

DESARROLLO DE SERVICIOS TELEMÁTICOS

Tema 2

3er. curso

Graduado en Informática

Mención de Tecnologías de la Información

Autora: Lidia Fuentes



Lidia Fuentes Fernández

Desarrollo de servicios web

- Clases java para aplicaciones Web
- Estructura y componentes de un sitio Web
- Los servlets y otras tecnologías
- Páginas JSP (y otras tecnologías similares)
- Otras tecnologías Web avanzadas



2.1. Clases Java para aplicaciones Web

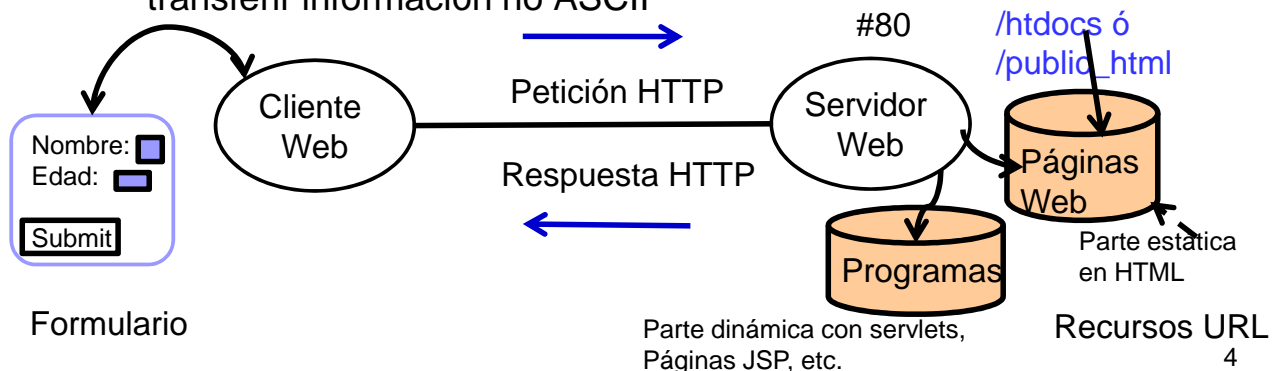
- El protocolo HTTP
- La URL
- Clases Java para la conexión con recursos Web

3



El protocolo HTTP

- Permite acceder a recursos almacenados en servidores web desde un navegador
 - Funciona sobre TCP (puerto 80 por defecto)
 - Modelo petición/respuesta
 - No tiene estado (*stateless*)
 - Al igual que SMTP, es un protocolo ASCII, que usa MIME para transferir información no ASCII



4



La URL

- Identificador de recursos URL (Universal Resource Locator)
 - <protocolo>://<nombre-serv-web[:<puerto>]/<path-recurso>**
 - <protocolo>:: <http>|<ftp>|<rtp>| ...**
 - <path-recurso>::<absolutos>|<relativos>**
- URL absoluta
 - ☐ <http://www.lcc.uma.es>, <http://www.abx.uma.es:8080/Inicio.html>
 - ☐ Normalmente utilizadas en la barra de navegadores
 - ☐ Usadas por servicios web al hacer referencia a otros servidores
- Path absoluto
 - ☐ /MyWeb/MainPage.jsp
 - ☐ Útil cuando se quiere hacer un enlace a una página dentro del mismo servidor, y no se prevé cambiarla de directorio
- Path relativo
 - ☐ Se utilizan para hacer un enlace a una página dentro del mismo servidor, ej: ../MainPage.jsp

5



El protocolo HTTP

- Peticiones HTTP (del cliente al servidor)

Sintaxis	Ejemplo
<método> <recurso (URL)> <versión HTTP> <crlf> [<head>: <valor> <crlf>] : <crlf> [cuerpo]	GET /path/fichero.html HTTP/1.0 User-Agent: Mozilla/4.05 [en] Accept: text/html

- <una línea de petición>
- <0 o más líneas de cabecera>
- <CRLF>
- [<cuerpo de la petición>]

6



Peticiones HTTP

Método	HTTP/1.0	HTTP/1.1	Descripción
GET	✓	✓	Recupera el URL especificado
HEAD	✓	✓	Idéntico a GET, excepto que el servidor no retorna el documento en respuesta; Sólo retorna los headers de respuesta. Los clientes lo utilizan para obtener "meta-datos" del recurso o para saber qué links son válidos.
POST	✓	✓	Envía datos a la URL especificada.
PUT		✓	Guarda estos datos en el URL especificado, re-escribiéndolo.
PATCH		✓	Similar a PUT, excepto que contiene una lista de diferencias entre la versión original del recurso y el contenido deseado.
COPY		✓	Copia el contenido del recurso a la(s) dirección(es) especificada(s).
MOVE		✓	Mueve el contenido del URL a otra(s) posición(es).
DELETE		✓	Borra el recurso identificado por el URL.
LINK		✓	Establece una o más relaciones de ligadura (<i>link</i>) entre el URL y otros recursos.
UNLINK		✓	Elimina la relación existente entre este URL y otro(s).
TRACE		✓	Responde, en el cuerpo del mensaje, la petición del cliente.
OPTIONS		✓	Pide información acerca de las opciones de comunicación disponibles para el URL especificado. Permite a un cliente obtener las características de un servidor sin recuperar ningún URL.
WRAPPED		✓	Permite que las peticiones se agrupen y, posiblemente se encripten para aumentar la seguridad de las mismas.

Tabla 3.3: Métodos HTTP

7



Respuestas HTTP

■ Formato

<versión HTTP><código de respuesta><descripción><CRLF>
 [<cabeceras>]
 CRLF
 [<cuerpo>]

■ En el cuerpo se transfiere el contenido de la URL solicitada (ej: página HTML)

8



Ejemplo HTTP

GET /index.html HTTP/1.1
Host: www.lcc.uma.es

Petición HTTP

Respuesta HTTP

HTTP/1.1 200 OK

Primera línea de respuesta

Date: Mon, 29 Oct 2012 15:42:52 GMT

Server: Apache/2.2.17 (EL)

Transfer-Encoding: chunked

Accept-Ranges: bytes

Content-Length: 3498

Content-Type: text/html

Cabeceras

línea en blanco

<html>

Contenido página HTML

<title>Bienvenido a la web de la asignatura DST</title>

... </html>

9



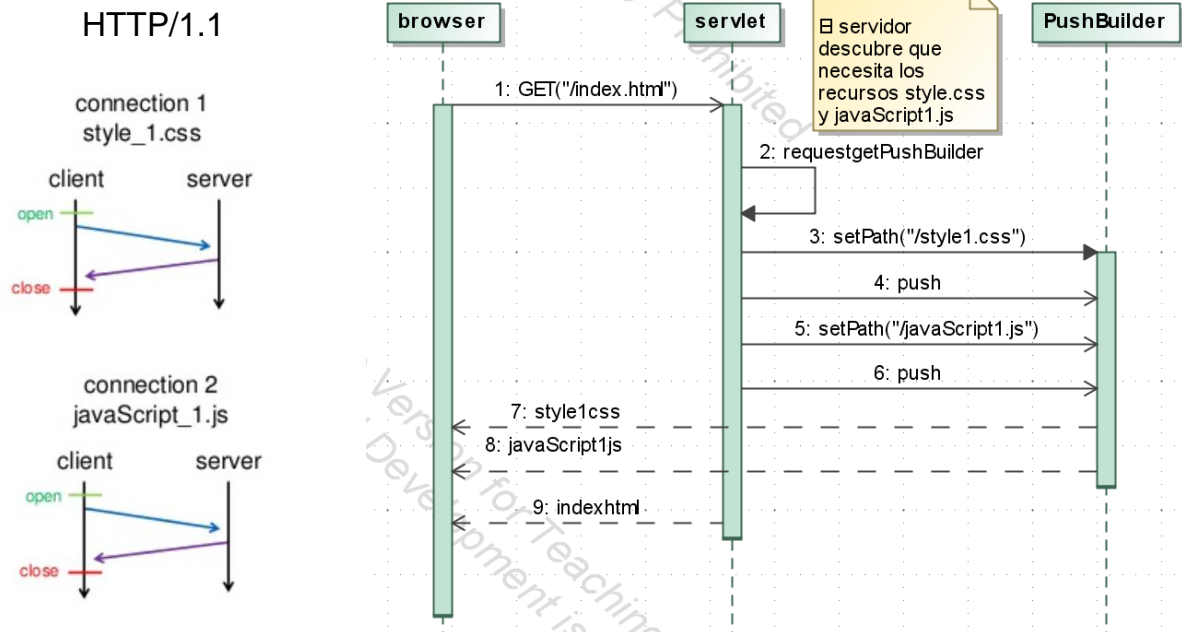
HTTP/2

- RFC 7540 (Mayo 2015)
 - Multiplexión req/resp
 - Codificación de datos en streams a nivel binario
 - Compresión de las cabeceras
 - Asignación de prioridad a cada stream
 - El servidor utilizar el mecanismo push para devolver los recursos de cada página solicitada
 - Cada recurso en un stream
 - Utiliza solamente una conexión TCP para todos los recursos de una página (ej: hojas de estilo, código javascript, imágenes, etc.)
 - Servlet 4.0 implementado por apache

10



HTTP/2



11



Clases Java para la conexión con recursos Web

- Java proporciona clases que nos facilitan la construcción de clientes y servidores Web
- Clases que modelan el identificador de recursos
 - URL
 - URI
- Clases que modelan la conexión a un recurso
 - URLConnection
 - HttpURLConnection

12



Clase URL

■ clase URL

- Representa un URL

```
URL(String oneURL)
URL(String protocol,String host,String file)
URL(String protocol,String host,int port,String file)
String getProtocol()
int getPort()
String getHost()
String getFile()
    /* devuelve cada uno de los campos de la referencia */
String toExternalForm()
    /* me devuelve una cadena con la URL completa */
URLConnection.openConnection() throws IOException
    /* abre una conexión con el servidor http identificado
    por la URL */
```

13



Clase URLConnection

■ clase URLConnection

- Clase que representa una conexión TCP con un servidor de http identificado por una URL (por defecto la petición es de tipo GET)

```
String getContentType()
    /* devuelve el tipo de documento que me ha enviado el
    servidor http: ej: .txt, .gif, .jpg, etc. */
void setRequestProperty(String key, String value)
    /* Establece el par cabecera/valor de la petición HTTP
    setRequestProperty("Content-Type","application/octet-
    stream"); */
Object getContent()
    /* devuelve el contenido completo del documento.
    ej: index.html, foto.gif, etc. */
InputStream getInputStream() throws IOException
    /* devuelve el descriptor de lectura */
OutputStream getOutputStream() throws IOException
    /* devuelve el descriptor de escritura */
```

14



```

import java.net.*;
class ClienteWWW {
    public static void main(String args[]) throws Exception {
        int c;
        if (args.length != 2) {
            throw new IllegalArgumentException("Parameter(s): <URL> <FileName>");
        }
        try {
            URL url = new URL(args[0]);
            URLConnection s = url.openConnection();

            InputStream in = s.getInputStream();

            FileOutputStream out = new FileOutputStream(args[1]);

            System.out.println(url);
            System.out.println(s.getContentType());
            System.out.println(s.getContentLength()+" bytes");

            while ((c=in.read()) != -1) {
                out.write(c);
            }
        } catch (UnknownHostException e){
            ...
        }
    }
}

```

sol10% java ClienteWWW http://www.gisum.uma.es index.html

http://www.gisum.uma.es/
text/html
15378 bytes

```

sol10% more index.html
<html>
<head>
  <title>GISUM-Grupo de Investigación de Ingeniería del Software
de la
      Universidad de Málaga </title>
  <meta http-equiv="content-type" content="text/html; charset=utf-8" />
  <link href="/stylesheets/estilo.css?1378817959" media="all"
rel="Stylesheet"
      title="Estilo" type="text/css" />
</head>
...

```



Clase HttpURLConnection

■ clase HttpURLConnection extends URLConnection

- Clase URLConnection que permite realizar configuraciones específicas sobre HTTP

■ Define constantes para los códigos de respuesta y mensajes HTTP

```

/* Status codes */
HttpURLConnection.HTTP_OK;
// HTTP Status-Code 200: OK
HttpURLConnection.HTTP_NOT_FOUND;
// HTTP Status_Code 404: Not Found
protected String method;
// El método HTTP (GET,POST,PUT,etc.).
protected int responseCode;
// El entero que representa el código HTTP Ej: 404
protected String responseMessage ;
// El mensaje de respuesta HTTP Ej: Not Found

```




Clase HttpURLConnection

■ Métodos principales

```
protected HttpURLConnection(URL)
// Constructor
boolean usingProxy()
// Indica si la conexión es a través de un proxy
void setRequestMethod(String)
/* GET, HEAD, .... */
int get/setResponseCode()
// Obtiene/establece el código de respuesta HTTP Ej: 200
String getResponseMessage()
// Obtiene el mensaje de respuesta HTTP Ej: HTTP/1.1. 200 OK
String getHeaderField(int n)
// Devuelve el valor de la cabecera número "n"
...
```

17



Clase HttpURLConnection

■ Obtención de una HttpURLConnection (el constructor no es público)

```
URL url = new URL("http://www.lcc.uma.es");
URLConnection urlc = url.openConnection();

if (urlc instanceof HttpURLConnection) {
    HttpURLConnection httpurlc = (HttpURLConnection) urlc;
    /* ya puede accederse a los métodos de la clase
       HttpURLConnection */

    // Cambio la petición GET por defecto a POST
    httpurlc.setRequestMethod("POST");
    ...
}
```

18



```
import java.net.*;

public class HeadRequest {
    public static void main(String[] args) throws Exception{
        URL url = new URL("http://www.lcc.uma.es/index.html");

        try {

            URLConnection urlc = url.openConnection();
            if (urlc instanceof HttpURLConnection) {
                HttpURLConnection httpurlc = (HttpURLConnection) urlc;
                httpurlc.setRequestMethod("HEAD");
                System.out.println("Request method: "+httpurlc.getRequestMethod());
                System.out.println("Response code: "+httpurlc.getResponseCode());
                System.out.println("HTTP Header 1: "+httpurlc.getHeaderField(1));
                System.out.println("Proxy? "+httpurlc.isUsingProxy());
            }
        }
        catch (UnknownHostException e) {
            System.out.println("non existant URL or no answer")
        }
    }
}
```

SALIDA:

Request method: HEAD
Response code: 200
HTTP Header 1: Apache-Coyote/1.1
Proxy? false

19



2.2. Estructura y componentes de un sitio Web

■ 2.2.1. Sitios Web dinámicos

- ☐ CGI
- ☐ FastCGI
- ☐ API Server Extensions

■ 2.2.2. Procesamiento de formularios en sitios Web dinámicos

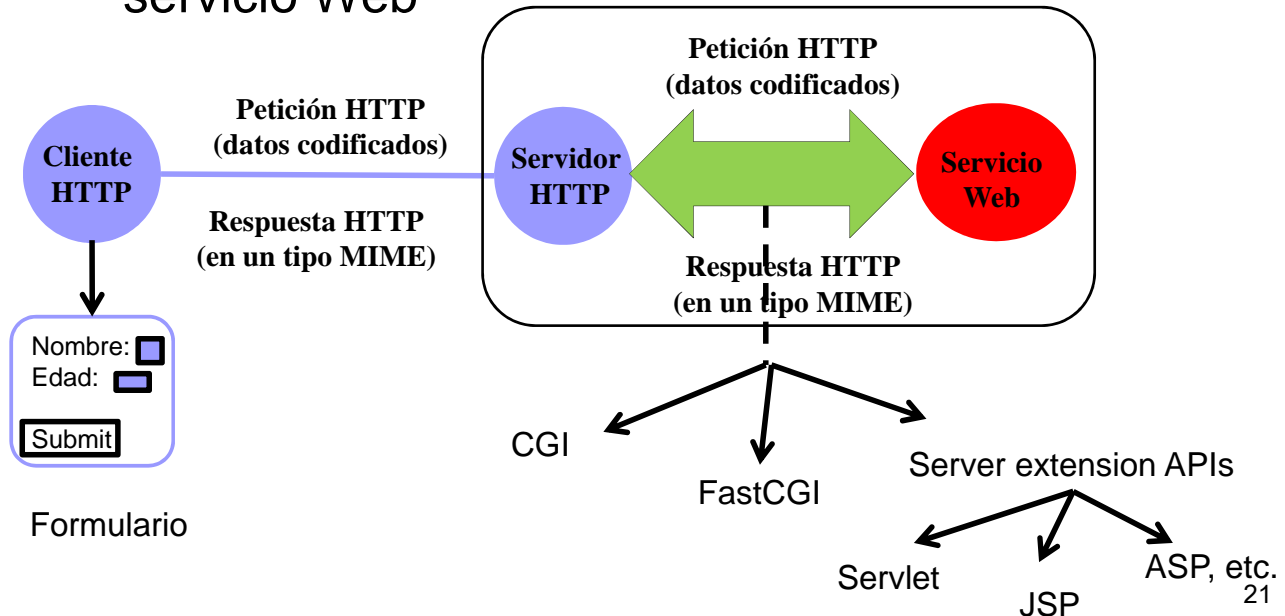
- ☐ Formularios HTML
- ☐ Codificación de datos en peticiones HTTP
- ☐ Variables de entorno
- ☐ Diagrama de flujo de un sitio Web dinámico

20



2.2.1. Sitios Web dinámicos

■ Mecanismos de interacción servidor Web – servicio Web



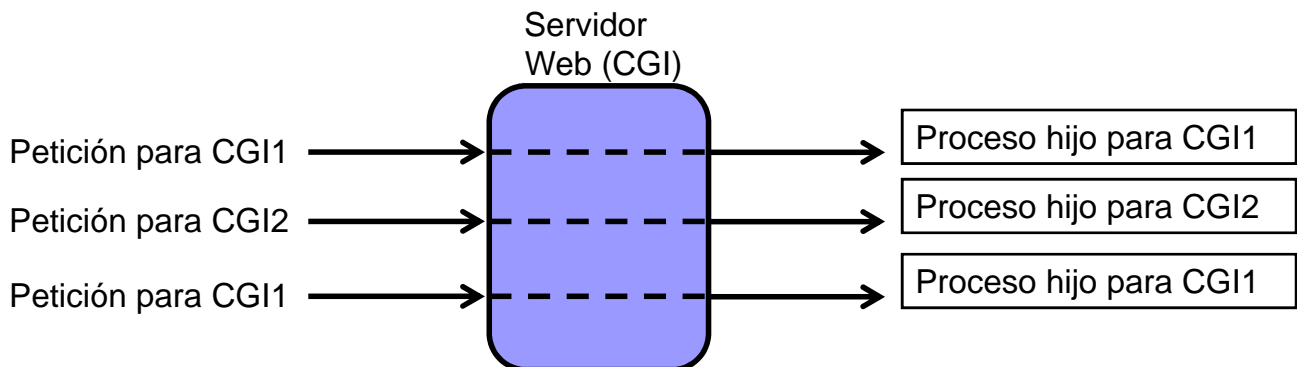
Programación CGI

■ CGI (Common Gateway Interface)

- Es una de las primeras técnicas para generar contenido Web dinámico
- Define un mecanismo de comunicación entre el servidor web y una aplicación externa cuyo resultado está codificado según MIME
- Lenguajes de programación: C/C++, Perl, Tcl, Java, scripts sistema operativo (C Shell, Visual Basic, ...)



Ciclo de vida CGI



- Se crea un proceso hijo por cada petición
- Procesos hijos, independientes del proceso servidor Web

23



Desventajas de la programación CGI

- Dependiente del sistema operativo
- Se carga en memoria tras cada petición
 - Bajo rendimiento
- No puede acceder a la memoria del servidor (ej: no puede escribir en el fichero de log del servidor)
- Se utilizan algunos lenguajes tipo *script* (Unix Shell, Perl, Tcl, ...)
- Normalmente no disponen de técnicas de comprobación dinámica de errores
- Mezcla código de funcionalidad con la generación de la página de respuesta
 - Código difícil de extender

24



Programación FastCGI

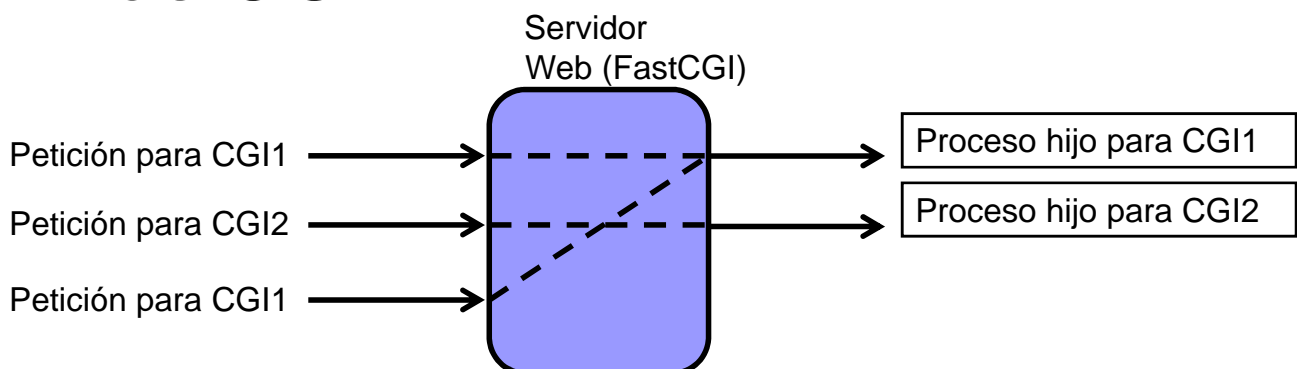
■ FastCGI (de OpenMarket)

- Su objetivo era mejorar el rendimiento del mecanismo CGI
- Puede implementarse en cualquier lenguaje que soporte sockets
 - Lenguajes de programación: PHP, C/C++, Perl, Python, Tcl, Ada, Haskell, Java, scripts sistema operativo (C Shell, Visual Basic, ...)
- Servidores web con FastCGI
 - Apache, Cherokee HTTP Server, Abyss Web Server, Microsoft IIS, Sun Java System Web Server

25



Ciclo de vida FastCGI



- Se crea un proceso hijo por cada servicio
- Procesos hijos, independientes del proceso servidor Web

26



Características de la programación FastCGI

- Dependiente del sistema operativo
- No puede acceder a la memoria del servidor (ej: no puede escribir en el fichero de log del servidor)
- Mezcla código de funcionalidad con la generación de la página de respuesta
 - Código difícil de extender
- Buen rendimiento
 - Si hay muchos servicios, puede distribuir la carga entre varios servidores
 - El rendimiento dependerá del lenguaje de programación (ej: en Perl, tendría que cargar un intérprete Perl por cada servicio)

27



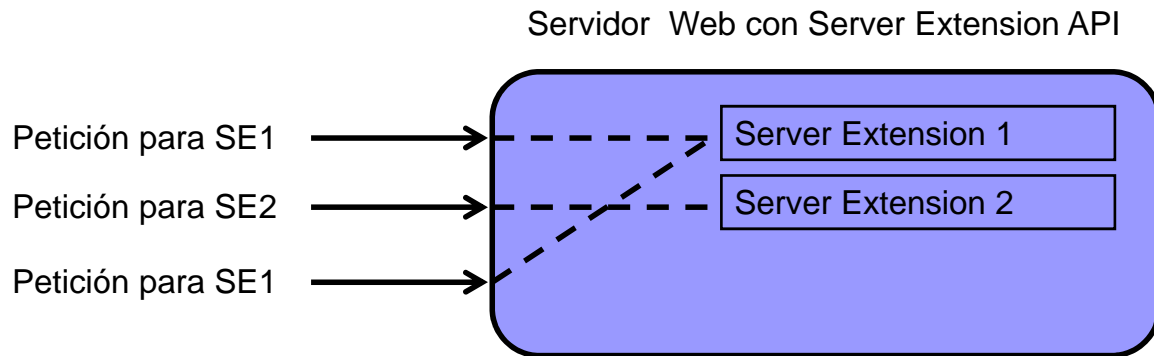
APIs Server Extension

- API que permite extender la funcionalidad del servidor Web para procesar peticiones HTTP y generar contenido dinámico como respuesta
- Propietarias
 - ASP/Microsoft, WAI/Netscape
 - Solamente funcionan en algunos servidores Web
- Basadas en Java
 - Servlets, páginas JSP
 - Son portables a todo aquel servidor Web que incluya una JVM

28



Ciclo de vida APIs Server Extension



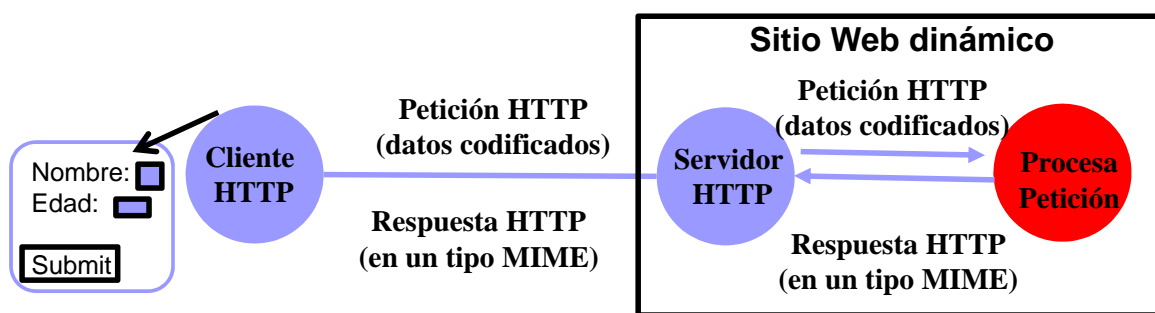
- Cada servicio se atiende de forma independiente, pero concurrentemente
- Hilos de ejecución creados dentro del servidor Web

29



2.2.2. Procesamiento de formularios en sitios Web dinámicos

- Procesamiento de formularios en sitios Web dinámicos
 - Formularios HTML
 - Codificación de datos en peticiones HTTP
 - Variables de entorno
 - Diagrama de flujo de un sitio Web dinámico



Formulario

30



HTML

■ *HyperText Markup Language*

- ☐ Lenguaje basado en etiquetas, interpretable por los navegadores web y utilizado como lenguaje de base para la creación de páginas web.
- ☐ Estandarizado por el W3C

ejemplo.html

```
<!DOCTYPE html>
<html>

  <head>
    <title> HTML </title>
  </head>

  <body>
    <h1>Documento Ejemplo</h1>
    <p>Texto.</p>
  </body>

</html>
```

31



Etiquetas HTML

■ Las etiquetas html tienen la siguiente forma

`<etiqueta>contenido</etiqueta>`

■ Elemento HTML

- ☐ Se denomina elemento HTML a una unidad de visualización formada por una etiqueta inicial, el contenido a visualizar y la etiqueta final

`<p>Esto es un parrafo</p>`

■ Declaración <!DOCTYPE>

- ☐ Se utiliza para definir la versión HTML utilizada en la creación de la página web

32



Versiones HTML

■ Declaración `<!DOCTYPE>`

☐ HTML5

```
<!DOCTYPE html>
```

☐ XHTML 1.0

```
<?xml version="1.0"
encoding="UTF-8"?>
<!DOCTYPE html>
```

Version	Year
HTML	1991
HTML+	1993
HTML 2.0	1995
HTML 3.2	1997
HTML 4.01	1999
XHTML 1.0	2000
HTML5	2012
XHTML5	2013



Editores HTML

■ Editores de texto

- ☐ notepad, wordpad, etc.

■ Editores profesionales de HTML

- ☐ Adobe Dreamweaver
- ☐ Microsoft Expression Web
- ☐ CoffeeCup HTML Editor

■ Entornos de desarrollo

☐ NetBeans

- New File -> Web -> HTML
- New File -> HTML5/JavaScript -> HTML (tema 3)



HTML básico

- Cabecera
- Cuerpo
 - Texto
 - Enlaces
 - Imágenes
 - Estructurar un documento
 - Formularios

35



HTML básico – cabecera

- Cabecera
 - `<head> ... </head>`
 - Título `<title>` (nombre de la pestaña)
`<title>TODO supply a title</title>`
 - Etiqueta `<meta>`
 - Propiedades del documento (ej: tamaño pantalla, cabecera http de fecha, etc. autor, etc.)
`<meta charset="UTF-8">`
`<meta name="viewport" content="width=device-width, initial-scale=1.0">`
 - Estilo `<style>` (ver Tema 3)
 - Código embebido `<script>` (ver Tema 3)

36



HTML básico – cuerpo - texto

■ Encabezados

- Se definen 6 estilos de cabeceras <h1> a <h6>

```
<h1> Título tipo 1 </h1>
```

■ Párrafo

- Texto separado por una línea antes y después

```
<p> Texto normal </p>
```

■ Texto

- Formato texto
- Etiqueta (deprecated)

```
<b>Texto en negrita</b>
<big>Texto grande</big>
<em>Texto enfatizado</em>
<i> Texto en itálica</i>
<small>Texto pequeño</small>
<strong>Texto fuerte</strong>
<sub> Texto subíndice</sub>
<sup> Texto superíndice</sup>
<ins> Texto subrayado</ins>
<del> Texto tachado</del>
<tt> Texto teletipo</tt>
```

```
<p> <font size=5 color="#0000ff">Texto distinto</font></p>
```

37



HTML básico – enlaces

■ Enlaces

- Etiqueta anchor <a>

```
<a href="localización">Enlace</a>
```

- Enlace página local

```
<a href="pagina_local.html">Enlace página local</a>
```

- Enlace página remota

```
<a href="http://www.dst.net">Enlace página remota</a>
```

- Enlaces dentro de la página

- Identificar el elemento a enlazar

```
<h1 id="titulo1">Título 1</h1>
```

- Enlace elemento página local

```
<a href="#titulo1">Enlace a Título 1</a>
```



HTML básico – imágenes

■ Imágenes

``

☐ Localización

``

☐ Texto alternativo

``

☐ Título

``

☐ Alto y ancho

``

☐ Como enlaces

``³⁹

navidad.jpg



HTML básico – estructurar doc

■ Estructurar un documento <div>

`<div id="bloque1" style="estilo propio" ...>`

☐ Agrupa un conjunto de elementos html dentro de una página web

■ Nos permite referenciarlos con un identificador único

■ Es posible definir un estilo conjunto (ej: tamaño letra)

☐ Definirlo con el atributo estilo (<style>)

☐ Definirlo con una página de estilo (ver tema 3)

■ Nos permite reemplazar el contenido de dicho bloque

☐ Permite modificar zonas de una página web (ver tema 3)

```
<div id="bloque1" style="estilo propio" ...>
  <h3> Título bloque </h3>
  <p> En este bloque se habla de ... </p>
</div>
```

```

<!DOCTYPE html>
<html>
  <head>
    <title>Desarrollo de Servicios Telemáticos</title>
    <meta charset="UTF-8"><meta name="viewport" content="width=device-width, initial-scale=1.0">
  </head>
  <!-- Esto es un comentario -->
  <body>
    <h1 id="inicio">Ilustramos algunos elementos HTML</h1>
    <p><a href="#t1">[Importante]</a> <a href="#t2">[Párrafo con varios estilos]</a>
      <a href="#t3">[No es importante]</a></p>
    <p> Copiar esta página web en el editor Netbeans y ejecutarla para ver su resultado </p>
    <h2 id="t1">Importante</h2>
    <p><big>Este texto es importante por eso lo pongo en grande</big></p>
    <h2 id="t2">Párrafo con varios estilos</h2>
    <p>Un <strong>subíndice</strong> o <strong>superíndice</strong>, es el número, letra o
    símbolo que aparece de tamaño <em>más pequeño</em> que la escritura normal y que se encuentra
    ligeramente por debajo (subíndice, ej: <tt>X<sub>i</sub></tt>) o por encima
    (superíndice, <tt>X<sup>i</sup></tt>) del texto. <a
    href="https://es.wikipedia.org/wiki/Sub%C3%ADndice_y_super%C3%ADndice">Ampliar esta definición en
    la Wikipedia</a></p>
    <h2 id="t3">No es importante</h2>
    <p><small>Esto no es importante por eso lo pongo en pequeño.</small>
    <br> Con lo que he puesto antes he conseguido cambiar de línea!
  </p>
  <h1>Se acerca la Navidad</h1>
  <div id="b1" style="border: 2px solid rgb(204, 102, 204);">
    <h3>Lista de deseos para Navidad</h3>
    <ul>
      <li> Terminar todas las prácticas a tiempo </li>
      <li> Sacar buena nota </li>
    </ul>
    <p> Pongamos alguna foto navideña !</p>
    
  </div>
  <a href="#inicio">Regresar a la parte superior</a>
  </body>
</html>

```



Lidia Fuentes Fernández

Formularios HTML

■ Formularios HTML (introducir datos del lado del cliente)

```

<form action="http://aplic_web" method="GET|POST">
  tipos de entrada de datos, input, textarea, etc.
</form>

```

■ Entrada de datos

- ☐ El elemento <input> es el más importante
- ☐ Atributos input →

```

<input type="tipo"
name="nombre_dato">

```

Atributo	Descripción
type	Define el tipo de entrada de datos
name	Da nombre al valor de entrada
value	Proporciona un valor inicial
size	Define el ancho del campo en caracteres
maxlength	Máximo número de caracteres permitido



Formularios HTML

■ Tipos entrada de datos

□ Campos de texto

■ Campo de una línea

```
<input type="text" name="nombre" value="Dato por defecto">
```

■ Campo multilínea

```
Campo Libre: <textarea rows="3"
               name="campolibre"> Completar ...</textarea>
```

■ Password

```
Password:<input type="password" name="clave" size=10 value="xxx">
```

□ Radio

```
<input type="radio" name="sexo" value="femenino">
```

43



Formularios HTML

■ Tipos entrada de datos

□ Checkboxes

Intereses:


```
<input type="checkbox" name="deporte" value="deport">Deportes
```

```
<input type="checkbox" name="politica" value="polit">Política
```

```
<input type="checkbox" name="ciencia" value="cienc">Ciencia
```

□ Select

Selecciona un departamento

```
<select name="departamento">
```

```
<option value="Lenguajes" selected>Lenguajes y CC
```

```
<option value="Comunicaciones">I. de Comunicaciones
```

```
</select>
```

44



Formularios HTML

■ Tipos entrada de datos

- ☐ Submit (envía los datos introducidos por el usuario)
- ☐ Reset (limpia los campos de formularios)

```
<form action="http://aplic_web" method="POST">
...
    <input type="submit" value="Enviar">
    <input type="reset" value="Limpiar">
</form>
```

☐ Clickable buttons

- Crea un botón usado para llamar a un código script (ver tema 3)

```
<input type="button" name="ok" value="OK">
```

45



```
<html><head><title> Ejemplo de formularios </title></head>
<body>
```

```
<FORM METHOD="GET" ACTION="http://xyz.lcc.uma.es/app-bin/form">
```

Campo de Texto

```
<INPUT TYPE="text" NAME="nombre" size=40 value="Dato por defecto">
```

Area de texto: <textarea NAME="textarea" rows=3 cols=20>Dato por defecto</textarea>

Password: <INPUT TYPE="password" NAME="password" size=10 value="xxx">

```
</FORM>
```

```
<FORM METHOD="GET" ACTION="http://xyz.lcc.uma.es/app-bin/cues1-1">
```

```
<h2>Pregunta 1 (Radio Buttons)</h2><p>La asignatura DST, ¿es sencilla?<ul>
```

```
<li><INPUT TYPE="radio" NAME="unouno" VALUE="Si">Si
```

```
<li><INPUT TYPE="radio" NAME="unouno" VALUE="No">No
```

```
<li><INPUT TYPE="radio" NAME="unouno" VALUE="En determinadas ocasiones">En
determinadas ocasiones</ul><p>
```

```
<h2>Pregunta 2 (Checkbox)</h2><p>Señala los temas que mas te han gustado<ul>
```

```
<li><INPUT TYPE="checkbox" NAME="unodos" VALUE="Sockets TCP">Sockets TCP
```

```
<li><INPUT TYPE="checkbox" NAME="unodos" VALUE="Sockets UDP">Sockets UDP
```

```
<li><INPUT TYPE="checkbox" NAME="unodos" VALUE="Sockets en Java">Sockets en
Java </ul><p>
```

```
<h2>Pregunta 3 (Select)</h2><p><ul>
```

Selecciona que sistemas operativos prefieres

```
<SELECT NAME="so"><OPTION SELECTED>Linux<option>Windows</select><p>
```

```
Pulsa el siguiente boton: <INPUT TYPE="submit" VALUE="RESPUESTA"><p>
```

```
</FORM></body></html>
```

46



Resultado formulario

Campo de Texto Area de texto: Password:

Pregunta 1 (Radio Buttons)

La asignatura Software de Comunicaciones, ¿es sencilla?

- ☐ Si
- ☐ No
- ☐ En determinadas ocasiones

Pregunta 2 (Checkbox)

Señala los temas que mas te han gustado

- ☒ Sockets TCP
- ☐ Sockets UDP
- ☒ Sockets en Java

Pregunta 3 (Select)

Selecciona que sistemas operativos prefieres

Pulsa el siguiente boton:

47



Codificación datos petición HTTP

- Cada elemento del formulario se pasa como un conjunto nombre=valor
 - ☐ Ej: edad=25 o unouno="Si"
- Reemplazar cada espacio por un signo de "+"
 - ☐ Ej: "Hola mundo" -> "Hola+mundo"
- Cada par nombre-valor se separa por "&"
 - ☐ Ej: edad=25&nombre="Pepa"
- Los caracteres alfanuméricos se traducen a su código de escape
 - ☐ Ej: "!" -> %21

Ej: nombre="Juan+Fuentes"&comentario="Bravo%21"

Content-Type: application/x-www-form-urlencoded

48



Mensajes HTTP (GET y POST)

GET /cgi-gin/Cuest1-1?datos-del-formulario-codificados

Connection: Keep-Alive

User-Agent: Mozilla/4.5 (compatible;MSIE 4.01;Windows NT)

Host: localhost

Accept: image/gif, text/html

POST /cgi-bin/Cuest1-1

Connection: Keep-Alive

User-Agent: Mozilla/4.5 (compatible;MSIE 4.01;Windows NT)

Host: miServidor.lcc.uma.es

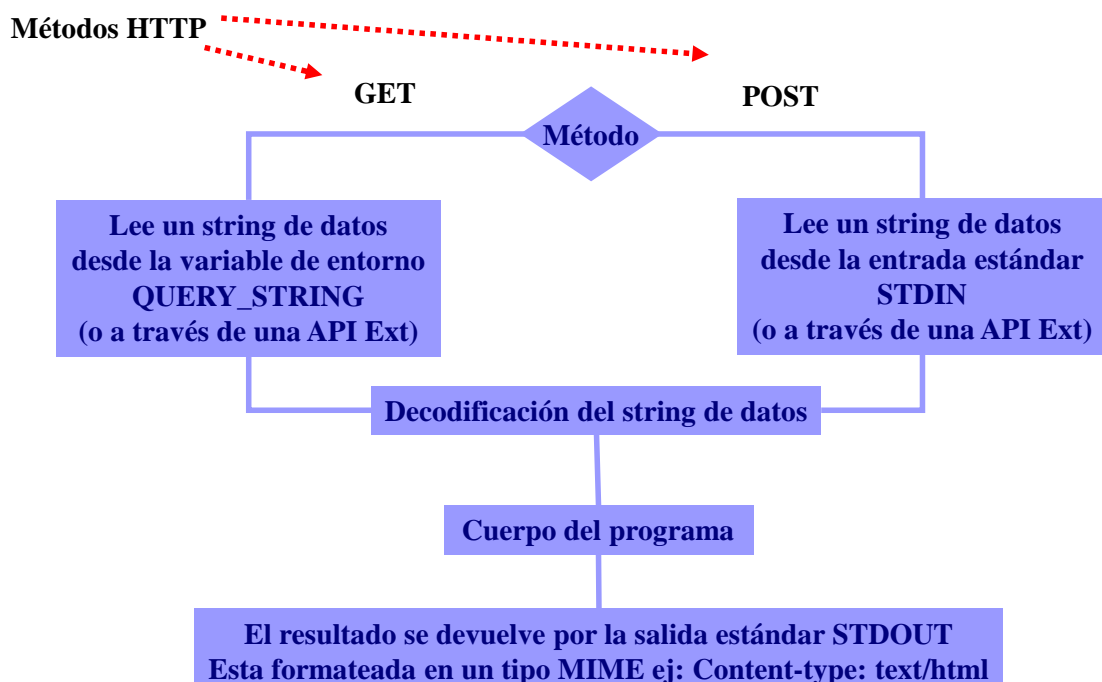
Accept: image/gif, text/html

datos-del-formulario-codificados

49



Diagrama de flujo de un programa CGI/FastCGI/Server Extension



50



Variables de entorno externas al servidor Web

Variable	Valor
SERVER_ADMIN	La dirección <i>e-mail</i> de la persona que se encarga del servidor.
SERVER_SOFTWARE	Identifica al servidor y su versión.
SERVER_NAME	Identifica el nombre DNS del servidor, o, en su defecto, su dirección IP.
GATEWAY_INTERFACE	Versión del protocolo CGI utilizada, normalmente CGI/1.x.
AUTH_TYPE	Muestra el protocolo específico de autenticación.
CONTENT_LENGTH	La longitud del cuerpo del mensaje que envió el cliente. Es utilizado por el CGI para saber cuándo dejar de leer de la entrada estándar.
CONTENT_TYPE	Tipo MIME de los datos del cliente.
HTTP_REFERER	El URL del que el <i>script</i> fue invocado.
HTTP_REQUEST_METHOD	El método utilizado por el cliente.
HTTP_USER_AGENT	User-Agent de la petición HTTP.
QUERY_STRING	Los datos del form del usuario (si la petición es GET).
REMOTE_ADDR	Dirección IP del cliente.
REMOTE_HOST	Nombre DNS del cliente.
SCRIPT_FILENAME	El valor del <i>path</i> completo al <i>script</i> CGI.
SCRIPT_NAME	Nombre del <i>script</i> .
SERVER_PORT	Indica el puerto al que la petición del cliente se lanzó.
SERVER_PROTOCOL	Indica el protocolo que el servidor está utilizando.

51



Ejemplo

- Ejemplo de programación CGI en “C”
 - Procesamiento del formulario sobre la asignatura



Programa CGI en C (con GET)

```
//Funciones para decodificar datos:
//Ver http://www.jmarshall.com/easy/cgi/getcgi.c.txt
void getword(char *word, char *line, char stop):
char x2c(char *what);
void unescape_url(char *url);
void plustospace(char *str);

//Estructura para almacenar los pares nombre-valor
typedef struct {
    char name[128];
    char val[128];
} entry;

main() {
    entry entries[10000];
    register i,j;
    char *cl;

    //Verificar que los datos se han introducido mediante el metodo GET
    if (strcmp(getenv("REQUEST_METHOD"),"GET")!= 0) {
        printf("Debe utilizar el metodo GET\n");exit(1); }
```

53



Programa CGI en C (con GET)

```
//Obtener la peticion HTTP de la variable de entorno
cl=getenv("QUERY_STRING");
if (cl ==NULL) { printf("Peticion vacia\n");exit(1); }

//Decodificar los datos hasta final de cadena
for (i=0;cl[i]!='\0';i++) {
    j=i;
    getword(entries[i].val,cl,&');
    plustospace(entries[i].val);
    unescape_url(entries[i].val);
    getword(entries[i].name,entries[i].val, '=');
}

//A partir de aqui escribo la respuesta en la salida estandar
printf("<p><h1>Respuesta</h1>\n");
if (!strcmp(entries[0].val,"Si"))
    printf("Me alegra que la asignatura te parezca sencilla");
else //resto del procesamiento .....
```

54



2.3. Los servlets y otras tecnologías

- Los Servlets de Java
- Seguimiento de sesiones
- Otras tecnologías



2.3.1 Los servlets de Java

- Características de los *Servlets*
- Funcionalidad de la API Servlets
- Diagrama de clases
- Ejemplos de *servlets*
- Invocación de un *servlet*



Definición de un *Servlet*

- Son la implementación en Java de la programación CGI
- Son programas escritos en Java que se ejecutan en un servidor Web y le añaden funcionalidad e interactividad con el cliente
- Están diseñados en base a un modelo pregunta/respuesta donde el *servlet* recibe y responde a las peticiones de varios clientes
- Traspase de información cliente/*servlet*

57



Características de los *Servlets*

- Independientes del servidor Web y del sistema operativo
- Se cargan en memoria de forma permanente
- Programación orientada a objetos, interacción entre *Servlets*, programas Java con RMI, etc.
- Fácil obtención de información del cliente mediante acceso a métodos (no es necesario consultar variables de entorno)
- Permiten utilizar *cookies* y sesiones
- Generación dinámica de páginas HTML

58



Entorno de ejecución

- Los servlets se ejecutan en un *servidor de aplicaciones web* (contenedor del servicio)
- El servidor web debe tener una JVM
- Servidores que soportan servlets
 - Apache Tomcat (desarrollado bajo el proyecto Jakarta)
 - Glassfish
 - Java Web Server
 - Netscape Fast Track y Enterprise
 - Lotus Domino Go WebServer
 - BEA WebLogic Server
 - IBM WebSphere Application Server
 - ...

59



API javax.servlet

- Formada por interfaces y clases que permiten heredar la funcionalidad de un *servlet*
- Paquetes java (Java EE, Servlet 3.2)
 - javax.servlet
 - Clases e interfaces genéricas independientes del protocolo de acceso al *servlet*
 - javax.servlet.http
 - Clases e interfaces específicas del protocolo HTTP
- Servlet 4.0
 - Implementada por Apache
 - javax.servlet.http.PushBuilder

60



Ciclo de vida de un Servlet

■ Init

- Llamado por el contenedor (servidor Web) al cargar la clase del *servlet*
- Inicializaciones (abrir una base de datos, configuración, etc.)

■ Service

- Petición GET, POST, etc. y respuesta

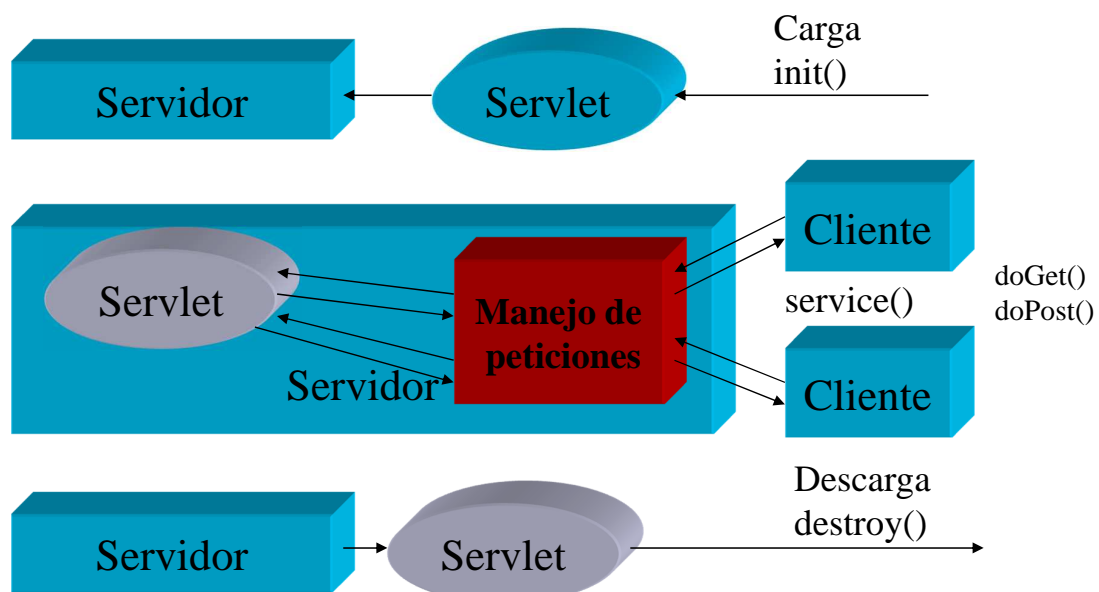
■ Destroy

- Llamado por el contenedor (servidor Web) después de ver que lleva cierto tiempo sin usarse, entonces descarga la clase del *servlet* de memoria

61



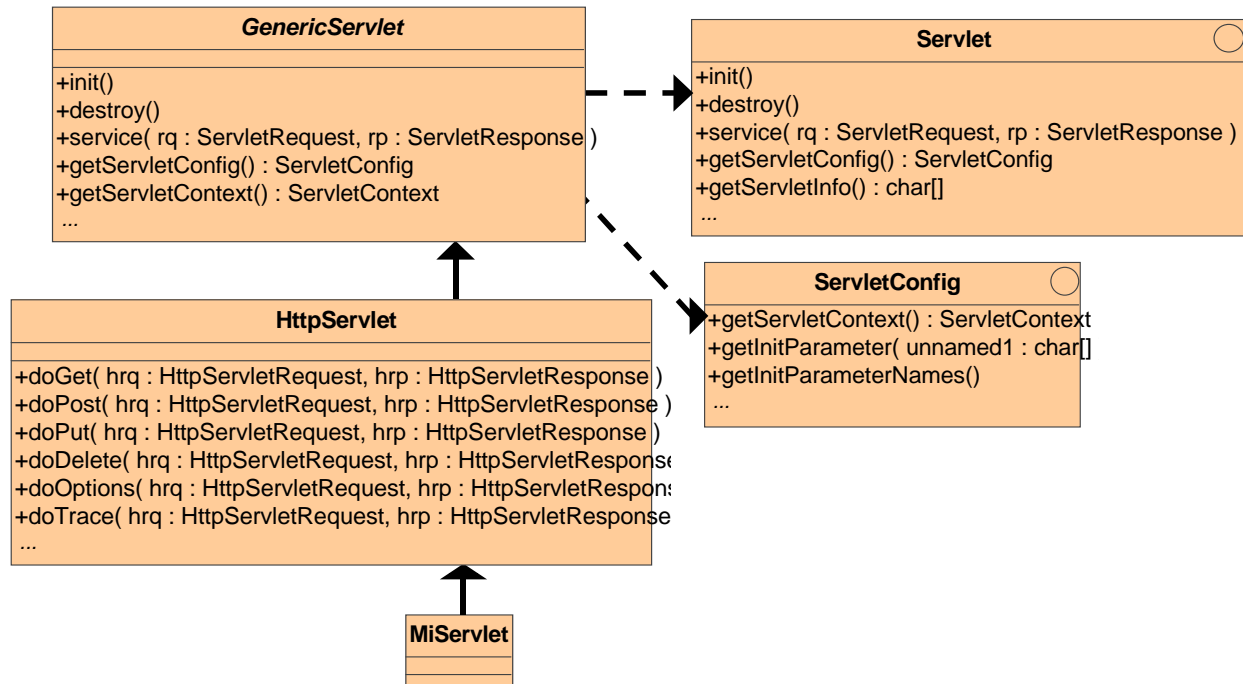
Ciclo de vida de un Servlet



62



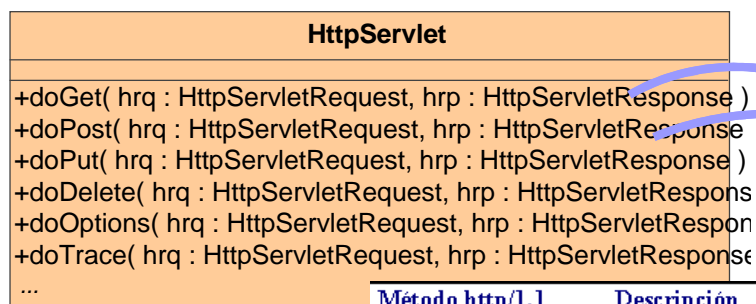
Diagrama de clases



63



Correspondencia métodos HTTP

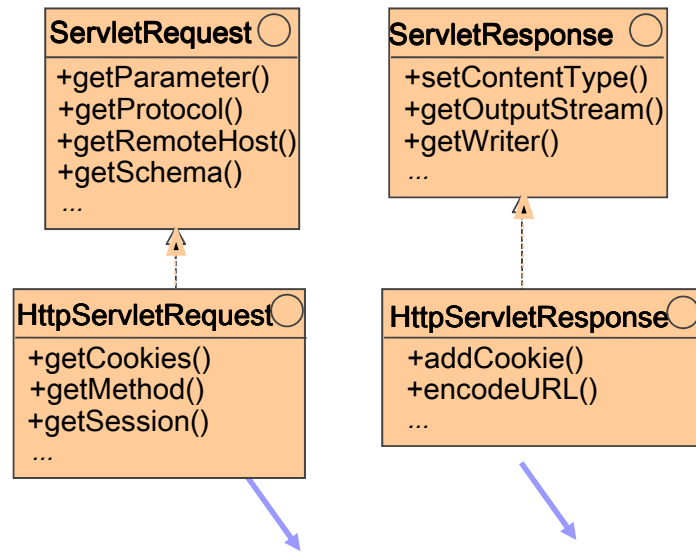


Método http/1.1	Descripción
GET	Recupera la URL especificada
POST	Envía datos a la URL especificada
PUT	Guarda estos datos en la URL especificada, reescribiéndolos
DELETE	Borra el recurso identificado por el URL
OPTIONS	Pide información acerca de las opciones de comunicación disponibles para la URL especificada
TRACE	Responde, en el cuerpo del mensaje, la petición del cliente
MOVE	Mueve el contenido del recurso a la(s) direcciones(es) especificada(s)
...	

64



Petición y Respuesta

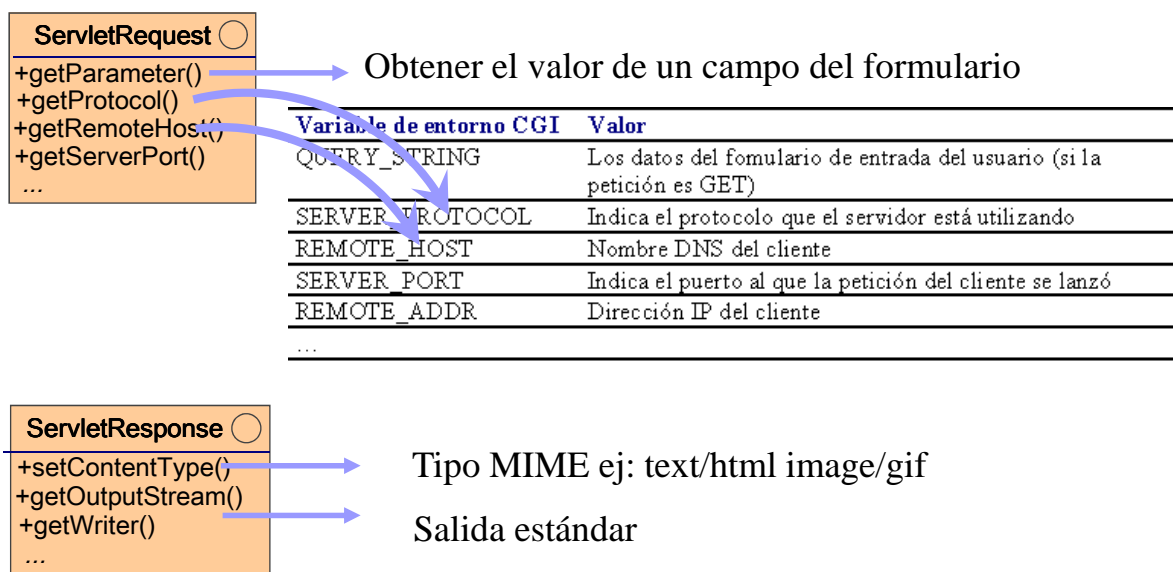


doGet ó doPost(hrq: HttpServletRequest, hrp: HttpServletResponse)

65



Petición y Respuesta



66



Ejemplo *HelloWorld* (GET)

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

```
public class HelloWorld extends HttpServlet {
```

```
    public void doGet(HttpServletRequest hrq, HttpServletResponse hrp)  
        throws IOException, ServletException  
    {  
        hrp.setContentType\("text/html;charset=UTF-8"\); // Primer método de la respuesta  
        PrintWriter out = hrp.getWriter\(\);  
        out.println("<html><head>");  
        out.println("<title>Servlet HelloWorld</title>");           out.println("</head>");  
        out.println("<body>");  
        out.println("<h1>Servlet HelloWorld at " + hrq.getContextPath() + "</h1>");  
        out.println("</body></html>");  
    }  
}
```



67



Codificación HTTP

[GET /EjemploServlet/HelloWorld?parametros+extra](#)

Connection: Keep-Alive

User-Agent: Mozilla/4.5 (compatible;MSIE 4.01;Windows NT)

Host: localhost

Accept: image/gif, text/html

68



Ejemplo completo (POST)

```
<html><head>
  <meta http-equiv="Content-Type" content="text/html">
  <meta name="GENERATOR" content="Mozilla/4.77 [en]">
  <title>Ejemplo de Servlet</title>
</head>
<body>
<h2>
Por favor, introduzca su nombre y apellidos</h2>
<form ACTION="http://localhost:8084/EjemploServlet/EjemploServlet" METHOD="post">
  Nombre:&nbsp;<input TYPE="TEXT" NAME="nombre" SIZE=15>
  <br>Apellidos:&nbsp;<input TYPE="TEXT" NAME="apellidos" SIZE=30>
  <input TYPE="SUBMIT" VALUE="Enviar">
<br>&nbsp;</form>
</body>
</html>
```

69



Ejemplo completo (POST)

POST /EjemploServlet/EjemploServlet

Connection: Keep-Alive

User-Agent: Mozilla/4.5 (compatible;
MSIE 4.01;

Windows NT)

Host: miServidor.lcc.uma.es

Accept: image/gif, text/html

nombre=Lidia&apellidos=Fuentes+FernA0ndez

70

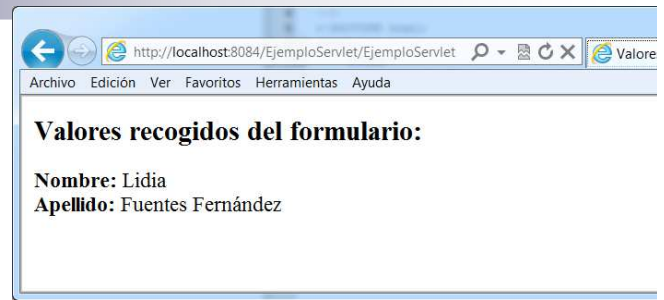


Ejemplo completo

```
public class EjemploServlet extends HttpServlet {
    private String nombre=null;
    private String apellidos=null;

    public void init(ServletConfig config) throws ServletException {
        super.init(config);
        System.out.println("Iniciando EjemploServlet...");
    }
    public void destroy() {
        System.out.println("No hay nada que hacer...");
    }
    public void doPost (HttpServletRequest hrq, HttpServletResponse hrp) {
        throws ServletException, IOException {
            nombre  = hrq.getParameter("nombre"); // Si no se ha introducido nada devuelve null
            apellidos = hrq.getParameter("apellidos");

            devolverPaginaHTML(hrp);
        }
        public void devolverPaginaHTML (HttpServletResponse hrp) {
            hrp.setContentType("text/html");
            PrintWriter out = hrp.getWriter();
            out.println("<html>.....</html>");
        }
    }
}
```



71



Lectura de parámetros

```
public void doPost(HttpServletRequest hrq,HttpServletResponse hrp) throws
ServletException,IOException
{
    //Tipo de respuesta
    hrp.setContentType("text/html");
    //Salida
    ServletOutputStream out=hrp.getOutputStream();

    out.println("<h2>Parametros recogidos: nombre y email</h2>\n");
    //Capturo los parametros de una vez y los almaceno en un Enumeration
    Enumeration parametros=hrq.getParameterNames();
    while(parametros.hasMoreElements()) {
        String par=(String) parametros.nextElement();
        String[] valores=hrq.getParameterValues(par);
        out.print("<h1>"+par+": <h1>");

        // Contempla parametros multivaluados
        for(int i=0;i<valores.length;i++) {
            if (valores[i].length()==0)
                out.println("valor nulo");
            else
                out.println(valores[i]);
        }
    }
    out.println("</body>\n"); out.println("</html>\n");
}
```

72



Salida

SERVLETS :
Formulario

Este es un ejemplo de un Formulario que recoge los datos y los procesa mediante servlets

Nombre:

E-mail:

Alias:

Aficiones:

Estudios:

Ciudad:

Parametros recogidos:

email:
jbh@yahooo.coma

name:
Juan Blanco Herrera

aficiones:
La lectura y los deportes

alias:
Juanito

ciudad:
Granada

estudios:
Universitarios

73



Obtener variables de entorno

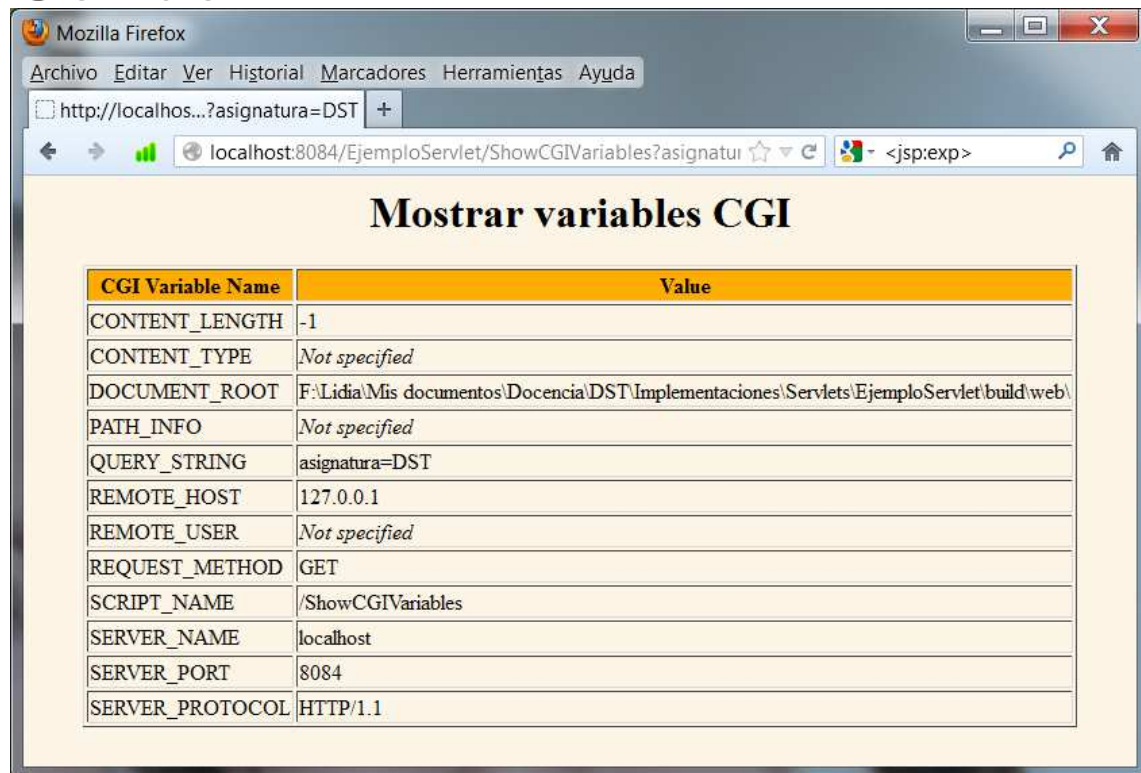
```
public void doGet(HttpServletRequest request, HttpServletResponse response) throws ... {
```

```
    response.setContentType("text/html");
    PrintWriter out = response.getWriter();
    String[][] variables =
    { { "CONTENT_LENGTH", String.valueOf(request.getContentLength()) },
      { "CONTENT_TYPE", request.getContentType() },
      { "DOCUMENT_ROOT", getServletContext().getRealPath("/") },
      { "PATH_INFO", request.getPathInfo() }, { "QUERY_STRING", request.getQueryString() },
      { "REMOTE_HOST", request.getRemoteHost() }, { "REMOTE_USER", request.getRemoteUser() },
      { "REQUEST_METHOD", request.getMethod() }, { "SCRIPT_NAME", request.getServletPath() },
      { "SERVER_NAME", request.getServerName() },
      { "SERVER_PORT", String.valueOf(request.getServerPort()) },
      { "SERVER_PROTOCOL", request.getProtocol() } };
    String title = "Servlet Example: Showing Environmental Variables";
    out.println("<BODY BGCOLOR=\"\#FDF5E6\">\n" +
        "<H1 ALIGN=CENTER>" + title + "</H1>\n" +
        "<TABLE BORDER=1 ALIGN=CENTER>\n" +
        "<TR BGCOLOR=\"\#FFAD00\">\n" +
        "<TH>CGI Variable Name<TH>Value");
    for(int i=0; i<variables.length; i++) {
        String varName = variables[i][0];
        String varValue = variables[i][1];
        if (varValue == null)
            varValue = "<I>Not specified</I>";
        out.println("<TR><TD>" + varName + "<TD>" + varValue);
    }
    out.println("</TABLE></BODY></HTML>");
}
```

74



Salida



The screenshot shows a Mozilla Firefox browser window with the address bar displaying `http://localhost:8084/EjemploServlet/ShowCGIVariables?asignatura=DST`. The page title is "Mostrar variables CGI". Below the title is a table with two columns: "CGI Variable Name" and "Value".

CGI Variable Name	Value
CONTENT_LENGTH	-1
CONTENT_TYPE	<i>Not specified</i>
DOCUMENT_ROOT	F:\Lidia\Mis documentos\Docencia\DST\Implementaciones\Servlets\EjemploServlet\build\web\
PATH_INFO	<i>Not specified</i>
QUERY_STRING	asignatura=DST
REMOTE_HOST	127.0.0.1
REMOTE_USER	<i>Not specified</i>
REQUEST_METHOD	GET
SCRIPT_NAME	/ShowCGIVariables
SERVER_NAME	localhost
SERVER_PORT	8084
SERVER_PROTOCOL	HTTP/1.1

75



Invocación de un *Servlet*

- URL
 - ☐ Invocado directamente desde el *browser* (método *doGet*)
- Formulario HTML resultado de una operación
 - ☐ GET (restricciones de tamaño en los datos de la petición)
 - ☐ POST (el más utilizado)
- Etiqueta servlet
 - ☐ En JSP (*Java Server Page*)
- Directamente desde otro *servlet*
 - ☐ `hrp.sendRedirect("/OtroServlet")`
 - ☐ `hrq.getRequestDispatcher("/OtroServlet").forward(hrq,hrp)`

76



Invocación de servlets desde código

- **sendRedirect**
 - Se envía una petición a otra URL
- **forward**
 - Nos movemos a otra URL local
 - Se conservan los objetos asociados a `HttpRequest`
 - Útil para tratar errores en formularios
 - Agregar un atributo de error (ej: errors de tipo Map) en la request con `setAttribute()`
 - El servlet que muestra el formulario de resultado comprueba si `HttpRequest` incluye un atributo de error (ej: errors)

77



Uso de `sendRedirect` y de `forward`

- Un `forward` siempre es más rápido dado que es una llamada local que no genera una nueva petición
- Es necesario usar `forward` cuando queremos pasar valores de atributos (dentro de `HttpRequest`)
 - Ej: Tratamiento de errores en la entrada de datos de formularios
- El `sendRedirect` es el único que muestra la URL nueva en el navegador (el `forward` al ser local, no es percibido por el navegador)
 - Mejor usarlo siempre por si el usuario decide recargar la página (con el `forward` se mantiene la URL antigua, entonces recargaría la página antigua lo cuál puede confundir al usuario)

78



2.3.2. Seguimiento de sesiones

- Cookies
- Sesiones
- Reescritura de URLs
- Ejemplo de servlets para autenticar usuario de un servicio Web

79



Seguir la trayectoria de los usuarios

- Obtener y mantener una información del cliente

- ☐ Ej: comercio electrónico (carrito de la compra)  Add to cart

- Problemas protocolo HTTP

- ☐ Sin estado (*stateless*)
- ☐ Anónimo (sin *login* ni *password*)

- Soluciones

- ☐ Campos ocultos en los formularios

- ☐ `<INPUT TYPE="hidden" NAME="user" VALUE="dst38">`

- ☐ Cookies ("galleta" o "ficha")
- ☐ Seguimiento de sesiones
- ☐ Reescritura de URLs

80



Cookies

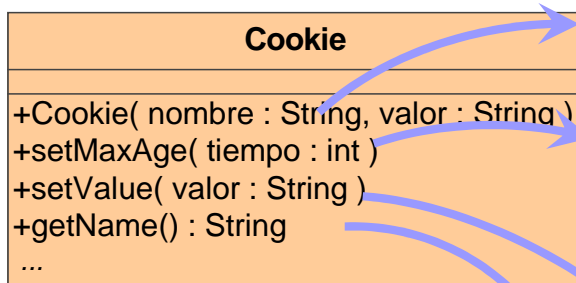
- Traspase de información cliente/*servlet* con *cookies*
- Respuesta de un *servlet* con *cookies*
 - Tiene un nombre asociado
 - El cliente debe ser capaz de soportarlas
 - Se almacenan temporalmente en la máquina del cliente
 - El *servlet* envía una *cookie* en la respuesta y los clientes la devuelven en la petición
 - Se envían sólo al servidor que las originó
 - Los *servlets* de un servidor comparten las *cookies*

81



Clase Cookie

- Clase `javax.servlet.http.Cookie`



Creación *cookie* por nombre y valor

Tiempo de caducidad de la *cookie*

- < 0, se borra tras cerrar el browser
- > 0, duración en segundos
- = 0, borrado de la *cookie*

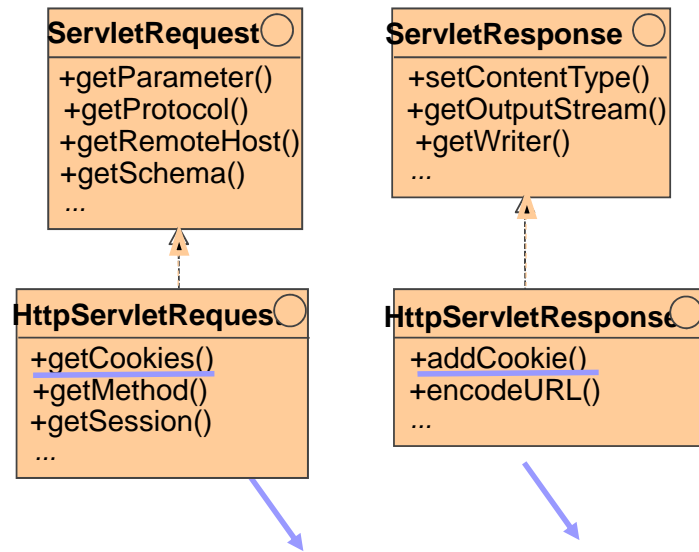
Establece el valor de la *cookie*
¡tipo String!

Devuelve el nombre de una *cookie*

82



Petición y Respuesta



doGet ó doPost(hrq: HttpServletRequest, hrp HttpServletResponse)

83



Implementación método doGet() o doPost()

+ Crear un objeto Cookie

```
String IdObject = new String("301");
if (IdObject != null)
    Cookie myCookie = new Cookie("Cesta", IdObject);
```

+ Establecer atributos de la *cookie*

```
myCookie.setValue("Libro301");
myCookie.setMaxAge(-1);
```

+ Enviar la cookie

```
public void doGet(HttpServletRequest hrq, HttpServletResponse hrp)... {
    ...
    hrp.addCookie(myCookie);
    PrintWriter out=hrp.getWriter();
    ...
}
```

84



Implementación método doGet() o doPost()

+ Recoger las *cookies*

```
Cookie myCookie = null;
Cookie[] arrayCookies = hrq.getCookies\(\);
```

+ Encontrar y obtener el valor de una *cookie*

```
...
String DeleteBook = hrq.getParameter("Delete");
if (DeleteBook != null) {
    for (i=0;i<arrayCookies.length;i++) {
        Cookie myCookie = arrayCookies[i];
        if (myCookie.getName\(\).equals("Cesta") &&
            (myCookie.getValue\(\).equals("Libro301"))) {
            myCookie.setMaxAge(0);           // Elimina la cookie
        }
    }
    ...
}
```

85



Ejemplo Cookies

The screenshot shows a web browser window with the address bar displaying `http://localhost:8084/EjemploServlet/Cookies.html`. The browser has a single tab titled 'Tienda virtual'. The page content includes a heading 'Compra' and two radio buttons: 'Chocolate' (which is selected) and 'Caramelo'. Below the radio buttons is a button labeled 'Enviar'.

```
<!doctype html public "-//w3c//dtd html 4.0 transitional//en">
<html><head>
<title>Tienda virtual</title>
</head>
<body>
<h2>Compra</h2>
<form ACTION="http://localhost:8080/servlet/Cookies" METHOD="POST"><p>
<br><input TYPE="RADIO" CHECKED NAME="item" VALUE="chocolate">Chocolate
<br><input TYPE="RADIO" NAME="item" VALUE="caramelo">Caramelo
<p><input TYPE="SUBMIT" NAME="botonEnviar" VALUE="Enviar"></form>
</body>
</html>
```

86



Ejemplo Cookies

```
public void doPost (HttpServletRequest hrq, HttpServletResponse hrp) {
    throws ServletException, IOException {
    hrp.setContentType("text/html");
    String item=hrq.getParameter("item");
    Cookie[] cookies = hrq.getCookies();
    int i=0;
    if (cookies == null) {
        cookie = new Cookie("Cesta", item);
        cookie.setMaxAge(120);
    } else {
        while ((i<cookies.length) && !(cookies[i].getName().equals("Cesta"))) i++;
        if (cookies[i].getName().equals("Cesta")) {
            cookie = cookies[i];
            item = cookie.getValue()+" "+item;
            cookie.setValue(item);
        }
    }
    hrp.addCookie(cookie);
    PrintWriter out = response.getWriter();
    out.println("<html>");
    out.println("<body bgcolor=\"white\">");
    // Resto formulario de respuesta
}
```

87



Codificación HTTP

Servidor -> Cliente (establecimiento de una cookie)

HTTP/1.0 200 OK

Server: Apache/1.3b3

Mime-Version: 1.0

Content-type: text/html

Set-Cookie: Cesta="chocolate";Version="1"; Max-Age="120";
Path="/EjemploServlet"

<HTML>

...

</HTML>

88



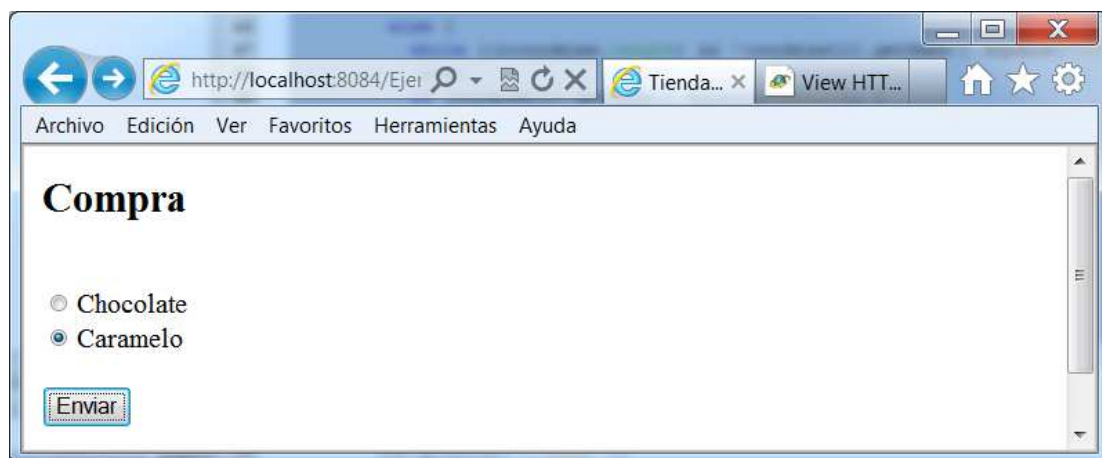
Fichero con cookies

- Fichero cookies (o cookies.txt) simplificado

#DOMINIO	PATH	EXPIRE	NOMBRE	VALOR
150.214.108.85:8080	/EjemploServlet	TRUE	Cesta	chocolate
bignosebird.com	/docs	FALSE	YourPage	here
.netscape.com	/	FALSE	UIDC	150.214.108.118:0975
.doubleclick.net	/	TRUE	id	80000004ca6a25a
.go.com	/	TRUE	UserId	909C26CDE025AE923...
...				



Ejecución del Ejemplo





Codificación HTTP

Cliente -> Servidor (envío de una cookie)

POST /EjemploServlet/Cookies

Connection: Keep-Alive

User-Agent: Mozilla/4.5 (compatible;Windows NT)

Host: miServidor.lcc.uma.es

Accept: image/gif, text/html

Cookie: \$Version="1";Cesta=chocolate;\$Path="/EjemploServlet"
item=caramelo

91



Ejecución del Ejemplo

Servidor -> Cliente (establecimiento de una cookie)

HTTP/1.1 200 OK

Server: Apache/2.3b3

Mime-Version: 1.0

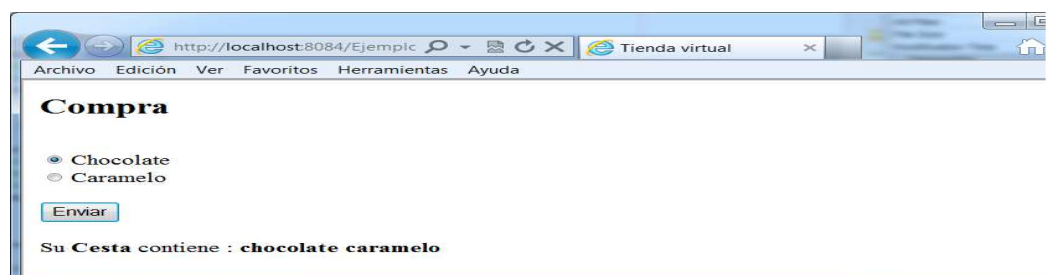
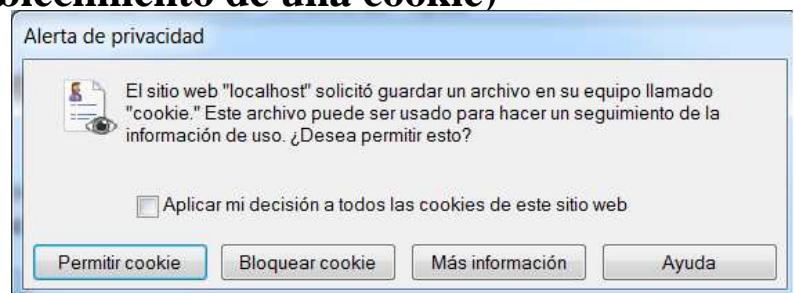
Content-type: text/html

Set-Cookie: Cesta="chocolate caramelo";Version="1";
Path="/EjemploServlet"

<HTML>

...

</HTML>





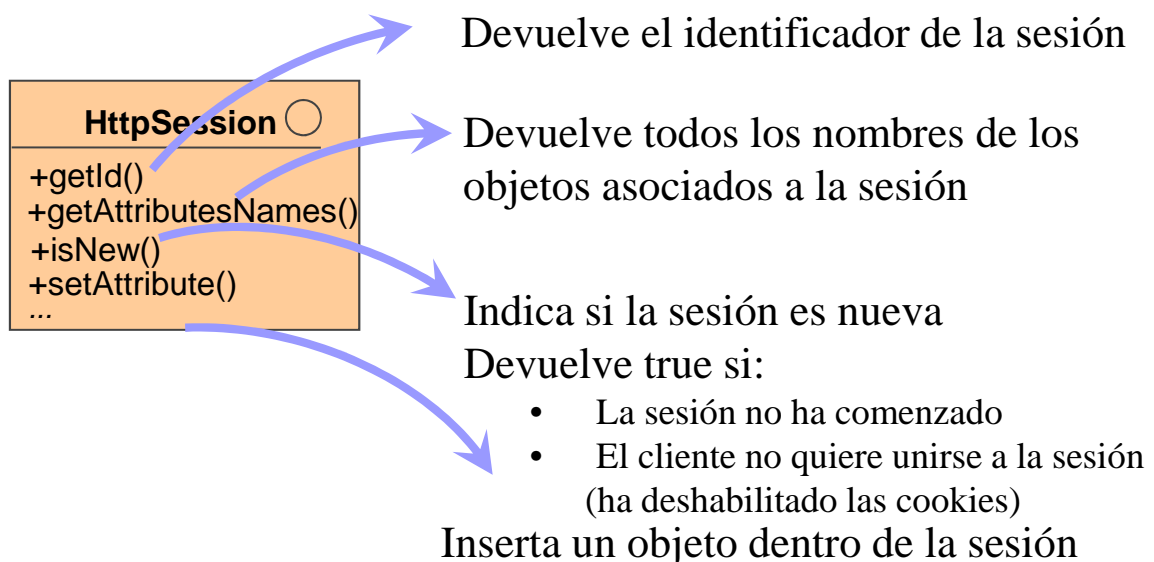
Sesiones

- “Conexión” continuada de un mismo cliente a un servidor
- La duración depende del servidor aunque puede controlarse
- El cliente debe ser capaz de soportarlas
- Los *servlets* de un servidor comparten las sesiones, pero el ID de sesión identifica a un usuario
- Las sesiones pueden contener objetos

93



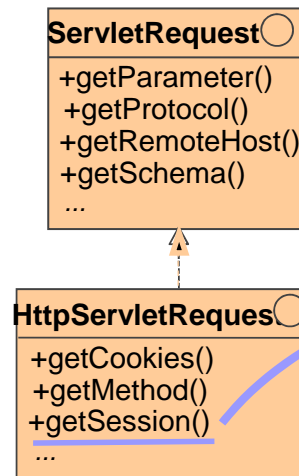
La interfaz HttpSession



94



Petición y Respuesta



Devuelve la sesión actual
o crea una nueva

doGet ó doPost(hrq: HttpServletRequest, hrp HttpServletResponse)

95



Ejemplo Sesiones

```
<h2>Compra</h2>
<form ACTION="http://localhost:8084/EjemploServlet/Sesiones" METHOD="POST">
<p>
<br><input TYPE="RADIO" CHECKED NAME="item" VALUE="chocolate">Chocolate
<br><input TYPE="RADIO" NAME="item" VALUE="caramelo">Caramelo
<p><input TYPE="SUBMIT" NAME="botonEnviar" VALUE="Enviar"></form>
</body>
</html>
```

96



Ejemplo Sesiones

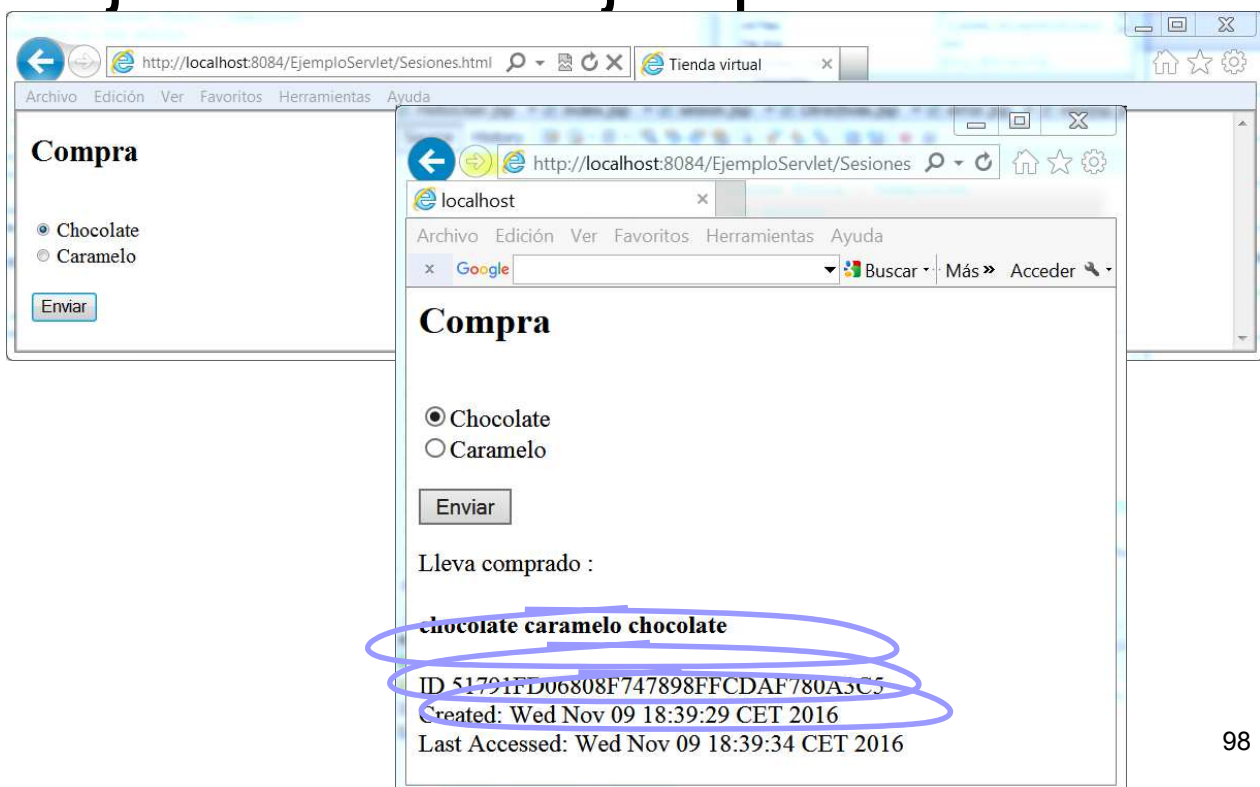
```
String item = hrq.getParameter("item");
HttpSession session = hrq.getSession(true); //Para ese usuario concreto
if (session.isNew()) {
    session.setAttribute(session.getId(), item);
}
else {
    String acumulado = (String) session.getAttribute(session.getId());
    acumulado = acumulado + " " + item;
    session.setAttribute(session.getId(), acumulado);
}

out.println("<html><body>");
out.println("<form ACTION=\"\"/EjemploServlet/Sesiones\" METHOD=\"\"POST\">");
//Resto formulario
out.println("<p>Lleva comprado : <h4>");
out.println(session.getAttribute(session.getId()) + "</h4>");
Date created = new Date(session.getCreationTime());
Date accessed = new Date(session.getLastAccessedTime());
out.println("<p>ID " + session.getId());
out.println("<br>Created: " + created);
out.println("<br>Last Accessed: " + accessed);
```

97



Ejecución del ejemplo



98



Reescritura de URLs

- Clientes (*browsers*) que no soportan *cookies*
- Opción de *cookies* desactivada en el cliente
 - ☐ Sólo reescritura de URLs
 - ☐ Podrían usarse conjuntamente con sesiones
- Implementación
 - ☐ `HttpServletResponse.encodeURL()` (codifica una URL incluyendo un identificador de sesión)
 - ☐ `HttpServletResponse.encodeRedirectURL()` (codifica una URL para usar en el método `sendRedirect()`, que envía una respuesta poniendo dicha URL)

99



Ejecución del ejemplo





Ejemplo Reescritura URLs

```
public void doGet(HttpServletRequest hrq, HttpServletResponse hrp)
    throws IOException, ServletException {

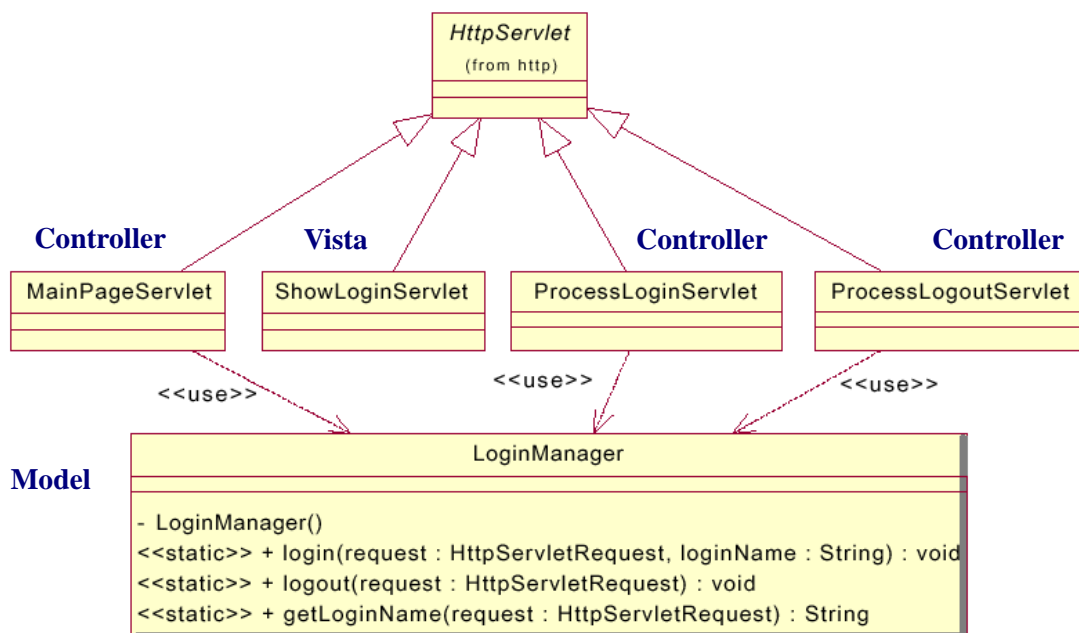
    hrp.setContentType("text/html");
    ServletOutputStream out = hrp.getOutputStream();
    ...
    String nombre=hrq.getParameter("sesion");
    if (nombre == null) {
        nombre = "Id"+System.currentTimeMillis();
        out.println("Se le ha asignado un identificador por "+
            "reescritura de URLs");
    }

    out.println("Identificador aleatorio: "+nombre);
    out.println("<p><a href='"+hrp.encodeURL
        ("/EjemploServlet/ReescrituraURLs?sesion="+nombre)+"'>Pulse aqui</a>");
    ...
}
```

101



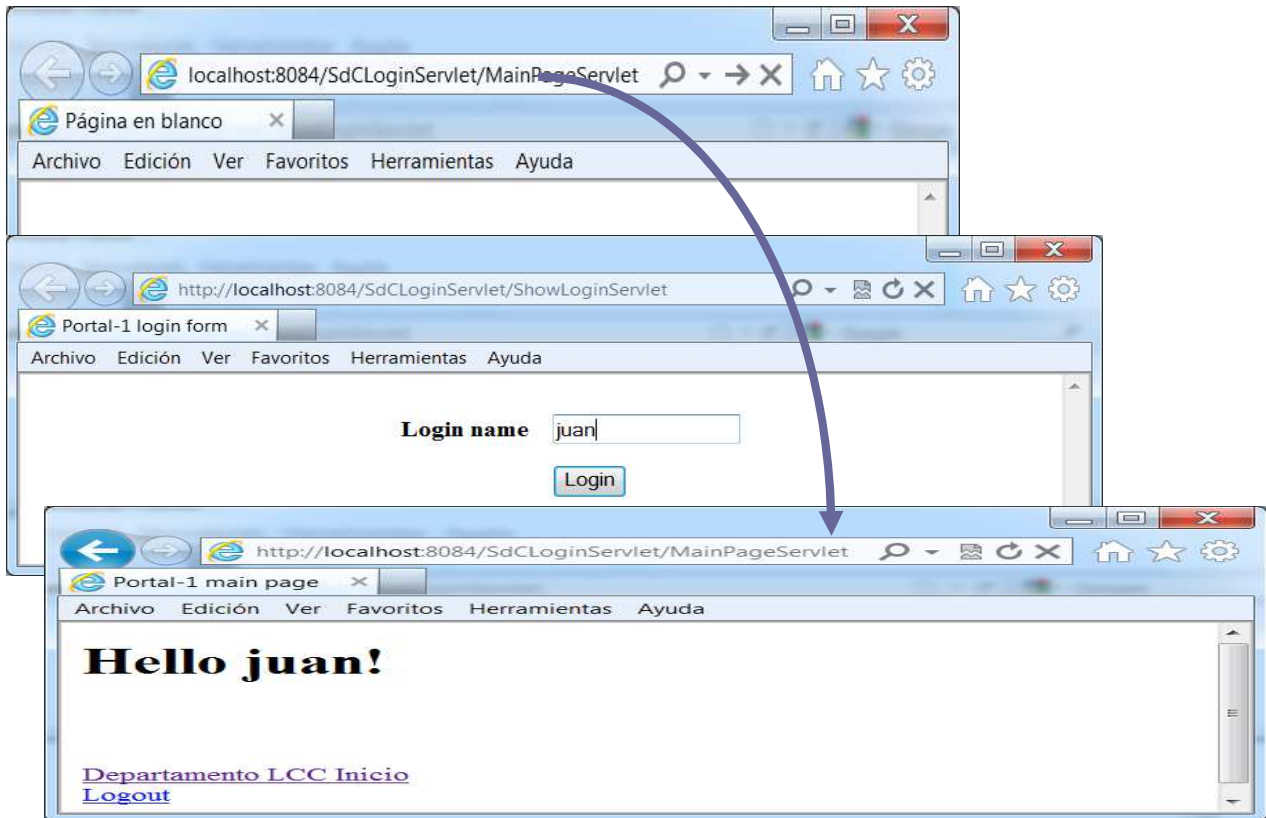
Sesión con Login



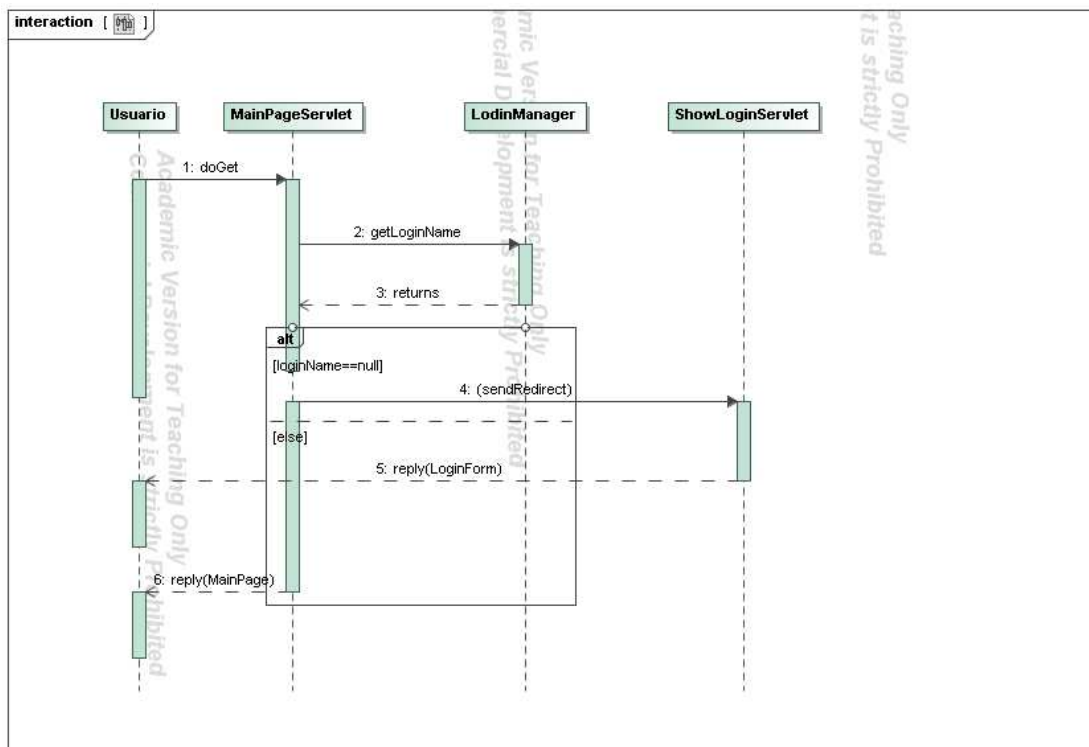
102



Sesión con Login

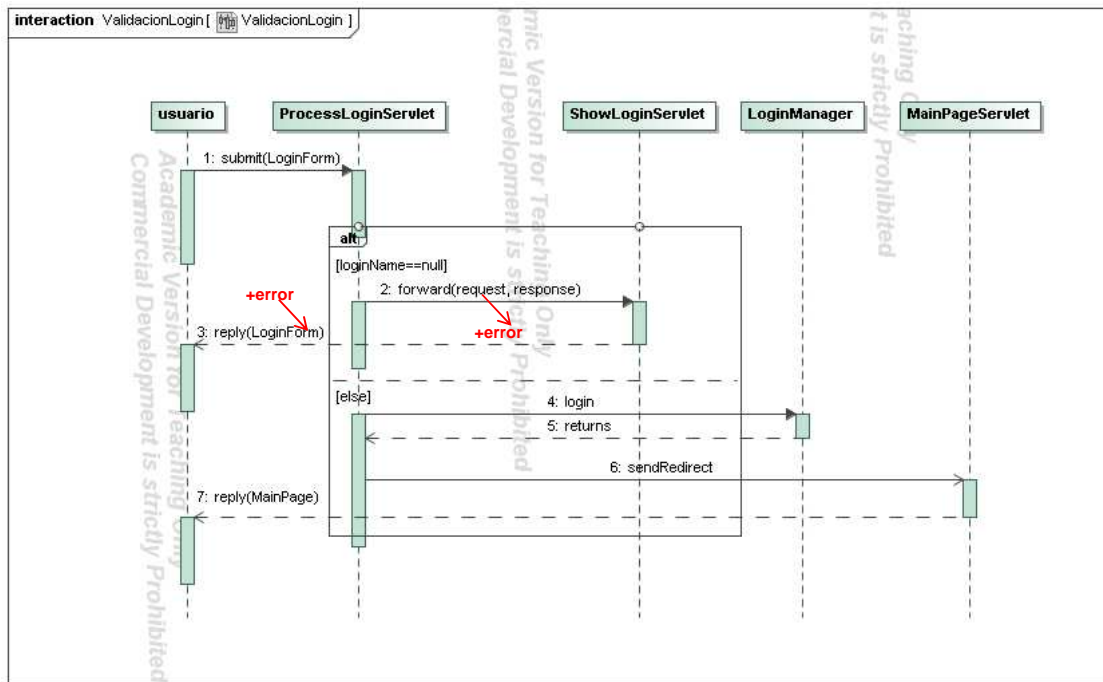


Autenticación





Validación Login



105



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

```
public class MainPageServlet extends HttpServlet {
```

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

```
        String loginName = LoginManager.getLoginName(request);
```

```
        if (loginName != null) { // El usuario ya se ha autenticado
```

```
            generateMainPage(response, loginName);
```

```
        } else { // El usuario no se ha autenticado aun
```

```
            response.sendRedirect("/servlet/ShowLoginServlet");
```

```
        }
```

```
    }
```

```
private void generateMainPage(HttpServletResponse response,String loginName) throws IOException
```

```
{
```

```
    response.setContentType("text/html; charset=ISO-8859-1");
```

```
    PrintWriter out = response.getWriter();
```

```
    out.println("<html><head><title>");out.println("Pagina Principal");out.println("</title></head>");
```

```
    out.println("<body text=\"\#000000\" bgcolor=\"\#ffffff\" " +
```

```
        "link=\"\#000ee0\" vlink=\"\#551a8b\" alink=\"\#000ee0\">");
```

```
    out.println("<h1>Hello " + loginName + "! </h1>");out.println("<br><br><br>");
```

```
    out.println("<a href=\"\http://www.lcc.uma.es\">Departamento LCC " + "Inicio</a><br>");
```

```
    out.println("<a href=\"\ProcessLogoutServlet\">Logout</a><br>");
```

```
    out.println("</body></html>");out.close();
```

```
}
```

```
}
```

106



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

public final class LoginManager {
    private final static String LOGIN_NAME_ATTRIBUTE ="loginName";

    private LoginManager() {}
    public final static void login(HttpServletRequest request,String loginName) {
        HttpSession session = request.getSession(true); // Crea una sesion nueva

        session.setAttribute(LOGIN_NAME_ATTRIBUTE, loginName); //ej: loginName="juan"
    }
    public final static void logout(HttpServletRequest request) {
        HttpSession session = request.getSession(false);
        if (session != null) {
            session.invalidate(); //Destruye la sesion actual
        }
    }
    public final static String getLoginName(HttpServletRequest request) {
        HttpSession session = request.getSession(false);
        if (session == null) {
            return null;
        } else {

            return (String) session.getAttribute(LOGIN_NAME_ATTRIBUTE);
        }
    }
}
```

107



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

public class ShowLoginServlet extends HttpServlet {
    public void doGet(HttpServletRequest request,HttpServletResponse response)
        throws IOException, ServletException {
        String loginName = "";
        String loginNameErrorMessage = "";

        /* Obtener los errores obtenidos por ProcessLoginServlet*/
        Map errors = (Map) request.getAttribute("errors");
        if (errors != null) {
            String errorHeader = "<font color=\"red\"><b>";
            String errorFooter = "</b></font>";
            if (errors.containsKey("loginName")) {
                loginName = request.getParameter("loginName");
                loginNameErrorMessage = errorHeader +errors.get("loginName") + errorFooter;
            }
        }
    }

    /* SIGUE SIGUIENTE PÁGINA*/
}
```

108



```

/* DE LA PÁGINA ANTERIOR */
/* Generar respuesta. */
response.setContentType("text/html; charset=ISO-8859-1");
PrintWriter out = response.getWriter();

/* Inicio y cabecera HTML. */
out.println("<html><head><title>");out.println("Formulario de Login");out.println("</title></head>");
out.println("<body text=\"#000000\" bgcolor=\"#ffffff\">");

/* Comienzo del formulario. */
out.println("<form method=\"POST\" action=\"ProcessLoginServlet\">");
/* Comienzo tabla. */
out.println("<table width=\"100%\" border=\"0\" align=\"center\" \" +\"cellspacing=\"12\">");

/* Login. */
out.println("<tr>");out.println("<th align=\"right\" width=\"50%\"> Login name </th>");
out.println("<td align=\"left\"> \" +\"<input type=\"text\" name=\"loginName\" \" +\" value=\"\" + loginName
+ \"\" size=\"16\" maxlength=\"16\"> \" +loginNameErrorMessage + "</td>");

/* Boton de Login. */
out.println("<tr>");out.println("<td width=\"50%\"></td>");out.println("<td align=\"left\" width=\"50%\"> \"
+\"<input type=\"submit\" value=\"Login\"></td>");out.println("</tr>");
out.println("</table>");out.println("</body></html>");
out.close();
}}

```

Login name Mandatory field

Login

109



```

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

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

        String loginName = request.getParameter("loginName");
        if ( (loginName == null) || (loginName.trim().length() == 0) ) {
            Map errors = new HashMap();
            errors.put("loginName", "Mandatory field");
            request.setAttribute("errors", errors);
            forwardToShowLogin(request, response);
        } else {
            LoginManager.login(request, loginName.trim());
            response.sendRedirect("/servlet/MainPageServlet");
        }
    }
    private void forwardToShowLogin(HttpServletRequest request, HttpServletResponse response)
        throws IOException, ServletException {
        RequestDispatcher requestDispatcher =request.getRequestDispatcher("/servlet/ShowLoginServlet");
        requestDispatcher.forward(request, response);
    }
}

```

Login name Mandatory field

Login

110



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

```
public class ProcessLogoutServlet extends HttpServlet {

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

        LoginManager.logout(request);
        response.sendRedirect("/servlet/MainPageServlet");
    }
}
```



Despliegue de un Servicio Web

■ Proyecto XXX

Estructura de ficheros (NetBeans)

□ web

■ WEB-INF

□ web.xml

■ META-INF

■ XXXX.html

■ ...

□ src

■ java

□ Paquete XXX

■ servlet1.java

■ ...

□ Otros directorios internos del NetBeans



Contenido web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="3.0" ...">
  <servlet>
    <servlet-name>HelloWorld</servlet-name>
    <servlet-class>Ejemplos.HelloWorld</servlet-class>
  </servlet>
  ...
  <servlet-mapping>
    <servlet-name>HelloWorld</servlet-name>
    <url-pattern>/HolaMundo</url-pattern>
  </servlet-mapping>
  ...
  <session-config>
    <session-timeout>
      30
    </session-timeout>
  </session-config>
</web-app>
```

nombre paquete Java

Por defecto se pone el mismo nombre que el servlet

Invocación (nombre proyecto: EjemploServlet)

<http://localhost:8084/EjemploServlet/HolaMundo>

Context Path:/EjemploServlet (se puede cambiar)

113



Localización ficheros de datos

■ Depende del servidor Web.

□ Apache Tomcat

- Se almacenan en la carpeta del proyecto. Para saber el path real, simplemente pulsar el botón derecho sobre el nombre del proyecto y en la opción propiedades vendrá el path del proyecto (ej: c:\Proyectos\EjemploServlet\web)

□ GlassFish

- Con este servidor debemos crear un directorio para almacenar el fichero. Concretamente si el path del proyecto es "path", y el nombre del proyecto es nProyecto, el directorio donde se almacenará/creará el fichero de datos sería:
path\build\web\nProyecto (ej:
c:\Proyectos\EjemploServlet\build\web\EjemploServlet). Dentro de este directorio, se creará el fichero de datos (**crear directorio nombre proyecto, sino path=null**).

114



Localización ficheros de datos

- Para que el código sea independiente del servidor Web obtener el path actual:

```
String path =
    request.getServletContext().getRealPath(request.getContextPath());
System.out.println("Real path"+path);
String sFile = path+"\\\" +\"fichero\"; // nombre \"fichero\"
File fichero = new File(sFile);
if (fichero.exists())
    System.out.println(\"fichero abierto\");
else
    try {
        //A partir del objeto File creamos el fichero físicamente
        if (fichero.createNewFile())
            System.out.println(\"El fichero se ha creado correctamente\");
        else
            System.out.println(\"No se ha podido crear el fichero\");
    } catch (IOException ioe) {
        ioe.printStackTrace();
    }
}
```

115



2.4. Páginas JSP (y otras tecnologías similares)

- Arquitectura MVC
- Definición de las páginas JSP
- Elementos de un script JSP
- Directivas y Acciones JSP
- JSP y JavaBeans

116



2.4.1. Arquitectura MVC

■ Motivación

- Las tecnologías vistas hasta ahora mezclan código de funcionalidad con la generación de la página de respuesta
- La desventaja es que el código es difícil de extender

■ ¿Cómo se debe diseñar un servicio Web para que sea **mantenible** y contenga **partes reutilizables**?

- Patrón arquitectónico Model-View-Controller (MVC)
 - F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, M. Stal, Pattern-Oriented Software Architecture: A System Of Patterns, John Wiley and Sons, 1996.

117



Patrón arquitectónico MVC

■ Separación clara entre el **modelo** (los datos) y la **vista** (interfaz gráfica), gracias a un **controlador** (lógica de negocio) que los mantiene desacoplados

■ Ventajas:

- El modelo es reutilizable con distintas vistas (ej.: una vista web y una con interfaz de ventanas)
- Se modifica cada concepto (*concern*) por separado, sin tocar la implementación del resto de conceptos
- División clara de trabajo entre los miembros de un equipo, que estará formado por personas con distintos niveles de especialización

118



MVC en servicios Web Java EE

■ Modelo

- Clases independientes de la vista y el controlador, relacionadas con el tratamiento de datos (BD, ...)
- Normalmente implementadas con JavaBeans
- Clases candidatas a reutilizarse en varios servicios Web

■ Controlador

- Definen la lógica de la aplicación
- Interactúan con el modelo y seleccionan la siguiente vista
- Normalmente es un servlet

■ Vista

- Conjunto de páginas JSP
 - No contienen código Java
 - Sólo visualizan datos

119



2.4.2. Definición de páginas JSP

■ Java Server Pages (JSP) estándar Java EE

- Separa la visualización de la generación de contenido (patrón MVC)
- Etiquetas XML y scriptlets escritos en Java con la lógica de generación de contenido

Hello ! The time is now <%= new java.util.Date() %>
- Páginas con extensión .jsp
- Pueden llamar a JavaBeans o Enterprise JavaBeans

120



Generación de páginas JSP

- Si es la primera vez, el servidor de aplicaciones genera un servlet que implementa:
 - `javax.servlet.jsp.HttpJspPage`a partir de la página JSP, lo compila y lo carga en memoria
- Si no es la primera vez, le pasa la petición al servlet de forma usual
- Si la página se ha modificado desde la última compilación, el servidor se da cuenta, genera el nuevo servlet, lo compila y lo carga de nuevo

121



```
public void _HttpServletRequest request, HttpServletResponse response)
    throws java.io.IOException, javax.servlet.ServletException {
```

index_jsp.java

```
    final javax.servlet.jsp.PageContext pageContext;
    javax.servlet.jsp.JspWriter out = null;
```

```
    try {
        response.setContentType("text/html");
        pageContext = _jspxFactory.getPageContext(this, request, response,
                                                    null, true, 8192, true);
        _jspx_page_context = pageContext;
        out = pageContext.getOut();
        _jspx_out = out;
```

```
        out.write("\n");
        out.write("<html>\n");
        out.write("    <head>\n"); out.write("        \n");
        out.write("        <title>JSP Page</title>\n");
        out.write("    </head>\n");
        out.write("    <body>\n");
        out.write("        <h1>Hello World!</h1>\n");
        out.write("        <b>Fecha actual: </b>");
        out.print( new java.util.Date() ); out.write("\n");
        out.write("    </body>\n");
        out.write("</html>\n");
    } catch (java.lang.Throwable t) {
        ...
    } finally {...}
}
```

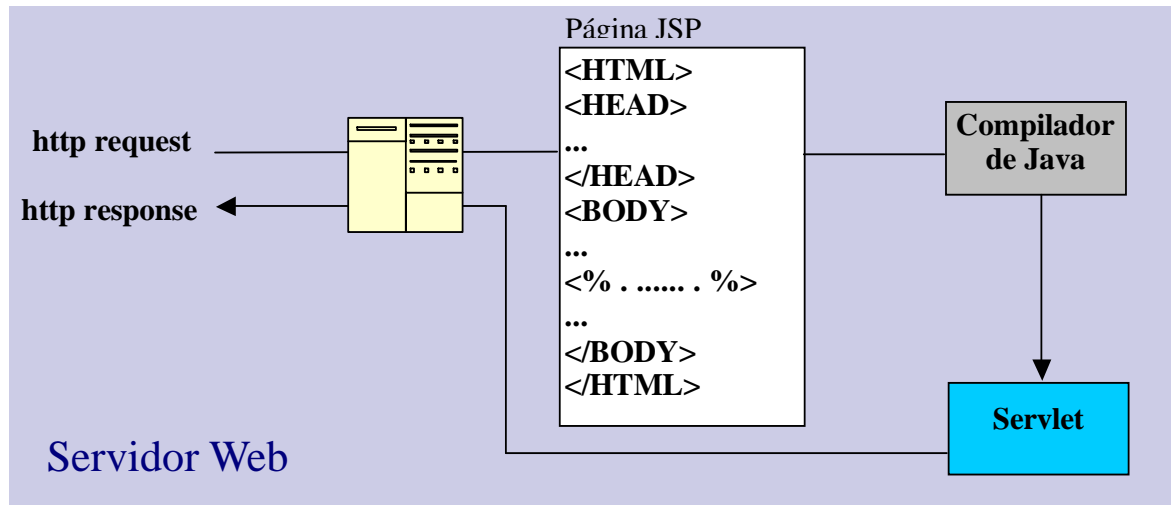
Código generado

En NetBeans con Apache Tomcat

ProyectoJSP/build/generated/src/org/apache/jsp



Generación de páginas JSP



Se pueden poner **comentarios** en una página JSP entre los símbolos `<%--` y `-->`. El contenedor JSP ignorará todo lo contenido entre ambos. Dentro de los fragmentos de código Java también se pueden colocar comentarios siguiendo la sintaxis habitual del lenguaje.

123



2.4. Índice

- Arquitectura MVC
- Definición de las páginas JSP
- Elementos de un script JSP
- Directivas y Acciones JSP
- JSP y JavaBeans



2.4.3. Elementos de un script JSP

- Nos permiten insertar código Java dentro del *servlet* que se generará para la página actual
- Tipos de elementos
 - ☐ Expresiones
 - ☐ Scriptlets
 - ☐ Declaraciones

125



Elementos de un script JSP

- Expresiones
 - `<%= expresion %>` ó `<jsp:exp></jsp:exp>`**
 - ☐ Ej: `<%= new java.util.Date() %>` (el resultado es un string)
 - ☐ Variables predefinidas:
 - request, (el `HttpServletRequest`)
 - response, (el `HttpServletResponse`)
 - session, (el `HttpSession` asociado al request si existe)
 - out, (el `PrintWriter` para enviar la salida al cliente)
 - ej:** `<%= request.getRemoteHost() %>`

126



Index.jsp

```
<html>
<head>
  <meta http-equiv="Content-Type" content="text/html;
  charset=UTF-8">
  <title>Página JSP</title>
</head>
<body>
  <h1>Página JSP con expresiones!</h1>
  <b>-Ruta:</b> <%= request.getContextPath() %>
  <br>
  <b>Método HTTP:</b> <%= request.getMethod() %>
  <br>
  <b>Fecha actual: </b><%= new java.util.Date() %>
</body>
</html>
```



127



Elementos de un script JSP

■ Scriptlets JSP

<% ...código java... %> ó <jsp:scriptlet>...</jsp:scriptlet>

Ej: <% if (request.getParameter("nombre") == null) {
 out.println("Hola, invitado"); }
 else {
 out.println("Hola, "+request.getParameter("nombre"));
 } %>

☐ Tienen acceso a las mismas variables predefinidas que las expresiones

☐ El código del scriptlet se inserta tal cual en el servlet

☐ Al incluir HTML estático se traduce por sentencias print

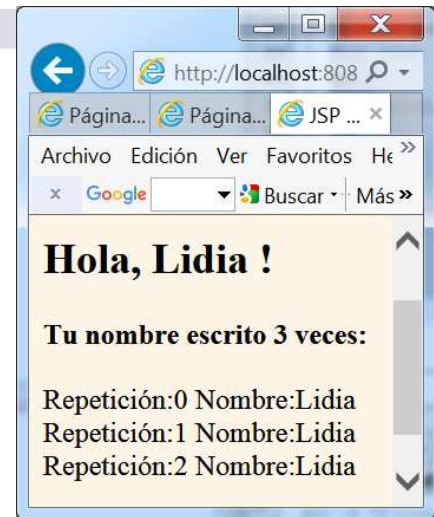
<% if (Math.random()<0.7) { %>	if (Math.random()<0.7) {
Mi mundo!	out.println("Mimundo!");
<% } %>	}

128



HelloUser.jsp

```
<html><head>
  <title>JSP Page</title></head>
<h2>Hola,
<%= request.getParameter("userName") %> !</h2>
<b>Tu nombre escrito 3 veces:</b><br>
<%
String name = request.getParameter("userName");
for (int i=0; i<3; i++) {  %>
  Repetición:<%= i %> Nombre:<%= name %> <br>
<%}
%>
</body></html>
```



129



Elementos de un script JSP

■ Declaraciones JSP

□ `<%! ...código java... %>`

Ej: `<%! private int contAccesos = 0; %>`

Accesos a la pagina:

`<%= ++contAccesos %>`

□ Como no generan ninguna salida, se usan en conjunción con las expresiones o scriptlets

130



HelloUser.jsp

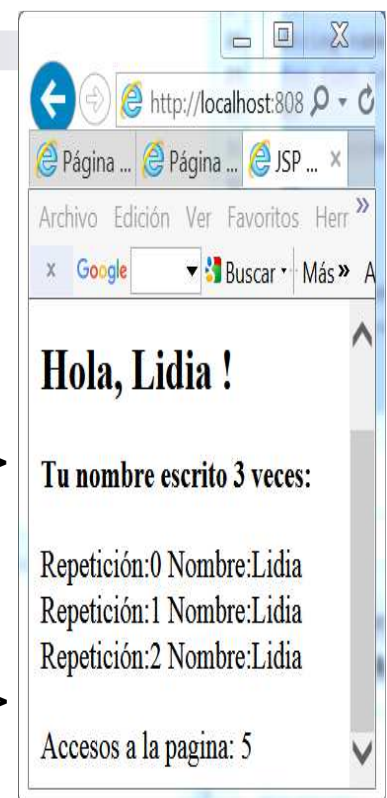
```
<html><head>
  <title>JSP Page</title></head>
<%! private int contAccesos = 0; %>
<h2>Hola,
<%= request.getParameter("userName") %> !</h2>
<b>Tu nombre escrito 3 veces:</b><br>
<%
String name = request.getParameter("userName");
for (int i=0; i<3; i++) {  %>
  Repetición:<%= i %> Nombre:<%= name %> <br>
<%}
%><br>
  Accesos a la pagina:
  <%= ++contAccesos %>
</body>
</html>
```

131



HelloUser.jsp

```
<html><head>
  <title>JSP Page</title></head>
<%! private int contAccesos = 0; %>
<h2>Hola,
<%= request.getParameter("userName") %> !</h2>
<b>Tu nombre escrito 3 veces:</b><br>
<%
String name = request.getParameter("userName");
for (int i=0; i<3; i++) {  %>
  Repetición:<%= i %> Nombre:<%= name %> <br>
<%}
%><br>
  Accesos a la pagina:
  <%= ++contAccesos %>
</body>
</html>
```



132



Evitar código Java en las páginas JSP

- Es difícil depurar el código (se escribe código Java en una página, pero el error surge en el servlet resultante)
- Java es un lenguaje que no es de tipo script, y lo estamos usando como tal
- Debe aplicarse el MVC, separando la visualización de la generación de contenido

133



2.4. Índice

- Arquitectura MVC
- Definición de las páginas JSP
- Elementos de un script JSP
- Directivas y Acciones JSP
- JSP y JavaBeans

134



2.4.4. Directivas JSP

- Una directiva permite controlar distintos parámetros del servlet resultante de la traducción automática de la página JSP

`<%@ directiva atributo1="valor1" ... atributoN="valorN" %>`

- Directiva page

- Nos permite asignar valores a atributos predefinidos

`<%@ page atributo1="valor1" %>` `<jsp:directive.page atr="v"\>`

- Directiva include

- Nos permite incluir ficheros al traducir la página JSP a un servlet

`<%@ include file="url relativa" %>` `<jsp:directive.include f="u"\>`

- Directiva taglib

- Permite insertar etiquetas y la librería que las procesa. Ej: "jsf"

`<%@ taglib uri="http://java.sun.com/jsf/html" prefix="h" %>`

`<h:form id="form1"></h:form>` (los componentes html llevan el prefijo `<h:>`)₁₃₅



Directivas JSP

- Atributos de la Directiva page

`<%@ page atributo1="valor1" %>`

- import

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

- contentType

`<%@ page contentType="text/plain" %>`

- session

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

- errorPage

`<%@ page errorPage="http://localhost/jsp/error.jsp" %>`

`<%@page isErrorPage="true" %>` //incluirlo en la pagina error.jsp

-



```

<%@page contentType="text/html" pageEncoding="UTF-8"%>
<%@page errorPage="error.jsp" %>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML><HEAD><TITLE>Página JSP</TITLE></HEAD>
<P> Contenido dinámico creado por diferentes mecanismos:<UL>
<LI><B>Expresion.</B><BR>
Nombre del servidor: <%= request.getServerName() %>.
<LI><B>Scriptlet.</B><BR>
<% out.println("Datos enviados con la petición GET: " + request.getQueryString()); %>
<% int i;
if (request.getQueryString()==null) {
i=0;
} else
i=1;
int valor=100/i;
%>
<LI><B>Declaracion (mas una expresion).
</B><BR>
<%! private int accessCount = 0; %>
Accesos a la pagina: <%= ++accessCount %>
<LI><B>Directiva (mas una expresion).</B>
<BR>
<%@page import = "java.util.*" %>
Fecha actual: <%= new Date() %>
</UL></BODY></HTML>

```

error.jsp

```

<%@ page import="java.io.PrintWriter"%>
<%@ page isErrorPage="true"%>
<!DOCTYPE html>
<html>
<head><title>JSP Page</title>
</head>
<body>
Ha ocurrido un error:<br/>
<i><%= exception %></i><br/>
El problema se encuentra en:<br/>
<pre>
<% exception.printStackTrace( new PrintWriter( out ) ); %>
</pre>
</body>
</html>

```



Acciones JSP

Normalmente sirven para alterar el flujo normal de ejecución de la página (p.ej. redirecciones), aunque tienen usos variados.

- Acción jsp:include
<jsp:include page="URL relativa" flush="true" />
- Acción jsp:forward
<jsp:forward page="URL relativa" />
- Acción jsp:useBean
<jsp:useBean id="nombre" scope="page|request|session|application"
class="clase.class" type="clase-int.class" />
- Acción jsp:setProperty
<jsp:setProperty name="nombre-Bean" property="*" />
- Acción jsp:getProperty
<jsp:getProperty name="nombre-Bean" property="nombre-propiedad" />



Acción jsp:include

■ Es otra forma de incluir ficheros

- Vimos `<%@ include file="url relativa" %>`
 - Incluye literalmente el fichero dentro del servlet en el momento de la traducción (#include de "C")
- `<jsp:include page="URL relativa" flush="true" />`
 - Ocurre al recibir una petición de la página (request)
 - Se ejecuta el fichero incluido y la salida se incluye como parte de la respuesta final (response)

Ej: `<jsp:include page="jsp/jsp-uno.jsp" flush="true" />`

139



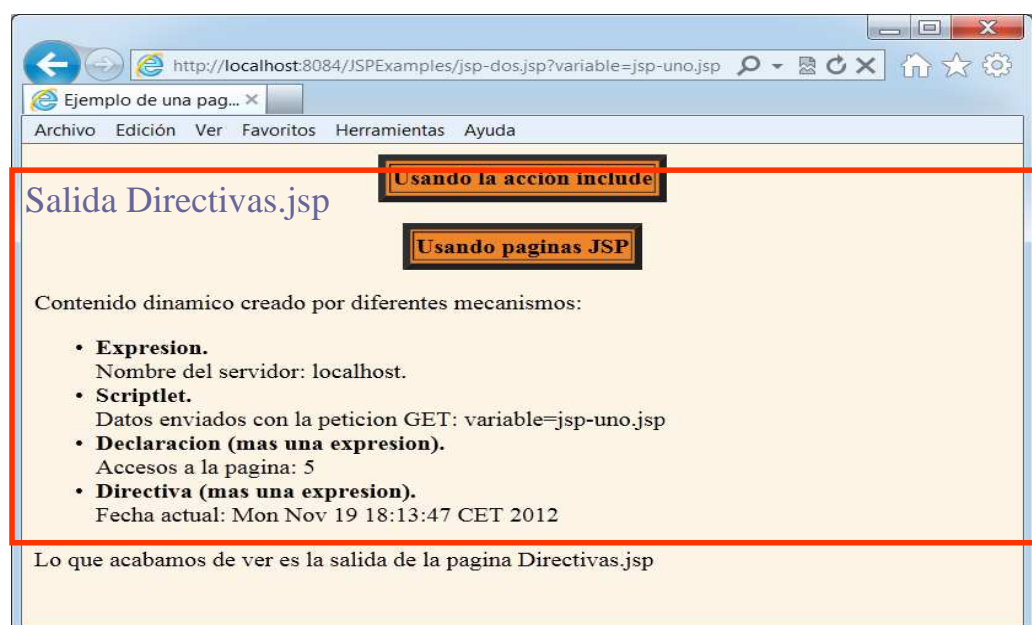
Acción jsp:include

`<table> Usando la acción include </table>`

`<jsp:include page="/jsp/Directivas.jsp" flush="true" />`

Lo que acabamos de ver es la salida de la página Directivas.jsp

....





Acción jsp:forward

```
<jsp:forward page="URL relativa" />
```

- Se pasa el control a otro recurso (página JSP, servlet, ...)
- Permite modularizar páginas JSP

Ej: <% if (session.getAttribute("user") == null) { %>
 <jsp:forward page="/login.jsp" />
 <% } %>



2.4. Índice

- Arquitectura MVC
- Definición de las páginas JSP
- Elementos de un script JSP
- Directivas y Acciones JSP
- JSP y JavaBeans



2.4.5. JSP y JavaBeans

- JSP soporta el uso de JavaBeans como mecanismo de acceder a objetos ligados a la petición `request` y a la sesión `session`
 - Permite desacoplar el modelo (los datos) de la vista (una página JSP) dentro del patrón MVC
 - La comunicación entre una página JSP y un componente (JavaBean) se hace a través de acciones JSP

143



JavaBeans

- **JavaBeans** (Sun Microsystems 1997) es un estándar sobre Java que define el modelo de componentes Sun.
- **Beans**: componentes del modelo
 - Componentes software reutilizables que pueden ser manipuladas de forma visual por herramientas de desarrollo de aplicaciones
 - Granularidad y funcionalidad de las beans muy distintas: botón, hoja de cálculo, etc.

144



JavaBeans

- Inspección y particularización mediante la forma de acceder a sus atributos o *propiedades*. Para cada atributo X de tipo T, la *bean* debe soportar métodos:
 - `public T getX();`
 - `public void setX(T x);`
- Las beans visuales heredan `java.awt.Component`
- Persistencia mediante la secuenciación, proporcionada gracias al paquete `Serialization` de Java.
- Extensiones: Glasgow, Edinburgh, Enterprise Java Beans (EJB).

145



Ejemplo de una Bean

```
public class BeanHola {  
    private String name="Hola, soy una bean";  
  
    public void setName(String name) {  
        this.name=name;  
    }  
  
    public String getName() {  
        return name;  
    }  
}
```

146



Acción jsp:useBean

```
<jsp:useBean id="nombre" scope="page | request | session | application"
  class="clase.class" type="clase-int.class" beanName="bn"
</jsp:useBean>
```

</jsp:useBean>

- id : Nombre de la bean
- scope : Ámbito de visibilidad de la bean. Si es page, sólo estará disponible para la página actual, request, sólo disponible para la petición actual del cliente (almacenada en el request), session, se almacena en la sesión actual (si existe), application, para todas las páginas que compartan el mismo ServletContext
- class : Designa el nombre de la clase de la bean (absoluta, con el path de los packages) **ej: class="com.uma.lcc.sdc.MiBeanImpl"**
- type : nombre de una interfaz o de la superclass (para castings)
ej: type="com.uma.lcc.sdc.MiBean"
- beanName : El atributo class se puede reemplazar por este atributo (se instancia como Beans.instantiate()), esto es, busca primero una versión serializada de la bean)

147



Acción jsp:set/getProperty

■ Acción jsp:get/setProperty

```
<jsp:setProperty name="bean" property="nom-prop" />
```

- Ej: Si una bean tiene la propiedad setSize(int size), y hay un parámetro en la página size=15, entonces se llama a bean.setSize(15)

```
<jsp:setProperty name="bean" property="*" />
```

- Idem para todas las propiedades

■ Acción jsp:setProperty

```
<jsp:setProperty name="bean" property="pr" param="param"/>
```

- Pone la propiedad "pr" al valor del parámetro de la página "param"

```
<jsp:setProperty name="bean" property="pr" value="valor" />
```

- Inicializa la propiedad "pr" al valor "valor"

148



Ejemplo JSP y Beans

```
<%-- jsp-tres.jsp --%>
```

```
<%@ page import="BeanHola" %>
```

```
<jsp:useBean id="Hola" class="BeanHola">
  <jsp:setProperty name="Hola" property="*" />
</jsp:useBean>
```

```
<HTML><HEAD><TITLE>Tercer ejemplo JSP</TITLE>
<CENTER><table BORDER=5 BGCOLOR="#EF8429"><TR><TH
  CLASS="TITLE">JSP y Beans</table>
</CENTER>
<H1>
Voy a saludar: <jsp:getProperty name="Hola" property="name" />
</H1>
</BODY>
</HTML>
```

149



Ejemplo JSP y Beans

```
<%-- jsp-tres.jsp --%>
```

```
<%@ page import="BeanHola" %>
```

```
<jsp:useBean id="Hola" class="BeanHola">
  <jsp:setProperty name="Hola" property="*" />
</jsp:useBean>
```

```
<HTML><HEAD><TITLE>Tercer ejemplo JSP</TITLE>
<CENTER><table BORDER=5 BGCOLOR="#EF8429"><TR><TH
  CLASS="TITLE">JSP y Beans</table>
</CENTER>
<H1>
Voy a saludar: <jsp:getProperty name="Hola" property="name" />
</H1>
</BODY>
</HTML>
```

BeanHola.class debe incluirse en el CLASSPATH
O crear un package e importarlo



150



Interacción JSP y Servlets

FORWARD

- Llamar a un servlet desde una página JSP

```
<jsp:forward page="/servlet/jspToServlet" />
```

- Llamar a una página JSP desde un servlet

```
public class servletToJsp extends HttpServlet {  
    public void doGet (HttpServletRequest req,HttpServletResponse resp) {  
        try {  
            req.setAttribute ("servletName", "servletToJsp");  
            getServletConfig().getServletContext().getRequestDispatcher  
                ("/jsp/jsptoserv/hello.jsp").forward(req, resp);  
        } catch (Exception ex) {  
            ex.printStackTrace ();  
        }  
    }  
}
```

151



Interacción JSP y Servlets

SENDREDIRECT

- Llamar a un servlet desde una página JSP

```
<%  
    response.sendRedirect("/servlet/servletToJsp");  
%>
```

- Llamar a una página JSP desde un servlet

```
response.sendRedirect("/jsp/jsptoserv/hello.jsp");
```

152



Tecnologías alternativas (resumen)

■ Active Server Pages (ASP) de Microsoft

- ☐ Lógica de generación de contenido incluida en la página
- ☐ Página incluye código en VBScript or Jscript

```
<% response.write("Hello World!") %>
```
- ☐ Dependiente del servidor Internet Information Server (IIS)

■ PHP3

- ☐ Lenguaje *script*

```
<?php echo "Hello World<p>"; ?>
```
- ☐ Similar a ASP pero de libre distribución
- ☐ Fácil acceso a bases de datos (Oracle, mSQL, ...) 153



Tecnologías alternativas (resumen)

■ Python Server Pages (PSP)

- ☐ Lenguaje de programación Python
- ☐ Página incluye código en Python

```
<html>
  <% import time
  %>
  Hello world, the time is:
  <%=time.strftime("%Y-%m-%d, %H:%M:%S")%>
</html>
```



Ventajas páginas JSP

■ Frente a PHP

- ☐ Lenguaje muy potente para generación dinámica de contenido
 - Basado en Java, por lo que hereda toda su potencia
 - Java es orientado a objetos
- ☐ Mejores herramientas de desarrollo y soporte

■ Frente a ASP

- ☐ Mejor rendimiento porque el código es compilado, no interpretado
- ☐ Lenguaje más potente para generación dinámica de contenido (Java)
- ☐ Independiente de la plataforma

■ Frente a PSP

- ☐ Mejor rendimiento porque el código es compilado, no interpretado

155



¿Usamos páginas JSP?

■ Desventajas páginas JSP

- ☐ No facilita la creación de interfaces web elaboradas
 - Hay que programar muchas líneas de código
 - Se debe usar mucho Javascript para validación de campos, etc.
- ☐ Complejidad de depuración
- ☐ La integración con las JavaBeans es tediosa
- ☐ La posibilidad de incluir código Java rompe el MVC

■ Tecnologías complementarias Java EE

- ☐ JavaServer Faces (JSF)
- ☐ Facelets

156



JavaServer Faces (JSF)

- Orientado al desarrollo de interfaces de usuario
 - Forma parte del estándar Java EE, complementa a las páginas JSP
- Usa las páginas JSP
 - Añade una biblioteca de etiquetas propia parecido al HTML estándar
 - Asocia a cada vista (con formularios) un conjunto de beans Java
 - Predefine procesamientos típicos como por ejemplo la validación, recuperación de elementos, etc.
 - Es extensible
 - Permite introducir Javascript de forma integrada

157



Facelets

- Usan XHTML para crear páginas Web
- Soporta JSF y otras extensiones para la interfaz de usuario
- Compilación más rápida y con comprobaciones dinámicas

158

**helloworld.xhtml**

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
xmlns:f="http://java.sun.com/jsf/core"
xmlns:h="http://java.sun.com/jsf/html">

<h:head>
<title>My First JSF Application</title>
</h:head>
<h:body>
<h3>JSF 2.2 Hello World Example with faces-config.xml</h3>
<h:form>
<h3>Now The time is
    <h:outputLabel>
        #{helloWorldBean.currentTime}
    </h:outputLabel></h3>
Your Name Please?
    <h:inputText value="#{helloWorldBean.name}">
    </h:inputText>
    <h:commandButton value="Say Hello"
        action="#{resultBean.result}">
    </h:commandButton>
</h:form></h:body></html>
```

```
package net.javaonline.jsf;
import javax.faces.bean.*;
import java.io.Serializable;
//use faces-config.xml
```

```
@ManagedBean(name="helloWorldBean")
@RequestScoped
public class HelloWorldBean implements
Serializable {
    private String name;
    private String currentTime;
    public String getCurrentTime() {}
    public void setName(String name) {}
    public String getName() {}
}
```

```
package net.javaonline.jsf;
import javax.faces.bean.*;
import java.io.Serializable;
//use faces-config.xml
```

```
@ManagedBean(name="resultBean")
@RequestScoped
public class ResultBean implements
Serializable {
    public String result() {}
}
```



Más allá de Java EE

■ Frameworks de desarrollo de servicios Web

- Implementan el estándar Java EE
 - Spring Web
 - MVC, acceso a JDBC, manejo de transacciones, etc.
 - Apache Struts
 - Framework *opensource* para desarrollar aplicaciones web Java EE, MVC, etc
 - Apache Wicket, Stripes, etc..
- Tapestry
 - Usa plantillas XML para separar código de la interfaz
 - Basado en componentes y orientado a eventos
- En Python
 - Django, TurboGears, web3py, etc.
- En Ruby
 - WEBrick, Mongrel, Sinatra, etc.



Conclusión

- Hay muchos frameworks para desarrollar servicios web del lado del servidor
- Todos los frameworks de desarrollo de servicios Web usan servlets
 - ¿Por qué?
 - Porque es la API que me permite comunicarme con el contenedor del servicio