

La capa de presentación de la arquitectura Java EE

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: Marty Hall



Contenido

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

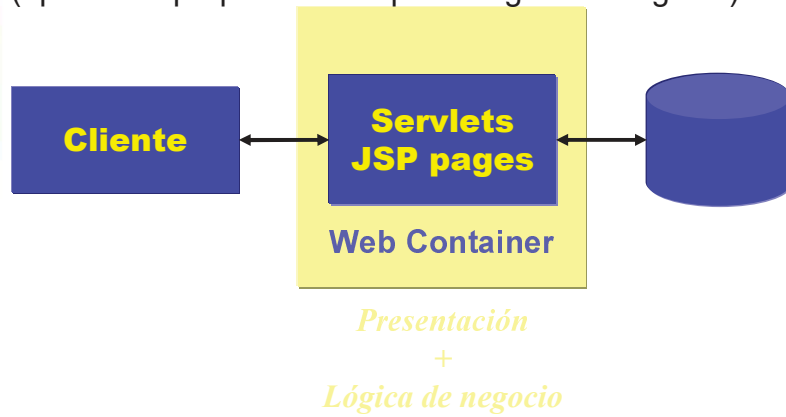
Bibliografía:

- **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



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*)



5

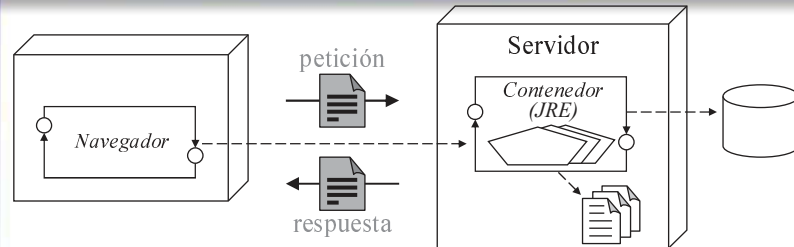
Introducción a los Servlets (1/2)

- 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



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`

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 *pools* para acceder a la base de datos, etc



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



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`



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



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



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



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`



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



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)



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>
```



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));
    }
}
```



Interface ServletConfig (3/3)

- Si el método `init` (con parámetros) de la interfaz `Servlet` (implementado el 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.



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`



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();
```



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()`, ...



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`

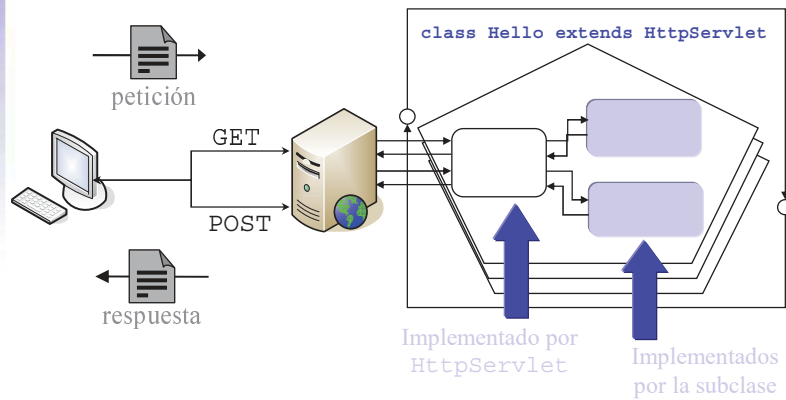


Métodos `doGet`, `doPost`, `doXxx`

- 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



HttpServlet



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.



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)



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();
    }
}
```



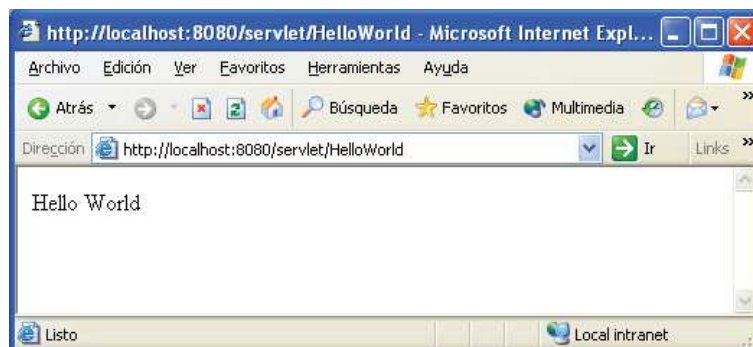
Ejemplo 1: Generación de texto (1/2)

```
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");
    }
}
```



Ejemplo 1: Generación de texto (2/2)



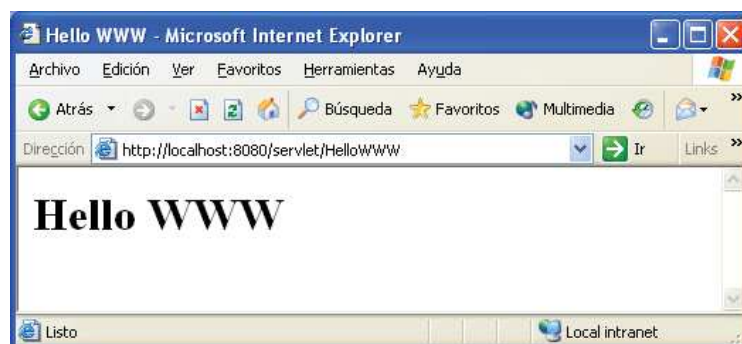
Ejemplo 2: Generación de HTML (1/2)

```
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>");
    }
}
```



Ejemplo 2: Generación de HTML (2/2)



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`)



33

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

`http://host/path?`

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)



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>");
    }
}
```

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\">";

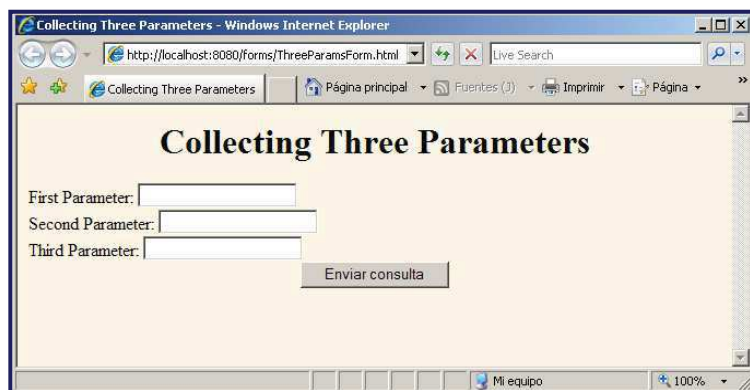
    public static String headWithTitle (String title)
        return(DOCTYPE + "\n" + "<html>\n" + "<head><title>" + title +
            "</title></head>\n");
    }
}
```

Ejemplo 3: Formulario HTML

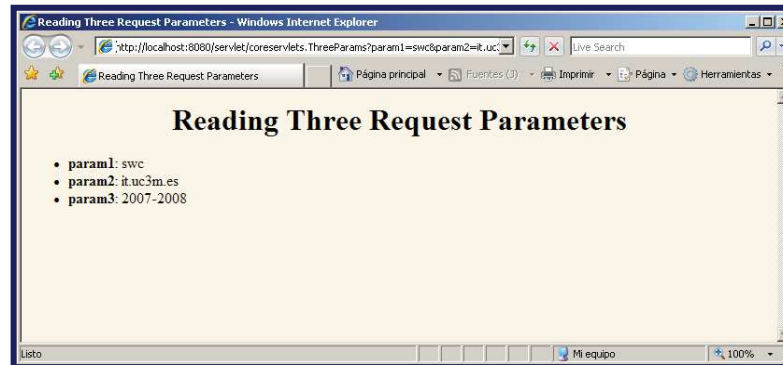
```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/1999/REC-html401-19991224/loose.dtd">
<html>
<head><title>Collecting Three Parameters</title></head>
<body bgcolor="#FDF5E6">
  <h1 align="center">Collecting Three Parameters</h1>
  <form action="/servlet/coreservlets.ThreeParams">
    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>
```



Ejemplo 3: Apariencia del formulario HTML



Ejemplo 3: Respuesta del Servlet



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();
```



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>");
}
```

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



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

    doGet(req, res);
}

}
```

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>
```



Ejemplo 4: Apariencia del formulario HTML

A sample FORM using POST

Item Number:

Quantity:

Price Each:

First name:

Last name:

Credit Card:

☐ Visa

☒ Master Card

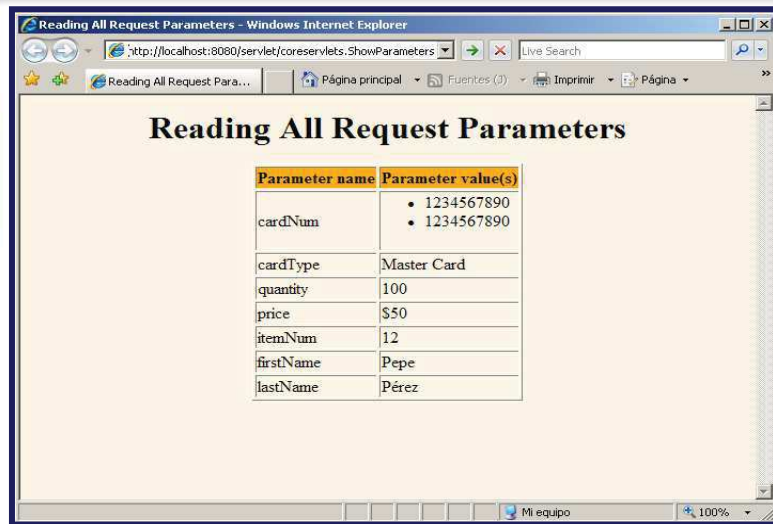
☐ American Express

Credit Card Number:

Repeat Credit Card Number:



Ejemplo 4: Respuesta del Servlet



The screenshot shows a web browser window titled 'Reading All Request Parameters - Windows Internet Explorer'. The address bar shows 'http://localhost:8080/servlet/coreervlets.ShowParameters'. The page content is titled 'Reading All Request Parameters' and displays a table with the following data:

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



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`



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)



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`



48

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

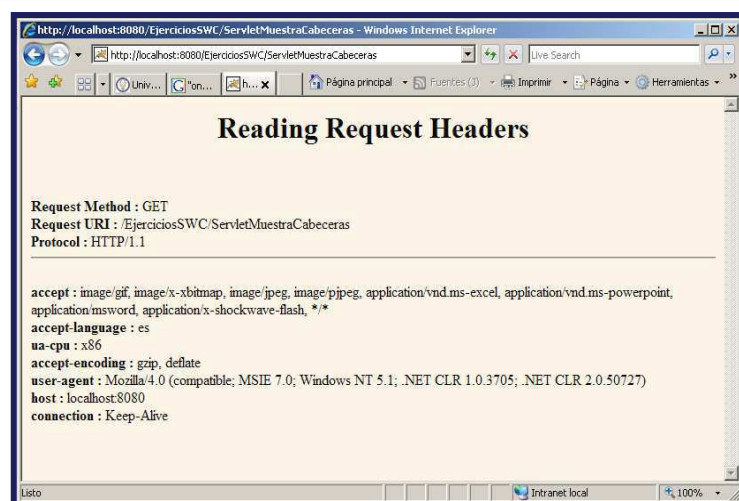


```
public class ShowHeadersServlet extends HttpServlet {

    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/>");
        }
    }
}
```

Ejemplo 5: Respuesta del Servlet



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

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**

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



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.");
    }
}
```



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);
    }
```

Ejemplo 6a: Autenticación (3/3)



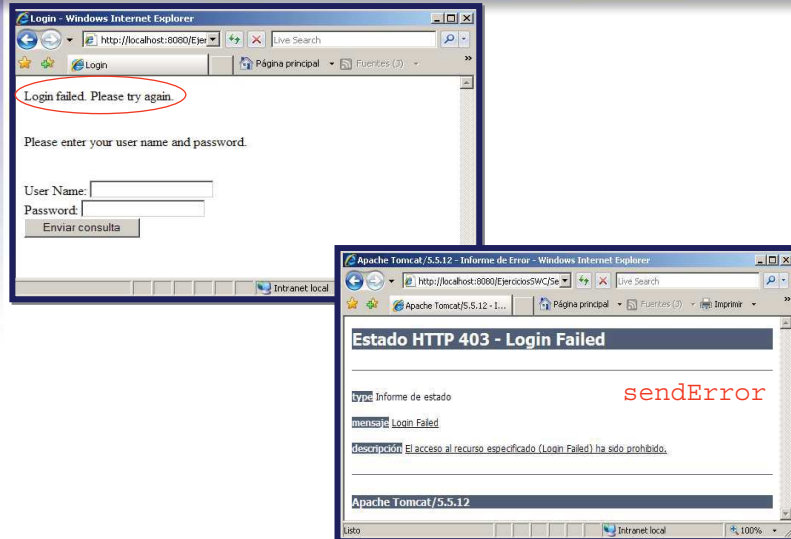
```
    public void doPost(HttpServletRequest request,
                      HttpServletResponse response)
                      throws ServletException, IOException {
        String userName = request.getParameter("userName");
        String password = request.getParameter("password");

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

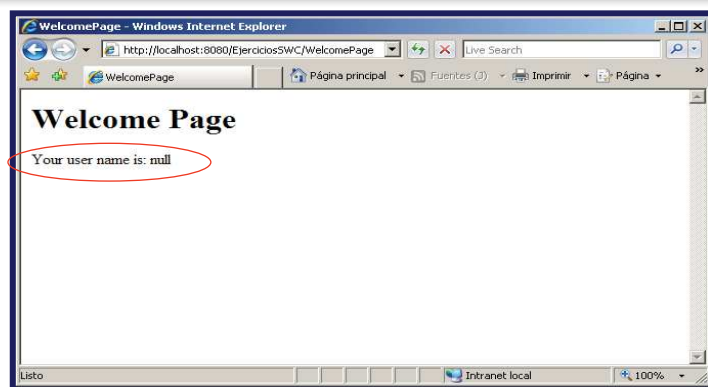
            response.sendRedirect("http://domain/WelcomePage");

        } else {
            sendLoginForm(response, true);
        }
    }
```

Ejemplo 6a: Respuesta del Servlet con Fallo



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



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



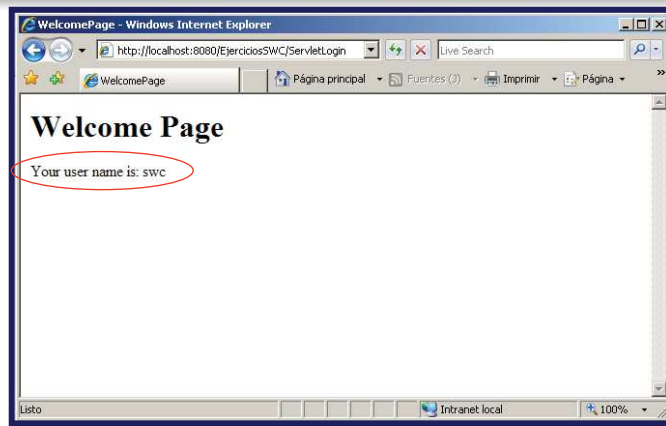
Ejemplo 6b: Autenticación (3/3)

```
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")) {
        
    } else {
        sendLoginForm(response, true);
    }
}
```



Ejemplo 6b: Respuesta en éxito



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

- El objeto `request` es el mismo



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



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

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.

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`



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`



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

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

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`



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()`



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()`



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`



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
 - `Mi primer servlet`
3. Desde otro servlet
 - `sendRedirect, RequestDispatcher`



Nivel de presentación: Java Server Pages (JSPs)

