Tema 6: Sesiones en Java

Sesiones en Java

El seguimiento de sesión es un mecanismo que los servlets utilizan para mantener el estado sobre una serie de peticiones desde un mismo navegador durante un periodo de tiempo.

Las sesiones son compartidas por los servlets a los que accede el cliente. Así, una aplicación compuesta por varios servlet es capaz de mantener el estado mediante el uso de sesiones
Para utilizar el seguimiento de sesión debemos:

- Obtener una sesión (un objeto **HttpSession**) para un usuario.
- Almacenar u obtener datos desde el objeto **HttpSession**.
- Invalidar la sesión (si así se desea).

1. **Obtener una Sesión** desde la petición

HttpSession ***getSession***(boolean create)

El método **getSession** del objeto **HttpServletRequest** devuelve una sesión de usuario.
Tiene un argumento de entrada para indicar si crear la sesión (en caso de no existir)

- Cuando llamamos al método con su argumento como **true** queremos decir: "inicia una nueva sesión en caso de no existir ya"
- Si pasamos false como argumento y no existe una sesión, getSession simplemente devolverá null

Para mantener la sesión apropiadamente, debemos llamar a **getSession** antes de escribir la respuesta.

Atención: Si respondemos utilizando un **PrintWriter**, entonces debemos llamar a **getSession** antes de acceder al **PrintWriter**, no sólo antes de enviar la respuesta.

```
public class CatalogoServlet extends HttpServlet {
        public void doGet (HttpServletRequest request, HttpServletResponse response)
                                                        throws ServletException,IOException{
                // Obtiene la variable sesión para el usuario.
                HttpSession session = request.getSession(true);
                ...
                out = response.getWriter();
                ...
        }
}
```

2. Manejar Atributos de sesión

Las sesiones de por sí no son muy útiles sin atributos de sesión. Un atributo de sesión es un objeto Java, identificado por un nombre(String), que está asociado a la sesión.HttpSession dispone de los métodos setAttribute y getAttribute para añadir y consultar atributos de la sesión, respectivamente:

void **setAttribute** (String nombre, Object valor);

Object **getAttribute**(String nombre);

Ejemplo:

```
public void doGet(HttpServletRequest request, HttpServletResponse response)  throws ……. {
        String nombre = request.getParameter("nombre");
        String password = request.getParameter("password");
        DbUser user = buscarEnBD(nombre, password);
        HttpSession sess = request.getSession(true);
        sess.setAttribute("USER", user);
        if (user == null) {
                // Mensaje “Error usuario-password”
        } else {
                // Mensaje “Login OK”
        }
}
```

En este ejemplo, suponemos que buscarEnDB() devuelve un objeto de la clase DbUser, que encapsula datos de un usuario determinado recuperados de una base de datos. Asumimos que este método devuelve null si el usuario no existe o si la contraseña no es la correcta.
En cualquier caso, añadimos a la sesión un atributo de nombre “USER” que contiene como valor un objeto DbUser con los datos de un usuario
Este código borraría toda la información de un usuario previo, cuando se realiza un login erróneo

Siguiendo el ejemplo, una vez que un usuario se ha logueado, podríamos realizar un seguimiento del mismo (conocer información del usuario actualmente conectado) utilizando el método getAttribute( )

```
private DbUser getCurrentUser(HttpServletRequest request) {
        HttpSession sess = request.getSession(false);
        if (sess == null)
                return null;
        return (DBUser) sess.getAttribute("USER");  //Tener en cuenta que un casting de null dará error
}
```

3. Invalidar la Sesión

Una sesión de usuario puede ser invalidada manual o automáticamente, dependiendo de dónde se esté ejecutando el servlet. (Las aplicaciones Java suelen invalidar una sesión cuando no hay peticiones de página por un periodo de tiempo de 30 minutos).
Invalidar una sesión significa eliminar el objeto **HttpSession** y todos sus atributos del sistema.

Para eliminar todo rastro de una sesión del servidor, llamaremos al método invalidate() de la sesión

```
private void doLogout(HttpServletRequest req) {
        HttpSession sess = req.getSession(false);
        if (sess != null) {
                sess.invalidate();
        }
}
```

Almacenar y Obtener Datos desde la Sesión

HttpSession proporciona, además, otros métodos que permiten almacenar y recuperar propiedades y datos de la sesión: getId( ), getCreationTime( ), getLastAccessedTime( ), getAttributeNames() , getValue( ), putValue(String, Object), removeAttribute(String), etc

Reescritura de URL

Por defecto, el seguimiento de sesión utiliza cookies para asociar un identificador de sesión con un usuario. Si queremos dar soporte a usuarios que accedan a un servlet con un navegador que no soporta cookies, se debe utilizar **reescritura de URL.** (como en php)

La reescritura de la URL consiste en que el servidor añada en las URLs presentes en la respuesta HTTP transmitida al cliente, un parámetro correspondiente al identificador de sesión.
Este mecanismo no es muy complejo de implementar gracias a los métodos

String **encodeURL** (String url)
y
String **encodeRedirectURL**(String url)

disponibles en el objeto HttpServletResponse.

Estos 2 métodos reciben como parámetro una cadena de caracteres que representa la URL a transformar, a la que añadirán un parámetro correspondiente al identificador de sesión. Además, contienen un mecanismo que permite verificar automáticamente si no hay otra solución que se pueda usar (las cookies) para transmitir el id de sesión al cliente. Si detectan que hay otra solución disponible para asegurar esta transferencia, dejan sin modificar la cadena de caracteres que se les ha pasado por parámetro.

Ejemplo 1

- Caso de un formulario HTML generado por un servlet.

```
  out.println("<FORM action=\"" +
response.encodeURL("/riJEE/Seguir") +"\">");
```

el código HTML construido por esta instrucción

```
<FORM
action="/riJEE/Seguir;jsessionid=43A9FBA967BBA4B68AE3114B7E4745CA">
```

Ejemplo 2

```
String originalURL = someURL;
String encodedURL = response.encodeRedirectURL(originalURL);
response.sendRedirect(encodedURL);
```

Cookies en Java

1. Crear una Cookie: new Cookie (nombre, valor)

   Cookie libro = new Cookie("comprar", "301");

2. Asignar atributos

   - Asignar su duración en segundos: setMaxAge (num_segundos)
     libro.setMaxAge(60*60*24);
     libro.setMaxAge(0); //se borra

   Por defecto, las cookies duran hasta el cierre del navegador, si no les damos mayor duración, mediante este método.

   - Asignar un valor nuevo: setValue(String)
     libro.setValue(“Madame Bovary”);

   - Asignar un comentario: setComment(String)
     libro.setComment("Almacena el código del libro");

3. Envío de la cookie: Método addCookie (Cookie)
   Se añaden con el metodo addCookie de la clase HttpServletResponse, antes de llamar al metodo getWriter()

   ```
   response.addCookie(libro);
   PrintWriter out = response.getWriter();
   ```

4. Leer, de vuelta, valores de cookies: Método getCookies() de HttpServletRequest

   Cookie[ ] getCookies( ) → Devuelve null si el navegador cliente no envía cookies de nuestro sitio web

```
Cookie[] cookies = request.getCookies();
if (cookies != null)
   for (Cookie ck : cookies) {
                if ("prefResultsPerPage".equals(ck.getName())) {
                    String prefValue = ck.getValue();
                ..
   }
}
```

Métodos de HttpSession

Object getAttribute(String nombreAtributo): devuelve el objeto almacenado en la sesión actual, identificado por el nombre pasado como parámetro.
Ya que este método devuelve un objeto de la clase genérica Object, a la hora de recuperarlo se deberá realizar el cast correspondiente. Este método devuelve null si el objeto indicado no existe.

Enumeration getAttributeNames(): este método devuelve en una Enumeration, los nombres de todos los objetos almacenados en la sesión actual.

long getCreationTime(): devuelve la fecha y hora en la que fue creada la sesión, como timestamp (milisegundos desde el 1 de enero de 1970)

String getId(): devuelve una cadena que se corresponde con el identificador único asignado a la sesión. Este valor se corresponde con el valor de el cookie.

JSESSIONID utilizada para poder realizar el mantenimiento de la sesión de un usuario determinado, y en el caso de no utilizar cookies se corresponde con la información que se añade al final de cada enlace cuando se utiliza el mecanismo de reescritura de URLs.

long getLastAccesesedTime(): devuelve como timestamp el momento de la última petición realizada en la sesión actual.

int getMaxInactiveInterval(): devuelve el máximo intervalo de tiempo, en segundos, en el que una sesión puede permanecer activa, sin que el cliente realice ninguna petición. Suele ser de 30 segundos. Una vez transcurrido este tiempo el contenedor de servlets (servlet container) destruye la sesión, liberando todos los atributos asociados a la misma

void invalidate(): destruye la sesión de forma explícita, y libera de memoria todos sus objetos (atributos)

boolean isNew(): devuelve verdadero si la sesión se acaba de crear en la petición actual.

void removeAttribute(String nombreAtributo): elimina el objeto almacenado en la sesión cuyo nombre se pasa por parámetro. Si el nombre no se corresponde a un atributo de sesión, no realizará ninguna acción.

void setAttribute(String nombre, Object valor): almacena un objeto en la sesión utilizando como referencia el nombre indicado como parámetro a través de un objeto String.

void setMaxInactiveInterval(int intervalo): establece, en segundos, el máximo tiempo que una sesión puede permanecer inactiva antes de ser destruida por el contenedor de servlets.

Atributos: ámbitos

Los atributos usados por los servlets, son OBJETOS (de cualquier tipo) accesibles por un nombre.

Así, para establecer, borrar y acceder a atributos, utilizaremos los métodos:

- …..**setAttribute**("nombre", objeto);
- …..**removeAttribute**("nombre");
- Objeto objeto= (Objeto)....**getAttribute**("nombre");

respectivamente.

Es importante diferenciarlos de los parámetros, que ya conoces.
Los parámetros son cadenas de texto asociadas a un nombre, que nos llegan bien desde campos de formulario o bien desde la URL (……?param1=valor1&param2=valor2&param3=valor3….).
Un parámetro no puede contener un objeto, sólo un String

Un atributo, por el contrario, ES UN OBJETO DE CUALQUIER TIPO y puede ser definido en distintos ámbitos (scopes), dependiendo de su alcance o duración (de cuánto queremos que dure el atributo)

En este momento, debes distinguir estos 3 ámbitos para 1 atributo:

- Ámbito de petición o request:
  El atributo tiene vigencia en la petición http actual

  Ejemplo: Atributo para guardar un error puntual que se visualizará en la siguiente página

- Ámbito de sesión o session: El atributo tiene vigencia durante la sesón navegación de un cliente.

  Ejemplo: Atributo para guardar el carro de compra del usuario actual (que podría ser un HashMap o cualquier colección compleja)

- Ámbito de contexto o application: El atributo tiene vigencia mientras el servlet esté atendiendo peticiones. Para compartir datos entre varias sesiones.
  - Estableceremos los atributos de ámbito “aplicación” en 1 lugar que se ejecute una única vez, durante el tiempo de vida de la aplicación web, por ejemplo, en el método init de un servlet.
  - Desde un servlet, accederemos a un atributo de ámbito “aplicación” mediante la sentencia:
    this.getServletContext().getAttribute(“nombreatributo”);

  Ejemplo: Atributo para guardar en forma de HashMap una tabla de nuestra BD, muy accedida por todos los clientes y que no cambia (con el objetivo de reducir el nº de accesos a la BD)