# Pro JSP 2

## Fourth Edition

Simon Brown, Sam Dalton, Daniel Jepp,
David Johnson, Sing Li, and Matt Raible
Edited by Kevin Mukhar

Apress®

**Pro JSP 2, Fourth Edition**



ISBN (pbk): 1-59059-513-0

Printed and bound in the United States of America 9 8 7 6 5 4 3 2 1



Lead Editors: Steve Anglin and Kevin Mukhar
Technical Reviewers: Scott Davis and Dilip Thomas
Editorial Board: Steve Anglin, Dan Appleman, Ewan Buckingham, Gary Cornell, Tony Davis, Jason Gilmore, Jonathan Hassell, Chris Mills, Dominic Shakeshaft, Jim Sumser
Associate Publisher: Grace Wong
Project Manager: Beckie Brand
Copy Edit Manager: Nicole LeClerc
Copy Editor: Sharon Wilkey
Assistant Production Director: Kari Brooks-Copony
Production Editor: Ellie Fountain
Compositor: ContentWorks
Proofreader: Sue Boshers
Indexer: Julie Grady
Artist: Kinetic Publishing Services, LLC
Interior Designer: Van Winkle Design Group
Cover Designer: Kurt Krames
Manufacturing Director: Tom Debolski

Distributed to the book trade worldwide by Springer-Verlag New York, Inc., 233 Spring Street, 6th Floor, New York, NY 10013. Phone 1-800-SPRINGER, fax 201-348-4505, e-mail `orders-ny@springer-sbm.com`, or visit `http://www.springeronline.com`.

For information on translations, please contact Apress directly at 2560 Ninth Street, Suite 219, Berkeley, CA 94710. Phone 510-549-5930, fax 510-549-5939, e-mail `info@apress.com`, or visit `http://www.apress.com`.



The source code for this book is available to readers at `http://www.apress.com` in the Source Code section.

# Contents at a Glance

# Contents

# About the Authors

**SIMON BROWN** works in London as a technical architect and has been using Java since its early beginnings, working in roles ranging from developer and architect to mentor and trainer. When not working with Java, he can usually be found speaking or writing about it. In the past few years, Simon has spoken at the JavaOne Conference and has authored or coauthored several books, including his own, entitled *Professional JSP Tag Libraries* (Peer Information, 2002). Simon maintains an active involvement within the Java community as a bartender (moderator) with JavaRanch and his open-source JSP custom tag-testing framework called TagUnit.

Simon graduated from the University of Reading in 1996 with a First Class BSc (Hons) degree in Computer Science and is a Sun Certified Enterprise Architect for J2EE, Web Component Developer for J2EE, and Developer for the Java 2 Platform.

For information about what Simon is currently up to, you can point your browser to his web log at `http://www.simongbrown.com/blog/`.

*I would like to thank my wife, Kirstie—you're always there for me.*

**SAM DALTON** has worked with Java and related technologies in London for a number of years, and has coauthored two titles, *Professional Java Servlets 2.3* (Peer Information, 2002) and *Professional SCWCD Certification* (Wrox Press, 2002). Sam is an active contributor to TagUnit, an open-source custom tag-testing framework (`http://www.tagunit.org`) and is also pursuing other open-source interests. He has embarked on the next stage of his career adventure by joining ThoughtWorks (`http://www.thoughtworks.co.uk`).

Sam graduated from the University of Reading in 1997 with a 2:1 honors degree in Computer Science. He has also achieved various certifications, including Sun Certified Web Component Developer and Sun Certified Developer. Please feel free to e-mail any questions or comments about this book and related topics to `books@samjdalton.com`.

*Well, here we are again! Who would have thought I would ever be involved in three books? Not me, that's for sure! There are a number of people that I would like to thank for supporting/putting up with me while I was contributing to this book. First of all, as ever, I would like to thank my darling wife, Anne, without whom I would not have the energy to do half of the things that I do. I would also like to thank my Mum and Dad; it means a great deal to me to see how proud my work makes you—thanks! Enjoy the book, and happy reading!*

**DANIEL JEPP** is a senior developer at Dresdner Kleinwort Wasserstein, based in London. He has been working with the Java platform and related technologies for a number of years and has spoken at several sessions at the JavaOne Conference. Dan coauthored *Professional SCWCD Certification* with Sam Dalton in 2002.

Dan graduated from the University of Kent, in Canterbury, England, where he attained a 2:1 honors degree in Computer Science, and has since gained the following Sun Certifications: Sun Certified Programmer, Developer, and Web Component Developer for the Java 2 Platform.

*Dedicated to my fiancée, Kelly, whose love, support, and encouragement will leave me forever grateful.*

**DAVID JOHNSON** is an experienced software developer who has worked in the commercial software development, telecommunications, and Geographic Information Systems industries. David has been working with Java since before the dawn of Java 1.0. Since then, he has been involved in the development of a number of Java-based commercial products, including the HAHTsite Application Server, HAHT eSyndication, Venetica's Jasper document viewer, and Rogue Wave Software's Object Factory IDE. David is also an active weblogger and the original developer of the open-source Roller Weblogger (`http://www.rollerweblogger.org`) software. David works at HAHT Commerce and lives in Raleigh, North Carolina, with his wife and three children.

*First and foremost, I must thank my beautiful wife, Andi, for giving me the encouragement and time needed to work on this book. She kept my three little boys, Alex, Linus, and Leo, happy and quiet while I toiled away in the back room on my chapters. I should also thank fellow Roller Weblogger developers Lance Lavandowska and Matt Raible. Lance helped me to get started with this project, and Matt helped to improve and perfect my example code. Finally, I would like to thank Bill Barnett and the whole HAHTsite Application Server team at HAHT Commerce for teaching me just about everything I know about web-application performance and scalability and for inspiring me to learn more.*

**SING LI** was first bitten by the computer bug in 1978 and has grown up with the microprocessor revolution. His first PC was a $99 do-it-yourself COSMAC ELF computer with 256 bytes of memory and a 1-bit LED display. For more than two decades, Sing has been a developer, author, consultant, speaker, instructor, and entrepreneur. His wide-ranging experience spans distributed architectures, web-application and service systems, computer telephony integration, and embedded systems. Sing is a regular book contributor, has been working with and writing about Java, Jini, and JXTA since their very first alpha releases, and is an evangelist of P2P technology and a participant in the JXTA community.

**MATT RAIBLE** is a Montana native who grew up in a log cabin without electricity or running water. After hiking to school a mile and a half every day (and skiing in the winter), he would arrive home to a very loving family. "The Cabin" is a beautiful and awesome place that will always be near and dear to Matt's entire family. Even without electricity, his father, Joseph, connected them to the Internet by using a 300 baud modem, a Commodore 64, and a small generator. CompuServe was the name, slow was the game. Matt became inspired by the Internet in the early 1990s, and has been developing websites and web applications ever since. He graduated from the University of Denver in 1997 with degrees in Russian, International Business, and Finance. To learn more about Matt and his life as a J2EE Developer, visit him at `http://raibledesigns.com`.

*I'd like to thank my beautiful wife, Julie, and adorable daughter, Abbie, for their love and support while writing these chapters. Abbie was born three weeks before I was asked to write my chapters, and her smiles and giggles were an incredible inspiration. Chris Alonso, thanks for motivating me to go into computers as a profession and for being such a good friend. Thanks to my dad for passing along his knack for computers and great memory, and to my Mom for giving me a passion for life, happiness, and humor. Kalin—you're the best sister in the world and you make this world a better place with your smiles and laughter. Last but not least, thanks to Matt Good for letting me write Java, and to Martin Gee and Brian Boelsterli for their mentoring.*

# About the Editor

**KEVIN MUKHAR** is a software developer from Colorado Springs, Colorado. For the past seven years, he has worked on various software systems using different Java Enterprise technologies. He has coauthored several other books, including *Beginning Java EE 5: From Novice to Professional* (Apress, 2005), *The Ultimate Palm Robot* (Osborne/McGraw-Hill, 2003), and *Beginning Java Databases* (Wrox Press, 2001). In addition to developing software during the day, he is working on a master's degree in computer science. His web page is `http://home.earthlink.net/~kmukhar/`.

# About the Technical Reviewers

**SCOTT DAVIS** is a senior software engineer at OpenLogic. He is passionate about open-source solutions and agile development. In addition to serving as technical editor for several Apress titles including *Pro Jakarta Tomcat 5* and *Beginning JSP 2: From Novice to Professional*, he coauthored *JBoss At Work* (O'Reilly Media, 2005). He is the author of the upcoming book *Pragmatic GIS* (Pragmatic Bookshelf, 2006), which focuses on free/open-source geography software solutions such as Google Maps and GeoServer.

Scott is a frequent presenter at national conferences (such as No Fluff, Just Stuff) and local user groups. He was the president of the Denver Java Users Group in 2003 when it was voted one of the top-ten JUGs in North America. Since a quick move north, he has been active in the leadership of the Boulder Java Users Group. Keep up with him at `http://www.davisworld.org`.

**DILIP THOMAS** is an open-source enthusiast who keeps a close watch on LAMP technologies, open standards, and the full range of Apache Jakarta projects. He is coauthor of *PHP MySQL Website Programming: Problem - Design - Solution* (Apress, 2003) and a technical reviewer/editor on several open-source/open standard book projects. Dilip is an editorial director at Software & Support Verlag GmbH.

Dilip resides in Bangalore, India, with his beautiful wife, Indu, and several hundred books and journals. Reach him via e-mail at `dilip.thomas@gmail.com`.

# Acknowledgments

When we started this revision, it was going to be just a simple update: change a few things here and there to make sure the book was consistent with JSP 2.1. Along the way, it turned into a much bigger task. Part of the reason for that was our desire to make this the best possible book about JSP 2.1 that we could. So I want to acknowledge everyone at Apress who helped make this the book it is, especially Sharon Wilkey, Beckie Brand, Ellie Fountain, Ami Knox, Steve Anglin, and the technical reviewers Scott Davis and Dilip Thomas.

While I worked on this book, my wife and I experienced a lot of changes and challenges in our lives. I'd like to thank some of the many people who helped us through that time: Tom and Marg Gimmy, Dave and Kris Johnson, my family, Anne's family, and Dawn Girard. And of course, no thanks would be complete without thanking my family, Anne and Christine, for letting me spend the time away from them needed to do this project.

Kevin Mukhar

# Introduction

Welcome to the fourth edition of *Professional JSP*, designed to help new and experienced Java developers alike discover the power (and even the joy) of creating Java-based server-side solutions for the Web by using **JavaServer Pages**, or **JSP** for short. If you've programmed with JSP before, you'll find that the new features in JSP 2.1 make developing JSP pages easier than ever before. If you only know a little Java, this is your chance to add JSP to your toolbox skills.

**JSP** is a server-side technology that takes the Java language, with its inherent simplicity and elegance, and uses it to create highly interactive and flexible web applications. In today's unsure economic climate, having the Java language as the cornerstone of JSP makes JSP particularly compelling for business: Because Java is an open language (meaning it doesn't require expensive licenses), JSP solutions can be highly cost-effective.

The founding premise of JSP is that HTML can be used to create the basic structure of a web page, and Java code can be mixed in with the HTML to provide the dynamic components of the page that modern web users expect. If you understand the concepts of HTML and web pages, JSP provides an unbeatable way to learn about creating innovative, interactive content as well as coming to grips with the popular language of Java. This book will be your guide as you step into this exciting new world.

## Who Is This Book For?

This book is aimed at anyone who knows the Java language and core APIs and wants to learn about web programming with the latest versions of the JSP and Servlet APIs.

Familiarity with HTML is required; however, no prior knowledge of server-side Java programming is necessary. Having said that, this book does not claim to be exhaustive in all areas, particularly in relation to other Java Enterprise Edition APIs such as Enterprise JavaBeans.

## What's Covered in This Book

This book covers the latest versions of the JSP and Servlet specifications—versions 2.1 and 2.5 respectively, both of which are new specifications developed through the **Java Community Process** (`http://www.jcp.org/`).

---

**Note** At the time this book was being published, the JSP specification was in Proposed Final Draft stage. It's possible that some small changes might be made before the specification reaches final release; however, any modifications are likely to be minor and the new specifications are already being implemented by a number of products such as Tomcat 5.5.

---

Those who have read previous editions of this book will notice that this edition is a revision of *Professional JSP, 3rd Edition*. Because the third edition covered JSP 2.0, most of the information was still current for this book, which covers JSP 2.1. However, we've gone through the entire book and ensured that the material is still correct for JSP 2.1. We've gone through every chapter and updated the text to make it clearer and more concise. Finally, we've added an entire new chapter on JavaServer Faces, one of the newest Java web-application technologies.

If you already have some exposure to server Java web development, you should pay attention to any changes in the technologies that are indicated throughout the book, or skip ahead to the sections that interest you the most. On the other hand, if you're new to JSP, servlets, and JSTL, and this is somewhat confusing, you've come to the right place; the early chapters in this book, especially, were written with you in mind.

Here is what you can expect to find in each chapter:

Chapter 1, *The Anatomy of a JavaServer Page*, looks at the JSP life cycle, JSP application architecture, and the fundamentals of JSP pages, and provides a feel for where JSP technology fits within the Java EE 5 and other web components such as servlets, tag libraries, and JavaBeans, which exist in the Java EE 5 web tier for providing dynamic web-based content.

Chapter 2, *Servlets and Deployment*, delves into what Java servlets are, and looks at the development and deployment of Java servlets. The Servlet and JSP specifications are developed in parallel, and this chapter is up to date for the latest release of JSP 2.1 and Servlets 2.5 (as is the rest of the book).

We discuss one of the new features of the JSP 2.1 specification in Chapter 3, *The JavaServer Pages Expression Language*. The JSP expression language is what you'll be using most often in JSP pages, an intentionally simple language that is, to a large extent, independent of JSP.

Chapter 4, *JavaServer Pages Standard Tag Library*, looks at the reasons for the creation of the JSTL, its details (it is in fact four different tag libraries), and how to install the JSTL.

Chapter 5, *JavaServer Faces*, is an introduction to JavaServer Faces (JSF), a framework for creating component-based user interfaces. You'll learn how to use JSF with JSP pages to create feature-rich user interfaces.

*Tag Files and Simple Tags* is the title of Chapter 6. Tags contained within JSTL are extremely valuable for improving the readability and maintainability of a JSP page. You can also build custom tags to enable your own functionality to be reusable and easily maintained. Tag files and simple tags are both new mechanisms for writing custom tags introduced as a part of the JSP 2.1 specification.

Chapter 7, *Classic Tags*, takes a look at the facilities provided by former versions of the JSP specification for writing custom tags. As you'll see throughout the chapter, these previous methods, now called classic tags, provide a great deal more flexibility and therefore are still useful in some scenarios.

After you learn the basics of building custom tags, Chapter 8, *Custom Tag Advanced Features and Best Practices*, wraps up your understanding by looking at some more advanced features and the best way to use custom tags.

Chapter 9, *Data Access Options for Web Applications*, discusses how best to access your back-end data from your JSPs and servlets. No matter what type of JSP application you're writing, you'll need to either store the data that is created by your application, or use data from an external source, and this chapter looks at examples using a MySQL database.

In Chapter 10, *Introduction to Filtering*, you'll look at filtering, a standard feature of all Servlet 2.5–compliant containers. You'll explore the life cycle of a filter as managed by the container, discuss the very important concept of filter chaining, and then create and deploy two simple filters.

Chapter 11, *Advanced Filtering Techniques*, is a cookbook for the application of filters, as you turn your attention to the more advanced techniques involved in applied filter programming by looking at five examples that can be used as the basis for your own filter implementation.

Chapter 12, *Security in Web Applications*, looks at making your web applications secure and explores different methods of authentication and authorization.

Chapter 13, *Improving Web-Application Performance and Scalability*, is your guide to a number of well-known tools and techniques such as page caching and database connection pooling that you can use to improve performance and stability, even after you've designed and coded your application.

Chapter 14, *Web-Application Design and Best Practices*, brings together the techniques covered in the earlier chapters and shows how to build maintainable, extensible Java-based web applications. It looks at the importance of good design and how it can help you build high-quality web applications that are easier to maintain and extend in the future.

In Chapter 15, *Using Struts, XDoclet, and Other Tools*, you'll develop a résumé building and viewing web application called `struts-resume`, by using a variety of third-party products. All of the products used in `struts-resume` are open source and help to facilitate and speed up various stages of the development process.

# What You Need to Use This Book

The first thing you'll need to use this book is a computer that supports the Java programming language. This includes computers that run Microsoft Windows, Sun Solaris, or Linux.

We don't use any proprietary software, and all code runs on open-source products, available free of charge over the Internet. Consequently, an Internet connection is pretty much essential in order to get hold of this software.

The primary piece of software you'll need is a web container that supports the JSP 2.1 and Servlet 2.5 specifications. Although there are a number of options to choose from, we've elected to use the Jakarta Tomcat web container throughout the whole book because it's the officially designated reference implementation. Version 5.5 is the latest and greatest, which supports the specs we require. You can get the latest release information about Tomcat 5.5 from `http://jakarta.apache.org/tomcat/index.html`.

As you need further software components during the course of the book, we'll indicate clearly where to download them from.

# Conventions

We've used several styles of text and layout in this book to differentiate between different kinds of information. Here are examples of the styles used and an explanation of what they mean.

Code has several fonts. If we're talking about code in the text, we use a nonproportional font like this: `for...next`. If it's a complete code listing that can be entered and used as part of an application, then it will appear in a nonproportional font with a caption like this:

**Listing 1-2.** *date.jsp*

```
<html>
  <body>
    <h2>Greetings!</h2>
    <p>The current time is <%=new java.util.Date()%> precisely</p>
  </body>
</html>
```

Code that is an extract or snippet from a larger listing will appear without a caption, like this:

```
import javax.servlet.http.*;

public class SessionTracker2 extends HttpServlet {
  public void doGet(HttpServletRequest req, HttpServletResponse res)
```

Sometimes you will need to type in commands on the command line, which we display using the following style:

```
> set classpath=.;%Java EE_HOME%\lib\j2ee.jar
\projsp> javac -d . client\*.java
```

We show the prompt using a > symbol or *dirname*> (where *dirname* is a directory name) and then the commands you need to type. As you can see, we tend to use the Windows directory separator character when showing directory paths. We do this because we expect that a lot of readers will be using a Windows platform when they try out the code. But we also develop on Linux or Solaris platforms, and if you do too, then you should use the directory separator that is correct for your platform.

---

**Note** Advice, hints, and background information come in this type of font offset by borders. Important pieces of information also come in this format. Depending on the type of information, we preface the text with the word Note, Tip, or Caution. Notes consist of incidental information of one type or another that defines, explains, or elaborates on the main discussion. Tips will make your programming easier. For instance, a Tip might point out another way to use a certain feature that's not obvious from the main discussion. Cautions indicate a potential hazard. For example, a Caution might be a method that if misused could crash your application server.

---

Bullets appear indented, with each new bullet marked as follows:

- **Important Words** are in a bold type font.
- Words that appear on the screen or in menus, such as `File` or `Window`, are in a monospaced font.

Numbered lists are similar to bulleted lists:

1. Do this action first.
2. Do this action next.

# What to Do If You Encounter Problems

Despite all our best efforts, and despite this book's numerous sharp-eyed editors, there is a possibility that errors managed to sneak through. It has been known to happen. If you are having problems with any of the text or code examples, the first place to go for corrections is the web page for the book (`http://www.apress.com/book/bookDisplay.html?bID=464`). If any errata have been identified, you will find a link for Corrections on the book's web page. If you click this link, you will find a page that lists known errors with the code or book text, and corrections for those problems.

If you can't find your problem listed on the Corrections page, you will find a link to Submit Errata on the main book page. If you've double-checked and triple-checked your problem and still can't get the code to work or the text to make sense, use the Submit Errata link to send us a description of the problem. We can't promise a speedy response, but we do see all submissions and post responses to the Corrections page after we've had a chance to check out the problem.

CHAPTER 1



# The Anatomy of a JavaServer Page

The Java Platform, Enterprise Edition 5 (Java EE 5) has two different but complementary technologies for producing dynamic web content in the presentation tier—namely **Java Servlet** and **JavaServer Pages (JSP)**.

Java Servlet, the first of these technologies to appear, was initially described as extensions to a web server for producing dynamic web content. JSP, on the other hand, is a newer technology but is equally capable of generating the same dynamic content. However, the way in which a servlet and a JSP page produce their content is fundamentally different; servlets embed content into logic, whereas JSP pages embed logic into content.

JSP pages contain markup interlaced with special JSP elements that provide logic for controlling the dynamic content. Servlets are built using Java classes that contain statements to output markup code. Of these two paradigms, JSP pages are preferred for presenting dynamic content in the presentation tier due to their greater readability, maintainability, and simplicity. Further increasing the simplicity and ease of use of JSP pages was one of the main objectives of the JSP 2.0 specification, which included several new features to make it easier than ever to embrace JSP technology, especially for developers who aren't fluent in the Java syntax.

The inclusion of a new **expression language (EL)** enables JavaScript-style JSP code to be embedded within pages, which makes it much easier for web developers not familiar with the Java syntax to understand the JSP logic. A library of standard actions known as the **JavaServer Pages Standard Tag Library (JSTL)** is also included to provide a host of useful, reusable actions such as conditional statements, iteration, and XML integration to name a few. These actions are applicable in some shape or form to most JSP web applications, and their use will greatly improve the reliability and ease of development for JSP page authors. Custom actions (also known as custom tags) also benefit from changes in the JSP specification, and it's now possible to write a custom action entirely in JSP syntax instead of Java syntax!

JSP 2.1 further eases the development of JSP pages by unifying the JSP expression language with the JavaServer Faces (JSF) expression language. These new features will help make JSP pages easier to write and maintain and are discussed in detail in the following chapters:

- The JSP 2.1 expression language (EL) (see Chapter 3)
- The JavaServer Pages Standard Tag Library (JSTL) (see Chapter 4)
- The JavaServer Faces custom tags (see Chapter 5)
- JSP custom tags (see Chapters 6, 7, and 8)

In this chapter, you'll take a look at some of the fundamental concepts based around JSP technology, such as the following:

- The mechanics of a JSP page
- Typical JSP architectures
- Core JSP syntax
- Tag libraries

The aim of this chapter is to help you gain a grounding in the basics of JSP technology so you can make full use of the rest of the chapters in this book that build on these basic principles.

# Before You Begin

To begin examining the basics of JSP technology, it's essential that you have a cursory familiarity with the alternative and complementary presentation-tier web component, Java servlets. The next chapter will discuss servlets in more detail.

## Java Servlets

As mentioned earlier, servlets can most simply be described as custom web-server extensions, whose jobs are to process requests and dynamically construct appropriate responses. In practice, such responses are usually returned in the form of HTML or XML and are the result of a user making an HTTP request via a web browser. Servlet technology has been an extremely popular choice for building dynamic web applications such as e-commerce sites, online banking, and news portals, for reasons of simplicity, extensibility, efficiency, and performance over alternative technologies such as Common Gateway Interface (CGI) scripts.

Some of the most basic advantages of servlet technology are as follows:

- **Simplicity:** Servlets are easy to write, and all the complicated threading and request delegating is managed by the servlet container.
- **Extensibility:** The Servlet API is completely protocol independent.
- **Efficiency:** Unlike CGI scripts, the execution of a servlet doesn't require a separate process to be spawned by the web server each time.
- **Performance:** Servlets are persistent, and their life cycle extends beyond that of each HTTP request.

Servlets are simply Java classes that inherit from the `javax.servlet.Servlet` interface. These servlets are compiled and deployed inside a servlet container, which is a Java environment that manages the life cycle of the servlet and deals with the lower-level socket-based communication. The servlet container may be part of an existing Java-enabled web server itself or may be used as a stand-alone product that is integrated with a third-party web server. The servlet Reference Implementation container, Apache Jakarta Tomcat for example, may be used as a stand-alone web server or as a separate servlet container inside a larger commercial web server such as the Apache web server.

Servlets are typically used for returning text-based content such as HTML, XML, WML, and so on. However, they are equally at home returning binary data such as images or serialized Java objects, which are often used by further servlets to generate some appropriate dynamic response.

# JSP Under the Hood

A JSP page is simply a regular text file that contains markup (usually HTML) suitable for display inside a browser. Within this markup are special JSP elements that you'll learn more about later. These are used to provide processing logic that enables dynamic content to be produced on a request-by-request basis.

In JSP terms, any markup that isn't a JSP element is known as template text, and this really can be any form of text-based content such as HTML, WML, XML, or even plain text! Of course the mixture of JSP elements and template text cannot simply be sent to the browser without any form of processing by the server. We mentioned earlier how JSP technology is an extension of servlet technology, and so you probably won't be surprised to learn that each JSP page is, in fact, converted into a servlet in order to provide this processing logic. Figure 1-1 shows a JSP page being translated and compiled into a servlet in response to a request. This servlet is known as the JSP **implementation servlet**.

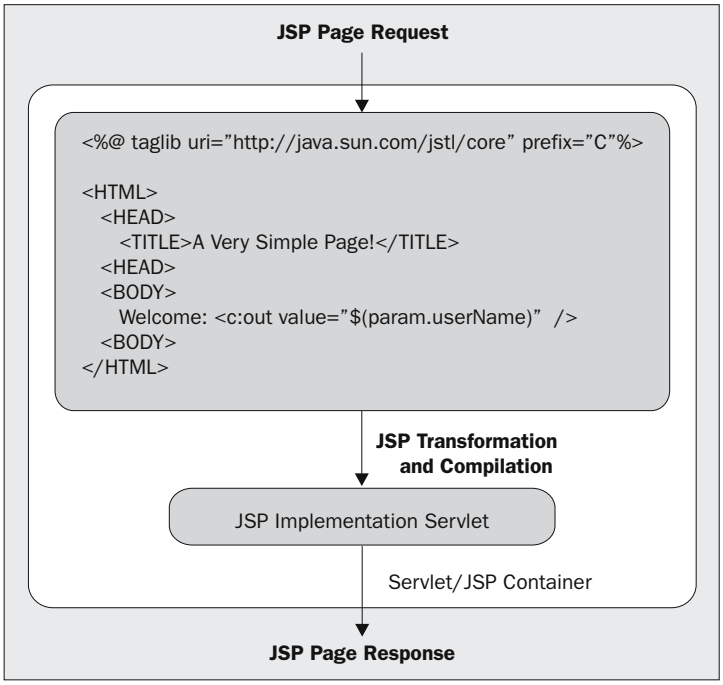


**Figure 1-1.** *The JSP container translates and compiles the JSP source into an implementation class, which is used to process all requests.*

A request for a JSP page is handled initially by the web server, which then delegates the request to the JSP container. The JSP engine will translate the contents of the JSP into its implementation servlet, which the container then uses to service the request. Usually a JSP container will check whether the contents of a JSP page have changed before deciding if it needs to retranslate the page in response to a request. This feature can make on-the-spot changes to JSP pages easy because the next request will automatically cause a retranslation and the most up-to-date content will be returned. Compare this with a purely servlet-based approach, which would require the servlet container to be shut down in order to have the necessary changes made, such as recompilation, testing, and finally, a restart!

Let's take a closer look at the process of taking a plain JSP text file and turning it into a dynamic web component; this process is also known as the JSP life cycle.

## The JSP Life Cycle

As you've just seen, JSP pages don't directly return content to the client browser themselves. Instead, they rely on some initial server-side processing that converts the JSP page into the JSP page implementation class (see Figure 1-2), which handles all requests made of the JSP.

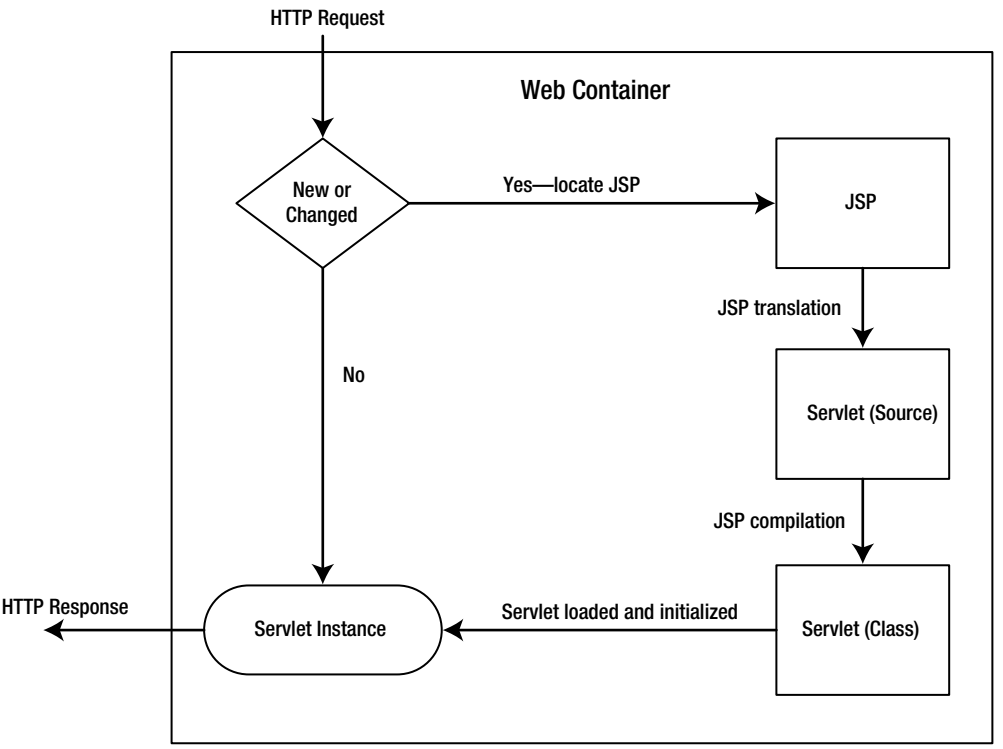


**Figure 1-2.** *Before processing a request, the container determines whether the JSP source is new or has changed. If so, the container translates and compiles the JSP page into a servlet class, or page implementation class, before passing the request to the servlet for processing.*

As you can see in Figure 1-2, the JSP servlet container decides whether the JSP page has been translated before. If not, the JSP container starts the translation phase to generate the JSP page implementation servlet, which is then compiled, loaded and initialized, and used to service the request. If the JSP container detects that a JSP page has already been translated and hasn't subsequently changed, the request is simply serviced by the implementation servlet that already exists inside the container.

The life cycle of a JSP page can be split into four phases: translation, initialization, execution, and finalization.

### Translation

The first stage in the life cycle of a JSP page is known as the translation phase.

When a request is first made for a JSP page (assuming it hasn't been precompiled), the JSP engine will examine the JSP file to check that it's correctly formed and that the JSP syntax is correct. If the syntax check is successful, the JSP engine will translate the JSP page into its page implementation class, which takes the form of a standard Java servlet. After the page's implementation servlet has been created, it will be compiled into a class file by the JSP engine and will be ready for use.

Each time a container receives a request, it first checks whether the JSP file has changed since it was last translated. If it has, it's retranslated so that the response is always generated by the most up-to-date implementation of the JSP file.

### Initialization

After the translation phase has been completed, the JSP engine will need to load the generated class file and create an instance of the servlet in order to continue processing the initial request. Therefore, the JSP engine works very closely with the servlet container and the JSP page implementation servlet and will typically load a single instance of the servlet into memory. This single instance will be used to service all requests for the JSP page. In a real-world web application, those requests will most likely happen concurrently, so your JSP page must be multithreaded.

Prior to the Servlet 2.5 specification, the Java Servlet specification provided two separate threading models that could be used for a servlet. The models determine whether single or multiple instances of a servlet can exist. The default threading model for any servlet is the multithreaded one that requires no additional work for the developer. In this model, the container creates only a single instance of the servlet class and sends multiple requests to the instance concurrently.

To select the single-threaded model for your JSP, you must set an attribute of the `page` directive called `isThreadSafe` to `false` to serialize all requests to the implementation servlet behind the JSP:

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

In the past, containers would support this feature by creating an implementation page that implements the `SingleThreadModel` interface. When the implementation page implements this interface, the JSP container creates multiple instances of the implementation class; each instance handles a single request at any given time. However, note that the JSP 2.1 specification advises developers against using `isThreadSafe="false"` because the Servlet 2.5 specification has deprecated `SingleThreadModel`.