World Wide Web

The World Wide Web

- NOT invented by Academics or Industry ☺
- Invented by Tim Berners-Lee at CERN:
  - CERN is a meeting place for physicists from all over the world, who collaborate on complex physics, engineering and information handling projects.
  - Thus, the need for the WWW system arose "from the geographical dispersion of large collaborations, and the fast turnover of fellows, students, and visiting scientists," who had to get "up to speed on projects and leave a lasting contribution before leaving.“

WWW Evolution

- Took off once first fully integrated graphical browser was developed: NCSA Mosaic (first version posted to NCSA servers in 1993)
  - Marc Andreesen co-wrote Mosaic as a student, went on to co-found Netscape
  - Continued development of core technologies, Tim Berners-Lee now heads the World Wide Web Consortium at MIT, www.w3c.org

The World Wide Web

- No need to introduce the Web, is there?
  - A uniform resource locator: URL
  - A protocol: HTTP (HyperText Transfer Protocol)
    - The client: a Web browser
    - The server: the Web server
  - A markup language: HTML (HyperText Markup Language)
    - Simple markup language
    - Very presentation-oriented
    - Designed for essentially rendering content on a PC-class device (as opposed to HDML: Handheld Device Markup Language, part of WAP)

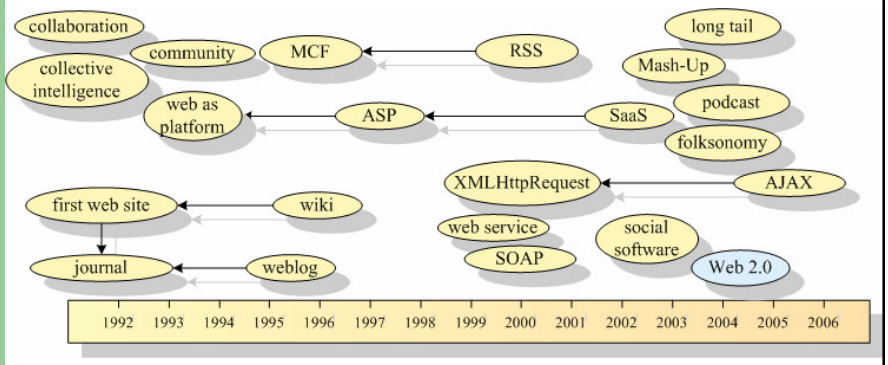
Static vs. Dynamic WWW

- Original vision: WWW content is dynamic, collaboratively edited/shared, sort of like the Wiki idea now (http://en.wikipedia.org/wiki/Wiki)
- Reality: first websites were collections of static HTML files, downloaded by user to browse information
- Soon: customization, access to dynamically changing information, interface to applications → how to make WWW content more dynamic?

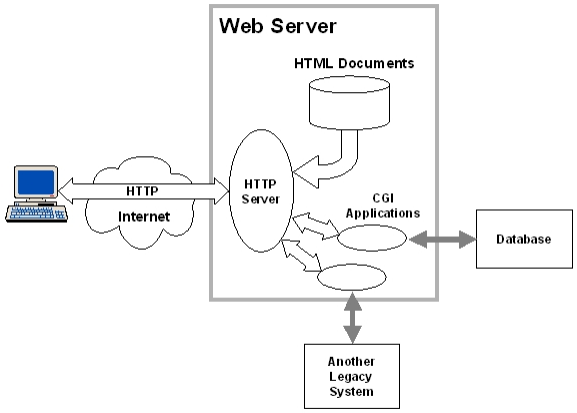
Dynamic WWW pages

- Client-side rendering: Stylesheets, JavaScript, Java, Flash…
  - Limited in functionality: basic input validation, simple games, animations, etc.
- Server-side dynamic generation of HTML documents: CGI, Servlets, ASPs, JSPs…
  - Focus in this course

WWW Buzzword Timeline



"Original" WWW Architecture



HTTP

- Hypertext Transport Protocol
  - Not limited to hypertext though!
- Client/server
- Transaction-oriented
- Stateless
- Based on TCP (opening one or multiple TCP connections)

URL: Uniform Resource Locator

- Not limited to HTTP:

  protocol://host:port/resource_path
- The browsers default to:
  - `http` protocol
  - Port 80
  - Index.html resource
- Resources can be *static* or *dynamic*

The HTTP Protocol

- RFCs 1945 and 2616 (previously 2068) for versions 1.0 and 1.1 respectively
  - To obtain these protocols, see www.ietf.org
- HTTP transactions consist of a *request* and a *response*
  - Two types of request *methods*: `GET` and `POST`

GET Request

- Simple get request: (HTTP 0.9)

```
GET /document.html [CRLF]
```

- Full get request:

```
GET /document.html HTTP/1.0 [CRLF]
```

- Full get request with headers:

```
GET /document.html HTTP/1.0 [CRLF]
If-Modified-Since: Sun 20 Oct 1996 04:07:51 GMT[CRLF]
[LF]
```

Post Request

- Post allows the client to include a body of data in a request:

```
POST /cgi-bin/code.cgi HTTP/1.0 [CRLF]
Content-type: application/octet-stream [CRLF]
Content-length: 2048 [CRLF]
[LF]
body
```

HTTP Responses

- Simple Response: `body`
- Full Response:

```
HTTP/1.0 200 OK[CRLF]
[LF]
body
```

- Full Response with headers:

```
HTTP/1.0 200 OK[CRLF]
Content-type: text/html[CRLF]
[LF]
body
```

HTTP Response Codes

- Here a few response codes:

```
200 OK
400 Bad Request
401 Unauthorized
404 Not Found
500 Internal Server Error
```

MIME Types

- Originally designed for email, associates a type with a message to help the receiver to decode/view (RFC 1521)
- Can be used in a HTTP header
- Type/subtype
- Common types/subtypes:

```
text/html, text/plain, image/gif…
```

Programming Web Applications

- CGI
- Servlets
- JSP

- What do these have in common?
  - They are all server-side technologies!

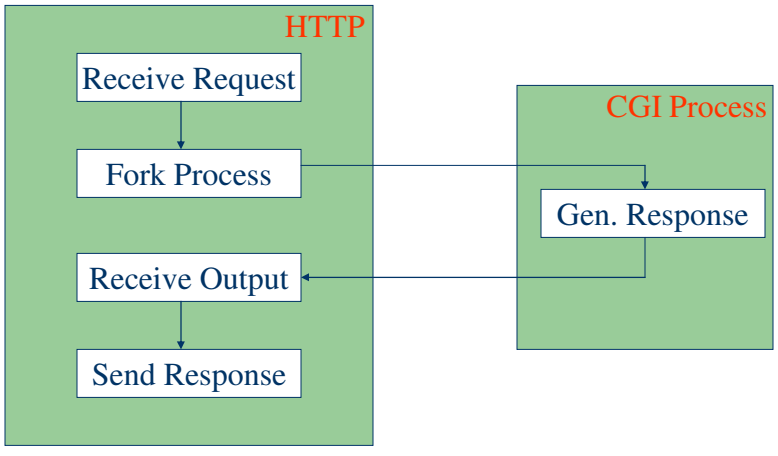
CGI

- What is CGI
- How does it work?
- Environment variables
- Processing Forms
- GET vs. POST
- Examples

What is CGI?

- Stands for “Common Gateway Interface”
- Server-side technology
- Can be used:
  - To process fill-out forms
  - To generate dynamic contents
  - By a web server to run external programs
  - By a web server to get/send data from databases and other apps

How does it work?



CGI

- CGI scripts can be written in any language, including Java
- Perl is the most popular for CGI scripting
- To experiment, you need a web server (see list posted on course website)

Content headers

- If your script generates HTML then use:

  Content-type: text/html\n\n

This tells the browser what content it is about to receive

- Other content headers (MIME!) include:
  - text/plain
  - image/gif
  - image/jpg

Sample Script (SIMPLE.PL)

```
#!/usr/bin/perl

print "Content-type:text/html\n\n";
print "<html><head><title>Test Page</title></head>\n";
print "<body>\n"; print "<h2>Hello, world!</h2>\n";
print "</body></html>\n";
```

Environment Variables

- Some of the environment variables:
  - DOCUMENT_ROOT
  - HTTP_HOST
  - HTTP_USER_AGENT
  - REMOTE_HOST
  - REQUEST_METHOD
  - QUERY_STRING
  - CONTENT_LENGTH …etc

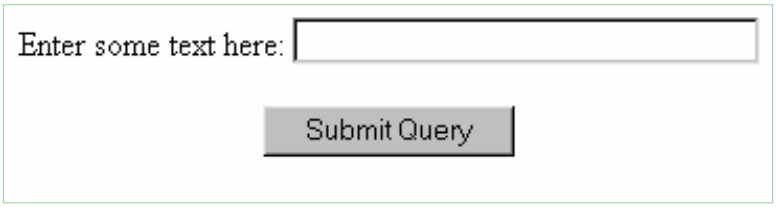
Script: environment variables (SIMPLE2.PL)

```
#!/usr/bin/perl
print "Content-type:text/html\n\n";
print “<html><head><title>Print Environment</title></head> <body>”;

foreach $key (sort(keys %ENV)) {
    print "$key = $ENV{$key}<br>\n";
}
print "</body></html>";
```

Forms

```
<form action=“simple.pl" method="GET">
Enter some text here: <input type="text"
    name="sample_text" size=30><input
    type="submit"><p>
</form>
```



Forms

- There are two ways to send data from an HTML form to a CGI script
  - GET
  - POST
- These methods determine how the form data is sent to the server

GET

- The input values from the form are sent as part of the URL
- They are saved in the QUERY_STRING environment variable
- If in the above example you type:
  "hello there John"
- The QUERY_STRING will be:
  Sample_text=hello+there+John
- Spaces have been replaced with +

GET….

- This is called URL Encoding!
- Some commonly encoded characters

\t (tab)	%09
\n (return)	%0A
/	%2F
~	%7E
:	%3A
;	%3B
@	%40
&	%26

GET….

<form action="simple2.pl" method="GET">
First Name: <input type="text" name="fname" size=30><p>
Last Name: <input type="text" name="lname" size=30><p>
<input type="submit"> </form>

If input is: Sarah Johnson

$ENV{'QUERY_STRING'} would be: fname=Sarah&lname=Johnson

GET….

- Parsing:

```
@values = split(/&/,$ENV{'QUERY_STRING'});
 foreach $i (@values) {
   ($varname, $mydata) = split(/=/,$i);
   print "$varname = $mydata\n";
}
```

GET….

- It is possible to send values as part of a URL
- Hidden values can be used to maintain session info

POST

- More sophisticated than GET
- Data is not sent as URL-encoded (I.e. not part of the URL)
- When POST is used, data is sent as a separate message (input stream)

POST….

- Parsing:

```
read(STDIN, $buffer, $ENV{'CONTENT_LENGTH'});
 @pairs = split(/&/, $buffer);
foreach $pair (@pairs) {
    ($name, $value) = split(/=/, $pair);
   $value =~ tr/+/ /;
   $value =~ s/%([a-fA-F0-9][a-fA-F0-9])/pack("C",
    hex($1))/eg;
   $FORM{$name} = $value;
}
```

Evaluation of CGI

- Advantages
  - Expands static WWW pages
  - Wide range of programming languages supported
  - “easy” to use
- Disadvantages
  - High overhead (new process per request)
  - Low level (explicitly generate HTML code)
  - No support for state across requests (sessions, etc.)

Java support for WWW

Java support…

- The URL class
- Applets
- Servlets and JSPs…

The `URL` class

- A facility to retrieve objects from the network
- Decodes the object based on its extension
  - For example, a .gif file will generate an `Image` object
  - Can be extended to any object type that you want
- The objects must obviously be addressable by a URL
  - So far, support for "`http:`" and "`file:`"
  - Can be extended to support "`ftp:`" and others

URL class example

```
URL url = new URL(http://java.sun.com/index.html);
InputStream in = url.openStream();
…
Or:
URL url = new URL(http://java.sun.com/duke.gif);
Image im = (Image) url.getContent();
```

- An HTTP connection is made
- See full URL doc

Java Applets

- Client-side
- See example applet on course web site
- Use somewhat limited:
  - Downloaded code has limited access to host machine
  - Usually used for simple validation of input fields in forms, games, animations, etc. (see example on course website)
  - Typically NOT used for complex business logic

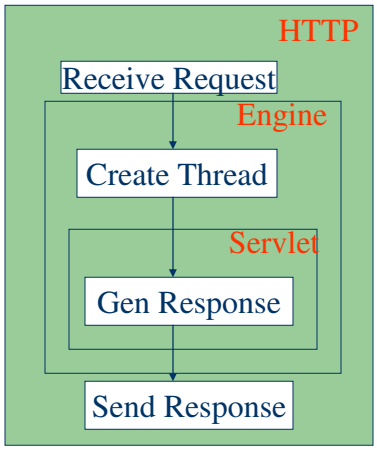
What is a Servlet?

- A server-side technology
- Designed to overcome some limitations of existing technologies (e.g. CGI is stateless)
- Characteristics:
  - A light-weight task that can be executed as a thread
  - A servlet can remain in memory (a CGI script terminates when it finished)
- Advantages:
  - A servlet can service multiple client requests
  - Can handle multiple clients without reloading/reinitialization

Servlets….

- Servlets are written in Java
- Can be used to:
  - Generate dynamic contents
  - Talk to databases
  - Work with cookies
  - Session management

How do they work?



Servlet Framework

- The package: javax.servlet
- At the top level there are three interfaces:
  - ServletConfig, Servlet, Serializable
- The servlet interface:

  init()
  service()
  destroy()
  getServiceConfig()
  getServiceInfo()

The GenericServlet

- It is an abstract class that implements Servlet

```
public abstract GenericServlet implements
                Servlet, ServletConfig, Serializable {
  void init()
  abstract void service()
  void destroy()
  ServletConfig getServletConfig()
  String getServiceInfo()
  void log()
}
```

The HttpServlet

- It is an abstract class that extends the GenericServlet abstract class
- It provides a convenient framework for handling the HTTP protocol
- These two classes (GenericServlet and HttpServlet) ease the task of writing servlets
- In the simplest case, you need to provide implementation for service()

Life Cycle of a Servlet

- Servlet is loaded into memory by server: init()
- Servlet processes client requests: service()
- Servlet is removed by server: destroy()
- service() is responsible for handling incoming client requests
  - public void service(ServiceRequest request, ServletResponse response) throws ServletException, IOException
- Delegates HTTP requests: doGet() & doPost()

Retrieving Parameters

- Use:
  - public String getParameter(String name)
  - public String[] getParameterValues(String name)
- So if you have a parameter name “username” in a form then to retrieve the value use:
  - String name = request.getParameter(“username”);

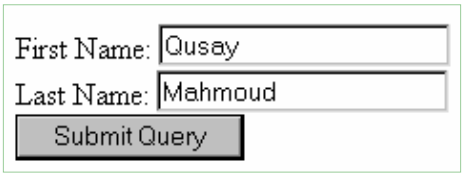
Example:

- Consider the following form:

```
<form action="RequestParamExample"
   method=POST>
First Name: <input type=text size=20
   name=firstname> <br>
Last Name: <input type=text size=20
   name=lastname> <br>
<input type=submit>
</form>
```

Example….

- In a browser, this would look like:



- When “Submit Query” is clicked we have the output:
  - First Name := Qusay
  - Last Name := Mahmoud

Example

- The servlet:

```
public class RequestParams extends HttpServlet {
  public doPost(HttpServletRequest re, HttpServletResponse response)
   {
      PrintWriter out = response.getWriter();
      out.println("<html><body><head><title>test</title></head");
      out.println("<body>");
      String firstName = req.getParameter("firstname");
      String lastName = req.getParameter("lastname");
      out.println("First Name := " + firstName + "<br>");
      out.println("Last Name := " + lastName);
      out.println("</body></html>");
}}
```

An Example Servlet


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

public class HelloWorld extends HttpServlet {
  public void doGet(HttpServletRequest request,
                    HttpServletResponse response)
        throws IOException, ServletException {
    response.setContentType("text/html");
    PrintWriter out = response.getWriter();

   out.println("<html><head><title>ServletExample</title></head>"+
               "<body><h1>Hello World!</h1>"+
               "This page was last updated: "+
               new java.util.Date()+
               "</body></html>");
  }
}
```

Requests

- Methods in **`HttpServletRequest`**
  - `getHeader`
  - `getParameter`
  - `getInputStream`
  - `getRemoteHost`, `getRemoteAddr`, `getRemotePort`
  - …

Example: `HttpServletRequest` (1/2)

```
public class Requests extends HttpServlet {
  public void doGet(HttpServletRequest request,
                    HttpServletResponse response)
       throws IOException, ServletException {
    response.setContentType("text/html");
    PrintWriter out = response.getWriter();
    out.println("<html><head><title>Requests</title></head><body>");
    out.println("<h1>Hello, visitor from "+request.getRemoteHost()+"</h1>");
    String useragent = request.getHeader("User-Agent");
    if (useragent!=null)
      out.println("You seem to be using "+useragent+"<p>");
    String name = request.getParameter("name");
    if (name==null)
      out.println("No <tt>name</tt> field was given!");
    else
      out.println("The value of the <tt>name</tt> field is: <tt>" +
                  htmlEscape(name) + "</tt>");
    out.println("</body></html>");
  }
```

Example: `HttpServletRequest` (2/2)

```
  public void doPost(HttpServletRequest request,
                     HttpServletResponse response)
        throws IOException, ServletException {
   doGet(request, response);
  }
  private String htmlEscape(String s) {
    StringBuffer b = new StringBuffer();
    for (int i = 0; i<s.length(); i++) {
      char c = s.charAt(i);
      switch (c) {
        case '<': b.append("&lt;"); break;
        case '>': b.append("&gt;"); break;
        case '"': b.append("&quot;"); break;
        case '\'': b.append("&apos;"); break;
        case '&': b.append("&amp;"); break;
        default: b.append(c);
    } }
    return b.toString();
} }
```

Hello, visitor from britney.widget.inc

You seem to be using Mozilla/5.0 (X11; U; Linux i686; en-US; rv:1.5) Gecko/20031007

The value of the `name` field is: `John Doe`

Responses

- Methods in **`HttpServletResponse`**
  - `setStatus`
  - `addHeader, setHeader`
  - `getOutputStream, getWriter`
  - `setContentType`
  - `sendError, sendRedirect`
  - …

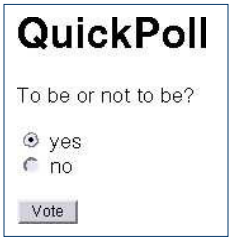
Servlet Contexts

- One `ServletContext` object for each Web application
- `getServerInfo`
- `getInitParameter`
- ...
- Shared state:
  - `setAttribute("`*name*`", `*value*`)`
  - `getAttribute("`*name*`")`
  - *don't use for mission critical data!*

Example: A Polling Service

A Web application consisting of

- `QuickPollQuestion.html`
- `QuickPollSetup.java`
- `QuickPollAsk.java`
- `QuickPollVote.java`
- `QuickPollResults.java`



Example: QuickPollSetup.html

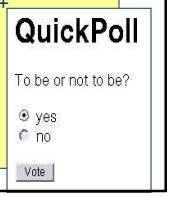
```
<html>
<head><title>QuickPoll</title></head>
<body>
<h1>QuickPoll</h1>
<form method=post action=setup>
What is your question?<br>
<input name=question type=text size=40>?<br>
<input type=submit name=submit
       value="Register my question">
</form>
</body>
</html>
```

Example: QuickPollQuestion.java

```
public class QuickPollSetup extends HttpServlet {
  public void doPost(HttpServletRequest request,
                     HttpServletResponse response)
       throws IOException, ServletException {
    String q = request.getParameter("question");
    ServletContext c = getServletContext();
    c.setAttribute("question", q);
    c.setAttribute("yes", new Integer(0));
    c.setAttribute("no", new Integer(0));
    response.setContentType("text/html");
    PrintWriter out = response.getWriter();
    out.print("<html><head><title>QuickPoll</title></head><body>"+
              "<h1>QuickPoll</h1>"+
              "Your question has been registered. "+
              "Let the vote begin!"+
              "</body></html>");
} }
```

Example: QuickPollAsk.java

```
public class QuickPollAsk extends HttpServlet {
  public void doGet(HttpServletRequest request,
                    HttpServletResponse response)
       throws IOException, ServletException {
    response.setContentType("text/html");
    PrintWriter out = response.getWriter();
    out.print("<html><head><title>QuickPoll</title></head><body>"+
              "<h1>QuickPoll</h1>"+
              "<form method=post action=vote>");
    String question =
      (String)getServletContext().getAttribute("question");
    out.print(question+"?<p>");
    out.print("<input name=vote type=radio value=yes> yes<br>"+
              "<input name=vote type=radio value=no> no<p>"+
              "<input type=submit name=submit value=Vote>"+
              "</form>"+
              "</body></html>");
} }
```



Example: QuickPollVote.java (1/2)

```
public class QuickPollVote extends HttpServlet {
  public void doPost(HttpServletRequest request,
                     HttpServletResponse response)
       throws IOException, ServletException {
    String vote = request.getParameter("vote");
    ServletContext c = getServletContext();
    if (vote.equals("yes")) {
      int yes = (Integer)c.getAttribute("yes")).intValue();
      yes++;
      c.setAttribute("yes", new Integer(yes));
    } else if (vote.equals("no")) {
      int no = ((Integer)c.getAttribute("no")).intValue();
      no++;
      c.setAttribute("no", new Integer(no));
    }
```

Example: QuickPollVote.java (2/2)

```
    response.setContentType("text/html");
    PrintWriter out = response.getWriter();
    out.print("<html><head><title>QuickPoll</title></head><body>"+
              "<h1>QuickPoll</h1>"+
              "Thank you for your vote!"+
              "</body></html>");
  }
}
```

Example: QuickPollResult.java (1/2)

```
public class QuickPollResults extends HttpServlet {
  public void doGet(HttpServletRequest request,
                    HttpServletResponse response)
        throws IOException, ServletException {
    ServletContext c = getServletContext();
    String question = (String)c.getAttribute("question");
    int yes = ((Integer)c.getAttribute("yes")).intValue();
    int no = ((Integer)c.getAttribute("no")).intValue();
    int total = yes+no;
    response.setContentType("text/html");
    response.setDateHeader("Expires", 0);
    response.setHeader("Cache-Control",
                       "no-store, no-cache, must-revalidate");
    response.setHeader("Pragma", "no-cache");
    PrintWriter out = response.getWriter();
```

Example: `QuickPollResult.java` (2/2)

```
    out.print("<html><head><title>QuickPoll</title></head><body>"+
              "<h1>QuickPoll</h1>");
    if (total==0)
      out.print("No votes yet...");
    else {
      out.print(question + "?<p>"+"<table border=0>"+
       "<tr><td>Yes:<td>"+drawBar(300*yes/total)+"<td>"+yes+
       "<tr><td>No:<td>"+drawBar(300*no/total)+"<td>"+no+
       "</table>");
    }
    out.print("</body></html>");
  }
  String drawBar(int length) {
    return "<table><tr><td bgcolor=black height=20 width="+
           length+"></table>";
} }
```

Summary

- Servlets closely follow the **request-response** pattern from HTTP
- Features:
  - Multi-threading
  - Declarative configuration
  - Request parsing, including decoding of form data
  - Shared state
  - Session management
  - Advanced code structuring: listeners, filters, wrappers
  - Client authentication, SSL

Essential Online Resources

- The servlet API:
  http://jakarta.apache.org/tomcat/tomcat-5.5-doc/servletapi/
- Sun's home page for servlets:
  http://java.sun.com/products/servlet/
- The Tomcat server:
  http://jakarta.apache.org/tomcat/

Limitations of Servlets

- Low-level **construction of HTML documents**
  - fragments (strings) written to output stream
  - no static well-formedness/validity guarantees
- Low-level **session management**
  - control-flow is often unclear
  - no enforcement of relation between showing a page and receiving form input
  - primitive session state management

JWIG

- Research project (http://www.jwig.org/)
- **Session threads**
  - showing a page and receiving form input modeled as a Remote Procedure Call (RPC)
    - explicit control-flow
    - simpler session state management
- **Template**-based page construction using XACT
- **Static checking** of
  - output validity
  - form field consistency

JSP

- Server-side technology
- Enables you to embed Java code within an HTML document
- JSP documents have the extension .jsp
- When an HTTP request is received, the compilation engine converts the JSP document into a Java Servlet then the servlet will be loaded
- Java code is embedded between <% and %>

Example

```
// file: hello.jsp
<html><head><title>example</title></head>
<body>
<% String visitor = request.getParameter("user");
   if (visitor == null) visitor = "there";
%>
Hello, <%= visitor %>!
</body>
</html>
```

Example

- Request parameters are passed into JSP pages using normal HTTP parameter mechanisms (using GET and POST)
- Therefore, for the hello.jsp:
  - If invoked with: **http://host…/hello.jsp** it will print "Hello, there!"
  - If invoked with: **http://host…/hello.jsp?user=Mary**, it will print: "Hello, Mary!"

A Tiny Example


```
<% response.addDateHeader("Expires", 0); %>
<html>
  <head><title>JSP</title></head>
  <body>
    <h1>Hello World!</h1>
    <%! int hits = 0; %>
    You are visitor number
    <% synchronized(this) { out.println(++hits); } %>
    since the last time the service was restarted.
    <p>
    This page was last updated:
    <%= new java.util.Date().toLocaleString() %>
  </body>
</html>
```

JSP Templates

- A text file with snippets of code:
  - directives
  - declarations
  - expressions
  - statements
- Implicitly declared variables:
  - HttpServletRequest request;
  - HttpServletResponse response;
  - HttpSession session;
  - ServletContext application;
  - ServletConfig config;
  - JspWriter out;
  - PageContext pageContext;

JSP Expressions

```
<html>
  <head><title>Addition</title></head>
  <body>
    The sum of <%= request.getParameter("x") %>
    and <%= request.getParameter("y") %> is
    <%= Integer.parseInt(request.getParameter("x")) +
        Integer.parseInt(request.getParameter("y")) %>
  </body>
</html>
```

JSP Statements

```
<html>
  <head><title>Numbers</title></head>
  <body>
    <ul>
      <% int n =
            Integer.parseInt(request.getParameter("n"));
         for (int i=0; i<n; i++)
           out.println("<li>"+i+"</li>");
      %>
    </ul>
  </body>
</html>
```

JSP Declarations

```
<%! int add(String x, String y) {
      return Integer.parseInt(x)+Integer.parseInt(y);
    }
%>
<html>
  <head><title>Addition</title></head>
  <body>
    The sum of <%= request.getParameter("x") %>
    and <%= request.getParameter("y") %> is
    <%= add(request.getParameter("x"),
            request.getParameter("y")) %>
  </body>
</html>
```

Be Careful About Declarations (1/2)

```
<% response.addDateHeader("Expires", 0); %>
<html>
  <head><title>JSP</title></head>
  <body>
    <h1>Hello World!</h1>
    <%! int hits = 0; %>
    You are visitor number
    <% synchronized(this) { out.println(++hits); } %>
    since the last time the service was restarted.
    <p>
    This page was last updated:
    <%= new java.util.Date().toLocaleString() %>
  </body>
</html>
```

Be Careful About Declarations (2/2)

```
<% response.addDateHeader("Expires", 0); %>
<html>
  <head><title>JSP</title></head>
  <body>
    <h1>Hello World!</h1>
    <% int hits = 0; %>
    You are visitor number
    <% synchronized(this) { out.println(++hits); } %>
    since the last time the service was restarted.
    <p>
    This page was last updated:
    <%= new java.util.Date().toLocaleString() %>
  </body>
</html>
```

This page counter is always 1...

JSP Directives

```
header.jsp
<html>
  <head><title><%= title %></title></head>
  <body>

footer.jsp
  </body>
</html>

<%! String title = "Addition"; %>
<%@ include file="header.jsp" %>
  The sum of <%= request.getParameter("x") %>
  and <%= request.getParameter("y") %> is
  <%= Integer.parseInt(request.getParameter("x")) +
      Integer.parseInt(request.getParameter("y")) %>
<%@ include file="footer.jsp" %>
```

Other Directives

- contentType="*type*"
- info="*String*"
- errorPage="*path*"
- isErrorPage="*boolean*"
- import="*package*"

Using Error Pages

```
<%@ page errorPage="error.jsp" %>
<html>
  <head><title>Division</title></head>
  <body>
    <% int n = Integer.parseInt(request.getParameter("n")); %>
    <% int m = Integer.parseInt(request.getParameter("m")); %>
    <%= n %>/<%= m %> equals <%= n/m %>
  </body>
</html>
```

```
<%@ page isErrorPage="true" %>
<html>
  <head><title>Error</title></head>
  <body>
     Something bad happened:
     <%= exception.getMessage() %>
  </body>
</html>
```

Translation into Servlets

```
<% response.addDateHeader("Expires", 0); %>
<html>
  <head><title>JSP</title></head>
  <body>
    <h1>Hello World!</h1>
    <%! int hits = 0; %>
    You are visitor number
    <% synchronized(this) { out.println(++hits); } %>
    since the last time the service was restarted.
    <p>
    This page was last updated:
    <%= new java.util.Date().toLocaleString() %>
  </body>
</html>
```

The Generated Servlet (1/5)

```
package org.apache.jsp;

import javax.servlet.*;
import javax.servlet.http.*;
import javax.servlet.jsp.*;

public final class hello_jsp
  extends org.apache.jasper.runtime.HttpJspBase
  implements org.apache.jasper.runtime.JspSourceDependent {

  int hits = 0;
  private static java.util.Vector _jspx_dependants;

  public java.util.List getDependants() {
    return _jspx_dependants;
  }
```

The Generated Servlet (2/5)

```
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;
   JspWriter _jspx_out = null;
   PageContext _jspx_page_context = null;
```

The Generated Servlet (3/5)

```
  try {
     _jspxFactory = JspFactory.getDefaultFactory();
     response.setContentType("text/html");
     pageContext = _jspxFactory.getPageContext(
                     this, request, response,
                     null, true, 8192, true);
     _jspx_page_context = pageContext;
     application = pageContext.getServletContext();
     config = pageContext.getServletConfig();
     session = pageContext.getSession();
     out = pageContext.getOut();
     _jspx_out = out;
```

The Generated Servlet (4/5)

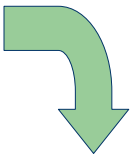
```
response.addDateHeader("Expires", 0);
out.write("\n");
out.write("<html><head><title>JSP</title></head>\n");
out.write("<body><h1>Hello World!</h1>\n");
out.write("\n");
out.write("You are visitor number ");
synchronized(this) { out.println(++hits); }
out.write("\n");
out.write("since the last time the service was restarted.\n");
out.write("<p>\n");
out.write("This page was last updated: ");
out.print(new java.util.Date().toLocaleString());
out.write("\n");
out.write("</body></html>\n");
```

The Generated Servlet (5/5)

```
    } catch (Throwable t) {
      if (!(t instanceof SkipPageException)){
        out = _jspx_out;
        if (out != null && out.getBufferSize() != 0)
          out.clearBuffer();
        if (_jspx_page_context != null)
          _jspx_page_context.handlePageException(t);
      }
    } finally {
      if (_jspxFactory != null)
        _jspxFactory.releasePageContext(_jspx_page_context);
    }
  }
}
```

Literal Translation

```
<% if (Math.random() < 0.5) { %>
  Have a <b>nice day!
<% } else { %>
  Have a <b>lousy day!
<% } %>
</b>
```



```
if (Math.random() < 0.5) {
     out.write("\n");
     out.write("  Have a <b>nice day!\n");
} else {
     out.write("\n");
     out.write("  Have a <b>lousy day!\n");
}
out.write("</b>\n");
```

Advanced Features

- XML version of JSP
- The expression language
- Tag files
- JSTL

JSP Pages Are Not XML

```
<html>
  <head><title>JSP Color</title></head>
  <body bgcolor=<%= request.getParameter("color") %>>
    <h1>Hello World!</h1>
    <%! int hits = 0; %>
    You are visitor number
    <% synchronized(this) { out.println(++hits); } %>
    since the last time the service was restarted.
    <p>
    This page was last updated:
    <%= new java.util.Date().toLocaleString() %>
  </body>
</html>
```

- This page generates HTML, not XHTML
- <%. . .%> is not well-formed XML

XML Version of JSP

```
<jsp:root xmlns:jsp="http://java.sun.com/JSP/Page" version="2.0"
          xmlns="http://http://www.w3.org/1999/xhtml">
  <jsp:directive.page contentType="text/html"/>
  <jsp:scriptlet>
    response.addDateHeader("Expires", 0);
  </jsp:scriptlet>
  <html>
    <head><title>JSP</title></head>
    <jsp:element name="body">
      <jsp:attribute name="bgcolor">
        <jsp:expression>
          request.getParameter("color")
        </jsp:expression>
      </jsp:attribute>
      <h1>Hello World!</h1>
      <jsp:declaration>
        int hits = 0;
      </jsp:declaration>
      You are visitor number
      <jsp:scriptlet>
        synchronized(this) { out.println(++hits); }
      </jsp:scriptlet>
      since the last time the service was restarted.
      <p/>
      This page was last updated:
      <jsp:expression>
        new java.util.Date().toLocaleString()
      </jsp:expression>
    </jsp:element>
  </html>
</jsp:root>
```

- Uses `<jsp:...>`
- No schema seems to be available
- No validation of the output
- No validation of Java code
- but it's there...

The Expression Language

- We want to **avoid explicit Java code** in JSP templates
- The syntax **`${exp}`** may be used in
  - template text
  - attribute values in markup
- The expression may access
  - variables in the various scopes
  - implicit objects, such as `param`
- The usual operators are available

An Expression Example

```
<html>
  <head><title>Addition</title></head>
  <body bgcolor="${param.color}">
    The sum of ${param.x} and ${param.y} is ${param.x+param.y}
  </body>
</html>
```

Tag Libraries

- Libraries of tags capturing common patterns:
  - pagination of large texts
  - date and times
  - database queries
  - regular expressions
  - HTML scraping
  - bar charts
  - cookies
  - e-mail
  - WML
  - ...

JSTL 1.1

- JSP Standard Tag Library covers:
  - assigning to variables
  - writing to the output stream
  - catching exceptions
  - conditionals
  - iterations
  - URL construction
  - string formatting
  - SQL queries
  - XML manipulation
- Current version 1.2 (a “maintenance” release, including the use of generics in Java 5.0)

Evaluation of Tags

- Make Web applications available to a wider range of developers
- May be used to structure applications

- A myriad of domain-specific languages
- Brittle implementation, hard to debug

Summary

- **JSP templates** are HTML/XHTML pages with embedded code
- The simple **expression language** is often sufficient in place of full-blown Java code
- **Tag files and libraries** allow code to be hidden under a tag-like syntax

Essential Online Resources

- Sun's home page for JSP:
  http://java.sun.com/products/jsp/
- JSTL:
  http://java.sun.com/products/jsp/jstl/