COOKIES DE SESIÓN

Las cookies de sesión permiten a los usuarios ser reconocidos en un sitio web de forma que cualquier cambio que realices, artículo que selecciones o dato que introduzcas se recuerda de una a otra página. El ejemplo más común de esta función es el de la función del carro de la compra en cualquier sitio de e-commerce (tienda online). Cuando visitas una página de un catálogo y seleccionar algunos artículos, la cookie de sesión recuerda tu selección de forma que tu carro de la compra tendrá los artículos que hayas ido seleccionando en el momento de hacer el pago. Sin las cookies de sesión, si haces click en CHECKOUT, la página nueva no reconocería tu actividad anterior en las páginas previas y tu carro de la compra estaría siempre vacío.
El siguiente código muestra cómo un servlet puede crear y enviar una cookie a un cliente:

```
public void doGet(HttpServletRequest peticion, HttpServletResponse respuesta
{
   Cookie miCookie = new Cookie(“idCliente”,”dato”);

   // hacemos que nuestra cookie tenga sentido durante un día
   miCookie.setMaxAge(60*60*24);

   respuesta.addCookie(miCookie);
}
```

Cuando el usuario se conecte a la misma web otra vez, un servlet puede recuperar las cookies del cliente a través del objeto HttpServletRequest tal como se muestra en el código siguiente:

```
Cookie [] cookies = peticion.getCookies();

for(int i=0; i<cookies.length; i++)
{
   Cookie cookieActual = cookies[i];

   String identificador = cookieActual.getName();
   String valor = cookieActual.getValue();

   If(identificador.equals(“idCliente”)
   {
   // tratamiento específico para ese usuario, como por ejemplo mostrar una web
   // personalizada con los últimos artículos que estuvo consultando.
```

```
    }
}
```

Las cookies permiten almacenar la información del usuario en el lado del cliente, lo cual tiene las siguientes ventajas:

• Sobreviven a las caídas del servidor.
• No sobrecargan la memoria del servidor
• Simplifican la recuperación tras una caída del servidor porque almacenan los últimos datos valiosos resultado de las selecciones del cliente antes de que el servidor cayera.

Desgraciadamente también tienen algunos inconvenientes:

• Ofrecen muy poca privacidad y pueden ser leídas por usuarios no autorizados muy fácilmente (especialmente en equipos compartidos o públicos).
• Incrementan la carga de la red.
• Los usuarios pueden deshabilitar su uso en los navegadores.

El objeto HttpSession

Otra solución para mantener la sesión del usuario es el uso de la clase javax.servlet.http.HttpSession. El contenedor de servlets crea un solo objeto de este tipo por cliente al cual se le asocia y este será compartido por todos los servlets. En estos objetos se pueden almacenar pares clave/valor con el método al estilo de los HashMap. Estos objetos no se crean automáticamente. Deberían ser creados por el primer servlet que se encuentra el usuario y el resto lo usarán para almacenar y recuperar información. Se crean de la siguiente manera:

```
HttpSession miSesion = peticion.getSession(true);
```

Esta llamada puede entenderse como “Encuentra mi objeto HttpSession o crea uno caso de no existir”. En este momento, el servidor de aplicaciones genera un identificador de sesión único que se envía al usuario automáticamente mediante una cookie o mediante reescritura de URL según convenga y de forma automática.

El resto de servlets usuarios pueden presuponer que la sesión se creó (si la aplicación está bien diseñada, así debería ser) y para obtener el objeto pueden hacer la llamada así:

HttpSession miSesion = peticion.getSession(false);

que se entiende como “encuentra mi objeto de sesión”. Si no lo encontrara, devolvería un valor null.

Ejemplo:

```
//Creación
public void doGet(HttpServletRequest peticion,
HttpServletResponse respuesta)
throws ServletException, IOException
{
...
HttpSession sesion = peticion.getSession(true);

LinkedList articulos = new LinkedList();

sesion.setAttribute(“articulos”,articulos);
...
// página HTML de bienvenida y formulario para comprar artículos
// y tratamiento del formulario por parte del siguiente servlet
respuesta.getWriter().println(“<html><body>...</body></html>”);
...
}


//Carga
public void doGet(HttpServletRequest peticion,
HttpServletResponse respuesta)
throws ServletException, IOException
{
...
HttpSession sesion = peticion.getSession(false);

LinkedList articulos = (LinkedList)sesion.getAttribute(“articulos”);

String nombreArticulo = peticion.getParameter("nombreArticulo");

Articulo a = new Articulo(nombreArticulo);
articulos.add(a);
...
// página HTML que contiene qué se ha comprado hasta el momento
```

```
// y da la opción de seguir comprando (nueva llamda a este servlet)
// o de pasar a la factura (llamada al siguiente servlet)
respuesta.getWriter().println("<html><body>...</body></html>");
...
}

//Uso y desuso
public void doGet(HttpServletRequest peticion,
HttpServletResponse respuesta)
throws ServletException, IOException
{
...
HttpSession sesion = peticion.getSession(false);
LinkedList articulos = (LinkedList)sesion.getAttribute("articulos");
...
// generar página web con todos los artículos comprados
respuesta.getWriter().println("<html><body>...</body></html>");
sesion.invalidate();
...
}
```

Como vemos, la última línea invalida la sesión con lo que tras esta llamada no se podrá seguir utilizando. Si no hay una llamada explícita a invalidate(), la sesión lo hará automáticamente dependiendo de cómo se haya configurado el servidor de aplicaciones o tras el tiempo indicado en segundos por una llamada previa al método setMaxInactiveInterval().