JSP Custom Tags and JSP Expression Language (EL)

1. JSP Custom Tags

✅ Overview

JSP Custom Tags allow developers to encapsulate reusable functionality into tags, which can be reused across multiple JSP pages. This helps maintain **clean separation of concerns**, improves **readability**, and encourages **code reusability**.

🧱 Structure of a Custom Tag

1. **Tag Handler Class** (extends TagSupport, BodyTagSupport, or implements SimpleTag)
2. **TLD File (Tag Library Descriptor)** - describes the tag and its attributes.
3. **JSP Page** - where the tag is used.

Example: Creating a Simple Custom Tag

Sure! Let's break down the example you've shared for creating a custom JSP tag step by step. This is a simple demonstration where a custom tag prints "Hello from Custom Tag!" on a JSP page.

✅ Step 1: Create Tag Handler Class

File: HelloTag.java

This is a Java class that defines the logic of your custom tag. It extends TagSupport, which is a convenience class for implementing simple tags.

```
public class HelloTag extends TagSupport {

  public int doStartTag() throws JspException {

    try {

      JspWriter out = pageContext.getOut();  // Used to write output to the JSP

      out.print("Hello from Custom Tag!");   // Outputs the message

    } catch (IOException e) {

      e.printStackTrace();

    }
```

```
        return SKIP_BODY; // Tells the JSP engine to skip the body of the tag (if any)
    }
}
```

- doStartTag(): This method is called when the JSP engine encounters the start of the tag.
- SKIP_BODY: Since the tag doesn't have any body content, we skip it.
- JspWriter: Used to send output to the client (i.e., to the browser).

✅ Step 2: Create TLD File (Tag Library Descriptor)

File: hello.tld

This is an XML file that describes your custom tag to the JSP container. It binds the tag name to the Java class that implements its logic.

```
<taglib xmlns="http://java.sun.com/xml/ns/javaee" version="2.1">
  <tlib-version>1.0</tlib-version>
  <short-name>hello</short-name>
  <uri>http://example.com/hello</uri>
  <tag>
    <name>sayHello</name> <!-- This is the tag name you will use in your JSP -->
    <tag-class>mytags.HelloTag</tag-class> <!-- Full class name of your tag handler -->
    <body-content>empty</body-content> <!-- Indicates the tag has no body -->
  </tag>
</taglib>
```

- uri: A unique identifier used to reference the tag library in a JSP file.
- tag-class: Points to the Java class (HelloTag) that implements the tag logic.
- name: Defines what the tag will be called in the JSP (sayHello).

✅ Step 3: Use in JSP

File: Some JSP file (e.g., index.jsp)

```
<%@ taglib uri="http://example.com/hello" prefix="h" %>

<html>

<body>

  <h:sayHello />

</body>

</html>
```

- **<%@ taglib ... %>: This directive registers the tag library using the URI from the TLD file.**
- **prefix="h": Allows you to use custom tags with the h: prefix.**
- **<h:sayHello />: This is where your custom tag is used. When the JSP is rendered, it will output:**
- **Hello from Custom Tag!**

🧠 Summary

You're building a custom tag in JSP, which works like a mini component:

1. **Tag Handler Class (HelloTag.java) – Implements what the tag does.**
2. **TLD File (hello.tld) – Tells the JSP container about your tag.**
3. **JSP Page – Uses the tag just like any built-in tag.**

This modular approach is great for reusability and cleaner JSP pages.

Let me know if you want to add attributes to the tag or make it support a body!

🎯 Advantages of Custom Tags

- Clean and reusable
- MVC-friendly
- Easy to integrate with business logic

🧮 2. JSP Expression Language (EL)

✅ Overview

Expression Language (EL) simplifies the accessibility of data stored in JavaBeans, request, session, and application scopes without writing Java code directly in JSP.

Syntax:

```
${expression}
```

📦 Common Uses

Context Object	Scope
${param.name}	Request Parameter
${requestScope.attr}	Request Attribute
${sessionScope.user}	Session Attribute
${applicationScope.data}	Application Attribute

🔍 Example: Basic EL Usage

Assume a JavaBean:

```
public class Student {
   private String name;
   private int marks;
   // getters and setters
}
```

In Servlet:

```
Student s = new Student();
s.setName("Alice");
s.setMarks(90);
request.setAttribute("student", s);
request.getRequestDispatcher("student.jsp").forward(request, response);
```

In student.jsp:

Name: ${student.name} <br>

Marks: ${student.marks}

🧠 EL Operators

✅ **Arithmetic: +, -, *, /, %**

${10 + 5} // Outputs 15

✅ **Logical: &&, ||, !**

${marks > 50 && passed}

✅ **Relational: ==, !=, <, >, <=, >=**

${student.marks >= 90}

✅ **Empty Operator**

${empty student.name}

🔧 EL Implicit Objects

Object	**Description**
param	Request parameter
paramValues	Request parameter (multi-value)
header	HTTP header
cookie	Cookie values
pageScope, requestScope, sessionScope, applicationScope	Scoped attributes

---

🛠️ Advanced: Accessing Maps/Collections

${studentMap["101"].name}

${list[0].marks}

📘 Conclusion

- **Custom Tags** encapsulate logic and are useful for reusable UI components.
- **EL** provides a clean, declarative way to access server-side data.
- Together, they promote **clean, maintainable**, and **MVC-compliant JSP applications**.

JSP Exception Handling

⚠️ 1. What is Exception Handling in JSP?

Exception handling in JSP refers to the mechanism of **gracefully managing runtime errors** that occur during the execution of JSP pages. Instead of showing raw server errors to users, we handle exceptions to maintain user experience and security.

🧠 2. Why Handle Exceptions in JSP?

- To avoid displaying stack traces to users
- To manage errors in a centralized way
- To provide user-friendly error messages
- To improve maintainability of web applications

🧱 3. Types of Exceptions in JSP

Type	Description
Checked Exceptions	Must be handled or declared (e.g., IOException)
Unchecked Exceptions	Occur at runtime (e.g., NullPointerException)
JSP Exceptions	JSP-specific errors during translation or execution

---

🔧 4. Exception Handling Techniques in JSP

✅ A. Using errorPage and isErrorPage Directives

- errorPage — set in the JSP page where the exception might occur
- isErrorPage — set in the error-handling JSP page

✏️ Example

📄 main.jsp — page where error may occur:

```
<%@ page errorPage="error.jsp" %>
```

```
<html>
<body>
  <%
    int a = 10 / 0; // This will cause ArithmeticException
  %>
</body>
</html>
```

📄 **error.jsp — error handler page:**

```
<%@ page isErrorPage="true" %>
<html>
<body>
  <h2>An error occurred!</h2>
  <p>Error Message: <%= exception.getMessage() %></p>
  <p>Type: <%= exception.getClass().getName() %></p>
</body>
</html>
```

📌 **Note**: exception is an implicit object available only in pages with isErrorPage="true".

✅ B. Using Web.xml for Centralized Exception Handling

You can configure exception handling in web.xml to direct specific exceptions or HTTP error codes to a common error page.

🛠️ Example

📄 **web.xml configuration:**

```
<error-page>
  <exception-type>java.lang.ArithmeticException</exception-type>
  <location>/error.jsp</location>
</error-page>
```

```
<error-page>
  <error-code>404</error-code>
  <location>/notfound.jsp</location>
</error-page>
```

✅ C. Try-Catch Block in Scriptlet (Not Recommended for Modern JSP)

```
<%
  try {
      int x = 10 / 0;
  } catch (Exception e) {
      out.println("Handled Error: " + e.getMessage());
  }
%>
```

🔴 **Not recommended** — breaks MVC principle and mixes Java code with HTML.

📌 5. Best Practices

- Use errorPage and isErrorPage for local handling.
- Use web.xml for centralized/global error management.
- Avoid using scriptlets; prefer servlets and EL for logic.
- Always log errors (using log4j, SLF4J, etc.) for debugging.
- Customize error pages for user-friendliness.

🧪 6. Practical Activity

Task: Create a JSP application that handles:

- Division by zero (ArithmeticException)
- Null value access (NullPointerException)
- A 404 error (missing page)

Use both errorPage directive and web.xml configuration.

🎯 Summary

Concept	Description
errorPage	Used to define the error handler page
isErrorPage	Set to true in the error page to access the exception object
web.xml	Allows central configuration of exception handling for the whole application
Try-Catch	Works, but discouraged in modern JSP