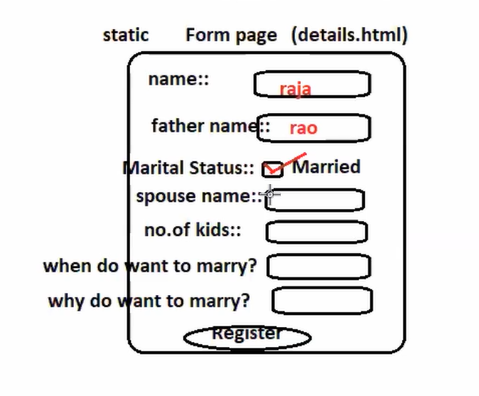
# Session Tracking

**Need of Session tracking/Management**

**Static form Page:** The form page that is generated by .html file is called static form page. In static form page components are fixed.

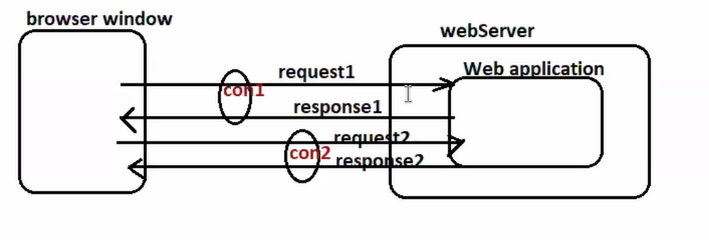
**Dynamic form Page:** The form page that is generated by servlet/jsp components as response is called dynamic form page. In dynamic form page the form components are not fixed and they can change dynamically as required.



**PROBLEM::** Gathering all the details from end user single form page is bad practice because the end user is forced to answer both related and unrelated information in this process end-user may get into confusions.

**Solution::** collect the common information using static form page and remaing details using dynamic form.

**Session:**

1. Session is set of contineous and related operation performed by the end user through browser on specific application. Example: Login to Logout in gmail, FB etc..
2. If the web application is remembering client data (enduser through browser) across the multiple request during a session then it is called state full web application.
3. If web application is not remembering client data across the multiple request during a session then it is called stateless web application.
4. By default every web application is stateless web application because the protocol http is given as stateless protocol.
   1. Protocol http is stateless because it create new connection between browser and web server for every request. Due to this we cannot use one connection data (request data) in another connection (another request)
5. To make web application as statefull though the protocol http is sateless we need to use one or another session tracking technique/ session management.

**Session tracking/ Session Management Technique:**

1. Hidden form field (Non-Standrad technique. Not part of specification.
2. Http Cookies
3. HttpSession with Cookies
4. HttpSession with ULR rewriting

All popular website like FB, Google and etc uses one or another session tracking technique so they behave like satefull web application. I.e. while processing current request they can use previous request data.

For example while processing 3rd request we can use 1st,2nd request data.

**Hidden Form Fields:**

* It is all about working with hidden boxes.
* It is a non-standard technique.

|  |  |
| --- | --- |
| **In form Page** | **In Servlet component** |
| <input type=”hidden” name=”h1” value=”hello”/> | String s1 = request.getParameter(“h1”); |

**Http Cookies/Cookies:**

* Cookies are textual information that allocate memory in client machine remembering client data across the multiple request.
* Cookies are basically created in web application at server side and comes to browser along with the response and allocate memory at client side.
* Cookies goes back to web application along with the request to make client data visible across the multiple request.
* Every cookie contains:
  + Cookie Name
  + Cookie Value
  + Domain Name and etc details.
* Cookies can carry only textual information, not the objects.
* Two types of Cookies:

1. In-Memory Cookies
2. Persistence Cookies

* **In-Memory Cookies** allocates memory in the browser memory of RAM.
* **In-Memory Cookies** will be deleted ones the browser window is closed.
* **In-Memory Cookies** do not have expiry time
* **In-Memory Cookies** are suitable for session tracking
* **Persistence Cookies** allocates memory in client machine file system having expiry time.
* **Persistence Cookies** will not be deleted once the browser window is closed, they will be deleted only when the expiry time is completed.
* **Persistence Cookies** are not suitable for session tracking, these cookies are useful to enable remember me option.

When the web component adds cookies to the response they go to browser as “set-cookie” cookie response header values.

When browser gives request back to web application the cookies representing same web application go back to web application as “cookie” request header value

**Cookie API (Working with javax.servlet.Cookie©:**

**To create cookie:** Cookie ck1 = new Cookie(“TS”,”hyd”);

response.addCookie(ck1); --> In memory cookies.

Cookie ck2 = new Cookie(“TN”,”Chennai”);

Ck2.setMaxAge(1800); // 1800 Sec 30 min expiry time

response.addCookie(ck2); --> persistance cookies

**To modify Cookies values:**

ck1.setValue(“hyd1”);

ck2.setValue(“chennai123”);

**To read cookies values:**

Cookie cookies[] = request.getCookies();

if(cookie != null)

SOP(ck.getName() + “ “ + ck.getValue();

**To delete cookies:**

* We cannot delete cookies pro grammatically but we can delete cookies through browser settings.
* In memory cookies can be destroyed ones the browser window is closed.
* Persistence cookies will be destroyed ones expiry time is completed.

**To know Max size (Expiry Time) of cookies:**

* Int time = ck1.getMaxAge(); //-> -1 for in-memory cookie
* Int time2 = ck2.getMaxAge(); // -> Give 1800 seconds

**To get the domain name of the cookie:**

* String doman1 = ck1.getDomain();
* String doman2 = ck2.getDoman();

**Different user cases of Session tracking:**

* Login to logout
* Registration by collection end-user info in multiple form
* Shopping cart
* Online Gaming Session
* Online examination
* Direct advertisement (Rendering the advertisement based on the activity of the end user)
* Continuing the theme chosen by end user across the multiple request during a session and etc.

**What is the difference between state management and session management?**

|  |  |
| --- | --- |
| **State Management** | **Session Management** |
| In state management, the web application remembers the client data across the multiple request but that data is not specific to one user/client and will not be vanished ones session is completed | In session management web application remembers client data across the multiple request but that data is specific to one client and can be remembered only during the session. I.e. ones the session is over that data will be vanished. |
| EG: Writing each form or request data to db table and using that data across the multiple request, | EG: all session tracking technique are session management techniques not the state management techniques. |
| EG: writing each form/req data to Servlet Context/application attributes and using that data across the multiple request. |  |

**Http Session With Cookies**

HttpSession allocates memory in the server 1 per browser software of each client machine.

Http Session object maintain data in the form of Session attributes. HttpSession objects and its attributes are visible in all the web components of web application only when they get request from same browser s/w of client machine for which Session object and Session attributes are created.

Every session object contains 1 Session Id and this session id goes to browser along with the response and comes back to web application along with the request in the form of in-memory cooki automatically.

**Session API (javax.servlet.http.HttpServlet(I) methods):**

**To create/locate session object:**

HttpSession ses = req.getSession(); If browser s/w of client machine which has generated request is not having session having Sesson object in the server then it creates new Session object otherwise it will get access to existing session object.

This method can make current request either joining in the existing session, if session is already started between browser and web application otherwise it creates new session between browser and web application.

This method will create new Session object in the following condition:

1. . if the current request is not getting any cookies
2. If the current request getting cookies but there is no cokkie with “jsessionId” name.
3. If the current request getting cookies and there is cooki having “jsessionId” but they having cookie value there is no session in the server.

HttpSession session = req.getSession(false):

This method get access to session object for the browser s/w of current request if session object is already there for that browser other wise return null I.e. does not create new session object.

This method makes the current request joining in the existing session that is there b/w browser software and web application but does not allow to create new Session between browser s/w and web application.

In the following situation this method returns null:

1. If current request is not having any cookies
2. If current request having cookies but does not have cookie named “jsessionId”
3. If the current request getting cookies and there is cookie having name “jsessionId” but the cookie value (sessionId) there is no session obj in the server.

**To get SessionId:**

String id = req.getId();

**To know the Session object creation date and time:**

Long ms = session.getCreationTime(); We will get Session object creation time in the form of miliseconds that elapsed b/w session creation date and time to **Jan 1st 1970 00:00 hrs (epoach)**.

***In software industry all date and time will be rememberd in the form of milli seconds with respect to Jab 1st 1970 00:00 hrs***

Date d = new Date(ms);

**To know the last accessed date and time of session object:**

Long ms = Session.getLastAccessTime();

**To check weather session is new or old:**

Boolean b = session.isNew(); To check weather current request is creating new session or accessing the existing session.

**To invalidate the session (making session object inactive and ready for garbage collection)**

1. By closing browser s/w or window (In-memory cookie having session id will be destroyed the moment browser or window is closed and without session id the session object becomes inaccessible so indireclty session object become invalidate object.
2. By calling **session.inValidate()**; method
3. When Session inactive period (idle timeout) is completed:
   1. Using declarative approach: in web.xml

|  |
| --- |
| <session-timout>15</session-timeout> (In this time is in minutes |

* 1. Progrramatic approach: session.setMaxInactiveIntervel(1800); Here time is mentions in seconds.

|  |
| --- |
| 1. -> If we specify Session inactive interval period both pragmatically and declarative.. which will be taken ?   Ans -> The programmatic approach will be taken because it overrides the declarative value given in web.xml file. |

**To access ServletContext object:**

ServletContext sc = session.getServletContext();

**HttpSession obj maintain client’s across multiple request as Session Attributes:**

**To create Session Attributes:**

Session.setAttribute(“name”,“raja”);

**To read Session Attributes:**

Session.getAttribute(“name”);

**To modifiy Session Attributes values:**

Session.setAttributes(“name”, “king”);

**HttpSession URL Rewriting**

The major problem with HttpSession with cookies is the session tracking technique fails, if cookies are restricted/blocked coming to browser.

To solve this problem don’t use cookies for sending session-id from web application to browser and for getting back to web application from browser.

We can append session id to that url goes to browser from web application along with response and come back to web application from browser along with the request.

The action url of dynamic form and href url of dynamic hyperlinks goes to browser from web application along with the response and comes back to web application from browser along with the request, if we can append session id to those urls then there will not be any dependency with cookies.

In servlet component:

Pw.print(“<form action=’secondurl;jssionId=54152365’ method =’post’”);

Or

Pw.print(“<form action=’res.encodeURL(‘secondurl’)’ method =’post’”);

**Conclusion on Session Tracking:**

If web application has huge customer base and its wants to insensitive data during the session management then perfer using http with cookies.

These cookies allocate memory to client side and does not give burden to the server.

Example: Remembering user activities in google, Remembering items added to the shopping cart.

If web application is having less customer base and its want to remember sensitive data during the session management then prefer using **HTTP session with url rewriting**

Session attributes allocate memory at server side and give data secrecy.

Example: Remembering Login credentials, remembering credit or debit card details during online payment and etc.

NOTE:: if needed we can use multiple session tracking techniques in one web application development. Like in an e-commerce application use Http cookies to remember items added to the shopping cart and use httpSession with URL re-writing for remembering login credentials and payment information.

DIfference between Generic Servlet and Http Servlets

|  |  |
| --- | --- |
| Generic Servlet | Http Servlets |
| It is an abstract class with one abstract method. | It is an abstract class with no abstract method. |
| Allows any protocol request | Allows only Http request |
| Supports Hidden form fields in session tracking | Supports all the four techniques of session tracking. |
| Gives only service(-,-) to process the request | Gives 7 doXxx(-,-) methods and 2 Service(-,-) method to process the request |
| Gives only ServletRequest, ServletResponse objects using which we cannot use protocol http features. | Gives both ServletRequest,HttpServletRequest and ServletResponse, HttpServletResponse objects using which we can use protocol http features. |