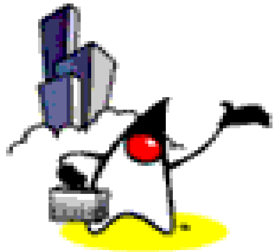
JSP Basics I

Sang Shin
Michèle Garoche
www.JPassion.com
“Learn with JPassion!”

Agenda

- JSP in big picture of Java EE
- Introduction to JSP
- Life-cycle of JSP page
- Steps for developing JSP-based Web application
- Dynamic contents generation techniques in JSP
- Invoking Java code using JSP scripting elements
- JavaBeans for JSP
- Error handling

What is JSP?



What is JSP page?

- A text-based document capable of returning dynamic content to a client browser
  - It looks and feels like a HTML page
- Contains both static and dynamic content
  - Static content: HTML, XML
  - Dynamic content: programming code, and JavaBeans, custom tags

Why JSP Technology?

- Enables separation of business logic from presentation
  - Presentation is in the form of HTML or XML/XSLT
  - Business logic is implemented as Java Beans or custom tags
  - Better maintainability, reusability
- Extensible via custom tags
- Builds on Servlet technology
  - A JSP gets compiled into Servlet before deployment

JSP Sample Code

```
<html>
   Hello World!
  <br>
  <jsp:useBean id="clock"
               class=“calendar.JspCalendar” />
   Today is
  <ul>
  <li>Day of month: <%= clock.getDayOfMonth() %>
  <li>Year: <%= clock.getYear() %>
  </ul>
</html>
```

Servlets and JSP - Comparison

Servlets

- HTML code in Java
- Any form of Data
- Not easy to author a web page

JSP

- Java-like code in HTML
- Structured Text
- Very easy to author a web page (compared to Servlet)
- Code is compiled into a servlet

JSP Benefits over Servlet

- Content and display logic are separated
- Simplify development with JSP, JavaBeans and custom tags
- Supports software reuse through the use of components
- Recompile automatically when changes are made to the source file
- Easier to author web pages (compared to Servlets)
- Platform-independent

When to use Servlet over JSP

- In general, you want to use JSP over Servlet for the presentation
  - Avoid returning HTML directly from your servlets whenever possible
- However, there could be cases where you might want to use Servlet over JSP as display
  - Extend the functionality of a Web server such as supporting a new file format
  - Generate objects that do not contain HTML such as graphs or pie charts

Should I Use Servlet or JSP?

- In practice, servlet and JSP are used together
  - via MVC (Model, View, Controller) architecture
  - Servlet handles Controller
  - JSP handles View

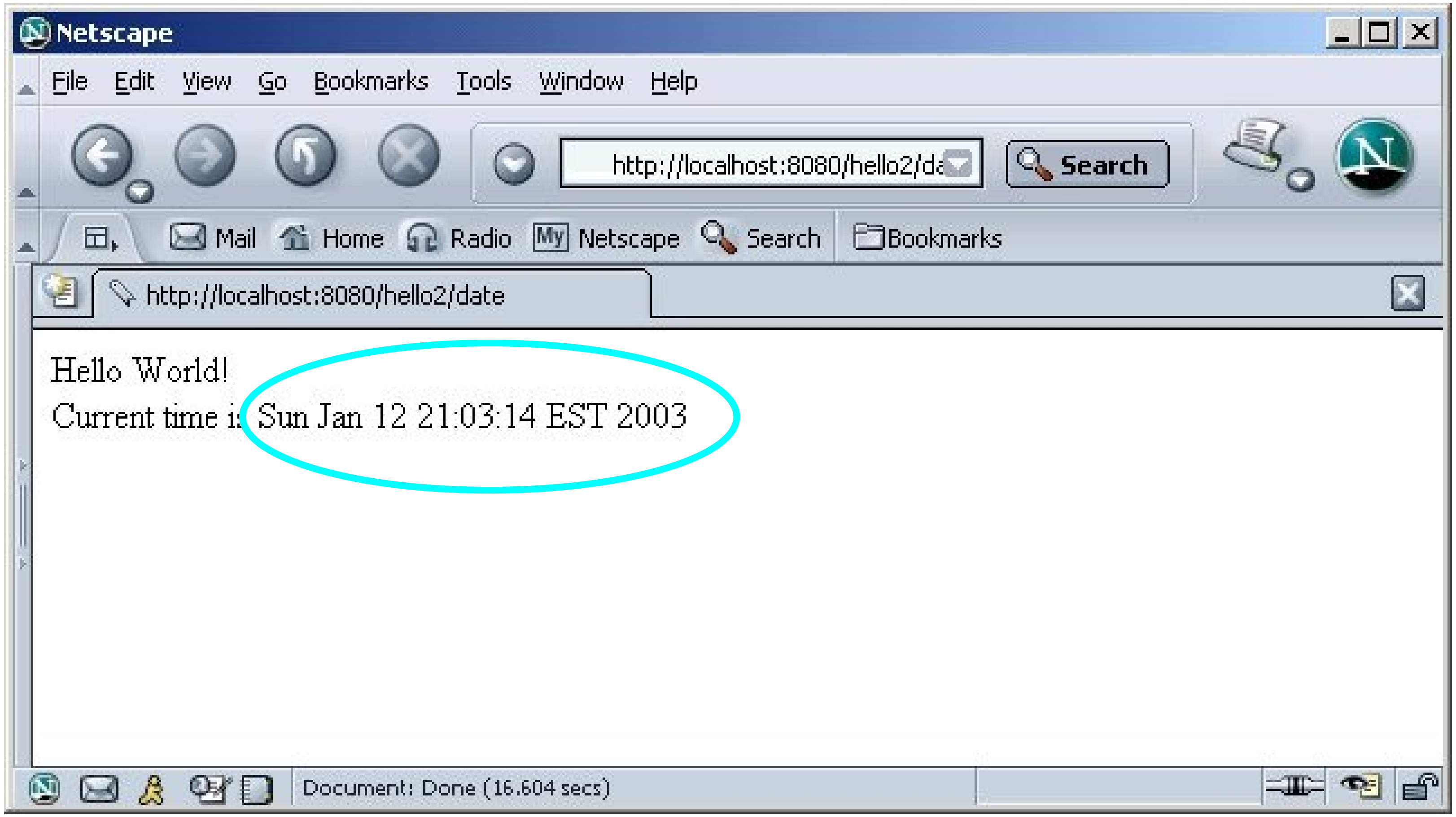
Static vs. Dynamic Contents

- Static contents
  - Typically static HTML page
  - Same display for everyone
- Dynamic contents
  - Contents is dynamically generated based on conditions
  - Conditions could be
    - User identity
    - Time of the day
    - User entered values through forms and selections
  - Examples
    - Etrade webpage customized just for you, my Yahoo

A Simple JSP Page (Blue: static, Red: Dynamic contents)

```
<html>
<body>
   Hello World!
  <br>
  Current time is <%= new java.util.Date() %>
</body>
</html>
```

Output

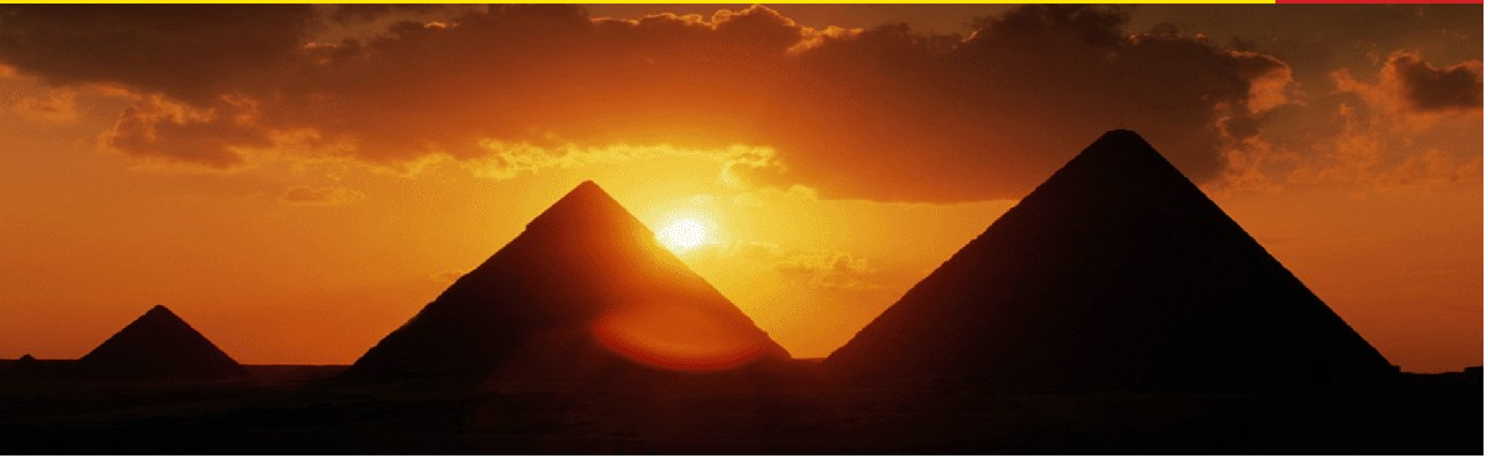

Netscape
File Edit View Go Bookmarks Tools Window Help
http://localhost:8080/hello2/da
Search
Mail Home Radio Netscape Search Bookmarks
http://localhost:8080/hello2/date

```
Hello World!
Current time is Sun Jan 12 21:03:14 EST 2003
```

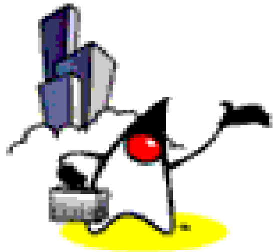
Document: Done (16.604 secs)

Servlet/JSP vs. Web Frameworks

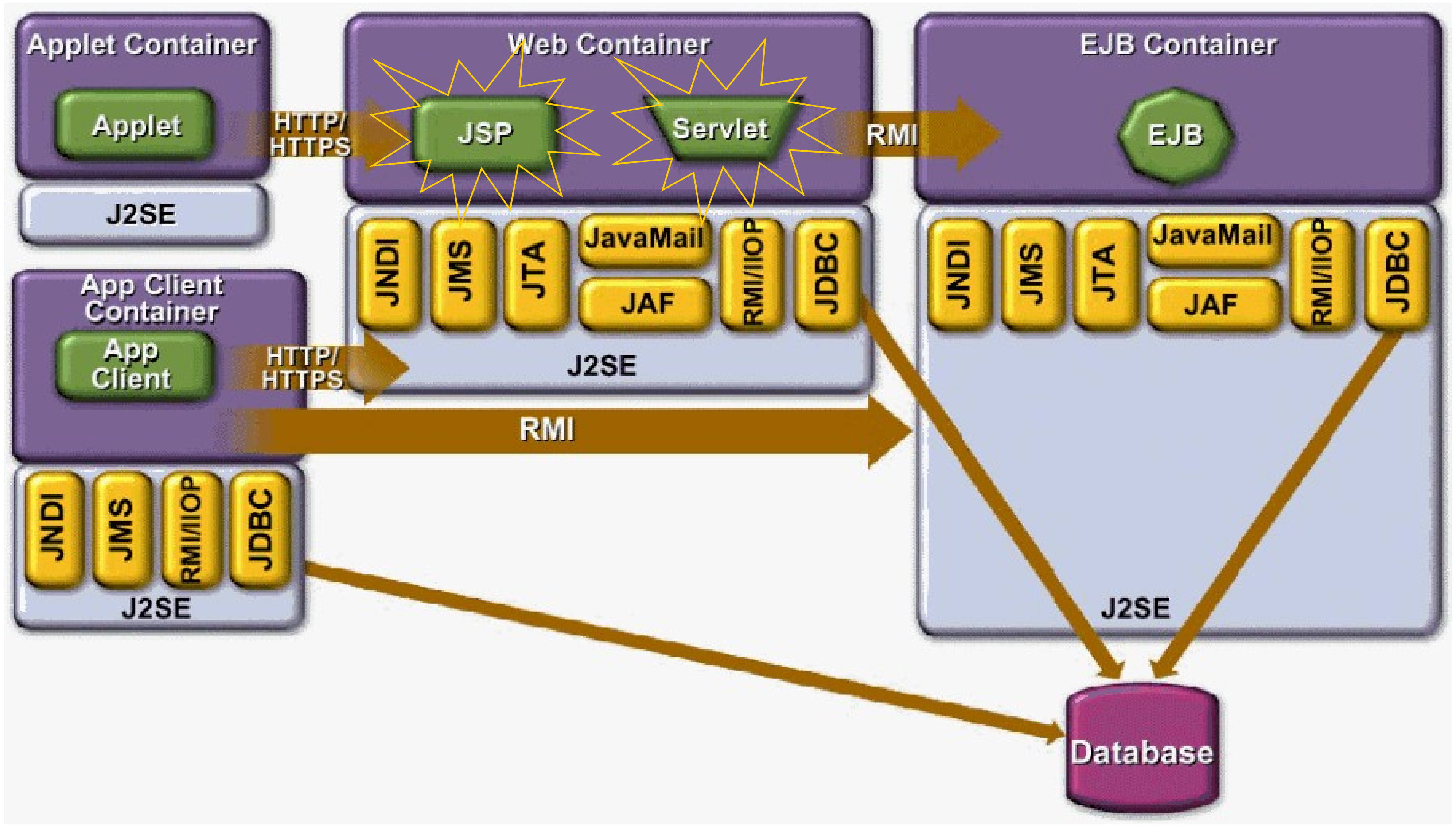
- Limitations of vanilla Servlet/JSP
  - Vanilla Servlets/JSP are considered too low-level for building real-life production-quality Web applications
  - Vanilla Servlets/JSP oo not provide common features needed for building web applications such as Dispatch framework, Data binding, validation, internationalization, etc
- So it is highly likely you will use popular MVC-based Web application frameworks, which provide extra features over vanilla Servlet/JSP
  - SpringMVC, Wicket, Tapestry, Struts, etc

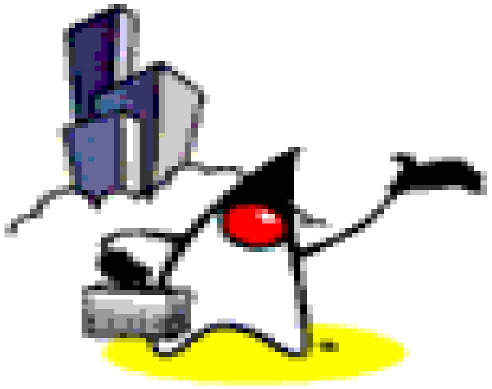


JSP in a Big Picture of Java EE



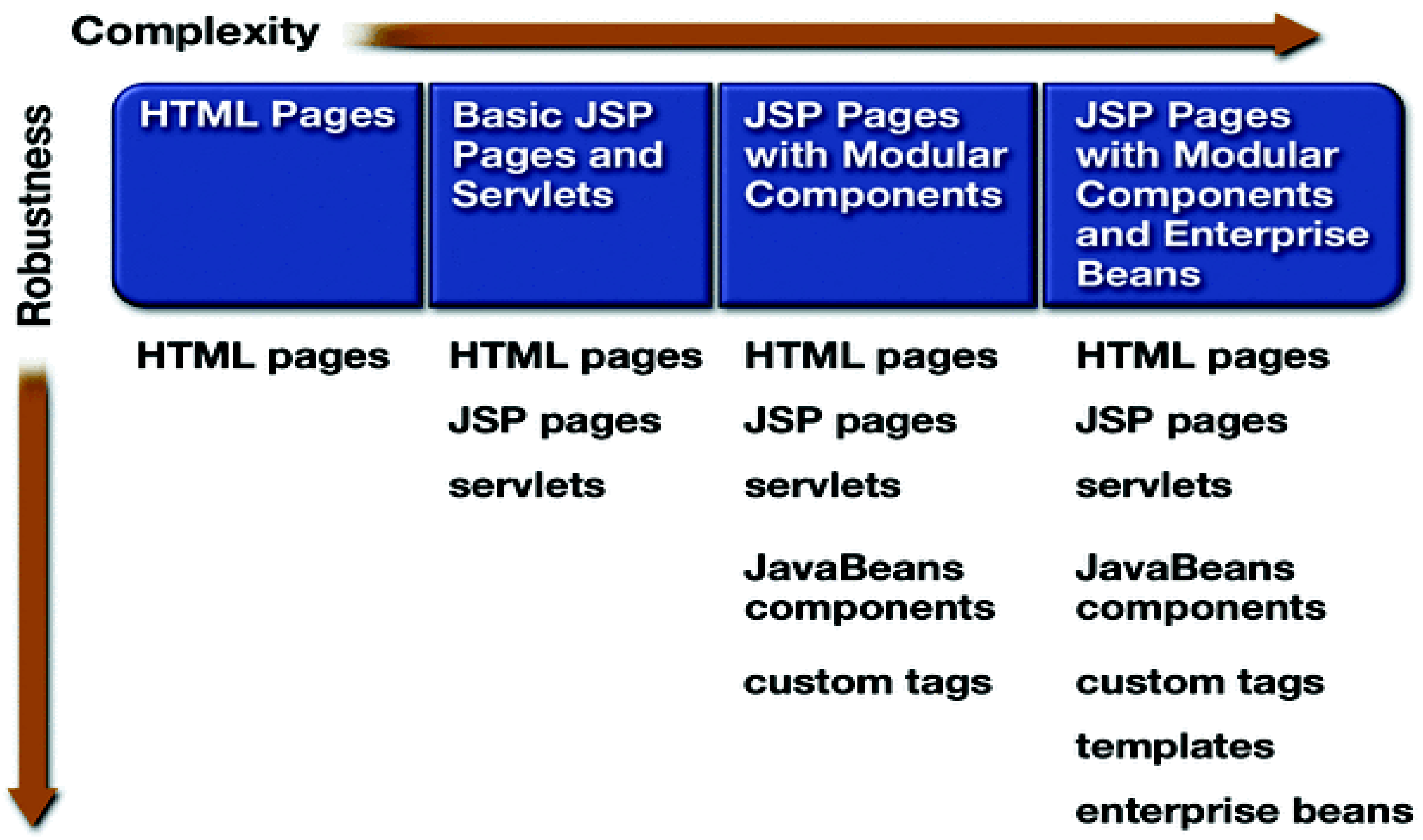
JSP & Servlet as Web Components



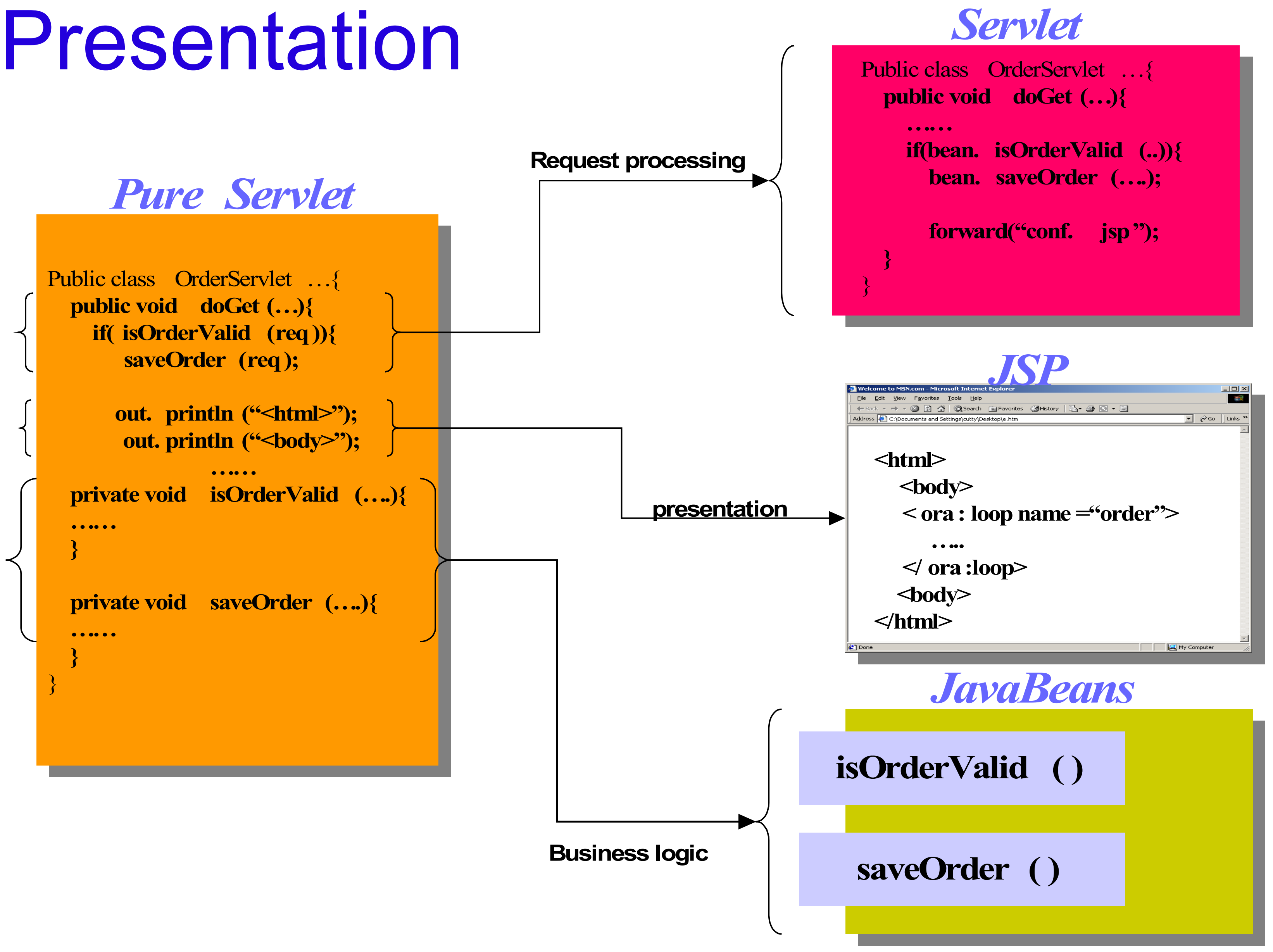


JSP Architecture

Web Application Designs



Separate Request processing From Presentation



Pure Servlet

```
Public class   OrderServlet  …{
    public void    doGet (…){
        if( isOrderValid   (req )){
            saveOrder  (req );

            out.  println  (“<html>”);
            out. println  (“<body>”);
                  ……
    private void    isOrderValid    (….){
    ……
    }

    private void    saveOrder   (….){
    ……
    }
}
```

Servlet

```
Public class   OrderServlet  …{
    public void    doGet (…){
        ……
        if(bean.  isOrderValid   (..)){
            bean.  saveOrder   (….);

            forward(“conf.    jsp ”);
    }
}
```

JSP

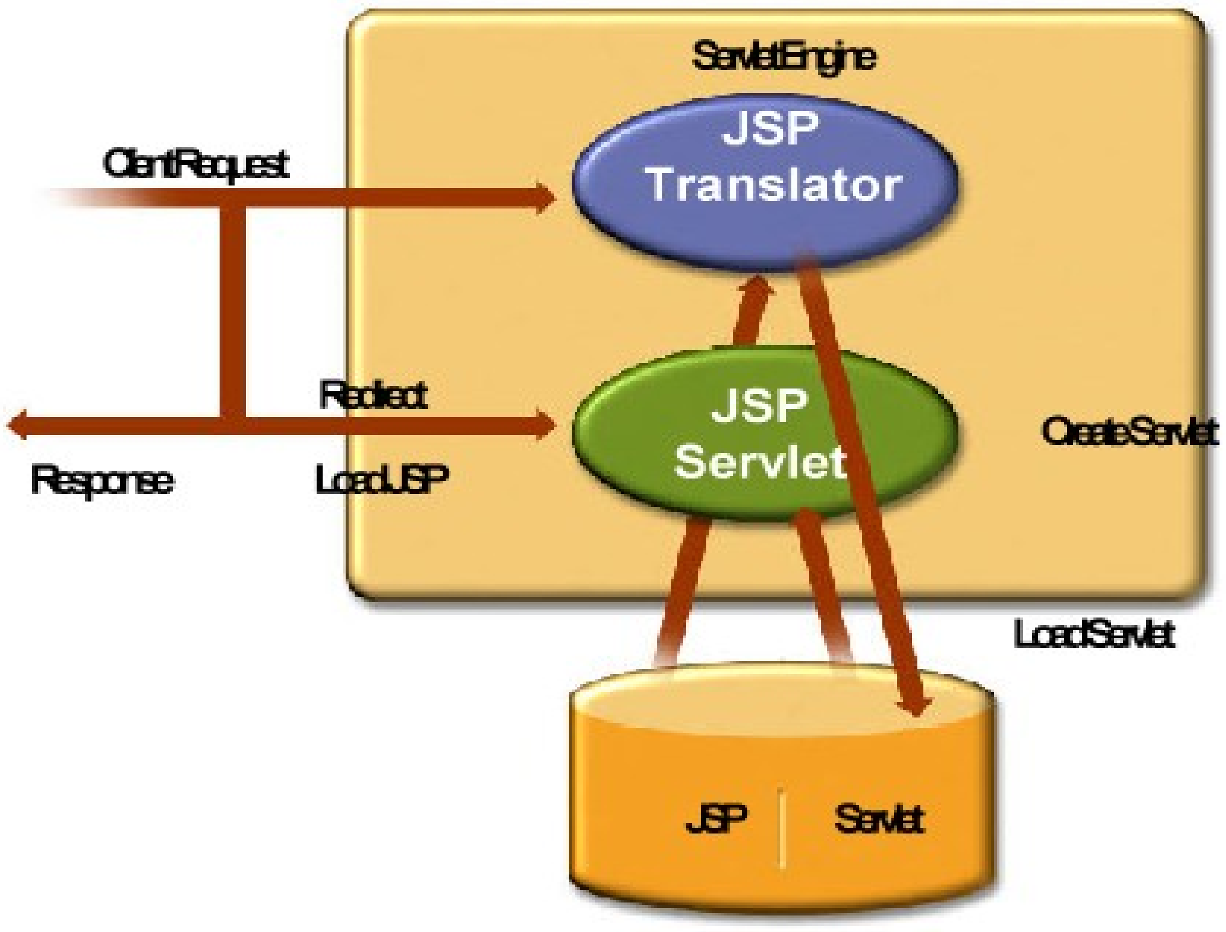
```
<html>
    <body>
    < ora : loop name =“order”>
        …..
    </ ora :loop>
    <body>
</html>
```

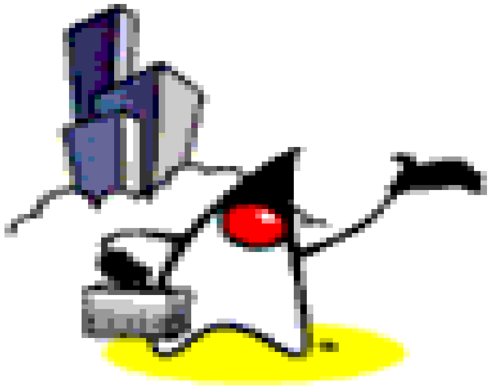
JavaBeans

isOrderValid ( )

saveOrder ( )

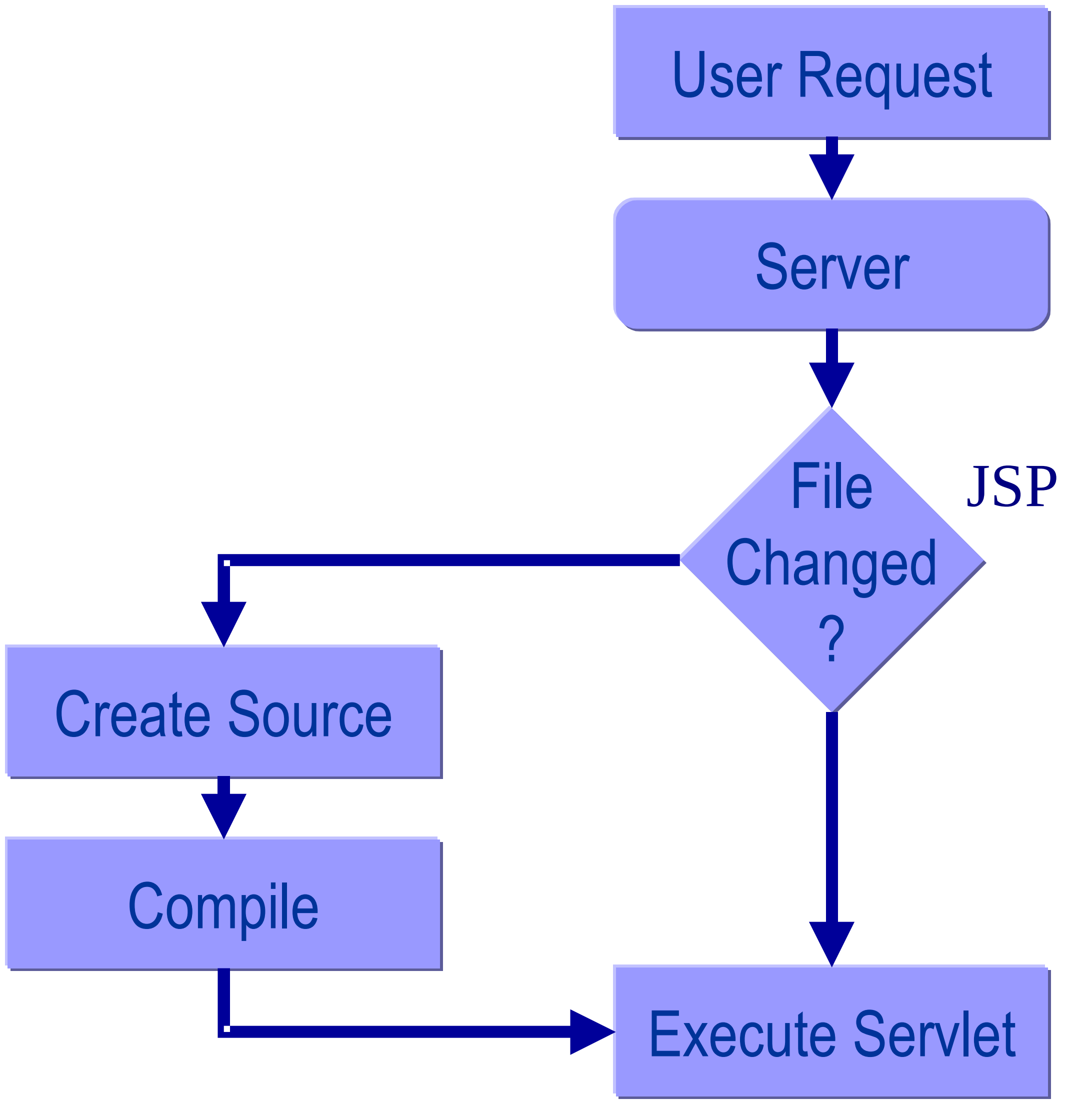
JSP Architecture





Life-Cycle of a JSP Page

How Does JSP Work?



JSP Page Lifecycle Phases

- Translation phase
- Compile phase
- Execution phase

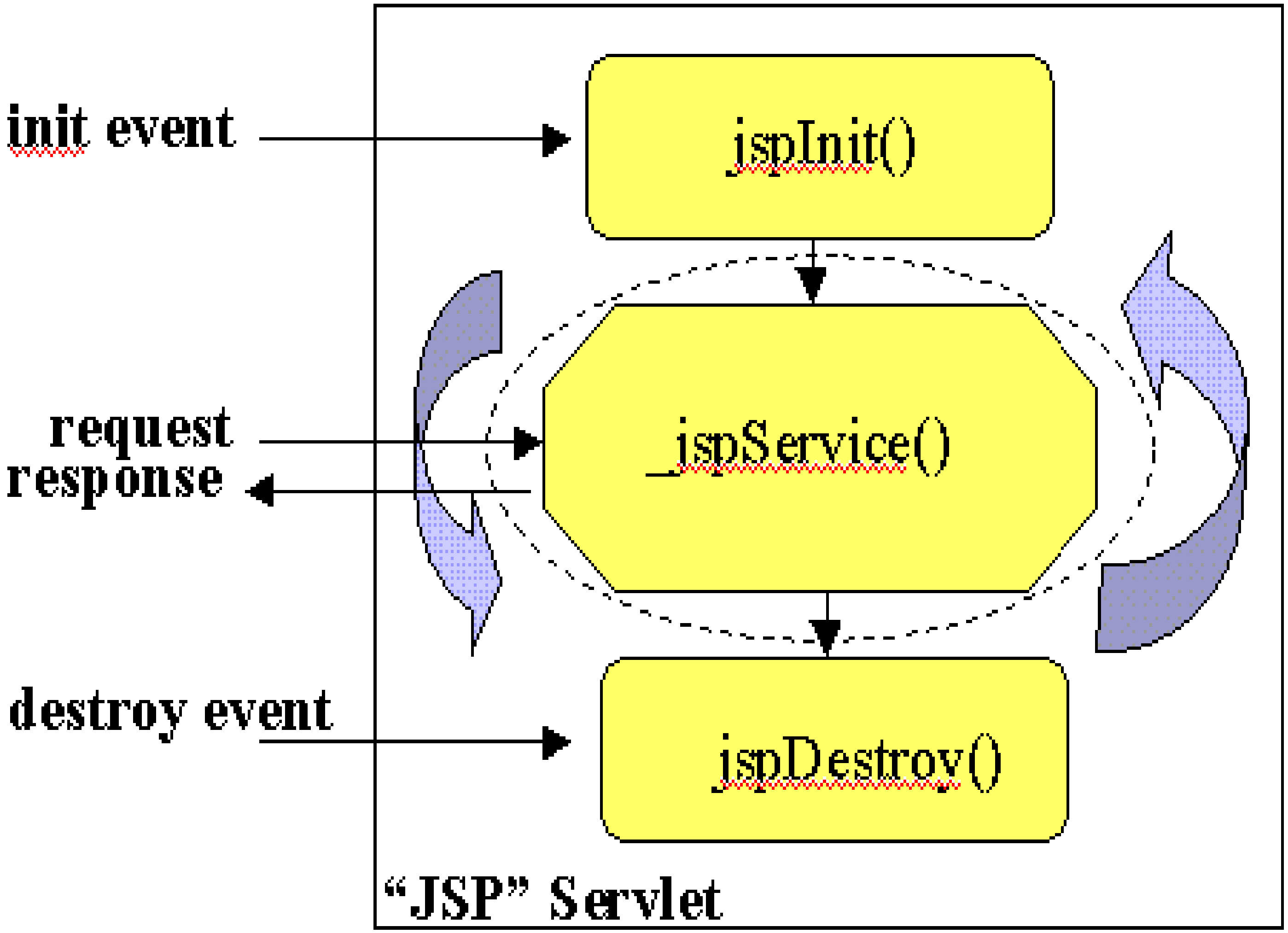
Translation/Compilation Phase

- JSP files get translated into servlet source code, which is then compiled
- Done by the container automatically
- The first time JSP page is accessed after it is deployed (or modified and redeployed)
- For JSP page “pageName”, the source code resides
  - <AppServer_HOME>/work/Standard Engine/localhost/context_root/pageName$jsp.java
  - <AppServer_HOME>/work/Standard Engine/localhost/date/index$jsp.java

Translation/Compilation Phase

- Static Template data is transformed into code that will emit data into the stream
- JSP elements (we have not covered them yet) are treated differently
  - Directives are used to control how Web container translates and executes JSP page
  - Scripting elements are inserted into JSP page's servlet class
  - Elements of the form <jsp:xxx .../> are converted into method calls to JavaBeans components

JSP Lifecycle Methods during Execution Phase



Initialization of a JSP Page

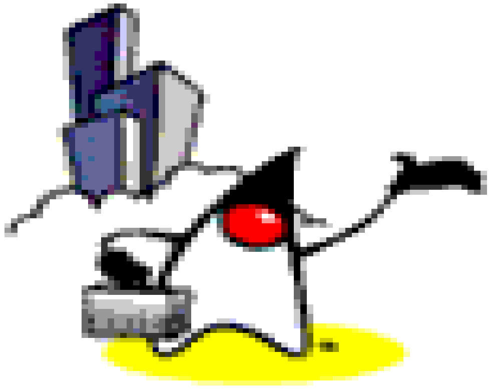
- Declare methods for performing the following tasks using JSP declaration mechanism
  - Read persistent configuration data
  - Initialize resources
  - Perform any other one-time activities by overriding jspInit() method of JspPage interface

Finalization of a JSP Page

- Declare methods for performing the following tasks using JSP declaration mechanism
  - Read persistent configuration data
  - Release resources
  - Perform any other one-time cleanup activities by overriding jspDestroy() method of JspPage interface

Example: initdestroy.jsp

```
<%@ page import="database.*" %>
<%@ page errorPage="errorpage.jsp" %>
<%--  Declare initialization and finalization methods using JSP declaration --%>
<%!

  private BookDBAO bookDBAO;
  public void jspInit() {

    // retrieve database access object, which was set once per web application
    bookDBAO =
      (BookDBAO)getServletContext().getAttribute("bookDB");
    if (bookDBAO == null)
        System.out.println("Couldn't get database.");
  }

  public void jspDestroy() {
    bookDBAO = null;
  }
%>
```



Steps for Developing JSP-based Web Application

Web Application Development and Deployment Steps

1. Write (and compile) the Web component code (Servlet or JSP) and helper classes referenced by the web component code
2. Create any static resources (for example, images or HTML pages)
3. Create deployment descriptor (web.xml)
4. Build the Web application (*.war file or deployment-ready directory)
5. Install or deploy the web application into a Web container
   - Clients (Browsers) are now ready to access them via URL

1. Write and compile the Web component code

- Create development tree structure
- Write either servlet code and/or JSP pages along with related helper code
- Create build.xml for Ant-based build (and other application life-cycle management) process

Development Tree Structure

- Keep Web application source separate from compiled files
  - facilitate iterative development
- If you are using an IDE or Maven, the default development tree structure is created for you
- Root directory
  - src: Java source of servlets and JavaBeans components
  - web: JSP pages and HTML pages, images

Example: Hello1 Example Tree Structure (before “ant build” command)

- hello1 directory (from Java EE tutorial)
  - web directory
    - duke.waving.gif
    - index.jsp
    - response.jsp
    - WEB-INF directory
      - web.xml
  - (it does not have *src* directory since this does not use any Java classes as utility classes)

2. Create any static resources

- HTML pages
  - Custom pages
  - Login pages
  - Error pages
- Image files that are used by HTML pages or JSP pages
  - Example: duke.waving.gif

3. Create deployment descriptor (web.xml)

- Deployment descriptor contains deployment time runtime instructions to the Web container
  - URN that the client uses to access the web component
- Every web application has to have it
  - Except Java EE 6 based Web application (in Java EE 6 based Web application, web.xml is optional since configuration information can be specified in the source file through annotation)

web.xml for Hello1

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<!DOCTYPE web-app PUBLIC '-//Sun Microsystems, Inc.//DTD Web
  Application 2.3//EN' 'http://java.sun.com/dtd/web-app_2_3.dtd'>

<web-app>
  <display-name>Hello2</display-name>
  <description>no description</description>
  <servlet>
    <servlet-name>greeting</servlet-name>
    <display-name>greeting</display-name>
    <description>no description</description>
    <jsp-file>/greeting.jsp</jsp-file>   <!-- what gets called -->
  </servlet>
  <servlet-mapping>
    <servlet-name>greeting</servlet-name>
    <url-pattern>/greeting</url-pattern> <!-- URL from browser -->
  </servlet-mapping>
</web-app>
```

4. Build the Web application

- Either *.WAR file or unpacked form of *.WAR file
  - Again, IDE or Maven handles all this
- Build process is made of
  - create build directory (if it is not present) and its subdirectories
  - copy *.jsp files under build directory
  - compile Java code into build/WEB-INF/classes directory
  - copy web.xml file into build/WEB-INF directory
  - copy image files into build directory

Example: Hello2 Tree Structure (after "ant build" command)

- Hello2
  - web directory
  - build directory
    - WEB-INF directory
      - classes directory
      - web.xml
    - duke.waving.gif
    - greeting.jsp
    - response.jsp

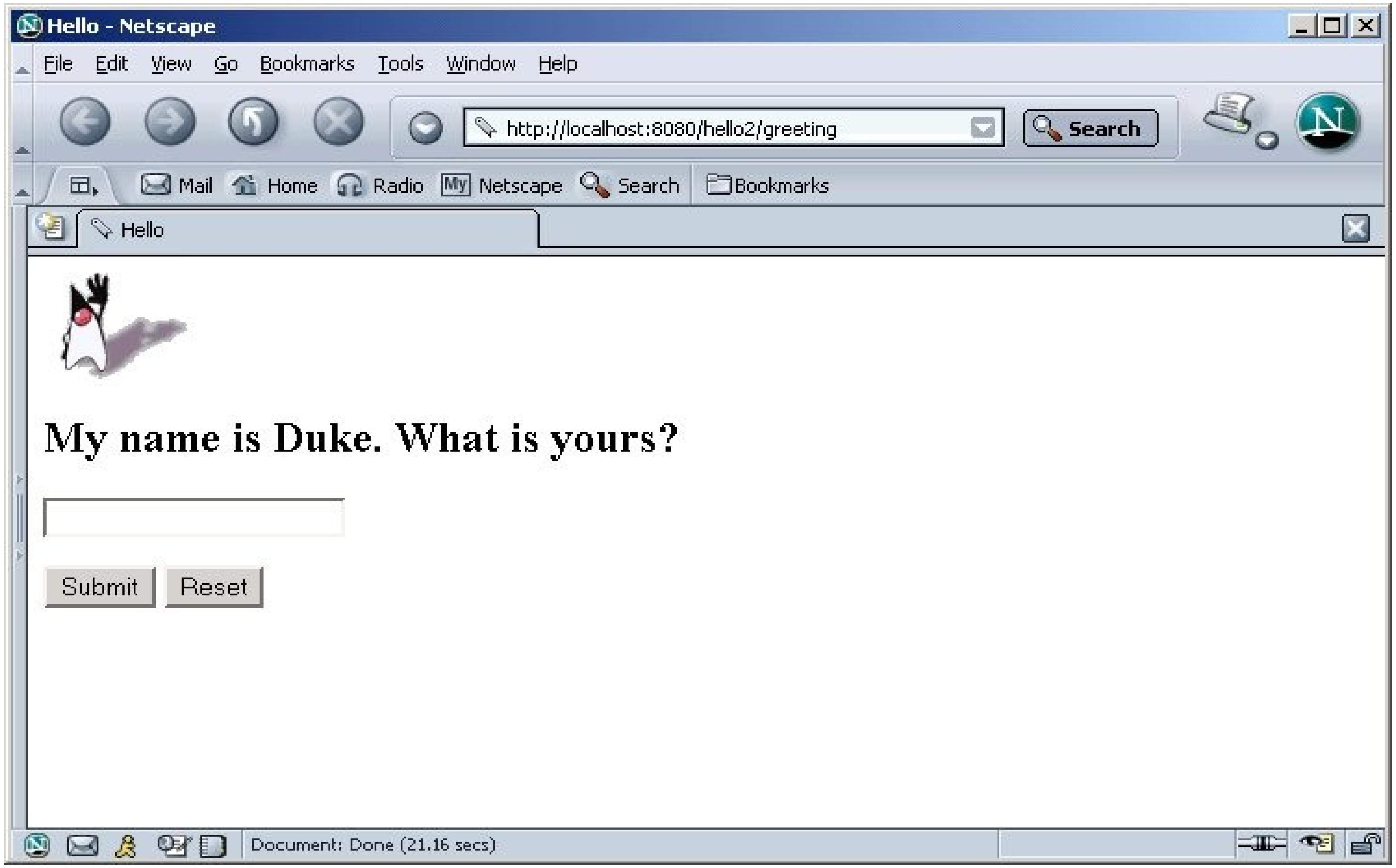
5. Install or Deploy Web application

- There are several different ways to install/deploy Web application
  - Through IDE
  - Through Maven
  - Through admin console of the server (Tomcat Manager for Tomcat or GlassFish Admin console.)
  - Most servers also support auto-deploy (you just copy the war file or exploded directory into their auto-deployment directory)

6. Perform Client Access to Web Application

- From a browser, go to URN of the Web application
  - http://localhost:8080/hello2/greeting

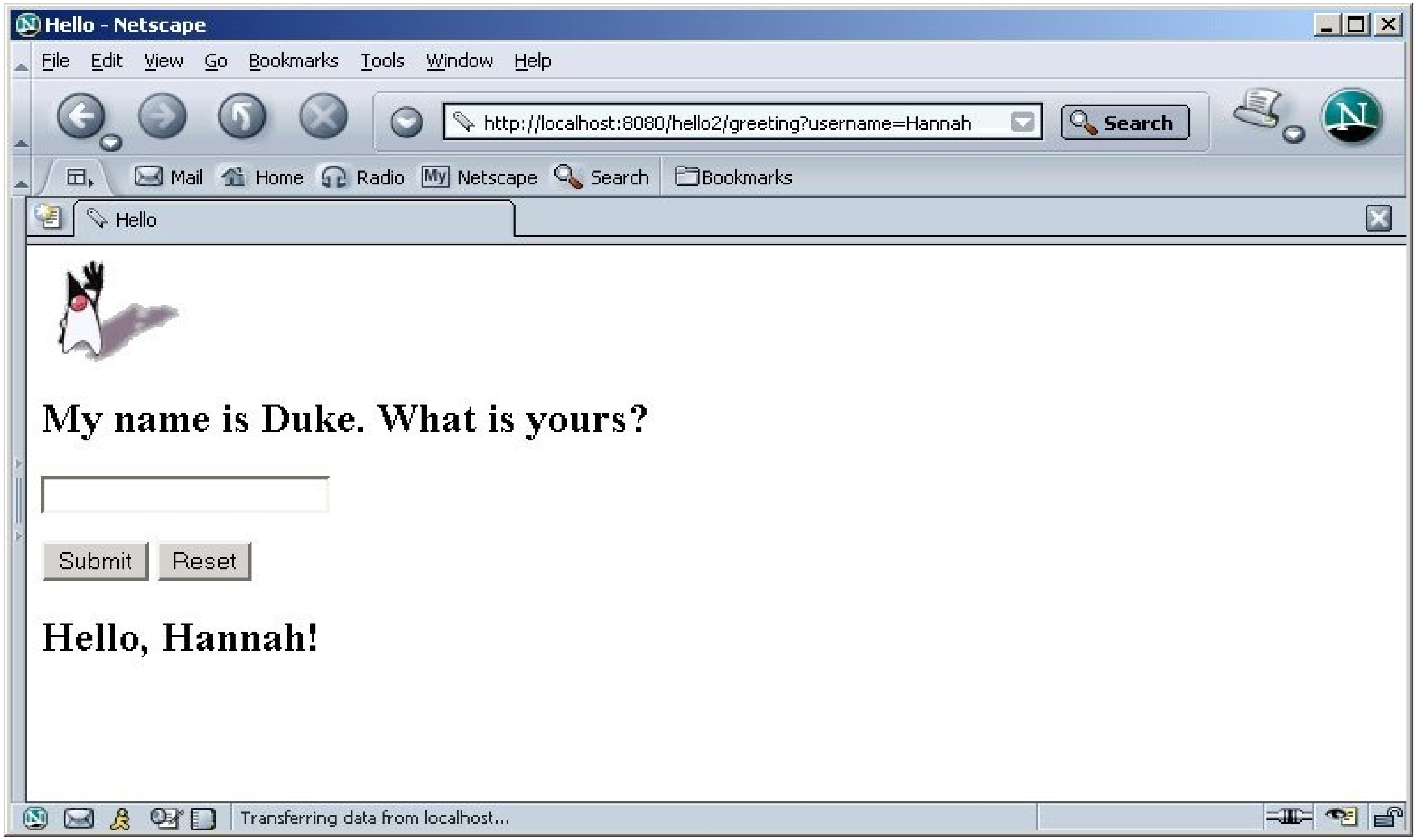
http://localhost:8080/hello2/greeting


Hello - Netscape
File Edit View Go Bookmarks Tools Window Help
http://localhost:8080/hello2/greeting
Search
Mail Home Radio Netscape Search Bookmarks
Hello

My name is Duke. What is yours?

Submit
Reset
Document: Done (21.16 secs)

response.jsp

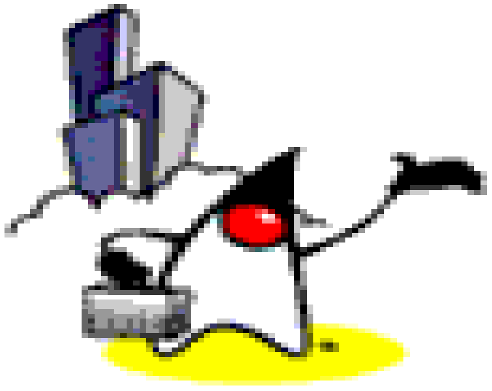

Hello - Netscape
File Edit View Go Bookmarks Tools Window Help
Search
Mail Home Radio Netscape Search Bookmarks
Hello
Submit Reset

http://localhost:8080/hello2/greeting?username=Hannah

My name is Duke. What is yours?

Hello, Hannah!

Transferring data from localhost...



Comparing Hello1 Servlet & Hello2 JSP code

GreetingServlet.java (1)

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

/**
 * This is a simple example of an HTTP Servlet.  It responds to the GET
 * method of the HTTP protocol.
 */
public class GreetingServlet extends HttpServlet {

    public void doGet (HttpServletRequest request,
                       HttpServletResponse response)
                       throws ServletException, IOException
 {

      response.setContentType("text/html");
      response.setBufferSize(8192);
      PrintWriter out = response.getWriter();

      // then write the data of the response
      out.println("<html>" +
                  "<head><title>Hello</title></head>");
```

GreetingServlet.java (2)

```
    // then write the data of the response
    out.println("<body  bgcolor=\"#ffffff\">" +
       "<img src=\"duke.waving.gif\">" +
       "<h2>Hello, my name is Duke. What's yours?</h2>" +
       "<form method=\"get\">" +
       "<input type=\"text\" name=\"username\" size=\"25\">" +
       "<p></p>" +
       "<input type=\"submit\" value=\"Submit\">" +
       "<input type=\"reset\" value=\"Reset\">" +
       "</form>");

    String username = request.getParameter("username");

    // dispatch to another web resource
    if ( username != null && username.length() > 0 ) {
          RequestDispatcher dispatcher =
             getServletContext().getRequestDispatcher("/response");

          if (dispatcher != null)
          dispatcher.include(request, response);

    }
    out.println("</body></html>");
    out.close();
  }
```

GreetingServlet.java (3)

```
    public String getServletInfo() {
        return "The Hello servlet says hello.";

    }
}
```

greeting.jsp

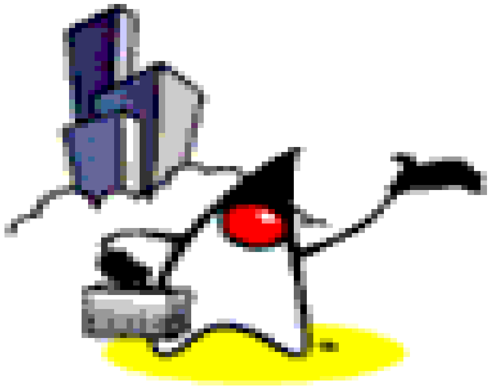
```
<html>
<head><title>Hello</title></head>
<body bgcolor="white">
<img src="duke.waving.gif">
<h2>My name is Duke. What is yours?</h2>

<form method="get">
<input type="text" name="username" size="25">
<p></p>
<input type="submit" value="Submit">
<input type="reset" value="Reset">
</form>

<%
    String username = request.getParameter("username");
    if ( username != null && username.length() > 0 ) {
%>
    <%@include file="response.jsp" %>
<%
    }
%>
</body>
</html>
```

ResponseServlet.java

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

// This is a simple example of an HTTP Servlet.  It responds to the GET
// method of the HTTP protocol.
public class ResponseServlet extends HttpServlet {

    public void doGet (HttpServletRequest request,
                       HttpServletResponse response)
                       throws ServletException, IOException{
        PrintWriter out = response.getWriter();

        // then write the data of the response
        String username = request.getParameter("username");
        if ( username != null && username.length() > 0 )
          out.println("<h2>Hello, " + username + "!</h2>");
    }

    public String getServletInfo() {
        return "The Response servlet says hello.";
    }
 }
```

response.jsp

```
<h2><font color="black">Hello, <%=username%>!</font></h2>
```



JSP “is” Servlet!

JSP is “Servlet”

- JSP pages get translated into servlet
  - Tomcat translates greeting.jsp into greeting$jsp.java
- Scriptlet (Java code) within JSP page ends up being inserted into jspService() method of resulting servlet
- Implicit objects for servlet are also available to JSP page designers, JavaBeans developers, custom tag designers

greeting$jsp.java (1) – no need to understand this code

```
package org.apache.jsp;

import javax.servlet.*;
import javax.servlet.http.*;
import javax.servlet.jsp.*;
import org.apache.jasper.runtime.*;


public class greeting$jsp extends HttpJspBase {


  static {
  }
  public greeting$jsp( ) {
  }

  private static boolean _jspx_inited = false;

  public final void _jspx_init() throws
   org.apache.jasper.runtime.JspException {
```

greeting$jsp.java (2)

```
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;
  String  _value = null;
```

greeting$jsp.java (3)

```
try {

  if (_jspx_inited == false) {
    synchronized (this) {
      if (_jspx_inited == false) {
        _jspx_init();
        _jspx_inited = true;
      }
    }
  }
  _jspxFactory = JspFactory.getDefaultFactory();
  response.setContentType("text/html;charset=ISO-8859-1");
  pageContext = _jspxFactory.getPageContext(this, request,
                               response,"", true, 8192, true);

  application = pageContext.getServletContext();
  config = pageContext.getServletConfig();
  session = pageContext.getSession();
  out = pageContext.getOut();
```

greeting$jsp.java (4)

```
    // HTML // begin [file="/greeting.jsp";from=(38,4);to=(53,0)]
      out.write("\n\n<html>\n<head><title>Hello</title></head>\n<body
bgcolor=\"white\">\n<img src=\"duke.waving.gif\"> \n<h2>My name is Duke.
What is yours?</h2>\n\n<form method=\"get\">\n<input type=\"text\"
name=\"username\" size=\"25\">\n<p></p>\n<input type=\"submit\"
value=\"Submit\">\n<input type=\"reset\"
value=\"Reset\">\n</form>\n\n");


    // end
    // begin [file="/greeting.jsp";from=(53,2);to=(56,0)]

         String username = request.getParameter("username");
          if ( username != null && username.length() > 0 ) {
    // end
    // HTML // begin [file="/greeting.jsp";from=(56,2);to=(57,4)]
      out.write("\n    ");

    // end
    // HTML // begin [file="/response.jsp";from=(38,4);to=(40,31)]
      out.write("\n\n<h2><font color=\"black\">Hello, ");

    // end
    // begin [file="/response.jsp";from=(40,34);to=(40,42)]
      out.print(username);
```

greeting$jsp.java (5)

```
      // HTML // begin [file="/response.jsp";from=(40,44);to=(55,0)]
        out.write("!</font></h2>\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n");

      // end
      // HTML // begin [file="/greeting.jsp";from=(57,37);to=(58,0)]
        out.write("\n");

      // end
      // begin [file="/greeting.jsp";from=(58,2);to=(60,0)]

            }
      // end
      // HTML // begin [file="/greeting.jsp";from=(60,2);to=(63,0)]
        out.write("\n</body>\n</html>\n");

      // end
    } catch (Throwable t) {
      if (out != null && out.getBufferSize() != 0) out.clearBuffer();
      if (pageContext != null) pageContext.handlePageException(t);
    } finally {
      if (_jspxFactory != null)
        _jspxFactory.releasePageContext(pageContext);
    }
  }
}
```

Thank you!

Sang Shin
Michèle Garoche
http://www.javapassion.com
"Learning is fun!"