Jess Monnier

CSD-430 Module 10 Assignment

5 October 2025

**Creating Custom Tags in JSP: Purpose, Process, and Practical Use**

Custom tags are one of many mechanisms used to create dynamic server-side web content with JavaServer Pages (JSP). With custom tags, we can define elements that encapsulate reusable functionality for JSP pages, minimizing both code reuse and use of servlets directly in webpages. This paper explores the purpose of custom tags, outlines their advantages and disadvantages, explains the steps required to develop them, and reflects on my own understanding and experience as a student learning how they are used.

**Understanding Custom Tags**

A custom tag in JSP is essentially a reusable component that allows developers to separate presentation logic from business logic by encapsulating functionality in a Java class. This class can then be invoked in JSP pages using a tag-like syntax. For example, instead of embedding a block of Java code in every JSP to perform a task like displaying a formatted date, you can create a tag like <my:formatDate> that handles the logic internally.

In the first half of this course, I worked a bit with WebServlets. When I initially read about custom tags, I mistakenly assumed they must serve a similar purpose. However, in researching this paper, I realized they operate at different layers. A servlet acts as a controller that processes HTTP requests, handles form data, and connects to back-end resources like databases. In contrast, custom tags are used primarily within JSP pages to generate parts of the HTML or output, almost like helper components or mini-templates.

**Advantages of Custom Tags**

The main advantage of custom tags is reusability. Instead of repeating code across multiple JSPs, developers can define a custom tag once and use it wherever needed. This makes JSPs cleaner and easier to read. It also promotes a separation of concerns; custom tags focus on presentation or view logic, while servlets and other back-end code manage application logic.

Another advantage is maintainability. When logic for display formatting or conditional rendering is wrapped in a tag, any changes only need to be made in the tag handler class. This reduces the risk of introducing bugs in multiple JSP pages.

Custom tags also help reduce or eliminate the use of scriptlets (<% %>), which are now considered poor practice. Scriptlets mix Java code directly into HTML, making JSPs harder to read and debug. Custom tags, especially when used with Expression Language (EL), support a more modern, component-based approach to building web pages.

**Disadvantages and Considerations**

While custom tags offer many benefits, there are some drawbacks. Creating them involves extra setup: defining a tag handler class, writing a Tag Library Descriptor (TLD) file, and sometimes updating web.xml. For small applications or one-off tasks, this may seem like overkill. Additionally, debugging custom tag behavior can be more complex than using simple servlets (<% %> code) embedded directly in the webpage, especially for beginners.

There is also a learning curve. As someone new to server-side Java, I initially struggled with understanding the difference between using servlets, WebServlets, and custom tags. Until I reviewed a number of practical examples, it wasn’t clear when a custom tag was worth the extra effort.

**Requirements to Create a Custom Tag**

To create a custom tag in JSP, several components are required:

1. Tag handler class – This is a Java class that extends TagSupport, BodyTagSupport, or SimpleTagSupport. The class must override specific methods like doStartTag(), doEndTag(), or doTag() to control the tag's behavior.
2. Tag Library Descriptor (TLD) – This is an XML file that defines the tag’s name, class, and attributes. The JSP container uses this to know how to process the tag.
3. Configuration (optional) – In older servlet versions (pre-3.0), web.xml must reference the TLD file. In newer versions, this can be skipped with annotations or modern packaging.

The first zipped-up example in this module’s resources used a tag handler extending BodyTagSupport, which allowed access to the tag’s body content and supported repeating logic. However, newer JSP applications often use SimpleTagSupport, which simplifies tag development and is part of the JSP 2.0+ API (GeeksforGeeks, n.d.). While BodyTagSupport is still valid, SimpleTagSupport is generally easier to work with and aligns better with modern practices.

**Realizations and Reflections**

One revelation I had while working on my assignment was how well-suited custom tags are for displaying pagination or navigation controls. This is a common UI element that requires logic (e.g., showing current page, disabling "Next" if on last page, etc.) and appears in multiple places. Rather than duplicating this logic across pages using scriptlets, a custom tag can encapsulate it. The JSP remains clean and readable:

<my:pagination currentPage="${page}" totalPages="${total}" baseUrl="results.jsp?page=" />

This kind of abstraction made the use of custom tags "click" for me, showing that they are not just an academic exercise but a genuinely useful tool. I will include an example of what the custom tag class might look like in this scenario separately from this paper.

**Conclusion**

Custom tags in JSP offer a structured and reusable way to manage view logic in server-side Java web applications. While they come with some setup overhead and require an understanding of JSP's tag API, they promote clean code and modular design. Although I was initially confused about their purpose compared to WebServlets, I now see their value as view-layer components. When used appropriately, custom tags improve code readability, reduce duplication, and make large web applications easier to manage.

**References**

GeeksforGeeks. (n.d.). *Custom tags with body in JSP*. GeeksforGeeks. <https://www.geeksforgeeks.org/java/custom-tags-with-body-in-jsp/>

Java EE. (n.d.). *Interface BodyTag*. Java EE 8 Specification. <https://javaee.github.io/javaee-spec/javadocs/javax/servlet/jsp/tagext/BodyTag.html>

SetGetWeb. (n.d.). *Creating custom tags*. <https://www.setgetweb.com/p/webservices/custom_tags.html>