TABLA DE CONTENIDO

[TABLA DE CONTENIDO 1](#_Toc503688144)

[Crear Servlet 2](#_Toc503688145)

[Objetivo: 2](#_Toc503688146)

[Crear Servlet. 2](#_Toc503688147)

[Notas: 3](#_Toc503688148)

[Probar el servlet método GET 5](#_Toc503688149)

[Prueba intern de envio de mensaje desde java a la consola del navegador 5](#_Toc503688150)

[Probar el servlet método POST 6](#_Toc503688151)

[Teclear el país: Venezuela e introducir click en botón enviar. 6](#_Toc503688152)

[Notas 7](#_Toc503688153)

[Java EE Servlet y JSP 7](#_Toc503688154)

[**Introducción a los Servlets** 7](#_Toc503688155)

[Estructura de un servlet 8](#_Toc503688156)

[Clases principales de servlets 9](#_Toc503688157)

[Métodos de información del servidor con servlets 10](#_Toc503688158)

[Ciclo de vida de un servlet 11](#_Toc503688159)

[Métodos implícitos en el ciclo de vida de un servlet 12](#_Toc503688160)

[**JSP (JavaServer Pages)** 14](#_Toc503688161)

[Directivas JSP: page 14](#_Toc503688162)

[Otras directivas JSP 14](#_Toc503688163)

[Scriptlets 14](#_Toc503688164)

[Scriptlets (aviso) 15](#_Toc503688165)

[Acciones: jsp:include 15](#_Toc503688166)

[Acciones: jsp:forward 15](#_Toc503688167)

[**Objetos predefinidos** 16](#_Toc503688168)

Crear Servlet

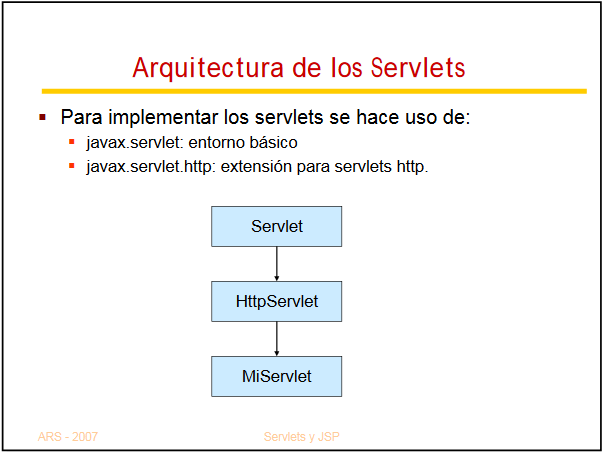
### Objetivo:

Crear un servlet, como caso de estudio.

Nota: Se esta utilizando el ambiente previo de “Liferay”, para utilizar el modelo OSGI.

### Crear Servlet.

1. Crear paquete: com.pais.servlet
2. Crear clase MiServlet que extienda de HTTPServlet.



**package com.pais.servlet;**

**import java.io.IOException;**

**import java.io.PrintWriter;**

**import javax.servlet.Servlet;**

**import javax.servlet.ServletException;**

**import javax.servlet.http.HttpServlet;**

**import javax.servlet.http.HttpServletRequest;**

**import javax.servlet.http.HttpServletResponse;**

**import org.osgi.service.component.annotations.Component;**

**@Component(**

**immediate = true,**

**property = {**

**"osgi.http.whiteboard.context.path=/",**

**"osgi.http.whiteboard.servlet.name=com.pais.servlet.MiServlet",**

**"osgi.http.whiteboard.servlet.pattern=/miservlet",**

**},**

**service = Servlet.class)**

**public class MiServlet extends HttpServlet {**

**private static final long serialVersionUID = -1970334471631842726L;**

**// Invocar http://localhost:8080/o/miservlet -> [Peticion GET]**

**@Override**

**protected void doGet(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException {**

**System.out.println("Peticion doGet Servlet...!");**

**resp.setContentType("text/html");**

**PrintWriter out = resp.getWriter();**

**out.println("<html>");**

**out.println("<head>");**

**out.println("<title>Response Servlet GET</title>");**

**out.println("</head>");**

**out.println("<body>");**

**out.println("<h1>Respuesta .. peticion doGet Servlet...!!</h1>");**

**out.println("<br>");**

**out.println("<h2>Realizar una peticion POST</h1>");**

**out.println("<hr>");**

**out.println("<form action='miservlet' method='post'>");**

**out.println("<lb>");**

**out.println("<strong>Pais: </strong> <input name='pais' />");**

**out.println("<input type='submit' />");**

**out.println("</form>");**

**out.println("</body>");**

**out.println("</html>");**

**}**

**// Invocar http://localhost:8080/o/miservlet -> [Peticion POST]**

**@Override**

**protected void doPost(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException {**

**System.out.println("Peticion doPost Servlet...!");**

**resp.setContentType("text/html");**

**String pais = req.getParameter("pais");**

**PrintWriter out = resp.getWriter();**

**out.println("<html>");**

**out.println("<head>");**

**out.println("<title>Response Servlet POST</title>");**

**out.println("</head>");**

**out.println("<body>");**

**out.println("<h1>Respuesta .. peticion doPOST Servlet...!!</h1>");**

**out.println("<hr>");**

**out.println("<lb>");**

**out.println("<strong>Pais:</strong>" + pais);**

**out.println("</body>");**

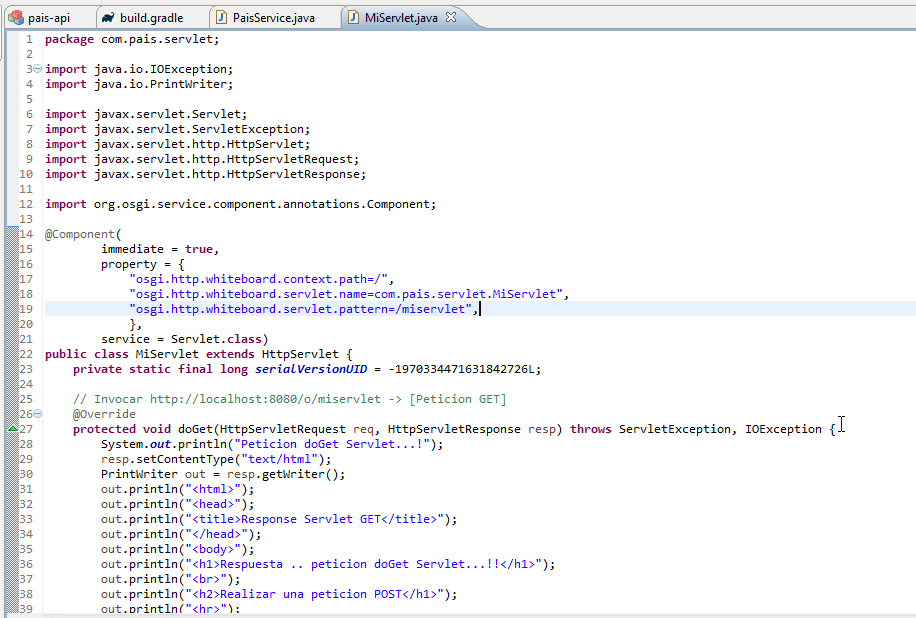
**out.println("</html>");**

**}**

**}**

### Notas:

* Como modulo OSGI con las propiedades:
  + **osgi.http.whiteboard.context.path=/**
  + **osgi.http.whiteboard.servlet.name=com.pais.servlet.MiServlet**
  + **osgi.http.whiteboard.servlet.pattern=/miservlet => define el patrón para definir el servlet**
* Se genera los metodos GET y POST
  + GET: responde un Html (‘JSP’) con formulario para realizar una petición submit
  + POST: responde la petición post.
  + Las respuestas de las peticiones se estan devolviendo a través del PrintWriter del response (a manera de ejercicio. Analisis de la petición y respuesta).



### Probar el servlet método GET

Con el servidor levantado. En el navegador invocar: <http://localhost:8080/o/miservlet>

Repuesta a petición GET:



### Prueba intern de envio de mensaje desde java a la consola del navegador

// Enviar un Alert de JavaScript como pruba

out.println("<script type=\"text/javascript\">");

out.println("console.log('Mensaje desde Java a la consola del navegador...!');");

out.println("</script>");

### Probar el servlet método POST

### Teclear el país: Venezuela e introducir click en botón enviar.

Repuesta a petición POST:

### 

Notas

### Java EE Servlet y JSP

**Introducción a los Servlets**

Los **Servlets** son clases de Java que implementan la clase **HttpServlet** de Java. Dichos servlets no contienen parte gráfica, por lo que su funcionamiento en el momento de recibir una petición no tiene por qué ser dar salida a código HTML en el cliente.

Un servlet se puede definir como un conjunto de acciones capturadas desde un cliente y que pueden derivar en una respuesta http o pueden realizar un flujo de trabajo.

El cliente, mediante el navegador, hace una solicitud al contenedor web, que captura dicha solicitud en un servlet mediante un request.

El servlet puede enviar una respuesta mediante un objeto response, y dicha respuesta puede ser implementada mediante un objeto **PrintWriter** para devolver código **HTML** que se generará de nuevo en el cliente.

La llamada a un servlet se debe realizar a través de contenidos estáticos HTML, tales como páginas HTML o JSP.

Las páginas **JSP** contienen lógica de servidor y lenguaje **HTML** en un mismo conjunto.

Teniendo en cuenta esto, las páginas JSP contienen parte gráfica y lógica de servidor.

En realidad, las páginas JSP utilizan un servlet interno al que se accede mediante etiquetas de servidor y que también implementan el diseño HTML. Las páginas JSP son la evolución de los servlets y permiten realizar acciones más dinámicas que un servlet, ya que podemos utilizar la lógica del servidor separada del diseño, pero en un mismo conjunto.

## Estructura de un servlet

Los servlets heredan de la clase **HttpServlet** y sobrescriben el método **doPost** o **doGet**, dependiendo de si los datos han sido enviados por el método POST o GET.   
Podemos realizar la misma acción tanto para peticiones GET y POST simplemente llamando a **doGet** desde **doPost** o viceversa.

Ambos métodos, doPost y doGet, ofrecen dos argumentos:

* Un objeto **HttpServletRequest**: Encapsula la petición HTTP
* Un objeto **HttpServletResponse**: Encapsula la respuesta HTTP.

El objeto **HttpServletRequest** nos permite acceder tanto a los datos como a las cabeceras de la petición **HTTP**.

El objeto **HttpServletResponse** nos permite modificar la información de salida, como los códigos de estado **HTTP** o las cabeceras de respuesta.

Además podemos obtener un objeto **PrintWriter** sobre el que generar el contenido dinámico del documento al cliente.

Los métodos doGet y doPost pueden lanzar las excepciones **ServletException** e **IOexception** por lo que deben controlar dichas excepciones, ya sea mediante throws o mediante try-catch.

## Clases principales de servlets

Para poder trabajar con servlets, debemos importar los paquetes **java.io** (para el PrintWriter), **javax.servlet** (para el HttpServlet) y **javax.servlet.http** (para el HttpServletRequest y HttpServletResponse).

Los objetos para la definición de servlets se encuentran divididos en dos paquetes: **javax.servlet.\*,** que proporciona clases necesarias para crear servlets genéricos, independientes del protocolo utilizado, y **javax.servlet.http.\*,** que proporciona las clases que definen un servlet específico para el protocolo HTTP.

Servlet define la funcionalidad básica que tiene un servlet como es su ciclo de vida (métodos init, destroy) y procesar requests (método service).

Es implementado por la clase GenericServlet.

**SingleThreadModel** es una interfaz utilizada para marcar los servlets que se desea que se ejecuten en forma secuencial por requests simultáneos, evitando posibles problemas de procesamiento paralelo.

**HttpServlet** agrega la funcionalidad para procesar los variados tipos de request HTTP (principalmente los métodos doGet, doPost para procesar formularios GET y POST, respectivamente).

**HttpServletRequest** proporciona información del request del cliente al servlet a través del protocolo HTTP. Se pueden obtener datos como encabezados, cookies y características genéricas como dirección del cliente y parámetros recibidos desde él (getParameter). Permite también asociar objetos (setAttribute) y acceder al objeto HttpSession.

**HttpServletResponse** asiste a un servlet para enviar una respuesta al cliente a través de un canal de comunicación binario o de texto (getWriter). Provee también funcionalidad específica para respuestas HTTP como enviar encabezados, cookies (addCookie) y errores con códigos numéricos como 403 (sendError).

**HttpSession** permite identificar un usuario a través de más de una página, para esto se le asigna un identificador único que se mantiene mientras el usuario navega en el sitio (getId). Permite asignar un tiempo máximo de inactividad y se le pueden asociar objetos para compartir entre servlets (setAttribute y getAttribute).

## Métodos de información del servidor con servlets

Los servlets pueden obtener información del servidor, ya sean datos propios del servidor o mediante el envío de información desde un formulario HTML. Tenemos los siguientes métodos:

* Nombre del servidor: request.getServerName()
* Número de puerto: request.getServerPort()
* Software del servidor: getServletContext().getServerInfo()
* Obtener la dirección IP del cliente: request.getRemoteAddr()
* Obtener el nombre del cliente Host: request.getRemoteHost()

O mediante formularios HTML por medio de GET como una cadena añadida a la URL o POST:

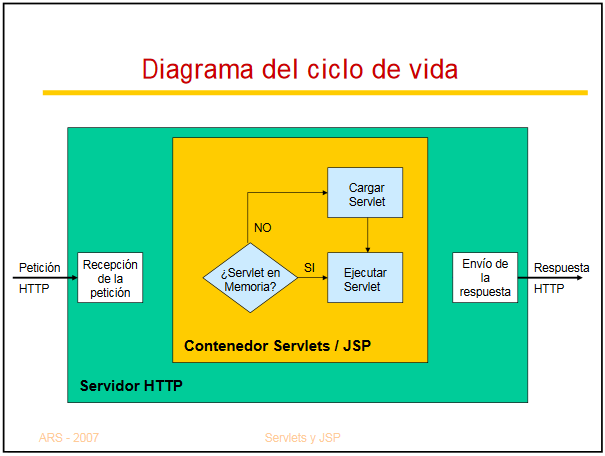
El método String **getParamete**r("nombreParámetro") devuelve el **valor asociado al objeto** que se ha denominado como nombreParámetro en la petición. Tenemos que tener en cuenta que se **diferencia entre mayúsculas y minúsculas**. Devuelve null si no aparece dicho parámetro en la petición.

* Método getParameterValues("NombreParametros"): devuelve un array de valores. Se utiliza para capturar objetos que forman conjuntos en un formulario, como puede ser un grupo de radiobutton HTML.
* GetParameterNames( ): devuelve una enumeración de los parámetros de petición sin ningún orden.

## Ciclo de vida de un servlet

Los servlets pueden obtener información del servidor, ya sean datos propios del servidor o mediante el envío de información desde un formulario HTML. Tenemos los siguientes métodos

* El servidor recibe una petición que ha de ser manejada por un servlet.ƒ
* El servidor comprueba si existe una instancia creada en memoria de la clase servlet correspondiente. Si no, la crea.ƒ
* Las peticiones posteriores de otros usuarios utilizarán la misma instancia.
* El objeto servlet permanece en memoria mientras el servidor siga en funcionamiento.

****

## Métodos implícitos en el ciclo de vida de un servlet

**init**ƒ

* Se ejecuta una vez, la primera vez que es invocado el servlet (el servlet se carga en memoria y se ejecuta sólo la primera vez que es invocado. El resto de peticiones generan un hilo).

ƒ

**Service (no debe sobreescribirse)**

ƒ

* Se ejecuta cada vez que se produce una nueva petición.
* Dentro de esta función se invoca a doGet o a doPost.

ƒ

**doGet y doPost**

ƒ

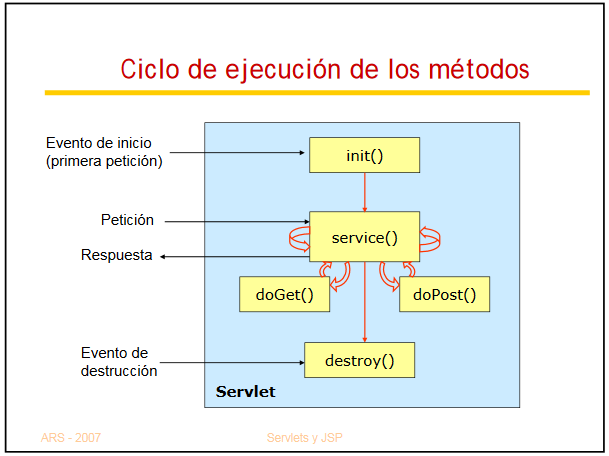
* Manejan las peticiones GET y POST.
* Incluyen el código principal del servletƒ
* La ejecución del servlet finalizará cuando termine la ejecución de estos métodos.

ƒ

**destroy**

ƒ

* Se invoca cuando el servidor decide eliminar el servlet de la memoria (NO después de cada petición).



Referencia: http://informatica.uv.es/it3guia/ARS/transparencias\_2c/Tema13\_Servlet-JSP.pdf

**JSP (JavaServer Pages)**

* Fichero con código (X)HTML que incluye scripts codificados en Java
* Permite usar (X)HTML para definir gran parte de la página
* E introducir código Java en las partes dinámicas de la página
* Mediante etiquetas especializadas (Custom Tags) que amplían la sintaxis de HTML
* Se compila y se convierte en un servlet (solo la primera vez que se invoca)
* Se ejecuta como un servlet.
* Las JavaServer Pages (JSP) permiten escribir código HTML e insertar código Java para las partes dinámicas.
* Con JSP es más fácil que se distribuya la tarea de diseño de la página web y la programación de la aplicación web
* Aunque no es estrictamente obligatorio, una página JSP se suele transformar en el código fuente de un servlet, que después se compila y ejecuta.

## Directivas JSP: page

* Todas las páginas JSP deberían incluirla. Atributos habituales:
  + language: lenguaje de programación (java por defecto).
  + contenttype: tipo de contenido de la página (text/html por defecto).
  + isErrorPage: indica si es una página de error (false por defecto).
  + errorPage: página a la que dirigirse si ocurre una excepción procesando esta página.

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

isErrorPage='false' errorPage='error.jsp' %>

## Otras directivas JSP

* include: permite incluir directamente el código de otro fichero en el punto en que aparezca la directiva.
* import: permite importar clases Java utilizadas en la página JSP.

<%@ include file='footer.html' %>

<%@ page import='java.util.\*' %>

## Scriptlets

* Permiten incrustar código escrito en otro lenguaje de programación (normalmente Java):
  + <%= expresión %>: evalúa la expresión y muestra el resultado en la página.
  + <% sentencias %>: ejecuta las sentencias, sin mostrar nada en la página.
  + <%! declaraciones %>: declaraciones de variables.

<%!-- los siguientes scriptlets son equivalentes --%>

<%= user.getName() %>

<% out.println(user.getName()); %>

## Scriptlets (aviso)

* Con <%! declaraciones %> se declaran atributos y métodos de instancia:
  + Todas las peticiones a la página JSP comparten la misma instancia del servlet asociado, por lo que también comparten la misma copia de los atributos de instancia.
  + Por tanto, no se debe declarar variables locales de la página con esta directiva, sino con:

<% String text = new String(); %>

## Acciones: jsp:include

* La acción jsp:include invoca al servlet o JSP dado e incluye en resultado de su ejecución en el punto actual del documento JSP desde el cual se incluya.
* El control retorna finalmente a la página inicial.
* Opcionalmente, se pueden pasar parámetros.

<jsp:include page='header.jsp'>

<jsp:param name='title' value='Welcome' />

</jsp:include>

## Acciones: jsp:forward

* La acción jsp:forward invoca al servlet o JSP dado e incluye en resultado de su ejecución en el punto actual del documento JSP desde el cual se incluya, sin que retorne el control a la página inicial.
* Opcionalmente, se pueden pasar parámetros.

<jsp:forward page='list.jsp'>

<jsp:param name='order' value='date' />

</jsp:forward>

**Objetos predefinidos**

**Request**

* Objeto HttpServletRequest que permite acceder a la información de la solicitud

**Response**

* Objeto HttpServletResponse para generar la respuesta

**Sesión**

* Objeto HttpSession asociado a la petición
* Si no hubiera sesión será null

**Out**

* Objeto JspWriter (similar a un PrintWriter) para generar la salida para el cliente

**Application**

* El objeto ServletContext del contenedor web.