmlns:h="http://java.sun.com/jsf/html"
    xmlns:ui="http://java.sun.com/jsf/facelets"
    xmlns:f="http://java.sun.com/jsf/core">
<h:head />
<!-- This page just displays the received params -->
<h:body>
    <!-- Different ways to retrieve the params from the flash scope -->
    <p>Param1: #{bean2.param}</p>
    <p>Param1: #{flash.param}</p>
    <p>Param1: #{flash['param']}</p>
    <p>Param2: #{flash['param2']}</p>

    <!-- Keep the first param for next redirection -->
    #{flash.keep.param}

    <!-- Return to page1 and see how the first param is retained -->
    <h:button outcome="page1?faces-redirect=true" value="return to 1" />
</h:body>
</html>
```

Read The Flash Scope in JSF 2 online: https://riptutorial.com/jsf/topic/3906/the-flash-scope-in-jsf-2

Credits

S. No	Chapters	Contributors
1	Getting started with jsf	BalusC, Community, Emil Sierżęga, Pixelstix
2	Ajax Integration	Xtreme Biker
3	Comments in JSF	albertoiNET
4	JSF Annotations	Right leg, YCF_L
5	JSF Templates	BalusC, DimaSan, Emil Sierżęga, Xtreme Biker
6	The Flash Scope in JSF 2	Xtreme Biker