

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

HTTP GET/POST

- 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

Fall 2007— Pinar Yolum

5

Execution

- Flow
 - HTML form passes information to the servlet
 - The servlet
 - processes data
 - returns an HTML file
 - The client presents the HTML file
- Container
 - Web server
 - Tomcat, JRun

Fall 2007— Pinar Yolum

6

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()`

Fall 2007— Pinar Yolum

7

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

Fall 2007— Pinar Yolum

8

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 `doXXX` 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

Fall 2007— Pinar Yolum

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

Fall 2007— Pinar Yolum

10

Example (doGet)

- Get order from an HTML page
 - Method for sending data is GET
- ```
<html>
<head><title>Get Order</title></head>
<body>
<h3>Enter an order</h3>
<form action="http://localhost:8100/servlet/GetOrder" method=GET>
 Order: <input type=text name=Order size=20>
</form></body></html>
```
- When form is submitted the URL looks like:  
`http://localhost:8100/servlet/GetOrder?Order="Food%20and%20Drink"`

Fall 2007— Pinar Yolum

11

## Example (doGet)

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class GetOrder extends HttpServlet {
 public void doGet(HttpServletRequest req,
 HttpServletResponse resp)
 throws ServletException, IOException {
 resp.setContentType("text/html");
 PrintWriter out = resp.getWriter();
 String message = req.getParameter("Order");
 out.println("<html>");
 out.println("<head><title>Get Order Servlet</title></head>");
 out.println("<body><h1>I'd like to order
");
 out.println(message+"</h1></body>");
 out.println("</html>");
 out.close();
 }
}
```

Fall 2007— Pinar Yolum

12

## Example (doPost)

**I'm a Simple Form**  
Enter some text and click the Submit button.  
Clicking Submit invokes [ExampServlet.java](#),  
which returns an HTML page to the browser.

  
 

### Button Clicked

Four score and seven years ago

Return to [Form](#)

- In the I'm a simple form page  
<FORM METHOD="POST" ACTION="/servlet/ExampServlet">  
  <INPUT TYPE="TEXT" NAME="DATA" SIZE=30>  
  <P>  
  <INPUT TYPE="SUBMIT" VALUE="Click Me">  
  <INPUT TYPE="RESET">  
</FORM>

Fall 2007— Pinar Yolum

13

## Example (doPost)

```
import java.io.*; import javax.servlet.*; import javax.servlet.http.*;
public class ExampServlet extends HttpServlet {
 public void doPost(HttpServletRequest request,
 HttpServletResponse response)
 throws ServletException, IOException {
 response.setContentType("text/html");
 PrintWriter out = response.getWriter();
 out.println("<title>Example</title>" + "<body bgcolor=FFFFFF>");
 out.println("<h2>Button Clicked</h2>");

 String DATA = request.getParameter("DATA");
 if(DATA != null) { out.println(DATA); }
 else { out.println("No text entered."); }
 }
 out.println("<P>Return to Form");
 out.close();
}
```

Fall 2007— Pinar Yolum

14

## Session Tracking (1)

- HTTP is connectionless (cannot maintain session)
- Session necessary to track history
  - E-commerce
- HttpSession object
  - setAttribute: To store values
  - getAttribute: To retrieve values

Fall 2007— Pinar Yolum

15

## Session Tracking (2)

```
public class SessionOrder extends HttpServlet {
 public void doGet(HttpServletRequest req, HttpServletResponse resp)
 throws ServletException, IOException {
 resp.setContentType("text/html");
 PrintWriter out = resp.getWriter();
 HttpSession session = req.getSession(true);
 String[] newItems = req.getParameterValues("Order");
 Vector items = (Vector)session.getAttribute("items");
 if(items==null)
 items = new Vector();
 for(int i=0; i<newItems.length; i++)
 items.addElement(newItems[i]);
 session.setAttribute("items",items);
 }
}
```

Fall 2007— Pinar Yolum

16

## Session Tracking (3)

```
out.println("<html>");
out.println("<head><title>Session Order Servlet</title></head>");
out.println("<body><h1>Your current order is
");
out.println(items + "</h1><p>");
out.println("Your session ID is:" + session.getId() + "
");
out.println("</body></html>");
out.close();
}
}
```

Fall 2007— Pinar Yolum

17

## 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)

Fall 2007— Pinar Yolum

18

## 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

Fall 2007— Pinar Yolum

19

## 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

Fall 2007— Pinar Yolum

20

## Example

```
<HTML>
<%@ page language="java" imports="java.util.*" %>
<H1>Welcome</H1>
<P>Today is </P>
<jsp:useBean id="clock" class="jspCalendar" />

Day: <%= clock.getDayOfMonth() %>
Year: <%= clock.getYear() %>

<!-- 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>
```

Fall 2007— Pinar Yolum

21

## JSP Components

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

Fall 2007— Pinar Yolum

22

## JSP Actions

- XML-like syntax
- Manage JavaBeans components
  - <jsp:useBean id=="clock" class=="jspCalendar" />
- 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>

F

## 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" imports=="java.util.*" %>
<%@ include file=="copyright.html" %>
```

Fall 2007— Pinar Yolum

24

## 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.getDayOfMonth() %>`
  - `<%= clock.getYear() %>`

## 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

Fall 2007— Pinar Yolum

29

## Example JSP Page

```
<html>
<head>
 <title>Using the hello tag</title>
 <%@ taglib uri="myExamples" prefix="examples" %>
</head>
<body>
 <h1><examples:hello /></h1>
</body>
</html>
```

Fall 2007— Pinar Yolum

30

## 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

Fall 2007— Pinar Yolum

31

## Example Tag Library Descriptor

```
<?xml version="1.0" ?>
<taglib>
 <shortname>examples</shortname>
 <tag>
 <name>hello</name>
 <tagclass>HelloTag</tagclass>
 </tag>
 <tag>
 <name>helloName</name>
 <tagclass>HelloNameTag</tagclass>
 <attribute>
 <name>name</name>
 <required>true</required>
 </attribute>
 </tag>
```

Fall 2007— Pinar Yolum

32

## 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

Fall 2007— Pinar Yolum

∞

## Example Tag Handler

```
import java.io.*;
import javax.servlet.jsp.*;
import javax.servlet.jsp.tagext.*;

public class HelloTag extends TagSupport {
 public int doStartTag() {
 JspWriter out = pageContext.getOut();
 try {
 out.write("Hello World!");
 } catch (IOException e) {
 System.out.println(e.getMessage());
 }
 return SKIP_BODY;
 }
}
```

## More Examples

```
<%@ page language=="java" info="Example JSP #1" %>
<html>
<body>
<%! String agent; %>
<%
agent = request.getHeader("User-Agent");
if (agent.startsWith("Mozilla/4.0") {
%>
<%-- Return content for 4.0 browsers --%>
<%@ include file="ver4.html" %>
<%
}
else if (agent.startsWith("Mozilla/3.0") {
%>
<%-- Return content for 3.0 browsers --%>
<%@ include file="ver3.html" %>
<%
}
else {
%>
<%-- Return content for other/unknown browsers --%>
<%@ include file="other.html" %>
<%
}
%>
</body>
</html>
```

request and response  
(`HttpServletResponse`)  
objects are always  
accessible.

Fall 2007— Pinar Yolum

35

## Usage of JSP



- Replace CGI bin with JSP
- Simple
- Follows C/S logic

Fall 2007— Pinar Yolum

∞

## Multi-Tier Application



- Separate Web and business tiers
- Scalable
  - EJBs can manage DB access for multiple users
- Transaction support
- Built-in security