Servlets

Presented By: Pankaj Kumar,
Assistant Professor-ISE, AIT

Servlet

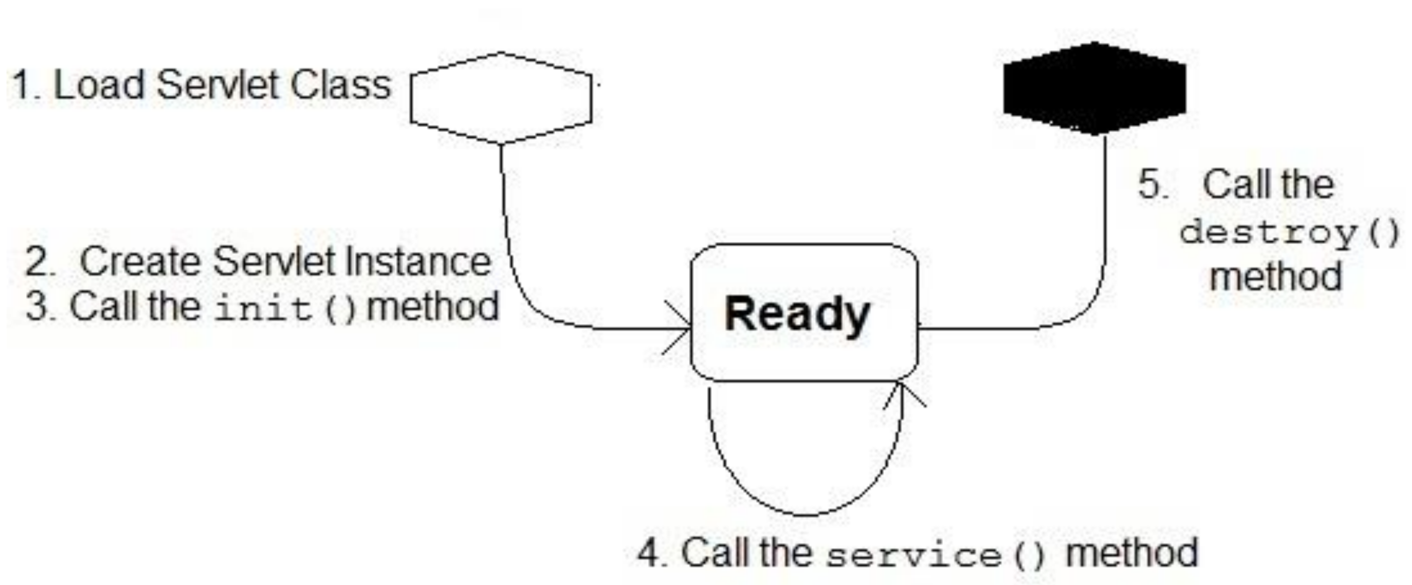
- **Servlet** technology is used to create web application
- Is an API that provides many interfaces and classes.
- Is an interface that must be implemented for creating any Servlet.
- Is a class that extends the capabilities of the servers and responds to the incoming requests. It can respond to any requests.

Life Cycle of a Servlet

The following steps show the different methods used in Servlet Life Cycle.

1. Loading the Servlet Class
2. Creating Servlet Instance
3. init Method
4. service Method
5. destroy Method

Figure below demonstrate the Servlet Life Cycle structure


1. Load Servlet Class
2. Create Servlet Instance
3. Call the init() method
Ready
5. Call the destroy() method
4. Call the service() method

Methods of life cycle of servlet

- ➢**Loading The Servlet Class:** When the web container gets the request from the browser for servlet then servlet class will be loaded.
- ➢**Creating Servlet Instance:** After the Servlet class is loaded, Web Container creates the instance (object) of it. Servlet instance is created only once in the life cycle.
- ➢**init Method:** This method will be called by the web container after an object was created to the servlet. The functionality of the init() method is to initialize the servlet.

  Syntax: **public void** init(ServletConfig config) **throws** ServletException

Contd….

➢**Service Method:** The web container calls the service method each time when request for the servlet is received

public void service(ServletRequest request, ServletResponse response) **throws** ServletException, IOException

➢**Destroy Method:** Called by the web container to indicate to a servlet that the servlet is being taken out of service

Syntax:**public void** destroy()

A Simple Servlet

Presented By: Pankaj Kumar,
Assistant Professor, ISE AIT

The basic steps are the following

1. Create and compile the servlet source code. Then, copy the servlet’s class file to the proper directory, and add the servlet’s name and mappings to the proper **web.xml** file.

2. Start Tomcat.

3. Start a web browser and request the servlet.

1. Create and Compile the Servlet Source Code

```
import java.io.*;

import javax.servlet.*;

public class HelloServlet extends GenericServlet {

public void service(ServletRequest request, ServletResponse response)

throws ServletException, IOException

{

response.setContentType("text/html");

PrintWriter pw = response.getWriter();

pw.println("<B>Hello!");

pw.close();   } }
```

2. Start Tomcat

Tomcat must be running before you try to execute a servlet.

3. **Start a Web Browser and Request the Servlet**

Start a web browser and enter the URL shown here:

http://localhost:8080/servlets-examples/servlet/HelloServlet

Alternatively, you may enter the URL shown here:

http://127.0.0.1:8080/servlets-examples/servlet/HelloServlet

The javax.servlet Package

Presented By: Pankaj Kumar,
Assistant Professor, ISE AIT

javax.sevlet package

- Contain the number of classes and interfaces that establish the framework in which servlet operates

Interfaces provided in the javax.servlet package

Interface	Description
Servlet	Declares life cycle methods for a servlet.
ServletConfig	Allows servlets to get initialization parameters.
ServletContext	Enables servlets to log events and access information about their environment.
ServletRequest	Used to read data from a client request.
ServletResponse	Used to write data to a client response.

classes provided in the javax.servlet package:

Classes	**Description**
GenericServlet	Implements the Servlet and ServletConfig interfaces.
ServletInputStream	Provides an input stream for reading requests from a client.
ServletOutputStream	Provides an output stream for writing responses to a client.
ServletException	Indicates a servlet error occurred.
UnavailableException	Indicates a servlet is unavailable.

Servlet Interface

- All servlets must implement the **Servlet** interface.
- Provides 3 life cycle methods that are used to
    1. initialize the servlet (init())
    2. to service the requests (service())
    3. to destroy the servlet (destroy())

Methods of Servlet interface

Method	Description
void init(ServletConfig config)	initializes the servlet. It is the life cycle method of servlet and invoked by the web container only once.
void service(ServletRequest request,ServletResponse response)	provides response for the incoming request. It is invoked at each request by the web container.
void destroy()	is invoked only once and indicates that servlet is being destroyed.
ServletConfig getServletConfig()	returns the object of ServletConfig.
String getServletInfo()	returns information about servlet such as writer, copyright, version etc.

GenericServlet class

- **GenericServlet** class implements **Servlet**, **ServletConfig**
- Provides the implementation of all the methods of these interfaces except the service method.
- create a generic servlet by inheriting the GenericServlet class and providing the implementation of the service method.

Reading Servlet Parameters

- The **ServletRequest** interface includes methods that allow you to read the names and values of parameters that are included in a client request

- ❖**String getParameter(String pname)** : Returns the value of the parameter named pname.
- ❖**Enumeration getParameterNames( )** : Returns an enumeration of the parameter names for this request.
- ❖**String[ ] getParameterValues(String name)** :Returns an array containing values associated with the parameter specified by name.

Example

```
<html>
<body>
<center>
<form action=“Display">
<table>
<tr>
<td><B>Employee</td>
<td><input type=textbox name="e" size="25" value=""></td>
</tr>
<tr>
<td><B>Phone</td>
<td><input type=textbox name="p" size="25" value=""></td>
</tr>
</table>
<input type=submit value="Submit">
</body>
</html>
```

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

public class PostParametersServlet
extends GenericServlet {

public void service(ServletRequest request,
ServletResponse response)
throws ServletException, IOException {

PrintWriter pw = response.getWriter();

Enumeration e = request.getParameterNames();
while(e.hasMoreElements()) {
String pname = (String)e.nextElement();
pw.print(pname + " = ");
String pvalue = request.getParameter(pname);
pw.println(pvalue);
}
pw.close();
}
}
```

javax.servlet.http Package

Presented By: Pankaj Kumar,
Assistant Professor, ISE, AIT

Interfaces that are provided in javax.servlet.http

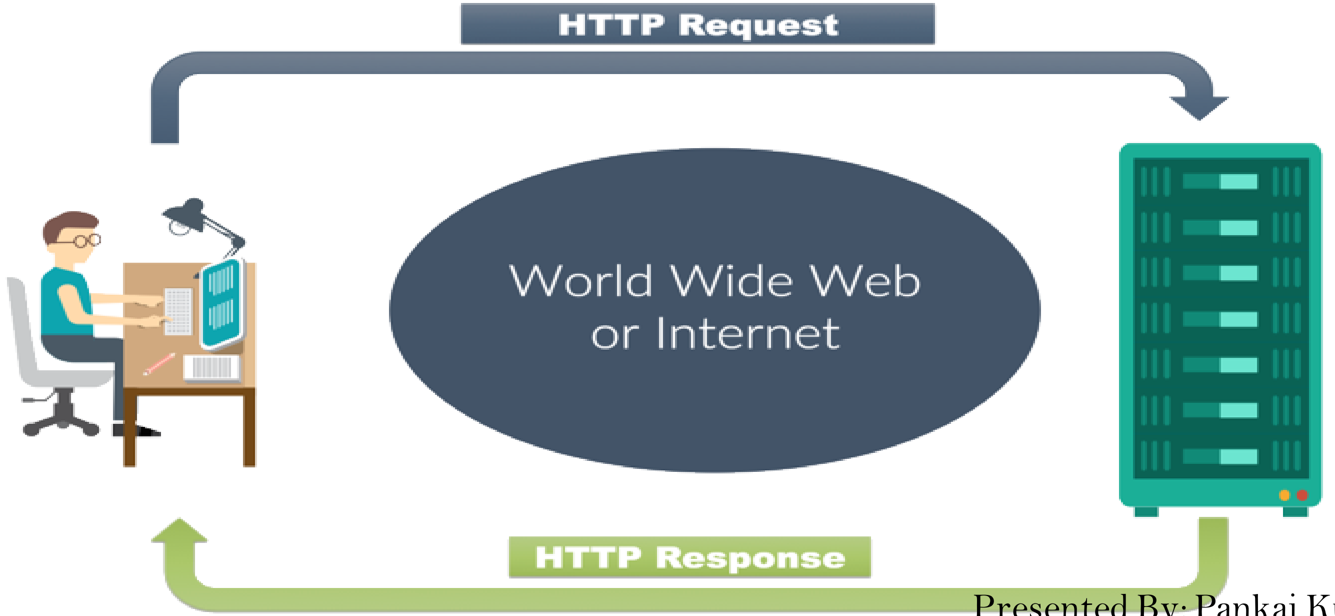
Interface	**Description**
HttpServletRequest	Enables servlets to read data from an HTTP request.
HttpServletResponse	Enables servlets to write data to an HTTP response.
HttpSession	Allows session data to be read and written.
HttpSessionBindingListener	Informs an object that it is bound to or unbound from a session.

Classes that are provided in javax.servlet.http

Class	**Description**
Cookie	Allows state information to be stored on a client machine.
HttpServlet	Provides methods to handle HTTP requests and responses.
HttpSessionEvent	Encapsulates a session-changed event.
HttpSessionBindingEvent	Indicates when a listener is bound to or unbound from a session value, or that a session attribute changed.

What is HTTP?

- ❖HTTP stands for **H**yper**T**ext **T**ransfer **P**rotocol.
- ❖This is a basis for data communication in the internet.
- ❖The data communication starts with a request sent from a client and ends with the response received from a web server.



Handling HTTP Requests and Responses

- A servlet developer typically overrides one of these methods. These methods are
- **doDelete( )**, **doGet( )**, **doHead( )**, **doOptions( )**, **doPost( )**, **doPut( )**, and **doTrace( )**

```
<html>
<body>
<center>
<form name="Form1“ action=“ABC">
<B>Color:</B>
<select name="color" size="1">
<option value="Red">Red</option>
<option value="Green">Green</option>
<option value="Blue">Blue</option>
</select>
<br><br>
<input type=submit value="Submit">
</form>
</body>
</html>
```

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

public class ColorGetServlet extends HttpServlet {

public void doGet(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException
{
String color = request.getParameter("color");
response.setContentType("text/html");
PrintWriter pw = response.getWriter();
pw.println("<B>The selected color is: ");
pw.println(color);
pw.close();
} }
```

- Parameters for an HTTP GET request are included as part of the URL that is sent to the web server.
- Assume that the user selects the red option and submits the form.
- The URL sent from the browser to the server is

http://localhost:8080/examples/servlets/servlet/ColorGetServlet?color=Red

Handling HTTP POST Requests

```
<html>
<body>
<center>
<form name="Form1“ method="post“ action=“display">
<B>Color:</B>
<select name="color" size="1">
<option value="Red">Red</option>
<option value="Green">Green</option>
<option value="Blue">Blue</option>
</select>
<br><br>
<input type=submit value="Submit">
</form> </body> </html>
```

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class ColorPostServlet extends HttpServlet {

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

String color = request.getParameter("color");
response.setContentType("text/html");
PrintWriter pw = response.getWriter();
pw.println("<B>The selected color is: ");
pw.println(color);
pw.close();
} }
```

GET	**POST**
In case of Get request, only limited amount of data can be sent because data is sent in header.	In case of post request, large amount of data can be sent because data is sent in body.
Get request is not secured because data is exposed in URL bar.	Post request is secured because data is not exposed in URL bar.
Get request can be bookmarked. Record the address of (a website, file, etc.) to enable quick access in future.	Post request cannot be bookmarked.
Get request is idempotent . It means second request will be ignored until response of first request is delivered	Post request is non-idempotent.
Get request is more efficient and used more than Post.	Post request is less efficient and used less than get.

Cookies

- ➢Is a small piece of information that is stored on the client machine.
- ➢Cookies are valuable for tracing the user activities.
- ➢Has a name, a single value, and optional attributes such as a comment, path, a maximum age, and a version number

How cookie works?

- ➢As HTTP is a stateless protocol(no session information is retained by the receiver,) so there is no way to identify that it is a new user or previous user for every new request.
- ➢In case of cookie a text file with small piece of information is added to the response of first request. They are stored on client's machine.
- ➢Now when a new request comes cookie is by default added with the request.
- ➢With this information we can identify that it is a new user or a previous user

Methods of Cookie Class

Methods	Description
public String getName( )	Returns the name of the cookies.
public String getPath( )	Returns the path of the server to which the browser returns the cookie.
public String getValue( )	Returns the value of the cookie.
public int getMaxAge( )	Returns the maximum age limit to the cookie, specified in seconds.
public void setMaxAge(int expiry)	Sets the maximum age of the cookies in seconds.
public void setValue(String newValue)	Allocates a new value to a cookie after the cookie is created.

How to create cookie?

```
//create cookie object

Cookie cookie=new Cookie(“cookieName”,”cookieValue”);

//add cookie object in the response

response.addCookie(cookie)
```

How to get cookie?

```
//get all cookie objects.
Cookie[]  cookies  = request.getCookies();
//iterate cookies array to get individual cookie objects.
for(Cookie  cookie  : cookies){
 out.println("Cookie  Name:  " + cookie.getName());
 out.println("Cookie  Value:  " + cookie.getValue());
```

Example

```
<html>

<body>

<center>

<form name="Form1“ method="post“ action=“abc">

<B>Enter a value for MyCookie:</B>

<input type=textbox name="data" size=25 value="">

<input type=submit value="Submit">

</form>

</body>

</html>
```

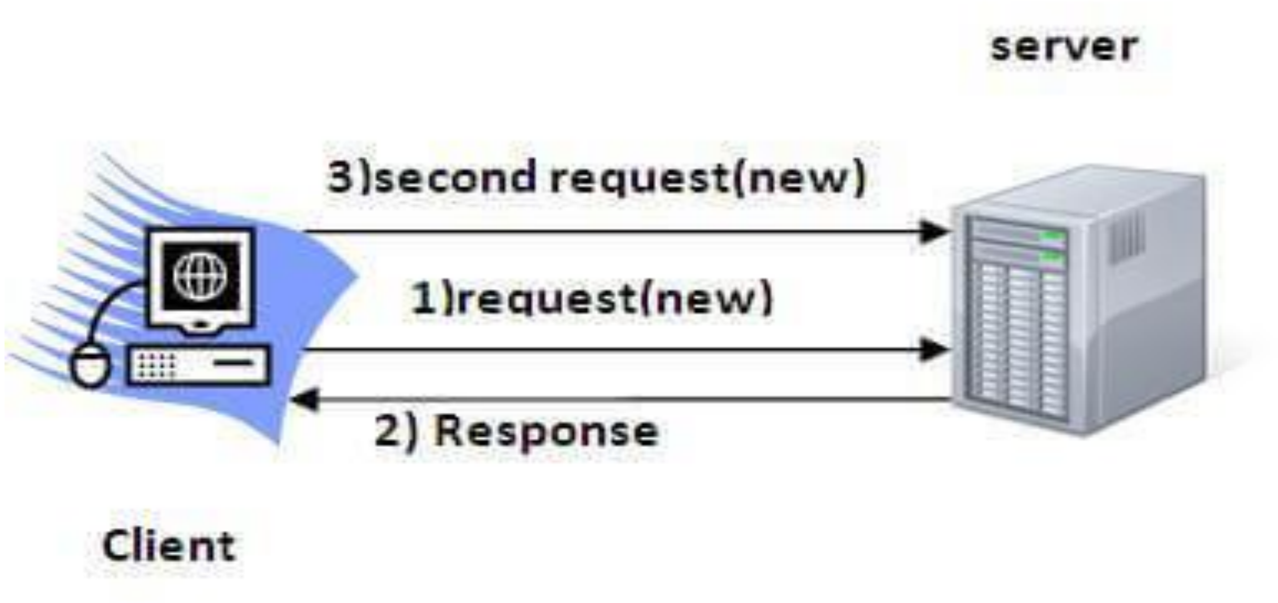
```
public class AddCookieServlet extends HttpServlet {

public void doPost(HttpServletRequest  request, HttpServletResponse  response)

throws  ServletException,  IOException  {

String  data = request.getParameter("data");                    // Get parameter from HTTP request.

Cookie cookie = new Cookie("MyCookie",  data);                   // Create cookie.

 response.addCookie(cookie);                                     // Add cookie to HTTP response

 response.setContentType("text/html");                           // Write  output to browser.

PrintWriter  pw = response.getWriter();

pw.println("<B>MyCookie  has been set to");

pw.println(data);

pw.close();

} }
```

```
public class GetCookiesServlet extends HttpServlet {
public void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException,
IOException {
Cookie[] cookies = request.getCookies();        // Get cookies from header of HTTP request.
response.setContentType("text/html");           // Display these cookies.
PrintWriter pw = response.getWriter();
pw.println("<B>");
for(int i = 0; i < cookies.length; i++) {
String name = cookies[i].getName();
String value = cookies[i].getValue();
pw.println("name = " + name + "; value = " + value);  }
pw.close();
} }
```

Session Tracking

- **Session** 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.

Each request is considered as the new request. It is shown in the figure given below:



```
public class DateServlet extends HttpServlet {
public void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException,
IOException {
HttpSession hs = request.getSession(true);          // Get the HttpSession object.
response.setContentType("text/html");               // Get writer.
PrintWriter pw = response.getWriter();
pw.print("<B>");
Date date = (Date)hs.getAttribute("date");          // Display date/time of last access.
if(date != null) {
pw.print("Last access: " + date + "<br>");
}
date = new Date();                                      // Display current date/time.
hs.setAttribute("date", date);
pw.println("Current date: " + date);
}   }
```

Java Server Pages (JSP)

Presented By: Pankaj Kumar,

Java Server Pages (JSP):

- Server side technology.
- Used for creating web application.
- JSP tags are used to insert JAVA code into HTML pages

JSP Tags

Presented By: Pankaj Kumar,
Assistant Professor, ISE AIT

JSP Tags

There are Five JSP tags

Syntax Name	**Code**
Scriptlet Tag	<%statement or java code%>
Declaration tag	<%! field or method declaration %>
Expression tag	<% = Statement %>
Comment tag	<%-----comments------ %>
Directive tag	<%@ directive name %>

Scriptlet tag

- ❖ Used to execute java source code in JSP
- ❖ Contain any number of JAVA language statements, variable or method declarations, or expressions that are valid in the page scripting language.
- ❖ syntax of Scriptlet

<% statement or java code%>

- ❖ **Example:** <% out.print("welcome to jsp"); %>

Expression tag

- ❖ Used to print out java language expression that is put between the tags.
- ❖ Can hold any java language expression that can be used as an argument to the **out.print()** method.
- ❖ Syntax of Expression Tag: **<%= statement %>**
- ❖ Example: **<%= (2*5) %>** // writes out.print((2*5));

Declaration tag

- Declaration tag is used *to declare fields and methods*.
- Can declare static member, instance variable and methods inside **Declaration Tag**
- Declaration is made inside the Servlet class but outside the service
- Syntax: **<%! field or method declaration %>**

Example for Declaration tag

```
<html>

<body>

<%! int data=50; %>

<%= "Value of the variable is:"+data %>

</body>

</html>
```

Directive Tag

- ❖Used primarily to import packages
- ❖Can use these tags to define error handling pages and for session information of JSP page.
- ❖Directive tags are of three types: **page**, **include** and **taglib**.

Directive	**Description**
<%@ page ... %>	defines page dependent properties such as language, session, errorPage etc.
<%@ include ... %>	defines file to be included.
<%@ taglib ... %>	declares tag library used in the page

Comment tag

- Describes the functionality of statements that follows the comment tag
- Syntax : **<% -- Comments %>**

Variables and objects

Presented By: Pankaj Kumar,
Assistant Professor, ISE AIT

Variables and objects

- ❖In jsp when we have to create a method or variable we usually declare it inside the declaration tag.
- ❖If we declare it inside the declaration directive then then the scope of the variables will be applicable in the whole page.
- ❖Works like a instance variable

Declaring and using a variable

```
<html>
<head>
<title>Declaration Tag</title>
</head>
<body>
<%! int age =10; %>
<% out.println("The age is " +age); %>
<p>your age is<%=age %></p>
</body>
</html>
```

Declaring objects and arrays within a single jsp tag

```
<html>
<head>
<title> JSP programing </title>
</head>
<body>
        <%! String name;
        String [] telephone = {"080-123456" ,"08158-23483" };
        String company == new String();
        Vector assignments = new Vector();
        int [] Grade = { 100,82,93  };       %>
</body>
 </html>
```

Methods

```
<html>
<head>
<title> JSP programing </title>
</head>
<body>
        <%! int curve (int grade)
                { return 10+grade; }
        %>
        <p> your curved grade is : <%= curve(80) %> </p>
</body>
 </html>
```

Control statement

```
<%! int day = 3; %>
<%
switch(day) {
case 0:
   <p> It\'s Sunday </p>
   break;
case 1:
   <p> It\'s Monday. </p>
   break;
case 2:
   <p> It\'s Tuesday. </p>
   break;
case 3:
   <p> It\'s Wednesday. </p>
   break;
case 4:
   <p> It\'s Thursday. </p>
   break;
case 5:
   <p> It\'s Friday. </p>
   break;
default:
   <p> It's Saturday. </p>
}
%>
```

Loops

```
<html>
<head>
<title>Jsp Example</title>
</head>
<body>
<%
for(int i=0;i<5;i++)
{
    for(int j=1;j<=i+1;j++)
    {
    out.print(j);
}
    out.print("<br>");
}
%>
</body>
</html>
```

Requesting a String

- Browser generates a user request string whenever the submit button is selected.
- The user request string consists of URL and query String

http://www.jimkeogh.com/jsp/myprogram.jsp?fname=“abc” &lname=“xyz”

```
<%!
 String FirstName= request.getParameter(fname);
String FirstName= request.getParameter(fname);
%>
```

User Session

- A JSP program must be able to track a session as a client moves between HTML pages and JSP Program
- There are Three commonly used method to track a session
  - By using hidden field
  - By using cookie
  - By using JavaBean

Cookies

- ❖ Cookies are the text files which are stored on the client machine.
- ❖ Used to track the information for various purposes.
- ❖ Stores various kind of information such as user preferences and ID that tracks a session with a JSP database system
- ❖ If the browser is configured to store cookies, it will keep information until expiry date.

How to create a cookie

```
<html>
<head>
<title> JSP programing </title>
</head>
<body>
        <%! String MyCookieName="userID";
            String MyCookieValue="JK1234";
            response.addCookie(new Cookie(MyCookieName, MyCookieValue));
        %>
</body>
 </html>
```

How to read a cookie

```
<html> <head>
<title> JSP programing  </title>
</head> <body>
            <%! String MyCookieName=“userID”;
                   String MyCookieValue;
                   String Cname, Cvalue;
                  int found=0;
                  Cookie[] cookies=request.getCookies();
                  for(int i=0;i<cookies.length;   i++)   {
                  Cname= cookies[i].getName();
                  Cvalue=cookies[i].getValue();
                  if(MyCookiName.equals(cookieNames[i]){
                  found=1;
                  MyCookieValue=cookieValue;      } }
                  if(found == 1) {
                  <p> Cookie Name =  <%= MyCookieName  %> </p>
                  <p> Cookie Value =  <%= MyCookieValue  %> </p>
                  }%>
</body>     </html>
```

Session Object

- ➢JSP database system is able to share information among JSP program within session by using session object.
- ➢Each time a session is created a unique ID is assigned to session and stored as cookie.
- ➢The unique ID enables JSP program to track multiple session simultaneously while maintaining data integrity of each session.

How to create a session attribute

```
<html>
<head>
<title> JSP programing </title>
</head>
<body>
        <%! String AtName=“Product”;
             String AtValue=“1234”;
        session.setAttribute(AtName, AtValue);
        %>
</body>
 </html>
```

How to read session attribute

```
html>

<head>

<title> JSP programing </title>

</head>

<body>

        <%! Enumeration purchases=session.getAttributeNames();

        While( purchases.hasMoreElements()) {

        String AtName= (String)attributeNames.nextElement();

        String AtValue=(String)session.getAttribute(AtName);      %>

        <p> Attribute Name <%= AtName %> </p>

        <p> Attribute Value <%=AtValue %> </p> <% } %>
</body>
 </html>
```