TATA CONSULTANCY SERVICES

JavaServerPages™

Version 1.0

Experience certainty. IT Services
Business Solutions
Outsourcing

JSP

- JavaServerPages (JSP) technology allows web developers to rapidly develop dynamic web pages and easily maintain them.

- JSP technology helps in embedding dynamic content within a HTML page. This avoids generating the complete HTML page using a Servlet.
- Hence, page design can be done by designer and wherever the dynamic content has to come, programmer can put the Java code.

What a JSP page contains ?

Text-based document that contains two types of text :

- Static components, expressed in any text-based format, such as HTML, WML, XML
- JSP elements, which construct dynamic content

Main Purpose of JSP

Separation of static from dynamic content

- Logic to generate the dynamic content is separated from the static presentation templates by using it within external JavaBeans components.
- The JavaBeans are then created and used by the JSP page using special tags and scriptlets.
- When a page designer makes any changes to the presentation template, the JSP page is automatically recompiled and reloaded into the web server by the JSP engine.

JSP -Advantages

- **Write Once Run Anywhere**

  JSP pages can be moved easily across platforms, and across web servers, without any changes

- **Dynamic content can be served in a variety of formats**

  There is nothing that mandates the static template data within a JSP page to be of a certain format.

  JSP can service a diverse clientele ranging from conventional browsers using HTML/DHTML, to handheld wireless devices like mobile phones and PDAs using WML, to B2B applications using XML.

JSP – Advantages(Contd..)

- Recommended Web access layer for n-tier architecture: Sun's J2EE™ Blueprints, which categorically recommends JSP over servlets for serving dynamic content.
- Completely leverages the Servlet API:

  You can do almost anything that can be done with servlets using JSP--but more easily!

JSP Architecture

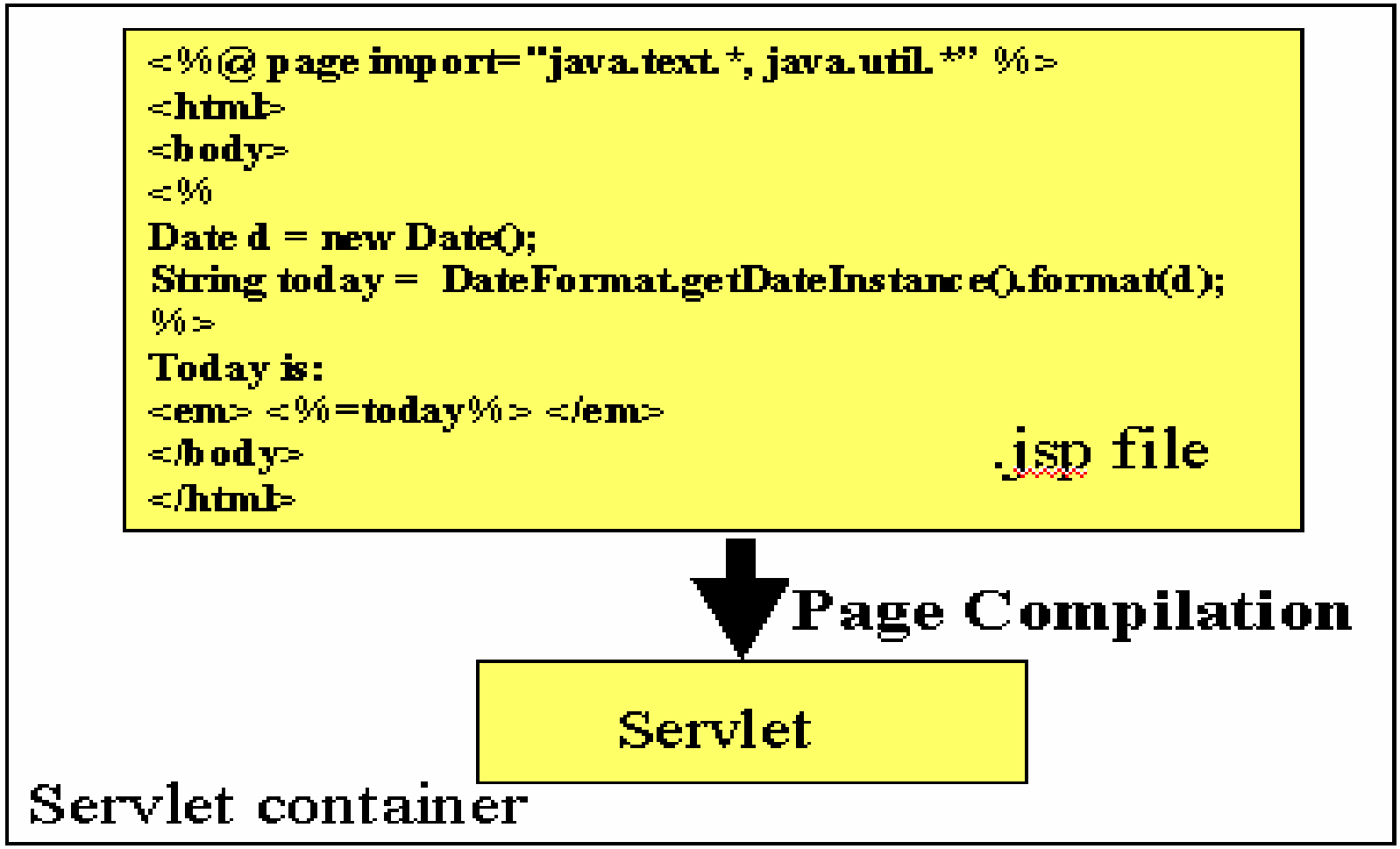
JSP pages are subject to two phases.

- Translation phase
- Request processing phase

Translation phase : The JSP page is translated only once into Servlet, until the JSP page changes again.

Request processing phase : After the page has been translated, the request processing phase is handled by the servlet.

JSP Translation



JSP Translation (Contd..)

- The translation phase is carried out by the JSP engine itself, when it receives an incoming request for the JSP page for the first time.
- The JSP 1.1 specification allows for JSP pages to be precompiled into class files. Precompilation may be useful in removing the start-up lag that occurs when a JSP page receives the first request from a client.

Note : Many details of the translation phase, like the location where the source and class files are stored are implementation dependent

JSP Translated Servlet

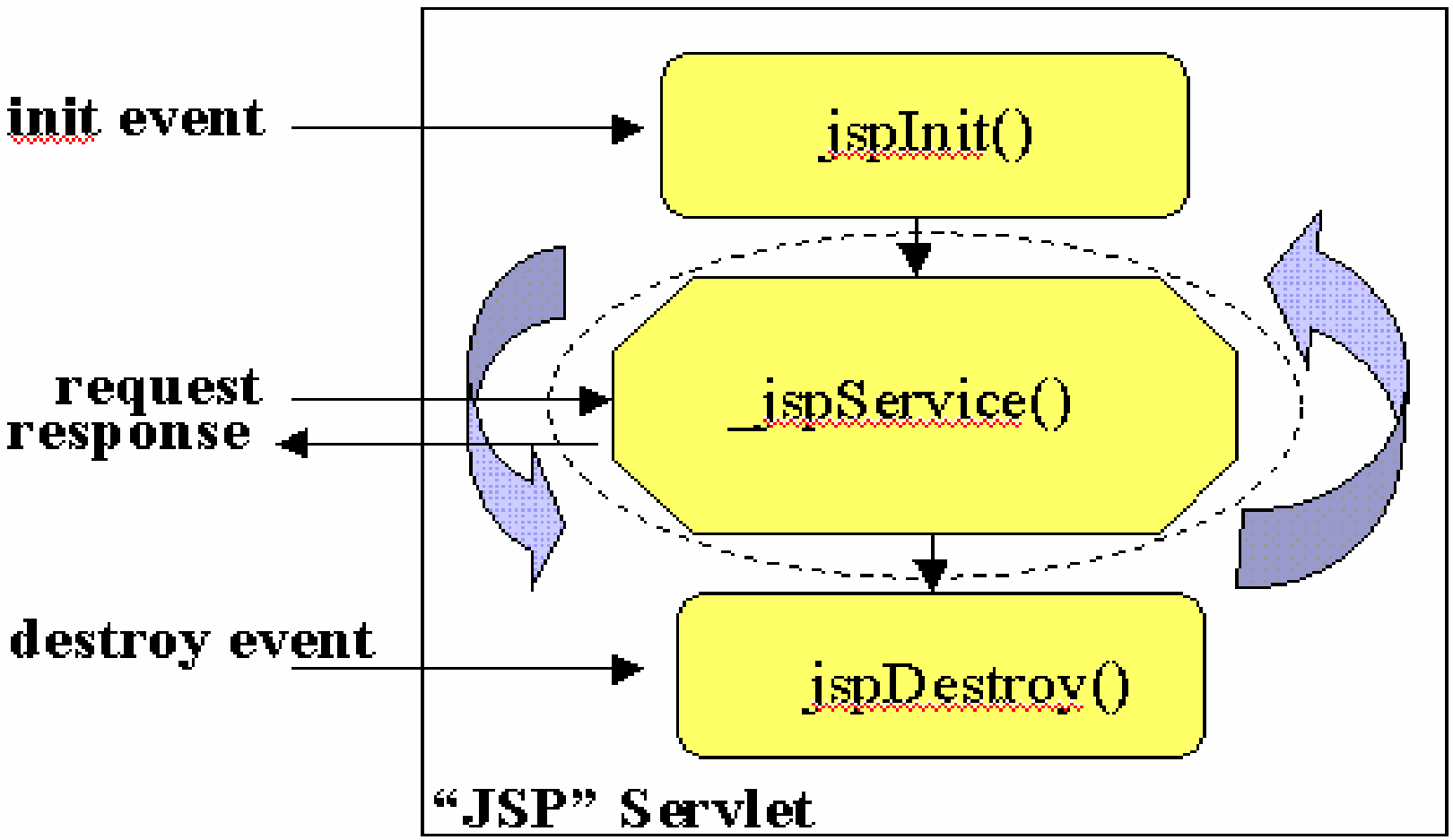
- The JSP page implementation class file extends **HttpJspBase**, which implements the **Servlet** interface. The service method of this class, _jspService(), essentially inline the contents of the JSP page.

  Note : Although _jspService() cannot be overridden, the developer can describe initialization and destroy events by providing implementations for the jspInit() and jspDestroy() methods within their JSP pages.

JSP Translated Servlet (Contd..)

- Once this class file is loaded within the servlet container, the _jspService() method is responsible for replying to a client's request.
- By default, the _jspService() method is dispatched on a separate thread by the servlet container in processing concurrent client requests

JSP Translated Servlet(Contd..)



JSP Syntax Basics

Directives :

JSP directives are messages for the JSP engine

- Tell the engine what to do with the rest of the JSP page
- JSP directives are always enclosed within the <%@ ... %> tag

Page Directive

- Typically, the page directive is found at the top of almost all of your JSP pages
- Any number of page directives could be added within a JSP page, although the attribute/value pair must be unique
- Simple page directive importing a java package:

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

Include Directive

- Helps in separating content into manageable elements. For example : a common header or footer in all the pages of a web application could be included with this directive.
- The page included can be a static HTML page or more JSP content. The page could be included at any location.

<%@ include file="copyright.html" %>

Declarations

- A typical declaration directive would be:
- <%! int i=0; %>

We can also declare methods. For example, you can override the initialization event in the JSP life cycle by declaring:

- <%! public void jspInit()

{ //some initialization code }

%>

Expressions

- Typically expressions are used to display simple values of variables or return values by invoking a bean's getter methods.
- The results of evaluating the expression are converted to a string and directly included within the output page.
- JSP expressions begin within <%= ... %> tags and do not include semicolons:

  <%= fooVariable %> <%= fooBean.getName() %>

Scriptlets

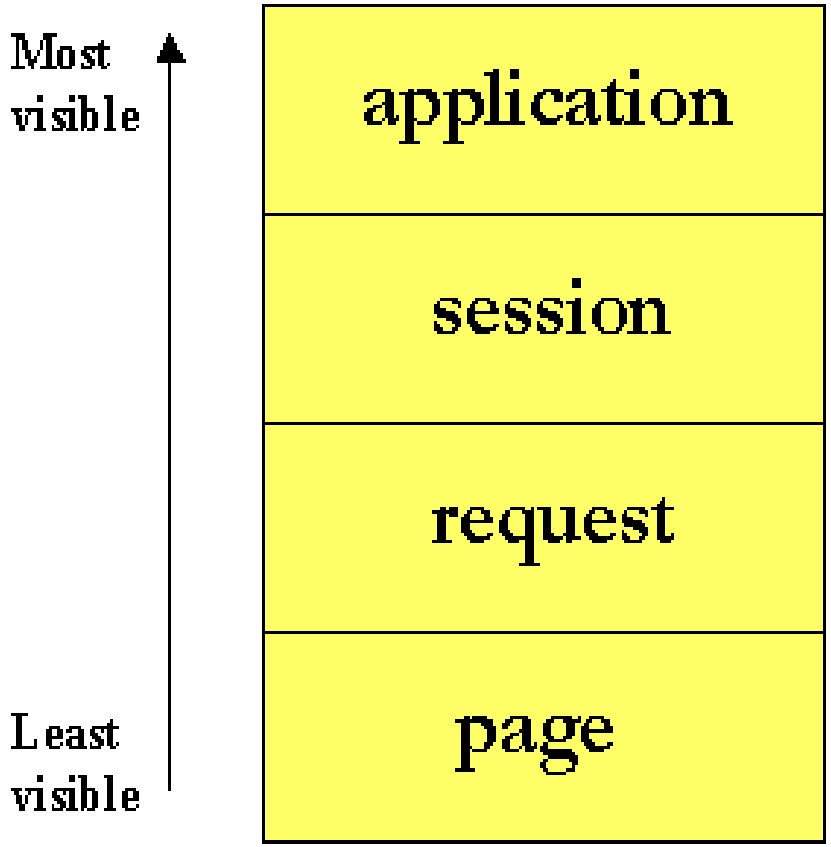
- JSP code fragments or scriptlets are embedded within <% ... %> tags.
- The java code is run when the request is serviced by the JSP page. You can have just about any valid Java code within a scriptlet, and is not limited to one line of source code. For example, the following displays the string "Hello" within H1, H2, H3, and H4 tags, combining the use of expressions and scriptlets:
- <% for (int i=1; i<=4; i++) { %>

  <H<%=i%>>Hello</H<%=i%>> <% } %>

Comments tag

- Although you can always include HTML comments in JSP pages, users can view these if they view the page's source. If you don't want users to be able to see your comments, embed them within the <%-- ... --%> tag:
- <%-- comment for server side only --%> A most useful feature of JSP comments is that they can be used to selectively block out scriptlets or tags from compilation. Thus, they can play a significant role during the debugging and testing process

Object Scopes



JSP Implicit Objects

- The JSP container makes available implicit objects that can be used within scriptlets and expressions, without the page author first having to create them.
- The nine implicit objects are :

1) **request**: Represents the HttpServletRequest triggering the service invocation. *Request scope.*

JSP Implicit Objects (Contd..)

2) **response**: Represents HttpServletResponse to the request. *Page scope*.

3) **pageContext**: Encapsulates implementation-dependent features in PageContext.

Page scope.

4) **application**: Represents the ServletContext obtained from servlet configuration object. *Application scope*.

5) **out**: A JspWriter object that writes into the output stream. *Page scope*.

JSP Implicit Objects(Contd..)

- **config**: Represents the ServletConfig for the JSP. *Page scope*.
- **page**: synonym for the "**this**" operator, as an HttpJspPage. Not used often by page authors. *Page scope*.
- **session**: An HttpSession. *Session scope*.
- **exception**: the uncaught Throwable object that resulted in the error page being invoked. *Page scope*.

JSP Implicit Objects (Contd..)

- These implicit objects are only visible within the system generated _jspService() method. They are not visible within methods you define yourself in declarations.

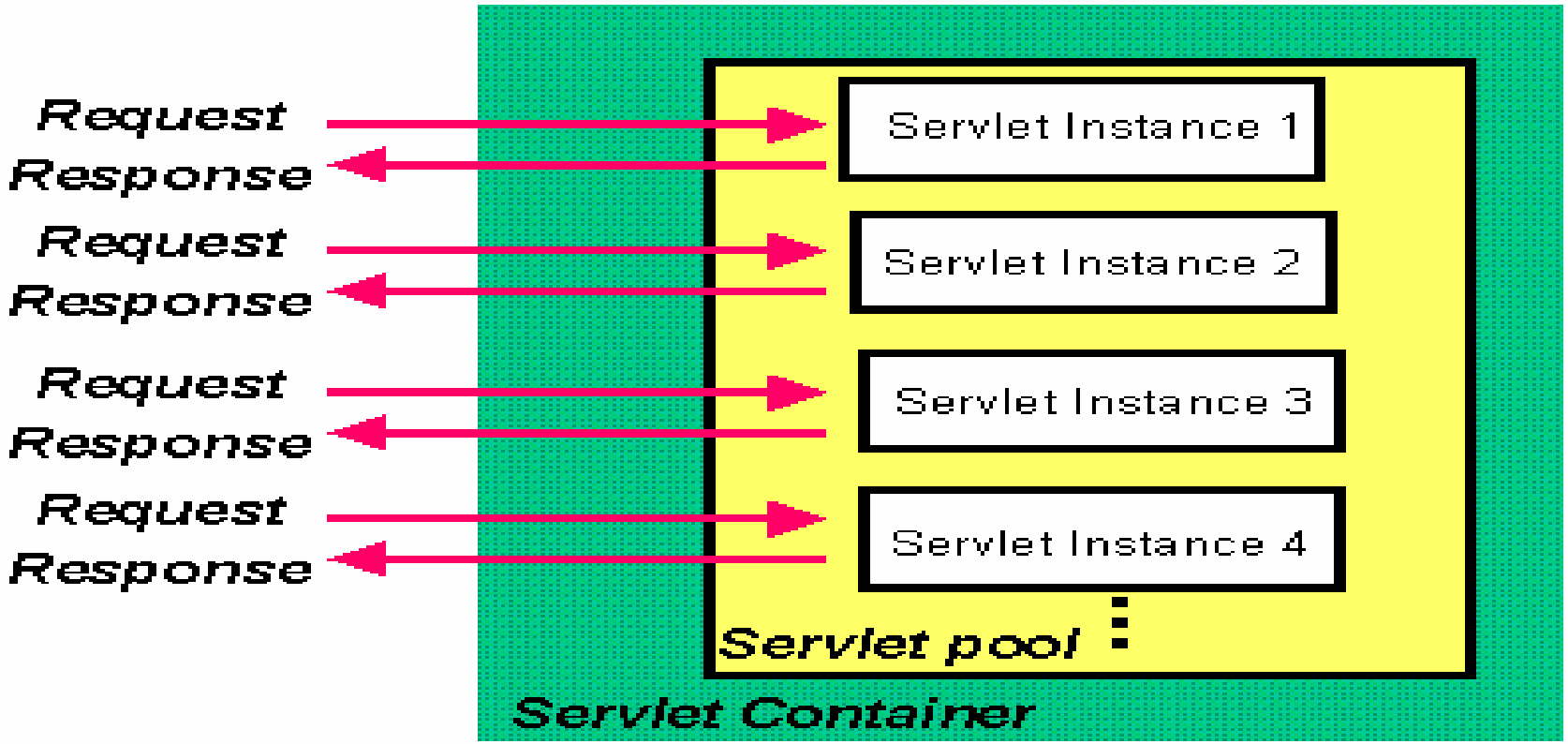
Synchronization Issues

- By default, the service method of the JSP page implementation class that services the client request is multithreaded. Thus, it is the responsibility of the JSP page author to ensure that access to shared state is effectively synchronized.
- There are a couple of different ways to ensure that the service methods are thread-safe. The easy approach is to include the JSP page directive.

Synchronization Issues(Contd..)

- <%@ page isThreadSafe="false" %>
- This causes the JSP page implementation class to implement the SingleThreadModel interface, resulting in the synchronization of the service method, and having multiple instances of the servlet to be loaded in memory.
- The concurrent client requests are then distributed evenly amongst these instances for processing in a round-robin fashion.

Multiple Servlet Instances



Drawback

- The downside of using the above approach is that it is not scalable. If the wait queue grows due to a large number of concurrent requests overwhelming the processing ability of the servlet instances, then the client may suffer a significant delay in obtaining the response.

Synchronization Issues(Contd..)

- A better approach is to explicitly synchronize access to shared objects (like those instances with application scope, for example) within the JSP page, using scriptlets:

```
<%
        synchronized (application) {
        SharedObject foo =
(SharedObject)application.getAttribute("sharedObject");
        foo.update(someValue);
                application.setAttribute("sharedObject",foo);
         }
%>
```

Exception handling

- JSP provides a rather **elegant mechanism** for handling runtime exceptions. Although you can provide your own exception handling within JSP pages, it may not be possible to anticipate all situations.
- By making use of the page directive's **errorPage** attribute, it is possible to forward an uncaught exception to an error handling JSP page for processing.

Exception handling(Contd..)

- For example :

<%@ *page isErrorPage="false"*errorPage="errorHandler.jsp" %>

- informs the JSP engine to forward any uncaught exception to the JSP page **errorHandler.jsp**. It is then necessary for errorHandler.jsp to flag itself as a error processing page using the directive:

Exception handling (Contd..)

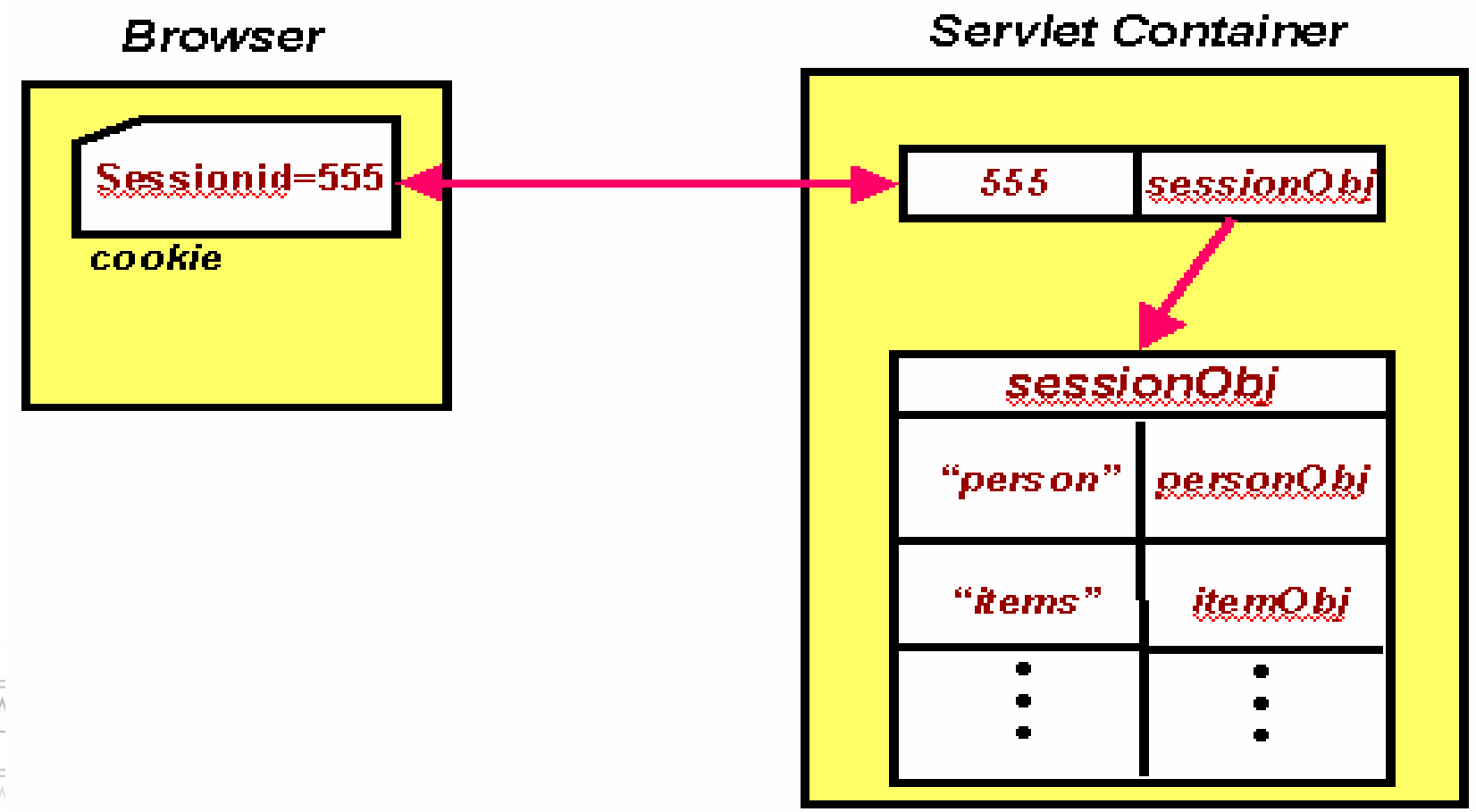
- <%@ page isErrorPage="true" %>

  - This allows the Throwable object describing the exception to be accessed within a scriptlet through the implicit exception object.

Session Management

- Sessions are a good place for storing beans and objects that need to be shared across other JSP pages and servlets that may be accessed by the user.
- By default, all JSP pages participate in an **Httpsession**.
- The HttpSession object can be accessed within scriptlets through the session implicit JSP object.
- The session objects is identified by a session ID and stored in the browser as a cookie.

Session Management(Contd..)



Example – Session Management

-
```
<%
        Foo foo = new Foo();
             session.putValue("foo",foo);
    %>
```

- makes available the Foo instance within all JSP pages and servlets belonging to the same session. The instance may be retrieved within a different JSP page as:

```
<%
        Foo myFoo = (Foo) session.getValue("foo");
%>
```

Session Management (Contd..)

- If cookies are unsupported by the browser, then the session ID may be maintained by URL rewriting.
- Support for URL rewriting is not mandated by the JSP specification and is supported only within a few servers.
- Although you cannot place primitive data types into the session, you can store any valid Java object by identifying it by a unique key.

Using JavaBean Components

- Before you can access a bean within a JSP page, it is necessary to identify the bean and obtain a reference to it.
- The <*jsp:useBean*> tag tries to obtain a reference to an existing instance using the specified id and scope, as the bean may have been previously created and placed into the session or application scope from within a different JSP page.
- The bean is newly instantiated using the Java class name specified through the class attribute only if a reference was not obtained from the specified scope.

Using Java Beans (Contd..)

- *<jsp:useBean id="user" class="com.jguru.Person" scope="session" />*
- In this example, the Person instance is created just once and placed into the session. If this useBean tag is later encountered within a different JSP page, a reference to the original instance that was created before is retrieved from the session.

Access Java Bean properties

- Once you have declared a JavaBean component, you have access to its properties to customize it.
- The value of a bean's property is accessed using the <*jsp:getProperty*> tag. With the <*jsp:getProperty*> tag, you specify the name of the bean to use (from the id field of useBean), as well as the name of the property whose value you are interested in. The actual value is then directly printed to the output:

<*jsp:getProperty name="user" property="name" />*

Access Java Bean properties(Contd..)

- Changing the property of a JavaBean component requires you to use the <*jsp:setProperty*> tag. For this tag, you identify the bean and property to modify and provide the new value:
- <jsp:setProperty name="user" property="name" value="jec" />
  or
- <jsp:setProperty name="user" property="name" value="<%=expression %>" />

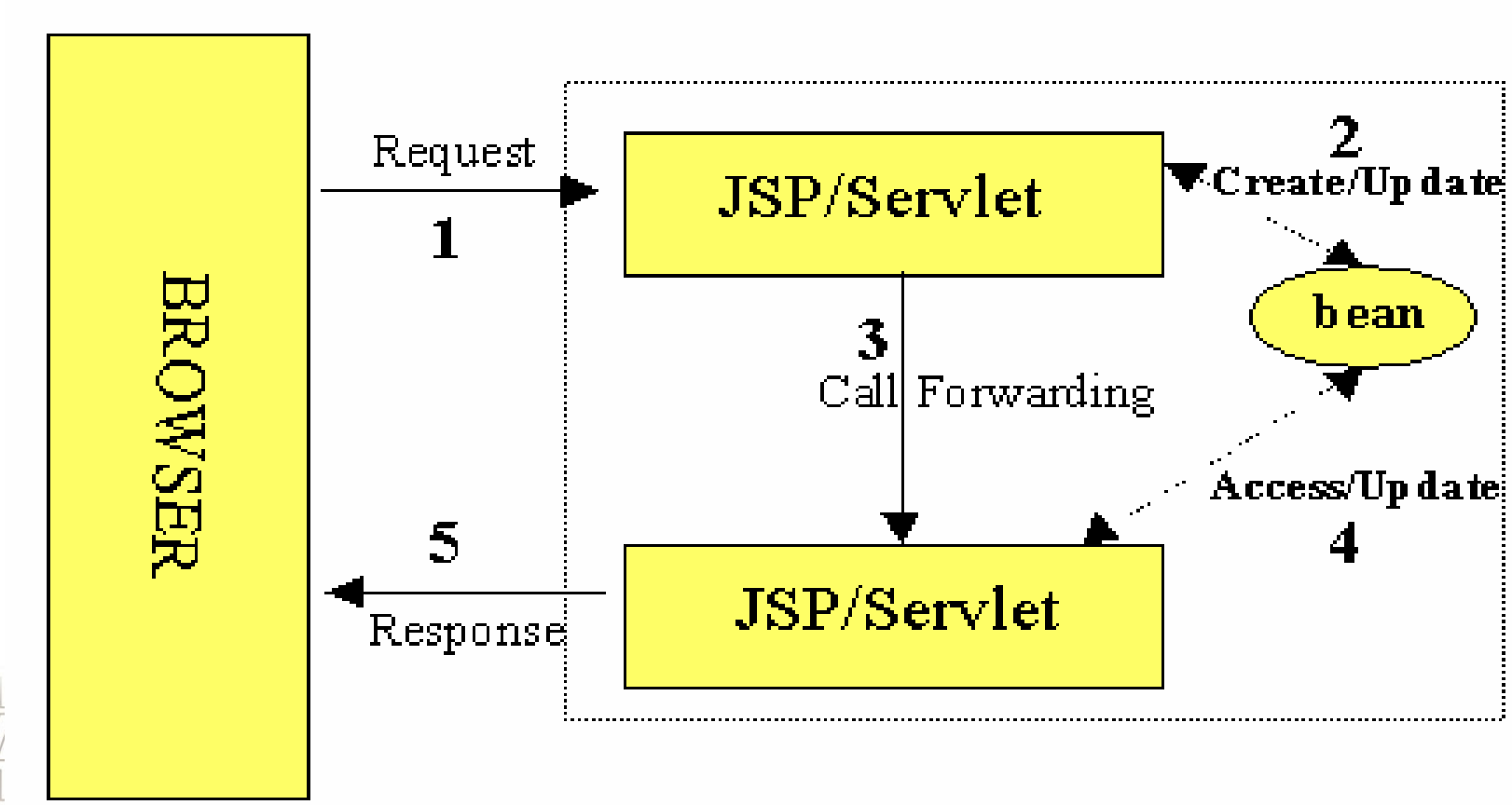
Better Approach

- When developing beans for processing form data, you can follow a **common design pattern** by matching the names of the bean properties with the names of the form input elements.

  You also need to define the corresponding getter/setter methods for each property within the bean. The advantage in this is that you can now direct the JSP engine to parse all the incoming values from the HTML form elements that are part of the request object, then assign them to their corresponding bean properties with a single statement, like this:

- *<jsp:setProperty name="user" property="*"/>*

Forwarding requests



Forwarding Requests (Contd..)

- A <*jsp:forward*> tag may also have jsp:param subelements that can provide values for some elements in the request used in the forwarding:
- For example :

<jsp:forward page="<%= somePage %>" > <jsp:param name="name1" value="value1" /> <jsp:param name="name2" value="value2" /> </jsp:forward>

Forwarding Requests(Contd..)

- With the <jsp:forward> tag, you can redirect the request to any JSP, servlet, or static HTML page within the same context as the invoking page. This effectively halts processing of the current page at the point where the redirection occurs, although all processing up to that point still takes place:
- <jsp:forward page="somePage.jsp" />

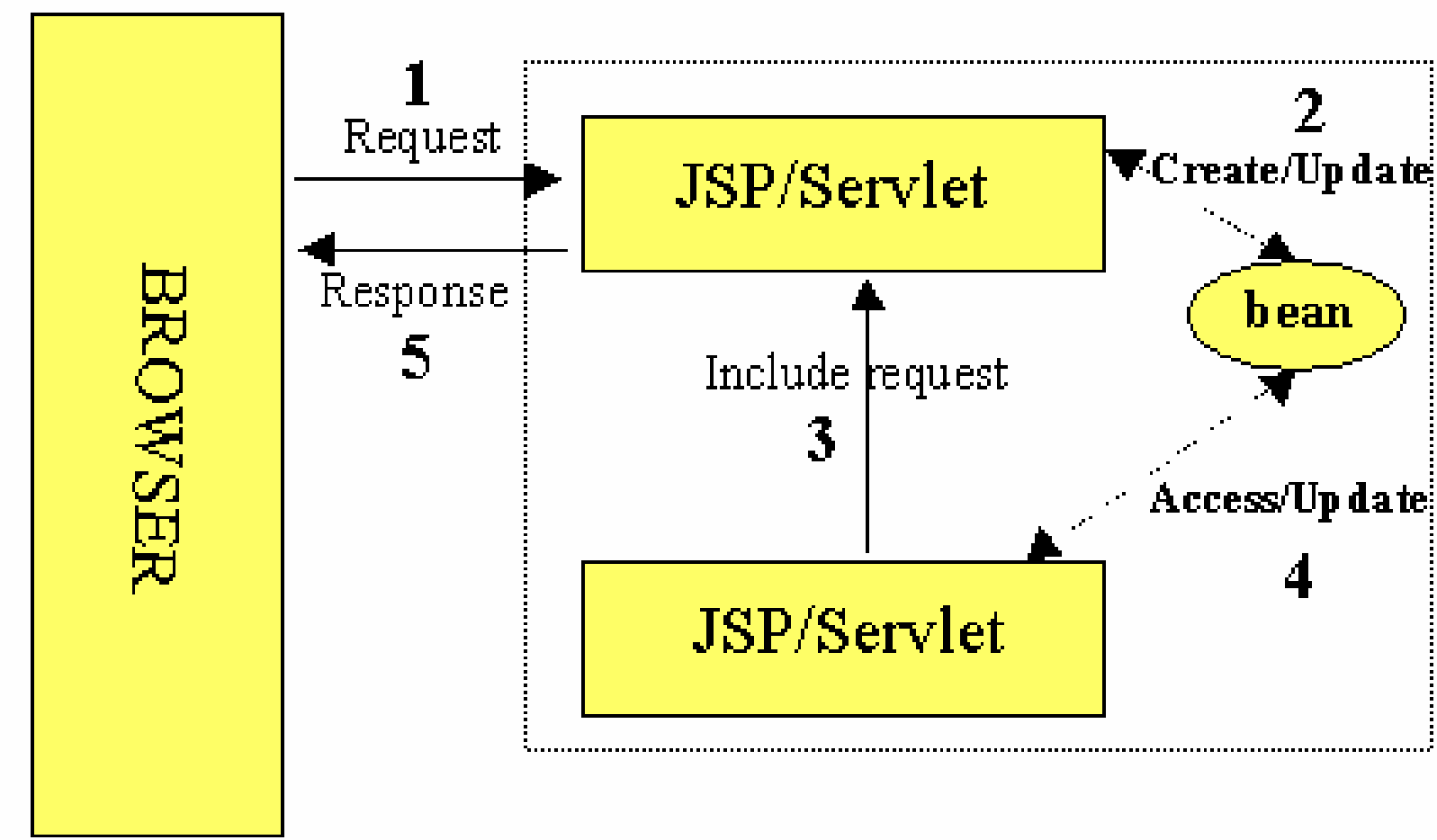
Including Requests

For example:

- *<jsp:include page="shoppingcart.jsp" flush="true"/>*

not only allows shoppingcart.jsp to access any beans placed within the request using a <jsp:useBean> tag, but the dynamic content produced by it is inserted into the calling page at the point where the *<jsp:include>* tag occurs. The included resource, however, cannot set any HTTP headers, which precludes it from doing things like setting cookies, or else an exception is thrown.

Including Requests



Reference

- Stephanie Bodoff, et. al., The J2EE Tutorial, Sun Microsystems.
- James McGovers, et. al., J2EE1.4 Bible, Wiely Publishing Inc.