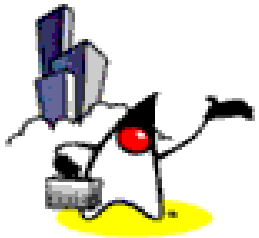
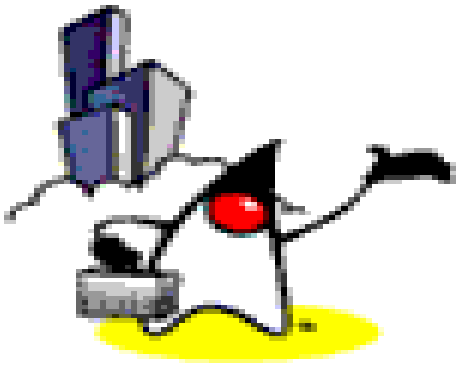
Session Tracking



Topics

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



Underlying Mechanisms of Session Tracking

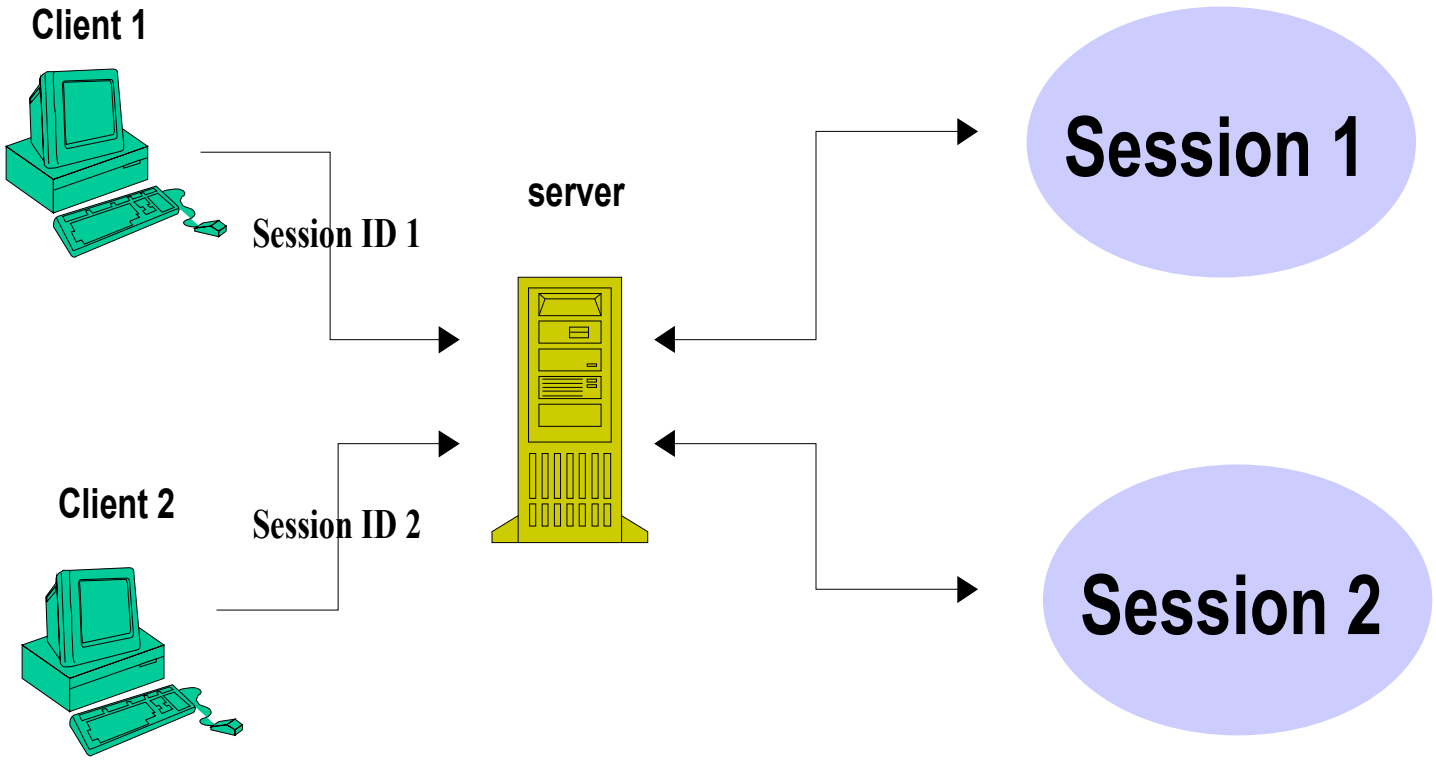
Why Session Tracking?

- 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 maintains "conversational state" of a user

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?

A Session Maintains Client Identity and State across multiple HTTP requests



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

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

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

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

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

Hidden Form Fields

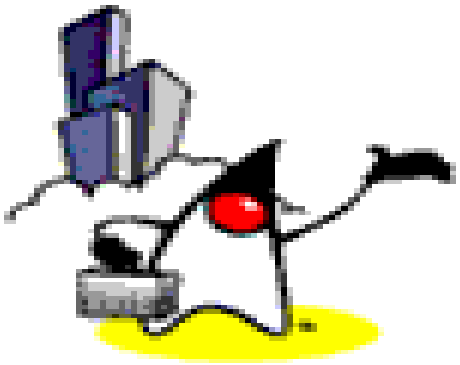
- Hidden form fields do not display in the browser, but can be sent back to the server by submit

```
<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()

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.



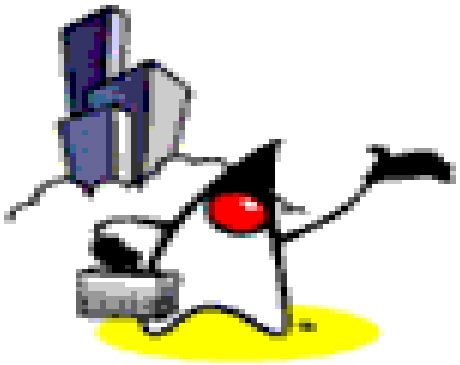
Why do we need Session Tracking feature of Servlet?

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

“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



Servlet Session Tracking Programming API

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

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();
            ...
        }
    }
    ...
```

HttpSession Java Interface

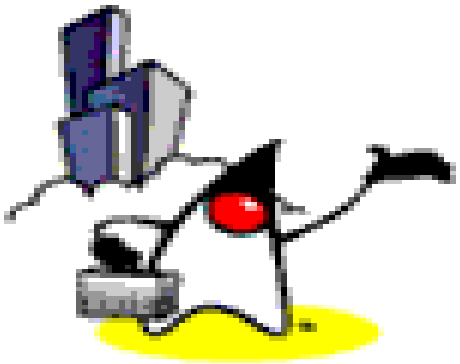
- Contains Methods to
  - View and manipulate information about a session, such as the session identifier, creation time, and last accessed time
  - Bind objects to sessions, allowing user information to persist across multiple user connections

Store and Retrieve of Attribute

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

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.
  }
}
```



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

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

String response.encodeURL(URL)

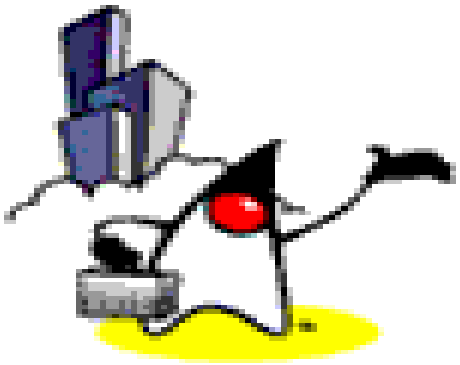
- Encodes the specified URL by including the session ID in it, or, if encoding is not needed, returns the URL unchanged
  - Implementation of this method includes the logic to determine whether the session ID needs to be encoded in the URL
  - For example, if the browser supports cookies, or session 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

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>");
```

Example: URL

- If cookies are turned off
  - http://localhost:8080/bookstore1/cashier;jsessionid=c007fszeb1
- If cookies are turned on
  - http://localhost:8080/bookstore1/cashier



Session Timeout & Invalidation

Session Timeout

- Used whn 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

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

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

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();
       ...
    }
}
```

“Session Object Unboud” 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)

Passion!