**Session Tracking In Servlet**

**Session**is the conversion of user within span of time. In general meaning particular interval of time.

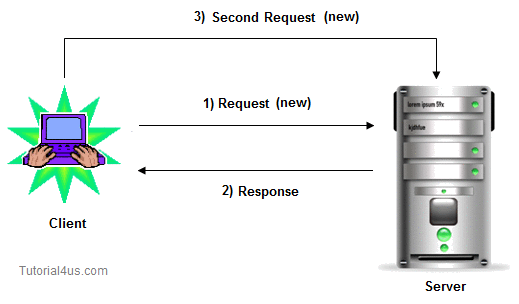
**Tracking** is the recording of the thing under session.

**Session Tracking** is remembering and recording of client conversion in span of time. It is also called as session management.

If web application is capable of remembering and recording of client conversion in span of time then that web application is called as **stateful web application**.

**Why need Session Tracking ?**

* Http protocol is stateless, to make stateful between client and server we need Session Tracking.
* Session Tracking is useful for online shopping, mailing application, E-Commerce application to track the conversion.
* Http protocol is stateless, that means each request is considered as the new request. You can see in below image.



**Why use Session Tracking ?**

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

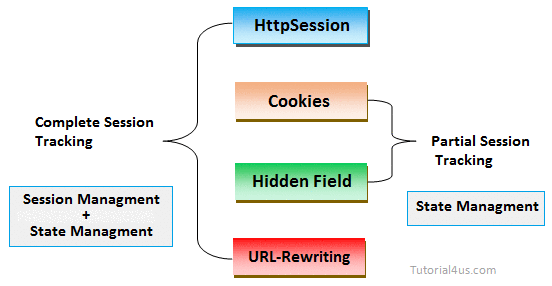
**Why Http is design as stateless protocol ?**

If Http is stateful protocol for multiple requests given by client to web application single connection will be used between browser and web server across the multiple requests. This may make clients to engage connection with web server for long time event though the connection are ideal. Due to this the web server reach to maximum connections even though most of its connection are idle. To overcome this problem Http is given as stateless.

**Session Tracking Techniques**

Servlet technology allows four technique to track conversion, they are;

* Cookies
* URL Rewriting
* Hidden Form Field
* HttpSession



**Cookies**

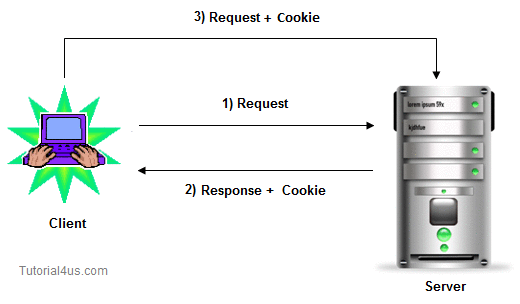
**Cookies** are text files stored on the client computer and they are kept for various information like name, a single value, and optional attributes such as a comment, path and domain qualifiers, a maximum age, and a version number.

Cookies are created using Cookie class present in Servlet API. Cookies are added to response object using the addCookie() method. This method sends cookie information over the HTTP response stream. getCookies() method is used access the cookies that are added to response object.

In Http Session technique, container internally generates a cookies for transferring the session ID between server and client. Apart from container generated cookie a servlet programmer can also generate cookies for storing the data for a client.

**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 (chrome, firefox) at client side. 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.



**When use cookies ?**

When session ID is not required and when less number of input values are submitted by client in that case in place of using HttpSession Technique you can use cookies Technique to reduce the burden on server.

**Points to Remember**

* Cookies is pressistance resource which is stores at client location.
* We can store 3000 cookies in cookies file at a time.
* The cookies are introduced by net scape communication.
* Cookies files exist up to 3 year.
* Size of cookies is 4 kb.

**Type of Coockies**

There are two types of cookies, those are given below;

* In-memory cookies or pre session cookies
* Persistent cookies

**In-memory cookies:**By default cookie is in-memory coockie, This type of cookie is lives until that browser is destroy(close). It is valid for **single session** only. It is removed each time when user closes the browser.

**Persistent cookies:**Presestent cookie lives on a browser until its expiration time is reached it means , eventhough you close or reopen the browser but still the cookie exists on the browser. 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.

**Cookie class**

**javax.servlet.http.Cookie** class provides the functionality of using cookies. It provides a some constructor and methods for cookies.

**Constructor of Cookie class**

|  |  |
| --- | --- |
| **Constructor** | **Description** |
| Cookie() | Used for constructs a cookie. |
| Cookie(String name, String value) | Used for constructs a cookie with a specified name and value. |

**Methods of Cookie class**

|  |  |
| --- | --- |
| **Methods** | **Description** |
| public void setMaxAge(int expiry) | It is used for Sets the maximum age of the cookie in seconds. |
| public String getName() | It is used for Returns the name of the cookie. The name cannot be changed after creation. |
| public String getValue() | It is used for Returns the value of the cookie. |
| public void setName(String name) | It is used for changes the name of the cookie. |
| public void setValue(String value) | It is used for changes the value of the cookie. |
| public void addCookie(Cookie ck) | It is method of HttpServletResponse interface which is used to add cookie in response object. |
| public Cookie[] getCookies() | It is method of HttpServletRequest interface which is used to return all the cookies from the browser. |

**Create Cookies**

To create cookies you need to use Cookie class of javax.servlet.http package.

**Syntax**

Cookie c=**new** Cookie(name, value);

// here name and value are string type

**Add Cookies**

To add a cookie to the response object, we use addCookie() mehtod.

**Syntax**

Cookie c=**new** Cookie(); //creating cookie object

response.addCookie(c1); //adding cookie in the response

**Read Cookies for browser**

To read Cookies from browser to a servlet, we need to call getCookies methods given by request object and it returns an array type of cookie class.

**Syntax**

response.addCookie(c1);

Cookie c[]=request.getCookie();

**Advantage of Cookie**

* Simplest technique of maintaining the state.
* Cookie are maintained at client side so they do not give any burden to server.
* All server side technology and all web server, application servers support cookies.
* Presistent cookies can remember client data during session and after session with expiry time.

**Limitation of Coockie**

* It will not work if cookie is disabled from the browser.
* Cookies are text files, It does not provides security. Any one can change this file.
* With coockies need client support that means if client disable the coockies then it does not store the client location.
* Cookies can not store java objects as values, they only store text or string.

**Example of session tracking by using Cookies**

**index.html**

**<form** action="servlet1"**>**

Name:**<input** type="text" name="userName"**/>** **<br/>**

**<input** type="submit" value="continue"**/>**

**</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='continue'>");

**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);}

}

}

**web.xml**

**<web-app>**

**<servlet>**

**<servlet-name>**s1**</servlet-name>**

**<servlet-class>**FirstServlet**</servlet-class>**

**</servlet>**

**<servlet-mapping>**

**<servlet-name>**s1**</servlet-name>**

**<url-pattern>**/servlet1**</url-pattern>**

**</servlet-mapping>**

**<servlet>**

**<servlet-name>**s2**</servlet-name>**

**<servlet-class>**SecondServlet**</servlet-class>**

**</servlet>**

**<servlet-mapping>**

**<servlet-name>**s2**</servlet-name>**

**<url-pattern>**/servlet2**</url-pattern>**

**</servlet-mapping>**

**</web-app>**

**Hidden Form Field**

Tracking client conversion using Html hidden variables in secure manner is known as hidden form field.

**How to use Hidden Form Field ?**

In Hidden Form Field we are use html tag is <input type="hidden"> and with this we assign session ID value.

**Syntax**

**<input** type="hidden" name="uname" value="porter"**>**

**Hidden Form Field Advantage**

* Basic knowledge of html is enough to work with this technique.
* It will always work whether cookie is disabled or not.
* Hidden boxes resides in web pages of browser window so they do not provide burden to the server.
* This technique can be used along with all kind of web server or application server.

**Hidden Form Field Dis-Advantage**

* More complex than URL Rewriting.
* It is maintained at server side.
* Extra form submission is required on each pages.
* Hidden form field can not store java object as values. They only store text value
* It Also increase network traffic because hidden boxes data travels over the network along with request and response.
* Hidden boxes does not provides data security because their data can be view through view source option.

**Example of session tracking by using Hidden Form Field**

**index.html**

**<form** action="servlet1"**>**

Name:**<input** type="text" name="userName"**/>** **<br/>**

**<input** type="submit" value="continue"**/>**

**</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='continue'>");

**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);}

}

}

**web.xml**

**<web-app>**

**<servlet>**

**<servlet-name>**s1**</servlet-name>**

**<servlet-class>**FirstServlet**</servlet-class>**

**</servlet>**

**<servlet-mapping>**

**<servlet-name>**s1**</servlet-name>**

**<url-pattern>**/servlet1**</url-pattern>**

**</servlet-mapping>**

**<servlet>**

**<servlet-name>**s2**</servlet-name>**

**<servlet-class>**SecondServlet**</servlet-class>**

**</servlet>**

**<servlet-mapping>**

**<servlet-name>**s2**</servlet-name>**

**<url-pattern>**/servlet2**</url-pattern>**

**</servlet-mapping>**

**</web-app>**

**URL Rewriting**

**URL Rewriting**track the conversion in server based on unique session ID value.

**When use URL Rewriting ?**

If the client has disabled cookie in the browser then coockie are not work for session management. In that case you can use URL rewriting technique for session managment. URL rewriting will always work.

**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

**Dis-advantage of URL Rewriting**

* Generate more network traffic.
* It will work only with links.
* It can send Only textual information.
* Less secqure because query string in session id displace on address bar.

**Example of session tracking by using URL Rewriting**

**index.html**

**<form** action="servlet1"**>**

Name:**<input** type="text" name="userName"**/>** **<br/>**

**<input** type="submit" value="continue"**/>**

**</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);

HttpSession session=request.getSession();

session.setAttribute("uname",n);

**out**.**print**("<a href='servlet2'>visit</a>");

**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);}

}

}

**web.xml**

**<web-app>**

**<servlet>**

**<servlet-name>**s1**</servlet-name>**

**<servlet-class>**FirstServlet**</servlet-class>**

**</servlet>**

**<servlet-mapping>**

**<servlet-name>**s1**</servlet-name>**

**<url-pattern>**/servlet1**</url-pattern>**

**</servlet-mapping>**

**<servlet>**

**<servlet-name>**s2**</servlet-name>**

**<servlet-class>**SecondServlet**</servlet-class>**

**</servlet>**

**<servlet-mapping>**

**<servlet-name>**s2**</servlet-name>**

**<url-pattern>**/servlet2**</url-pattern>**

**</servlet-mapping>**

**</web-app>**

**HttpSession**

**HttpSession**is another kind of session management technique, In this technique create a session object at server side for each client.

Session is available until the session time out, until the client log out. The default session time is 30 minutes and can configure explicit session time in web.xml file.

**Configure session time in web.xml**

**Example**

**<web-app>**

**<session-config>**

**<session-timeout>**40**</session-timeout>**

**</session-config>**

**</web-app>**

**HttpSession Api**

Http session is an interface define in java.http package.

**Getting session object**

**Example**

HttpSession hs=req.getSession(); // create new session object

**Methods of HttpSession interface**

|  |  |
| --- | --- |
| **Method** | **Description** |
| public HttpSession getSession(): | It returns the current session associated with this request, or if the request does not have a session, creates one. |
| public HttpSession getSession(boolean create) | It returns the current HttpSession associated with this request or, if there is no current session and create is true, returns a new session. |
| public String getId() | It returns a string containing the unique identifier value. |
| public long getCreationTime() | It returns the time when this session was created, measured in milliseconds since midnight January 1, 1970 GMT. |
| public long getLastAccessedTime() | It returns the last time the client sent a request associated with this session, as the number of milliseconds since midnight January 1, 1970 GMT. |
| public void invalidate() | Invalidates this session then unbinds any objects bound to it. |

**Dis-Advantage of HttpSession**

* Http session objects allocate memory on the server so this increase burden on the server.
* If cookies are restricted coming to browser window this technique fails to perform session tracking.

**Example of session tracking by using httpsession**

**index.html**

**<form** action="servlet1"**>**

Name:**<input** type="text" name="userName"**/>** **<br/>**

**<input** type="submit" value="continue"**/>**

**</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);

HttpSession session=request.getSession();

session.setAttribute("uname",n);

**out**.**print**("<a href='servlet2'>visit</a>");

**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();

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

String n=(String)session.getAttribute("uname");

**out**.**print**("Hello "+n);

**out**.close();

}**catch**(Exception e){System.**out**.println(e);}

}

}

**web.xml**

**<web-app>**

**<servlet>**

**<servlet-name>**s1**</servlet-name>**

**<servlet-class>**FirstServlet**</servlet-class>**

**</servlet>**

**<servlet-mapping>**

**<servlet-name>**s1**</servlet-name>**

**<url-pattern>**/servlet1**</url-pattern>**

**</servlet-mapping>**

**<servlet>**

**<servlet-name>**s2**</servlet-name>**

**<servlet-class>**SecondServlet**</servlet-class>**

**</servlet>**

**<servlet-mapping>**

**<servlet-name>**s2**</servlet-name>**

**<url-pattern>**/servlet2**</url-pattern>**

**</servlet-mapping>**

**</web-app>**