trainologic

JSF

trainologic

Custom Components

- There are several ways of creating custom components in JSF.
  - Custom tags.
  - Composites.
  - Custom components.

- Remember before you go into one of these, search if the components at hand can provide a solution for your problem.

trainologic

Custom Tags

- Custom tags allow for reducing code duplications (Facelets code).
- It's a way to register a template under a tag name.

- Let's see how it is done…

trainologic

Custom Tags

- The template for the tag: (showandget.xhtml):

```
<!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:ui="http://xmlns.jcp.org/jsf/facelets"
xmlns:h="http://xmlns.jcp.org/jsf/html"
xmlns:f="http://xmlns.jcp.org/jsf/core">


<ui:composition>
  <h:outputText value="#{show_attr}" />
  <h:inputText value="#{get_attr}" />
</ui:composition>
</html>
```

trainologic

Custom Tags

- A descriptor file (suffix ‘taglib.xml’ – example.taglib.xml):

```
<?xml version="1.0" encoding="UTF-8"?>
<facelet-taglib 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-facelettaglibary_2_2.xsd"
version="2.2">
<namespace>http://trainologic.com/example</namespace>
    <tag>
        <tag-name>showAndGet</tag-name>
        <source>tags/showandget.xhtml</source>
    </tag>
</facelet-taglib>
```

trainologic

Custom Tags

- web.xml registration:

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://xmlns.jcp.org/xml/ns/javaee"
xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
http://xmlns.jcp.org/xml/ns/javaee/web-app_3_1.xsd"
version="3.1">

<servlet> …..</…>

<context-param>
  <param-name>javax.faces.FACELETS_LIBRARIES</param-name>
  <param-value>/WEB-INF/example.taglib.xml</param-value>
</context-param>


</web-app>
```

trainologic

Custom Tags

- Now use in Facelets: (hello.xhtml)

```
<html xmlns="http://www.w3.org/1999/xhtml"
xmlns:ui="http://java.sun.com/jsf/facelets"
xmlns:h="http://java.sun.com/jsf/html"
xmlns:f="http://java.sun.com/jsf/core"
xmlns:example="http://trainologic.com/example">
<h:body>
Hello <h:outputText value="#{name.fullName}"></h:outputText>
<h:form>


<example:showAndGet  show_attr="please enter you age" get_attr="#{name.age}"/>

</h:form>
</h:body>
</html>
```

trainologic

Composites

- Composite pretty much provides the same, however without the registration of custom tag.

- Let’s see an example…

trainologic

Composites

- First the template file content:

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


<composite:interface name="promptAndAsk">
  <composite:attribute name="prompt" />
  <composite:attribute name="ask" />
</composite:interface>

<composite:implementation>
  <h:outputText value="#{cc.attrs.prompt}" />
  <h:inputText value="#{cc.attrs.ask}" />
</composite:implementation>

</html>
```

trainologic

Composites

- The previous file must be saved under a path:

  /resources/*your_namespace* (in our example resources/trainologic).

- The file name will by default be the name of the tag.

trainologic

Composites

- Now, the usage:

```
<html xmlns="http://www.w3.org/1999/xhtml"
xmlns:ui="http://java.sun.com/jsf/facelets"
xmlns:h="http://java.sun.com/jsf/html"
xmlns:f="http://java.sun.com/jsf/core"
xmlns:trainologic="http://java.sun.com/jsf/composite/trainologic"
>
<h:body>
<h:form>

<trainologic:promptAndAsk prompt="please enter you age"
ask="#{name.age}"/>

</h:form>
</h:body>
</html>
```

trainologic

Discussion

- Although it looks like the composite approach is better than the custom tag, it's important to note the differences:
- A composition generate a single UIComponent.
- A custom tag may result in multiple components.

- Keep that an mind and apply each technique to the appropriate context.

trainologic

Custom Components

- Sometimes you can’t reuse existing components in the previous ways.
- You’ll need to create your own custom component.

- You can either create a custom renderer or a custom UI component or you can create both.

- Let’s go through an example..

trainologic

Custom Renderer

- A custom component:

```
@FacesComponent(value = "ShoutComponent", createTag = true, tagName = "shout",
namespace = "http://com.trainologic/tags")

public class ShoutComponent extends UIComponentBase {
 public ShoutComponent() {
  setRendererType("ShoutComponent");
 }


 @Override
 public String getFamily() {
  return "Shout";
 }
}
```

trainologic

Custom Renderer

- A custom renderer:

```
@FacesRenderer(componentFamily = "Shout", rendererType = "ShoutComponent")
public class ShoutRenderer extends Renderer {
 @Override
 public void encodeEnd(FacesContext context, UIComponent component) throws
       IOException {
  Map<String, Object> attributes = component.getAttributes();
  String message = (String) attributes.get("message");

  ResponseWriter writer = context.getResponseWriter();
  writer.startElement("h1", component);
  writer.write(message);
  writer.endElement("h1");
 }
}
```

trainologic

PrimeFaces

- A modern and comprehensive component library for JSF.

- Let’s go through the showcase first:
- https://www.primefaces.org/showcase/

trainologic

PF

- As you can note, some component examples use the PF(‘some id’) function.
- You can see an example in the ‘Signature’ component in the showcase.
- This is a JavaScript PrimeFaces function that accesses the relevant component by id.
- Client-side of-course.

trainologic

Exercise

File System Mind Map

trainologic

dataTable

- Somewhat resembles the regular h:dataTable.
- Much more powerful and performant.

- Let’s explore…

trainologic

AJAX

- Before JSF 2, there was no support for AJAX in JSF.
- Many component providers provides workarounds for AJAX support.
- Due to the fact that regular JSF lifecycle is not appropriate for AJAX.
- I.e., not all the components should participate in an AJAX request and we definitely don't want them to go through 'full render'.

trainologic

AJAX

- Starting with JSF 2, JSF can process partial requests.
- Can be checked by *PartialViewContext.isAjaxRequest()* method.
- You can also use the *isPartialRequest()* method as a generalization over AJAX.

trainologic

AJAX

- Ajax support in JSF2 is most commonly seen with the usage of <f:ajax> tag.
- It has two use-cases that depend on its placing:
  - Nested within an element.
  - Surrounding a group of elements.

- Let’s discuss these cases…

trainologic

Single Component Case

- When the <f:ajax> is nested inside a single element, it will specify which operation of this element will be Ajax-based.
- It can be nested within all of the standard JSF components (the HTML ones) and inside any custom component that implements the *ClientBehaviorHolder* interface.

- Let's get familiar with its attributes…

trainologic

Attributes

- event – the event of the component to be ajaxified. By default its either ‘action’ or ‘valueChange’ (depending on the component type.
- execute – see next page.
- render – see next page.
- immediate.
- disabled – default is ‘false’.
- listener – a method expression of a listener.
- onevent, onerror – JavaScript method names.

trainologic

execute & render

- The *execute* and *render* attributes can specify a whitespace separated list of component ids to participate in the ‘execute’ or ‘render’ lifecycle phases.
- By default the keyword ‘@this’ is provided (which specifies the current component that triggered the event.
- More supported keywords: @all, @form and @none.

trainologic

Multiple Components Case

- When the f:ajax tag surrounds a group of components, it ajaxifies them all.