SERVLETS

- **INTRODUCCIÓN:**

Servlet son una serie de aplicaciones programadas en Java que se ejecutan completamente en un servidor (Web Server). Un servlet va a aceptar una petición de un cliente a través del Web Server, hará su tarea y devolverá al cliente una respuesta.

Los servlets son el sustituto de los antiguos CGI (Common Gateway Interface), puesto que los CGI estaban escritos en C ó Perl y los servlets estarán escritos en Java, aportando este lenguaje la independencia de plataforma. Algunas ventajas de los servlets frente a CGI son:

- Persistencia de los servlets: Los servlets se cargan una sola vez por el Web Server y pueden mantener la conexión entre varias peticiones.
- Rapidez de los servlets: puesto que sólo se cargan una vez.
- Independencia de plataforma.
- Extensibilidad de los servlets. Como están escritos en Java, aportan todos los beneficios de este lenguaje. Java es lenguaje robusto y orientado a objetos, por lo que es fácilmente extensible a nuestras necesidades.
- Seguridad de los servlets: La única forma de invocar un servlet es a través de un Web Server. Esto da un alto nivel de seguridad, especialmente si el Web Server está protegido por un muro de contención (firewall). Esto significa que el cliente no puede borrar ni modificar nada del propio servidor. Para ampliar la seguridad, puedo definir usuarios y grupos de usuarios. Por último decir que se pueden usar características nativas de seguridad, como el encriptamiento de mensajes.
- Los servlets pueden ser usados por cualquier número de clientes.

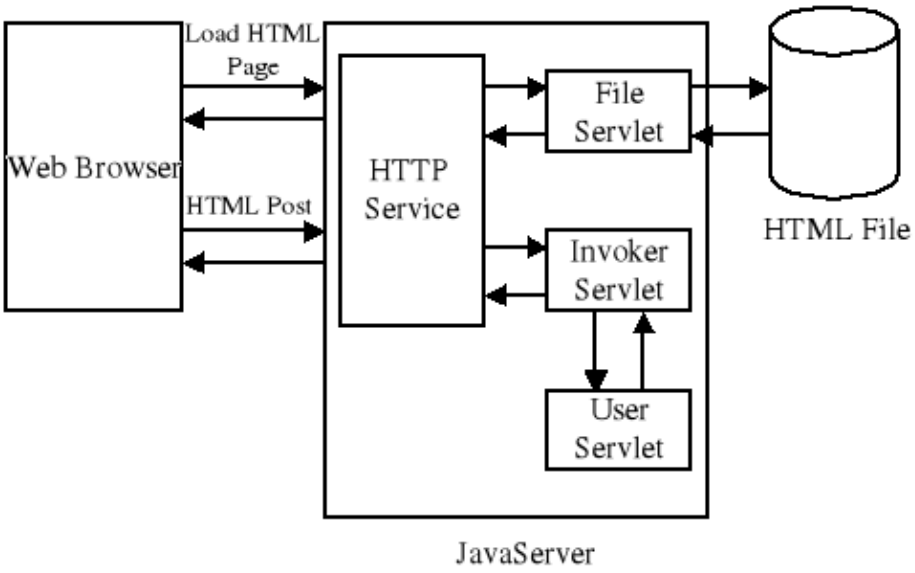
- **SERVLETS:**

Vamos a estudiar con detenimiento el flujo que va a tener lugar:

En primer lugar, el cliente (Navegador Web) hace una petición para cargar una página HTML. El http Web service (que está dentro del Web Server) recibe la petición reconociendo que se trata de una petición de lectura de una página HTML. Tras esto, invoca el File Servlet para buscar el fichero de E/S. La página HTML será devuelta al cliente expuesta en el navegador.

Si el navegador Web hace una petición POST de HTML, el http service recibirá de nuevo la petición. Si el POST requiere que se cargue un servlet, la petición será reenviada la invoker servlet, que invocará el servlet deseado. El servlet hará algún tipo de proceso y devolverá datos de vuelta al cliente a través de http.

Esto es lo que se muestra en la siguiente figura:



¿Cómo sabe exactamente el http Web service si tiene que invocar un servlet? En el lado del cliente habrá que especificar un URL que llame específicamente al servlet que queremos invocar.

http:// localhost:8080/servlet/ nombreServlet

El nombre del servlet suele ser un alias.

Ahora vamos a ver el ciclo de vida del servlet:

- Carga del servlet: Si no estaba cargado, se carga con el invoker. El servlet se carga una sola vez, y después se lanzan hilos del mismo servlet a todo cliente que lo solicite.
- Inicialización del servlet: Se llama al método init() del servlet para proceder a su inicialización. La llamada a este método se hace una sola vez tras ser cargado el servlet, aunque otros clientes quieran acceder a él.
- Desde la petición HTML POST se llama el método doPost() del servlet.
- El servlet realiza sus procesos y devuelve algo sobre el output stream.
- La respuesta que viene del servlet la recibe inicialmente el http Web service, que hará también sus procesos.

- **El API Servlet:**

El API Servlet es claro y simple. Un servlet es una clase Java que implementa la interfaz Servlet, que define cinco métodos:

- service(): Es el corazón de los servlets. El servidor invoca al método service() para ejecutar respuestas. El método service() acepta como parámetros objetos ServletRequest, que encapsula la petición del cliente y ServletResponse, que dispone de métodos para devolver información al cliente.
- init(): Es el lugar donde se inicaliza el servlet. Es un método que acepta como parámetros un objeto de tipo ServletConfig, que contiene la configuración del servidor, y se garantiza que solamente se llamará una vez durante la vida del servlet

- getServletConfig(): Debe devolver el objeto ServletConfig que se pasa como parámetro al método init().
- destroy(): Libera el servlet. Se llama cada vez que el servlet debe ser descargado. Todos los recursos del sistema bloqueados por init() son liberados al invocar este método y se garantiza que solo se le llamará una vez durante el la vida del servlet.
- getServletInfo(): devuelve una cadena con la información de Copyright.

Para asegurar un óptimo rendimiento, el servidor solamente carga una vez el servlet. Una vez cargado, permanece en memoria, estando disponible en cualquier instante para procesar cualquier petición. Por lo tanto, varias tareas pueden llamar simultáneamente al método service(), por lo que la sincronización dentro de service debe ser una premisa a no olvidar jamás.

- **La clase HttpServlet:**

La clase HttpServlet es una clase que implementa la interfaz Servlet incorporando además métodos específicos para servidores Web. Un uso típico de HttpServlet es el procesamiento de formularios html. Antes de poder el primer servlet, es necesario tener unas nociones básicas sobre el protocolo HTTP (HyperText Transfer Protocol), que es un protocolo de comunicaciones que se utiliza para que un cliente, por ejemplo, envíe peticiones a un servidor web.

HTTP es un protocolo orientado a petición−respuesta. Una petición HTTP está formada por unos campos de cabecera y cuerpo, que puede estar vacío. Una respuesta HTTP contiene un código de resultado y de nuevo una cabecera y un cuerpo.

Todos los métodos de esta clase, excepto getLastModified, tienen la misma declaración:

protected void método (HttpServletRequest req, HttpServletResponse res)

throws ServletException, Java.io.IOException { }

- Método doDelete: realiza la operación DELETE de http. La operación delete permite al cliente una petición para borrar un URI del servidor.
- Método doGet: realiza la operación GET de http.
- Método doHead: realiza la operación POST de http. Por defecto está realizado por la implementación de la operación GET, pero no devuelve datos acerca del cliente, sino tan solo las cabeceras.
- Método doOptions: realiza la operación OPTIONS de http. La implementación por defecto determina qué opciones de http se soportan. Este método no necesita ser sobrescrito al no ser que el servlet implemente nuevos métodos que no son soportados por el protocolo http.
- Método doPost: realiza la operación POST de http. Sirve para leer datos desde el request (como parámetros), poner las cabeceras en el response y escribir datos en response usando el output stream. Es decir, Post solamente estará disponible para el tratamiento de formularios.
- Método doPut: realiza la operación PUT de http. Consiste en enviar un fichero a través del FTP (file transport protocol).
- Método doTrace: realiza la operación TRACE de http. Devuelve un mensaje que contiene todas las cabeceras enviadas con el trace request.
- Método getLastModified: Devuelve el tiempo que estuvo el request sin modificarse. Su declaración es:

protected long getLastModified(HttpServletRequest req);

- Método service: realiza un servicio de http. Este método raramente se sobrescribe. En su lugar se sobrescriben doGet y doPost.

El método service tiene otra declaración que es:

protected void service (ServletRequest req, ServletResponse res)

throws ServletException, Java.io.IOException { }

Este método realiza el método SERVLET.SERVICE() llamando al servicio específico de http. Tampoco se suele sobrescribir.

Los métodos doOptions() y doTrace() disponen de una implementación por defecto y no está permitido sobrecargarlos. Los métodos que se pueden utilizar son doGet(), doPut(), doPost() y doDelete(), cuyas implementaciones por defecto en la clase HttpServlet devuelven un error HTTP de tipo `Bad Request'. Cualquier subclase HttpServlet debe sobrecargar uno o más de estos métodos para proporcionar la adecuada implementación a las acciones que desea realizar.

Los datos de petición se pasan a todos los métodos como primer argumento de tipo HttpServletRequest, que es una subclase de la clase más general ServletRequest. Las repuestas que pueden crear los distintos métodos se devuelven en el segundo argumento de tipo HttpServletResponse, que es una clase de ServletResponse.

- **La clase HttpServletRequest:**

Tiene los siguientes métodos:

- getAuthTipe(): Devuelve el esquema auténtico del request, o null si no hay.
- getCookies(): Devuelve un array de cookies.
- getDateHeader(String): Devuelve el valor del campo de tipo fecha de la cabecera del response.
- getHeader(String): Devuelve el valor del campo indicado de la cabecera.
- getHeaderNames(): Devuelve los nombres de las cabeceras.
- getIntHeader(String): Devuelve un entero correspondiente al campo de la cabecera introducido.
- getMethod(): Devuelve el método con el que se está haciendo el request.
- getPathInfo(): Devuelve información a cerca del path del servlet.
- getPathTranslated(): Devuelve información extra a cerca de un path que ha sido trasladado.
- getQueryString(): Devuelve la sentencia de consulta del URI.
- getRemoteUser(): Devuelve el nombre del usuario que hace el request.
- getRequestSessionId(): Devuelve el identificador de la sesión en este request.
- getRequestURI(): Devuelve el URI del request.
- getServletPath(): Devuelve el servlet que ha sido invocado.
- getSession(): Devuelve o crea la sesión asociada al request.
- getSession(boolean): Devuelve o crea la sesión asociada al request.
- isRequestedSessionIdFromCookie(): Devuelve true si el identificador para el request viene de un cookie.
- isRequestedSessionIdFromURL(): Devuelve true si el identificador para el request es parte del URL.
- isRequestedSessionIdValid(): Devuelve true si el identificador para el request es válido para esta sesión.

- **La clase HttpServletResponse:**

Sus métodos son:

- addCookie(Cookie): Añade un cookie al response actual.
- containsHeader(String): devuelve true si el nombre del campo que hemos pasado está en el mensaje de cabecera.

- encodedRedirectURL(String): codifica el URL para usarlo en el método sendRedirect.
- encodeURL(String): Codifica el URL incluyendo el identificador de la sesión.
- sendError(int): Envía un error al cliente usando el código de error.
- sendError(int, String): Envía un error al cliente usando el código de error y el mensaje.
- sendRedirect(String): reenvía un response al URL indicado o al cliente.
- setDateHeader(String, long): Añade el campo indicado a la cabecera del response con un valor de tipo fecha.
- setHeader(String, String): Añade el campo indicado a la cabecera del response con un valor de tipo String. Usaremos este método para forzar al navegador a cargar la página HTML desde el servidor en lugar de desde la caché.
- setIntHeader(String, int): Añade el campo indicado a la cabecera del response con un valor de tipo entero.
- setStatus(int): pone el código de estado para el response.

En el ejemplo `Propiedades.java' se ve el funcionamiento básico de los servlets.

- **Creación de servlets:**

Hay sólo dos pasos básicos para escribir un servlet que sirva una respuesta para una petición a través de http:

- Crear una nueva clase servlet que extienda Javax.servlet.http.HttpServlet. Esta clase a su vez extiende la clase Javax.servlet.GenericServlet y contiene un código especial para analizar información sobre la cabecera y el paquete del cliente. Este código se encuentra en la clase Javax.servlet.http.HttpServletRequest. Para evitar hacer referencia a estas clases con el nombre tan largo basta con poner estas sentencias de importación en la cabecera de nuestro servlet:

import Javax.servlet.;*

import Javax.servlet.http.;*

- Sobrescribir los métodos doGet y doPost. Aquí es donde se realiza realmente el trabajo para que el servlet tenga sentido. Estos métodos reciben por parámetros la petición del cliente y la respuesta al mismo. Ambos métodos lanzan excepciones. Una cabecera para estos métodos es:

public void doGet/doPost (HttpServletRequest req,

HttpServletResponse rep)

throws ServletException, Java.io.IOException {

HttpServletRequest proporciona los datos del solicitante, como número de sesión, información, etc. Es una especie de array cuyos elementos puedo pasar como parámetros a través del URL:

http://localhost:8080/servlet/NombreServlet ¿ nombre=Pepe & edad=20

HttpServletResponse proporciona servicios para dar una réplica al cliente. Representa la comunicación de vuelta al cliente.

El método doGet se sobreescribirá cuando el cliente quiera cargar una página HTML. doPost se implementará cuando el cliente quiera cargar un servlet, pasándole a éste ciertos parámetros.

Las primeras instrucciones que tenemos que programar tanto en doGet como en doPost son siempre las

mismas: tenemos que indicar el tipo de respuesta que le vamos a dar al cliente y crear un objeto de la clase PrintWriter para escribir la respuesta a través de un output stream:

/ indicamos que la respuesta va a ser una página HTML */*

resp.setContentType(text/html);

/ Preparamos la salida */*

new Java.io.PrintWriter(resp.getOutStream());

Opcionalmente, el servlet puede sobrescribir los métodos init y destroy para realizar algún tipo de inicialización y destrucción (en init podemos conectar a una base de datos y en destroy nos desconectamos).

La programación básica de estos métodos es:

public void init (ServletConfig cfg) throws ServletException {

super.init(cfg);

}

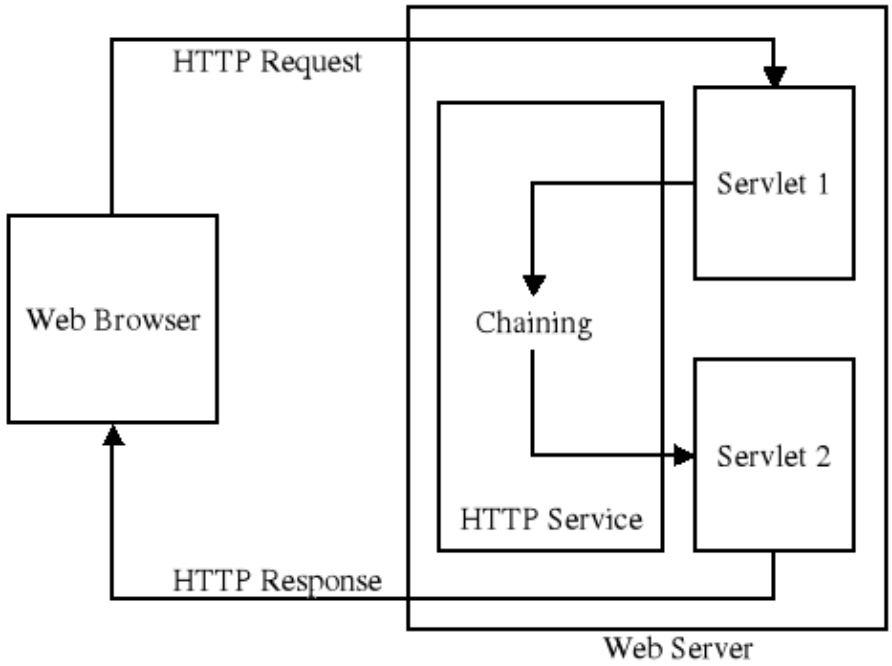
public void destroy () {

super.destroy();

}

- **ENLACE DE SERVLETS (SERVLET CHAINING):**

El chaining de los servlet es similar a las tuberías (piping) de C. Esto significa que la salida de un servlet va a ser la entrada de otro servlet. La salida del último servlet será la respuesta que se devolverá al cliente.



Por lo general, el cliente va a tener en su navegador una página HTML, que puede ser un formulario en el cual tendrá que rellenar textFields o cosas parecidas. El primer enlace se hace llamando a un servlet que maneje la información introducida por el usuario.

Por ejemplo, podemos tener un formulario en el que el cliente introduce su nombre en un textField (txtNombre) y luego pulsa el botón de aceptar. El botón de aceptar estará preparado de forma que tras ser pulsado, hace una llamada a un servlet que implementará un método doPost para trabajar con la información introducida por el usuario. La instrucción HTML para llamar al servlet es del tipo:

*<form action="http://localhost:8080/servlet/Nombre**(1)**" method="POST">*

El servlet llamado NombreServlet, tendrá una variable de la clase HttpServletRequest, que traerá información acerca del cliente e información de los parámetros introducidos por el cliente. De esta forma, en el método doPost del servlet NombreServlet, podemos extraer el parámetro que metió el cliente:

String nombre = (String) req.getParameter("txtNombre");

Si ahora quisiéramos hacer referencia a otro servlet desde el servlet actual, tendríamos que redireccionar la salida de NombreServlet al nuevo servlet. Esto lo podemos hacer porque NombreServlet tiene también un objeto de la clase HttpServletResponse. Con este objeto (res) redireccionamos el output stream de NombreServlet a un nuevo servlet con el método del objeto res llamado sendRedirect, que tiene como parámetro el URL del siguiente servlet:

*res.sendRedirect("http://localhost:8080/servlet/NombreServlet2**(2)**");*

De esta forma, podremos redireccionar todas las veces que queramos.

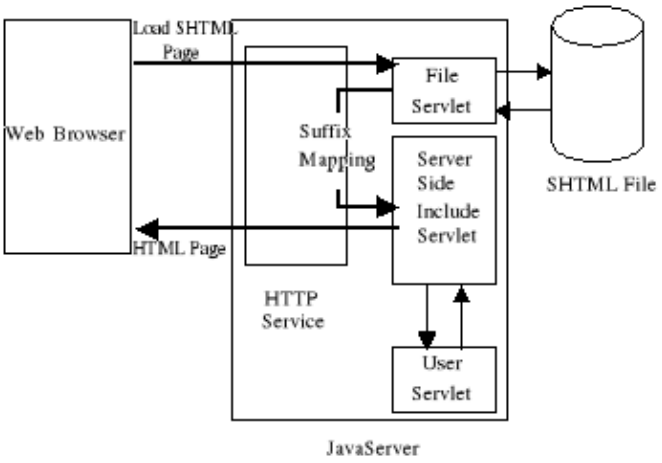
Cuando estemos en el último servlet, la salida del mismo será lo que devolvamos al cliente.

Ejemplo en los ficheros `TableFilter.java' y `TablaPeriodica.java'.

- **SERVER−SIDE INCLUDES:**

Server−side includes va a permitir `empotrar' un servlet que contiene documentos HTML usando una etiqueta especial de servlet. Esto se realiza usando:

- Un servlet interno especial (server−side include servlet), que procesa las etiquetas de servlet.
- Un mapa de sufijos (suffix mapping) que invoca el server−side include en cualquier momento que se requiere un fichero de un cierto tipo (suele ser indicado en un fichero shtml).



Flujo que se da lugar:

- En primer lugar el cliente solicita una página shtml. El Web server procesa la petición, invoca el servlet a leer y sirve el fichero.
- El Web server mira en el mapa en busca de sufijos o prefijos. En este caso, el fichero shtml lleva un sufijo, que indica al Web server que debe invocar el servlet `server−side include'.
- El servlet `server−side include' es invocado (en caso de que sea necesario) e invocado. Este servlet va a analizar el stream de datos del shtml buscando etiquetas.
- Para cada etiqueta que se encuentra, se invoca (incluye) el servlet correspondiente.
- El Web server devuelve la página HTML formada por todos los servlets invocados.

El código para indicad una etiqueta, es decir, para indicar que queremos incluir el código de otro servlet, es el que sigue:

<servlet name=SERVLET_NAME

code=SERVELT.CLASS

codebase=SERVELT_CODE_BASE

INIT_PARAM1=VALUE1

INIT_PARAM2=VALUE2

INIT_PARAMn=VALUEn>

<param name=PARAM1 value=PARAM_VALUE1

param name=PARAM2 value=PARAM_VALUE2

param name=PARAMn value=PARAM_VALUEn>

</servlet>

Cuando se encuentra una etiqueta de inserción de servlet, se va a cargar el servlet de nombre `SERVELT_NAME. Si el servlet no se encuentra de esta forma (bien porque el nombre está equivocado, o porque no damos nombre), se carga usando la clase dada en el campo `code'. Los demás campos se usan para inicializar el servlet que se quiere incluir.

- **SESIONES (SESSIONS):**

El API de los servlets nos da un modelo para guardar información sobre una sesión. Una sesión consiste en todas las peticiones (request) hechas durante una invocación desde un navegador. En otras palabras, una sesión comienza cuando se abre el navegador y termina cuando este se cierra. El primer problema con el que nos encontramos es identificar a cada cliente que se conecta. Esto se soluciona asignando un identificador a cada cliente cada vez que este hace una petición a través de http. Se podría pensar como identificador en la dirección IP de cada navegador, pero esto no es posible porque puede haber varios clientes usando el mismo navegador, y por tanto, con la misma dirección IP. Lo que usaremos son las llamadas cookies o una técnica conocida como la reescritura del URL.

- **La clase HttpSession:**

Métodos de esta clase:

- getCreationTime(): Devuelve el tiempo en que la sesión fue creada.
- getId(): Devuelve el identificador de la sesión.
- getLastAccessedTime(): Devuelve la última vez que el cliente hizo una petición para esta sesión.
- getMaxInactiveInterval(): Devuelve la máxima cantidad de tiempo en el que la sesión se mantiene en el motor del servlet.
- getValue(String): Devuelve el valor del objeto asociado al campo que se le pasa por parámetro.
- getValueNames(): Devuelve un array de los datos de la sesión.
- invalidate(): invalida la sesión y la borra del contexto de la sesión.
- isNew(): Devuelve true si el contexto ha sido creado por el Server pero no ha sido usado por el cliente.
- putValue(String, Object): Pone el dato objeto en el campo de nombre String.
- removeValue(String): Borra el campo del array de datos de la sesión.
- setMaxInactiveInterval(int): pone un tiempo límite que una sesión puede estar inactiva antes de que el motor de servlets la elimine.

Ejemplo: tenemos un formulario en el que se le pide al usuario que introduzca su login y password en sendos textfields llamados tLogin y tPass. Esta es la secuencia que hay que poner para crear una nueva sesión:

public void doPost(HttpServletRequest req, HttpServlet Response)

throws ServletException, Java.io.Exception {

/ Creamos una sesión. Si ya la teníamos creada, la mantiene */*

HttpSession sess = req.getSession(true);

/ Añadimos a la sesión el campo Login con el valor de tLogin */*

sess.putValue (Login, req.getParameter(tLogin));

sess.putValue (Password, req.getParameter(tPass));

}

De esta forma, tendremos creada una sesión. Cuando tengamos que navegar a otro servlet, la sesión vendrá conmigo, de manera que pueda utilizar la información en ella almacenada en cualquier momento.

Ejemplo de las sesiones en `Sesiones.java' y `InformaciónServlet.java'.

- **Manejo de los datos de sesión:**

Hay tres aspectos del manejo de datos que deben de estar muy claros: el intercambio de sesiones, la relocalización de sesiones y la persistencia de las sesiones.

- **Intercambio de sesiones (session swapping):** Todos los servidores de servelts tienen una cantidad limitada de recursos para la información de la sesión. En un esfuerzo por mantener el consumo de estos recursos bajo control, la mayoría de los servidores de servlet tendrán un número máximo de sesiones concurrentes en un mismo tiempo. Del mismo modo, tendremos un tiempo máximo de espera, de forma que cuando una sesión permanece inactiva durante demasiado tiempo dentro de un servidor, esa sesión será sustituída por otra. Esto sólo se podrá hacer si todos los objetos (datos) de la sesión son serializables (implementan la interfaz java.io.Serializable). Si la sesión no es serializable deberemos tenerla en memoria hasta que salga por sí sola.
- **Relocalización de las sesiones (session relocation):** No hay ningún convenio que diga que un servlet requerido sea servido por la misma JVM (Java Virtual Machine) o ni siquiera por el mismo servidor físico. Un servidor debe de enviar el servlet lo más rápido posible. Por eso, para trabajar con datos almacenados dentro de una sesión, el objeto sesión debe de ser relocalizable. De nuevo, esto conlleva que los datos (objetos) almacenados en la sesión deben de ser serializables. Si los datos de la sesión son serializables será fácil moverlos de una JVM a otra.

- Persistencia de las sesiones (session persistence): Los servidores WEB deben de estar activados 24 horas al día, 7 días a la semana. Sin embargo, de vez en cuando es necesario tumbar los servidores para su mantenimiento. De nuevo, la serialización nos va a salvar el día. El servidor de servlets serializará todas las sesiones y las volcará en un disco auxiliar. Cuando la limpieza del servidor esté completada, se recuperarán las sesiones sin que el usuario note el cambio.

- **Cookies:**

Hemos visto como ver información del usuario a través de un objeto session. Esta información suele ser bastante general (ID de la sesión, inicio de la sesión, etc.). Sin embargo hay casos en que queremos más que eso. Supongamos el caso de una tienda virtual. El usuario coge su carro virtual y visita las diversas subpáginas (servlets) de la tienda virtual en busca de sus artículos. Con cada cambio de página (de servlet), es conveniente mantener información de lo que el usuario ya ha comprado. Para ello se usan las cookies.,

Una cookie es una serie de datos que pueden ser empotrados en el request o el response. Un escenario típico

es en el que el Web Server empotra una cookie el la cabecera del response y el navegador devuelve el mismo cookie con cada subsecuencia request. Una de las piezas de información que puede ser almacenada es el identificador de sesión. Las cookies pueden ser desactivadas por el usuario, y pueden contener otros tipos de informaciones, como comentarios, tiempo de vida, etc.

Métodos de la clase cookie:

- clone(): Devuelve una copia del objeto.
- getComent (): Devuelve un comentario describiendo el cookie o null si no existe.
- getDomain(): Devuelve el dominio de la cookie.
- getMaxAge(): Devuelve la duración de la cookie.
- getName(): Devuelve el nombre de la cookie.
- getPath: Devuelve el prefijo de todas las URL para las que se instanció la cookie.
- getSecure(): Devuelve el valor del flag de seguridad.
- getValue(): devuelve el valor de la cookie.
- getVersion(): Devuelve la versión de la cookie.
- setCommet(String): Escribe un comentario describiendo el objetivo de la cookie.
- setDomain(String): Escribe el dominio de la cookie.
- setMaxAge(int): Escribe la duración máxima de la cookie.
- setPath(String): Esta cookie se presentará solo a las request que empiezan con el URL indicado.
- setSecure(boolean): Si es true, la cookie será usada solo enviando un protocolo de seguridad de http.
- setValue(String): Escribe el valor de la cookie.
- setVersion(int): Escribe la versión de esta cookie.

La clase HttpServletRequest tiene un método llamado getCookies que devuelve un array de objetos cookies para la petición actual. De esta forma, podemos hacer referencia a las cookies de la siguiente forma:

Cookie cookies [] = req.getCookies();

Ejemplo de esto en `VectorCookies.java'.

Hay ocasiones en las que el usuario desactivará las cookies. Para poder trabajar en estos casos vamos a utilizar la reescritura del URL. Todos los links y redireccionamientos que hagamos serán incorporando el identificador de la sesión como parte del URL. Ejemplo en `CounterRewrite.java'.

- **Los eventos en las sessions:**

Los eventos que pueden ocurrir dentro de una sesión es que hemos introducido o sacado un objeto de la sesión.

A veces querremos tener notificaciones a cerca de cuando un objeto está ligado o no a una sesión. Cuando un objeto está ligado a una sesión es el momento ideal para hacer algunas operaciones de inicialización, como abrir un fichero, conectarse a una base de datos, etc. Cuando el objeto deja de estar ligado a una sesión, querremos hacer tareas de clausura. Para trabajar con esto, vamos a tener la interfaz HttpSessionBindingListener, que tiene dos métodos para implementar:

- valueBound(HttpSessionBindingEvent): notifica al listener que está siendo ligado a una sesión.
- valueUnbound(HttpSessionBindingEvent): notifica al listener que está siendo desligado de una sesión.

Los usos de todo esto aparecen en los ejemplos `Binder.java' y `LigaduraObjetos.java'. Este último va a implementar la interfaz HttpSessionBindingListener, por lo que se sobrescriben los métodos valueBound() y

valueUnbound(). Ahora, cuando un objeto de la clase `LigaduraObjetos' sea añadido a una sesión, el método valueBound() será invocado. De la misma manera, se invocará el método valueUnbound() cuando el objeto se saque de la sesión.

El programa `Binder.java' va a incluir objetos de la clase LigaduraObjetos dentro de una sesión, para ver el funcionamiento de los eventos.

- **SEGURIDAD EN LOS APPLETS:**

Para comprobar la autenticidad del usuario que va a usar nuestro servlet vamos a hacer que este devuelva un nuevo request en el que se pide el login y password del usuario. Cuando el usuario rellena el login y el password, podremos comprobar la validez del mismo en una base de datos.

Tenemos dos formas de usar la autentificación http:

- **Autentificación básica:**

La primera de ellas es la autentificación básica. Esta autentificación la soportan casi todos los navegadores, aunque es de fácil decodificación puesto que utiliza un encriptamiento conocido como codificación Base64. Para usar la autentificación básica hay que seguir estos pasos:

  - Preparar la cabecera del response para forzar al navegador a cargar la página HTML desde el servlet en lugar desde la caché:

resp.setHeader(Expires, Tues, 01 Jan 2001 00:00:00 GMT);

  - Tenemos que hacer que el servidor proteja nuestro servlet con un esquema de seguridad. Cada servidor lo hará de una manera diferente. Una vez configurado como es debido, cuando intentemos acceder al servlet, el servidor nos mostrará una pantalla en la que nos pedirá el login y el password.
  - Cuando el cliente indique su información de autentificación, esta se codifica y es enviada al servidor para su validación. El servidor la decodificará y comprobará su validez en una base de datos.
  - Una vez validada, se invoca el servlet y se usa el método getRemoteUser() para extraer el nombre del cliente actual. De ninguna manera podremos obtener el password del cliente.

String user = req.getRemoteUser();

- **Autentificación personalizada:**

Es la que se usa cuando queremos tener un mayor control sobre la gente que accede a nuestro servlet. Se usará cuando tengamos información almacenada en una base de datos externa, o tengamos un esquema de autentificación diferente al del servidor. La autentificación personalizada es semejante a la anterior, pero en lugar de usar el servidor para validar login y password, tenemos que hacerlo ahora nosotros. Aun usamos la validación Base64.

En este ejemplo aparece parte del código para la autentificación personalizada:

// Creación de la sesión:

HttpSession session = req.getSession(true);

/ Tenemos el campo String USER_KEY que se encarga de almacenar en la sesión el nombre del usuario */*

String sessionUser = null;

if (session != null) // Si la sesión ya existía

sessionUser = (String) session.getValue(USER_KEY);

/ Si la sesión existe, pero aun no se tiene usuario para la misma, obtenemos el password de la cabecera y la validamos con un método llamado validarUsuario que tenemos que construir nosotros */*

String user = null;

if (sessionUser == null) user = validarUsuario(req);

/ Si no hay usuario para la sesión y el usuario no ha sido autentificado, forzamos un login */*

if ((sessionUser == null) && (user == null) {

/ El usuario no tiene autorización para acceder a la página. Mostramos un diálogo para introducir login y password */*

resp.setStatus(HttpServletResponse.SC_UNAUTHORIZED);

resp.setHeader(WWW−Authenticate, BASIC realm= \custom\);

/ Aquí vendría código HTML para mostrar en pantalla */*

}

else {

/ Hemos autentificado el nombre del usuario, pero aun no tenemos usuario para la sesión. Lo ponemos */*

if ((sessionUser == null) && (session ¡= null)) {

session.putValue(USER_KEY, user);

sessionUser = user;

}

}

Por último, decir que podemos usar los textFields para comunicarnos con el usuario, de forma que podamos obtener fácilmente su login y password.

- **FORMULARIOS DE HTML:**

Como otras etiquetas de HTML, los formularios tienen etiqueta inicial y final (<FORM ... /FORM>). Dentro del formulario podemos meter cajas de texto, botones, check box, etc. También puede contener elementos de HTML, como texto e imágenes. Cuando el usuario haya cargado la página, seguramente pulsará un botón submit, que hará cualquier acción. Esta acción va a ser, en la mayor parte de los casos, una invocación a un servlet o para enviar un e−mail. Cuando se pulse el botón, el navegador recogerá información almacenada en

cajas de textos o similares y la envía para ser procesada en el servidor. El servidor buscará el servlet invocado, lo ejecutará y devolverá otra información también en formato HTML.

El formulario debe tener tres atributos para que al pulsar un botón se haga algo:

- Atributo ACTION: Es un atributo requerido. En él vamos a indicar el URL del servlet al que queremos invocar cuando se pulse el botón. Este servlet va a ser el que reciba la información contenida en el formulario dentro de cajas de texto o similares. Podemos referenciar el URL con un alias que tenga definido en el servidor.

<FORM ACTION = http://localhost:8080/servlet/nombreServlet ...>

Si en lugar de invocar un servlet, queremos enviar un e−mail a cierto buzón cuando pulsemos el botón, escribiremos:

<FORM ACTION = mailto:rschocano@hotmail.es ...>

- Atributo ENCTYPE: Este atributo no es obligatorio. El navegador va a codificar los datos del formulario después de pasarlos al servidor, quien deberá decodificarlo o pasarlos codificados a otra aplicación. Podemos cambiar el tipo de codificación usando este atributo. Las posibles codificaciones que tendremos serán:
  - ♦ application/x−www−form−urlencoded: Es la codificación por defecto. Este codificador va a cambiar los espacios en blanco de los valores de los parámetros en el símbolo `+'. Los caracteres que no sean alfanuméricos los va a cambiar por el símbolo `%' seguido del valor hexadecimal en el código ASCII del carácter.
  - ♦ multipart/form−data: Se usa solo con formularios que contienen un campo de selección de ficheros.
  - ♦ text/plain: Se usa para enviar los parámetros del formulario a través de un e−mail. Cada elemento en este formulario se pone en una línea con el nombre y el valor separados por un símbolo `='.
- Etiqueta METHOD: Es un atributo requerido. Especifica el método que quiere que se implemente en el servlet al que se invoca (GET o POST).

- **La etiqueta INPUT:**

Se utiliza para definir campos en el formulario, como cajas de texto o botones. Una estructura genérica para esta etiqueta es:

<INPUT TYPE= tipo NAME= nombre [atributos adicionales]>

El atributo TYPE es obligatorio e indica el tipo de campo que vamos a introducir en el formulario. El atributo nombre indica el nombre del campo y también es obligatorio.

Vamos a ver los campos que se pueden introducir en un formulario a través de INPUT:

- ♦ **Botón:** Los botones normales en HTML tienen deben de tener al menos dos atributos:
  - ◊ NAME: El nombre del botón.
  - ◊ VALUE: la etiqueta del botón

<INPUT TYPE=BUTTON NAME=accion VALUE=Aceptar>

- ♦ **Check box:** Sirve para dar al usuario del formulario una forma de seleccionar o deseleccionar

un ítem particular. Estos son sus atributos:

- ◊ CHECKED: Dice que por defecto, este ítem estará seleccionado.
- ◊ NAME: Es obligatorio e indica el nombre del ítem.
- ◊ VALUE: Es obligatorio e indica el valor que será enviado al servlet si el check box es seleccionado.

Un ejemplo de declaración para check box es el siguiente:

<INPUT TYPE=CHECKBOX NAME=age VALUE=34>

<INPUT TYPE=CHECKBOX NAME=age VALUE=12>

<INPUT TYPE=CHECKBOX NAME=age VALUE=38>

<INPUT TYPE=CHECKBOX NAME=age VALUE=22>

- ♦ **Ficheros:** Nos va a permitir seleccionar un fichero almacenado en el disco local del usuario y enviar los contenidos de este fichero al servidor cuando el botón submit se presiona. El navegador creará una caja de texto que aceptará la cadena de texto del usuario (nombre fichero). Los atributos son:
  - ◊ ACCEPT: Dice el tipo de ficheros que el usuario puede seleccionar.
  - ◊ MAXLENGTH: Longitud máxima (en caracteres) del nombre del fichero.
  - ◊ NAME: es obligatorio e indica el nombre de la caja de texto.
  - ◊ SIZE: Tamaño (en caracteres) de la caja de texto.
  - ◊ VALUE: Nombre del fichero por defecto.

Un ejemplo de ficheros es:

<INPUT TYPE=file NAME=myfile SIZE=25>

- ♦ **El tipo oculto (HIDDEN INPUT TYPE):** Es un tipo que se oculta a la vista del usuario. Es una forma de ocultar información adicional en el formulario HTML. Esta información no va a ser modificada por el usuario. Lo vamos a usar, por ejemplo, para enviar información que no concierne al usuario hacia el servlet. Los atributos que debe llevar son el nombre (NAME) y el valor (VALUE):

<INPUT TYPE=hidden NAME=versión VALUE=1.0>

- ♦ **Imágenes:** Este tipo va a crear un botón con una imagen. Este botón personalizado se va a crear usando la imagen que el usuario especifique y, cuando hagamos clic sobre él, se enviarán las coordenadas (X,Y) del clic del ratón cuando el clic se haga dentro de la imagen. Los valores de las coordenadas se envían como <name>.x y <name>.y. Los atributos son:
  - ◊ ALIGN: El tipo de alineamiento de la imagen con el texto: TOP, TEXTTOP, MIDDLE, ABSMIDDLE, CENTER, BOTTOM, BASELINE y ABSBOOTOM.
  - ◊ BORDER: Especifica el grosor del borde de la imagen en píxeles.
  - ◊ NAME: Es obligatorio e indica el nombre del botón con imagen
  - ◊ SRC: Es obligatorio e indica el URL de la imagen.

Un ejemplo es:

<INPUT TYPE=image NAME=submit SRC=imagen.gif ALIGN=middle>

- **Password:** Este tipo es como una caja de texto, con la diferencia de que su contenido se enmascara con asteriscos. Los atributos son:
    - ◊ MAXLENGTH: Número de caracteres que se aceptan.
    - ◊ NAME: Es obligatorio e indica el nombre del campo password.
    - ◊ SIZE: Tamaño de la caja de texto.
    - ◊ VALUE: Valor por defecto.

Un ejemplo es:

<INPUT TYPE=password NAME=pass SIZE=10>

- **Radio Button:** Sirve para presentar al usuario una lista con varias opciones de las que solo se puede seleccionar una. Los atributos son:
    - ◊ CHECKED: Indica si el botón está activo.
    - ◊ NAME: Es obligatorio e indica el nombre del ítem.
    - ◊ VALUE: Es obligatorio e indica el valor que será enviado al servlet si el Radio Button es seleccionado.

Un ejemplo es:

<INPUT TYPE=radio NAME=time VALUE=NEVER CHECKED>

- **Botón Reset:** Sirve para reiniciar todos los campos que hay en el formulario a sus valores por defecto. El Servlet no se hará eco del efecto de este botón. Su único atributo es VALUE, que es la etiqueta de este botón.
- **Botón Submit:** Es el que se utiliza para hacer llamadas a servlets. Sus atributos son:
    - ◊ NAME: Es obligatorio e indica el nombre del botón Submit
    - ◊ VALUE: Es obligatorio e indica la etiqueta del botón Submit.
- **Caja de texto:** Es una región en la que el usuario puede escribir lo que quiera. Sus atributos son:
    - ◊ MAXLENGTH: Número total de caracteres aceptados
    - ◊ NAME: Es obligatorio e indica el nombre de la caja de texto.
    - ◊ SIZE: Longitud en caracteres de la caja de texto.
    - ◊ VALUE: Valor por defecto.

Un ejemplo es:

<INPUT TYPE=text NAME=txt1 SIZE=60 MAXLENGTH=60>

- **La etiqueta SELECT:**

Los check box y radio button son buenos, pero a veces tendremos necesidad de usar menús desplegables o listas. Esto lo vamos a hacer usando la etiqueta SELECT. La estructura genérica es:

<SELECT NAME=name SIZE=n MÚLTIPLE>

<OPTIONS> tags

</SELECT>

El atributo NAME es obligatorio e indica el nombre del parámetro que será enviado al servlet. El atributo MÚLTIPLE se utilizará cuando se desee que el usuario pueda seleccionar más de un elemento. El atributo

SIZE indica el número máximo de opciones que podemos tener en la caja de texto. Si SIZE es menor que el número de opciones, tendremos un menú desplegable.

Ejemplo de lista desplegable:

<SELECT NAME=time SIZE=1>

<OPTION VALUE=never> NEVER

<OPTION VALUE=6> LESS THAN 6 MONTHS

<OPTION VALUE=6−12> 6 − 12 MONTHS

<OPTION VALUE=always> ALWAYS

</SELECT>

- **La etiqueta TEXTAREA:**

Crea una caja de texto que puede contener más de una línea. Su estructura genérica es:

<TEXTAREA NAME=name COLS=N ROWS=M>

valor por defecto

</TEXTAREA>

En el siguiente ejemplo creamos una caja de texto multilínea de 60 columnas y 5 filas:

<TEXTAREA NAME=comentario COLS=60 ROWS=5>

</TEXTAREA>

APÉNDICE: EJECUCIÓN DE SERVLETS CON J2SDK:

Como ya se ha dicho, los servlets sólo pueden verse en un navegador Web. A continuación se explica la forma de arrancar un applet a través del servidor Web que se proporciona con el JSDK (Java Servlet Development Kit).

* Editar el fichero default.cfg