TRAINING

JAVA DEVELOPER CONNECTION™
java.sun.com

Products & APIs
Developer Connection
Docs & Training
Online Support
Community Discussion
Industry News
Solutions Marketplace
Case Studies
Printable Page

JavaServer Pages™ Fundamentals
About This Short Course

By **jGuru**

[Short Course](#) | [Exercises](#) | [Download](#)

The Java Developer Connection℠ (JDC) presents a Short Course written by Java™ Software licensee jGuru (formerly named the MageLang Institute). This Short Course provides an introduction to JavaServer Pages™ (JSP™) technology.

jGuru has been dedicated to promoting the growth of the Java technology community through evangelism, education, and software since 1995. You can find out more about their activities, including community-driven FAQs and online learning, at [jGuru.com](#).

Please send comments about the course to [jdc-tutorial-feedback@jguru.com](#).

JSP is the way to separate the look and feel of the web page from the underlying business logic, so that it is web server and platform independent. This course teaches you how to use JSP to develop dynamic web sites, by exploring the syntax and components of JSP development.

NOTE: Using JSP with JDBC will be covered in a future JDC course that explains JDBC.

Concepts

After completing this module you will understand the:

- Advantages of JSP technology
- JSP architecture
- Life cycle of a JSP page
- JSP syntax and semantics
- Role of JavaBean™ components within JSP pages

Objectives

By the end of this module you will be able to:

- Manage session-related information from JSP
- Communicate between JSP pages
- Process forms with JSP

Prerequisites

A general familiarity with object-oriented programming concepts and the Java programming language. If you are not familiar with these capabilities, see the Java Tutorial. The exercises require the ability to modify and build simple Java programs and HTML-like pages. It may also help to understand the fundamentals of Web computing and servlets. For help on servlet-specific issues, see the earlier Fundamentals of Java Servlets course, though that is based on the Servlets 2.1 API, instead of the newer 2.2 version.

About the Author

Govind Seshadri is Software Guru for the jGuru portal. He is a frequent speaker on advanced Java development issues at Java/Object conferences worldwide, and has over 10 years of software development experience. Govind is also the author of Enterprise Java Computing: Applications and Architecture from Cambridge University Press (1999).

Start the Short Course

Reader Feedback

Tell us what you think of this tutorial.

Very worth reading   Worth reading   Not worth reading

If you have other comments or ideas for future technical content, please type them here:

Have a question about Java™ programming? Use [Java Online Support.](#)

Printable Page 🖨 [This page was updated: 21-Sep-2000]

[Products & APIs](#) | [Developer Connection](#) | [Docs & Training](#) | [Online Support](#)
[Community Discussion](#) | [Industry News](#) | [Solutions Marketplace](#) | [Case Studies](#)

[Glossary](#) | [Feedback](#) | [A-Z Index](#)

For more information on Java technology
and other software from Sun Microsystems, call:
(800) 786-7638
Outside the U.S. and Canada, dial your country's
[AT&T Direct Access Number](#) first.

Copyright © 1995-2000 [Sun Microsystems, Inc.](#)
All Rights Reserved. [Terms of Use](#). [Privacy Policy](#).

A-Z Index •

JAVA DEVELOPER CONNECTION℠

java.sun.com

TRAINING

Products & APIs

Developer Connection

Docs & Training

Online Support

Community Discussion

Industry News

Solutions Marketplace

Case Studies

Printable Page

JavaServer Pages™ Fundamentals
Exercises

By jGuru

[Short Course| About This Short Course]

Exercise Outline

- About Exercises
    - The Anatomy of an Exercise
    - Exercise Design Goals
- JavaServer Pages Fundamentals Exercises
    - Installing and Configuring Tomcat
    - Exception Handling in JSP
    - Understanding JSP Object Scope
    - Form Processing Using JSP

Welcome to the jGuru exercises for the JavaServer Pages™ Fundamentals Short Course.

These exercises demonstrate how to use Tomcat -- the JSP 1.1 Reference Implementation, as well as how to design, implement, and deploy JSPs.

When you finish these exercises, you will know the basic steps for designing, compiling, and deploying JSP web components.

About Exercises

A jGuru exercise is a flexible exercise that provides varying levels of help according to the student's needs. Some students may complete the exercise using only the information and the task list in the exercise body; some may want a few hints (Help); while others may want a step-by-step guide to successful completion (Solution). Since complete solutions are provided in addition to help, students can skip an exercise and still complete later exercises that required the skipped one(s).

The Anatomy of an Exercise

Each exercise includes a list of any prerequisite exercises, a list of skeleton code to start with, links to necessary API pages, and a text description of the exercise's educational goal. In addition, buttons link you to the following information:

- **Help**: Gives you help or hints on the current exercise, an annotated solution. For ease of use, the task information is duplicated on the help page with the actual help information indented beneath it.
- **Solution**: The <applet> tag and Java source resulting in the expected behavior.
- **API Documentation**: A link directly to any necessary online API documentation.

Exercise Design Goals

There are three fundamental exercise types that you may encounter:

"Blank screen"

You are confronted with a "blank screen" and you create the entire desired functionality yourself.

Extension

You extend the functionality of an existing, correctly-working program.

Repair

You repair undesirable behavior in an existing program.

To make learning easier, exercises, where possible, address only the specific technique being taught in that exercise. Irrelevant, unrelated, and overly complex materials are avoided.

Where possible, exercises execute on the Web. However, exercises that must access Java features or library elements that could cause security violations are not executed on the web.

JavaServer Pages Fundamentals Exercises

Exercises

1. **Installing and Configuring Tomcat**

   This exercise steps you through the process of downloading and installing Tomcat--the JSP 1.1 Reference Implementation (RI). Tomcat comprises of a simple HTTP server as well as a Web container that can run JSP pages and servlets. Tomcat supports the Servlet 2.2 and JSP 1.1 specifications. We use this server for the subsequent exercises.

Educational goal(s):

- ❍ Install Tomcat.
- ❍ Configure your machine properly for compiling and deploying JSPs.

2. Exception Handling in JSP

In this exercise you learn how to redirect runtime exceptions occurring within a JSP page to an error handling page.

Educational goal(s):

- ❍ Learn how to handle runtime exceptions occurring within JSP pages by automatically forwarding them to an error handler page.
- ❍ Understand how exceptions can be accessed from within a JSP error handler page.

3. Understanding JSP Object Scope

In this exercise you observe the behavior of a Counter bean when used within a JSP page with different scope attributes.

Educational goal(s):

- ❍ Understand the importance of the scope attribute when instantiating beans using the useBean tag.
- ❍ Examine the difference between session and application scope.

4. Form Processing Using JSP

In this exercise you learn how to process HTML forms using JSPs, and understand the introspective features provided by the JSP engine.

Educational goal(s):

- ❍ Understand the ease with which HTML forms can be processed using JSP pages.
- ❍ Understand the role played by beans in streamlining form processing.

Printable Page 🖶

[This page was updated: 21-Sep-2000]

Products & APIs | Developer Connection | Docs & Training | Online Support
Community Discussion | Industry News | Solutions Marketplace | Case Studies

Glossary | Feedback | A-Z Index

http://developer.java.sun.com/developer/onlineTraining/JSPIntro/exercises.html (3 of 4) [9/22/2000 8:42:03 AM]

For more information on Java technology
and other software from Sun Microsystems, call:
(800) 786-7638
Outside the U.S. and Canada, dial your country's
AT&T Direct Access Number first.

A-Z Index •

JAVA DEVELOPER CONNECTION™
java.sun.com

TRAINING

Products & APIs
Developer Connection
Docs & Training
Online Support
Community Discussion
Industry News
Solutions Marketplace
Case Studies

Printable Page

JavaServer Pages™ Fundamentals
Short Course

By jGuru™

[About This Short Course](#) | [Exercises](#)

Course Outline

- [Introduction](#)
  - [JSP Advantages](#)
  - [Comparing JSP with ASP](#)
  - [JSP or Servlets?](#)
- [JSP Architecture](#)
- [JSP Access Models](#)
- [JSP Syntax Basics](#)
  - [Directives](#)
    - [Page Directive](#)
    - [Include Directive](#)
  - [Declarations](#)
  - [Expressions](#)
  - [Scriptlets](#)
  - [Comments](#)
- [Object Scopes](#)
- [JSP Implicit Objects](#)
- [Synchronization Issues](#)
- [Exception Handling](#)
- [Session Management](#)
- [Standard Actions](#)
  - [Using JavaBean Components](#)
  - [Forwarding Requests](#)
    - [Request Chaining](#)
  - [Including Requests](#)
- [Web Sites](#)
- [Documentation and Specs](#)
- [Articles](#)

Introduction

While there are numerous technologies for building web applications that serve dynamic content, the one that has really caught the attention of the development community is JavaServer Pages™ (JSP™). And not without ample reason either. JSP not only enjoys cross-platform and cross-Web-server support, but effectively melds the power of server-side Java technology with the WYSIWYG features of static HTML pages.

JSP pages typically comprise of:

- Static HTML/XML components.
- Special JSP tags
- Optionally, snippets of code written in the Java programming language called "scriptlets."

Consequently, you can create and maintain JSP pages by conventional HTML/XML tools.

It is important to note that the JSP specification is a standard extension defined on top of the Servlet API. Thus, it leverages all of your experience with servlets.

There are significant differences between JSP and servlet technology. Unlike servlets, which is a programmatic technology requiring significant developer expertise, JSP appeals to a much wider audience. It can be used not only by developers, but also by page designers, who can now play a more direct role in the development life cycle.

Another advantage of JSP is the inherent separation of presentation from content facilitated by the technology, due its reliance upon reusable component technologies like the JavaBeans™ component architecture and Enterprise JavaBeans™ technology. This course provides you with an in-depth introduction to this versatile technology, and uses the Tomcat JSP 1.1 Reference Implementation from the Apache group for running the example programs.

JSP Advantages

Separation of static from dynamic content: With servlets, the logic for generation of the dynamic content is an intrinsic part of the servlet itself, and is closely tied to the static presentation templates responsible for the user interface. Thus, even minor changes made to the UI typically result in the recompilation of the servlet. This tight coupling of presentation and content results in brittle, inflexible applications. However, with JSP, the logic to generate the dynamic content is kept separate from the static presentation templates by encapsulating it within external JavaBeans components. These 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.

Write Once Run Anywhere: JSP technology brings the "Write Once, Run Anywhere" paradigm to interactive Web pages. 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. Consequently, 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 other B2B applications using XML.

Recommended Web access layer for n-tier architecture: Sun's J2EE™ Blueprints,

which offers guidelines for developing large-scale applications using the enterprise Java APIs, categorically recommends JSP over servlets for serving dynamic content.

Completely leverages the Servlet API: If you are a servlet developer, there is very little that you have to "unlearn" to move over to JSP. In fact, servlet developers are at a distinct advantage because JSP is nothing but a high-level abstraction of servlets. You can do almost anything that can be done with servlets using JSP--but more easily!

Comparing JSP with ASP

Although the features offered by JSP may seem similar to that offered by Microsoft's Active Server Pages (ASP), they are fundamentally different technologies, as shown by the following table:

	JavaServer Pages	Active Server Pages
Web Server Support	Most popular web servers including Apache, Netscape, and Microsoft IIS can be easily enabled with JSP.	Native support only within Microsoft IIS or Personal Web Server. Support for select servers using third-party products.
Platform Support	Platform independent. Runs on all Java-enabled platforms.	Is fully supported under Windows. Deployment on other platforms is cumbersome due to reliance on the Win32-based component model.
Component Model	Relies on reusable, cross-platform components like JavaBeans, Enterprise JavaBeans, and custom tag libraries.	Uses the Win32-based COM component model.
Scripting	Can use the Java programming language or JavaScript.	Supports VBScript and JScript for scripting.
Security	Works with the Java security model.	Can work with the Windows NT security architecture.
Database Access	Uses JDBC for data access.	Uses Active Data Objects for data access.
Customizable Tags	JSP is extensible with custom tag libraries.	Cannot use custom tag libraries and is not extensible.

JSP or Servlets?

It is true that both servlets and JSP pages have many features in common, and can be used for serving up dynamic web content. Naturally, this may cause some confusion as to when to opt for one of the technologies over the other. Luckily, Sun's J2EE Blueprints offers some guidelines towards this.

According to the Blueprints, use servlets strictly as a web server extension technology. This could include the implementation of specialized controller components offering services like authentication, database validation, and so forth. It is interesting to note that what is commonly known as the "JSP engine"

itself is a specialized servlet running under the control of the servlet engine. Since JSP only deals with textual data, you will have to continue to use servlets when communicating with Java applets and applications.

Use JSP to develop typical web applications that rely upon dynamic content. JSP should also be used in place of proprietary web server extensions like server-side includes as it offers excellent features for handling repetitive content.

Exercise

1. [Installing and Configuring Tomcat](#)

JSP Architecture

The purpose of JSP is to provide a declarative, presentation-centric method of developing servlets. As noted before, the JSP specification itself is defined as a standard extension on top the Servlet API. Consequently, it should not be too surprisingly that under the covers, servlets and JSP pages have a lot in common.

Typically, JSP pages are subject to a translation phase and a request processing phase. The translation phase is carried out only once, unless the JSP page changes, in which case it is repeated. Assuming there were no syntax errors within the page, the result is a JSP page implementation class file that implements the Servlet interface, as shown below.
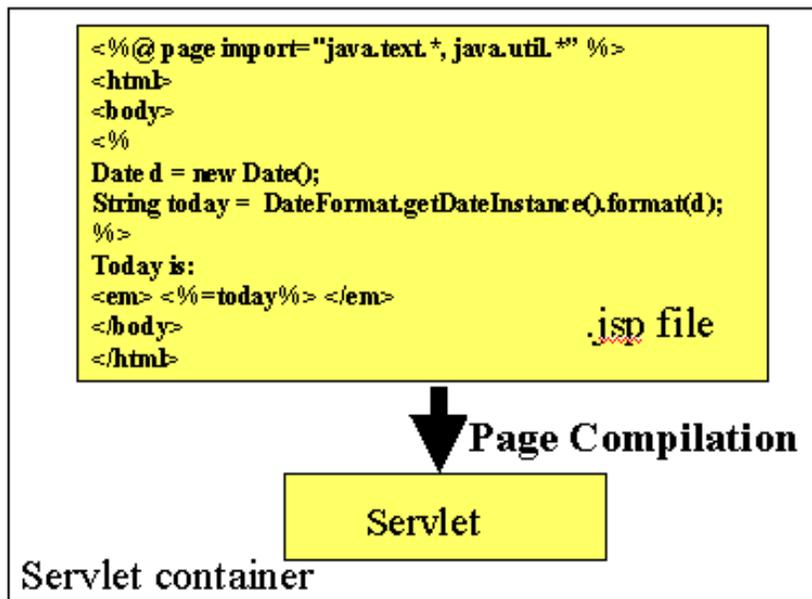
The translation phase is typically carried out by the JSP engine itself, when it receives an incoming request for the JSP page for the first time. Note that the JSP 1.1 specification also allows for JSP pages to be precompiled into class files. Precompilation may be especially useful in removing the start-up lag that occurs when a JSP page delivered in source form receives the first request from a client. Many details of the translation phase, like the location where the source and class files are stored are implementation dependent. The source for the class file generated by Tomcat for this example JSP page (shown in the above figure) is as follows:

```
package jsp;
```

```java
import javax.servlet.*;
import javax.servlet.http.*;
import javax.servlet.jsp.*;
import javax.servlet.jsp.tagext.*;
import java.io.PrintWriter;
import java.io.IOException;
import java.io.FileInputStream;
import java.io.ObjectInputStream;
import java.util.Vector;
import org.apache.jasper.runtime.*;
import java.beans.*;
import org.apache.jasper.JasperException;
import java.text.*;
import  java.util.*;

public class _0005cjsp_0005cjsptest_0002ejspjsptest_jsp_0
        extends HttpJspBase {

 static {
 }
 public _0005cjsp_0005cjsptest_0002ejspjsptest_jsp_0( ) {
 }

 private static boolean _jspx_inited = false;
 public final void _jspx_init() throws JasperException {
 }

 public void _jspService(HttpServletRequest request,
    HttpServletResponse  response)
      throws 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;
    try {
         if (_jspx_inited == false) {
             _jspx_init();
             _jspx_inited = true;
         }
         _jspxFactory = JspFactory.getDefaultFactory();
         response.setContentType("text/html");
         pageContext = _jspxFactory.getPageContext(this,
                 request,response, "", true, 8192, true);

         application = pageContext.getServletContext();
         config = pageContext.getServletConfig();
         session = pageContext.getSession();
         out = pageContext.getOut();
         // begin
         out.write("\r\n<html>\r\n<body>\r\n");
        // end
```

```
              // begin [file="E:\\jsp\\jsptest.jsp";from=(3,2);to=(5,0)]
              Date d = new Date();
              String today =  DateFormat.getDateInstance().format(d);
              // end
              // begin
              out.write("\r\nToday is: \r\n<em> ");
              // end
              // begin [file="E:\\jsp\\jsptest.jsp";from=(7,8);to=(7,13)]
              out.print(today);</b>
              // end
              // begin
              out.write(" </em>\r\n</body>\r\n</html>\r\n");
              // end
        } catch (Exception ex) {
           if (out.getBufferSize() != 0)
           out.clear();
           pageContext.handlePageException(ex);
        } finally {
           out.flush();
           _jspxFactory.releasePageContext(pageContext);
        }
    }
}
```

The JSP page implementation class file extends HttpJspBase, which in turn implements the Servlet interface. Observe how the service method of this class, _jspService(), essentially inlines the contents of the JSP page. 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.

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, as shown below:

JSP Access Models

The early JSP specifications advocated two philosophical approaches, popularly

known as Model 1 and Model 2 architectures, for applying JSP technology. These approaches differ essentially in the location at which the bulk of the request processing was performed, and offer a useful paradigm for building applications using JSP technology.

Consider the Model 1 architecture, shown below:
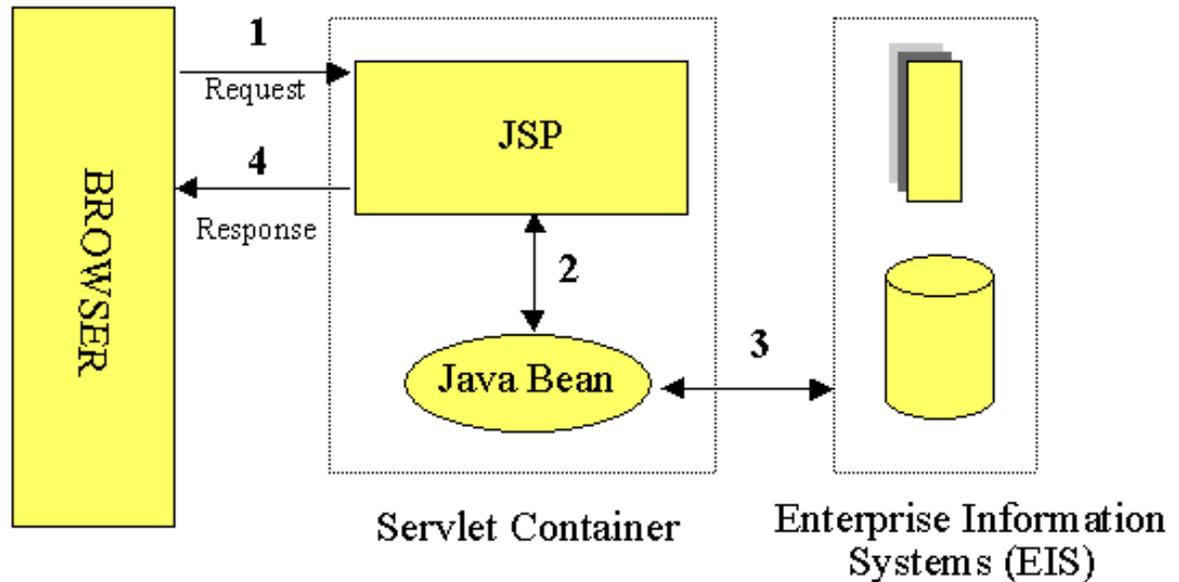
In the Model 1 architecture, the incoming request from a web browser is sent directly to the JSP page, which is responsible for processing it and replying back to the client. There is still separation of presentation from content, because all data access is performed using beans.

Although the Model 1 architecture is suitable for simple applications, it may not be desirable for complex implementations. Indiscriminate usage of this architecture usually leads to a significant amount of scriptlets or Java code embedded within the JSP page, especially if there is a significant amount of request processing to be performed. While this may not seem to be much of a problem for Java developers, it is certainly an issue if your JSP pages are created and maintained by designers--which is usually the norm on large projects. Another downside of this architecture is that each of the JSP pages must be individually responsible for managing application state and verifying authentication and security.
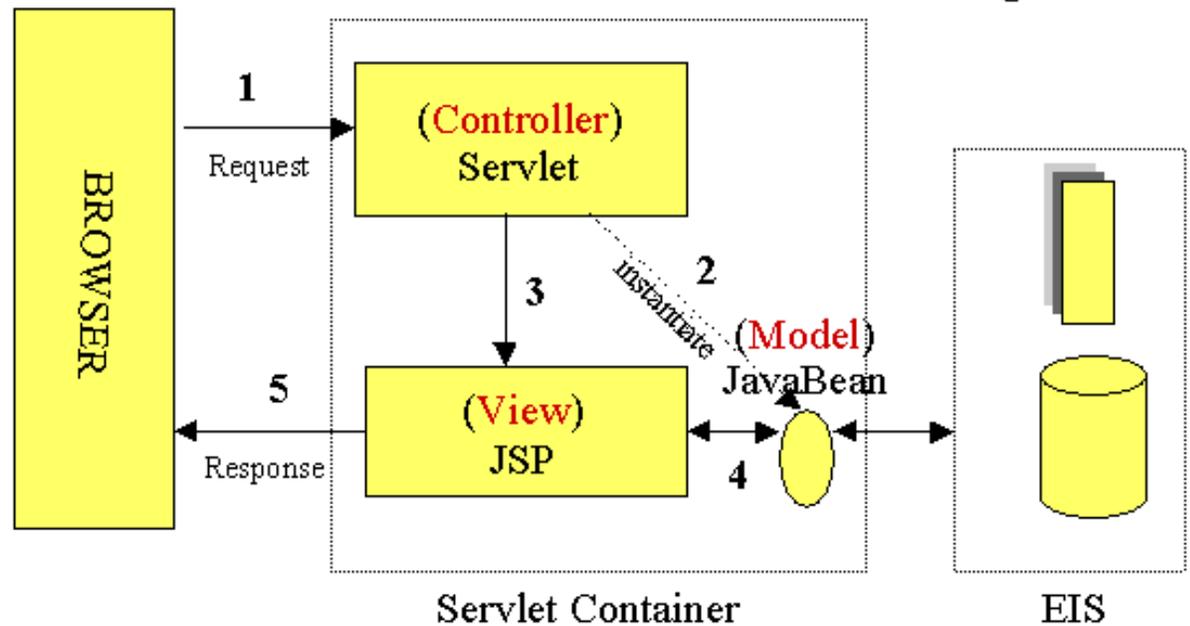
The Model 2 architecture, shown above, is a server-side implementation of the popular Model/View/Controller design pattern. Here, the processing is divided between presentation and front components. Presentation components are JSP pages that generate the HTML/XML response that determines the user interface when rendered by the browser. Front components (also known as controllers) do not handle any presentation issues, but rather, process all the HTTP requests. Here, they are responsible for creating any beans or objects used by the presentation components, as well as deciding, depending on the user's actions, which presentation component to forward the request to. Front components can be implemented as either a servlet or JSP page.

The advantage of this architecture is that there is no processing logic within the presentation component itself; it is simply responsible for retrieving any objects or beans that may have been previously created by the controller, and extracting the dynamic content within for insertion within its static templates. Consequently, this clean separation of presentation from content leads to a clear delineation of the roles and responsibilities of the developers and page designers on the programming team. Another benefit of this approach is that the front components present a single point of entry into the application, thus making the management of application state, security, and presentation uniform and easier to maintain.

JSP Syntax Basics

JSP syntax is fairly straightforward, and can be classified into directives, scripting elements, and standard actions.

Directives

JSP directives are messages for the JSP engine. They do not directly produce any visible output, but tell the engine what to do with the rest of the JSP page. JSP directives are always enclosed within the <%@ ... %> tag. The two primary directives are page and include. (Note that JSP 1.1 also provides the taglib directive, which can be used for working with custom tag libraries, although this isn't discussed here.)

Page Directive

Typically, the page directive is found at the top of almost all of your JSP pages. There can be any number of page directives within a JSP page, although the attribute/value pair must be unique. Unrecognized attributes or values result in a translation error. For example,

```
<%@ page import="java.util.*, com.foo.*" buffer="16k" %>
```

makes available the types declared within the included packages for scripting and sets the page buffering to 16K.

Include Directive

The include directive lets you separate your content into more manageable elements, such as those for including a common page header or footer. The page included can be a static HTML page or more JSP content. For example, the directive:

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

can be used to include the contents of the indicated file at any location within the JSP page.

Declarations

JSP declarations let you define page-level variables to save information or define supporting methods that the rest of a JSP page may need. While it is easy to get led away and have a lot of code within your JSP page, this move will eventually turn out to be a maintenance nightmare. For that reason, and to improve reusability, it is best that logic-intensive processing is encapsulated as JavaBean components.

Declarations are found within the <%! ... %> tag. Always end variable declarations with a semicolon, as any content must be valid Java statements:

```
<%! int i=0; %>
```

You 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

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

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

Scriptlets

JSP code fragments or scriptlets are embedded within <% ... %> tags. This 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

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

Before we look at JSP syntax and semantics, it is important to understand the scope or visibility of Java objects within JSP pages that are processing a request. Objects may be created implicitly using JSP directives, explicitly through actions, or, in rare cases, directly using scripting code. The instantiated objects can be associated with a scope attribute defining where there is a reference to the object and when that reference is removed. The following diagram indicates the various scopes that can be associated with a newly created object:
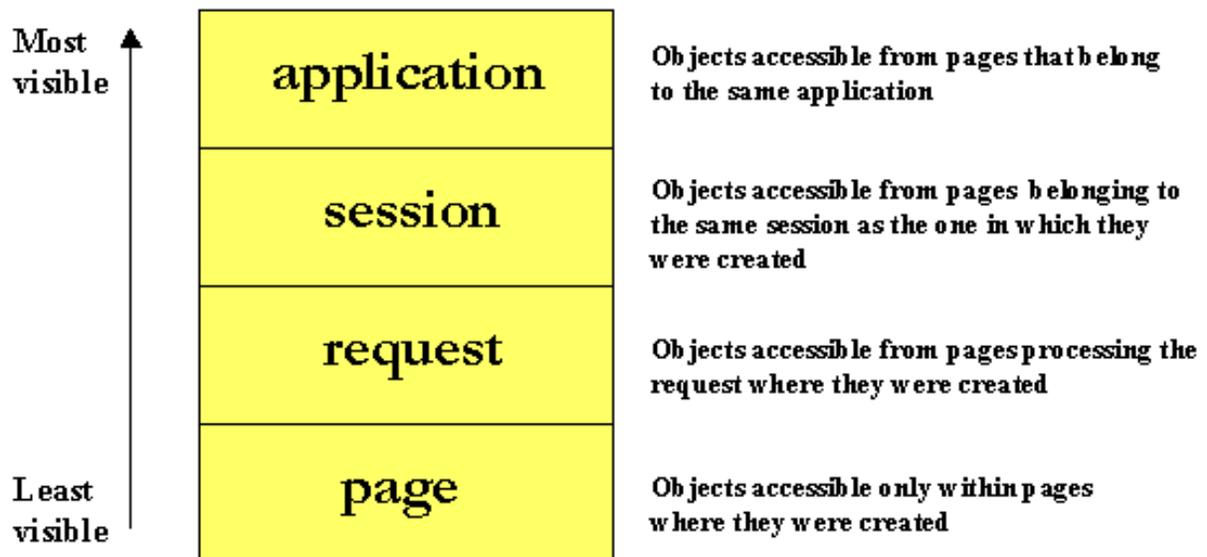
JSP Implicit Objects

As a convenience feature, the JSP container makes available implicit objects that can be used within scriptlets and expressions, without the page author first having to create them. These objects act as wrappers around underlying Java

classes or interfaces typically defined within the Servlet API. The nine implicit objects:

- request: represents the HttpServletRequest triggering the service invocation. Request scope.
- response: represents HttpServletResponse to the request. Not used often by page authors. Page scope.
- pageContext: encapsulates implementation-dependent features in PageContext. Page scope.
- application: represents the ServletContext obtained from servlet configuration object. Application scope.
- out: a JspWriter object that writes into the output stream. Page scope.
- 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. More on sessions shortly.
- exception: the uncaught Throwable object that resulted in the error page being invoked. Page scope.

Note that 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:

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

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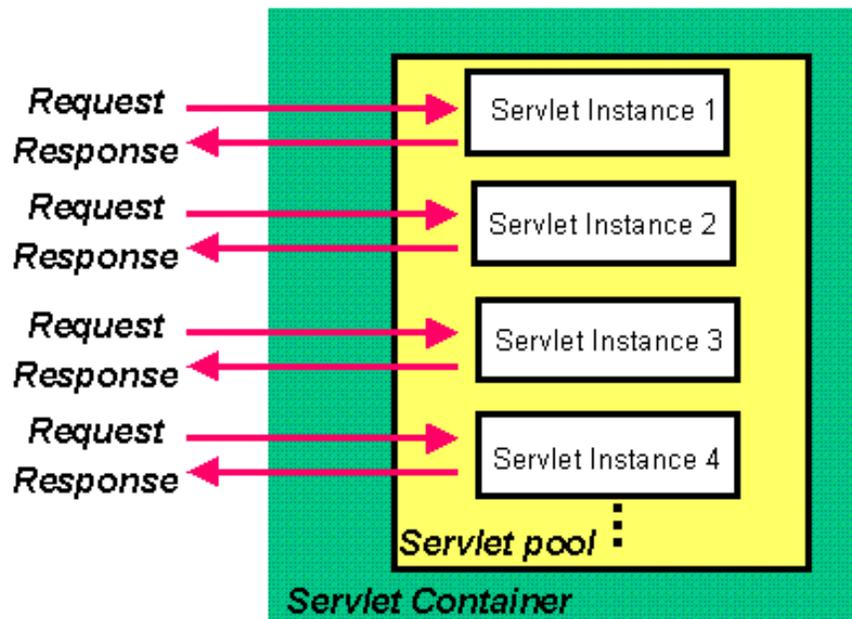
The downside of using this approach is that it is not scalable. If the wait queue grows due to a large number of concurrent requests overwhelming the processing ability of the servlet instances, then the client may suffer a significant delay in obtaining the response.

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. 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:

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

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

Exercise

2. Exception Handling in JSP

Session Management

By default, all JSP pages participate in an HTTP session. The HttpSession object can be accessed within scriptlets through the session implicit JSP object. 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. The session objects is identified by a session ID and stored in the browser as a cookie. 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. For example:

```
<%
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");
%>
```

The call to session.getValue() returns a reference to the generic Object type. Thus it is important to always cast the value returned to the appropriate data type before using it. It is not mandatory for JSP pages to participate in a session; they may choose to opt out by setting the appropriate attribute of the page directive:

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

There is no limit on the number of objects you can store into the session. However, placing large objects into the session may degrade performance, as they take up valuable heap space. By default, most servers set the lifetime of a session object to 30 minutes, although you can easily reset it on a per session basis by invoking setMaxInvalidationInterval(int secs) on the session object. The figure below highlights the general architecture of session management:
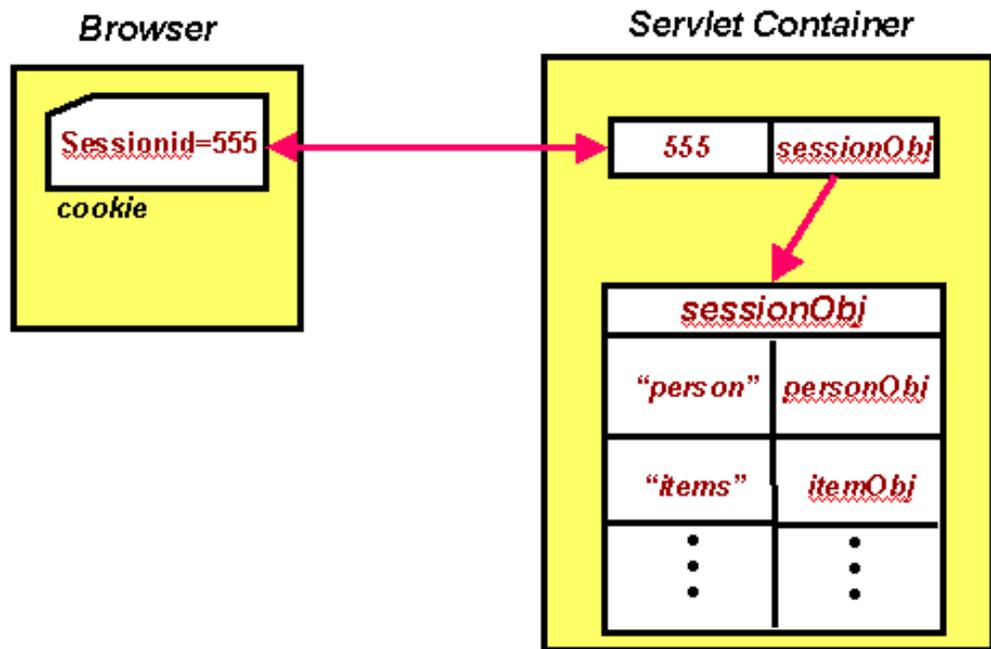
The JSP engine holds a live reference to objects placed into the session as long as the session is valid. If the session is invalidated or encounters a session timeout, then the objects within are flagged for garbage collection.

Standard Actions

Actions allow you to perform sophisticated tasks like instantiating objects and communicating with server-side resources like JSP pages and servlets without requiring Java coding. Although the same can be achieved using Java code within scriptlets, using action tags promotes reusability of your components and enhances the maintainability of your application.

Using JavaBean Components

The component model for JSP technology is based on JavaBeans component architecture. JavaBeans components are nothing but Java objects which follow a well-defined design/naming pattern: the bean encapsulates its properties by declaring them private and provides public accessor (getter/setter) methods for reading and modifying their values.

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. Consider the tag:

```
<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.

The <jsp:useBean> tag can also optionally include a body, such as

```
<jsp:useBean id="user" class="com.jguru.Person"
   scope="session">
<%
  user.setDate(DateFormat.getDateInstance(
                        ).format(new Date()));
  //etc..
%>
</jsp:useBean>
```

Any scriptlet (or <jsp:setProperty> tags, which are explained shortly) present within the body of a <jsp:useBean> tag are executed only when the bean is instantiated, and are used to initialize the bean's 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" />
```

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="jGuru" />
```

or

```
<jsp:setProperty name="user" property="name"
  value="<%=expression %>" />
```

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="*"/>
```

This runtime magic is possible through a process called introspection, which lets a class expose its properties on request. The introspection is managed by the JSP engine, and implemented through the Java reflection mechanism. This feature alone can be a lifesaver when processing complex forms containing a significant number of input elements.

If the names of your bean properties do not match those of the form's input elements, they can still be mapped explicitly to your property by naming the parameter as:

```
<jsp:setProperty name="user" property="address"
  param="parameterName" />
```

Exercises

Forwarding Requests

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" />
```

The invoking page can also pass the target resource bean parameters by placing them into the request, as shown in the diagram:
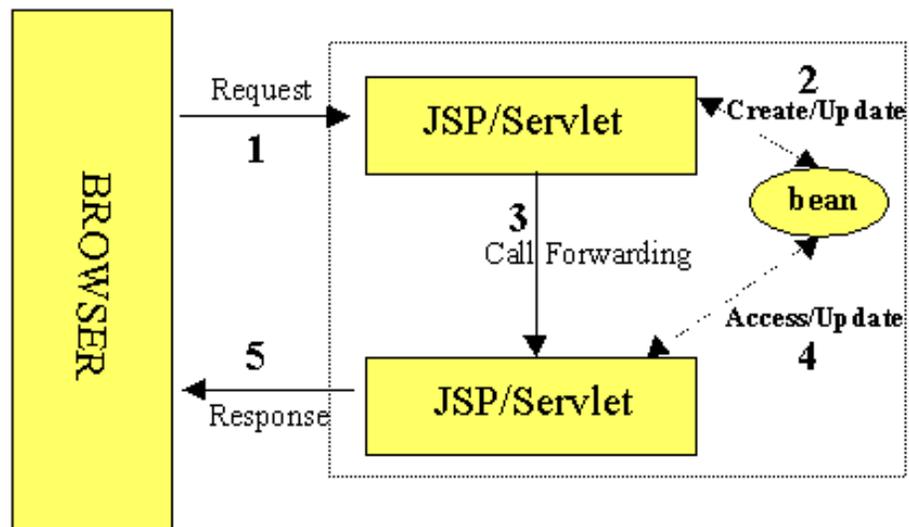
A <jsp:forward> tag may also have jsp:param subelements that can provide values for some elements in the request used in the forwarding:

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

Request Chaining

Request chaining is a powerful feature and can be used to effectively meld JSP pages and servlets in processing HTML forms, as shown in the following figure:
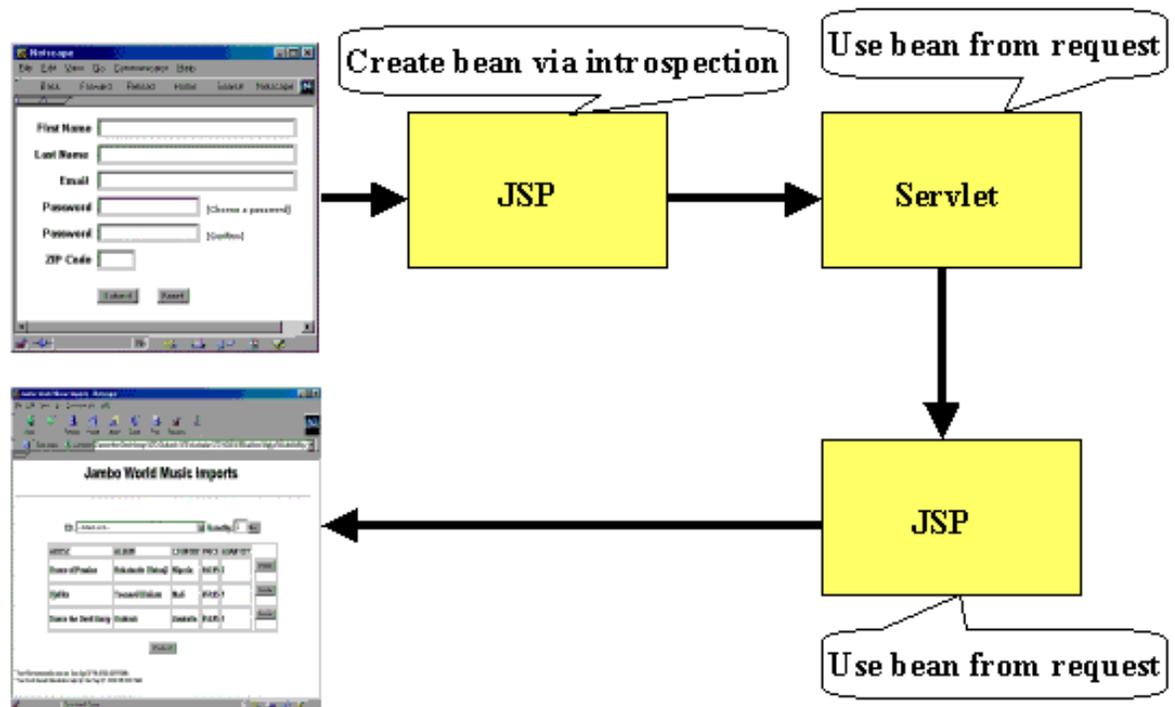
Consider the following JSP page, say Bean1.jsp, which creates a named instance fBean of type FormBean, places it in the request, and forwards the call to the servlet JSP2Servlet. Observe the way the bean is instantiated--here we automatically call the bean's setter methods for properties which match the names of the posted form elements, while passing the corresponding values to the methods.

```
<jsp:useBean id="fBean" class="govi.FormBean"
  scope="request"/>
<jsp:setProperty name="fBean" property="*" />
<jsp:forward page="/servlet/JSP2Servlet" />
```

The servlet JSP2Servlet now extracts the bean passed to it from the request, makes changes using the appropriate setters, and forwards the call to another JSP page Bean2.jsp using a request dispatcher. Note that this servlet, acting as a controller, can also place additional beans if necessary, within the request.

```
public void doPost (HttpServletRequest request,
                    HttpServletResponse response) {
  try {
    FormBean f = (FormBean) request.getAttribute
                                      ("fBean");
    f.setName("Mogambo");
    // do whatever else necessary
    getServletConfig().getServletContext().
      getRequestDispatcher("/jsp/Bean2.jsp").
        forward(request, response);
  } catch (Exception ex) {
    . . .
  }
}
```

The JSP page Bean2.jsp can now extract the bean fBean (and whatever other beans that may have been passed by the controller servlet) from the request and extract its properties.
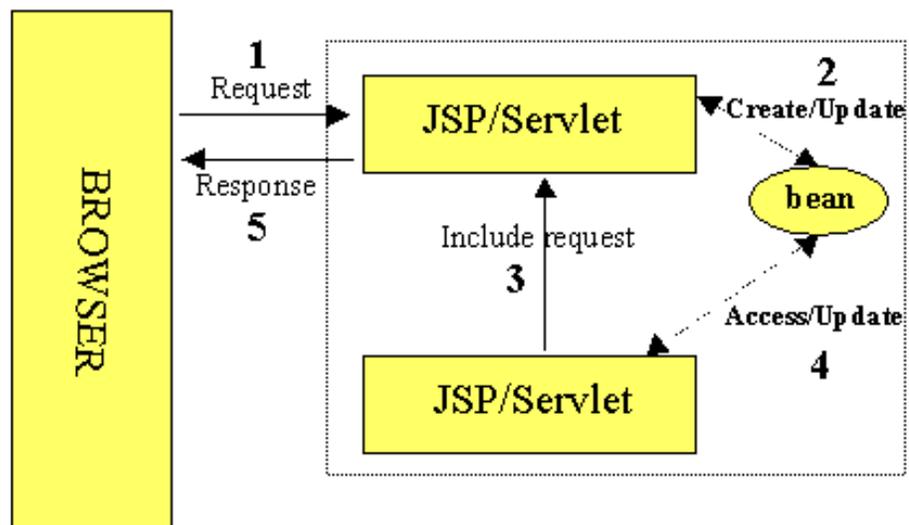
```
<html>
<body>
<jsp:useBean id="fBean" class="govi.FormBean"
   scope="request"/>
<jsp:getProperty name="fBean" property="name" />
</body>
</html>
```

Including Requests

The <jsp:include> tag can be used to redirect the request to any static or dynamic resource that is in the same context as the calling JSP page. The calling page can also pass the target resource bean parameters by placing them into the request, as shown in the diagram:

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.

Web Sites

The following sites have product information as well as whitepapers on JSP and Servlets:

- [Sun Microsystems, JSP Home Page](#)
- [JSP-INTEREST Mailing List Archive](#)
- [jGuru's JSP FAQ](#)
- [jGuru's Servlets FAQ](#)

Documentation and Specs

The Java Technology site at Sun Microsystems includes a Products and APIs page which lists enterprise-related products and APIs. Several of the ones relevant to JSP are listed here:

- JSP 1.1 Specification
- Sun Microsystems, Inc. Java 2 Enterprise Edition (J2EE) Home page
- The Tomcat Project
- JSP Technical Resources
- Java Servlet API
- JSP Whitepaper
- JSP Syntax Card

Articles

Some articles on JSP computing include:

- Advanced Form Processing using JSP by Govind Seshadri (JavaWorld, March 2000)
- JSP Architectures by Lance Lavandowska, brainopolis.com
- Java serves the Web by John Zukowski, (Builder.com, February 2000)
- Internationalize JSP-based Websites by Govind Seshadri (JavaWorld, February 2000)
- The Problems with JSP by Jason Hunter (Servlets.com, January, 2000)
- Understanding JSP Model 2 Architecture by Govind Seshadri (JavaWorld, December 1999)
- JSP for the ASP Developer by Cindy Nordahl (ASP Today, October 1999)

Printable Page

[This page was updated: 21-Sep-2000]

Products & APIs | Developer Connection | Docs & Training | Online Support
Community Discussion | Industry News | Solutions Marketplace | Case Studies

Glossary | Feedback | A-Z Index

For more information on Java technology
and other software from Sun Microsystems, call:
(800) 786-7638
Outside the U.S. and Canada, dial your country's
AT&T Direct Access Number first.

A-Z Index •

JAVA DEVELOPER CONNECTION
java.sun.com

TRAINING

Products & APIs
Developer Connection
Docs & Training
Online Support
Community Discussion
Industry News
Solutions Marketplace
Case Studies

Printable Page

Installing and Configuring Tomcat

by **jGuru**

[Help](#) | [API Docs](#) | [Short Course](#)| [Exercises](#)

This exercise steps you through the process of downloading and installing Tomcat--the JSP 1.1 Reference Implementation (RI). The exercises are specific to Tomcat--if you want to use a different server for the remainder of these exercises, you should ensure that it is JSP 1.1 compliant and install it now.

Prerequisites

None

Tasks

1. Check your system requirements to make sure you have an adequate hardware and software for installing and running Tomcat.

2. Download the appropriate version of Tomcat 3.1 from the Apache [website](#).

3. Uncompress the file.

4. Set the environment variable JAVA_HOME to point to the root directory of your JDK hierarchy. Be sure the Java interpreter is in your PATH environment variable.

5. Change to the bin directory and start Tomcat using the command-line command startup.

6. Tomcat is now installed and running on port 8080 by default. Explore the Tomcat documentation within the [documentation site](#) to familiarize yourself more with Tomcat.

Where help exists, the task numbers above are linked to the step-by-step [help](#) page.

Solution Source

There is no solution to this exercise. When the tasks in this exercise have been completed, Tomcat will be installed, running, and available for the subsequent exercises.

Demonstration

When you complete the tasks in this exercise, Tomcat is installed and running and available for the subsequent exercises.

Running Tomcat using the command-line command startup should produce output similar to the following:

```
myhost> startup

Tomcat Web Server Version 3.0
Loaded configuration from: file:E:/tomcat/server.xml
Configuring web service using "default"
Configuring web service using
  "file:E:/tomcat/examples/WEB-INF/web.xml"
default: init
jsp: init
Configuring web service using "default"
Configuring web service using
  "file:E:/tomcat/webpages/WEB-INF/web.xml"
default: init
jsp: init
Starting tcp endpoint on 8080 with
 org.apache.tomcat.service.http.HttpConnectionHandler
Starting tcp endpoint on 8007 with
 org.apache.tomcat.service.connector.Ajp12ConnectionHandler
```

Next Exercise

Exercises

Short Course

Copyright 1996-2000 jGuru.com. All Rights Reserved.

Printable Page

[This page was updated: 21-Sep-2000]

Products & APIs | Developer Connection | Docs & Training | Online Support
Community Discussion | Industry News | Solutions Marketplace | Case Studies

Glossary | Feedback | A-Z Index

http://developer.java.sun.com/developer/onlineTraining/JSPIntro/exercises/SetupTomcat/index.html (2 of 3) [9/22/2000 8:44:04 AM]

For more information on Java technology
and other software from Sun Microsystems, call:
(800) 786-7638
Outside the U.S. and Canada, dial your country's
AT&T Direct Access Number first.

A-Z Index •

JAVA DEVELOPER CONNECTION™
java.sun.com

TRAINING

Installing and Configuring Tomcat

by **jGuru**

Exercise | API Docs | Short Course | Exercises

Help is available for each task.

Task 1

Check your system requirements to make sure you have an adequate hardware and software platform for installing and running Tomcat.

Tomcat has been tested under the following platforms: Windows NT 4.0 Service Pack 4, Solaris 2.6 and 2.7, Windows 98, and Red Hat Linux 6.0. You will also need to install JDK 1.1.8 or the Java 2 SDK version onto your system.

Task 2

Download the appropriate version of Tomcat 3.1 from the Apache website.

Either grab the ZIP or tar.gz version. Most Windows users should just grab the ZIP version, though **WinZip** can read the smaller tar.gz version.

Task 3

Uncompress the file.

This should create a new subdirectory named jakarta-tomcat.

Task 4

Set the environment variable JAVA_HOME to point to the root directory of your JDK hierarchy. Be sure the Java interpreter is in your PATH environment variable.

Something like the following will work for Windows, depending upon

the JDK installation directory:

```
SET JAVA_HOME=C:\jdk1.2.2
```

Task 5

Change to the bin directory and start Tomcat using the command-line command startup.

Running Tomcat using the command-line command startup should produce output similar to the following:

```
myhost> startup

Tomcat Web Server Version 3.1
Loaded configuration from: file:E:/tomcat/server.xml
Configuring web service using "default"
Configuring web service using
    "file:E:/tomcat/examples/WEB-INF/web.xml"
default: init
jsp: init
Configuring web service using "default"
Configuring web service using
    "file:E:/tomcat/webpages/WEB-INF/web.xml"
default: init
jsp: init
Starting tcp endpoint on 8080 with
 org.apache.tomcat.service.http.HttpConnectionHandler
Starting tcp endpoint on 8007 with
 org.apache.tomcat.service.connector.Ajp12ConnectionHandler
```

Task 6

Tomcat is now installed and running on port 8080 by default. Explore the Tomcat documentation within the documentation site to familiarize yourself more with Tomcat.

There is extensive documentation on all aspects of JSP, along with example code, bundled with your Tomcat install. For additional information, please see Sun's JSP technical resource page

Copyright 1996-2000 jGuru.com. All Rights Reserved.

Printable Page 🖨

[This page was updated: 21-Sep-2000]

Products & APIs | Developer Connection | Docs & Training | Online Support
Community Discussion | Industry News | Solutions Marketplace | Case Studies

Glossary | Feedback | A-Z Index

http://developer.java.sun.com/developer/onlineTraining/JSPIntro/exercises/SetupTomcat/help.html (2 of 3) [9/22/2000 8:44:10 AM]

For more information on Java technology
and other software from Sun Microsystems, call:
(800) 786-7638
Outside the U.S. and Canada, dial your country's
AT&T Direct Access Number first.

JAVA DEVELOPER CONNECTION™
java.sun.com

TRAINING

Products & APIs

Developer Connection

Docs & Training

Online Support

Community Discussion

Industry News

Solutions Marketplace

Case Studies

Printable Page

Exception Handling in JSP

by jGuru®

[Help | API Docs | Short Course | Exercises]

This exercise implements a JSP page (errhandler.jsp), which processes a POST operation and throws an exception in case of an "incorrect" answer. You will see how these exceptions can be automatically forwarded by the JSP engine to an "error handler." You also develop an error processing JSP page (errorpage.jsp), which receives the exception by means of the exception implicit variable.

Prerequisites

- **Installing and Configuring Tomcat**

Skeleton Code

- errhandler.jsp
- errorpage.jsp

Tasks

1.

   Design a JSP page called errhandler.jsp that can process a POST
2. operation.

   Indicate an error page, errorpage.jsp, using the page directive for
3. the JSP page.

   Process the posted form elements. Throw an exception if the value posted for the input element is not equal to an expected
4. value, else print an acknowledgment back to the user.

   Develop an error page, errorpage.jsp, which can access the
5. runtime exception.

6. Deploy the JSP files for the example within Tomcat.

   Run the error handling example.

Where help exists, the task numbers above are linked to the

step-by-step [help](#) page.

Solution Source

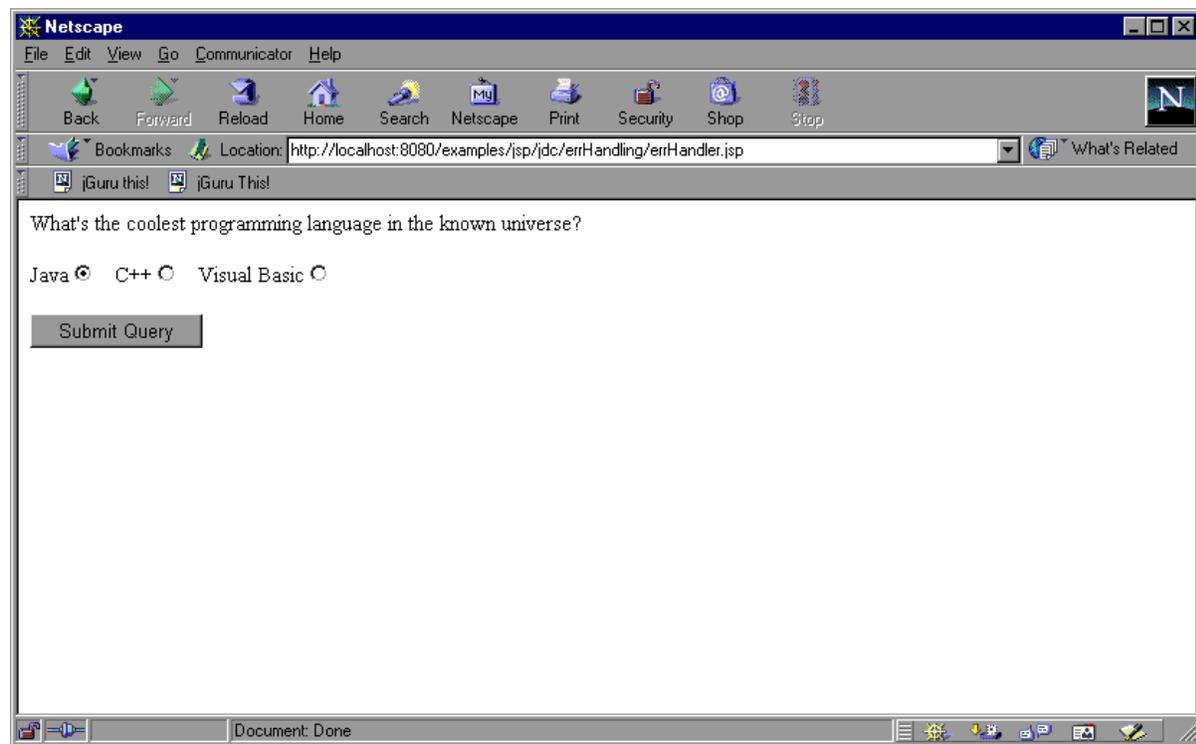The following files contain a complete implementation of the JSP error handling example:

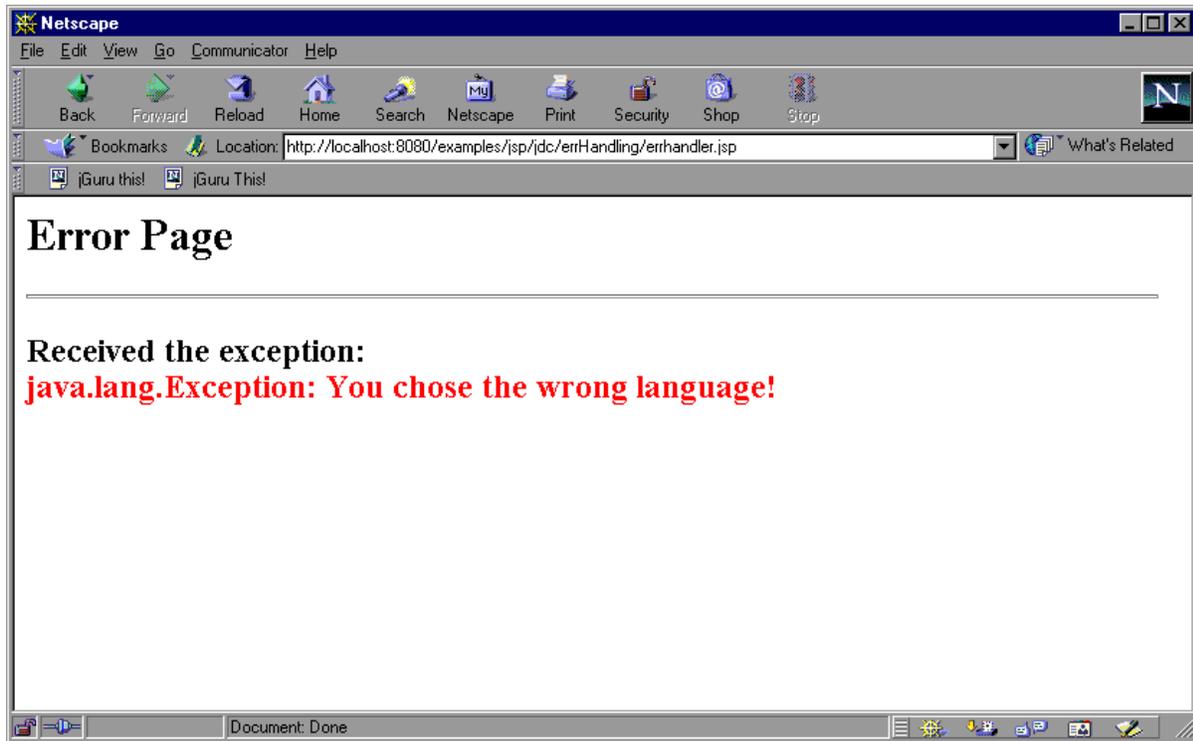- [errhandler.jsp](#)
- [errorpage.jsp](#)

Demonstration

From your browser, access the URL
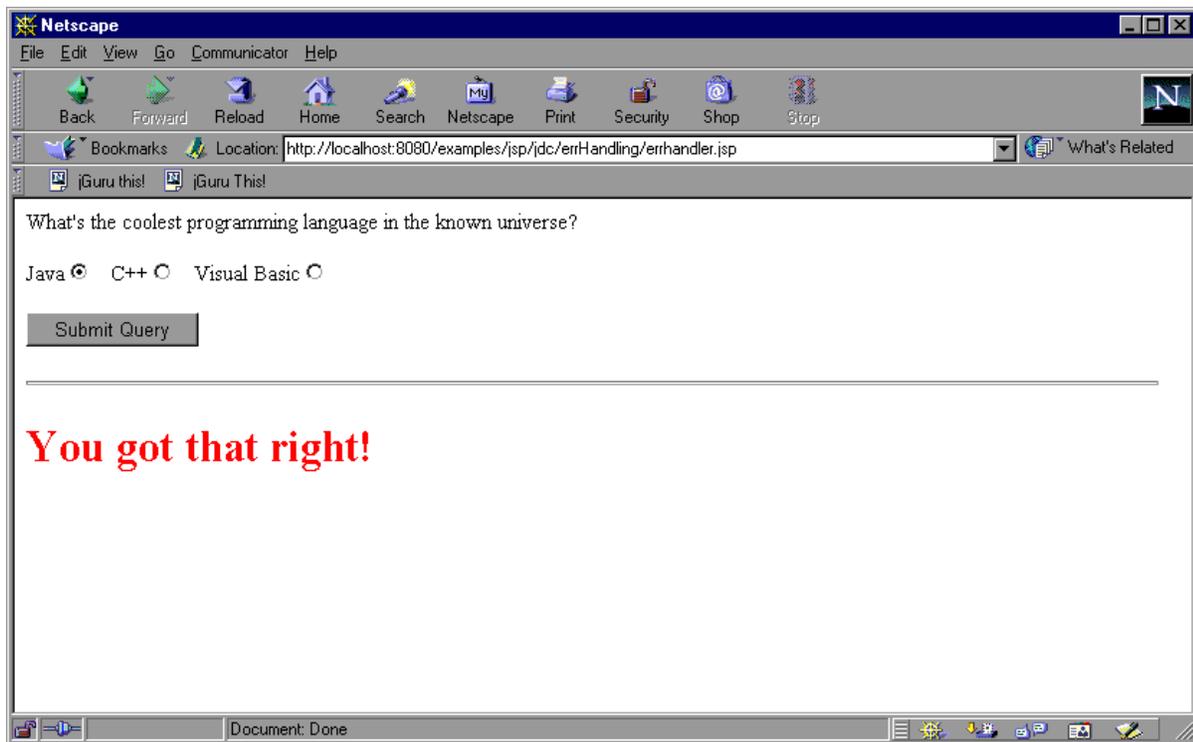http://localhost:8080/examples/jsp/jdc/errHandling/errhandler.jsp

You should see an HTML form as shown below:

Now, make a selection and submit the form. If you made an incorrect selection, an exception is thrown and the request is forwarded to the error handler page, which extracts and displays the exception:

Error Page

Received the exception:
java.lang.Exception: You chose the wrong language!

If you made the right choice, you get an acknowledgment from the JSP page itself:

What's the coolest programming language in the known universe?

Java ⦿   C++ ○   Visual Basic ○

Submit Query

You got that right!

Next Exercise

Exercises

Short Course

Copyright 1996-2000 jGuru.com. All Rights Reserved.

Printable Page 🖨

[This page was updated: 21-Sep-2000]

For more information on Java technology
and other software from Sun Microsystems, call:
(800) 786-7638
Outside the U.S. and Canada, dial your country's
AT&T Direct Access Number first.

JAVA DEVELOPER CONNECTION™
java.sun.com

TRAINING

Products & APIs
Developer Connection
Docs & Training
Online Support
Community Discussion
Industry News
Solutions Marketplace
Case Studies

Printable Page

Exception Handling in JSP

by jGuru®

[Exercise | API Docs | Short Course | Exercises]

Help is available for each task.

Task 1

Design a JSP page called errhandler.jsp that can process a POST operation.

For convenience, the HTML form can be developed as part of the JSP page itself. Note that the form's ACTION clause must point to the JSP page itself. Also, make sure that the form contains an input element like a radio button.

Task 2

Indicate an error page, errorpage.jsp, using the page directive for the JSP page.

Uncaught runtime exceptions are automatically forwarded to an error page, if specified, by the JSP engine. The location of the error page can be specified using the errorPage attribute of the page tag.

Task 3

Process the posted form elements. Throw an exception if the value posted for the input element is not equal to an expected value, else print an acknowledgement back to the user.

The form elements can be obtained using the request.getParameter() method. You can echo a message to the browser using response.println(). You can throw a generic exception using new Exception("some message").

Task 4

Develop an error page, errorpage.jsp, which can access the runtime exception.

A JSP error page is developed by setting the isErrorPage attribute of the page tag to true. The runtime exception can be accessed by means of the JSP implicit variable exception.

Task 5

Deploy the JSP files for the example within Tomcat.

Assuming you have installed Tomcat in say, \jakarta-tomcat, copy the two JSP files to \jakarta-tomcat\examples\jsp\jdc\errHandling\

Task 6

Run the error handling example.

From your browser, access the URL http://localhost:8080/examples/jsp/jdc/errHandling/errhandler.jsp

Printable Page

[This page was updated: 21-Sep-2000]

Products & APIs | Developer Connection | Docs & Training | Online Support
Community Discussion | Industry News | Solutions Marketplace | Case Studies

---

Glossary | Feedback | A-Z Index

For more information on Java technology
and other software from Sun Microsystems, call:
(800) 786-7638
Outside the U.S. and Canada, dial your country's
AT&T Direct Access Number first.

TRAINING

JAVA DEVELOPER CONNECTION™
java.sun.com

Products & APIs
Developer Connection
Docs & Training
Online Support
Community Discussion
Industry News
Solutions Marketplace
Case Studies

Printable Page

Understanding JSP Object Scope

by jGuru®

[[Help](#) | [API Docs](#) | [Short Course](#)| [Exercises](#)]

This exercise implements a simple JSP page (Counter.jsp), which instantiates two instances of a bean which maintains a counter (CounterBean.java), but with differing scope. One bean is attributed with session scope, and the other with application scope. Each time the JSP page in invoked, the count of each of the beans is incremented by one. You can observe the difference between session and application scope when you access the counter page from different browsers. Each browser maintains a distinct count for their session, but share the counter with application scope, since it is treated as a global variable.

Prerequisites

- [Installing and Configuring Tomcat](#)

Skeleton Code

- [Counter.jsp](#)
- [CounterBean.java](#)

Tasks

1.

2. Develop a simple counter bean, CounterBean.java.

3. Compile the counter bean.

4. Deploy the bean within Tomcat.

5. Develop a JSP page, Counter.jsp, which creates two instances of the counter bean, one with session scope, and the other with application scope.

6. Deploy the JSP file for the example within Tomcat.

Run the example.

Where help exists, the task numbers above are linked to the step-by-step help page.
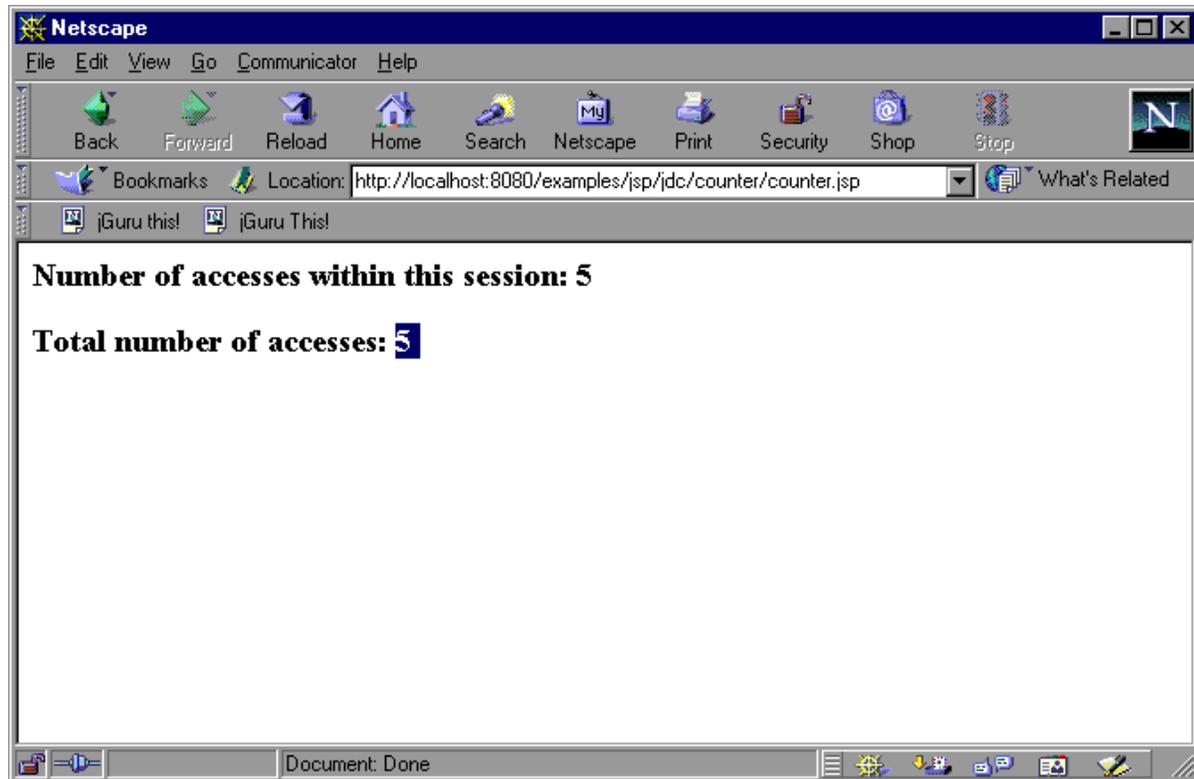
Solution Source

The following files contain a complete implementation of the example demonstrating JSP variable scope:

- Counter.jsp
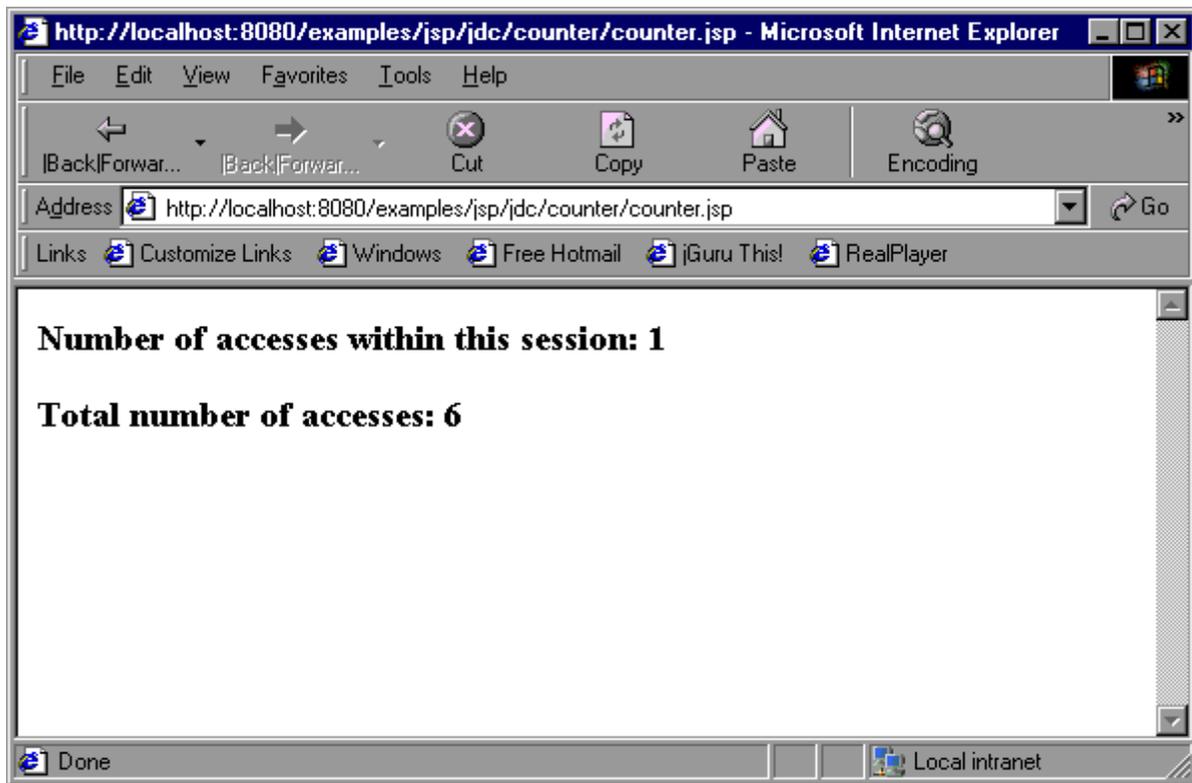- CounterBean.java

Demonstration

From your browser (say, Netscape Navigator), access the URL http://localhost:8080/examples/jdc/counter/Counter.jsp

Reload the page a few times. You should see the counters incremented as shown below:

From using a different browser (say, MSIE), access the same URL. Observe the difference in the counts:

Number of accesses within this session: 1

Total number of accesses: 6

Next Exercise

Exercises

Short Course

Copyright 1996-2000 jGuru.com. All Rights Reserved.

Printable Page 🖨   [This page was updated: 21-Sep-2000]

Products & APIs | Developer Connection | Docs & Training | Online Support
Community Discussion | Industry News | Solutions Marketplace | Case Studies

---

Glossary | Feedback | A-Z Index

For more information on Java technology
and other software from Sun Microsystems, call:
(800) 786-7638
Outside the U.S. and Canada, dial your country's
AT&T Direct Access Number first.

Copyright © 1995-2000 Sun Microsystems, Inc.
All Rights Reserved. Terms of Use. Privacy Policy.

A-Z Index •

JAVA DEVELOPER CONNECTION™
java.sun.com

TRAINING

Understanding JSP Object Scope

by **jGuru**

[[Exercise](#) | [API Docs](#) | [Short Course](#)| [Exercises](#)]

Products & APIs

Developer Connection

Docs & Training

Online Support

Community Discussion

Industry News

Solutions Marketplace

Case Studies

Printable Page

Help is available for each task.

Task 1

Develop a simple counter bean, CounterBean.java.

Develop a bean which can maintain a count within say, an integer property. Provide at least a getter method for this property.

Task 2

Compile the counter bean.

You can compile the bean as javac CounterBean.java

Task 3

Deploy the bean within Tomcat.

Copy CounterBean.class to
jakarta-tomcat\webapps\examples\WEB-INF\jsp\classes\com\jguru\CounterBean.class.
You will need to create the directories below classes for the package the
bean is in.

Task 4

Develop a JSP page, Counter.jsp, which creates two instances of the
counter bean, one with session scope, and the other with application
scope.

You can use the jsp:useBean tag for instantiating the beans. Make sure
you provide the appropriate scope for the scope attribute.

Task 5

Deploy the JSP file for the example within Tomcat.

Assuming you have installed Tomcat in say, \jakarta-tomcat, copy the JSP file to \jakarta-tomcat\webapps\examples\jsp\jdc\counter\Counter.jsp

Task 6

Run the example.

From your browser (say, Netscape Navigator), access the URL
http://localhost:8080/examples/jsp/jdc/counter/Counter.jsp

Now, start up a different browser (say, MSIE) access the URL
http://localhost:8080/examples/jsp/jdc/counter/Counter.jsp

(Note: If you use only MSIE, you can simply double click on the browser icon again to run a second instance of MSIE as a separate process. This is important to ensure that the browser creates a new session and does not reuse the one created by an earlier instance.)

Click a few times within both browser to increment the counters for the beans with session and application scope. Observe the difference between the two counts.

Printable Page

[This page was updated: 21-Sep-2000]

Products & APIs | Developer Connection | Docs & Training | Online Support
Community Discussion | Industry News | Solutions Marketplace | Case Studies

Glossary | Feedback | A-Z Index

Form Processing Using JSP

by **jGuru**

[[Help](#) | [API Docs](#) | [Short Course](#) | [Exercises](#)]

In this exercise, you develop a simple JSP page (form.jsp), which can process an HTML form containing typical input elements like textboxes, radio buttons, and checkboxes. You also develop a bean (FormBean.java), whose property names mirror the input elements of the form. You will then examine the automatic instantiation of the bean on a form POST operation, using the introspective features provided by the JSP engine.

Prerequisites

- [Installing and Configuring Tomcat](#)

Skeleton Code

- [FormBean.java](#)
- [form.jsp](#)

Tasks

1. You are given the JSP page containing the form. Observe that the form posts to itself recursively. Instantiate the bean FormBean when you recognize that a POST operation has taken place. Allow the setter methods to be called on the bean using introspection.

2. Deploy the JSP page within Tomcat.

3. Develop the bean, FormBean.java, with properties matching the form elements.

4. Compile the bean source FormBean.java.

5. Deploy the bean within Tomcat.

6. Run the example.

Where help exists, the task numbers above are linked to the
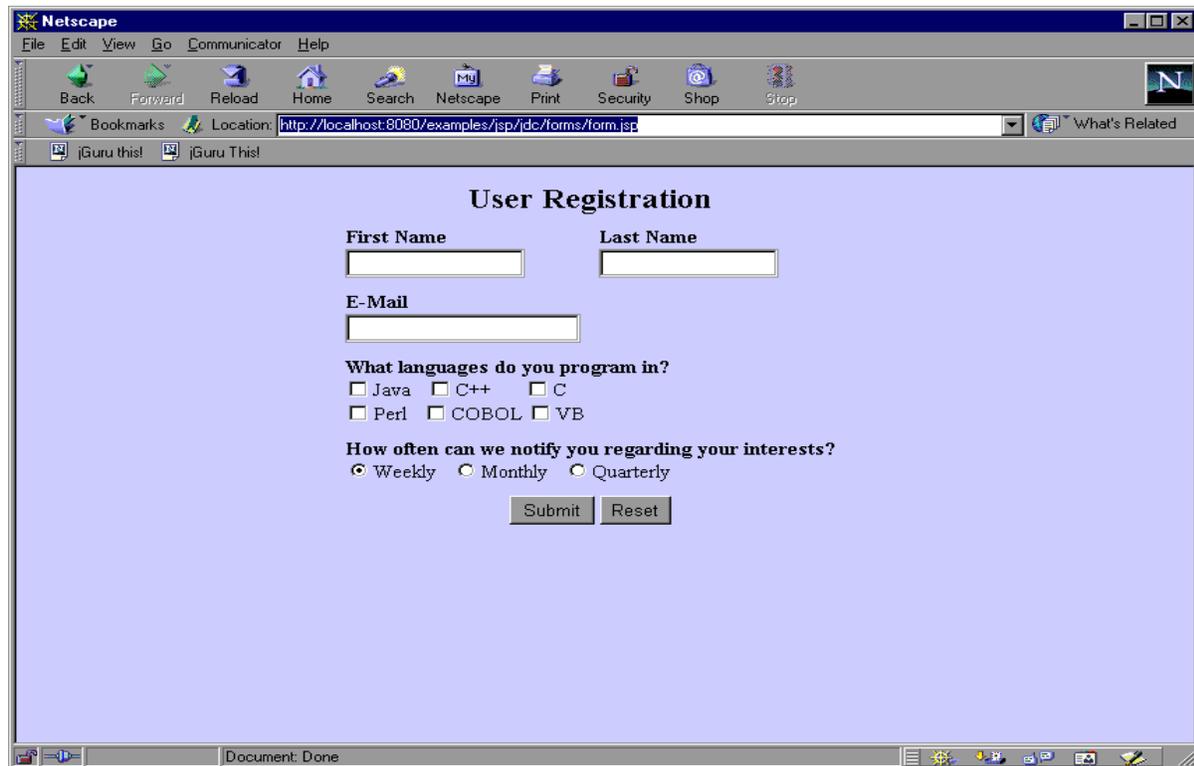
step-by-step [help](#) page.

Solution Source

- [FormBean.java](#)
- [form.jsp](#)

Demonstration

From your browser, access the URL
http://localhost:8080/examples/jsp/jdc/forms/form.jsp.

You should see an HTML form as shown below:

Fill in data for all the form input elements, before performing a submit:

On submission, you should see the data you entered extracted from the bean and displayed beneath the form:

Exercises

Short Course

Copyright 1996-2000 jGuru.com. All Rights Reserved.

Printable Page 🖨️　　　[This page was updated: 21-Sep-2000]

Products & APIs | Developer Connection | Docs & Training | Online Support
Community Discussion | Industry News | Solutions Marketplace | Case Studies

Glossary | Feedback | A-Z Index

For more information on Java technology
and other software from Sun Microsystems, call:
(800) 786-7638
Outside the U.S. and Canada, dial your country's
AT&T Direct Access Number first.

Form Processing Using JSP

by **jGuru**

[[Exercise](#) | [API Docs](#) | [Short Course](#) | [Exercises](#)]

Help is available for each task.

Task 1

You are given the JSP page containing the form. Observe that the form posts to itself recursively. Instantiate the bean FormBean when you recognize that a POST operation has taken place. Allow the setter methods to be called on the bean using introspection.

You can use the useBean tag for instantiating the bean. By indicating property="*" within the setProperty tag, you can direct the JSP engine to parse all the incoming values from the HTML form elements that are part of the request object and assign them to their corresponding bean properties.

Task 2

Deploy the JSP page within Tomcat.

Assuming you have installed Tomcat in say, \jakarta-tomcat, copy the JSP file to \jakarta-tomcat\webapps\examples\jsp\jdc\forms\form.jsp

Task 3

Develop the bean, FormBean.java with properties matching the form elements.

Develop a bean with the names of the bean properties matching the names of the form input elements. Note that values for certain form elements like checkbox, need to be mirrored within an indexed property. Define the corresponding getter/setter methods for each property within the bean.

Task 4

Compile the bean source FormBean.java.

You can compile the bean as javac FormBean.java

Task 5

Deploy the bean within Tomcat.

Copy FormBean.class to
\jakarta-tomcat\webapps\examples\WEB-INF\classes\com\jguru\FormBean.class

Task 6

Run the example.

From your browser, access the URL
http://localhost:8080/examples/jsp/jdc/forms/form.jsp.

Fill in data for the form input elements and hit submit. You should see the data you entered displayed at the bottom of the page.

Printable Page

[This page was updated: 21-Sep-2000]

```html
<html>
<body bgcolor="#c8d8f8">
<form action="/examples/jsp/jdc/forms/form.jsp" method=post>
<center>
<table cellpadding=4 cellspacing=2 border=0>

<th bgcolor="#CCCCFF" colspan=2>
<font size=5>User Registration</font>
</th>

<tr>
<td valign=top>
<b>First Name</b>
<br>
<input type="text" name="firstName" size=15></td>
<td  valign=top>
<b>Last Name</b>
<br>
<input type="text" name="lastName" size=15></td>
</tr>

<tr>
<td valign=top colspan=2>
<b>E-Mail</b>
<br>
<input type="text" name="email" size=20>
<br></td>
</tr>

<tr>
<td  valign=top colspan=2>
<b>What languages do you program in?</b>
<br>
<input type="checkbox" name="languages" value="Java">Java  
<input type="checkbox" name="languages" value="C++">C++     
<input type="checkbox" name="languages" value="C">C<br>
<input type="checkbox" name="languages" value="Perl">Perl  
<input type="checkbox" name="languages" value="COBOL">COBOL
<input type="checkbox" name="languages" value="VB">VB<br>
</td>
</tr>

<tr>
<td  valign=top colspan=2>
<b>How often can we notify you regarding your interests?</b>
<br>
<input type="radio" name="notify" value="Weekly" checked>Weekly  
<input type="radio" name="notify" value="Monthly">Monthly  
<input type="radio" name="notify" value="Quarterly">Quarterly
<br></td>
</tr>

<tr>
<td  align=center colspan=2>
<input type="submit" value="Submit"> <input type="reset"  value="Reset">
</td>
</tr>

</table>
</center>
</form>
```

```
<%-- Create the bean only when the form is posted --%>
<%
if (request.getMethod().equals("POST")) {
%>

<jsp:useBean id="formHandler" class="com.jguru.FormBean">
<%-- provide a setProperty tag and ensure that the setter methods are invoked via
introspection --%>
</jsp:useBean>
<p>
<hr>
<font color=red>
<b>You submitted:<P>
First Name:</b><br>
<%-- invoke the getter method to display the firstName using the getProperty tag --%>
<br><b>Last Name:</b><br>
<%-- invoke the getter method to display the lastName using the getProperty tag --%>
<br><b>Email:</b><br>
<%-- invoke the getter method to display the email address using the getProperty tag --%>
<b>Languages:</b><br>
<%
        String[] lang = formHandler.getLanguages();
        if (!lang[0].equals("1")) {
                out.println("<ul>");
                for (int i=0; i<lang.length; i++)
                        out.println("<li>"+lang[i]);
                out.println("</ul>");
        } else out.println("Nothing was selected<br>");
%>
<b>Notification:</b><br>
<%-- invoke the getter method to display the ntotification status using the getProperty
tag --%>
<br>
<%
}
%>
</font>
</body>
</html>
```

```
package com.jguru;

public class FormBean {
// declare properties for firstname, lastname, notify and email with the names
//matching the corrosponding form input elements
        private String[] languages;

        public FormBean() {
         firstName="";
         lastName="";
         email="";
         languages = new String[] { "1" };
         notify="";
        }

        //write getter methods for firstname, lastname, notify, email and languages

        //write setter methods for firstname, lastname, notify, email and languages

}
```

```
<%@ page import="com.jguru.CounterBean" %>

<%-- provide appropriate values for the class and scope attributes --%>

<jsp:useBean id="session_counter" class="" scope="" />
<jsp:useBean id="app_counter" class="" scope="" />

<% session_counter.increaseCount();
 synchronized(page) {
    app_counter.increaseCount();
 }
%>
<h3>
Number of accesses within this session:

<%-- provide appropriate values for the name attribute --%>

<jsp:getProperty name="" property="count" />
</h3>
<p>
<h3>
Total number of accesses:

<%-- provide appropriate values for the name attribute --%>

<% synchronized(page) { %>
<jsp:getProperty name="" property="count" />
<% } %>
</h3>
```

```java
package com.jguru;
public class CounterBean {
        //declare a integer for the counter

        public int getCount() {
                //return count
        }

        public void increaseCount() {
                //increment count;
        }
}
```

```
<%-- Indicate the location of the error handler using the page tag --%>
<html>
<body>
<form method=post action="errhandler.jsp">
What's the coolest programming language in the known universe?<p>
Java<input type=radio name=language value="JAVA" checked>
C++<input type=radio name=language value="CPP">
Visual Basic<input type=radio name=language value="VB">
<p>
<input type=submit>
</form>

<%
        if (request.getMethod().equals("POST")) {
          if (request.getParameter("language").equals("JAVA")) {
                out.println("<hr><font color=red>You got that right!</font>");
          } else {
                        //thow a new exception initializing it with some message
          }
        }
%>

</body>
</html>
```

```
<%-- Indicate that this is an error page using the page tag --%>
<html>
<body>
<h1>
Error Page
</h1>
<hr>
<h2>
Received the exception:<br>
<font color=red>
<%= exception.toString() %>
</font>
</h2>
</body>
</html>
```

```
<%@ page import="com.jguru.CounterBean" %>
<jsp:useBean id="session_counter" class="com.jguru.CounterBean" scope="session" />
<jsp:useBean id="app_counter" class="com.jguru.CounterBean" scope="application" />

<% session_counter.increaseCount();
  synchronized(page) {
    app_counter.increaseCount();
 }
%>
<h3>
Number of accesses within this session:
<jsp:getProperty name="session_counter" property="count" />
</h3>
<p>
<h3>
Total number of accesses:
<% synchronized(page) { %>
<jsp:getProperty name="app_counter" property="count" />
<% } %>
</h3>
```

```
package com.jguru;
public class CounterBean {
        int count;

        public int getCount() {
                return count;
        }

        public void increaseCount() {
                count++;
        }
}
```

```
<%@ page errorPage="errorpage.jsp" %>
<html>
<body>
<form method=post action="errhandler.jsp">
What's the coolest programming language in the known universe?<p>
Java<input type=radio name=language value="JAVA" checked>
C++<input type=radio name=language value="CPP">
Visual Basic<input type=radio name=language value="VB">
<p>
<input type=submit>
</form>

<%
        if (request.getMethod().equals("POST")) {
          if (request.getParameter("language").equals("JAVA")) {
                out.println("<hr><font color=red>You got that right!</font>");
            } else {
                throw new Exception("You chose the wrong language!");
            }
        }
%>

</body>
</html>
```

```
<%@ page isErrorPage="true" %>
<html>
<body>
<h1>
Error Page
</h1>
<hr>
<h2>
Received the exception:<br>
<font color=red>
<%= exception.toString() %>
</font>
</h2>
</body>
</html>
```