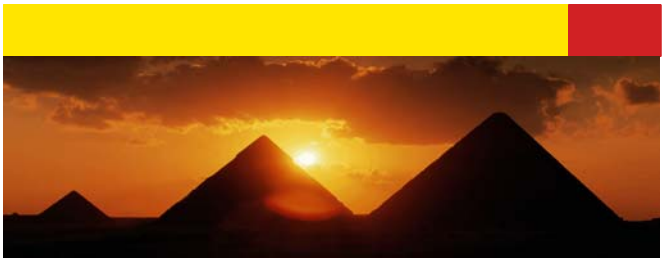


Session Tracking

1

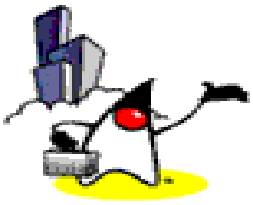
Now let's talk about session tracking, which is one of the most frequently used features of Servlet yet somehow people still have a fuzzy idea on how it works.

Topics

- Underlying mechanisms
- Why do we need Session Tracking feature of Servlet?
- Servlet Session Tracking Programming APIs
- Session timeout and invalidation

2

This is a list of sub-topics under Session tracking. We will first look into the underlying mechanisms of session management. Then we will talk a bit on why we need session tracking. We will spend some time talking about programming APIs. And we will talk about session timeout and invalidation.



Underlying Mechanisms of Session Tracking

3

Let's start our discussion on session tracking by talking about underlying mechanisms of passing session identification between browser and web container.

Why Session Tracking?

- Need a mechanism to maintain state across a series of requests from the same user (or originating from the same browser) over some period of time
  - Example: Online shopping cart
- Yet, HTTP is stateless protocol
  - Each time, a client talks to a web server, it opens a new connection
  - Server does not automatically maintain "conversational state" of a user

4

Many applications require a series of requests from a same client to be associated with one another. For example, the Duke's Bookstore application saves the state of a user's shopping cart across multiple requests.

Web-based applications are responsible for maintaining such state, called a session, because the HTTP protocol is stateless in that every time a client sends a request, it opens a new connection and the HTTP server does not automatically maintains this conversational state of a user.

To support applications that need to maintain state, Java Servlet technology provides an API for managing sessions and allows several mechanisms for implementing sessions.

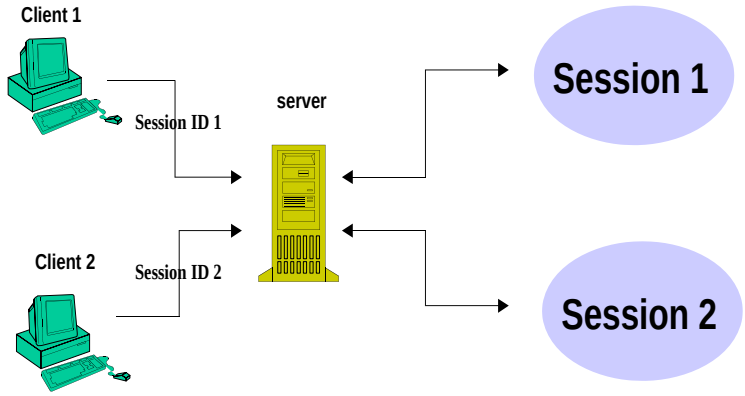
Session Tracking Use Cases

- When clients at an on-line store add an item to their shopping cart, how does the server know what's already in the cart?
- When clients decide to proceed to checkout, how can the server determine which previously created shopping cart is theirs?

5

There are many use cases in which client conversational state has to be maintained over a series of HTTP requests. One of the most frequently used use case scenario is online shopping cart. For example, when a user at an online store adds an item to their shopping cart, it basically sends a new HTTP request to the server. The server in this case has to add a new item to the existing list of items. Also when the client decides to proceed to checkout, the server should know which items are present in the shopping cart.

A Session Maintains Client Identity and State across multiple HTTP requests



6

This picture shows the relationship between a client and a session. As it shows, there is a one-to-one correspondence between a client and a session. In other words, a particular session maintains an identity and state of a particular client across the multiple HTTP requests. (By the way, "client" typically means an end user interacting with website via a browser.)

Now how does the client indicate that the set of HTTP requests it sends to the web server belong to a single session? This is accomplished by client sending the same "session id" to the web server. How does the "session id" get created in the first place? The web server when it receives the the first HTTP request from a particular client, it creates a new "session id" and sends it back to the client and then the client send the "session id" for all the subsequent HTTP requests it sends to the server.

Now what is the underlying mechanism that web server uses in order to create the "session id" and pass it to the client and how does client send the "session id" to the web server for subsequent HTTP requests? This mechanism is called "session tracking" and that is what we will talk about in the following slides.

Three "underlying" Session Tracking Mechanisms

- Cookies
- URL rewriting
- Hidden form fields
- Note that these are just underlying mechanisms of passing "session id"
  - do not provide high-level programming APIs
  - do not provide a framework for managing sessions
  - This is what Servlet Session Tracking feature provides

7

So there are three different session tracking mechanisms - (1) cookie based (2) URL rewriting (3) hidden form fields. The first and second are used mostly frequently while the third is used rarely.

Please note that these are just underlying mechanisms and do not provide any high-level programming APIs nor a framework for managing sessions. And as we will talk about again later on, this is where Servlet session tracking feature comes into the picture.

What is HTTP Cookie?

- Cookie is a small amount of information sent by a servlet to a Web browser
- Saved by the browser, and later sent back to the server in subsequent requests
  - A cookie has a name, a single value, and optional attributes
  - A cookie's value can uniquely identify a client
- Server uses cookie's value to extract information about the session from some location on the server

8

(read the slide)

Cookies as Session Tracking Mechanism

- Advantages:
  - Very easy to implement
  - Highly customizable
  - Persist across browser shut-downs
- Disadvantages:
  - Often: users turn off cookies for privacy or security reason
  - Not quite universal browser support

9

(read slide)

URL Rewriting

- URLs can be rewritten or encoded to include session information.
- URL rewriting usually includes a session id
- Session id can be sent as an added parameter:
  - http://.../servlet/Rewritten?sessionid=688

(read slide)

URL Rewriting as Session Tracking Mechanism

- Advantages:
  - Let user remain anonymous
  - They are universally supported(most styles)
- Disadvantages:
  - Tedious to rewrite all URLs
  - Only works for dynamically created documents

11

The advantages of using URL rewriting over cookie as session tracking mechanism are first it lets user to remain anonymous and second URL rewriting is universally supported. In fact, in Servlet Session scheme, you as a programmer can specify configuration option something like “if cookie is disabled in the browser, then automatically use URL Rewriting as session tracking mechanism”.

Hidden Form Fields

- Hidden form fields do not display in the browser, but can be sent back to the server by submit

<INPUT TYPE="HIDDEN" NAME="session" VALUE="...">

- Fields can have identification (session id) or just some thing to remember (occupation)
- Servlet reads the fields using req.getParameter()

12

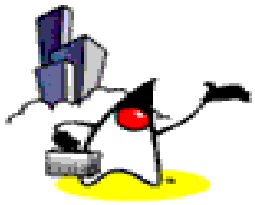
(read slide)

Hidden Form Fields as Session Tracking Mechanism

- Advantages:
  - Universally supported.
  - Allow anonymous users
- Disadvantages:
  - Only works for a sequence of dynamically generated forms.
  - Breaks down with static documents, emailed documents, bookmarked documents.
  - No browser shutdowns.

13

(read slide)



Why do we need Session Tracking feature of Servlet?

14

So far we talked about the underlying mechanisms of passing session id. This is certainly not enough for your servlet code to perform session tracking. So let's talk about why the session tracking feature of Servlet is needed.

Now Without "Session Tracking" Feature of Servlet

- Servlet programmers have to perform the following tasks themselves by using one of three session-tracking mechanisms
  - Generating and maintaining a session id for each session
  - Passing session id to client via either cookie or URL
  - Extracting session id information either from cookie or URL
  - Creating and maintaining a hashtable in which session id and session information are stored
  - Coming up with a scheme in which session information can be added or removed

15

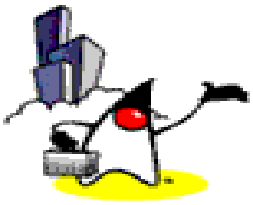
As was mentioned before, the 3 schemes we talked about provide underlying mechanism of passing session id information between the client and the server but it does not provide any framework nor programming APIs for managing these "session id's". That is, somebody has to perform the tasks mentioned above. And as we will see in the following slide, Servlet session tracking feature does all this for you so you as a programmer do not have to deal with this yourself. And this is one of the frequently mentioned advantages of using Servlet over CGI - session management.

"Session Tracking" Features of Servlet

- Provides higher-level API for session tracking
  - Built on top of Cookie or URL rewriting
- Servlet container maintains
  - internal hashtable of session id's
  - session information in the form of HttpSession
- Generates and maintains session id transparently
- Provides a simple API for adding and removing session information (attributes) to HttpSession
- Could automatically switch to URL rewriting if cookies are unsupported or explicitly disabled

16

So session tracking feature of Servlet supports the features mentioned in the slide. Please note the last bullet item mentioned above in which session tracking scheme of servlet can switch to URL rewriting automatically if cookies are unsupported or explicitly disabled by the user. And you as a programmer can specify this "switch automatically" feature in your code.



Servlet Session Tracking Programming API

Now let's talk about concrete session tracking programming APIs that are supported by Servlet.

HttpSession

- To get a user's existing or new session object:
  - HttpSession session = request.getSession(true);
    - "true" means the server should create a new session object if necessary
  - HttpSession is Java interface
    - Container creates a object of HttpSession type

Sessions are represented by an HttpSession object. You access a session by calling the getSession() method of a request object. This method returns the current session associated with this request, or, if the request does not have a session, it creates one. Since getSession() may modify the response header (if cookies are the session tracking mechanism), it needs to be called before you retrieve a PrintWriter or ServletOutputStream.

You can associate object-valued attributes with a session by name. Such attributes are accessible by any Web component that belongs to the same Web context and is handling a request that is part of the same session.

The Duke's Bookstore application stores a customer's shopping cart as a session attribute. This allows the shopping cart to be saved between requests and also allows cooperating servlets to access the cart. CatalogServlet adds items to the cart; ShowCartServlet displays, deletes items from, and clears the cart; and CashierServlet retrieves the total cost of the books in the cart.

Example: Getting HttpSession Object

```
public class CatalogServlet extends HttpServlet {
    public void doGet (HttpServletRequest request,
                       HttpServletResponse response)
      throws ServletException, IOException {

      // Get the user's session and shopping cart
      HttpSession session =request.getSession(true);
            ...
      out = response.getWriter();
            ...
        }
    }
    ...
```


This example code shows how to get a session object.

HttpSession Java Interface

- Contains Methods to
  - View and manipulate information about a session, such as the session identifier, creation time, and last accessed time
  - Bind objects to sessions, allowing user information to persist across multiple user connections

(read the slide)

Store and Retrieve of Attribute

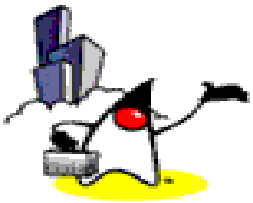
- To stores values:
  - session.setAttribute("cartItem", cart);
- To retrieves values:
  - session.getAttribute("cartItem");

(read the slide)

Setting and Getting Attribute

```
public class CatalogServlet extends HttpServlet {
   public void doGet (HttpServletRequest request,
                                HttpServletResponse response)
                      throws ServletException, IOException {
     // Get the user's session and shopping cart
     HttpSession session = request.getSession(true);
     ShoppingCart cart = (ShoppingCart)session.getAttribute(
                          "examples.bookstore.cart");
     // If the user has no cart, create a new one
     if (cart == null) {
         cart = new ShoppingCart();
         session.setAttribute("examples.bookstore.cart", cart);
     }
     ...
     //see next slide.
  }
}
```


This example code shows how to get and set an attribute for a session object.



How Servlet supports both Cookie-enabled and Cookie-disable browsers?

23

Now let's talk about how your servlet can supports a browser that disabled Cookie. That is to say, how you program your servlet or JSP pages so that you can handle both cookie-enabled and cookie-disabled browser.

If Cookie is turned off..

- If your application makes use of session objects
  - you must ensure that session tracking is enabled by having the application rewrite URLs whenever the client turns off cookies
  - by calling the response's encodeURL(URL) method on all URLs returned by a servlet
  - This method includes the session ID in the URL only if cookies are disabled; otherwise, it returns the URL unchanged

24

A Web container can use several methods to associate a session with a user, all of which involve passing an identifier between the client and server. The identifier can be maintained on the client as a cookie or the Web component can include the identifier in every URL that is returned to the client.

If your application makes use of session objects, you must ensure that session tracking is enabled by having the application rewrite URLs whenever the client turns off cookies. You do this by calling the response's encodeURL(URL) method on all URLs returned by a servlet. This method includes the session ID in the URL only if cookies are disabled; otherwise, it returns the URL unchanged.

String response.encodeURL(URL)

- Encodes the specified URL by including the session ID in it, or, if encoding is not needed, returns the URL unchanged
  - 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

25

Since people get confused about the usage of encodeURL(URL) method of HttpServletResponse object, let me clarify the usage of this method one more time in this slide.

Again, this method contains the logic to determine whether the session id needs to be encoded in the URL or not and then returns the URL with session ID if it is needed or URL without session ID otherwise. How does it decide? If browser supports cookies or session tracking is turned off, URL encoding is not necessary.

Now whenever your servlet has to return a response that contains URL (which is to be invoked by the client sometime), you want to run through this method for all those URLs. Otherwise, URL rewriting based session tracking cannot be used when the browsers do not support cookies.

Example: response.encodeURL()

```
out.println("<p>   <p><strong><a href=\"" +
  response.encodeURL(request.getContextPath() + "/catalog") +
    "\">" + messages.getString("ContinueShopping") +
    "</a>      " +
    "<a href=\"" +
  response.encodeURL(request.getContextPath() + "/cashier") +
    "\">" + messages.getString("Checkout") +
    "</a>      " +
    "<a href=\"" +
  response.encodeURL(request.getContextPath() +
    "/showcart?Clear=clear") +
    "\">" + messages.getString("ClearCart") +
    "</a></strong>");
```

26

The doGet() method of ShowCartServlet of bookstore example in Java WSDP encodes the three URLs at the bottom of the shopping cart display page as above.

Again, please not that the URL's you are returning as part of response message will be invoked by the client sometime. In that case, the browser will be able to send URL with a session if cookie scheme is not supported in that browser.

Now let's take a look at an example URL that is to be transmitted from the browser when a user clicks on the web page item whose link is set to “cachier” servlet.

Example: URL

- If cookies are turned off
  - http://localhost:8080/bookstore1/cashier;jsessionid=c0o7fszeb1
- If cookies are turned on
  - http://localhost:8080/bookstore1/cashier

27

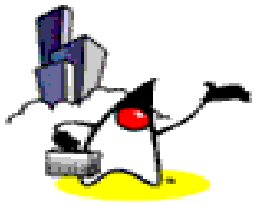
If cookies are turned off, the session is encoded in the Check Out URL as follows:

http://localhost:8080/bookstore1/cashier; jsessionid=c0o7fszeb1

If cookies are turned on, the URL is simply

http://localhost:8080/bookstore1/cashier

Again this is the value that will be sent from the browser. Please note that these URL values are the ones you set in the response message in the doGet() method of ShowCartServlet of bookstore example we have seen in previous slides.



Session Timeout & Invalidation

28

Now let's talk about how a session can be closed (or more technically how session object can be removed - garbage collected- by the container).

There are two different cases a session can be closed: One case is when an enduser just leaves the browser without actively closing a session. The other case is when a user in fact indicates that s/he is done with the session.

Session Timeout

- Used when an end-user can leave the browser without actively closing a session
- Sessions usually timeout after 30 minutes of inactivity
  - Product specific
  - A different timeout may be set by server admin
- getMaxInactiveInterval(), setMaxInactiveInterval() methods of HttpSession interface
  - Gets or sets the amount of time, session should go without access before being invalidated

29

Since there is no easy way for an HTTP client to signal the servlet that it no longer needs a session, each session has an associated timeout so that its resources can be reclaimed by the container. The timeout period can be accessed with a session's [get|set]MaxInactiveInterval methods. You can also set the time-out period through vendor management tool.

To ensure that an active session does not get timed out, you should periodically access the session via service methods because this resets the session's time-to-live counter.

Issues with “Stale” Session Objects

- The number of “stale” session objects that are in “to be timed out” could be rather large
- Example
  - 1000 users with average 2 minutes session time, thus 15000 users during the 30 minutes period
  - 4K bytes of data per session
  - 15000 sessions * 4K -= 60M bytes of session data
  - This is just for one Web application
- Could have an performance impact
  - Use the data space in Session object with care

30

Since the number of “stale” session objects that are waiting to be timed out could be rather large, you are recommended to use the space in the Session object with care.

For example, if your web application handles 1000 users and if a user stays in average 2 minutes before leaving your application, and if your application saves 4K bytes of session data, the system would need 60Mbytes of memory just to maintain the session data and this is just for a single web application.

Session Invalidation

- Can be used by servlet programmer to end a session proactively
  - when a user at the browser clicks on “log out” button
  - when a business logic ends a session (“checkout” page in the example code in the following slide)
- public void invalidate()
  - Expire the session and unbinds all objects with it
- Caution
  - Remember that a session object is shared by multiple servlets/JSP-pages and invalidating it could destroy data that other servlet/JSP-pages are using

31

When an end-user at the browser indicates his intention of ending a session by clicking on a “log out” button on a page, for example, servlet programmers can use the session's invalidate() method to invalidate a session and remove any session data.

Please remember that a session is associated with a user (i.e client) not with a particular servlet. And invalidating a session could result in destroying the data that is being used by other servlets and JSP pages.

Example: Invalidate a Session

```
public class ReceiptServlet extends HttpServlet {
    public void doPost(HttpServletRequest request,
                       HttpServletResponse response)
       throws ServletException, IOException {
                ...
       scart = (ShoppingCart)
          session.getAttribute("examples.bookstore.cart");
                ...
       // Clear out shopping cart by invalidating the session
       session.invalidate();

       // set content type header before accessing the Writer
       response.setContentType("text/html");
       out = response.getWriter();
       ...
    }
}
```

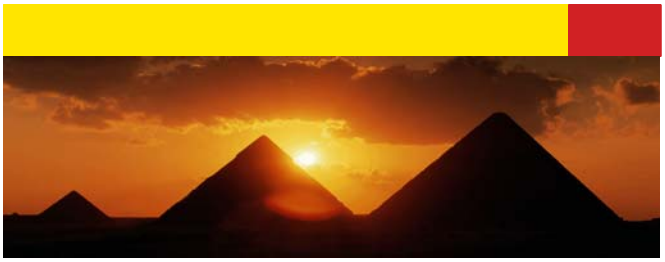
This is an example code in the bookstore application that comes with Java WSDP that shows how to invoke invalidate() method. In this example, the ReceiptServlet is the last servlet to access a client's session, so it has responsibility for invalidating the session:

“Session Object Unbound” Event Notification

- Any attribute object that implements HttpSessionBindingListener interface gets an notification
  - valueUnbound(HttpSessionBindingEvent event)
- Note: when your servlet adds an attribute to HttpSession via setAttribute() method, the container calls
  - valueBound(HttpSessionBindingEvent event)

33

When a session becomes inactive either via timeout or invalidated by the servlet, any objects bound to the HttpSession object automatically get unbound. Then any attached objects are automatically notified if they implement the HttpSessionBindingListener interface.



Passion!

34