



Componentes Web

Sesión 3: Cookies y sesiones



Puntos a tratar

- Cookies
 - Enviar y obtener una cookie
 - Ejemplo
- Sesiones
 - Obtener una sesión
 - Guardar y recuperar datos en la sesión
 - Invalidar la sesión
 - Reescribir URLs
 - Oyentes
 - Ejemplos



Definición de cookie

- Una cookie es un objeto de tipo *nombre = valor* donde se asigna un valor *valor* a una variable *nombre*, de forma que el servidor Web y el navegador mantienen la variable en memoria durante un tiempo
- En Javascript se tiene un objeto *document.cookie* con la forma:

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



Características de las cookies

- Un navegador puede soportar hasta 20 cookies por servidor, de al menos 4 KB cada una
- Los servlets que se ejecutan en el mismo servidor comparten las cookies
- Podremos tener así una lista de *cookies* para almacenar valores relevantes para cada usuario, y poder:
 - Identificar a un usuario durante una o varias sesiones
 - Personalizar un sitio web según el usuario que entre
- No debemos depender de las cookies, pues muchos navegadores las deshabilitan



Enviar una cookie al cliente

- Primero se crea la cookie con el nombre y valor

```
Cookie c = new Cookie("nombre", "valor");
```

- Después se establecen los atributos de la cookie

```
c.setComment("Comentario");
```

```
c.setMaxAge(120);
```

...

- Si a *setMaxAge()* le pasamos un valor negativo se borrará la cookie al cerrar el navegador, un valor de 0 la borra instantáneamente, y un valor positivo *n* la borra tras *n* segundos
- Finalmente, se envía la cookie

```
response.addCookie(c);
```



Cookies y cabeceras

- Las cookies son parte de la cabecera, y deben enviarse ANTES de obtener el *Writer* donde escribir la respuesta:

```
public void doGet(HttpServletRequest request,
                  HttpServletResponse response)
    throws ServletException, IOException
{
    Cookie c = new Cookie("nombre", "Pepe");
    c.setMaxAge(180);
    response.addCookie( c );
    PrintWriter out = response.getWriter();
    ...
}
```



Obtener una cookie del cliente

- Primero se obtienen todas las cookies de la petición

```
Cookie[] c = request.getCookies();
```

- Después se busca la cookie por su nombre, y se obtiene su valor

```
String valor;  
for (int i = 0; i < c.length; i++)  
{  
    if (c[i].getName().equals("nombre"))  
        valor = c[i].getValue();  
}
```



Qué es el seguimiento de sesiones

- El seguimiento de sesiones es un mecanismo empleado por los servlets para gestionar el estado de las peticiones de un mismo cliente a lo largo de un período de tiempo
- Las sesiones se comparten por los servlets a los que accede un cliente
- Para utilizar sesiones se debe:
 - Obtener el objeto sesión para el usuario o cliente
 - Almacenar u obtener datos de dicho objeto
 - Invalidar la sesión (opcionalmente)



Obtener una sesión

- Utilizamos el método `getSession()` de `HttpServletRequest` para obtener una sesión

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

- El primer método crea una si no existe, y el segundo sólo la crea si su parámetro es *true*.
- El método `isNew()` de `HttpSession` permite comprobar si es una sesión nueva o existente
- Se debe obtener la sesión antes de escribir nada en la respuesta



Guardar y obtener datos de la sesión

- *HttpSession* tiene métodos para obtener algunas propiedades particulares

```
String id = ses.getId();
```

- Además, podemos asignar, obtener y eliminar atributos de la sesión, identificándolos con un nombre

```
ses.setAttribute("objeto1", new MiObjeto());  
MiObjeto mo = (MiObjeto)(ses.getAttribute("objeto1"));  
ses.removeAttribute("objeto1");
```



Invaldar la sesión

- Al invalidar una sesión eliminamos el objeto *HttpSession* asociado. Se tienen los métodos

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

- El primero obtiene el tiempo en segundos entre dos accesos a partir del cual la sesión se invalida automáticamente, y el segundo establece dicho tiempo
- El tercero invalida la sesión manualmente



Compatibilidad con los navegadores

- El seguimiento de sesiones por defecto emplea cookies para almacenar sus datos
- Algunos navegadores no utilizan cookies
- Para solucionarlo, se emplea la *reescritura de URLs*
- En *HttpServletResponse* tenemos los métodos

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

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

- El primero se utiliza cuando pongamos URLs en el contenido de la página
- El segundo se utiliza cuando redirijamos a otra página
- Devuelven la URL reescrita (si ha sido necesaria la reescritura porque no hay cookies), donde se le ha añadido la información de sesión a la propia URL



Ejemplo de reescritura

```
public void doGet(...)...
{
    String url = response.encodeURL("http://host/pagina.jsp");
    out.println("<a href=\"\" + url + \"\">Enlace</a>");

    String url2 = response.encodeRedirectURL("http://host/pagina.jsp");
    response.sendRedirect(url2);
}
```



Oyentes

- Existen tres tipos de oyentes sobre las sesiones
 - *HttpSessionListener*: para eventos de crear/terminar sesión

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

Las clases que implementen esta interfaz deben configurarse en el descriptor (web.xml) con etiquetas <listener>

```
<listener>
    <listener-class>clase</listener-class>
</listener>
```
 - *HttpSessionBindingListener*: para eventos sobre objetos añadidos a una sesión

```
public void valueBound(HttpSessionBindingEvent e)
public void valueUnbound(HttpSessionBindingEvent e)
```
 - *HttpSessionActivationListener*: para eventos de cambios entre máquinas virtuales distintas