

■ Tema 4: Aplicaciones web con Java (II)

■ Más sobre el API

Mantenimiento de sesiones

Seguridad



Tecnologías  Web

■ 1. Más sobre el API de JSP



Tecnologías  Web

Incluir ficheros en un JSP

- Se puede incluir un fichero cualquiera o un JSP dentro de otro JSP

```
<jsp:include page="fichero" />
```

- Cuando se incluye un JSP dentro de otro, se le pueden pasar parámetros, como si vinieran de la petición HTTP (el JSP incluido recibe además todos los que recibe el JSP "padre")

```
<jsp:include page="pie.jsp">
```

```
<jsp:param name="color" value="rojo"/>
```

```
</jsp:include>
```



Tratamiento de errores

- Para indicar que cuando se produzca una excepción se debe saltar a una determinada página:

```
<%@ page errorPage="pagina-de-error" %>
```

- En la página a la que se salta hay que poner

```
<%@ page isErrorPage="true" %>
```

- En dicha página tenemos el objeto predefinido **exception** que contiene la excepción producida

```
<%= exception %> //imprime tipo de excepción y mensaje
```

```
<%= exception.getMessage() %> //imprime solo mensaje
```



2. Mantenimiento de sesiones

Información permanente entre peticiones HTTP



Sesiones

- Almacena variables propias de cada usuario en el servidor
- Parecidas a las *cookies*, pero:
 - Se almacenan en el servidor, no en el cliente
 - Cualquier tipo de datos (no sólo cadenas)
 - No hay límite de tamaño (una *cookie* como máx. 4K)
- Objeto implícito `session`: en él se pueden meter objetos Java asignándoles un nombre y recuperarlos con ese nombre
 - `session.setAttribute(nombre,objeto)`
 - `session.getAttribute(nombre)`



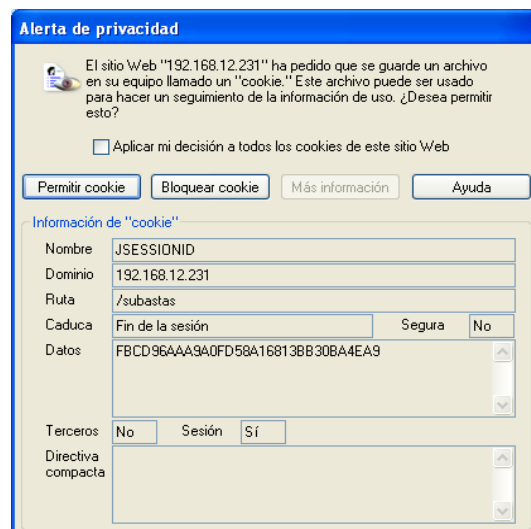
Sesiones (II)

- Los objetos guardados en la sesión se mantienen mientras que el cliente siga navegando por el sitio web
- ¿Cómo saber si sigue navegando por el sitio? (en HTTP no hay conexión permanente)
- Se define un intervalo de tiempo, transcurrido el cual, si el cliente no ha hecho ninguna petición HTTP se considera que se ha “desconectado”
 - `session.getMaxInactiveInterval()` devuelve dicho intervalo en segundos (en Tomcat por defecto es 1800 = 30 min.)
 - `session.invalidate()` “desconecta” al usuario, borrando los datos de la sesión (el típico “logout”)



Cómo funcionan las sesiones

- En realidad, las sesiones usan cookies.
- El servidor asigna automáticamente a cada usuario una cookie que hace de identificador único.
- Este identificador sirve como “clave” para acceder a la información propia del usuario



Reescritura de URLs

- ¿Qué pasa si el cliente se niega a aceptar la cookie o no puede hacerlo (porque no es un navegador web)?
- Idea: forzar al cliente a enviar su código identificador en cada petición (cuando se pulse un enlace o un submit)
 - Los enlaces pasan de ser “normales” a tener el identificador “empotrado”

 - Esto nos fuerza a que cada vez que pongamos un enlace o un submit en un JSP tengamos que “empotrar” el identificador (en el **href** o en el **action**). El método **response.encodeURL(direccion)** nos devuelve la dirección con el identificador “empotrado” en ella

<% String url = "lista.jsp?categoria=1" %>
<a href="<%=response.encodeURL(url)%>">
- Cuando el cliente no maneje cookies (ej. aplicación J2ME) este será el método a usar



Almacenamiento global

- En el objeto `session` cada cliente tiene su propio valor
- El objeto `application` funciona de manera parecida pero cuando se almacena un objeto todos los clientes acceden al mismo
 - `application.getAttribute(nombre)`
 - `application.setAttribute(nombre, objeto)`
- Lo almacenado en el objeto `application` se conserva en memoria mientras el servidor esté en marcha



3. Autenticación

Programada vs. declarativa



El problema de la autenticación

- **Autenticación:** verificar la identidad del usuario. Normalmente se hace con login + password, aunque puede hacerse con certificados digitales.
- Formas de gestionar la autenticación en aplics. web java
 - **Programada:** en cada página restringida hay que chequear si el usuario tiene permiso de acceso. Normalmente cuando el usuario hace login se coloca un objeto en la sesión que lo indica, y las páginas restringidas simplemente verifican la presencia de este objeto
 - **Declarativa:** el servidor controla el acceso automáticamente
 - en el web.xml se ponen ciertas etiquetas indicando qué páginas son restringidas
 - Hay que configurar el servidor web para decirle dónde están los logins+passwords de los usuarios (esto es **dependiente** del servidor)



Autenticación programada

- Cuando se comprueba que login y password son correctos se mete algo en la sesión que lo indique

```
//Sup. que este método chequea login y pw
if (Usuario.esOKLoginYPassword(login, password))
    session.setAttribute("autenticado","OK")
```

- Desde páginas restringidas se comprueba que el objeto esté en la sesión. En caso contrario, se va a otra página

```
//Primeras líneas de la página restringida
String ok = (String) session.getAttribute("autenticado");
if ((ok==null)||(!ok.equals("OK"))) {
    response.sendRedirect("index.jsp");
    return;
}
//Resto del JSP...
```



■ 4. JavaBeans, JSTL, y EL

- El trío que eliminó el Java de los JSPs



JavaBeans

- Son simplemente objetos con datos privados y métodos públicos `getXXX/setXXX` para acceder/cambiar los datos
 - ¿Hay algún *JavaBean* en la implementación de la práctica que os hemos pasado?
- En un JSP podemos instanciar un bean sin código Java, solo con etiquetas pseudoHTML

```
<jsp:useBean id="entrada" class="tw.dominio.Entrada" scope="page"/>
```

es casi lo mismo que

```
<% entrada = new tw.dominio.Entrada() %>
```

 - scope puede ser "page" (por defecto), "request", "session" o "application"



EL: expression language

- Con JavaBeans instanciados con `jsp:useBean` podemos usar una sintaxis simplificada: `${objeto.propiedad}`
`${entrada.texto}` sería lo mismo que `<%= entrada.getTexto() %>`
¡Cuidado! esto solo funciona si la variable se ha instanciado con `jsp:useBean`, no si se hace llamando a `new`
- Con parámetros HTTP, objetos en la sesión o en la aplicación podemos usar la misma sintaxis
`<%= session.getAttribute("loginActual") %>` es lo mismo que `${session.loginActual}` o que `${loginActual}`
 - Objetos predefinidos en EL
 - request, session, application
 - param (parámetros HTTP)
 - cookie
 - header (cabeceras HTTP)

si no se pone, va buscando en ese orden
`index.jsp?id=1` → `${param.id}`
`${cookie.usuario}`
`${header["User-Agent"]}`



JSTL (JSP Standard Tag Library)

- Etiquetas pseudoHTML que son en realidad “código Java disfrazado”, sirven para hacer bucles, condicionales, formatear fechas y números, llamar a SQL, etc.
 - Para usarlas se necesitan 2 librerías: standard.jar y jstl.jar
 - En la página hay que poner una directiva llamada *taglib* con una URL que le indica a Tomcat qué módulo/s vamos a usar de JSTL y un “prefijo corto” para marcar las etiquetas JSTL en el código

```
<%@taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core"%>
```

```
...
```

```
<c:if test="${application.visitas > 1000}">
```

```
    <h1>¡Mas de 1000 visitas!</h1>
```

```
</c:if>
```

- JSTL es un estándar, pero cada usuario se puede definir etiquetas aparte que encapsulen su código Java (no es trivial...)



Tecnológicas  Web

JSTL + EL + JavaBeans

- Cuando unimos estos tres componentes, podemos eliminar casi en su totalidad el Java de los JSPs...*(bueno, el Java sigue estando pero simplificado y disfrazado de HTML)*
- Por suerte o por desgracia la utilidad de cada cosa por separado es limitada, tenemos que usar los 3 juntos
 - Todo el código Java que llame un JSP deberían ser beans.
 - Todos los métodos sin parámetros deben ser getXXX. Por ejemplo, en lugar de verUltimasEntradas() usar getUltimasEntradas()
 - Si un método tiene parámetros, llamaremos primero a uno o varios setXXX, que guardarán el valor dentro del objeto
 - Cuando queramos mostrar el valor de un bean, lo hacemos con EL, no con `<%= ... %>`
 - Cuando queramos hacer bucles, condicionales, etc. lo hacemos con JSTL, no con código Java `<% ... %>`



Tecnológicas  Web

Página JSP “clásica”

```
<%@ page import="tw.datos.*, tw.negocio.*, java.util.*" %>
<html>
<head></head>
<body>
<%
    EntradasBD buscador = new EntradasBD();
    ArrayList<EntradaResumen> ultEntradas = buscador.verUltimasEntradas();
    for(EntradaResumen e : ultEntradas) { %>
        <%= e.getFecha() %> <br>
        <%= e.getTexto() %> <br>
    <% } %>
</body>
</html>
```



Página JSP “moderna”

- Aquí se ha cambiado en la clase EntradasBD el método verUltimasEntradas por getUltimasEntradas

```
<%@taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<jsp:useBean id="buscador" class="tw.datos.EntradasBD" scope="page"/>
<html>
<head></head>
<body>
<c:forEach items="${buscador.ultimasEntradas}" var="e">
    ${e.fecha} <br>
    ${e.texto} <br>
</c:forEach>
</body>
</html>
```

