P.O.O. – Trabajo Voluntario

Mayo, 2002

Manejo de sesiones con JSP

Alberto Velasco Florines

Juan Francisco De Paz Santana

Departamento de Informática y Automática

Universidad de Salamanca

Información de los autores:

Alberto Velasco Florines
3ºI.T.I.S
Facultad de Ciencias - Universidad de Salamanca
betovf@wanadoo.es

Juan Francisco De Paz Santana
3ºI.T.I.S
Facultad de Ciencias - Universidad de Salamanca
fcofds@ole.com

Resumen

Este documento trata sobre el manejo de las sesiones con JSP, de cómo se pueden utilizar para el almacenamiento, recuperación e intercambio de información entre páginas web de un mismo servidor.

Abstract

This document deals with session managing with JSP, how they can be used to storage, recovering and interchange of information between web pages in a same server.

Tabla de Contenidos

1. Introducción

El protocolo HTTP permite acceder a páginas web y enviar datos de un formulario pero tiene una limitación que consiste en que no puede almacenar cuando se cambia de servidor o de página dentro de un mismo servidor. Por esta razón a este protocolo se le conoce como protocolo sin estado.

Cuando se solicita una página independientemente del tipo que sea, el servidor abre una conexión por la que envía los datos y luego ésta es cerrada una vez que ha terminado.

2. Qué es una sesión

Una sesión es una serie de comunicaciones entre un cliente y un servidor en la que se realiza un intercambio de información. Por medio de una sesión se puede hacer un seguimiento de un usuario a través de la aplicación.

El tiempo de vida de una sesión comienza cuando un usuario se conecta por primera vez a un sitio web pero su finalización puede estar relacionada con tres circunstancias:

- Cuando se abandona el sitio web.
- Cuando se alcanza un tiempo de inactividad que es previamente establecido, en este caso la sesión es automáticamente eliminada. Si el usuario siguiera navegando se crearía una nueva sesión.
- Se ha cerrado o reiniciado el servidor

Una posible aplicación de las sesiones es en el comercio electrónico. En este caso una sesión permite ir eligiendo una serie de productos e irlos añadiendo a nuestro "carrito" y así hasta finalizar la compra. Sin el uso de sesiones no se podría hacer porque al ir navegando de una página a otra se iría perdiendo toda la información.

También se utilizan para la identificación de usuarios, en la que se deben de introducir un *login* y un *password*. Después de haber hecho esto el usuario tendrá una serie de permisos sobre las páginas que va a visitar, de tal forma que si un usuario intenta pasar a una página si haberse identificado, el sistema comprobará que no se ha identificado y sería redireccionado a la página de identificación. Para poder realizarse estas operaciones es necesario almacenar en unas tipo sesión la información necesaria para saber que el usuario se ha identificado correctamente.

Para poder hacer uso de las sesiones en JSP hay que poner el atributo *session* de la directiva *page* a *true*, de esta forma se notifica al contenedor que la página interviene en un proceso que utiliza las sesiones del protocolo HTTP:

```
<%@page session='true'%>
```

El manejo de las sesiones impide el intercambio de datos entre ellas ya que se trata información específica para cada usuario e incluso si se trata del mismo usuario.

3. Manejo de las sesiones

En JSP las acciones que se pueden realizar sobre las sesiones se lleva a cabo mediante la interface HttpSession y los métodos que implementa. Esta interfaz está incluida dentro del

paquete javax.servlet.http y es utilizada por el contenedor de páginas JSP para crear una sesión entre el servidor y el cliente.

Para obtener la sesión de un usuario se utiliza el método getSession() que devuelve una interfaz de tipo HttpSession.

```
<%
        HttpSession sesion=request.getSession();
%>
```

Una vez creado el objeto de tipo sesión es posible acceder a una serie de datos sobre la misma. Uno de estos datos es idSession que devuelve un identificador único asociado a una sesión:

```
<%
        HttpSession sesion=request.getSession();
        out.println("IdSesion: "+sesion.getId());
%>
```

Cada interprete de JSP generará un identificador diferente. Así en el caso del servidor Jakarta-Tomacat3.2.3 , el resultado sería similar a :

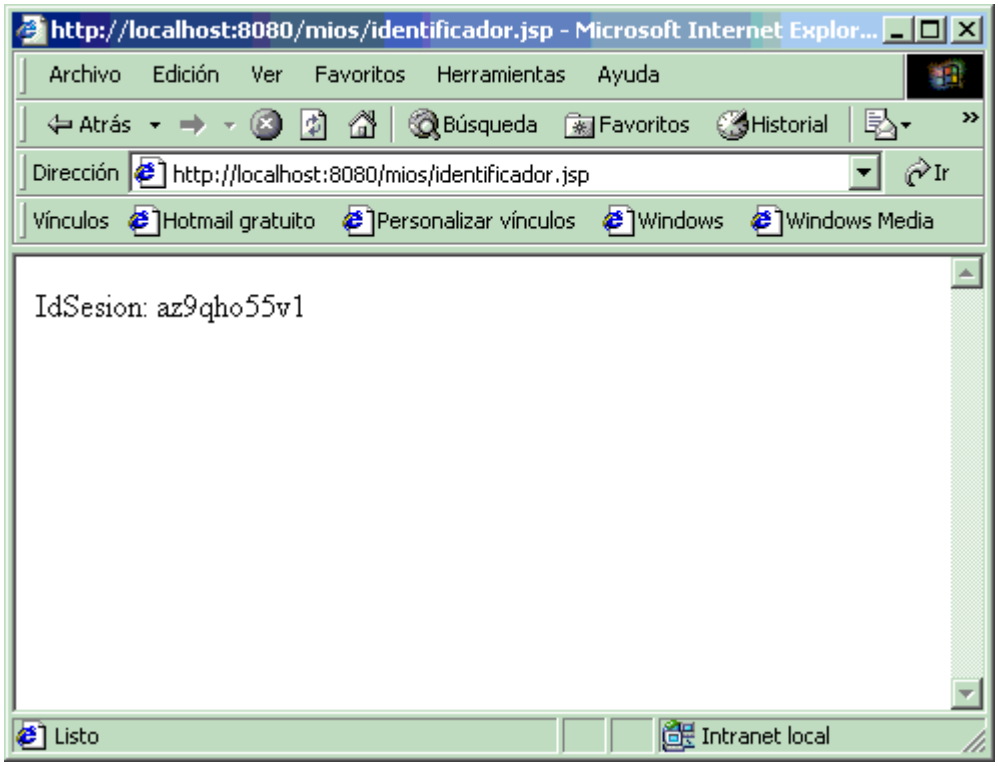

Figura 1. Todas las sesiones que se crean tienen asociado un identificador (id) que es posible conocer a través del método getId()

Es posible conocer el momento en el que se creó la sesión:

```
<%@page import="java.util.*" session="true"%>
<%
```

```
        HttpSession sesion=request.getSession();
        out.println("Creación: "+sesion.getCreationTime());
        Date momento=new Date(sesion.getCreationTime());
        out.println("<BR>Creación: "+momento);
%>
```

En el primer caso se muestra el dato tal cual lo devuelve el método getCreationTime(), que es una fecha en formato long, mientras que en el segundo caso se formatea para que tenga un aspecto más común.

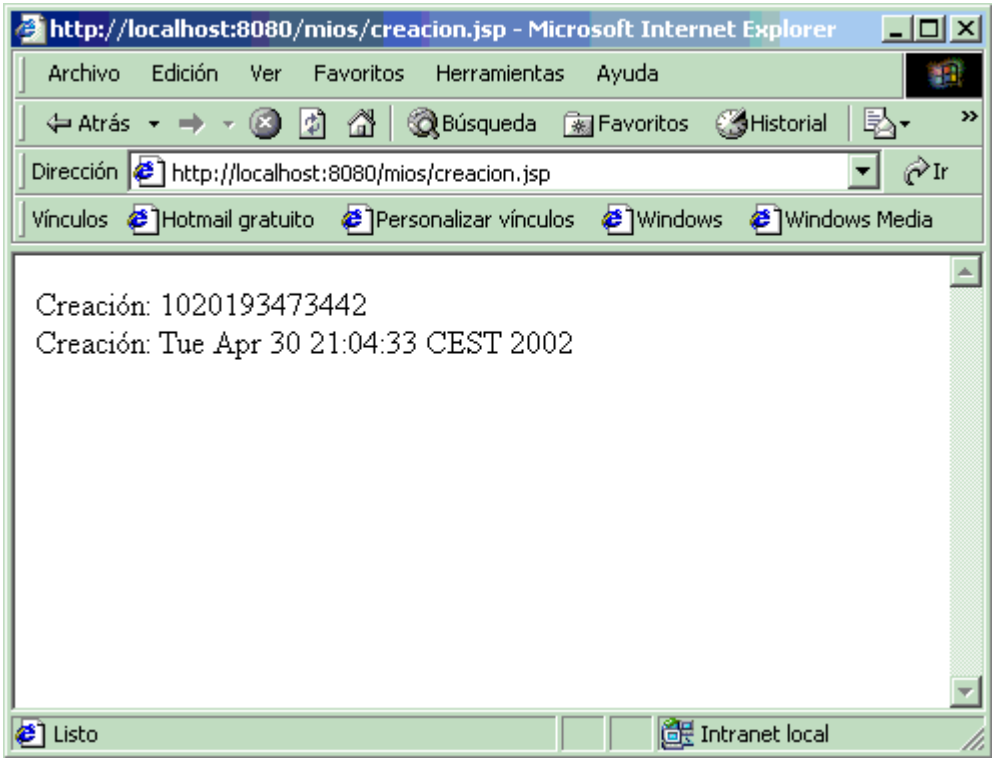

Figura 2. A través del método getCreationTime() es posible conocer la Fecha y la hora en que se creó la sesión.

También se puede conocer la fecha y hora de la última vez que el cliente accedió al servidor con el que se creó la sesión, utilizando el método getLastAccesedTime():

```
<%
        Date acceso=new Date(sesion.getLastAccessedTime());
        out.println("Último acceso: "+acceso+"<br>");
%>
```

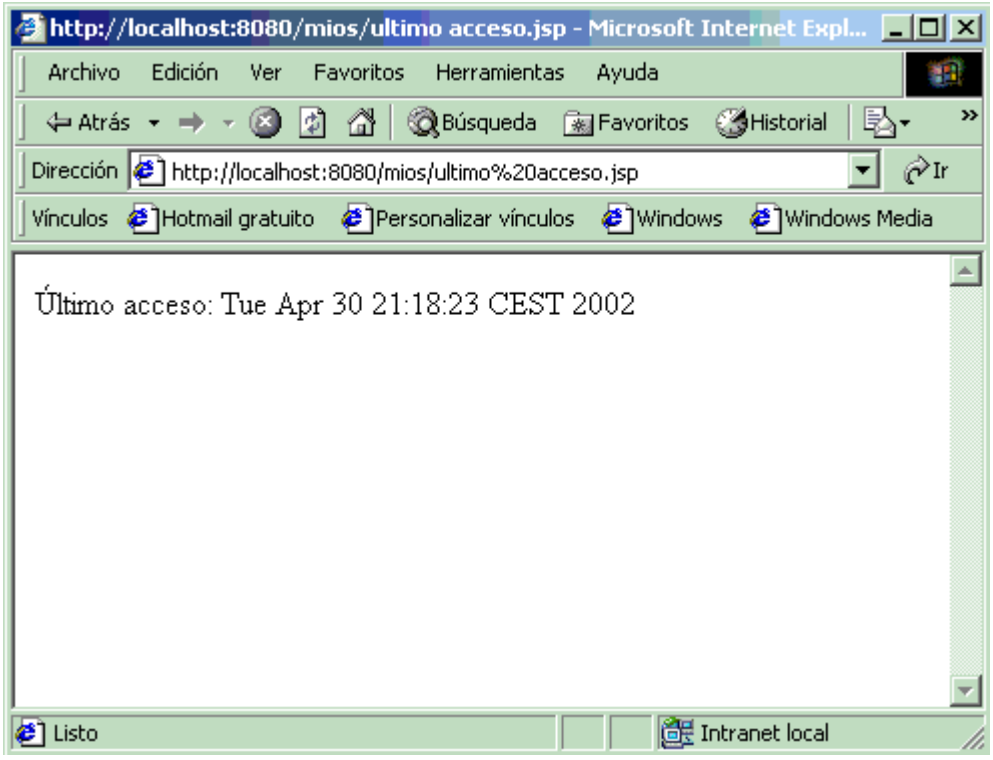

Figura 3. El objeto HttpSession también proporciona un método para conocer cuando fue la última vez que se actualizó la sesión creada.

Teniendo en cuenta el momento en el que ser creó la sesión y la última vez que se accedió al servidor, se puede conocer el tiempo que lleva el cliente conectado al servidor, o lo que es lo mismo el tiempo que lleva el usuario navegando por la páginas JSP:

```
<%
      long longDuracion=sesion.getLastAccessedTime()
      sesion.getCreationTime();
      Date duracion=new Date(longDuracion);
      out.println("Duracion:
      "+duracion.getMinutes()+"min."+duracion.getSeconds()+"seg")
      ;
%>
```

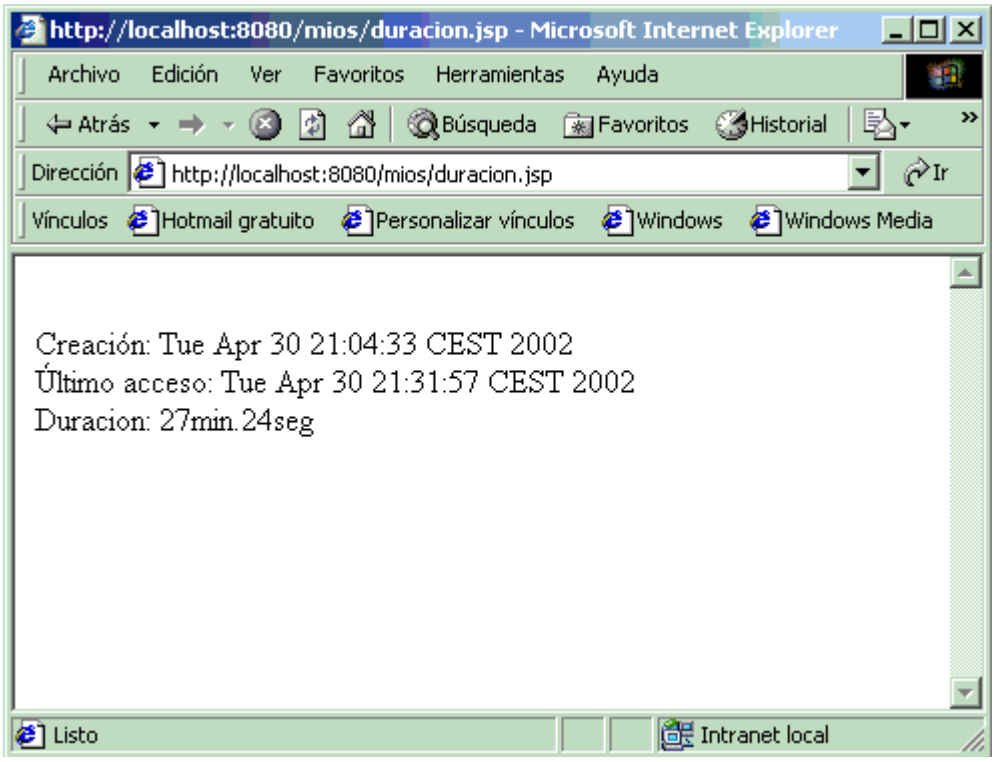

Figura 4. Utilizando el momento en el que se creó la sesión y la última vez accedió se puede llegar a saber el tiempo que ha estado el usuario visitando nuestras páginas web.

La interfaz HttpSession ofrece el método isNew() mediante el cual es posible saber si la sesión creada es nueva o se está tomando de una previamente creada:

```
<%
        HttpSession sesion=request.getSession();
        out.println("nueva: "+sesion.isNew());
%>
```

Si se ejecuta el ejemplo la primera vez el método devolverá true, ya que previamente no había ninguna sesión y ha sido creada en ese instante. Si se recarga la página devolverá false ya que la sesión ya ha sido creada.

Las sesiones no necesitan ningún tipo de mantenimiento, una vez creadas no es necesario utilizarlas de forma obligatoria o tener que refrescar los datos asociados a las mismas, se deben ver como una variable más con la diferencia que pueden ser utilizadas en cualquier página independientemente del lugar en el que hayan sido creadas.

4. Guardar objetos en una sesión

Para guardar un objeto en una sesión se utiliza el método setAttribute(), que ha sustituido al método putValue(). Este método utiliza dos argumentos:

- El primero es el nombre que identificará a esa variable.
- El segundo es el dato que se va a guardar.

```
SetAttribute(java.lang.String name, java.lang.Object value)
```

Un ejemplo de cómo guardar una cadena de texto en la sesión:

```
<%@page import=”java.util.*” session=”true” %>
<%
      HttpSession sesion=request.getSession();
      sesion.setAttribute(“trabajo”,”Paginas de JSP”);
%>
```

Si se quiere pasar un parámetro que no sea un objeto es necesario realizar una conversión:

```
<%@page import=”java.util.*” session=”true” %>
<%
      HttpSession sesion=request.getSession();
      Integer edad=new Integer(26);
      sesion.setAttribute(“edad”,edad);
%>
```

Si se hubiera utilizado el valor entero en vez del objeto Integer, el resultado habría sido similar al siguiente.

```
Incompatible type for meted. Can’t convert int to java.lang.Object.
```

En el primer ejemplo este no sucedería puesto que una cadena es un objeto de tipo String, no así un entero . Así habría sido igual si en el primer caso ponemos:

```
<%@page import=”java.util.*” session=”true” %>
<%
      HttpSession sesion=request.getSession();
      String nombre=new String(“Paginas de JSP·);
      sesion.setAttribute(“trabajo”,nombre);
%>
```

En caso de tratarse objeto de tipo Vector (parecido a un array con dos diferencias: la primera es que puede almacenar todo tipo de objetos, y la segunda es que no es necesario establecer de forma previa el tamaño que va a tener) que almacene los 7 días de la semana. El código sería el siguiente:

```
<%@page import=”java.util.*” session=”true” %>
<%
      HttpSession sesion=request.getSession();
      Vector v=new Vector();
      v.addElement(new String(“Lunes”));
      v.addElement(new String(“Martes”));
      v.addElement(new String(“Miercoles”));
      v.addElement(new String(“Jueves”));
      v.addElement(new String(“Viernes”));
      v.addElement(new String(“Sábado”));
      v.addElement(new String(“Domingo”));
```

```
        sesion.setAttribute(“diasSemana”,v);
%>
```

5. Recuperar objetos de una sesión

Los datos que se guardan en la sesión permanecen ahí a la espera de ser utilizados. Para ello es necesario realizar el proceso contrario a cuando se graban, comenzando por la recuperación del objeto de la sesión para empezar a ser tratado.

Para poder realizar este paso se utiliza el método getAttribute() (anteriormente se utilizaba el método getValue(), pero este método se encuentra en desuso), utilizando como argumento el nombre que identifica al objeto que se quiere recuperar.

```
getAttribute(java.lang,String nombre)
```

Un ejemplo de recuperación de objetos almacenados en la sesión:

```
<%
        HttpSession sesion=request.getSession();
        Sesion.getAttribute(“nombre”);
%>
```

Cuando este método devuelve el objeto no establece en ningún momento de qué tipo de objeto se trata(String, Vector...)

Por ello si se conoce previamente el tipo de objeto que puede devolver tras ser recuperado de la sesión es necesario realizar un *casting*, para convertir el objeto de tipo genérico al objeto exacto que se va a usar. Para realizar esta operación se añade el tipo de objeto al lado de tipo HttpSession que utiliza el método getAttribute() para obtener el objeto que devuelve:

```
<%
        HttpSession sesion=request.getSession();
        String nombre=(String)sesion.getAttribute(“nombre”);
        out.println(“Contenido de nombre: “+nombre);
%>
```

Si no existe ningún objeto almacenado en la sesión bajo el identificador que se utiliza en el método getAttribute(), el valor devuelto será *null.* Por ello habrá que prestar especial atención ya que si se realiza el *casting* de un valor *null* el contenedor JSP devolverá un error. Lo mejor en estos casos es adelantarse a los posibles errores que pueda haber.

```
<%
        if(sesion.getAttribute(“nombre”)!=null)
        {
                String nombre=(String)sesion.getAttribute(“nombre”);
                out.println(“Contenido de nombre: “+nombre);
        }
%>
```

En el caso de tipos primitivos deben de ser convertidos a objetos previamente a su integración sesión de tal forma que su proceso de extracción viene a ser similar:

```
<%
      HttpSession sesion=request.getSession();
      Integer edad=(Integer)sesion.getAttribute(“edad”);
      out.println(“Edad: “+edad.intValue());
%>
```

En esta ocasión el objeto devuelto y convertido a Integer no es del todo válido ya que es necesario obtener de él el valor entero. Para ello se utiliza el método intValue() que devuelve el valor que realmente se había guardado previamente en la sesión.

Por último, el ejemplo del vector guardado en la sesión tiene un tratamiento similar al de los casos anteriores. El primer paso es recuperar el objeto de la sesión:

```
<%@page import=”java.util.*” session=”true” %>
<%
      HttpSession sesion=request.getSession();
      sesion.getAttribute(“diasSemana”);
%>
```

Como se sabe que el objeto es de tipo Vector se puede recuperar y convertir en un solo paso:

```
Vector v= (Vector) sesion.getAttribute(“diasSemana”);
```

A partir de este momento se puede acceder a los elementos del vector independientemente de si venía de una sesión o ha sido creado. Para ello se utiliza el método size() que devuelve el tamaño del vector para ir leyendo cada uno de sus elementos:

```
<%
      for(int i=0; i<v.size(); i++)
      {
            out.println(“<b>Dia: </b>”+(String)v.get(i)+”<br>”);
      }
%>
```

Se ha realizado otro proceso de conversión que sin ser necesario, ayuda a entender mejor el funcionamiento. Cuando se recupera un elemento de un vector(que se trata de un objeto) es necesario realizar el *casting* y convertirlo a su tipo de objeto definitivo. El resultado será el siguiente.

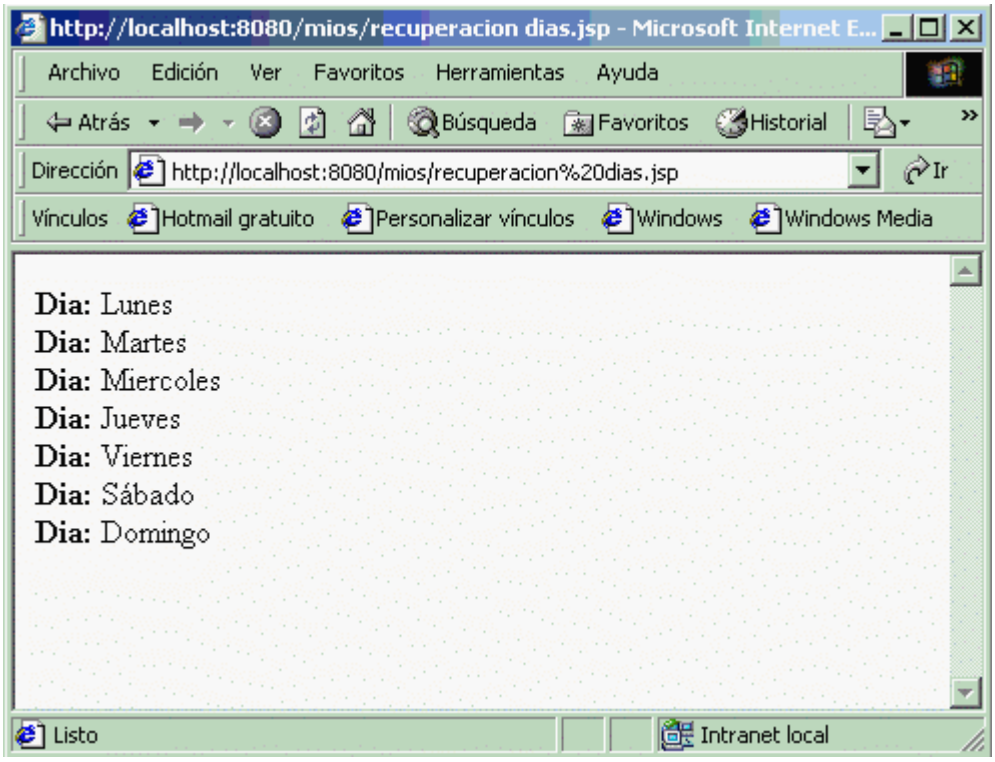

Figura 5. Para extraer los datos guardados primero se recupera el objeto Vector de la sesión y a continuación se recorren cada uno de sus elementos.

Para recuperar todos los objetos de una sesión se puede hacer uso también del método getAttributeNames() de la interfaz HttpSession. Para recoger todos los objetos almacenados en la sesión se recorre el objeto Enumeration que contiene el nombre de todos los objetos que contiene la sesión y que ha sido devuelto por el método getAttributeNames(). Cada nombre de atributo de la sesión se utiliza en la llamada a cada método getAttribute(), que devolverá el objeto correspondiente.

```
<%@page contentType="text/html; charset=iso-8859-1"
session="true" language="java" import="java.util.*" %>

<%
      HttpSession sesion=request.getSession();
      String nombre=”Práctica de POO”;
      Vector v=new Vector();

      v.addElement(new String("Lunes"));
      v.addElement(new String("Martes"));
      v.addElement(new String("Miercoles"));
      v.addElement(new String("Jueves"));
      v.addElement(new String("Viernes"));
      v.addElement(new String("Sábado"));
      v.addElement(new String("Domingo"));
      sesion.setAttribute("diasSemana",v);
```

```
        sesion.setAttribute("nombre",nombre);
%>
```

Con el siguiente código se recupera los objetos

```
<%@page import="java.util.*" session="true" %>
<%
        HttpSession sesion=request.getSession();
        Enumeration enum=sesion.getAttributeNames();
        String nombreAtributo;

        while (enum.hasMoreElements())
        {
                nombreAtributo=(String)enum.nextElement();
                out.println("<b>Atributo:</b>"+nombreAtributo);

                if(nombreAtributo.equals("diasSemana"))
                {
                        Vector v= (Vector)
                    sesion.getAttribute("diasSemana");

                    for(int i=0; i<v.size(); i++)
                    {
                            out.println("<br><b>Dia:
                          </b>"+(String)v.get(i));
                    }
                }
                else
                        out.println("<b>"+sesion.getAttribute(nombr
                    eAtributo)+"</b><BR><BR>");
        }
%>
```

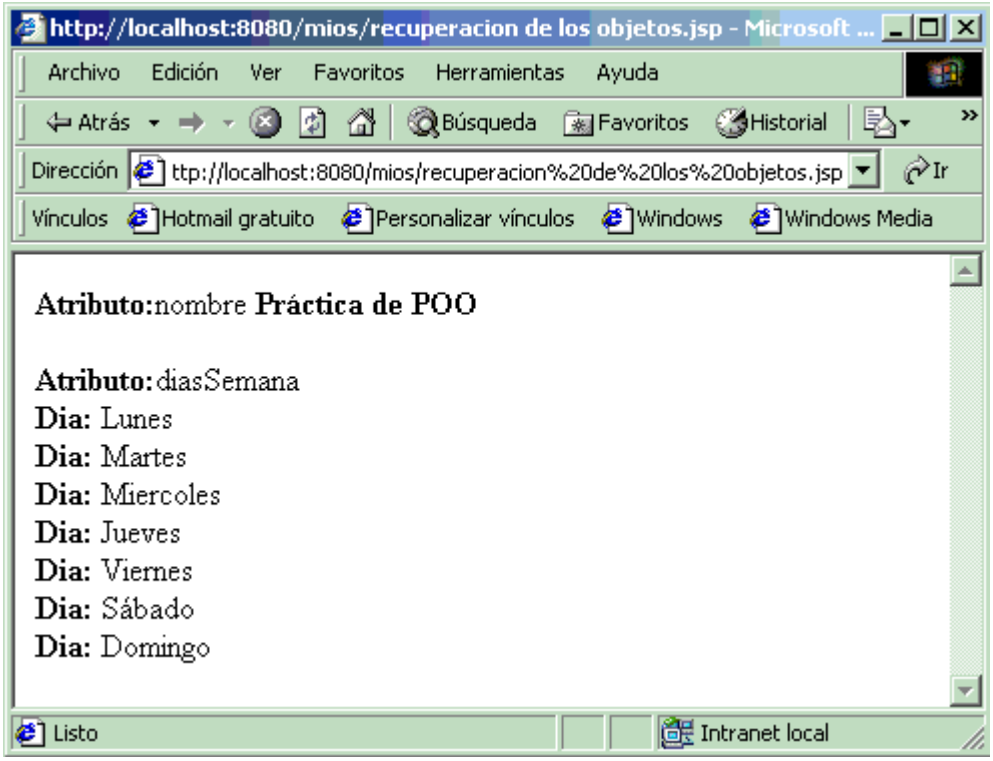

Figura 6. Para extraer varios objetos guardados en la sesión

6. Cómo se destruye una sesión

Como se ha visto, los datos almacenados por las sesiones pueden destruirse en tres casos:

- El usuario abandona aplicación web (cambia de web o cierra el navegador)
- Se alcanza el tiempo máximo permitido de inactividad de un usuario (*timeout*).
- El servidor se para o se reincia.

Pero la situación más probable es querer iniciar las sesiones o dar por finalizada una si se ha cumplido una o varias condiciones. En este caso no es necesario esperar a que ocurra alguno de los tres casos citados anteriormente, ya que mediante el método invalidate() es posible destruir una sesión concreta.

En el siguiente caso la sesión “sesión” se destruye al invocar el método invalidate(); y por la tanto el valor u objeto que está asociado a la misma.

```
<%
      [...]
      sesion.invalidate();
%>
```

7. Resumen de la interfaz HttpSession

Principales métodos:

• Object getAttribute(String nombreAtributo): devuelve el objeto almacenado en la sesión actual, y cuya referencia se corresponde con el nombre de atributo indicado por parámetro como un objeto de la clase String. Sea cual sea la clase del objeto almacenado en la sesión, este método siempre devolverá un objeto de la clase genérica Object, a la hora de recuperarlo se deberá realizar la transformación de clases correspondiente. Este método devuelve null si el objeto indicado no existe.

• Enumeration getAttributeNames(): este método devuelve en un objeto Enumeration del paquete java.util, que contiene 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, medido en milisegundos desde el 1 de enero de 1970.

• String getId(): devuelve una cadena que se corresponde con el identificador único asignado a la sesión. Luego se ve que 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 en milisegundos la fecha y hora de la última vez que el cliente realizó una petición asociada con la sesión actual.

• int getMaxInactiveInterval(): devuelve el máximo intervalo de tiempo, en segundos, en el que una sesión permanece activa entre dos peticiones distintas de un mismo cliente, es decir, es el tiempo de espera máximo en el que pertenece activa una sesión sin que el cliente realice ninguna petición relacionada con la sesión actual. El valor por defecto que puede permanecer inactiva unas sesión es de 30 segundos. Una vez transcurrido este tiempo el contenedor de *servlets* (*servlet container*) destruirá la sesión, liberando de la memoria todos los objetos que contiene la sesión que ha caducado.

• void invalidate(): este método destruye la sesión de forma explícita, y libera de memoria todos los objetos (atributos) que contiene la sesión.

• boolean isNew(): devuelve verdadero si la sesión se acaba de crear en la petición actual o el cliente todavía no ha aceptado la sesión (puede rechazar el *cookie* de inicio de sesión). Este método devolverá falso si la petición que ha realizado el cliente ya pertenece a la sesión actual, es decir, la sesión ya ha sido creada previamente,

• void removeAttribute(String nombreAtributo): elimina el objeto almacenado en la sesión cuyo nombre se pasa por parámetro. Si el nombre del objeto indicado no se corresponde con ninguno de los almacenados en la sesión, este método 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*.

8. Ejemplo práctico: Administración de usuarios.

Un caso práctico donde poder usar las sesiones es en las páginas a las que se debe acceder habiendo introducido previamente un usuario y una clave. Si no se introducen estos datos no se podrán visualizar y de igual modo si alguien intenta entrar directamente a una de estas páginas sin haberse identificado será redirigido a la página principal para que se identifique y, de este modo, no pueda acceder de forma anónima.

8.1. login.jsp

La primera página de la aplicación JSP es en la que el usuario se debe identificar con un nombre de usuario y una clave por lo que su aspecto será el de un formulario.

La página JSP contiene el formulario el cual especifica la página destino cuando el usuario pulse el botón de enviar los datos. Además se ha añadido una comprobación en la que en caso de recibir un parámetro llamado "error" se muestra el mensaje que contenga. De esta forma el usuario ve qué tipo de error se ha producido.

Figura 7. Para poder acceder al área reservada será necesario identificarse mediante un nombre de usuario y una contraseña.

```
   <%@page contentType="text/html; charset=iso-8859-1"
session="true" language="java" import="java.util.*" %>


<html>
<head><title>Proceso de login</title>
</head>


<body>
      <b>Proceso de identificación</B>
      <p>
<%
      if(request.getParameter("error")!=null)
```

```
        {
                out.println(request.getParameter("error"));
        }
%>
<form action="checklogin.jsp" methop="post">
usuario: <input type="text" name="usuario" size=20><br>
clave: <input type="text" name="clave" size=20><br>
<input type="submit" value="enivar"><br>
</form>
</body>
</html>
```

8.2. checklogin.jsp

Esta página es la encargada de recoger del usuario y la clave enviados desde el formulario. Una vez recibidos se almacenan en dos variables(“usuario” y “clave”) de tipo String. A continuación se comparan con los valores correctos del usuario y la clave.

Si esta comprobación es correcta se crea un objeto de tipo session y se guarda el valor en la variable “usuario” en la sesión mediante el método setAttribute().

A continuación y mediante la opción estándar <jsp: fordward> se redirecciona al usuario a la página final en la que se encuentra el menú de opciones al que se accede después de haber completado de forma satisfactoria el proceso de identificación.

En caso que la comprobación de usuario y clave no se cumpla se redirecciona al usuario hacia la página de inicio, para que vuelva a identificarse incluyendo esta vez un parámetro llamado “error” con un mensaje que avisará de qué es lo que le ha ocurrido.

```
<%@ page session="true" %>
<%
        String usuario = "";
        String clave = "";

        if (request.getParameter("usuario") != null)
                usuario = request.getParameter("usuario");

        if (request.getParameter("clave") != null)
                clave = request.getParameter("clave");

        if (usuario.equals("spiderman") &&
        clave.equals("librojsp")) {
                HttpSession sesionOk = request.getSession();
                sesionOk.setAttribute("usuario",usuario);
                %>
```

```
            <jsp:forward page="menu.jsp" />
            <%
      } else {
            %>
            <jsp:forward page="login.jsp">
            <jsp:param name="error" value="Usuario y/o clave
            incorrectos.<br>Vuelve a intentarlo."/>
            </jsp:forward>
            <%
      }
%>
```

8.3. menu.jsp

La página contiene la comprobación que se ha realizado el proceso de *login* previamente. En caso de no ser así, es decir, se ah entrado de forma “no correcta”, se redirige de nuevo al usuario a la página de login.jsp, para que se identifique. Para esta comprobación se recupera el valor que supuestamente ya estaría en la sesión. Si este valor es nulo, quiere decir que la sesión no se ha creado y por lo tanto es necesario que se identifique. Para esta comprobación se recupera el valor que supuestamente ya estaría en la sesión. Si este valor es nulo, quiere decir que la sesión no se ha creado y por lo tanto es necesario que se identifique. En esta ocasión el proceso para redireccionar al usuario a la página de login.jsp también lleva el parámetro “error” con un mensaje que será visto en la primera página.

Si por el contrario, la sesión sí ha sido creada, quiere decir que el usuario ha sido identificado de forma correcta, por lo que en la variable usuario creada previamente se almacena el nombre del usuario utilizado, para ser mostrado posteriormente.

```
<%@ page session="true" %>
<%
      String usuario = "";
      HttpSession sesionOk = request.getSession();
      if (sesionOk.getAttribute("usuario") == null) {
            %>
            <jsp:forward page="login.jsp">
                  <jsp:param name="error" value="Es
                  obligatorio identificarse"/>
            </jsp:forward>
            <%
      } else {
            usuario = (String)sesionOk.getAttribute("usuario");
      }
%>
<html>
```

```
<head><title>Proceso de login</title>
</head>
<body>
<b>PROCESO DE IDENTIFICACIÓN</b><p>
<b>Menú de administración</b><br>
<b>Usuario activo:</b> <%=usuario%><p>
<li> <a href="opc1.jsp">Crear nuevo usuario</a>
<li> <a href="opc2.jsp">Borrar un usuario</a>
<li> <a href="opc3.jsp">Cambiar clave</a>
<p>
<li> <a href="cerrarsesion.jsp">Cerrar sesión</a>
</body>
</html>
```

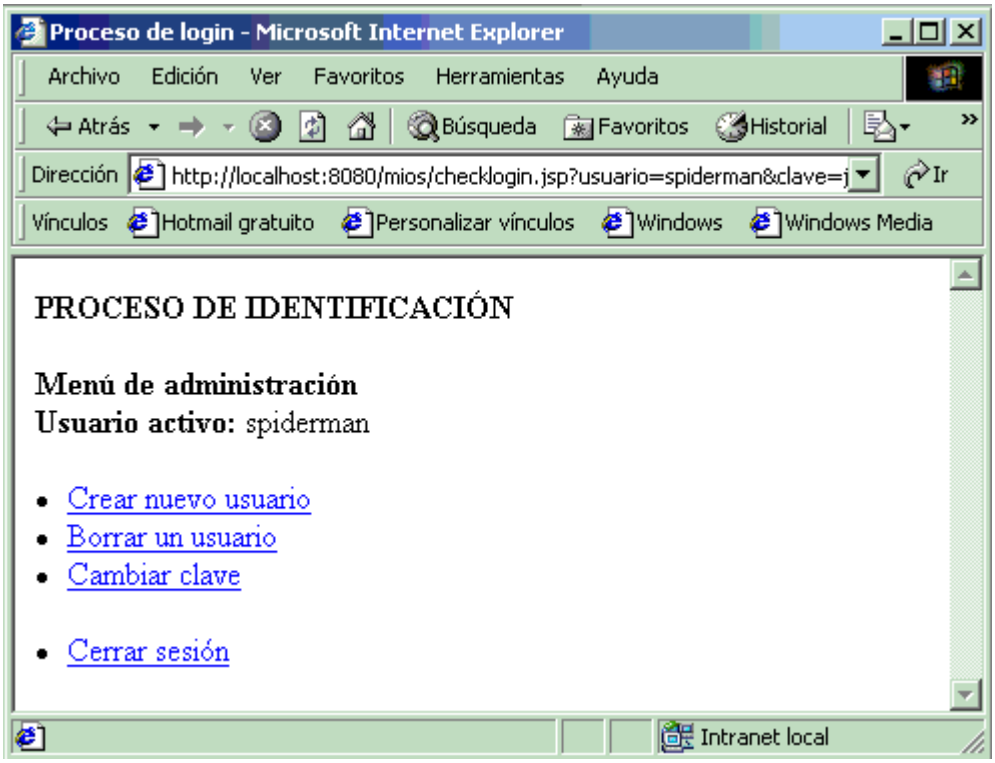

Figura 8. Una vez que el usuario se ha identificado correctamente puede acceder al menú.

En la siguiente figura se puede ver la pantalla cuando se intenta acceder a la página del menú sin haber realizado previamente el proceso de login:

Figura 9. Si el usuario intenta entrar directamente al menú, será redirigido a la página de identificación mostrando el mensaje de aviso.

8.4. cerrarsesion.jsp

La última opción que incorpora el menú es la de "Cerrar sesión", que será de gran utilidad cuando se haya finalizado el trabajo y queremos estar seguro que nadie realiza ninguna acción con nuestro usuario y clave.

Al pulsar este enlace, se recupera de nuevo la sesión y mediante el método invalidate() se da por finalizada la sesión.

```
<%@ page session="true" %>
<%
      HttpSession sesionOk = request.getSession();
      sesionOk.invalidate();
%>
<jsp:forward page="login.jsp"/>
```

9. Cookies

Las sesiones vistas anteriormente basan su funcionamiento en los *cookies*. Cuando se hace uso de la interfaz HttpSession de forma interna y totalmente transparente al programador se está haciendo uso de los *cookies*. De hecho cuando a través de una página JSP se comienza una sesión, se crea un *cookie* llamado JSSESIONID. La diferencia es que este *cookie* es temporal y durará el tiempo que permanezca el navegador ejecutándose, siendo borrada cuando el usuario cierre el navegador.

El objetivo de utilizar *cookies* es poder reconocer al usuario en el momento en el que se conecta al servidor. Una de las páginas que recoge la petición del usuario puede comprobar si existe un *cookie* que ha dejado anteriormente, si es así, sabe que ese usuario ya ha visitado ese *website* y por lo tanto puede leer valores que le identifiquen. Otro de los usos de los *cookies* es ofrecer una personalización al usuario. En muchos sitos web es posible elegir el color de fondo, el tipo de letra utilizado, etc... Estos valores pueden ser almacenados en *cookies* de forma que cuando acceda de nuevo al web y se compruebe la existencia de esos valores, serán recuperados para utilizarlos en la personalización de la página tal y como el usuario estableció en su momento.

Un ejemplo que se ha podido encontrar en muchas webs es en el momento de realizar un registro o solicitar el alta en un área restringida, ya que en muchas ocasiones existe un *checkbox* que cuando se selecciona permite recordar el nombre de usuario a falta de que sólo se escriba la clave.

Utilizando también el identificador idSession que se genera en un sesión como ya hemos visto y guardándolo en el *cookie* se pueden mostrar mensajes personalizados en el momento en el que el usuario acceda de nuevo al website.

Para trabajar con *cookies* se utiliza la clase Cookie que está disponible en paquete javax.servlet.http. Por medio de esta clase se pueden crear *cookies*, establecer sus valores y nombres, alguna de sus propiedades, eliminarlas, leer valores que almacenan, etc.

9.1. Crear un cookie

Un *cookie* almacenado en el ordenador de un usuario está compuesto por un nombre y un valor asociado al mismo. Además, asociada a este *cookie* pueden existir una serie de atributos que definen datos como su tiempo de vida, alcance, dominio, etc.

Cabe reseñar que los *cookies*, no son más que ficheros de texto, que no pueden superar un tamaño de 4Kb, además los navegadores tan sólo pueden aceptar 20 *cookies* de un mismo servidor web (300 *cookies* en total).

Para crear un objeto de tipo Cookie se utiliza el constructor de la clase Cookie que requiere su nombre y el valor a guardar. El siguiente ejemplo crearía un objeto Cookie que contiene el nombre “nombre” y el valor “objetos”.

```
<%
      Cookie miCookie=new Cookie(“nombre”,”objetos”);
%>
```

También es posible crear *cookies* con contenido que se genere de forma dinámica. El siguiente código muestra un *cookie* que guarda un texto que está concatenado a la fecha/hora en ese momento:

```
<%@page contentType="text/html; charset=iso-8859-1"
session="true" language="java" import="java.util.*" %>
<%
      Cookie miCookie=null;
      Date fecha=new Date();
      String texto= “Este es el texto que vamos a guardar en el
      cookie”+fecha;
      miCookie=new Cookie(“nombre”,texto);
```

```
%>
```

En esta ocasión el contenido del valor a guardar en el *cookie* está en la variable "texto". También se pueden guardar valores o datos que provengan de páginas anteriores y que hayan sido introducidas a través de un formulario:

```
<%
      Cookie miCookie=null;
      String ciudad= request.getParameter("formCiudad");
      miCookie=new Cookie("ciudadFavorita",ciudad);
%>
```

Una vez que se ha creado un *cookie*, es necesario establecer una serie de atributos para poder ser utilizado. El primero de esos atributos es el que se conoce como tiempo de vida.

Por defecto, cuando creamos un *cookie*, se mantiene mientras dura la ejecución del navegador. Si el usuario cierra el navegador, los *cookies* que no tengan establecido un tiempo de vida serán destruidos.

Por tanto, si se quiere que un *cookie* dure más tiempo y esté disponible para otras situaciones es necesario establecer un valor de tiempo (en segundos) que será la duración o tiempo de vida del *cookie*. Para establecer este atributo se utiliza el método setMaxAge(). El siguiente ejemplo establece un tiempo de 31 días de vida para el *cookie* "unCookie":

```
<%
      unCookie.setMaxAge(60*60*24*31);
%>
```

Si se utiliza un valor positivo, el *cookie* será destruido después de haber pasado ese tiempo, si el valor es negativo el *cookie* no será almacenado y se borrará cuando el usuario cierre el navegador. Por último si el valor que se establece como tiempo es cero, el *cookie* será borrado.

Otros de los atributos que se incluye cuando se crea un *cookie* es el *path* desde el que será visto, es decir, si el valor del *path* es "/" (raíz), quiere decir que en todo el *site* se podrá utilizar ese *cookie*, pero si el valor es "/datos" quiere decir que el valor del *cookie* sólo será visible dentro del directorio "datos". Este atributo se establece mediante el método setPath().

```
<%
      unCookie.setPath("/");
%>
```

Para conocer el valor de *path*, se puede utilizar el método getPath().

```
<%
      out.println("cookie visible en: "+unCookie.getPath());
%>
```

Existe un método dentro de la clase Cookie que permite establecer el dominio desde el cual se ha generado el *cookie*. Este método tiene su significado porque un navegador sólo envía al servidor los *cookies* que coinciden con el dominio del servidor que los envió. Si en alguna ocasión se requiere que estén disponibles desde otros subdominios se especifica con el método setDomain(). Por ejemplo, si existe el servidor web en la página www.paginasjsp.com , pero al mismo tiempo también existen otros subdominios como usuario1.paginasjsp.com, usuario2.paginasjsp.com, etc.

Si no se establece la propiedad *domain* se entiende que el *cookie* será visto sólo desde el dominio que lo creó, pero sin embardo si se especifica un nombre de dominio se entenderá que el *cookie* será visto en aquellos dominios que contengan el nombre especificado.

En el siguiente ejemplo hace que el *cookie* definido en el objeto "unCookie" esté disponible para todos los dominios que contengan el nombre ".paginasjsp.com". Un nombre de dominio debe comenzar por un punto.

```
<%
      unCookie.setDomain(".paginasjsp.com");
%>
```

Igualmente, para conocer el dominio sobre el que actúa el *cookie*, basta con utilizar el método getDomain() para obtener esa información.

Una vez que se ha creado el objeto Cookie, y se ha establecido todos los atributos necesarios es el momento de crear realmente, ya que hasta ahora sólo se tenía un objeto que representa ese *cookie*.

Para crear el fichero *cookie* real, se utiliza el método addCookie() de la interfaz HttpServletResponse:

```
<%
      response.addCookie(unCookie);
%>
```

Una vez ejecutada esta línea es cuando el *cookie* existe en el disco del cliente que ha accedido a la página JSP.

Es importante señalar que si no se ejecuta esta última línea el *cookie* no habrá sido grabado en el disco, y por lo tanto, cualquier aplicación o página que espere encontrar dicho *cookie* no lo encontrará.

9.2. Recuperar un cookie

El proceso de recuperar un *cookie* determinado puede parecer algo complejo, ya que no hay una forma de poder acceder a un *cookie* de forma directa. Por este motivo es necesario recoger todos los *cookies* que existen hasta ese momento e ir buscando aquél que se quiera, y que al menos, se conoce su nombre.

Para recoger todos los *cookies* que tenga el usuario guardados se crea un array de tipo Cookie, y se utiliza el método getCookies() de la interfaz HttpServletRequest para recuperarlos:

```
<%
      Cookie [] todosLosCookies=request.getCookies();
         /* El siguiente paso es crear un bucle que vaya leyendo
      todos los cookies. */
         for(int i=0;i<todosLosCookies.length;i++)
         {
            Cookie unCookie=todosLosCookies[i];

         /* A continuación se compara los nombres de cada uno de
      los cookies con el que se está buscando. Si se encuentra un
```

```
        cookie con ese nombre se ha dado con el que se está
        buscando, de forma que se sale del bucle mediante break. */
              if(unCookie.getName().equals("nombre"))
                    break;
        }
           /* Una vez localizado tan sólo queda utilizar los
        métodos apropiados para obtener la información necesaria
        que contiene. */
        out.println("Nombre: "+unCookie.getName()+"<BR>");
        out.println("Valor: "+unCookie.getValue()+"<BR>");
        out.println("Path: "+unCookie.getPath()+"<BR>");
        out.println("Tiempo de vida:"+unCookie.getMaxAge()+"<BR>");
        out.println("Dominio: "+unCookie.getDomain()+"<BR>");
  %>
```

9.3. Utilizar los cookies

Para realizar un ejemplo práctico se va a seguir con el ejemplo de Sesiones. El objetivo será modificar las páginas necesarias para que si el usuario selecciona un campo de tipo *checkbox* (que será necesario añadir) el nombre de usuario le aparezca por defecto cuando vuelva a entrar a esa página. Este nombre de usuario estará guardado en un *cookie* en su ordenador.

El primer paso es añadir el *checkbox* en la página login.jsp:

```
<%@ page session="true" import="java.util.*"%>
<%
      String usuario = "";
      String fechaUltimoAcceso = "";


         /*Búsqueda del posible cookie si existe para recuperar
      su valor y ser mostrado en el campo usuario */
      Cookie[] todosLosCookies = request.getCookies();
      for (int i=0; i<todosLosCookies.length; i++) {
            Cookie unCookie = todosLosCookies[i];
            if (unCookie.getName().equals("cokieUsu")) {
                  usuario = unCookie.getValue();
            }
      }
      /* Para mostrar la fecha del último acceso a la página.
Para ver si el cookie que almacena la fecha existe, se busca en los
cookies existentes. */
      for (int i=0; i<todosLosCookies.length; i++) {
            Cookie unCookie = todosLosCookies[i];
            if (unCookie.getName().equals("ultimoAcceso")) {
```

```
                    fechaUltimoAcceso = unCookie.getValue();
              }
        }
        /* Se comprueba que la variable es igual a vacío, es decir
        no hay ningún cookie llamado “ultimoAcceso“, por lo que se
        recupera la fecha, y se guarda en un nuevo cookie. */
        if (fechaUltimoAcceso.equals(""))
        {
              Date fechaActual = new Date();
              fechaUltimoAcceso = fechaActual.toString();
                 Cookie cookieFecha = new
              Cookie("ultimoAcceso",fechaUltimoAcceso);
              cookieFecha.setPath("/");
              cookieFecha.setMaxAge(60*60*24);
              response.addCookie(cookieFecha);
        }
  %>
  <html>
  <head><title>Proceso de login</title>
  </head>
  <body>
  <b>PROCESO DE IDENTIFICACIÓN</b>
     <br>Última vez que accedió a esta
  página:<br><%=fechaUltimoAcceso%>
  <p>
  <%
        if (request.getParameter("error") != null) {
              out.println(request.getParameter("error"));
        }
  %>
  <form action="checklogin.jsp" method="post">
     usuario: <input type="text" name="usuario" size="20"
  value="<%=usuario%>"><br>
  clave: <input type="password" name="clave" size="20"><br>
     Recordar mi usuario: <input type="checkbox"
  name="recordarUsuario" value="on"><br>
  <input type="submit" value="enviar">
  </form>
  </body>
  </html>
```

Teniendo un aspecto final similar a la siguiente figura.

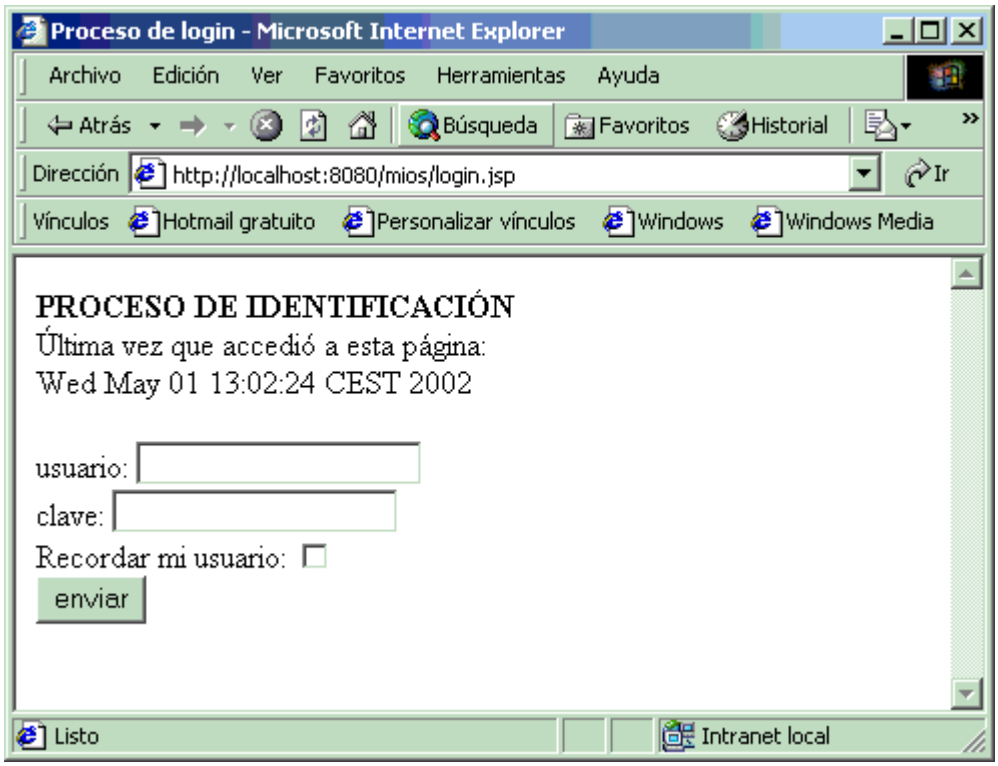

Figura 10. El formulario de identificación contiene ahora un checkbox para que se puede elegir guardar o no el nombre del usuario

El siguiente paso es modificar la página checklogin.jsp que recoge el usuario y clave introducidos y por lo tanto ahora también la nueva opción de "Recordar mi usuario". Dentro de la condición que se cumple si el usuario y la clave son correctos, y después de crear la sesión, escribimos el código que creará el *cookie* con el usuario. El primer paso es comprobar que el usuario ha activado esta opción, es decir, ha seleccionado el *checkbox*. También se realiza la comprobación de que el campo "recordarUsuario" no llegue con el valor nulo y produzca un error en la aplicación, en caso de que el usuario deje sin seleccionar el *checkbox*:

```
<%@ page session="true" import="java.util.*"%>
<%
      String usuario = "";
      String clave = "";

      if (request.getParameter("usuario") != null)
            usuario = request.getParameter("usuario");

      if (request.getParameter("clave") != null)
            clave = request.getParameter("clave");

      if (usuario.equals("spiderman") &&
      clave.equals("librojsp")) {
```

```
                out.println("checkbox: " +
                request.getParameter("recordarUsuario") + "<br>");
            HttpSession sesionOk = request.getSession();
            sesionOk.setAttribute("usuario",usuario);


            if ((request.getParameter("recordarUsuario") != null) &&
            (request.getParameter("recordarUsuario").equals("on")))
            {
                out.println("entra");
                   Cookie cookieUsuario = new Cookie
                ("cokieUsu",usuario);
                cookieUsuario.setPath("/");
                cookieUsuario.setMaxAge(60*60*24);
                response.addCookie(cookieUsuario);
            }
            /* Se realiza un proceso similar a la creación de cookie de
recordar el usuario. En este caso se trata de crear un nuevo cookie
con el nuevo valor de la fecha y guardarlo con el mismo nombre. De
esta forma será borrado el anterior y prevalecerá el valor del último.
*/
            Date fechaActual = new Date();
            String fechaUltimoAcceso = fechaActual.toString();
               Cookie cookieFecha = new
            Cookie("ultimoAcceso",fechaUltimoAcceso);
            cookieFecha.setPath("/");
            cookieFecha.setMaxAge(60*60*24);
            response.addCookie(cookieFecha);
      %>
            <jsp:forward page="menu.jsp" />
            <%
                } else {
            %>
      <jsp:forward page="login.jsp">
          <jsp:param name="error" value="Usuario y/o clave
      incorrectos.<br>Vuelve a intentarlo."/>
      </jsp:forward>
      <%
            }
      %>
```

Señalar que los *cookies* están envueltos en una mala fama relacionada con la intromisión de la privacidad de los usuarios por parte de determinados *websites* e incluso con la seguridad. Se

trata de pequeños ficheros de texto que poco daño pueden causar, y que como es lógico no pueden contener ningún código ejecutable.

Desde el punto de vista es cierto que determinados *sites* pueden saber que clase de palabras se consultan en un buscador o cual es nuestra frecuencia de visita a un determinado web. En cualquier caso son datos que no descubren la verdadera identidad del usuario, aunque si bien por este motivo, muchos de ellos deciden desactivar la recepción de *cookies*. Si esto es así las páginas en las que se utilicen *cookies* no funcionarán de forma correcta al no permitir ser recibidas por los navegadores.

10. Conclusión

Con JSP el manejo de sesiones permite controlar toda la información que se intercambia con el usuario desde el momento en el que se conecta al website, independientemente de que cambie de página dentro del web ya que permite almacenar información y recuperarla en cualquier momento de forma transparente para el usuario. Así mismo el uso de *cookies*, aunque controvertido, permite proveer un servicio personalizado para cada usuario almacenando la información que lo caracteriza en su propio ordenador.

11. Bibliografía

Esteban Trigos García. *"JSP"*. ANAYA MULTIMEDIA. 2001.

Ángel Esteban. "*Tecnología de servidor con Java: Servlets, JavaBeans, JSP"*. Grupo EIDOS. 2000.

Miguel Angel García. *"Tutorial de JavaSever Pages"*. www.javahispano.com