Manejo de Cookies y Sesiones

Índice

Veremos en este tema aspectos sobre el seguimiento de las acciones de los usuarios sobre un sitio web. Para ello veremos cómo trabajar con cookies en los servlets, y cómo manejar información sobre las sesiones de los usuarios.

1. Cookies

1.1. Cookies en HTTP

Las **cookies** son un mecanismo general mediante el que los programas de un servidor web pueden almacenar información en la parte del cliente de la conexión. Es una forma de añadir estado a las conexiones HTTP, aunque el manejo de cookies no es parte del protocolo HTTP, pero es soportado por la mayoría de los clientes.

Las cookies son objetos de tipo: *nombre = valor*, donde se asigna un *valor* determinado (una cadena de texto) a una variable del *nombre* indicado. Dicho objeto es almacenado y recordado por el servidor web y el navegador durante un período de tiempo (indicado como un parámetro interno de la propia *cookie*). Así, se puede tener una lista de *cookies* con distintas variables y distintos valores, para almacenar información relevante para cada usuario (se tienen listas de cookies independientes para cada usuario).

El funcionamiento es: el servidor, con la cabecera `Set-Cookie`, envía al cliente información de estado que éste almacenará. Entre la información se encuentra la descripción de los rangos de URLs para los que este estado es válido, de forma que para cualquier petición HTTP a alguna de esas URLs el cliente incluirá esa información de estado, utilizando la cabecera `Cookie`.

La sintaxis de la cabecera `Set-Cookie` es:

```
Set-Cookie: CLAVE1=VALOR1;...;CLAVEN=VALORN [OPCIONES]
```

donde OPCIONES es una lista opcional con cualquiera de estos atributos:

```
expires=FECHA;path=PATH;domain=DOMINIO;secure
```

- Las parejas de *CLAVE* y *VALOR* representan la información almacenada en la cookie
- Los atributos `domain` y `path` definen las URL en las que el navegador mostrará la cookie. `domain` es por defecto el *hostname* del servidor. El navegador mostrará la cookie cuando acceda a una URL que se empareje correctamente con ambos atributos. Por ejemplo, un atributo `domain="eps.ua.es"` hará que el navegador muestra la cookie cuando acceda a cualquier URL terminada en `"eps.ua.es"`. `path` funciona de forma similar, pero con la parte del path de la URL. Por ejemplo, el path `"/foo"` hará que el navegador muestre la cookie en todas las URLs que comiencen por `"/foo"`.
- `expires` define la fecha a partir de la cual la cookie caduca. La fecha se indica en formato GMT, separando los elementos de la fecha por guiones. Por ejemplo:

```
expires=Wed, 09-Nov-1999 23:12:40 GMT
```

- `secure` hará que la cookie sólo se transmita si el canal de comunicación es seguro (tipo de conexión HTTPS).

Por otra parte, cuando el cliente solicita una URL que empareja con el dominio y path de alguna cookie, envía la cabecera:

```
Cookie: CLAVE1=VALOR1;CLAVE2=VALOR2;...;CLAVEN=VALORN
```

El número máximo de cookies que está garantizado que acepte cualquier navegador es de 300, con un máximo de 20 por cada servidor o dominio (los servlets que se ejecutan en un mismo servidor comparten las cookies). El tamaño máximo de una cookie es de 4096 bytes.

En Javascript, por ejemplo, el objeto `document.cookie` contiene como valor una lista de la forma:

```
nombre1=valor1;nombre2=valor2;...;nombreN=valorN
```

donde se almacenan así los valores de las cookies que se tengan definidas.

Se pueden emplear `cookies`, entre otras cosas, para:

- **Identificar a un usuario durante una o varias sesiones**. Por ejemplo, a la hora de realizar compras a través de una tienda web, se almacena su identidad (login y password) como una cookie y se recuerda a lo largo de diferentes visitas qué es lo que lleva almacenado en su cesta de la compra cada usuario.
- **Personalizar un sitio web de acuerdo a las preferencias de cada usuario**: definir el contenido, apariencia, etc, que queremos que tenga una determinada página en función de las preferencias del usuario que la esté visitando.

A la hora de trabajar con cookies, debemos tener en cuenta que nuestro sitio web no debe depender de ellas, puesto que muchos navegadores y usuarios las deshabilitan para evitar problemas de privacidad y otras cuestiones.

Veremos ahora cómo trabajar con cookies desde servlets.

1.2. Enviar una cookie

Para crear una nueva cookie y enviarla, se siguen los pasos:

1. Crear la cookie

Las cookies se manejan con la clase **`Cookie`**. Se tiene el constructor:

```
public Cookie (String nombre, String valor)
```

que crea una cookie de nombre `nombre`, dándole el valor `valor`.

2. Establecer los atributos de la cookie

Una vez creada la cookie, podemos establecer los atributos que queramos, con los métodos de la clase `Cookie`. Por ejemplo, se tienen:

```
public void setComment(String comentario)
public void setMaxAge(int edad)
...
```

El primero asigna una cadena descriptiva sobre la cookie. El segundo indica cuántos segundos de vida tiene. Si es un valor negativo, se borrará la cookie cuando se cierre el navegador. Si el valor es 0, se borra la cookie instantáneamente, y si es positivo, se borrará la cookie cuando pasen los segundos indicados (si cerramos y volvemos a abrir el navegador dentro de ese tiempo, la cookie todavía persistirá). Se tienen otros métodos para establecer atributos de la cookie.

3. Enviar la cookie

Las cookies se añaden a la cabecera de la respuesta, y se envían así al cliente, mediante el método de `HttpServletResponse`:

```
public void addCookie (Cookie cookie)
```

Ejemplo

Vemos un ejemplo completo de envío de cookie:

```
public class MiServlet extends HttpServlet
{
        public void doGet (HttpServletRequest request,
                           HttpServletResponse response)
                        throws ServletException, IOException {
                Cookie miCookie = new Cookie ("nombre", "Pepe");
                miCookie.setMaxAge(120);
                response.addCookie(miCookie);
                PrintWriter out = response.getWriter();
                ...
        }
}
```

Hay que tener en cuenta que las cookies son parte de la cabecera HTTP, con lo cual hay que enviarlas ANTES de escribir la respuesta (o antes de obtener el objeto `Writer` si lo queremos utilizar).

1.3. Obtener una cookie

Para obtener una cookie que envía el cliente se trabaja sobre la petición del cliente (`HttpServletRequest`), siguiendo los pasos:

1. Obtener todas las cookies

Obtenemos todas las cookies con el método **getCookies()** de la clase `HttpServletRequest`:

```
public Cookie[] getCookies()
```

Con esto se tiene un array con todas las cookies actuales para el usuario. Si no hay cookies el método devuelve `null`.

2. Obtener el valor de una cookie

Con lo anterior, para obtener el valor de una cookie simplemente recorremos el array de cookies buscando la que concuerde con el nombre que queramos. Pueden ser útiles los métodos de `Cookie`:

```
public String getName()
public String getValue()
```

El primero obtiene el nombre de la cookie, y el segundo el valor.

Ejemplo

Un ejemplo de uso, para obtener el nombre del usuario, guardado en la cookie "nombre":

```
public void doGet (HttpServletRequest request,
                   HttpServletResponse response)
                throws ServletException, IOException {
        Cookie[] cookies = request.getCookies();
        String nombre;
        for (int i = 0; i < cookies.length; i++)
                if (cookies[i].getName().equals("nombre"))
                        nombre = cookies[i].getValue();
}
```

1.4. Ejemplo

Aquí tenéis un ejemplo de uso de cookies. El servlet `ServletCookies` cuenta el número de visitas a una página con una cookie que dura 3 minutos.

```
package ejemplos;

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

public class ServletCookies extends HttpServlet
{
   // Metodo para GET

   public void doGet(HttpServletRequest request,
                     HttpServletResponse response)
                  throws ServletException, IOException {
        response.setContentType("text/html");
        response.setHeader("Cache-Control", "no-cache");

        Cookie[] cookies = request.getCookies();
        Cookie contador = buscaCookie("contador", cookies);

        if (contador == null)
        {
           // Creamos la cookie con el contador

           Cookie cookie = new Cookie ("contador", "1");
           cookie.setMaxAge(180);
```

```
            response.addCookie(cookie);

            // Mostramos el mensaje de primera visita

            PrintWriter out = response.getWriter();
            out.println ("<HTML>");
            out.println ("<BODY>");
            out.println ("Primera visita");
            out.println ("<BR>");
            out.println ("</BODY>");
            out.println ("</HTML>");

         } else {

            // Obtenemos el valor actual del contador

            int cont = Integer.parseInt(contador.getValue());
            cont++;

            // Modificamos el valor de la cookie
            // incrementando el contador

            Cookie cookie = new Cookie ("contador", "" + cont);
            cookie.setMaxAge(180);
            response.addCookie(cookie);

            // Mostramos el numero de visitas

            PrintWriter out = response.getWriter();
            out.println ("<HTML>");
            out.println ("<BODY>");
            out.println ("Visita numero " + cont);
            out.println ("<BR>");
            out.println ("</BODY>");
            out.println ("</HTML>");
         }
   }

   // Busca la cookie 'nombre'
   // en el array de cookies indicado.
   // Devuelve null si no esta

   private Cookie buscaCookie(String nombre,
                              Cookie[] cookies)
   {
        if (cookies == null)
           return null;

        for (int i = 0; i < cookies.length; i++)
           if (cookies[i].getName().equals(nombre))
                return cookies[i];

        return null;
   }
}
```

Podéis probar el ejemplo con:

```
http://localhost:8080/appcs/servlet/ejemplos.ServletCookies
```

2. Seguimiento de sesiones

El seguimiento de sesiones es un mecanismo empleado por los servlets para gestionar un estado sobre las peticiones realizadas desde un mismo cliente (un mismo navegador) a lo largo de un período de tiempo determinado. Las sesiones se comparten por los servlets a los que accede un cliente (algo útil si queremos construir una aplicación basada en múltiples servlets).

Para utilizar el seguimiento de sesiones se tienen los pasos:

- Utilizar una sesión (objeto **HttpSession**) para un usuario
- Almacenar u obtener datos del objeto `HttpSession`
- Opcionalmente, invalidar la sesión

2.1. Obtener una sesión

El método **`getSession()`** del objeto **`HttpServletRequest`** obtiene una sesión de usuario.

```
public HttpSession getSession()
public HttpSession getSession(boolean crear)
```

El primer método obtiene la sesión actual, o crea una si no existe. Con el segundo método podemos establecer, mediante el flag booleano, si queremos crear una nueva si no existe (`true`) o no (`false`). Si la sesión es nueva, el método **isNew()** del `HttpSession` devuelve `true`, y la sesión no tiene ningún dato asociado. Deberemos ir añadiéndole datos tras crearla.

Para mantener la sesión de forma adecuada, debemos llamar a `getSession()` antes de que se escriba nada en la respuesta `HttpServletResponse` (y si utilizamos un `Writer`, debemos obtenerla antes de obtener el `Writer`, no antes de escribir datos).

Por ejemplo:

```
public class MiServlet extends HttpServlet
{
        public void doGet (HttpServletRequest request,
                           HttpServletResponse response)
                        throws ServletException, IOException {
                HttpSession sesion = request.getSession(true);
                ...
                PrintWriter out = response.getWriter();
                ...
        }
}
```

2.2. Guardar y obtener datos de la sesión

La interfaz **`HttpSession`** proporciona métodos que permiten almacenar y obtener:

- Propiedades de la sesión, como por ejemplo su identificador:

```
public String getId()
```

- Datos de la aplicación, que se almacenan y obtienen como pares nombre-valor, donde el nombre es un `String` que identifica al dato, y el valor es un `Object` con el valor asociado. Tendremos que tener cuidado de no sobreescribir datos de un servlet desde otro accidentalmente. Se tienen los métodos:

```
public Object getAttribute(String nombre)
public void   setAttribute(String nombre, Object valor)
public void   removeAttribute(String nombre)
```

que obtienen / establecen / eliminan, respectivamente, valores de atributos. Estos`métodos eran `getValue()`, `putValue()` y `removeValue()` hasta la versión 2.2 de servlets. Se tienen además métodos como `getAttributeNames()` para obtener los nombres de los atributos que se tienen almacenados para la sesión, y otros métodos de utilidad en la clase `HttpSession`.

Por ejemplo:

```
public class MiServlet extends HttpServlet
{
        public void doGet (HttpServletRequest request,
                           HttpServletResponse response)
                         throws ServletException, IOException {
                HttpSession sesion = request.getSession(true);
                String nombreUsuario =
                   (String)(sesion.getAttribute("nombre"));
                sesion.setAttribute("password", "secreto");
        }
}
```

El ejemplo lee el atributo "nombre" de la sesión, y establece el atributo "password" al valor "secreto".

2.3. Invalidar la sesión

Una sesión de usuario puede invalidarse manualmente, o automáticamente (dependiendo de dónde esté ejecutando el servlet). Esto implica eliminar el objeto `HttpSession` y sus valores de la memoria. Se tienen los métodos de `HttpSession`:

```
public int getMaxInactiveInterval()
public void setMaxInactiveInterval(int intervalo)
public void invalidate()
```

Para invalidarla automáticamente, la sesión expira cuando transcurre el tiempo indicado por el método **`getMaxInactiveInterval()`** entre dos accesos del cliente (en segundos). Se puede establecer dicho valor con **`setMaxInactiveInterval(...)`**.

Para invalidar manualmente una sesión, se emplea el método **`invalidate()`** de la misma. Esto puede ser interesante por ejemplo en comercio electrónico: podemos mantener una sesión donde se vayan acumulando los productos que un usuario quiera comprar, e invalidar la sesión (borrarla) cuando el usuario compre los productos.

Por ejemplo:

```
public class MiServlet extends HttpServlet
{
        public void doGet (HttpServletRequest request,
                           HttpServletResponse response)
                        throws ServletException, IOException {
                HttpSession sesion = request.getSession(true);
                ...
                sesion.invalidate();
                ...
        }
}
```

2.4. Compatibilidad con los navegadores

Una alternativa para el seguimiento de sesiones consiste en la **`reescritura de URLs`**. Con esta técnica, se añaden como parámetros de la URL los datos relativos a la sesión actual, de forma que se van conservando entre las páginas.

El seguimiento de sesiones por defecto emplea cookies para almacenar el identificador de una sesión. Para poder utilizar seguimiento de sesiones con usuarios que accedan desde navegadores que no utilicen cookies, los servlets aplican automáticamente la reescritura de URLs cuando no se utilizan cookies.

Para poder utilizar esto, debemos codificar todas las URLs que referenciemos. Para esto se emplean los métodos:

```
public String encodeURL(String url)
public String encodeRedirectURL(String url)
```

El primero se emplea para asociar identificadores con URLs, se utilizará cuando pongamos urls en el contenido de la página que generamos. El segundo se utiliza para asociar identificadores con redirecciones. Lo emplearemos cuando utilicemos métodos `sendRedirect()`, para codificar la URL que se le pasa. Ambos devuelven la URL sobreescrita si la sobreescritura ha sido necesaria, o la misma URL si no ha sido necesario sobreescribir.

Por ejemplo:

```
public class MiServlet extends HttpServlet
{
        public void doGet (HttpServletRequest request,
                           HttpServletResponse response)
                        throws ServletException, IOException {
                ...
                String url = response.encodeURL(
                   "http://localhost:8080/mipagina.html");
                out.println ("<a href=\"" + url + "\">...</a>");
                ...
                String url2 = response.encodeRedirectURL(
                   "http://otrapagina.com");
                response.sendRedirect(url2);
        }
}
```

2.5. Oyentes

Existen tres tipos de oyentes (`listeners`) que podemos utilizar en sesiones, para dar respuesta a eventos que produzcan las propias sesiones, o que desde fuera se provoquen sobre las mismas:

- **`HttpSessionListener`** se emplea para eventos de crear la sesión y terminarla. Tiene los métodos:

```
public void sessionCreated(HttpSessionEvent e)
public void sessionDestroyed(HttpSessionEvent e)
```

  El objeto que implemente esta interfaz ejecutará el código de `sessionCreated()` cuando se inicie la sesión, y el de `sessionDestroyed()` cuando se termine. Las clases que implementen este oyente deben llevar la anotación `@WebListener` en caso de estar utilizando la API de Servlet 3.0, o bien configurarse en el fichero descriptor de la aplicación, mediante etiquetas **<listener>**:

```
<listener>
        <listener-class>clase</listener-class>
</listener>
```

  donde en **<listener-class>** se pone el nombre (completo) de la clase que implementa el listener. Así, el listener se registra, y el servidor ya sabe que tiene que notificarle en los momentos oportunos.

- **HttpSessionBindingListener**: si un objeto asociado a una sesión implementa esta interfaz, la sesión se encarga de notificarle de cuándo son añadidos y eliminados de la sesión. Para ello la interfaz tiene los métodos:

```
public void valueBound(HttpSessionBindingEvent e)
public void valueUnbound(HttpSessionBindingEvent e)
```

  El objeto que implemente esta interfaz ejecutará el código de `valueBound()` cuando la sesión lo añada, y el método `valueUnbound()` cuando la sesión lo elimine.

- **HttpSessionActivationListener**: si un objeto asociado a una sesión implementa esta interfaz, la sesión se encarga de notificarles de cuándo el contenedor cambia la sesión entre máquinas virtuales distintas, para un sistema distribuido. Para ello tiene los métodos:

```
public void sessionDidActivate(HttpSessionEvent e)
public void sessionWillPassivate(HttpSessionEvent e)
```

  El objeto que implemente esta interfaz ejecutará el código de `sessionDidActivate()` cuando la sesión se active, y `sessionWillPassivate()` cuando se vuelva pasiva.

2.6. Ejemplos

Aquí tenéis un ejemplo de uso de sesiones. El servlet `ServletSesiones` cuenta el número de visitas a una página en una sola sesión (en una sola ventana de navegador).

```
package ejemplos;

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

public class ServletSesiones extends HttpServlet {

    // Metodo para GET

    public void doGet(HttpServletRequest request,
                      HttpServletResponse response)
                  throws ServletException, IOException {
        response.setContentType("text/html");
        response.setHeader("Cache-Control", "no-cache");

        HttpSession sesion = request.getSession();
        PrintWriter out = response.getWriter();

        if (sesion.isNew()) {
            // Mostramos un mensaje de primera visita

            out.println ("<HTML>");
            out.println ("<BODY>");
            out.println ("Primera visita a la pagina");
            out.println ("<BR>");
            out.println ("</BODY>");
            out.println ("</HTML>");

            sesion.setAttribute("contador", new Integer(1));

        } else {

            // Mostramos el numero de visitas
            // y actualizamos el contador

            int contador = ((Integer)
                (sesion.getAttribute("contador"))).intValue();
            contador++;

            out.println ("<HTML>");
            out.println ("<BODY>");
            out.println ("Visita numero " +
                         contador +
                         " a la pagina en esta sesion");
            out.println ("<BR>");
            out.println ("</BODY>");
            out.println ("</HTML>");

            sesion.setAttribute("contador",
                                new Integer(contador));
        }
   }
}
```

Podéis probar el ejemplo con:

```
http://localhost:8080/appcs/servlet/ejemplos.ServletSesiones
```

El siguiente servlet utiliza como atributo de sesión una clase interna `ObjetoSesion`, que

implementa la interfaz `HttpSessionBindingListener`. Dicho objeto tiene dentro un valor entero, y una cadena. Cada vez que el objeto se añade a la sesión se modifica un mensaje de texto, mostrando el valor entero actual del objeto:

```
package ejemplos;

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

public class EjemploServletListener extends HttpServlet {
    public void doGet(HttpServletRequest request,
                      HttpServletResponse response)
                  throws ServletException, IOException {
        response.setContentType("text/html");
        response.setHeader("Cache-Control",
                           "no-cache");

        HttpSession sesion = request.getSession();
        PrintWriter out = response.getWriter();

        if (sesion.isNew()) {
            // Mostramos mensaje de inicio

            out.println ("<HTML><BODY>" +
                         "Mensaje de inicio" +
                         "</BODY></HTML>");
            sesion.setAttribute("contador",
                                new ObjetoSesion(1));

        } else {

            // Mostramos el valor actual del
            // objeto "contador"

            int contador = ((ObjetoSesion)
                (sesion.getAttribute("contador"))).getValor();
            String mensaje = ((ObjetoSesion)
                (sesion.getAttribute("contador"))).getEnlazado();

            out.println ("<HTML><BODY>");
            out.println ("Valor: " + contador +
                         "<br>Enlazado: " + mensaje);
            out.println ("</BODY></HTML>");

            sesion.setAttribute("contador",
                                new ObjetoSesion(contador+1));
        }
    }
}

class ObjetoSesion implements HttpSessionBindingListener
{
    int valor;
    String enlazado = "NO";

    public ObjetoSesion(int valor) {
        this.valor = valor;
    }

    public void valueBound(HttpSessionBindingEvent e) {
        enlazado = "Objeto enlazado a la sesion " +
                   valor + " veces";
    }
```

```
    public void valueUnbound(HttpSessionBindingEvent e) {
    }

    public String getEnlazado() {
        return enlazado;
    }

    public int getValor() {
        return valor;
    }
}
```

Si cargamos el servlet por primera vez en la sesión, muestra el mensaje:

```
Mensaje de bienvenida
```

Luego, cada vez que recarguemos el servlet se entrará en el bloque `else`, y al llamar al método `setAttribute()` se disparará el método `valueBound()`, actualizando la cadena. Con esto, con cada recarga se mostrará el mensaje:

```
Valor: i
Enlazado: Objeto enlazado a la sesion i veces
```

siendo `i` el número de veces que se ha enlazado (que coincide con el número de veces que se ha cargado el servlet, en este caso). Lo esencial aquí es que esta variable `enlazado` se modifica sólo desde el método `valueBound`, y éste es llamado sólo cuando el objeto se añade a la sesión.