सी डैक | प्रगत संगणन विकास केंद्र
CDAC | CENTRE FOR DEVELOPMENT OF ADVANCED COMPUTING

Jakarta Server Pages -JSP

(Java Server Pages)

STDC

What Is a JSP Page?

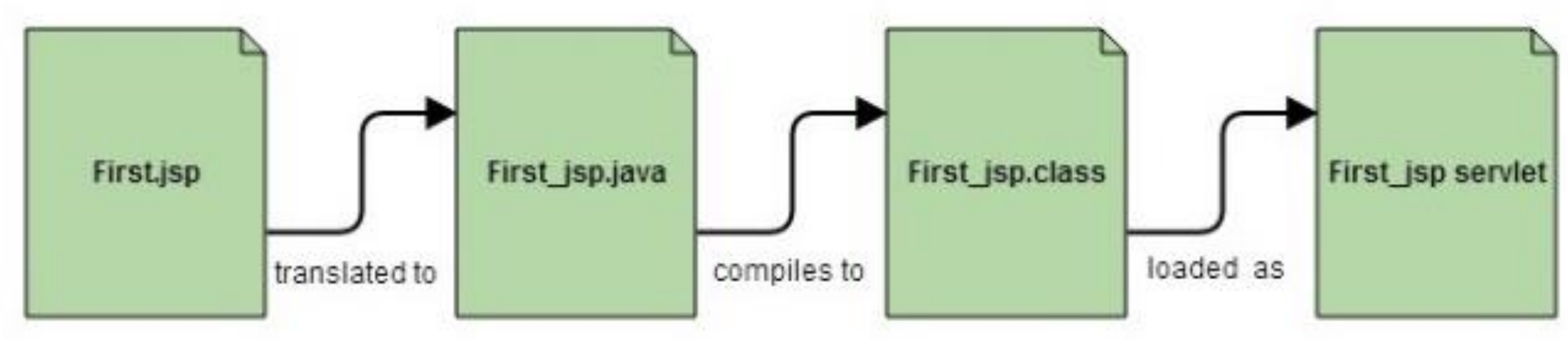
- A **JSP page** is a text document that contains two types of text:
  - ***static data*** : which can be expressed in any text-based format (such as HTML, SVG, WML, and XML)
  - ***JSP elements:*** which construct dynamic content.
- JSP allows as to separate the dynamic content of a webpage from its static presentation content.
- In servlet programming we requires additional files such as web.xml, .html and java class files to generate required web pages.
- JSP pages are opposite of Servlets as *a servlet adds HTML code inside Java code.*
- *JSP adds Java code inside HTML using JSP tags*.
- The Servlet embed html into Java code through out.println statements.

What Is a JSP Page?

- JSP enables us to write HTML pages containing tags that run powerful Java programs.
- **JSP separates presentation and business logic** as Web designer can design and update JSP pages without learning the Java language and Java Developer can also write code without concerning the web design.

Relation between JSP and Servlets

- Usually JSP and Servlets are very closely related to each other.
- Each JSP is first compiled into a Servlet before it can be used.
- When a first call is made to JSP, it translates to Java Servlet source code and then with the help of compiler it gets compiled to Java Servlet class file.
- This class file gets executed in the server and result are returned back to client.



JSP Processing:

The following steps explain how the web server creates the web page using JSP:

1. As with a normal page, your browser sends an HTTP request to the web server.
2. The web server recognizes that the HTTP request is for a JSP page and forwards it to a JSP engine. This is done by using the URL or JSP page which ends with .jsp instead of .html.
3. The JSP engine loads the JSP page from disk and converts it into a servlet content. This conversion is very simple in which all template text is converted to println( ) statements and all JSP elements are converted to Java code that implements the corresponding dynamic behavior of the page.
4. The JSP engine compiles the servlet into an executable class and forwards the original request to a servlet engine.

JSP Processing:

5. A part of the web server called the servlet engine loads the Servlet class and executes it. During execution, the servlet produces an output in HTML format, which the servlet engine passes to the web server inside an HTTP response.
6. The web server forwards the HTTP response to your browser.

JSP API

The JSP API consists of two packages:

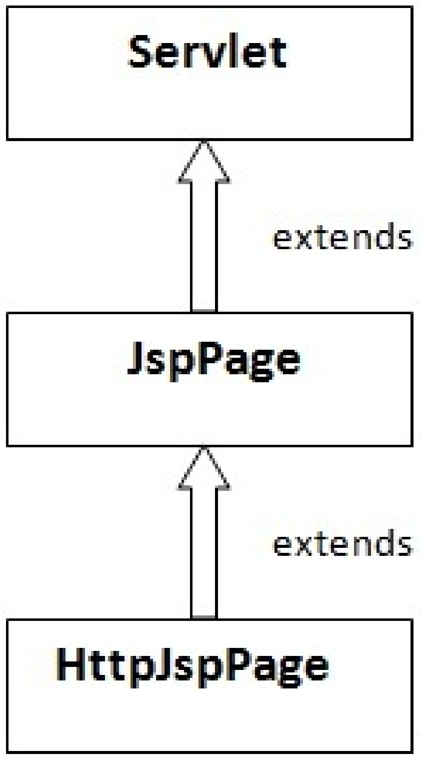
- javax.servlet.jsp
- javax.servlet.jsp.tagext

The javax.servlet.jsp package has two interfaces and classes.The two interfaces are as follows:

- JspPage
- HttpJspPage

JspPage interface

- According to the JSP specification, all the generated servlet classes must implement the JspPage interface. It provides two life cycle methods.



Methods of JspPage interface

- **public void jspInit():** It is invoked only once during the life cycle of the JSP when JSP page is requested firstly. It is used to perform initialization. It is same as the init() method of Servlet interface.

- **public void jspDestroy():** It is invoked only once during the life cycle of the JSP before the JSP page is destroyed. It can be used to perform some clean up operation.

HttpJspPage interface

- The HttpJspPage interface provides the one life cycle method of JSP. It extends the JspPage interface.

Method of HttpJspPage interface:

- **public void _jspService():** It is invoked each time when request for the JSP page comes to the container. It is used to process the request. The underscore _ signifies that you cannot override this method.

JSP-Life Cycle

- A JSP life cycle can be defined as the entire process from its creation till the destruction which is similar to a servlet life cycle with an additional step which is required to compile a JSP into servlet.
- The following are the paths followed by a JSP.

  (1) JSP page translation

  (2) JSP page compilation

  (3) load class

  (4) create instance

  (5) call the jspInit method

  (6) call the_jspService method

  (7) call the jspDestroy method.

JSP page translation

- The translation of jsp is automatically done by the web server.
- The translation of JSP can take place in pre-compilation phase.

- Following are the operations…………
  - Locate the Requested JSP page.
  - Validate syntactically correctness of the JSP page.
  - Interprets the standard JSP directives, actions, and custom actions used in the JSP page.
  - Write the source code of equivalent Servlet for the JSP page.

JSP page translation

The contract on the JSP page implementation class:

- Implements HttpJspPage if the protocol is HTTP, or JspPage otherwise.
- All of the methods in the Servlet interface are declared final.

JSP Compilation:

- When a browser asks for a JSP, the JSP engine first checks to see whether it needs to compile the page.
- If the page has never been compiled, or if the JSP has been modified since it was last compiled, the JSP engine compiles the page.
- In this stage, the JSP container compiles the java source code for the corresponding servlet and converts it into java byte code.

JSP Initialization

- When a container loads a JSP it invokes the jspInit() method before servicing any requests.
- If you need to perform JSP-specific initialization, override the jspInit() method:

```
public void jspInit()
{
 // Initialization code…
 }
```

- Typically initialization is performed only once and as with the servlet init method, you generally initialize database connections, open files, and create lookup tables in the jspInit method.

JSP Execution

- This phase of the JSP life cycle represents all interactions with requests until the JSP is destroyed.
- Whenever a browser requests a JSP and the page has been loaded and initialized, the JSP engine invokes the **_jspService()** method in the JSP.
- The _jspService() method takes an **HttpServletRequest** and an **HttpServletResponse** as its parameters as follows:

```
void _jspService(HttpServletRequest request, HttpServletResponse
   response)
{ // Service handling code...
}
```

- The _jspService() method of a JSP is invoked once per a request and is responsible for generating the response for that request and this method is also responsible for generating responses.

JSP Cleanup:

- The destruction phase of the JSP life cycle represents when a JSP is being removed from use by a container.
- The **jspDestroy()** method is the JSP equivalent of the destroy method for servlets.
- Override jspDestroy when you need to perform any cleanup, such as releasing database connections or closing open files.
- The jspDestroy() method has the following form:

```
public void jspDestroy()
{
 // Your cleanup code goes here.
}
```

JSP – Syntax

- A Scriptlet can contain any number of JAVA language statements, variable or method declarations, or expressions that are valid in the page scripting language.
- Following is the syntax of Scriptlet:
- <% code fragment %>

```
<html>
  <head>
    <title>My First JSP Page</title>
  </head>
 <% int count = 0;  %>
 <body>
    Page Count is:
    <% out.println(++count); %>
 </body>
</html>
```

JSP (Scripting elements)

- The scripting elements provides the ability to insert java code inside the jsp.
- JSP Scripting element are written inside <% %> tags.
- These code inside <% %> tags are processed by the JSP engine during translation of the JSP page.
- Any other text in the JSP page is considered as HTML code or plain text.

Scripting Element	Example
Comment	<%-- comment --%>
Scriptlet	<% scriplets %>
Declaration	<%! declarations %>
Directive	<%@ directive %>
Expression	<%= expression %>

JSP Comment

- JSP Comment is used when you are creating a JSP page and want to put in comments about what you are doing. JSP comments are only seen in the JSP page. These comments are not included in servlet source code during translation phase, nor they appear in the HTTP response. Syntax of JSP comment is as follow
- <%-- JSP comment --%>

- **Simple Example of JSP Comment**

```
<html>
  <head>
    <title>My First JSP Page</title>
  </head>
 <%
    int count = 0;
 %>
 <body>
    <%-- Code to show page count  --%>
    Page Count is  <% out.println(++count); %>
 </body>
```

JSP scriptlet

- Scriptlet Tag allows you to write java code inside JSP page. Syntax of Scriptlet Tag is as follows :

- <% *java code* %>

Example of Scriptlet

```
<html>
  <head>
    <title>My First JSP Page</title>
  </head>
 <%
   int count = 0;
 %>
 <body>
    Page Count is
    <% out.println(++count); %>
 </body>
</html>
```

Mixing scriptlet Tag and HTML

- Let's see how we can utilize the power of JSP scripting with HTML to build dynamic webpages with help of a few examples.

```
<table border = 1>
<%
   for ( int i = 0; i < n; i++ ) {
%>
      <tr>
      <td>Number</td>
      <td><%= i+1 %></td>
      </tr>
<%
   }
%>
</table>
```

Declaration Tag

- We know that at the end a JSP page is translated into Servlet class.
- So when we declare a variable or method in JSP inside **Declaration Tag**, it means the declaration is made inside the Servlet class but outside the service(or any other) method.
- You can declare static member, instance variable and methods inside **Declaration Tag.**

Syntax of Declaration Tag :

<%! *declaration* %>

Example of Declaration Tag

```
<html>
  <head>
    <title>My First JSP Page</title>
  </head>
 <%!
   int count = 0;
 %>
 <body>
    Page Count is:
    <% out.println(++count); %>
 </body>
</html>
```

Example of Declaration Tag

```
public class hello_jsp extends HttpServlet
{
 int count=0;
 public void _jspService(HttpServletRequest request, HttpServletResponse
    response) throws IOException, ServletException
   {
    PrintWriter out = response.getWriter();
    response.setContenType("text/html");
    out.write("<html><body>");
        out.write("Page count is:");
    out.print(++count);
    out.write("</body></html>");
  }
}
```

In the above servlet, we can see that variable count is declared outside the _jspservice() method. If we declare the same variable using scriptlet tag, it will come inside the service method, as seen before.

When to use Declaration tag and not scriptlet tag

- If you want to include any method in your JSP file, then you must use the declaration tag, because during translation phase of JSP, methods and variables inside the declaration tag, becomes instance methods and instance variables and are also assigned default values.

```
<html>
   <head>
      <title>My First JSP Page</title>
   </head>
  <%!
     int count = 0;
     int getCount() {
        System.out.println( "In getCount() method" );
        return count;
     }
  %>
 <body>
      Page Count is:
      <% out.println(getCount()); %>
 </body>
</html>
```

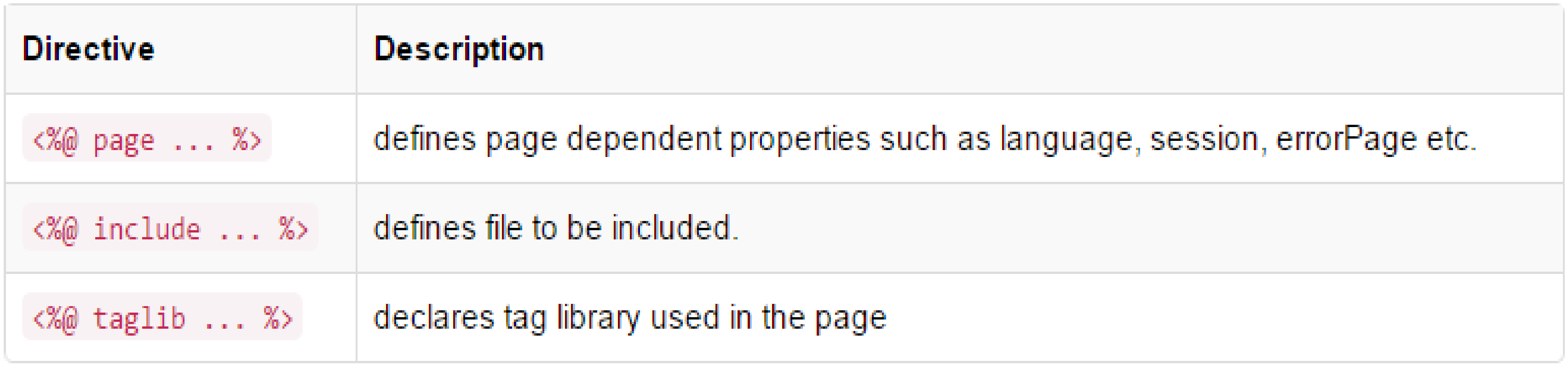
Previous code will be translated into following servlet :

```
public class hello_jsp extends HttpServlet
{
 int count = 0;
 int getCount() {
    System.out.println( "In getCount() method" );
    return count;
 }
 public void _jspService(HttpServletRequest request, HttpServletResponse response)
                       throws IOException,ServletException
  {
    PrintWriter out = response.getWriter();
    response.setContenType("text/html");
    out.write("<html><body>");
    out.write("Page count is:");
    out.print(getCount());
    out.write("</body></html>");   } }
```

- While, anything we add in scriptlet tag, goes inside the _jspservice() method, therefore we cannot add any function inside the scriptlet tag, as on compilation it will try to create a function getCount() inside the service method, and in Java, method inside a method is not allowed.

Directive Tag

- **Directive Tag** gives special instruction to Web Container at the time of page translation. Directive tags are of three types: **page**, **include** and **taglib**.



Directive	Description
`<%@ page ... %>`	defines page dependent properties such as language, session, errorPage etc.
`<%@ include ... %>`	defines file to be included.
`<%@ taglib ... %>`	declares tag library used in the page

- The **Page directive** defines a number of page dependent properties which communicates with the Web Container at the time of translation.
- Basic syntax of using the page directive is

```
<%@ page attribute="value" %>
```

where attributes can be one of the following :

import attribute

- The import attribute defines the set of classes and packages that must be imported in servlet class definition. For example

```
<%@ page import="java.util.Date" %>
```

or

```
<%@ page import="java.util.Date,java.net.*" %>
```

language attribute

- language attribute is added to specify the scripting language used in JSP page. It's default value is "java" and this is the only value it can have. May be in future, JSPs provide support to include other scripting languages like C++ or PHP too.

```
<%@ page language="java" %>
```

extends attribute

- This attribute is used to define the super class of the generated servlet code. This is very rarely used and we can use it if we have extended HttpServlet and overridden some of it's implementations.
  - `<%@ page extends="somePackage.SomeClass" %>`

session attribute

- By default JSP page creates a session but sometimes we don't need session in JSP page. We can use this attribute to indicate compiler to not create session by default. It's default value is true and session is created.

```
<%@ page session="true" %>
```

isThreadSafe attribute

- isThreadSafe attribute declares whether the JSP is thread-safe. The value is either true or false. By default, all JSPs are considered thread-safe. If you set the isThreadSafe option to false, the JSP engine makes sure that only one thread at a time is executing your JSP.

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

***errorPage* attribute**

- errorPage attribute indicates another JSP page that will handle all the run time exceptions thrown by current JSP page. It specifies the URL path of another page to which a request is to be dispatched to handle run time exceptions thrown by current JSP page.

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

***contentType* attribute**

- contentType attribute defines the MIME type for the JSP response.

```
<%@ page contentType="text/xml" %>
```

***autoFlush* attribute**

- autoFlush attribute is to control the buffer output. Its default value is true and output is flushed automatically when buffer is full. If we set it to false, the buffer will not be flushed automatically and if it's full, we will get exception for buffer overflow. We can use this attribute when we want to make sure that JSP response is sent in full or none.

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

***buffer* attribute**

- The buffer attribute sets the buffer size in kilobytes to handle output generated by the JSP page.The default size of the buffer is 8Kb.

```
<%@ page buffer="8kb" %>
```

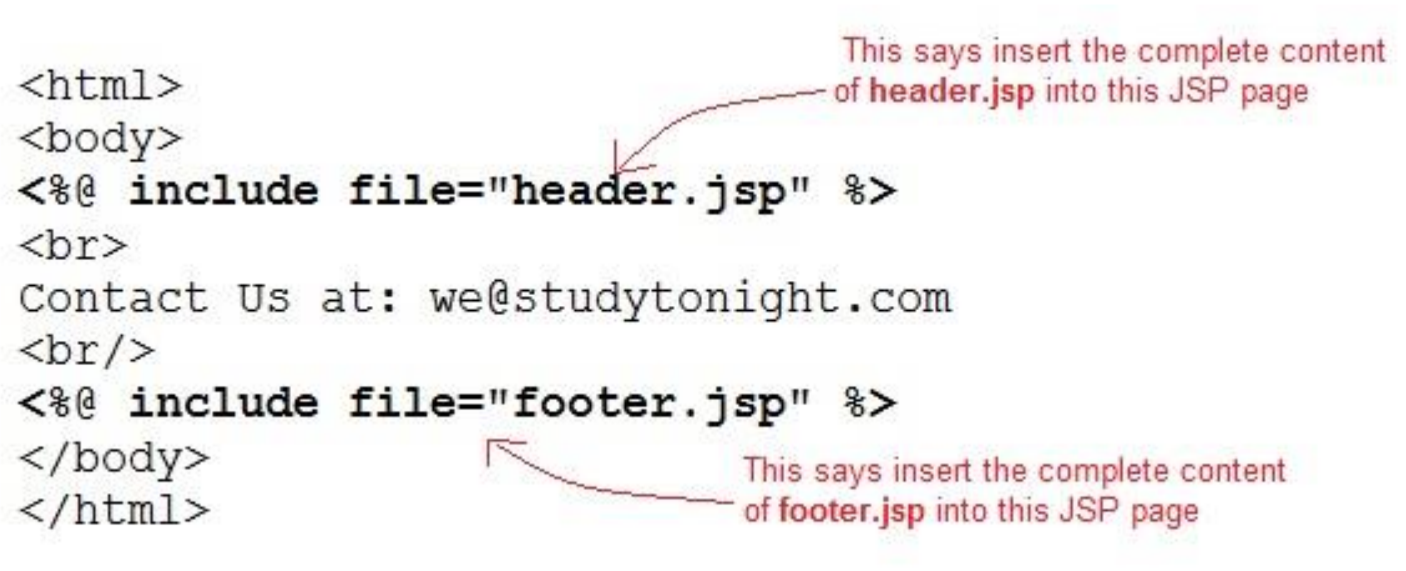
isErrorPage attribute

- isErrorPage attribute declares whether the current JSP Page represents another JSP's error page.

```
<%@ page isErrorPage="true" %>
```

Include Directive

- The *include* directive tells the Web Container to copy everything in the included file and paste it into current JSP file. Syntax of **include** directive is:
- <%@ **include** file="filename.jsp" %>



```
<html>
<body>
<%@ include file="header.jsp" %>
<br>
Contact Us at: we@studytonight.com
<br/>
<%@ include file="footer.jsp" %>
</body>
</html>
```

This says insert the complete content of header.jsp into this JSP page
This says insert the complete content of footer.jsp into this JSP page

Example of include directive

- welcome.jsp

```
<html>
   <head>
      <title>Welcome  Page</title>
   </head>
<body>
      <%@ include file="header.jsp"  %>
      Welcome, User
 </body>
</html>
```

- **header.jsp**

```
<html>
  <body>
    <img src="header.jpg" alt="This is Header image" / >
  </body>
</html>
```

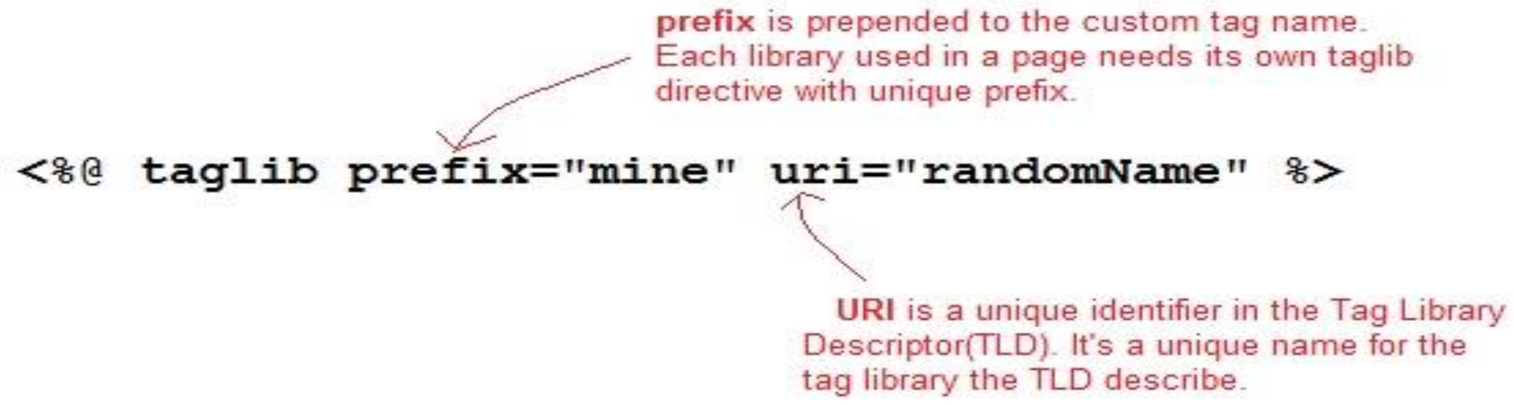
- The example above is showcasing a very standard practice. Whenever we are building a web application, with webpages, all of which have the top navbar and bottom footer same. We make them as separate jsp files and include them using the include directive in all the pages. Hence whenever we have to update something in the top navbar or footer, we just have to do it at one place. Handy, isn't it?

Taglib Directive

- The **taglib** directive is used to define tag library that the current JSP page uses. A JSP page might include several tag library.
- Java Server Pages Standard Tag Library (JSTL), is a collection of useful JSP tags, which provides many commonly used core functionalities.
- It has support for many general, structural tasks such as iteration and conditionals, readymade tags for manipulating XML documents, internationalization tags, and for performing SQL operations.

- Syntax of taglib directive is:

  <%@ *taglib* **prefix**="prefixOfTag" **uri**="uriOfTagLibrary" %>
- The prefix is used to distinguish the custom tag from other libary custom tag.
- Prefix is prepended to the custom tag name. Every custom tag must have a prefix.
- The URI is the unique name for Tag Library.



```
<%@ taglib prefix="mine" uri="randomName" %>
```

prefix is prepended to the custom tag name. Each library used in a page needs its own taglib directive with unique prefix.

URI is a unique identifier in the Tag Library Descriptor(TLD). It's a unique name for the tag library the TLD describe.

Using Taglib Directive

- To use the JSTL in your application you must have the jstl.jar in your webapps /WEB-INF/lib directory.
- There are many readymade JST Libraries available which you use to make your life easier. Following is a broad division on different groups of JST libraries :

Core Tags - URI → *http://java.sun.com/jsp/jstl/core*

Formatting Tags - URI → *http://java.sun.com/jsp/jstl/fmt*

SQL Tags - URI → *http://java.sun.com/jsp/jstl/sql*

XML Tags - URI → *http://java.sun.com/jsp/jstl/xml*

JSTL Functions - URI → *http://java.sun.com/jsp/jstl/functions*

Expression Tag

- Expression Tag is used to print out java language expression that is put between the tags. An expression tag can hold any java language expression that can be used as an argument to the **out.print()** method. Syntax of Expression Tag

  <%= *JavaExpression* %>

- **When the Container sees this**

  <%= (2*5) %> **It turns it into this:**

  out.print((2*5));

- **Note:** Never end an expression with semicolon inside Expression Tag. Like this:

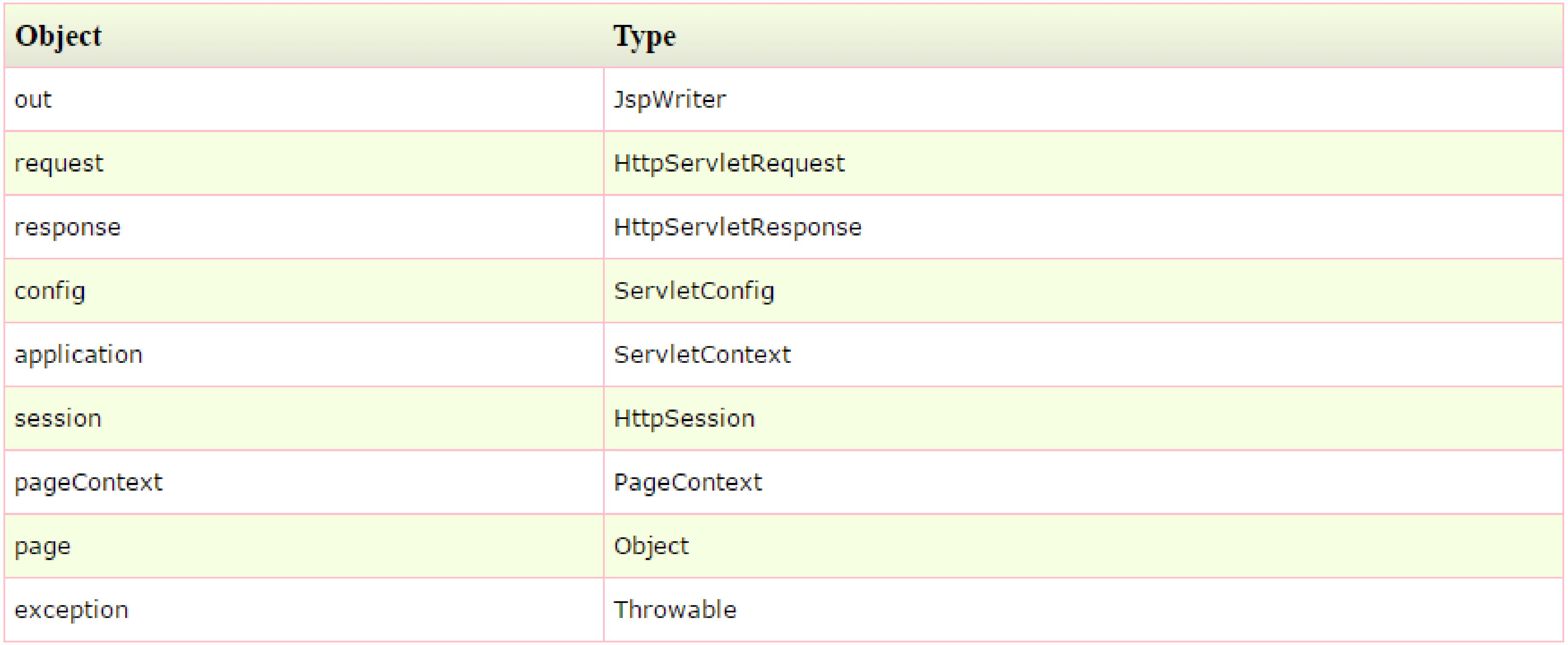
  <%= (2*5); %>

Example of Expression Tag

```
<html>
  <head>
    <title>My First JSP Page</title>
  </head>
 <%
   int count = 0;
 %>
 <body>
    Page Count is  <%= ++count %>
 </body>
</html>
```

Implicit Objects in JSP

- There are **9 jsp implicit objects**. These objects are *created by the web container* that are available to all the jsp pages.
- The available implicit objects are out, request, config, session, application etc.



Object	Type
out	JspWriter
request	HttpServletRequest
response	HttpServletResponse
config	ServletConfig
application	ServletContext
session	HttpSession
pageContext	PageContext
page	Object
exception	Throwable

1. JSP out implicit object

- For writing any data to the buffer, JSP provides an implicit object named out. It is the object of JspWriter. In case of servlet you need to write:

```
PrintWriter out=response.getWriter();
```

- But in JSP, you don't need to write this code.

```
<html>
<body>
<% out.print("Today is:"+java.util.Calendar.getInstance().getTime()); %>
</body>
</html>
```

2. JSP request implicit object

- The **JSP request** is an implicit object of type HttpServletRequest i.e. created for each jsp request by the web container.
- It can be used to get request information such as parameter, header information, remote address, server name, server port, content type, character encoding etc.
- It can also be used to set, get and remove attributes from the jsp request scope.

Index.html

```
<form action="welcome.jsp">
<input type="text"  name="uname">
<input type="submit"  value="go"><br/>
</form>
```

Welcome.jsp

```
<%
String name=request.getParameter("uname");
out.print("welcome "+name);
%>
```

3. JSP response implicit object

- In JSP, response is an implicit object of type HttpServletResponse. The instance of HttpServletResponse is created by the web container for each jsp request.
- It can be used to add or manipulate response such as redirect response to another resource, send error etc.

index.html

```
<form action="welcome.jsp">
<input type="text"  name="uname">
<input type="submit"  value="go"><br/>
</form>
```

welcome.jsp

```
<%
response.sendRedirect("http://www.google.com");
%>
```

4) JSP config implicit object

- In JSP, config is an implicit object of type *ServletConfig*.
- This object can be used to get initialization parameter for a particular JSP page.
- The config object is created by the web container for each jsp page.
- Generally, it is used to get initialization parameter from the web.xml file.

index.html

```
<form action="welcome.jsp">
<input type="text"  name="uname">
<input type="submit"  value="go"><br/>
</form>
```

```
</servlet-mapping>
        <servlet>
        <servlet-name>myServlet</servlet-name>
        <jsp-file>/configJsp.jsp</jsp-file>

        <init-param>
            <param-name>Institute</param-name>
            <param-value>Config Software Solutions</param-value>
        </init-param>

    </servlet>
    <servlet-mapping>
        <servlet-name>myServlet</servlet-name>
        <url-pattern>/configJsp</url-pattern>
    </servlet-mapping>
```

```
<%
out.print(request.getParameter("uname"));
String driver=config.getInitParameter("dname");
out.print("driver name is="+driver);
%>
```

5) JSP application implicit object

- In JSP, application is an implicit object of type *ServletContext*.
- The instance of ServletContext is created only once by the web container when application or project is deployed on the server.
- This object can be used to get initialization parameter from configuration file (web.xml).
- It can also be used to get, set or remove attribute from the application scope.
- This initialization parameter can be used by all jsp pages.

Index.html

```
<form action="welcome">
<input type="text" name="uname">
<input type="submit" value="go"><br/>
</form>
```

```
<context-param>
<param-name>dname</param-name>
<param-
value>sun.jdbc.odbc.JdbcOdbcDriver</param-value>
</context-param>
```

```
<%
out.print("Welcome "+request.getParameter("uname"));
String driver=application.getInitParameter("dname");
out.print("driver name is="+driver);
%>
```

6) session implicit object

- In JSP, session is an implicit object of type HttpSession.The Java developer can use this object to set,get or remove attribute or to get session information.

Index.html

```
<html>
<body>
<form action="welcome.jsp">
<input type="text"  name="uname">
<input type="submit" value="go"><br/>
</form>
</body>
</html>
```

Welcome.jsp

```
<html>
<body>
<%
String name=request.getParameter("uname");
out.print("Welcome "+name);
session.setAttribute("user",name);
<a href="second.jsp">second jsp page</a>
%>
</body>
</html>
```

Second.jsp

```
<html>
<body>
<%
String name=(String)session.getAttribute("user");
out.print("Hello "+name);
%>
</body>
</html>
```

7) pageContext implicit object

- In JSP, pageContext is an implicit object of type PageContext class.The pageContext object can be used to set,get or remove attribute from one of the following scopes:
  - page
  - request
  - session
  - application
- In JSP, page scope is the default scope.

Index.html

```
<html>
<body>
<form action="welcome.jsp">
<input type="text"  name="uname">

<input type="submit" value="go">
<br/>
</form>
</body>
</html>
```

Welcome.jsp

```
<html>
<body>
<%
String name=request.getParameter("uname");
out.print("Welcome "+name);
pageContext.setAttribute("user",name,PageContext.SESSION_SCOPE);
<a href="second.jsp">second jsp page</a>
%>
</body>
</html>
```

second.jsp

```
<html>
<body>
<%
String name=(String)pageContext.getAttribute("user",PageContext.SESSION_SCOPE);
out.print("Hello "+name);
%>
</body>
</html>
```

8) page implicit object

- In JSP, page is an implicit object of type Object class. This object is assigned to the reference of auto generated servlet class. It is written as:

```
Object page=this;
```

- For using this object it must be cast to Servlet type.
- For example:

```
<% (HttpServlet)page.log("message"); %>
```

Since, it is of type Object it is less used because you can use this object directly in jsp.

For example:

```
<% this.log("message"); %>
```

9) exception implicit object

- In JSP, exception is an implicit object of type java.lang.Throwable class. This object can be used to print the exception. But it can only be used in error pages.It is better to learn it after page directive. Let's see a simple example:

<u>error.jsp</u>

```
<%@ page isErrorPage="true" %>
<html>
<body>
 Sorry following exception occured:<%= exception %>
</body>
</html>
```

Exception Handling

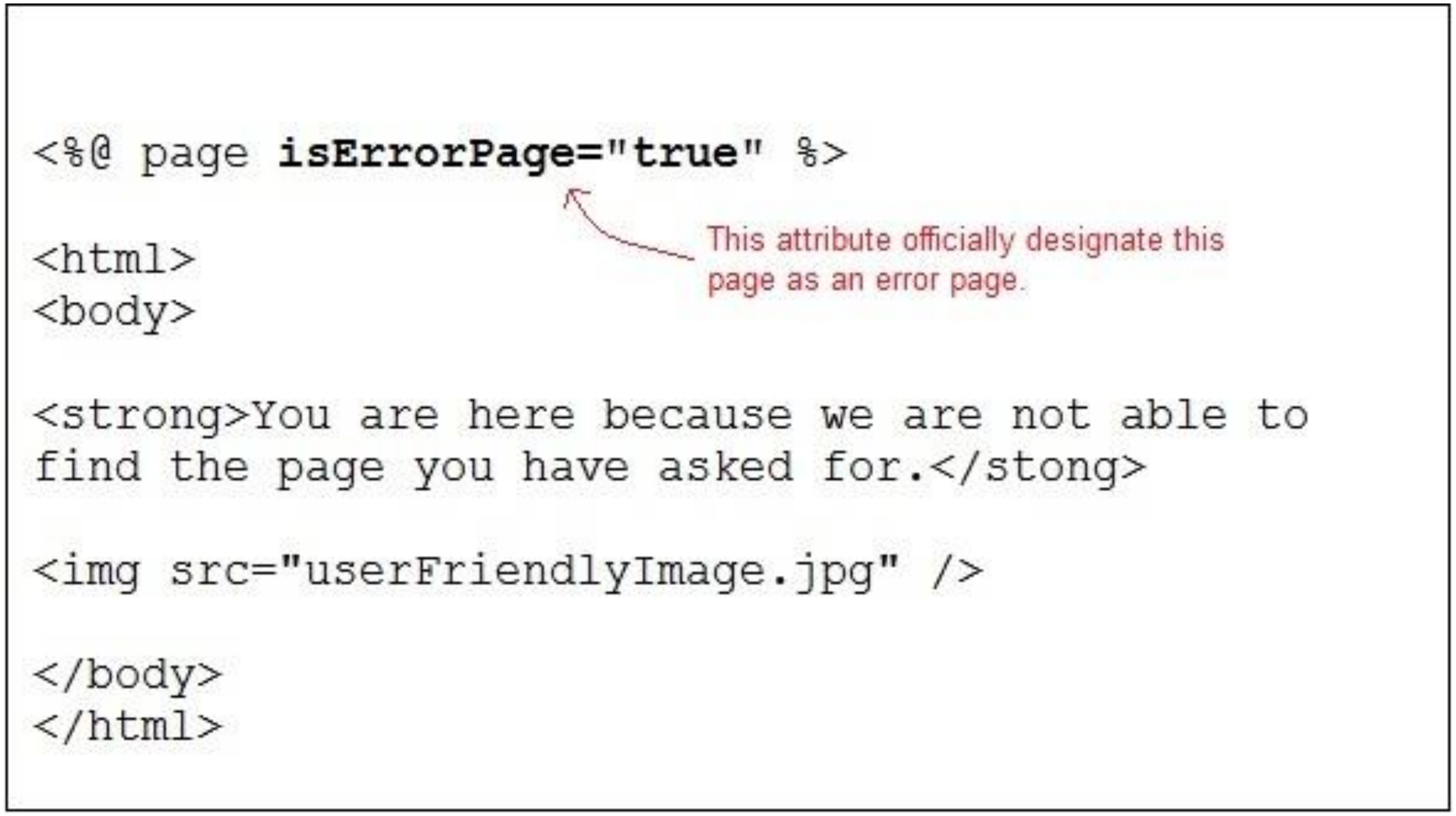
- Exception Handling is a process of handling exceptional condition that might occur in your application.
- Exception Handling in JSP is much easier than Java Technology exception handling.
- Although JSP Technology also uses the same exception class objects.
- It is quite obvious that you dont want to show error stack trace to any random user surfing your website.
- You can't prevent all errors in your application but you can atleast give a user friendly error response page.

Ways to perform Exception Handling in JSP

- JSP provide 3 different ways to perform exception handling:
  - Using **isErrorPage** and **errorPage** attribute of page directive.
  - Using **<error-page>** tag in **Deployment Descriptor**.
  - Using simple try...catch block.

isErrorPage attribute

- isErrorPage attribute in page directive officially appoints a JSP page as an error page.

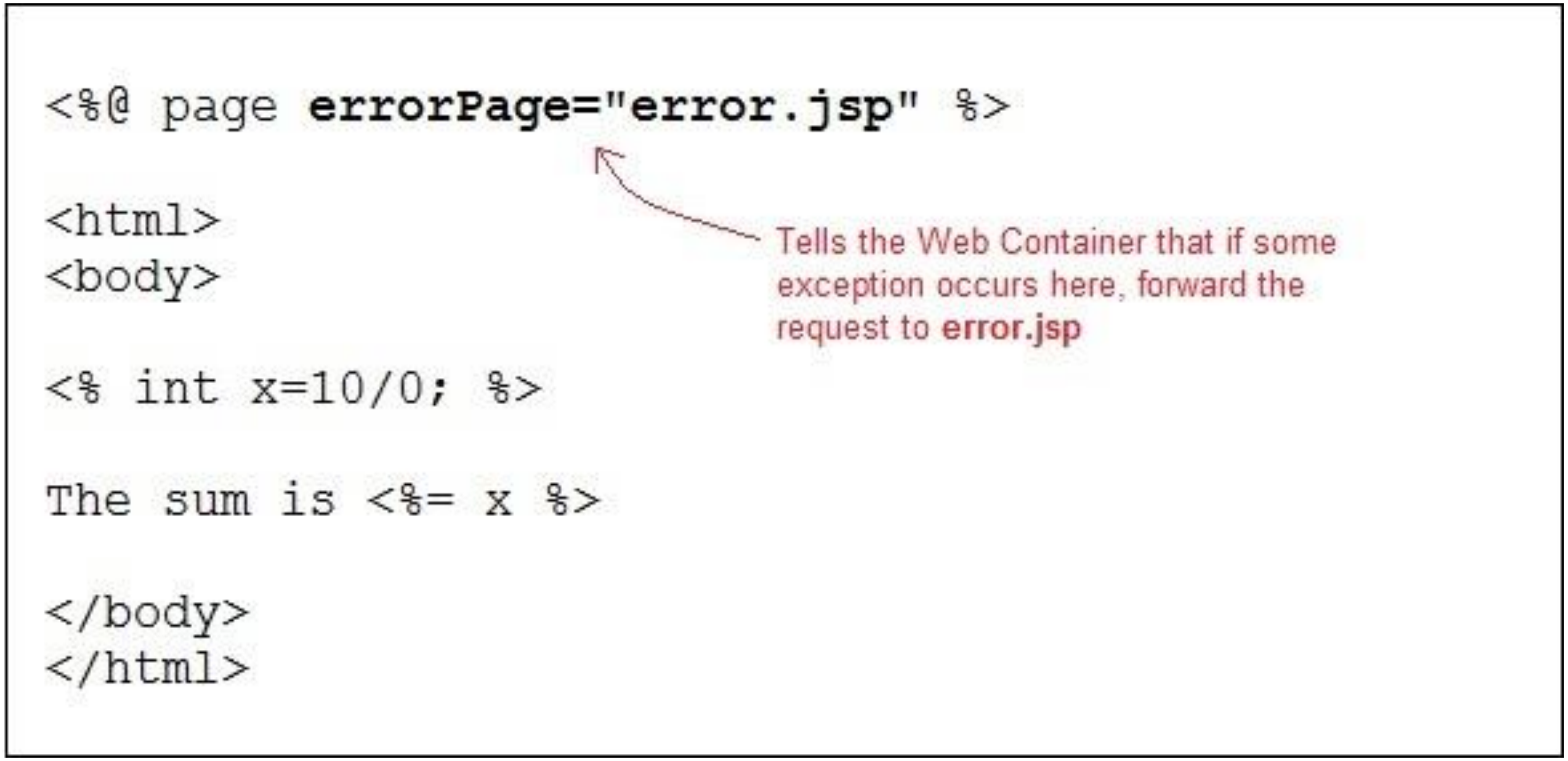


```
<%@ page isErrorPage="true" %>

<html>
<body>

<strong>You are here because we are not able to
find the page you have asked for.</stong>

<img src="userFriendlyImage.jpg" />

</body>
</html>
```

This attribute officially designate this page as an error page.

errorPage attribute

- errorPage attribute in a page directive informs the Web Container that if an exception occurs in the current page, forward the request to the specified error page.



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

<html>
<body>

<% int x=10/0; %>

The sum is <%= x %>

</body>
</html>
```

Tells the Web Container that if some exception occurs here, forward the request to **error.jsp**

Declaring error page in Deployment Descriptor

- You can also declare error pages in the DD for the entire Web Application.
- Using <error-page> tag in the **Deployment Descriptor**.
- You can even configure different error pages for different exception types, or HTTP error code type(503, 500 etc).

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

Declaring an error page based on HTTP Status code

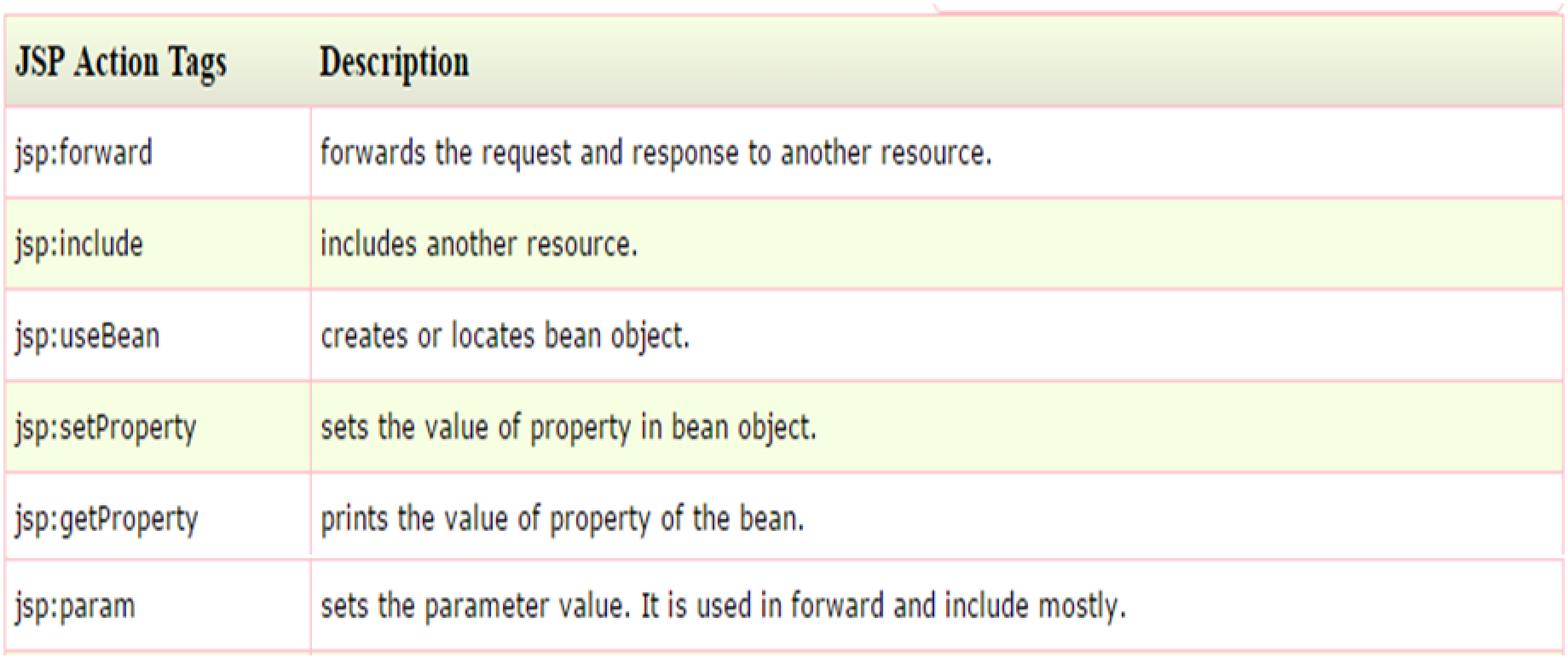
```
<error-page>
 <error-code>
       404
</error-code>
 <location>
      /error.jsp
</location>
</error-page>
```

Using the try...catch block

```
<html>
<head>
  <title>Try...Catch  Example</title>
</head>
<body>
 <%
  try{
     int i = 100;
     i = i / 0;
     out.println("The answer is " + i);
  }
  catch (Exception e){
     out.println("An exception occurred: " + e.getMessage());
  }
 %> </body></html>
```

Standard Tag(Action Element)

- JSP specification provides **Standard**(Action) tags for use within your JSP pages.
- These tags are used to remove or eliminate scriptlet code from your JSP page because scriptlet code are technically not recommended now a days.
- It's considered to be bad practice to put java code directly inside your JSP page.
- Standard tags begin with the jsp: prefix.
- There are many JSP Standard Action tag which are used to perform some specific task.
- There are many JSP action tags or elements. Each JSP action tag is used to perform some specific tasks.
- The action tags are used to control the flow between pages and to use Java Bean. The Jsp action tags are given below.



JSP Action Tags	Description
jsp:forward	forwards the request and response to another resource.
jsp:include	includes another resource.
jsp:useBean	creates or locates bean object.
jsp:setProperty	sets the value of property in bean object.
jsp:getProperty	prints the value of property of the bean.
jsp:param	sets the parameter value. It is used in forward and include mostly.

jsp:forward action tag

- The jsp:forward action tag is used to forward the request to another resource it may be jsp, html or another resource.
- If the resource is dynamic we can use a jsp:param tag to pass the name and value of the parameter to the resource.
- Jsp:forward action tag works as same as forward() in requestdispatcher.

```
<jsp:forward path=“relativeURL”/>
                    Or
<jsp:forward path=“relativeURL”>
<jsp:param name=“key” value=“value”/>
</jsp:forward>
```

jsp:include action tag

- The **jsp:include action tag** is used to include the content of another resource it may be jsp, html or servlet.
- The jsp include action tag includes the resource at request time so it is **better for dynamic pages** because there might be changes in future.
- The jsp:include tag can be used to include static as well as dynamic pages.

Advantage of jsp:include action tag

- **Code reusability** : We can use a page many times such as including header and footer pages in all pages. So it saves a lot of time.

Java Bean

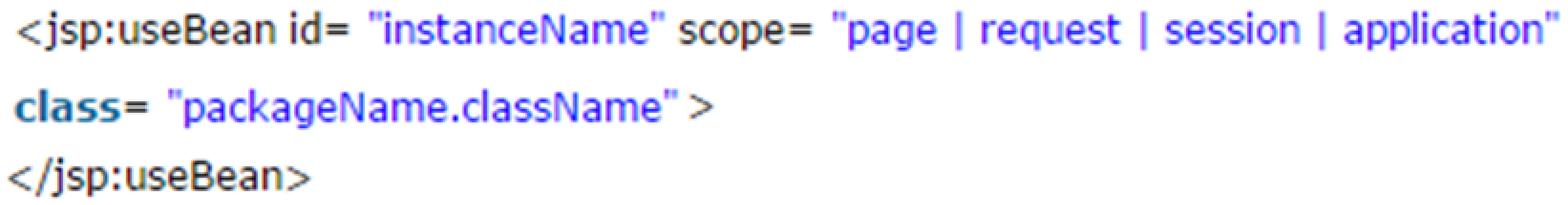
- A Java Bean is a java class that should follow following conventions:
  - It should have a no-arg constructor.
  - It should be Serializable.
  - It should provide methods to set and get the values of the properties, known as getter and setter methods.

Why use Java Bean?

- According to Java white paper, it is a reusable software component. A bean encapsulates many objects into one object, so we can access this object from multiple places. Moreover, it provides the easy maintenance.

jsp:useBean action tag

- The jsp:useBean action tag is used to locate or instantiate a bean class.
- If bean object of the Bean class is already created, it doesn't create the bean depending on the scope.
- But if object of bean is not created, it instantiates the bean.



```
<jsp:useBean id= "instanceName" scope= "page | request | session | application"
class= "packageName.className" >
</jsp:useBean>
```

Attributes and Usage of jsp:useBean action tag

- **id:** is used to identify the bean in the specified scope.
- **scope:** represents the scope of the bean. It may be page, request, session or application. The default scope is page.
  - **page:** specifies that you can use this bean within the JSP page. The default scope is page.
  - **request:** specifies that you can use this bean from any JSP page that processes the same request. It has wider scope than page.
  - **session:** specifies that you can use this bean from any JSP page in the same session whether processes the same request or not. It has wider scope than request.
  - **application:** specifies that you can use this bean from any JSP page in the same application. It has wider scope than session.
- **class:** instantiates the specified bean class (i.e. creates an object of the bean class) but it must have no-arg or no constructor and must not be abstract.

jsp:setProperty and jsp:getProperty action tags

- The setProperty and getProperty action tags are used for developing web application with Java Bean.
- In web devlopment, bean class is mostly used because it is a reusable software component that represents data.
- The jsp:setProperty action tag sets a property value or values in a bean using the setter method.

Syntax of jsp:setProperty action tag

```
<jsp:setProperty name="instanceOfBean" property= "*"   |
property="propertyName" param="parameterName"  |
property="propertyName" value="{ string | <%= expression %>}"
/>
```

Example of jsp:setProperty action tag if you have to set all the values of incoming request in the bean

```
<jsp:setProperty name="bean" property="*" />
```

Example of jsp:setProperty action tag if you have to set value of the incoming specific property

```
<jsp:setProperty name="bean" property="username" />
```

Example of jsp:setProperty action tag if you have to set a specific value in the property

```
<jsp:setProperty name="bean" property="username" value="Kumar" />
```

jsp:getProperty action tag

The jsp:getProperty action tag returns the value of the property.

Syntax of jsp:getProperty action tag

```
<jsp:getProperty name="instanceOfBean" property="propertyName" />
```

Simple example of jsp:getProperty action tag

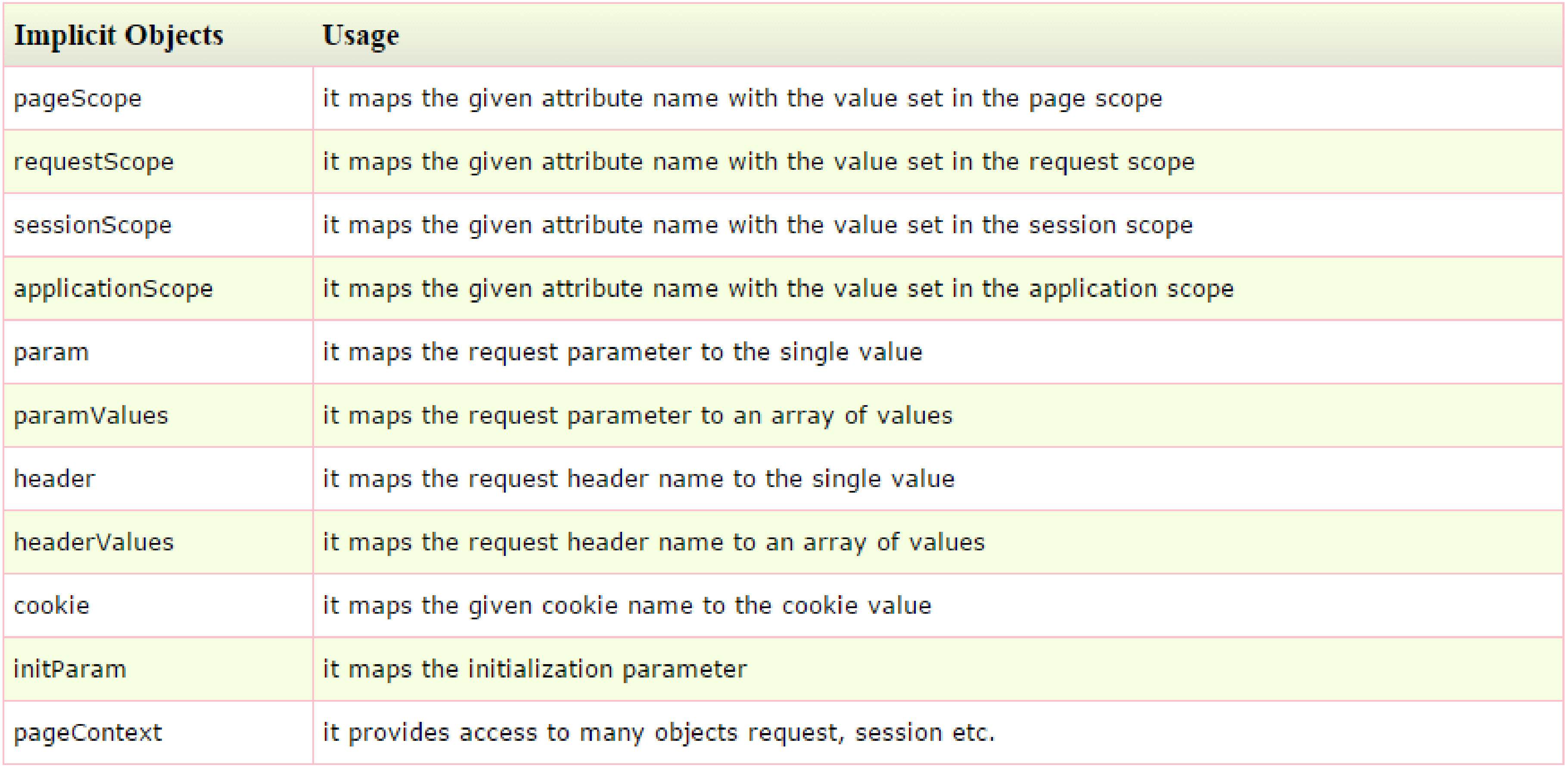
```
<jsp:getProperty name="obj" property="name" />
```

Expression Language (EL) in JSP

- The **Expression Language** (EL) simplifies the accessibility of data stored in the Java Bean component, and other objects like request, session, application etc.
- There are many implicit objects, operators and reserve words in EL.

Syntax for Expression Language (EL)

```
${ expression }
```



Implicit Objects	Usage
pageScope	it maps the given attribute name with the value set in the page scope
requestScope	it maps the given attribute name with the value set in the request scope
sessionScope	it maps the given attribute name with the value set in the session scope
applicationScope	it maps the given attribute name with the value set in the application scope
param	it maps the request parameter to the single value
paramValues	it maps the request parameter to an array of values
header	it maps the request header name to the single value
headerValues	it maps the request header name to an array of values
cookie	it maps the given cookie name to the cookie value
initParam	it maps the initialization parameter
pageContext	it provides access to many objects request, session etc.

Example

index.jsp

```
<form action="process.jsp">
Enter Name:<input type="text" name="name" /><br/><br/>
<input type="submit" value="go"/>
</form>
```

process.jsp

```
Welcome, ${ param.name }
```

JSTL (JSP Standard Tag Library)

- The JSP Standard Tag Library (JSTL) represents a set of tags to simplify the JSP development.

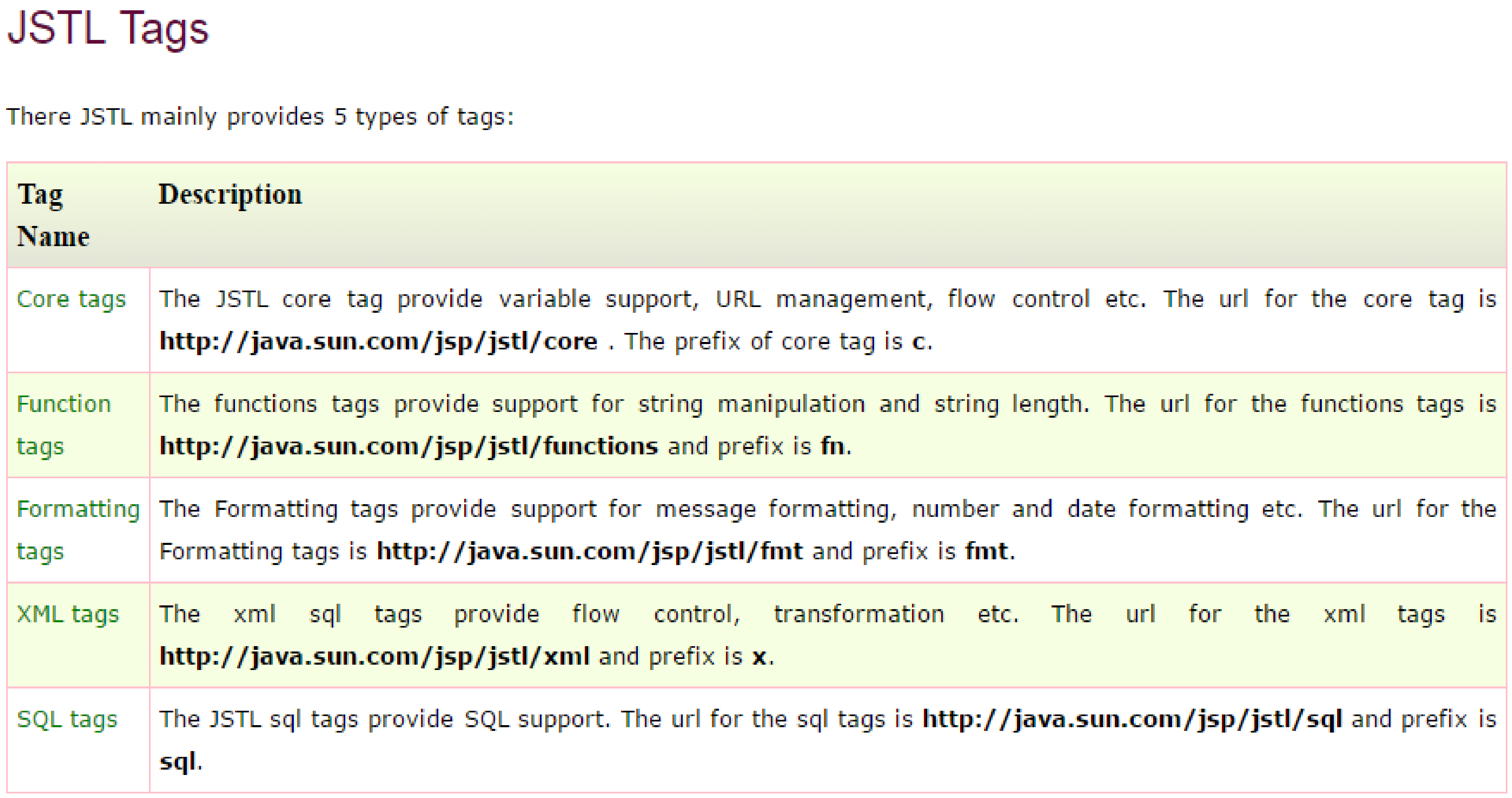
Advantage of JSTL

Fast Developement JSTL provides many tags that simplifies the JSP.

Code Reusability We can use the JSTL tags in various pages.

No need to use scriptlet tag It avoids the use of scriptlet tag.

JSTL Tags



JSTL Tags

There JSTL mainly provides 5 types of tags:

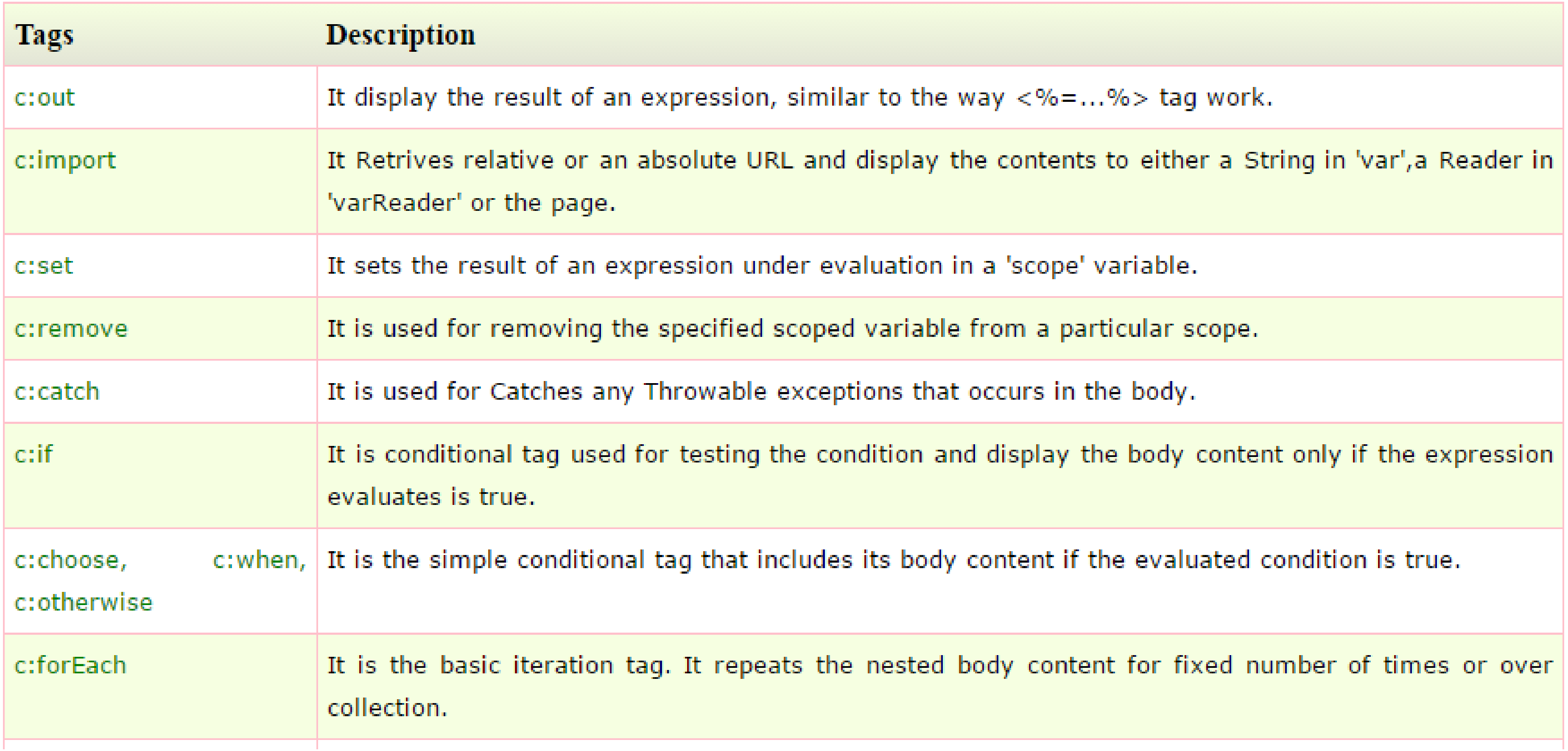
Tag Name	Description
Core tags	The JSTL core tag provide variable support, URL management, flow control etc. The url for the core tag is **http://java.sun.com/jsp/jstl/core** . The prefix of core tag is **c**.
Function tags	The functions tags provide support for string manipulation and string length. The url for the functions tags is **http://java.sun.com/jsp/jstl/functions** and prefix is **fn**.
Formatting tags	The Formatting tags provide support for message formatting, number and date formatting etc. The url for the Formatting tags is **http://java.sun.com/jsp/jstl/fmt** and prefix is **fmt**.
XML tags	The xml sql tags provide flow control, transformation etc. The url for the xml tags is **http://java.sun.com/jsp/jstl/xml** and prefix is **x**.
SQL tags	The JSTL sql tags provide SQL support. The url for the sql tags is **http://java.sun.com/jsp/jstl/sql** and prefix is **sql**.

JSTL Core Tags

- The JSTL core tag provides variable support, URL management, flow control etc.
- The syntax used for including JSTL core library in your JSP is:

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
```

JSTL Core Tags List



Tags	Description
c:out	It display the result of an expression, similar to the way <%=...%> tag work.
c:import	It Retrives relative or an absolute URL and display the contents to either a String in 'var',a Reader in 'varReader' or the page.
c:set	It sets the result of an expression under evaluation in a 'scope' variable.
c:remove	It is used for removing the specified scoped variable from a particular scope.
c:catch	It is used for Catches any Throwable exceptions that occurs in the body.
c:if	It is conditional tag used for testing the condition and display the body content only if the expression evaluates is true.
c:choose, c:when, c:otherwise	It is the simple conditional tag that includes its body content if the evaluated condition is true.
c:forEach	It is the basic iteration tag. It repeats the nested body content for fixed number of times or over collection.

JSTL Core <c:out> Tag

- The <c:out> tag is similar to JSP expression tag, but it can only be used with expression. It will display the result of an expression, similar to the way < %=...% > work.

Let's see the simple example of c:out tag:

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core"  prefix="c" %>
<html>
<head>
<title>Tag Example</title>
</head>
<body>
<c:out value="${'Welcome to JSTL'}"/>
</body>
</html>
```

Output: **Welcome to JSTL**

JSTL Core <c:import> Tag

- The <c:import> is similar to jsp 'include', with an additional feature of including the content of any resource either within server or outside the server.
- This tag provides all the functionality of the <include > action and it also allows the inclusion of absolute URLs.
- Using an import tag the content from a different FTP server and website can be accessed.

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<html>
<head>
<title>Tag Example</title>
</head>
<body>
<c:import var="data" url="http://www.javatpoint.com"/>
<c:out value="${data}"/>
</body>  </html>
```

Above example would fetch the complete content from javatpoint.com and would store in a variable "data" which will printed eventually.

JSTL Core <c:set> Tag

- **It is used to set the result of an expression evaluated in a 'scope'. The <c:set> tag is helpful because it evaluates the expression and use the result to set a value of java.util.Map or JavaBean.**
- This tag is similar to jsp:setProperty action tag.
- <%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>

```
<html>
<head>
<title>Core Tag Example</title>
</head>
<body>
<c:set var="Income" scope="session" value="${4000*4}"/>
<c:out value="${Income}"/>
</body>
</html>
```

Output:

```
16000
```

JSTL Core <c:remove> Tag

- It is used for removing the specified variable from a particular scope. This action is not particularly helpful, but it can be used for ensuring that a JSP can also clean up any scope resources.
- The <c:remove > tag removes the variable from either a first scope or a specified scope.

example of c:remove tag:

- <%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>

```
<html>
<head>
<title>Core Tag Example</title>
</head>
<body>
<c:set var="income" scope="session" value="${4000*4}"/>
<p>Before Remove Value is: <c:out value="${income}"/></p>
<c:remove var="income"/>
<p>After Remove Value is: <c:out value="${income}"/></p>
</body>
</html>
```

Output:

```
Before Remove Value is: 16000
After Remove Value is:
```

JSTL Core <c:catch> Tag

- It is used for Catches any Throwable exceptions that occurs in the body and optionally exposes it.
- In general it is used for error handling and to deal more easily with the problem occur in program.
- The < c:catch > tag catches any exceptions that occurs in a program body.

example of c:catch tag:

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core"
    prefix="c" %>
<html>
<head>
<title>Core Tag Example</title>
</head>
<body>
 <c:catch var ="catchtheException">
  <% int x = 2/0;%>
</c:catch>
<c:if test = "${catchtheException != null}">
  <p>The type of exception is : ${catchtheException} <br />
  There is an exception: ${catchtheException.message}</p>
</c:if>
 </body>
</html>
```

Output:

```
The type of exception is : java.lang.ArithmaticException: / by zero
There is an exception: / by zero
```

JSTL Core <c:if> Tag

The < c:if > tag is used for testing the condition and it display the body content, if the expression evaluated is true.

It is a simple conditional tag which is used for evaluating the body content, if the supplied condition is true.

example of c:if tag:

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core"  prefix="c"  %>
<html>
<head>
<title>Core Tag Example</title>
</head>
<body>
<c:set var="income" scope="session" value="${4000*4}"/>
<c:if test="${income > 8000}">
  <p>My income is: <c:out value="${income}"/><p>
</c:if>
</body>  </html>
```

Output:My income is: 16000

JSTL Core <c:choose>, <c:when>, <c:otherwise> Tag

- The < c:choose > tag is a conditional tag that establish a context for mutually exclusive conditional operations.
- It works like a Java **switch** statement in which we choose between a numbers of alternatives.
- The <c:when > is subtag of <choose > that will include its body if the condition evaluated be 'true'.
- The < c:otherwise > is also subtag of < choose > it follows &l;twhen > tags and runs only if all the prior condition evaluated is 'false'.
- The c:when and c:otherwise works like if-else statement. But it must be placed inside c:choose tag.

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<html>
<head>
<title>Core Tag Example</title>
</head>
<body>
<c:set var="income" scope="session" value="${4000*4}"/>
<p>Your income is : <c:out value="${income}"/></p>
<c:choose>
   <c:when test="${income <= 1000}">
      Income is not good.
   </c:when>
<c:when test="${income > 10000}">
      Income is very good.
   </c:when>
   <c:otherwise>
   Income is undetermined...
   </c:otherwise>
</c:choose>
</body>  </html>
```

This will produce the following result:

```
Your income is : 16000
Income is very good.
```

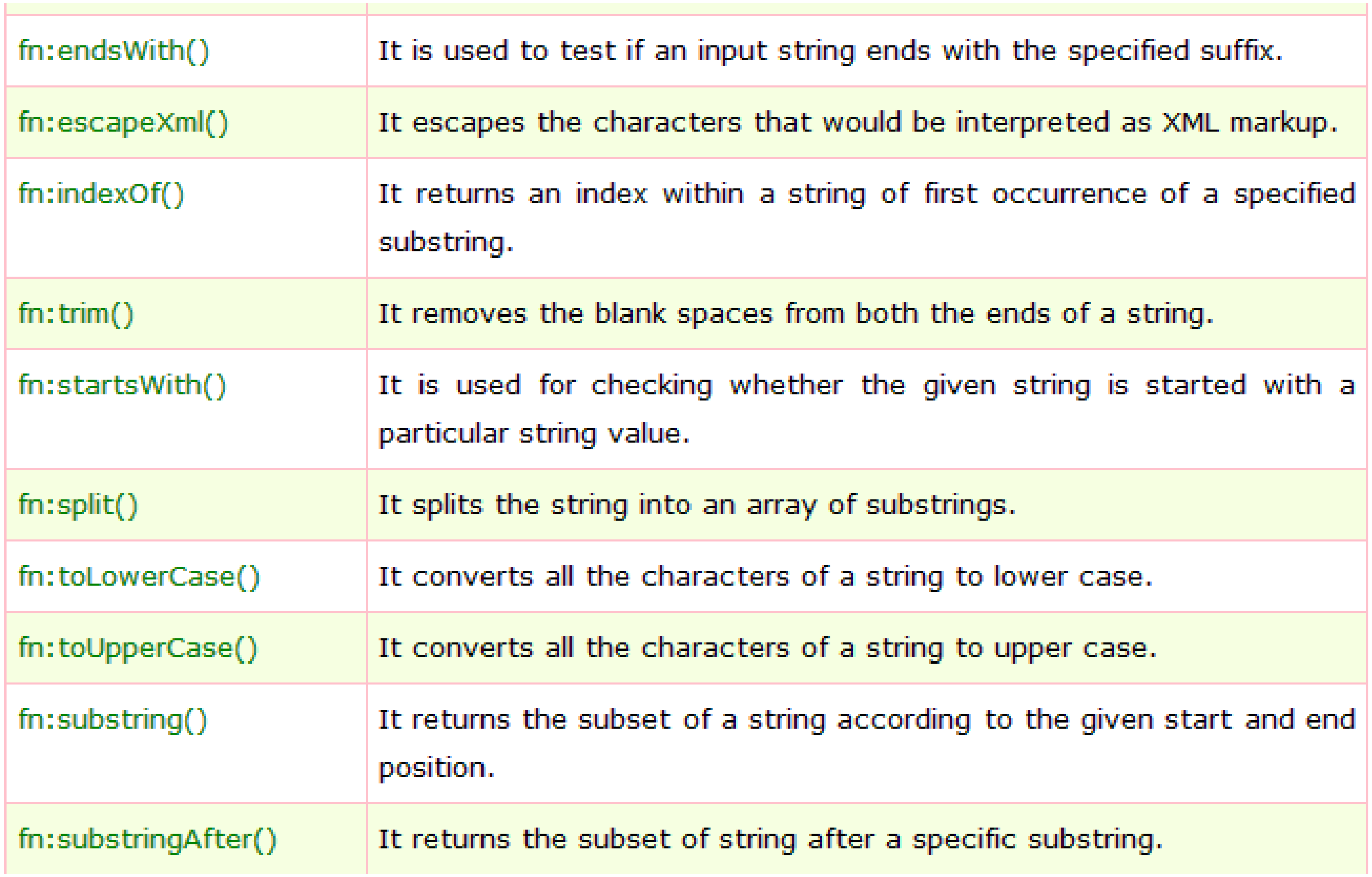
JSTL Core <c:forEach> Tag

- The <c:for each > is an iteration tag used for repeating the nested body content for fixed number of times or over the collection.
- These tag used as a good alternative for embedding a Java **while, do-while, or for** loop via a scriptlet.
- The < c:for each > tag is most commonly used tag because it iterates over a collection of object.

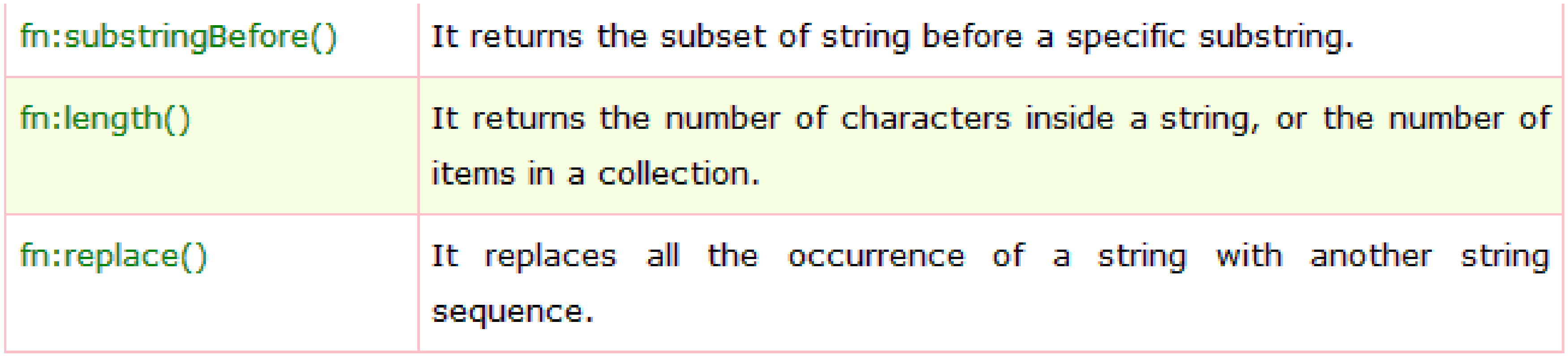
```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core"  prefix="c" %>
<html>
<head>
<title>Core Tag Example</title>
</head>
<body>
<c:forEach var="j" begin="1" end="3">
  Item <c:out value="${j}"/><p>
</c:forEach>
</body>
</html>
```

```
Output:
Item 1
Item 2
Item 3
```

Function Tags



fn:endsWith()	It is used to test if an input string ends with the specified suffix.
fn:escapeXml()	It escapes the characters that would be interpreted as XML markup.
fn:indexOf()	It returns an index within a string of first occurrence of a specified substring.
fn:trim()	It removes the blank spaces from both the ends of a string.
fn:startsWith()	It is used for checking whether the given string is started with a particular string value.
fn:split()	It splits the string into an array of substrings.
fn:toLowerCase()	It converts all the characters of a string to lower case.
fn:toUpperCase()	It converts all the characters of a string to upper case.
fn:substring()	It returns the subset of a string according to the given start and end position.
fn:substringAfter()	It returns the subset of string after a specific substring.



fn:substringBefore()	It returns the subset of string before a specific substring.
fn:length()	It returns the number of characters inside a string, or the number of items in a collection.
fn:replace()	It replaces all the occurrence of a string with another string sequence.

Thank You