Tema 6. Sesiones

Concepto de sesión

Una sesión es una herramienta utilizada por programas hospedados y ejecutados en un servidor (servlets y jsps, en nuestro caso) para "estar al tanto" o "rastrear" las acciones que llevan a cabo los clientes que realizan una petición a alguno de esos programas.

Las sesiones nacen debido a la imposibilidad de un servidor http para:

- Detectar el cliente que realiza una petición.
- Almacenar y mantener datos asociados a la petición de un cliente, entre peticiones de ese mismo cliente. Es una consecuencia de lo anterior.

Lo que ocurre entre un cliente y un servidor que hospeda una aplicación web, a grandes rasgos sería algo así: el cliente realiza una petición, el servidor detecta que está dirigida a un servlet y delega su procesamiento al mismo. A continuación, el servlet genera una respuesta adecuada y es enviada al cliente. Fin de la historia: el servidor no guarda información de la interacción con el cliente ni almacena ningún tipo de dato relacionado con la petición. Si ese mismo cliente realizara otra petición al mismo servlet, para el servidor sería como un cliente nuevo.

Este comportamiento se debe al hecho de que el protocolo http empleado para la comunicación cliente-servidor, es un protocolo "sin estado" o "stateless" protocol.

NOTA IMPORTANTE: cuando se trabaja con sesiones, a los clientes, en el momento en que realizan la primera petición, el servidor les asocia un identificador (ID) distinto para cada uno, que le permite reconocerlos en posteriores peticiones y mantener la información que cada uno haya podido generar durante su interacción con la aplicación. Los mecanismos más empleados para asociar este ID son:

- **Cookies**
- **Reescritura de URLs**

Estos dos mecanismos se estudiarán a lo largo del tema.

El proceso que se realiza cuando se trabaja con sesiones, a grandes rasgos, puede resumirse como sigue:

- Un cliente realiza una primera petición a un servlet que contiene código para crear una sesión. Al ser la primera petición, el cliente no envía ninguna información sobre el ID o identificador de sesión.
- El servidor, a través del servlet, crea una sesión y le asigna un número de identificación ID . La sesión se considera nueva: esto significa que al aplicarle el método isNew(), devuelve true. El servidor envía la respuesta generada por el servlet y, además, el ID de sesión. Este ID suele enviarse en el encabezado de respuesta Set-Cookie o agregándolo al final del url de la petición (Reescritura de URL).
- El cliente captura el ID de sesión y lo guarda para futuras peticiones. Esto se hace para cada cliente que realiza una petición.

- El cliente decide realizar otra petición. Ahora, con esa petición viaja el ID de sesión asignado por el servidor en la primera, a través del encabezado de petición Cookie.
- El servidor recibe la petición, analiza el ID de sesión y, automáticamente, liga la petición con la sesión creada anteriormente. A partir de ese momento el cliente se considera unido a la sesión. Si se aplica el método isNew() a la sesión devolverá false ya que no se ha creado una nueva sesión, sino que se ha asignado la creada en base a la primera petición, a la segunda y posteriores peticiones.

Hay muchas aplicaciones en las que se trabaja con sesiones. Por ejemplo, las aplicaciones de comercio electrónico en las que cada cliente tiene asignado un carrito de compra donde almacena sus productos. Si no se trabajara con sesiones, a medida que fuera comprando un producto, perdería el anterior.

Se van a mostrar dos ejemplos en los que se utilizan sesiones:

- **Ejemplo 1)** Tienda de comercio electrónico

  Imaginemos que un cliente accede a una aplicación cuyo objetivo es vender a través de Internet libros de todo tipo: dispone de varias secciones como Lenguajes de programación, Internet, Filosofía, Novela romántica, etc. Al usuario le interesa adquirir una novela romántica y pulsa un botón que le lleva a otra página almacenada en el mismo servidor, con un listado de todas las novelas románticas a la venta. Se decide a comprar una y para ello cuenta con un botón "Comprar" que apunta a un servlet. El servlet está programado para agregar a un carrito de la compra virtual todo lo que el usuario compra y mostrar su contenido en una agradable interface gráfica. El usuario quiere comprar ahora un libro de Java y pulsa en un enlace que le lleva a la sección Lenguajes de programación, disponible en la página desde la que ha comprado la novela. Pues bien, cuando el usuario visualiza el listado de los libros de programación y compra uno de Java que le ha parecido interesante, observa que en su carrito de compra virtual no aparece la novela romántica. En el momento en que se ha accedido a la página de Lenguajes de programación se le ha asignado otro carrito de compra nuevo sin capacidad para mantener lo comprado en otras secciones.

  Esto es lo que ocurre si no se emplean sesiones, y se debe a que el protocolo http no es capaz de acumular la información de todas las acciones realizadas por el cliente cuando éste va a otras páginas o incluso cuando va a secciones distintas de una misma página.

- **Ejemplo 2)** Registro basado en múltiples formularios

  Otro ejemplo del uso de sesiones son los procesos de registro basados en varios formularios no contenidos en una misma página html. Imaginemos que el proceso de registro se estructura en dos formularios con dos páginas html y que los datos se insertarán en una BD (base de datos) previa validación. Podría hacerse validando cada grupo de datos por separado e insertarlos en la BD en dos veces. En este caso, se realizarían dos conexiones a la BD por cada petición y no se emplearían sesiones. Es preferible optar por una validación conjunta de todos los datos y su inserción posterior en la BD empleando una sola conexión por petición. En este caso, sí sería preciso trabajar con sesiones.

Ejemplo: ir a www.ya.com, pulsar el enlace situado en la parte izda superior con el nombre Tarifa plana y acceder a un producto cualquiera de la gama. Por ejemplo, Tarifa plana 24h. Luego, botón Date de Alta.

En otras muchas aplicaciones, no interesa trabajar con sesiones. Por ejemplo, para acceder al catálogo de libros de programación de una librería.

Interface HttpSession

Esta interface modela a una sesión. Una sesión, como se ha comentado antes, permite a un servlet identificar a cada cliente y almacenar información persistente asociada a cada petición. Dicha información se mantiene aun cuando un cliente abandone la aplicación o cambie de sección. Como se verá a lo largo de la lección, se suele definir un tiempo límite de mantenimiento de sesión, de modo que cuando se supere, se elimina automáticamente la misma y todo lo que almacena. Forma parte del paquete javax.servlet.http

Métodos

- **HttpSession getSession()**: este método pertenece a la interface HttpServletRequest y devuelve la sesión actual asociada a una petición de un cliente. Si no hay ninguna sesión ligada con la petición, se crea una nueva.

  Hay un método getSession sobrecargado que admite un booleano como argumento.

  El que se acaba de explicar es equivalente a getSession(true). El getSession(false) se comporta de igual forma que los anteriores, pero devolviendo null en caso de que no haya sesión asociada a la petición, es decir, no crea una nueva si no hay sesión ligada con la petición.

- **void setAttribute(String clave, Object valor)**: se emplea para almacenar atributos en la sesión. Se estructuran en forma de pares de datos clave/valor al estilo de los Hashtable. Establece una relación entre la clave y el valor.
- **Object getAttribute(String clave)**: devuelve el valor del atributo asociado a la clave. Suele ser habitual recuperarlo en el tipo de variable con la que fue guardado en la sesión. Para ello, se empleará casting. Si la clave que se le pasa no coincide con ninguna de las existentes devuelve null.
- **void removeAttribute(String clave)**: elimina un atributo en base a su clave.
- **String getId()**: devuelve una String asociada al número de identificación único o ID de la sesión. Cada sesión está asociada a un cliente y tiene un ID de sesión distinto.

NOTA: este número de identificación o ID de sesión es creado por el contenedor web, varía en función del servidor utilizado, y en la mayoría de los casos, es enviado al cliente mediante el encabezado de respuesta Set-Cookie.
Por tanto, para que el cliente soporte sesiones debe permitir el envío de cookies. ¿Qué ocurre si mantiene bloqueado dicho envío?

Puede utilizarse el mecanismo alternativo de trabajo con sesiones llamado Reescritura de URL o URL Rewriting. Se estudiará más adelante. Las cookies se verán en profundidad en el siguiente tema.

- **void setMaxInactiveInterval(int segundos)**: establece el tiempo, expresado en segundos, durante el que la sesión se mantendrá activa entre dos peticiones de cliente, antes de destruirse.

  La idea es que transcurrido un tiempo de inactividad de sesión, ésta se destruye o invalida. Se pierde todo lo que almacena. No tiene sentido mantener la sesión a un cliente que no hace nada o se va a otra página por tiempo indefinido. No conviene asignar demasiado tiempo, dado que los objetos HttpSession consumen memoria. Un valor habitual es 30 minutos. En las webs de tiendas on-line o en aquellas que ofrecen cuentas de correo gratuitas como Hotmail, Yahoo, etc. su valor por defecto suele ser 1800 segundos (media hora).

  Si se desea configurar este valor en el descriptor,

```
<session-config>
    <session-timeout>Tiempo en minutos (debe ser entero)</session-timeout>
</session-config>
```

  Ojo porque este elemento afecta a todas las sesiones de la aplicación mientras que el anterior a aquellas sobre las que se aplica el método.

  Debe ir **después de los elementos servlet y servlet-mapping.** Un número negativo indica que la sesión no tiene timeout, es decir que las sesiones existirán durante la vida de la aplicación.

- **long getCreationTime()**: devuelve los milisegundos transcurridos desde el epoch hasta la fecha en la que se ejecuta por primera vez el código asociado a la creación de la sesión, es decir, cuando se ejecute "HttpSession sesion=request.getSession()". Utilizado para conocer la fecha de creación de la sesión sobre la que se aplica.
- **long getLastAccessedTime()**: devuelve los milisegundos transcurridos desde el epoch hasta la fecha en la que se ejecutó por última vez el código asociado a la creación de la sesión, pero siendo vieja. Se emplea, junto con el anterior y siempre que no se haya sobrepasado el tiempo máximo de inactividad de sesión, para conocer el tiempo que tarda un usuario en volver a usar la sesión.
- **boolean isNew()**: devuelve true si la sesión es nueva o false si no.
- **void invalidate()**: destruye una sesión y, por tanto, todos sus atributos. Una sesión, además de con una llamada a este método, se destruye cuando se supera un determinado tiempo

entre peticiones realizadas por el usuario fijado con el método “void setMaxInactiveInterval(int segundos)”.

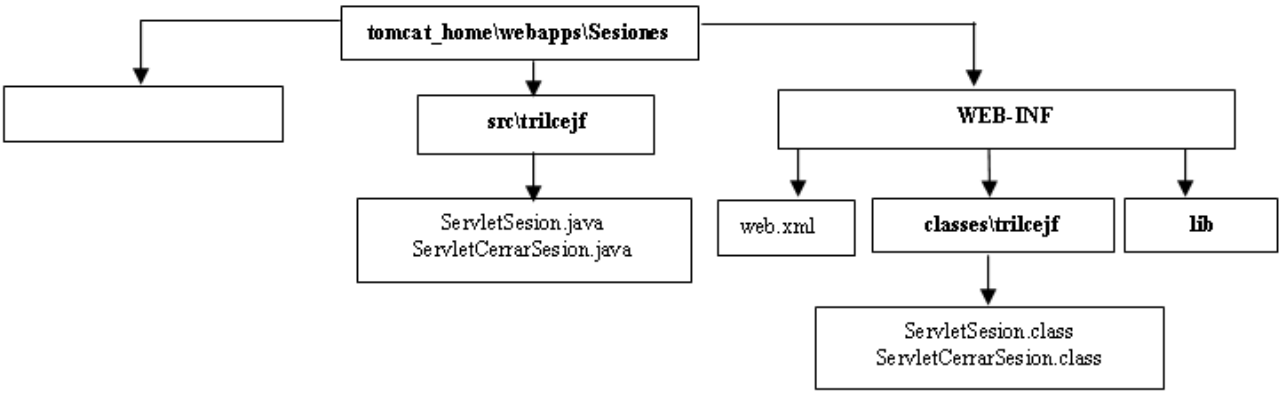
Ejemplo básico

Crear una aplicación web cuyo directorio raíz sea Sesiones. Contendrá un servlet de nombre **ServletSesion** que mostrará a los clientes información de la sesión asociada a cada petición y contabilizará el número de accesos realizados durante el tiempo de vida de la sesión.

Se mostrará información de la fecha y hora en la que se inicia la sesión así como de la última fecha y hora en la que produjo el último acceso a la misma.

Este servlet dispondrá de un botón que permitirá cerrar la sesión actual, accediendo a otro servlet de nombre **ServletCerrarSesion**, que contará con un enlace al servlet original.

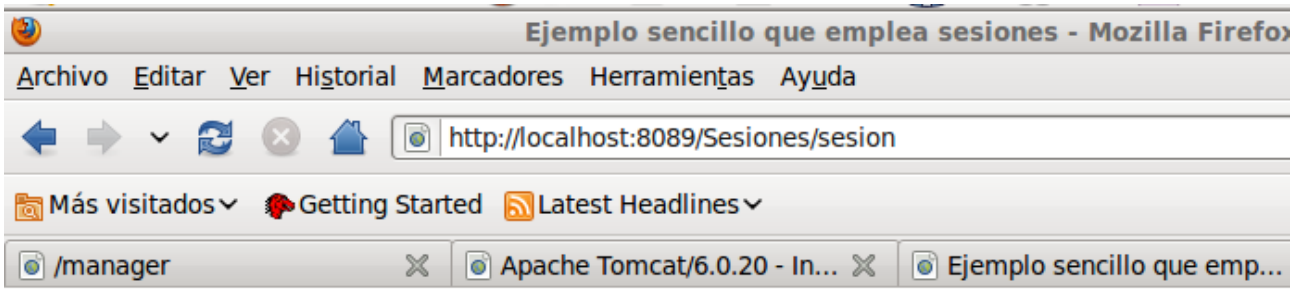
Estructura de directorios:



Descargar el war: Sesiones.war

Inicio de aplicación: http://localhost:8089/Sesiones/sesion

La primera vez que un cliente accede al servlet se mostrará:



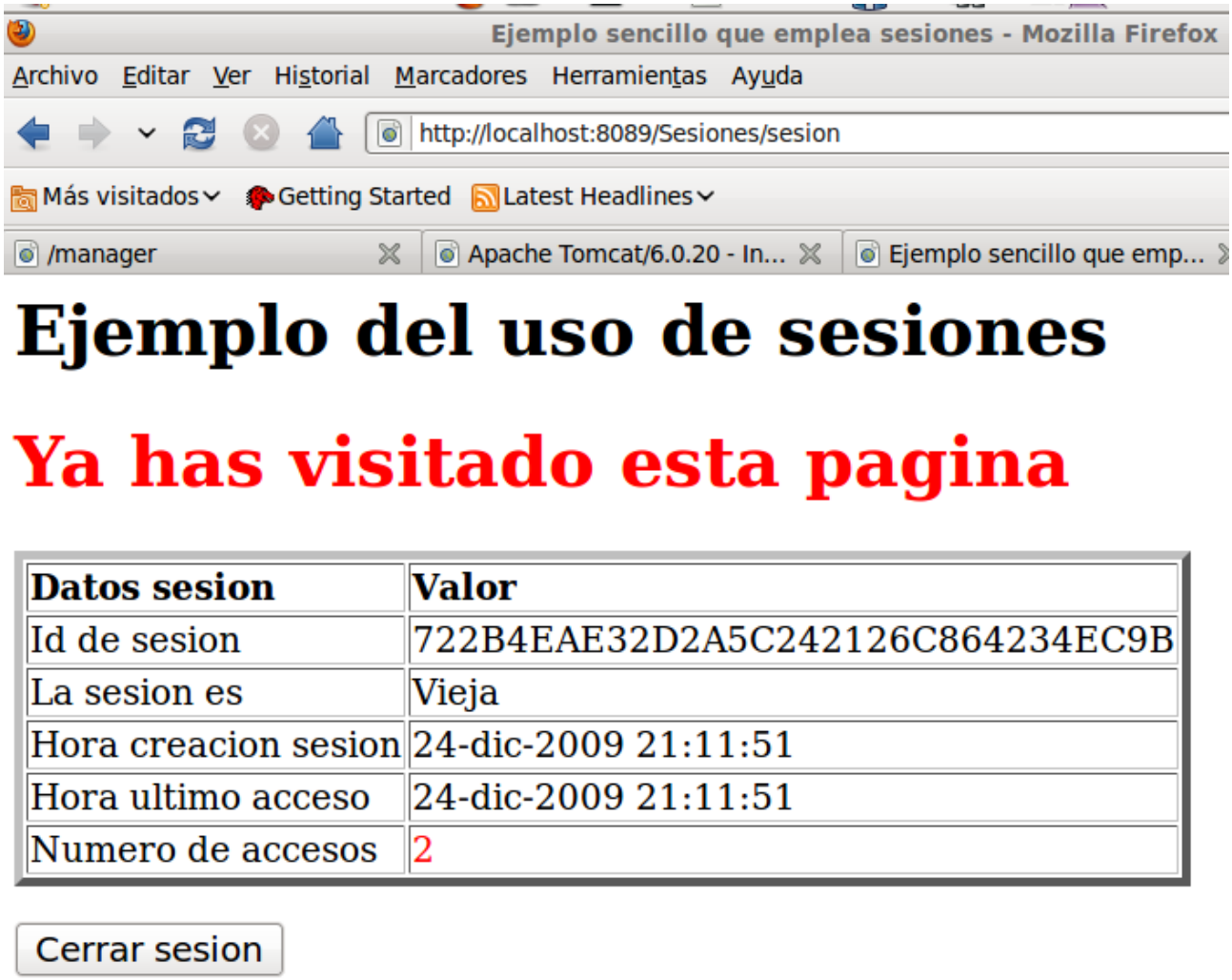
Ejemplo del uso de sesiones

Esta es tu primera visita

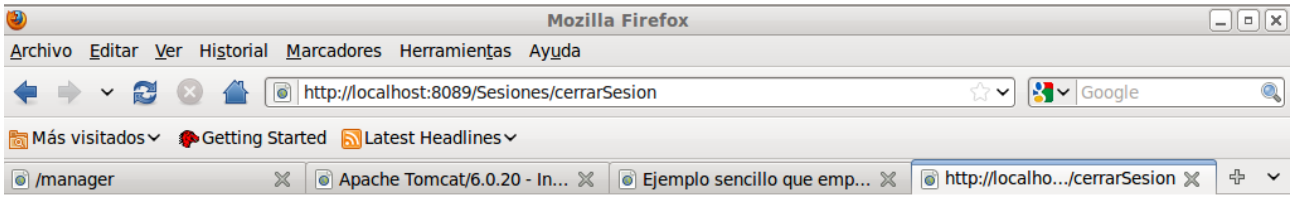
Datos sesion	Valor
Id de sesion	722B4EAE32D2A5C242126C864234EC9B
La sesion es	Nueva
Hora creacion sesion	24-dic-2009 21:11:51
Hora ultimo acceso	24-dic-2009 21:11:51
Numero de accesos	1

Cerrar sesion

El resto de veces que acceda ese cliente bien por pulsar Actualizar, bien porque abra otra ventana del navegador, se mostrará:



Al cerrar la sesión



La sesión 722B4EAE32D2A5C242126C864234EC9B se ha cerrado

Inicio

NOTA: si se bloquean las cookies en el navegador del cliente qué ocurre?
Se accede a la primera pantalla (la de "Esta es su primera visita"), pero cuando se actualiza no se mantiene la sesión y no se visualiza la segunda sino que se crea otra sesión nueva.

Si se accede localmente a la aplicación se usará

http://localhost:8080/Sesiones/sesion

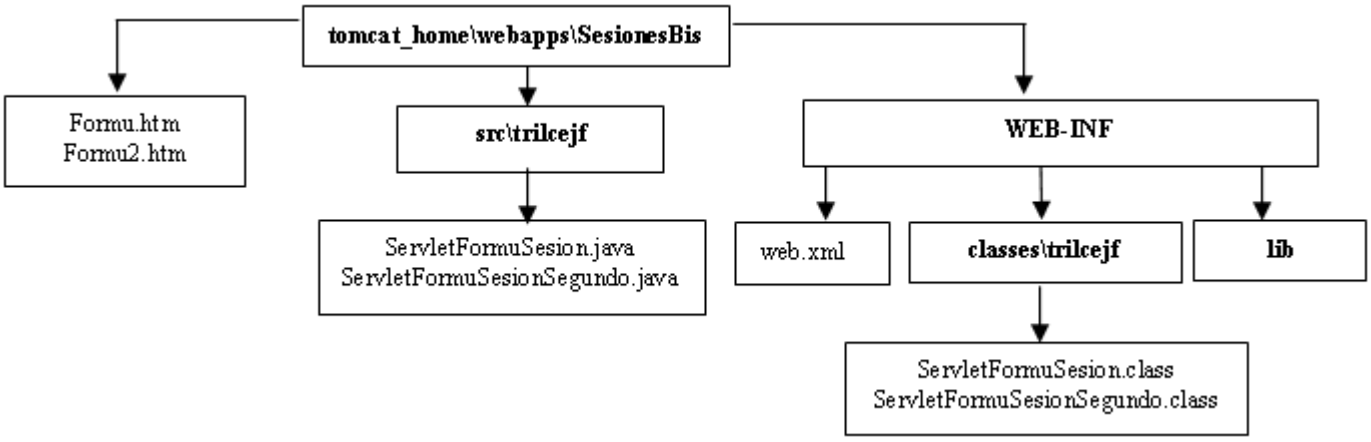
En este caso, el otro mecanismo para trabajar con sesiones, es decir, la reescritura de URL no serviría pues sólo funciona con enlaces html y con el atributo action de formularios, y como en la aplicación anterior no hay, no puede utilizarse. En el siguiente ejemplo se muestra cómo trabajar con sesiones mediante reescritura de URL

Reescritura de URL

Primero se trabajará con soporte para cookies y, a continuación, se eliminará dicho soporte y se observará cómo cambia el funcionamiento.

Para solventar los problemas, se realizarán los cambios pertinentes usando reescritura de URL, con el fin de que la aplicación siga funcionado del mismo modo que antes.

Estructura de directorios de la aplicación cuyo directorio raíz es SesionesBis:



Inicio de aplicación: http://127.0.0.1:8089/SesionesBis

Descarga del war: SesionesBis.war

La primera vez que un cliente accede al servlet se mostrará:

Página primera del formulario

NOMBRE:

APELLIDOS:

Enviar

Después de rellenar los cuadros de texto y pulsar Enviar

Página segunda del formulario

EMAIL

Enviar

Tras rellenar el email

NOMBRE: Juliana

APELLIDOS: Garcia Perez

EMAIL: juliana@gmail.com

NOTA: una vez ejecutada la aplicación, se bloquearán las cookies del cliente y se intentará ejecutar de nuevo (http://localhost:8089/SesionesBis)

¿Cuál va a ser el resultado?

```
NOMBRE: null
APELLIDOS: null
EMAIL: juliana@gmail.com
```

¿Por qué?

Porque no se mantiene la sesión. La línea de trilcejf.ServletFormuSesionSegundo

```
HttpSession sesion=request.getSession();
```

no captura la sesión asociada a la petición sino que crea una nueva. Por eso, al intentar obtener los datos mediante

```
String nombre=(String)sesion.getAttribute("nombre");
String apellidos=(String)sesion.getAttribute("apellidos");
```

devuelven null.

¿Cómo se soluciona?

Mediante **reescritura de URL**: se utiliza cuando el cliente no soporta cookies. **Se basa en agregar el ID de sesión a todos los URLs de la aplicación (enlaces y atributos action de formularios)**. De este modo, cuando el usuario pulse sobre un enlace o un botón de envío de formulario, la información de la ID de sesión se enviará automáticamente.

¿Cómo se hace?

Mediante el método de javax.servlet.http.HttpServletResponse

String encodeURL(String url): devuelve el URL que se le pasa, pero con la información de ID de sesión

Ejemplo:

http://127.0.0.1:8089/SesionesBisSinCookies/sesion2;jsessionid=7B4EFC378649C8C72D74A933F5D56F2E?email=ewew

Este método sólo actúa cuando el cliente no soporta cookies.

En el caso particular de esta aplicación, debería sustituirse Formu2.html por el siguiente servlet

```java
package trilcejf;

import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class ServletFormu2 extends HttpServlet{
    public void doGet(HttpServletRequest request,HttpServletResponse response)
        throws ServletException,IOException{

        response.setContentType("text/html");

        PrintWriter out=response.getWriter();

        out.println("<html>");
        out.println("<head><title></title></head>");
        out.println("<body>");
        out.println("<h1>Página segunda del formulario</h1>");

        //String action="/SesionesBisSinCookies/sesion2";
        //La línea de código siguiente es la clave ya que agrega al final del url de la petición
        //el ID de sesión. Si se comentara esta línea y se descomentara la del principio del
        //comentario, la aplicación no funcionaría según lo esperado.
        String action=response.encodeURL("/SesionesBisSinCookies/sesion2");
        System.out.println(action);
        out.println("<form method=\"get\" action="+action+">");
        out.println("<b>EMAIL</b><input type=\"text\" name=\"email\" size=\"25\"><p>");
        out.println("<input type=\"submit\" value=\"Enviar\">");
        out.println("</body>");
        out.println("</html>");
    }
}
```

Además, deberían modificarse las líneas de trilcejf.ServletFormuSesionSegundo

```java
if(email==null || email.length()==0){
    getServletContext().getRequestDispatcher("/Formu2.html").include(request,response);
}
```

y el descriptor de despliegue.

Se proporciona la aplicación SesionesBisSinCookies.war para comprobar todo lo comentado anteriormente.

Descarga del war: SesionesBisSinCookies.war

Desventajas de la reescritura de URL a la hora de trabajar con sesiones:

- Debe utilizarse el método encodeURL(..) para todos los enlaces y atributos action de los componentes de la aplicación que intervengan en la sesión.
- Todas las páginas html de la aplicación tienen que convertirse en servlets o JSPs para poder aplicar a los enlaces y a los action de formularios, el método anterior.