CmpE 473
Internet Programming

Pınar Yolum
pinar.yolum@boun.edu.tr

Department of
Computer Engineering
Boğaziçi University

Servlets and JavaServer Pages

Examples from java.sun.com

Web pages

- Content resides on a server
- Viewed by client browsers
- Static Web pages
  - Always the same content
  - HTML
- Dynamic Web pages
  - Content changes based on information from the client (e.g., authentication)
  - CGI, Servlet, JSP

Common Gateway Interface

- Typically client sends data to the server through a form
- Sending data to the server triggers a new process (c program, perl script, and so on)
- Same process cannot be used for more than one request

Common Gateway Interface

- Two common methods to send data
  - Get
    - Data passed as part of the URL
    - URL?NAME=VALUE&NAME=VALUE
    - CGI picks the NAME=VALUE pairs in the QUERY_STRING environment variable
    - <FORM ACTION="/cgi-bin/color.cgi" METHOD="GET">
    - Replace white space with +
    - Replace special characters with %hexadecimal ASCII
    - Might have restrictions on the number of bytes appended
  - Post
    - Server receives POST and keeps listening
    - Receive the data through STDIN
    - CONTENT_LENGTH environment variable is checked to determine how much to read

Servlets (1)

- All servers implement javax.servlet.Servlet interface
- Contains five methods
  - `public void init(ServletConfig config) throws ServletException`
  - `public void service(ServletRequest request, ServletResponse response) throws ServletException, java.io.IOException`
  - `public void destroy()`
  - `public ServletConfig getServletConfig()`
  - `public java.lang.String getServletInfo()`

Servlets (2)

- `init`, `service`, and `destroy` manage the lifecycle of a servlet
- `init`: Puts the servlet into service
  - Load a database driver
  - Initialize values
- `service`: Called by the servlet container
  - `ServletRequest`: Client's request
  - `ServletResponse`: Server's answer
- `destroy`: Called after the service is over
  - Release memory, threads, unload drivers

Servlets (3)

- `GenericServlet` (abstract class)
  - Independent of protocol
  - Only override abstract `service` method
- `HttpServlet` (abstract class)
  - `service:` Receives standard HTTP requests from the public service method and dispatches them to the do*XXX* methods defined in this class.
  - Provide methods that are called from the service method (with HttpServletRequest and HttpServletResponse)
    - `doGet`, if the servlet supports HTTP GET requests
    - `doPost`, for HTTP POST requests
    - `doPut`, for HTTP PUT requests

Servlets (4)

- Override `doGet`(or `doPut`)
  - Get request data
  - Write response headers
  - Get response output stream (or writer)
  - Write the response data
  - Set encoding
  - Set content type
- Get method should be safe
  - Should not change data on the server
- Repeatable
  - Performing it again should yield the same results

Example (1)

- In the I'm a simple form page

```
<FORM METHOD="POST" ACTION="guestBook">

    <INPUT TYPE="TEXT" NAME="DATA" SIZE=30>

    <P>
    <INPUT TYPE="SUBMIT" VALUE="Click Me">
    <INPUT TYPE="RESET">
    </FORM>
```

- A text box to enter a word with Click Me and Reset buttons.
- The server is going to print the word in uppercase.
- Demo

Example (2)

```
package guestbook;
import java.io.IOException;
import java.io.*;
import javax.servlet.http.*;
public class GuestbookServlet extends HttpServlet {
        public void doPost(HttpServletRequest req, HttpServletResponse resp)
                        throws IOException {
                resp.setContentType("text/html");
                PrintWriter out = resp.getWriter();
                out.println("<title>Example</title>" + "<body bgcolor=FFFFFF>");
                out.println("<h2>Button Clicked</h2>");
                String DATA = req.getParameter("DATA");
                if(DATA != null) { out.println("<h2> Look uppercase! " +
DATA.toUpperCase() + "</h2>"); }
                        else { out.println("No text entered."); }
        out.close(); }}
```

JSP

- For creating dynamic Web pages
- Platform independent
  - Any browser can view JSP pages
  - Any java-compliant server can process them
- HTML page + application logic
  - Separate user interface and logic
  - Application logic
    - JavaBeans
    - JDBC objects
    - EJBs
    - RMI objects
  - Change presentation without changing logic (No compilation by the developer)

JSP vs. Servlet

- Servlets
  - Embed HTML code and application code into Java classes
  - Compile the code if you modify the HTML part
  - Require expertise in Java even if the modification is in HTML
- JSP
  - Extends Servlets
  - Compiled dynamically into servlets before usage

JSP Pages

- Resemble XML pages (code inside tags)
- Processed by the Web server to generate content
- JSP Tags can define
  - Call to a JavaBeans getMethod
  - Include Java code (called *scriptlets*)
- HTML tags are standard

Example JSP File

```
<HTML>
<%@ page language="java" import="java.util.*" %>
<H1>Welcome</H1>
<P>Today is </P>
<jsp:useBean id="clock" class="GregorianCalendar" />
<UL>
<LI>Day: <%= clock.get(Calendar.DAY_OF_MONTH) %>
<LI>Year: <%= clock.get(Calendar.YEAR) %>
</UL>
<%-- Check for AM or PM --%>
<%! int time = Calendar.getInstance().get(Calendar.AM_PM); %>
<%
if (time == Calendar.AM) {
%>
Good Morning
<%
}
else {
%>
Good Afternoon
<%
}
%>
<%@ include file="copyright.html" %>
</HTML>
```

Example Output

Welcome

Today is

Day: 4
Year: 2012
Good Afternoon
PINAR YOLUM

JSP Components

- JSP actions (or tags)
- Directives
- Declarations
- Expressions
- Scriptlets
- Comments

JSP Actions

- XML-like syntax
- Manage JavaBeans components

<jsp:useBean id=="clock" class="GregorianCalendar" />

- Getproperty and Setproperty methods
  - <jsp:getProperty name="bean" property="property" />
    <jsp:setProperty name="bean" property="property"
    value="value" />
  - <h2>
    Clock of **<jsp:getProperty name="clock"**
    property="username" />
    </h2>

Directives

- Instructions for JSP engine
- Contain meta-information about the page
  - Specify custom tag libraries
  - Insert external files
- Exist between **<%@** and **%>** tags
- Example:

```
<%@ page language="java" import="java.util.*" %>
<%@ include file="copyright.html" %>
```

Declarations

- Variable declarations for use in expressions and in scriptlets
- Exist between <%! and %>
- Example

```
<%! int time =
   Calendar.getInstance().get(Calendar.AM_PM); %>
```

Expressions

- Variables or constants returned by the Web server
- Exist between <%= and %>
- Example: Making calls to a JavaBean

<%= clock.get(Calendar.DAY_OF_MONTH) %>

<%= clock.get(Calendar.YEAR) %>

Scriptlets

- Block of Java code
- Inserted directly into the generated servlet
- Exist between <% and %> tags

```
<%
if (time == Calendar.AM) {
%>
Good Morning
<%
}
else {
%>
Good Afternoon
<%
}
%>
```

Comments

- Similar to HTML comments
- Not processed by JSP engine
- Exist between <%-- and --%> tags
- Example:
  - <%-- Check for AM or PM --%>

Custom Tags

- Alternative to inserting scriptlets
- Define new tags
  - Move the code from the JSP page to another place
  - Link from the JSP page to the other page through the custom tag
  - Simpler JSP; no need to declare variables, import java libraries, and so on
  - Need to ensure linking is done right

Example JSP Page

- <HTML>
  <%@ taglib uri="/tlds/menuDB.tld" prefix="menu" %>

  <H1>Today's Menu</H1>

  <P>Lunch</P>
  <%@ include file="lunch_menu**.html" %>**

  <P>Our Special of the Day</P>
  <menu: insertCatchOfDay meal="lunch" >

  </HTML>

Components of a Custom Tag

- JSP page that contains the custom tag
  - Must specify the *taglib* directive to provide the location of the tag library descriptor
- Tag library descriptor
  - XML file that defines the custom tag
  - Includes the attributes of the tag
    - Name and location of the handler class
    - Any other information needed to process the tag
- Tag handler
  - Java class that executes the operations of the tag
  - Ex: The class for *insertCatchOfDay* pulls the menu item from the DB

Tag Handler

- Java class that implements a *Tag* interface

(javax.servlet.jsp.Tag)

- Executed when a custom tag is processed by a JSP engine
- Must implement
  - public int doStartTag()
- Must define attributes defined for the tag and its get/set methods

Example Tag Handler

- Must define

```
private String meal = null;

public void setMeal(String s) {
meal = s;
}
public String getMeal() {
return meal;
}
```

Example Tag Library Descriptor

```
<? xml version="1.0" ?>
<taglib>
<!-- May also have tlibversion, shortname, uri -->
<tag>
<name>insertCatchOfDay</name>
<tagclass>com.sun.CatchOfDayHandler</tagclass>
<info>
Queries menu database for the catch of the day.
</info>

<attribute>
<name>meal</name>
</attribute>
</tag>

</taglib>
```

Web Archive (WAR)

- *.html, *.jsp --- for browsers
- /WEB-INF/web.xml: Web application deployment descriptor
- /WEB-INF/classes/: For running the application
- /WEB-INF/lib/: JAR files
- /META-INF/MANIFEST.MF: