Facelets & Web services

GÖTEBORGS UNIVERSITET

DAT076/DIT126 Web Applications

Adam Waldenberg

Facelets in short

Lightweight page declaration language used to build Java Server Faces views using HTML style templates and component trees

- XHTML
- Support for comprehensive templating
- Composite components
- Original version developed back in 2005 by Jacob Hookom as a response to the default JSP-based view declaration method
- If used correctly, it saves you huge amounts of time and simplifies your view definitions

Facelets templating

Facelets offers us a number of methods to facilitate templating

- `<ui:composition>` Defines a composition which essentially is the section that defines the templating input
- `<ui:decorate>` Like a composition, but does not disregard content outside of the tag
- `<ui:insert>` Used to define a templateable section
- `<ui:define>` Used to define content to be inserted into a templateable section

Facelets templating continued

Time for an example template

```
<html xmlns="http://www.w3.org/1999/xhtml" xmlns:h="http://xmlns.jcp.org/jsf/html"
      xmlns:ui="http://xmlns.jcp.org/jsf/facelets"
      xmlns:f="http://xmlns.jcp.org/jsf/core">
    <h:head>
        <title>Example</title>
    </h:head>
    <h:body>
        <section>
            <div class="container">
                <ui:insert name="content" />
            </div>
        </section>
    </h:body>
</html>
```

- What is happening here?
- How do we utilize it?

Facelets templating continued

This is how the consuming page would look

```
<html xmlns="http://www.w3.org/1999/xhtml" xmlns:h="http://xmlns.jcp.org/jsf/html"
      xmlns:ui="http://xmlns.jcp.org/jsf/facelets"
      xmlns:f="http://xmlns.jcp.org/jsf/core">
    <ui:composition>
        <ui:define name=”content”>
            <p>
                Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do
                eiusmod tempor incididunt ut labore et dolore magna aliqua.
            </p>
        </ui:define>
    </ui:composition>
</html>
```

- We can define as many templating definitions as we need
- Everything outside the composition is ignored in the resulting page

Facelets templating example

Let’s take an example to see how templating works!

Facelets composite components

A mechanism for declaring custom components based on a collection of other components

- Can utilize external component frameworks
- Compared to a normal JSF component, there is no need for a Java file defining how the component is rendered
- Allows for the definition of a custom attributes
- Combined with JSTL it allows for the inclusion of programmatic logic, allowing us to define very rich components without the need of any Java code

Composite components

Facelets offers us a number of methods to facilitate composite components. These are the most commonly used

- `<cc:interface>` Defines the prototype and usage contract of the element
- `<cc:attribute>` Defines an attribute. Can later be accessed via expression language and `` `#{cc.attrs.attributeName}` ``

  `<cc:implementation>` Defines the composite component and what it renders
- `<cc:facet>` Similar to `<define>` in facelet compositions allowing you to template composite components
- `<cc:renderFacet>` Companion to the above tag – similar to `<insert>`
- `<cc:insertChildren>` Inserts all the passed children of the component here. Used under the implementation section

Configuring for composite components

We need to give `javax.faces.WEBAPP_RESOURCES_DIRECTORY` a path to `WEB-INF/` in order to make the composite components inaccessible from the outside

```
<context-param>
    <param-name>javax.faces.WEBAPP_RESOURCES_DIRECTORY</param-name>
    <param-value>WEB-INF/resources</param-value>
</context-param>
```

- Added to your *web.xml*

Composite components continued

Defining a composite component is easy

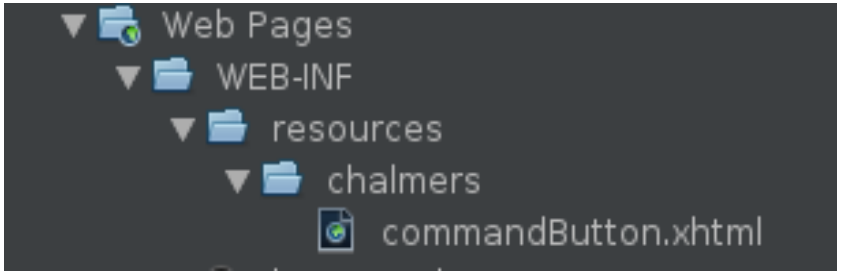
```
<ui:composition
    Xmlns="http://www.w3.org/1999/xhtml"
    xmlns:ui="http://java.sun.com/jsf/facelets"
    xmlns:h="http://java.sun.com/jsf/html"
    xmlns:cc="http://java.sun.com/jsf/composite">
    <cc:interface>
        <cc:attribute name="value"/>
    </cc:interface>
    <cc:implementation>
        <h:commandButton image=”...” value="#{cc.attrs.value}"/>
    </cc:implementation>
</ui:composition>
```

- Defines a replacement commandButton that sets an image by default
- The interface section defines the prototype

Composite components naming scheme

Facelets will look for your locally defined component libraries in your resources directory

- *The path where it looks is defined as;*
  `[javax.faces.webapp.FacesServlet]/<library>/<componentName>.xhtml`



- Will define the namespace `http://xmlns.jcp.org/jsf/composite/<library>` and define the component `<componentName>` under it

- To use it, you simply reference it in your XML file
- Once defined, they are used like any other JSF component

Facelets templating example

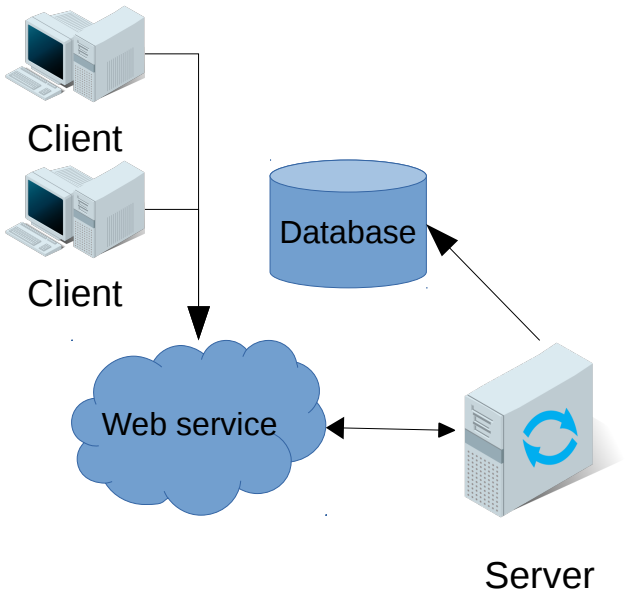
Let’s take another example to see how composite components actually work!

What are web sevices?

A collection of open protocols and standards used for exchanging data between applications or systems

- Is available over the Internet or private (intranet) networks
- Uses a standardized messaging system (protocol)
- Is not tied to any one operating system or programming language
- Is discoverable via a simple find mechanism

What are web sevices?, continued



A communication layer between clients, frontends and the backend

- Even if Java Server Faces is used, many sites still expose a web service to allow external devices to access and communicate with the backend
- A web service is an API – but an API is not necessarily a web service
- Many standards exist; SOAP, REST, RPC and probably a few more. REST being the most common

RESTful web services

"By using a stateless protocol and standard operations, RESTful systems aim for fast performance, reliability, and the ability to grow by reusing components that can be managed and updated without affecting the system as a whole, even while it is running"

- Defined back in 2000 by Roy Fielding
- Very simple, which is why it has become so popular
- Accessed via a base URL, such as ***`http://api.example.com/collection`***
- Utilizes standard HTTP methods (*GET*, *POST*, *PUT*, *PATCH* and *DELETE*)
- Defines media types that the web service can consume or produce as a response to a request

REST - A product register

Here is an example of a product register implemented via a REST API

• /products	GET	Fetches a list of all products
• /products	POST	Creates a new product
• /products/id	GET	Returns any product with <id>
• /products/id	PUT	Replace the product with <id>
• /products/id	DELETE	Delete the product with <id>
• /productgroups/id	GET	Return product group with <id>
• /products/id	PATCH	Update an existing product <id>

Web services in EE

Java / Jakarta EE offers an extensive tool set for developing web services and connecting them to the backend

- **JAX-B** API for serializing and de-serializing XML back and forth between Java classes and XML
- **JSON-B** The above equivalent for JSON (new with EE8)
- **JSON-P** API to programmatically construct JSON strings, used before EE8 and should not really be used with newly developed applications
- **JAX-RS** Annotation-driven API to define methods, paths and other properties that define a REST web service

Some JSON-B annotations

JSON-B has a number of annotation that allow you to control the behavior of serialization and de-serialization

• **`@JsonbProperty("name")`**	Change the name of one particular property
• **`@JsonbPropertyOrder(...)`**	Change the order of properties in the generated JSON
• **@JsonbTransient**	Used to ignore properties
• **@JsonbNillable, @JsonbProperty(nillable=true)**	Allows null fields to be serialized
• **@JsonbCreator**	Used to customize the constructor

Some JAX-RS annotations

With JAX-RS you define the location of your web service, the path locations, endpoints, parameters and the behavior of each endpoint

• **`@Path("name")`**	Identifies the URI path. It can be specified on class or method
• **`@PathParam("name")`**	Represents a parameter of the URI path
• **@HTTP_METHOD**	Can be *@POST, @PUT, @DELETE, @GET* etc
• **@Consumes("mime"...)** **@Produces("mime"...)**	Specifies the mime type that a particular endpoint or path consumes or produces

More JAX-RS annotations

- **@FormParam** Represents a parameter coming from the request form
- **@QueryParam** Represents a parameter coming from the query part of the URL
- **@CookieParam** Represents a parameter coming from the specified cookie
- **@HeaderParam** Represents a parameter coming from the reqest header

- These parameter types are more useful than they seem and allows the web service to store data hidden from view. They can even be used for things like security tokens

A simple REST web service in EE

A very simple web service serving a list of products

```
@Path("shop")
public class ShopResource {
    @EJB
    private Shop shop;

    @GET
    public List<Product> list() {
        return shop.getProducts();
    }
}
```

- Automatically unserializes the list of products and creates a JSON response!
- All the bells and whistles of EJB’s and CDI can be used directly in the web service definition

An example with path parameters

```
@Path("shop")
public class ShopResource {
    @EJB
    private Shop shop;

    @GET
    @Path("{from}/{to}")
    public List<Product> range(@PathParam(“from”) int from
                               @PathParam(“to”) int to) {
        return shop.getProducts().subList(from, to);
    }
}
```

- Notice the correlation between the defined parameters in *@Path* and *@PathParam*

Setting up for web services

Before web services work we need to make sure the JAX-RS servlet is running and serving the web service

```
@ApplicationPath("ws")
public class JAXRSConfiguration extends Application {
    /* Intentionally left blank */
}
```

- This starts the JAX-RS servlet and tells it to serve all our resources under ***<application-context>/ws/***
- Note! With prior versions of EE we had to edit the ***web.xml*** file and add the JAX-RS servlet manually – since JavaEE 8, this is no longer needed!

Web services example

Let’s take a look at an example of a simple web service implemented with JAX-RS!