JSP

Java Server Pages

Generating Dynamic web content

Generating dynamic web content can be achieved through several technologies, mentioning few technologies like

CGI – Common Gateway Interface

ASP– Active Server Pages

Servlet

Though these technologies are working well in their domains, due to increased demands of client these technologies fall short of demands.

Let us have a glance over these technologies and problems encountered through these technologies.

ASP – Active Sever Pages

A Microsoft based scripting language that allows you to create HTML templates that insert dynamic content from the server (usually a database)

Problems

Runs on Microsoft Windows only

- Limited number of CPU architectures that can handle Windows

Windows is not as stable (opinion)

Servlet

Benefits:

- Conceivably faster since is partially compiled and hence it is
- Easier to “interpret.”
- Load the JVM once
- Platform independence

Problems:

not be appropriate for everyone;

no separation between content and presentation

Idea

Use regular HTML for most of page

Mark servlet code with special tags

Need ?

To meet current requirements, we need a technology which

- Works on any webserver or application server
- Seperates the application logic from the presentation
- Allowing fast development and testing
- Simplifying the process of developing interactive web-based applications

The solution is JSP – Java Server Pages

The need for JSP

- With Servlets,it is easy to

-Read form data

-Read Http Request Headers

-Use Cookies and session Tracking

-Share data among servlets

-Remember data between requests

- But, sure it is a pain to

-use println to generate html

-Maintain that html

What is JSP

A JSP page is a text based document that describes how to process a request to create a response.

Advantages of JSP

- Separation of content and display logic
- Write once run anywhere
- Reuse of components and tag libraries
- Recompile automatically
- Support for scripting and actions

Actions permit the encapsulation of useful functionality in a convenient form that can be used by tools. Scripts, glue together this functionality in a per-page manner

- Web access for N-tier enterprise application architecture

What is a Java Server Page

- A JSP is a simple text file consisting of HTML or XML content along with JSP elements.
- A JSP combines Java code and template HTML in a single file.
- This is similar to the way PHP works.
- Scripting elements are used to provide dynamic pages

What is JSP?

- Mostly HTML page, with extension .jsp
- Include JSP tags to enable dynamic content creation
- Translation: JSP → Servlet class
- Compiled at Request time

  (first request, a little slow)
- Execution: Request → JSP Servlet's service method

What is a JSP?

```
<html>
   <body>
      <jsp:useBean.../>
      <jsp:getProperty.../>
      <jsp:getProperty.../>
   </body>
</html>
```

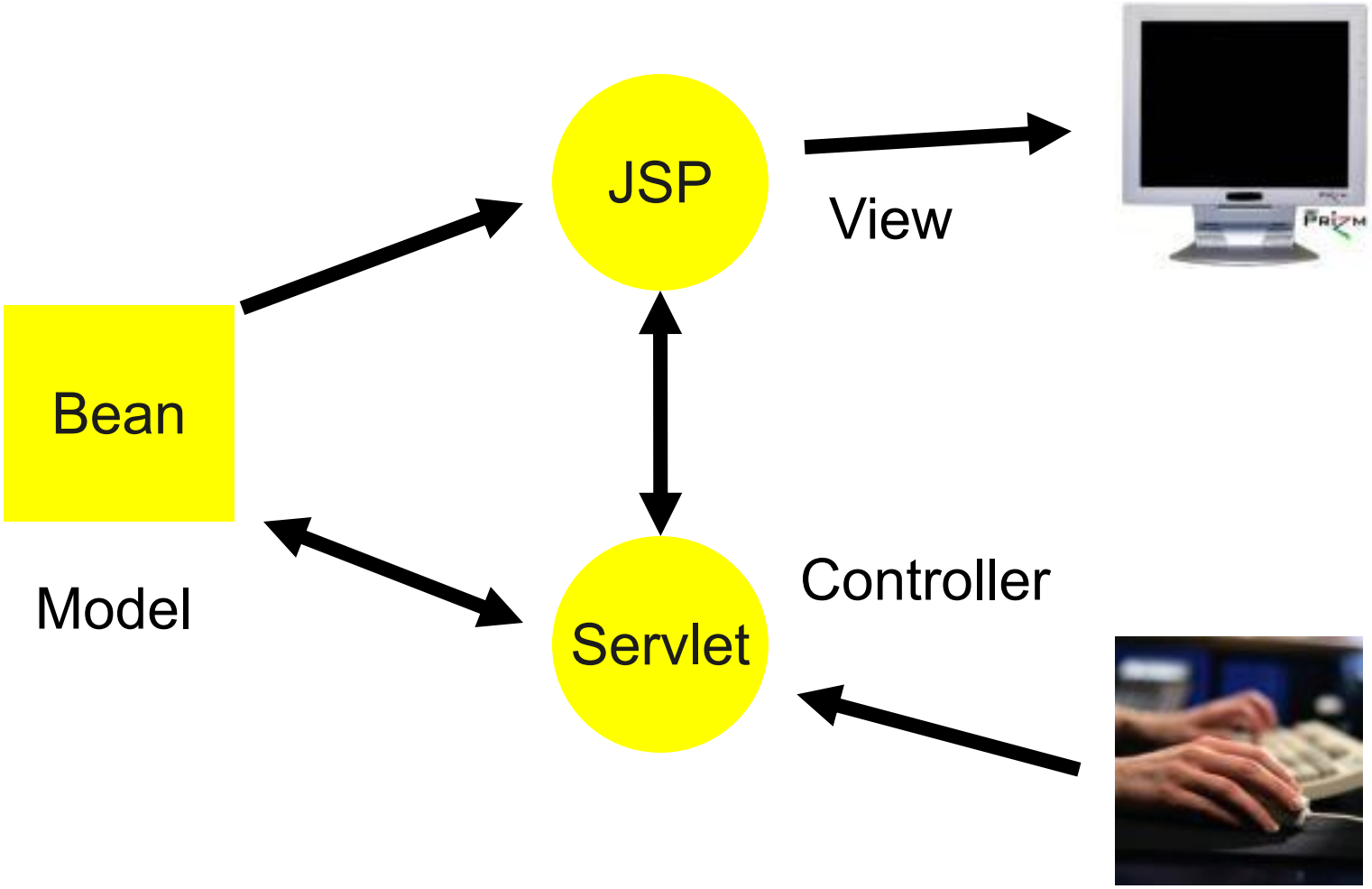
Advantages

- Code -- Computation
- HTML -- Presentation
- Separation of Roles
  - Developers
  - Content Authors/Graphic Designers/Web Masters
  - Supposed to be cheaper... but not really...

Model-View-Controller

- A Design Pattern
- Controller -- receives user interface input, updates data model
- Model -- represents state of the world (e.g. shopping cart)
- View -- looks at model and generates an appropriate user interface to present the data and allow for further input

Model-View-Controller



Pure Servlet

```
public class OrderServlet … {
  public void dogGet(…){
    if(isOrderValid(req)){
      saveOrder(req);
      …
      out.println("<html><body>");
      …
  }
  private void isOrderValid(…){
     …
  }
  private void saveOrder(…){
    …
  }
}
```

Servlet

```
Public class OrderServlet … {
  public void doGet(…){
    …
     if(bean.isOrderValid(…)){
      bean.saveOrder(req);
      …
      forward("conf.jsp");
  }
}
```

JSP

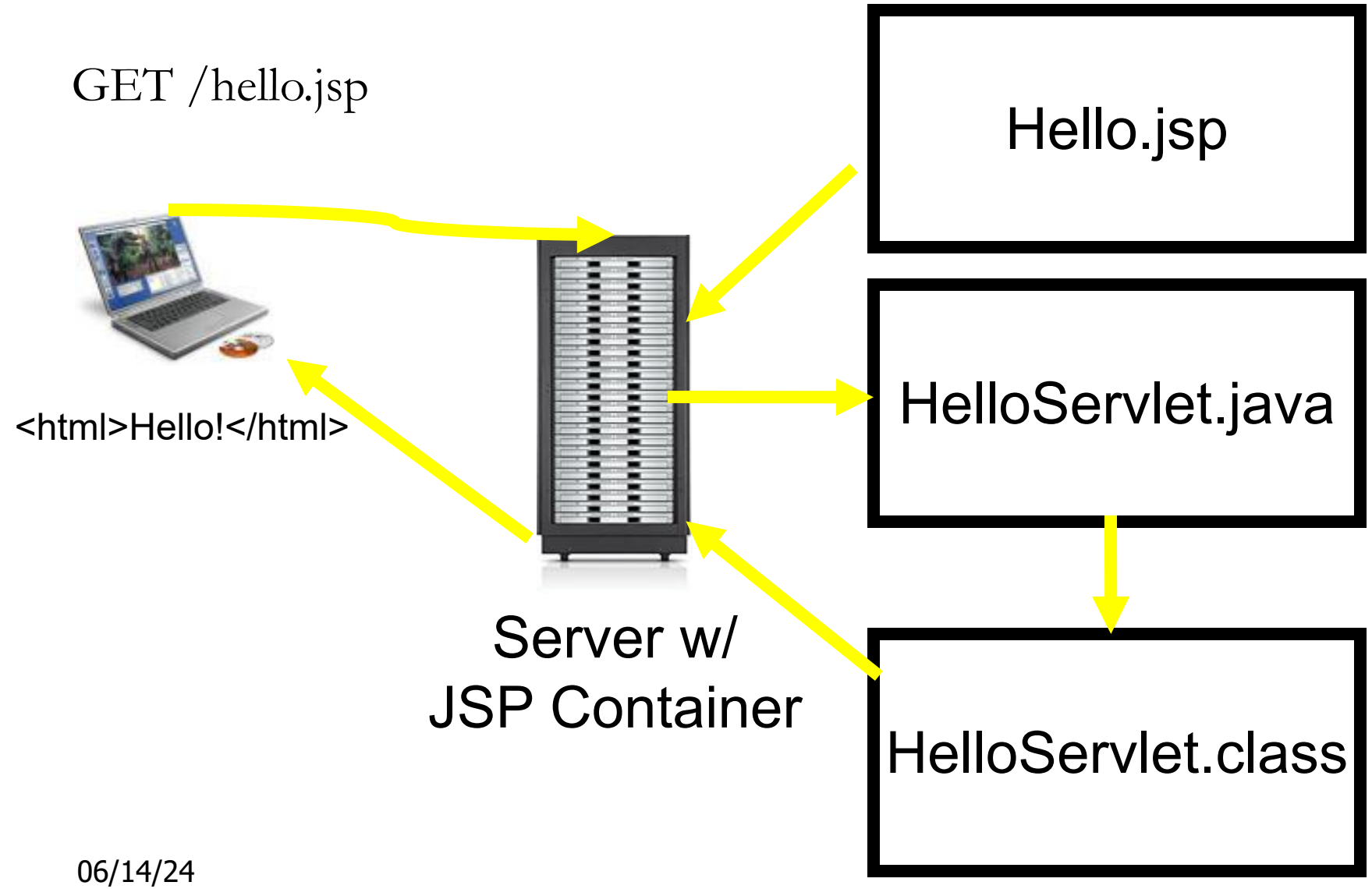
```
<html>
  <body>
    <c:forEach items="${order}">
    …
    </c:forEach>
  </body>
</html>
```

Java Bean

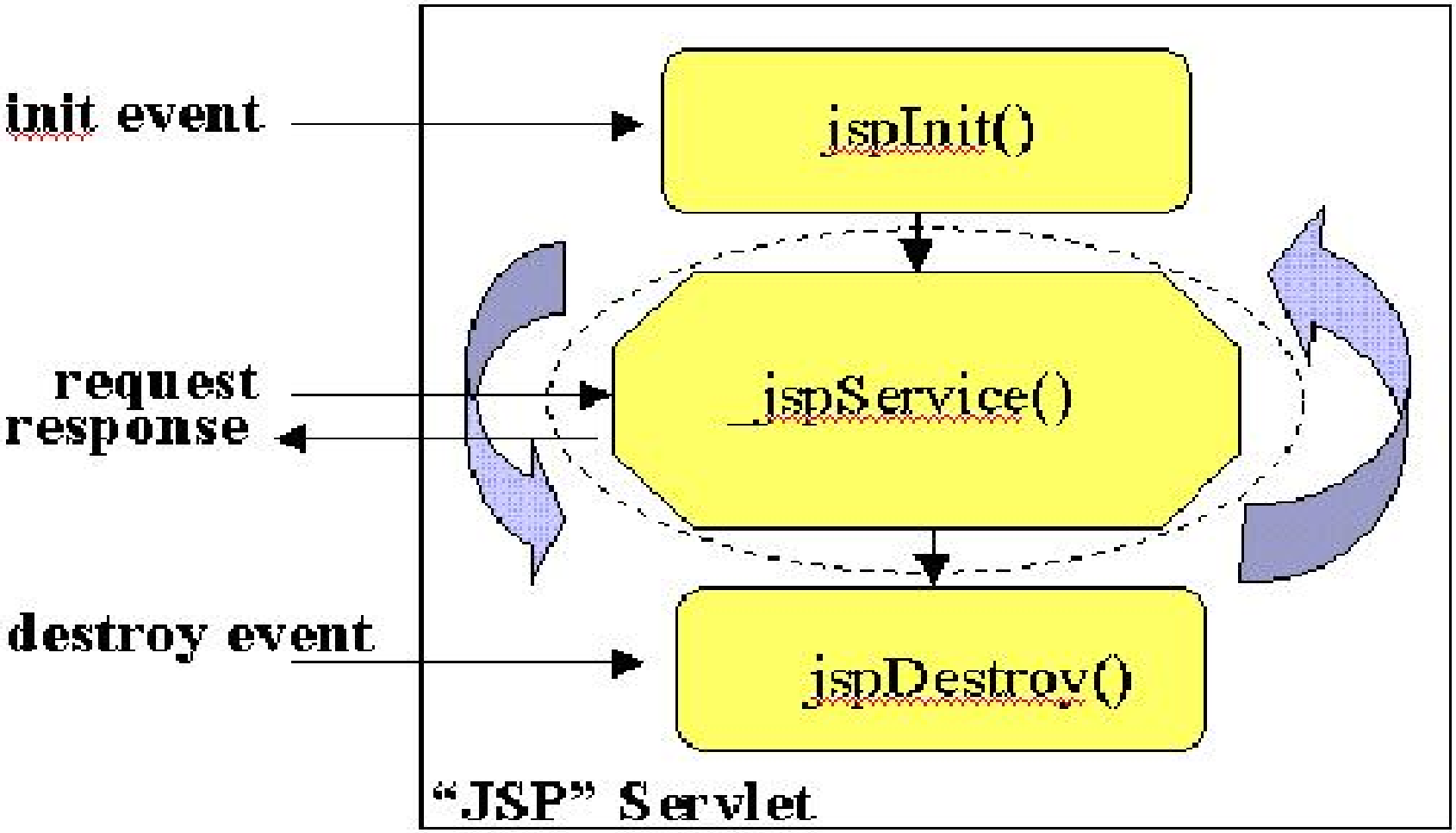
```
isOrderValid()
saveOrder()

------------------
private state
```

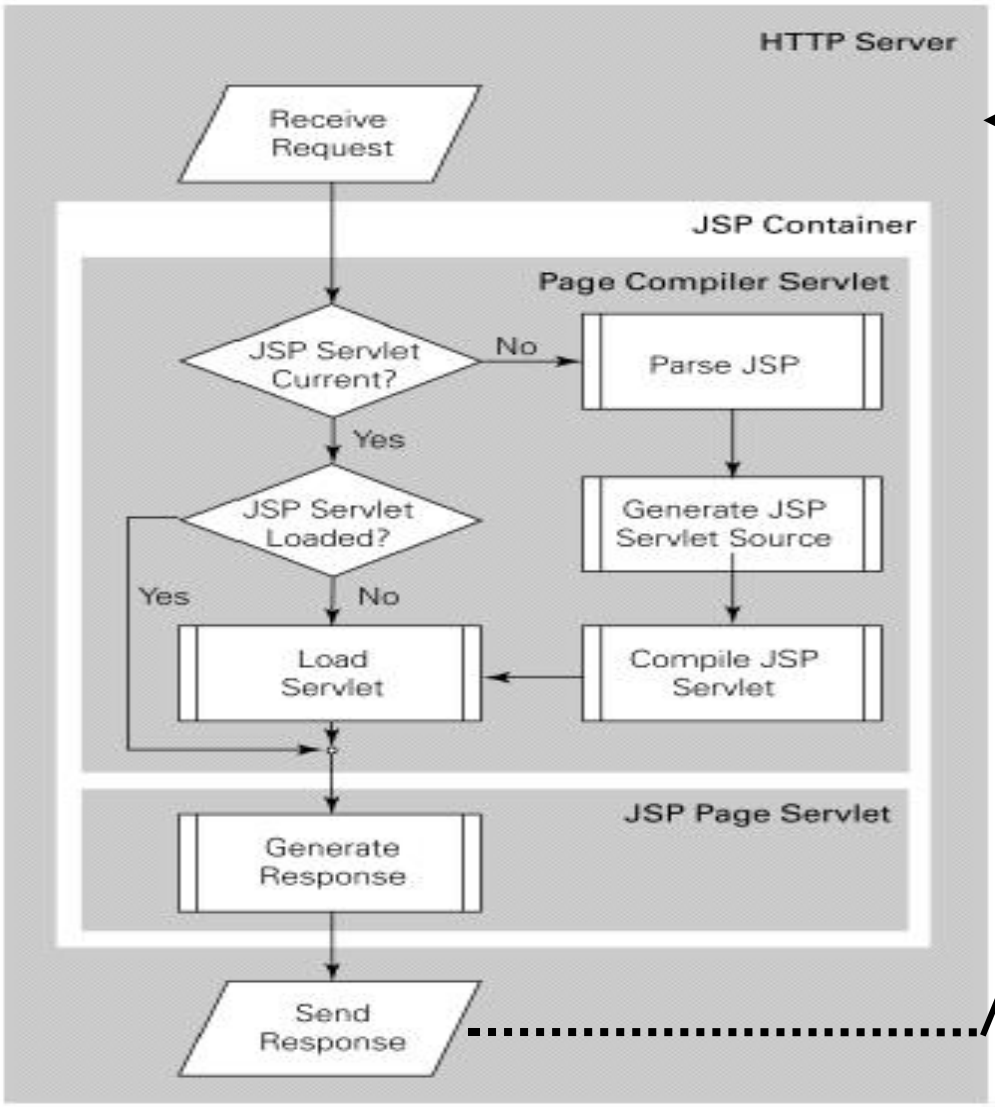
JSP Big Picture



JSP Life Cycle



JSP Page Processing



request

response

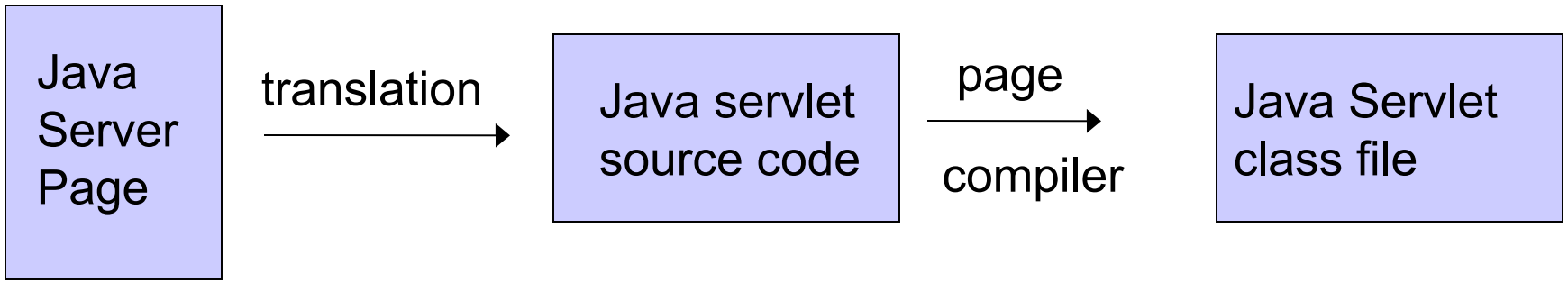
Client

Invoking Dynamic Code (from JSPs)

- Call Java Code Directly
  (Expressions, Declarations, Scriptlets)
- Call Java Code Indirectly
  (Separate Utility Classes, JSP calls methods)
- Use Beans
  (jsp:useBean, jsp:getProperty, jsp:setProperty)
- Use MVC architecture (servlet, JSP, JavaBean)
- Use JSP expression Language
  (shorthand to access bean properties, etc)
- Use custom tags
  (Develop tag handler classes; use xml-like custom tags)

Connection with servlets

- Each Java server page is compiled into a servlet before it can be used
- This is normally done when the first request is made so there could be a short wait.
- However, JSP's can be precompiled so there is no wait.



Translated servlets

- You can example the source code produced by the JSP translation process.
- There is a directory called work in the main tomcat directory where you can find the source code.
- Note that the _jspService method corresponds to the servlet service method (which is called by doGet or doPost)

JSP Syntax

JSP Syntax - All

Comments

Output Comment
Hidden Comment

Scripting Elements

Declaration
Expression
Scriplet

Directives

Page Include
Taglib

Actions

<jsp:forward>
<jsp:include>
<jsp:useBean>
<jsp:setProperty>
<jsp:getProperty> <jsp:plugin>

Built In Objects

request response
application page
pageContext out
session exception
config

Comments

Comments are used for adding documentation strings to a JSP page. The two types of comments are the

one which enables documentation to appear in the output from the page.

```
<!--   comment      -->
```

and the other JSP comments can only be viewed in the original JSP file, or in the source code for the servlet into which the page is translated.

```
<%--  comment --%>
```

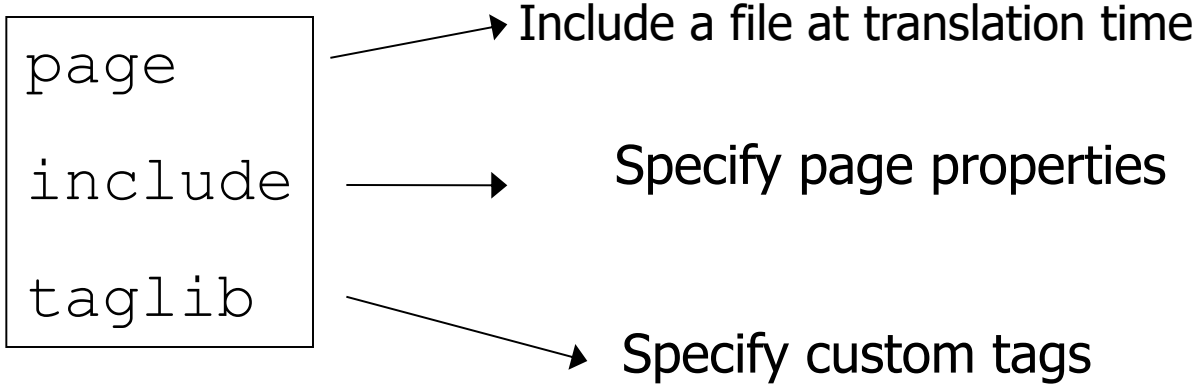
JSP elements (overview)

- Directives of the form <%@ ... %>
- Scripting elements
  - Expressions of the form <%= expr %>
  - Scriptlets of the form <% code %>
  - Declarations of the form <%! code %>
  - JSP Comments <%-- ... --%>
- Standard actions
  - Example: <jsp:useBean> ... </jsp:useBean>
- Implicit variables like request, response, out

Directives

- These provide global information to the page, for example, import statements, the page for error handling or whether the page is part of a session.
- They have the form

```
<%@ name attribute1="...", attribute2="..." ... %>
```



I. Page directive

The page directive defines a number of page dependent attributes and communicates these to the jsp container

A translation unit (Jsp Source file) can contain more than one instance of the page directive.

All the attributes will apply to the complete translation unit

There shall be only one occurrence of any attribute/value defined by this directive expect for the “import” attribute.

Page directive - Syntax

<%@ page *attribute1="value1" attribute2="value2" attribute3=…* %>

White space after the opening <%@ and before the closing %> is optional, but recommended to improve readability.

Like all JSP tag elements, the page directive supports n XML-based syntax, as follows:

< j s p : d i r e c t i v e . p a g e *attribute1="value1" attribute2="value2"*.. />

Page directive Attributes

There are eleven attributes for the page directive

Attribute	Value	Default	Examples
info	Text string	None	info="Registration form."
language	Scripting language name	"java"	language="java"
contentType	MIME type, character set	See first example	contentType="text/html; charset=ISO-8859-1" contentType="text/xml"
extends	Class name	None	extends="com.taglib.wdjsp.MyJspPage"
import	Class and/or package names	None	import="java.net.URL" import="java.util.*, java.text.*"
session	Boolean flag	"true"	session="true"
buffer	Buffer size, or false	"8kb"	buffer="12kb" buffer="false"
autoFlush	Boolean flag	"true"	autoFlush="false"
isThreadSafe	Boolean flag	"true"	isThreadSafe="true"
errorPage	Local URL	None	errorPage="results/failed.jsp"
isErrorPage	Boolean flag	"false"	isErrorPage="false"

Attribute description

Info attribute

The info attribute allows the page author to add a documentation string to the page that summarizes its functionality.

```
<%@ page info="Sample JSP Page directive - Sarath."
%>
```

Language attribute

The language attribute specifies the scripting language to be used in all scripting elements on the page.

```
<%@ page language="java" %>
```

Attribute description

ContentType attribute

This attribute is used to indicate the MIME type of the response being generated by the JSP page.

<%@ page contentType="text/xml" %>

The default MIME type for JSP pages is "text/html".

Extends attribute

The extends attribute identifies the superclass to be used by the JSP container when it is translating the JSP page into a Java servlet

<%@ page extends="myJspPage" %>

Attribute description

Import attribute

it extends the set of Java classes which may be referenced in a JSP page.

Importing a Class

```
<%@ page import="java.util.List" %>
```

Importing a Package

```
<%@ page import="java.util.*" %>
```

Importing a Class(es) / Package(s)

```
<%@ page import="java.util.List, java.util.ArrayList,
java.text.*" %>
```

Attribute description

Session attribute

The session attribute is used to indicate whether or not a JSP page participates in session management

<%@ page session= “false” %>

Buffer attribute

The buffer attribute controls the use of buffered output for a JSP page.

<%@ page buffer="none" %>

<%@ page buffer=“14kb" %>

Default value is 8kb

Attribute description

AutoFlush attribute

This attribute is also used for controlling buffered output.

```
<%@ page autoFlush="true" %>
```

IsThreadSafe attribute

The isThreadSafe attribute is used to indicate whether your JSP page, once it is compiled into a servlet, is capable of responding to multiple simultaneous requests.

If not, this attribute should be set to false

```
<%@ page isThreadSafe="false" %>
```

Attribute description

ErrorPage attribute

This attribute is used to specify an alternate page to display if an (uncaught) error occurs while the JSP container is processing the page.

```
<%@ page errorPage="/error.jsp" %>
```

IsErrorPage attribute

The isErrorPage attribute is used to mark a JSP page that serves as the error page for one or more other JSP pages.

```
<%@ page isErrorPage="true" %>
```

Scripting elements: expression

- For an expression scripting element like <%= expr %>, expr is evaluated and the result is converted to a string and placed into the JSP's servlet output stream. In a Java servlet this would be equivalent to

```
PrintWriter out = response.getWriter();
...
out.print(expr);
```

Formats the expression as a string for inclusion in the output of the page

Expression examples

- Displaying request parameters (request is an implicit object available in a JSP)

```
Your name is <%= request.getParameter("name") %>
and your age is <%= request.getParameter("age") %>
```

- Doing calculations

```
The value of pi is <%= Math.PI %> and the square root
of two is <%= Math.sqrt(2.0) %> and today's date is
<%= new java.util.Date() %>.
```

Scripting elements: scriptlet

- For a scriplet <% statements %> the Java statements are placed in the translated servlet's _jspService method body (it's like the servlet service method which calls either doGet or doPost)

```
public void _jspService(HttpServletRequest
   request, HttpServletResponse response)
   throws java.io.IOException, ServletException
{...
   statements
 ...
}
```

Scriplet examples

- Check a request parameter

```
<% String name = request.getParameter("name");
   if (name == null)
   { %>
      <h3>Please supply a name</h3>
<% }
   else
   { %>
      <h3>Hello <%= name %></h3>
<% } %>
```

There are 3 scriptlets here and an expression element

Scripting elements:declaration

- For a declaration <%! declarations %> the Java statements are placed in the class outside the _jspService method. Typical declarations can be Java instance variable declarations or Java methods

```
// declarations would go here
public void _jspService(...)
{
    ...
}
```

Declaration examples

- Declaring instance variables

```
<%! private int count = 0; %>
...
The count is <%= count++ %>.
```

- Declaring methods

```
<%!
private int toInt(String s)
{
    return Integer.parseInt(s);
}
%>
```

Including files

- Including files at translation time (when JSP is translated to a servlet)

```
<%@ include file="filename" %>
```

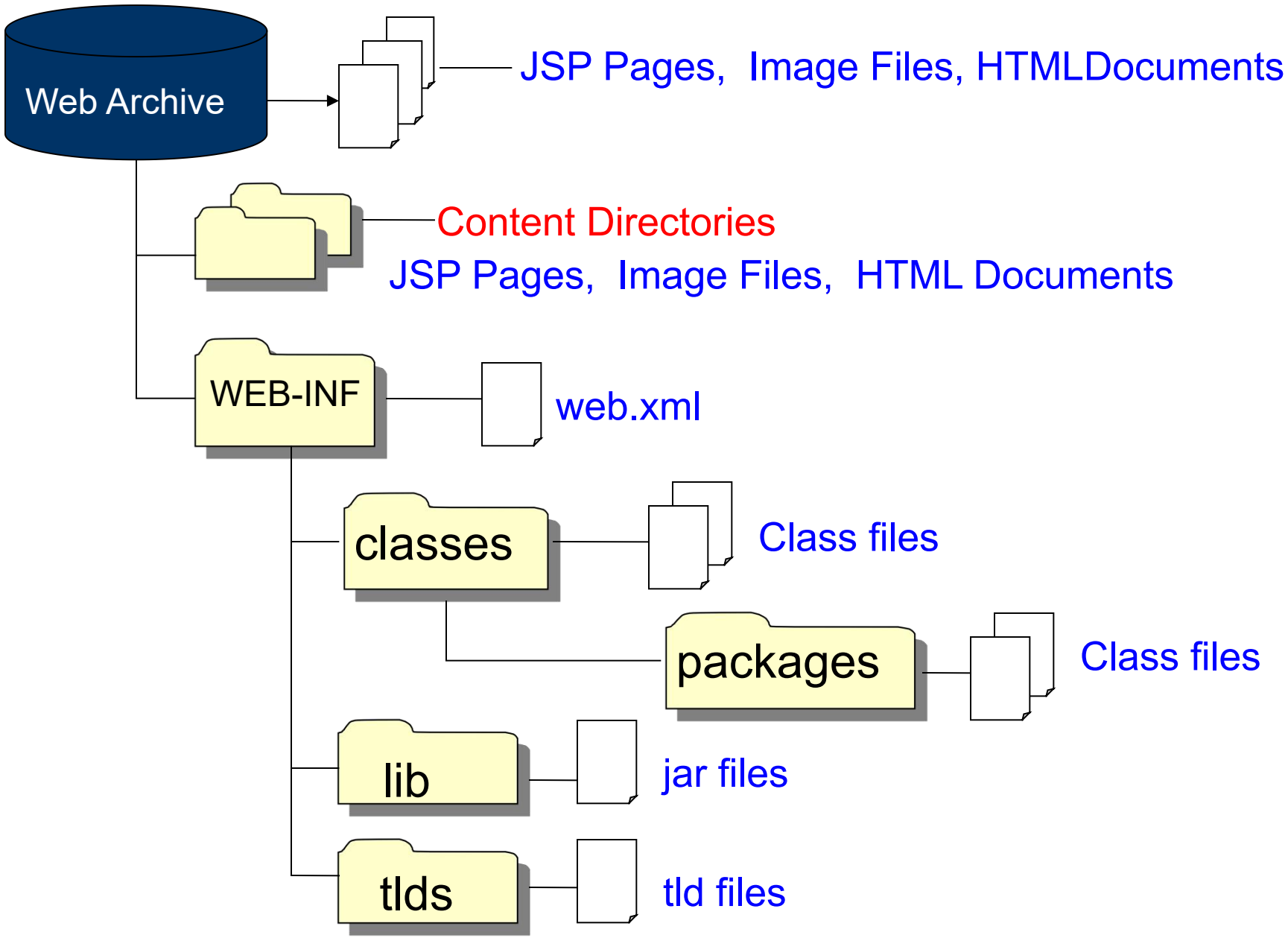
- Including files at request time

```
<jsp:include page="filename" flush = "true" />
```

Where do you put JSP's ?

- If you have a web application called examples then create a directory called jsp below it and put JSP's there:
- For example a JSP called test.jsp would have the path examples/jsp/test.jsp
- To run this JSP use the URL
  - http://localhost:8080/examples/jsp/test.jsp

War Directory Structure



A simple JSP

```
<html>
<head><title>JSP Test</title></head>
<body>
<h1>JSP Test</h1>
Time: <%= new java.util.Date() %>
</body>
</html>
```

The expression scripting element `<%= ... %>` is equivalent to the scriptlet `<% out.print(...); %>`

The implicit out object

- In a scriptlet <% ... %> you can use the out object to write to the output stream:
- Example:

```
<%
    out.print("The sum is ");
    out.print("1 + 2 = " + (1+2));
%>
```

The implicit request object

- Example

```
<html>
<head><title>...</title></head>
<body>
<h1>...</h1>
<p><%= request.getParameter("greeting") %></p>
</body></html>
```

Try this using
http://localhost:8080/examples/jsp/simple/greeting0.jsp?greeting=Hello

Implicit Objects	Description
`request`	The client's request. This is usually a subclass of `HttpServletRequest`. This has the parameter list if there is one.
`response`	The JSP page's response, a subclass of `HttpServletResponse`.
`pageContext`	Page attributes and implicit objects (essentially what makes up the server environment in which the JSP runs) need to be accessible through a uniform API, to allow the JSP engine to compile pages. But each server will have specific implementations of these attributes and objects. The solution to this problem is for the JSP engine to compile in code that uses a factory class to return the server's implementation of the `PageContext` class. That `PageContext` class has been initialized with the `request` and `response` objects and some of the attributes from the page directive (`errorpage`, `session`, `buffer` and `autoflush`) and provides the other implicit objects for the page request. We'll see more on this in a moment.
`session`	The HTTP session object associated with the request.
`application`	The servlet context returned by a call to `getServletConfig().getContext()` (see Chapter 5).
`out`	The object representing the output stream.

Processing form using GET

```
<html>
<head><title>JSP Processing ...</title></head>
<body>
<h1>JSP Processing form with GET</h1>
<form action="doForm1.jsp" method="GET">
First name: <input type="text" name="firstName"><br />
Last name: <input type="text" name="lastName">
<p><input type="submit" name="button"
      value="SubmitName"></p>
</form>
</body>
</html>
```

examples/jsp/forms/form1_get.html

Processing form using POST

```
<html>
<head><title>JSP Processing ...</title></head>
<body>
<h1>JSP Processing form with POST</h1>
<form action="doForm1.jsp" method="POST">
First name: <input type="text" name="firstName"><br />
Last name: <input type="text" name="lastName">
<p><input type="submit" name="button"
      value="SubmitName"></p>
</form>
</body>
</html>
```

examples/jsp/forms/form1_post.html

doForm1.jsp

```
<%@ include file="../doctype.html" %>
<head>
   <title>JSP Form Results</title>
</head>
<body>
<h1>JSP Form Results</h1>
Hello <%= request.getParameter("firstName") %>
<%= request.getParameter("lastName") %>
</body>
</html>
```

examples/jsp/forms/doForm1.jsp

Try this using
http://localhost:8080/examples/jsp/forms/form1_get.html

Java Beans

- Special classes that encapsulate some data
- They have a default constructor
- get and set methods for data fields (properties)
- A bean can be constructed in JSP using
  - `<jsp:useBean id = "..." class = "..." />`
- If the bean already exists this statement does nothing

setting properties

- To set a property of a bean use

```
<jsp:setProperty name="..."
    property="..." value="..." />
```

- To set a property using the value of a request parameter use

```
<jsp:setProperty name="..."
    property="..." param="..." />
```

getting properties

- To get a property of a bean use

```
<jsp:getProperty name="..."
    property="..." />
```

useBean Attributes	Description
`id="name"`	The name by which the instance of the bean can be referenced in the page. Other Bean tags use `name` to refer to it, so do note the difference.
`scope="page"`	The scope over which the bean can be called. More below.
`class="package.class"`	Fully qualified name of the bean class. Because this needs to be the full name, you don't need to `import` these classes in the `page` directive.
`beanName="name"`	This allows you to specify a serialized bean (`.ser` file) or a bean's name that can be passed to the `instantiate()` method from the `java.beans.Beans`. Needs an associated `type` tag rather than `class`.
`type="package.class"`	A synonym for `class` that is used for `beanName`.

A Greeting bean

```
package beans;
public class Greeting
{
    private String greeting; // the property

    public Greeting()
    { greeting = "Hello World"; }

    public String getGreeting()
    { return greeting; }

    public void setGreeting(String g)
    {  greeting = (g == null) ? "Hello World" : g;
    }
}
```

beans/Greeting.java

Naming convention

- If the property name is `greeting`
- the get method must have the name `getGreeting`
- the set method must have the name `setGreeting`

Creating a Greeting bean (1)

- Create a bean and use default property

```
<jsp:useBean id="hello" class="beans.Greeting" />
```

- Create a bean and set its property when it is constructed

```
<jsp:useBean id="hello" class="beans.Greeting" >
<jsp:setProperty name="hello" property="greeting"
   value="Hello JSP World" />
</jsp:useBean>
```

- Here `<jsp:setProperty>` is in the body of the `<jsp:useBean>` element.

Creating a Greeting bean (2)

- Create a bean and set its property after it has been constructed

```
<jsp:useBean id="hello" class="beans.Greeting" />
<jsp:setProperty name="hello" property="greeting"
   value="Hello JSP World" />
```

- The <jsp:setProperty> tag is now outside the <jsp:useBean> tag so it will always set the property, not just when the bean is constructed

greeting1.jsp

```
<jsp:useBean id="hello" class="beans.Greeting" />
<jsp:setProperty name="hello" property="greeting"
   value="Hello JSP World" />
<html>
<head>
<title>Greeting JSP that uses a Greeting bean</title>
</head>
<body>
<h1>Greeting JSP that uses a Greeting bean</h1>
<p><jsp.getProperty name="hello" property="greeting" />
</p>
</body>
</html>
```

test/jsp/greeting1.jsp http://localhost:8080/test/jsp/greeting1.jsp

Two beans

One initialized explicitly
and the other is initialized
using a request parameter

greeting2.jsp

```
<jsp:useBean id="greet1" class="beans.Greeting" />
<jsp:useBean id="greet2" class="beans.Greeting" />
<jsp:setProperty name="greet1" property="greeting"
   value="Hello JSP World" />
<jsp:setProperty name="greet2" property="greeting"
   param="greeting" />
<html><head><title>Greeting JSP using two Greeting
beans</title></head><body>
<h1>Greeting JSP using two Greeting beans</h1>
<p>1st bean: <jsp:getProperty name="greet1"
               property="greeting" /></p>
<p>2nd bean: <jsp:getProperty name="greet2"
               property="greeting" /></p></body></html>
```

test/jsp/greeting2.jsp http://localhost:8080/test/jsp/greeting2.jsp

http://localhost:8080/test/jsp/greeting2.jsp?greeting=Hello+friend

three beans and include file

One initialized explicitly, one
is
initialized using a request
parameter, and one is
initialized using getParameter

greeting3.jsp (1)

```
<jsp:useBean id="greet1" class="beans.Greeting" />
<jsp:useBean id="greet2" class="beans.Greeting" />
<jsp:useBean id="greet3" class="beans.Greeting" />
<jsp:setProperty name="greet1" property="greeting"
   value="Hello JSP World" />
<jsp:setProperty name="greet2" property="greeting"
   param="greeting" />

<%-- Following works but param method is better --%>

<jsp:setProperty name="greet3" property="greeting"
  value="<%= request.getParameter(\"greeting\") %>" />
```

greeting3.jsp (2)

```
<%-- Include file contains doctype and html tag --%>
<jsp:include page="doctype.html" flush="true" />
<head><title>Greeting JSP</title></head><body>
<h1>Greeting JSP</h1>
<p>1st bean: <jsp:getProperty name="greet1"
                property="greeting" /></p>
<p>2nd bean: <jsp:getProperty name="greet2"
                property="greeting" /></p>
<p>3rd bean: <jsp:getProperty name="greet3"
                property="greeting" /></p>
<p>request: <%= request.getParameter("greeting") %></p>
</body></html>
```

test/jsp/greeting3.jsp http://localhost:8080/test/jsp/greeting3.jsp

http://localhost:8080/test/jsp/greeting3.jsp?greeting=Hello+Fred