CAPÍTULO 11

Sesión en Servlets

11.1. Formas de seguir la trayectoria de los usuarios

Los ***servlets*** permiten seguir la trayectoria de un cliente, es decir, obtener y mantener una determinada información acerca del cliente. De esta forma se puede ***tener identificado a un cliente*** (usuario que está utilizando un browser) durante un determinado tiempo. Esto es muy importante si se quiere disponer de aplicaciones que impliquen la ejecución de varios ***servlets*** o la ejecución repetida de un mismo ***servlet***. Un claro ejemplo de aplicación de esta técnica es el de los ***comercios vía Internet*** que permiten llevar un ***carrito de la compra*** en el que se van guardando aquellos productos solicitados por el cliente. El cliente puede ir navegando por las distintas secciones del comercio virtual, es decir realizando distintas conexiones ***HTTP*** y ejecutando diversos ***servlets***, y a pesar de ello no se pierde la información contenida en el carrito de la compra y se sabe en todo momento que es un mismo cliente quien está haciendo esas conexiones diferentes.

El mantener información sobre un cliente a lo largo de un proceso que implica múltiples conexiones se puede realizar de tres formas distintas:

- Mediante ***cookies***
- Mediante ***seguimiento de sesiones*** (***Session Tracking***)
- Mediante la ***reescritura*** de URLs

11.2. Cookies

Estrictamente hablando "cookie" significa galleta. Parece ser que dicha palabra tiene otro significado: se utilizaría también para la ficha que le dan a un cliente en un guardarropa al dejar el abrigo y que tiene que entregar para que le reconozcan y le devuelvan dicha prenda. Éste sería el sentido de la palabra ***cookie*** en el contexto de los ***servlets***: algo que se utiliza para que un ***servidor HTTP*** reconozca a un cliente como alguien que ya se había conectado anteriormente. Como era de esperar, los ***cookies*** en ***Java*** son objetos de la clase ***Cookie***, en el package ***javax.servlet.http***.

El empleo de ***cookies*** en el seguimiento de un cliente requiere que dicho cliente sea capaz de soportarlas. Sin embargo, puede ocurrir que a pesar de estar disponible, dicha opción esté ***desactivada*** por el usuario, por lo que puede ser necesario emplear otras alternativas de seguimiento de clientes como la reescritura de ***URLs***. Esto es debido a que los ***servlets*** envían ***cookies*** a los clientes junto con la respuesta, y los clientes las devuelven junto con una petición. Así, si un cliente tiene activada la opción ***No cookies*** o similar en su navegador, no le llegará la ***cookie*** enviada por el ***servlet***, por lo que el seguimiento será imposible.

Cada ***cookie*** tiene un nombre que puede ser el mismo para varias ***cookies***, y se almacenan en un directorio o fichero predeterminado en el disco duro del cliente. De esta forma, puede mantenerse información acerca del cliente durante días, ya que esa información queda almacenada en el ordenador del cliente (aunque no indefinidamente, pues las ***cookies*** tienen una fecha de caducidad).

La forma en que se envían ***cookies*** es bastante sencilla en concepto. Añadiendo una ***clave*** y un ***valor*** al *header* del mensaje es posible enviar ***cookies*** al cliente, y desde éste al servidor. Adicionalmente, es posible incluir otros parámetros adicionales, tales como ***comentarios***. Sin embargo, estos no suelen ser tratados correctamente por los browsers actuales, por lo que su empleo es desaconsejable. Un servidor puede enviar más de una ***cookie*** al cliente (hasta veinte ***cookies***).

Las ***cookies*** almacenadas en el cliente son enviadas en principio sólo al servidor que las originó. Por este motivo (porque las ***cookies*** son enviadas al ***servidor HTTP*** y **no** al ***servlet***), los ***servlets*** que se ejecutan en un mismo servidor comparten las mismas ***cookies***.

La forma de implementar todo esto es relativamente simple gracias a la clase ***Cookie*** incluida en el ***Servlet API***. Para enviar una ***cookie*** es preciso:

- Crear un objeto ***Cookie***
- Establecer sus atributos
- Enviar la ***cookie***

Por otra parte, para obtener información de una ***cookie***, es necesario:

- Recoger todas las ***cookies*** de la petición del cliente
- Encontrar la ***cookie*** precisa
- Obtener el valor recogido en la misma

11.2.1. Crear un objeto Cookie

La clase ***javax.servlet.http.Cookie*** tiene un constructor que presenta como argumentos un ***String*** con el ***nombre*** de la ***cookie*** y otro ***String*** con su ***valor***. Es importante hacer notar que toda la información almacenada en ***cookies*** lo es en forma de ***String***, por lo que será preciso convertir cualquier valor a ***String*** antes de añadirlo a una ***cookie***.

Hay que ser cuidadoso con los ***nombres*** empleados, ya que aquellos que contengan caracteres especiales pueden no ser válidos. Adicionalmente, aquellos que comienzan por el símbolo de dólar (**$**) no pueden emplearse, por estar reservados.

Con respecto al ***valor*** de la ***cookie***, en principio puede tener cualquier forma, aunque hay que tener cautela con el valor ***null***, que puede ser incorrectamente manejado por los browsers, así como espacios en blanco o los siguientes caracteres:

```
[ ] ( ) = , " / ? @:;
```

Por último, es importante saber que es necesario crear la ***cookie*** antes de acceder al *Writer* del objeto ***HttpServletResponse***, pues como las ***cookies*** son enviadas al cliente en el *header* del mensaje, y éstas deben ser escritas antes de crear el *Writer*.

Por ejemplo, el siguiente código crea una ***cookie*** con el nombre "***Compra***" y el valor de ***IdObjetoAComprar***, que es una variable que contiene la identificación de un objeto a comprar (301):

```
...
String IdObjetoAComprar = new String("301");
if(IdObjetoAComprar!=null)
    Cookie miCookie=new Cookie("Compra", IdObjetoAComprar);
```

11.2.2. Establecer los atributos de la cookie

La clase ***Cookie*** proporciona varios métodos para establecer los valores de una ***cookie*** y sus atributos. Entre otros, los mostrados en la Tabla 11.1.

Todos estos métodos tienen sus métodos ***getXXX()*** correspondientes incluidos en la misma clase.

Por ejemplo, se puede cambiar el valor de una ***cookie*** de la siguiente forma:

```
...
Cookie miCookie=new Cookie("Nombre", "ValorInicial");
miCookie.setValue("ValorFinal");
```

o hacer que sea eliminada al cerrar el browser:

```
miCookie.setMaxAge(-1);
...
```

Métodos de la clase Cookie	Comentarios
public void **setComment(String)**	Si un browser presenta esta ***cookie*** al usuario, el cometido de la ***cookie*** será descrito mediante este comentario.
public void **setDomain(String)**	Establece el patrón de dominio a quien permitir el acceso a la información contenida en la ***cookie***. Por ejemplo .yahoo.com permite el acceso a la ***cookie*** al servidor www.yahoo.com pero no a a.b.yahoo.com
public void **setMaxAge(int)**	Establece el tiempo de caducidad de la ***cookie*** en segundos. Un valor -1 indica al browser que borre la ***cookie*** cuando se apague. Un valor 0 borra la ***cookie*** de inmediato.
public void **setPath(String)**	Establece la ruta de acceso del directorio de los ***servlets*** que tienen acceso a la ***cookie***. Por defecto es aquel que originó la ***cookie***.
Public void **setSecure(boolean)**	Indica al browser que la ***cookie*** sólo debe ser enviada utilizando un protocolo seguro (***https***). Sólo debe utilizarse en caso de que el servidor que haya creado la ***cookie*** lo haya hecho de forma segura.
public void **setValue(String)**	Establece el valor de la ***cookie***
public void **setVersion(int)**	Establece la versión del protocolo de la ***cookie***.

Tabla 11.1. Métodos de la clase ***Cookie***.

11.2.3. Enviar la cookie

Las ***cookies*** son enviadas como parte del *header* de la respuesta al cliente. Por ello, tienen que ser añadidas a un objeto ***HttpServletResponse*** mediante el método ***addCookie(Cookie)***. Tal y como se ha explicado con anterioridad, esto debe realizarse antes de llamar al método ***getWriter()*** de ese mismo objeto. Sirva como ejemplo el siguiente código:

```
...
public void doGet(HttpServletRequest req, HttpServletResponse resp)
                  throws ServletException, IOException {
    ...
    Cookie miCookie=new Cookie("Nombre","Valor");
    miCookie.setMaxAge(-1);
    miCookie.setComment("Esto es un comentario");
    resp.addCookie(miCookie);

    PrintWriter out=resp.getWriter();
    ...
}
```

11.2.4. Recoger las cookies

Los clientes devuelven las ***cookies*** como parte integrante del *header* de la petición al servidor. Por este motivo, las ***cookies*** enviadas deberán recogerse del objeto ***HttpServletRequest*** mediante el método ***getCookies()***, que devuelve un array de objetos ***Cookie***. Véase el siguiente ejemplo:

```
...
Cookie miCookie = null;
Cookie[] arrayCookies = req.getCookies();
miCookie = arrayCookies[0];
...
```

El anterior ejemplo recoge la primera ***cookie*** del ***array de cookies***.

Por otra parte, habrá que tener cuidado, pues tal y como se ha mencionado con anterioridad, puede haber más de una ***cookie*** con el mismo nombre, por lo que habrá que detectar de alguna manera cuál es la ***cookie*** que se necesita.

11.2.5. Obtener el valor de la cookie

Para obtener el valor de una ***cookie*** se utiliza el método ***getValue()*** de la clase ***Cookie***. Obsérvese el siguiente ejemplo. Supóngase que se tiene una tienda virtual de libros y que un usuario ha decidido

eliminar un libro del carro de la compra. Se tienen dos ***cookies*** con el mismo nombre (***compra***), pero con dos valores (***libro1***, ***libro2***). Si se quiere eliminar el valor ***libro1***:

```
...
String libroABorrar=req.getParameter("Borrar");
...
if(libroABorrar!=null) {
    Cookie[] arrayCookies=req.getCookies();
    for(i=0;i<arrayCookies.length;i++) {
        Cookie miCookie=arrayCookies[i];
        if(miCookie.getName().equals("compra")
             &&miCookie.getValue.equals("libro1") {
               miCookie.setMaxAge(0); // Elimina la cookie
        } // fin del if
    } // fin del for
} // fin del if
...
```

Tal y como se ha dicho con anterioridad, una ***cookie*** contiene como valor un ***String***. Este ***String*** debe ser tal que signifique algo para el ***servlet***. Con esto se quiere decir que es responsabilidad exclusiva del programador establecer que formato o codificación va a tener ese ***String*** que almacena la ***cookie***. Por ejemplo, si se tiene una tienda on-line, pueden establecerse tres posibles tipos de status de un producto (con referencia al interés de un cliente determinado por dicho producto): el cliente se ha solicitado información sobre el producto (**A**), el cliente lo tiene contenido en el carrito de la compra (**B**) o el cliente ya ha comprado uno anteriormente (**C**). Así, si por ejemplo el código del producto fuera el **301** y estuviera contenido en el carrito de la compra, podría enviarse una ***cookie*** con el siguiente valor:

```
Cookie miCookie = new Cookie("NombreDeCookie", "301_B");
```

El programador deberá establecer una codificación propia y ser capaz de descodificarlo posteriormente.

11.3. Sesiones (Session Tracking)

Una ***sesión*** es una conexión continuada de un mismo browser a un servidor durante un tiempo prefijado de tiempo. Este tiempo depende habitualmente del servidor, aunque a partir de la versión ***2.1*** del ***Servlet API*** puede establecerse mediante el método ***setMaxInactiveInterval(int)*** de la interface ***HttpSession***. Esta interface es la que proporciona los métodos necesarios para mantener ***sesiones***.

Al igual que las ***cookies***, las ***sesiones*** son compartidas por todos los ***servlets*** de un mismo servidor. De hecho, por defecto se utilizan ***cookies*** de una forma implícita en el mantenimiento de ***sesiones***. Por ello, si el browser no acepta ***cookies***, habrá que emplearse las ***sesiones*** en conjunción con la reescritura de **URLs** (Ver apartado 11.4).

La forma de obtener una ***sesión*** es mediante el método ***getSession(boolean)*** de un objeto ***HttpServletRequest***. Si este ***boolean*** es ***true***, se crea una sesión nueva si es necesario mientras que si es ***false***, el método devolverá la ***sesión*** actual. Por ejemplo:

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

crea una nueva ***sesión*** con el nombre ***miSesion***.

Una vez que se tiene un objeto ***HttpSession***, es posible mantener una colección de pares ***nombre de dato/valor de dato***, de forma que pueda almacenarse todo tipo de información sobre la ***sesión***. Este valor puede ser cualquier objeto de la clase ***Object*** que se desee. La forma de añadir valores a la ***sesión*** es mediante el método ***putValue(String ,Object)*** de la clase ***HttpSession*** y la de obtenerlos es mediante el método ***getValue(String , Object)*** del mismo objeto. Esto puede verse en el siguiente ejemplo:

```
...
HttpSession miSesion=req.getSesion(true);
CarritoCompras compra = (CarritoCompras) miSesion.getValue(miSesion.getId());

if(compra==null) {
    compra = new CarritoCompras();
    miSesion.putValue(miSesion.getId(), compra);
}
...
```

En este ejemplo, se supone la existencia de una clase llamada ***CarritoCompras***. En primer lugar se obtiene una nueva ***sesión*** (en caso de que fuera necesario, si no se mantendrá una creada previamente), y se trata de obtener el objeto ***CarritoCompras*** añadido a la ***sesión***. Obsérvese que para ello se hace una llamada al método ***getId()*** del objeto ***miSesion***. Cada ***sesión*** se encuentra identificada por un identificador único que la diferencia de las demás. Este método devuelve dicho identificador. Esta es una buena forma de evitar confusiones con el nombre de las ***sesiones*** y el de sus valores. En cualquier caso, al objeto ***CarritoCompras*** se le podía haber asociado cualquier otra clave. Si no se hubiera añadido previamente el objeto ***CarritoCompras*** a la ***sesión***, la llamada al método ***getValue()*** tendría como resultado ***null***. Obsérvese además, que es preciso hacer un **cast** para pasar el objeto ***Object*** a objeto ***CarritoCompras***.

En caso de que compra sea ***null***, es decir, que no existiera un objeto añadido previamente, se crea un nuevo objeto ***CarritoCompras*** y se añade a la sesión ***miSesion*** mediante el método ***putValue()***, utilizando de nuevo el identificador de la ***sesión*** como nombre.

Además de estos métodos mencionados, la interface ***HttpSession*** define los siguientes métodos:

- ***getCreationTime():*** devuelve el momento en que fue creado la ***sesión*** (en milisegundos).
- ***getLastAccessedTime():***devuelve el último momento en que el cliente realizó una petición con el identificador asignado a una determinada ***sesión*** (en milisegundos)
- ***getValueNames():*** devuelve un array con todos los nombres de los objetos asociados con la ***sesión***.
- ***invalidate():*** invalida la ***sesión*** en curso.
- ***isNew():*** devuelve un ***boolean*** indicando si la ***sesión*** es "nueva".
- ***removeValue(String):*** elimina el objeto asociado con una determinada clave.

De todos los anteriores métodos conviene comentar dos en especial: ***invalidate()*** y ***isNew()***.

El método ***invalidate()*** invalida la sesión en curso. Tal y como se ha mencionado con anterioridad, una sesión puede ser invalidada por el propio servidor si en el transcurso de un intervalo prefijado de tiempo no ha recibido peticiones de un cliente. *Invalidar* quiere decir eliminar el objeto ***HttpSession*** y los valores asociados con él del sistema.

El método ***isNew()*** sirve para conocer si una sesión es *"nueva"*. El servidor considera que una sesión es nueva hasta que el cliente se una a la sesión. Hasta ese momento ***isNew()*** devuelve ***true***. Un valor de retorno ***true*** puede darse en las siguientes circunstancias:

- El cliente todavía no sabe nada acerca de la ***sesión***
- La ***sesión*** todavía no ha comenzado.
- El cliente no quiere unirse a la ***sesión***. Ocurre cuando el browser tiene la aceptación de ***cookies*** desactivada.

11.4. Reescritura de URLs

A pesar de que la mayoría de los browser más extendidos soportan las ***cookies*** en la actualidad, para poder emplear ***sesiones*** con clientes que o bien no soportan ***cookies*** o bien las rechazan, debe utilizarse la reescritura de ***URLs***. No todos los servidores soportan la reescritura de ***URLs*** (por ejemplo el ***servletrunner*** que acompaña el ***JSDK***).

Para emplear está técnica lo que se hace es incluir el código identificativo de la ***sesión*** (***sessionId***) en el ***URL*** de la petición. Los métodos que se encargan de reescribir el ***URL*** si fuera necesario son ***HttpServletResponse.encodeUrl()*** y ***HttpServletResponse.encodeRedirectUrl()*** (sustituidas en el ***API 2.1*** por ***encodeURL()*** **y** ***encodeRedirectURL()*** respectivamente). El primero de ellos lee un ***String*** que

representa un ***URL*** y si fuera necesario la reescribe añadiendo el identificativo de la ***sesión***, dejándolo inalterado en caso contrario. El segundo realiza lo mismo sólo que con ***URLs*** de redirección, es decir, permite reenviar la petición del cliente a otro ***URL*** .

Véase el siguiente ejemplo:

```
...
HttpSession miSesion=req.getSesion(true);
CarritoCompras compra = (CarritoCompras)miSesion.getValue(miSesion.getId());

if(compra==null) {
    compra = new CarritoCompras();
    miSesion.putValue(miSesion.getId(), compra);
}
...

PrintWriter out = resp.getWriter();
resp.setContentType("text/html");
...
out.println("Esto es un enlace reescrito");
out.println("<a href\""+
resp.encodeUrl("/servlet/buscador?nombre=Pedro")+"\"</a>");
...
```

En este caso, como hay una ***sesión***, la llamada al método ***encodeUrl()*** tendría como consecuencia la reescritura del enlace incluyendo el identificativo de la ***sesión*** en él.