

CmpE 473 Internet Programming

Pinar Yolum
pyolum@cmpe.boun.edu.tr

Department of
Computer Engineering
Boğaziçi University

Spring 2005—Pinar Yolum

4

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

Spring 2005—Pinar Yolum

4

Chapter 10 JavaServer Pages

Examples from java.sun.com

Spring 2005—Pinar Yolum

5

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

Spring 2005—Pinar Yolum

5

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

Spring 2005—Pinar Yolum

3

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()`

Spring 2005—Pinar Yolum

6

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

Example from Sun (1)

Button Clicked

Four score and seven years ago
Return to [Form](#)

- In the I'm a simple form page

```
<FORM METHOD="POST" ACTION="/servlet/ExamServlet">
  <INPUT TYPE="TEXT" NAME="DATA" SIZE=30>
  <P>
  <INPUT TYPE="SUBMIT" VALUE="Click Me">
  <INPUT TYPE="RESET">
</FORM>
```

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

Example from Sun (2)

```
import java.io.*; import javax.servlet.*; import javax.servlet.http.*;
public class ExamServlet 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 <A HREF=../simpleHTML.html>Form</A>");
        out.close();
    }
}
```

Servlets (4)

- Override `doGet`
 - 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

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)

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 Components

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

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

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>`

Example

```
<HTML>
<%@ page language="java" imports="java.util.*" %>
<H1>Welcome</H1>
<P>Today is <P>
<jsp:useBean id="clock" class="jspCalendar" />
<UL>
<LI>Day: <%= clock.getDayOfMonth() %>
<LI>Year: <%= clock.getYear() %>
</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>
```

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" %>
```

Declarations

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

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

Comments

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

Expressions

- Variables or constants returned by the Web server
- Exist between `<%=` and `%>`
- Example: Making calls to a JavaBean
 - `<%= clock.getDayOfMonth() %>`
 - `<%= clock.getYear() %>`

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

Scriptlets

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

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

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

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>
```

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

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 (*HttpServletRequest*) objects are always accessible.

Example Tag Handler

- Must define

```
private String meal = null;

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

More Examples

```
<html>
<body>
<%@ include file="header.html" %>
<jsp:useBean id="db" class="DbBean" />
<p>Here are your current selections:</p>
<%
selections = request.getParameterValues("items");
if (selections != null) {
%><ul>
<%
for(int x = 0; x < selections.length; x++) {
%><li>
<%= selections[x] %> : <%= db.getInfo(selections[x]) %>
<%
}
%></li></ul>
}
else {
%>
<p>(no items selected)</p>
<%
}
%>
<br>
<%@ include file="footer.html" %>
</body>
</html>
```

C/S Application



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

Spring 2005—Pinar Yolum

31

Multi-Tier Application



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

Spring 2005—Pinar Yolum

32