Alisa Steensen 7/15/25

Module 10.2 Server Side Development

Creating Custom Tags

Custom tags are special tags that developers create to perform certain actions in Java Server Pages. Each custom tag is linked to a tag handler; this will contain the logic for what the tag should do. This setup helps keep the business logic separate from the Java Server Page itself. Now by using custom tags, the developers can avoid having to use scriplets. The Java code is written directly inside the JSP. Scriplets can make the page messy and hard to read, so it’s generally better to use customer tags instead for cleaner code.

However, there are some reasons why you should not be using scriplets in your JSP. While both scriplets and custom tags ultimately rely on servlets and offer similar functionality, custom tags are generally favored for several important reasons:

* Cleaner separation of logic – Custom tags help keep the business logic out of the JSP by moving it into a separate Java class called a tag handler. This allows the JSP to manage page flow while delegating the actual processing elsewhere.
* Improved Reading – Pages that don’t include scriplets are much easier to read. Custom tags make the code cleaner, especially for larger applications.
* Better for Front End Development – Easier for front end developers to work with JSP’s without needing to write or understand embedded Java code.
* Reuse – Custom tags can be reused throughout different JSP pages. Scriplets, however, are stuck in the page they’re written in and can’t be easily reused.

When working with JavaServer Pages, custom tags follow a specific process that's handled by the JSP container. As the page runs, the container creates an object for the tag using a constructor that doesn’t take any arguments. If the tag is inside another tag, it sets the parent tag. It also sets the page context, which gives the tag access to things like request and session data. Then, it sets up the tag’s attributes by calling the matching setter methods based on what’s written in the JSP.

Now, once the setup is complete, the container calls the tag lifecycle methods to handle processing. And depending on the type of tag handler used, these methods may include doStartTag(), doAfterBody(), and doEndTag(). Tag pooling may also be used for improving performance or reusing handler objects when attribute values are the same. However, when using SimpleTag or SimpleTagSupport, the server doesn’t always reuse the tag objects and usually creates a new one each time by default.

There are several interfaces and classes available for implementing tag handlers:

* The Tag interface - This provides basic lifecycle methods, which includes the doStartTag() and doEndTag(), and it controls whether the tag’s body and the rest of the page should be processed.
* The IterationTag - This interface extends the Tag interface and allows repeated execution of the tag body through the doAfterBody() method.
* The TagSupport - This class implements Tag and IterationTag and provides default implementations for development.
* The BodyTag – This interface allows buffered evaluation of the tag body and includes methods like doInitBody() and setBodyContent().
* BodyTagSupport - This is a helper class that combines the functionality of BodyTag and TagSupport.
* The SimpleTag – This interface simplifies tag creation. It has a single doTag() method where all logic is written. The tag’s body can be set with setJspBody() using a JspFragment, which encapsulates reusable template code without scriptlets.
* SimpleTagSupport – This provides a basic implementation of SimpleTag.

To set up and describe custom tags, you need a Tag Library Descriptor (TLD) file. This is an XML file, usually found in the WEB-INF folder. It explains how your custom tags work and what they look like. The file starts with a <taglib> section and includes tags like <tlib-version>, <jsp-version>, and <uri> to describe the tag library. Inside the <taglib>, each custom tag is defined using a <tag> element. This includes the tag's name, the Java class that runs it, the type of content it uses, and its attributes. For each attribute, the TLD says if it’s required, what kind of data it accepts, whether you can use expressions, and if the value is a JSP code block (called a JspFragment).

The TLD file and the tag handler class work together to create a tag library. A good example of this is JavaServer Pages Standard Tag Library. To use your own custom tag library in a JSP page, you need to add a taglib directive at the top of the file. This tells the page where to find your tag library (using a URI) and sets a prefix so you can use your custom tags easily in the page.

Custom Tags in JavaServer Pages create a powerful way to organize and simplify the logic behind dynamic web applications. By separating Java code from the top presentation layer, they make JSP pages cleaner and easier to read, especially compared to using scriplets. Custom tags help improve code reuse for front-end and back-end developers, and allows for better organization of logic. Even though setting up custom tags takes a little more effort, it's worth it. Developers can pick from different types of tag handlers and use built-in classes and tools that make the job easier.

Overall, custom tags promote good coding practices. While they may not be necessary for every project, they are a great option for medium to large applications that require a clean structured approach. Understanding how to use custom tags and how to create them, will give developers more flexibility when building Java based web applications.

References:

Rathee, P. (2025, April 28). *Custom Tags with Attributes in JSP*. GeeksforGeeks. Retrieved July 16, 2025, from <https://www.geeksforgeeks.org/java/custom-tags-with-attributes-in-jsp/>

Tutorials Point. (n.d.). *JSP – Custom Tags*. Retrieved July 16, 2025, from <https://www.tutorialspoint.com/jsp/jsp_custom_tags.htm>

BeginnersBook. (2014, January 1). *JSP custom tags with example | JSP tutorial*. Retrieved July 16, 2025, from <https://beginnersbook.com/2014/01/jsp-custom-tags-with-example-jsp-tutorial/>