IBD – Intergiciels et Bases de Données

JavaServer Pages for building distributed web applications

Fabien Gaud, Fabien.Gaud@inrialpes.fr

http://www-ufrima.imag.fr/ ⇨ Placard électronique ⇨ M1 Info ⇨ IBD

Overview of lectures and practical work

- Lectures
  - Introduction to distributed systems and middleware
  - Socket-based distributed systems
  - RMI-based distributed systems
  - Servlet-based distributed systems
  - **JavaServer Pages for building distributed web applications**
  - Introduction to multi-tier distributed Internet services

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

```
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 correspond to Java code

Outline

- Motivations
- **The lifecycle of a JSP page**
- Creating static and dynamic content
- 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

The lifecycle of a JSP page (2)

- 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
      - Directives
      - Actions
      - Predefined variables

Outline

- Motivations
- The lifecycle of a JSP page
- Creating static and dynamic content
  - **Creating static content**
  - Creating dynamic content
    - Scripting elements
    - Predefined variables
    - Directives
    - Actions
  - Errors
- 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.
  - Example: A JSP page that contains data expressed in WML includes the following directive:
    <%@ page contentType="text/vnd.wap.wml"%>
- The static part can be created by tools for building web pages

Outline

- Motivations
- The lifecycle of a JSP page
- Creating static and dynamic content
  - Creating static content
  - Creating dynamic content
    - **Scripting elements**
    - Predefined variables
    - Directives
    - Actions
  - Errors
- Example

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
  - Declarations that are inserted into the body of the servlet class, outside of any existing methods, they have the form
    `<%! code %>`
  - Scriptlets that are inserted into the servlet's service method, they have the form of
    `<% code %>`
  - Expressions that are evaluated and inserted into the output, they have the form of:
    `<%= expression %>`

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)
  - Persistent among requests
- 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 from the start:
  `<%! private int accessCount = 0; %>`

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 help 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);
accessCount++; %>
```


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.

JSP scriptlets

- Example

```
<% 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 expressions

- JSP expressions are used to insert 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, and thus has full access to information about the request
- Example

```
Current time: <%= new java.util.Date() %>
Accesses to page since server reboot:
<%= accessCount %>
```


Outline

- Motivations
- The lifecycle of a JSP page
- Creating static and dynamic content
  - Creating static content
  - Creating dynamic content
    - Scripting elements
    - **Predefined variables**
    - Directives
    - Actions
  - Errors
- 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,
  - response, this is the HttpServletResponse associated with the request
  - session, this is the HttpSession object associated with the request
  - application,the context for the JSP page's servlet and any web components contained in the same application
  - page, alternative to *this*
  - ...

Outline

- Motivations
- The lifecycle of a JSP page
- Creating static and dynamic content
  - Creating static content
  - Creating dynamic content
    - Scripting elements
    - Predefined variables
    - **Directives**
    - Actions
  - Errors
- 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
- Processed when the page is translated

JSP include directive (2)

- Example

```
<html>
      <%@ include file="banner.jsp" %>

      Here we have some work

      <%@ include file ="catalog.html" %>
</html>
```

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.class1,...,package.classN".
    - Specify what packages should be imported
  - Example
    ```
    <%@ page import="java.util.*" %>
    ```
  - The import attribute is allowed to appear multiple times
- contentType
  - Specify the content-type of the page returned
  - Example:
    ```
    <%@ page contentType="text/vnd.wap.wml"%>
    ```


JSP page directive examples (2)

- session
  - session="true|false"
  - If true (the default), 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
- IsThreadSafe
  - session="true|false"
  - If true, the JSP can be accessed concurrently (default = true)
- ...

JSP taglib directive

- Custom tags are user-defined JSP **actions** that help in 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" />
  ```
- Composed of a tag library description associated with a Java implementation

JSP taglib directive (2)

- To use a custom tag in a JSP page:
  - Declare the tag library containing the tag as follows
    <%@ taglib prefix="mytl" uri=”/WEB-INF/mytaglib.tld” %>
- Example

```
<html>
  <body>
    <mytl:func param1=”value”/>
  </body>
</html>
```

- JSP Standard Tag Library (JSTL)
  - A common tag library helping for common tasks (ex. XML processing, SQL queries, ...)

JSP taglib directive (3)

```
<%--
  * 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>
```


Outline

- Motivations
- The lifecycle of a JSP page
- Creating static and dynamic content
  - Creating static content
  - Creating dynamic content
    - Scripting elements
    - Predefined variables
    - Directives
    - **Actions**
- 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" />
- Flush value means flushing prior data before including JSP content if the page output is buffered (since JSP 1.2).
- 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 %>" />
```


JavaBeans

- Classic Java class
- Follow recommendations
  - The class must be public
  - The class must provide a no-args public constructor
  - The class must be serializable
  - Attributes are accessible through well-known methods
    - <typeOfX> getX()
    - setX(<typeOfX> X)

JSP and JavaBeans

- The useBean action loads in a JavaBean to be used in the JSP page
- The simplest syntax for specifying that a bean should be used is:

```
<jsp:useBean id="name" class="package.class" />
```

  - Can also specify a scope
    - application
    - page
    - session
    - request
  - Return the bean if exiting, creating it otherwise

JSP and JavaBeans (2)

- Example

```
<jsp:useBean id="c" scope="page" class="Counter">
</jsp:useBean>
<HTML>
   <HEAD>
      <TITLE>Example</TITLE>
   </HEAD>

   <BODY>
   <H3>
      <% c.increase(); %>
      Accessing page <%= c.getCounter(); %> times
   </H3>
</HTML>
```


JSP and JavaBeans (3)

- Can access to a bean property
  - getProperty, return the value of the specified property
  - setProperty, set the value of the specified property
- Example
  - <jsp:getProperty name="c" property="counter">
  - <jsp:setProperty name="c" property="counter" value="1">
  - Are equivalent to
    - <%= c.getCounter() >
    - <% c.setCounter(1) >

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

- Motivations
- The lifecycle of a JSP page
- Creating static and dynamic content
  - Creating static content
  - Creating dynamic content
    - Scripting elements
    - Directives
    - Predefined variables
    - Actions
  - **Errors**
- Example

Errors

- Can be thrown due to
  - Syntax error in the jsp code
  - Java execution error (ex: NullPointerException)
- By default exception are sent to the client
  - Using a default page
  - Could be customize through *errorPage* and *isErrorPage* directives

Errors

error.jsp

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

[...]
```

```
<%@ page isErrorPage="true" %>
<html>
<head>
   <title>Error Page</title>
</head>

<body>
<h2>Your application has
generated an error</h2>
<b>Exception:</b>
<br />
<%= exception.toString() %>
</body>
</html>
```


Outline

- Motivations
- The lifecycle of a JSP page
- Creating static and dynamic content
- **Example**

A simple example

```
<%! private int counter = 0; %>
<HTML>
        <HEAD><TITLE>Hello</TITLE></HEAD>
                <BODY>
                <H1> Hello
                        <%      counter++;
                                String pname;
                                pname = request.getParameter("name");
                                if (pname== null) {
                                        out.println("World");
                                }
                                else {
                        %>

                        Mister <%=pname%>

                        <% } // fin du else %>
                </H1>
        </BODY>
</HTML>
```

©D.Donsez

A simple example (2)

```
public class myJSPName extends HttpJspBase {
  private int counter = 0;

  public myJSPName( ) {}

  private static boolean _jspx_inited = false;

  public final void _jspx_init() throws JspException {}

  public void _jspService(HttpServletRequest request, HttpServletResponse
response) throws java.io.IOException, ServletException {

        JspFactory _jspxFactory = null;
        PageContext pageContext = null;
        HttpSession session = null;
        ServletContext application = null;
        ServletConfig config = null;
        JspWriter out = null;
        Object page = this;
        String  _value = null;

//...
```


A simple example (3)

```
// ...

        try {

            if (_jspx_inited == false) {
                synchronized (this) {
                    if (_jspx_inited == false) {
                        _jspx_init();
                        _jspx_inited = true;
                    }
                }
            }
            _jspxFactory = JspFactory.getDefaultFactory();
            response.setContentType("text/html");
            pageContext = _jspxFactory.getPageContext(this, request, response,
                        "", true, 8192, true);

            application = pageContext.getServletContext();
            config = pageContext.getServletConfig();
            session = pageContext.getSession();
            out = pageContext.getOut();

// ...
```


A simple example (4)

```
// ...
                application = pageContext.getServletContext();
                config = pageContext.getServletConfig();
                session = pageContext.getSession();
                out = pageContext.getOut();

                out.write("<HTML> ... ");

                String pname;
                pname = request.getParameter("name");
                if (pname== null) { out.println("World"); }
                else {
                        out.write("Mister");
                        out.println(pname);
                }

                out.write("</H1> ... ");

                } catch (Throwable t) {
                   ...
                }
        }
   }
}
```


Packaging

- Goal: distributing of web applications
- Package must contains all application needs
  - Libraries, resources, ...
- All contained in a .war
  - A jar with a special organization
  - Use of WEB-INF/web.xml to specify parameters
    - Example: Mapping URL <-> Servlet
- Example: Use with Tomcat
  - Put war in webapps/
  - Start (or restart) the server

Incoming lectures and practical work on middleware

- Lectures
  - Introduction to distributed systems and middleware
  - Socket-based distributed systems
  - RMI-based distributed systems
  - Servlet-based distributed systems
  - Introduction to multi-tier distributed Internet services
- Practical work
  - Programming distributed systems with Sockets
  - Programming distributed systems with RMI
  - Programming distributed systems with Servlets
  - Project on multi-tier Internet services

References

- This lecture is built from:
  - 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/
  - Courses from D. Donsez http://membres-liglab.imag.fr/donsez/
  - Courses from Lionel Seinturier http://www2.lifl.fr/~seinturi/
  - http://course.cs.ust.hk/comp201/2007summer/web/slides/ch35.ppt
- is mostly based on lectures given by Sara Bouchenak,
  - http://sardes.inrialpes.fr/~bouchena/