

## Nivel de presentación Aplicaciones Web

Autores: Simon Pickin  
Natividad Martínez Madrid  
Florina Almenárez Mendoza  
Pablo Basanta Val

Dirección: Departamento de Ingeniería Telemática  
Universidad Carlos III de Madrid

Versión: 1.0

Agradecimientos: Florina Almenares, IT/UC3M  
Natividad Martínez Madrid, IT/UC3M  
Simon Pickin, IT/UC3M



Software de  
Comunicaciones

## Contenido

1. Java Servlets
2. Java Server Pages (JSPs)
3. Integración de servlets y JSPs

### Bibliografía:

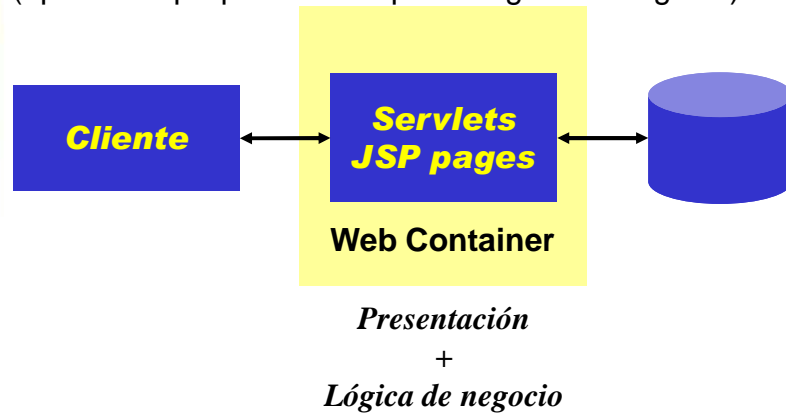
- **Core Servlets and JavaServer Pages.** Marty Hall and Larry Brown. Second Edition. Prentice Hall. 2004
- **Java for the Web with Servlets, JSP, and EJB.** Budi Kurniawan. New Riders. 2002. Part I, capítulos 1-5, 8-11, 17
- **Tecnologías de servidor con Java: Servlets, JavaBeans, JSP.** Ángel Esteban. Grupo EIDOS. 2000



Software de  
Comunicaciones

## Arquitectura de una aplicación Web Servlets/JSPs

Aplicación con una arquitectura “**Three-Tier**”  
(aplicación pequeña sin capa de lógica de negocio)



3

## Nivel de presentación: Java Servlets



4

## Contenido

- Generalidades
  - Introducción
  - Ventajas
  - Ciclo de vida
- API de Servlets
  - Interfaces, clases y métodos
  - Servlets HTTP
  - *Forwarding / Including*
  - Gestión de Sesiones (*Session Tracking*)



Software de  
Comunicaciones

5

## Introducción a los Servlets (1/2)

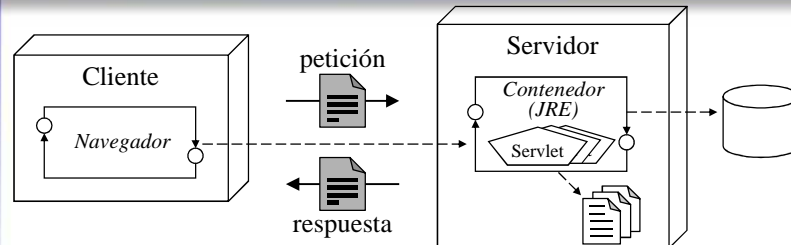
- Un **servlet** es una clase java usada para extender las capacidades de los servidores que albergan aplicaciones accedidas mediante un modelo de programación cliente-servidor
  - Usado para extender las capacidades de la web
- Comparable a un programa CGI (*Common Gateway Interface*)
  - pero con una arquitectura de ejecución diferente
- Gestionados por un contenedor de servlets o un motor
  - JVM + implementación del API del servlet



Software de  
Comunicaciones

6

## Introducción a los Servlets (2/2)



Fuente: Web Component Development With Servlet and JSP Technologies  
Sun Microsystems (course SL-314-EE5)

- **Interfaces y clases**

- Paquetes `javax.servlet` y `javax.servlet.http`

- Todos los servlets tienen que implementar el interfaz **Servlet**, que define los métodos de ciclo de vida, o bien heredar de la clase:

- `GenericServlet` para implementar servicios genéricos.
- `HttpServlet` para manejar servicios HTTP específicos.
- `extends GenericServlet`



Software de  
Comunicaciones

7

## Ventajas de utilizar servlets (1/2)

- **Eficiencia**

- Un hilo por cada petición pero una única instancia de cada servlet
  - Ventajas en rendimiento: no hay retrasos en las peticiones.
  - Ventajas espaciales: menor consumo de memoria
  - Escalabilidad
- El servlet mantiene su estado entre diferentes invocaciones:
  - conexiones a bases de datos, conexiones de red, etc.
- Ejecución de peticiones mediante la invocación de un método.

- **Utilidades para realizar las típicas tareas de servidor**

- logging, gestión de errores, cookies, sesiones, ...

- **Comunicación**

- Manera estandarizada de comunicación con el servidor
- Los servlets pueden compartir datos
  - Permite la creación de *pool/s* para acceder a la base de datos, etc



Software de  
Comunicaciones

8

## Ventajas de utilizar servlets (2/2)

- Ventajas de Java
  - Gran número de APIs: JDBC, hilos, RMI, red, etc.
  - Portabilidad entre plataformas y servidores
  - Seguridad:
    - máquina virtual, chequeo de tipos, gestión de memoria, excepciones, etc.
    - Gestor de seguridad Java
  - Orientación a objetos
  - Gran comunidad de desarrolladores
  - Disponibilidad de código externo



Software de  
Comunicaciones

9

## Ciclo de vida del servlet

- Instanciación e inicialización (en la primera petición)
  - Si no existen instancias del servlet, el contenedor web:
    - Carga la clase del servlet
    - Crea una instancia
    - Inicializa la instancia del servlet llamando a `init`
- Manejo de sucesivas peticiones
  - El contenedor crea un hilo que llama al método `service` de la instancia
  - El método `service` determina lo que ha llegado en la petición y llama a un método apropiado
- Destrucción
  - Cuando el contenedor decide destruir el servlet, llama a su método `destroy`



Software de  
Comunicaciones

10

## Consecuencias del ciclo de vida del servlet (1/2)

- Una única máquina virtual:
  - Compartición de datos entre varias instancias
- Persistencia (en memoria) de las instancias
  - Consumo de memoria reducido
  - Eliminación de los tiempos de inicialización e instanciación
  - Persistencia (en memoria) del estado, los datos y los recursos
    - Atributos persistentes del servlet
    - Conexiones a bases de datos persistentes, etc
  - Persistencia (en memoria) de los hilos



Software de  
Comunicaciones

11

## Consecuencias del ciclo de vida del servlet (2/2)

- Peticiones concurrentes
  - Se necesita de sincronización para manejar el acceso concurrente
    - Clases, instancias de atributo, bases de datos, etc
  - Si el hilo implementa la interfaz `SingleThreadModel`
    - No existe acceso concurrente a las instancias de los atributos (puede haber acceso concurrente a los atributos de la clase)
    - Puede minar el rendimiento de la máquina virtual
    - Ha sido marcado como obsoleto (*deprecated*) desde la versión 2.4



Software de  
Comunicaciones

12

## Contenido: Servlets Java

- Generalidades
  - Introducción
  - Ventajas
  - Tareas de los servlets
  - Ciclo de vida
- API de Servlets
  - Interfaces, clases y métodos
  - Servlets HTTP
  - Forwarding / Including
  - Gestión de Sesiones



Software de  
Comunicaciones

13

## API de Servlets

- Paquetes
  - `javax.servlet`
- 7 interfaces
  - `Servlet`
  - `ServletConfig`
  - `ServletContext`
  - `ServletRequest`
  - `ServletResponse`
  - `SingleThreadModel`
  - `RequestDispatcher`
- 3 clases
  - `GenericServlet`
  - `ServletInputStream`
  - `ServletOutputStream`
- 2 clases de excepciones
  - `ServletException`
  - `UnavailableException`



Software de  
Comunicaciones

14

## Interfaz Servlet Métodos (1/2)

- **void init(ServletConfig config)**
  - Sólo se llama una vez después de instanciar el servlet
  - El servlet puede instanciarse según como se haya registrado:
    - Cuando el primer usuario accede a la URL del servlet
    - O bien cuando se arranca el servidor Web
  - **Sin argumentos:** inicialización independiente del servidor
    - Inicialización de variables, conexión a base de datos, etc
  - **Con argumentos:** inicialización dependiente del servidor
    - Información obtenida del descriptor de despliegue `web.xml` (desde la especificación 2.3) y almacenado en un objeto `ServletConfig`
    - Configuración de base de datos, ficheros de password, parámetros de prestaciones del servidor, etc.
- **void service(ServletRequest req, ServletResponse res)**
  - Es invocado por el contenedor para permitir que el servlet responda a una petición



15

## Interfaz Servlet Métodos (2/2)

- **void destroy()**
  - El contenedor puede decidir descargar una instancia de un servlet
    - Decisión del administrador
    - Timeout: demasiado tiempo inactivo
  - Previamente llama al método `destroy`
    - Cerrar conexiones a bases de datos
    - Parar hilos
    - Escribir cookies o contador de impactos (hits) a disco
    - ...
  - Si se cae el servidor Web, no se llama al método `destroy`
    - Conclusión: mantener el estado de manera proactiva (guardar los trastos de forma de regular)



16



## Interfaz ServletConfig (1/3)

- Objeto de configuración usado por el contenedor para pasar información al servlet durante la inicialización
  - Se recupera del descriptor de despliegue `web.xml`
  - Por cada servlet registrado, se pueden especificar un conjunto de parámetros iniciales (nombre/valor)

```
<web-app>
  <servlet>
    <servlet-name>ConfigExample</servlet-name>
    <servlet-class>ConfigExampleServlet</servlet-class>
    <init-param>
      <param-name>adminEmail</param-name>
      <param-value>admin@it.uc3m.es</param-value>
    </init-param>
    <init-param> . . . </init-param>
  </servlet>
  . . .
</web-app>
```



Software de  
Comunicaciones

17

## Interfaz ServletConfig (2/2)

- Ejemplo: sobrescribir el método `init` para imprimir la información contenida en el objeto `ServletConfig`

```
public void init(ServletConfig config) throws ServletException
{
    Enumeration parameters = config.getInitParameterNames();
    while (parameters.hasMoreElements()) {
        String parameter = (String) parameters.nextElement();
        System.out.println("Parameter name : " + parameter);
        System.out.println("Parameter value : " +
            config.getInitParameter(parameter));
    }
}
```



Software de  
Comunicaciones

18

## Interface ServletConfig (3/3)

- Si el método `init` (con parámetros) de la interfaz `Servlet` (implementado en la clase `GenericServlet`) es redefinido, el objeto `ServletConfig` no será salvado y no estará disponible **después** de la inicialización.
- Solución: o bien llamar a `Servlet.init(super.init)` si se extiende la clase `GenericServlet` o `HttpServlet` desde dentro del `init` redefinido o si explícitamente se salva:

```
ServletConfig servlet_config;  
public void init(ServletConfig config) throws  
    ServletException {  
    servlet_config = config;  
}
```

La ventaja de esta última solución es que en este caso el objeto `ServletConfig` estará disponible a través del método `getServletConfig` mientras que la segunda solución no lo estará.



Software de  
Comunicaciones

19

## Interfaz ServletContext

- Define un conjunto de métodos usados por el servlet para comunicarse
  - Con su contenedor (obtener el tipo MIME de un fichero, repartidores de peticiones ("*dispatcher*"), etc.)
  - Con otros servlets de la misma aplicación Web
- Hay un contexto
  - Por cada aplicación Web
  - Por cada JVM
- Aplicación Web
  - colección de servlets, JSPs y otros recursos instalados en un subconjunto específico (subdirectorio) del espacio de nombres del servidor
- La información sobre la aplicación web a la que pertenece un servlet se almacena en el objeto `ServletConfig`



Software de  
Comunicaciones

20

## Atributos de ServletContext

- El contexto se obtiene a partir de la configuración

```
ServletContext sc =  
    Servlet.getServletConfig().getServletContext();
```

- Los objetos se almacenan como atributos, identificándolos por un nombre

```
sc.setAttribute("miObjeto", objeto);
```

Si existiera el nombre, el contexto se actualiza con el contenido del nuevo objeto

- Cualquier servlet en el mismo contexto puede recuperar el objeto que hemos almacenado

```
MiClase mc = (MiClase)sc.getAttribute("miObjeto");
```

- Se puede recuperar una colección con los nombres de atributos almacenados

```
Enumeration att = sc.getAttributeNames();
```



Software de  
Comunicaciones

21

## Interfaces ServletRequest y ServletResponse

- Objetos creados por el contenedor y pasados como argumentos a los métodos de servicio

- Interfaz **ServletRequest** encapsula información acerca de la petición del usuario

- Incluye parámetros, atributos y un stream de entrada

- Métodos: `getParameterNames()`, `getAttributeNames()`, `getRemoteAddr()`, `getRemoteHost()`, `getProtocol()`, `getContentType()`, ...

- Interfaz **ServletResponse** representa la respuesta al usuario

- Métodos: `getWriter()`, `reset()`, `getBufferSize()`, `getLocale()`, `getOutputStream()`, `isCommitted()`, ...



Software de  
Comunicaciones

22

## Servlets HTTP (`javax.servlet.http`)

- Hereda de `javax.servlet.HttpServlet`
- Implementa `service()`, que invoca al método correspondiente de la petición:
  - `void doGet(HttpServletRequest request, HttpServletResponse response)`
  - `void doPost(HttpServletRequest request, HttpServletResponse response)`
  - `void doXXX(HttpServletRequest request, HttpServletResponse response)`
- No se suele redefinir el método `service()`
- Se suele sobrecargar los métodos `doXXX()`:
  - Para procesar peticiones GET redefine `doGet`



Software de  
Comunicaciones

23

## Métodos `doGet`, `doPost`, `doXXX`

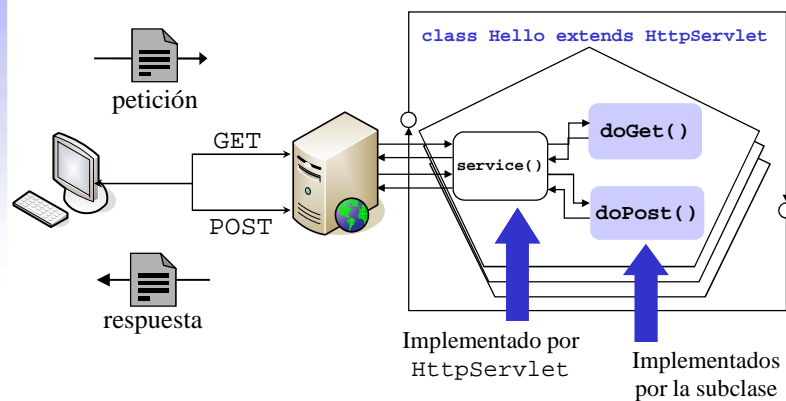
- 99% de las veces el servlet sólo reescribe los métodos `doGet` y `doPost`
- Además: `doDelete`, `doPut`, `doOptions`, `doTrace`
- No hay `doHead`
  - El método `service` llama a `doGet` y devuelve el código de estado y cabeceras, y omite el cuerpo
- `doOptions`, en general, no es necesario definirlo
  - El método `service` le da soporte automático
  - Si existe un método `doGet`, el método `service` devuelve la cabecera `Allow` indicando que soporta GET, HEAD, OPTIONS y TRACE



Software de  
Comunicaciones

24

## HttpServlet



Software de  
Comunicaciones

25

## Tareas de los servlets (1/2)

1. Leer datos enviados por el usuario
  - Típicamente través de un formulario HTML
  - Pero también desde un applet o aplicación cliente
2. Recuperar otra información de usuario embebida en la petición HTTP
  - Capacidades del navegador,
  - cookies,
  - nombre de la máquina del cliente, etc.
3. Generar resultados
  - Cálculo directo de la respuesta,
  - llamando a otro servidor (posiblemente remoto vía RMI o CORBA)
  - accediendo a una base de datos, etc.



Software de  
Comunicaciones

26

## Tareas de los servlets (2/2)

### 4. Formatear los resultados

- En un documento HTML

### 5. Asignar los parámetros de la respuesta HTTP

- Tipo de documento devuelto (HTML)
- Cookies
- Parámetros de cache.

### 6. Enviar el documento al cliente

- En formato texto (e.g.HTML),
- Formato binario (e.g. GIF)
- Comprimido (e.g. gzip)



Software de  
Comunicaciones

27

## Plantilla de servlet básico

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

public class ServletTemplate extends HttpServlet {

    // Use "request" to read incoming HTTP headers (e.g. cookies)
    // and HTML form data (e.g. data user entered and submitted).
    // Use "response" to specify the HTTP response status code
    // and headers (e.g. the content type, cookies).
    public void doGet(HttpServletRequest request,
                      HttpServletResponse response)
        throws ServletException, IOException {

        // Use "out" to send content to browser.
        PrintWriter out = response.getWriter();
    }
}
```



Software de  
Comunicaciones

28

## Ejemplo 1: Generación de texto (1/2)

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

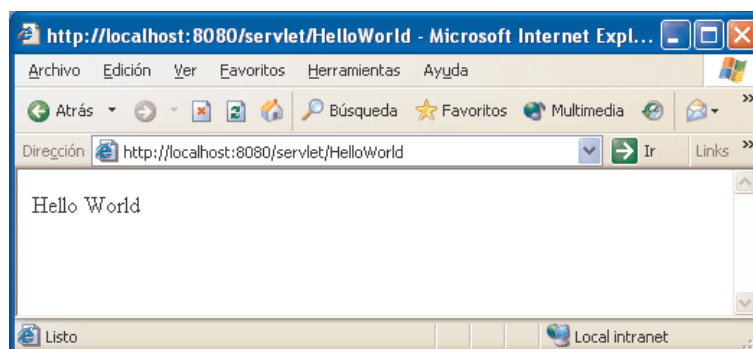
public class HelloWorld extends HttpServlet {
    public void doGet(HttpServletRequest request,
        HttpServletResponse response)
        throws ServletException, IOException {
        PrintWriter out = response.getWriter();
        out.println("Hello World");
    }
}
```



Software de  
Comunicaciones

29

## Ejemplo 1: Generación de texto (2/2)



Software de  
Comunicaciones

30

## Ejemplo 2: Generación de HTML (1/2)

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

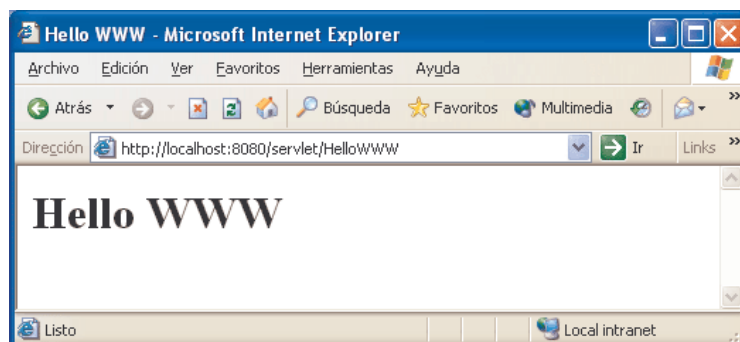
public class HelloWWW extends HttpServlet {
    public void doGet(HttpServletRequest request,
        HttpServletResponse response)
        throws ServletException, IOException {
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        String docType =
            "<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "
            + "\"http://www.w3.org/TR/1999/REC-html401-19991224/loose.dtd\" >\n";
        out.println(docType +
            "<HTML>\n" +
            "<HEAD><TITLE>Hello WWW</TITLE></HEAD>\n" +
            "<BODY>\n" +
            "<H1>Hello WWW</H1>\n" +
            "</BODY></HTML>");
    }
}
```



Software de  
Comunicaciones

31

## Ejemplo 2: Generación de HTML (2/2)



Software de  
Comunicaciones

32



## Lectura de datos de un programa CGI (con el fin de establecer comparaciones)

```
http://host/path?user=Marty+Hal&origin=bwi&dest=lax
```

Datos del formulario/petición (GET)

```
public String getParameter(String name)
```

- Método de `HttpServletRequest` heredado de `ServletRequest`
- Aplica a datos enviados con GET o POST (el servidor conoce cuál)
- **name**: nombre del parámetro cuyo valor es requerido
- valor retornado:
  - Valor decodificado (url-decoded) de la primera ocurrencia de **name**
  - Cadena vacía si el parámetro existe pero no tiene valor
  - Null si el parámetro no existe
- Para parámetros que potencialmente tienen varios valores:
  - `getParameterValues` (devuelve un array de Strings)
- Para obtener una lista completa de parámetros (depuración):
  - `getParameterNames` (retorna una enumeración con los valores que se amoldan a Strings y se usan en llamadas a `getParameter`)



Software de  
Comunicaciones

33

## Reading Form Data from a CGI Program (for Comparison Purposes)

```
http://host/path?user=Marty+Hall&origin=bwi&dest=lax
```

form data / query data (GET)

CGI:

- Métodos diferentes para **GET** y **POST**
- Procesar el "query string" para extraer nombres y valores:
  1. Leer datos de la variable `QUERY_STRING` (GET) o la entrada estándar (POST)
  2. Detectar pares con "&" (separador) y separarlos de los nombres (texto antes de "=") de valores (después de "=")
  3. Decodificar los datos que me pasan
- Tomar en cuenta que puede haber muchos parámetros
  - Cuyos valores pueden ser omitidos
  - Para los cuales múltiples valores son enviados (separadamente)



Software de  
Comunicaciones

34

## Ejemplo 3: Leer 3 parámetros explícitos

```
package coreservlets

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

public class ThreeParams extends HttpServlet {
    public void doGet(HttpServletRequest req, HttpServletResponse res)
        throws ServletException, IOException {
        res.setContentType("text/html");
        PrintWriter out = res.getWriter();
        String title = "Reading Three Request Parameters";
        out.println(ServletUtilities.headWithTitle(title) +
            "<body bgcolor=\"#FDF5E6\">\n" +
            "<h1 align=\"center\">" + title + "</h1>\n <ul>\n" +
            " <li><b>param1</b>: " +
            req.getParameter("param1") + "</li>\n" +
            " <li><b>param2</b>: " +
            req.getParameter("param2") + "</li>\n" +
            " <li><b>param3</b>: " +
            req.getParameter("param3") + "</li>\n" +
            "</ul>\n</body></html>");
    }
}
```



Software de  
Comunicaciones

35

## Ejemplo 3: Clase ServletUtilities

```
public class ServletUtilities {

    public static final String DOCTYPE =
        "<!DOCTYPE HTML PUBLIC \"-//W3C//DTD HTML 4.01 Transitional//EN\" +
        \"http://www.w3.org/TR/1999/REC-html401-19991224/loose.dtd\">";

    public static String headWithTitle (String title)
        return(DOCTYPE + "\n" + "<html>\n" + "<head><title>" + title +
            "</title></head>\n");
}
}
```



Software de  
Comunicaciones

36

## Ejemplo 3: Formulario HTML

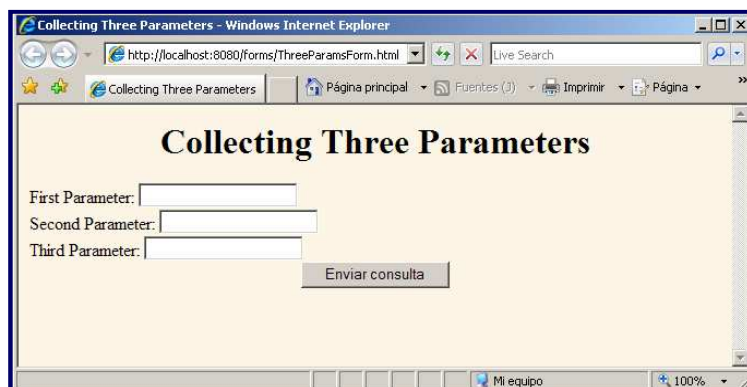
```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/1999/REC-html401-19991224/loose.dtd">
<html>
<head><title>Collecting Three Parameters</title></head>
<body bgcolor="#FDF5E6">
  <h1 align="center">Collecting Three Parameters</h1>
  <form action="/servlet/coreservlets.ThreeParams">
    First Parameter: <input type="text" name="param1"><br />
    Second Parameter: <input type="text" name="param2"><br />
    Third Parameter: <input type="text" name="param3"><br />
    <center><input type="submit" value="Enviar consulta"></center>
  </form>
</body>
</html>
```



Software de  
Comunicaciones

37

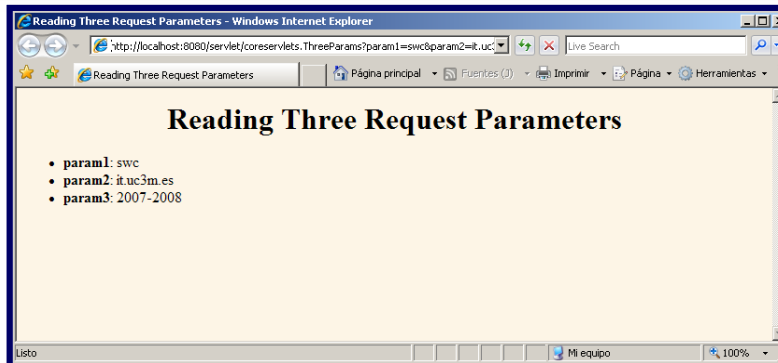
## Ejemplo 3: Apariencia del formulario HTML



Software de  
Comunicaciones

38

## Ejemplo 3: Respuesta del Servlet



Software de  
Comunicaciones

39

## Ejemplo 4: Lectura de todos los parámetros (1/3)

```
package coreservlets

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

public class ShowParameters extends HttpServlet {

    public void doGet(HttpServletRequest req, HttpServletResponse res)
        throws ServletException, IOException {
        res.setContentType("text/html");
        PrintWriter out = res.getWriter();
        String title = "Reading All Request Parameters";
        out.println(ServletUtilities.headWithTitle(title) +
            "<body bgcolor=#FDF5E6>\n" +
            "<h1 align=center>" + title + "</h1>\n" +
            "<table border=1 align=center>\n" +
            "<tr bgcolor=#FFAD00>\n" +
            "<th>Parameter name</th><th>Parameter value(s)</th></tr>");

        Enumeration paramnames = req.getParameterNames();
```



Software de  
Comunicaciones

40

## Ejemplo 4: Lectura de todos los parámetros Servlet (2/3)

```
while (paramnames.hasMoreElements()) {
    String paramname = (String)paramnames.nextElement();
    out.print("<tr><td>" + paramname + "</td>\n<td>");
    String[] paramvalues = req.getParameterValues(paramname);

    if (paramvalues.length == 1) {
        String paramvalue = paramvalues[0];
        if (paramvalue.length() == 0)
            out.println("<i>No value</i>");
        else
            out.println(paramvalue);
    } else {
        out.println("<ul>");
        for(int i=0; i<paramvalues.length; i++)
            out.println("<li>" + paramvalues[i] + "</li>");
        out.println("</ul>");
    } // if
    out.println("</td></tr>");
} // while

out.println("</table>\n</body></html>");
}
```



Software de  
Comunicaciones

41

## Ejemplo 4: Lectura de todos los parámetros Servlet (3/3)

```
public void doPost(HttpServletRequest req, HttpServletResponse res)
    throws ServletException, IOException {

    doGet(req, res);
}
}
```



Software de  
Comunicaciones

42

## Ejemplo 4: Formulario HTML

```
<form action="/servlet/coreservlets.ShowParameters" method="POST">

  Item Number: <input type="text" name="itemNum"><br />
  Quantity: <input type="text" name="quantity"><br />
  Price Each: <input type="text" name="price" value="$"><br />
  <hr />
  First name: <input type="text" name="firstname"><br />
  Last name: <input type="text" name="lastname"><br />
  Credit Card:<br />
    &nbsp;&nbsp;&nbsp;<input type="radio" name="cardType"
      value="Visa">Visa<br />
    &nbsp;&nbsp;&nbsp;<input type="radio" name="cardType"
      value="Master Card">Master Card<br />
    &nbsp;&nbsp;&nbsp;<input type="radio" name="cardType"
      value="Amex">American Express<br />
  Credit Card Number: <input type="password" name="cardNum"><br />
  Repeat Credit Card Number:
    <input type="password" name="cardNum"><br />

  <center><input type="SUBMIT" value="Submit Order"></center>

</form>
```



Software de  
Comunicaciones

43

## Ejemplo 4: Apariencia del formulario HTML

A sample FORM using POST

Item Number: 12

Quantity: 100

Price Each: \$50

---

First name: Pepe

Last name: Pérez

Credit Card:

Visa

Master Card

American Express

Credit Card Number: .....

Repeat Credit Card Number: .....

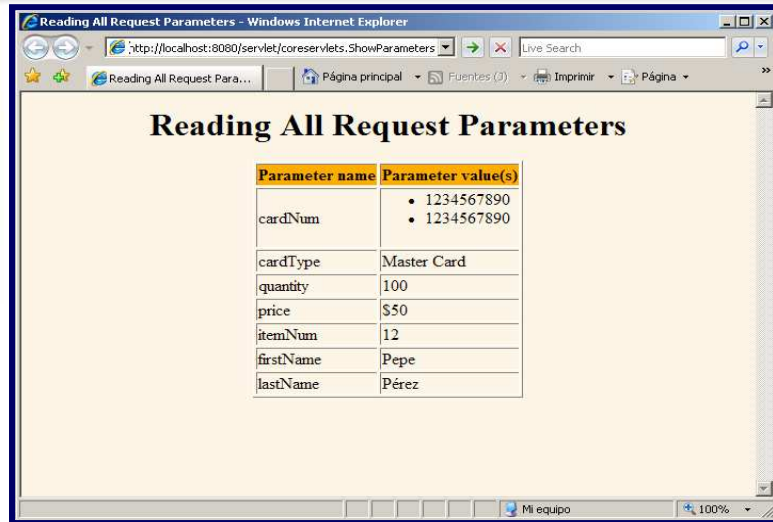
Submit Order



Software de  
Comunicaciones

44

## Ejemplo 4: Respuesta del Servlet



| Parameter name | Parameter value(s)                                                                |
|----------------|-----------------------------------------------------------------------------------|
| cardNum        | <ul style="list-style-type: none"><li>• 1234567890</li><li>• 1234567890</li></ul> |
| cardType       | Master Card                                                                       |
| quantity       | 100                                                                               |
| price          | \$50                                                                              |
| itemNum        | 12                                                                                |
| firstName      | Pepe                                                                              |
| lastName       | Pérez                                                                             |



45

## Manejo de cabeceras de petición Interfaz HttpServletRequest (1/2)

- **String getHeader (String name)**
  - Recibe un String con el nombre de la cabecera (no case sensitive)
  - Devuelve el contenido de la cabecera, o null si no se encuentra
- **Cookie[] getCookies()**
  - Devuelve todos los objetos `Cookie` que el cliente envió junto con la petición en un array de `Cookie`
- **String getAuthType() y String getRemoteUser()**
  - Devuelve los componentes de la cabecera `Authorization`
- **int getContentLength()**
  - Devuelve la cantidad en bits del cuerpo de la petición, o -1 si no se conoce la longitud
- **String getContentType()**
  - Devuelve el valor de la cabecera `Content-Type`



46

## Manejo de cabeceras de petición Interfaz HttpServletRequest (2/2)

- **long getDateHeader (String name) y  
int getIntHeader (String name)**
  - Devuelve el valor de una cabecera de petición como long o int.
  - long es el resultado en milisegundos desde 1970
- **Enumeration getHeaderNames ()**
  - Devuelve una enumeración con todos los nombres de cabeceras recibidos en la petición
- **Enumeration getHeaders (String name)**
  - Devuelve una enumeración con todos los valores de todas las ocurrencias en una cabecera (por ejemplo, Accept-Language puede aparecer varias veces)



Software de  
Comunicaciones

47

## Manejo de primera línea de petición Métodos de HttpServletRequest

- **String getMethod ()**
  - Devuelve el método de la petición (GET, POST, ...)
- **String getRequestURI ()**
  - Devuelve la parte de la URL de la petición entre el host y el puerto y antes de la siguiente petición ( *sq://host.port/path?query\_string*). Por ejemplo, retorna /a/b.html para peticiones HTTP de la siguiente manera:  

```
GET /a/b.html?name=simon HTTP/1.1  
Host: www.it.uc3m.es
```
- **String getProtocol ()**
  - Devuelve el nombre y versión del protocolo en la forma: *protocol/majorVersion.minorVersion*
  - Ejemplo: HTTP/1.1



Software de  
Comunicaciones

48



## Ejemplo 5: Mostrando las cabeceras de petición (1/2)

```
public class ShowHeadersServlet extends HttpServlet {

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

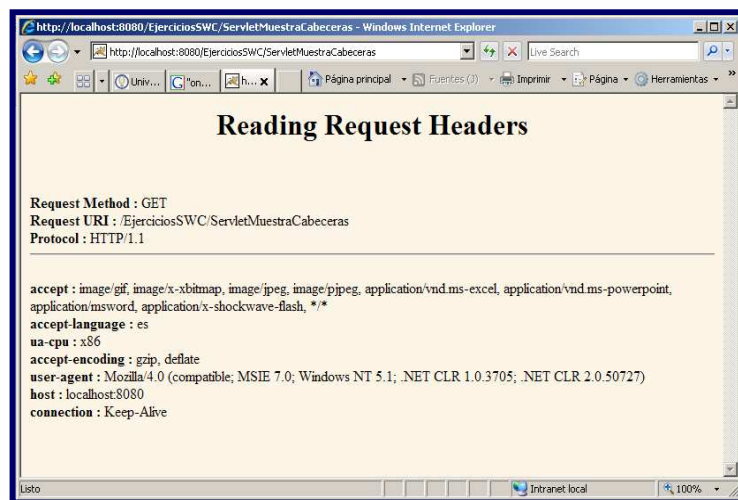
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        out.println("Request Method: " + request.getMethod() +
            "<br />");
        out.println("Request URI: " + request.getRequestURI() +
            "<br />");
        out.println("Protocol: " + request.getProtocol() + "<br />");
        out.println("<hr /><br />");
        Enumeration enumeration = request.getHeaderNames();
        while (enumeration.hasMoreElements()) {
            String header = (String) enumeration.nextElement();
            out.println(header + ": " + request.getHeader(header) + "<br />");
        }
    }
}
```



Software de  
Comunicaciones

49

## Ejemplo 5: Respuesta del Servlet



Software de  
Comunicaciones

50

## Generación de la respuesta

### Métodos de HttpServletResponse

- **void setStatus (int sc)**
  - Importante:
    - línea de estado y cabeceras se pueden poner en cualquier orden
    - pero siempre ANTES de escribir en el `PrintWriter`
    - Desde la versión 2.2 se permite *buffering* de salida (las cabeceras y líneas de estado se pueden modificar hasta que el buffer se llene)
  - Acepta una de las constantes definidas como código de status
- **void sendError(int sc)**  
**void sendError(int sc, String msg)**
  - Manda el código de error y un mensaje que aparecerá en el navegador del cliente dentro de su HTML
- **void sendRedirect (String location)**
  - Redirección temporal al cliente con la nueva URL de parámetro.
    - Puede ser relativa al raíz de servlets (empieza con "/") o al directorio actual, el contenedor completa la URL
  - Genera tanto el código de estado como la cabecera



Software de  
Comunicaciones

51

## Generación de las cabeceras de respuesta

### Métodos de HttpServletResponse

- **void setHeader(String name, String value)**
  - Establece la cabecera "name" a "value"
- **void setDateHeader (String name, long date)**
  - Valor en milisegundos desde 1970  
(`System.currentTimeMillis`)
  - Establece la cabecera "name" al valor como GMT time string
- **void setIntHeader(String name, int value)**
  - Acepta valores como enteros
  - Pone la cabecera "name" al valor pasado como string
- A partir de la versión 2.2
  - Estas funciones reescriben las cabeceras si más de una vez
  - Para añadir una cabecera más de una vez utilizar
    - **addHeader**
    - **addDateHeader**
    - **addIntHeader**



Software de  
Comunicaciones

52

## Generación de cabeceras de respuesta

### Métodos de HttpServletResponse

- **void setContentType (String type)**
  - Establece la cabecera Content-Type (tipo MIME del contenido). Usado por la mayoría de servlets
- **void setContentLength (int len)**
  - Establece la cabecera Content-Length
- **void addCookie (Cookie cookie)**
  - Inserta una cookie en la cabecera Set-Cookie
- **void sendRedirect(String location)**
  - Ya mencionado



Software de  
Comunicaciones

53

## Ejemplo 6a: Autenticación (1/3)

```
public class LoginServlet extends HttpServlet {  
  
    private void sendLoginForm(HttpServletResponse response,  
                               boolean withErrorMessage)  
        throws ServletException, IOException {  
  
        response.setContentType("text/html");  
        PrintWriter out = response.getWriter();  
        out.println("<html>");  
        out.println("<head><title>Login</title></head>");  
        out.println("<body>");  
  
        if (withErrorMessage)  
            out.println("Login failed. Please try again.<br />");  
  
        out.println("<br />");  
        out.println("<br />Please enter your user name and  
                    password.");  
    }  
}
```



Software de  
Comunicaciones

54

## Ejemplo 6a: Autenticación (2/3)

```
out.println("<br /><form method=\"POST\">");
out.println("<br />User Name: <input type=\"text\"
           name=\"userName\">");
out.println("<br />Password: <input type=\"password\"
           name=\"password\">");
out.println("<br /><input type=\"submit\"
           name=\"Submit\">");
out.println("</form>");
out.println("</body>");
out.println("</html>");
}
public void doGet(HttpServletRequest request,
                  HttpServletResponse response)
    throws ServletException, IOException {
    sendLoginForm(response, false);
}
```



Software de  
Comunicaciones

55

## Ejemplo 6a: Autenticación (3/3)

```
public void doPost(HttpServletRequest request,
                  HttpServletResponse response)
    throws ServletException, IOException {
    String userName = request.getParameter("userName");
    String password = request.getParameter("password");

    if (userName!=null && password!=null &&
        userName.equals("swc") && password.equals("it")) {

        response.sendRedirect("http://domain/WelcomePage");

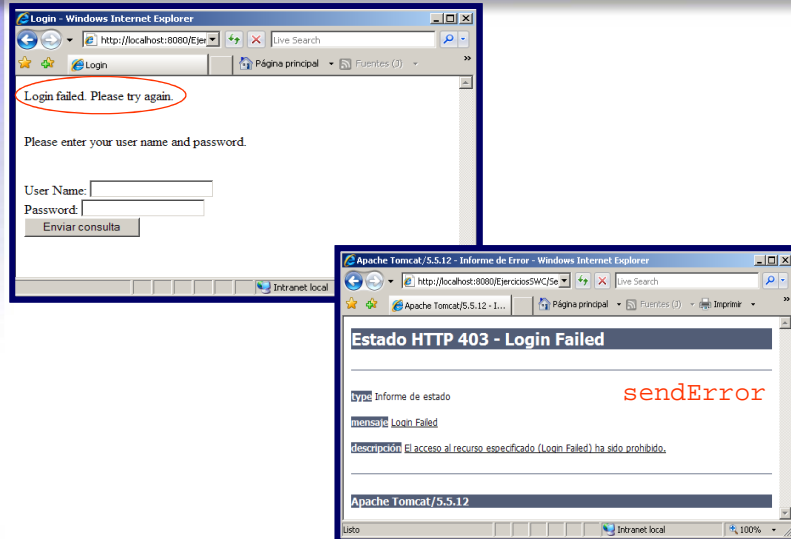
    } else {
        sendLoginForm(response, true);
    }
    sponse.sendError(response.SC_FORBIDDEN, "Login Failed");
}
```



Software de  
Comunicaciones

56

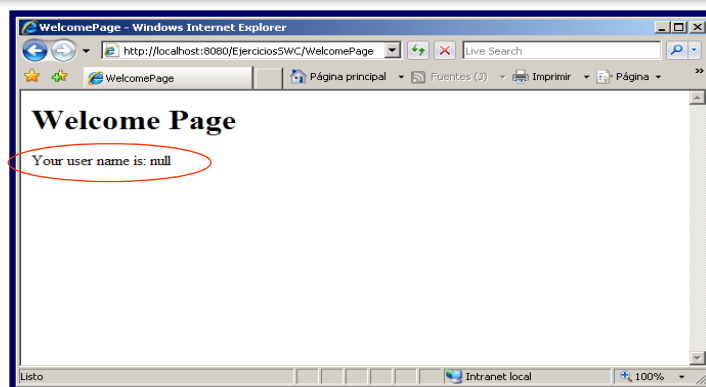
## Ejemplo 6a: Respuesta del Servlet con Fallo



Software de  
Comunicaciones

57

## Ejemplo 6a: Respuesta bajo éxito



```
String userName = request.getParameter("userName");  
...  
out.println("<p>Your user name is: " + userName + "</p>");
```

- El objeto `request` será uno nuevo para esta redirección



Software de  
Comunicaciones

58

## Forwarding / Including de peticiones

### Usar un objeto `RequestDispatcher`

- Llamar al método `getRequestDispatcher` de:
  - `ServletContext`
    - Proporcionar **URL relativa a la raíz** del servidor como argumento
  - `ServletRequest`
    - Proporcionar **URL relativa a la petición HTTP** como argumento
- Llamar al método `getNamedDispatcher` de `ServletContext`
- Para pasar el control al recurso de la URL: **forward**
  - Proporcionar objetos `request` y `response` como argumentos
  - El servlet de origen no puede escribir el cuerpo de la respuesta
  - El servlet de origen puede escribir las cabeceras de la respuesta
  - Cambia el camino para ser relativo al destino y no el origen
- Para incluir la salida generada por el recurso de la URL: **include**
  - Proporcionar objetos `request` y `response` como argumentos
  - El recurso (JSP/HTML/Servlet) objetivo no puede modificar las cabeceras de la respuesta



Software de  
Comunicaciones

59

## Ejemplo 6b: Autenticación (3/3)

```
public void doPost(HttpServletRequest request,
                   HttpServletResponse response)
    throws ServletException, IOException {
    String userName = request.getParameter("userName");
    String password = request.getParameter("password");

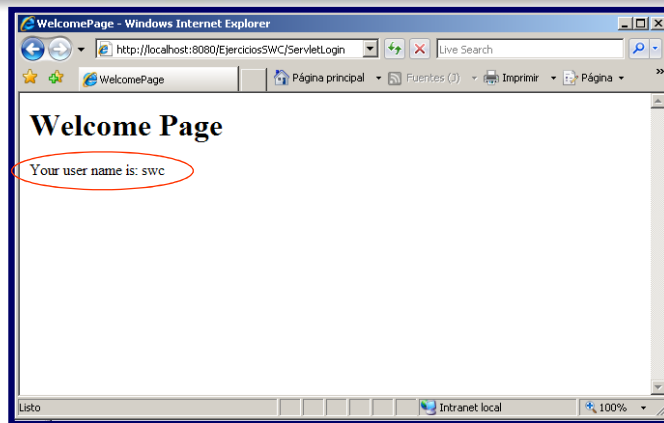
    if (userName!=null && password!=null &&
        userName.equals("swc") && password.equals("it")) {
        RequestDispatcher rd = request.getRequestDispatcher("WelcomePage");
        rd.forward(request, response);
    } else {
        sendLoginForm(response, true);
    }
}
```



Software de  
Comunicaciones

60

## Ejemplo 6b: Respuesta en éxito



Software de  
Comunicaciones

```
String userName = request.getParameter("userName");  
...  
out.println("<p>Your user name is: " + userName + "</p>");
```

- El objeto `request` es el mismo

61

## Cookies

- HTTP es un protocolo sin estado
- Las cookies son piezas pequeñas de información
  - Enviadas del servidor al cliente en respuestas HTTP
  - Retornadas del cliente al servidor in sucesivas peticiones
- Una cookie es por tanto un medio para que el servidor almacene información en el cliente
- Tienen
  - Un nombre e identificador
  - Opcionalmente, atributos como `path`, `comment`, `domain`, `maximum lifespan`, `version number`



Software de  
Comunicaciones

62

## Uso de cookies

- Identificación de un usuario durante una sesión de comercio (“*session tracking*”)
  - Por ejemplo, carro de la compra
- Evitar recordar usuario y contraseña (“login” y “password”), y demás datos del usuario
  - Sólo una alternativa para acceso de baja seguridad
  - El sitio Web puede recordar los datos de usuario
- Configurar el acceso al sitio
  - El sitio puede recordar los intereses del usuario
- Publicidad dirigida
  - Los sitios pueden enfocar la publicidad en función del perfil del usuario



Software de  
Comunicaciones

63

## Problemas con las Cookies

- No es tanto un problema de seguridad
  - Ni se ejecutan o se interpretan
  - El tamaño y su número (por lugar y número total) está limitado (4KB, 20, 300)
- Es un problema de privacidad
  - Los servidores pueden recordar tus acciones previas
  - Las cookies pueden ser compartidas entre servidores
    - Por ejemplo, cargar una imagen con una cookie asociada de un tercer lugar, esas imágenes vienen hasta en correo HTML !
  - Información secreta (tarjeta de crédito) no se deben de guardar en cookies sino en el servidor
    - Las cookies sólo almacenan un identificador; como usuario cómo puedo estar seguro.
- Muchos usuarios las desactivan
  - Los servlets pueden usar cookies pero no son imprescindibles.



Software de  
Comunicaciones

64



## Creando y populando Cookies en Servlets

### Métodos de la clase `Cookie`

- `Cookie(String name, String value)`
  - Crea una cookie con nombre y valor
  - Caracteres prohibidos: [] () = , " / ? @ : ;
- `getXxx` y `setXxx`,
  - siendo `Xxx` el nombre del atributo
  - Atributos:
    - Tipo `String`: `Comment`, `Domain`, `Name`, `Path`, `Value`
    - Tipo `int`: `MaxAge`, `Version`
    - Tipo `Boolean`: `Secure`



Software de  
Comunicaciones

65

## Lectura y escritura de Cookies en Servlets

- Se leen del objeto petición  

```
Cookie[] cookies = request.getCookies();
```
- Se escriben en el objeto respuesta  

```
void HttpServletResponse.addCookie(Cookie cookie)
```
- Para reutilizar una cookie de la petición:
  - Se tiene que usar también `addCookie` (no basta usar `setValue`)
  - Se deben resetear los atributos (los valores no se transmiten en la petición)
- Ver también el método `getCookieValue`



Software de  
Comunicaciones

66

## Session Tracking

- Cliente en una tienda on-line añade algo al carro de compra:
  - ¿Cómo sabe el servidor lo que hay dentro del carro?
- Cliente en una tienda on-line va a la caja:
  - ¿Cómo sabe el servidor cuál de los carros de compra es suyo?
- Implementar *session tracking* con **cookies**
  - Complicado: generar ID de sesión único, asociar ID con información de sesión vía *hash-table*, poner tiempo de expiración de la cookie, ...
- Implementar *session tracking* con **URL-rewriting**
  - Se debe añadir la información de sesión a todas las URLs que refieren al sitio Web propio
  - No se puede usar páginas estáticas que contienen tales URLs
- Implementar *session tracking* con **hidden form fields**
  - Tedioso:
  - Todas las páginas deben de ser resultado de formularios previos



Software de  
Comunicaciones

## Interfaz HttpSession: Session Object

- Crea una sesión entre el cliente y el servidor HTTP, que persiste a través de distintas peticiones
- Permite a los servlets:
  - ver y manipular información de una sesión, como el identificador de sesión, momento de creación, ...
  - enlazar objetos a sesiones, permitiendo que la información de usuario persista a través de varias conexiones
- Para obtener la sesión asociada con una petición
  - `getSession()` y `getSession(boolean create)` de `HttpServletRequest`
  - si no existe sesión asociada a la petición:
    - `getSession() / getSession(true)` crea una nueva
    - `getSession(false)` devuelve null



Software de  
Comunicaciones

68

## Almacenar información en Session Object

- Dentro de una sesión se pueden almacenar objetos arbitrarios
  - Usando mecanismos similares a las tablas hash
  - Se guardan y recuperan con `setAttribute` y `getAttribute`
- Para dar apoyo a
  - Aplicaciones Web distribuidas
  - Sesiones persistentes

los datos de la sesión deben implementar `java.io.Serializable`



Software de  
Comunicaciones

69

## Gestión de objetos HttpSession (1/2)

- Asociar información con una sesión
  - `void setAttribute(String name, Object value)`
  - `void setMaxInactiveInterval(int interval)`
  - `void removeAttribute(String name)`
- Terminar sesiones completadas o abandonadas
  - Automáticamente, después de que pase `MaxIntervalInterval`
  - Mediante el método `void invalidate()`



Software de  
Comunicaciones

70

## Gestión de objetos HttpSession (2/2)

- Buscar información asociada a una sesión
  - Object `getAttribute(String name)`
  - Enumeration `getAttributeNames()`
  - String `getId()`
  - long `getCreationTime()`
  - long `getLastAccessedTime()`
  - ServletContext `getServletContext()`
  - Int `getMaxInactiveInterval()`
  - boolean `isNew()`



Software de  
Comunicaciones

71

## HttpSession con cookies deshabilitadas

- Por detrás, el mecanismo de control de sesión usa:
  - Cookies, o si están deshabilitadas,
  - Reescritura de URLs en el resto de los casos
- Para garantizar que la reescritura funciona: codificar URLs
  - El servidor utiliza cookies: sin efecto
  - El servidor utiliza URL-rewriting: se añade el ID a la URL  
`http://host/path/file.html;jsessionid=1234`
- Para cualquier enlace hipertextual al mismo sitio
  - Utiliza `response.encodeURL`
- Para cualquier uso de `sendRedirect` in el código
  - Usa `response.encodeRedirectURL`



Software de  
Comunicaciones

72

## Ejecutar Servlets

- Configuración del servidor web Tomcat ( en el archivo web.xml):

```
<servlet>
  <servlet-name>PrimerServlet</servlet-name>
  <description>Mi primer Servlet HTTP</description>
  <servlet-class>PrimerServlet</servlet-class>
</servlet>
<servlet-mapping>
  <servlet-name>PrimerServlet</servlet-name>
  <url-pattern>/servlets/miPrimerServlet</url-pattern>
</servlet-mapping>
```

1. Introducir la siguiente URL en un navegador
  - `http://host/app/servlets/miPrimerServlet`
2. Llamarlo desde dentro de una página HTML
  - Enlace, form action o recargando la etiqueta META
  - `<a href="servlets/miPrimerServlet">Mi primer servlet</a>`
3. Desde otro servlet
  - `sendRedirect, RequestDispatcher`



Software de  
Comunicaciones

73

## Nivel de presentación: Java Server Pages (JSPs)



Software de  
Comunicaciones

## Contenido: Java Server Pages

- Introducción
- Variables predefinidas
- Instrucciones JSP
  - Script
  - Directive
  - Action
- JavaBeans
- JSP Standard Tag Library (JSTL)
  - Expression Language (EL)



75

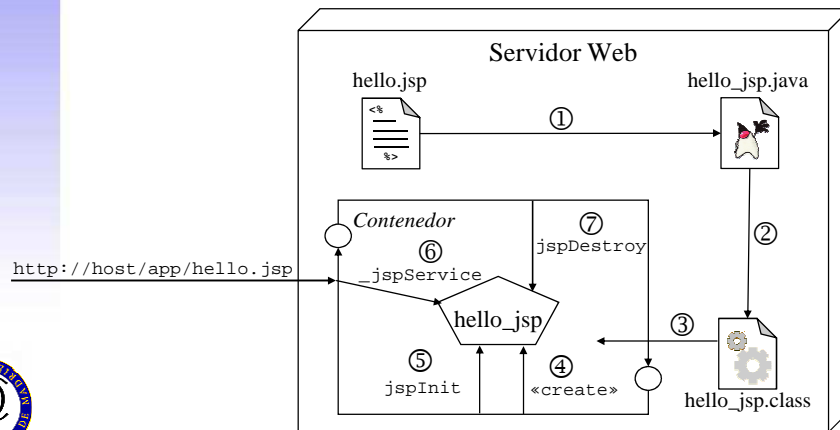
## Introducción

- Los servlets generan siempre toda la página
  - en muchos casos casi toda la página es estática
- Solución: Java Server Pages (JSPs)
  - Permite mezclar tanto de
    - HTML estático
    - Contenido dinámico generado por servlets
- Ventajas JSP:
  - Ampliamente soportado por plataformas y servidores Web
  - Acceso completo a servlets y tecnologías Java (JavaBeans, etc.) en la parte dinámica
- Las JSP son convertidas por el servidor en servlets
  - La primera vez que se usan o se despliegan



76

## Procesamiento JSP



Fuente: Web Component Development With Servlet and JSP Technologies  
Sun Microsystems (course SL-314-EE5)



77

## Variables predefinidas/ Objetos implícitos (1/2)

- **request**
  - El objeto `HttpServletRequest`
- **response**
  - El objeto `HttpServletResponse`
- **session**
  - El objeto `HttpSession` asociado a la petición
- **out**
  - El objeto `PrintWriter` usado para enviar la salida al cliente (es un *buffered* `PrintWriter` llamado `JspWriter`)
- **page**
  - Sinónimo de `this` (no muy usado)



78

## Variables predefinidas/ Objetos Implícitos (2/2)

- **Exception**
  - Páginas de error
- **application**
  - Representa el objeto `ServletContext`
  - Permite almacenar datos persistentes mediante `getAttribute` y `setAttribute`
  - Recordar que los datos almacenados en el `ServletContext` son accesibles desde otro servlet
- **config**
  - El objeto `ServletConfig`
- **pageContext**
  - Objeto de la clase `pageContext` específica a JSP
  - Punto de acceso a los atributos de la página
  - Lugar de almacenamiento de datos compartidos



Software de  
Comunicaciones

79

## Instrucciones JSP

- Tres tipos de instrucciones embebidas
  - *Guiones (scripts)*
    - Especifican código Java que formará parte del servlet
  - *Directivas*
    - Controlan la estructura general del servlet
  - *Acciones*
    - Etiquetas HTML interpretadas en la fase de traducción
    - Controlan la ejecución del motor de JSPs
- Comentarios:
  - `<%-- comentario --%>`



Software de  
Comunicaciones

80



## Elementos Script

- Expresiones: `<%= expression %>`
  - Son evaluadas y el resultado se incluye en la salida
  - Por ejemplo: `<%= new java.util.Date() %>`
- Scriptlets: `<% code %>`
  - Bloques de código de una página JSP que se insertan en el método `_jspService` (llamado por `service`)
  - Por ejemplo: `<% try { . . . } catch() { . . . } %>`
- Declaraciones: `<%! code %>`
  - El código se inserta en la clase servlet, fuera de métodos existentes (código de inicialización)
  - Por ejemplo: `<%! int i = 0; %>`



Software de  
Comunicaciones

81

## Expresiones: `<%= expresion %>`

- Expresiones Java
- Salida convertida a String
- Evaluada en el momento de la llamada
  - Acceso a información sobre la petición
- No se añade “,” al final de la expresión
- Ejemplos
  - `<%=java.util.Calendar.getInstance().getTime() %>`
  - `<p>Your session Id: <%= session.getId() %>`
  - Petición:  
`http://host/confirmation.jsp?title=core+web`  
respuesta:  
`Thanks for ordering <%=request.getParameter("title")%>`



Software de  
Comunicaciones

82

## Ejemplo 1: expresiones

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<!-- Taken from Core Servlets and JavaServer Pages. 2nd Edition -->
<html>
<head>
<title>JSP Expressions</title>
<meta name="keywords"
      content="JSP,expressions,JavaServer Pages,Servlets" />
<meta name="description"
      content="A quick example of JSP expressions" />
<link rel="stylesheet" href="JSP-Styles.css" type="text/css"/>
</head>
<body>
<h1>JSP Expressions</h1>
<ul>
<li>Current time: <%= new java.util.Date() %> </li>
<li>Server: <%= application.getServerInfo() %> </li>
<li>Session ID: <%= session.getId() %></li>
<li>The <code>testParam</code> form parameter:
      <%= request.getParameter("testParam") %></li>
</ul>
</body>
</html>
```



Software de  
Comunicaciones

83

## Ejemplo 1: Respuesta del Servidor

**JSP Expressions**

- Current time: Tue Oct 30 09:48:50 CET 2007
- Server: Apache Tomcat/5.5.12
- Session ID: 624B312D81F324D5ADD8A1EC39E78254
- The testParam form parameter: null



Software de  
Comunicaciones

84

## Scriptlets: `<% code %>`

- Tareas que no pueden realizarse mediante expresiones
  - Generar cabeceras de respuesta
  - Escribir en el log del servidor
  - Actualizar una base de datos
  - Ejecutar código que contenga bucles, etc
- Ejemplos:
  - Poner cabecera de respuesta

```
<% response.setContentType("text/plain"); %>
```

- Código condicional

```
<% if (Math.random() < 0.5) { %>
    <p>Have a <b>nice</b> day!</p>
<% } else { %>
    <p>Have a <b>lousy</b> day!</p>
<% } %>
```

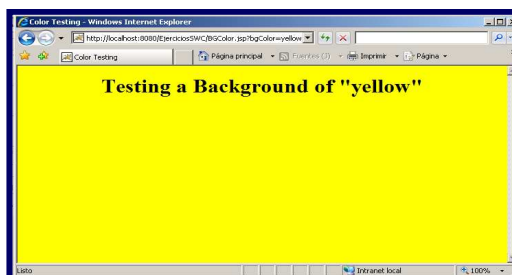


Software de  
Comunicaciones

85

## Ejemplo 2: Scriptlets

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<!-- Taken from Core Servlets and JavaServer Pages 2nd Edition -->
<html>
<head><title>Color Testing</title></head>
<% String bgColor = request.getParameter("bgColor");
    if ((bgColor == null) || (bgColor.trim().equals("")) {
        bgColor = "WHITE";
    }
%>
<body bgcolor="<%= bgColor %>">
<h1 align="center">Testing a Background of "<%= bgColor %>"
</h1>
</body>
</html>
```



Software de  
Comunicaciones

86

## Declaraciones: `<%! declaración %>`

- Definición de métodos o campos
  - Se insertan en el servlet fuera de los métodos existentes
- No produce salida alguna
  - Normalmente se usan en conjunción con expresiones o scriptlets
- Ejemplos
  - ```
<%! String getSystemTime() {  
    return Calendar.getInstance().getTime.toString();  
} %>
```
  - ```
<%! private int accessCount = 0; %>  
<h2>Accesses to page since server reboot:  
    <%= ++accessCount %></h2>
```

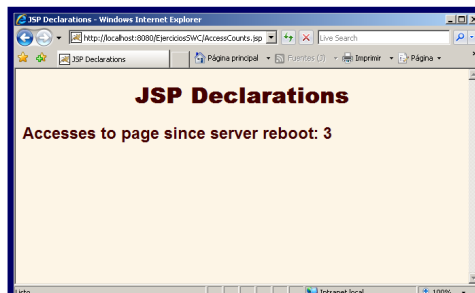


Software de  
Comunicaciones

87

## Ejemplo 3: Declaraciones

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">  
<!-- Taken from Core Servlets and JavaServer Pages 2nd Edition -->  
<html>  
<head><title>JSP Declarations</title>  
    <link rel="stylesheet" href="JSP-Styles.css" type="text/css" />  
</head>  
<body>  
    <h1>JSP Declarations</h1>  
    <%! private int accessCount = 0; %>  
    <h2>Accesses to page since server reboot: <%= ++accessCount %>  
    </h2>  
</body>  
</html>
```



Software de  
Comunicaciones

88

### Ejemplo 3: Algunas observaciones

- Peticiones de múltiples clientes al mismo servlet
  - No dan como resultado la creación de múltiples instancias del servlet
  - Sino la creación de múltiples threads que llaman al método `service` de la misma instancia
    - Aunque cuidado en caso de usar `singleThreadModel`
- Por tanto:
  - Variables de instancia se comparten entre múltiples peticiones
  - No hace falta declarar `accessCount` como `static`



Software de  
Comunicaciones

89

### Directivas: `<%@ atributos de directiva %>`

- Afectan a la estructura global del servlet que se genera de la página JSP
- Sintaxis:

```
<%@ directive attribute="value" %>
<%@ directive attribute1="value1"
...
attributeN="valueN" %>
```
- Tres tipos de directivas:
  - **page**: Controla la estructura del servlet importando clases, adaptando la superclase, configurando el tipo de contenido, etc.
  - **include**: Permite insertar el contenido de otros ficheros (HTML, JSP) en el servlet en el momento de la traducción de JSP a servlet
  - **taglib**: Extiende la funcionalidad de JSP. Define etiquetas de marcado personalizadas ("*custom tags*")



Software de  
Comunicaciones

90

## Atributos de la directiva page

- `import`
- `contentType`
- `isThreadSafe`
- `session`
- `buffer`
- `autoflush`
- `extends`
- `info`
- `errorPage`
- `isErrorPage`
- `language`



Software de  
Comunicaciones

91

## Directiva page: atributo import

- Especifica los paquetes importados por el servlet
- Por defecto, el servlet generado importará
  - `java.lang.*`
  - `javax.servlet.*`
  - `javax.servlet.http.*`
  - `javax.servlet.jsp.*`Y posiblemente otros (depende del servidor)

- Ejemplo:

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



Software de  
Comunicaciones

92

## Directiva page : atributo contentType

- Define la cabecera de respuesta Content-Type

```
<%@ page contentType="MIME-type" %>
<%@ page contentType="MIME-type;
      charset=Character-Set" %>
```
- Ejemplo

```
<%@ page contentType="text/html;
      charset=ISO-8859-1" %>
```



Software de  
Comunicaciones

93

## Directiva include

- Uso:

```
<%@ include file="URL relative" %>
```
- Añade el contenido del archivo especificado antes de comenzar la fase de traducción al servlet
  - Las páginas incluidas pueden contener construcciones JSP
- Reutilización de código
- Problema
  - El servidor puede no detectar cuando un fichero incluido ha cambiado
  - Forzar recompilación: cambiar la fecha de modificación del fichero principal
    - Comando touch en unix
    - Modificando explícitamente un comentario en el fichero principal

```
<%-- navbar.jsp modified 30/10/2007 --%>
<%@ include file="navbar.jsp" %>
```
- Ver también el elemento **jsp:include**



Software de  
Comunicaciones

94

## Directiva taglib

- Permite definir etiquetas JSP personalizadas
- El desarrollador define la interpretación de:
  - La etiqueta
  - Sus atributos
  - Su cuerpo
- Agrupa las etiquetas en librerías de etiquetas
- Elementos en el uso de una librería de etiquetas personalizadas
  - Clase manejadora o ficheros JSP :
    - define el comportamiento de la etiqueta
  - Fichero de descripción (*tag library descriptor file, TLD*):
    - información sobre la librería y cada una de sus etiquetas
  - Fichero JSP
    - Que utiliza la librería de tags



Software de  
Comunicaciones

95

## Acciones `< jsp:acción atributos >`

- Etiquetas embebidas en una página JSP y que se interpretan en tiempo de ejecución
  - `jsp:include`
  - `jsp:forward`
  - `jsp:param`
  
  - `jsp:useBean`
  - `jsp:setProperty`
  - `jsp:getProperty`

} **Java Beans**

  
  - `jsp:plugin`
  - `jsp:params`
  - `jsp:fallback`

} **Etiqueta HTML <object>**



Software de  
Comunicaciones

96



## Acción `jsp:include`

- Añade el contenido del fichero especificado cuando se gestiona la petición del cliente
  - Por lo tanto, después de la traducción del servlet
- Atributos de `include`:
  - `page`: una URL relativa (se permite usar expresiones JSP)
  - `flush`:
    - Valor a cierto: obliga a volcar los valores a la salida estándar
    - JSP 1.1: siempre con el valor `true`
- Ficheros incluidos
  - Normalmente ficheros de texto o HTML
  - No puede contener instrucciones JSP
  - Puede ser el resultado de recursos que usan JSP para generar su salida
    - Por lo tanto, la URL puede apuntar a JSPs o servlets



97

## Acción `jsp:forward`

- Contenido generado por JSP o un servlet indicado
  - Añadido a la respuesta
- El control no vuelve a la página original
  - pasa completamente a la segunda página
- Atributos:
  - `page`: una URL relativa (se permite usar expresiones JSP)
- Interacción con el buffer de salida (directiva `page`, atributo `buffer`):
  - Forwarding conlleva que el buffer de salida se borre
  - Forwarding después de que la salida se halla llevado al browser: excepción
    - Por ejemplo sin buffer y datos enviados a la salida
    - Por ejemplo el tamaño de buffer excedido y definido como `autoflush`
- Ejemplo

```
<jsp:forward page="list.jsp" />
```



98

## Acción `jsp:param`

- Para especificar los parámetros
  - Añadido al objeto request
  - Recuperado con `request.getParameter`
- Atributos:
  - `name`: nombre del parámetro
  - `value`: valor (se permite usar expresiones JSP)
- Ejemplos:

```
<jsp:include page="header.jsp" flush="true">
  <jsp:param name="title" value="Welcome" />
</jsp:include>

<jsp:forward page="list.jsp">
  <jsp:param name="order" value="date" />
</jsp:forward>
```



Software de  
Comunicaciones

99

## Sintaxis XML de JSP

- Scripting
  - Expresiones  
`<jsp:expression> Expresión Java </jsp:expression>`
  - Scriptlets  
`<jsp:scriptlet> scriptlet code </jsp:scriptlet>`
  - Declaraciones  
`<jsp:declaration> declaration code </jsp:declaration>`
- Directivas  
`<jsp:directive:directiveName attribute_list />`
- Template Data  
`<jsp:text> text </jsp:text>`



Software de  
Comunicaciones

100

## Ejemplo 4: Hola Mundo JSP

```
<%@ page language="java"
    contentType="text/html; charset=iso-8859-1" %>
<%@ page import="java.util.Date" %>
<html>
  <head>
    <title>Hola Mundo</title>
  </head>
  <body>
    <%! private int accessCount = 0; %>
    <p>Hola, esto es una página JSP.</p>
    <p>La hora del servidor es <%= new Date() %></p>
    <p>La página ha sido accedida <%= ++accessCount %>
      veces desde el arranque del servidor</p>
  </body>
</html>
```



Software de  
Comunicaciones

101

## Ejemplo 4: JSP transformado en un servlet (Tomcat 5.x)

```
import java.util.Date;

public class hello_jsp extends HttpJspBase {

    private int accessCount = 0;

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

        . . .
        response.setContentType("text/html; charset=iso-8859-1");
        pageContext = _jspxFactory.getPageContext(this, request,
            response, null, true, 8192, true);

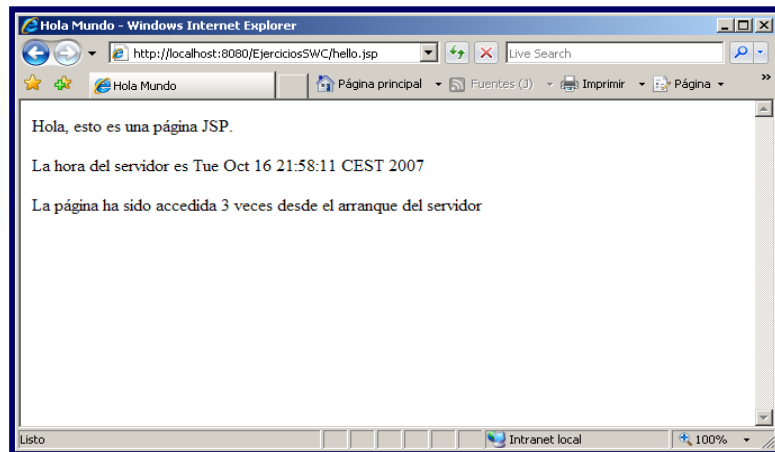
        out = pageContext.getOut();
        out.write("\r\n");
        out.write("<html><head><title>Hola Mundo</title></head>\r\n");
        out.write(" <body>\r\n");
        out.write(" <p>Hola, esto es una página JSP.</p>\r\n");
        out.write(" <p>La hora del servidor es " + new Date() + "</p>\r\n");
        out.write(" <p>La página ha sido accedida " + ++accessCount );
        out.write(" veces desde el arranque del servidor</p>\r\n");
        out.write("</body></html>\r\n");
        . . .
    }
}
```



Software de  
Comunicaciones

102

## Ejemplo 4: Ventana de salida



Software de  
Comunicaciones

103

## Java Beans

- Todos los atributos son privados (*properties*)
- Escritura/lectura de propiedades del bean
  - Método `getXxx` (accesor)
  - Método `setXxx` (mutador)
- Eventos
  - Los beans pueden mandar notificaciones de cambios en propiedades
- Introspección
  - Conocimiento de sí mismos
- Serializables
- Personalizables
  - Facilidades para la edición de propiedades



Software de  
Comunicaciones

104

## ¿Por qué usar Java Beans?

- Reusabilidad y modularidad
  - Clases java separadas
    - más fácil escribir, compilar, probar, depurar y reutilizar.
  - En lugar de grandes cantidades de código embebido en páginas JSP
- Separación más fuerte entre el contenido y la presentación
  - Podrían manipular objetos Java usando simplemente sintaxis compatible XML
- Más fácil compartir objetos entre páginas y servlets
- Pueden simplificar el proceso de lectura de parámetros de las peticiones



105

## Acciones `jsp:useBean`

```
<jsp:useBean id="name" class="package.Class" />
```

- Significado:
  - Instancia un objeto de la clase referenciada por `class`
  - Asigna el nombre indicado en `id`
- Alternativa :
  - En vez del atributo `class` usar el atributo `beanName`
  - Atributo `beanName` puede referirse a un fichero con un Bean serializado
- Acceso a las propiedades:

```
<jsp:getProperty name="book1" property="title" />
```

e equivalente a:

```
<%= book1.getTitle() %>
```
- Asignación de propiedades:

```
<jsp:setProperty name="book" property="title" value="Bible" />
```

es equivalente a:

```
<%= book.setTitle("Bible") %>
```



106

## Acción `jsp:setProperty` (1/2)

- Los valores de atributos deben ser normalmente `Strings`
  - Se permiten usar expresiones JSP en los atributos: `name`, `value`
- Ejemplo
  - Poner el valor de un atributo al de un parámetro de la petición:

```
<jsp:setProperty
  name="entry"
  property="itemId"
  value=<%= request.getParameter("itemId") %>' />
```
- ¿Qué pasa si la propiedad no es de tipo `String`?
  - Conversión explícita (Dentro de `try ... catch`) pero ...  
... ver la siguiente transparencia



Software de  
Comunicaciones

107

## Acción `jsp:setProperty` (2/2)

- Asociar el valor de una propiedad con el de un parámetro de la petición:
  - Parámetro y propiedad con distintos nombres:

```
<jsp:setProperty name="cust" property="email"
                  param="emailAdd" />
```

equivale a:

```
cust.setEmail(request.getParameter("emailAdd"));
```
  - Parámetro y nombre con nombres idénticos

```
<jsp:setProperty name="cust" property="email" />
```

equivale a:

```
cust.setEmail(request.getParameter("email"));
```
- Asociar el valor de cada una de las propiedades con el de un parámetro de la petición con el mismo nombre
  - ```
<jsp:setProperty name="entry" property="*" />
```
- En estos casos, la conversión de tipos es automática



Software de  
Comunicaciones

108

## Ejemplo 5: Mi primer Bean

```
package coreservlets;

public class MessageBean {
    private String message = "No message specified";

    public String getMessage() {
        return message;
    }

    public void setMessage(String message) {
        this.message = message;
    }
}
```



Software de  
Comunicaciones

109

## Ejemplo 5: JSP incluyendo Simple Bean

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<html>
<head><title>Using JavaBeans with JSP</title>
    <link rel="stylesheet" href="JSP-Styles.css" type="text/css" />
</head>
<body>
    <table border="5" align="center">
        <tr><th class="title">Using JavaBeans with JSP</th></tr></table>
        <jsp:useBean id="messageBean" class="coreservlets.MessageBean" />
        <ol>
            <li>Initial value (getProperty):
                <i><jsp:getProperty name="messageBean" property="message" /></i></li>
            <li>Initial value (JSP expression):
                <i><%= messageBean.getMessage() %></i></li>
            <li><jsp:setProperty name="messageBean" property="message" value="Best
                message bean: Fortex" />
                Value after setting property with setProperty:
                <i><jsp:getProperty name="messageBean" property="message" /></i></li>
            <li><%= messageBean.setMessage("My favorite: Kentucky Wonder"); %>
                Value after setting property with scriptlet:
                <i><%= messageBean.getMessage() %></i></li>
        </ol>
    </body>
</html>
```



Software de  
Comunicaciones

110

## Ejemplo 5: Ventana JSP



Software de  
Comunicaciones

111

## Ámbito de los Beans

- En el contexto de los JSPs
  - Los beans se crean con `jsp:useBean`
  - Están asociados a una variable local
  - Existen 4 posibilidades en cuanto a su almacenamiento
    - Atributo `scope`



Software de  
Comunicaciones

112



## Ámbito del Bean

Atributo `scope` de `jsp:useBean` toma uno de los siguientes valores:

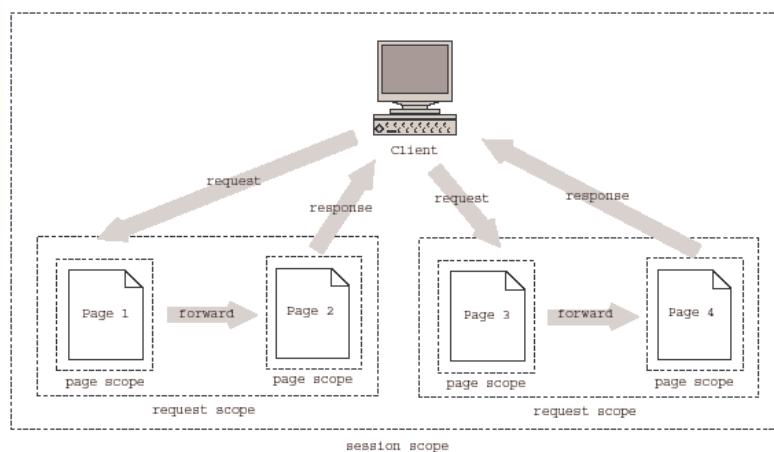
- **page** (valor por defecto)
  - Se almacena en el `pageContext`
  - accesible mediante la variable predefinida `pageContext`
- **request**
  - Se almacena en el `ServletRequest`
  - accesible mediante la variable predefinida `request`
- **session**
  - Se almacena en el `HttpSession`
  - accesible mediante la variable predefinida `session`
- **application**
  - Se almacena en el `ServletContext`
  - accesible mediante la variable predefinida `application`



Software de  
Comunicaciones

113

## Ámbito del Bean



Software de  
Comunicaciones

114

## Creación Condicional de Beans

- La acción `jsp:useBean`
  - Instancia un nuevo bean si no se encontró ningún bean con los mismos `id` y `scope`
  - En otro caso, se asocia el bean existente a la variable referenciada por `id`
- Si en vez de  

```
<jsp:useBean ... />
```

se escribe  

```
<jsp:useBean ... >  
  sentencias  
</jsp:useBean>
```

, `sentencias` se ejecuta sólo cuando un nuevo bean se crea
  - Conveniente para inicializar propiedades de beans compartidos
  - Se comparte el mismo código de inicialización



Software de  
Comunicaciones

115

## Expression Language (EL)

- Introducido con JSTL (JSP Standard Tag Library )
  - Más tarde extendido para usarse en cualquier lado (fuera de tags JSTL): JSP 2.0 EL
- Ayudas para producir scriptlets en páginas JSP
- Sintaxis: `${expression}`
  - Una expresión EL puede ser escapada y no evaluada con `\`
- Pueden ser usadas:
  - Como valores de atributos en acciones  

```
<jsp:include page="${location}">
```
  - Dentro del texto de una plantilla, como HTML  

```
<h1>Welcome ${variable}</h1>
```
- Ejemplo:
  - Fijar el atributo (en este caso en un servlet)  

```
request.setAttribute("endMessage","That's all Folks!");
```
  - Fijar el atributo en JSP (cuatro scopes por atributo):  

```
<h2>${endMessage}</h2>
```



Software de  
Comunicaciones

116

## JSP Standard Tag Library (JSTL)

- JSTL especifica un conjunto de librerías de etiquetas
  - Encapsula funcionalidad JSP común a muchas aplicaciones
- Sintaxis

```
<prefix:tagName (attributeName="attributeValue")* />
<prefix:tagName>body</prefix:tagName>
```
- Áreas funcionales

| Área                                            | URI                                                                                         | Prefijo |
|-------------------------------------------------|---------------------------------------------------------------------------------------------|---------|
| Acciones core                                   | <a href="http://java.sun.com/jsp/jstl/core">http://java.sun.com/jsp/jstl/core</a>           | c       |
| Acciones de procesamiento XML                   | <a href="http://java.sun.com/jsp/jstl/xml">http://java.sun.com/jsp/jstl/xml</a>             | x       |
| Acciones de formato                             | <a href="http://java.sun.com/jsp/jstl/fmt">http://java.sun.com/jsp/jstl/fmt</a>             | fmt     |
| Acciones de acceso a base de datos relacionales | <a href="http://java.sun.com/jsp/jstl/sql">http://java.sun.com/jsp/jstl/sql</a>             | sql     |
| Acciones de funciones                           | <a href="http://java.sun.com/jsp/jstl/functions">http://java.sun.com/jsp/jstl/functions</a> | fn      |



Software de Comunicaciones

117

## Algunos tags de JSTL (1/2)

- **set**
  - Crea una variable EL
  - Actualiza el valor de una variable ya existente o el valor de una propiedad de una JavaBean

```
<c:set var="varName" value="value"
[scope="{page|request|session|application}"]
[target="variable.bean"][property="bean.property"] />
```
- **remove**
  - Borra una variable EL

```
<c:remove var="varName"
[scope="{page|request|session|application}"] />
```
- **url**
  - Proporciona una URL (relativa) con re-escritura, si las cookies están deshabilitadas (para gestión de sesión)

```
<c:url value="value" [var="varName"]
[scope="{page|request|session|application}"] />
```



Software de Comunicaciones

118

## Algunas de las etiquetas JSTL (2/2)

- **if** (ver también **choose** / **when**/ **otherwise**)  

```
<c:if test="expression" var="varName"
  [scope="{page|request|session|application}"]
  body if expression is true
</c:if>
```
- **forEach**
  - Mecanismo de iteración sobre el cuerpo de la etiqueta  

```
<c:forEach items="collection" [var="varName"] [...] >
  body content
</c:forEach>
```
- **out**
  - Evalúa una expresión y escribe el resultado al `JSPWriter`  

```
<c:out value="value" [default="defaultValue"]
  [escapeXml="{true|false}"] />
```



Software de  
Comunicaciones

119

## Ejemplo 6: JSTL y EL

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core"
  prefix="c" %>
<c:if test="${not empty errorMsgs}">
<p>Please correct the following errors:
<ul>
  <c:forEach var="message" items="${errorMsgs}">
    <li>${message}</li>
  </c:forEach>
</ul>
</p>
<p>Accessing messageBean properties:
  ${messageBean.message}</p>
</c:if>
```



Software de  
Comunicaciones

previamente almacenado en uno de los objetos:  
pageContext, request, session, o application

120

## Nivel de presentación: Integración de servlets y JSPs



Software de  
Comunicaciones

## Contenido: Integración de Servlets y JSPs

- Integración de servlets y JSPs
- Gestión de una petición
- *Forwarding* en servlets y JSPs
- El patrón de diseño MVC



Software de  
Comunicaciones

122

## Integración de servlets y JSPs

- Ventaja de JSP frente a servlets
  - Más sencillo generar la parte estática de HTML
- Problema:
  - Un documento JSP proporciona una única presentación
- Solución:
  - Combinar servlets y JSP
  - Servlet
    - Maneja la petición inicial
    - Procesa parcialmente los datos
    - Configura los beans
    - Pasa los resultados a diferentes páginas JSP dependiendo de las circunstancias



123

## Gestión de una petición

- Solución sólo servlet adecuada cuando
  - La salida es un tipo binario y no hay salida
  - Formato/layout de la página altamente variable
- Solución sólo JSP adecuada cuando
  - La salida básicamente caracteres
  - Formato/layout casi fijo
- Combinación servlets & JSPs adecuada cuando
  - Una única petición tiene varias respuestas posibles, con distintos layouts
  - Lógica de negocio y presentación web desarrolladas por distintos equipos
  - Aplicación con procesamiento de datos complicado pero layout relativamente fijo



124

## Forwarding en Servlets y JSPs

- Desde un JSP

```
<jsp:forward page="login.jsp">
<jsp:param name="login" value="pepe" />
</jsp:forward>
```

LoginServlet

- Desde un servlet

```
RequestDispatcher rd =
    getServletContext().getRequestDispatcher(
        "login.jsp?login=pepe");
rd.forward(request, response);
```

LoginServlet?login=pepe



Software de  
Comunicaciones

125

## Ejemplo 1: Request Forwarding

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

    String operation = request.getParameter("operation");
    if (operation == null) {
        operation = "unknown";
    }
    String address;
    if (operation.equals("order")) {
        address = "/WEB-INF/Order.jsp";
    } else if (operation.equals("cancel")) {
        address = "/WEB-INF/Cancel.jsp";
    } else
        address = "/WEB-INF/UnknownOperation.jsp";
    }
    RequestDispatcher rd =
        request.getRequestDispatcher(address);
    rd.forward(request, response);
}
```



Software de  
Comunicaciones

126

## Patrón Model-View-Controller (MVC)

- Patrón de diseño
  - Solución repetible a un problema software común
- Patrón MVC adecuado cuando
  - Una única petición tiene varias respuestas posibles
  - Distintas páginas tienen procesado común
- Modelo
  - Datos que manipular y mostrar
- Vista
  - Lo que se ve en pantalla
- Controlador
  - Gestiona la petición
  - Decide qué lógica invocar
  - Decide qué vista mostrar

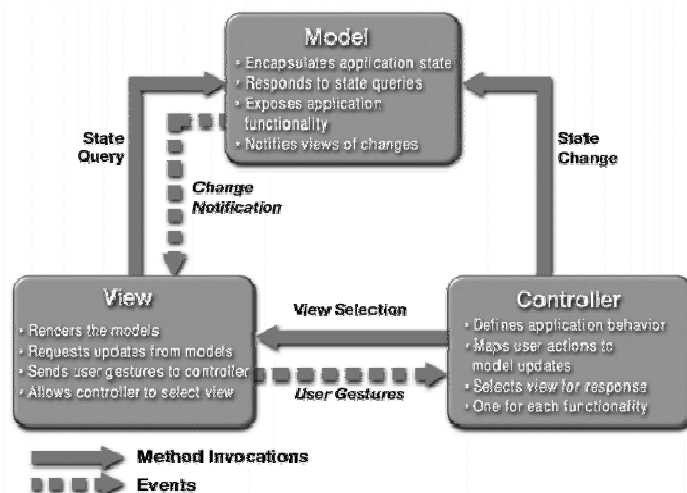


Software de  
Comunicaciones

127

## Patrón MVC: más detalle

Fuente: Java BluePrints, Model-View-Controller. Disponible en:  
<http://java.sun.com/blueprints/patterns/MVC-detailed.html>

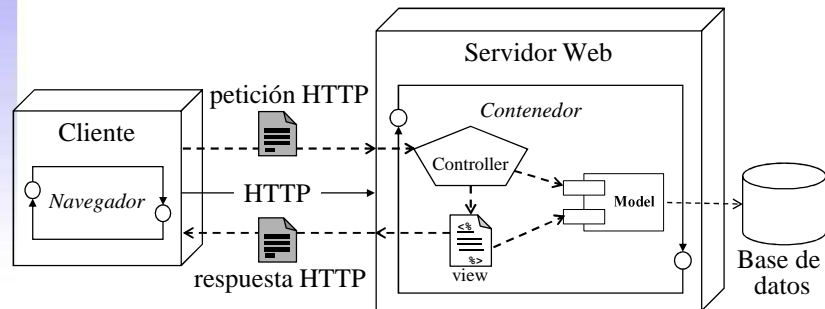


Software de  
Comunicaciones

128



## Arquitectura patrón MVC



Fuente: Web Component Development With Servlet and JSP Technologies  
Sun Microsystems (course SL-314-EE5)



También denominado "Model 2"

129

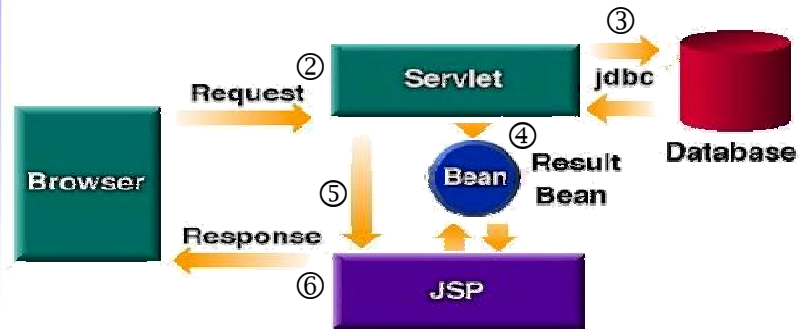
## Implementación de MVC con RequestDispatcher

1. Definir beans para representar los datos
2. Servlet maneja petición
  - Lee parámetros y comprueba los datos
3. Servlet rellena los beans
  - Invoca lógica de negocio, código de acceso a datos
  - Coloca los resultados en los beans (sólo servlet crea y modifica los beans)
4. Servlet almacena beans
  - Invoca `setAttribute` en petición, sesión o contexto
5. Servlet redirige petición al JSP adecuado
  - Usa `forward` del `RequestDispatcher`
6. JSP extrae los datos de los beans
  - Usa `jsp:useBean` y `jsp:getProperty`
  - O usa JSP 3.0 EL: más potente, conciso y legible



130

## Implementación de MVC

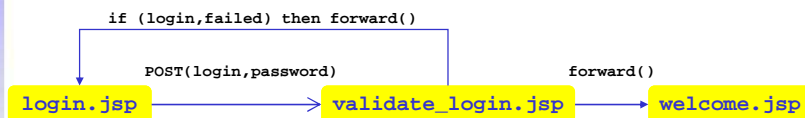


Fuente: IBM Labs. JSP – Dynamic Content Generation Made Simple

131

## Ejemplo: Aplicación Web con JSP

- Usando sólo JSP

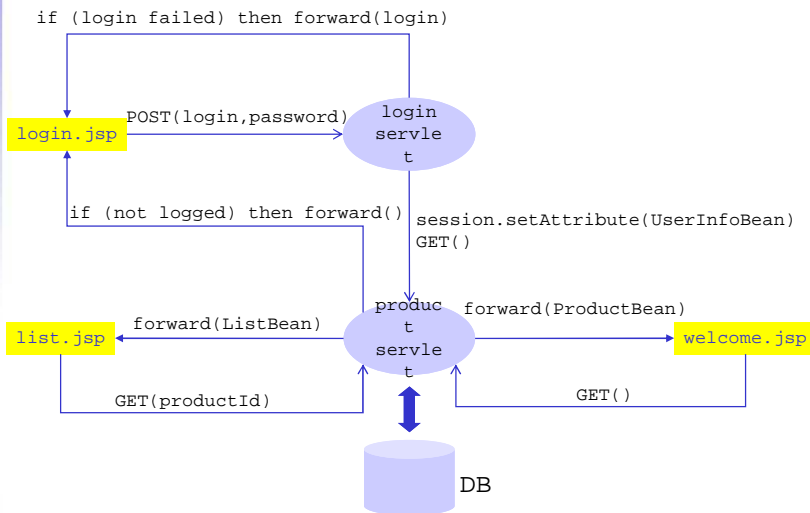


- Para aplicaciones Web más grandes, debería utilizarse el patrón MVC
  - JSP para la presentación
  - Servlets puros para la lógica



132

## Ejemplo: Aplicación Web usando MVC



Software de Comunicaciones

133

## Ejemplo paso de información mediante beans

- **JSP:**

```

<!-- instantiate/use bean -->
<jsp:useBean id='user' class='UserInfoBean' scope='session'>
  <jsp:setProperty name='user' property='lastName' value='Perez' />
</jsp:useBean>
<!-- set property -->
<jsp:setProperty name='user' property='firstName' value='Pepe' />
<!-- get property -->
<jsp:getProperty name='user' property='firstName' />

```
- **Servlet:**

```

// instantiate/use bean
UserInfoBean user = (UserInfoBean) session.getAttribute("user");
if (user == null) {
  user = new UserInfoBean();
  user.setLastName("Perez");
  session.setAttribute("user", user);
}
// set property
user.setFirstName = "Pepe";
// get property
out.println(user.getFirstName());

```



Software de Comunicaciones

134

## MVC: Manejo de Beans en JSPs

- MVC: el bean se inicializa en el servlet y lo usa en el JSP

- Usando el bean en JSP sin el EL:

```
<!-- use bean -->
<jsp:useBean id='user' class='UserInfoBean' scope='session' />
<!-- get property -->
<jsp:getProperty name='user' property='firstName' />
```

- Usando el bean en JSP con JSTL EL:

```
<!-- use bean and get property -->
<c:out value="${user.firstName}" />
```

- Usando el bean en JSP con JSP 2.0 EL:

```
<!-- use bean and get property -->
${user.firstName}
```



Software de  
Comunicaciones

135

## MVC: Paso de info. en la redirección

- La página destino puede obtener la información sin procesar del objeto `request` pero:

- Más fácil de programar en los servlets origen que en las JSP destino
- Múltiples JSPs pueden requerir los mismos datos

- Valor sencillo: pasar en atributo del objeto `request`

- En el origen:

```
request.setAttribute("key1", value1);
```

- En el destino:

```
Type1 value1 = (Type1) request.getAttribute("key1");
```

o como parámetro de la acción `forward`

- Valor complejo: representar como bean compartido

- `scope = application`: se guarda en el `servletContext` object
- `scope = session`: se guarda en el `session` object
- `scope = request` almacenado en `request` object



Software de  
Comunicaciones

136

## Otros temas de interés

- Listeners
  - Monitorizar y reaccionar a eventos del ciclo de vida del servlet
- API de filtrado: **Filter**, **FilterChain**, **FilterConfig**
  - Filtro: un objeto que puede transformar la cabecera o el contenido (o ambos) de una petición o de una respuesta (desde servlet 2.3)
  - Autenticación, logs, conversión de imágenes, compresión, cifrado, transformaciones XML, ...
- Java Server Faces (JSF)
  - Armazón (framework) Java basado en JSPs que simplifica el desarrollo de interfaces de usuario para aplicaciones Java EE
- Jakarta Struts
  - Armazón Java de software libre que extiende el API de servlets para facilitar el desarrollo de aplicaciones Web conformes al patrón MVC



137

## Bibliografía on-line

- Sun's servlet pages / JSP pages / JSTL pages
  - <http://java.sun.com/products/servlet/>
  - <http://java.sun.com/products/jsp/>
  - <http://java.sun.com/products/jsp/jstl/>
- Marty Hall's book *Core Servlets and JavaServer Pages*, vol. 1, 2nd edition
  - <http://pdf.coreservlets.com/>
- Marty Hall's book *More Servlets and JavaServer Pages*, 1st edition (including information on filters and lifecycle listeners)
  - <http://pdf.moreservlets.com/>
- Marty Hall's slides *Beginning and Intermediate-Level Servlet, JSP and JDBC Tutorials*
  - <http://courses.coreservlets.com/Course-Materials/csajsp2.html>
- Apache Tomcat home page / Marty Hall's Apache Tomcat tutorial
  - <http://tomcat.apache.org/>
  - <http://www.coreservlets.com/Apache-Tomcat-Tutorial/>
- Sun's *Java Server Faces Technology* pages
  - <http://java.sun.com/javaee/javaserverfaces/>
- Jakarta Struts
  - <http://struts.apache.org/>



138