Adv. Java means DURGA SIR..

ADV.JAVA With SCWCD / OCWCD

Servlets Material

4. Session Management

DURGA M.Tech

(Sun certified & Realtime Expert)

Ex. IBM Employee

Trained Lakhs of Students for last 14 years across INDIA

India's No.1 Software Training Institute

DURGASOFT

www.durgasoft.com Ph: 9246212143 ,8096969696

Session Management

Agenda:

1) **Session API**
    - **Creation of Session object**
    - **Invalidating a Session**
        - **invalidate( )**
        - **By Session timeout mechanism**
    - **Important Methods of HttpSession**
    - **Attribute Management in Session Scope**
    - **Exchanging Session Id between Client and Server**

2) **Cookies**
    - **Important methods of Cookie class**
    - **Persistant (Vs) Non-Persistant Cookies**
    - **Advantages of Cookies**
    - **Limitations of Cookie**
    - **Differences between Session-API and Cookies**

3) **URL Rewriting**
    - **HttpServletResponse two methods to append Session id to the url**
    - **Advantage of URL Re-writing**
    - **Limitations**

4) **Hidden Variables**

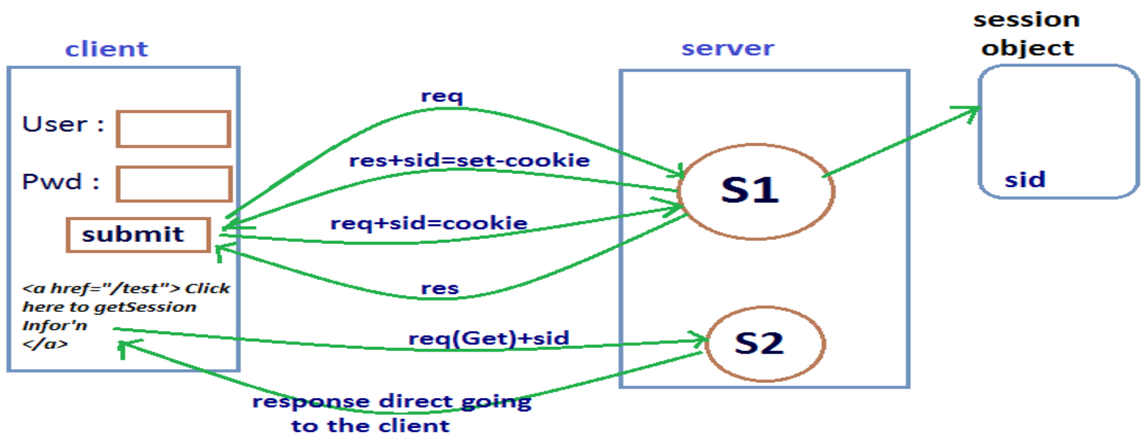
5) **Listeners**
    - **Request Listeners**
        - **ServletRequestListener**
        - **ServletRequestAttributeListener**
    - **Context Listeners**
        - **ServletContextListener**
        - **ServletContextAttributeListener**
    - **Session Listeners**
        - **HttpSessionListener**
        - **HttpSessionAttributeListener**
        - **HttpSessionBindingListener**
        - **HttpSessionActivationListener**

Objective:

1. For the given scenario describe the Session API.
2. Explain the process of creating a Session object.
3. What are various different mechanisms to invalidate a session
4. The basic limitation of Http is , Its a stateless protocol i.e., it is unable to remember client state across multiple request.
5. Every request to the server is considered as a new request. Hence some mechanism is required to remember client information across multiple requests. This mechanism is nothing but Session management /Session Tracking Mechanism.
6. The following are various Session management mechanisms.
   1. Session-API
   2. Cookies
   3. URL Rewriting
   4. Hidden Variables

      It is not official Session management from Sun. It is just a programmers trick to remember client information across multiple requests.

1.Session-API :



1. When ever client sends 1st request to the server, If the server wants to remember client information for the future purpose then it will create a Session object stores the required Session information in the form of Session scoped attributes.
2. Server sends the corresponding session-id as the part of 1st response.
3. Client saves that Session-id and send back to the Server with every consecutive request.
4. By accessing Session-id and corresponding Session object , Server can able to remember client information accross multiple requests. This mechanism is nothing but Session management by using Session-API.

5. **In this mechanism entire Session information will be stored at Server side , and only Session-id will be maintained by Client.**
   Ex:Bank Locker

Creation of Session object :

HttpServletRequest interface defines the following methods for the creation of Session object.

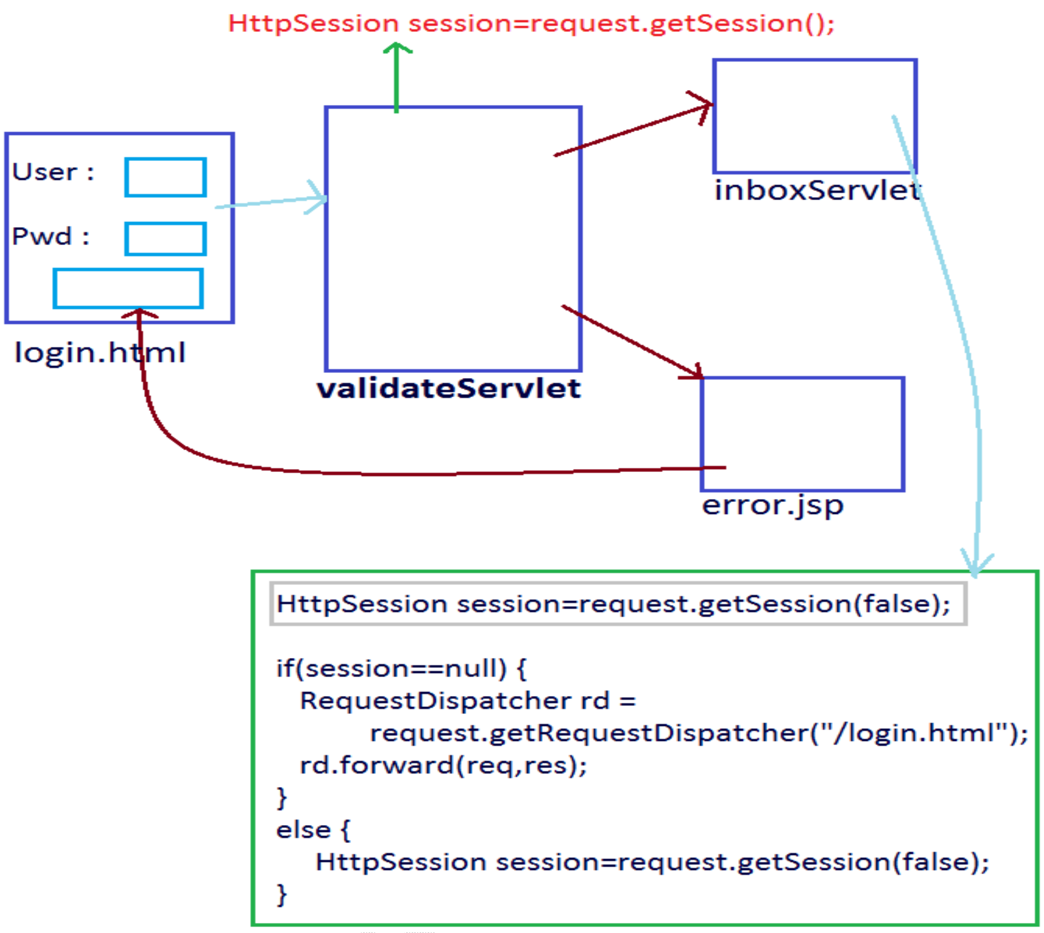
<u>public HttpSession getSession( )</u>

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

- **First this method will check whether the request is already associated with any Session or not. If the request associated with any session then existing Session object will be returned.**
- **If the request is not associated with any Session , then only a new Session object will be created.**

<u>public HttpSession getSession(boolean b)</u>

- **if the argument is true , then this method simply acts as getSession( )**
- **If the argument is false , then this method first checks whether the request is already associated with any Session or not, If it is already associated with a Session , then existing Session object will be returned.**
- **If the request is not associated with a Session then this method simply returns null , without creating new object.**
- **Note: There is always guarantee that first method returns a Session object. It may be newly created or already existing one.**
- **There is no guarantee that getSession(false) method will return Session object.**



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

if(session==null) {
  RequestDispatcher rd =
        request.getRequestDispatcher("/login.html");
  rd.forward(req,res);
}
else {
    HttpSession session=request.getSession(false);
}
```

- **After providing credentials in the login page the request will be forwarded to the ValidateServlet.**
- **With in the ValidateServlet , we will check whether the credentials are valid or not. If the credentials are valid , a new Session object will be created and forward the request to inbox.jsp , Hence with in the Validate Servlet. We have to use request.getSession( ) method because we can create a new Session object , if it is not already there .**
- **To access inbox.jsp , compulsory request should be associated with Session.**
- **If the Session is not already there inbox.jsp is not responsible to create new Session object and just simply forwards the request to login page.**
- **Hence to meet this requirement inside inbox.jsp we have to use request.getSession(false);**

<u>Which of the two statements are equal ?</u>

- **HttpSession session = request.getSession( );**
- **HttpSession session=context.getSession(true);**

- HttpSession session = request.getSession(false);
- HttpSession session = request.getSession(true);

Answer : 1 & 4

Invalidating a Session :

We can invalidate a Session by using the following 2 ways

1. By invalidate method
2. By timeout mechanism

1.invalidate( ) :

HttpSession interface contains invalidate( ) to invalidate a Session explicitly.

```
public void invalidate( )
```

When ever we click logout button , internally this method will be executed.

Ex: session.invalidate( )

LogoutServlet.java

```
public class LogoutServlet extends HttpServlet {
        public void doGet(HttpServletRequest request, HttpServletResponse response)
                                    throws ServletException, IOException {
            response.setContentType("text/html");
            PrintWriter out = response.getWriter();
            HttpSession session=request.getSession(false);
            if(session != null)
                session.invalidate();
            else
                out.println("No session is  invalidate()");
     }
}
```

web.xml

```
<servlet>
      <servlet-name>LogoutServlet</servlet-name>
      <servlet-class>session.LogoutServlet</servlet-class>
</servlet>
<servlet-mapping>
      <servlet-name>LogoutServlet</servlet-name>
      <url-pattern>/logout</url-pattern>
</servlet-mapping>
```

2.By Session timeout mechanism :

1. If we are not performing any operation for a predefined amount of time on the session object, then session will be expired automatically
2. This pre-defind amount of time is called "Session time out ".
3. We can configure Session time out at Server level or a particular web-application level or for a particular Session object level.

- ***Automatic Support from the WebServer :***

  Most of the Web-Servers provide default support for Session time out mostly it is 30 minutes. We can customize this value based on our requirement. This Session timeout is application for all sessions , Which are created in that Server irrespective of application.

- ***Configuring Session timeout at application level :***

  1. We can configure Session timeout for entire web-application in web.xml as follows

```
<web-app>
      <session-config>
                  <session-timeout> 10 </session-timeout>
      </session-config>
</web-app>
```

  2. The session-config is direct child tag of web-app .Hence we can take any where with in web-app
  3. The unit to the session timeout is in minutes.
  4. Zero (or) negative value indicates that session never expires.(untill we click the logout button)
  5. This session timeout value is applicable for all sessions , which are created as part of web application.

Configuring Session timeout for a particular Session object :

6. We can set the Session timeout by using setMaxInactiveInterval( ) method for a perticular Session object.

```
public void setMaxInactiveInterval(int seconds)
```

7. Ex: session.setMaxInactiveInterval(120);

8. The argument is in seconds, -ve value indicates session never expires, Zero value indicates session expires immediately .
9. This session time out is applicable only for a particular session object on which this method has called.

Comparision between two Session timeout mechanism :

Property	<session-timeout>	setMaxInactiveInterval( )
Scope	It is applicable for all the Sessions which are created in that application.	It is applicable only for a particular Session object on which we called this method
Units	Minutes	Seconds
Zero value	Session never expires	Session expires immediately
- ve	Session never expires	Session never expires

Important Methods of HttpSession :

1) <u>public booleanisNew()</u>
We can use this method to check whether the Session object is newly created or not.

2) <u>public void inValidate()</u>
To expires a session forcefully.

3) <u>public void setMaxInactiveInterval(int seconds)</u>
To set session timeout for a particular Session object .

4) <u>public void setMaxInactiveInterval()</u>
It returns the session timeout value in seconds.

5) <u>public String getId()</u>
Returns the session id.

6) <u>public long getCreationTime()</u>
- Returns the time when the Session was created in milliseconds , Since Jan 1st1970.
- If we are passing this long value to the Date constructor , we will get exact Date and time.
Date d = new Date(l);

7) <u>public long getLastAccessedTime()</u>

Returns the time when the client accessed recently the ....

8) <u>public ServletContext getServletContext()</u>
Returns the ServletContext object to which this Session belongs.

Methods in HttpSession to perform Attribute Management in Session Scope :

1. public void setAttribute(String name, Object value)
2. public Object getAttribute(String name)
3. public void removeAttribute(String name)
4. public Enumeration getAttributeNames( )

Note : Once session expires we can't call any of the above methods , validation leads to RuntimeException saying "IllegalStateException".But this rule is not applicable for getServletContext( ) method .

```
HttpSession session=reqest.getSession();
session.invalidate();
------------------
session.isNew(); //java.lang.IllegalStateException
------------------
out.println(session.getServletContext()); //valid
```

Ex: Demo program for Session Management by session API

login.html

```
<form action="./sessionone">
      <table>
          <tr><td>Name :</td><td> <input type="text" name="uname"></td></tr>
          <tr><td>Value : </td><td> <input type="text" name="value"></td></tr>
          <tr><td><input type="submit" value="submit"></td></tr>
      </table>
</form>
<a href="./sessiontwo">Session Information</a>
```

SessionServletOne.java

```
public class SessionServletOne extends HttpServlet {
        public void doGet(HttpServletRequest request, HttpServletResponse response)
                                             throws ServletException, IOException {
              response.setContentType("text/html");
              PrintWriter out = response.getWriter();
              String name=request.getParameter("uname");
              String value=request.getParameter("value");
              HttpSession session=request.getSession();
               if(session.isNew()) {
                     out.println("New Session got created "+session.getId());
```

```
            }
          else {
               out.println("With existing Session id : "+session.getId());
          }
              session.setAttribute(name, value);
              session.setMaxInactiveInterval(120);
              RequestDispatcher rd=request.getRequestDispatcher("login.html");
              rd.include(request, response);
     }
}
```

SessionServletTwo.java

```
public class SessionServletTwo extends HttpServlet {
        public void doGet(HttpServletRequest request, HttpServletResponse response)
                                          throws ServletException, IOException {
             response.setContentType("text/html");
             PrintWriter out = response.getWriter();
             HttpSession session=request.getSession(false);
             if(session == null) {
                   out.println("No Session id  associated with request ");
             }
             else {
                    Enumeration  e=session.getAttributeNames();
                    out.println("<table border=3><tr><th>Session Attribute Name</th>
                                               <th>Session Attribute Value</th></tr>");
                    while(e.hasMoreElements()) {
                           String name=(String) e.nextElement();
                           String value=(String) session.getAttribute(name);
                           out.println("<tr><td>"+name+"</td>  <td>"+value+"</td></tr>");
                    }
                    out.println("</table>");
                   out.println("<br>The Session creation Time is :"+ new Date(session.getCreationTime()));
       out.println("<br>The session last accessed time is :"+ new Date(session.getLastAccessedTime()));
            out.println("<br>The Session max inactvate interval is :"+ session.getMaxInactiveInterval());
      }
      out.println("<br><a href=login.html> login page </a>");
    }
}
```

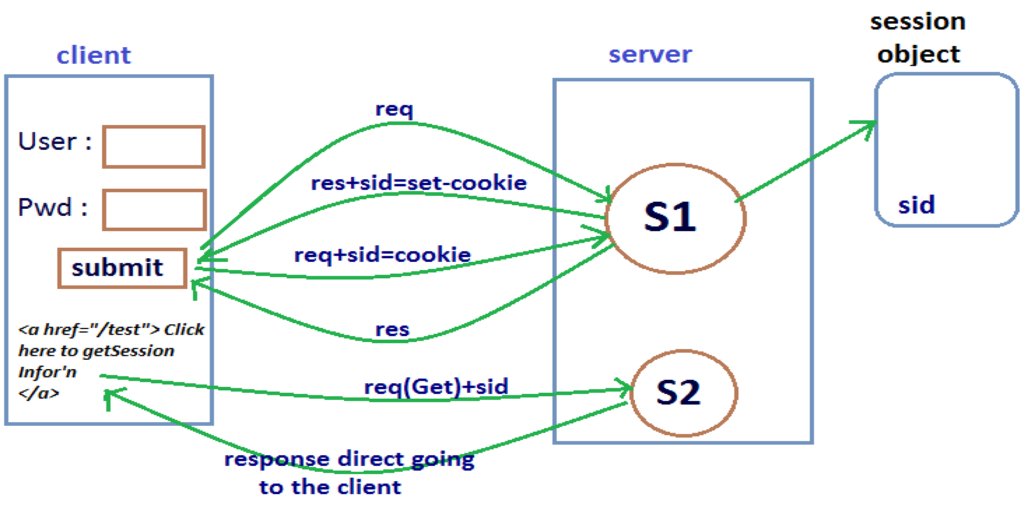
web.xml

```
<web-app>
      <servlet>
         <servlet-name>SessionServletOne</servlet-name>
         <servlet-class>session.SessionServletOne</servlet-class>
      </servlet>

      <servlet>
```

```
            <servlet-name>SessionServletTwo</servlet-name>
            <servlet-class>session.SessionServletTwo</servlet-class>
      </servlet>

      <servlet-mapping>
             <servlet-name>SessionServletOne</servlet-name>
             <url-pattern>/sessionone</url-pattern>
      </servlet-mapping>

      <servlet-mapping>
            <servlet-name>SessionServletTwo</servlet-name>
            <url-pattern>/sessiontwo</url-pattern>
      </servlet-mapping>

      <session-config>
            <session-timeout>1</session-timeout>
      </session-config>

</web-app>
```

Exchanging Session Id between Client and Server :

1. Whenever Client sends 1st request to the Sever , If any request information is required to remember for the future purpose then server creates a Session object and score the required information in the form of Session scopped attributes.
2. Server sends corresponding session id to the client as the part of 1st response.For this server uses "set-cookie" response header.
3. Whenever client got the response , it retrives the session id and stored in the local file System.
4. With every consecutive request client send back the corresponding Session id to the server, for this client uses "cookie" request header.
5. By accessing Session id and corresponding Session object Server can able to remember client information across multiple requests.
6. By using set-cookie response header and cookie request header session id will be exchanged between client and server.



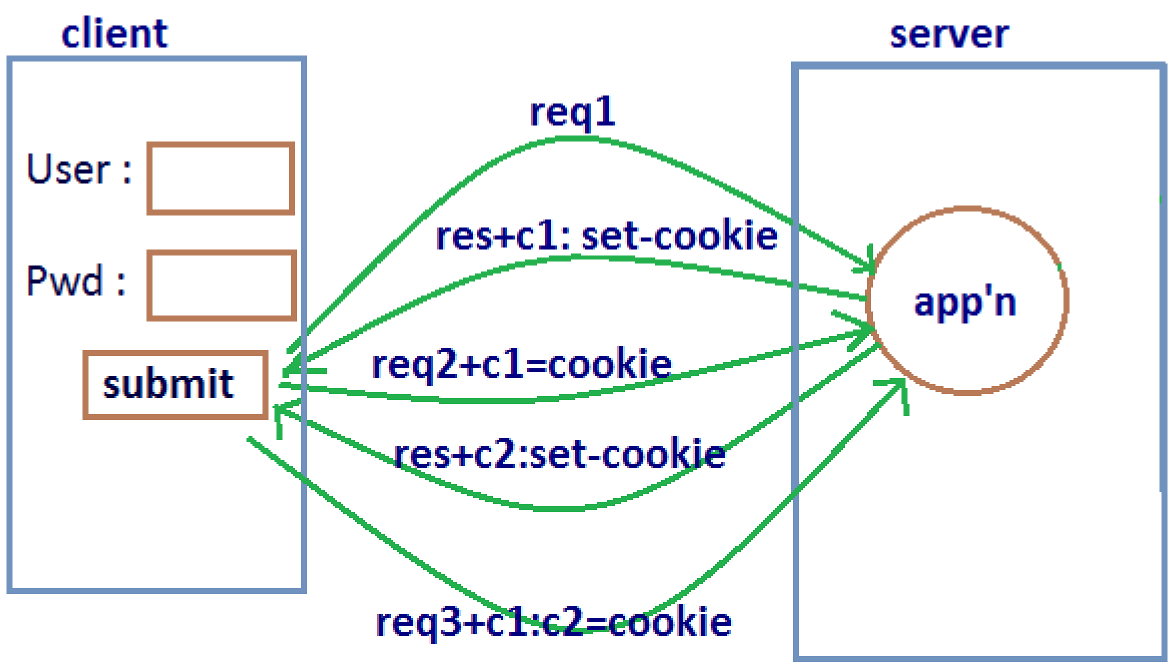
Session id will be stored as per application in the client side.

header	value
cookie	sid

Ex: Note:(Bank locker):

1. **If the required Session information is very less than creating a separate Session object and maintaining that object at Server side is not recommended because it impacts performense of the System.**
2. **To resolve this, we should go for cookie Session management mechanism , where required Session information is maintained at client side and Server is not responsible to maintain any Session information.**

Cookies :



1. Cookie is a small amount of information (key , value) pair.
2. Whenever client sends a request to the Server , if any information required to remember for future purpose then server creates a Cookie object with that information and send that Cookie back to the client as the part of response . For, this , server uses set-cookie response header.
3. Whenever client got the response it retrives the cookies send by the Server and stores those cookies in the local file system.
4. Client sendback all the Cookies send by the server with every consecutive request for this client use cookie request header.
5. By accessing those cookies server can able to remember client information across multiple requests.
6. We can create Cookie object by using Cookie class constructor.

```
Cookie cookieobject = new Cookie(String key , String value );
```

7. After creating the Cookie object we have to add that Cookie to the response by using addCookie( ).

```
response.addCookie(cookieobject) ;
```

8. At server side , we can retrive all cookies send by the client by using getCookies( ).

```
Cookies[ ] cookieobject = request.getCookies( );
```

9. If the request doesn't associated with any Cookies then this method returns null.

Important methods of Cookie class :

public String getName()
This method returns the name of the Cookie.

public String getValue()
Returns value of the Cookie.

public int getMaxAge()
Returns the maximum age of Cookie in seconds.

public void setMaxAge(int seconds)
To set max age of cookie.

- "+ve" value indicates that cookie will be expires after that how many seconds have passed.
- Note that the value is max age of the cookie, when the cookie will expires , not the cookie current age.
- "-ve" value indicates the cookie is not stored permanently and it will be deleted when browser exists/close.
- "0" value indicates cookies to be deleted.

Demo Program for session management by cookies

login.html

```
<form action="./cookieone">
      <table>
            <tr><td>Name :</td><td> <input type="text" name="uname"></td></tr>
            <tr><td>Value : </td><td> <input type="text" name="value"></td></tr>
            <tr><td><input type="submit" value="submit"></td></tr>
      </table>
</form>
<a href="./cookietwo">Cookie Information</a>
```

CookieServletOne.java

```
public class CookieServletOne extends HttpServlet {
        public void doGet(HttpServletRequest request, HttpServletResponse response)
                                        throws ServletException, IOException {
            response.setContentType("text/html");
            PrintWriter out = response.getWriter();
            String name=request.getParameter("uname");
            String value=request.getParameter("value");
            Cookie c=new Cookie(name, value);
            c.setMaxAge(120);
            response.addCookie(c);
            out.println("Cookie added Successfully");
            out.println("domain"+c.getDomain());
            out.println("comment"+c.getComment());
            out.println("maxage"+c.getMaxAge());
            out.println("version"+c.getVersion());
            out.println("hashcode"+c.hashCode());
            RequestDispatcher rd=request.getRequestDispatcher("login.html");
            rd.include(request, response);
      }
}
```

CookieServletTwo.java

```
public class CookieServletTwo extends HttpServlet {
        public void doGet(HttpServletRequest request, HttpServletResponse response)
                                        throws ServletException, IOException {
                  response.setContentType("text/html");
                  PrintWriter out = response.getWriter();
                  Cookie[] c = request.getCookies();
                  if(c == null) {
                        out.println("No cookies associated with this request");
                  }
                  else {
              out.println("<table border=3><tr><th>Cookie Name</th><th>Cookie Value</th></tr>");
                          for(Cookie c1:c) {
                              String name = c1.getName();
                              String value = c1.getValue();
                              out.println("<tr><td>"+name+"</td><td>"+value+"</td></tr>");
                          }
                          out.println("</table>");
             }
      }
}
```

web.xml

```
<web-app>

      <servlet>
           <servlet-name>CookieServletOne</servlet-name>
           <servlet-class>session.CookieServletOne</servlet-class>
      </servlet>

      <servlet>
          <servlet-name>CookieServletTwo</servlet-name>
```

```
            <servlet-class>session.CookieServletTwo</servlet-class>
        </servlet>

        <servlet-mapping>
              <servlet-name>CookieServletOne</servlet-name>
              <url-pattern>/cookieone</url-pattern>
        </servlet-mapping>

        <servlet-mapping>
             <servlet-name>CookieServletTwo</servlet-name>
             <url-pattern>/cookietwo</url-pattern>
  </servlet-mapping>

</web-app>
```

Persistant (Vs) Non-Persistant Cookies :

Persistant Cookies	Non-Persistant Cookies
1. If we set max age for the Cookie , such type of Cookies are called "Persistant Cookies". 2. These cookies will be stored permanently in the local file system. 3. Once the time expires , these Cookies will be disabled automatically.	1. If we are not setting max age for the Cookie , such type of Cookies are called Non-Persistant/temporary Cookies. 2. These will be stored in the browsers cache and disabled automatically Once browser will be closed.

Advantages of Cookies :

1. It is very easy to implement.
2. Persist across browser shutdowns, server shutdowns and application redeployments.
3. If very less Session information is available or if huge no. of end-users are available then the best suitable mechanism is Cookies.

Limitations of Cookie :

1. To meet security constraints there may be a chance of disabling cookies at client side. In this case Session management by Cookies wan't work .
2. The maximum no. of Cookies supported by the browser is fixed.(maximum of 30, based on browser)
3. The size of Cookie is also fixed. Hence we can't store huge amount of information by using Cookies.
4. The Cookies should be travelled every time across the network. Hence there may be a chance of network over heads. i.e., it impact performance

Differences between Session-API and Cookies :

Session-API	Cookies
If huge amount of Session information is available , then we should go for Session-API .	If very less amount of Session information is available , then we should go for Cookie.
Session information will be maintained at Server side.	Session information will be maintained at client side.
The Session information can be any type and need not be String type.	Session information should be String type
Session information won't persist across server shutdown & application re-deploys.	Session information will be persist across Server shutdowns & application deployments.(The cookies should be persist)

Note: To meet Security constraints there may be a chance of disabled cookies at client side . In this case Session management by using Session-API and Cookies won't work. To handle this requirement we should go for url-rewriting .

URL - Rewriting :

1. Whenever Cookies are disabled at client side , browser unable to see "set-cookie" response header and hence browser unable to get session id and cookies send by the Server .
2. Due to this browser can't send session id & Cookies to the server and hence server is unable to remember client information across multiple requests . So that Session management fails.
3. To resolve this we should go for url rewriting technique.
4. The central idea in this technique is append required Session information to the url , instead of appending to "set-cookie" response header.
5. Whenever client clicks url for further communication server can get required Session information with the url . So that Server can able to remember client information across multiple requests.

URL Re-writing = URL + session information

OR

URL Re-writing = URL ; jsessionid=1234

HttpServletResponse defines the following two methods to append Session id to the url

1) public String encodeURL(String url)
   Returns url by appending jsession id

2) **public String encodeRedirectURL(String url)**
   Returns the url by appending session id . This URL can be used as an argument to send sendRedirect() method

- The above 2 methods will append jsessionid to the url iff cookies are disabled at client side.
- If cookies are enabled then these methods returns the same url without appending jsessionid .
- HttpServletRequest defines the following methods to identify whether the sessionid is coming as the part of url or with cookie request header.
  1) public boolean isRequestedSessionIdFromURL()
  2) public boolean isRequestedSessionIdFromCookie()
- By using these methods we can identify underlying Session management technique.

Advantage of URL Re-writing :

There is no chance of disabling url re-writing technic hence this technique will work always . It is universally supported.

Limitations :

1. It is very difficult to rewrite every url to append Session information .
2. Url rewriting works only for dynamic document i.e., response should contain at least one url .
3. To keep our web-application as robust we can use both cookies and url rewriting together.
4. If cookies are enabled then cookies will work otherwise url rewriting will work.

Demo program on URL rewriting

login.html

```
<form action="./redirectone">
      <table>
          <tr><td>Name :</td><td> <input type="text" name="uname"></td></tr>
          <tr><td><input type="submit" value="submit"></td></tr>
      </table>
</form>
```

UrlRedirectServletOne.java

```
public class UrlRedirectServletOne extends HttpServlet {
        @SuppressWarnings("deprecation")
        public void doGet(HttpServletRequest request, HttpServletResponse response)
                                    throws ServletException, IOException {
                response.setContentType("text/html");
                PrintWriter out = response.getWriter();
                String name = request.getParameter("uname");
                HttpSession session = request.getSession();
                session.setAttribute("uname", name);
                out.println("Welcome to Aksahay");
             // out.println("<br> <a href=./redirecttwo?name=" +name+ "> click here to get User </a>");
```

```
 out.println("<br> <a href= "+response.encodeUrl("./redirecttwo") +" >  click here to get User </a> ");
        }
}
```

UrlRedirectServletTwo.java

```
public class UrlRedirectServletTwo extends HttpServlet {
        public void doGet(HttpServletRequest request, HttpServletResponse response)
                                        throws ServletException, IOException {
             response.setContentType("text/html");
             PrintWriter out = response.getWriter();
             HttpSession session = request.getSession(false);
             String name = (String)session.getAttribute("uname");
            //String name = request.getParameter("name");
            out.println("Good Morning :"+name);
       }
}
```

web.xml

```
<web-app>

       <servlet>
            <servlet-name>UrlRedirectServletOne</servlet-name>
            <servlet-class>session.UrlRedirectServletOne</servlet-class>
       </servlet>

       <servlet>
            <servlet-name>UrlRedirectServletTwo</servlet-name>
            <servlet-class>session.UrlRedirectServletTwo</servlet-class>
       </servlet>

       <servlet-mapping>
            <servlet-name>UrlRedirectServletOne</servlet-name>
           <url-pattern>/redirectone</url-pattern>
       </servlet-mapping>

       <servlet-mapping>
             <servlet-name>UrlRedirectServletTwo</servlet-name>
             <url-pattern>/redirecttwo</url-pattern>
       </servlet-mapping>

</web-app>
```

Listeners

Objective :

- Describe the web-container event life cycle model for the request, Session and webapplication(context).
- Create and configure Listener class for each Scope.
- Create and configure attribute Listeners for each scope.
- For the given scenario identify proper attribute Listener.
- In the webapplication there may be a chance of occuring several events like ...
    1. Request object creation/destruction
    2. Session object creation/destruction
    3. Context object creation/destruction
    4. Attribute addition in request/session Scope
    5. Removal of attribute in request/session Scope
- We can configure Listeners classes to listen these events and they can do appropriate things whenever that event occurs.

When ever a particular event occurs if we want to perform certain operation then we should go for Listeners.

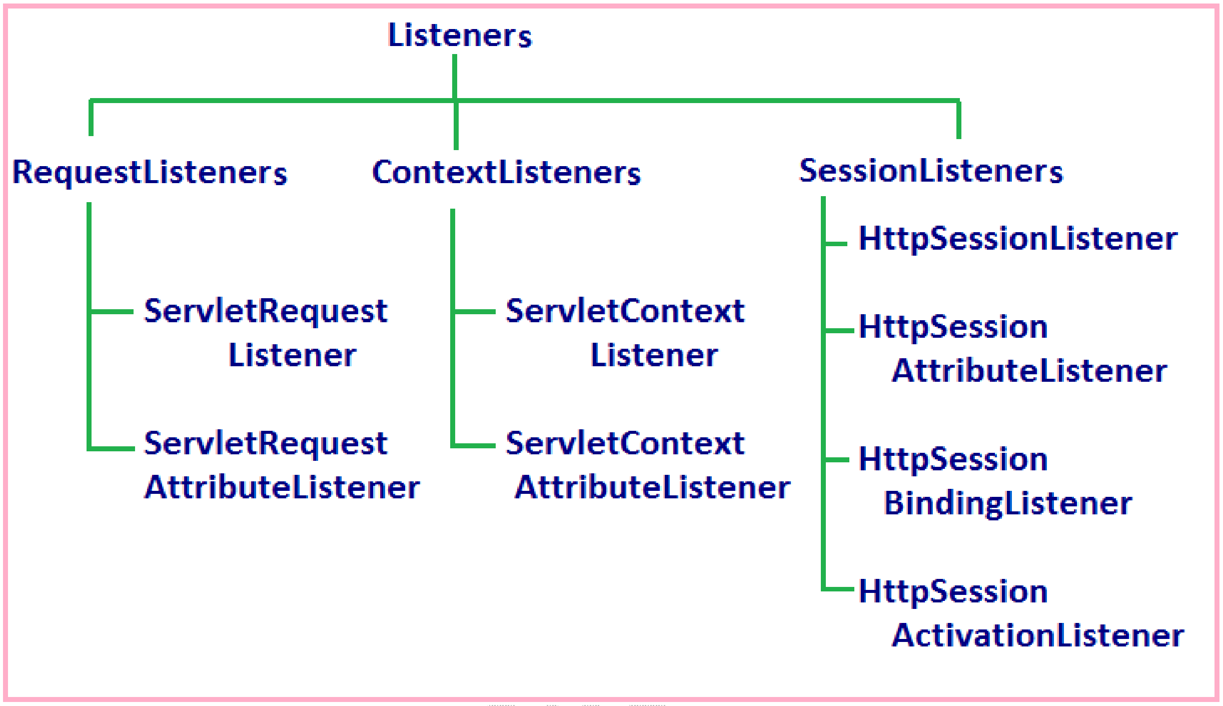
All the Listeners are divided into 3 groups.

1) **RequestListeners :**
    - These Listen the events related to request.
    - There are 2 types of RequestListeners.
        - ServletRequestListener
        - ServletRequestAttributeListener

2) **Session Listeners :**
    - Listens the events related to Sessions.
    - There are 4 types of Session Listeners.
        - HttpSessionListener
        - HttpSessionAttributeListener
        - HttpSessionBindingListener
        - HttpSessionActivationListener

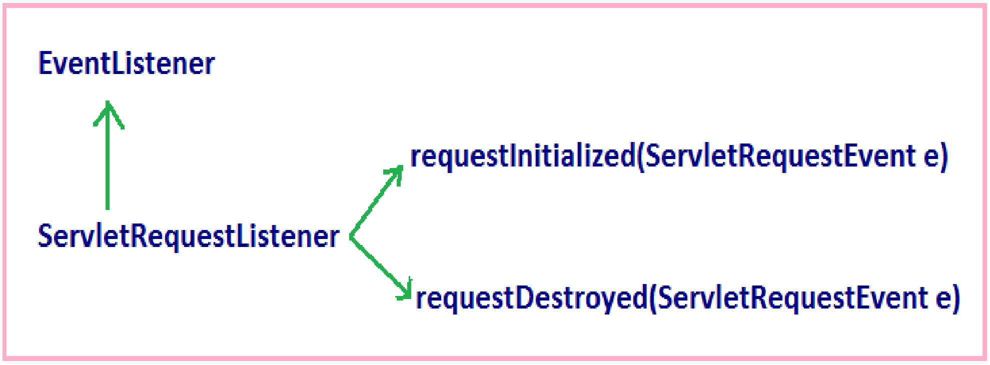
3) **ContextListener :**
    - Listens the events related to context.
    - There are 2 types of context Listeners.
        - ServletContextListener
        - ServletContextAttributeListener

Note: java.util.EventListener is the super interface for all the Listeners in our servlets.
This Listener doesn't contain any method and acts as marker interface.


Listeners
RequestListeners
ContextListeners
SessionListeners
ServletRequest Listener
ServletRequest AttributeListener
ServletContext Listener
ServletContext AttributeListener
HttpSessionListener
HttpSession AttributeListener
HttpSession BindingListener
HttpSession ActivationListener

RequestListeners :

ServletRequestListener


EventListener
ServletRequestListener
requestInitialized(ServletRequestEvent e)
requestDestroyed(ServletRequestEvent e)

This Listener life cycle events of request object like request object creation and destruction.

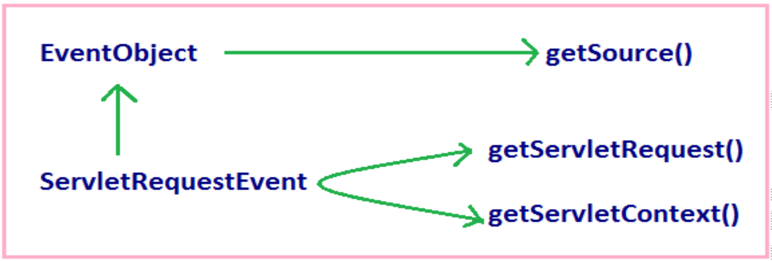
This interface defines 2 methods

public void requestInitialized(ServletRequestEvent e)
This method will be executed automatically by the WebContainer, at the time of request object creation, i.e., just before executing service()

public void requestDestroyed(ServletRequestEvent e)

This method will be executed automatically by the WebContainer at the time of request object destruction, i.e., just after completing service().

ServletRequestEvent(C) :



It defines the following 2 methods

1. public ServletRequest getServletRequest()
2. public ServletContext getServletContext()

java.util.EventObject

ServletRequestEvent is the child class of EventObject, it contains only one method getSource()

public Object getSource()

It returns source of the event, in this case the source of the event is web application which is represented by ServletContext object hence getSource() method returns ServletContext object.

ServletRequestListenerDemo.java

```
public class ServletRequestListenerDemo implements ServletRequestListener {
    static int count;
    static {
        System.out.println("ServletRequestListener class is  loading ");
    }
    public ServletRequestListenerDemo() {
         System.out.println("ServletRequestListener Object is  created ");
    }
    public void requestDestroyed(ServletRequestEvent event) {
         System.out.println("The  request object destroyed at :"+new java.util.Date());
         System.out.println("The source of destroyed request is :"+event.getSource());
    }
    public void requestInitialized(ServletRequestEvent event) {
         count++;
         System.out.println("new request object is created at :"+ new java.util.Date());
         System.out.println("The source of  creation request is :"+event.getSource());
```

```
            System.out.println("The context is :"+event.getServletContext());
            System.out.println("The  request is :"+event.getServletRequest());
      }
}
```

RequestServlet.java

```
public class RequestServlet extends HttpServlet {
          public void doGet(HttpServletRequest request, HttpServletResponse response)
                                         throws ServletException, IOException {
               response.setContentType("text/html");
               PrintWriter out = response.getWriter();
             out.println("The hit count of the web application is :"+ ServletRequestListenerDemo.count);
          }
}
```

web.xml

```
<web-app>
      <listener>
           <listener-class>listener.ServletRequestListenerDemo</listener-class>
                // fully qualified name
           </listener>
           <servlet>
                <servlet-name>RequestServlet</servlet-name>
                <servlet-class>listener.RequestServlet</servlet-class>
           </servlet>
           <servlet-mapping>
                 <servlet-name>RequestServlet</servlet-name>
                 <url-pattern>/request</url-pattern>
          </servlet-mapping>
</web-app>
```

- By using listener tag we can configure listener class in web.xml
- listener tag is the direct child tag of web-app, hence we can take any where with in the web-app.
- web container is responsible for the creation of listener object for this WC always calls public no-arg constructor, hence every listener class should compulsory contains public no-arg constructor otherwise Instantiation Exception.
- Instantiation of the listener will be happen at the time of web-application deployment or sever start-up.

<u>Note:</u> In the above program count value won't persist across server shutdown or application un-deployment but we can persist count value by using ServletContextListener.

ServletRequestAttributeListener

This Listener listens to events related to request scoped attributes like attribute addition, attribute removel, attribute replacement.

This interface defines following methods

1. **public void attributeAdded(ServletRequestAttributeEvent e)**
   This method will be executed automatically by the by the WC, when ever we are adding an attribute an request scope.
2. **public void attributeRemovel(ServletRequestAttributeEvent e)**
   This method will be executed automatically by the by the WC, when ever we are removing an attribute from request scope.
3. **public void attributeReplaced(ServletRequestAttributeEvent e)**
   This method will be executed automatically by the by the WC, when ever we are replacing an existing attribute named with new value. This method returns old value, but not new value.

ServletRequestAttributeEvent

It is the child class of ServletRequestEvent, this class defines the following 2 methods

1. **public String getName()**
   It returns name of the attribute which is adding request scope.
2. **public Object getValue()**
   It returns the value of attribute which is added or removed , In this case of replacement this method returns old value.

ServletRequestAttributeListenerDemo.java

```
public class ServletRequestAttributeListenerDemo implements  ServletRequestAttributeListener {
		public ServletRequestAttributeListenerDemo() {
				System.out.println("ServletRequestAttributeListenerDemo obj created ");
		}
		public void attributeAdded(ServletRequestAttributeEvent event) {
				System.out.println("Attribute added name :"+event.getName());
				System.out.println("Attribute added value :"+event.getValue());
		}
		public void attributeRemoved(ServletRequestAttributeEvent event) {
				System.out.println("Attribute removed name :"+event.getName());
				System.out.println("Attribute removed value :"+event.getValue());
		}
		public void attributeReplaced(ServletRequestAttributeEvent event) {
				System.out.println("Attribute replaced name :"+event.getName());
				System.out.println("Attribute replaced value :"+event.getValue());
		}
}
```

ServletRequestAttribute.java

```
public class ServletRequestAttribute extends HttpServlet {
	public void doGet(HttpServletRequest request, HttpServletResponse response)
```

```
                                        throws ServletException, IOException {
                response.setContentType("text/html");
              //PrintWriter out = response.getWriter();
                request.setAttribute("Ashok", "SCJP");
                request.setAttribute("Ashok", "SCWCD");
                request.removeAttribute("Ashok");
      }
}
```

web.xml

```
<listener>
      <listener-class>listener.ServletRequestAttributeListenerDemo</listener-class>
 </listener>
 <servlet>
       <servlet-name>ServletRequestAttribute</servlet-name>
       <servlet-class>listener.ServletRequestAttribute</servlet-class>
 </servlet>
 <servlet-mapping>
       <servlet-name>ServletRequestAttribute</servlet-name>
       <url-pattern>/requestattribute</url-pattern>
 </servlet-mapping>
```

ContextListener:

ServletContextListener

This Listener listens life cycle events of context object like creation & destruction. This Listener defines the following 2 methods

1. public void contextInitialized(ServletContextEvent e)
   This method will be executed at the time of context object created or application deployment.
2. public void contextDestroyed(ServletContextEvent e)
   This method will be executed at the time of context object destruction. i.e., at the time of application un-deployment.

ServletContextEvent:

It is the child class java.util.EventOject and it contains only one method public ServletContext getServletContext().

Demo program to ContextListener to print hitcount of the web-application which should persist across server shutdown.

ServletRequestListenerDemo.java

ServletContextListenerDemo.java

```
public class ServletContextListenerDemo implements ServletContextListener {
      public void contextDestroyed(ServletContextEvent event) {
            System.out.println("context destroyed ");
            String path = event.getServletContext().getRealPath("abc.txt");
```

```
                try {
                        PrintWriter out=new PrintWriter(path);
                        out.print(ServletRequestListenerDemo.count);
                        out.flush();
                } catch (Exception e) {}
        }
        public void contextInitialized(ServletContextEvent event) {
                System.out.println("context initialized ");
                String path = event.getServletContext().getRealPath("abc.txt");
                 try {
                        BufferedReader br = new BufferedReader(new FileReader(path));
                        String s = br.readLine();
                        if(s != null)  {
                                int c = Integer.parseInt(s);
                                ServletRequestListenerDemo.count = c;
                        }
                 }catch (Exception e) {}
        }
}
```

ServletContext.java

```
public class ServletContext extends HttpServlet {
        public void doGet(HttpServletRequest request, HttpServletResponse response)
                                        throws ServletException, IOException {
                response.setContentType("text/html");
                PrintWriter out = response.getWriter();
                out.println("Hitcount persist acoss server shutdown
                                        in Servlet context listener :"+ServletRequestListenerDemo.count);
        }
}
```

web.xml

```
 <listener>
        <listener-class>listener.ServletContextListenerDemo</listener-class>
 </listener>
 <servlet>
        <servlet-name>ServletContext</servlet-name>
        <servlet-class>listener.ServletContext</servlet-class>
 </servlet>
 <servlet-mapping>
        <servlet-name>ServletContext</servlet-name>
        <url-pattern>/context</url-pattern>
 </servlet-mapping>
<listener>
        <listener-class>listener.ServletRequestListenerDemo</listener-class>
</listener>
```

- We can configure more than one listener of the same type but the order of the execution is based on the order of Listener tag in the web.xml
- If both listeners are 2 different types the order is not important.

Demo Program on ServletContextListener

It is not a Dog object in realtime it is DataSource object.

Dog.java

```
public class Dog {
        public String breed;
        public Dog(String breed) {
                this.breed = breed;
        }
        public String getBreed() {
                return breed;
        }
}
```

ServletContextListenerDog.java

```
public class ServletContextListenerDog implements ServletContextListener {
            public void contextInitialized(ServletContextEvent event) {
                    System.out.println("context listener dog initialized");
                    ServletContext context=event.getServletContext();
                    String dogBreed = context.getInitParameter("breed");
                    Dog d = new Dog(dogBreed);
                    context.setAttribute("dog",d);
            }
            public void contextDestroyed(ServletContextEvent event) {
                            System.out.println("context listener dog destroyed");
            }
}
```

ServletContextDog.java

```
public class ServletContextDog extends HttpServlet {
            public void doGet(HttpServletRequest request, HttpServletResponse response)
                                        throws ServletException, IOException {
                    response.setContentType("text/html");
                    PrintWriter out = response.getWriter();
                    out.println("Servlet Context Listener  Dog ");
                    Dog d = (Dog) getServletContext().getAttribute("dog");
```

```
                  out.println("The dog breed is : "+d.getBreed());
        }
}
```

web.xml

```
 <listener>
           <listener-class>listener.ServletContextListenerDog</listener-class>
              //The whole point is to be initialized the app'n before any servlet is initialized
</listener>
 <context-param>
         <param-name>breed</param-name>
        <param-value>Great Puppy</param-value>
 </context-param>
 <servlet>
          <servlet-name>ServletContextDog</servlet-name>
          <servlet-class>listener.ServletContextDog</servlet-class>
 </servlet>
 <servlet-mapping>
         <servlet-name>ServletContextDog</servlet-name>
         <url-pattern>/contextdog</url-pattern>
 </servlet-mapping>
```

1. Container read the deployment descriptor for the application including <listener> and <context-param> elements.
2. Container creates a new ServletContext object for this application that all parts of application will share.
3. Container creates name/value pairs of strings for each context init-parameters.
4. Container gives the ServletContext reference to the name value parameters.
5. Container creates a new instance of the ServletContetListenerDemo class.
6. Container calls the listener context initialized method by passing ServletContextEvent, the EventObject has a reference to the ServletContext so the Eventhandling code can get the context form the event & get the ContextInitParameters from the Context.
7. Listener asks ServletContextEvent for a reference to ServletContext.
8. Listener asks ServletContext for the ContextInitParameter(breed).
9. Listener uses this initParameter to construct a new Dog object.
10. Listener sets the dog object as attributes in the servlet context scope.
11. Container makes a new servlet(i.e., makes a new ServletConfig) with initParameter gives a ServletConfig reference to the ServletContext, when it calls servlet init method.
12. Servlet gets a request and ask the ServletContext for the attribute "dog".
13. Servlets calls getBreed() on the Dog object.

ServletContextAttributeListener

This listener listens events related to context scoped attributes , i.e., attribute added in the context scope or replacement or removal. This interface defines the following methods

1. public void attributeAdded(ServletContextAttributeEvent e)

2. public void attributeRemoved(ServletContextAttributeEvent e)
3. public void attributeReplaced(ServletContextAttributeEvent e)

ServletContextAttributeEvent

It is child class of ServletContextEvent, it contains the following 2 methods.

1. public String getName()
2. public Object getValue()

Session Listeners:

1) **HttpSessionListener**

HttpSessionListener listens lifecycle events of session object like session object creation and destruction, this interface defines the following 2 methods.

- public void sessonCreated(HttpSessionEvent e)
- public void sessonDestroyed(HttpSessionEvent e)

HttpSessionEvent

It is the child class of java.util.EventObject, it contains only one method.

public HttpSession getSession()

HttpSessionListenerDemo.java

```
public class HttpSessionListenerDemo implements HttpSessionListener {
        static int count;
        public void sessionCreated(HttpSessionEvent event) {
              System.out.println("new session object created at :"+new  java.util.Date());
              count++;
        }
        public void sessionDestroyed(HttpSessionEvent event) {
               System.out.println("session object is destroyed :"+ new java.util.Date());
               count--;
        }
}
```

SessionServlet.java

```
public class SessionServlet extends HttpServlet {
          public void doGet(HttpServletRequest request, HttpServletResponse response)
                                        throws ServletException, IOException {
                 response.setContentType("text/html");
                 PrintWriter out = response.getWriter();
                 HttpSession session = request.getSession();
              //session.setMaxInactiveInterval(120);
                 out.println("The no, of users online is :"+HttpSessionListenerDemo.count);
        }
}
```

web.xml

```
 <listener>
         <listener-class>listener.HttpSessionListenerDemo</listener-class>
  </listener>
  <session-config>
        <session-timeout>3</session-timeout>
  </session-config>
  <servlet>
         <servlet-name>SessionServlet</servlet-name>
         <servlet-class>listener.SessionServlet</servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>SessionServlet</servlet-name>
    <url-pattern>/session</url-pattern>
  </servlet-mapping>
```

HttpSessionAttributeListener

This Listener listens events related to session scoped attributes like attribute addition in the session scope, or removal or replacement.

This interface defines the following methods

1) public void attributeAdded(HttpSessionBindingEvent e)
   This method will be executed automatically by the WC, when ever we are adding any type of objects in the session scope.
2) public void attributeReplaced(HttpSessionBindingEvent e)
   This method will be executed automatically by the WC, when ever we are replacing an existing object with new object.
3) public void attributeRemoved(HttpSessionBindingEvent e)
   This method will be executed automatically by the WC, when ever we are removed any type of attribute in the session scope.

Note: There is no class named with HttpSessionAttributeEvent for this requirement the request class is HttpSessionBindingEvent.

HttpSessionBindingEvent

It is the child class of HttpSessionEvent , and it contains 2 methods

- public String getName()
- public Object getValue()

HttpSessionBindingListener

When ever we are trying to add or remove or replacement a perticular type of object in session scope, if we want to perform any operation then we should go for HttpSessionBindingListener.

This interface defines the following 2 methods

1) public void valueBound(HttpSessionBindingEvent e)
   This method will be executed automatically by the WC, when ever we are trying to add a perticular type of object in session scope.

2) public void valueUnbound(HttpSessionBindingEvent e)
   This method will be executed automatically by the WC, when ever we are trying to remove a perticular type of object in session scope.

- For replacement operations both methods will be executed but valueBound() first and followed by valueUnbound().
- It is not required to configuure HttpSessionBindingListener in web.xml
- When ever we are trying to add an attribute in session scope , WC will check whether the corresponding class implements HttpSessinBindingListener or not , if it is implements HttpSessionBindingListener then valueBound() will be executed automatically by the WC.
- WC follows the same approach for attributeRemoval and attributeReplacement also.
- If we configure both attribute & binding listeners then binding listener will executed first followed by attribute listener.

<u>Demo program</u>

HttpSessionAttributeListenerDemo.java

```
public class HttpSessionAttributeListenerDemo implements HttpSessionAttributeListener {
        public void attributeAdded(HttpSessionBindingEvent event) {
                        System.out.println("Attributes added");
        }
        public void attributeRemoved(HttpSessionBindingEvent event) {
                        System.out.println("Attribute removed");
        }
        public void attributeReplaced(HttpSessionBindingEvent event) {
                        System.out.println("Attribute replaced");
        }
}
```

HttpSessionBindingListenerDemo.java

```
public class HttpSessionBindingListenerDemo implements HttpSessionBindingListener {
          public void valueBound(HttpSessionBindingEvent event) {
                System.out.println("Session object has  added  to the Session scope : bound ");
          }
          public void valueUnbound(HttpSessionBindingEvent event) {
                System.out.println("Session object has removed to the Session scope : unbound ");
          }
}
```

ServletSessionAttributeBind.java

```
public class ServletSessionAttributeBind extends HttpServlet {
        public void doGet(HttpServletRequest request, HttpServletResponse response)
                                        throws ServletException, IOException {
            response.setContentType("text/html");
            PrintWriter out = response.getWriter();
            HttpSession session=request.getSession();
            session.setAttribute("a1", "SCJP");
            session.setAttribute("a1", "SCWCD");
            session.setAttribute("a2", new HttpSessionBindingListenerDemo());
            session.setAttribute("a3", new HttpSessionBindingListenerDemo());
            session.setAttribute("a2", new HttpSessionBindingListenerDemo());
            session.removeAttribute("a2");
            session.setAttribute("a1", new HttpSessionBindingListenerDemo());
            out.println("HttpSessionBindingListener not required to configure");
            }
}
```

web.xml

```
 <listener>
        <listener-class>listener.HttpSessionAttributeListenerDemo</listener-class>
 </listener>
 <servlet>
         <servlet-name>ServletSessionAttributeBind</servlet-name>
         <servlet-class>listener.ServletSessionAttributeBind</servlet-class>
 </servlet>
 <servlet-mapping>
          <servlet-name>ServletSessionAttributeBind</servlet-name>
          <url-pattern>/sessionbind</url-pattern>
 </servlet-mapping>
```

HttpSessionActivationListener

- If a web-application is destributed across several JVM's such type of web-application is called distributed web applications.

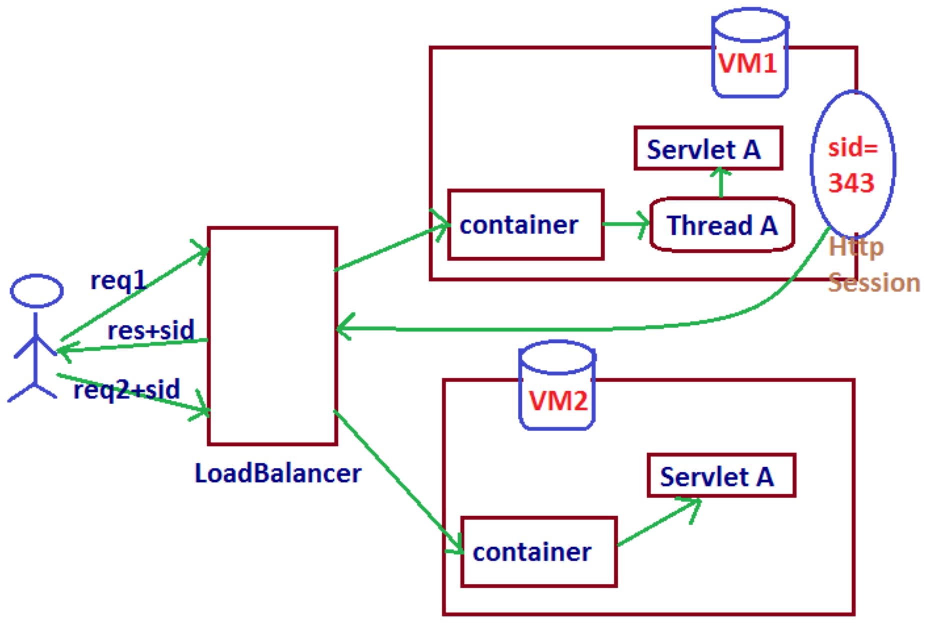
- The main advantage of distributed web-applications are --
  By Load Balancing: we can improve performance of the application.
  By Handling: fail over situations we can keep our application has robust.

- In distributed web-applications the session object is required to migrate from one JVM to another JVM.
- When ever a session object is migrating from one JVM to another JVM , the corresponding session scoped attributes also will migrate across the network , hence session scoped attributes should serializable.
- At the time of sesson object migration , if want to perform any operation then we should go for HttpSessionActivationListener.

This Listener defines the following 2 methods.

1) **<u>public void sessionWillPassivate(HttpSessionEvent e)</u>**
   This method is called on each implementing object bound to the session just before serialization.
2) **<u>public void sessionDidActivate(HttpSessionEvent e)</u>**
   This methode wii be executed on each implements object bound to the session just after de-serialization.

The session id 343 migrates from one VM1 to another VM2, in other words it is no longer exists on VM1 once it is moves to VM2. This migration means the session was passivated on VM1 and activated on VM2.



The container get the request the corresponding session id and realize this session id is on a different virtual machine i.e., VM1

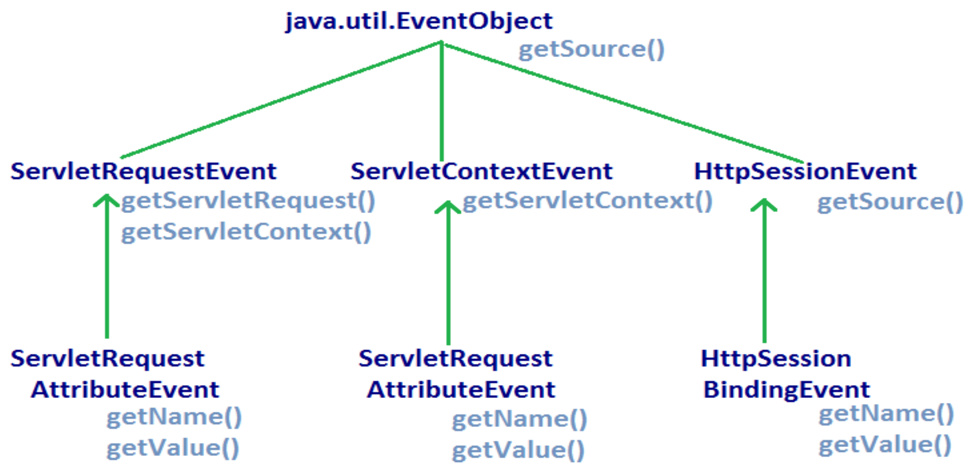
<u>Q : requestA for SercletA could happend for VM1 and requestB for ServletA could endup and different VM , what happens to the things ServletContext, ServletConfig and HttpSession objects ?</u>

Only HttpSession object (and their attributes) moves from one VM to another VM.
There is one ServletContext per one JVM, there is one ServletConfig per Servlet,per VM but there is only one HttpSession object for a given session id per web-app'n, regardless of how many virtual machines the app'n distributed across.

<u>Note:</u> HttpSessionActivationListener is also not required to configure in web.xml

HttpSessionActivationListenerDemo.java

```
public class HttpSessionActivationListenerDemo implements HttpSessionActivationListener {
        public void sessionDidActivate(HttpSessionEvent arg0) {
            System.out.println("object bound to the session just after De-Serialization");
        }
        public void sessionWillPassivate(HttpSessionEvent arg0) {
            System.out.println("object bound to the session just before Serialization");
        }
}
```



Listener	Purpose	Corresponding Methods	Corresponding Events	Corresponding Event Methods	Is required to configure web.xml
Servlet Request Listener	To listens life cycle events of request object. i.e., request object creation & destruction	requestInitialized() requestDestroyed()	ServletRequest Event	getServletRequest() getServletContext()	Yes
Servlet Request Attribute Listener	To listens events related request scoped attributes	attributeAdded() attributeReplaced() attributeRemoved()	ServletRequest AttributeEvent	getName() getValue()	Yes
Servlet Context Listener	To listens life cycle events of context object. i.e., context object creation & destruction	contextInitialized() contextDestroyed()	ServletContext Event	getServletContext()	Yes

Servlet Context Attribute Listener	To listen events related to context scoped attributes	attributeAdded() attributeReplaced() attributeRemoved()	ServletContext AttributeEvent	getName() getValue()	Yes
HttpSession Listener	To listens life cycle events of session object. i.e., session object creation & destruction	sessionCreated() sessionDestroyed()	HttpSession Event	getSession()	Yes
HttpSession Attribute Listener	To listen events related to session scoped attributes	attributeAdded() attributeReplaced() attributeRemoved()	~~HttpSession AttributeEvent~~ HttpSession BindingEvent	getName() getValue()	Yes
HttpSession Binding Listener	When ever we are adding or removing or replacing a perticular type of object in session scope , to perform certain activity then we should go for this Listener	valueBound() valueUnbound()	HttpSession BindingEvent	getName() getValue()	not required
HttpSession Activation Listener	If we want to perform any activity just before serialization and just after de-serialization in distributed web-applications	sessionWillPassivate() sessionDidActivate()	HttpSession Event	getSession()	not required