OBJECT™
TECHNOLOGIES

Servlet III

What will be covered

OBJECT TECHNOLOGIES

- Attributes in servlet
- Handling listeners
- Need of state management
- What is state management
- State management techniques
- What are cookies
- Cookie functionalitites

Attributes in servlet

OBJECT

- Generally, a parameter is a string value that is most commonly known for being sent from the client to the server (e.g. a form post) and retrieved from the servlet request.
- There are even other types of parameters which are specific to particular servlet like servlet initial parameters and ServletContext initial parameters which are string parameters that are configured in web.xml and exist on the server.
- But attributes are in the form of object.
- Strings are even objects, so attributes can be in the form of string also.
- Parameters and attributes both are in the form of name and value pairs.
- For retrieving them, name needs to be specified.

Attributes in servlet

OBJECT

- An attribute is an object that is used to share information in a web app.
- Attribute allows Servlets to share information among themselves.
- Attributes can be SET and GET from one of the following scopes :
  - request
  - session
  - application
- public void setAttribute(String name, Object obj) method is used to SET an Attribute.

```
ServletContext sc = getServletContext();
sc.setAttribute("user","Abhijit");  //setting attribute on context
```

- Object getAttribute(String name) method is used to GET an attribute.

Attributes in servlet

OBJECT

```
ServletContext sc = getServletContext();

String str = sc.getAttribute("user");   //getting attribute

out.println("Welcome"+str);  // Prints : Welcome Abhijit
```

- Note : In the above examples context object can be replaced by request or session object to set or get the attributes from request and session scope respectively.
- **Request Scope:**
- Request Scope starts when an HTTP request hits the Servlet and ends when the Servlet sends an HTTP response.
- Request Scope is denoted as javax.servlet.http.HttpServletRequest interface.
- Web Container uses the request object as an argument of type HttpServletRequest.

Attributes in servlet

- **Session Scope:**
- Session scope starts when a connection is established between the user and the web application and remains until the browser is closed.
- Session Scope is denoted as javax.servlet.http.HttpSession interface.
- Session object can be retrieved using request.getSession().
- **Application Scope:**
- Application scope starts when the browser is open and stops when the browser is closed.
- Application Scope is denoted as javax.servlet.ServletContext interface.
- Application object can be retrieved using getServletContext() directly or by making an explicit call to getServletConfig().getServletContext().

Handling listeners

OBJECT TECHNOLOGIES

- Application events provide notifications of a change in state of the servlet context (each Web application uses its own servlet context) or of an HTTP session object.
- Event listener classes are written that respond to these changes in state, and can be configured and deployed them in a Web application.
- The servlet container generates events that cause the event listener classes to do something. In other words, the servlet container calls the methods on a user's event listener class.
- The following is an overview of this process:
  - The user creates an event listener class that implements one of the listener interfaces.
  - This implementation is registered in the deployment descriptor or using annotation.
  - At deployment time, the servlet container constructs an instance of the event listener class. (This is why the public constructor must exist, as discussed in Writing an Event Listener Class.)
  - At runtime, the servlet container invokes on the instance of the listener class

Handling listeners

- For servlet context events, the event listener classes can receive notification when the Web application is deployed or undeployed (or when WebLogic Server shuts down), and when attributes are added, removed, or replaced.
- For HTTP session events, the event listener classes can receive notification when an HTTP session is activated or is about to be passivated, and when an HTTP session attribute is added, removed, or replaced.
- For servlet request events, the event listener classes can receive notification when request is initialized and destroyed and when request attribute is added, removed or replaced.

Handling listeners

OBJECT

Following tables lists the types of events, the interface event listener class must implement to respond to the event, and the methods invoked when the event occurs

Type of Event	Interface	Method
Servlet context is created.	`javax.servlet.ServletContextListener`	`contextInitialized()`
Servlet context is about to be shut down.	`javax.servlet.ServletContextListener`	`contextDestroyed()`
An attribute is added.	`javax.servlet.ServletContextAttributesListener`	`attributeAdded()`
An attribute is removed.	`javax.servlet.ServletContextAttributesListener`	`attributeRemoved()`
An attribute is replaced.	`javax.servlet.ServletContextAttributesListener`	`attributeReplaced()`
An HTTP session is activated.	`javax.servlet.http.HttpSessionListener`	`sessionCreated()`
An HTTP session is about to be passivated.	`javax.servlet.http.HttpSessionListener`	`sessionDestroyed()`

Handling listeners

OBJECT

`void requestInitialized(ServletRequestEvent sre)`	Receive notification that the request is about to come into scope within the web application.
`void requestDestroyed(ServletRequestEvent sre)`	Receive notification that the request is about to go out of scope within the web application.

- Event listener classes can be used in web application to:
  - Manage database connections when a Web application is deployed or shuts down
  - Create standard counter utilities
  - Monitor the state of HTTP sessions and their attributes

Handling listeners

OBJECT

- Create a listener class implementing appropriate listener interface

```
@WebListener
public class MyContextListener implements
                        ServletContextListener {
```

- @WebListener annotation is used to register a class as a listener of a web application. The annotated class must implement one or more of the listener interfaces.
- Implement the methods in the listener interface for implementing the required task. e.g. Database connectioncan be established in contextInitialized and can be closed in contextDestroyed method.

Handling listeners

OBJECT

```
public void contextInitialized(ServletContextEvent sce)  {
     //con established
    //read context parameters and first 2 steps of JDBC
    try
    {
        String driver = sce.getServletContext().getInitParameter("driver");
        Class.forName(driver);
        String jdbcurl = sce.getServletContext().getInitParameter("url");
        String user = sce.getServletContext().getInitParameter("user");
        String pwd = sce.getServletContext().getInitParameter("pwd");
        con = DriverManager.getConnection(jdbcurl,user,pwd);
        System.out.println("connected to db");
        sce.getServletContext().setAttribute("jdbccon", con );
        System.out.println("connection set as ctx attribute");
    }
    catch(Exception e)
    {
        e.printStackTrace();
    }
```

Handling listeners

OBJECT

```
//this web app is undeployed
public void contextDestroyed(ServletContextEvent sce)   {
    // con close
    try
    {
        con.close();
    }
    catch(Exception e)
    {
        e.printStackTrace();
    }
}
```

Need of state management

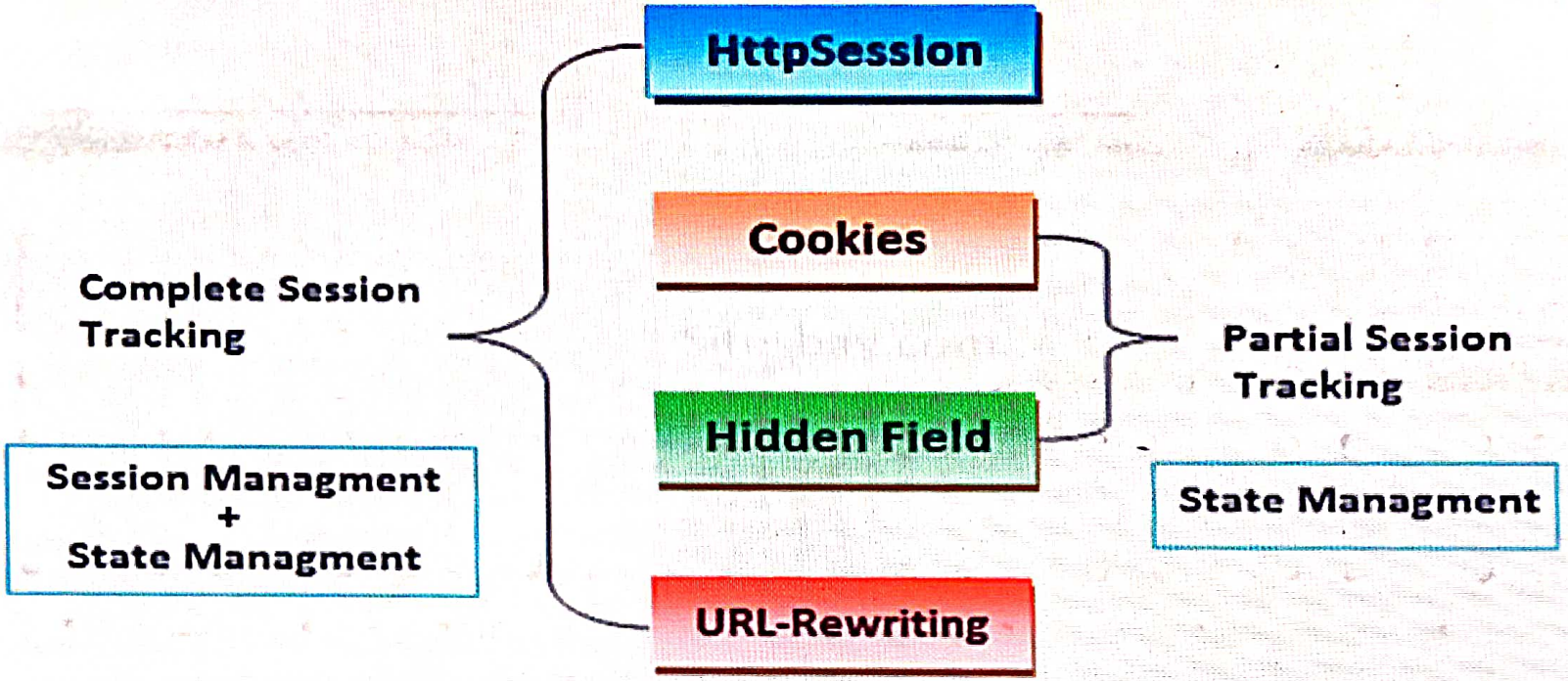
OBJECT

- Web applications are stateless. A user request is received to display a Web page and a Web server obliges by sending the HTML to the user's browser to display.
- It is important to realize that at this point the communication is over until the user requests another page.
- When the next user request comes in the Web server has no idea who the user is. The Web server doesn't know what pages it has sent before to that user or if the user has ever been sent any page at all.
- After the Web server processes each user request it wipes the slate clean and gets ready for the next incoming request, be it from you or someone else.
- If the Web server is continually wiping its memory clean after every request, how can it know who you are and how (or if) it has served you in the past?

What is state management

- An every request from the client is always considered as fresh or new request, there is a need to store information to help web server remember who you are.
- State is what an application knows about the user, their current interaction with the application, and other pieces of global information like who the user is, where they are in the application, what information they have entered so far, and other application configuration information.
- Data generated in the request- response cycle can be saved for further reference either on the client side or on the server side.
- Data saved as a state information is always in the form of name-value pairs.

State management techniques

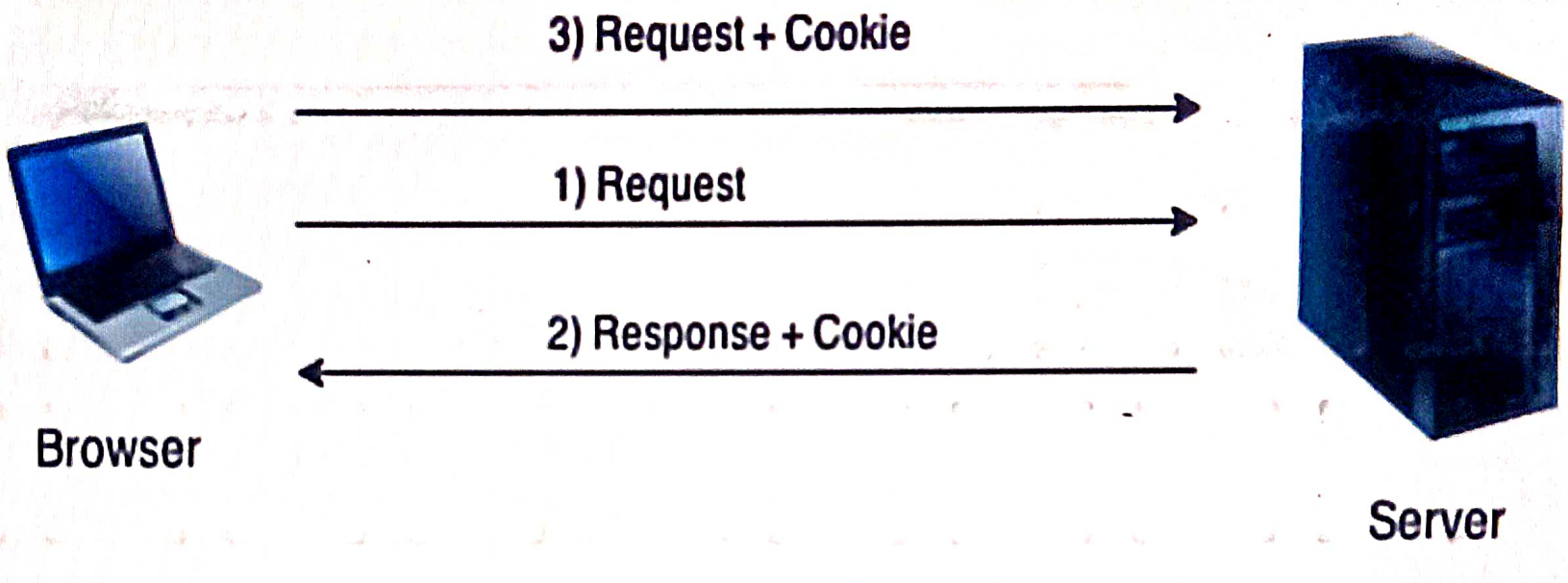


- Since HTTP and Web Server both are stateless, there are differnt ways to keep the information on client as we as server. Some of the common ways of state management in servlets are:
  - HTML Hidden Field
  - Cookies
  - URL Rewriting
  - Session Management API

State management techniques

- **URL Rewriting**
- In URL rewriting, a token or identifier is appended to the URL of the next Servlet or the next resource. We can send parameter name/value pairs using the following format:
- url?name1=value1&name2=value2&??
- A name and a value is separated using an equal = sign, a parameter name/value pair is separated from another parameter using the ampersand(&). When the user clicks the hyperlink, the parameter name/value pairs will be passed to the server. From a Servlet, we can use getParameter() method to obtain a parameter value
- **HTML Hidden Field**
- In case of Hidden Form Field a hidden (invisible) textfield is used for maintaining the state of a user.
- In such a case, we store the information in the hidden field and get it from another servlet. This approach is better if we have to submit form in all the pages and we don't want to depend on the browser.
- Let's see the code to store value in hidden field.
- <input type="hidden" name="uname" value="Edureka">
- Here, uname is the hidden field name and Edureka is the hidden field value.

What are cookies



- A cookie is a small piece of information that is persisted between the multiple client requests stored on client side.
- A cookie has a name, a single value, and optional attributes such as a comment, path and domain qualifiers, a maximum age, and a version number.
- In cookies technique, we add cookie with response from the servlet. So cookie is stored in the cache of the browser. 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.

Cookie functionalitites

OBJECT

- **Creating cookie**
- To store a cookie in the web browser, first create a new Cookie object:
- String name = "Cookie name";
- String value = "Cookie value";

```
Cookie cookie = new Cookie(name, value);
```

- Then call the addCookie() method of the HttpServletResponseobject in a Servlet class like this:

```
response.addCookie(cookie);
```

- This will send appropriate HTTP headers to the client, and the browser stores the cookie on user's computer.
- Besides the mandatory attributes name and value, other methods of cookie can be used to specify additional information for a cookie.
- **Reading cookie**
- To read cookies sent from the browser to the server, call getCookies() method on a HttpServletRequest object in a Java servlet class. This method returns an array of Cookie objects that are visible to the current request

Cookie functionalitites

OBJECT

```
Cookie[] cookies = request.getCookies();

PrintWriter writer = response.getWriter();

for (Cookie aCookie : cookies) {
    String name = aCookie.getName();
    String value = aCookie.getValue();

    writer.println(name + " = " + value);
}
```

- If a specific cookie needs to read, you need to check the cookie's name in the loop.

```
for (Cookie aCookie : cookies) {
    String name = aCookie.getName();

    if (name.equals("username")) {
        username = aCookie.getValue();

        break;
    }
```

Cookie functionalitites

OBJECT

- Updating a cookie
- To update an existing cookie, you need to create a new cookie with the same name and add it to the response. This will overwrite (update) the cookie with the same name.

```
String name = "Cookie name";
String value = "New value";

Cookie cookie = new Cookie(name, value);

response.addCookie(cookie);
```

- Deleting a cookie
- To remove a cookie from the browser's cache, you need to create a new cookie with the same name, set its max age to zero and add it to the response.

```
Cookie cookie = new Cookie("username", "");
cookie.setMaxAge(0);
response.addCookie(cookie);
```

Cookie functionalitites

OBJECT

```
//create the cookie and redirect to login page
Cookie c = new Cookie("loginerror","Wrong_ID_pwd");
response.addCookie(c);
response.sendRedirect("/ShoppingApp/login.jsp");
```

```
//delete the cookie if available
Cookie [] carr = request.getCookies();
if(carr != null)
{
    for(Cookie c : carr)
    {
        if(c.getName().equals("loginerror"))
        {
            c.setMaxAge(0);
            response.addCookie(c);
        }
    }
}
```

Servlet III

In this chapter we will understand about how information can be shared between servlets and how state menagement techniques can be implemented in servlets.

Attributes and Listeners

Attributes :

An object, which can be set, get, or removed, is referred to as an Attribute. The Servlet attribute is used mainly to pass the information from one servlet to another or from one filter to another, etc. It is just like passing an object from one class to another so that we can reuse the same object again and again. In a way it is used to share information between multiple servlets and JSP.

Attributes in Servlet can get or set using any of the below scopes:

1. Request: Request Scope starts when an HTTP request hits the Servlet and ends when the Servlet sends an HTTP response. This is used while processing the results of a submitted form.
2. Session: This is used to create the session by the Web Container when a user visits the web application. Session level attributes will be valid till user session is active.
3. Application: This scope persists until the application is deployed.

Methods used :

- public void setAttribute(String name, Object object): This method will set Attribute provided object in the application scope.
- public Object getAttribute(String name): This method will return the attribute for the specified name.
- public void removeAttribute(String name): This will remove the attribute with the given name from the servlet context.

Listeners :

Servlet Listener is **used for listening to events in web containers**, such as when you create a session, insert an attribute, passivate and activate in another container. The servlet container generates events that trigger the action of event listener classes.

There are total eight type of listeners available in servlet framework which listens to a particular event and they are :

1. ServletContextListener
2. ServletContextAttributeListener
3. HttpSessionListener
4. HttpSessionAttributeListener

Methods used :

- public void setAttribute(String name, Object object): This method will set Attribute provided object in the application scope.
- public Object getAttribute(String name): This method will return the attribute for the specified name.
- public void removeAttribute(String name): This will remove the attribute with the given name from the servlet context.

Listeners :

Servlet Listener is **used for listening to events in web containers**, such as when you create a session, insert an attribute, passivate and activate in another container. The servlet container generates events that trigger the action of event listener classes.

There are total eight type of listeners available in servlet framework which listens to a particular event and they are :

1. ServletContextListener
2. ServletContextAttributeListener
3. HttpSessionListener
4. HttpSessionAttributeListener
5. ServletRequestListener
6. ServletRequestAttributeListener
7. HttpSessionActivationListener
8. HttpSessionBindingListener

Theses listeners are interfaces which needs to be implemented by classes and these classes should be annotated with @WebListener annotation. The @WebListener annotation is used to register a class as a listener of a web application.

Example of listener implementing ServletContextListener :

```
@WebListener
public class ShoppingAppListener implements ServletContextListener {
   Connection con;
   public void contextDestroyed(ServletContextEvent sce)  {
      try  {
         con.close();
      }
      catch(Exception e)  {
         e.printStackTrace();
      }
   }
```

Example of listener implementing ServletContextListener :

```
@WebListener
public class ShoppingAppListener implements ServletContextListener {
   Connection con;
   public void contextDestroyed(ServletContextEvent sce) {
      try {
         con.close();
      }
      catch(Exception e) {
         e.printStackTrace();
      }
   }
   public void contextInitialized(ServletContextEvent sce) {
      //reading from context level parameters
      String driver = sce.getServletContext().getInitParameter("diverclass");
      String url = sce.getServletContext().getInitParameter("jdbcurl");
      String user = sce.getServletContext().getInitParameter("user");
      String pwd = sce.getServletContext().getInitParameter("password");
      //establish the connection
      try {
         Class.forName(driver);
         con = DriverManager.getConnection(url, user, pwd);
         sce.getServletContext().setAttribute("jdbccon", con);
         System.out.println("con is set as ctx level attribute");
      }
      catch(Exception e) {
         e.printStackTrace();
      }
   }
}
```

contextInitialized() method gets executed when application is deployed and contextDestroyed() method gets executed when application is undeployed.

contextInitialized() method gets executed when application is deployed and contextDestroyed() method gets executed when application is undeployed.

---

state Management

Need of state management :

HTTP is a stateless (or non-persistent) protocol. Each request is treated by its own. i.e. Each request is considered as fresh request. A request will not know what was done in the previous requests. The protocol is designed to be stateless for simplicity. Since the protocol is stateless, it is the responsibility of the application to maintain state information within their application.

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.

What is state management

"State" is any data that describes the current behavior of an application. This could include values like "a list of objects fetched from the server", "which item is currently selected", "name of the currently logged-in user", and "is this modal open?".

"State Management" is the process of dealing with changes to state over time. This means having ways to:

- store an initial value
- read the current value
- update a value

In short state management is nothing but saving data on the server or client which might be required in future for processing further requests..

State menagement techniques

State Management is a mechanism used by the Web container to store information for a particular user. There are four different techniques used by Servlet application for session management.

They are as follows:

1. Hidden form field

State menagement techniques

State Management is a mechanism used by the Web container to store information for a particular user. There are four different techniques used by Servlet application for session management.

They are as follows:

1. **Hidden form field**

   We can create a unique hidden field in the HTML form and we can set its value unique to the user and keep track of the session. This method can't be used with links because it needs the form to be submitted every time request is made from client to server with the hidden field. Also it's not secure because we can get the hidden field value from the HTML source and use it to hack the session.

   In case of Hidden Form Field a hidden (invisible) field is used for maintaining the state of a user.

   In such a case, we store the information in the hidden field and get it from another servlet. This approach is better if we have to submit form in all the pages and we don't want to depend on the browser.

   ```
   <input type="hidden" name="uname" value="object">
   ```

   Here, uname is the hidden field name and object is the hidden field value

2. **URL Rewriting**

   It's a technique to maintain session state across multiple requests when cookies are disabled or not supported. It involves appending the session ID as a parameter to all URLs generated by the server.

   The browser includes this session ID in subsequent requests, allowing the server to associate the requests with the correct session. For every link or redirect generated by the server, it appends the session ID to the URL using either:

   ```
   HttpServletResponse.encodeURL(String url)
   HttpServletResponse.encodeRedirectURL(String url)
   ```

   The server extracts the session ID from the URL and retrieves the associated session object.

3. **Cookies**

   Cookies are small piece of information that is sent by web server in response header and gets stored in the browser cookies. When client make further request, it adds the cookie to the request header and we can utilize it to keep track of the session. We will see more details about it further.

4. **HttpSession**

3. **Cookies**
   Cookies are small piece of information that is sent by web server in response header and gets stored in the browser cookies. When client make further request, it adds the cookie to the request header and we can utilize it to keep track of the session. We will see more details about it further.
4. **HttpSession**
   It's the mechanism for maintaining state across multiple requests from the same user in a stateless HTTP environment. It allows web applications to associate data with a specific user session, providing a personalized experience. We will see more details about it further.

Handling Cookies

A cookie is a small amount of data which is stored in the web browser and transferred between requests and responses through HTTP headers. A cookie has a name and value, plus optional attributes like comment, path, domain, max age etc. Cookies are commonly used to implement simple, temporary data storage on the client side, such as session management, remember password, shopping cart items, etc. Cookies are generated by server and sent to the client along with the response. Browser stores cookies and sent is automatically to the server in the subsequent request. (Note : It is applicable for requests made to the same domain form where cookie is received)

Pros :

- All the data related to cookies are stored on the hard drive without the use of server resources. No extra load or weight is added to the server.
- Cookies are extremely user friendly. The client can choose what they need to do with cookies.
- Cookies can also set to be made available for a longer period of time.
- Cookies can also be configured as per the requirement.

Cons :

- Whenever a user surfs the web, more and more cookies will be accumulated. This eventually slows down or lags the browser.
- Since cookies are stored in the hard drive as text files, it posses some serious security risks. Any intruder can easily open these files and view the information.
- Size limitations also exist on cookies. They cannot store large amount of information. Most cookies are able to store information only up to 4kb.
- Whenever the user browses the internet, the cookie enabled sites will be recording all the online activities.
- Browsers also comes with the option to disable cookies. Users who are highly security conscious could simply disable them.

Cons :

- Whenever a user surfs the web, more and more cookies will be accumulated. This eventually slows down or lags the browser.
- Since cookies are stored in the hard drive as text files, it posses some serious security risks. Any intruder can easily open these files and view the information.
- Size limitations also exist on cookies. They cannot store large amount of information. Most cookies are able to store information only up to 4kb.
- Whenever the user browses the internet, the cookie enabled sites will be recording all the online activities.
- Browsers also comes with the option to disable cookies. Users who are highly security conscious could simply disable them. Even some browsers disable cookies automatically if the security level is set to high. Therefore, web applications will not work without cookies.

Creating cookie on server

1. Create a Cookie object:
   Cookie myCookie = new Cookie("name", "value");
2. Set optional attributes:
   setMaxAge(int seconds): Expiration time in seconds.
   setPath(String path): Path on the server where the cookie is accessible.
   setDomain(String domain): Domain for which the cookie is valid.
   setSecure(boolean secure): Send only over HTTPS connections.
3. Add the cookie to the response:
   response.addCookie(myCookie);

Retrieving cookie from request

Get an array of cookies from the request:

```
Cookie[] cookies = request.getCookies();
```

Use code with caution. Learn more

Iterate through cookies to find the desired one:

```
if (cookies != null) {
  for (Cookie cookie : cookies) {
    if (cookie.getName().equals("name")) {
      String value = cookie.getValue();
```

Retrieving cookie from request

Get an array of cookies from the request:

```
Cookie[] cookies = request.getCookies();
```

Use code with caution. Learn more

Iterate through cookies to find the desired one:

```
if (cookies != null) {
   for (Cookie cookie : cookies) {
      if (cookie.getName().equals("name")) {
         String value = cookie.getValue();
         // Process the cookie value
      }
   }
}
```

Updating cookie

Create a new cookie with the same name and updated value, and add it to the response. The browser will overwrite the old cookie with the new one.

Deleting cookie

Set the cookie's expiration time to 0:

```
myCookie.setMaxAge(0);
response.addCookie(myCookie);
```

Assignments

1. Create a shppoing application and provide following features
   a. Establish the database connection in a context listener by reading context parameters for name of driver class, conection string , user name and password.
   b. Provide the login feature which autheticates user from database
   c. After successful login forward request to Home page which displays the information about existing categories in the form of hyperlinks.
   d. Clicking category hyperlink displays respective products as drop down list
2. Display a visit count on the web page. Provide a referesh link to display the visited count(Use cookies)

Deleting cookie

Set the cookie's expiration time to 0:

```
myCookie.setMaxAge(0);
response.addCookie(myCookie);
```

Assignments

1. Create a shppoing application and provide following features
   a. Establish the database connection in a context listener by reading context parameters for name of driver class, conection string , user name and password.
   b. Provide the login feature which autheticates user from database
   c. After successful login forward request to Home page which displays the information about existing categories in the form of hyperlinks.
   d. Clicking category hyperlink displays respective products as drop down list
2. Display a visit count on the web page. Provide a referesh link to display the visited count(Use cookies)
3. Display error message on the login page in case of failed login. Messages should be stored in cookies. After successful login remove the cookie for storing the msg of failed login
4. Display list of depts on the web page as hyperlinks. After clicking on the hyperlink display a list of all the employees in that dept in the form of table. Consider following-
   a. Establish the database connection in a context listener event by reading context parameters for name of driver class, conection string , user name and password.
   b. Set database connection as a context attribute and use this connection from above servlets.
5. Create quiz application and provide login and registration feature. After successful login display all the subjects as in the form of hyperlinks. When user selects the subject, display the question one at a time on the page and allow user to move to next quesion.
   a. Establish the database connection in a context listener event by reading context parameters for name of driver class, conection string , user name and password.
   b. Set database connection as a context attribute and use this connection from above servlets.