Web Programming: Java Servlets and JSPs

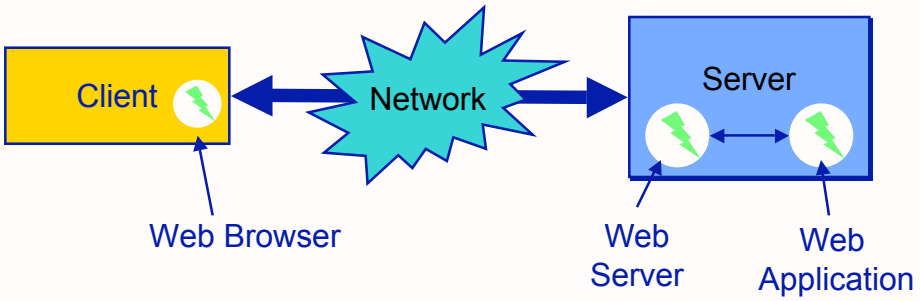
Sara Sprenkle

August 3, 2006

Announcements

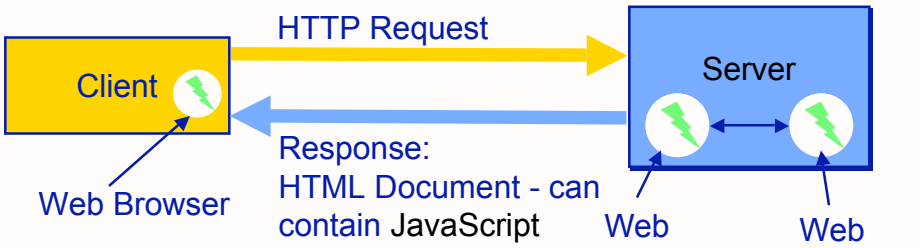
- Assignment 6 due today
- Project 2 due next Wednesday
- Review
  - XML

Web Programming



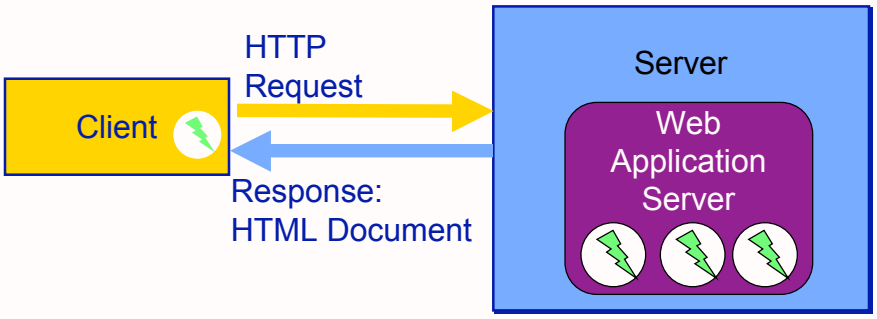
- Specialized network programming
- Web browser: makes requests, renders responses; executes JavaScript, client-side code
- Web Server: handles static requests
- Web Application: handles dynamic requests

Web Programming



- Web Application
  - Parses request, including data
  - Executes request
  - Returns response (often an HTML document)
    - May do other things, like send email, …

Java Web Application Server



- Web Application Server
  - Container to run the web applications
  - Listens on another port besides 80, typically 8080

Servlets

- A servlet is a Java program that extends the functionality of web servers
  - Processing occurs at the server and then the results (typically as an HTML file) are sent back to the client
  - In javax.servlet.* packages
    - Part of J2EE or as a separate download
- Servlets are Java's answer to CGI
- Servlets are supported by many major web servers
  - including Netscape, iPlanet, and Apache (with Tomcat/Jakarta)
  - Specialized web servers: Resin

The **Servlet** Interface

- All servlets implement the **Servlet** interface
  - Many key methods of Servlet interface are invoked automatically
  - Web application server calls methods
    - The program itself does not call

Servlet Interface: Key Methods

- init(ServerConfig config)
  - Called once by the web server to initialize the servlet
- ServletConfig getServletConfig()
  - Returns a reference to a ServletConfig that provides access to the servlet's configuration information and the servlet's ServletContext, which provides access to the server itself
- void service(ServletRequest, ServletResponse)
  - Called to respond to a client request
- String getServletInfo()
  - Returns a String that describes the servlet (name, version, etc.)
- void destroy()
  - Called by the server to terminate a servlet
  - Should close open files, close database connections, etc.

The service() Method

- Called for every client request by application server
  - Generally not overridden
- Method receives both a ServletRequest and a ServletResponse object
  - **ServletRequest** gives the servlet access to input streams and methods to read data from the client
  - **ServletResponse** gives the servlet access to output streams and methods to write data back to the client

The HttpServlet Class

- Web-based servlets (almost all of them) typically extend the **HttpServlet** class
  - HttpServlet overrides the service() method to distinguish between the typical types of requests (HTTP commands/requests)
  - Most common request types are GET and POST
- GET - data encoded in URL
  - Request a resource (file) or retrieve data
- POST - data encoded in body of message
  - Upload data; processing; hide data from URL

The doGet() & doPost() Methods

- HttpServlet defines the doGet() and doPost() methods
  - service() calls the respective method in response to a HTTP GET or POST request
- doGet() and doPost() receive
  - HttpServletRequest
    - From the client
  - HttpServletResponse
    - To the client

The HttpServletRequest Object

- String getParameter(String)
  - Returns the value of a parameter sent to the servlet
- String[] getParameterValues (String)
  - Returns an array of Strings containing the values for a specific servlet parameter
- Enumeration getParameterNames()
  - Returns the names of all of the parameters passed to the servlet

Requests for a digital publication library:

GET dspace/simple-search?search=xxx&sort=date&title=Title

GET dspace/simple-search?search=xxx&sort=name&header=head

The HttpServletRequest Object

- Cookie[] getCookies()
  - Returns an array of Cookie class objects that have been stored on the client by the server
  - Cookies can be used to uniquely identify clients to the server
- HttpSession getSession (boolean create)
  - Returns an HttpSession associated with the client's current browser session
  - Sessions can also be used to uniquely identify clients

The HttpServletResponse Object

- void addCookie(Cookie)
  - Add a Cookie to the header in the response to the client
  - The cookie will be stored on the client, depending on the max-life and if the client allows cookies
- ServletOutputStream getOutputStream()
  - obtains a byte output stream that enables the servlet to send binary data to the client
- PrintWriter getWriter()
  - obtains a text writer that enables the servlet to send character data to the client
- void setContentType(String)
  - Specifies the MIME type of the response to the client so that the browser will know what it received and how to format it
  - "text/html" specifies an HTML document

A Simple Example

- Let's create a very basic servlet
- Create a HTML page with a very basic form, one submit button
- When the button is pressed, browser passes the servlet a basic GET request


```
<HTML>
<HEAD>
<TITLE>Servlet HTTP GET Example (simple!)</TITLE>
</HEAD>
<BODY>
 <FORM ACTION=
        "http://localhost:8080/080306/SimpleServlet"
      METHOD="GET">
 <P>Click on the button to have the servlet send back an
    HTML document.</P>
 <INPUT TYPE="submit" VALUE="Get HTML Document">
 </FORM>
</BODY>
</HTML>
```

Creates a submit button

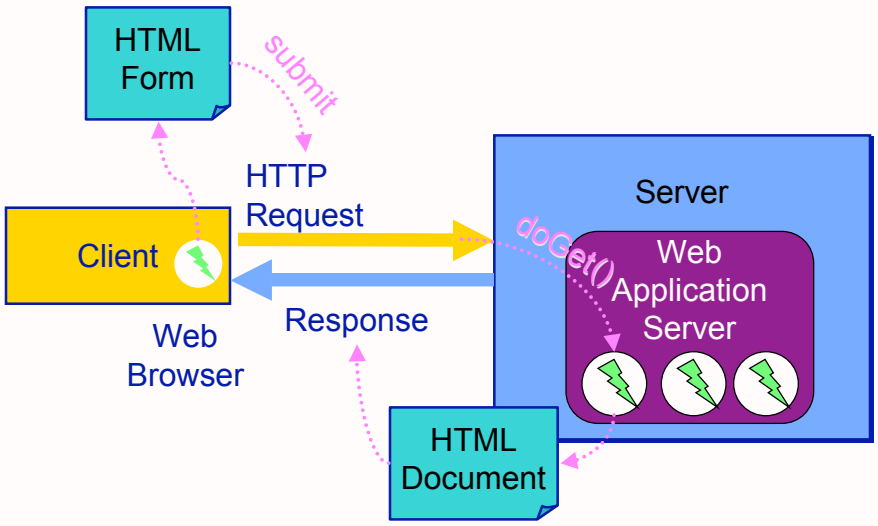
When the submit button is pressed, the browser makes a GET request to the web application server on the local machine listening to port 8080.
The application server then calls the doGet() method on the servlet, which is named HTTPGetServlet and located in a webapp directory

The Actual Servlet

- We can design the server to only accept/handle GET requests
  - Extend the HttpServlet class and override the doGet() method
  - Could return an error inside of doPost()
- doGet() method needs to
  - Obtain an output stream writer to write back to the client
  - Generate an HTML page
  - Write out the HTML page to the client using the writer
  - Close the writer


```
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
public class HTTPGetServlet extends HttpServlet {
   public void doGet( HttpServletRequest request,
                      HttpServletResponse response)
   throws ServletException, IOException
   {
      PrintWriter output;
      response.setContentType("text/html");
      output = response.getWriter();
      StringBuffer buffer = new StringBuffer();
      buffer.append("<HTML><HEAD><TITLE>\n");
      buffer.append("A Simple Servlet Example\n");
      buffer.append("</TITLE></HEAD><BODY>\n");
      buffer.append("<H1>Welcome to Servlets!</H1>\n");
      buffer.append("</BODY></HTML>\n");
      output.println( buffer.toString());
      output.close();
   }
}
```

The Example Servlet Flow



A More Complex Example

- This example was the most basic type of servlet
  - it always did the same thing
- The power of servlets is that the web server can receive data from the client, perform substantial processing, and then generate results to send back to the client
- As a more complex example, let's create a survey system
  - An HTML document that asks the user for their favorite type of pet
  - After they submit the form, the server sends back the current results of the survey.

Assumes on same server, Same path

```
<HTML>
   <HEAD>
      <TITLE>Pet Survey</TITLE>
   </HEAD>
 <BODY>
   <FORM METHOD="POST"
         ACTION="SurveyServlet">
    What is your favorite pet?<BR><BR>
   <INPUT TYPE=radio NAME=animal VALUE=dog>Dog<BR>
   <INPUT TYPE=radio NAME=animal VALUE=cat>Cat<BR>
   <INPUT TYPE=radio NAME=animal VALUE=bird>Bird<BR>
   <INPUT TYPE=radio NAME=animal VALUE=snake>Snake<BR>
   <INPUT TYPE=radio NAME=animal VALUE=fish>Fish<BR>
   <INPUT TYPE=radio NAME=animal VALUE=none CHECKED>Other<BR>
   <INPUT TYPE=submit VALUE="Submit">
   <INPUT TYPE=reset>
   </FORM>
 </BODY>
</HTML>
```

Create a radio button for each type of animal in the survey.


```
public class SurveyServlet extends HttpServlet {
   private String animalNames[] =
      {"dog", "cat", "bird", "snake", "fish", "none" };

   public void doPost( HttpServletRequest request,
                       HttpServletResponse response)
   throws ServletException, IOException
   {
      int animals[] = null, total = 0;
      File f = new File("survey.results");
      if (f.exists()) {
          try {
             ObjectInputStream in = new ObjectInputStream(
                new FileInputStream( f ));
             animals = (int []) in.readObject();
             in.close();
             for (int x = 0; x < animals.length; x++)
                total += animals[x];
          } catch (ClassNotFoundException exp) { };
       } else animals = new int[6];
```

```
        // read current response (that caused this to run)
        String value = request.getParameter("animal");
        total++;

        // determine which was selected and update the total
        for (int x = 0; x < animalNames.length; x++)
           if(value.equals(animalNames[x]))
              animals[x]++;

        // write updated total to the disk
        ObjectOutputStream out = new ObjectOutputStream(
           new FileOutputStream( f ));
        out.writeObject(animals);
        out.flush();
        out.close();

        // determine precentages
        double percentages[] = new double[animals.length];
        for (int x = 0; x < percentages.length; x++)
           percentages[x] = 100.0 * animals[x] / total;
```



```
      // now sent a thanks to the client and the results
      response.setContentType("text/html");
      PrintWriter clientOutput = response.getWriter();
      StringBuffer buffer = new StringBuffer();
      buffer.append("<HTML><TITLE>Thanks!</TITLE>\n");
      buffer.append("Thanks for your input.<BR>Results:\n<PRE>");
      DecimalFormat twoDigits = new DecimalFormat("#0.00");
      for (int x = 0; x < percentages.length; x++) {
         buffer.append("<BR>" + animalNames[x] + ": ");
         buffer.append(twoDigits.format(percentages[x]));
         buffer.append("% Responses: " + animals[x] + "\n");
      }
      buffer.append("\n<BR><BR>Total Responses: ");
      buffer.append(total + "</PRE>\n</HTML>");

      clientOutput.println(buffer.toString());
      clientOutput.close();
   }
}
```

Multiple Clients

- The survey servlet stores the results of the survey in a static file on the web server
- What happens if more than one client connects to the server at one time?
- The server handles both of the clients concurrently
  - More than one thread can open/close/modify that file at one time
  - Can lead to inconsistent data!
- Need to use Java's synchronization mechanisms
  - How would you synchronize SurveyServlet?

Cookies

- A popular way to customize web pages is to use cookies
  - Cookies are sent from the server (servlet) to the client
  - Small files, part of a header in the response to a client
    - Every HTTP transaction includes HTTP headers
  - Store information on the client's computer that can be retrieved later in the same browser session or in a future browsing session

Cookies

- When a servlet receives a request from the client, the header contains the cookies stored on the client by the server
- When the servlet sends back a response, the headers can include any cookies that the server wants to store on the client
- For example, the server could store a person's book preferences in a cookie
  - When that person returns to the online store later, the server can examine the cookies and read back the preferences

Cookie Structure

- Cookies have the name/value structure (similar to a hashtable)
- Creating a Cookie object is very easy
  - pass the constructor a name and a value
- For example, to store a user's preferred language on the client (so the servlet only has to ask for this information once)…

```
String cookie_name = new String("Pref_language");
String cookie_value = new String("English");
Cookie new_cookie = new Cookie(cookie_name, cookie_value);
```

Sending the Cookie to the Client

- Construct the Cookie object
- Call addCookie() on the HttpServletResponse object before you call the getWriter() method
  - HTTP header is always sent first, so the cookie(s) must be added to the response object before you start writing to the client

```
public void doPost( HttpServletRequest request,
                    HttpServletResponse response)
throws ServletException, IOException {
      . . .
      Cookie c = new Cookie("Pref_language", "English");
      c.setMaxAge(120);       // max age of cookie
      response.addCookie(c);
      . . .
      output = response.getWriter();
}
```


Cookies: Maximum Ages

- The maximum age of the cookie is how long the cookie can live on the client (in seconds)
- When a cookie reaches it maximum age, the client automatically deletes it

Retrieving Cookies

- Call getCookies() on the HttpServletRequest object
  - returns an array of Cookie objects, representing the cookies that the server previously sent to the client
- For example…

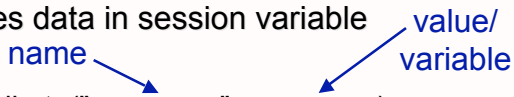
```
public void doPost( HttpServletRequest request,
                    HttpServletResponse response)
throws ServletException, IOException
{
      . . .
      Cookie[] cookies = request.getCookies();
      . . .
}
```


Retrieving Cookies

- The client will send all cookies that the server previously sent to it in the HTTP headers of its requests
- Client's cookies are available immediately upon entry into the doPost() and doGet() methods

Session variables

- A **session** is one user's visit to an application
  - Can be made up of lots of accesses
- Associate data with a session rather than a request
- Example:
  - User gives application data
  - Application stores data in session variable
    - session.setAttribute("username", username);



  - Application can use later, without user having to give information every time
    - String username = session.getAttribute("username");

JavaServer Pages (JSPs)

- Simplify web application development
- Separate UI from backend code
- Difficult to write HTML in print statements

- Merge HTML and Java
  - Separate static HTML from dynamic
  - Make HTML templates, fill in dynamic content
- Web application server compiles JSPs into Servlet code

JSP Syntax

- Enclose code in <% %>

```
<html>
<body>
Hello!  The time is now <%= new java.util.Date() %>
</body>
</html>
```

  - Aside: new convention is all lowercase HTML tags

JSP Syntax

```
<html>
<body>
<%
    // This is a scriptlet.  Notice that the "date"
    // variable we declare here is available in the
    // embedded expression later on.
    java.util.Date date = new java.util.Date();
%>
Hello!  The time is now <%= date %>
</body>
</html>
```

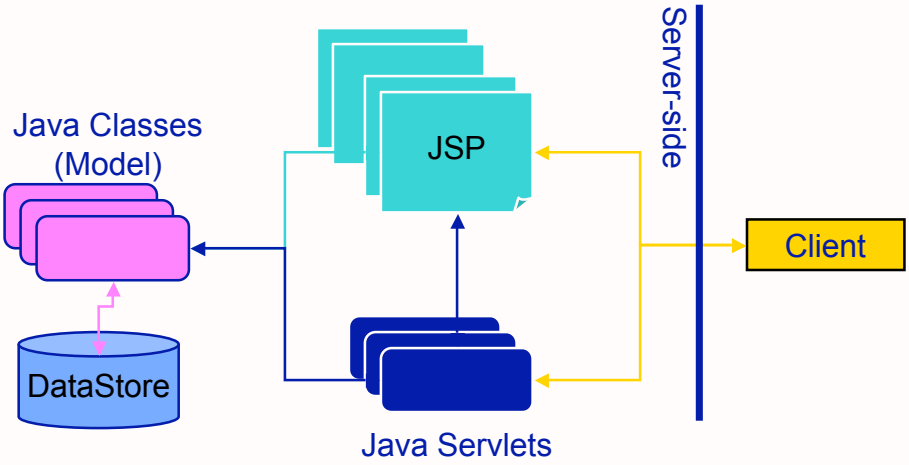
JSP Directives

- Page Directive
  - Java files to import (like import statement in Java)
  - `<%@ page import="java.util.*,java.text.*" %>`
  - `<%@ page import="servlets.SurveyServlet2"%>`
- Include Directive
  - Include contents of another file: JSP or HTML or text …
  - Could include common headers or footers for a site
  - `<%@ include file="hello.jsp" %>`

JSP Variables

- By default, JSPs have some variables
  - not explicitly declared in the file
  - HttpServletRequest request
  - HttpServletResponse response
  - HttpSession session
- From JSPs, can get access to request parameters, session data

Web Application Architecture



- Using traditional Java classes, JSPs and Servlets together

Communicating Between JSPs and Servlets

- Attributes
  - Name/Value pairs
  - Values are **Objects**
  - Can get/set attributes on the request object
- Rather than Parameters
  - All Strings
  - From forms or in URLs

SurveyServlet2
pet.jsp

A Real Example: CPM

- The Course Project Manager that we use
- Technologies
  - JSPs, Servlets
  - Filestore backend

Deployment: WAR files

- Web Archives
- Copy into webapp directory of web application server
  - Server will automatically extract files and run
  - Procedure for Apache/Tomcat and Resin
- Can create WAR files from Eclipse, with Web Tools Plugin

Configuration: web.xml

- Contains configuration parameters
- For security
  - Map URLs to Servlet names
  - Map Servlet classes to Servlet names

Web Programming Tools

- Eclipse plugin
  - Web Tools Platform
    http://www.eclipse.org/webtools/
- Firefox plugins
  - Firebug
  - Web developer plug-in:
    http://chrispederick.com/work/webdeveloper/

Newer Web Technologies

- Struts
  - Controller component of MVC
  - Model and Viewer are from other standard technologies
    - JSPs, JDBC
- JavaServer Faces (JSF)
  - Framework to make creating UIs easier
  - Custom JSP tag libraries

Exam Preparation

- Similar format to quizzes
- Variety of questions
  - May have some “find the bug” and coding questions
- Covers all topics this semester
  - What we did, how/when to apply/use
- Bring your questions next Tuesday
  - Also have quiz
  - Brief overview of other packages

Project2

- Replay tool to help automate testing
  - Generate different test cases
  - Makes requests, saves responses