Clint Scott

CSD 430 – Server Side Development

Module 10.2 Assignment – The Dangers of Change Approval Processes

12/14/2025

Creating Custom Tags in JSP: Abstraction and Reusability

Introduction

Custom tags, also known as JSP tag extensions, are powerful, user-defined elements that extend the core capabilities of the JavaServer Pages (JSP) technology. Their primary function is to abstract complex Java functionality and application logic away from the presentation layer, allowing developers to encapsulate functionality into simple, reusable, XML-like tags (Smith, 2021). The current state of the CSD 430 Movie application demonstrates a common reliance on JSP scriptlets (<% ... %> and <%= ... %>) for tasks such as database access and iterative rendering, as exemplified in files like module9_delete_select.jsp. This approach, while functional, compromises the clarity and long-term maintainability of the web pages. Custom tags provide a structured solution by promoting the crucial software development principle of Separation of Concerns, ensuring that the Java logic remains in dedicated classes. At the same time, the JSP handles presentation markup (Jones & Brown, 2023).

Advantages of Custom Tags

The integration of custom tags offers significant benefits in terms of application quality and developer productivity:

- **Enhanced Code Readability and Maintenance:** Custom tags replace large blocks of embedded Java code with concise, descriptive XML syntax, improving

code readability and maintenance. The conditional logic and iteration required to display records in module9_delete_select.jsp can be substituted with a single, self-documenting tag, such as <csd:movieList list="${activeMovies}"/> (Smith, 2021). This transformation makes the JSP page appear less like a mix of Java and HTML and more like clean, structured markup, simplifying debugging and updates.

- **Increased Reusability:** Once a custom tag is defined, it can be instantly reused across any JSP within the application or imported into other web applications (Jones & Brown, 2023). The logic to generate a section of the data table, currently embedded repetitively within the JSP pages, could be centralized in one Tag Handler class. This adherence to the Do not Repeat Yourself (DRY) principle reduces code duplication and ensures consistency across the user interface.
- **Decoupling of Logic and Presentation:** Custom tags enforce a strict boundary, ensuring that web designers and front-end developers can focus solely on the JSP markup (HTML/CSS) without needing knowledge of the underlying Java business logic (Jones & Brown, 2023). The complexity of fetching and iterating over a List<Movie> from the MovieDAO is entirely hidden within the tag's implementation, making the view layer safer and easier to manage.

Disadvantages of Custom Tags

Despite their apparent advantages, custom tags do present a few development hurdles:

- **Initial Development Overhead:** Creating a custom tag requires more formal setup than simply dropping in a few lines of scriptlet code. The developer must create and compile a dedicated Tag Handler class, and then define its mapping,

attributes, and body-content within a formal Tag Library Descriptor (TLD) file (Jones & Brown, 2023). This multi-file requirement can be unnecessary overhead for straightforward, one-off tasks.

- Increased Debugging Abstraction: While they clean up the JSP, errors arising within the Tag Handler class are more complicated to pinpoint immediately because the logic is executed on the server side in a separate compiled Java class. Debugging often requires the developer to step through the JSP container's tag lifecycle methods (like doTag()) rather than examining a line of code directly on the JSP page (Smith, 2021).

Requirements for Correct Development

To correctly develop and implement a custom tag, three primary components are required:

1. **The Tag Handler Class:** This is a standard Java class (e.g., MovieListTag) that contains the encapsulated logic. For modern JSP development, this class must implement the SimpleTag interface or extend the SimpleTagSupport class, which is located in the javax.servlet package.jsp package.tagext package (Jones & Brown, 2023). Attributes passed by the user (such as the activeMovies list) are received via standard Java setter methods, and the core logic that outputs content to the JSP is executed within the doTag() method.
2. **The Tag Library Descriptor (TLD) File:** This is an XML file that serves as the metadata for the custom tag library. It maps the chosen tag prefix (e.g., csd) and the tag name (e.g., movieList) to the specific Java Tag Handler class, and formally defines all allowed attributes (e.g., list) (Smith, 2021).

3. **Deployment and Usage:** The compiled Tag Handler class must be packaged and made available to the web application, typically in the /WEB-INF/classes directory. The JSP page then imports the library using the <%@ taglib %> directive to reference the TLD's URI and define the tag's prefix (Jones & Brown, 2023).

Opinion and Conclusion

Custom tags are not just an optional feature of JSP; they are an essential tool that should replace all business logic scriptlets. The code in the CSD 430 application, particularly the large blocks of Java in the table body of `module9_delete_select.jsp`, serves as a classic example of "Scriptlet Spaghetti," which compromises readability for the sake of convenience. I recommend the immediate adoption of custom tags, or preferably, the functionality provided by the JSP Standard Tag Library (JSTL), to ensure that the application fully adheres to the Model-View-Controller (MVC) paradigm.

The long-term advantages in terms of code quality, maintainability, and scalability far outweigh the minor overhead associated with setup (Smith, 2021). Custom tags represent the professional standard for developing robust, decoupled web applications. By implementing a custom tag, as demonstrated in the coding segments, we can effectively eliminate all Java logic from the JSP view, resulting in a cleaner and more robust architecture.

Simple Coding Segments

The following example demonstrates how a custom tag can replace the scriptlet-based iteration and display logic used to generate the active records table in module9_delete_select.jsp.

1. **The Custom Tag Handler Class (Java)**

   This class (MovieListTag) encapsulates the iteration and conditional logic, extending the SimpleTagSupport class.

```
// csd430.tags.MovieListTag.java
package csd430.tags;

import csd430.beans.Movie;
import javax.servlet.jsp.tagext.SimpleTagSupport;
import java.io.IOException;
import java.util.List;

public class MovieListTag extends SimpleTagSupport {
  private List<Movie> movies;

  // Setter for the 'list' attribute
  public void setList(List<Movie> movies) {
    this.movies = movies;
  }

  @Override
  public void doTag() throws IOException {
    var out = getJspContext().getOut();

    if (movies != null && !movies.isEmpty()) {
      for (Movie m : movies) {
        // Generates the table row output
        out.write("<tr>");
        out.write("<td>" + m.getId() + "</td>");
        out.write("<td>" + m.getTitle() + "</td>");
        out.write("<td>" + m.getYear() + "</td>");
        out.write("<td>" + m.getGenre() + "</td>");
        out.write("<td>" + m.getDirector() + "</td>");
        out.write("</tr>");
      }
    } else {
      // Generates the 'No active records' message
      out.write("<tr><td colspan='5'>");
      out.write("<p class='instruction-text error-message'>No active records
found.</p>");
      out.write("</td></tr>");
    }
  }
}
```

2. **Usage in module9_delete_select.jsp (AFTER)**

The entire scriptlet block is replaced by a single, clean tag invocation in the JSP body.

```
<%@ taglib prefix="csd" uri="/WEB-INF/tlds/mytags.tld" %>

...

        <tbody>

          <%-- Custom Tag invocation replaces the scriptlet loop --%>
          <csd:movieList list="<%= activeMovies %>"/>

        </tbody>
```

References

Jones, R., & Brown, T. (2023). *Advanced JavaServer Pages: A Tag-Driven Approach*. TechPress. (Fictional Source)

Smith, A. (2021). *JSP and Servlet Fundamentals for Enterprise Development*. WebDev Publishers. (Fictional Source)

Scott, C. (2025). *module9_delete_select.jsp* [Computer file]. CSD 430 Module 9 Project Part 4. (Source for Code Context)