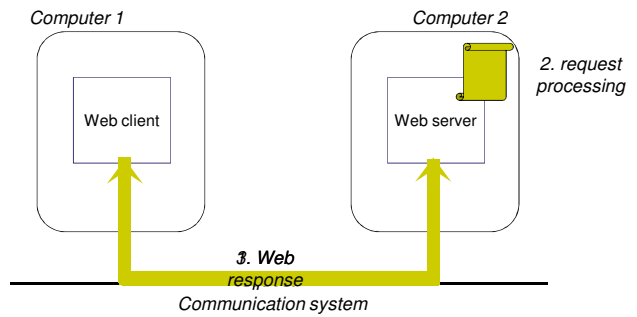
JavaServer Pages for Building Distributed Web Applications

Sara Bouchenak

Sara.Bouchenak@imag.fr
http://membres-liglab.imag.fr/bouchenak/teaching/

Introduction – Web applications



Motivations

- Web response includes
  - A static part:
    - Part of the response that does not change between different requests
    - Usually, static HTML
  - A dynamic part
    - Part of the response that depends on the actual request
    - Usually, processing/program result
- JavaServer Pages (JSP)
  - A technology used to build web applications
  - Allows to build web responses in such a way that the static part is separated from the dynamic part

JavaServer Pages overview

- With JavaServer Pages (JSP)
  - For the static parts of the web response
    - Simply write regular HTML in the normal manner
  - For the dynamic parts of the web response
    - Enclose code for the dynamic parts using special tags
  - Example
    - A JSP page that results in the following "Thanks for ordering *Core Web Programming*"
    - URL http://host/OrderConfirmation.jsp?title=Core+Web+Programming
    - Section of the JSP page:

```
Thanks for ordering <I>
<%= request.getParameter("title") %>
</I>
```

JavaServer Pages overview (2)

- JSP files
  - A JSP file has a *.jsp* extension
  - A JSP file is installed in any place a normal web page could be placed
- JSP – How it works
  - A JSP page often looks more like a regular HTML page
  - A JSP page automatically gets converted to a normal Servlet
  - The static HTML is printed to the output stream associated with the servlet's service method
  - While the dynamic part correponds to Java code

Outline

1. Motivations
2. **The lifecycle of a JSP page**
3. Creating static and dynamic content
4. Example

The lifecycle of a JSP page

- When a web request is mapped to a JSP page, the web container first checks whether the JSP page's servlet exists and whether it is older than the JSP page
- If the servlet does not exists or is older than the JSP page, the web container translates the JSP page into a servlet class and compiles the class
- After the JSP page has been translated and compiled, the JSP page's servlet follows the servlet life cycle
  - If an instance of the JSP page's servlet does not exist, the container
    - Loads the JSP page's servlet class
    - Instantiates an instance of the servlet class
    - Initializes the servlet instance by calling the *jspInit* method
  - The container invokes the *_jspService* method, passing request and response objects.
  - If the container needs to remove the JSP page's servlet, it calls the *jspDestroy* method.
- During development, one of the advantages of JSP pages over servlets is that the build process is performed automatically.

Translation and compilation

- During the translation phase each type of data in a JSP page is treated differently.
  - Static data
    - It is transformed into code that will print the data into the output response stream associated with the servlet's service method
  - Dynamic data
    - Several JSP elements are used to build dynamic reponse:
      - *Scripting elements*
        - Specify Java code that will become part of the resultant servlet class
      - *Directives*
        - Control how the web container translates and executes the JSP page
      - *Actions*
        - Specify existing components that should be used
      - *Predefined variables*
        - Simplify the scripting elements
        - Example: the *request* variable in the previous JSP snippet

Outline

1. Motivations
2. The lifecycle of a JSP page
3. **Creating static and dynamic content**
4. Example

Creating static content in a JSP

- Static content is created in a JSP page simply by writing it as if a static web page that consists only of that content is created
- Static content can be expressed in any text-based format
  - Examples: HTML, WML (Wireless Markup Language), and XML (eXtensible Markup Language)
- The default format is HTML.
- If another format is used, it must be specified at the beginning of your JSP page with a page directive with the *contentType* attribute set to the content type.
  - Example: A JSP page that contains data expressed in WML includes the following directive:
    `<%@ page contentType="text/vnd.wap.wml"%>`
- A registry of content type names is kept by the IANA at : http://www.iana.org/assignments/media-types/
- The static part can also be created by tools for building web pages

Creating dynamic content in a JSP

- In the following, we will detail several JSP elements:
  - *Scripting elements*
    - Specify Java code that will become part of the resultant servlet class
  - *Directives*
    - Control how the web container translates and executes the JSP page
  - *Actions*
    - Specify existing components that should be used
  - *Predefined variables*
    - Simplify the scripting elements
    - Example: the *request* variable in the previous JSP snippet

JSP scripting elements

- JSP scripting elements allow inserting Java code into the servlet that will be generated from the current JSP page
- There are three forms of scripting elements
  - *Expressions* that are evaluated and inserted into the output, they have the form of:
    `<%= expression %>`
  - *Scriptlets* that are inserted into the servlet's service method, they have the form of
    `<% code %>`
  - *Declarations* that are inserted into the body of the servlet class, outside of any existing methods, they have the form
    `<%! code %>`

JSP expressions

- A JSP *expression* is used to insert Java values directly into the output
- An expression has the following form:
  `<%= Java Expression %>`
- The Java expression is evaluated, converted to a string, and inserted in the page
- This evaluation is performed at run-time (when the page is requested), and thus has full access to information about the request
- Example: the following shows the date/time that the page was requested:
  `Current time: <%= new java.util.Date() %>`

JSP expressions and predefined varibales

- To simplify expressions, there are a number of predefined variables that can be use
- Some predefined variables:
  - *request*, the HttpServletRequest;
  - *response*, the HttpServletResponse;
  - *session*, the HttpSession associated with the request (if any); and
  - *out*, the PrintWriter (a buffered version of type JspWriter) used to send output to the client.
- Example
  `Your hostname: <%= request.getRemoteHost() %>`

Outline

1. Motivations
2. The lifecycle of a JSP page
3. **Creating static and dynamic content**
   - Creating static content
   - **Creating dynamic content**
     - **Scripting elements:** Expressions, **Scriptlets**, Declarations
     - Directives
     - Predefined variables
     - Actions
4. Example

JSP scriptlets

- In order to do something more complex than insert a simple expression, JSP *scriptlets* allow inserting arbitrary code into the servlet method that will be built to generate the page
- Scriptlets have the following form:
  `<% Java Code %>`
- Scriptlets have access to the same automatically predefined variables as expressions
- Example:

```
<% String queryData = request.getQueryString();
out.println("Attached GET data: " + queryData); %>
```

JSP scriptlets (2)

- Code inside a scriptlet gets inserted *exactly* as written
- Any static HTML (template text) before or after a scriptlet gets converted to print statements.
- Scriptlets need not contain complete Java statements, and blocks left open can affect the static HTML outside of the scriptlets.
- Example: a JSP fragment with mixed template text and scriptlets

```
<% if (Math.random() < 0.5) { %>
Have a <B>nice</B> day!
<% } else { %>
Have a <B>lousy</B> day!
<% } %>
```

will get converted to something like:

```
if (Math.random() < 0.5) {
     out.println("Have a <B>nice</B> day!");
} else {
     out.println("Have a <B>lousy</B> day!");
}
```


JSP declarations

- A JSP *declaration* allows defining methods or fields that get inserted into the main body of the servlet class (outside of the service method processing the request)
- A JSP declaration has the following form:
  `<%! Java Code %>`
- Since declarations do not generate any output, they are normally used in conjunction with JSP expressions or scriptlets
- Example: a JSP fragment that prints out the number of times the current page has been requested since the server booted (or the servlet class was changed and reloaded):

```
<%! private int accessCount = 0; %>
Accesses to page since server reboot:
<%= ++accessCount %>
```


Outline

1. Motivations
2. The lifecycle of a JSP page
3. **Creating static and dynamic content**
   - Creating static content
   - **Creating dynamic content**
     - Scripting elements
     - **Directives**
     - Predefined variables
     - Actions
4. Example

JSP directives

- A JSP *directive* affects the overall structure of the servlet class
- A directive has usually the following form:
  `<%@ directive attribute="value" %>`
- However, you can also combine multiple attribute settings for a single directive, as follows:

```
<%@ directive    attribute1="value1"
                 attribute2="value2"
                 ...
                 attributeN="valueN" %>
```

- There are three main types of directives:
  - *include* directive
  - *page* directive
  - *taglib* directive

JSP include directive

- The *include* directive is used to insert the text contained in another file into the including JSP document
- The inserted text is either static content or another JSP page
- This directive can placed anywhere in the JSP page
- Its syntax is:
  `<%@ include file="relativeURLspec" %>`
- The URL specified is normally interpreted relative to the JSP page that refers to it, but the URL can be interpreted relative to the home directory of the Web server by starting the URL with a forward slash
- The HTML/XML/WML view of a JSP document does not contain *include* directives, rather the included file is expanded in place; this is done to simplify validation

JSP page directive

- The *page* directive defines a number of page-dependent properties and communicates these to the JSP container
- This directive has the following syntax:
  `<%@ page page_directive_attr_list %>`
- Example: a page directive that tells the JSP container to load an error page when it throws an exception

```
...
<%@ page errorPage="errorpage.jsp" %>
...
```

If there is an error when the JSP page is requested, the error page is accessed

JSP page directive examples

- import
  - import="*package.class*" or import="*package.class1,...,package.classN*". Specify what packages should be imported
  - Example
    `<%@ page import="java.util.*" %>`
  - The import attribute is allowed to appear multiple times
- extends
  - extends="*package.class*"
  - Indicate the superclass of servlet that will be generated.
  - To use this with extreme caution, since the server may be using a custom superclass already
- session
  - session="true|false"
  - A value of true (the default) indicates that the predefined variable session (of type HttpSession) should be bound to the existing session if one exists, otherwise a new session should be created and bound to it.
  - A value of false indicates that no sessions will be used, and attempts to access the variable session will result in errors at the time the JSP page is translated into a servlet

JSP taglib directive

- Custom tags are user-defined JSP language elements that encapsulate recurring tasks.
- Custom tags are distributed in a *tag library*, which defines a set of related custom tags and contains the objects that implement the tags
- Custom tags have the syntax:
  `<prefix:tag attr1="value" ... attrN="value" />`
  where *prefix* distinguishes tags for a library,
  tag is the tag identifier, and
  attr1 ... attrN are attributes that modify the behavior of the tag
- To use a custom tag in a JSP page:
  - Declare the tag library containing the tag as follows
    `<%@ taglib prefix="tt" [tagdir=/WEB-INF/tags/dir | uri=URI ] %>`
  - Make the tag library implementation available to the web application

Outline

1. Motivations
2. The lifecycle of a JSP page
3. **Creating static and dynamic content**
   - Creating static content
   - **Creating dynamic content**
     - Scripting elements
     - Directives
     - **Predefined variables**
     - Actions
4. Example

JSP predefined variables

- To simplify code in JSP expressions and scriptlets, several automatically defined variables, sometimes called *implicit objects*, are provided
- Examples
  - *request*, this is the *HttpServletRequest* associated with the request, and lets you look at the request parameters, the request type, and the incoming HTTP headers, etc.
  - *response*, this is the *HttpServletResponse* associated with the response to the client
  - *session*, this is the HttpSession object associated with the request
  - *servletContext*, the context for the JSP page's servlet and any web components contained in the same application

Outline

1. Motivations
2. The lifecycle of a JSP page
3. **Creating static and dynamic content**
   - Creating static content
   - **Creating dynamic content**
     - Scripting elements
     - Directives
     - Predefined variables
     - **Actions**
4. Example

JSP actions

- JSP *actions* use constructs in XML syntax to control the behavior of the servlet engine
- Available actions include:
  - *jsp:include* - Include a file at the time the page is requested
  - *jsp:forward* - Forward the requester to a new page
  - *jsp:useBean* - Find or instantiate a JavaBean
  - *jsp:setProperty* - Set the property of a JavaBean
  - *jsp:getProperty* - Insert the property of a JavaBean into the output

JSP include action

- The *include* action inserts files into the JSP page being generated
- The files are inserted at the time the JSP page is requested
- The syntax looks like this:
  `<jsp:include page="relative URL" flush="true" />`
- Unlike the *include directive*, which inserts the file at the time the JSP page is translated into a servlet, the *include action* inserts the file at the time the page is requested

JSP forward action

- The *forward* action lets you forward the request to another page
- This action has a single attribute, *page*, which should consist of a relative URL
- This could be a static value, or could be computed at request time
- Example 1: the target page is a static value
  `<jsp:forward page="/utils/errorReporter.jsp" />`
- Example 2: the target page is computed at request time
  `<jsp:forward page="<%= someJavaExpression %>" />`

JSP useBean, setProperty and getProperty action

- The *useBean* action loads in a *JavaBean* to be used in the JSP page
- This is a a very useful capability to exploit the reusability of Java classes without sacrificing the convenience that JSP adds over servlets alone
- The simplest syntax for specifying that a bean should be used is:
  `<jsp:useBean id="name" class="package.class" />`
- This usually means "instantiate an object of the class specified by *class*, and bind it to a variable with the name specified by *id*."
- Modify JavaBean properties via the *jsp:setProperty* action
- Read existing JavaBean properties via the jsp:getProperty action

Comments

- JSP comments
  - `<%-- comment --%>`
  - Ignored by JSP-to-scriptlet translator
  - Any embedded JSP scripting elements, directives, or actions are ignored
- HTML comments
  - `<!-- comment -->`
  - Passed through to resultant HTML
  - Any embedded JSP scripting elements, directives, or actions are executed normally

Outline

1. Motivations
2. The lifecycle of a JSP page
3. Creating static and dynamic content
   - Creating static content
   - Creating dynamic content
     - Scripting elements
     - Directives
     - Predefined variables
     - Actions
4. **Example**

Example – index.jsp

```
<%--
  * A simple JSP page example.
--%>

<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<%@ taglib uri="http://java.sun.com/jsp/jstl/functions" prefix="fn" %>

<html>
    <head>
        <title>Hello</title>
    </head>
<body bgcolor="white">
    <img src="duke.waving.gif">
    <h2>Hello, my name is Duke. What's yours?</h2>
    <form method="get">
        <input type="text" name="username" size="25">
    <p></p>
        <input type="submit" value="Submit">
        <input type="reset" value="Reset">
    </form>

    <c:if test="${fn:length(param.username) > 0}" >
        <%@include file="response.jsp" %>
    </c:if>

</body>
</html>
```


Example – response.jsp

```
<%--
  * A simple JSP page example.
--%>


<h2><font color="black">
    Hello,
    ${param.username}
    !</font>
</h2>
```


Outline

1. Motivations
2. The lifecycle of a JSP page
3. Creating static and dynamic content
4. Example

References

This lecture is extensively based on:

- H. Bergsten. *JavaServer Pages*. O'Reilly, 2003.
- M. Hall. *Servlets and Java ServerPages: A Tutorial*. http://www.apl.jhu.edu/~hall/java/Servlet-Tutorial/
- Sun Microsystem. *The J2EE Tutorial*. http://java.sun.com/j2ee/1.4/docs/tutorial/doc/