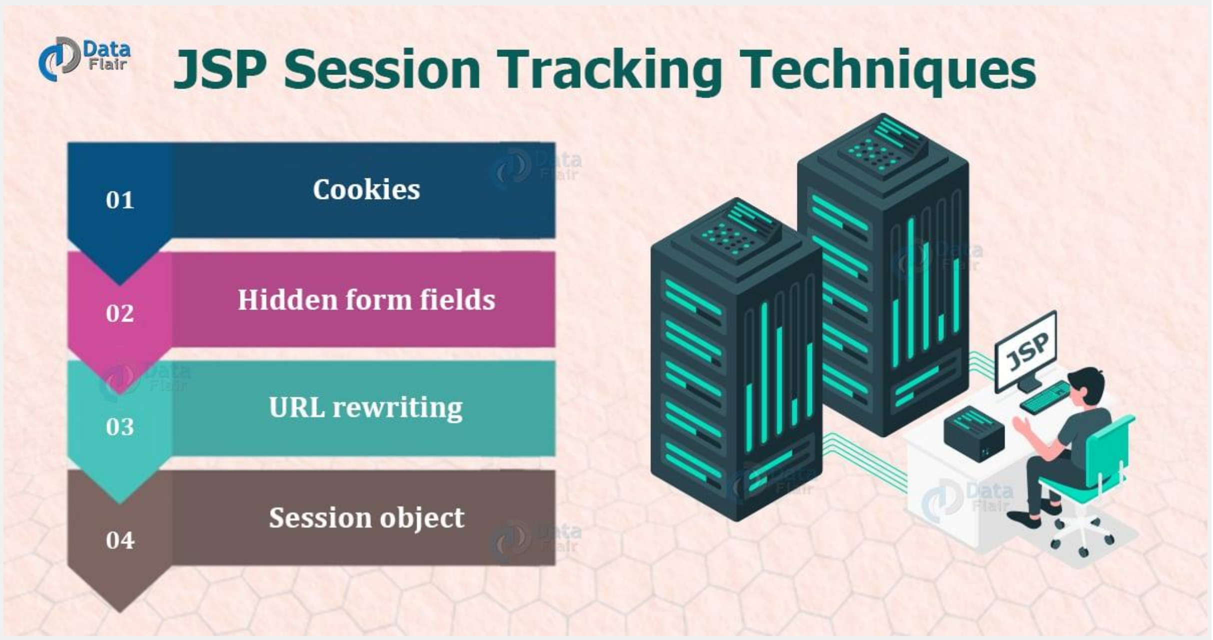
JSP Session Tracking Techniques



Introduction to JSP Session Tracking

While using a web application, a client may trigger multiple requests, or a server might be handling multiple requests from different clients at a given point of time. At times like these, a need for Session tracking arises so that the server identifies a client. It is an efficient way to maintain the state of requests that originate from the same browser for a given period of time. In order to do so, a unique session id is assigned to a user for the given session.

Need for Session Tracking in JSP

HTTP is a stateless protocol that implies that whenever a client makes a request, then for each request a new connection to the web server is established. As new connections are made every time, it becomes difficult for servers to identify that requests are coming from the same user or not. In addition to this, the server doesn’t keep the information of the previous request, if a new request is made.

For example, a client is using a shopping web application; a client may keep adding items to the cart. A server should know that if it is the same client or not. In scenarios like these session tracking is required.

To solve such problems, whenever a user introduces itself to the server, it should possess and provide a unique identifier. This will help to maintain a complete conversation between client and server. There are four methods available to maintain a session between server and client.

JSP Session Tracking Techniques

1. Cookies

Cookie is useful for Session management. The server may store a session id on the client-side as a cookie. When the method getSession() is called then this session ID can be sent by the client to the server to identify it.

```
Cookie myCookie = new Cookie("SessionID", "Session Value");
myCookie.setMaxAge(60*60*24)
Response.addCookie(myCookie);
```

Advantage: A user can be identified using this method.

Disadvantage: This method is not recommended for session tracking as it fails for the users that keep their cookies blocked as no information can be saved on the client’s side.

2. Hidden Form Fields

HTML forms may have hidden fields like:

```
<input type="hidden" name="sessionId" value="unique value"/>
```

They are similar to the field input of a form. The only difference between them is that when we submit a form, the values in the field will pass through the specified method but will not be displayed on the screen to the user. It’s value can be retrieved through request.getParameter(sessionId).

Advantage: The sessionId remains hidden and can be passed through simple HTML pages.

Disadvantage: However, we will need a logic that will dynamically generate unique values. So, this way of session tracking is not suitable for dynamic pages. Also, form submission can’t occur by clicking on a simple hypertext link. Thus this way is not much suitable.

3. URL Rewriting

We can rewrite the URL by appending the sessionId at the end of the link as:

```
http://localhost:8080/jsp/home.jsp?sessionid="ABC"
```

This will help the server identify the user with the help of a unique identifier and match it with data stored from the session. This is also called handling the browsers.

HttpServletResponse.encodeUrl will associate session id with URL.

HttpServletResponse.encodeRedirectUrl will associate session id with the redirected URL in case the user is sent to a redirected page.

Servlet engine gets and recognizes the Session ID once the user will click on the rewritten link. With this session ID, the httpSession method will get the Session ID.

Advantage: This technique is independent of the browser. Unlike the approaches above, even if the user has disabled cookies, this method is useful. We also need not pass hidden values.
Disadvantage: The client needs to keep a track of this unique identifier until the conversation between client and server completes. Thus the client needs to regenerate the URL and append the sessionId with each request made in that session.

4. Session object

Apart from the above methods, there is a method that is most reliable to use. A session begins when the client opens the browser and sends the first request to the server. The server generates a unique identifier that gets attached to the session. This sessionId and value are sent by the server to the client, and client(browser) sends it back with every request and gets uniquely identified.

Advantage: This method is the most reliable as each user possesses their unique session id. The session id will help the user to access the session data stored at the server side.
Disadvantage: This method is the most reliable method thus it has no such disadvantages.
A session is always created until it is set false in the page directive as:

<%@ page session = "**true**|**false**" %>

New session gets created with each new request using httpSession.

4.1 How to get Session object

We can access it using the implicit object of HttpServletRequest request along with the getSession() API of the HttpSession interface.

a) HttpSession session = request.getSession()
b) HttpSession session = request.getSession(Boolean)

4.2 Methods of session object

Since session object is available to JSP programmers, these methods can be used with this object:

S.No.	Method	Description
1	**public Object getAttribute(String name)**	This method will return the object bound to the session whose name is specified. It returns null if no object is found.
2	**public Enumeration getAttributeNames()**	This method will return an enumeration of all the bounded objects with the session.

3	**public long getCreationTime()**	This method will return the creation time of the session.
4	**public String getId()**	This method will return the unique identifier of the session.
5	**public long getLastAccessedTime()**	This method will return the last time of access that is the last time the client sent a request for the session.
6	**public int getMaxInactiveInterval()**	This method will return the time interval for which session will remain open between the client accesses.
7	**public void invalidate()**	This method will invalidate the session and will free the objects that are bound to it.
8	**public boolean isNew()**	This method will return true if: • Client doesn't know about the session • Or doesn't want to join the session.
9	**public void removeAttribute(String name)**	This method will remove the object of the specified name bound to the session.
10	**public void setAttribute(String name, Object value)**	This method will bind the object with the given name and value to the session.
11	**public void setMaxInactiveInterval(int interval)**	This method will set the time interval between client accesses before the session gets invalidated.

4.3 Session Tracking Example

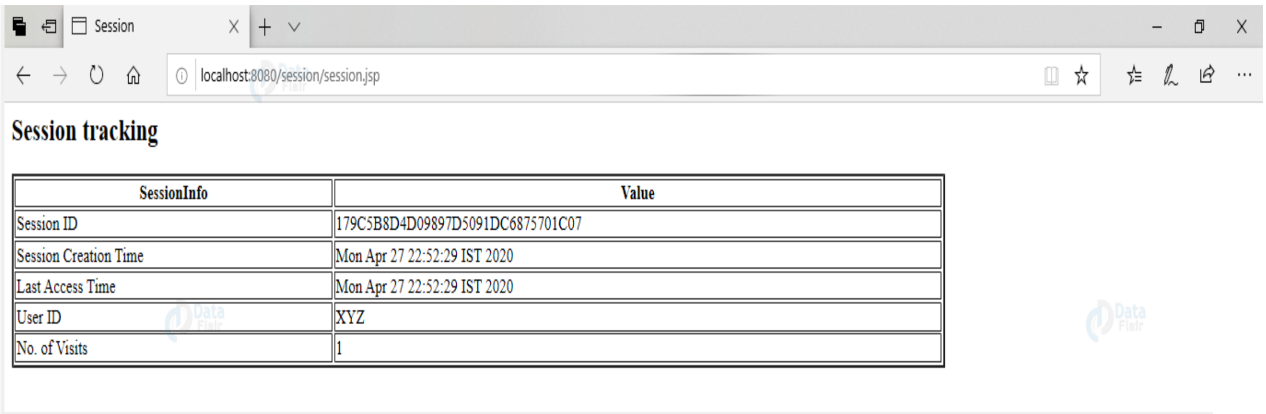
Session.jsp

```
<%@page import = "java.io.*,java.util.*"%>
<%
Date creationTime = new Date(session.getCreationTime());
Date lastaccessTime = new Date(session.getLastAccessedTime());
String title = "My website";
Integer visit = new Integer(0);
String visitCount = new String("visit");
String userID = new String("XYZ");
String userIDCount = new String("userID");
if (session.isNew())
{
title = "My website";
session.setAttribute(userIDCount, userID);
session.setAttribute(visitCount, visit);
}
visit = (Integer)session.getAttribute(visitCount);
visit = visit+1;
userID =(String)session.getAttribute(userIDCount);
session.setAttribute(visitCount, visit);
%>
<html>
```

```
<head><title>Session</title>
</head>
<body>
<h2>Session tracking</h2>
<table width = "75%" border = "2" align = "left">
<tr>
<th>SessionInfo</th>
<th>Value</th>
</tr>
<tr>
<td>Session ID</td>
<td><%out.print(session.getId());%></td>
</tr>
<tr>
<td>Session Creation Time</td>
<td><%out.print(creationTime);%></td>
</tr>
<tr>
<td>Last Access Time</td>
<td><%out.print(lastaccessTime);%></td>
</tr>
<tr>
<td>User ID</td>
<td><%out.print(userID);%></td>
</tr>
<tr>
<td>No. of Visits</td>
<td><%out.print(visit);%></td>
</tr>
</table>
</body>
</html>
```

Explanation: In the above example we are displaying the session information like sessionID, user ID, creation time, last access time and number of visits to a website using the session object of HttpSession Interface.

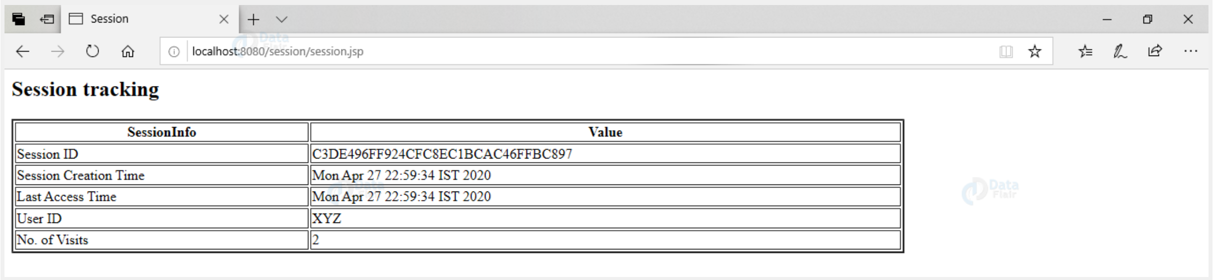
Output:



Session tracking

SessionInfo	Value
Session ID	179C5B8D4D09897D5091DC6875701C07
Session Creation Time	Mon Apr 27 22:52:29 IST 2020
Last Access Time	Mon Apr 27 22:52:29 IST 2020
User ID	XYZ
No. of Visits	1

When we load the page again the no. of visits increases by 1.



Session tracking

SessionInfo	Value
Session ID	C3DE496FF924CFC8EC1BCAC46FFBC897
Session Creation Time	Mon Apr 27 22:59:34 IST 2020
Last Access Time	Mon Apr 27 22:59:34 IST 2020
User ID	XYZ
No. of Visits	2

4.4 Invalidate Example

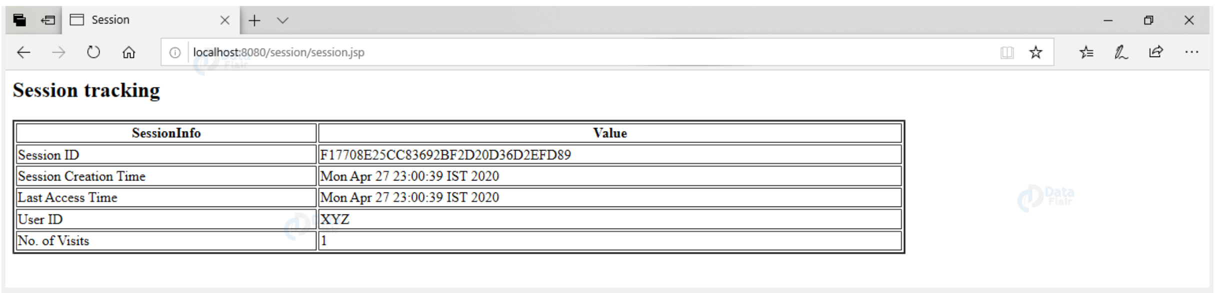
For invalidating the session we append the above code as following:

```
<%@page import = "java.io.*,java.util.*"%>
<%
Date creationTime = new Date(session.getCreationTime());
Date lastaccessTime = new Date(session.getLastAccessedTime());
String title = "My website";
Integer visit = new Integer(0);
String visitCount = new String("visit");
String userID = new String("XYZ");
String userIDCount = new String("userID");
if (session.isNew())
{
title = "My website";
session.setAttribute(userIDCount, userID);
session.setAttribute(visitCount, visit);
}
visit = (Integer)session.getAttribute(visitCount);
visit = visit+1;
userID =(String)session.getAttribute(userIDCount);
session.setAttribute(visitCount, visit);
%>
<html>
```

```
<head><title>Session</title>
</head>
<body>
<h2>Session tracking</h2>
<table width = "75%" border = "2" align = "left">
<tr>
<th>SessionInfo</th>
<th>Value</th>
</tr>
<tr>
<td>Session ID</td>
<td><%out.print(session.getId());%></td>
</tr>
<tr>
<td>Session Creation Time</td>
<td><%out.print(creationTime);%></td>
</tr>
<tr>
<td>Last Access Time</td>
<td><%out.print(lastaccessTime);%></td>
</tr>
<tr>
<td>User ID</td>
<td><%out.print(userID);%></td>
</tr>
<tr>
<td>No. of Visits</td>
<td><%out.print(visit);%></td>
</tr>
</table>
<%session.invalidate();%>
</body>
</html>
```

Explanation: The current session gets invalidated and the number of visits gets decreased by 1. Even on refreshing the page the value does not increase as the session has been invalidated and objects are free.

Output:



Session tracking

SessionInfo	Value
Session ID	F17708E25CC83692BF2D20D36D2EFD89
Session Creation Time	Mon Apr 27 23:00:39 IST 2020
Last Access Time	Mon Apr 27 23:00:39 IST 2020
User ID	XYZ
No. of Visits	1

4.5 Session Management Example

As one of the important purposes of session tracking is to remember a user, session management can increase the scope. Session management is done so that only the authenticated user can use that session. If the information given by the user matches the database then only he is allowed to go to the next page. Following is the Example:

index.jsp

```
<%@ page isErrorPage="true" %>
<html>
<head>
<title>Session Management</title>
</head>
<body>
<form action="first.jsp" method="post">
Enter your name:<input type="text" name="name"><br/>
Enter your password:<input type="password" name="password">
<br/>
<input type="submit" name="submit" value="submit">
</form>
</body>
</html>
```

first.jsp

```
<%
String name = request.getParameter("name");
String password = request.getParameter("password");
if (name.equals("dataflair") && password.equals("admin")) {
session.setAttribute("username", name);
response.sendRedirect("second.jsp");
}
else {
response.sendRedirect("index.jsp");
}
%>
```

second.jsp

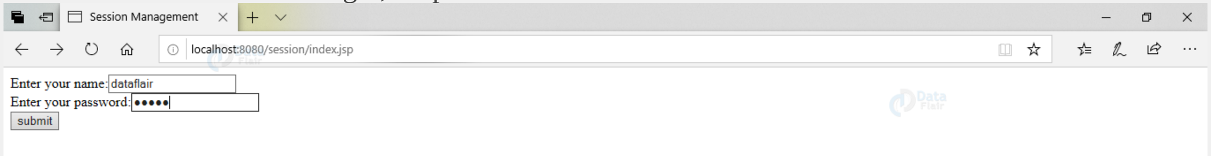
```
<html>
```

```
<head>
<title>Welcome to the session</title>
</head>
<body>
Hello
<%= session.getAttribute("username") %>
</body>
</html>
```

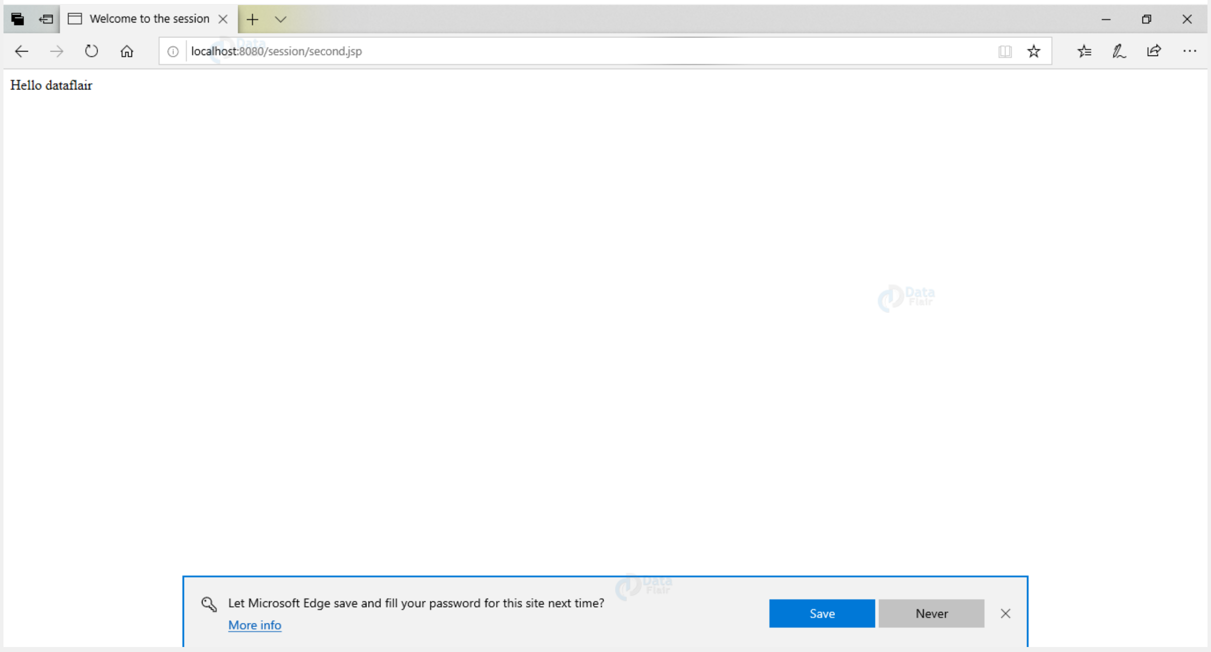
Explanation: In this example, we can see the application of session management. We get the name and password and it is set using session.setAttribute( ). If the information matches then only the user gets prompted to the next page else the **index page opens again.**

Output:

a. When the information is right, output is:



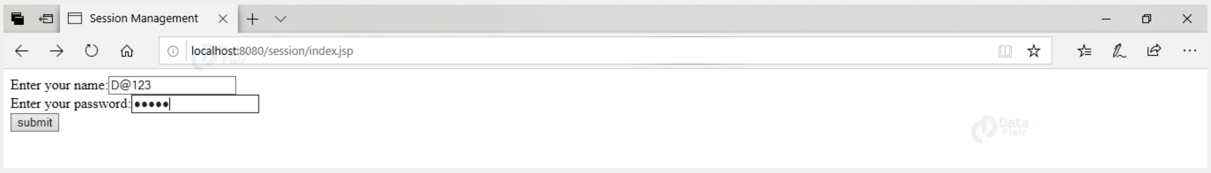
Enter your name: dataflair
Enter your password: ●●●●●



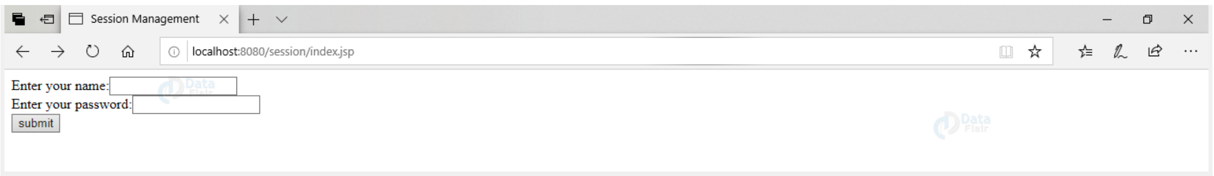
Hello dataflair

Let Microsoft Edge save and fill your password for this site next time?

b. When the information is wrong, output is:



Enter your name: D@123
Enter your password: ●●●●●



Enter your name:
Enter your password:
submit

4.6 Deleting a JSP Session

There are various options to delete a session once you are done with it. They are:

- **Remove a particular attribute** – public void removeAttribute(String name) as discussed above in methods can remove an attribute from a session
- **Delete the whole session** – As the invalidate example shown above, you can call session.invalidate() method to invalidate a session. It will free all the objects bound to the session.
- **Setting Session timeout** – Using method public void setMaxInactiveInterval(int interval) we can set the timeout for a session.
- **Log the user out** – Logging out will invalidate all the sessions related to the user.
- **web.xml Configuration** – You can use config method to set the timeout for the session if working on the tomcat platform.

4.7 Browser session vs. Server session

As the session tracking at the browser side occurs with the help of cookies, the session information will get discarded once the user closes the browser.

Unlike it, server session stores session information at the server side. This information will remain until the session invalidates, or the maximum inactive time gets over irrespective of the fact that the client has logged out of the browser or not.

4.8 Use session objects wisely

Use session objects wisely as it stores some important data.

For example, you develop a banking system that will fetch balance and update balance. Say your first page fetches balance and stores it in a variable called acc_ balance. Then the second page update balance uses this variable acc_balance and updates it.

Now let’s say another developer also works on it and uses the same name variable acc_balance and initializes it with zero. If it happens that your acc_balance variable has value 1000 when the developer calls the first page. Then a program made by another developer gets called. Now the acc_balance=0 and then update balance runs. It uses the acc_balance variable and updates it with say 100. acc_balance should have 1100 as result but it now contains 100.

Thus, use session objects wisely.