Managing Session in Servlets

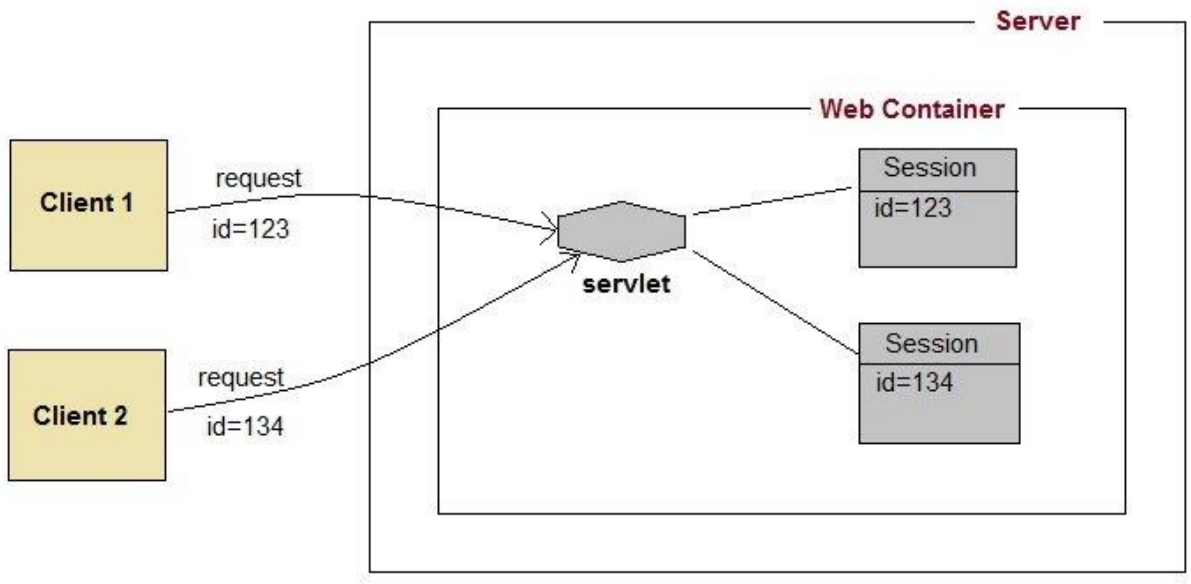
We all know that **HTTP** is a stateless protocol. All requests and responses are independent. But sometimes you need to keep track of client's activity across multiple requests. For eg. When a User logs into your website, not matter on which web page he visits after logging in, his credentials will be with the server, until he logs out. So this is managed by creating a session.

Session Management is a mechanism used by the **Web container** to store session information for a particular user. There are four different techniques used by Servlet application for session management. They are as follows:

1. **Cookies**
2. **Hidden form field**
3. **URL Rewriting**
4. **HttpSession**

Session is used to store everything that we can get from the client from all the requests the client makes.

How Session Works



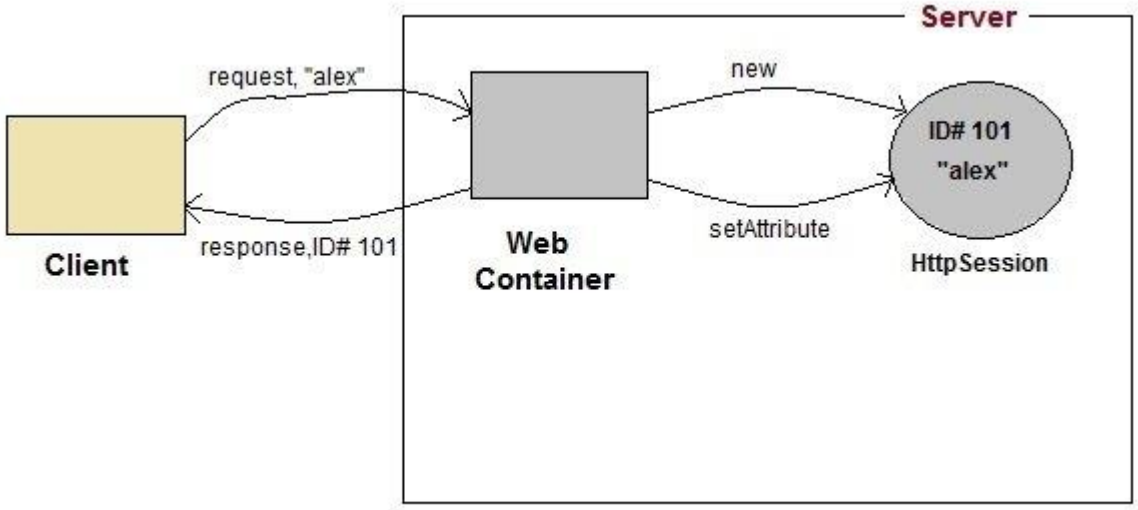
The basic concept behind session is, whenever a user starts using our application, we can save a unique identification information about him, in an object which is available throughout the application, until its destroyed.

So wherever the user goes, we will always have his information and we can always manage which user is doing what. Whenever a user wants to exit from your application, destroy the object with his information.

What is HttpSession?

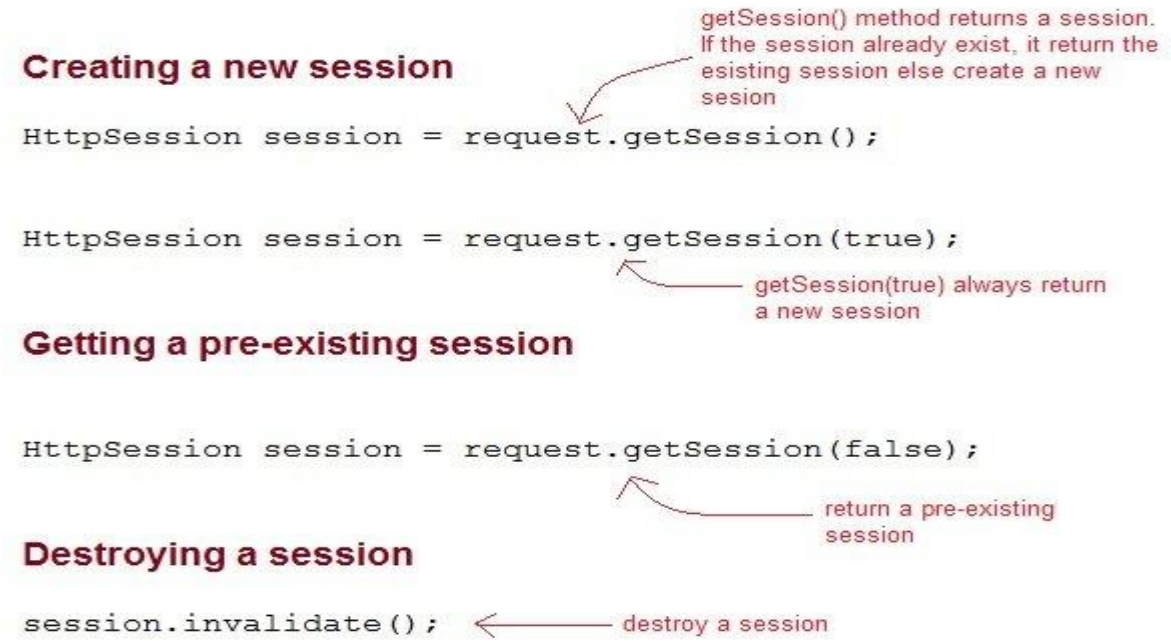
HttpSession object is used to store entire session with a specific client. We can store, retrieve and remove attribute from **HttpSession** object. Any servlet can have access to **HttpSession** object throughout the getSession() method of the **HttpServletRequest** object.

Servlet: How HttpSession works



1. 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.
2. 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.
3. The **Web Container** uses this ID, finds the matching session with the ID and associates the session with the request

HttpSession Interface



JSP SESSION

JSP makes use of the servlet provided HttpSession Interface. This interface provides a way to identify a user across.

- a one-page request or
- visit to a website or
- store information about that user

What Is a Session?

Different approaches to maintain a session between client and server are:

1. **Cookies**: Cookies are text files that allow programmers to store some information on a client computer, and they are kept for usage tracking purposes.

2. **Passing Session ID in URL**: Adding and passing session ID to URL is also a way to identify a session. However, this method is obsolete and insecure because the URL can be tracked.

The session Implicit Object in JSP

- A session object is the most commonly used implicit object implemented to store user data to make it available on other JSP pages until the user's session is active.
- The session implicit object is an instance of a **javax.servlet.http.HttpSession** interface.
- This session object has different session methods to manage data within the session scope.
- By default, JSPs have session tracking enabled and a new HttpSession object is instantiated for each new client automatically. Disabling session tracking requires explicitly turning it off by setting the page directive session attribute to false as follows −

  <%@ page session = "false" %>
  - The JSP engine exposes the HttpSession object to the JSP author through the implicit **session** object. Since **session** object is already provided to the JSP programmer, the programmer can immediately begin storing and retrieving data from the object without any initialization or **getSession()**.

JSP Sessions Methods

1. public Object **getAttribute(String name)**: is used for returning the object bound with the specified name for a session and null if there is no object.
2. public Enumeration **getAttributeNames()**: is used for returning an Enumeration of String objects that will hold the names of all the objects to this session.
3. public long **getCreationTime()**: is used for returning the time when the session was created right from midnight January 1, 1970, GMT.

4. public String **getId()**: is used for returning a string that will hold a unique identifier assigned to your session.
5. public long **getLastAccessedTime()**: is used for returning the latest time your client sent a request linked with the session.
6. public int **getMaxInactiveInterval()**: is used for returning the highest time interval (in seconds), which has to be maintained by the servlet container as a session gets opened between client accesses.
7. public void **invalidate()**: is used for invalidating a session and unbinds its objects bound to it.
8. public boolean **isNew()**: is used for returning a true when the client does not know anything about the session or when the client chooses not to join the session.
9. public void **removeAttribute(String name)**: is used for removing the object bound specifically to a session.
10. public void **setAttribute(String name, Object value)**: is used for binding an object to your session with the help of a specified name.
11. public void **setMaxInactiveInterval(int interval)**: is used for specifying the time (in seconds) between client requests where the servlet container will nullify this session.

COOKIES

Using Cookies for Session Management in Servlet

Cookies are small pieces of information that are sent in response from the web server to the client. **Cookies** are the simplest technique used for storing client state.

Cookies are stored on client's computer. They have a lifespan and are destroyed by the client browser at the end of that lifespan.

Using Cookies for storing client state has one shortcoming though, if the client has turned of COokie saving settings in his browser then, client state can never be saved because the browser will not allow the application to store cookies.

Cookies API

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 to access the cookies that are added to response object.

Example demonstrating usage of Cookies