Servlets and Sessions

## Jsessionid

JSESSIONID is **a cookie generated by Servlet containers and used for session management in J2EE web applications for HTTP protocol**. If a Web server is using a cookie for session management, it creates and sends JSESSIONID cookie to the client and then the client sends it back to the server in subsequent HTTP requests.

### [Automatic Parameterization of JSESSIONID Cookie Values](https://www.loadtestingtool.com/help/jsessionid-cookie-values.shtml" \l ":~:text=JSESSIONID%20is%20a%20cookie%20generated,server%20in%20subsequent%20HTTP%20requests.)

## How to create a cookie?

**How to create Cookie?**

1. Cookie ck=new Cookie("user","sonoo jaiswal");//creating cookie object.
2. response.addCookie(ck);//adding cookie in the response.

## HTTP location header field:

https://en.wikipedia.org/wiki/HTTP\_location

The **HTTP Location** [header field](https://en.wikipedia.org/wiki/HTTP_header) is returned in responses from an [HTTP](https://en.wikipedia.org/wiki/HTTP) [server](https://en.wikipedia.org/wiki/Server_(computing)) under two circumstances:

1. To ask a web browser to load a different web page ([URL redirection](https://en.wikipedia.org/wiki/URL_redirection)). In this circumstance, the Location header should be sent with an [HTTP status code](https://en.wikipedia.org/wiki/HTTP_status_code) of 3xx. It is passed as part of the response by a web server when the requested [URI](https://en.wikipedia.org/wiki/URI) has:
   * Moved temporarily;
   * Moved permanently; or
   * Processed a request, e.g. a POSTed form, and is providing the result of that request at a different URI
2. To provide information about the location of a newly created resource. In this circumstance, the Location header should be sent with an HTTP status code of [201 or 202](https://en.wikipedia.org/wiki/List_of_HTTP_status_codes#2xx_success).[[1]](https://en.wikipedia.org/wiki/HTTP_location#cite_note-RESTful_Web_Services-1)

## PD-S-Session-id

 A different **PD**-**S**-**SESSION**-**ID** cookie is generated for unauthenticated **sessions**, **sessions** in process of authentication, and authenticated **sessions**.

<https://community.ibm.com/community/user/security/communities/community-home/digestviewer/viewthread?GroupId=2863&MessageKey=ca1c2694-1d21-4808-b871-064911323c94&CommunityKey=e7c36119-46d7-42f2-97a9-b44f0cc89c6d&tab=digestviewer&ReturnUrl=%2Fcommunity%2Fuser%2Fibmz-and-linuxone%2Fgroups%2Fnewtoz%2Fntz-discussions>

In a situation where I have an authenticated web session with both PD-S-SESSION-ID & PD-ID cookies set on the browser, when a call to pkmslogout is done, is it normal that I receive a "Set-Cookie: PD-ID=; Max-Age=0; Domain=blahblah; Path=/; Expires="Sun, 01-Jan-1995 01:00:00 GMT"; Secure; HttpOnly" for the PD-ID but nothing for the PD-S-SESSION-ID?

If so, I am Assuming that the session represented by the PD-S-SESSION-ID is nevertheless invalidated in ISAM. What would be the simplest most direct way to proove this? Is there a way to question ISAM directly on the status of a session giving it the value of the PD-S-SESSION-ID token?

Trying to access a protected ressource triggers ISAM to refuse the access to the ressource and send back another set-cookie for PD-S-SESSION-ID but with a different value. Am I right in assuming that this new PD-S-SESSION-ID represents a (new) unauthenticated session?

also, if required is there a setting that would force ISAM to send a "Set-Cookie: PD-S-SESSION-ID=; Max-Age=0;" similar to the one for PD-ID when pkmslogout is used? And why is that not the case by default, any issues/draw back to this that I am not seeing?

### Configuring session cookie names

https://www.ibm.com/docs/en/sva/9.0.6?topic=sharing-configuring-session-cookie-names

The cookie name used for WebSEAL session cookies is specified by the **tcp-session-cookie-name** and **ssl-session-cookie-name** stanza entries in the **[session]** stanza of the WebSEAL configuration file. For example (default WebSEAL cookie names for TCP and SSL sessions):

[session]

tcp-session-cookie-name = PD-H-SESSION-ID

ssl-session-cookie-name = PD-S-SESSION-ID

### Configuring DNS domains

<https://www.ibm.com/docs/en/sva/9.0.6?topic=configuration-standard-webseal-junctions>

[session-cookie-domains]

domain = example.com

domain = abc.ibm.com

## setHeader(), addHeader()

https://stackoverflow.com/questions/53071229/httpclient-what-is-the-difference-between-setheader-and-addheader

As you can read from documentation:

[addHeader(String name, String value](http://hc.apache.org/httpcomponents-core-ga/httpcore/apidocs/org/apache/http/message/AbstractHttpMessage.html?is-external=true#addHeader(java.lang.String,%20java.lang.String))

Adds a header to this message. The header will be appended to the end of the list.

[setHeader(String name, String value](http://hc.apache.org/httpcomponents-core-ga/httpcore/apidocs/org/apache/http/message/AbstractHttpMessage.html?is-external=true#setHeader(org.apache.http.Header))

Overwrites the first header with the same name. The new header will be appended to the end of the list, if no header with the given name can be found.

## What if the user doesn’t accept cookies?

<https://stackoverflow.com/questions/65796479/how-to-do-sessions-if-client-doesnt-accept-cookies-understanding-response-enco#:~:text=When%20a%20client%20will%20not%20accept%20a%20cookie%2C,as%20a%20path%20parameter%20in%20the%20URL%20string>.

1

As per [the specification](https://download.oracle.com/otn-pub/jcp/servlet-3.0-fr-oth-JSpec/servlet-3_0-final-spec.pdf), the server should support a few ways of tracking sessions: with cookies, SSL sessions, or URL rewriting.

You are asking about URL rewriting, which works like this:

URL rewriting is the lowest common denominator of session tracking. When a client will not accept a cookie, URL rewriting may be used by the server as the basis for session tracking. URL rewriting involves adding data, a session ID, to the URL path that is interpreted by the container to associate the request with a session.

The session ID must be encoded as a path parameter in the URL string. The name of the parameter must be jsessionid. Here is an example of a URL containing encoded path information:

http://www.myserver.com/catalog/index.html;jsessionid=1234

URL rewriting exposes session identifiers in logs, bookmarks, referer headers, cached HTML, and the URL bar. URL rewriting should not be used as a session tracking mechanism where cookies or SSL sessions are supported and suitable.

Notice that it's a path parameter, not a query parameter. Your query params will follow that, like this:

http://www.myserver.com/catalog/index.html;jsessionid=1234?param1=value1&param2=value2&...

This mechanism is supported automatically by the server to track sessions, but it becomes pretty obvious that you need to give the server a helping hand. And you do that by making sure that all your links include the jsessionid otherwise your server won't identify your request with a session.

You can use [encodeURL](https://javaee.github.io/javaee-spec/javadocs/javax/servlet/http/HttpServletResponse.html" \l "encodeURL-java.lang.String-) in your Java code:

Encodes the specified URL by including the session ID, or, if encoding is not needed, returns the URL unchanged. The implementation of this method includes the logic to determine whether the session ID needs to be encoded in the URL. For example, if the browser supports cookies, or session tracking is turned off, URL encoding is unnecessary.

For robust session tracking, all URLs emitted by a servlet should be run through this method. Otherwise, URL rewriting cannot be used with browsers which do not support cookies.

You need to do the same inside your JSP files. That's usually done with something like [<c:url>](https://docs.oracle.com/javaee/5/tutorial/doc/bnakh.html#bnakn) instead of writing URLs directly into the file:

[...] You can use the url tag to rewrite URLs returned from a JSP page. The tag includes the session ID in the URL only if cookies are disabled; otherwise, it returns the URL unchanged. Note that this feature requires that the URL be relative. [...]

## Invoking a JSP Page from a Servlet

You can invoke a JSP page from a servlet through functionality of the standard javax.servlet.RequestDispatcher interface. Complete the following steps in your code to use this mechanism:

1. Get a servlet context instance from the servlet instance:
2. ServletContext sc = this.getServletContext();
4. Get a request dispatcher from the servlet context instance, specifying the page-relative or application-relative path of the target JSP page as input to the getRequestDispatcher() method:
5. RequestDispatcher rd = sc.getRequestDispatcher("/jsp/mypage.jsp");

Prior to or during this step, you can optionally make data available to the JSP page through attributes of the HTTP request object. See the next section, ["Passing Data Between a JSP Page and a Servlet"](https://docs.oracle.com/cd/A87860_01/doc/java.817/a83726/basics4.htm#1014110), for information.

Invoke the include() or forward() method of the request dispatcher, specifying the HTTP request and response objects as arguments. For example:

rd.include(request, response);

or:

rd.forward(request, response);

The functionality of these methods is similar to that of jsp:include and jsp:forward actions. The include() method only temporarily transfers control; execution returns to the invoking servlet afterward.

Note that the forward() method clears the output buffer.

# [RequestDispatcher.forward() vs HttpServletResponse.sendRedirect()](https://stackoverflow.com/questions/2047122/requestdispatcher-forward-vs-httpservletresponse-sendredirect)

## RequestDispatcher - forward() method

1. When we use the forward method, the request is transferred to another resource within the same server for further processing.
2. In the case of forward, the web container handles all processing internally and the client or browser is not involved.
3. When forward is called on the requestDispatcherobject, we pass the request and response objects, so our old request object is present on the new resource which is going to process our request.
4. Visually, we are not able to see the forwarded address, it is transparent.
5. Using the forward() method is faster than sendRedirect.
6. When we redirect using forward, and we want to use the same data in a new resource, we can use request.setAttribute() as we have a request object available.

## SendRedirect

1. In case of sendRedirect, the request is transferred to another resource, to a **different domain**, or to a different server for further processing.
2. When you use sendRedirect, the container transfers the request to the client or browser, so the URL given inside the sendRedirect method is visible as a new request to the client.
3. In case of sendRedirect call, **the old request and response objects** are lost because it’s treated as new request by the browser.
4. In the address bar, we are able to see the new redirected address. It’s not transparent.
5. sendRedirect is slower because one extra round trip is required, because a completely new request is created and the old request object is lost. Two browser request are required.
6. But in sendRedirect, if we want to use the same data for a new resource we have to store the data in session or pass along with the URL.

## Which one is good?

Its depends upon the scenario for which method is more useful.

If you want control is transfer to new server or context, and it is treated as completely new task, then we go for sendRedirect. Generally, a forward should be used if the operation can be safely repeated upon a browser reload of the web page and will not affect the result.

# [Sending session key in the header vs HTTP-Only cookie](https://stackoverflow.com/questions/54525490/sending-session-key-in-the-header-vs-http-only-cookie)

You are right, Cookies and Authorization headers are not compatible out of the box. As you pointed out, you are looking at two use-cases: one for browser usage and another for API (cli, desktop app).

If you want to support both via a single authentication scheme, you will need to work a bit more. As a good rule of thumb, browsers work well with cookies and its easy to set it up securely. You should opt for cookie-based session management with browsers.

Given the situation, I am thinking about extending the API server to use either one of Authorization header or cookie to authorize users. Is this a feasible option, and are there alternatives?

Yes, this is feasible, it will make your browser use-case more secure. As for alternatives, I put together a [Web Authentication Guide](https://www.securitydrops.com/the-web-api-authentication-guide/) that will greatly assist you in exploring your options.

https://www.securitydrops.com/the-web-api-authentication-guide/