Session Tracking using HttpSession

HyperText Transfer Protocol (HTTP) was designed so that users could travel from one web site to another quickly. Hypertext means that a document has links to other web pages and sites. It was not designed for web sites to track users' activities.

When a user makes a request for a browser page, the server responds to the request and then disconnects. The server can store the IP (Internet Protocol) address, but it may not be unique. Often several computers share the same IP address. Web sites need some other way to keep track of their visitors. This is especially true of on-line stores. Users typically put something in their shopping cart and then continue shopping on the site.

There are two ways for web sites to track user activity. One is by depositing a *cookie* on the user's hard drive, and the other is URL rewriting. There are several kinds of cookies, but the one that we will look at just puts data in temporary storage and deletes it when finished. URL rewriting involves adding an identification number to the URL string. This is actually less safe than storing a cookie, since the string is sent unencrypted, and use of the back button on the browser can destroy it. Some web stores have decided not to deal with users that have set their browsers to refuse cookies.

HttpSession

Java supplies a *session* object that implements javax.servlet.http.HttpSession. It is associated with HttpServletRequest and can be accessed by a servlet using

```
    HttpSession session = request.getSession (true);
```

The boolean parameter, true, is used to tell the server to use the current session if there is one, or to create a new session if no current session exists. If the parameter is omitted, the default is true.

When a session is created, a cookie containing a session ID is stored on the user's hard drive. The name of the ID is JSESSIONID. It is a long string made up of a random sequence of letters and digits. It is probably not sufficiently random for very large web stores, but for smaller ones it is unlikely that two sessions would receive the same ID. If the user's browser does not accept cookies, the server can use

```
    String url = request.getRequestURI ();
    String codedUrl = reponse.encodeURL (url);
```

Sessions have a life-time. They begin when the user first contacts the web-site and end when the user closes the browser. The server can terminate sessions after a given number of minutes. This information is contained in web.xml with the lines

```
    <session-config>
        <session-timeout>30</session-timeout>
    </session-config>
```

These lines go right after the ones on <servlet-mapping>. If the time given is negative, the session will not timeout.

Cookies

When the server gets a session object, a cookie is created and stored on the user's computer. The server can also create cookies and deposit them on the user's computer. A cookie is created by

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

where name and value are both Strings made up of ascii alphanumeric values. The following code will add a cookie to the user's computer:

```
    Cookie cookie = new Cookie ("Your name", "Some value such as an ID");
    response.addCookie (cookie);
```

Unless the server specifies otherwise, the cookie will be deleted when the browser is closed. That can be changed by setting the maximum age for the cookie. The code for this is

```
	cookie.setMaxAge (time_in_seconds);
```

If you wish the cookie to be available for an hour, use

```
	cookie.setMaxAge (3600);
```

You can also set a comment with cookie.setComment ("This is an example of a cookie."); However, comments are not returned to the browser. The following servlet illustrates this. It can be accessed in a browser by typing http://localhost/servlet/http_session.MakeCookie.

```
package http_session;

import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

/*	MakeCookie creates a cookie, stores it, and then checks for cookies. */
public class MakeCookie extends HttpServlet
{
	public void doGet (HttpServletRequest request, HttpServletResponse response)
	{
		try
		{
			response.setContentType ("text/html");
			PrintWriter out = response.getWriter ();
			Cookie cookie = new Cookie ("Pace", "CS 396S");
			cookie.setComment ("This is an example of a cookie.");
			cookie.setMaxAge (3600); // Set the maximum age to be an hour.
			response.addCookie (cookie);
			Cookie cookies [] = request.getCookies ();

			Page.createHeader (out, "Cookies");
			if ((cookies == null) || (cookies.length == 0))
				out.println ("<h3>No Cookies Found</h3>");
			else
			{
				out.println ("<h3>Cookies Found</h3>");
				for (int count = 0; count < cookies.length; count ++)
				{
					out.println ("<br>Name: " + cookies [count].getName ());
					out.println ("<br>Value: " + cookies [count].getValue ());
					out.println ("<br>Comment: " + cookies [count].getComment ());
					out.println ("<br>MaxAge: " + cookies [count].getMaxAge ());
				}
			}
			Page.createFooter (out);
		} catch (IOException ex) {System.out.println ("<h3>IO Exception.</h3>");}
	} // doGet
} // MakeCookie
```

Session Attributes

The session object is available to all servlets in the application. So session data can be passed from one servlet to another while the session is active. This is done by storing data as a session attribute. Attributes are maintained in a hash table, so you need a key (String) for each attribute. These strings can be constants in your servlets.

Once the servlet has gotten a session, it can set an attribute, such as a customer's ID.

```
    HttpSession session = request.getSession (true);
    session.setAttribute (CustomerKey, customerId);
```

where CustomerKey is a constant String used throughout the application to locate the customer's data.

Attribute data is retrieved using getAttribute (key), as follows:

```
    String customerId = (String) session.getAttribute (CustomerKey);
```

Note that this hash table stores objects, so when the ID is retrieved, it must be cast to a String.

A very simple servlet illustrates this.

```
package http_session;

import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

/*  SessionAttribute stores a customer's ID in a session attribute.*/
public class SessionAttribute extends HttpServlet
{
    static final String CustomerKey = "SessionKey";

    public void doGet (HttpServletRequest request, HttpServletResponse response)
    {
        try
        {
            HttpSession session = request.getSession (true);
            String sessionId = session.getId ();
            String customerId = sessionId.substring (0, 6);
            session.setAttribute (CustomerKey, customerId);

            response.setContentType ("text/html");
            PrintWriter out = response.getWriter ();
            Page.createHeader (out, "Session Attributes");
            out.println ("<h3>Customer ID: " + (String) session.getAttribute (CustomerKey) + "<h3>");
            Page.createFooter (out);
        } catch (IOException ex) {System.out.println ("<h3>IO Exception.</h3>");}
    }
} // SessionAttribute
```

Session attributes can be used to store any object including IDs, shopping carts, customer orders, etc. There is a limit on the number, but you are unlikely to exceed it.

Shopping Carts

On-line stores use shopping carts to store customer purchases before they decide to check out. There are many ways to implement these, but probably the simplest is as a vector of items. The Item object can store information about the item ordered, such as the product's ID, name of table it is in, quantity ordered, etc.

```
package orders;

// Item stores data for a single ordered item.
public class Item
{
	private String tablename, productId;
	private int quantityOrdered;
	private double subtotal;

	public Item (String t, String p, int q, double s)
	{
		tablename = t; productId = p;
		quantityOrdered = q;
		subtotal = s;
	} // constructor

	public String getTablename () {return tablename;}
	public String getProductId () {return productId;}
	public int getQuantityOrdered () {return quantityOrdered;}
	public double getSubtotal () {return subtotal;}
} // class Item
```

The shopping cart then maintains a vector of items. It also keeps track of the ID for the order, the customer's ID, and the running total cost of the order.

```
package orders;

import java.util.*;
// The Shopping cart is used to store the items a customer wishes to order.
public class ShoppingCart
{
	private String orderId, customerId;
	private Vector items;
	private double total = 0;

	public ShoppingCart (String ordId, String custId)
	{
		items = new Vector ();
		orderId = ordId;
		customerId = custId;
	} // constructor

	public int getSize () {return items.size ();}
	public String getOrderId () {return orderId;}
	public String getCustomerId () {return customerId;}
	public double getTotal () {return total;}
```

```
        public void addItem (Item item)
        {
                items.add (item);
                total += item.getSubtotal ();
        } // addItem

        public Item getItem (int index)
        {
                return (Item) items.elementAt (index);
        }
} // class ShoppingCart
```

The cart can be created and saved as a session attribute either when the customer first visits the web site or after some login procedure. Web sites use both methods. Then when the customer decides to add something to the cart, it can be retrieved from the session. Since the cart is an object all that is actually stored is a reference (pointer) to the cart, so adding an item changes the contents of the cart.

```
        ShoppingCart cart = (ShoppingCart) session.getAttribute (CartId);
        Item item = new Item (table, productId, quantityOrdered, subtotal);
        cart.addItem (item);
```

If the customer then decides to buy the items in the cart and check out, the cart can again be retrieved from the session and the order processed.