# JSP

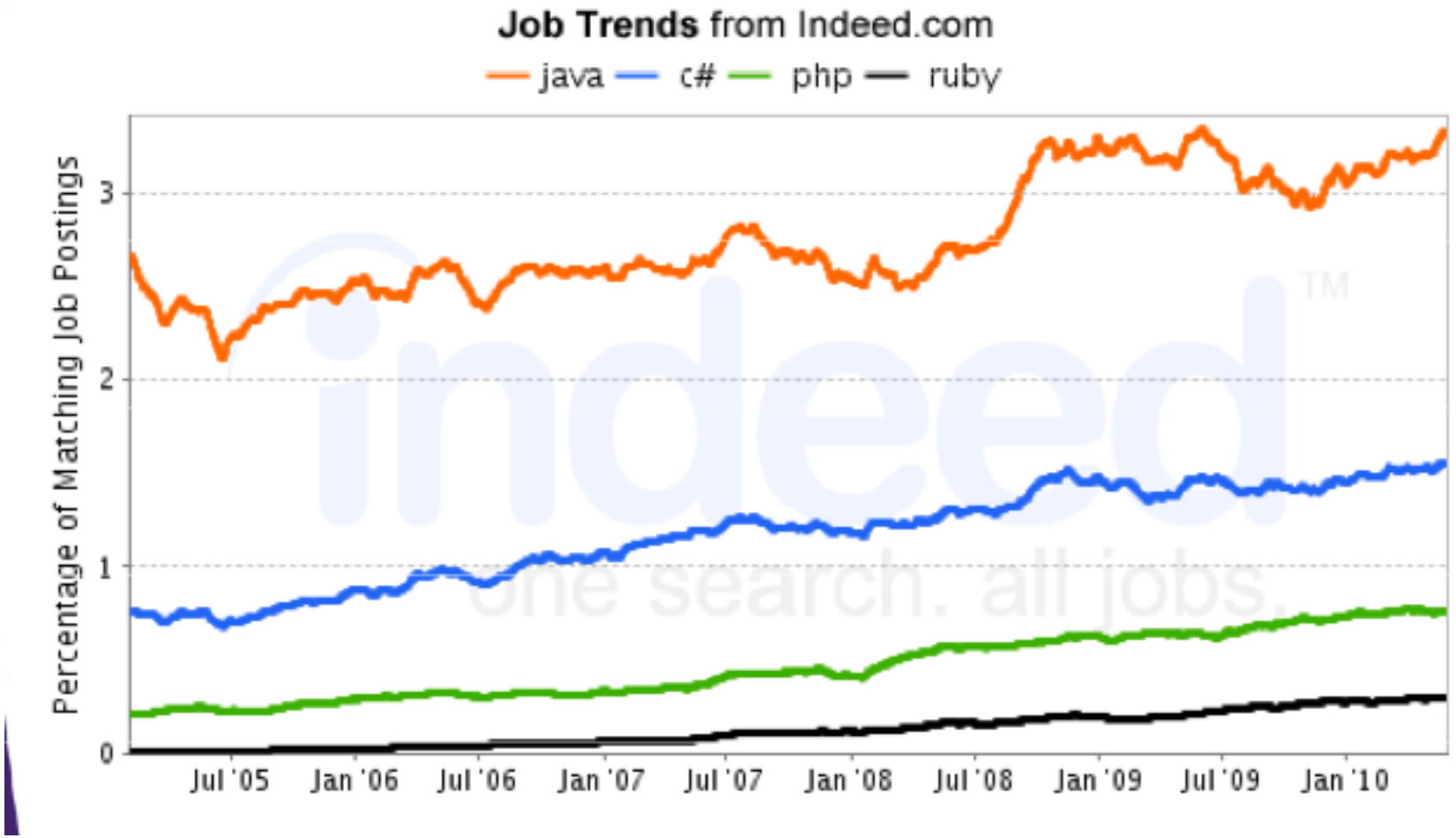
Thanks to

http://courses.coreservlets.com/
Course-Materials/csajsp2.html

# The 10 most popular sites (Summer.10)

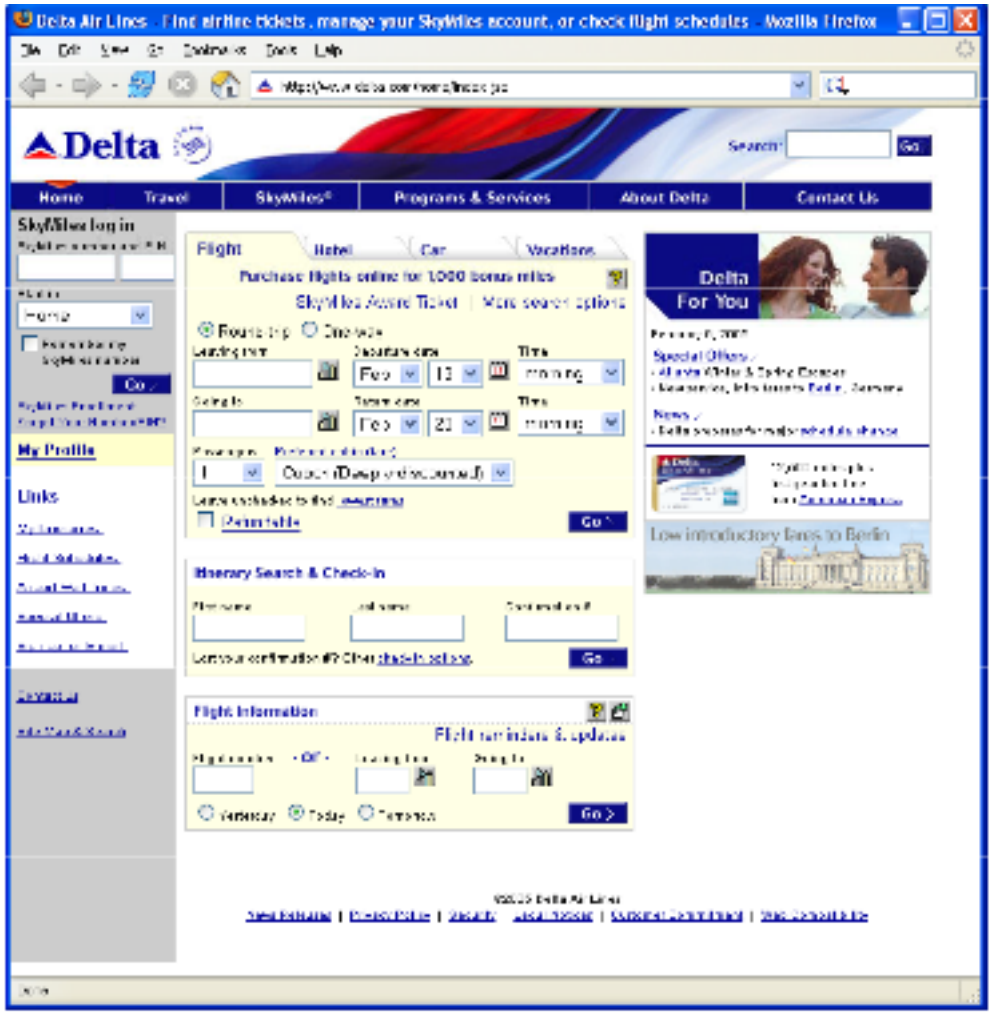
1. **Google**
   - Java (Web), C++ (indexing)
2. **Facebook**
   - PHP
3. **YouTube**
   - Flash, Python, Java
4. **Yahoo**
   - PHP and Java
5. **Microsoft Live.com**
   - .NET
6. **Baidu**
   - Unknown
7. **Wikipedia**
   - PHP
8. **Blogger**
   - Java
9. **MSN**
   - .NET
10. **Twitter**
    - Ruby on Rails, Scala, Java

# Keywords in job posting



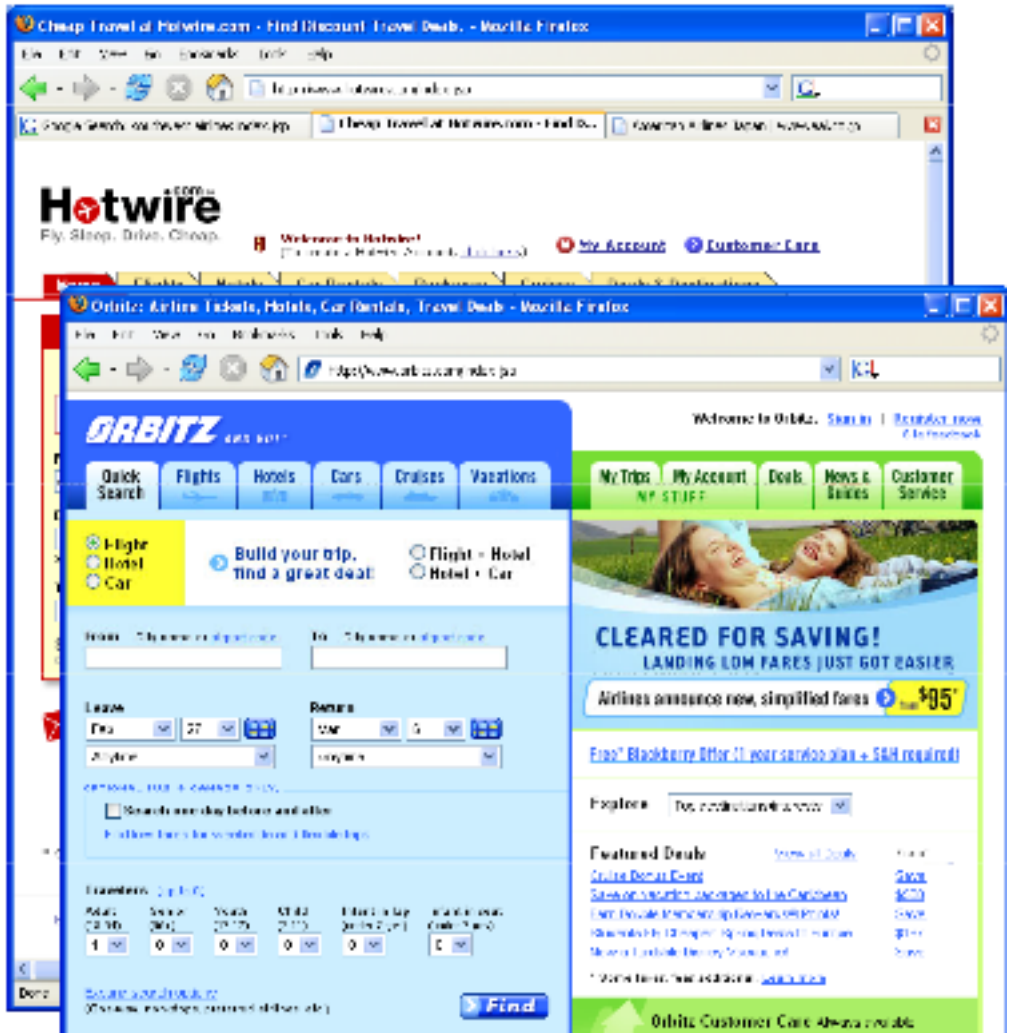
# Java Web Technology in… AIRLINES

- Delta Airlines
- United Airlines
- AirTran
- American Airlines
- British Airways
- KLM
- Air China
- Saudi Arabian Airlines
- Iceland Air



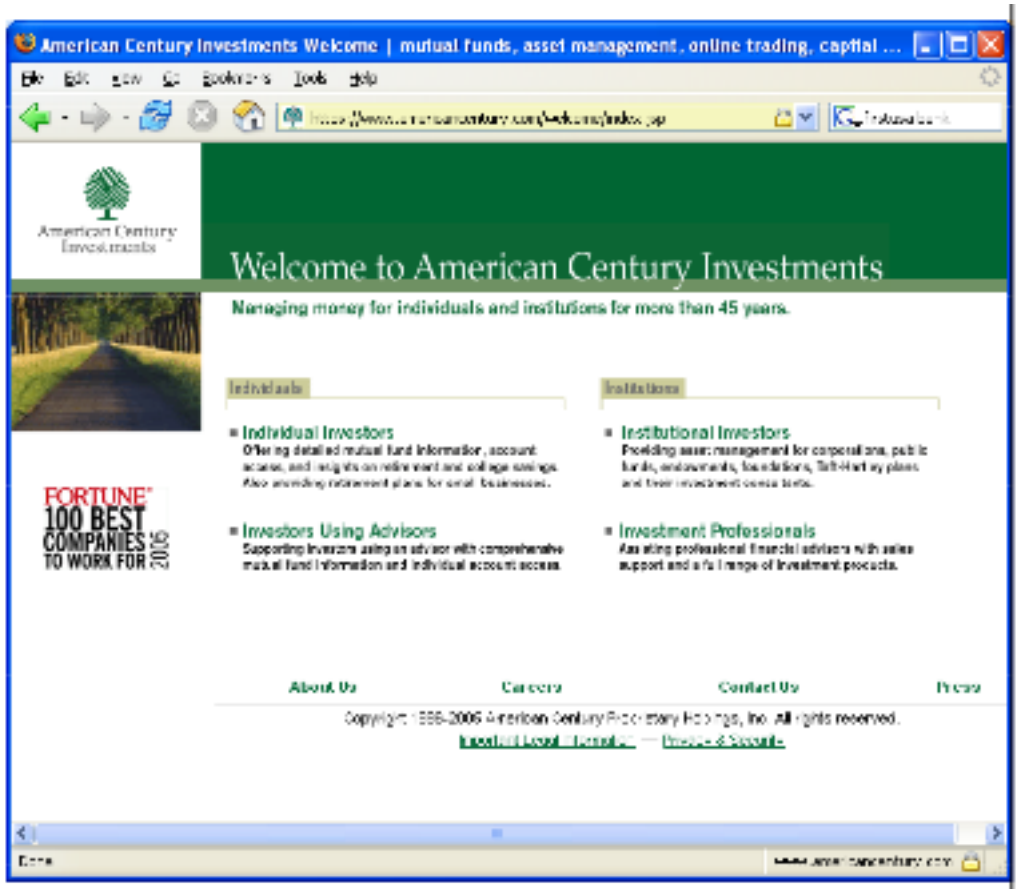
# Java Web Technology in… TRAVEL

- Travelocity.com
- Orbitz.com
- HotWire.com
- Hotels.com
- CheapTickets.com
- National Car Rental
- Avis Car Rental
- Enterprise Car Rental
- Hertz Car Rental



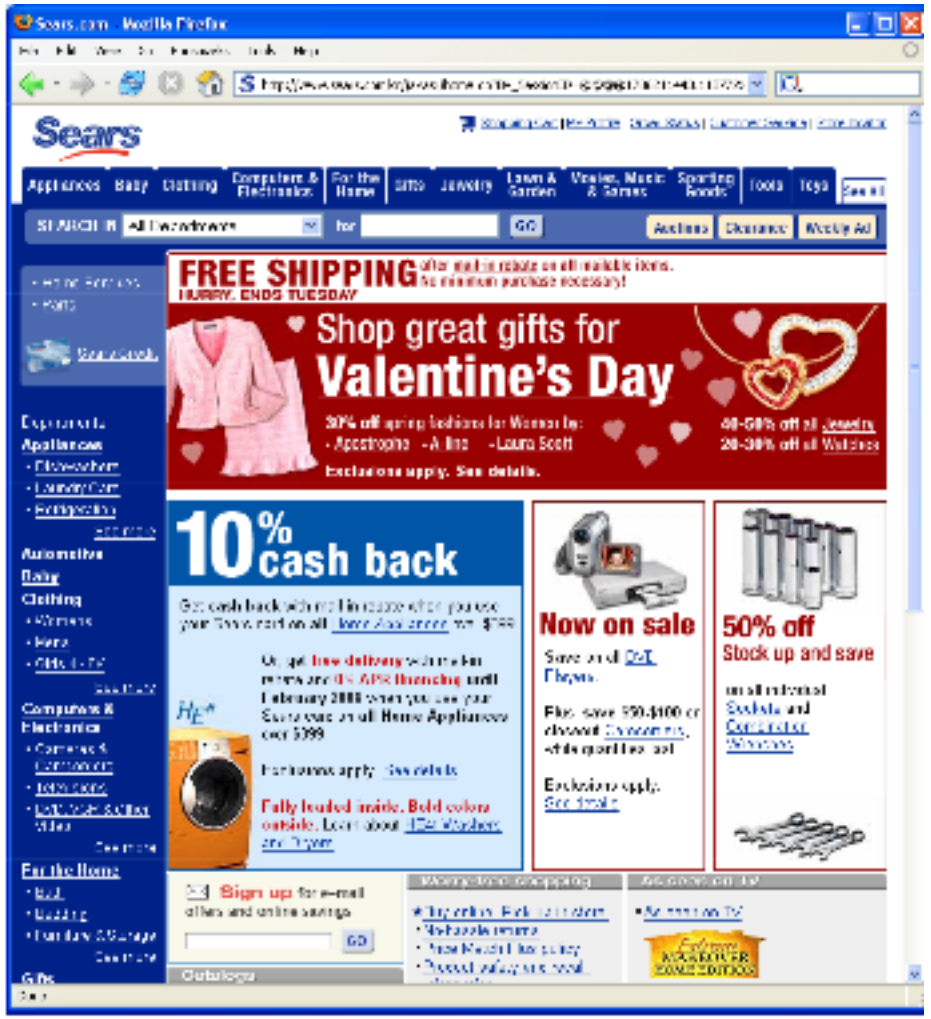
# Java Web Technology in… FINANCIAL SERVICES

- American Century
- Vanguard
- Fidelity
- NY Stock Exchange
- First USA Bank
- Royal Bank of Scotland
- Banco Popular de Puerto Rico
- Bank of America
- China Construction Bank



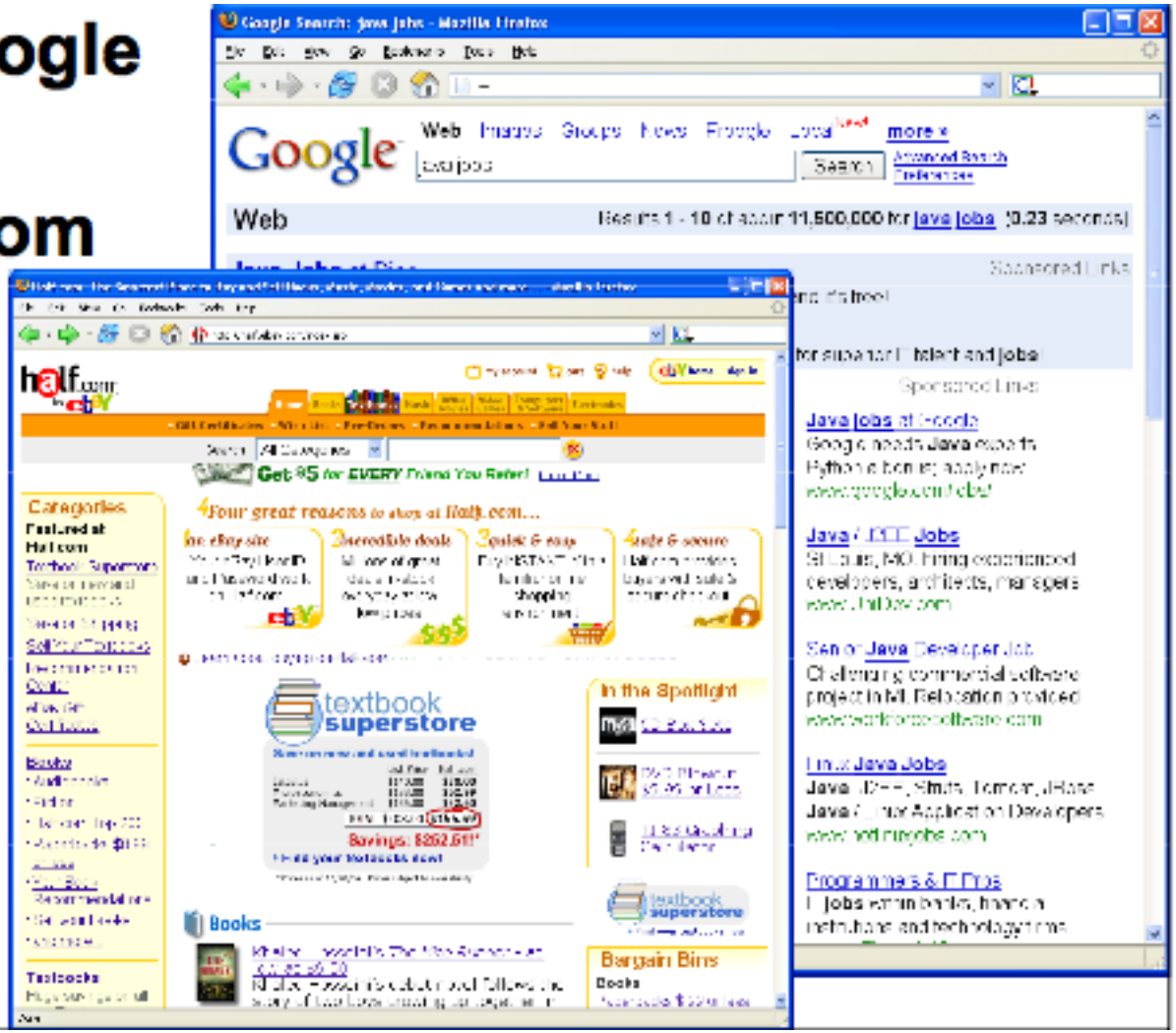
# Java Web Technology in… RETAIL

- Sears.com
- Walmart.com
- HomeDepot.com
- SamsClub.com
- Macys.com
- llbean.com
- Kohls.com
- Ikea.com
- Target.com
- Longaberger.com
- Nike.com
- CircuitCity.com



# Java Web Technology in… SEARCH PORTALS

- **Most of Google**
- **All of Ebay**
- **netscape.com**
- **excite.com**
- **dice.com**
- **hi5**
- **Paypal**



# Servlets pros and cons

- **With servlets, it is easy to**
  - Read form data
  - Read HTTP request headers
  - Set HTTP status codes and response headers
  - Use cookies and session tracking
  - Share data among servlets
  - Remember data between requests
  - Get fun, high-paying jobs
- **But, it sure is a pain to**
  - Use those println statements to generate HTML
  - Maintain that HTML

# JSP - The idea

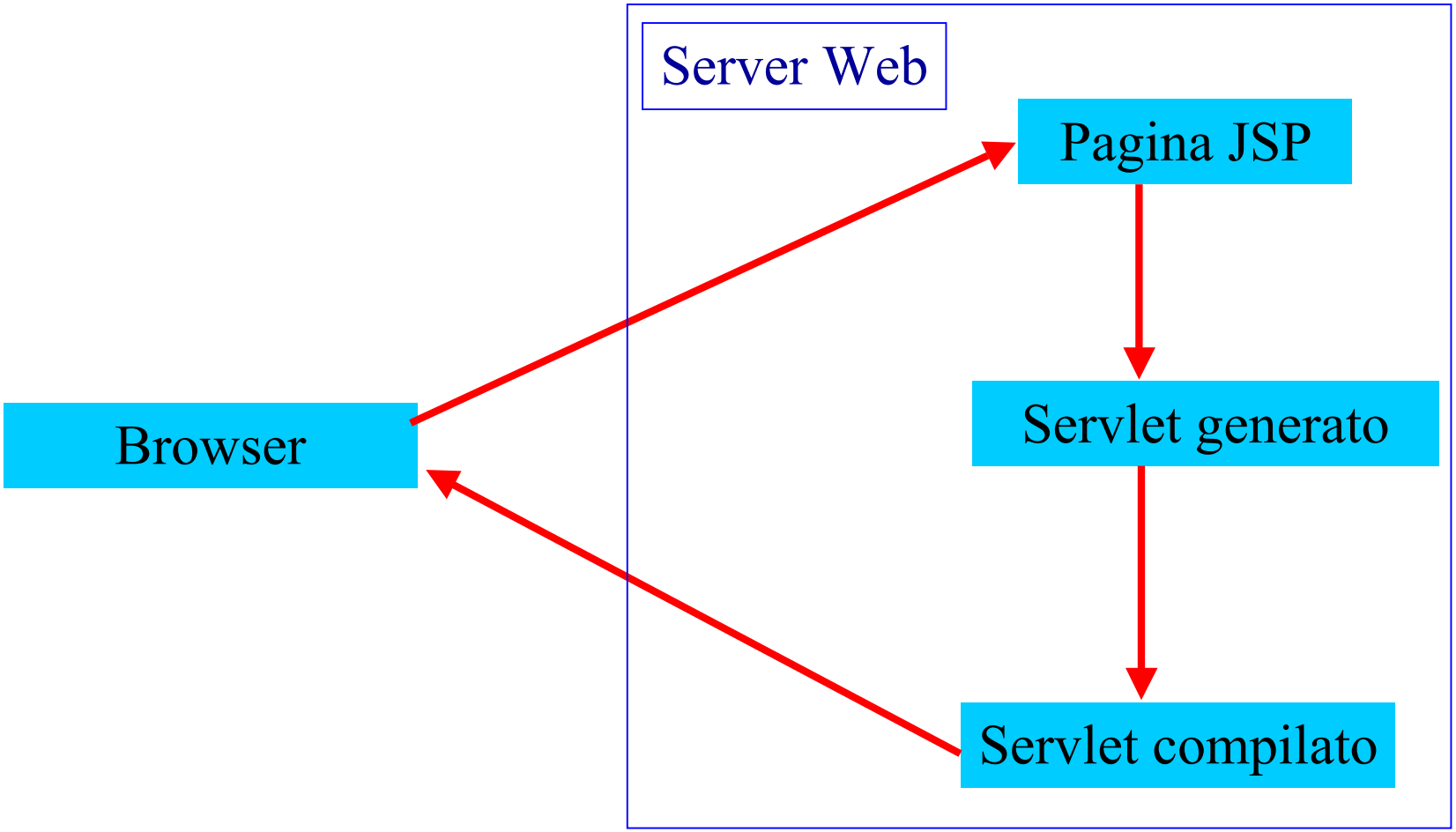
- **Idea:**
  - Use regular HTML for most of page
  - Mark servlet code with special tags
  - Entire JSP page gets translated into a servlet (once), and servlet is what actually gets invoked (for each request)
- **Example:**

```
<!DOCTYPE …>
<HTML>
<HEAD>
<TITLE>Order Confirmation</TITLE>
<LINK REL=STYLESHEET
      HREF="JSP-Styles.css"
      TYPE="text/css">
</HEAD>
<BODY>
<H2>Order Confirmation</H2>
Thanks for ordering
<I><%= request.getParameter("title") %></I>!
</BODY></HTML>
```

# JSP advantages

- **Although JSP technically can't do anything servlets can't do, JSP makes it easier to:**
  - Write HTML
  - Read and maintain the HTML
- **JSP makes it possible to:**
  - Use standard HTML tools such as Macromedia DreamWeaver or Adobe GoLive.
  - Have different members of your team do the HTML layout than do the Java programming
- **JSP encourages you to**
  - Separate the (Java) code that creates the content from the (HTML) code that presents it

# JSP Lifecycle



# JSP - MISCONCEPTIONS

- **Very common question**
  - I can't do such and such with HTML. Will JSP let me do it?
- **Why doesn't this question make sense?**
  - JSP runs entirely on server
  - It doesn't change content the client (browser) can handle
- **Similar questions**
  - How do I put a normal applet in a JSP page? Answer: send an <applet…> tag to the client
  - How do I put an image in a JSP page? Answer: send an <img…> tag to the client
  - How do I use JavaScript/Acrobat/Shockwave/Etc? Answer: send the appropriate HTML tags

# What happens when?

- **What happens at page translation time?**
  - JSP constructs get translated into servlet code.
- **What happens at request time?**
  - Servlet code gets executed. *No* interpretation of JSP occurs at request time. The original JSP page is totally ignored at request time; only the servlet that resulted from it is used.
- **When does page translation occur?**
  - Typically, the first time JSP page is accessed after it is modified. This should never happen to real user (developers should test all JSP pages they install).
  - Page translation does *not* occur for each request.

# Event table

| | Page first written | Request #1 | Request #2 | Server restarted | Request #3 | Request #4 | Page modified | Request #5 | Request #6 |
|---|---|---|---|---|---|---|---|---|---|
| JSP page translated into servlet | | Yes | No | | No | No | | Yes | No |
| Servlet compiled | | Yes | No | | No | No | | Yes | No |
| Servlet instantiated and loaded into server's memory | | Yes | No | | Yes | No | | Yes | No |
| init (or equivalent) called | | Yes | No | | Yes | No | | Yes | No |
| doGet (or equivalent) called | | Yes | Yes | | Yes | Yes | | Yes | Yes |

# JSP nuts and bolts

- Syntactic elements:
- <%@ directives %>
- <%! declarations %>
- <% scriptlets %>
- <%= expressions %>
- <jsp:actions/>
- <%-- Comment --%>
-

Implicit Objects:

- request
- response
- pageContext
- session
- application
- out
- config
- page

# JSP nuts and bolts

- Syntactic elements:
- <%@ directives %> → Interaction with the CONTAINER
- <%! declarations %> → In the initialization of the JSP
- <% scriptlets %> → In the service method
- <%= expressions %> → In the service method
- <jsp:actions/>

# Two sintaxes - why?

- **Classic syntax is not XML-compatible**
  - <%= ... %>, <% ... %>, <%! ... %> are illegal in XML
  - HTML 4 is not XML compatible either
  - So, you cannot use XML editors like XML Spy
- **You might use JSP in XML environments**
  - To build xhtml pages
  - To build regular XML documents
    - You can use classic syntax to build XML documents, but it is sometimes easier if you are working in XML to start with
      - For Web services
      - For Ajax applications
- **So, there is a second syntax**
  - Following XML rules

# xml syntax for HTML4 files

- **Many extra steps required**
  - Enclose the entire page in jsp:root
  - Enclose the HTML in CDATA sections
    - Between <![CDATA[ and ]]>
    - Because HTML 4 does not obey XML rules
  - Usually not worth the bother

# XML syntax for XHTML files

```
<?xml version="1.0" encoding="UTF-8" ?>
<html xmlns:jsp="http://java.sun.com/JSP/Page">
<jsp:output
    omit-xml-declaration="true"
    doctype-root-element="html"
    doctype-public="-//W3C//DTD XHTML 1.0 Transitional//EN"
    doctype-system="http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd" />
<jsp:directive.page contentType="text/html"/>
<head><title>Some Title</title></head>
<body bgcolor="#fdf5e6">
Body
</body></html>
```

The jsp namespace is required if you use jsp:blah commands. You can use other namespaces for other custom tag libraries.

Needed because of Internet Explorer bug where xhtml pages that have the XML declaration at the top run in quirks mode.

Builds DOCTYPE line.

For JSP pages in XML syntax, default content type is text/xml.

Normal xhtml content, plus JSP commands that use jsp:blah syntax, plus JSP custom tag libraries.

# JSP expressions

- **Format**
  - <%= Java Expression %>
- **Result**
  - Expression evaluated, converted to String, and placed into HTML page at the place it occurred in JSP page
  - That is, expression placed in _jspService inside out.print
- **Examples**
  - Current time: <%= new java.util.Date() %>
  - Your hostname: <%= request.getRemoteHost() %>
- **XML-compatible syntax**
  - <jsp:expression>Java Expression</jsp:expression>
  - You cannot mix versions within a single page. You must use XML for *entire* page if you use jsp:expression.
    - See slides at end of this lecture

# How is it translated?

- **Original JSP**

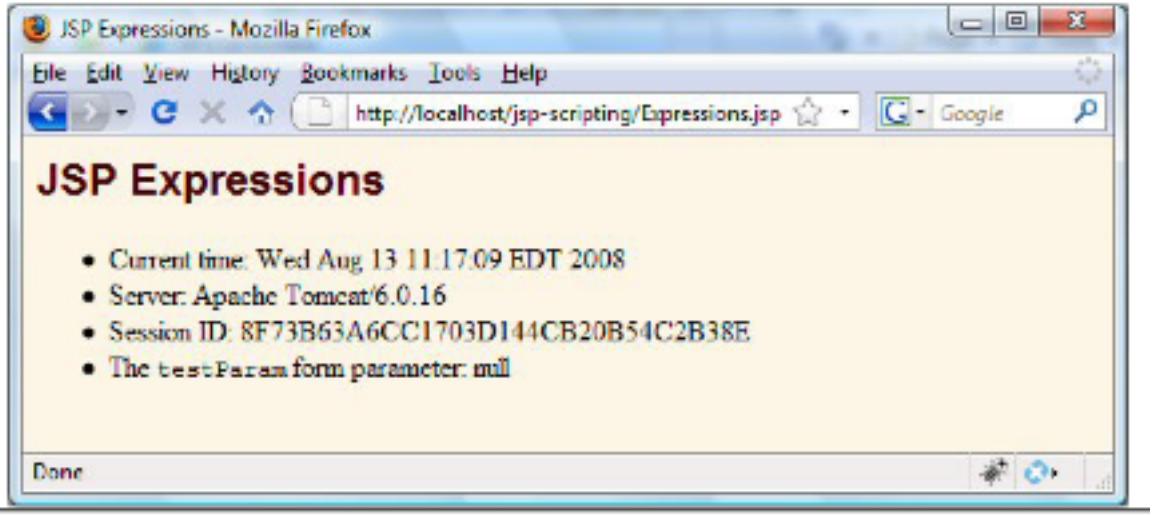
```
<H1>A Random Number</H1>
<%= Math.random() %>
```

- **Representative resulting servlet code**

```
public void _jspService(HttpServletRequest request,
                        HttpServletResponse response)
    throws ServletException, IOException {
  response.setContentType("text/html");
  HttpSession session = request.getSession();
  JspWriter out = response.getWriter();
  out.println("<H1>A Random Number</H1>");
  out.println(Math.random());
  ...
}
```

# Example

```
…<BODY>
<H2>JSP Expressions</H2>
<UL>
  <LI>Current time: <%= new java.util.Date() %>
  <LI>Server: <%= application.getServerInfo() %>
  <LI>Session ID: <%= session.getId() %>
  <LI>The <CODE>testParam</CODE> form parameter:
      <%= request.getParameter("testParam") %>
</UL>
</BODY></HTML>
```



# Predefined variables

- **request**
  - The HttpServletRequest (1st argument to service/doGet)
- **response**
  - The HttpServletResponse (2nd arg to service/doGet)
- **out**
  - The Writer (a buffered version of type JspWriter) used to send output to the client
- **session**
  - The HttpSession associated with the request (unless disabled with the session attribute of the page directive)
- **application**
  - The ServletContext (for sharing data) as obtained via getServletContext().

# Example

```
<!DOCTYPE …>
<HTML>
<HEAD>
<TITLE>Reading Three Request Parameters</TITLE>
<LINK REL=STYLESHEET
      HREF="JSP-Styles.css"
      TYPE="text/css">
</HEAD>
<BODY>
<H1>Reading Three Request Parameters</H1>
<UL>
  <LI><B>param1</B>:
      <%= request.getParameter("param1") %>
  <LI><B>param2</B>:
      <%= request.getParameter("param2") %>
  <LI><B>param3</B>:
      <%= request.getParameter("param3") %>
</UL>
</BODY></HTML>
```

# Scope of the predefined variables

- **Problem**
  - The predefined variables (request, response, out, session, etc.) are *local* to the _jspService method. Thus, they are not available to methods defined by JSP declarations or to methods in helper classes. What can you do about this?
- **Solution: pass them as arguments. E.g.**

```
public class SomeClass {
  public static void someMethod(HttpSession s) {
    doSomethingWith(s);
  }
}
...
<% somePackage.SomeClass.someMethod(session); %>
```

# Scriptlets

- **Format**
  - <% Java Code %>
- **Result**
  - Code is inserted verbatim into servlet's _jspService
- **Example**
  - <%
    String queryData = request.getQueryString();
    out.println("Attached GET data: " + queryData);
    %>
  - <% response.setContentType("text/plain"); %>
- **XML-compatible syntax**
  - <jsp:scriptlet>Java Code</jsp:scriptlet>

# How gets is translated?

- **Original JSP**

```
<H2>foo</H2>
<%= bar() %>
<% baz(); %>
```

- **Representative resulting servlet code**

```
public void _jspService(HttpServletRequest request,
                        HttpServletResponse response)
    throws ServletException, IOException {
  response.setContentType("text/html");
  HttpSession session = request.getSession();
  JspWriter out = response.getWriter();
  out.println("<H2>foo</H2>");
  out.println(bar());
  baz();
  ...
}
```

# Example

```
<!DOCTYPE …>
<HTML>
<HEAD>
  <TITLE>Color Testing</TITLE>
</HEAD>
<%
String bgColor = request.getParameter("bgColor");
if ((bgColor == null)||(bgColor.trim().equals(""))){
  bgColor = "WHITE";
}
%>
<BODY BGCOLOR="<%= bgColor %>">
<H2 ALIGN="CENTER">Testing a Background of
"<%= bgColor %>".</H2>
</BODY></HTML>
```

# A scriptlet does NOT need to be a complete Java expression

- **Point**
  - Scriplets are inserted into servlet exactly as written
  - Need not be complete Java expressions
  - Complete expressions are usually clearer and easier to maintain, however
- **Example**
  - ```
    <% if (Math.random() < 0.5) { %>
    Have a <B>nice</B> day!
    <% } else { %>
    Have a <B>lousy</B> day!
    <% } %>
    ```
- **Representative result**
  - ```
    if (Math.random() < 0.5) {
      out.println("Have a <B>nice</B> day!");
    } else {
      out.println("Have a <B>lousy</B> day!");
    }
    ```

# JSP declarations

- **Format**
  - <%! Java Code %>
- **Result**
  - Code is inserted verbatim into servlet's class definition, outside of any existing methods
- **Examples**
  - <%! private int someField = 5; %>
  - <%! private void someMethod(...) {...} %>
- **Design consideration**
  - Fields are clearly useful. For methods, it is usually better to define the method in a separate Java class.
- **XML-compatible syntax**
  - <jsp:declaration>Java Code</jsp:declaration>

# How gets it translated? 1

- **Original JSP**

```
<H1>Some Heading</H1>
<%!
  private String randomHeading() {
    return("<H2>" + Math.random() + "</H2>");
  }
%>
<%= randomHeading() %>
```

- **Better alternative:**
  - Make randomHeading a static method in a separate class

# How gets it translated - 2

- **Possible resulting servlet code**

```
public class xxxx implements HttpJspPage {
 private String randomHeading() {
   return("<H2>" + Math.random() + "</H2>");
 }

 public void _jspService(HttpServletRequest request,
                         HttpServletResponse response)
     throws ServletException, IOException {
   response.setContentType("text/html");
   HttpSession session = request.getSession();
   JspWriter out = response.getWriter();
   out.println("<H1>Some Heading</H1>");
   out.println(randomHeading());
   ...
 } ...
}
```

# Example

```
<!DOCTYPE …>
<HTML>
<HEAD>
<TITLE>JSP Declarations</TITLE>
<LINK REL=STYLESHEET
      HREF="JSP-Styles.css"
      TYPE="text/css">
</HEAD>
<BODY>
<H1>JSP Declarations</H1>
<%! private int accessCount = 0; %>
<H2>Accesses to page since server reboot:
<%= ++accessCount %></H2>
</BODY></HTML>
```

# jspInit - jspDestroy

- **JSP pages, like regular servlets, sometimes want to use init and destroy**
- **Problem: the servlet that gets built from the JSP page might already use init and destroy**
  - Overriding them would cause problems.
  - Thus, it is illegal to use JSP declarations to declare init or destroy.
- **Solution: use jspInit and jspDestroy.**
  - The auto-generated servlet is guaranteed to call these methods from init and destroy, but the standard versions of jspInit and jspDestroy are empty (placeholders for you to override).

# Let's recap…

- **JSP Expressions**
  - Format: <%= expression %>
  - Wrapped in out.print and inserted into _jspService
- **JSP Scriptlets**
  - Format: <% code %>
  - Inserted verbatim into the servlet's _jspService method
- **JSP Declarations**
  - Format: <%! code %>
  - Inserted verbatim into the body of the servlet class
- **Predefined variables**
  - request, response, out, session, application
- **Limit the Java code that is directly in page**
  - Use helper classes, beans, servlet/JSP combo (MVC), JSP expression language, custom tags
- **XML Syntax**
  - There is alternative JSP syntax that is sometimes useful when generating XML-compliant documents, probably for Ajax apps.
    - But is more trouble than it is worth for most HTML applications

# import page directive

- **Format**
  - <%@ page import="package.class" %>
  - <%@ page import="package.class1,...,package.classN" %>
- **Purpose**
  - Generate import statements at top of servlet definition
- **Notes**
  - Although JSP pages can be almost anywhere on server, classes used by JSP pages must be in normal servlet dirs
  - E.g.:
    .../WEB-INF/classes or
    …/WEB-INF/classes/*directoryMatchingPackage*
    - *Always* use packages for utilities that will be used by JSP!

# contentType -pageEncoding

- **Format**
  - <%@ page contentType="MIME-Type" %>
  - <%@ page contentType="MIME-Type;
    charset=Character-Set" %>
  - <%@ page pageEncoding="Character-Set" %>
- **Purpose**
  - Specify the MIME type of the page generated by the servlet that results from the JSP page

# Session

- **Format**
  - <%@ page session="true" %> <%-- Default --%>
  - <%@ page session="false" %>
- **Purpose**
  - To designate that page not be part of a session
- **Notes**
  - By default, it is part of a session
  - Saves memory on server if you have a high-traffic site

# Error page

- **Format**
  - <%@ page errorPage="Relative URL" %>
- **Purpose**
  - Specifies a JSP page that should process any exceptions thrown but not caught in the current page
- **Notes**
  - The exception thrown will be automatically available to the designated error page by means of the "exception" variable
  - The web.xml file lets you specify *application-wide* error pages that apply whenever certain exceptions or certain HTTP status codes result.
    - The errorPage attribute is for *page-specific* error pages

# isErrorPage

- **Format**
  - <%@ page isErrorPage="true" %>
  - <%@ page isErrorPage="false" %> <%-- Default --%>
- **Purpose**
  - Indicates whether or not the current page can act as the error page for another JSP page
- **Notes**
  - A new predefined variable called exception is created and accessible from error pages
  - Use this for emergency backup only; explicitly handle as many exceptions as possible
    - Don't forget to always check query data for missing or malformed values

# isThreadSafe

- **Format**
  - <%@ page isThreadSafe="true" %> <%-- Default --%>
  - <%@ page isThreadSafe="false" %>
- **Purpose**
  - To tell the system when your code is not threadsafe, so that the system can prevent concurrent access
    - Normally tells the servlet to implement SingleThreadModel

Don't you be lazy….

# What's wrong here?

```
<%! private int idNum = 0; %>
<%
String userID = "userID" + idNum;
out.println("Your ID is " + userID + ".");
idNum = idNum + 1;
%>
```

# Should you use "isThreadSafe" ?

- **No! It is not needed. Synchronize normally:**

```
<%! private int idNum = 0; %>
<%
synchronized(this) {
  String userID = "userID" + idNum;
  out.println("Your ID is " + userID + ".");
  idNum = idNum + 1;
}
%>
```

- **Better performance in high-traffic environments**

# @include

- **Format**
  - <%@ include file="Relative address" %>
- **Purpose**
  - To reuse JSP content in multiple pages, *where JSP content affects main page*
- **Notes**
  - Servers are not required to detect changes to the included file, and in practice they don't.
  - Thus, you need to change the JSP files whenever the included file changes.
  - You can use OS-specific mechanisms such as the Unix "touch" command, or
    - <%-- Navbar.jsp modified 4/1/09 --%>
      <%@ include file="Navbar.jsp" %>

# jsp:include

- **Format**
  - <jsp:include page="Relative address" />
- **Purpose**
  - To reuse JSP, HTML, or plain text content
  - To permit updates to the included content without changing the main JSP page(s)
- **Notes**
  - JSP content cannot affect main page: only *output* of included JSP page is used
  - Don't forget that trailing slash
  - Relative URLs that starts with slashes are interpreted relative to the Web app, not relative to the server root.
  - You are permitted to include files from WEB-INF

# jsp:include vs @ include

| | jsp:include | <%@ include ...%> |
|---|---|---|
| Basic syntax | <jsp:include page="..." /> | <%@ include file="..." %> |
| When inclusion occurs | Request time | Page translation time |
| What is included | Output of page | Contents of file |
| Number of resulting servlets | Two | One |
| Can included page set response headers that affect the main page? | No | Yes |
| Can included page define fields or methods that main page uses? | No | Yes |
| Does main page need to be updated when included page changes? | No | Yes |

# Augmenting request params

- **Code**

```
<jsp:include page="/fragments/StandardHeading.jsp">
  <jsp:param name="bgColor" value="YELLOW" />
</jsp:include>
```

- **URL**
  - http://host/path/MainPage.jsp?fgColor=RED
- **Main page**
  - fgColor: RED
  - bgColor: null
    - Regardless of whether you check before or after inclusion
- **Included page**
  - fgColor: RED
  - bgColor: YELLOW

# Static pages

To let Tomcat serve static pages, we must define a “Web Application”.

That is, in the Tomcat Document Root (by default *$CATALINA_HOME/webapps/*) we must create a folder named after our Web Application (e.g. myApp).

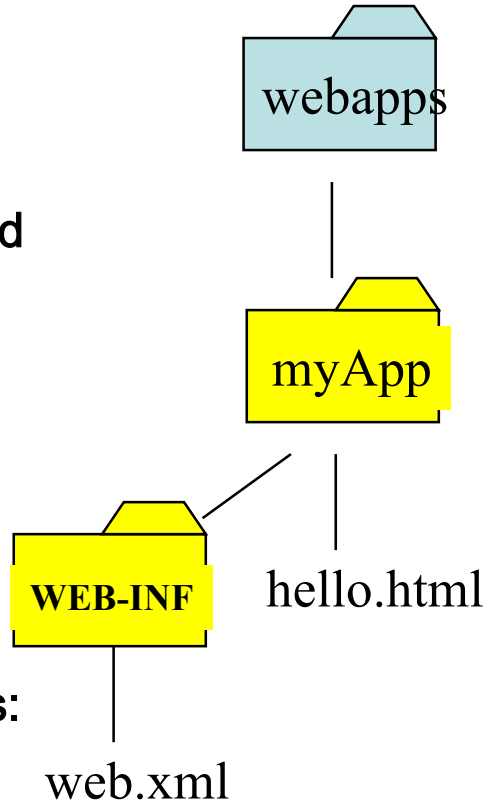
In that “myApp” folder, we MUST create a WEB-INF folder (that can be empy).

In the myApp folder we can then depost the static html files.

On our Tomcat server, the URL for the hello.html file becomes:

http://*machine/port*/myApp/hello.html

To actually see the webapp, we might have to restart Tomcat



# JSP pages

To let Tomcat serve JSP pages, we follow the same procedure that we described for static pages.

In the myApp folder we can depost the JSP files.

On our Tomcat server, the URL for the hello.jsp file becomes:

http://*machine/port*/myApp/hello.jsp

The WEB-INF directory is still empty.

To actually see the webapp, you might have to restart Tomcat (depending on the version you have)

The same web.xml file as in the static case must be provided.