

J2EE: Servlets

Antonio Espín Herranz

Contenidos

- Introducción: Servlets, ciclo de Vida.
- Interfaz Servlet.
- Contexto del Servlet.
- Peticiones / Respuestas.
- Sesiones.
- Envío de peticiones.
- Aplicaciones Web.
- Eventos de aplicación.
- Distribución de Peticiones al Servlet.
- Paquetes servlet, servlet.http.

Introducción

Ciclo de vida de los Servlets

¿Qué es un Servlet?

- Los Servlets son módulos escritos en Java que se ejecutan en contenedor Web para extender sus capacidades de respuesta a los clientes al utilizar las potencialidades de Java.
- Los Servlets son para los servidores lo que los applets para los navegadores, aunque los servlets no tienen una interfaz gráfica.
- Un servlet nos dará una respuesta en código HTML, que visualizamos desde un navegador.

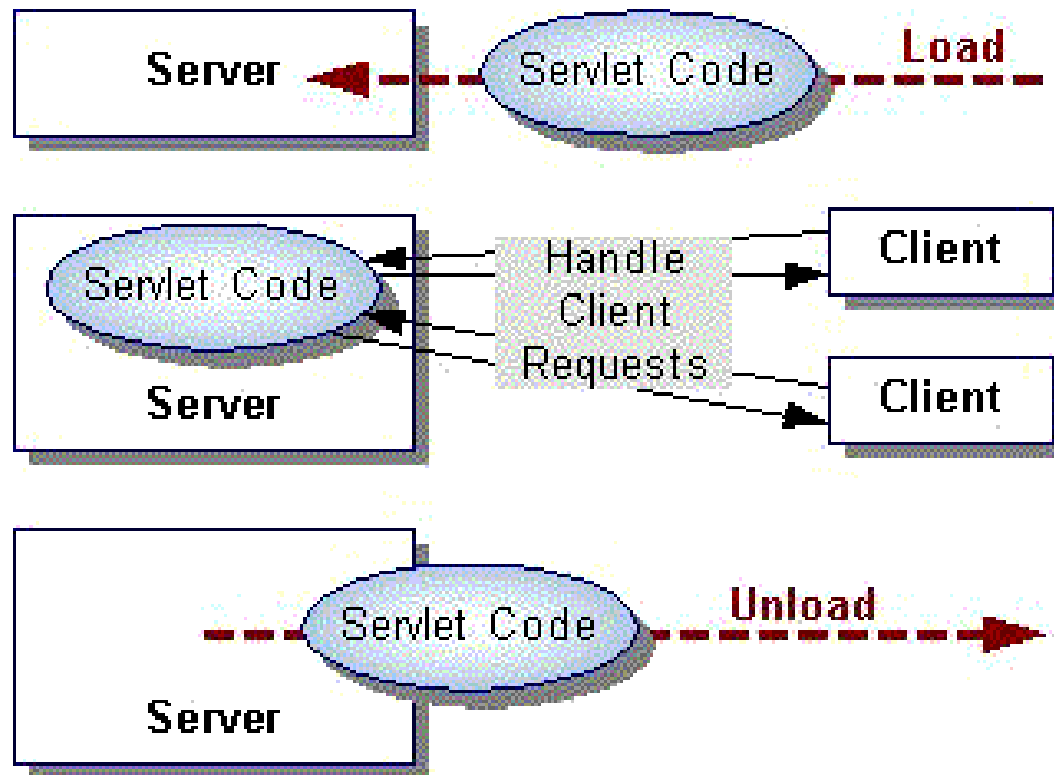
Servlet

- Los Servlets son un reemplazo efectivo para los CGI en los servidores que los soporten ya que proporcionan una forma de generar documentos dinámicos utilizando las ventajas de la programación en Java como conexión a alguna base de datos, manejo de peticiones concurrentes, programación distribuida, etc.
- Por ejemplo,
 - Un servlet podría ser responsable de procesar los datos desde un formulario en HTML como registrar la transacción, actualizar una base de datos, contactar algún sistema remoto y retornar un documento dinámico o redirigir a otro servlet u otro recurso.

Ciclo de vida de un Servlet

- **El Servidor controla el ciclo de vida de los Servlets.**
- **Inicializar un Servlet** → Cuando un servidor carga un servlet, ejecuta el método init del servlet. La inicialización se completa antes de manejar peticiones de clientes y antes de que el servlet sea destruido.
- **Interactuar con Clientes** → Después de la inicialización, el servlet puede manejar peticiones de clientes.
- **Destruir un Servlet** → Los servlets se ejecutan hasta que el servidor los destruye, por cierre el servidor o bien a petición del administrador del sistema.

Esquema del ciclo de vida de un Servlet



¿Qué es una JSP?

- Java Server Page:
- Es una tecnología similar a los Servlets que ofrece una conveniente forma de agregar contenido dinámico a un archivo HTML por utilizar código escrito en Java dentro del archivo utilizando tags especiales que son procesados por el servidor Web antes de enviarlos al cliente.
- La posibilidad de usar APIs de Java hacen de JSP una poderosa herramienta de desarrollo ya que se obtiene la ventaja de la programación orientada al objeto, como creación de clases especiales llamadas componentes o Java Beans, independencia de la plataforma propia de la programación en Java, etc.

Diferencias entre un Servlet y JSP

- Los **Servlets** son clases que deben implementar la clase abstracta **HttpServlet** → **implementa los servicios del protocolo http**, en especial el método **doGet()** o **doPost()**.
 - *Los servlets se compilan con javac.*
- **JSP** contienen código Java entre código HTML utilizando los símbolos **<% y %>**. Por esto un archivo JSP debe ser interpretado por el servidor al momento de la petición por parte del usuario.
 - *La páginas no hay que compilarlas, automáticamente se convierten a un servlet y se compilan, estas operaciones se realizan al llamarlas por 1ª vez.*

Ejemplo de Servlet

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

public class Servlet1 extends HttpServlet {

    public void doGet (HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
        PrintWriter out;
        String title = "el titulo";
        // primero selecciona el tipo de contenidos y otros campos de cabecera de la
        // respuesta
        response.setContentType("text/html");
        // Luego escribe los datos de la respuesta
        out = response.getWriter();
        out.println("<HTML><HEAD><TITLE>");
        out.println(title);
        out.println("</TITLE></HEAD><BODY>");
        out.println("<H1>" + title + "</H1>");
        out.println("<P>El mensaje que queremos dar</P>");
        out.println("</BODY></HTML>");
        out.close();
    }
}
```

Como llamar al servlet desde HTML

- El servlet le podemos llamar desde el **action** del **FORM**.

```
<html>
<head><title>Titulo de la pagina</title></head>
<body>
  <center>
    <form method="POST" action="Servlet1">
      <input type="submit" value="Conectar">
    </form>
  </center>
</form>
</html>
```

Ejemplo de JSP

<!-- La página JSP también se llama desde el action -->

```
<%@ page language='java' contentType="text/html" %>
```

```
<%! int count=0; %>
```

```
<html> <head>
```

```
<title>Hola y números. Intro to JSP</title></head>
```

```
<body bgcolor="white"> Hola, mundo. Te lo repito
```

```
<%= count++ %>
```

```
<% if (count == 1) { %> vez <% }
```

```
else { %> veces <% } %>
```

```
</body>
```

```
</html>
```

Interfaz del Servlet

Métodos http vs Servlet

- El **protocolo http** dispone de los siguientes métodos:
 - GET, HEAD, POST, PUT, DELETE, OPTIONS, TRACE.
- El **Servlet** dispone de los métodos equivalentes:
 - doGet, doHead, doPost, doPut, doDelete, doOptions, doTrace.

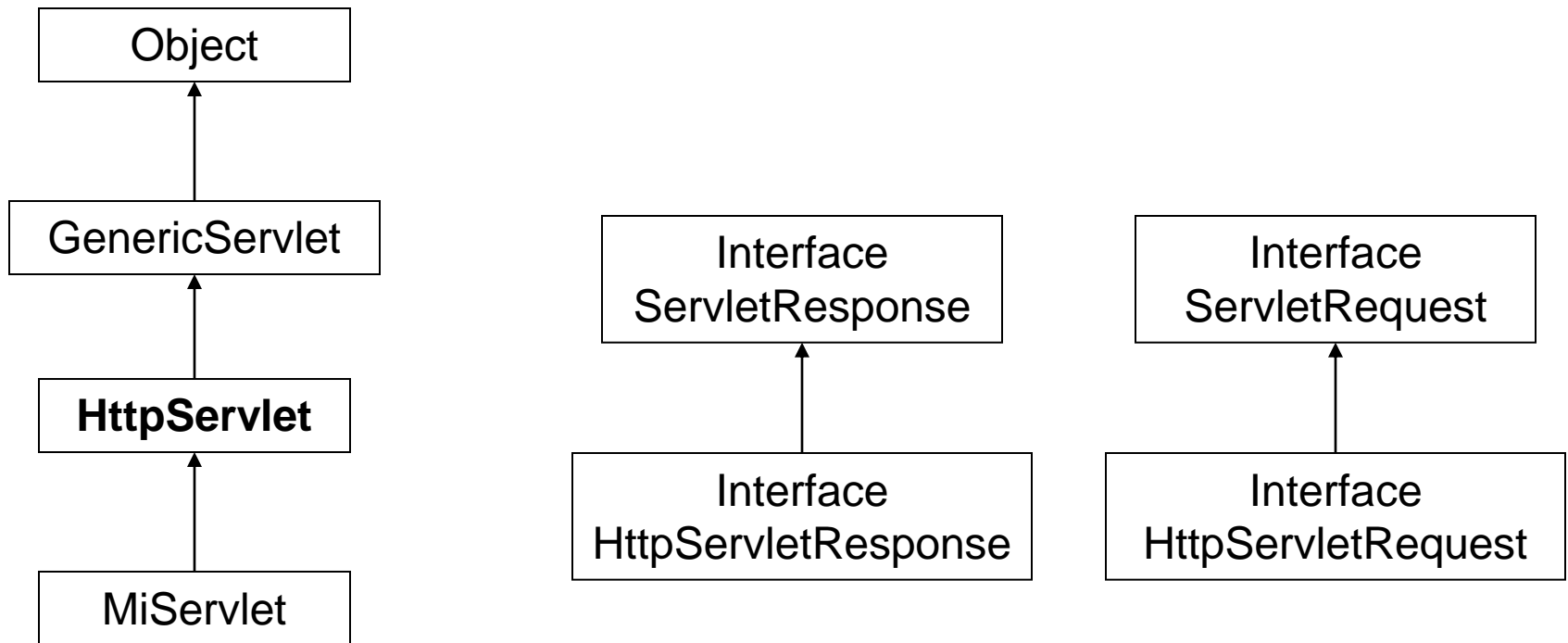
Métodos http vs Servlet

- **doDelete:** Realiza la operación DELETE de http. La operación delete permite al cliente una petición para borrar un URI del servidor.
- **doGet:** Realiza la operación GET de http.
- **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.
- **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.
- **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.
- **doPut:** Realiza la operación PUT de http. Consiste en enviar un fichero a través del FTP (file transport protocol).
- **doTrace:** Realiza la operación TRACE de http. Devuelve un mensaje que contiene todas las cabeceras enviadas con el trace request.
- **service:** Recibe peticiones Get y Post.

Métodos http vs Servlet

- **TODOS** Los métodos del Servlet **do...()** reciben dos parámetros: **request** y **response**.
- request:
 - Representa la petición que envía el cliente.
 - De esta se puede extraer información:
 - Cookies, la IP, el puerto, el protocolo, parámetros, campos de un formulario, crear la sesión.
- response:
 - Un canal de comunicación con el cliente.
 - Escribimos en el cliente en **HTML**.

Jerarquía de Clases



Para crear Servlets,
heredamos de
HttpServlet

Para comunicarnos con el cliente.

Response: La respuesta.

Request: La petición

**Serán los parámetros de los
métodos del Servlet.**

Servlet

- El paquete **javax.servlet** proporciona clases e interfaces para escribir servlets.
- La clase principal que implementa un servlet es **HttpServlet** implementa el interface **servlet**, de esta clase heredaremos para escribir servlets.
- Para interactuar con el cliente hay dos interfaces:
 - **HttpServletRequest**: que encapsula la comunicación desde el cliente al servidor.
 - **HttpServletResponse**: que encapsula la comunicación de vuelta desde el servlet hacia el cliente.

HttpServletRequest

- Hereda del interface ServletRequest.
- El Interface **ServletRequest** permite al servlet acceder a
 - Información como los nombres de los parámetros pasados por el cliente.
 - El protocolo (esquema) que está siendo utilizado por el cliente, y los nombres del host remoto que ha realizado la petición y la del server que la ha recibido.
- El stream de entrada, ServletInputStream. Los Servlets utilizan este stream para obtener los datos desde los clientes que utilizan protocolos como los métodos POST y PUT del HTTP.

HttpServletResponse

- Hereda del interface `ServletResponse`.
- El Interface **`ServletResponse`** le da al servlet los métodos para responder al cliente.
- Permite al servlet seleccionar la longitud del contenido y el tipo MIME de la respuesta.
- Proporciona un stream de salida, `ServletOutputStream` y un **`Writer`** a través del cual el servlet puede responder datos.

Contexto del Servlet

Interface ServletContext

- Representa el **contexto de ejecución** del Servlet dentro del Servidor Web.
- Permite **pasar información** entre los servlets que forman parte de la misma aplicación Web.
- Se pueden definir atributos, parámetros de inicialización.

Parámetros de inicialización

- Cuando definimos un Servlet podemos declarar una serie de parámetros, del tipo clave / valor.
- Se pueden utilizar para pasar valores constantes al Servlet.
 - Por ejemplo: El nombre de la clase que representa el driver de la BD.
 - El nombre de la BD.

Parámetros de inicialización

- Estos parámetros se declaran dentro de la declaración del Servlet en el descriptor de despliegue: **web.xml**.
- Desde el servlet los podemos recuperar.
- Solo son visibles en el Servlet que se declaran.
- **Métodos:**
 - String getInitParameter(String key)
 - Enumeration getInitParameterNames().



```
<servlet>
  <description></description>
  <display-name>Controlador</display-name>
  <servlet-name>Controlador</servlet-name>
  <servlet-class>control.Controlador</servlet-
    class>
  <init-param>
    <description></description>
    <param-name>param1</param-name>
    <param-value>valor1</param-value>
  </init-param>
  <init-param>
    <description></description>
    <param-name>param2</param-name>
    <param-value>valor2</param-value>
  </init-param>
</servlet>
```


Parámetros de inicialización

- Dentro del Servlet, podemos recuperar los parámetros:

// En este caso se imprimen por la consola del Server:

```
protected void doGet(HttpServletRequest request, HttpServletResponse  
    response) throws ServletException, IOException {
```

```
    System.out.println(this.getInitParameter("param1"));
```

```
    System.out.println(this.getInitParameter("param2"));
```

```
    Enumeration<String> e = this.getInitParameterNames();
```

```
    while (e.hasMoreElements())
```

```
        System.out.println(e.nextElement());
```

```
}
```

Atributos

- Todos los att con los que trabajamos estarían disponibles a nivel de aplicación.
- En las páginas JSP se corresponde con el objeto application.
- Almacenar, recuperar, eliminar parámetros:
 - void setAttribute(String nombre, Object valor) ;
 - Object getAttribute(String nombre):
 - Si recuperamos un att que no existe nos devuelve null.
 - Enumeration getAttributeNames();
 - void removeAttribute(String nombre);
- Los atributos que se establecen a este nivel son visibles por TODOS los servlets.

Tenemos 3 niveles

- Parámetros a nivel de **petición** (request):
`request.getAttribute("param1");`
- Atributos a nivel de **Sesión**:
`HttpSession session = request.getSession();`
`session.getAttribute("param1");`
- Atributos a nivel de **contexto / aplicación**:
`getServletContext().getAttribute("param1");`
- Parámetros a nivel de contexto:
`getServletContext().getInitParameter("nombre_param");`

Recursos

- Disponemos de dos métodos para cargar recursos desde un Servlet.
 - URL `getResource(String path)`
 - `InputStream getResourceAsStream(String path)`
 - A partir de este método se puede leer un fichero utilizando la clase `Scanner`.
 - Ejemplo:
 - `InputStream in = getServletContext().getResourceAsStream("nombres.txt");`
 - `Scanner s = new Scanner(in);`

Peticiones

Peticiones GET / POST

- La petición queda representada por **request**.
- La clase `HttpServlet` implementa las funcionalidades del protocolo HTTP.
- Por cada funcionalidad de HTTP tenemos el método equivalente en la clase `HttpServlet`.

Método http	Método HttpServlet
GET	<code>doGet()</code>
POST	<code>doPost()</code>

Parámetros en peticiones GET

- En esta petición los parámetros van en la URL. En pares nombre=valor, separados por un &.
- `url?param1=valor1¶m=valor2`
- Ejemplo en una página Web:
``

Parámetros en peticiones POST

- Se suelen utilizar para enviar datos en un formulario de HTML.
- No se ven en la URL.
- Ejemplo en una página:

```
<form name="miFormulario" method="post"  
  action="../servlet/ServletListar_libros">  
  Categoría: <input type="text" name="categoria">  
</form>
```

OJO si en el formulario NO se pone method, por defecto ira por GET.

Parámetros del protocolo http

- **ServletRequest:**

- Todos los parámetros que vienen de un formulario o de un enlace los podemos capturar a partir de los siguientes métodos.
 - `getParameter(String clave)`: Devuelve el valor.
 - `getParameterNames()`: `Enumeration<String>`.
 - `getParameterValues()`: Devuelve una matriz de `String` de todos los valores asociados al nombre.
 - `getParameterMap ()`: nombres y valores en un `Map`.
- Tenemos que utilizar los nombres que hayamos puesto en las etiquetas HTML.
- O en los parámetros de los enlaces.

Atributos

- Los atributos son objetos asociados a una petición. Se utilizan para pasar información de un Servlet a otro (RequestDispatcher).
- Son pares de clave / valor. El valor puede ser un objeto y la clave un String.
 - `getAttribute(String)`: Devuelve Object. Utilizar un casting para recuperar.
 - `getAttributeNames()`
 - `setAttribute(String, Object)`
- Se cargan a partir de la request (interface `ServletRequest`).

Cabeceras

- Desde la interface `HttpServletRequest` podemos acceder a las cabeceras HTTP.
 - `String getHeader(String key)`: Valor del campo de la cabecera.
 - `Enumeration<String> getHeaders()`: Devuelve las claves de las cabeceras.
 - Ejemplo:

```
Enumeration<String> cabs = request.getHeaderNames();

while (cabs.hasMoreElements()){
String cab = cabs.nextElement();
System.out.println(cab + " " + request.getHeader(cab));
}
```

Cabeceras

- **Clave / Valor de las cabeceras:**
- **accept**
 - image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, application/x-shockwave-flash, application/x-ms-application, application/x-ms-xbap, application/vnd.ms-xpsdocument, application/xaml+xml, application/x-silverlight, application/vnd.ms-excel, application/vnd.ms-powerpoint, application/msword, */*
- **referer**
 - http://localhost:8080/ServletVotaciones/index.html
- **accept-language**
 - es,en-US;q=0.5
- **ua-cpu**
 - x86
- **accept-encoding**
 - gzip, deflate
- **user-agent**
 - Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 5.1; .NET CLR 1.1.4322; .NET CLR 2.0.50727; .NET CLR 3.0.4506.2152; .NET CLR 3.5.30729; InfoPath.1; InfoPath.2; .NET4.0C; .NET4.0E)
- **host**
 - localhost:8080
- **connection**
 - Keep-Alive

Elementos del path de la Petición

- Desglose de la URL de la petición:
- <http://localhost:8080/ServletVotaciones/Controlador?op=1&nombre=juan>
 - Protocolo: http.
 - Servidor: localhost.
 - Puerto: 8080
 - Contexto Aplicación: ServletVotaciones
 - Recurso: Controlador.
 - Parámetros: op=1 nombre=juan
 - QueryString: [op=1&nombre=juan](#)

Consideraciones del protocolo Http

- El protocolo http, es un protocolo basado en petición – respuesta (request / response).
- No almacena ningún valor sobre las peticiones anteriores.
- De alguna forma tenemos que identificar a un cliente.
- Disponemos de tres mecanismos:
 - Cookies, campos ocultos, sesiones.

cookies

- Son ficheros que almacena un servidor en el cliente con datos de la sesión.
- El cliente acepta / rechaza las cookies, se configura en el navegador.
- Normalmente se almacena el nombre y un valor.
- Cuando el cliente hace una petición al mismo recurso se envía automáticamente esa cookie.

Manejo de cookies

- La cookie hereda de **javax.servlet.http.Cookie**.

Al crearla se define el nombre y el valor, aunque se puede modificar después de crearlo mediante **setName()** y **setValue()**, **getName()**, **getValue()**.

Cookie cookie = **new Cookie**("nombre", "valor");

- Si no se especifica la duración de la cookie, su duración es la **duración de la sesión**, es decir, se elimina cuando termina de usar el navegador.
- En el siguiente ejemplo la duración es de un año:

```
int SECONDS_PER_YEAR = 60*60*24*365;  
Cookie c = new Cookie( "nombre", "valor" );  
c.setMaxAge(SECONDS_PER_YEAR);
```


Manejo de cookies

- Una vez creada, **se envía al cliente:**
`response.addCookie(cookie);`

// Ejemplo para buscar una cookie en el cliente.

```
public static Cookie getCookie( HttpServletRequest request, String
    cookieName ) {
    Cookie[] cookies = request.getCookies();
    for(int i=0; cookies != null && i < cookies.length; i++) {
        Cookie cookie = cookies[i];
        if (cookieName.equals(cookie.getName()))
            return cookie;
    }
    return null;
}
```

Cookies

- Cuando queramos **enviar** alguna **cookie** al cliente, hay que hacerlo **ANTES** de mandar cualquier cabecera HTML.
- Las cookies dependen de cómo tenga configurado el Navegador el cliente.
 - No es una buena forma de identificar a los clientes → Mejor sesiones.

Otra información

- Además podemos obtener la siguiente información del cliente:
- **SSL:**
 - boolean isSecure():
 - Devuelve true si la petición es Https.
- **i18n:**
 - Locale getLocale():
 - Devuelve el idioma por defecto. Esto se extrae de la cabecera de la petición (accept-language).
 - Enumeration<Locale> getLocales():
 - Una enumeración con la preferencia de idiomas.
- **Encoding:**
 - String getEncodingCharacter():
 - Por defecto la codificación es ISO-8859-1.

Ámbito

- Tener en cuenta que el **ámbito** del objeto **request** es **sólo** para la **petición en curso**.
- Si volvemos a hacer otra petición el objeto request se destruye y se crea uno nuevo.
- Si queremos almacenar información entre una petición y otra NO nos vale con almacenarla en la request.
 - Tendremos que utilizar otros mecanismos: cookies, sesiones, campos ocultos.

Respuestas

Respuesta

- La respuesta que un Servlet devuelve al cliente queda representada por el objeto **Response**.
- Se puede añadir información en las cabeceras y el cuerpo del mensaje.
- El mensaje normalmente se devolverá en HTML. Aunque se soportan otros formatos.

Objeto Response

- Interface **ServletResponse**:
- A través de este objeto volcamos información al cliente.
`PrintWriter out = response.getWriter();`
`out.print("<p>Contenido HTML</p>");`
- Indicamos el tipo de información que le vamos a enviar:
tipo MIME:
 - text / html → HTML.
 - text / plain → Texto plano.`response.setContentType("text/html");`

Métodos: Response

- **sendRedirect:**
 - Con este método podemos redirigir a otro recurso.
 - `response.sendRedirect("index.html");`
 - La URL del navegador se modifica con la nueva dirección.
- **sendError:**
 - `response.sendError(int):`
 - Indica el código de error.
 - Enviar un error al cliente.

Otros métodos

- **Buffering:** Representa el buffer de respuesta al cliente.
 - `int getBufferSize():` Preguntar por el tamaño del buffer.
 - `void setBufferSize(int):` Establecer el tamaño del buffer.
 - `flushBuffer():` Fuerza a que el contenido del buffer sea escrito en el cliente.
 - **Normalmente utilizaremos `PrintWriter` para escribir en el cliente.**
- **Cabeceras:** de respuesta al cliente:
 - `setHeader(String nombre, String valor):` Modificar un valor de un campo de la cabecera.
 - `addHeader(String nombre, String valor):` Añade una nueva cabecera.

Otros métodos

- **i18n:**

- Un servlet puede establecer la respuesta con `setLocale` cuando el cliente solicita un documento en un lenguaje en particular.
- El Locale va asociado a una codificación en especial. Se especificaría dentro del descriptor de despliegue de la aplicación.

```
<locale-encoding-mapping-list>  
  <locale-encoding-mapping>  
    <locale>ja</locale>  
    <encoding>ISO-2022-JP</encoding>  
  </locale-encoding-mapping>  
</locale-encoding-mapping-list>
```

- `response.setLocale(Locale l)`:
 - Se debe de lanzar antes de que el Servlet capture el Writer para escribir en el cliente.

Ámbito

- El ámbito del objeto Response se sólo es válido para la **petición en curso**.
- Para otra petición al igual que ocurre con el objeto request el contenedor de Servlets (el servidor) destruye los objetos request y response y los vuelve a crear.

Gestión de Sesiones

¿Qué son?

- Hay que tener en cuenta que el protocolo HTTP es del tipo petición / respuesta.
- No almacena ningún valor.
- Las sesiones nos permiten mantener identificado a un usuario de nuestra aplicación de tal forma que todas las peticiones que realice el usuario formen parte de la misma sesión.

Sesiones

- Por ejemplo:
 - Un usuario se conecta a nuestra aplicación, se identifica y a partir de aquí puede realizar una serie de compras en una tienda virtual.
 - En las diferentes peticiones que realiza el usuario tengo que mantener información de este.
 - Por ejemplo:
 - El carrito de la compra, con los productos que ha seleccionado.
 - Después va a formalizar el pedido.

Sesiones

- Dentro de la Sesion, podemos almacenar objetos, variables, etc.
- Tener en cuenta que estas variables se mantienen en el Servidor.
- Cuando el usuario cierra el navegador, o le damos la posibilidad de cerrar la sesión el Servidor libera esa información.

Sesiones

- Normalmente hay un **tiempo de expiración** para la sesión (suelen ser 20' o menos), si el usuario no realiza ninguna petición la sesión se invalida.
- Tener en cuenta que a la aplicación se pueden conectar múltiples usuarios, si estamos almacenando mucha información en la sesión podemos **sobrecargar el Servidor** con los consiguientes fallos.

Sesiones



IDENTIFICACIÓN

USUARIO:

PASSWORD:



El usuario teclea la identificación.
El formulario se envía al **ServletValidar**.

El ServletValidar, comprueba los datos en una BD.
Si es correcto: crea la sesión y redirige al menú principal y ya tiene registrado al usuario.

Si no existe: Mostraremos una página de error.

Con la sesión: También podemos controlar que el usuario se ha identificado antes de acceder al menú principal. **¿Qué pasa si conoce la URL del menú principal y la teclea directamente en el Navegador? → Tenemos que obligarle a que pase por el Form de identificación.**

Seguimiento de las Sesiones

- Disponemos del objeto:
 - `javax.servlet.http.HttpSession`
 - Este objeto lo podemos obtener a partir del objeto `HttpServletRequest`.
- Ejemplo:

```
HttpSession sesion = request.getSession(true);  
// Devuelve la session actual, si no existe se  
  crea una.
```

Métodos HttpSession

- Almacenar, recuperar, eliminar parámetros:
 - void setAttribute(String nombre, Object valor) ;
 - Object getAttribute(String nombre):
 - Si recuperamos un att que no existe nos devuelve null.
 - Enumeration getAttributeNames();
 - void removeAttribute(String nombre);
- Mas métodos:
 - isNew(): ¿Es nueva la sesión?
 - invalidate(): Invalidar la sesión.
 - long getCreationTime(): los milisegundos que han pasado desde la creación.
 - long getLastAccessedTime(): Cuando se accedió por última vez.

Ejemplo

- Cargar objetos en la session:
 - `sesion = getRequest().getSession(true);`
 - `sesion.setAttribute("user", usuarioSesion);`
 - `sesion.setAttribute("conectado", true);`
- **getSession(true)**: Recupera la sesión, si no está creada la crea.
- Cerrar la session:
 - `request.getSession(true).invalidate();`

Enviar peticiones

El objeto RequestDispatcher

- Con este objeto vamos a poder reenviar solicitudes a otros recursos, por ejemplo: a otro Servlet o un JSP.
- El objeto RequestDispatcher lo obtenemos de la petición a través del método `getRequestDispatcher("otro_recurso")`.
 - `request.getRequestDispatcher("otro_recurso")`
- Antes de redirigirnos a otro recurso podemos pasar algún parámetro por la request.
- Una vez que tenemos el recurso podemos hacer uso de dos métodos: **include** y **forward**.

Métodos

- La invocación al recurso se puede hacer de dos formas:
- **Incluir (include)** el recurso en el flujo de salida. La salida de `"/servlet/request_imagen"` se incluye en la salida del primer servlet.
 - `dispatcher.include(request, response);`
- **Redirigir (forward)** la petición al recurso. Funcionalmente semejante a `sendRedirect()`. Se trata de **redirigir la petición** a otro componente:
 - `dispatcher.forward(request, response);`
- La **diferencia entre `sendRedirect()` y `forward()` (o también `include()`)** es:
 - En `sendRedirect()` “del objeto Response” la petición acaba bajo el control del segundo servlet. La URL que se puede ver en el navegador es la del segundo servlet.
 - En `forward()` o `include()` la petición se controla por el primer servlet.
 - **La URL que se puede ver en el navegador es la del primer servlet.**


RequestDispatcher

- Podemos cargar un recurso Web en el navegador.
- Podremos cargar:
 - Otro Servlet.
 - Una página JSP.
 - Una página HTML.

Ejemplo

- En el método doGet de un Servlet:

```
public void doGet(HttpServletRequest request,  
                  HttpServletResponse response){  
    request.setAttribute("param1", "value1");  
    RequestDispatcher rq =  
    request.getRequestDispatcher("Servlet2");  
    rq.forward(request, response);  
}
```



Vamos a enviar una solicitud al
Servlet2

Aplicaciones Web

Aplicaciones Web

- Las aplicaciones Web se componen de una determina organización de carpetas y recursos.
- Los recursos pueden ser:
 - Estáticos:
 - HTMLs, Imagen, iconos, etc.
 - Dinámico:
 - Servlets, JSPs.
 - Clases Java.
 - Ficheros de configuración: XML.
 - Ficheros JAR.

Directorios de la App Web

- **MiApp**
 - META-INF (opcional) *Fichero de manifiesto.*
 - WEB-INF
 - classes (ficheros *.class)
 - empresa
 - » Persona.class
 - » OtraClase.class
 - lib (*librerías auxiliares de mi app*)
 - *.jar*(Descriptores de despliegue de CAPA WEB)*
 - web.xml → *estándar*
 - *.jsp
 - *.html *(Estos html, jsp, etc. Se pueden organizar en DIR)*
(Otros contenidos Web, imagen, iconos, ...)

CON ECLIPSE SE EMPAQUETA UN FICHERO WAR, OPCIÓN EXPORT:
MiApp.war

Ubicación en el servidor

- Tomcat (carpetas del Servidor):
 - bin
 - lib
 - Ficheros JAR compartidos por todas las aplicaciones desplegadas en el Servidor.
 - conectorMySQL.jar
 - websapp
 - **MiApp**
 - MiApp2
 - MiApp3
 - Etc.

Aplicaciones Web y Web.xml

- WEB-INF\web.xml: Es el descriptor de la aplicación. Siempre almacena información de configuración.
- WEB-INF\classes: Contiene las clases compiladas, los .class. En Eclipse se almacena en build\classes.
- WEB-INF\lib: Ficheros jar, clases adicionales que necesite nuestra aplicación.

El descriptor: web.xml

- Empieza con el nodo principal **web-app**. El descriptor indica toda la información relevante de la configuración de la aplicación.
- El nombre de la aplicación: *Normalmente el del proyecto.*
`<display-name>ServletVerInfoPeticion</display-name>`
- Por cada Servlet:
`<servlet>`
 `<description></description>`
 `<display-name>VerInfoPeticion</display-name>`
 `<servlet-name>VerInfoPeticion</servlet-name>`
 `<servlet-class>controlador.VerInfoPeticion</servlet-class>`
`</servlet>` *La clase indica paquete.clase*

El descriptor: web.xml 2

- También por cada Servlet tendremos la cadena por la que se mapea, como le vamos a llamar:

```
<servlet-mapping>  
  <servlet-name>VerInfoPeticion</servlet-name>  
  <url-pattern>/VerInfoPeticion</url-pattern>  
</servlet-mapping>
```

Desde el action de un form podríamos llamar:
action="VerInfoPeticion"

- También se pueden mapear como: ***.do**, y luego haremos peticiones del estilo: validar.do, crear.do, borrar.do, etc.

El descriptor: web.xml 3

- Ficheros de bienvenida al sitio:

```
<welcome-file-list>
```

```
  <welcome-file>index.html</welcome-file>
```

```
  <welcome-file>index.htm</welcome-file>
```

```
  <welcome-file>index.jsp</welcome-file>
```

```
  <welcome-file>default.html</welcome-file>
```

```
  <welcome-file>default.htm</welcome-file>
```

```
  <welcome-file>default.jsp</welcome-file>
```

```
</welcome-file-list>
```

Indican el orden en el que se van a buscar.

El descriptor: web.xml 4

- También podemos indicar parámetros de inicialización:
- Que luego pueden ser recuperados en los métodos del Servlet.

```
<init-param>  
  <description></description>  
  <param-name>param1</param-name>  
  <param-value>valor1</param-value>  
</init-param>
```

El descriptor: web.xml 5

- Configuración de la Session:

```
<session-config>  
    <session-timeout>30</session-timeout>  
</session-config>
```

- Parámetros del contexto: Son a nivel global, no exclusivos de cada Servlet:

```
<context-param>  
    <param-name>some-port</param-name>  
    <param-value>5000</param-value>  
</context-param>
```

Distribución de Peticiones al Servlet

Distribución de Peticiones

- Dentro del fichero web.xml se definen las URLs a las que atiende cada servlet.
- Mediante url-pattern.

```
<servlet>
  <display-name>Contralador</display-name>
  <servlet-name>Contralador</servlet-name>
  <servlet-class>controlador.Contralador</servlet-class>
</servlet>
<servlet-mapping>
  <servlet-name>Contralador</servlet-name>
  <url-pattern>/Contralador</url-pattern>
</servlet-mapping>
</servlet>
```

Distribución de Peticiones

- Mediante el patrón de diseño **MVC**, centralizaremos las llamadas al mismo servlet, se le pueden pasar parámetros en la misma petición:
 - **Controlador?op=1, Controlador?op=2 ...**
 - También se puede hacer como lo hace **Struts**:
Coloca el patrón a ***.do**
 - Todo lo que termine do, lo recibe el Servlet.
 - Grabar.do, Borrar.do ...
 - Como **Struts2**: ***.action.**

Paquetes:
servlet / servlet.http

Métodos de HttpServlet

- Es una clase abstracta que implementa el interface Servlet.
- Está será nuestra clase base para implementar Servlet.
- Paquete: `javax.servlet.http.*`,
- void **doGet**(HttpServletRequest peticion, HttpServletResponse respuesta)
 - La petición se realiza con parámetros tipo GET.
- void **doPut**(HttpServletRequest peticion, HttpServletResponse respuesta)
 - La petición se realiza con el método POST.
- void **service**(HttpServletRequest peticion, HttpServletResponse respuesta)
 - Cuando no sabemos si la petición viene por GET o POST.

Métodos de HttpServlet II

- Estos métodos no se suelen utilizar pero también proporciona:
 - doHead():
 - procesa solicitudes HTTP HEAD.
 - Ejecuta el método doGet() y devuelve al cliente solo lo generado en las cabeceras.
 - doOptions(): procesa solicitudes HTTP OPTIONS.
 - doTrace(): procesa solicitudes HTTP TRACE.
 - doPut(): procesa solicitudes HTTP PUT.
 - doDelete(): procesa solicitudes HTTP DELETE.

Métodos de GenericServlet

- Por la jerarquía de herencia también tenemos acceso a los siguientes métodos:
 - `init()`: Se ejecutaría al crear el Servlet. Sólo se ejecuta una vez.
 - `destroy()`: Igual pero al destruir el Servlet. Se puede utilizar para liberar recursos o cerrar conexiones. Sólo se ejecuta una vez.
 - `String getInitParameter(String name)`: Capturar un parámetro inicial. Estos parámetros son por cada Servlet.
 - `Enumeration getInitParameterNames()`: Capturar todos los parámetros de inicialización. Estos parámetros son por cada Servlet.
 - `ServletConfig getServletConfig()`: Configuración del Servlet. Por ejemplo, los parámetros iniciales antes descritos, podemos acceder a través de los métodos directamente o través de este objeto.
 - `ServletContext getServletContext()`: Representa el contexto del Servlet. Se pueden definir parámetros en el `web.xml` y que serán luego recuperados. La diferencia con los parámetros anteriores es que estos son a nivel de la aplicación no exclusivos de cada servlet como los anteriores.
 - `String getServletInfo()`: Información del Servlet.
 - `String getServletName()`: El nombre del Servlet.

Métodos de ServletRequest

- **Paquete:** `javax.servlet.*`;
- `String getParameter(String nombre)` → Recupera el valor de un parámetro.
- `Enumeration getParameterNames()` → Recupera todos los nombres de los parámetros.
- `String getProtocol()` → El nombre del protocolo.
- `String getRemoteHost()` → El nombre del HOST.
- `String getScheme()` → Retorna el nombre del esquema: http / ftp / https
- `String getServerName()` → Devuelve el nombre del servidor.
- `int getServerPort()` → Devuelve el número del puerto.
- `String getContentType()` → Devuelve el tipo MIME. Ejemplo text/html
- `int getContentLength()` → Devuelve la longitud de la petición, -1 si es desconocida.

Métodos de HttpServletRequest 2

- Almacenar, recuperar, eliminar parámetros:
 - void setAttribute(String nombre, Object valor) ;
 - Object getAttribute(String nombre):
 - Si recuperamos un att que no existe nos devuelve null.
 - Enumeration getAttributeNames();
 - void removeAttribute(String nombre);

Métodos de HttpServletRequest

- **Paquete:** `javax.servlet.http.*`;
- `Cookies[] getCookies()` → Devuelve un array con las cookies del cliente.
- `String getMethod()` → Devuelve si el método de petición fue GET / POST.
- `String getQueryString()` → La cadena de parámetros que va en la URL. Para peticiones tipo Get.
- `String getServletPath()` → Devuelve la parte de URL que llamó al servlet.
 - Para esta url: http://localhost:8082/Servlet_Prueba/Ejemplo
 - Devuelve: /Ejemplo
- `String getContextPath()` → Devuelve el path del contexto de la aplicación.
 - Para esta url: http://localhost:8082/Servlet_Prueba/Ejemplo
 - Devuelve: /Servlet_Prueba

Métodos de HttpServletResponse

- **Paquete: javax.servlet.*;**
- `PrintWriter getWriter()` → Devuelve un objeto `PrintWriter` que usaremos para escribir al cliente la respuesta, es decir, los tags de HTML.
 - *PrintWriter* está en *java.io.**;
- `void setContentType(String tipo)` → Establece el tipo MIME con el que vamos a escribir al cliente.
Ejemplo: `text/html`

Métodos de HttpServletResponse

- **Paquete: `javax.servlet.http.*`;**
- `void addCookie(Cookie c)` → Añade una cookie al cliente.
- `void sendRedirect(String location)` → Para redirigir a otra página. Por ejemplo a una página de error.

PRACTICAS: SERVLETS

Apéndice

- Conviene tener presente la lista de los códigos devueltos por el protocolo HTTP.
- Ejemplo:
 - 404** → Recurso no encontrado.
 - 500** → Error interno dentro del Servidor.
 - 200** → Respuesta OK.
- [http://es.wikipedia.org/wiki/Anexo:C%C3%B3digos de estado HTTP](http://es.wikipedia.org/wiki/Anexo:C%C3%B3digos_de_estado_HTTP)