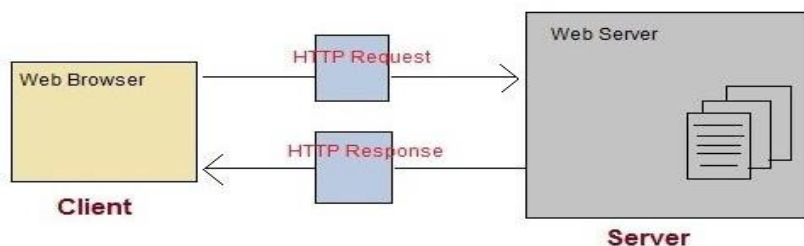


MODULE - 4

SERVLETS

Introduction to Web

The web clients make requests to web server. When a server answers a request, it usually sends some type of content(**MIME**- Multi purpose Internet Mail Exchange) to the client. The client uses web browser to send request to the server. The server often sends response to the browser with a set of instructions written in HTML(HyperText Markup Language)



Before Servlets, **CGI(Common Gateway Interface)** programming was used to create web applications. Here's how a CGI program works :

Drawbacks of CGI programs

- High response time because CGI programs execute in their own OS shell.
- CGI is not scalable.
- CGI programs are not always secure or object-oriented.
- It is Platform dependent.

Because of these disadvantages, developers started looking for better CGI solutions. And then Sun Microsystems developed **Servlet** as a solution.

Servlet

Servlet technology is used to create web application (resides at server side and generates dynamic web page).

Servlet can be described in many ways, depending on the context.

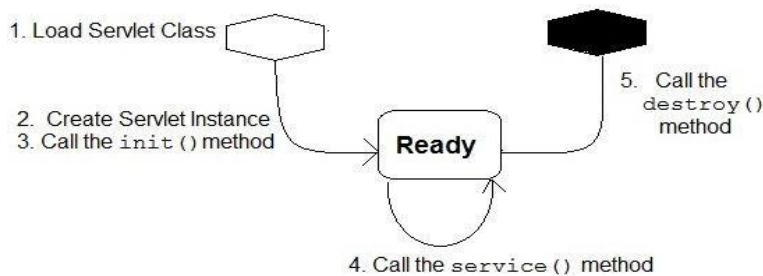
- Servlet is server side program.
- Servlet is an API that provides many interfaces and classes.
- Servlet is a web component that is deployed on the server to create dynamic web page.

Advantages of using Servlets

1. **better performance:** because it creates a thread for each request not process.
2. **Portability:** Servlets are platform independent because it uses java language.
3. **Robust:** Servlets are managed by JVM so we don't need to worry about memory leak, garbage collection etc.
4. **Secure:** servlets are object oriented and runs inside JVM.
5. Servlets are scalable.

Life Cycle of a Servlet

The web container maintains the life cycle of a servlet instance



1. **Loading Servlet Class :** A Servlet class is loaded when first request for the servlet is received by the Web Container.
2. **Servlet instance creation :** After the Servlet class is loaded, Web Container creates the instance of it. Servlet instance is created only once in the life cycle.
3. **Call to the `init()` method :** `init()` method is called by the Web Container on servlet instance to initialize the servlet.

`public void init(ServletConfig config) throws ServletException`

4. **Call to the service() method :** The containers call the service() method each time the request for servlet is received. The service() method will then call the doGet() or doPost() methods based on the type of the HTTP request, as explained in previous lessons.
`public void service(ServletRequest request, ServletResponse response) throws ServletException, IOException`
5. **Call to destroy() method:** The Web Container calls the destroy() method before removing servlet instance, giving it a chance for cleanup activity.

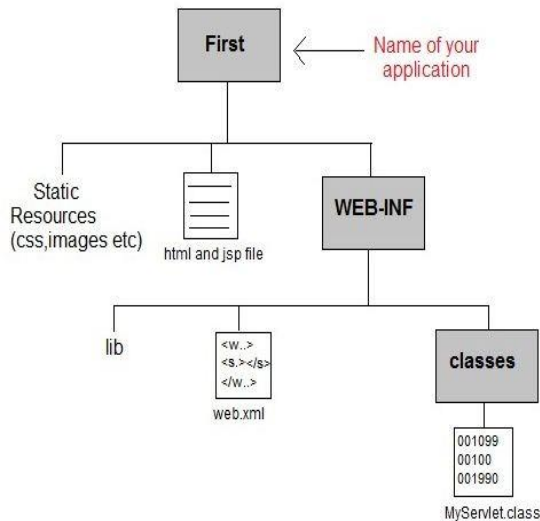
Steps to Create Servlet Application using tomcat server

To create a Servlet application you need to follow the below mentioned steps. These steps are common for all the Web server. In our example we are using Apache Tomcat server. Apache Tomcat is an open source web server for testing servlets and JSP technology. Create directory structure for your application.

1. Create directory structure for your application.
2. Create a Servlet
3. Compile the Servlet
4. Create Deployment Descriptor for your application
5. Start the server and deploy the application
- 6.

1. Creating the Directory Structure

Sun Microsystems defines a unique directory structure that must be followed to create a servlet application.



Create the above directory structure inside **Apache-Tomcat\webapps** directory.

All HTML, static files(images, css etc) are kept directly under **Web application** folder.

While all the Servlet classes are kept inside **classes** folder.

The **web.xml** (deployment descriptor) file is kept under **WEB-INF** folder.

2. Creating a Servlet

There are three different ways to create a servlet.

- By extending **HttpServlet** class
- By extending **GenericServlet** class
- By implementing **Servlet** interface

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

*// extending **HttpServlet** class*

```
public MyServlet extends HttpServlet
{
    public void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException
    {
        response.setContentType("text/html"); // set content type
        //get the stream to write the data
```

```
        PrintWriter out = response.getWriter(); // create printwriter object
        out.println("<h1>Hello Readers</h1>"); // print ur content on client web browser
    }
}
```

Write above code and save it as **MyServlet.java** anywhere on your PC. Compile it from there and paste the class file into WEB-INF/classes/ directory that you have to create inside **Tomcat/webapps** directory.

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

*// extending **GenericServlet** class*

```
public MyServlet extends GenericServlet
{
    public void service(ServletRequest request,ServletResponse response)
        throws IOException,ServletException{
    {
        response.setContentType("text/html"); // set content type
        //get the stream to write the data
        PrintWriter out = response.getWriter(); // create printwriter object
        out.println("<h1>Hello Readers</h1>"); // print ur content on client web browser
    }
}
```

3. Compiling a Servlet

To compile a Servlet a JAR file is required. Different servers require different JAR files. In Apache Tomcat server servlet-api.jar file is required to compile a servlet class.

- ☐ Download **servlet-api.jar** file.
- ☐ Paste the servlet-api.jar file inside Java\jdk\jre\lib\ext directory.

NOTE: After compiling your Servlet class you will have to paste the class file into WEB-INF/classes/ directory.

4. Create Deployment Descriptor

Deployment Descriptor(DD) is an XML document that is used by Web Container to run Servlets and JSP pages.

web.xml file

```
<web-app>
    <servlet>
        <servlet-name> MyServlet </servlet-name>
        <servlet-class> MyServlet </servlet-class>
    </servlet>

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

5. Starting Tomcat Server for the first time

set JAVA_HOME or JRE_HOME in environment variable (It is required to start server).

Go to My Computer properties -> Click on advanced tab then environment variables -> Click on the new tab of user variable -> Write JAVA_HOME in variable name and paste the path of jdk folder in variable value -> ok

Run Servlet Application

Open Browser and type **http:localhost:8080/First/hello**

Servlet API

Servlet API consists of two important packages that encapsulates all the important classes and interface, namely :

1. **javax.servlet.*;**

INTERFACES	CLASSES
Servlet	ServletInputStream
ServletContext	ServletOutputStream
ServletConfig	ServletException
ServletRequest	UnavailableException
ServletResponse	GenericServlet

Interface Summary	
<u>Servlet</u>	Defines methods that all servlets must implement.
<u>ServletRequest</u>	Defines an object to provide client request information to a servlet.
<u>ServletResponse</u>	Defines an object to assist a servlet in sending a response to the client.
<u>ServletConfig</u>	A servlet configuration object used by a servlet container to pass information to a servlet during initialization.
<u>ServletContext</u>	Defines a set of methods that a servlet uses to communicate with its servlet container, for example, to get the MIME type of a file, dispatch requests, or write to a log file.

Interface Servlet

Method	Description
<code>void destroy()</code>	Called when the servlet is unloaded.
<code>ServletConfig getServletConfig()</code>	Returns a ServletConfig object that contains any initialization parameters.
<code>String getServletInfo()</code>	Returns a string describing the servlet.
<code>void init(ServletConfig sc) throws ServletException</code>	Called when the servlet is initialized. Initialization parameters for the servlet can be obtained from <i>sc</i> . An UnavailableException should be thrown if the servlet cannot be initialized.
<code>void service(ServletRequest req, ServletResponse res) throws ServletException, IOException</code>	Called to process a request from a client. The request from the client can be read from <i>req</i> . The response to the client can be written to <i>res</i> . An exception is generated if a servlet or IO problem occurs.

Interface ServletRequest

Method	Description
<code>Object getAttribute(String attr)</code>	Returns the value of the attribute named <i>attr</i> .
<code>String getCharacterEncoding()</code>	Returns the character encoding of the request.
<code>int getContentLength()</code>	Returns the size of the request. The value <code>-1</code> is returned if the size is unavailable.
<code>String getContentType()</code>	Returns the type of the request. A <code>null</code> value is returned if the type cannot be determined.
<code>ServletInputStream getInputStream() throws IOException</code>	Returns a ServletInputStream that can be used to read binary data from the request. An IllegalStateException is thrown if getReader() has already been invoked for this request.
<code>String getParameter(String pname)</code>	Returns the value of the parameter named <i>pname</i> .
<code>Enumeration getParameterNames()</code>	Returns an enumeration of the parameter names for this request.
<code>String[] getParameterValues(String name)</code>	Returns an array containing values associated with the parameter specified by <i>name</i> .
<code>String getProtocol()</code>	Returns a description of the protocol.
<code>BufferedReader getReader() throws IOException</code>	Returns a buffered reader that can be used to read text from the request. An IllegalStateException is thrown if getInputStream() has already been invoked for this request.
<code>String getRemoteAddr()</code>	Returns the string equivalent of the client IP address.
<code>String getRemoteHost()</code>	Returns the string equivalent of the client host name.
<code>String getScheme()</code>	Returns the transmission scheme of the URL used for the request (for example, "http", "ftp").
<code>String getServerName()</code>	Returns the name of the server.
<code>int getServerPort()</code>	Returns the port number.

Interface [ServletResponse](#)

Method	Description
String getCharacterEncoding()	Returns the character encoding for the response.
ServletOutputStream getOutputStream() throws IOException	Returns a ServletOutputStream that can be used to write binary data to the response. An IllegalStateException is thrown if getWriter() has already been invoked for this request.
PrintWriter getWriter() throws IOException	Returns a PrintWriter that can be used to write character data to the response. An IllegalStateException is thrown if getOutputStream() has already been invoked for this request.
void setContentLength(int size)	Sets the content length for the response to <i>size</i> .
void setContentType(String type)	Sets the content type for the response to <i>type</i> .

Interface [ServletConfig](#)

Method	Description
ServletContext getServletContext()	Returns the context for this servlet.
String getInitParameter(String <i>param</i>)	Returns the value of the initialization parameter named <i>param</i> .
Enumeration getInitParameterNames()	Returns an enumeration of all initialization parameter names.
String getServletName()	Returns the name of the invoking servlet.

Interface [ServletContext](#)

Method	Description
Object getAttribute(String <i>attr</i>)	Returns the value of the server attribute named <i>attr</i> .
String getMimeType(String <i>file</i>)	Returns the MIME type of <i>file</i> .
String getRealPath(String <i>vpath</i>)	Returns the real path that corresponds to the virtual path <i>vpath</i> .
String getServerInfo()	Returns information about the server.
void log(String <i>s</i>)	Writes <i>s</i> to the servlet log.
void log(String <i>s</i> , Throwable <i>e</i>)	Writes <i>s</i> and the stack trace for <i>e</i> to the servlet log.
void setAttribute(String <i>attr</i> , Object <i>val</i>)	Sets the attribute specified by <i>attr</i> to the value passed in <i>val</i> .

Class Summary

<u>GenericServlet</u>	Defines a generic, protocol-independent servlet.
<u>ServletInputStream</u>	Provides an input stream for reading binary data from a client request, including an efficient readLine method for reading data one line at a time.
<u>ServletOutputStream</u>	Provides an output stream for sending binary data to the client.
<u>ServletException</u>	Defines a general exception a servlet can throw when it encounters difficulty.
<u>UnavailableException</u>	Defines an exception that a servlet or filter throws to indicate that it is permanently or temporarily unavailable.

Class GenericServlet

java.lang.Object

└ **javax.servlet.GenericServlet**

All Implemented Interfaces: [Servlet](#), [ServletConfig](#)

Class ServletInputStream

java.lang.Object

└ java.io.InputStream

└ **javax.servlet.ServletInputStream**

Method Summary

int	<u>readLine</u> (byte[] b, int off, int len)
	Reads the input stream, one line at a time.

Class ServletOutputStream

Method Summary

void	<u>println</u> ()
void	<u>print</u> (java.lang.String s)
	Writes a String to the client, without a carriage return-line feed (CRLF) character at the end.

void	println() Writes a carriage return-line feed (CRLF) to the client.
void	println(java.lang.String s) Writes a String to the client, followed by a carriage return-line feed (CRLF).

2. javax.servlet.http.*;

CLASSES	
INTERFACES	
Cookie	HttpServletRequest
HttpServlet	HttpServletResponse
HttpSessionBindingEvent	HttpSession

Interface Summary	
HttpServletRequest	Extends the ServletRequest interface to provide request information for HTTP servlets.
HttpServletResponse	Extends the ServletResponse interface to provide HTTP-specific functionality in sending a response.
HttpSession	Provides a way to identify a user across more than one page request or visit to a Web site and to store information about that user.

Interface HttpServletResponse

Method	Description
void addCookie(Cookie <i>cookie</i>)	Adds <i>cookie</i> to the HTTP response.
boolean containsHeader(String <i>field</i>)	Returns true if the HTTP response header contains a field named <i>field</i> .
String encodeURL(String <i>url</i>)	Determines if the session ID must be encoded in the URL identified as <i>url</i> . If so, returns the modified version of <i>url</i> . Otherwise, returns <i>url</i> . All URLs generated by a servlet should be processed by this method.
String encodeRedirectURL(String <i>url</i>)	Determines if the session ID must be encoded in the URL identified as <i>url</i> . If so, returns the modified version of <i>url</i> . Otherwise, returns <i>url</i> . All URLs passed to sendRedirect() should be processed by this method.
void sendError(int <i>c</i>) throws IOException	Sends the error code <i>c</i> to the client.
void sendError(int <i>c</i> , String <i>s</i>) throws IOException	Sends the error code <i>c</i> and message <i>s</i> to the client.
void sendRedirect(String <i>url</i>) throws IOException	Redirects the client to <i>url</i> .
void setDateHeader(String <i>field</i> , long <i>msec</i>)	Adds <i>field</i> to the header with date value equal to <i>msec</i> (milliseconds since midnight, January 1, 1970, GMT).
void setHeader(String <i>field</i> , String <i>value</i>)	Adds <i>field</i> to the header with value equal to <i>value</i> .
void setIntHeader(String <i>field</i> , int <i>value</i>)	Adds <i>field</i> to the header with value equal to <i>value</i> .
void setStatus(int <i>code</i>)	Sets the status code for this response to <i>code</i> .

Interface HttpServletRequest

Method	Description
String getAuthType()	Returns authentication scheme.
Cookie[] getCookies()	Returns an array of the cookies in this request.
long getDateHeader(String <i>field</i>)	Returns the value of the date header field named <i>field</i> .
String getHeader(String <i>field</i>)	Returns the value of the header field named <i>field</i> .
Enumeration getHeaderNames()	Returns an enumeration of the header names.
int getIntHeader(String <i>field</i>)	Returns the int equivalent of the header field named <i>field</i> .
String getMethod()	Returns the HTTP method for this request.
String getPathInfo()	Returns any path information that is located after the servlet path and before a query string of the URL.
String getPathTranslated()	Returns any path information that is located after the servlet path and before a query string of the URL after translating it to a real path.
String getQueryString()	Returns any query string in the URL.
String getRemoteUser()	Returns the name of the user who issued this request.
String getRequestedSessionId()	Returns the ID of the session.
String getRequestURI()	Returns the URI.
StringBuffer getRequestURL()	Returns the URL.
String getServletPath()	Returns that part of the URL that identifies the servlet.
HttpSession getSession()	Returns the session for this request. If a session does not exist, one is created and then returned.
HttpSession getSession(boolean <i>new</i>)	If <i>new</i> is true and no session exists, creates and returns a session for this request. Otherwise, returns the existing session for this request.
boolean isRequestedSessionIdFromCookie()	Returns true if a cookie contains the session ID. Otherwise, returns false .
boolean isRequestedSessionIdFromURL()	Returns true if the URL contains the session ID. Otherwise, returns false .
boolean isRequestedSessionIdValid()	Returns true if the requested session ID is valid in the current session context.

Interface

HttpSession

Method	Description
Object getAttribute(String attr)	Returns the value associated with the name passed in <i>attr</i> . Returns null if <i>attr</i> is not found.
Enumeration getAttributeNames()	Returns an enumeration of the attribute names associated with the session.
long getCreationTime()	Returns the time (in milliseconds since midnight, January 1, 1970, GMT) when this session was created.
String getId()	Returns the session ID.
long getLastAccessedTime()	Returns the time (in milliseconds since midnight, January 1, 1970, GMT) when the client last made a request for this session.
void invalidate()	Invalidates this session and removes it from the context.
boolean isNew()	Returns true if the server created the session and it has not yet been accessed by the client.
void removeAttribute(String attr)	Removes the attribute specified by <i>attr</i> from the session.
void setAttribute(String attr, Object val)	Associates the value passed in <i>val</i> with the attribute name passed in <i>attr</i> .

Class Summary

[Cookie](#)

Creates a cookie, a small amount of information sent by a servlet to a Web browser, saved by the browser, and later sent back to the server.

[HttpServlet](#)

Provides an abstract class to be subclassed to create an HTTP servlet suitable for a Web site.

[HttpSessionBindingEvent](#)

Events of this type are either sent to an object that implements [HttpSessionBindingListener](#) when it is bound or unbound from a session, or to a [HttpSessionAttributeListener](#) that has been configured in the deployment descriptor when any attribute is bound, unbound or replaced in a session.

[HttpSessionEvent](#)

This is the class representing event notifications for changes to sessions within a web application.

Class Cookie

Method	Description
Object clone()	Returns a copy of this object.
String getComment()	Returns the comment.
String getDomain()	Returns the domain.
int getMaxAge()	Returns the maximum age (in seconds).
String getName()	Returns the name.
String getPath()	Returns the path.
boolean getSecure()	Returns true if the cookie is secure. Otherwise, returns false .
String getValue()	Returns the value.
int getVersion()	Returns the version.
void setComment(String <i>c</i>)	Sets the comment to <i>c</i> .
void setDomain(String <i>d</i>)	Sets the domain to <i>d</i> .
void setMaxAge(int <i>secs</i>)	Sets the maximum age of the cookie to <i>secs</i> . This is the number of seconds after which the cookie is deleted.
void setPath(String <i>p</i>)	Sets the path to <i>p</i> .
void setSecure(boolean <i>secure</i>)	Sets the security flag to <i>secure</i> .
void setValue(String <i>v</i>)	Sets the value to <i>v</i> .
void setVersion(int <i>v</i>)	Sets the version to <i>v</i> .

Class

HttpServlet

Method	Description
void doDelete(HttpServletRequest req, HttpServletResponse res) throws IOException, ServletException	Handles an HTTP DELETE request.
void doGet(HttpServletRequest req, HttpServletResponse res) throws IOException, ServletException	Handles an HTTP GET request.
void doHead(HttpServletRequest req, HttpServletResponse res) throws IOException, ServletException	Handles an HTTP HEAD request.
void doOptions(HttpServletRequest req, HttpServletResponse res) throws IOException, ServletException	Handles an HTTP OPTIONS request.
void doPost(HttpServletRequest req, HttpServletResponse res) throws IOException, ServletException	Handles an HTTP POST request.
void doPut(HttpServletRequest req, HttpServletResponse res) throws IOException, ServletException	Handles an HTTP PUT request.
void doTrace(HttpServletRequest req, HttpServletResponse res) throws IOException, ServletException	Handles an HTTP TRACE request.
long getLastModified(HttpServletRequest req)	Returns the time (in milliseconds since midnight, January 1, 1970, GMT) when the requested resource was last modified.
void service(HttpServletRequest req, HttpServletResponse res) throws IOException, ServletException	Called by the server when an HTTP request arrives for this servlet. The arguments provide access to the HTTP request and response, respectively.

Class HttpSessionBindingEvent

Method Summary

java.lang.String	getName() Returns the name with which the attribute is bound to or unbound from the session.
HttpSession	getSession() Return the session that changed.
java.lang.Object	getValue()

	Returns the value of the attribute that has been added, removed or replaced.
--	--

Class HttpSessionEvent

Method Summary

HttpSession	getSession() Return the session that changed.
-----------------------------	--

Reading Servlet parameters

In this example, we will show how a parameter is passed to a **index.html**

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

MyServlet.java

```
public class MyServlet extends HttpServlet {

    protected void doPost(request, response){
        ... // set content type . . .

        String user=request.getParameter("user");

        out.println("<h2> Welcome "+user+"</h2>");
    }
}
```

NOTE: getParameter() returns string to get int value use

```
Integer.parseInt("sting");
```

Handling HTTP request and Response

HttpServlet class provides various methods that handle various types of HTTP request.

A servlet typically must override at least one method, usually one of these:

- doGet, if the servlet supports HTTP GET requests
- doPost, for HTTP POST requests
- doPut, for HTTP PUT requests
- doDelete, for HTTP DELETE requests

GET and POST methods are commonly used when handling form input.

NOTE: By default a request is Get request.

Difference between GET and POST requests

GET Request	POST Request
Data is sent in header to the server	Data is sent in the request body
Get request can send only limited amount of data	Large amount of data can be sent.
Get request is not secured because data is exposed in URL	Post request is secured because data is not exposed in URL.

Session

Session simply means a particular interval of time. **Session Tracking** is a way to maintain state (data) of an user. It is also known as **session management** in servlet.

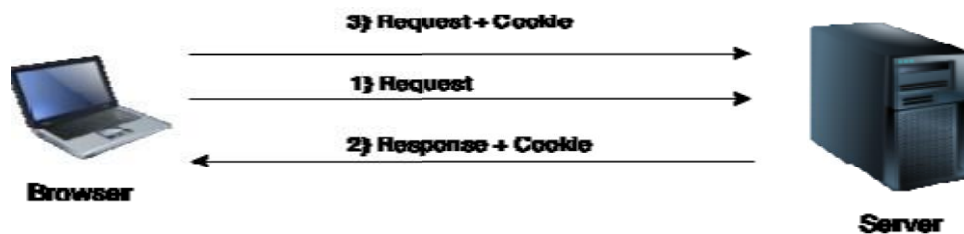
Http protocol is a stateless so we need to maintain state using session tracking techniques. Each time user requests to the server, server treats the request as the new request. So we need to maintain the state of an user to recognize to particular user.

There are 2 techniques used in Session tracking:

1. **Cookies**
2. **HttpSession**

Cookies in Servlet

A **cookie** is a small piece of information that is persisted between the multiple client requests. By default, each request is considered as a new request. In cookies technique, we add cookie with response from the servlet. So cookie is stored in the cache of the browser. After that if request is sent by the user, cookie is added with request by default. Thus, we recognize the user as the old user.



Advantage of Cookies

1. Simplest technique of maintaining the state.
2. Cookies are maintained at client side.

Disadvantage of Cookies

1. It will not work if cookie is disabled from the browser.
2. Only textual information can be set in Cookie object.

Creating a new Cookie

```
Cookie ck = new Cookie("username", name);
```

← creating a new cookie object

Setting up lifespan for a cookie

```
ck.setMaxAge(30*60);
```

← setting maximum age of cookie

Sending the cookie to the client

```
response.addCookie(ck);
```

← adding cookie to response object

Getting cookies from client request

```
Cookie[] cks = request.getCookies();
```

← getting the cookie for request object

Example demonstrating usage of Cookies

index.html

```
<form method="post" action=" MyServlet ">
    Name:<input type="text" name="user" /><br/>
    Password:<input type="text" name="pass" ><br/>
    <input type="submit" value="submit">
</form>
```

MyServlet.java

```
public class MyServlet extends HttpServlet {

    protected void doPost(HttpServletRequest request, HttpServletResponse response)
    {    // .....
        String name = request.getParameter("user");
        String pass = request.getParameter("pass");

        if(pass.equals("1234"))
        {
            Cookie ck = new Cookie("username",name);
            response.addCookie(ck);

            //response.sendRedirect("First");//call ur servlet

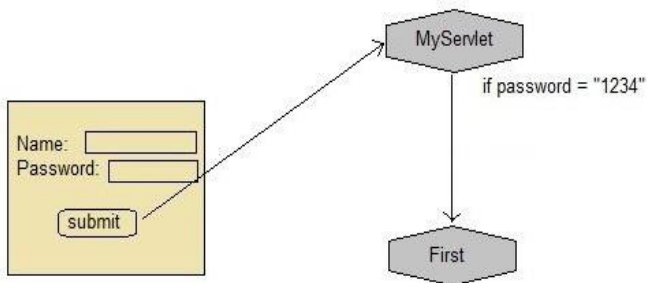
            //creating submit button
            out.print("<form action= First >");
            out.print("<input type='submit' value='go'>");
            out.print("</form>");
        }
    }
}
```

First.java

```
public class First extends HttpServlet {
```

```
protected void doGet(HttpServletRequest request, HttpServletResponse response)
```

```
{    // ...  
    Cookie[] cks = request.getCookies();  
    out.println("Welcome "+cks[0].getValue());  
}  
}
```



Useful Methods of Cookie class

Method	Description
public void setMaxAge(int expiry)	Sets the maximum age of the cookie in seconds.
public String getName()	Returns the name of the cookie. The name cannot be changed after creation.
public String getValue()	Returns the value of the cookie.
public void setName(String name)	changes the name of the cookie.
public void setValue(String value)	changes the value of the cookie.

Other methods required for using Cookies

For adding cookie or getting the value from the cookie, we need some methods provided by other interfaces. They are:

1. **public void addCookie(Cookie ck):**method of HttpServletResponse interface is used to add cookie in response object.
2. **public Cookie[] getCookies():**method of HttpServletRequest interface is used to return all the cookies from the browser.

HttpSession

HttpSession object is used to store entire session with a specific client. We can store, retrieve and remove attribute from **HttpSession** object. Any servlet can have access to **HttpSession** object throughout the getSession() method.

Creating a new session

```
HttpSession session = request.getSession();
```

getSession() method returns a session.
If the session already exist, it return the
existing session else create a new
session

```
HttpSession session = request.getSession(true);
```

getSession(true) always return
a new session

Getting a pre-existing session

```
HttpSession session = request.getSession(false);
```

return a pre-existing
session

Destroying a session

```
session.invalidate();
```

destroy a session

Some Important Methods of HttpSession

Methods	Description
long getCreationTime()	returns the time when the session was created, measured in milliseconds since midnight January 1, 1970 GMT.

String getId()	returns a string containing the unique identifier assigned to the session.
int getMaxInactiveInterval()	returns the maximum time interval, in seconds.
void invalidate()	destroy the session
boolean isNew()	returns true if the session is new else false

Complete Example demonstrating usage of HttpSession

index.html

```
<form method="post" action="Validate">  
  User: <input type="text" name="uname " /><br/>  
  <input type="submit" value="submit">  
</form>
```

Validate.java

```
public class Validate extends HttpServlet {  
  
    protected void doPost(request, response)  
    {  
        // ....  
        String name = request.getParameter("user");  
        //creating a session  
        HttpSession session = request.getSession();  
        session.setAttribute("user", uname);  
        response.sendRedirect("Welcome");  
    }  
}
```

Welcome.java

```
public class Welcome extends HttpServlet {  
  
    protected void doGet(request, response){  
        // ....  
        HttpSession session = request.getSession();  
        String user = (String)session.getAttribute("user");  
        out.println("Hello "+user);  
    }  
}
```

JSP

- **Java Server Page** technology is used to create dynamic web applications.
- **JSP** pages are easier to maintain than a **Servlet**.
- JSP pages are opposite of Servlets as a servlet adds HTML code inside Java code, while JSP adds Java code inside HTML using JSP tags.
- Everything a Servlet can do, a JSP page can also do it.
- JSP enables us to write HTML pages containing tags, inside which we can include powerful Java programs

Why JSP is preferred over servlets?

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

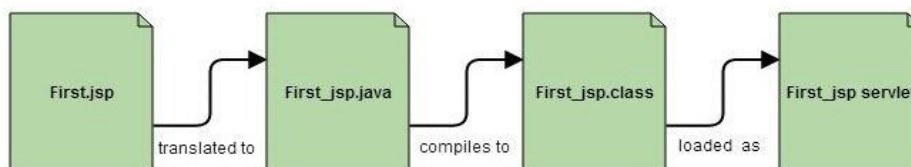
Advantage of JSP

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

Life cycle of a JSP Page

The JSP pages follow these phases:

- Initialization (`jspInit()` method is invoked by the container).
- Request processing (`_jspService()` method is invoked by the container).
- Destroy (`jspDestroy()` method is invoked by the container).



In the end a JSP becomes a Servlet

- As depicted in the above diagram, JSP page is translated into servlet by the help of JSP translator. The JSP translator is a part of webserver that is responsible to translate the JSP page into servlet. After that Servlet page is compiled by the compiler and gets converted into the class file.
- **Web Container** translates JSP code into a **servlet class source(.java) file**, then compiles that into a java servlet class. In the third step, the servlet class bytecode is loaded using classloader. The Container then creates an instance of that servlet class.
- The initialized servlet can now service request. For each request the **Web Container** call the `_jspService()` method. When the Container removes the servlet instance from service, it calls the `jspDestroy()` method to perform any required clean up.

Do we need to follow directory structure to run a simple JSP ?

No, put jsp files in a folder directly and deploy that folder. It will be running fine. But if you are using bean class, Servlet or tld file then directory structure is required.

JSP Scripting Element

JSP Scripting element are written inside `<% %>` tags. These code inside `<% %>` tags are processed by the JSP engine during translation of the JSP page. Any other text in the JSP page is considered as HTML code or plain text.

There are five different types of scripting elements

1. **JSP scriptlet tag** A scriptlet tag is used to execute java source code in JSP.
`<% java source code %>`

In this example, we are displaying a welcome message.

```
<html>
<body>
<% out.print("welcome to jsp"); %>
</body>
</html>
```

2. **JSP Declaration Tag**

The **JSP declaration tag** is used to *declare variables, objects and methods*.

The code written inside the jsp declaration tag is placed outside the `service()` method of auto generated servlet.

So it doesn't get memory at each request.

`<%! field or method declaration %>`

Difference between JSP Scriptlet tag and Declaration tag

Jsp Scriptlet Tag	Jsp Declaration Tag
The jsp scriptlet tag can only declare variables not methods.	The jsp declaration tag can declare variables as well as methods.
The declaration of scriptlet tag is placed inside the <code>_jspService()</code> method.	The declaration of jsp declaration tag is placed outside the <code>_jspService()</code> method.

declaration tag with variable

In

index.jsp

```
<html>
<body>
<%! int data=50; %>
<%= "Value of the variable is:"+data %>
</body>
</html>
```

declaration tag that declares method

index.jsp

```
<html>
<body>
<%!
int cube(int n){
return n*n*n*;
}
%>
<%= "Cube of 3 is:"+cube(3) %>
```

3. **JSP Expression Tag**

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

Syntax of Expression Tag

```
<%= JavaExpression %>
```

```
<%= (2*5) %>    //note no ; at end of statement.
```

4. JSP directives

The **jsp directives** are messages that tells the web container how to translate a JSP page into the corresponding servlet.

Syntax `<%@ directive attribute="value" %>`

There are three types of directives:

- A. import directive
- B. include directive
- C. taglib directive

a. **import**

The import attribute is used to import class, interface or all the members of a package. It is similar to import keyword in java class or interface.

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

Today is: `<%= new Date() %>`

b. **Include Directive**

The *include* directive tells the Web Container to copy everything in the included file and paste it into current JSP file. The include directive is used to include the contents of any resource it may be jsp file, html file or text file. Syntax of **include** directive is:

```
<%@ include file="filename.jsp" %>
```

c. **JSP Taglib directive**

The JavaServer Pages API allow you to define custom JSP tags that look like HTML or XML tags and a tag library is a set of user-defined tags that implement custom behavior. The **taglib** directive declares that your JSP page uses a set of custom tags, identifies the location of the library, and provides means for identifying the custom tags in your JSP page.

syntax `<%@ taglib uri = "uri" prefix = "prefixOfTag" >`

For example, suppose the **custlib** tag library contains a tag called **hello**. If you wanted to use the hello tag with a prefix of **mytag**, your tag would be `<mytag:hello>` and it will be used in your JSP file as follows

```
<%@ taglib uri = "http://www.example.com/custlib" prefix = "mytag" %>
```

```
<html>
```

```
    <mytag:hello/>
```

```
  </body>
```

```
</html>
```

5. JSP Comments

JSP comment marks text or statements that the JSP container should ignore.

syntax of the JSP comments `<%- - This is JSP comment - - %>`

Request String

- The query string contains the attribute & the value of the html form or JSP form, which sends with the POST /GET method to Servlet or JSP page for any request.

- A query string is the part of a URL which is attached to the end, after the file name. It begins with a question mark and usually includes information in pairs. The format is parameter=value, as in the following example:
`www.mediacollege.com/cgi-bin/myscript.cgi?name="Umesh"`
- Query strings can contain multiple sets of parameters, separated by an ampersand (&) like so:
`www.mediacollege.com/cgi-bin/myscript.cgi?fname="Umesh"&lname="M"`
- to parse this info we use method of JSP **request** object
- we can easily get using **request.getParameter()** with name of parameter as argument, to get its value.

index.html

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

welcome.jsp

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

welcome.jsp

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

    <a href="second.jsp">second jsp page</a>

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

Following are the JSP implicit object

Implicit Object	Description
request	The HttpServletRequest object associated with the request.
response	The HttpServletResponse object associated with the response that is sent back to the browser.
out	The JspWriter object associated with the output stream of the response.
session	The HttpSession object associated with the session for the given user of request.

Session implicit object (concept same as servlet)

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

Example of session implicit object

index.html

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

```
</body>
</html>
```

second.jsp

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

*//other methods of **Session** what u used in servlets can be used here.*

Cookie (concept same as servlet)

Step 1: Creating a Cookie object

Cookie cookie = new Cookie("key","value");

Step 2: Setting the maximum age

You use **setMaxAge** to specify how long (in seconds) the cookie should be valid. The following code will set up a cookie for 24 hours.

cookie.setMaxAge(60*60*24);

Step 3: Sending the Cookie into the HTTP response headers

You use **response.addCookie** to add cookies in the HTTP response header as follows
response.addCookie(cookie);

```
<%
    // Create cookies for first and last names.
    Cookie firstName = new Cookie("first_name", request.getParameter("first_name"));
    Cookie lastName = new Cookie("last_name", request.getParameter("last_name"));

    // Set expiry date after 24 Hrs for both the cookies.
    firstName.setMaxAge(60*60*24);
    lastName.setMaxAge(60*60*24);

    // Add both the cookies in the response header.
    response.addCookie( firstName );
    response.addCookie( lastName );
%>

<html>
<body>
    p><b>First Name:</b>
        <%= request.getParameter("first_name")%>
    </p>
    p><b>Last Name:</b>
```

```
        <%= request.getParameter("last_name")%>
    </p>
</body>
</html>
```

Reading Cookies with JSP

```
<html>
<body>
    <%
        Cookie cookie = null;
        Cookie[] cookies = null;

        // Get an array of Cookies associated with the this domain
        cookies = request.getCookies();

        if( cookies != null ) {
            out.println("<h2> Found Cookies Name and Value</h2>");

            for (int i = 0; i < cookies.length; i++) {
                cookie = cookies[i];
                out.print("Name : " + cookie.getName() + ", ");
                out.print("Value: " + cookie.getValue() + " <br/>");
            }
        } else {
            out.println("<h2>No cookies founds</h2>");
        }
    %>
</body>
```

```
</html>
```

//other methods of **Cookie** what u used in servlets can be used here.