**MODULE-V**

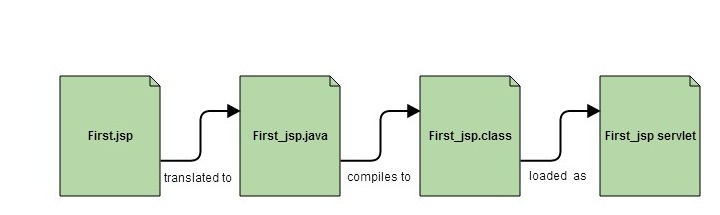
### Introduction to JSP

**JSP** technology is used **to create dynamic web applications**. **JSP** pages are **easier t o maintain than a Servlet. JSP pages are opposite of Servlets** as a servlet adds HTML code inside Java code, while **JSP adds Java code inside HTML using JSP tags**. **Everything a Servlet can do, a JSP page can also do it.**

JSP enables us to write HTML pages containing tags, inside which we can include powerful Java programs. **Using JSP, one can easily separate Presentation and Business logic** as a web designer can design and update JSP pages creating the presentation layer and java developer can write server side complex computational code without concerning the web design. And both the layers can easily interact over HTTP requests.

#### Execution of JSP

**JSP** pages are converted into **Servlet** by **the Web Containe**r. The Container translates a JSP page into servlet **class source(.java)** file and then compiles into a Java Servlet class.



#### Why JSP is preffered over Servlets?

* JSP provides **an easier way to code dynamic web pages**.
* JSP **does not require additional files** like, java class files, web.xml etc
* **Any change in the JSP** code is handled by Web Container(Application server like tomcat), and doesn't require re-compilation.
* **JSP pages can be directly accessed**, and web.xml mapping is not required like in servlets.

#### Advantage of JSP

* Easy to maintain and code.
* High Performance and Scalability.
* JSP is built on Java technology, so it is platform independent.

**Example:**

<%-- JSP comment --%>

<HTML>

<HEAD>

<TITLE>MESSAGE</TITLE>

</HEAD>

<BODY>

<%out.print("Hello, Sample JSP code");%>

</BODY>

</HTML>

**Servlet Advantages**

1. Servlets provide a way to generate dynamic documents that is both easier to write and faster to run.
2. provide all the powerfull features of JAVA, such as Exception handling and garbage collection.
3. Servlet enables easy portability across Web Servers.
4. Servlet can communicate with different servlet and servers.
5. Since all web applications are stateless protocol, servlet uses its own API to maintain  session.

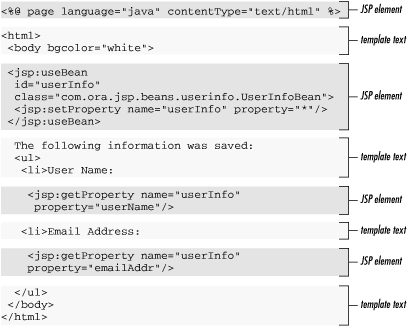
**Servlets Disadvantage**

1. Designing in servlet is difficult and slows down the application.
2. Writing complex business logic makes the application difficult to understand.
3. We need a Java Runtime Environment on the server to run servlets. CGI is a completely language independent protocol, so you can write CGIs in whatever languages you have available (including Java if you want to).

# The Anatomy of a JSP Page

A JSP page is simply a regular web page with JSP **elements for generating the parts of the page that differ for each request**, as shown in Figur(1)

Everything in the page that is not a JSP element is **called template text . Template text can really be any text: HTML, XML, or even plain text**. Template text is always passed straight through to the browser.



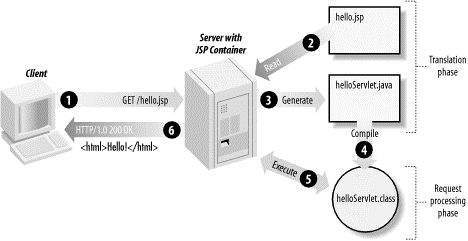
*Figure (1) Template text and JSP elements*

When a JSP page request is processed, **the template text and the dynamic content generated by the JSP elements are merged**, and **the result is sent as the response to the browse**r.

## JSP Processing

The following steps explain how the web server creates the Webpage using JSP −

* As with a normal page, your browser sends an HTTP request to the web server.
* 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**.m` n
* 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( ) statement**s and **all JSP elements are converted to Java code**. This code implements the corresponding dynamic behavior of the page.
* The **JSP engine compiles the servlet into an executable class** and **forwards the original request to a servlet engine**.
* 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. **The output is furthur passed on to the web server by the servlet engine inside an HTTP response.**
* The web server forwards the HTTP response to your browser in terms of static HTML content.
* Finally, the web browser handles the dynamically-generated HTML page inside the HTTP response exactly as if it were a static page.

All the above mentioned steps can be seen in the following diagram 

## JSP Components(or) Elements of JSP

JSPs are comprised of standard HTML tags and JSP tags. The structure of JavaServer pages are simple and easily handled by the servlet engine. In addition to HTML,we can categorize JSPs as following -

* Directives
* Declarations
* Scriptlets
* Comments
* Expressions

**Directives :**

A directives tag always appears at the top of your JSP file.

It is **global definition** sent to the JSP engine. Directives **contain special processing instruction**s for the web container.

we can import packages, define error handling pages or the session information of the JSP page.

Directives are defined by using **“<%@ and %>”** tags.

**Syntax -**

<%@ directive attribute="value" %>

**Example**:

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

or

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

**Declarations :**

This tag is used for **defining the functions and variables to be used in the JSP**.

This element of JSPs **contains the java variables and metho**ds which we can call **in expression block of JSP page**.

Declarations are defined by using **<%! and %>** tags. Whatever you declare within these tags will be visible to the rest of the page.

**Syntax -**

<%! declaration(s) %>

Example:

<html>

<head>

<title>My First JSP Page</title>

</head>

<%!

int count = 0;

%>

<body>

Page Count is:

<% out.println(++count); %>

</body>

</html>

**Scriptlets:**

In this tag **we can insert any amount of valid java code** and these codes are placed in \_**jspService method** by **the JSP engine**.

**Sriptlets can be used anywhere** in the page. Scriptlets are defined by using tags.

**Syntax -“<% and %>”**

<% Scriptlets%>

Example:

<html>

<head>

<title>My First JSP Page</title>

</head>

<%

int count = 0;

%>

<body>

Page Count is <% out.println(++count); %>

</body>

</html>

**Comments :**

* Comments help in understanding what is actually code doing.
* JSPs provides two types of comments for putting comment in your page.

(1)First type of comment is **for output comment which is appeared in the output stream on the browser**. It is written by using the

**“<!-- and -->”** tags.

**Syntax -**

<!-- comment text -->

(2)Second type of comment **is not delivered to the browser**. It is

written by using the **“<%-- and --%>”** tags.

**Syntax -**

<%-- comment text --%>

Example:

<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>

**Expressions :**

Expressions in JSPs **is used to output any data on the generated page**.

These data are **automatically converted to string** and **printed on the output stream.**

It is **an instruction to the web container for executing the code with in the expression and replace it with the resultant output conte**nt. For writing expression in JSP, we can use **“<%= and %>” tags.**

**Syntax -**

<%= expression %>

Example:

When the Container sees this

<%= (2\*5) %>

It turns it into this:

out.print((2\*5));

# JSP - Standard Tag Library (JSTL)

The Java Server Pages Standard Tag Library (JSTL) is a collection of useful JSP tags which encapsulates the core functionality common to many JSP applications.

JSTL has support for common, structural tasks such as iteration and conditionals, tags for manipulating XML documents, internationalization tags, and SQL tags. It also provides a framework for integrating the existing custom tags with the JSTL tags.

## Install JSTL Library

To begin working with JSP tages you need to first install the JSTL library. If you are using the Apache Tomcat container, then follow these two steps −

**Step 1** − Download the “jstl.jar” file.

**Step 2** − Simply copy the JAR files in the distribution's 'lib' directory to our application's **Tomcat7.0\lib** directory.

To use any of the libraries, we must include a <taglib> directive at the top of each JSP that uses the library.

## Classification of The JSTL Tags

The JSTL tags can be classified, according to their functions, into the following JSTL tag library groups that can be used when creating a JSP page −

* **Core Tags**
* **Formatting tags**
* **SQL tags**
* **XML tags**
* **JSTL Functions**

## Core Tags

The JSTL core tag provides variable support, URL management, flow control etc.

Following is the syntax to include the JSTL Core library in our JSP −

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

Following table lists out the core JSTL Tags −

|  |  |
| --- | --- |
| **S.No.** | **Tag & Description** |
| 1 | [**<c:out>**](https://www.tutorialspoint.com/jsp/jstl_core_out_tag.htm)  Diaplays the output.  Like <%= ... >, but for expressions. |
| 2 | [**<c:set >**](https://www.tutorialspoint.com/jsp/jstl_core_set_tag.htm)  Sets the result of an expression evaluation in a **'scope'** |
| 3 | [**<c:remove >**](https://www.tutorialspoint.com/jsp/jstl_core_remove_tag.htm)  Removes a **scoped variable** (from a particular scope, if specified). |
| 4 | [**<c:catch>**](https://www.tutorialspoint.com/jsp/jstl_core_catch_tag.htm)  Catches any **Throwable** that occurs in its body and optionally exposes it. |
| 5 | [**<c:if>**](https://www.tutorialspoint.com/jsp/jstl_core_if_tag.htm)  Simple conditional tag which evalutes its body if the supplied condition is true. |
| 6 | [**<c:choose>**](https://www.tutorialspoint.com/jsp/jstl_core_choose_tag.htm)  Simple conditional tag that establishes a context for mutually exclusive conditional operations, marked by **<when>** and **<otherwise>**. |
| 7 | [**<c:when>**](https://www.tutorialspoint.com/jsp/jstl_core_choose_tag.htm)  Subtag of **<choose>** that includes its body if its condition evalutes to **'true'**. |
| 8 | [**<c:otherwise >**](https://www.tutorialspoint.com/jsp/jstl_core_choose_tag.htm)  Subtag of **<choose>** that follows the **<when>** tags and runs only if all of the prior conditions evaluated to **'false'**. |
| 9 | [**<c:import>**](https://www.tutorialspoint.com/jsp/jstl_core_import_tag.htm)  Retrieves an absolute or relative URL and exposes its contents to either the page, a String in **'var'**, or a Reader in **'varReader'**. |
| 10 | [**<c:forEach >**](https://www.tutorialspoint.com/jsp/jstl_core_foreach_tag.htm)  The basic iteration tag, accepting many different collection types and supporting subsetting and other functionality . |
| 11 | [**<c:forTokens>**](https://www.tutorialspoint.com/jsp/jstl_core_foreach_tag.htm)  Iterates over tokens, separated by the supplied delimeters. |
| 12 | [**<c:param>**](https://www.tutorialspoint.com/jsp/jstl_core_param_tag.htm)  Adds a parameter to a containing **'import'** tag's URL. |
| 13 | [**<c:redirect >**](https://www.tutorialspoint.com/jsp/jstl_core_redirect_tag.htm)  Redirects to a new URL. |
| 14 | [**<c:url>**](https://www.tutorialspoint.com/jsp/jstl_core_url_tag.htm)  Creates a URL with optional query parameters |

## Formatting Tags

The JSTL formatting tags are used to format and display text, the date, the time, and numbers for internationalized Websites. Following is the syntax to include Formatting library in your JSP −

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

Following table lists out the Formatting JSTL Tags −

|  |  |
| --- | --- |
| **S.No.** | **Tag & Description** |
| 1 | [**<fmt:formatNumber>**](https://www.tutorialspoint.com/jsp/jstl_format_formatnumber_tag.htm)  To render numerical value with specific precision or format. |
| 2 | [**<fmt:parseNumber>**](https://www.tutorialspoint.com/jsp/jstl_format_parsenumber_tag.htm)  Parses the string representation of a number, currency, or percentage. |
| 3 | [**<fmt:formatDate>**](https://www.tutorialspoint.com/jsp/jstl_format_formatdate_tag.htm)  Formats a date and/or time using the supplied styles and pattern. |
| 4 | [**<fmt:parseDate>**](https://www.tutorialspoint.com/jsp/jstl_format_parsedate_tag.htm)  Parses the string representation of a date and/or time |
| 5 | [**<fmt:bundle>**](https://www.tutorialspoint.com/jsp/jstl_format_bundle_tag.htm)  Loads a resource bundle to be used by its tag body. |
| 6 | [**<fmt:setLocale>**](https://www.tutorialspoint.com/jsp/jstl_format_setlocale_tag.htm)  Stores the given locale in the locale configuration variable. |
| 7 | [**<fmt:setBundle>**](https://www.tutorialspoint.com/jsp/jstl_format_setbundle_tag.htm)  Loads a resource bundle and stores it in the named scoped variable or the bundle configuration variable. |
| 8 | [**<fmt:timeZone>**](https://www.tutorialspoint.com/jsp/jstl_format_timezone_tag.htm)  Specifies the time zone for any time formatting or parsing actions nested in its body. |
| 9 | [**<fmt:setTimeZone>**](https://www.tutorialspoint.com/jsp/jstl_format_settimezone_tag.htm)  Stores the given time zone in the time zone configuration variable |
| 10 | [**<fmt:message>**](https://www.tutorialspoint.com/jsp/jstl_format_message_tag.htm)  Displays an internationalized message. |
| 11 | [**<fmt:requestEncoding>**](https://www.tutorialspoint.com/jsp/jstl_format_requestencoding_tag.htm)  Sets the request character encoding |

## SQL Tags

The JSTL SQL tag library provides tags for interacting with relational databases (RDBMSs) such as **Oracle, mySQL**, or **Microsoft SQL Server**.

Following is the syntax to include JSTL SQL library in your JSP −

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

Following table lists out the SQL JSTL Tags −

|  |  |
| --- | --- |
| **S.No.** | **Tag & Description** |
| 1 | [**<sql:setDataSource>**](https://www.tutorialspoint.com/jsp/jstl_sql_setdatasource_tag.htm)  Creates a simple DataSource suitable only for prototyping |
| 2 | [**<sql:query>**](https://www.tutorialspoint.com/jsp/jstl_sql_query_tag.htm)  Executes the SQL query defined in its body or through the sql attribute. |
| 3 | [**<sql:update>**](https://www.tutorialspoint.com/jsp/jstl_sql_update_tag.htm)  Executes the SQL update defined in its body or through the sql attribute. |
| 4 | [**<sql:param>**](https://www.tutorialspoint.com/jsp/jstl_sql_param_tag.htm)  Sets a parameter in an SQL statement to the specified value. |
| 5 | [**<sql:dateParam>**](https://www.tutorialspoint.com/jsp/jstl_sql_dateparam_tag.htm)  Sets a parameter in an SQL statement to the specified java.util.Date value. |
| 6 | [**<sql:transaction >**](https://www.tutorialspoint.com/jsp/jstl_sql_transaction_tag.htm)  Provides nested database action elements with a shared Connection, set up to execute all statements as one transaction. |

## XML tags

The JSTL XML tags provide a JSP-centric way of creating and manipulating the XML documents. Following is the syntax to include the JSTL XML library in your JSP.

The JSTL XML tag library has custom tags for interacting with the XML data. This includes parsing the XML, transforming the XML data, and the flow control based on the XPath expressions.

<%@ taglib prefix = "x"

uri = "http://java.sun.com/jsp/jstl/xml" %>

Following is the list of XML JSTL Tags −

|  |  |
| --- | --- |
| **S.No.** | **Tag & Description** |
| 1 | [**<x:out>**](https://www.tutorialspoint.com/jsp/jstl_xml_out_tag.htm)  Like <%= ... >, but for XPath expressions. |
| 2 | [**<x:parse>**](https://www.tutorialspoint.com/jsp/jstl_xml_parse_tag.htm)  Used to parse the XML data specified either via an attribute or in the tag body. |
| 3 | [**<x:set >**](https://www.tutorialspoint.com/jsp/jstl_xml_set_tag.htm)  Sets a variable to the value of an XPath expression. |
| 4 | [**<x:if >**](https://www.tutorialspoint.com/jsp/jstl_xml_if_tag.htm)  Evaluates a test XPath expression and if it is true, it processes its body. If the test condition is false, the body is ignored. |
| 5 | [**<x:forEach>**](https://www.tutorialspoint.com/jsp/jstl_xml_foreach_tag.htm)  To loop over nodes in an XML document. |
| 6 | [**<x:choose>**](https://www.tutorialspoint.com/jsp/jstl_xml_choose_tag.htm)  Simple conditional tag that establishes a context for mutually exclusive conditional operations,marked by **<when>** and **<otherwise>** tags. |
| 7 | [**<x:when >**](https://www.tutorialspoint.com/jsp/jstl_xml_choose_tag.htm)  Subtag of **<choose>** that includes its body if its expression evalutes to 'true'. |
| 8 | [**<x:otherwise >**](https://www.tutorialspoint.com/jsp/jstl_xml_choose_tag.htm)  Subtag of **<choose>** that follows the **<when>** tags and runs only if all of the prior conditions evaluates to 'false'. |
| 9 | [**<x:transform >**](https://www.tutorialspoint.com/jsp/jstl_xml_transform_tag.htm)  Applies an XSL transformation on a XML document |
| 10 | [**<x:param >**](https://www.tutorialspoint.com/jsp/jstl_xml_param_tag.htm)  Used along with the **transform** tag to set a parameter in the XSLT stylesheet |

## JSTL Functions

JSTL includes a number of standard functions, most of which are common string manipulation functions.

Following is the syntax to include JSTL Functions library in your JSP −

<%@ taglib prefix = "fn" uri = "http://java.sun.com/jsp/jstl/functions" %>

Following table lists out the various JSTL Functions −

|  |  |
| --- | --- |
| **S.No.** | **Function & Description** |
| 1 | [**fn:contains()**](https://www.tutorialspoint.com/jsp/jstl_function_contains.htm)  Tests if an input string contains the specified substring. |
| 2 | [**fn:containsIgnoreCase()**](https://www.tutorialspoint.com/jsp/jstl_function_containsignorecase.htm)  Tests if an input string contains the specified substring in a case insensitive way. |
| 3 | [**fn:endsWith()**](https://www.tutorialspoint.com/jsp/jstl_function_endswith.htm)  Tests if an input string ends with the specified suffix. |
| 4 | [**fn:escapeXml()**](https://www.tutorialspoint.com/jsp/jstl_function_escapexml.htm)  Escapes characters that can be interpreted as XML markup. |
| 5 | [**fn:indexOf()**](https://www.tutorialspoint.com/jsp/jstl_function_indexof.htm)  Returns the index withing a string of the first occurrence of a specified substring. |
| 6 | [**fn:join()**](https://www.tutorialspoint.com/jsp/jstl_function_join.htm)  Joins all elements of an array into a string. |
| 7 | [**fn:length()**](https://www.tutorialspoint.com/jsp/jstl_function_length.htm)  Returns the number of items in a collection, or the number of characters in a string. |
| 8 | [**fn:replace()**](https://www.tutorialspoint.com/jsp/jstl_function_replace.htm)  Returns a string resulting from replacing in an input string all occurrences with a given string. |
| 9 | [**fn:split()**](https://www.tutorialspoint.com/jsp/jstl_function_split.htm)  Splits a string into an array of substrings. |
| 10 | [**fn:startsWith()**](https://www.tutorialspoint.com/jsp/jstl_function_startswith.htm)  Tests if an input string starts with the specified prefix. |
| 11 | [**fn:substring()**](https://www.tutorialspoint.com/jsp/jstl_function_substring.htm)  Returns a subset of a string. |
| 12 | [**fn:substringAfter()**](https://www.tutorialspoint.com/jsp/jstl_function_substringafter.htm)  Returns a subset of a string following a specific substring. |
| 13 | [**fn:substringBefore()**](https://www.tutorialspoint.com/jsp/jstl_function_substringbefore.htm)  Returns a subset of a string before a specific substring. |
| 14 | [**fn:toLowerCase()**](https://www.tutorialspoint.com/jsp/jstl_function_tolowercase.htm)  Converts all of the characters of a string to lower case. |
| 15 | [**fn:toUpperCase()**](https://www.tutorialspoint.com/jsp/jstl_function_touppercase.htm)  Converts all of the characters of a string to upper case. |
| 16 | [**fn:trim()**](https://www.tutorialspoint.com/jsp/jstl_function_trim.htm)  Removes white spaces from both ends of a string. |

**JSP with Database Connectivity:**

Here, we will see how to connect to the database from JSP. It is same as in Servlets.

**Example:**

<%@ page import = "java.io.\*,java.util.\*,java.sql.\*"%>

<%@ page import = "javax.servlet.http.\*,javax.servlet.\*" %>

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

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

<html>

<head>

<title>SELECT Operation</title>

</head>

<body>

<sql:setDataSource var = "snapshot" driver = "com.mysql.jdbc.Driver"

url = "jdbc:mysql://localhost:3306/test"

user = "root" password = "root"/>

<sql:query dataSource="${snapshot}" var= "result"> SELECT \* from Employee;</sql:query>

<table border = "1" width = "100%">

<tr>

<th>Emp ID</th>

<th>Emp Name</th>

<th>Salary</th>

</tr>

<c:forEach var = "row" items = "${result.rows}">

<tr>

<td><c:out value = "${row.Eno}"/></td>

<td><c:out value = "${row.Ename}"/></td>

<td><c:out value = "${row.Salary}"/></td>

</tr>

</c:forEach>

</table>

</body>

</html>

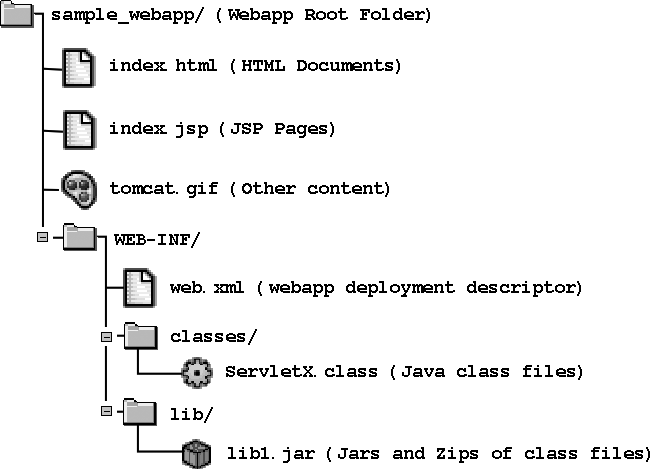
**Chapter-2**

**Introduction to Web Servers:**

**Structure of Web Application:**

The structure of a Web Application that may contain

* static content
* JSP Pages
* servlet classes
* the deployment descriptor
* tag libraries
* JAR files
* *Java class files*



**Deploying A Web Application**

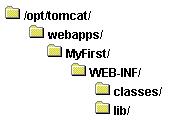
Deployment of a web application in Tomcat consists of the following tasks :

1. [Creating the directory structure](http://linux-sxs.org/internet_serving/c292.html#DIRECTORY_STRUCTURE)
2. [Creating the Context Descriptor file](http://linux-sxs.org/internet_serving/c292.html#ADD_CONTEXT)
3. [Creating a web.xml file for the web application](http://linux-sxs.org/internet_serving/c292.html#EDITING_WEB_XML)
4. [Copying the servlets, JSPs and support files to their respective directories](http://linux-sxs.org/internet_serving/c292.html#COPY_FILES)

**1. Creating the directory structure**

Before you begin deployment, you will need to create a directory structure under the $CATALINA\_HOME/webapps directory that conforms to the servlet specifications.

->You should get a directory structure similar to the one below:



## 2.Creating the Context Descriptor file

The context descriptor file, according to the Tomcat official documentation, is "used to define Tomcat specific configuration options, such as loggers, data sources, session manager configuration and more".

## 3. Creating a web.xml file for the web application

The file web.xml is sometimes known as a deployment descriptor, and it is basically the configuration file for your web application. In this file, you determine, among other things :

* the URL of the servlets in your web application
* the authentication method you wish to use
* Your filter definitions

**Example:** we will simply register HelloWorld as a servlet in the web application called MyFirst. We create a new web.xml file with the following contents inside it :

|  |
| --- |
| <?xml version="1.0" encoding="ISO-8859-1"?>  <!DOCTYPE web-app  PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"  "http://java.sun.com/dtd/web-app\_2\_3.dtd">  <web-app>  <servlet>  <servlet-name>HelloWorld</servlet-name>  <servlet-class>HelloWorld</servlet-class>  </servlet>  <servlet-mapping>  <servlet-name>HelloWorld</servlet-name>  <url-pattern>/HelloWorld</url-pattern>  </servlet-mapping>  </web-app> |

## 4. Copying the servlets, JSPs and support files to their respective directories

Now we can copy the Java class file that we compiled earlier, into the deployment directory. Because this is a servlet, it will go into /webapps/MyFirst/WEB-INF/classes directory. There is no need to copy the source file into that directory.

For more complicated deployments, such as servlets that require additional classes, such as JDBC drivers, or libraries, these additional files must be stored inside the /WEB-INF/lib subdirectory of the web application directory.

## ****Web Servers****

### What is a Web Server?

Web servers use HTTP to allow access to the Internet. They search through and use HTML files that are sent to web browsers and translated so the user can understand them.

It is also capable of accessing and storing other types of files, but they are often attached in some way to the HTML files it has, such as having images that are placed upon the HTML.

### What are Web Servers Used For?

Web servers are primarily used to store process and deliver the pages of a web site to users. In layman’s terms, this means that web servers are what make websites appear when you type in a URL.

### Types of Web Servers:

There are 4 primary web servers:

* [Apache](https://httpd.apache.org/) (provided by Apache)
* [IIS](http://www.iis.net/) (provided by Microsoft)
* [nginx](https://www.nginx.com/) (provided by NGINX, Inc. and pronounced like “Engine X”)
* and [GWS](https://en.wikipedia.org/wiki/Google_platform#Software) (provided by Google and short for Google Web Server)

**Apache**

Cost : Open source, free, no licensing fees

**Advantages:**

* There is flexibility in selecting various modules
* Enhanced security is offered
* Has strong user-community support
* Runs on UNIX, Windows, Linux, Mac OS

Disadvantages:

* It is a process based server; this means that every simultaneous connection requires a thread that can take significant overhead

**Microsoft IIS**

Cost : Comes with Windows (this means additional cost is involved for licensing)

Advantages:

* It is supported by Microsoft
* Access to .NET framework & ASP scripts is provided
* Microsoft IIS integrates with the other Microsoft services like MS SQL, ASP, Active Directory etc.

Disadvantages:

* With IIS, you cannot customize as much as open-source web servers

**GWS:**

* The Google Web Server - custom-built server software used only by Google - now runs nearly 13 per cent of all active web sites, according to the [latest survey data](http://news.netcraft.com/archives/2010/01/07/january_2010_web_server_survey.html) from the web-server-tracking UK research outfit Netcraft.
* Netcraft data has the Google Web Server (GWS) running nearly 11 million active sites - i.e., sites with recently updated content. This total includes not only sites run solely by Google, but also sites the company operates on behalf of third parties via services like Blogger, Google Docs, and Google App Engine.