HttpSession

Session Introduction

J2EE helps us to develop "Dynamic Web Application & there are 2 types of Dynamic Web Application.

1. User Dependent Dynamic Web Applications
2. User Independent Dynamic Web Applications

1. User Dependent Dynamic Web Applications :-

- These are the Web Applications where "Response Data is specific to a User"

  Ex:- Gmail, Facebook, Twitter, Linked-In, etc.,
- Login & Logout is "Must" for these Web Applications
- Also these web applications MUST know "from which User it's getting the request"

2. User Independent Dynamic Web Applications :-

- These are the Web Applications where "Response Data is independent of the User"

  For Ex:- Google Search, Google Maps, Youtube, Grepcode, Any Company Related Web Applications (www.jspiders.com) etc.,
- Login & Logout is "Optional" for these Web Applications
- Also these webapplications Need Not know "from which User it's getting the request"

Why HttpSession (a.k.a Session) Functionality ?

- HTTP is a "Stateless Protocol" i.e. it doesn't maintain a relationship/state between requests. Each request is un-related to any previous requests.
- Also HTTP don't help web application to uniquely identify the user.
- Hence even if user sends sequence of requests to web application then it will not able to identify, those are from the same user. Also web application will not be able to relate them
- To address to this problem, Servlet API provides "HttpSession functionality". With HttpSession we can overcome the above problems

Thumb Rule: Any web application which has Login & Logout then we MUST make use of HttpSession functionality.

HttpSession

- A session, w.r.t web application, is a time difference between Login & Logout
- "HttpSession" is a functionality provided by Servlet API which helps web application to
    1. Uniquely identify the user
    2. It maintain "State / Relationship" between requests with the help of "Session Attributes".
    3. Avoids Authentication for each Request
    4. Tightly couples the different pages of the web application.
- To make use of "HttpSession" functionality we must follow 3 Steps
    1. Create a Session (Only Once; during Login)
    2. Validate a Session (Many Times; for every request)
    3. Invalidate a Session (Only Once; during Logout)
- HttpSession uses one of the following two mechanisms to handle session:
    1. Cookies (it's default)
    2. URL Re-Writing
    3.

HttpSession Related Methods

1. HttpSession HttpServletRequest.getSession()
2. HttpSession HttpServletRequest.getSession(boolean create)
3. void HttpSession.invalidate()
4. String HttpSession.getId()
5. void HttpSession.setMaxInactiveInterval(int timeInterval)
6. String HttpServletResponse.encodeURL(String url)
7. Refer "Attributes Section" for Attributes related Methods

Creating a New Session Object Means:

- Container generates "Unique ID"
- Container maintains the Status of this "UniqueID" as "Active" @ Server Side
- Container sends this Unique ID to User in the form of Cookie along with the Response where, Cookie Name = "JSESSIONID", Cookie Value = "Unique ID"
- Container caches this Session Object in it's cache where,Key is "Unique ID" &Value is "Corresponding Session Object"

NOTE: Container does All the above steps when we create HttpSession Object [i.e. when we invoke getSession() or getSession(true)] & it happens behind the scene.

void HttpSession.invalidate(): Invalidates the current session & garbage collects the associated session object

String HttpSession.getId(): This method returns the unique identifier generated for each session

void HttpSession .setMaxInactiveInterval(int timeInterval):

- Specifies the valid session time, in seconds, A negative time indicates the session should never timeout.
- This method helps us to set different Session Time Out for different types of users

String HttpServletResponse.encodeURL(String url): This method encodes the specified URL by appending session ID to it.

<u>Steps to Handle HttpSession</u>

1. Create the Session: Whenever user login & after successful authentication, create session for the First Time.

HttpSession session = req.getSession(true);

HttpSession session = req.getSession();

2. Validate the Session Once Session is created, for subsequent requests validate the session.

Note: Any Servlet/JSP which get the request from Browser, after successful login, then it shoud have "Session validation logic"

Get the Current Session Object

```
HttpSession session = req.getSession(false);
if(session == null)
{
        //Invalid Session !!!
        //Generate "Login Page With Error Info as Response"
}
else{
        //Valid Session ...
        //Generate "Proper Response"
}
```

3. **In-Validate the Session :-**A Session get Invalidated in Following ways

When Application / Server goes down

When user logout of the application

When user wants to logout of the application, then invoke invalidate() method on the Current Session Object.

```
//Get the Current Session Object
HttpSession session = req.getSession(false);
if(session != null)
```

```
{
        session.invalidate();
}
//Generate "Login Page with Success Message"
//as a Response
```

When user is in-active for configured amount of time, There are two ways to configure session time out

1. **In web.xml**

```
<session-config>
        <!-- Time in Min (ex: 7 Days)-->
        <session-timeout>10080</session-timeout>
</session-config>
```

2. **In Program**

```
HttpSession session = req.getSession(true);
//Time in Seconds, ex: 7 Days
session.setMaxInactiveInterval(7*24*60*60);
```

<u>**URL Rewriting**</u>

- While handling user session "URL Rewriting" comes into picture ONLY if we encode the URL's with "Session ID"
- If we encode the URL's, then Container ALWAYS first attempt to use the Cookies to get Session ID & fall back to URL Rewriting only if Cookie approch fails
- In real time scenerios we will not make use of URL Rewriting to handle session (because of "Session Hijacking" issue)