Master of Computer Applications (MCA)

OOP's Using JAVA – 21MCAC101

Unit – 3 : Servlet & JSP

Prepared by
Raghavendra R
Assistant professor
School of CS & IT

r.raghavendra@jainuniversity.ac.in +91-86601-61661 raguramesh88@gmail.com
+91-96116-77907

HttpServlet and it's Methods

HttpServlet is an abstract class given under the servlet-api present. It is present in javax.servlet.http package and has no abstract methods. It extends GenericServlet class.

When the servlet container uses HTTP protocol to send request, then it creates

HttpServletRequest and HttpServletResponse objects. HttpServletRequest binds the request information like header and request methods and HttpServletResponse binds all information of HTTP protocol.

HttpServlet does not override in it or destroy method. However, it uses service (-,-) method. ServletRequest and ServletResponse references are cast into HttpServletRequest and HttpServletResponse, respectively.

Methods in HttpServlet

The methods in httpservlet are given as follows:

1. *protected void doDelete (HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException* - Called by the server (via the service method) to allow a servlet to handle a DELETE request. The DELETE operation allows a client to remove a document or web page from the server.

2. *protected void doGet(HttpServletRequest req,HttpServletResponse resp) throws ServletException, IOException* - Called by the server (via the service method)"to allow a servlet to handle a GET request. Overriding this method to support a GETrequest also automatically supports an HTTP HEAD request.

3. *protected void doHead(HttpServletRequest req,HttpServletResponse resp) throws ServletException, IOException* - Receives an HTTP HEAD request from the protected service method and handles the request.

4. *protected void doOptions(HttpServletRequest req,HttpServletResponse resp) throws ServletException, IOException* - Called by the server (via the service method) to allow a servlet to handle OPTIONS request. The OPTIONS request determines which HTTP methods the server supports and returns an appropriate header.

5. *protected void doPost(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException* - Called by the server (via the service method) to allow a servlet to handle a POST request. The HTTP POST method allows the client to send data of

unlimited length to the Web server a single time and is useful when posting information such as credit card numbers.

6. *protected void doPut(HttpServletRequest req,HttpServletResponse resp) throws ServletException, IOException* - Called by the server (via the service method) to allow a

servlet to handle a PUT request. The PUT operation allows a client to place a file on the server and is similar to sending a file by FTP.

7. *protected void doTrace(HttpServletRequest req,HttpServletResponse resp) throws ServletException, IOException* - Called by the server (via the service method) to allow a servlet to handle a TRACE request. ATRACE returns the headers sent with the TRACE request to the client, so that they can be used in debugging. There's no need to override this method.

8. *protected long getLastModified(HttpServletRequest req)* - Returns the time the HttpServletRequest object was last modified, in milliseconds since midnight 1January 1970GMT.If the time is unknown, this method returns a negative number (the default).

9. *protected void service(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException* - Receives standard HTTP requests from the public service method and dispatches them to the doXXX methods defined in this class. This method is an HTTP· specific version of the service method. There's no need to override this method.

10. *public void service (ServletRequest req/ ServletRespon,se res) throws ServletException, IOException* - Dispatches client request to the protected service method. There's no need to override this method.

Servlets Life Cycle

• The life cycle is the process from the construction till the destruction of any object. •
A servlet also follows a certain life cycle.

• The life cycle of the servlet is managed by the servlet container.

The container performs the following steps:
1. Loads Servlet Class
2. Creates instance of Servlet
3. Calls init( ) method
4. Calls service( ) method
5. Calls destroy( ) method

Architecture Diagram

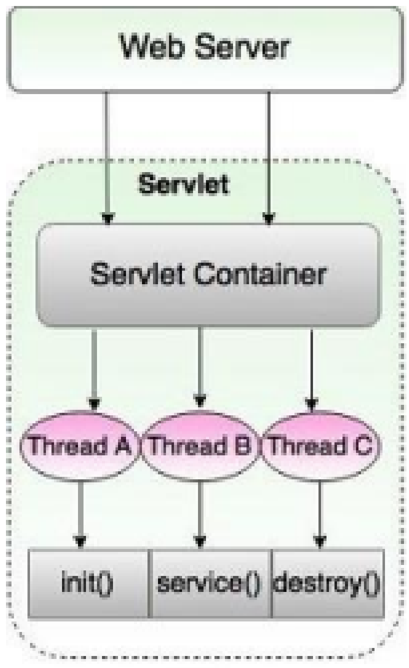


Fig: Servlet Life Cycle

1. **Loads the Servlet Class -** The **classloader** is responsible to load the Servlet class into the container. The servlet class is loaded when the first request comes from the web container.

2. **Creates instance of Servlet -** After loading of Servlet class into the container, the web container creates its instance. The instance of Servlet class is created only once in the servlet life cycle.
3. **Calls the init( ) method -** The **init( )** method performs some specific action constructor of a class. It is automatically called by Servlet container. It is called only once through the complete servlet life cycle.
   Syntax:
   public void init(ServletConfig config) throws ServletException

4. **Calls the service( ) method -** The **service( )** method performs actual task of servlet container. The web container is responsible to call the service method each time the request for servlet is received. The service( ) invokes the **doGet( ), doPost( ), doPut ( ), doDelete( )** methods based on the HTTP request.
   Syntax:
   public void service(ServletRequest request, ServletResponse response) throws ServletException, IOException

5. **Calls the destroy( ) method -** The **destroy( )** method is executed when the servlet container

remove the servlet from the container. It is called only once at the end of the life cycle of servlet.

Syntax:
public void destroy( )

ServletRequest class

True job of a Servlet is to handle client request. Servlet API provides two important interfaces **javax.servlet.ServletRequest** and **javax.servlet.http.HttpServletRequest** to encapsulate client request. Implementation of these interfaces provide important information about client request to a servlet.

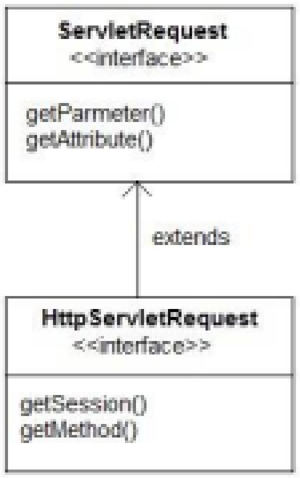
Some Important Methods of ServletRequest

Methods	**Description**
Object getAttribute(String name)	return attribute set on request object by name
Enumeration getAttributeName()	return an Enumeration containing the names of the attributes available inthis request
int getContentLength()	return size of request body
int getContentType()	return media type of request content
ServletInputStream getInputStream()	returns a input stream for reading binary data
String getParameter(String name)	returns value of parameter by name
String getLocalAddr()	returns the Internet Protocol(IP) address of the interface on which the request was received
Enumeration getParameterNames()	returns an enumeration of all parameter names
String[] getParameterValues(String name)	returns an array of String objects containing all of the values the given request parameter has, or null if the parameter does not exist
ServletContext getServletContext()	return the servlet context of current request.

String getServerName()	returns the host name of the server to which the request was sent
int getServerPort()	returns the port number to which the request was sent
boolean isSecure()	returns a boolean indicating whether this request was made using a secure channel, such as HTTPS.
void removeAttribute(String name)	removes an attribute from this request
void setAttribute(String name, Object o)	stores an attribute in this request.

HttpServletRequest interface

HttpServletRequest interface adds the methods that relates to the **HTTP**



protocol.

Some important methods of HttpServletRequest

Methods	**Description**
String getContextPath()	returns the portion of the request URI that indicates the context of the request
Cookies getCookies()	returns an array containing all of the Cookie objects the client sent with this request
String getQueryString()	returns the query string that is contained in the request URL after the path

HttpSession getSession()	returns the current HttpSession associated with this request or, if there is no current session and create is true, returns a new session
String getMethod()	Returns the name of the HTTP method with which this request was made, for example, GET, POST, or PUT.
Part getPart(String name)	gets the Part with the given name
String getPathInfo()	returns any extra path information associated with the URL the client sent when it made this request.
String getServletPath()	returns the part of this request's URL that calls the servlet

Example demonstrating Servlet Request

In this example, we will show how a parameter is passed to a Servlet in a request object from HTML page.

index.html

```
<form method="post" action="check">
 Name <input type="text" name="user" >
 <input type="submit" value="submit">
</form>
```

web.xml

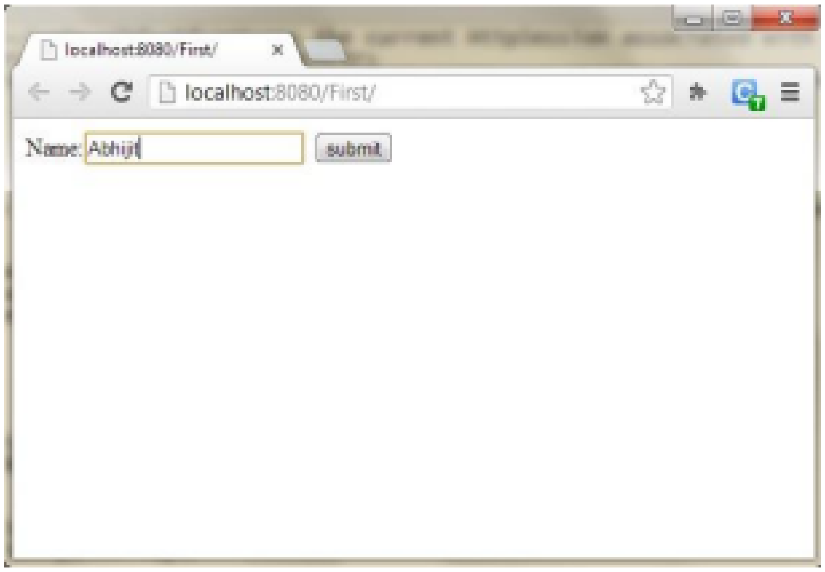
```
<servlet>
 <servlet-name>check</servlet-name>
 <servlet-class>MyServlet</servlet-class>
</servlet>
<servlet-mapping>
 <servlet-name>check</servlet-name>
 <url-pattern>/check</url-pattern>
</servlet-mapping>
```

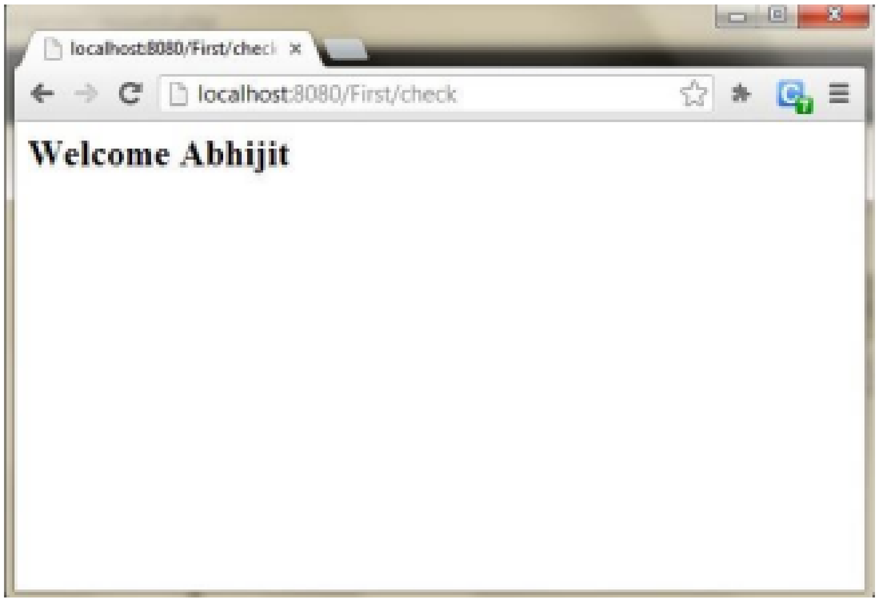
MyServlet.java

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class MyServlet extends HttpServlet {

 protected void doPost(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
```

```
response.setContentType("text/html;charset=UTF-8");
PrintWriter out = response.getWriter();
try {

String user = request.getParameter("user");
out.println("<h2> Welcome "+user+"</h2>");
} finally {
out.close();
}
}
}
```

Output :





```
Welcome Abhijit
```

ServletResponse

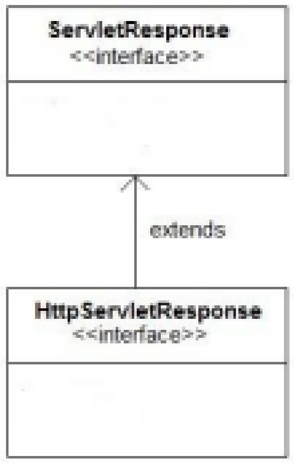
Servlet API provides two important interfaces **ServletResponse** and **HttpServletResponse** to assist in sending response to client.

Some Important Methods of ServletResponse

Methods	**Description**
PrintWriter getWriter()	returns a PrintWriter object that can send character text to the client.
void setBufferSize(int size)	Sets the preferred buffer size for the body of the response
void setContentLength(int len)	Sets the length of the content body in the response In HTTP servlets, this method sets the HTTP Content-Length header
void setContentType(String type)	sets the content type of the response being sent to the client before sending the respond.
void setBufferSize(int size)	sets the preferred buffer size for the body of the response.
boolean isCommitted()	returns a boolean indicating if the response has been committed
void setLocale(Locale loc)	sets the locale of the response, if the response has not been committed yet.

HttpServletResponse Interface

HttpServletResponse interface adds the methods that relates to the **HTTP**



response.

Some Important Methods of HttpServletResponse

Methods	**Description**
void addCookie(Cookie cookie)	adds the specified cookie to the response.
void sendRedirect(String location)	Sends a temporary redirect response to the client using the specified redirect location URL and clears the buffer
int getStatus()	gets the current status code of this response
String getHeader(String name)	gets the value of the response header with the given name.
void setHeader(String name, String value)	sets a response header with the given name and value
void setStatus(int sc)	sets the status code for this response
void sendError(int sc, String msg)	sends an error response to the client using the specified status and clears the buffer

HTTP Request

Servlet is a Java program which exists and executes in the J2EE servers and is used to receive the HTTP protocol request, process it and send back the response to the client. Servlets make use of the Java standard extension classes in the packages javax.servlet and javax.servlet.http. Since Servlets are written in the highly portable Java language and follow a standard framework, they provide a means to create the sophisticated server extensions in a server and operating system in an independent way.

Typical uses for HTTP Servlets include:

• Processing and/or storing the data submitted by an HTML form

• Providing dynamic content i.e. returning the results of a database query to the client • Managing state information on top of the stateless HTTP i.e. for an online shopping cart system which manages the shopping carts for many concurrent customers and maps every request to the right customer

As Servlet technology uses the Java language, thus web applications made using Servlet are

Secured, **Scalable**, and **Robust**.

HTTP Request Header

HTTP Request Header is used to pass the additional information about the requestor itself to the server. It can be used by the client to pass the useful information. getHeaderNames() and getHeader() methods of the javax.servlet.http.HttpServletRequest interface can be used to get the header information. Following are the important header information which comes from the browser side and can be frequently used while programming:

· **Accept**: This specifies the certain media types that are acceptable in the response · **Accept-Charset**: This indicates the character sets that are acceptable in the response. For e.g.: ISO-8859-1
· **Accept-Encoding**: This restricts the content-coding values that are acceptable in the response · **Accept-Language**: This restricts the set of language that is preferred in the response · **Authorization**: This type indicates that user agent is attempting to authenticate itself with a server
· **From**: This type contains the internet email address for the user who controls the requesting user agent
· **Host**: This type indicates the internet host and port number of the resource being requested · **If-Modified-Since**: This header indicates that the client wants the page only if it has been

changed after the specified date. The server sends a 304 code (i.e. a *Not Modified* header) if no newer result is available
· **Range**: This type requests one or more sub-ranges of the entity, instead of the entire entity · **Referrer**: This type enables the client to specify for the servers benefit, the address (URL) of the resources from which the request url was obtained
· **User-Agent**: This type contains the information about the user agent originating the request

```
// Import required java libraries
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import java.util.*;

// Extend HttpServlet class
public class DisplayHeader extends HttpServlet {

 // Method to handle GET method request.
 public void doGet(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {

 // Set response content type
 response.setContentType("text/html");

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

```
 String title = "HTTP Header Request Example";
 String docType =
 "<!doctype html public \"-//w3c//dtd html 4.0 " + "transitional//en\">\n";

 out.println(docType +
 "<html>\n" +
 "<head><title>" + title + "</title></head>\n"+
 "<body bgcolor = \"#f0f0f0\">\n" +
 "<h1 align = \"center\">" + title + "</h1>\n" +
 "<table width = \"100%\" border = \"1\" align = \"center\">\n" +
 "<tr bgcolor = \"#949494\">\n" +
 "<th>Header Name</th><th>Header Value(s)</th>\n"+
 "</tr>\n"
 );

 Enumeration headerNames = request.getHeaderNames();

 while(headerNames.hasMoreElements()) {
 String paramName = (String)headerNames.nextElement();
 out.print("<tr><td>" + paramName + "</td>\n");
 String paramValue = request.getHeader(paramName);
 out.println("<td> " + paramValue + "</td></tr>\n");
 }
 out.println("</table>\n</body></html>");
 }

 // Method to handle POST method request.
 public void doPost(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {

 doGet(request, response);
```

```
}
}
```

Steps to Create Servlets Application

A web server is required to develop a servlet application. We are using Apache Tomcat server to develop servlet application.

Following are the steps to develop a servlet application:

1. Create directory structure
2. Create a servlet
3. Compile the servlet
4. Create a deployment descriptor
5. Start the server and deploy the application

1. Create directory structure

There is a unique directory structure that must be followed to create Servlet application. This structure tells where to put the different types of files.

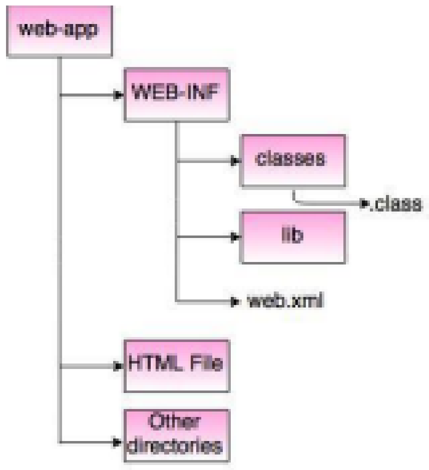


Fig: Directory Structure of Servlet Application

2. Create a Servlet

//ServletDemo.java

```
import javax.servlet.http.*;
import javax.servlet.*;
import java.io.*;
public class ServletDemo extends HttpServlet
{
 public void doGet(HttpServletRequest req, HttpServletResponse res) throws
ServletException, IOException
 {
```

```
 res.setContentType("text/html");
 PrintWriter pw = res.getWriter();
 pw.println("<html><body>");
 pw.println("First Program of Servlet");
 pw.println("</body></html>");
 pw.close();
 }
}
```

3. Compile the Servlet program

Assuming the classpath and environment is setup properly, run the above Java program.

C:\javac ServletDemo.java

After compiling the Java file, paste the class file of servlet in **WEB-INF/classes** directory.

4. Create a deployment descriptor

The deployment descriptor is an **xml** file. It is used to map URL to servlet class, defining error page.
//web.xml

```
<web-app>
 <servlet>
 <servlet-name>World</servlet-name>
 <servlet-class>ServletDemo</servlet-class>
 </servlet>

 <servlet-mapping>
 <servlet-name>Hello</servlet-name>
 <url-pattern>/Hello</url-pattern>
 </servlet-mapping>
</web-app>
```

Note:

- Copy the ServletDemoclass into **<Tomcat-installation-directory>/webapps/ROOT/WEB INF/classes.**

- Save the web.xml file in **<Tomcat-installation directoryt>/webapps/ROOT/WEB-INF/**

5. Start the server and deploy the application

Now start the Tomcat server by using

<Tomcat installation-direcory>\bin\startup.bat

Open browser and type

http://localhost:8080/demo/Hello

Web.xml (Deployment Descriptor)

This is the standard file for any web application and container comes to know about all the detail of web application through this file only.

All configurations of all components of a web application are done in web.xml and it is placed directly under WEB-INF folder.

Servlet Mapping

All servlets mapping need to be done in web.xml so that servlet container will be aware of the servlets and also to make it accessible from a browser. We must tell the servlet container what servlets to deploy, and what URL's to map the servlets to.

Below web.xml configures the Servlet "*MyFirstServlet*" with two init param.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<web-app xmlns="http://java.sun.com/xml/ns/j2ee"
xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance
xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd" version="2.4">
<servlet>
 <servlet-name> MyFirstServlet </servlet-name>
 <servlet-class> MyFirstServlet </servlet-class>
 <init-param>
 <param-name>name</param-name>
 <param-value>First Servlet</param-value>
 </init-param>
 <init-param>
 <param-name>purpose</param-name>
 <param-value>learning</param-value>
 </init-param>
```

```
 </servlet>
 <servlet-mapping>
 <servlet-name> MyFirstServlet </servlet-name>
 <url-pattern>*.html</url-pattern>
 </servlet-mapping>
</web-app>
```

Servlet is configured using the <servlet> element where we assign the servlet a unique name, and writes the qualified class of the servlet. Then map servlet to a specific URL or a URL pattern. This is done in the <servlet-mapping> element. In the above example, all URL's ending in .html are sent to the servlet.

In above sample we have defined one init param as well which can be accessed in a servlet. As the init params are defined within a servlet tag, they are limited to that servlet only. One init param tag is needed for one init parameter.

The * is a wild card, meaning any text.

When a request with URL ending with “html” , container-

• check the servlet mapping of web.xml and finds the servlet name based on url pattern. • Once servlet name is found, gets the servlet class for matching servlet name. • Delegate request to the servlet class configured in corresponding servlet-class tag.

Context Parameters

init parameters are limited to the servlet in which they are defined but context parameters are available to all the servlets of the web application. Below is the sample configuration of context parameter.

Since these parameters are not limited to any servlet, they are defined outside the <servlet> tag

```
<context-param>
 <param-name>param</param-name>
 <param-value>the value</param-value>
</context-param>
```

Early servlet loading & Initialization

As discussed earlier, servlet is loaded when first request for that servlet comes. There are scenarios where we need to get it loaded even before that or we can say at the time of application deployment. To do that, we can set the <load-on-startup> property of servlet configuration in web.xml

If we do not specify a <load-on-startup> element, the servlet container will typically load your servlet when the first request arrives for it.By setting a <load-on-startup> element, we tell the servlet container to load the servlet as soon as the servlet container starts.

```
<servlet>
<servlet-name>MyFirstServlet</servlet-name>
<servlet-class>MyFirstServlet</servlet-class>
<init-param>
<param-name>param-name</param-name>
<param-value>param-value</param-value>
</init-param>
<load-on-startup>1</load-on-startup>
</servlet>
```

The number inside the **<load-on-startup>1</load-on-startup>** tells the servlet container in what sequence the servlets should be loaded. The lower numbers are loaded first. If the value is negative, or unspecified, the servlet container can load the servlet at any time.

ServletConfig

Servlet Container creates ServletConfig object for each Servlet during initialization, to pass information to the Servlet. This object can be used to get configuration information such as parameter name and values from deployment descriptor file(web.xml).

Methods of ServletConfig interface

1. **public String getInitParameter(String name)**: Returns the value of given parameter as String, or null if the given parameter doesn't exist in web.xml.
2. **public Enumeration getInitParameterNames()**: Returns an enumeration of all the parameter names.
3. **public String getServletName()**: Returns the name of the servlet instance. 4.
public ServletContext getServletContext(): Returns an object of ServletContext.

Example:

In this example, we will use two methods getInitParameter() and getInitParameterNames() to get all the parameters from web.xml along with their values. The getInitParameterNames() method returns an enumeration of all parameters names and by passing those names during the call of getInitParameter() method, we can get the corresponding parameter value from web.xml.

DemoServlet.java

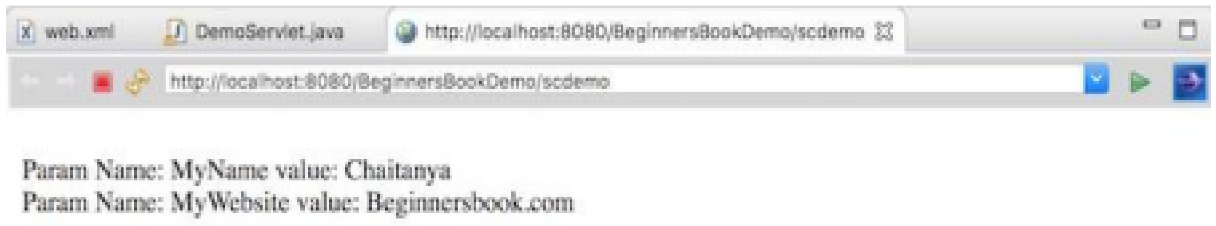
```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import java.util.Enumeration;

public class DemoServlet extends HttpServlet {

 protected void doGet(HttpServletRequest request,
HttpServletResponse response)
 throws ServletException, IOException
 {

 response.setContentType("text/html;charset=UTF-8");
PrintWriter pwriter = response.getWriter();  ServletConfig
sc=getServletConfig();

 Enumeration<String> e=sc.getInitParameterNames();
String str;
 while(e.hasMoreElements()) {
 str=e.nextElement();
 pwriter.println("<br>Param Name: "+str);
pwriter.println(" value: "+sc.getInitParameter(str));  }
 }
}
```

web.xml

```
<web-app>
 <display-name>BeginnersBookDemo</display-name
 > <welcome-file-list>
 <welcome-file>index.html</welcome-file>
 </welcome-file-list>
 <servlet>
 <servlet-name>MyServlet</servlet-name>
 <servlet-class>DemoServlet</servlet-class
 > <init-param>
 <param-name>MyName</param-name>
 <param-value>Chaitanya</param-value>

 </init-param>
 <init-param>
 <param-name>MyWebsite</param-name>
 <param-value>Beginnersbook.com</param-value
 > </init-param>
```

```
 </servlet>
 <servlet-mapping>
 <servlet-name>MyServlet</servlet-name>
 <url-pattern>/scdemo</url-pattern>
 </servlet-mapping>
</web-app>
```

Output:


web.xml
DemoServlet.java
http://localhost:8080/BeginnersBookDemo/scdemo
http://localhost:8080/BeginnersBookDemo/scdemo

```
Param Name: MyName value: Chaitanya
Param Name: MyWebsite value: Beginnersbook.com
```

ServletContext

In the last tutorial, we discussed ServletConfig, the Servlet Container creates ServletConfig object for each Servlet during initialization. The main difference between ServletConfig and ServletContext is that unlike ServletConfig, the ServletContext is being created once per web application, i.e. ServletContext object is common to all the servlets in web application.

This is how we can create ServletContext object. In this code we are creating object in init() method, however you can create the object anywhere you like.

```
ServletContext sc;
public void init(ServletConfig scfg)
{
 sc=scfg.getServletContext();
}
```

Once we have the ServletContext object, we can set the attributes of the ServletContext object by using the setattribute() method. Since the ServletContext object is available to all the servlets of the Web application, other servlets can retrieve the attribute from the ServletContext object by using the getAttribute() method.

Context Initialization Parameter

Context Initialization parameters are the parameter name and value pairs that you can specify in the deployment descriptor file (the web.xml file). Here you can specify the parameters that will be accessible to all the servlets in the web application.

When we deploy the Web application, the Servlet container reads the initialization parameter from the web.xml file and initializes the ServletContext object with it. We can use the getInitParameter() and getInitParameternames() methods of the ServletContext interface to get the parameter value and enumeration of parameter names respectively.

For example, here I have specified the parameter email_id with the value, since this is common to all the servlets, you can get the parameter name and value in any servlet.

```
<context-param>
<param-name>email_id</param-name>
<param-value>beginnersbook@gmail.com</param-value>
</context-param>
```

ServletContext complete example: To get the initialization parameters

In this example we have two context initialization parameters (user name and user email) in web.xml file and we are getting the value in Servlet using getInitParameter() method that returns the value of given parameter.

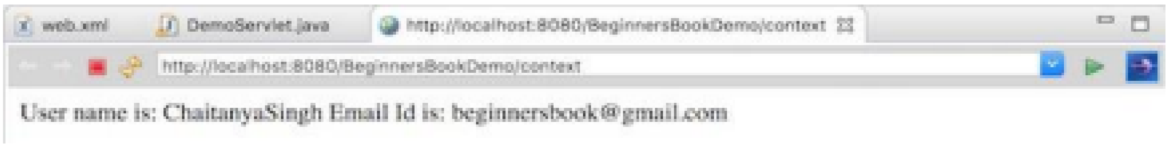
DemoServlet.java

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class DemoServlet extends HttpServlet{
 public void doGet(HttpServletRequest request,HttpServletResponse response)
throws ServletException,IOException
 {
 response.setContentType("text/html");
 PrintWriter pwriter=response.getWriter();

 //ServletContext object creation
 ServletContext scontext=getServletContext();

 //fetching values of initialization parameters and printing it
 String userName=scontext.getInitParameter("uname");
 pwriter.println("User name is="+userName);
 String userEmail=scontext.getInitParameter("email");
 pwriter.println("Email Id is="+userEmail);
 pwriter.close();
 }
}
```

web.xml

```
<web-app>
<servlet>
 <servlet-name>BeginnersBook</servlet-name>
 <servlet-class>DemoServlet</servlet-class>
</servlet>
<context-param>
 <param-name>uname</param-name>
 <param-value>ChaitanyaSingh</param-value>
</context-param>
<context-param>
 <param-name>email</param-name>
 <param-value>beginnersbook@gmail.com</param-value>
</context-param>
<servlet-mapping>
 <servlet-name>BeginnersBook</servlet-name>
 <url-pattern>/context</url-pattern>
 </servlet-mapping>
</web-app>
```

Output:


web.xml
DemoServlet.java
http://localhost:8080/BeginnersBookDemo/context
http://localhost:8080/BeginnersBookDemo/context

```
User name is: ChaitanyaSingh Email Id is: beginnersbook@gmail.com
```

Methods of ServletContext interface

Here is the list of frequently used methods of ServletContext interface.

1. **public String getInitParameter(String param)**: It returns the value of given parameter or null if the parameter doesn't exist.
2. **public Enumeration getInitParameterNames()**: Returns an enumeration of context parameters names.
3. **public void setAttribute(String name,Object object)**: Sets the attribute value for the given attribute name.
4. **public Object getAttribute(String name)**:Returns the attribute value for the given name or null if the attribute doesn't exist.
5. **public String getServerInfo()**: eturns the name and version of the servlet container on which the servlet is running.
6. **public String getContextPath()**: Returns the context path of the web application.

HttpSession

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

1. **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.
2. **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.
3. **public Enumeration getAttributeNames()**: Returns an Enumeration that contains the name of all the objects that are bound as attributes to the session object.
4. **public void removeAttribute(String name)**: Removes the given attribute from session. 5. **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.

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

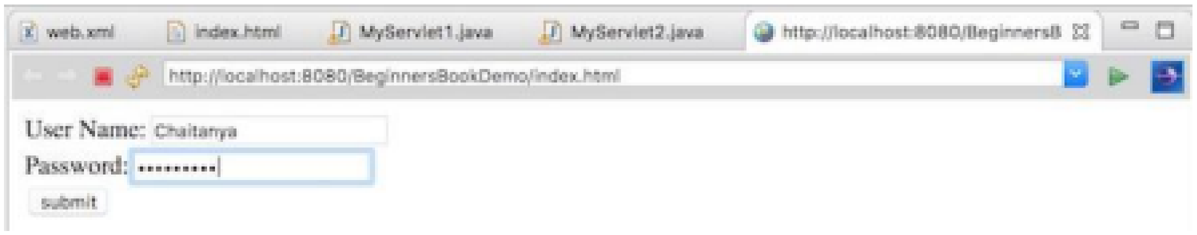
```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class MyServlet2 extends HttpServlet {
 public void doGet(HttpServletRequest request, HttpServletResponse response){
try{
 response.setContentType("text/html");
 PrintWriter pwriter = response.getWriter();
 HttpSession 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>
```

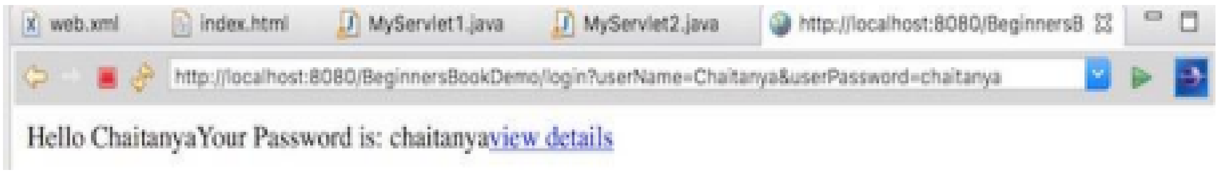
Output:
First Screen:



http://localhost:8080/BeginnersBookDemo/index.html

User Name: Chaitanya
Password: ••••••••••
submit

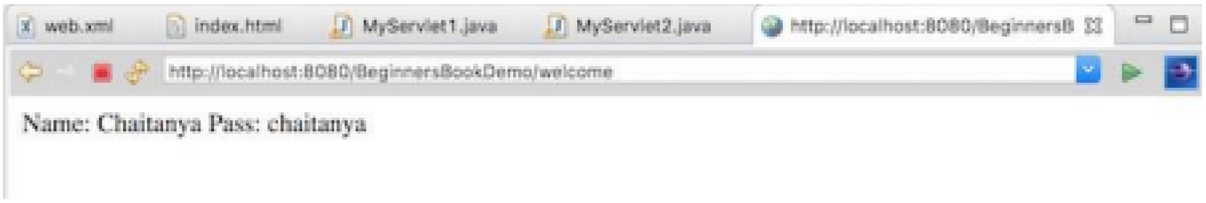
After clicking Submit:



http://localhost:8080/BeginnersBookDemo/login?userName=Chaitanya&userPassword=chaitanya

Hello ChaitanyaYour Password is: chaitanyaview details

After clicking view details:



http://localhost:8080/BeginnersBookDemo/welcome

Name: Chaitanya Pass: chaitanya

Cookies

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.*;
```

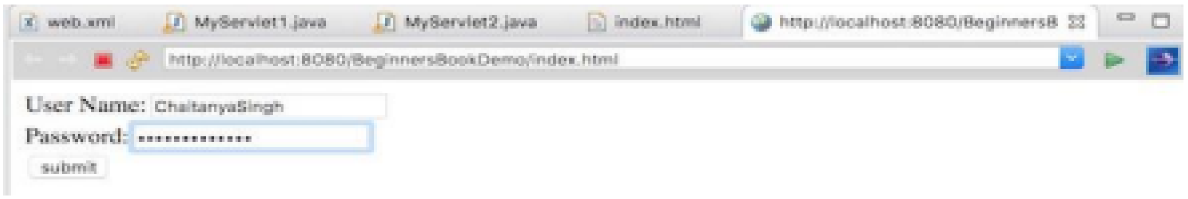
```
public class MyServlet2 extends HttpServlet {
public void doGet(HttpServletRequest request,
HttpServletResponse response){
 try{
 response.setContentType("text/html");
PrintWriter pwriter = response.getWriter();

 //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>
```

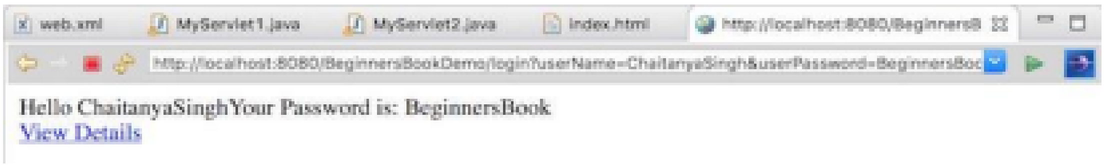
Output:
Welcome Screen:



http://localhost:8080/BeginnersBookDemo/index.html

User Name: ChaitanyaSingh
Password: ••••••••••••••
submit

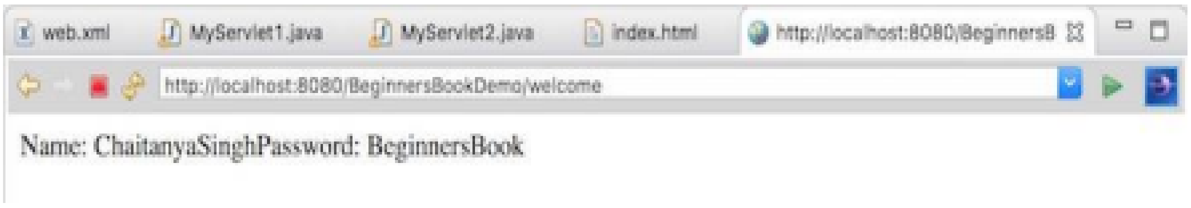
After clicking Submit:



http://localhost:8080/BeginnersBookDemo/login?userName=ChaitanyaSingh&userPassword=BeginnersBoo

Hello ChaitanyaSinghYour Password is: BeginnersBook
View Details

After clicking View Details:



http://localhost:8080/BeginnersBookDemo/welcome

Name: ChaitanyaSinghPassword: BeginnersBook

Methods of Cookie class

1. **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. 2. **public String getComment()**: Returns the comment describing the purpose of this cookie, or null if the cookie has no comment.
3. **public void setMaxAge(int expiry)**: Sets the maximum age of the cookie in seconds. 4. **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.

5. **public String getName()**: Returns the name of the cookie. The name cannot be changed after creation.
6. **public void setValue(String newValue)**: Assigns a new value to this Cookie.
7. **public String getValue()**: Gets the current value of this Cookie.

Java Server Pages

JSP is a server side technology that does all the processing at server. It is used for creating dynamic web applications, using java as programming language.

Basically, any html file can be converted to JSP file by just changing the file extension from ".html" to ".jsp", it would run just fine. What differentiates JSP from HTML is the ability to use java code inside HTML. In JSP, you can embed Java code in HTML using JSP tags. for e.g. run the code below, every time you run this, it would display the current time. That is what makes this code dynamic.

```
<HTML>
<BODY>
Hello BeginnersBook Readers!
Current time is: <%= new java.util.Date() %>
</BODY>
</HTML>
```

Your First JSP

Let's start learning JSP with a **simple JSP**.

```
<%-- JSP comment --%>
<HTML>
<HEAD>
<TITLE>MESSAGE</TITLE>
</HEAD>
<BODY>
<%out.print("Hello, Sample JSP code");%>
</BODY>
</HTML>
```

The above JSP generates the following output:
Hello, Sample JSP code.

Explanation of above code

1) The line **<%–JSP Comment–%>** represents the JSP element called JSP Comment, While adding comments to a JSP page you can use this tag, we will discuss this in detail in coming

posts.

Note: JSP Comments must starts with a tag **<%–** and ends with **–%>**

2) Head, Title and Body tags are HTML tags – They are HTML tags, frequently used for static web pages. Whatever content they have is delivered to client(Web browser) as such.

3) **<%out.print(" Hello, Sample JSP code ");%>** is a JSP element, which is known as Scriptlet. Scriptlets can contain Java codes. **syntax of scriptlet is:** <%Executable java code%>. As the code in Scriptlets is java statement, they must end with a semicolon(;). out.print(" Hello, Sample JSP code ") is a java statement, which prints" Hello, Sample JSP code".

As discussed, JSP is used for creating dynamic webpages. Dynamic webpages are usually a mix of static & dynamic content.

The **static content** can have text-based formats such as HTML, XML etc and the **dynamic content** is generated by JSP tags using java code inside HTML .

Servlet Vs JSP

Like JSP, Servlets are also used for generating dynamic webpages. Here is the comparison between them.

The major difference between them is that servlet adds HTML code inside java while JSP adds java code inside HTML. There are few other noticeable points that are as follows: **Servlets** –

1. Servlet is a Java program which supports HTML tags too.

2. Generally used for developing business layer(the complex computational code) of an enterprise application.
3. Servlets are created and maintained by Java developers.

JSP –

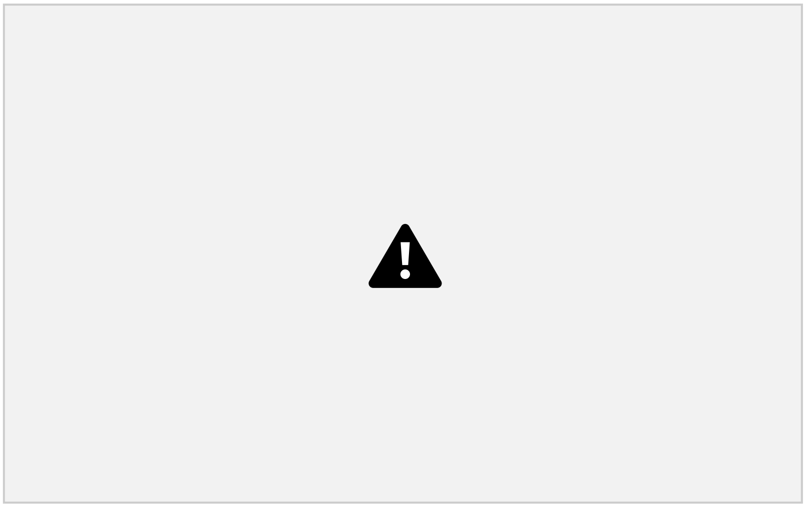
1. JSP program is a HTML code which supports java statements too.To be more precise, JSP embed java in html using JSP tags.
2. Used for developing presentation layer of an enterprise application
3. Frequently used for designing websites and used by web developers.

Advantages of JSP

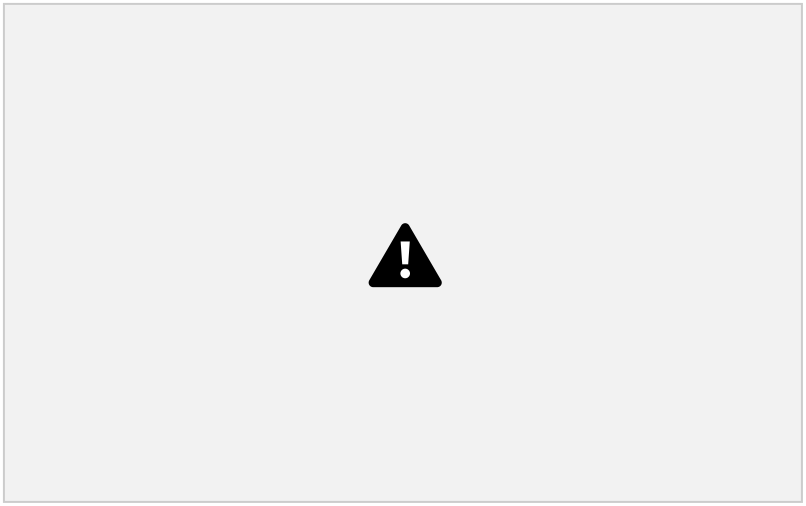
1. JSP has all the advantages of servlet, like: Better performance than CGI Built in session features, it also inherits the features of java technology like – multithreading, exception handling, Database connectivity, etc.
2. JSP Enables the separation of content generation from content presentation. Which makes it more flexible.
3. With the JSP, it is now easy for web designers to show case the information what is needed.
4. Web Application Programmers can concentrate on how to process/build the information.

Architecture of a JSP Application

Before we start developing web application, we should have a basic idea of architectures. Based on the location where request processing happens (Servlet OR JSP(java server pages)) there are two architectures for JSP. They are – Model1 Architecture & Model2 Architecture. **1) Model1 Architecture**: In this Model, JSP plays a key role and it is responsible for of processing the request made by client. Client (Web browser) makes a request, JSP then creates a bean object which then fulfils the request and pass the response to JSP. JSP then sends the response back to client. Unlike Model2 architecture, in this Model most of the processing is done by JSP itself.



2) Model2 Architecture: In this Model, Servlet plays a major role and it is responsible for processing the client's(web browser) request. Presentation part (GUI part) will be handled by JSP and it is done with the help of bean as shown in image below. The servlet acts as controller and in charge of request processing. It creates the bean objects if required by the jsp page and calls the respective jsp page. The jsp handles the presentation part by using the bean object. In this Model, JSP doesn't do any processing, Servlet creates the bean Object and calls the JSP program as per the request made by client.



Life Cycle of JSP

As the term suggests, the Life cycle can be defined as the process from creation to destruction along with the series of changes. The same applies to the life cycle of the Java Server pages. It can be defined as the procedure of origination till destruction, including all the phases a JSP follows. Further, we will discuss-

· Difference between Web Server, Web Container and Application Server ·
Different stages in the life of JSP

· Discussion of each stage in detail.

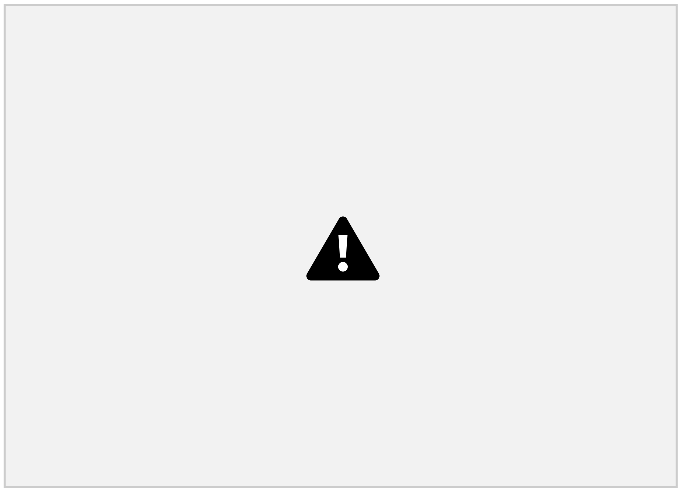
Before starting off with the life cycle, it is of utmost importance that we know the difference between a web server, web container and an application server.

Difference between a Web Server, a Web Container, and an Application Server

Web server: It receives HTTP request, intercepts it. It also processes the HTTP response to the web browser of the client. E.g., Apache web server.

Web container: A web container is J2EE compliant implementation. Its main job is to run the runtime environment to JSP and servlets. Request receiver at web browser is forwarded here, and the result generated is sent back to the web server. E.g., Tomcat.

Application Server: It is regarded as the complete server as it provides the facility of both a web server as well as a web container. It is a combination of both formers. E.g., Oracle Application Server, Bea WebLogic.



Stages of Java Server Page

JSP passes through the following phases:

1) Translation of JSP page

2) Compilation of JSP page

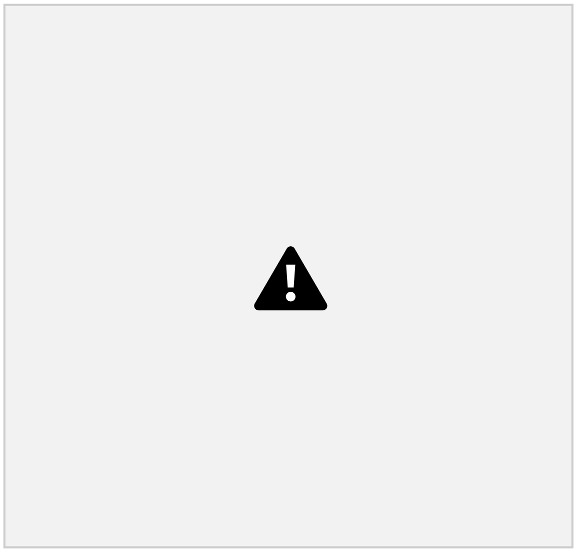
3) Class loading

4) Instantiation

5) Initialization

6) Request Processing

7) Destruction



Let's understand each and every phase in detail with the methods that are invoked:

1) JSP Translation

When the client sends the request to the web server, it goes to the web container. Now evidently, if the JSP page is newer and the servlet class is older, it means that the page got modified. In that case, the translation is done otherwise, this step is skipped so as to improve the performance of the system as it takes time.

2) JSP Compilation

Prior to compilation, JSP engine probes if the page has ever been compiled. If the page has some modifications or if it has never been compiled, then JSP engine compiles the page. Now the compilation follows three steps:

a) Parsing JSP

b) Conversion of JSP to Servlet

c) Compiling the servlet

3) Class Loading

At this stage, the .class file gets loaded by the loader i.e., corresponding servlet class file is loaded.

4) Instantiation

As we compiled the servlet, the object of the generated servlet is created. This process takes place after the class file is loaded in the memory. As soon as the instantiation takes place the objects: ServletConfig and ServletContext get accessible to the JSP class.

ServletConfig is a pre-defined interface used to develop flexible servlets. **ServletContext** is generated by the web container when the project is deployed. ServletConfig object exists as one per servlet program, whereas ServletContext object exists as one per web application. 5) Initialization

After instantiation, container invokes jspInit() method better known as initialization of JSP. It basically initializes the instance we created. Talking in detail, it is done to initialize resources needed to process the request i.e. resources like databases, allow JSP pages to read data, create lookup tables, and manage network connections.

Though it is invoked by the container only once, it can be overridden by the

author. public **void** jspInit() { / /Initialization code

}

/

6) Request Processing

After the initialization of servlet instances, a new thread gets created. All the interactions necessary to process requests occur at this stage. Thus, container invokes _jspService() method. This method contains two parameters:

- HTTPServletRequest req
- HTTPServletResponse res

```
public void _jspService(HTTPServletRequest req, HTTPServletResponse res)

{ // Enter jsp Service code here

}
```

This service method is responsible for generating a response to the request. Not only this, but it also generates response to all seven HTTP methods, namely:

a) GET

GET is a read only request. It tends to get information from the server. GET request posses 3 standard triggers:

- Typing in the address line of the browser and clicking enter or go.
- Pressing submit button in HTML form.
- Clicking on the link of a web page in order to access it.

b) POST

This method gets triggered when the form settings is made as a post.

c) HEAD

This method is almost similar to the GET method. Though this method doesn't return a message body, it can check if a resource hasn't been recently updated, valid, and accessible.

d) PUT

Using the request payload, PUT method replaces current representations of the target resource.

e) DELETE

This method deletes the specified resource.

f) CONNECT

This method establishes a path identified by the target resource to reach the server.

g) OPTIONS

This method describes the communication options that we need to communicate with the target resource.

7) Destruction

The destruction basically represents cleanup. For the cleanup jspDestroy() method gets invoked. This cleanup means that the JSP page after processing gets removed from the container. This method destroys the instance of the servlet class code, and all the connections to database and network are released. Opened files are also closed. jspDestroy() is overridden whenever the cleanup needs to be done.

```
public void jspDestroy() {

//Cleanup code

}
```

JSP Syntax and Semantics

This JSP deals with the writing structure and the use of it by the JSP containers. It describes the main elements of a JSP page, namely scripting elements, directives, and actions. It also discusses additional elements like comments, implicit objects, custom tags.

JSP Syntax and Semantics

As JSP deals with the writing of codes, a proper way to write that code becomes important. Thus we need to understand the syntax and semantics of a JSP code. To understand it better, we first need to know what syntax and semantics itself mean.

Syntax – Syntax is the coding structure to represent elements.

Semantics – It is the meaning of that element to the container i.e., what happens when that element is used.

Components of a JSP Page

1) JSP elements
2) Fixed template data
3) Or a combination of both

JSP elements
JSP containers will recognize these instructions. They tell about the generation of the code and how it needs to be operated. They have specific start and end tags for their recognition and identification by the web container.

JSP Elements

There are 3 most important elements in JSP

1. JSP scripting elements

1.1 Expressions

1.2 Scriptlets

1.3 Declarations

2. Directives

2.1page

2.2include

2.3taglib

3. Actions

Fixed template data

As this data is static or fixed, the compiler will not recognize it. This data won't change so that when the HTML file generates at the end, this data remains intact.

Above mentioned elements form the most part of JSP code. In addition to this, various other elements are there like:

- Implicit objects
- JSP comments
- JSP custom tags

1. JSP Scripting Elements

1.1 Expressions

They are a simple means for accessing the value of a Java variable. They can be an access point to other expressions and merging that value with HTML as well.

Syntax of expressions is:
<%=expression/statement%>

This code gets written to the output stream of the response. Now, this expression or statement can be any valid Java statement. This code will convert to out.print() statement when an auto generated servlet forms until and unless the expression is convertible to string format.

For Example:

```
<html>

<head><title>Expression</title>

 </head>

<body>

<h6>--Java Programming--</h6>

 <% String s1="Raghavendra"; %>

<% out.print(s1); %>

<% String s2=s1.toUpperCase(); %>

<% out.println( " "+s2); %>

 </body>

</html>
```

Explanation: Each line in the code is an expression. In this code we convert lower case to upper case.

Output:

1.2 Scriptlets

Scriptlets embed Java code in HTML for JSP code. It is like an expression, the only difference is that these are a set of statements written in Java language. It allows writing java code in it and is written between <%—-%> are called scriptlet tags. **Syntax of Scriptlet tag is as follows:** <% statement; [statement;….]%>

Everything that is inside a scriptlet goes to the _jspService() of the servlet code for processing the request. In case a code has multiple scriptlets then they will append in the _jspService in an ordered fashion.

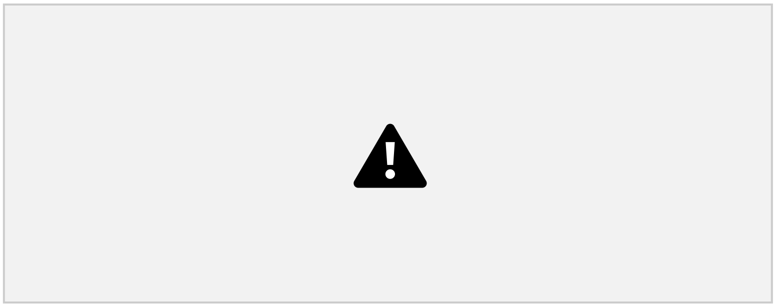
For Example:

```
<html>

<head><title>Expression</title>

</head>

<body>

<h6>--Java Programming--</h6>

<% int a=1;

 int b=2;

 int c=3;

 out.println("The number is " + a*b*c);

%>

 </body>
```

```
</html>
```

Explanation: In this code, we multiply three numbers and print the result in scriptlet tags.

Output:



1.3 Declarations

Declarations too have java language statements. Yet there lies a major difference. This code doesn't go to _jspService(). This code incorporates into the source file that gets generated outside the _jspService method. **Syntax is:**
<%! statement,[statement,….] %>

Important points to note about declaration are that:

- They have no access to implicit objects(discussed further).
- They declare class/instance/variable/methods/inner classes.
- Declaration is inside the servlet class but remains outside the service method. As said earlier this code doesn't go to _jspService() method.
- If a method that is inside declaration wants to use requested object from scriptlet then it needs to pass the object as parameter.

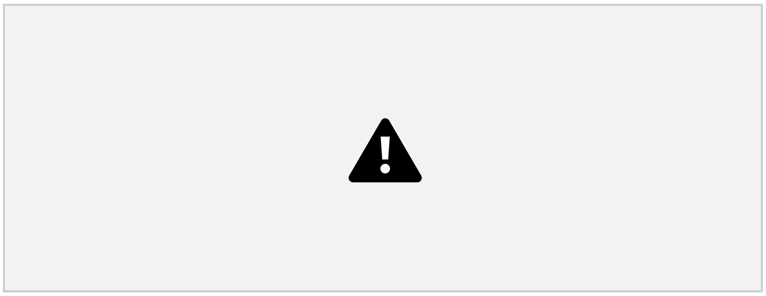
Example of declaration in JSP:

```
<html>

<head><title>Expression</title>

 </head>

<body>

<h6>--Java Programming--</h6>

 <%! int num=1, n=0; %>

<% n=num+1;
```

```
 out.println("The number is " + n);

%>

 </body>

</html>
```

Explanation: In this code, we have used declaration at code line 6, where we have declared the variables.

Output:



2. Directives

It is a type of instruction to the web container at the time of translation. Directives tell what code needs to be generated. It affects the compilation of the JSP page done by the container. **Syntax:** <%@ directive name [attribute="value"attribute="value"…..]%>

Note: They <%@……%> should lie in the same physical file.

Classification of Directives under 3 categories is as follows:

2.1 Page Directive

It defines the attributes for the JSP page as a whole.

Syntax:
<%@ page[attribute="value"attribute="value"]%>

The attributes of the page directive are:

- language
- extends
- import
- session
- buffer

- autoflush
- isThreadsafe
- info
- isErrorpage
- errorpage
- contentType

For Example:

<%@ page language="java" contentType="text/.html">

Explanation:
This code implies that coding of the page has been done in Java where the contentType of output is text or an HTML response. We won't get an output here as directives are an offset to the JSP code only.

2.2 Include Directive

It works similarly to the #include of the C preprocessor directory. It's work is to merge the contents of another file into .jsp input stream at the translation phase. **Syntax:** <%@ include file="filename"%>

For Example:
<%@ include file="header.html">

<%@ include file="sortmethod">

Explanation: The "filename" can be the absolute name or the relative path name as shown in the example. There is no body coding here, only the use of include tag is shown so output is not there.

2.3 Taglib Directive

With the use of tag library this directive makes the custom tags available in the current page. **Syntax can be shown as:**
<%@ taglib uri="taglib URI"prefix="tagPrefix"%>

The attributes of this directive are:

- tagLibrary URI
- tagPrefix

3. Actions

Actions also known as standard actions unlike others are written in XML. These are also known as high level JSP elements. They create, modify or use other objects. **Syntax:** <tagname attr="value":….>….body…….</tagname>

Proper nesting is important here. For e.g. <A><B>………</B></A> is right whereas <A><B>………</A></B> is wrong.
Important tags in actions are namely:

- <jsp:usebean>
- <jsp:setProperty>
- <jsp:getProperty>
- <jsp:include>
- <jsp:forward>
- <jsp:param>
- <jsp:plugin>

JSP Comments

Comments contain statements and texts but are ignored. They simply describe the code so that the usability and understanding increases.
There are two types of comments in a JSP page. They are:
Firstly, JSPcomments that are written in the JSP code. JSP container ignores these comments. **Syntax:**
<%----Hidden JSP comment----%>

Secondly the HTML comments that are written in the XML or HTML code. Browser ignores these comments. **Syntax:**
<!----included **in** generated HTML----!>

Example:

<html>

<head><title>Comments</title>

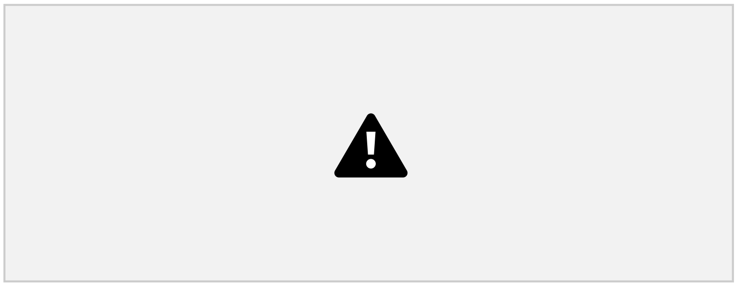
<!---This is an HTML comment--->

<h6>--Java Programming--</h6>

```
</head>

<body>


<%---This is a JSP comment---%>

<% int a=1;

 int b=2;

 out.println("The addition of a and b is:" + a+b);

%>

</body>

</html>
```

Explanation: In the generated output you can compare, the code and result. JSP and HTML comments get ignored by the JSP container and browser respectively.

Output:



Implicit Objects

As we know that only scriptlet, expression and HTML data goes to _jspService() and these variables are implicitly available to all except declaration. Thus they are implicitly available; they don't need to be declared.

Various implicit variables are request, response, pageContext, session, application, out, config, page, exception. They can also be created using tag libraries.

Custom Tags

These tags can be created so that the functionality of JSP can be extended. They allow the encapsulation of functionality. This is thus made available to non-expert page authors.

Configuring JSP File

Configuring a JSP into a web.xml file is optional because JSP is a public file to the web application.

A JSP called a public file and servlet is called a private file of the web application. Because JSP files stored in root directory of the web application and Servlet class file stored in sub directory of the web application. Because JSP is the public file, we can directly send a request from a browser by using its file name.

If we want ot configure a JSP in web.xml file the the xml elements are same as Servlet configuration. We need to replace <servlet-class> element with <jsp-file>

JSP's ***config*** object holds the *configuration details* like the username, password, parameter names and their values set in the configuration file(web.xml) and it is an object of type **javax.servlet.ServletConfig** interface.

Methods of ServletConfig interface

Methods	**Description**
String **getInitParameter(*String* name)**	This method gets an object stored in a session with a name, or null.
Enumeration **getInitParameterNames()**	This method sets an object with a name in a session.
ServletContext **getServletContext()**	This method removes an object with a name from the session.

String **getServletName()**	This method returns a RequestDispatcher which acts as a wrapper for the resource at the path.

An example of using the config object

In the upcoming example, we are first going to set the configuration data by setting a few variables and their values in the configuration file, **web.xml** as shown below. web.xml

```
<web-app>
<servlet>
<servlet-name>Raghavendra</servlet-name>
<jsp-file>/index.jsp</jsp-file>
</servlet>
<servlet-mapping>
<servlet-name>Raghavendra</servlet-name>
<url-pattern>/index.jsp</url-pattern>
</servlet-mapping>
</web-app>
```

and next we are going to retrieve the values of these configured variables using the JSP inbuilt **config** object and display it to the user.

index.jsp

```
<html>
<head> <title> Config Implicit Object</title>
</head>
<body>
<%
String sname=config.getServletName();
out.print("Servlet Name is: "+sname);
%>
</body>
</html>
```

Executing the index.jsp gives us the following window displaying the values of initialized variables/parameters accessed using ***config*** JSP implicit object.

JSP actions

JSP actions controls the behavior of the underneath servlet engine.

With JSP actions we can dynamically insert a file, reuse JavaBeans components, or forward the user to another page.

The following code is the syntax for the Action element:

```
<jsp:action_name attribute="value" />
```

Example

We can use the following JSP actions.

Syntax	**Purpose**
jsp:include	Includes a file
jsp:useBean	Initialize a JavaBean
jsp:setProperty	Sets the property of a JavaBean
jsp:getProperty	Get the property from a JavaBean
jsp:forward	Forwards the request to another page
jsp:plugin	Generates an OBJECT or EMBED tag for the Java plugin
jsp:element	Defines XML elements dynamically.
jsp:attribute	Defines XML element's attribute.
jsp:body	Defines XML element's body.
jsp:text	Write template text in JSP pages and documents.

Common Attributes in JSP Actions

There are two common attributes for all Action elements: the id attribute and the scope attribute.

Id attribute uniquely identifies the Action element. We can use the id to reference the action in the JSP page.

Scope attribute identifies the lifecycle of the Action element. The scope attribute has four possible values:

- page
- request
- session
- application

jsp:include Action

jsp:include Action inserts files into the JSP pages. The syntax is listed as follows:

```
<jsp:include page="relative URL" flush="true" />
```

The include directive includes a file at the time the JSP page is translated into a servlet, while jsp:include action inserts the file at the time the page is requested.

The following table lists the attributes associated with include action.

Attribute	Description
page	The relative URL of the page to be included.
flush	The boolean attribute determines whether the included resource should have its buffer flushed before including.

The following code shows how to use include action.

Here is the data.jsp.

```
<p>
 Today's date: <%= (new java.util.Date()).toLocaleString()%>
</p>
```

Here is the content of main.jsp file:

```
<html>
<body>
<jsp:include page="date.jsp" flush="true" />
</body>
</html>
```

jsp:useBean Action

The useBean action first searches for an existing object using the id and scope attributes. If an object is not found, it then will create the specified object.

The simplest way to load a bean is as follows:

```
<jsp:useBean id="name" class="java.lang.String" />
<jsp:useBean id="yourName" class="com.yourCom.YourClassName" />
```

Once a bean is loaded, we can use jsp:setProperty and jsp:getProperty actions to modify and retrieve bean properties.

The following table list attributes associated with useBean action.

Attribute	**Description**
class	Defines the full package name of the bean.
type	Specifies the type of the variable.
beanName	Named the loaded bean.

jsp:setProperty Action

The setProperty action sets the properties of a Bean.

We can use jsp:setProperty after outside of a jsp:useBean element as follows.

```
<jsp:useBean id="myName" ... />
<jsp:setProperty name="myName" property="someProperty" .../>
```

Or we can include jsp:setProperty inside the body of a jsp:useBean element as follows.

```
<jsp:useBean id="myName" ... >
 ...
 <jsp:setProperty name="myName" property="someProperty" .../>
</jsp:useBean>
```

In the code above, the jsp:setProperty is executed only if a new object was instantiated, not if an existing one was found.

The following table lists the attributes associated with setProperty action.

Attribute	Description
name	The name of the bean whose property will be set.
property	Property to set. A value of "*" means that all request parameters whose names match bean property names will be passed to the appropriate setter methods.
value	Value to set to the property.
param	The param attribute is the name of the request parameter.

jsp:getProperty Action

The getProperty action retrieves the value of a given property and converts it to a string, and then inserts it into the output.

The getProperty action has two attributes listed as follows:

```
<jsp:useBean id="myName" ... />
...
<jsp:getProperty name="myName" property="someProperty" .../>
```

Following is the list of required attributes associated with getProperty action.

Attribute	Description
name	The name of the Bean which must have been previously defined.
property	The name of the Bean property.

Example for userBean, set/get Property

Create a JavaBean as follows.

```
package action;

public class TestBean {
 private String message = "No message specified";

 public String getMessage() {
 return(message);
 }
 public void setMessage(String message) {
 this.message = message;
 }
}
```

Compile above code to TestBean.class file and copy TestBean.class to C:\apache tomcat\webapps\WEB-INF\classes\action folder.

Save the following code to hellow.jsp file.

```
<html>
```

```
<body>
<jsp:useBean id="test" class="action.TestBean" />
<jsp:setProperty name="test"
 property="message"
 value="Hello JSP..." />
<jsp:getProperty name="test" property="message" />
</body>
</html>
```

The jsp:forward Action

The forward action forwards the request to another a static page, or another JSP page, or a Java Servlet.

The simple syntax of this action is as follows:

```
<jsp:forward page="relativeURL | <%= expression %>" />
```

The following table lists the required attributes for forward action.

Attribute	Description
page	Relative URL for another resource.

The following code shows how to use the forward action.

The date.jsp file:

```
<p>
 Today's date: <%= (new java.util.Date()).toLocaleString()%>
</p>
```

Here is the main.jsp file:

```
<html>
<body>
<jsp:forward page="date.jsp" />
</body>
</html>
```

The jsp:plugin Action

The plugin action can insert Java applet, wrapped in the <object> or <embed> tags, into a JSP page.

The following code lists the typical syntax of using plugin action.

```
<jsp:plugin type="applet" codebase="dirname"
code="MyApplet.class"  width="60" height="80">
 <jsp:param name="fontcolor" value="red" />
 <jsp:param name="background" value="black" />

 <jsp:fallback>
 Unable to initialize Java Plugin
 </jsp:fallback>

</jsp:plugin>
```

The jsp:body Action

The <jsp:element>, <jsp:attribute> and <jsp:body> actions defines XML elements dynamically.

The following code shows how to define XML elements dynamically.

```
<%@page language="java" contentType="text/html"%>
<html xmlns="http://www.w3c.org/1999/xhtml"
 xmlns:jsp="http://java.sun.com/JSP/Page">
<body>
 <jsp:element name="xmlElement">
 <jsp:attribute name="xmlElementAttr">
 Value
 </jsp:attribute>
 <jsp:body>
 Body
 </jsp:body>
 </jsp:element>
</body>
</html>
```

Exception handling in JSP

Exception: These are nothing but the abnormal conditions which interrupts the normal flow of execution. Mostly they occur because of the wrong data entered by user. It is must to handle exceptions in order to give meaningful message to the user so that user would be able to understand the issue and take appropriate action.

Error: It can be a issue with the code or a system related issue. We should not handle errors as they are meant to be fixed.

Methods of handling exceptions:

We can handle exceptions using the below two methods.

- Exception handling using exception implicit object
- Exception handling using try catch blocks within scriptlets

Exception handling using exception implicit object

In the below example – we have specified the exception handling page using **errorPage** attribute of Page directive. If any exception occurs in the main JSP page the control will be transferred to the page mentioned in errorPage attribute.

The handler page should have **isErrorPage set to true** in order to use exception implicit object.

That's the reason we have set the isErrorPage true for errorpage.jsp.

index.jsp

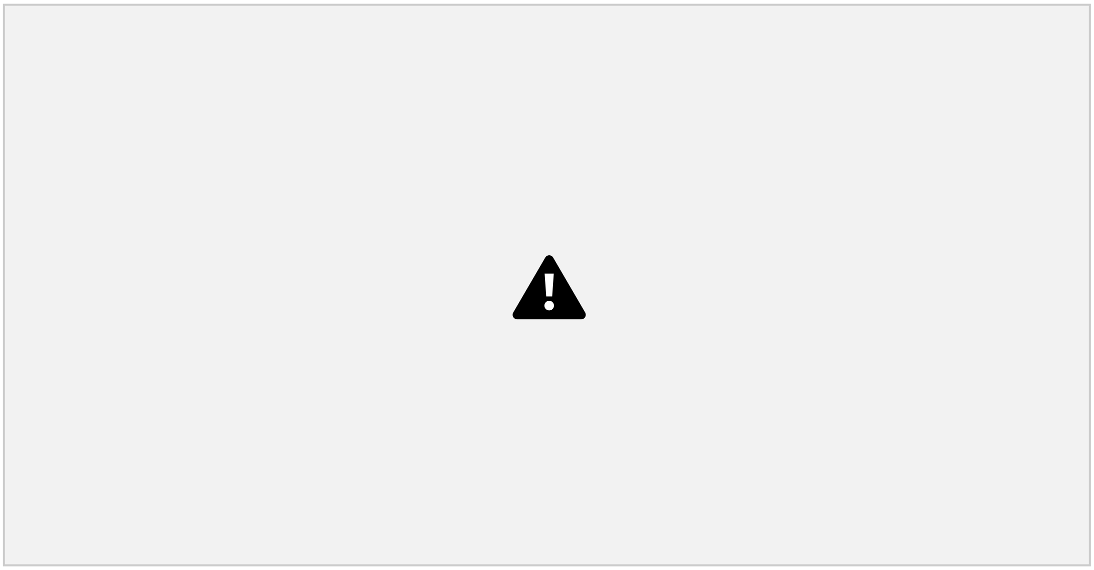
```
<%@ page errorPage="errorpage.jsp" %>
<html>
<head>
 <title>JSP exception handling example</title>
</head>
<body>
<%
 //Declared and initialized two integers
 int num1 = 122;
 int num2 = 0;

 //It should throw Arithmetic Exception
 int div = num1/num2;
%>
</body>
</html>
```

errorpage.jsp

```
<%@ page isErrorPage="true" %>
<html>
<head>
 <title>Display the ExceptAion Message here</title>
</head>
<body>
 <h2>errorpage.jsp</h2>
 <i>An exception has occurred in the index.jsp Page.
 Please fix the errors. Below is the error message:</i>
 <b><%= exception %></b>
</body>
```

```
</html>
```

output:



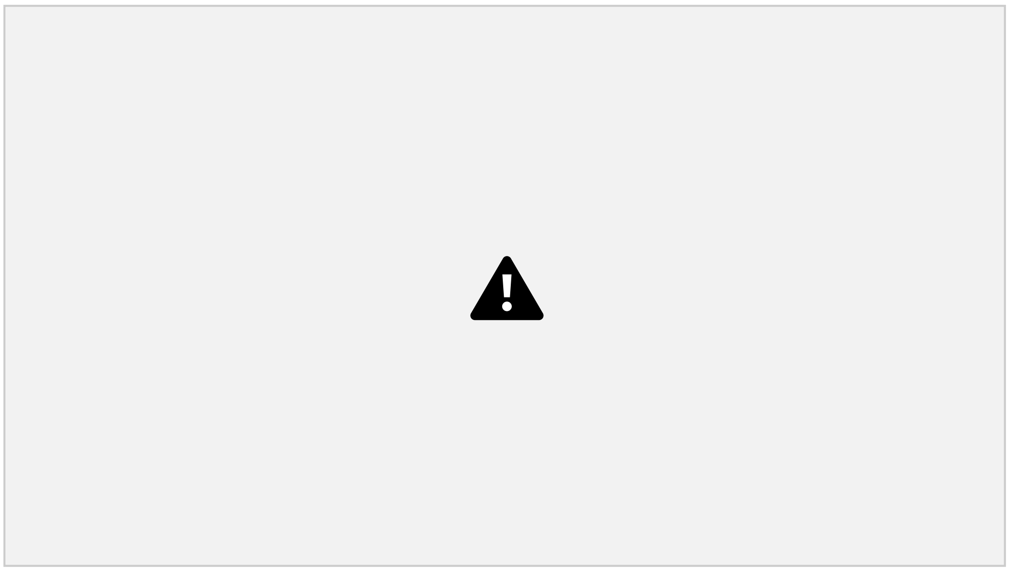
Exception handling using try catch blocks within scriptlets

We have handled the exception using try catch blocks in the below example. Since try catch blocks are java code so it must be placed inside sciptlet. In the below example I have declared an array of length 5 and tried to access the 7th element which doesn't exist. It caused Array Index out of bounds exception.

error.jsp

```
<html>
 <head>
 <title>Exception handling using try catch blocks</title>
 </head>
 <body>
 <%
 try{
 //I have defined an array of length 5
 int arr[]={1,2,3,4,5};
 //I'm assinging 7th element to int num
 //which doesn't exist
 int num=arr[6];
 out.println("7th element of arr"+num);
  }
 catch (Exception exp){
```

```
 out.println("There is something wrong: " + exp);
  }
 %>
 </body>
 </html>
```
output of example 2:



Let us know which method do you prefer for handling exceptions and why. If you have any questions, feel free to drop it in the below comment section.

Action Tags

In this tutorial we will see how to use a bean class in JSP with the help of **jsp:useBean, jsp:setProperty and jsp:getProperty** action tags.

Syntax of jsp:useBean:

```
<jsp: useBean id="unique_name_to_identify_bean"
class="package_name.class_name" />
```
Syntax of jsp:setProperty:

```
<jsp:setProperty name="unique_name_to_identify_bean"
property="property_name" />
```
Syntax of jsp:getProperty:

```
<jsp:getProperty name="unique_name_to_identify_bean"
property="property_name" />
```

A complete example of useBean, setProperty and getProperty

1) We have a bean class Details where we are having three variables username, age and password. In order to use the bean class and it's properties in JSP we have initialized the class like this in the userdetails.jsp page –

<jsp:useBean id="userinfo" class="beginnersbook.com.Details"></jsp:useBean> We have used useBean action to initialize the class. Our class is in beginnersbook.com package so we have given a fully qualified name **beginnersbook.com.Details**.

2) We have mapped the properties of bean class and JSP using setProperty action tag. We have given '*' in the property field to map the values based on their names because we have used the same property name in bean class and index.jsp JSP page. In the name field we have given the unique identifier which we have defined in useBean tag.

<jsp:setProperty property="*" name="userinfo"/>

3) To get the property values we have used getProperty action tag.

<jsp:getProperty property="*propertyname*" name="userinfo"/>

<u>Details.java</u>

```
package beginnersbook.com;
public class Details {
        public Details() {
 }
 private String username;
 private int age;
 private String password;
        public String getUsername() {
                return username;
        }
        public void setUsername(String username) {
                this.username = username;
        }
        public int getAge() {
                return age;
        }
        public void setAge(int age) {
                this.age = age;
        }
        public String getPassword() {
                return password;
        }
        public void setPassword(String password) {
                this.password = password;
        }

}
```

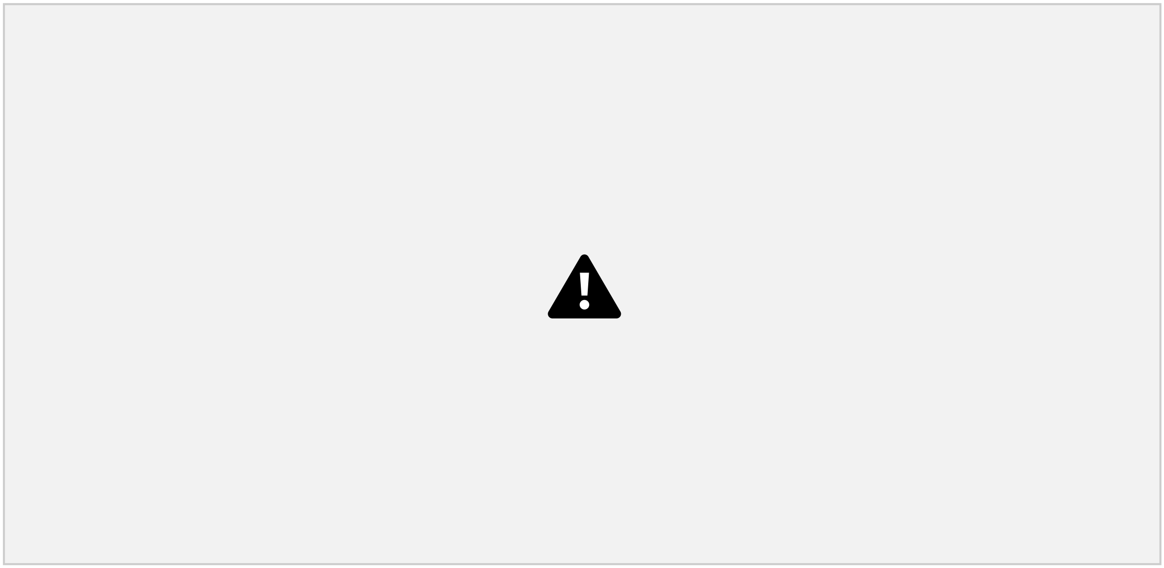
<u>index.jsp</u>

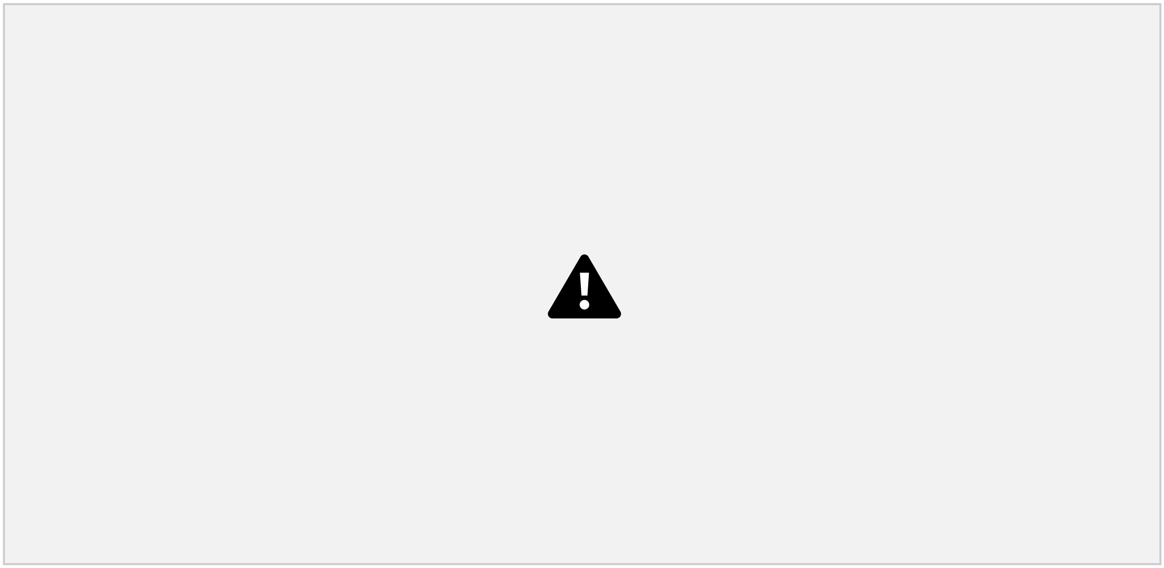
```
<html>
<head><title>
useBean, getProperty and setProperty example
</title></head>
<form action="userdetails.jsp" method="post">
User Name: <input type="text" name="username"><br>
User Password: <input type="password"
name="password"><br> User Age: <input type="text"
name="age"><br>
<input type="submit" value="register">
</form>
</html>
```

<u>userdetails.jsp</u>

```
<jsp:useBean id="userinfo"
class="beginnersbook.com.Details"></jsp:useBean> <jsp:setProperty
property="*" name="userinfo"/>
You have enterted below details:<br>
<jsp:getProperty                          property="username"
name="userinfo"/><br>                          <jsp:getProperty
property="password"                       name="userinfo"/><br>
<jsp:getProperty property="age" name="userinfo" /><br>
```

Output:





JSP Expression Language (EL)

Expression language (EL) has been introduced in JSP 2.0. The main purpose of it to simplify the process of accessing data from bean properties and from implicit objects. EL includes arithmetic, relational and logical operators too.

Synatx of EL:

${expression}
whatever present inside braces gets evaluated at runtime and being sent to the output stream.

Example 1: Expression language evaluates the expressions

In this example we are evaluating the expressions with the help of EL.

```
<html>
<head>
 <title>Expression language example1</title>
</head>
<body>
${1<2}
${1+2+3}
</body>
</html>
```

Output:

Example 2: Value fetch using param variable of expression language

In this example we are prompting user to enter name and roll number. On the other JSP page we are fetching the entered details using param variable of EL.

index.jsp

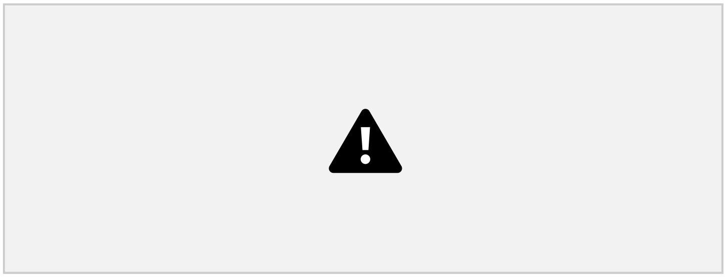
```
<html>
<head>
 <title>Expression language example2</title>
</head>
<body>
<form action="display.jsp">
Student Name: <input type="text" name="stuname" /><br>
Student RollNum:<input type="text" name="rollno" /><br>
```

```
<input type="submit" value="Submit Details!!"/>
</form>
</body>
</html>
```

display.jsp

```
<html>
<head>
<title>Display Page</title>
</head>
<body>
 Student name is ${ param.stuname } <br>
 Student Roll No is ${ param.rollno }
</body>
</html>
```

Output:



Example 3: Getting values from application object.

In this example we have set the attributes using application implicit object and on the display page we have got those attributes using **applicationScope** of Expression language.

index.jsp

```
<html>
 <head>
 <title>EL example3</title>
```

```
</head>
<body>
<%
application.setAttribute("author", "Raghavendra");
```

```
application.setAttribute("Site", "www.jainuniversity.ac.in");
%>
<a href="display.jsp">Click</a>
</body>
</html>
```

display.jsp

```
<html>
 <head>
 <title>Display Page</title>
 </head>
 <body>
 ${applicationScope.author}<br>
 ${applicationScope.Site}
 </body>
 </html>
```

Output:

EL predefined variables:

Similar to implicit objects in JSP we have predefined variables in EL. In the above examples we have used param and applicationScope, they are also the part of these variables.

pageScope: It helps in getting the attribute stored in Page scope.

pageContext: Same as JSP PageContext object.

sessionScope: Fetches attributes from session scope, set by session object.

requestScope: It used for getting the attributes from request scope. The attribute which are set by request implicit object.

param: Similar to **ServletRequest.getParameter**. Refer Example 2.

applicationScope: Used for getting Applicaton level attributes. Same what we see in Example 3. **header**: It helps in getting HTTP request headers as Strings.

headerValues: Used for fetching all the HTTP request headers.

initParam: It links to context initialization parameters.

paramValues: Same as ServletRequest.getParmeterValues.

cookie: It maps to Cookie object.

JSP Standard Tag Library (JSTL)

JSTL is part of the Java EE API and included in most servlet containers. But to use JSTL in our JSP pages, we need to download the JSTL jars for your servlet container. Most of the times, you can find them in the example projects of server download and you can use them. You need to include these libraries in your web application project **WEB-INF/lib** directory.

JSTL Tags

Based on the JSTL functions, they are categorized into five types.

1. **JSTL Core Tags**: JSTL Core tags provide support for iteration, conditional logic, catch exception, url, forward or redirect response etc. To use JSTL core tags, we should include it in the JSP page like below.

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

2. **JSTL Formatting and Localisation Tags**: JSTL Formatting tags are provided for formatting of Numbers, Dates and i18n support through locales and resource bundles. We can include these jstl

tags in JSP with below syntax:

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

3. **JSTL SQL Tags**: JSTL SQL Tags provide support for interaction with relational databases such as Oracle, MySql etc. Using JSTL SQL tags we can run database queries, we include these JSTL tags in JSP with below syntax:

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

4. **JSTL XML Tags**: JSTL XML tags are used to work with XML documents such as parsing XML, transforming XML data and XPath expressions evaluation. Syntax to include JSTL XML tags in JSP page is:

```
<%@ taglib uri="https://java.sun.com/jsp/jstl/xml" prefix="x" %>
```

5. **JSTL Functions Tags**: JSTL tags provide a number of functions that we can use to perform common operation, most of them are for String manipulation such as String Concatenation, Split String etc. Syntax to include JSTL functions in JSP page is:

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

Note that all the JSTL standard tags URI starts with https://java.sun.com/jsp/jstl/ and we can use any prefix we want but it's best practice to use the prefix defined above because everybody uses them, so it will not create any confusion.

JSTL Core Tags

JSTL Core Tags are listed in the below table.

JSTL Core Tag	Description
<c:out>	To write something in JSP page, we can use EL also with this tag
<c:import>	Same as <jsp:include> or include directive
<c:redirect>	redirect request to another resource
<c:set>	To set the variable value in given scope.

<c:remove>	To remove the variable from given scope
<c:catch>	To catch the exception and wrap it into an object.
<c:if>	Simple conditional logic, used with EL and we can use it to process the exception from <c:catch>
<c:choose>	Simple conditional tag that establishes a context for mutually exclusive conditional operations, marked by <c:when> and <c:otherwise>
<c:when>	Subtag of <c:choose> that includes its body if its condition evalutes to ‘true’.
<c:otherwise>	Subtag of <c:choose> that includes its body if its condition evalutes to ‘false’.
<c:forEach>	for iteration over a collection
<c:forTokens>	for iteration over tokens separated by a delimiter.
<c:param>	used with <c:import> to pass parameters
<c:url>	to create a URL with optional query string parameters

Database Connectivity & Web Application

Insert Data Into MySQL Using JSP Source Code

For insert data in MySQL using JSP first we have to create a table in data

base. The INSERT INTO statement is used to insert new data to a MySQL

table:

INSERT INTO table_name (column1, column2, column3,...)
VALUES (value1, value2, value3,...)

For creating table the SQL query is:

SQL Query

CREATE TABLE users

```
(
id int NOT NULL AUTO_INCREMENT,
first_name varchar(50),
last_name varchar(50),
city_name varchar(50),
email varchar(50),
PRIMARY KEY (id)
);
```

Here we using 2 files for insert data in MySQL:

- **index.html**:for getting the values from the user
- **process.jsp**:A JSP file that process the request

index.html

```
<!DOCTYPE html>
<html>
<body>
<form method="post" action="process.jsp">
First name:<br>
<input type="text" name="first_name">
<br>
Last name:<br>
<input type="text" name="last_name">
<br>
City name:<br>
<input type="text" name="city_name">
<br>
Email Id:<br>
<input type="email" name="email">
<br><br>
<input type="submit" value="submit">
</form>
</body>
</html>
```

process.jsp

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
```

```
pageEncoding="ISO-8859-1"%>
<%@page import="java.sql.*,java.util.*"%>

<%
String first_name=request.getParameter("first_name");
String last_name=request.getParameter("last_name");
String city_name=request.getParameter("city_name");
String email=request.getParameter("email");

try
{
Class.forName("com.mysql.jdbc.Driver");
```

```
Connection conn = DriverManager.getConnection("jdbc:mysql://localhost:3306/test", "root", "");
Statement st=conn.createStatement();

int i=st.executeUpdate("insert into
users(first_name,last_name,city_name,email)values('"+first_name+"','"+last_name+"','"+city_n
a me+"','"+email+"')");
out.println("Data is successfully inserted!");
}
catch(Exception e)
{
System.out.print(e);
e.printStackTrace();
}
%>
```

Output

First name:

Last name:

City name:

Email Id:

submit

How to Retrieve Data from MySQL Using JSP

For retrieve data from MySQL database using JSP first we have to create a table in data base. After create a table in the MySQL database you need to insert record or data on it. The SELECT statement is used to retrieve data from one or more tables:

The SQL query for retrieve specific column.

SELECT column_name(s) FROM table_name

or we can use the * character to retrieve ALL columns from a

table: SELECT * FROM table_name

In this example we retrieve the employee data of a company.

The JSP file for retrieving data from database is:

retrieve.jsp

```
<%@page import="java.sql.DriverManager"%>
<%@page import="java.sql.ResultSet"%>
<%@page import="java.sql.Statement"%>
<%@page import="java.sql.Connection"%>
<%
String id = request.getParameter("userid");
String driver = "com.mysql.jdbc.Driver";
String connectionUrl = "jdbc:mysql://localhost:3306/";
String database = "test";
String userid = "root";
String password = "";
try {
Class.forName(driver);
} catch (ClassNotFoundException e) {
e.printStackTrace();
}
Connection connection = null;
Statement statement = null;
ResultSet resultSet = null;
%>
<!DOCTYPE html>
<html>
<body>

<h1>Retrieve data from database in jsp</h1>
<table border="1">
<tr>
<td>first name</td>
<td>last name</td>
<td>City name</td>
```

```
<td>Email</td>

</tr>
<%
try{
connection = DriverManager.getConnection(connectionUrl+database, userid, password);
statement=connection.createStatement();
String sql ="select * from users";
resultSet = statement.executeQuery(sql);
while(resultSet.next()){
%>
<tr>
<td><%=resultSet.getString("first_name") %></td>
<td><%=resultSet.getString("last_name") %></td>
<td><%=resultSet.getString("city_name") %></td>
<td><%=resultSet.getString("email") %></td>
</tr>
<%
}
connection.close();
} catch (Exception e) {
e.printStackTrace();
}
```

```
%>
</table>
</body>
</html>
```

After retrieve the data from the data base the table look like this.

id first name last name City name Email Id

1 Divyasundar Sahu Mumbai divyasundar@gmail.com 2 Hritika Sahu Pune

hritika@gmail.com 3 Milan Jena Chennai milanjena@gmail.com

How to Update Data From MySql using JSP

To update the data in mysql table **UPDATE statement** is used.

```
UPDATE table_name
SET column1=value, column2=value2,...
WHERE some_column=some_value
```

Note: The WHERE clause specifies which data that should be updated. If you omit the WHERE clause, all records or data will be updated!

In the below example we update the employee data from MySQL database using JSP.

we used 3 file for update data

- index.jsp- To retrieve data from database with a update option.
- update.jsp- Show the employee data as per the selected id of employee(Suppose you select employee id 2, then it show only the information of id 2) .
- update-process.jsp- Process the user data after edit.

index.jsp

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-
1" pageEncoding="ISO-8859-1"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<%@ page import="java.sql.*" %>
<%@ page import="java.io.*" %>
```

```
<%@page import="java.sql.DriverManager"%>
<%@page import="java.sql.ResultSet"%>
<%@page import="java.sql.Statement"%>
<%@page import="java.sql.Connection"%>
<%
String id = request.getParameter("id");
String driver = "com.mysql.jdbc.Driver";
String connectionUrl = "jdbc:mysql://localhost:3306/";
String database = "test";
String userid = "root";
String password = "";
try {
Class.forName(driver);
} catch (ClassNotFoundException e) {
e.printStackTrace();
}
Connection connection = null;
Statement statement = null;
```

```
ResultSet resultSet = null;
%>
<html>
<body>
<h1>Retrieve data from database in jsp</h1>
<table border="1">
<tr>
<td>id</td>
<td>first name</td>
<td>last name</td>
<td>City name</td>
<td>Email</td>
<td>update</td>
</tr>
<%
try{
connection = DriverManager.getConnection(connectionUrl+database, userid, password);
statement=connection.createStatement();
String sql ="select * from users";
resultSet = statement.executeQuery(sql);
while(resultSet.next()){
%>
<tr>
<td><%=resultSet.getString("id") %></td>
<td><%=resultSet.getString("first_name") %></td>
<td><%=resultSet.getString("last_name") %></td>
<td><%=resultSet.getString("city_name") %></td>
<td><%=resultSet.getString("email") %></td>
<td><a href="update.jsp?id=<%=resultSet.getString("id")%>">update</a></td>
</tr>
<%
}
connection.close();
} catch (Exception e) {
e.printStackTrace();
}
%>
</table>
</body>
</html>
```

To update a record you must have insert data in the table. The users table look like this after run the index file.

id first name last name City name Email Id Action 1 Divyasundar Sahu Mumbai divyasundar@gmail.com Update 2 Hritika Sahu Pune hritika@gmail.com Update 3 Milan Jena Chennai milanjena@gmail.com Update

Now i am going to edit the city name and email id of employee **Divyasundar**.

update.jsp

```
<%@page import="java.sql.DriverManager"%>
<%@page import="java.sql.ResultSet"%>
<%@page import="java.sql.Statement"%>
```

```
<%@page import="java.sql.Connection"%>
<%
String id = request.getParameter("id");
String driver = "com.mysql.jdbc.Driver";
String connectionUrl = "jdbc:mysql://localhost:3306/";
String database = "test";
String userid = "root";
String password = "";
try {
Class.forName(driver);
} catch (ClassNotFoundException e) {
e.printStackTrace();
}
Connection connection = null;
Statement statement = null;
ResultSet resultSet = null;
%>
<%
try{
connection = DriverManager.getConnection(connectionUrl+database, userid, password);
statement=connection.createStatement();
String sql ="select * from users where id="+id;
```

```
resultSet = statement.executeQuery(sql);
while(resultSet.next()){
%>
<!DOCTYPE html>
<html>
<body>
<h1>Update data from database in jsp</h1>
<form method="post" action="update-process.jsp">
<input type="hidden" name="id" value="<%=resultSet.getString("id")
%>"> <input type="text" name="id" value="<%=resultSet.getString("id")
%>"> <br>
First name:<br>
<input type="text" name="first_name" value="<%=resultSet.getString("first_name")
%>"> <br>
Last name:<br>
<input type="text" name="last_name" value="<%=resultSet.getString("last_name")
%>"> <br>
City name:<br>
<input type="text" name="city_name" value="<%=resultSet.getString("city_name") %>">
<br>
Email Id:<br>
<input type="email" name="email" value="<%=resultSet.getString("email")
%>"> <br><br>
<input type="submit" value="submit">
</form>
<%
}
connection.close();
} catch (Exception e) {
e.printStackTrace();
}
%>
</body>
</html>
```

update-process.jsp

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-
 1" pageEncoding="ISO-8859-1"%>
<%@ page import="java.sql.*" %>
<%! String driverName = "com.mysql.jdbc.Driver";%>
```

```
<%!String url = "jdbc:mysql://localhost:3306/test";%>
<%!String user = "root";%>
<%!String psw = "";%>
<%
String id = request.getParameter("id");
String first_name=request.getParameter("first_name");
String last_name=request.getParameter("last_name");
String city_name=request.getParameter("city_name");
String email=request.getParameter("email");
if(id != null)
{
Connection con = null;
PreparedStatement ps = null;
int personID = Integer.parseInt(id);
```

```
try
{
Class.forName(driverName);
con = DriverManager.getConnection(url,user,psw);
String sql="Update users set id=?,first_name=?,last_name=?,city_name=?,email=? where
id="+id;
ps = con.prepareStatement(sql);
ps.setString(1,id);
ps.setString(2, first_name);
ps.setString(3, last_name);
ps.setString(4, city_name);
ps.setString(5, email);
int i = ps.executeUpdate();
if(i > 0)
{
out.print("Record Updated Successfully");
}
else
{
out.print("There is a problem in updating Record.");
}
}
catch(SQLException sql)
{
request.setAttribute("error", sql);
out.println(sql);
```

```
}
}
%>
```

After update the data the table look like this.

id first name last name City name Email Id Action 1 Divyasundar Sahu Delhi divyasundarsahu@gmail.com Update 2 Hritika Sahu Pune hritika@gmail.com Update 3 Milan Jena Chennai milanjena@gmail.com Update

How to Delete Data From MySql using JSP

In this example we will discuss about how to delete a record or data from MySQL database using JSP.

The DELETE statement is used to delete records from a table:

```
DELETE FROM table_name
WHERE some_column = some_value
```

Notice : The WHERE clause specifies which record or records that should be deleted. If you omit the WHERE clause, all records will be deleted!

The following examples delete the record with id=3 in the "users" table:

To delete a record from table you must have insert data. The users table look like this:

id first name last name City name Email Id

1 Divyasundar Sahu Mumbai divyasundar@gmail.com 2 Hritika Sahu Pune

hritika@gmail.com 3 Milan Jena Chennai milanjena@gmail.com

Now i am going to delete the id=3 record.

index.jsp

```
<%@page import="java.sql.DriverManager"%>
<%@page import="java.sql.ResultSet"%>
<%@page import="java.sql.Statement"%>
<%@page import="java.sql.Connection"%>
<%
String driver = "com.mysql.jdbc.Driver";
String connectionUrl = "jdbc:mysql://localhost:3306/";
String database = "student";
String userid = "root";
String password = "";
```

```
try {
Class.forName(driver);
 } catch (ClassNotFoundException e) {
e.printStackTrace();
 }
Connection connection = null;
Statement statement = null;
ResultSet resultSet = null;
%>
<!DOCTYPE html>
<html>
<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.2.1/jquery.min.js"></script>
<body>
<h1>Retrieve data from database in jsp</h1>
<table border="1">
<tr>
<td>first name</td>
<td>last name</td>
<td>City name</td>
<td>Email</td>
<td>Action</td>
</tr>
<%
try{
connection = DriverManager.getConnection(connectionUrl+database, userid, password);
statement=connection.createStatement();
String sql ="select * from users";
resultSet = statement.executeQuery(sql);
int i=0;
```

```
while(resultSet.next()){
%>
<tr>
<td><%=resultSet.getString("fname") %></td>
<td><%=resultSet.getString("lname") %></td>
<td><%=resultSet.getString("city_name") %></td>
<td><%=resultSet.getString("email") %></td>
<td><a href="delete.jsp?id=<%=resultSet.getString("id")
%>"><button type="button" class="delete">Delete</button></a></td>
</tr>
<%
i++;
}
connection.close();
} catch (Exception e) {
e.printStackTrace();
}
%>
</table>
</body>
</html>
```

delete.jsp

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-
 1" pageEncoding="ISO-8859-1"%>
<%@page import="java.sql.*,java.util.*"%>
<%
String id=request.getParameter("id");
try
{
Class.forName("com.mysql.jdbc.Driver");
Connection conn = DriverManager.getConnection("jdbc:mysql://localhost:3306/student", "root",
"");
Statement st=conn.createStatement();
int i=st.executeUpdate("DELETE FROM users WHERE id="+id);
out.println("Data Deleted Successfully!");
}
catch(Exception e)
{
System.out.print(e);
```

```
e.printStackTrace();
}
%>
```

After delete the id=3 record from the table the table is now

id first name last name City name Email Id

1 Divyasundar Sahu Mumbai divyasundar@gmail.com 2 Hritika Sahu Pune

hritika@gmail.com