UPPSALA
UNIVERSITET

Web Services and JSF

Carl Nettelblad *2015-05-13*

UPPSALA UNIVERSITET

Outline

- Assignment 2
- Web services
- Java Server Faces

UPPSALA UNIVERSITET

Assignment 2

- Model View Controller

- Model
  - Beans representing data
  - Beans representing logic
  - Beans should ideally perform business logic operations
    - Not “perform database query”
    - Rather “get user object [from database]”
  - Anything that is not directly tied to your web application

UPPSALA UNIVERSITET

MVC

- Controller
  - The controller can be our servlet
    - We can have a command pattern as well
  - The controller makes business logic requests to the beans
  - Decides what *view* to render
  - Sending data to the view

UPPSALA UNIVERSITET

View

- Should be concerned with the actual presentation to the user
  - Through HTML
- In our case JSP
  - The logic should be concerned with how to create HTML from our data
  - The data should already be ready from the controller and beans
  - Avoid actions that modify state inside view code
- So, do not:
  - Add database rows
  - Modify the session halfway through the page
  - Those things most likely belong in the controller
    - Which might ask the model layer to execute them

UPPSALA UNIVERSITET

How to specify data sources

- In general, avoid “magic constants” in code
  - Things that appear in *multiple places* should surely go into the class, not individual methods
- This include connection strings
- Connection strings are dependent on deployment environment
  - Should be changeable without changing code
  - Properties files would be one option

UPPSALA UNIVERSITET

Configuration

- Java EE provides the facilities to provide configuration
  - Servlet initial parameters
    - `getServletConfig().getInitParameter("name")`
  - Application context parameters
    - `getServletContext().getInitParameter("name")`

UPPSALA UNIVERSITET

From web.xml

```
<!-- This is a context init parameter -->
   <context-param>
       <param-name>email</param-name>
       <param-value>admin@example.com</param-value>
   </context-param>

   <servlet>
       …
       <!-- This is a servlet init parameter -->
       <init-param>
           <param-name>name</param-name>
           <param-value>John Doe</param-value>
       </init-param>
   </servlet>
```

UPPSALA UNIVERSITET

What's the difference?

- Per servlet or for the full application
- The data source can probably be application-wide

UPPSALA UNIVERSITET

Resources

- Data sources can be using connection pools
  - Different JDBC drivers
  - Secret user names and passwords
    - Well, at least you don’t want them all over the place
- Specify the full data source as a resource in the container
- In GlassFish, they can be per-container or per-application

UPPSALA UNIVERSITET

JNDI

- Names for resources etc are handled using Java Naming and Discovery Interface (JNDI)
- You can have different contexts and contexts that map to each other
- Once a name is establised, it can hide mapping to other names

- The `@Resource` annotation is dangerous in the sense that you can specify a name for two purposes
  - Bind to an existing name
  - Put a resource to a new name
- Use the `@Resource(lookup="JNDI-name")` annotation instead
  - Will give you errors rather than reassign a name to the default data source

UPPSALA UNIVERSITET

SQL injection

- `String sql = "SELECT * FROM Users WHERE username = '" + request.getParameter("username") + "'";`

- Wrapped in single quotes, what if the request parameter contains single quotes as well

- Solution?
  - Verify your inputs
    - Might be good, but hard to do right for SQL syntax
  - Better up: parametrized queries

UPPSALA UNIVERSITET

Prepared statements

- Parsing SQL strings can be time-consuming
- Stored procedures
  - Store complex SQL scripts in the database manager
- Prepared statements
  - Give the JDBC driver a SQL string where parameter blanks are later filled in
  - Both driver and database can cache these
  - Higher performance
  - Separation between data and code (avoid SQL injection)

UPPSALA UNIVERSITET

Example, prepared statements

```
PreparedStatement pstmt;
pstmt = connection.prepareStatement(
"INSERT INTO PERSON (pnr,firstname,lastname,age) VALUES (?,?,?,?)");

pstmt.setInt(1, 1);
pstmt.setString(2, "Nisse");
pstmt.setString(3, "Nilsson");
pstmt.setInt(4, 37);
int x = pstmt.executeUpdate();
```

- Note: 1-based indexing (parameters start at 1).
- Why?
  - Legacy history, JDBC was inspired by ODBC
  - Visual Basic and other technologies in use at the time were more likely to be 1-based

UPPSALA UNIVERSITET

Cross-site scripting

- `<%= "<script>" %>`
- `<c:out value="<script>" />`
- `${"<script>"}`
- `out.println("<script>");`

- What's the difference?

UPPSALA UNIVERSITET

Cross-site scripting

- `<%= request.getParameter("username") %>`
- `<c:out value="${request.username}" />`
- `${request.username}`
- `out.println(request.getParameter("username");`

- Only `c:out` is safe out of these!
- The JSTL `fn:escapeXml` function (bring in the fn taglib) can also be used in EL

UPPSALA UNIVERSITET

XML API in Java

- Streaming data:
  - Simple API for XML `org.xml.sax`
    - Event-driven, callbacks
  - XML stream API `javax.xml.stream`
    - Reading methods

- Reading full file:
  - Document object model `org.w3c.dom`
    - Object model
  - Java XML Binding JAXB `javax.xml.bind`
    - Mapping XML elements to Java objects

UPPSALA UNIVERSITET

DOM, pros and cons

- Pro
  - Navigate freely in the full structure
- Cons
  - Very verbose syntax
    - Explicitly look up attributes, elements, inner text as separate nodes and look at them
  - Not the best performance
    - Full document structure created in memory in rather inefficient objects
      - Time used to create objects
      - Memory used to store them

UPPSALA UNIVERSITET

SAX, pros and cons

- Pro
  - Resource-efficient
- Cons
  - Complex keeping track of state
  - Only pass everything once
    - To go back, you need to parse the full file again
    - Or store the relevant parts yourself

UPPSALA UNIVERSITET

StAX, pros and cons

- Pros
  - Somewhat easier programming model (pull rather than push)
  - Object-oriented structure and “real” iterator for iterator API
- Cons
  - Still need to handle the stream of individual XML events
  - Iterator API can result in very many object allocations in real high-performance niches

UPPSALA UNIVERSITET

Pros and cons JAXB

- Pros
  - Very convenient programming model
  - Easy mapping to/from schemas
- Cons
  - Slower than SAX/StAX
    - Actually not necessarily slower than full DOM!
  - Loss of control
  - Updating a single element means messing up the full file structure
    - Comments, formatting lost

UPPSALA UNIVERSITET

XSL Transforms

- XML Stylesheet Transforms, namespace http://www.w3.org/1999/XSL/Transform
- Map from one XML document into
  - Another XML document
  - HTML
  - Text
- Not quite a programming language, but pretty close
  - Declarative
- Tightly based on XPath

UPPSALA UNIVERSITET

Overall structure

- Templates matching XPath expressions
  - Some similarities to JSTL, but XPath rather than EL
  - Each template can match a specific expression defined in the `match` attribute
  - Common to have a root template, matching `/`
  - Inside the template we can have arbitrary XML and XSL instructions within the XSL namespace

UPPSALA UNIVERSITET

Two ways to do a <singer> tag

```
<xsl:template match="/">
  <xsl:for-each select="catalog/cd">
    <xsl:element name="singer">
      <xsl:value-of
          select="artist" />
    </xsl:element>
    <br />
  </xsl:for-each>
</xsl:template>
```

```
<xsl:template match="/">
  <xsl:for-each select="catalog/cd">
    <singer>
      <xsl:value-of select="artist" />
    </singer>
    <br />
  </xsl:for-each>
</xsl:template>
```

UPPSALA UNIVERSITET

Using XSLT in Java

- Go from an XML source to an XML result
  - All the Java XML APIs provide workable sources and results
  - `javax.xml.transform` API complements DOM, SAX, StAX and JAXB
- Example using StAX

```
Source xmlInput = new StreamSource(new File("input.xml"));
Source xsl = new StreamSource(new File("file.xsl"));
Result xmlOutput = new StreamResult(new File("output.xml"));

try {
    Transformer transformer = TransformerFactory.newInstance().newTransformer(xsl);
    transformer.transform(xmlInput, xmlOutput);
} catch (TransformerException e) {
    // Handle.
}
```

UPPSALA UNIVERSITET

XSLT on the client

- Imagine you have an XML file
- Users might try to read that directly
- You can provide a stylesheet to render it as XML in the browser

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" href="cdcatalog.xsl"?>
<catalog>
  <cd>
    <title>Empire Burlesque</title>
    <artist>Bob Dylan</artist>
    <country>USA</country>
    <company>Columbia</company>
    <price>10.90</price>
    <year>1985</year>
  </cd>…
```

UPPSALA UNIVERSITET

Web services

- SOAP
  - Standard for communicating requests and responses representing XML infosets
  - Generally method calls to stateless methods
  - HTTP is one transport for this
- WSDL (Web Services Description Language)
  - XML language describing the interface to a SOAP service
- SOAP can go over HTTP
  - But also (in theory) many other transports
- JAX-WS API in Java

UPPSALA UNIVERSITET

Annotations

- Netbeans offers wizard to create a web service wrapper for an EJB
- Annotate your class using `@WebService`
  - `serviceName` parameter controls name of end-point
    - Otherwise “unqualified class name + Service”
- Cross-platform WSDL auto-generated and published at end-point URL + “`?wsdl`”

UPPSALA UNIVERSITET

Using a SOAP service with WSDL

- You will need to generate a client for your web service
  - `wsimport` tool
  - **New Web Service Client** in Netbeans
- Generates code based on WSDL

UPPSALA UNIVERSITET

REST

- Representational state transfer
- HTTP was not created to do method calls
- HTTP *was* created to handle downloads and uploads of resources
- Represent our data as such resources
  - Use the verbs of HTTP
  - GET means get, PUT means replace, POST means add, DELETE means delete
  - Use a relevant *content type* (generally JSON, XML)
  - The response should only be the data

UPPSALA UNIVERSITET

JSON

- JavaScript Object Notation
- Represent structured data using a restricted syntax similar to a Javascript dictionary
  - Braces, key : value
  - Nested dictionaries and arrays possible (braces and brackets)
- Never actually evaluate JSON from an unsafe source as full Javascript
  - Fantastic injection/cross-site scripting attacks possible
- But very easy to use in a web browser script

UPPSALA UNIVERSITET

JSON in REST

- Don’t include all information in one big response
- Use subpaths and links instead
  - Refer to other resources by their URLs

- Facebook Graph API is a somewhat realistic example of this
- Refer to any user, page etc by their ID (handle or numerical ID)
  - https://graph.facebook.com/uppsala.universitet
  - https://graph.facebook.com/uppsala.universitet/photos
  - https://graph.facebook.com/uppsala.universitet/events
  - https://graph.facebook.com/uppsala.universitet/feed

UPPSALA UNIVERSITET

/uppsala.universitet


```
{
   "id": "54601820767",
   "about": "H\u00e4r kan du st\u00e4lla fr\u00e5gor om studier och studentliv eller annat
som r\u00f6r universitetet. Vad h\u00e4nder p\u00e5 campus? Var finns de b\u00e4sta
pluggst\u00e4llena? Vilken nation ska man v\u00e4lja?  ",
   "can_post": false,
   "category": "University",
   "checkins": 0,
   "cover": {
      "cover_id": "10152023174380768",
      "offset_x": 0,
      "offset_y": 0,
      "source": "https://scontent.xx.fbcdn.net/hphotos-xfp1/t31.0-
8/s720x720/887088_10152023174380768_409813962_o.jpg",
      "id": "10152023174380768"
   },
   "description": "Kvalitet, kunskap och kreativitet sedan 1477.  Uppsala universitet \u00e4r
ett av norra Europas h\u00f6gst rankade l\u00e4ros\u00e4ten. Forskning i v\u00e4rldsklass och
h\u00f6gklassig utbildning till global nytta f\u00f6r samh\u00e4lle, n\u00e4ringsliv och
kultur.\n\n",
   "founded": "1477",...
```

UPPSALA UNIVERSITET

/uppsala.universitet/photos


```
{
   "data": [
      {
         "id": "10152354841805768",
         "created_time": "2014-05-05T07:03:54+0000",
         "from": {
            "category": "University",
            "name": "Uppsala universitet",
            "id": "54601820767"
         },
         "height": 180,
         "icon": "https://fbstatic-a.akamaihd.net/rsrc.php/v2/yz/r/StEh3RhPvjk.gif",
         "images": [
            {
               "height": 180,
               "source": "https://scontent.xx.fbcdn.net/hphotos-xaf1/v/t1.0-
9/10338254_10152354841805768_7394580631421209662_n.jpg?oh=270dd47d761b8984af1db95db94fdf1f&oe=55C99E02",
               "width": 180
            },
            {
               "height": 130,
               "source": "https://scontent.xx.fbcdn.net/hphotos-xaf1/v/t1.0-
9/p130x130/10338254_10152354841805768_7394580631421209662_n.jpg?oh=ec4e109c97e567ea596f7c96de986643&oe=560C
865A",
               "width": 130
```

UPPSALA UNIVERSITET

/uppsala.universitet/events


```
{
   "error": {
      "message": "An access token is required to request this
resource.",
      "type": "OAuthException",
      "code": 104
   }
}
```

UPPSALA UNIVERSITET

What?

- Yeah, many REST APIs require you to have some kind of token or API key
  - Often sent as a query string
  - Sometimes cookies

UPPSALA UNIVERSITET

Why is REST useful?

- The unique URL per resource pattern matches the structure of the web
- Caching, intercepting requests, logging etc all work “straight away”
- In contrast, it is hard to ever properly cache a SOAP response
- Responses very content-focused and concise compared to arbitrary XML wrapped into SOAP

UPPSALA UNIVERSITET

JAX-RS

- An EJB can also be exposed directly as REST services
- `@Path` annotation on class level
- `@GET`, `@PUT`, `@POST`, `@DELETE`
  - Which verbs are handled by a specific method
- `@Produces({"application/xml", "application/json"})`
  - Return value will be encoded as this content type, depending on client request (ordering determines default)
  - You can specify just one as well, or things like `text/plain` or `text/html` if you return strings
  - JAXB is extended to also create corresponding mapping as JSON
- Matching `@Consumes` tag for how to interpret posted data
- Subpaths for individual methods

UPPSALA UNIVERSITET

Application config

- XML or a specific class
  - What classes are exposed
  - What's the entry point path for all RESTLET services
- Optional
  - Register serialization and deserialization mechanisms
  - Enable other features
    - Security layers etc

UPPSALA UNIVERSITET

Path parameters

- Subpaths can include placeholders, like

```
@Path("image/{id}")
public Image getImageData(@PathParam("id") String id);
```

- The Java API for this method is then `bean.getImageData(id)`
- The RESTFUL API is `http://server/.../image/`*`id`*
- `@CookieParam` – map to request cookie
- `@QueryParam` – map to named query string parameter

UPPSALA UNIVERSITET

Design

- Overall, good REST APIs send data in the content
  - Intent is conveyed in the path
  - Different simple methods, not a lot of arguments to the same method
  - You might wrap a more complex method with many options by several simpler paths
- A single parameter without any decoration and a proper `@Consumes` tag is enough in many case
  - You can have multiple level of path params `/{user}/{language}/{id}`
  - But this is rarely clean

UPPSALA UNIVERSITET

From the JavaEE 7 tutorial

- A coffee house keeping beans
  - Pun on words…
- Accessed through `/coffeebeans` in the web application
- Just get on that URL gets a list of all beans
- You can also refer to specific ones by their ID
- You can add a new one by a POST to the main URL
- You can delete a specific ID
- You can initiate roasting of a bean by calling `/coffeebeans/roaster/id`

UPPSALA UNIVERSITET

Example

```
@Path("coffeebeans")
public class CoffeeBeansResource {

    @Context
    ResourceContext rc;

    Map<String, Bean> bc;

    @PostConstruct
    public void init() {
        this.bc = new ConcurrentHashMap<>();
    }

    @GET
    public Collection<Bean> allBeans() {
        return bc.values();
    }

    @GET
    @Path("{id}")
    public Bean bean(@PathParam("id") String id) {
        return bc.get(id);
    }
```

Example

```
    @POST
    public Response add(Bean bean) {
        if (bean != null) {
            bc.put(bean.getName(), bean);
        }
        final URI id = URI.create(bean.getName());
        return Response.created(id).build();
    }

    @DELETE
    @Path("{id}")
    public void remove(@PathParam("id") String id) {
        bc.remove(id);
    }

    @Path("/roaster/{id}")
    public RoasterResource roaster(){
        return this.rc.initResource(new RoasterResource());
    }
}
```

UPPSALA UNIVERSITET

Example

- The last part actually hands off the handling of the `/roaster` URL to a separate class
- The roaster starts to demonstrate asynchronous processing, but actually completes it directly here, see the full Java EE 7 JAX-RS tutorial from Oracle

- The ID and path information is kept in the `ResourceContext` and injected by the `initResource` call

UPPSALA UNIVERSITET

Example

```
public class RoasterResource {

    @PathParam("id")
    private String id;

    @POST
    public void roast(@Suspended AsyncResponse ar, Bean bean) {
        try {
            Thread.sleep(2000);
        } catch (InterruptedException ex) {
        }
        bean.setType(RoastType.DARK);
        bean.setName(id);
        bean.setBlend(bean.getBlend() + ": The dark side of the bean");
        Response response = Response.ok(bean).header("x-roast-id", id).build();
        ar.resume(response);
    }
}
```

UPPSALA UNIVERSITET

Simple and complex responses

- JAX-RS provides two main ways to do responses
- Simply return the type you want to use
  - Serialized based on content type
- Or create a `Response` instance
  - In the latter case, you can control the return code
  - In this case 200 (OK), by calling the `ok` method with the object to return back
- Similar control to the `Response` object in servlets, but tailored towards web services

UPPSALA UNIVERSITET

All in one

- A RESTLET *can* be an EJB which uses JPA to access entities which are mapped using JAXB to both JSON and XML…
- Security annotation can work in RESTLETs as well
  - In some implementations, configuration needed
- These days, many applications can be closer to static HTML + CSS + Javascript
  - All interaction with the server backend (your Java code) actually taking place using web services

UPPSALA UNIVERSITET

JAX-RS for clients

- In JAX-RS 2.0 (in Java EE 7), JAX-RS was introduced for clients as well
  - Consume REST APIs in Java
- Difference to JAX-WS
  - Not really any widespread counterpart to WSDL

UPPSALA UNIVERSITET

API style

- Fluent interface
- Natural style of successive method calls

```
Client client = ClientBuilder.newClient();
String name = client.target("http://example.com/webapi/hello")
        .request(MediaType.TEXT_PLAIN)
        .get(String.class);
```

- You can of course save objects in variables along the way
- The response can be anything (no WSDL)
  - You specify expected type
  - Deserialization supported from JSON and XML

UPPSALA UNIVERSITET

Getting an object

```
int beanId = 42;
CoffeeBean bean =
client.target(REST_SERVICE_URL).path("/bean/{beanId}")
        .resolveTemplate("beanId",
        beanId).request().get(CoffeeBean.class);
```

UPPSALA UNIVERSITET

Java Server Faces

- Java Server Pages have a very simple lifecycle
- Remember the design of HTTP
  - Request followed by response
- The JSP is a template for generating a single response to a single request
- All code is executed in order to prepare that request

- One group tried to make a log-out link in assignment 2
  - `<form action="<%request.logout()%>">`

UPPSALA UNIVERSITET

Comparison to a GUI

- HTML is only a text representation of what is going to be presented on the client
  - Nothing is left on the server

- What if you did a desktop or mobile app like that
  - Draw the full window of Microsoft Word
  - The user presses a key
  - First determine where that keypress went
    - No reference of the window that has been drawn
  - Then redraw the full thing again

UPPSALA UNIVERSITET

GUI frameworks

- Swing and other GUI frameworks are stateful
  - They have an object representation of the windows (and controls/widgets) that are shown
  - You tie event handlers to specific events in those windows
  - An event handler might update a widget in some way
  - An event handler might also close the window and open another one
- You don’t write code to explicitly keep the window “alive” and maintain its state

UPPSALA UNIVERSITET

Purpose of JSF

- The design with a set of views is convenient
- In a rich application, you don't want each HTTP request to result in a "new" view
  - The user just pressed a button within an existing view
  - You *want* to be able to associate a logout method call to an event
  - And just update the parts of the view that are affected

UPPSALA UNIVERSITET

Actual structure of JSF

- XML document
  - Formally specified to be XHTML
  - In modern versions actually rendered as HTML5 (`<!doctype html>` sent to client no matter what you write in the source)
- Document can be a mix of
  - Plain HTML tags
  - JSF components
  - Expression language for data binding and actions
- Interacting with backing beans (model) and navigation rules

UPPSALA UNIVERSITET

Life cycle

- Create/**restore view**
  - If the view has already been shown, all state (not just the resulting HTML) is kept for each client on the server – **the component tree**
- **Apply request values**
  - Update the **component tree** based on information in the request
  - A model, not model in the MVC sense
- Process events
  - *Continued…*

UPPSALA UNIVERSITET

Life cycle

- **Process validations**
  - Rules can be set on valid field content
  - Automated mechanisms for reporting validation errors
- **Update model values**
  - Data is valid – update our backing beans
  - Errors can happen here as well
- **Invoke application**
  - Only now are we ready to take the next action (log in, log out, add an order, perform a search, etc)
- **Render response**
  - Based on the action, the view is updated or we navigate to a new view

UPPSALA UNIVERSITET

What would this look like for JSP?

- Process request (possibly in servlet)
- Render response

- Do you want to validate stuff? Do you want to update a bean based on form entry? Do it yourself. Are there multiple UI components on the same page, each having its own form? You'll have to keep them in sync yourself.

UPPSALA UNIVERSITET

Tag libraries

- Tags are brought in using XML namespaces
- Certain prefixes are common (just like for JSTL)
- http://xmlns.jcp.org/jsf/html h prefix
  - Basic elements wrapping HTML, `h:commandLink`, `h:inputText`, …
- http://xmlns.jcp.org/jsf/core f prefix
  - Actions in the processing steps
- http://xmlns.jcp.org/jsf/facelets ui prefix
  - Additional templating support
- http://xmlns.jcp.org/jsf/c parts of JSTL core
- http://xmlns.jcp.org/jsp/jstl/functions JSTL functions

UPPSALA UNIVERSITET

Special EL

- In JSP, expression language expressions are translated to values directly
  - “Immediate” expressions
- That’s no use if you want to specify where a form value should be stored
  - You want a reference, not the value itself
- `${}` and `#{}`
  - You will rarely explicitly want `${}` in JSF
    - Maybe useful as an experiment to understand the lifecycle
- Deferred expressions can refer to methods
  - Specify the action for an element using a deferred method expression

UPPSALA UNIVERSITET

Difference between component tree and rendered HTML

- You want the component tree to stay mostly the same
- You can decide what elements are really rendered (sent as client HTML) using the `rendered` attribute
- Not the same thing as making the element invisible in HTML

UPPSALA UNIVERSITET

Example

```
<html xmlns="http://www.w3.org/1999/xhtml"
xmnls:h="http://java.sun.com/jsf/html"
xmnls:f="http://java.sun.com/jsf/core">
<head><title>Newsletter Subscription</title></head>
<body>
<f:view>
<h:form>
<table>
<tr>
<td>Email Address:</td><td><h:inputText value="#{subscr.emailAddr}" /></td>
</tr>
<tr>
```

UPPSALA UNIVERSITET

Example

```
<td>Newsletters:</td>
<td>
<h:selectManyCheckbox value="#{subscr.subscriptionIds}">
<f:selectItem itemValue="1" itemLabel="JSF News" />
<f:selectItem itemValue="2" itemLabel="IT Industry News"/>
<f:selectItem itemValue="3" itemLabel="Company News"/>
</h:selectManyCheckbox>
</td>
</tr>
</table>
<h:commandButton value="Save" action="#{subscrHandler.saveSubscriber}" />
</h:form>
</f:view>
</body>
</html>
```

UPPSALA UNIVERSITET

Backing beans

- Lifetime of bean objects can be handled by JSF
- This is set in `faces-config.xml`

```
<managed-bean>
<managed-bean-name>subscr</managed-bean-name>
<managed-bean-class>com.mycompany.newsservice.models.Subscriber</managed-bean-class>
<managed-bean-scope>session</managed-bean-scope>
</managed-bean>
<managed-bean>
<managed-bean-name>subscrHandler</managed-bean-name>
<managed-bean-class>com.mycompany.newsservice.handlers.SubscriberHandler</managed-bean-class>
<managed-bean-scope>request</managed-bean-scope>
<managed-property>
<property-name>subscriber</property-name>
<value>#{subscr}</value>
</managed-property>
</managed-bean>
```

UPPSALA UNIVERSITET

Navigation rules

- We can move between views when all steps have succeeded
- Action methods can also return a string, which is considered the “outcome”
  - They can also return a page name directly (identified by file suffix)
  - Null means “stay in same view”
- `faces-config.xml` contains navigation rules for how to treat outcomes

UPPSALA UNIVERSITET

navigation-rule example

```
<navigation-rule>
  <from-view-id>*</from-view-id>
  <navigation-case>
    <from-outcome>logout</from-outcome>
    <to-view-id>logout.xhtml</to-view-id>
  </navigation-case>
</navigation-rule>
```

- With this rule, any action method can return "`logout`" to move to that view
  - We can rearrange the flow in the XML file

UPPSALA UNIVERSITET

Converters and validation

- HTML forms contain strings
  - Many values are of different types
- In the f prefix, tags like convertNumber can be used

```
<h:inputText value="#{someBean.someValue}">
        <f:convertNumber/>
        <f:validateLongRange minimum="10" maximum="100" />
</h:inputText>
```

- JSF has standard facilities for putting conversion and validation messages next to components.
  - Layout be customized
  - Possible to add custom validators and converters

UPPSALA UNIVERSITET

AJAX

- AJAX used to be the buzz term for dynamic content in the client
  - Asynchronous JAvascript and XML
  - Nowadays JSON-based development is more common
- JSF has the full component tree
  - This makes it relatively straightforward to just reprocess part of a view

UPPSALA UNIVERSITET

f:ajax

- Scenario:
  - You have a button performing some action
  - You don’t want the full page to reload
  - You know what parts of the component tree might be affected

```
<h:commandButton value="Add user“ action="#{userBean.doAddUser}">
    <f:ajax execute="@form" render=":userlist"/>
</h:commandButton>
```

- Execute all parts of JSF processing for the current form and then re-render the component with the ID userlist

UPPSALA UNIVERSITET

Ajax shortcuts

- `@all` for executing/rendering everything
- `@none` for nothing (default)
- `@form` for current form
- Multiple IDs can be specified like `":userlist :usercounter"` if there is also a counter of the total number of users

- Might be a convenient way to add some more modern dynamic content in an overall server-based application

UPPSALA UNIVERSITET

Passthrough elements and attributes

- The idea of JSF was to make the user ignore HTML and the client
  - Give the illusion of a development paradigm similar to desktop GUI frameworks
  - Far more convenient for a line-of-business form-based application compared to JSP + servlet
- Hard to integrate the rigid JSF components with a front-end developer wanting to do HTML and client-side scripting
- JSF pass-through options make life easier

UPPSALA UNIVERSITET

Pass-through attributes

- Add extra attributes to the resulting HTML tags
  - I.e. use an h:inputText tag, but use the new HTML5 `placeholder` attribute for giving explanation text and `type` for browser-based validation
  - Map passtrhough namespace `xmlns:pt="http://xmlns.jcp.org/jsf/passthrough"`

```
<h:inputText id="email" value="#{bean.email}"
        pt:type="email" pt:placeholder="Enter email"/>
```

UPPSALA UNIVERSITET

Pass-through elements

- Keep a clean HTML5 design
- Use the native `<form>`, `<input>`, `<button>`, `<a>` tags etc
  - Anyone familiar with HTML can read the source
- Specific elements can still be included in a “mini” component tree by adding attributes from the JSF name space http://xmlns.jcp.org/jsf
  - Attributes like `value`, `action`, `id`
- Benefit from JSF life-cycle while allowing full HTML creativity

UPPSALA UNIVERSITET

Pass-through example

```
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:jsf="http://xmlns.jcp.org/jsf">
<head jsf:id="head"><title>JSF 2.2</title></head>
<body jsf:id="body">
  <form jsf:id="form">
    <input type="text" jsf:id="name"
        placeholder="Enter name"
        jsf:value="#{bean.name}"> <f:validateLength
minimum="3"/> </input>
    <button jsf:action="#{bean.save}">Save</button>
  </form>
</body>
</html>
```

UPPSALA UNIVERSITET

Listen to arbitrary events

- f:ajax can also be tied to arbitrary events
- Scenario
  - A block of text shows a counter, with the count determined by a backing bean
  - User clicks on this block, on the client (no button or anything)
  - Click triggers a server action and partial re-render

UPPSALA UNIVERSITET

JSF listener to div click

```
<div jsf:id="clickCounter">
  Clicks: #{bean.clickCount}
  <f:ajax event="click" render="@this"
      listener="#{bean.inc}"/>
</div>
```

UPPSALA UNIVERSITET

JSF, pros and cons

- Life cycle management can be convenient
  - If your application logic matches the lifecycle
- If you find yourself spending lots of time on validating input, sending messages about errors etc, then it might be good

- Storing the full component tree for any view is cumbersome
- Bookmarking used to be a problem
  - Now somewhat fixed
- Still not always ideal experience if user session times out
- Would not scale well to very high counts of simultaneous users
- Done right, you can “scale out” to several servers

- Many real applications never need to handle tens of thousands of simultaneous users

UPPSALA UNIVERSITET

Q&A