JSP

Table of Contents

[JSP Lifecycle 2](#_Toc100559508)

[JSP API 3](#_Toc100559509)

[Implicit Objects 4](#_Toc100559510)

[Syntactic Elements 4](#_Toc100559511)

A **Java Server Page** (JSP) is essentially an HTML page with some Java code in it. HTML itself only allows **static content**, but the ability to include Java code allows JSP pages to have **dynamic content**.

We can technically generate dynamic content using just servlets, but it should quickly become obvious that generating complicated HTML using just the out.println() method is both frustrating and limited in functionality. Instead, we should use **template engines**. JSP is one such template engine.

Servlets and JSPs are similar. In fact, JSPs are converted to servlets before they are deployed. Servlets work on the server-side, communicating with model classes, while JSPs work on the client-side, with the **presentation layer**.

## JSP Lifecycle

The JSP lifecycle consist of 7 stages:

1. **Translation** – The web container parses the JSP page and converts it to a servlet source code. home.jsp becomes home\_jsp.java.
2. **Compilation** – The web container compiles the servlet Java file into a class file.
3. **Class Loading** – The class file gets loaded into memory.
4. **Instance Creation** – An object for the class is instantiated by the web container.
5. **Initialization** – The JSP class is initialized and is transformed into a servlet class. This makes the *ServletConfig* and *ServletContext* objects accessible.
6. **Request Processing** – For every client request, a **new thread** is spawned with the corresponding *ServletRequest* and *ServletResponse* objects to process the request.
7. **Destruction** – The JSP is destroyed.

## JSP API

JSP mainly works with two interfaces, *JspPage* and *HttpJspPage*.

The *JspPage* interface has two methods, jspInit(), which is invoked during the **initialization** of the JSP, and jspDestroy(), which is invoked just as the JSP is about to be **destroyed**, giving us the opportunity to perform any clean-up required.

The *HttpJspPage* interface has a single method, \_jspService(), which is invoked every time a request is made to the container for the JSP. This is where the request is processed. This method is similar to the service() method from servlets, except that it is highly recommended that we do not override this method. This is indicated using the *\_* at the start of the method name.

There are actually several other classes which can be used in JSPs. All the methods are under the javax.servlet.jsp package.

### Implicit Objects

There are a total of **9 implicit objects**. These are objects which can be used directly in the JSP page, without initializing them from a class.

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

### Syntactic Elements

There are a total of **6 syntactic elements**. There use will become self-evident soon.

* Scriplets
* Expressions
* Directives
* Declarations
* Actions
* Comments

**Scriplets** are blocks of code inside a JSP page which are executed during the request processing time. They are written inside <% %> tags. Any code written here goes into the \_jspService() method.

<html>  
<body>  
<%  
 // this is a scriptlet inside which we can write Java code  
 out.println(request.getParameter("employee"));  
 int z = 10;  
 out.println(z);  
%>  
</body>  
</html>

JSP

**Expressions** are shorthand notations that evaluate a Java expression as a string. This is somewhat of a shortcut for the out.println() method. JSP expressions are contained in the <%= %> tags. The content inside these tags are sent to the out.println() method. Any code written here goes into the \_jspService() method.

<html>  
<body>  
<%  
 String username = "Adam";  
%>  
<h1>Hello <%=username%>!</h1>  
</body>  
</html>

JSP

**Declarations** are used to define class-wide variables and methods. They are initialized when the JSP page is initialized and they are only **initialized once**. Declarations are also **persistent across users**. If we have some variable here and we update the variable from one client, the update will be reflected in other clients as well. The code written here, unlike the previous two elements, does not go into the \_jspService() method. Declarations are contained inside <%! %> tags.

<html>  
<body>  
<%!  
 // this is a declaration  
 String name = "Adam";  
 String returnName(){  
 return name;  
 }  
%>  
<h1>Hello <%=returnName()%>!</h1>  
</body>  
</html>

JSP

**Directives** are used to give special instructions to the container when it is converting the JSP page to servlet source code. They are contained inside <%@ %> tags.

There are mainly three types of directives:

1. **Page Directives** – These define attributes that apply to the entire JSP page.
2. **Include Directives** – These are used to include other static resources during compilation.
3. **Taglib Directives** – These are used to include custom tag libraries. We will study what these are later on.

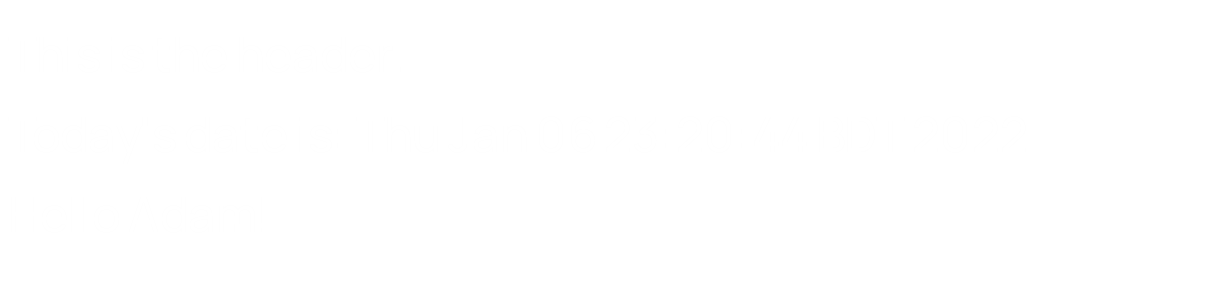
**Page Directives** allow several attributes, such as import, contentType, extends, info, buffer, language, isELIgnored, isThreadSafe, autoFlush, session, pageEncoding, errorPage, isErrorPage. We will not end up using most of these.

**Include Directives** only have a single attribute, file, which takes the value of the filename to be included in the JSP page.

<html>  
<body>  
<%@include file="header.html"%>  
<%@page import="java.util.Date"%>  
<%!  
 String name = "Adam";  
%>  
<h1>Today's date is: <%=new Date()%></h1>  
<h1>Hello <%=name%>!</h1>  
</body>  
</html>

JSP

In the above code, we have included the content of a header file first. This content will be printed where it is. Afterwards, we have used a Java library to print the current date. The output for this code looks like this:



The interesting thing about directives is that the client has no way of knowing that all of this happened. If they inspect the webpage, all they will see is HTML code.

**JSP Standard Actions** are tags that affect the runtime behaviour of the JSP and the response sent back to the client. They are used to include **static or dynamic resources** at the time of the client request.

|  |  |
| --- | --- |
| **JSP Action Tag** | **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 a property in a bean object |
| jsp:getProperty | Prints the value of a property in a bean object |
| jsp:plugin | Embeds another component, such as an applet |
| jsp:param | Sets the parameter value; used in forward and include operations |
| jsp:fallback | Can be used to print the message if the plugin is working; used in jsp:plugin |

<html>  
<body>  
<%@include file="header.html"%>  
<jsp:include page="footer.jsp"/>  
</body>  
</html>

JSP

The jsp:include tag, which is used to include another resource, is very similar to the @include directive. However, there is one important difference. The @include directive includes the specified resource during **translation time**. This means changing something here will require the JSP file to be recompiled. Thus, the @include directive is more suitable for **static content**. The jsp:include tag on the other hand includes the specified resource during **request time**. This means changing something will require a new request to be made before the change becomes visible. Thus, the jsp:include tag is more suitable for **dynamic content**.

JSP supports **comments** of two types, HTML comments, such as <!--Comment-->, as well as JSP comments, such as *<%--Comment--%>*. **JSP comments** should be used for documentation purposes. This is because when the JPS file is converted to an HTML file, the JSP comments are not kept. **HTML comments** should only be used if we specifically want the comment to appear in the resulting HTML document.