**welcome-file-list tag in web.xml file of Project**

Have you ever seen <welcome-file-list> tag in your web.xml file and wondering what it is? In this text, I will explain what is this tag and why we use it.

The tag <welcome-file-list> is used for specifying the files that needs to be invoked by server by default, if you do not specify a file name while loading the project on browser.

For e.g. You have created a project named “MyServletProject” and you have few html pages and servlet classes defined in the project. However in browser you have given the url like this:

http://localhost:8080/MyServletProject

Usually we give the complete path like this:http://localhost:8080/MyServletProject/index.html. However if you have given the path like above then the webserver will look for the <welcome-file-list> tag in your project’s web.xml file. Lets say you have the following content in your web.xml file:

<web-app>

....

<welcome-file-list>

<welcome-file>myhome.htm</welcome-file>

<welcome-file>myindex.htm</welcome-file>

<welcome-file>mydefaultpage.htm</welcome-file>

</welcome-file-list>

....

</web-app>

Based on the welcome file list, server would look for the myhome.htm page if this doesn’t exist then the second welcome file myindex.html and so on till it finds a valid welcome file.

**Note**: If the <welcome-file-list> tag is not defined in web.xml or the welcome files defined in the <welcome-file> tags does not exist then the server would look for the following files in the given sequence:  
1) index.html  
2) index.htm  
3) index.jsp

I hope I covered everything related to this tag. Should you have any questions, feel free to drop a comment below.

# How to use load-on-startup tag in web.xml file with Example

By default Servlet is not loaded until servlet container receives a request for the particular servlet. This may cause a delay in accessing the servlet for the first time. To avoid the delay in access time, you can use **<load-on-startup> tag in your web.xml** file that allows you to force the servlet container to load (instantiated and have its init() called) the servlet as soon as the server starts.

## How to use <load-on-startup>?

Here is a sample web.xml file:

<web-app>

…

<servlet>

<servlet-name>MyServlet</servlet-name>

<servlet-class>com.beginnersbook.DemoServlet</servlet-class>

<load-on-startup>1</load-on-startup>

</servlet>

…

</web-app>

If I didn’t specify the <load-on-startup>, the web container would not have loaded the servlet until it receives a request for DemoServlet servlet class. Since I have specified a value >=0, this servlet (DemoServlet class) would be loaded on the startup.

value >= 0 means that the servlet is loaded when the web-app is deployed or when the server starts, if the value < 0 then servlet would be loaded whenever the container feels like.

### How to specify the order of servlet loading using <load-on-startup> tag?

<web-app>

…

<servlet>

<servlet-name>MyServlet1</servlet-name>

<servlet-class>com.beginnersbook.DemoServlet1</servlet-class>

<load-on-startup>5</load-on-startup>

</servlet>

<servlet>

<servlet-name>MyServlet2</servlet-name>

<servlet-class>com.beginnersbook.DemoServlet2</servlet-class>

<load-on-startup>0</load-on-startup>

</servlet>

<servlet>

<servlet-name>MyServlet3</servlet-name>

<servlet-class>com.beginnersbook.DemoServlet3</servlet-class>

<load-on-startup>-2</load-on-startup>

</servlet>

…

</web-app>

In this example we have three Servlets specified in the web.xml file, since servlet classes DemoServlet1 and DemoServlet2 has load-on-startup value >=0, they both will be loaded as soon as the server starts. However servlet class DemoServlet2 would be loaded before the DemoServlet1 class because it has lower load-on-startup value.

Servlet class DemoServlet3 would not be loaded on startup as it has negative load-on-startup value.

# ServletResponse Interface

The servlet container is connected to the web server that receives Http Requests from client on a certain port. When client sends a request to web server, the servlet container creates HttpServletRequest and HttpServletResponse objects and passes them as an argument to the servlet service() method.

The response object allows you to format and send the response back to the client. First we will see the commonly used methods in the ServletReponse interface and then we will see an example.

## Method of ServletResponse interface

1) String getCharacterEncoding(): It returns the name of the MIME charset used in body of the response sent to the client.  
2) String getContentType(): It returns the response content type. e.g. text, html etc.  
3) ServletOutputStream getOutputStream(): Returns a ServletOutputStream suitable for writing binary data in the response.  
4) java.io.PrintWriter getWriter(): Returns the PrintWriter object.  
5) void setCharacterEncoding(java.lang.String charset): Set the MIME charset (character encoding) of the response.  
6) void setContentLength(int len): It sets the length of the response body.  
7) void setContentType(java.lang.String type): Sets the type of the response data.  
8) void setBufferSize(int size): Sets the buffer size.  
9) int getBufferSize(): Returns the buffer size.  
10) void flushBuffer(): Forces any content in the buffer to be written to the client.  
11) boolean isCommitted(): Returns a boolean indicating if the response has been committed.  
12) void reset(): Clears the data of the buffer along with the headers and status code.  
To get the complete list of methods. Refer the [official documentation](https://docs.oracle.com/javaee/7/api/javax/servlet/ServletResponse.html).

## Example:

In the below example, we have used setContentType() and getWriter() methods of ServletResponse interface.

index.html

<form action="mydetails" method="get">

User name: <input type="text" name="uname">

<input type="submit" value="login">

</form>

MyServletDemo.java

import javax.servlet.http.\*;

import javax.servlet.\*;

import java.io.\*;

public class MyServletDemo extends HttpServlet{

public void doGet(HttpServletRequest req,HttpServletResponse res)

throws ServletException,IOException

{

res.setContentType("text/html");

PrintWriter pwriter=res.getWriter();

String name=req.getParameter("uname");

pwriter.println("User Details Page:");

pwriter.println("Hello "+name);

pwriter.close();

}

}

web.xml

<web-app>

<servlet>

<servlet-name>DemoServlet</servlet-name>

<servlet-class>MyServletDemo</servlet-class>

</servlet>

<servlet-mapping>

<servlet-name>DemoServlet</servlet-name>

<url-pattern>/mydetails</url-pattern>

</servlet-mapping>

</web-app>

# HttpSession with example in Servlet

The HttpSession object is used for session management. A session contains information specific to a particular user across the whole application. When a user enters into a website (or an online application) for the first time HttpSession is obtained via request.getSession(), the user is given a unique ID to identify his session. This unique ID can be stored into a cookie or in a request parameter.

The HttpSession stays alive until it has not been used for more than the timeout value specified in tag in deployment descriptor file( web.xml). The default timeout value is 30 minutes, this is used if you don’t specify the value in tag. This means that when the user doesn’t visit web application time specified, the session is destroyed by servlet container. The subsequent request will not be served from this session anymore, the servlet container will create a new session.

This is how you create a HttpSession object.

protected void doPost(HttpServletRequest req,

HttpServletResponse res)

throws ServletException, IOException {

HttpSession session = req.getSession();

}

You can store the user information into the session object by using setAttribute() method and later when needed this information can be fetched from the session. This is how you store info in session. Here we are storing username, emailid and userage in session with the attribute name uName, uemailId and uAge respectively.

session.setAttribute("uName", "ChaitanyaSingh");

session.setAttribute("uemailId", "myemailid@gmail.com");

session.setAttribute("uAge", "30");

This First parameter is the attribute name and second is the attribute value. For e.g. uName is the attribute name and ChaitanyaSingh is the attribute value in the code above.

TO get the value from session we use the getAttribute() method of HttpSession interface. Here we are fetching the attribute values using attribute names.

String userName = (String) session.getAttribute("uName");

String userEmailId = (String) session.getAttribute("uemailId");

String userAge = (String) session.getAttribute("uAge");

## Methods of HttpSession

**public void setAttribute(String name, Object value)**: Binds the object with a name and stores the name/value pair as an attribute of the HttpSession object. If an attribute already exists, then this method replaces the existing attributes.

**public Object getAttribute(String name)**: Returns the String object specified in the parameter, from the session object. If no object is found for the specified attribute, then the getAttribute() method returns null.

**public Enumeration getAttributeNames()**: Returns an Enumeration that contains the name of all the objects that are bound as attributes to the session object.

**public void removeAttribute(String name)**: Removes the given attribute from session.

**setMaxInactiveInterval(int interval)**: Sets the session inactivity time in seconds. This is the time in seconds that specifies how long a sessions remains active since last request received from client.

For the complete list of methods, refer the [official documentation](https://docs.oracle.com/javaee/7/api/javax/servlet/http/HttpSession.html).

## Session Example

index.html

<form action="login">

User Name:<input type="text" name="userName"/><br/>

Password:<input type="password" name="userPassword"/><br/>

<input type="submit" value="submit"/>

</form>

MyServlet1.java

import java.io.\*;

import javax.servlet.\*;

import javax.servlet.http.\*;

public class MyServlet1 extends HttpServlet {

public void doGet(HttpServletRequest request, HttpServletResponse response){

try{

response.setContentType("text/html");

PrintWriter pwriter = response.getWriter();

String name = request.getParameter("userName");

String password = request.getParameter("userPassword");

pwriter.print("Hello "+name);

pwriter.print("Your Password is: "+password);

HttpSession session=request.getSession();

session.setAttribute("uname",name);

session.setAttribute("upass",password);

pwriter.print("<a href='welcome'>view details</a>");

pwriter.close();

}catch(Exception exp){

System.out.println(exp);

}

}

}

MyServlet2.java

import java.io.\*;

import javax.servlet.\*;

import javax.servlet.http.\*;

public class MyServlet2 extends HttpServlet {

public void doGet(HttpServletRequest request, HttpServletResponse response){

try{

response.setContentType("text/html");

PrintWriter pwriter = response.getWriter();

HttpSession session=request.getSession(false);

String myName=(String)session.getAttribute("uname");

String myPass=(String)session.getAttribute("upass");

pwriter.print("Name: "+myName+" Pass: "+myPass);

pwriter.close();

}catch(Exception exp){

System.out.println(exp);

}

}

}

web.xml

<web-app>

<servlet>

<servlet-name>Servlet1</servlet-name>

<servlet-class>MyServlet1</servlet-class>

</servlet>

<servlet-mapping>

<servlet-name>Servlet1</servlet-name>

<url-pattern>/login</url-pattern>

</servlet-mapping>

<servlet>

<servlet-name>Servlet2</servlet-name>

<servlet-class>MyServlet2</servlet-class>

</servlet>

<servlet-mapping>

<servlet-name>Servlet2</servlet-name>

<url-pattern>/welcome</url-pattern>

</servlet-mapping>

</web-app>

# Cookies in Servlet with example

In the last guide, I have covered [Sessions in Servlet](https://beginnersbook.com/2013/05/http-session/). Here we will discuss Cookies which is also used for session management. Let’s recall few things here from last tutorial so that we can relate sessions and cookies. When a user visits web application first time, the servlet container crates new HttpSession object by calling request.getSession(). A unique Id is assigned to the session. The **Servlet container also sets a Cookie in the header of the HTTP response with cookie name and the unique session ID as its value.**

The cookie is stored in the user browser, the client (user’s browser) sends this cookie back to the server for all the subsequent requests until the cookie is valid. **The Servlet container checks the request header for cookies and get the session information from the cookie and use the associated session from the server memory.**

The session remains active for the time specified in tag in web.xml. If tag in not set in web.xml then the session remains active for 30 minutes. **Cookie remains active as long as the user’s browser is running**, as soon as the browser is closed, the cookie and associated session info is destroyed. So when the user opens the browser again and sends request to web server, the new session is being created.

## Types of Cookies

We can classify the cookie based on their expiry time:

1. Session
2. Persistent

**1) SessionCookies:**  
Session cookies do not have expiration time. It lives in the browser memory. As soon as the web browser is closed this cookie gets destroyed.

**2) Persistent Cookies:**  
Unlike Session cookies they have expiration time, they are stored in the user hard drive and gets destroyed based on the expiry time.

## How to send Cookies to the Client

Here are steps for sending cookie to the client:

1. Create a Cookie object.
2. Set the maximum Age.
3. Place the Cookie in HTTP response header.

#### 1) Create a Cookie object:

Cookie c = new Cookie("userName","Chaitanya");

#### 2) Set the maximum Age:

By using **setMaxAge ()** method we can set the maximum age for the particular cookie in seconds.

c.setMaxAge(1800);

#### 3) Place the Cookie in HTTP response header:

We can send the cookie to the client browser through response.addCookie() method.

response.addCookie(c);

## How to read cookies

Cookie c[]=request.getCookies();

//c.length gives the cookie count

for(int i=0;i<c.length;i++){

out.print("Name: "+c[i].getName()+" & Value: "+c[i].getValue());

}

## Example of Cookies in java servlet

**index.html**

<form action="login">

User Name:<input type="text" name="userName"/><br/>

Password:<input type="password" name="userPassword"/><br/>

<input type="submit" value="submit"/>

</form>

**MyServlet1.java**

import java.io.\*;

import javax.servlet.\*;

import javax.servlet.http.\*;

public class MyServlet1 extends HttpServlet

{

public void doGet(HttpServletRequest request,

HttpServletResponse response) {

try{

response.setContentType("text/html");

PrintWriter pwriter = response.getWriter();

String name = request.getParameter("userName");

String password = request.getParameter("userPassword");

pwriter.print("Hello "+name);

pwriter.print("Your Password is: "+password);

//Creating two cookies

Cookie c1=new Cookie("userName",name);

Cookie c2=new Cookie("userPassword",password);

//Adding the cookies to response header

response.addCookie(c1);

response.addCookie(c2);

pwriter.print("<br><a href='welcome'>View Details</a>");

pwriter.close();

}catch(Exception exp){

System.out.println(exp);

}

}

}

**MyServlet2.java**

import java.io.\*;

import javax.servlet.\*;

import javax.servlet.http.\*;

public class MyServlet2 extends HttpServlet {

public void doGet(HttpServletRequest request,

HttpServletResponse response){

try{

response.setContentType("text/html");

PrintWriter pwriter = response.getWriter();

//Reading cookies

Cookie c[]=request.getCookies();

//Displaying User name value from cookie

pwriter.print("Name: "+c[1].getValue());

//Displaying user password value from cookie

pwriter.print("Password: "+c[2].getValue());

pwriter.close();

}catch(Exception exp){

System.out.println(exp);

}

}

}

**web.xml**

<web-app>

<display-name>BeginnersBookDemo</display-name>

<welcome-file-list>

<welcome-file>index.html</welcome-file>

</welcome-file-list>

<servlet>

<servlet-name>Servlet1</servlet-name>

<servlet-class>MyServlet1</servlet-class>

</servlet>

<servlet-mapping>

<servlet-name>Servlet1</servlet-name>

<url-pattern>/login</url-pattern>

</servlet-mapping>

<servlet>

<servlet-name>Servlet2</servlet-name>

<servlet-class>MyServlet2</servlet-class>

</servlet>

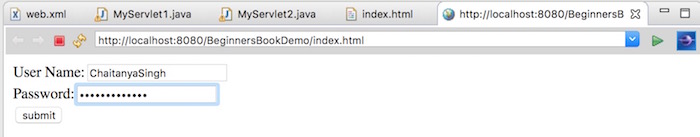
<servlet-mapping>

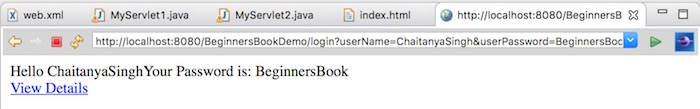
<servlet-name>Servlet2</servlet-name>

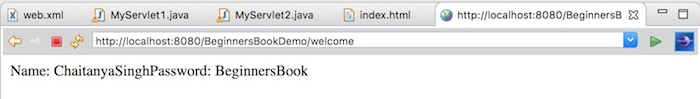
<url-pattern>/welcome</url-pattern>

</servlet-mapping>

</web-app>

**Output:**  
**Welcome Screen:**  


**After clicking Submit:**  


**After clicking View Details:**  


## Methods of Cookie class

**public void setComment(String purpose)**: This method is used for setting up comments in the cookie. This is basically used for describing the purpose of the cookie.

**public String getComment()**: Returns the comment describing the purpose of this cookie, or null if the cookie has no comment.

**public void setMaxAge(int expiry)**: Sets the maximum age of the cookie in seconds.

**public int getMaxAge()**: Gets the maximum age in seconds of this Cookie.  
By default, -1 is returned, which indicates that the cookie will persist until browser shutdown.

**public String getName()**: Returns the name of the cookie. The name cannot be changed after creation.

**public void setValue(String newValue)**: Assigns a new value to this Cookie.

**public String getValue()**: Gets the current value of this Cookie.