Session Management

Table of Contents

[Cookies 2](#_Toc100552441)

[HttpSessions 4](#_Toc100552442)

[URL Re-Writing 6](#_Toc100552443)

[Hidden Form Field 7](#_Toc100552444)

There are mainly four methods that are used to **manage sessions** in servlets. Two of these, using Cookie objects and using *HttpSession* objects, are used only if the client has **cookies enabled**. The other two, **URL Re-Writing** and **Hidden Form Fields** are used only if the client has **cookies disabled**.

## Cookies

As we have discussed before, **Cookies** are **temporary files** stored on the **client’s browser**. The steps involved in handling cookies are as follows:

1. Create a cookie with some value and an expiry time
2. Add the cookie to a response object
3. For subsequent requests, check if the cookie is valid
4. On user logout, invalidate the cookie

Suppose a user logs in. The servlet which handles logins will verify that the credentials are correct and then attach a cookie to the response. The cookie’s value will be the unique username of the user. Afterwards, the user can be sent to a new page automatically.

Cookie usernameCookie = new Cookie("currentUser", username);  
usernameCookie.setMaxAge(30 \* 60); // 30 minutes  
response.addCookie(usernameCookie);  
  
*RequestDispatcher* rd = request.getRequestDispatcher("LoginSuccessful");  
rd.forward(request, response);

JAVA

Now, suppose, instead of sending the user to a **servlet** (like we have above), we send them directly to an **HTML page**. What would happen? Since the HTML page is a resource, we would have no way of verifying the user’s credentials. Because of this, anyone could directly enter the path to the HTML page and access the resource without logging in. To avoid this, we have used a servlet which will send a respond with the required HTML code.

When **verifying** the cookie, we actually have to first retrieve **all the cookies** in an array and loop through them to find the right one.

Cookie [] cookies = request.getCookies();  
String user = null;  
  
if (cookies == null) response.sendRedirect("Login.html");  
else {  
 for (Cookie cookie : cookies) {  
 if (cookie.getName().equals("currentUser")) user = cookie.getValue();  
 }  
}

JAVA

Finally, we can delete the cookie again by first retrieving it from the array and then setting its **expiry time** to 0. There is no explicit method to delete a cookie.

Cookie [] cookies = request.getCookies();  
Cookie currentUserCookie = null;  
if (cookies != null) {  
 for (Cookie cookie:cookies) {  
 if (cookie.getName().equals("currentUser"))currentUserCookie = cookie;  
 }  
}  
if (currentUserCookie != null) {  
 currentUserCookie.setMaxAge(0);  
 response.addCookie(currentUserCookie);  
}  
response.sendRedirect("Login.html");

JAVA

## HttpSessions

If we choose to use an *HttpSession* object instead of a Cookie object, the method of handling things remains the same. However, the *HttpSession* object provides several benefits. This object creates a **unique ID** for each client, unlike the Cookie object, where we have to essentially create the unique ID ourselves. This unique ID is automatically stored inside an auto-generated Cookie object with JSESSIONID as the key and the unique ID as the value.

protected void doPost(*HttpServletRequest* request, *HttpServletResponse* response) throws ServletException, IOException {  
 String username = request.getParameter("username-field");  
  
 *HttpSession* session = request.getSession();  
 session.setAttribute("currentUser", username); // we can add more attributes  
 session.setMaxInactiveInterval(30 \* 60); // expiry time in seconds  
  
 *RequestDispatcher* rd = request.getRequestDispatcher("loginsuccessful");  
 rd.include(request, response);  
}

JAVA

The request.getSession() method creates a new session if one does not already exist. Otherwise, it retrieves the old one.

*HttpSession* objects include several other important methods:

|  |  |
| --- | --- |
| **Methods** | **Description** |
| getSession() | Always returns a session. Returns a previous session if it already exists. Otherwise, returns a new session. |
| getCreationTime() | Returns the time when the session was created, measured in milliseconds, since midnight January 1, 1970 GMT. |
| getId() | Returns a string containing the unique identifier assigned to the session. |
| getLastAccessedTime() | Returns the last time the client sent a request associated with the session. |
| getMaxInactiveInterval() | Returns the maximum time interval, in seconds. |
| invalidate() | Destroys the session. |
| isNew() | Returns true if the session is new. |
| setMaxInactiveInterval(interval) | Specifies the time in seconds after which the servlet container will invalidate the session. |

## URL Re-Writing

If the client has **cookies disabled**, we cannot use cookies. Instead, we can make use of **URL Re-Writing**, which involves adding the JSESSIONID cookie created by an *HttpSession* object and attaching it to the URL using either the encodeURL() method or the encodeRedirectURL() method if the client is simultaneously being redirected. Literally nothing else changes.

protected void doPost(*HttpServletRequest* request, *HttpServletResponse* response) throws ServletException, IOException {  
 String username = request.getParameter("username-field");  
  
 *HttpSession* session = request.getSession();  
 session.setAttribute("currentUser", username);  
 session.setMaxInactiveInterval(30 \* 60);  
  
 // encoding with JSESSIONID  
 String encodedUrl = response.encodeURL("LoginSuccessful");  
 response.sendRedirect(encodedUrl);  
}

The encoded URL looks like this:

[www.example.com/jsessionid=1234](http://www.example.com/jsessionid=1234)

The good thing is URL re-writing only gets activated if the client has cookies disabled, so it is like a fallback approach.

## Hidden Form Field

A final and unrecommended approach is to simply add a **hidden field** inside a form to store the session identifier. This is not recommended because this adds the extra overhead of having a form on every page.

protected void doPost(*HttpServletRequest* request, *HttpServletResponse* response) throws ServletException, IOException {  
 String username = request.getParameter("username-field");  
  
 PrintWriter out = response.getWriter();  
 out.println("<form action=\"SecondServlet\">\n" +  
 " <input type=\"hidden\" name=\"user\" value=\"" + username + "\">\n" +  
 " <input type=\"submit\" value=\"submit\">\n" +  
 " </form>");  
}

JAVA