

Name of the School: School of Computer Science and Engineering

Course Code: R1UC602C

Course Name: Web Technology

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Subject Name: Web Technology

Day: 27

**Topics Covered: Java Servlet- Session
Management**

Prerequisites, Objectives and Outcomes

Prerequisite of topic: Basic concepts related to web programming

Objective: To make students aware about the session Management using Servlet.

Outcome : 1. Students will be able to use Session Management techniques.

2. Students will be able to apply session Management using different ways

3. Students will be able to implement in practical applications.

Session Tracking in Servlets

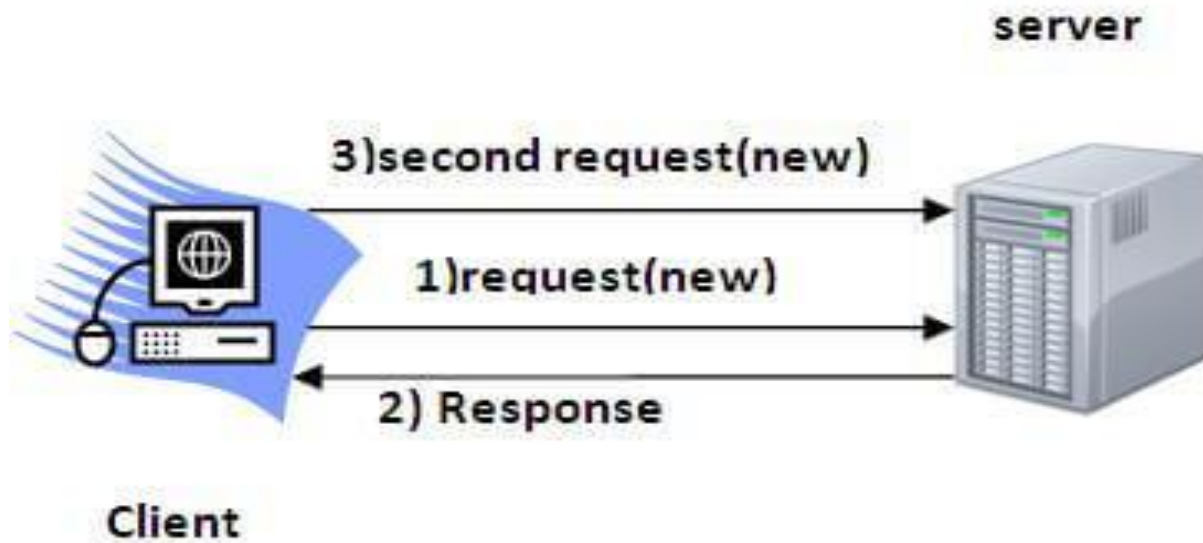
To recognize the user It is used to recognize the particular user.

- HTTP protocol and Web Servers are stateless, what it means is that for web server every request is a new request to process and they can't identify if it's coming from client that has been sending request previously.
- But sometimes in web applications, we should know who the client is and process the request accordingly. For example, a shopping cart application should know who is sending the request to add an item and in which cart the item has to be added or who is sending checkout request so that it can charge the amount to correct client.

Session Tracking

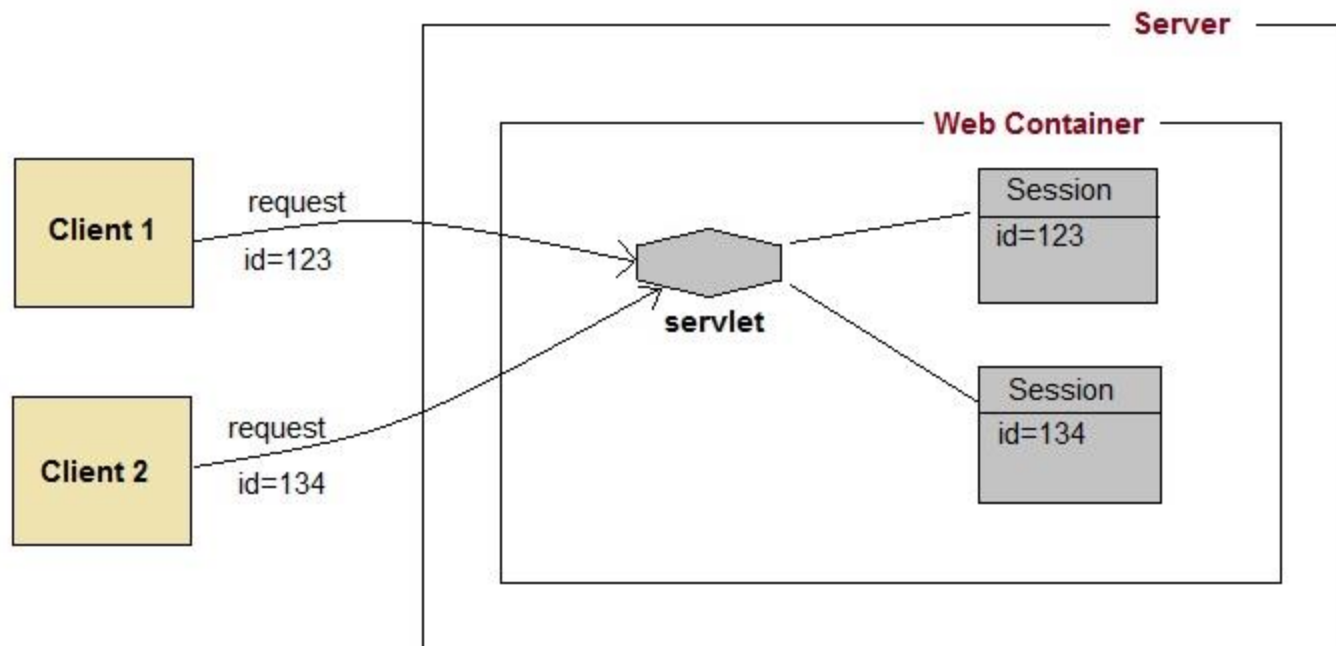
- **Session** simply means a particular interval of time.
- **Session Tracking** is a way to maintain state (data) of an user. It is also known as **session management** in servlet.
- Http protocol is a **stateless** so we need to maintain state using session tracking techniques. Each time user requests 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.

- HTTP is stateless that means each request is considered as the new request. It is shown in the figure given below:



Session Tracking Techniques

- There are four techniques used in Session tracking:
- **Cookies**
- **Hidden Form Field**
- **URL Rewriting**
- **HttpSession**

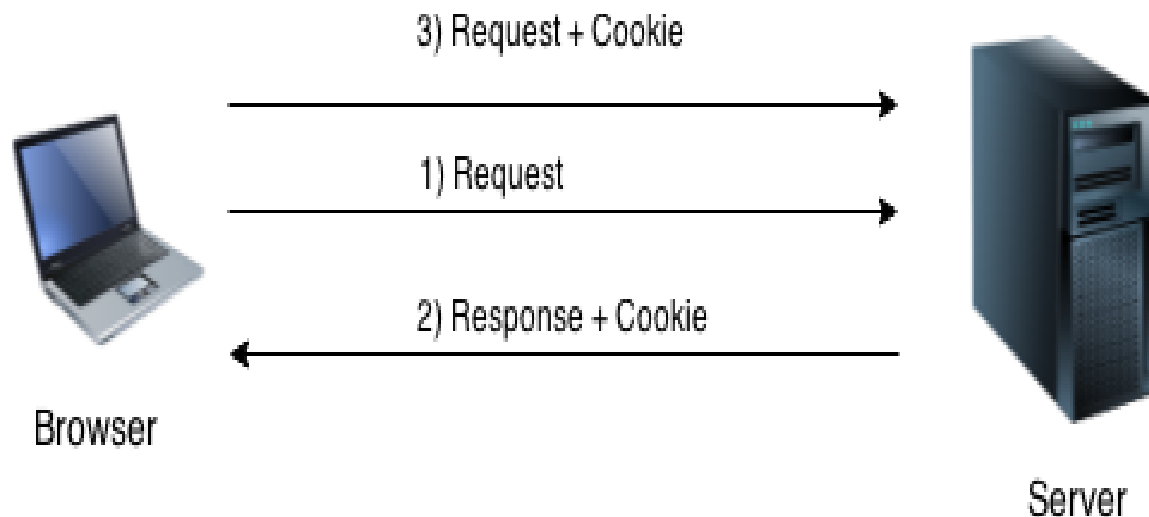


Cookies in Servlet

- A **cookie** is a small piece of information that is persisted between the multiple client requests.
- A cookie has a name, a single value, and optional attributes such as a comment, path and domain qualifiers, a maximum age, and a version number.

How Cookie works

- 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.



Types of Cookie

- There are 2 types of cookies in servlets.
 - Non-persistent cookie
 - Persistent cookie
- **Non-persistent cookie**
- It is **valid for single session** only. It is removed each time when user closes the browser.
- **Persistent cookie**
- It is **valid for multiple session** . It is not removed each time when user closes the browser. It is removed only if user logout or signout.

- Advantage of Cookies
 - Simplest technique of maintaining the state.
 - Cookies are maintained at client side.
- Disadvantage of Cookies
 - It will not work if cookie is disabled from the browser.
 - Only textual information can be set in Cookie object.
- *Note: Gmail uses cookie technique for login. If you disable the cookie, gmail won't work.*

Cookies API

- Cookies are created using **Cookie** class present in Servlet API. Cookies are added to **responseobject** using the `addCookie()` method.

This method sends cookie information over the HTTP response stream. `getCookies()` method is used to access the cookies that are added to response object.

Creating a new Cookie

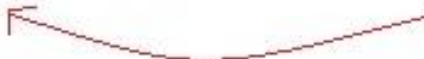
```
Cookie ck = new Cookie("username", name);
```



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

How to create Cookie?

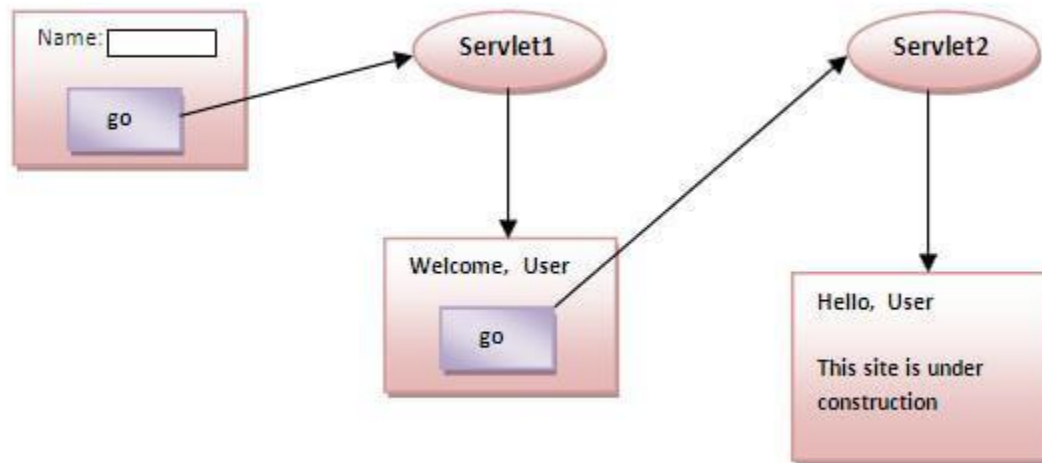
- `Cookie ck=new Cookie("user","Amit kumar");`
`//creating cookie object`
- `response.addCookie(ck);`//adding cookie in the response
- **How to delete Cookie?**
- Let's see the simple code to delete cookie. It is mainly used to logout or signout the user.
- `Cookie ck=new Cookie("user","");`//deleting value of cookie
- `ck.setMaxAge(0);` //changing the maximum age to 0 seconds
- `response.addCookie(ck);` //adding cookie in the response

How to get Cookies?

- Let's see the simple code to get all the cookies.
- `Cookie ck[]=request.getCookies();`
- `for(int i=0;i<ck.length;i++){`
- `out.print("
" + ck[i].getName() + " " + ck[i].getValue());`
- `//printing name and value of cookie`
- `}`

Simple example of Servlet Cookies

- In this example, we are storing the name of the user in the cookie object and accessing it in another servlet. As we know well that session corresponds to the particular user. So if you access it from too many browsers with different values, you will get the different value.



Servlet Program using Cookie

- index.html
- `<form action="servlet1" method="post">`
- Name:`<input type="text" name="userName"/`
`>
`
- `<input type="submit" value="go"/>`
- `</form>`

FirstServlet.java

- **import** java.io.*; **import** javax.servlet.*; **import** javax.servlet.http.*;
- **public class** FirstServlet **extends** HttpServlet {
 public void doPost(HttpServletRequest request, HttpServletResponse response){
 try{
- response.setContentType("text/html");
- PrintWriter out = response.getWriter();
- String n=request.getParameter("userName");
- out.print("Welcome "+n);
- Cookie ck=**new** Cookie("uname",n); //creating cookie object
- response.addCookie(ck); //adding cookie in the response
- //creating submit button
- out.print("<form action='servlet2'>");
- out.print("<input type='submit' value='go'>");
- out.print("</form>");
- out.close();
- **catch**(Exception e){System.out.println(e);} } }

SecondServlet.java

- **import** java.io.*;
- **import** javax.servlet.*;
- **import** javax.servlet.http.*;
- **public class** SecondServlet **extends** HttpServlet {
- **public void** doPost(HttpServletRequest request, HttpServletResponse response){
- **try**{
- response.setContentType("text/html");
- PrintWriter out = response.getWriter();
- Cookie ck[]=request.getCookies();
- out.print("Hello "+ck[0].getValue());
- out.close();
-
- **catch**(Exception e){System.out.println(e);}
- }
- }

Hidden Form Field

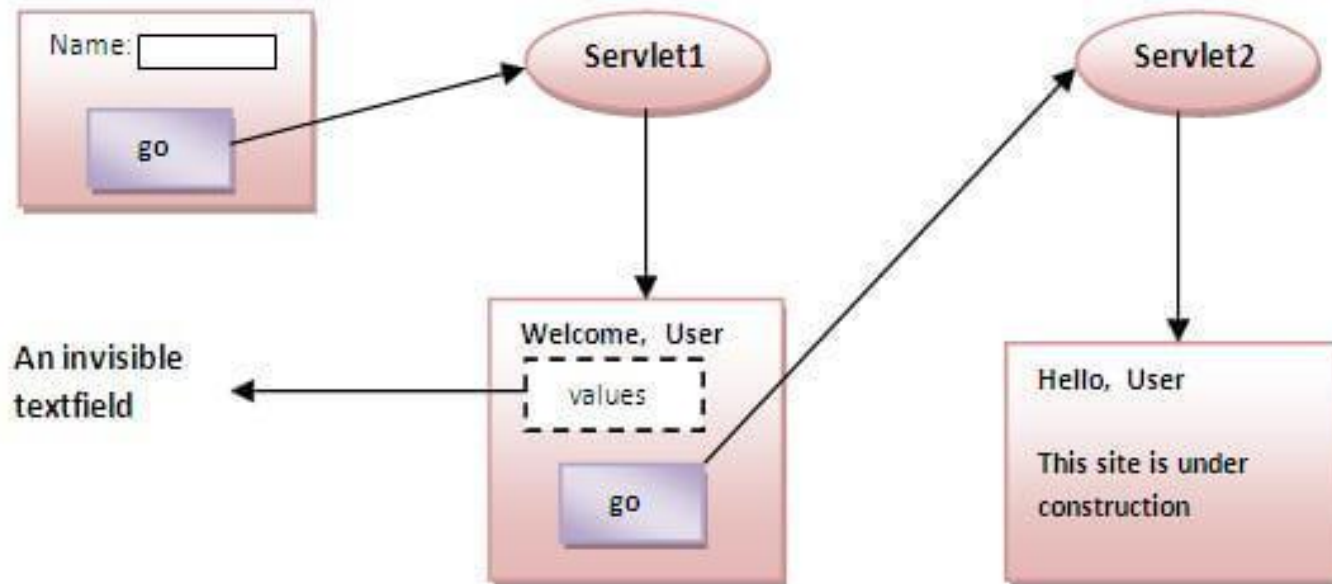
- In case of Hidden Form Field a **hidden (invisible) textfield** is used for maintaining the state of an user.
- In such case, we store the information in the hidden field and get it from another servlet. This approach is better if we have to submit form in all the pages and we don't want to depend on the browser.
- Let's see the code to store value in hidden field.
- `<input type="hidden" name="uname" value="Amit Kumar">`
- Here, uname is the hidden field name and Amit Kumar is the hidden field value.

Advantage of Hidden Form Field

- It will always work whether cookie is disabled or not.
- **Disadvantage of Hidden Form Field:**
- It is maintained at server side.
- Extra form submission is required on each pages.
- Only textual information can be used.

Example of using Hidden Form Field

- In this example, we are storing the name of the user in a hidden textfield and getting that value from another servlet.



index.html

- `<form action="servlet1">`
- Name:`<input type="text" name="userName"/>`
`
`
- `<input type="submit" value="go"/>`
- `</form>`

FirstServlet.java

- **import** java.io.*; **import** javax.servlet.*; **import** javax.servlet.http.*;
- **public class** FirstServlet **extends** HttpServlet {
- **public void** doGet(HttpServletRequest request, HttpServletResponse response){
- **try**{
- response.setContentType("text/html");
- PrintWriter out = response.getWriter();
- String n=request.getParameter("userName");
- out.print("Welcome "+n);
- //creating form that have invisible textfield
- out.print("<form action='servlet2'>");
- out.print("<input type='hidden' name='uname' value='"+n+"'>");
- out.print("<input type='submit' value='go'>");
- out.print("</form>");
- out.close();
- **catch**(Exception e){System.out.println(e);}
- **}** }

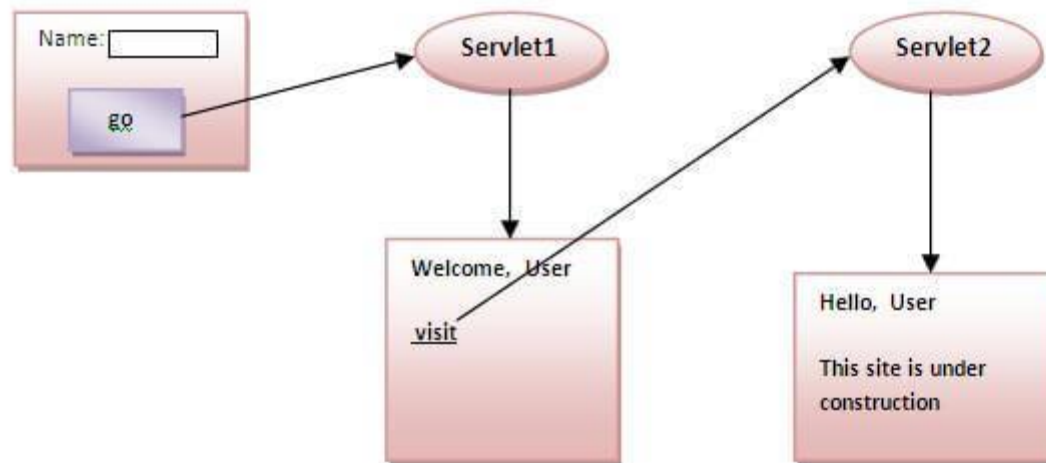
SecondServlet.java

- **import** java.io.*;
- **import** javax.servlet.*;
- **import** javax.servlet.http.*;
- **public class** SecondServlet **extends** HttpServlet {
- **public void** doGet(HttpServletRequest request, HttpServletResponse response)
- **try**{
- response.setContentType("text/html");
- PrintWriter out = response.getWriter();
- //Getting the value from the hidden field
- String n=request.getParameter("uname");
- out.print("Hello "+n);
- out.close();
- **catch**(Exception e){System.out.println(e);}
- **}** }

URL Rewriting

- In URL rewriting, we append a token or identifier to the URL of the next Servlet or the next resource. We can send parameter name/value pairs using the following format:
- `url?name1=value1&name2=value2&??`
- A name and a value is separated using an equal = sign, a parameter name/value pair is separated from another parameter using the ampersand(&). When the user clicks the hyperlink, the parameter name/value pairs will be passed to the server. From a Servlet, we can use `getParameter()` method to obtain a parameter value.
- the following format:
- `url?name1=value1&name2=value2&??`

- A name and a value is separated using an equal = sign, a parameter name/value pair is separated from another parameter using the ampersand(&). When the user clicks the hyperlink, the parameter name/value pairs will be passed to the server. From a Servlet, we can use `getParameter()` method to obtain a parameter value.



Advantage of URL Rewriting

- It will always work whether cookie is disabled or not (browser independent).
- Extra form submission is not required on each pages.
- Disadvantage of URL Rewriting
 - It will work only with links.
 - It can send Only textual information.

Example of using URL Rewriting

- In this example, we are maintaining the state of the user using link. For this purpose, we are appending the name of the user in the query string and getting the value from the query string in another page.
- index.html
- `<form action="servlet1">`
- `Name:<input type="text" name="userName"/>
`
- `<input type="submit" value="go"/>`
- `</form>`

FirstServlet.java

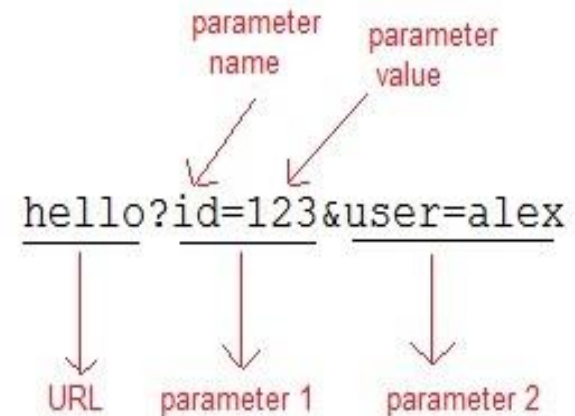
- **import** java.io.*; **import** javax.servlet.*; **import** javax.servlet.http.*;
- **public class** FirstServlet **extends** HttpServlet {
- **public void** doGet(HttpServletRequest request, HttpServletResponse response){
- **try**{
- response.setContentType("text/html");
- PrintWriter out = response.getWriter();
- String n=request.getParameter("userName");
- out.print("Welcome "+n);
- //appending the username in the query string
- out.print("visit");
- out.close();
- **catch**(Exception e){System.out.println(e);}
- }
- }

SecondServlet.java

- **import** java.io.*;
- **import** javax.servlet.*;
- **import** javax.servlet.http.*;
- **public class** SecondServlet **extends** HttpServlet {
- **public void** doGet(HttpServletRequest request, HttpServletResponse response)
- **try**{
- response.setContentType("text/html");
- PrintWriter out = response.getWriter();
-
- //getting value from the query string
- String n=request.getParameter("uname");
- out.print("Hello "+n);
- out.close();
-
- **catch**(Exception e){System.out.println(e);}
- }
- }
-

- In URL rewriting, a token(parameter) is added at the end of the URL. The token consist of name/value pair seperated by an equal(=) sign.

When the User clicks on the URL having parameters, the request goes to the **Web Container** with extra bit of information at the end of URL. The **Web Container** will fetch the extra part of the requested URL and use it for session management.



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

MyServlet.java

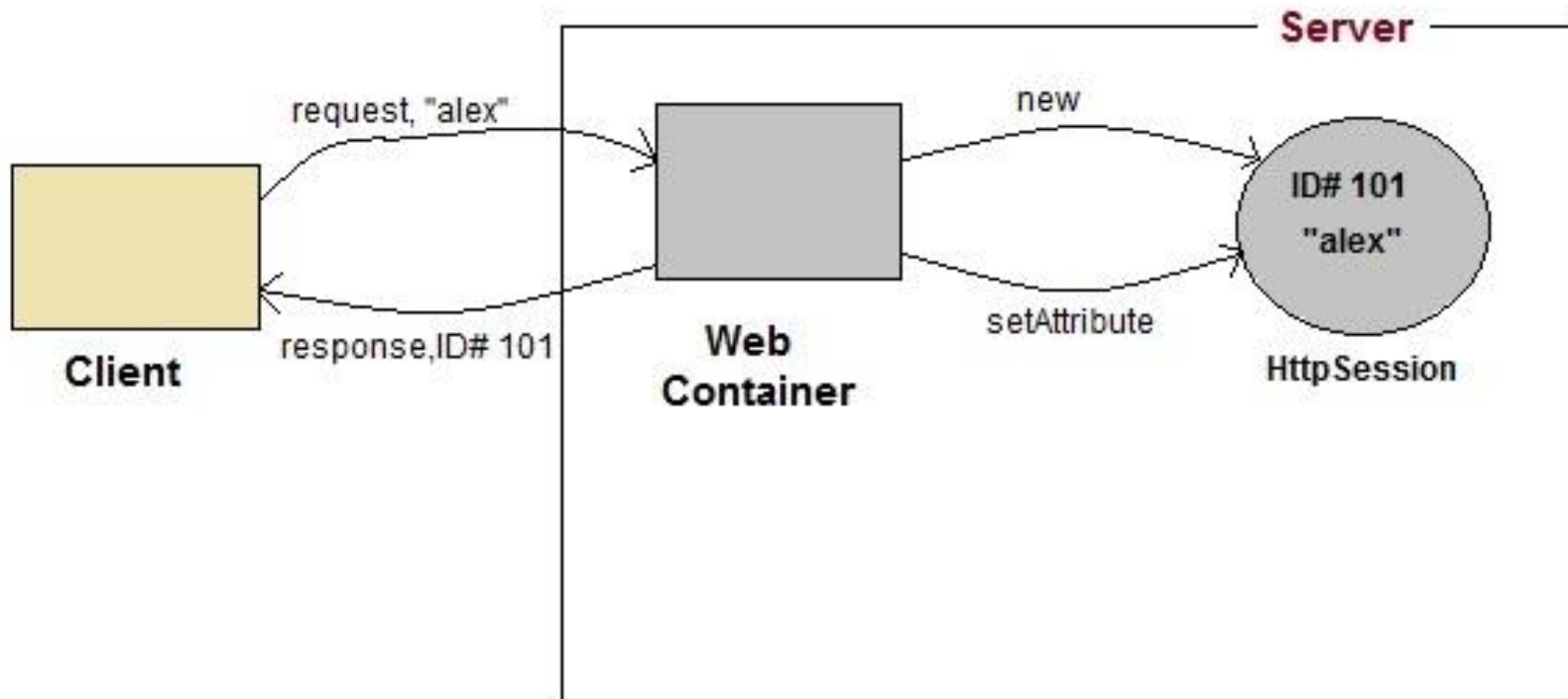
- `import java.io.*;`
- `import javax.servlet.*;`
- `import javax.servlet.http.*;`
- `public class MyServlet extends HttpServlet {`
- `protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {`
- `response.setContentType("text/html;charset=UTF-8");`
- `String name = request.getParameter("user");`
- `String pass = request.getParameter("pass");`
- `if(pass.equals("1234"))`
- `{`
- `response.sendRedirect("First?user_name="+name+"");`
- `} } }`

First.java

- `import java.io.*;`
- `import javax.servlet.*;`
- `import javax.servlet.http.*;`
- `public class First extends HttpServlet {`
- `protected void doGet(HttpServletRequest request, HttpServletResponse response)`
- `throws ServletException, IOException {`
- `response.setContentType("text/html;charset=UTF-8");`
- `PrintWriter out = response.getWriter();`
- `String user = request.getParameter("user_name");`
- `out.println("Welcome "+user);`
- `}`
- `}`

What is 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 of the **HttpServletRequest** object.
- **How HttpSession works**




- On client's first request, the **Web Container** generates a unique session ID and gives it back to the client with response. This is a temporary session created by web container.
- The client sends back the session ID with each request. Making it easier for the web container to identify where the request is coming from.
- The **Web Container** uses this ID, finds the matching session with the ID and associates the session with the request.

HttpSession Interface

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.
long getLastAccessedTime() ()	returns the last time the client sent a request associated with 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
void setMaxInactiveInterval(int interval)	Specifies the time, in seconds, after servlet container will invalidate the session.

Index.html

- `<form method="post" action="Validate">`
User: `<input type="text" name="user" />
`
Password: `<input type="text" name="pass"`
`>
`
- `<input type="submit" value="submit">`
`</form>`

Validate.java

- `import java.io.*; import javax.servlet.*;`
- `import javax.servlet.http.*;`
- `public class Validate extends HttpServlet {`
- `protected void doPost(HttpServletRequest request, HttpServletResponse response)`
- `throws ServletException, IOException {`
- `response.setContentType("text/html;charset=UTF-8");`
- `String name = request.getParameter("user");`
- `String pass = request.getParameter("pass")`
- `if(pass.equals("1234")) {`
- `//creating a session`
- `HttpSession session = request.getSession();`
- `session.setAttribute("user", name);`
- `response.sendRedirect("Welcome");`
- `} }`

Welcome.java

- import java.io.*; import javax.servlet.*;
 - import javax.servlet.http.*;
- ```
public class Welcome extends HttpServlet
{
 protected void doGet(HttpServletRequest request, HttpServletResponse
 response) throws ServletException, IOException {
 response.setContentType("text/html;charset=UTF-8");
 PrintWriter out = response.getWriter();
 HttpSession session = request.getSession();
 String user = (String)session.getAttribute("user");
 out.println("Hello "+user);
 }
}
```