



**ACCIÓN FORMATIVA:**  
**< PROGRAMACIÓN JAVA INTERMEDIO >**

## ICONO TRAINING



 [www.iconotc.com](http://www.iconotc.com)

 [linkedin.com/company/icono-training-consulting](https://linkedin.com/company/icono-training-consulting)

 [training@iconotc.com](mailto:training@iconotc.com)

## FORMADORA



Ana Isabel Vegas

Consultora / Formadora en Tecnologías de la Información.

¡Síguenos en las Redes Sociales!




# JAVA INTERMEDIO

## DURACIÓN

 28 horas

## LUGAR/FECHAS /HORARIO

 Días 17, 19, 24, 26 y 31 de Octubre, 2 y 7 de Noviembre. De 15:00 - 19:00 horas.

## CONTENIDO:

 Acceso a Bases de datos con Java

 Desarrollo de aplicaciones web.

- Introducción aplicaciones web
- Peticiones http
- Servlets
- JSP
- Manejo de sesiones
- Parámetros iniciales del servlet
- Parámetros iniciales de contexto
- Cookies
- Filtros

## INDICE

|  |    |
|--|----|
| INDICE .....                                     | 4  |
| 1.- CONEXIÓN CON BASES DE DATOS .....            | 7  |
| API JDBC .....                                   | 7  |
| TIPOS DE CONTROLADORES.....                      | 7  |
| ABRIR Y CERRAR CONEXIONES.....                   | 10 |
| CREAR CONSULTAS CON SQL .....                    | 11 |
| PROCESAR DATOS .....                             | 14 |
| MANEJO DE TRANSACCIONES.....                     | 15 |
| METADATOS.....                                   | 16 |
| 2. INTRODUCCION APLICACIONES WEB.....            | 17 |
| ESTRUCTURA DE UN WAR.....                        | 17 |
| PATRON MVC .....                                 | 18 |
| 3. PETICIONES HTTP .....                         | 20 |
| METODO GET .....                                 | 20 |
| METODO POST .....                                | 21 |
| SOBREESCRITURA URL .....                         | 21 |
| ENVIO DE PARAMETROS DE UN FORMULARIO .....       | 22 |
| 4. SERVLETS.....                                 | 23 |
| CARACTERÍSTICAS DE LOS SERVLETS .....            | 23 |
| API DE DESARROLLO DE SERVLETS.....               | 24 |
| CICLO DE VIDA DE UN SERVLET.....                 | 28 |
| MAPEO DEL SERVLET EN EL DESCRIPTOR WEB.XML ..... | 30 |
| MAPEO DEL SERVLET A TRAVÉS DE ANOTACIONES.....   | 31 |
| ATRIBUTOS .....                                  | 32 |
| LANZAMIENTO DE SOLICITUDES .....                 | 33 |
| 5. JSP.....                                      | 35 |
| CICLO DE VIDA DE LAS PÁGINAS JSP .....           | 35 |
| INCLUIR CODIGO JAVA EN UNA JSP .....             | 36 |
| DIRECTIVAS.....                                  | 39 |
| ACCIONES .....                                   | 41 |

|  |    |
|--|----|
| OBJETOS IMPLICITOS .....                               | 47 |
| LENGUAJE DE EXPRESIONES EL.....                        | 48 |
| ETIQUETAS JSTL.....                                    | 52 |
| 6. MANEJO DE SESIONES.....                             | 59 |
| API DE HTTPSESSION.....                                | 59 |
| CREAR UNA SESION.....                                  | 60 |
| ALMACENAMIENTO DE ATRIBUTOS DE SESIÓN .....            | 60 |
| ACCESO A ATRIBUTOS DE SESIÓN .....                     | 61 |
| DESTRUCCIÓN DE LA SESIÓN .....                         | 61 |
| EJEMPLO.....   | 62 |
| 7. PARAMETROS INICIALES DEL SERVLET.....               | 63 |
| DECLARACION PARAMETROS INICIALES DEL SERVLET .....     | 63 |
| LA API DE SERVLETCONFIG.....                           | 63 |
| RECUPERAR LOS PARAMETROS EN EL SERVLET.....            | 64 |
| EJEMPLO.....   | 64 |
| 8. PARAMETROS INICIALES DE CONTEXTO .....              | 66 |
| API DE SERVLETCONTEXT.....                             | 66 |
| DECLARAR LOS PARAMETROS DE CONTEXTO EN EL WEB.XML..... | 67 |
| CICLO DE VIDA DE LA APLICACIÓN WEB .....               | 67 |
| LISTENERS DE CONTEXTO .....                            | 67 |
| CONFIGURACIÓN DEL RECEPTOR DE EVENTOS .....            | 68 |
| RECUPERAR LOS PARAMETROS DE CONTEXTO .....             | 68 |
| EJEMPLO.....   | 69 |
| 9. COOKIES .....                                       | 70 |
| API DE COOKIE.....                                     | 70 |
| FUNCIONAMIENTO DE LAS COOKIES .....                    | 70 |
| CREAR COOKIES.....                                     | 71 |
| ESTABLECER EL TIEMPO DE PERMANENCIA .....              | 71 |
| ALMACENAR LA COOKIE EN EL NAVEGADOR .....              | 71 |
| BORRAR LA COOKIE.....                                  | 71 |
| OBTENER TODAS LAS COOKIES DEL NAVEGADOR .....          | 72 |
| BUSCAR UNA COOKIE DETERMINADA .....                    | 72 |
| MODIFICAR EL VALOR DE LA COOKIE.....                   | 72 |
| ESTABLECER Y LEER COMENTARIOS DE LA COOKIE .....       | 72 |

|   |    |
|---|----|
| ESPECIFICAR PARA QUE DOMINIO FUE CREADA .....       | 73 |
| EJEMPLO.....  | 73 |
| 10.FILTROS .....                                    | 74 |
| APLICACIÓN DE FILTROS A SOLICITUDES ENTRANTES ..... | 74 |
| USO DE FILTROS .....                                | 75 |
| APLICACION DE VARIOS FILTROS .....                  | 76 |
| API DE FILTRO.....                                  | 76 |
| CICLO DE VIDA DE UN FILTRO .....                    | 77 |
| CONFIGURACIÓN DEL FILTRO .....                      | 78 |
| EJEMPLO.....  | 80 |
| ÍNDICE DE GRÁFICOS.....                             | 81 |

## 1.- CONEXIÓN CON BASES DE DATOS

### API JDBC

**JDBC (Java Database Connectivity)** está compuesto por un número determinado de clases e interfaces que permiten a cualquier programa escrito en Java acceder a una base de datos.

Este conjunto de clases reside en los paquetes: java.sql y javax.sql

javax.sql añade funciones de servidor como los RowSet, los pools de conexiones o las transacciones.

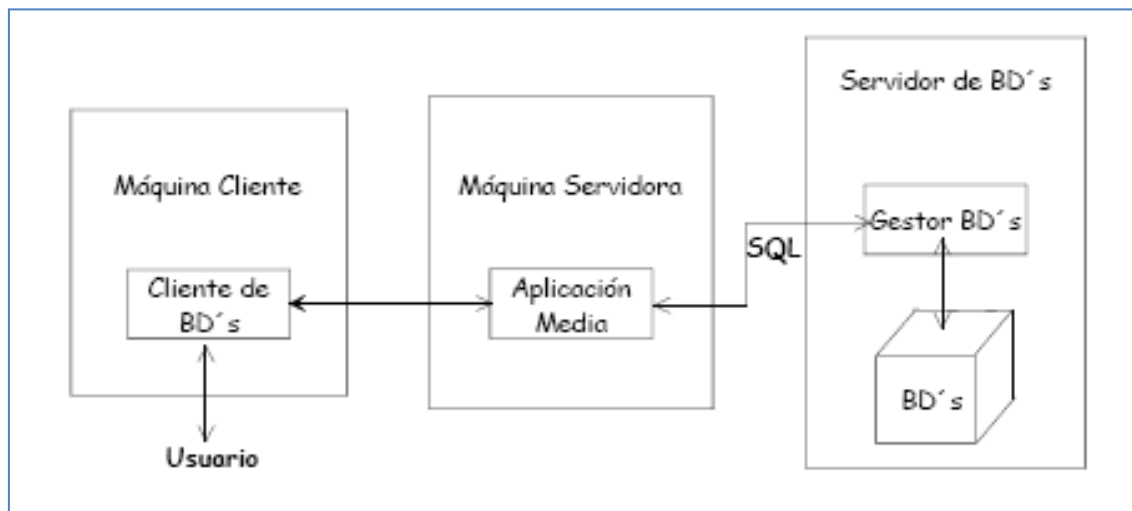


Gráfico 1. Esquema de conexión

Para podernos conectar a la base de datos necesitamos un controlador o también conocido como Driver. Este driver no es más que la implementación del API JDBC para un gestor de Base de datos concreto Oracle, MySQL, Derby, ...etc.

### TIPOS DE CONTROLADORES

Los diferentes tipos de controladores que tenemos son:

- Puente JDBC-ODBC
- 100 % Java nativo
- 100 % Java / Protocolo independiente
- 100 % Java / Protocolo nativo

## PUENTE JDBC-ODBC

Muchos servidores de BD's utilizan protocolos específicos para la comunicación. Esto implica que el cliente tiene que aprender un lenguaje nuevo para comunicarse con el servidor. Microsoft ha establecido una norma común para comunicarse con las bases de datos llamada Conectividad Abierta de Bases de Datos (ODBC). Hasta que esta norma no apareció, los clientes eran específicos del servidor. Utilizando la API ODBC lo que conseguimos es desarrollar software independiente del servidor.

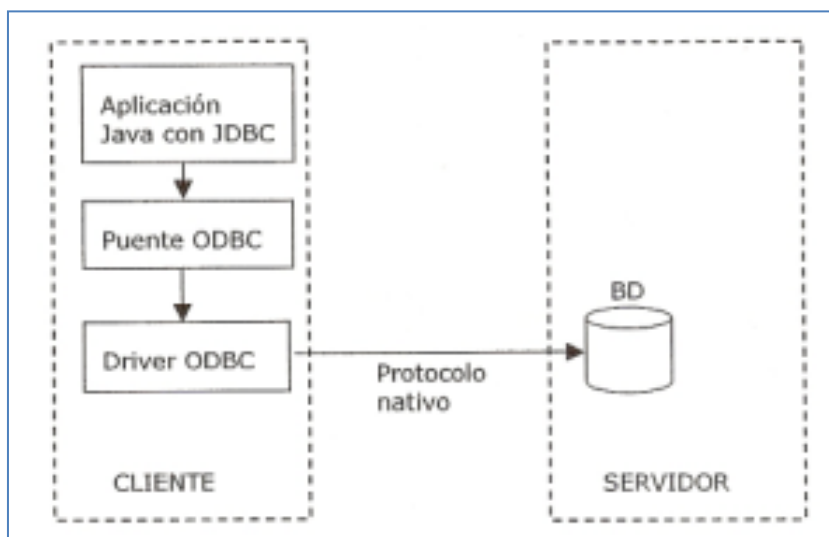


Gráfico 2. Conexión a través de ODBC

## 100 % JAVA NATIVO

Son controladores que usan el API de Java JNI (Java native interface) para presentar una interfaz Java a un controlador binario nativo del SGBD.

Su uso, igual que los anteriores, implica instalar el controlador nativo en la máquina cliente. Suelen tener un rendimiento mejor que los controladores escritos en Java completamente, aunque un error de funcionamiento de la parte nativa del controlador puede causar problemas en la máquina virtual de Java.



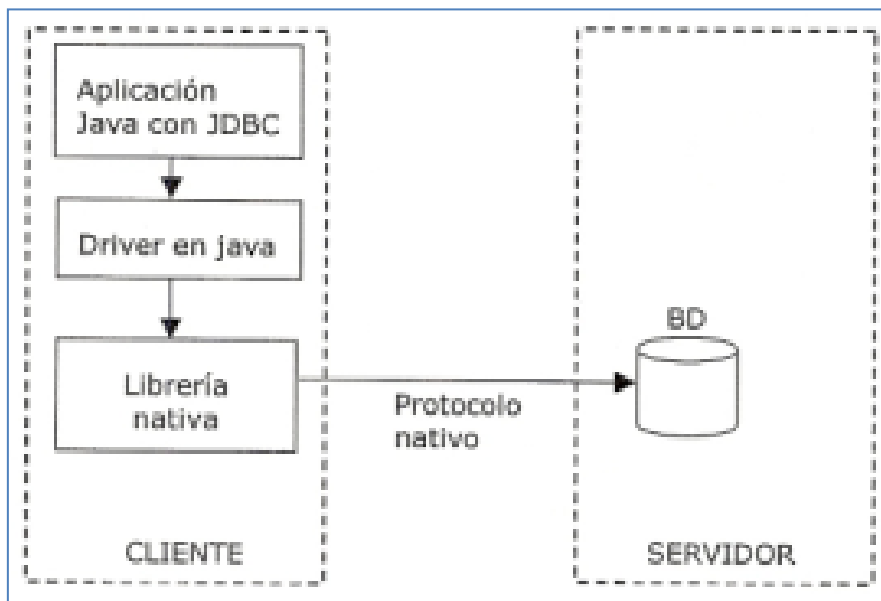


Gráfico 3. Conexión a través de Librería nativa

### 100 % JAVA / PROTOCOLO INDEPENDIENTE

Controladores escritos en Java que definen un protocolo de comunicaciones que interactúa con un programa de middleware que, a su vez, interacciona con un SGBD. El protocolo de comunicaciones con el middleware es un protocolo de red independiente del SGBD y el programa de middleware debe ser capaz de comunicar los clientes con diversas bases de datos.

El inconveniente de esta opción estriba en que debemos tener un nivel más de comunicación y un programa más (el middleware).

### 100 % JAVA / PROTOCOLO NATIVO

Son los controladores más usados en accesos de tipo intranet (los usados generalmente en aplicaciones web).

Son controladores escritos totalmente en Java, que traducen las llamadas JDBC al protocolo de comunicaciones propio del SGBD. No requieren ninguna instalación adicional ni ningún programa extra.

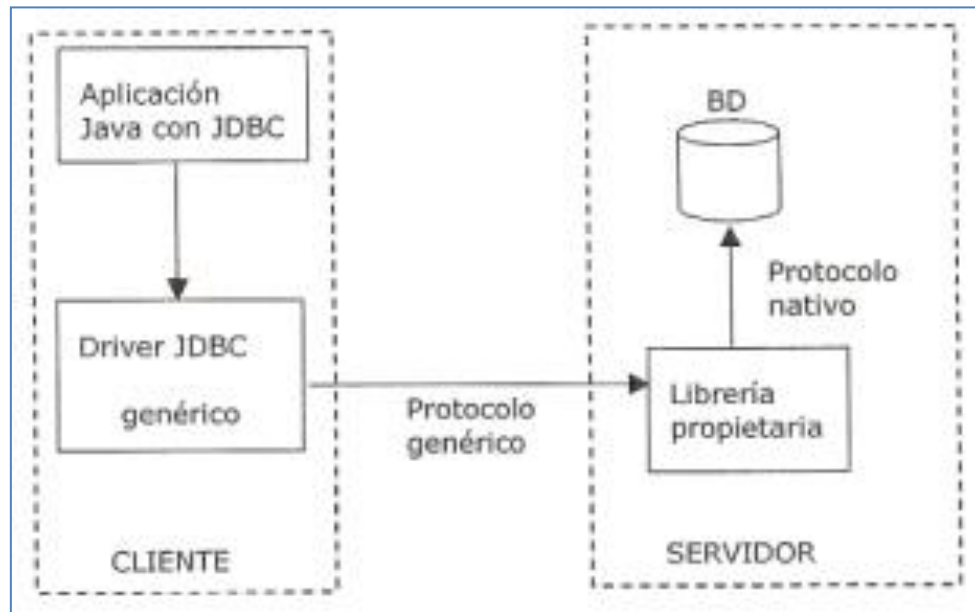


Gráfico 4. Conexión a través de protocolo nativo

## ABRIR Y CERRAR CONEXIONES

Para poder acceder a una base de datos es necesario establecer una conexión.

Antes de conectarnos con la base de datos hay que registrar el controlador apropiado. Java utiliza la clase **DriverManager** para cargar inicialmente todos los controladores JDBC disponibles y que deseemos utilizar.

Para cargar un controlador se emplea del método **forName()** de la clase **Class** que devuelve un objeto **Class** asociado con la clase que se le pasa como parámetro.

La interfaz **Connection** es la que se encarga de la conexión con la base de datos. Mediante el método **getConnection()**, obtenemos un objeto de la clase **Connection**. Esta clase contiene todos los métodos que nos permiten manipular la base de datos.

Para cerrar la conexión utilizamos el método **close()** de la interfaz **Connection**.

```
private Connection conexion;  
  
public void abrirConexion() {  
    try {  
        // Cargar el driver de Derby  
        Class.forName("org.apache.derby.jdbc.ClientDriver");  
  
        // Abrir la conexion a la BBDD  
        conexion = DriverManager.getConnection("jdbc:derby://localhost:1527/Tienda",  
            "curso", "curso");  
  
    } catch (SQLException ex) {  
        System.out.println("Error al abrir la conexion");  
        ex.printStackTrace();  
    } catch (Exception ex) {  
        System.out.println("error al cargar el driver");  
    }  
}
```

Gráfico 5. Fragmento para abrir una conexión a Derby

```
public void cerrarConexion() {  
    try {  
        // cerrar la conexion  
        conexion.close();  
    } catch (SQLException ex) {  
        System.out.println("Error al cerrar la conexion");  
    }  
}
```

Gráfico 6. Fragmento para cerrar la conexión

## CREAR CONSULTAS CON SQL

Para crear una consulta a la base de datos podemos utilizar 3 interfaces diferentes, cada uno tiene un uso específico:

- **Statement**; Para ejecutar queries completas, es decir, tenemos todos los datos de la query.
- **PreparedStatement**; Para ejecutar queries parametrizadas. No tenemos todos los datos en este momento y utilizamos parámetros.
- **CallableStatement**; Para ejecutar queries a través de procedimientos almacenados definidos en la BBDD.

### Statement

Para incluir una sentencia SQL utilizaremos la interfaz **Statement**, con el método **createStatement()** de la interfaz Connection.

Al crear una instancia del objeto Statement, podremos realizar sentencias SQL sobre la base de datos. Existen dos tipos de sentencias a realizar:

- Sentencias de modificación (update); Engloban a todos los comandos SQL que no devuelven ningún tipo de resultado como pueden ser los comandos INSERT, UPDATE, DELETE o CREATE.
- Sentencias de consulta (query); Son sentencias del tipo SELECT (sentencias de consulta) que retornan algún tipo de resultado.

Para las sentencias “update” la clase Statement nos proporciona el método siguiente que devolverá el número de filas afectadas por la sentencia SQL:

```
public abstract int executeUpdate(String sentenciaSQL)
```

```
// 1.- Abrir la conexion
abrirConexion();

// 2.- Preparar la query
Statement stm = conexion.createStatement();

// 3.- Ejecutar la query
int resultados = stm.executeUpdate("delete * from APP.Productos");
```

Gráfico 7. Ejemplo executeUpdate

Para las sentencias “query” la clase Statement utiliza el método siguiente:

```
public abstract ResultSet executeQuery(String sentenciaSQL)
```

```
// 1.- Abrir la conexion
abrirConexion();

// 2.- Preparar la query
Statement stm = conexion.createStatement();

// 3.- Ejecutar la query
ResultSet resultados = stm.executeQuery("select * from APP.Productos");
```

Gráfico 8. Ejemplo executeQuery

## PreparedStatement

Define métodos para trabajar con instrucciones SQL pre compiladas, que son más eficientes.

También se usan para poder utilizar instrucciones SQL parametrizadas. Para establecer parámetros en una instrucción SQL basta con sustituir el dato por un signo de interrogación. (Select \* from Tabla where Nombre=?).

El parámetro se sustituye por el valor mediante el método setXXX(int lugar, XXX Valor).

PreparedStatement no utiliza los métodos de Statement para ejecutar las queries, en su lugar, se pasa la query en el propio constructor.

Igual que antes podemos ejecutar dos tipos de consultas query y update.

```
// 1.- Abrir la conexion
abrirConexion();

// 2.- Preparar la query
String query = "insert into APP.Productos values (?, ?, ?) ";
PreparedStatement stm = conexion.prepareStatement(query);

// sustituimos los parámetros
stm.setInt(1, p.getId());
stm.setString(2, p.getDescripcion());
stm.setDouble(3, p.getPrecio());

// 3.- Ejecutar la query
int resultados = stm.executeUpdate();
```

Gráfico 9. Ejemplo PreparedStatement con executeUpdate

```
// 1.- Abrir la conexion
abrirConexion();

// 2.- Preparar la query
String query = "select * from APP.Productos where ID = ? ";
PreparedStatement stm = conexion.prepareStatement(query);

// asignar parámetros
stm.setInt(1, id);

// 3.- Ejecutar la query
ResultSet resultados = stm.executeQuery();
```

Gráfico 10. Ejemplo PreparedStatement con executeQuery

## CallableStatement

Los objetos de este tipo se crean a través del método de Connection, prepareCall(). Se pasa como argumento de este método la llamada al procedimiento almacenado. Los CallableStatement también pueden ser parametrizados.

Un procedimiento almacenado es una macro registrada en la base de datos y que realiza operaciones de cualquier tipo.

```
// 1.- Abrir la conexion
abrirConexion();

// 2.- Preparar la query
String query = "{call procedimiento(?)}";
CallableStatement cstm = conexion.prepareCall(query);
cstm.setDouble(1, precio);

// 3.- Ejecutar la query
ResultSet resultados = cstm.executeQuery();
```

Gráfico 11. Ejemplo CallableStatement con executeQuery

## PROCESAR DATOS

Para procesar los datos recibidos tras ejecutar la query depende del tipo de consulta que hemos lanzado.

- Recordamos que si era una query de consulta devolvía un objeto de tipo ResultSet.
- Si era una query de modificación entonces recogemos una valor entero que será el número de registros modificados

La interfaz ResultSet contendrá las filas o registros obtenidas mediante la ejecución de la sentencia de tipo “query”. Cada una de esas filas obtenidas se divide en columnas.

La interfaz ResultSet contiene un puntero que está apuntando a la fila actual. Inicialmente está apuntando por delante del primer registro. Para avanzar el puntero utilizamos el método **next()**.

Una vez posicionados en una fila concreta, podemos obtener los datos de una columna específica utilizando los métodos `getxxx()` que proporciona la interfaz `ResultSet`, la “xxx” especifica el tipo de dato presente en la columna.

Para cada tipo de dato existen dos métodos `getxxx()`:

- `getxxx(String nombreColumna)`; Donde especificamos el nombre de la columna donde se encuentra el dato.
- `getxxx(int numeroColumna)`; Donde se especifica la posición de la columna dentro de la consulta. Siempre empezando desde 1.

```
// 3.- Ejecutar la query
ResultSet resultados = cstm.executeQuery();

// 4.- Recorrer la coleccion de registros
while (resultados.next()) {
    lista.add(new Producto(resultados.getInt("ID"),
                           resultados.getString("DESCRIPCION"),
                           resultados.getDouble("PRECIO")));
}
```

Gráfico 12. Procesar resultados

## MANEJO DE TRANSACCIONES

Para poder manejar transacciones debemos utilizar el método `setAutocommit` para especificar si queremos un commit implícito o no.

```
Connection.setAutocommit(boolean);
```

- Si lo ponemos a `true`; estamos diciendo que se hace un commit implícito.
- Si se pone a `false`; no se realizará ninguna modificación en la base de datos hasta que explícitamente se haga un commit.

Si la conexión no hace commit implícito:

- `connection.commit()`: valida la transacción
- `connection.rollback()`: anula la transacción

## METADATOS

Los metadatos se pueden considerar como información adicional a los datos que almacena la tabla.

Se utiliza un objeto `ResultSetMetaData` para recoger los metadatos de una consulta. Dicho objeto expone los siguientes métodos:

- `getColumnName()`; devuelve el nombre de la columna.
- `getColumnCount()`; nos indica el número de columnas de la consulta.
- `getTableName()`; devuelve el nombre de la tabla.
- `getColumnType()`; nos dice el tipo de datos de la columna.
- `isReadOnly()`; especifica si los datos son de solo lectura.

```
// 5.- Obtener los metadatos
ResultSetMetaData metadataRS = resultados.getMetaData();
int columnCount = metadataRS.getColumnCount();
for (int i = 0; i < columnCount; i++) {
    System.out.println("nombre columna " +
                      metadataRS.getColumnName(i));
}
```

Gráfico 13. Obtención de los metadatos de la consulta



### RECUERDA QUE...

- Con el API JDBC podemos acceder una base de datos sin importarnos el fabricante.
- JDBC permite abrir y cerrar conexiones, así como ejecutar consultas a la BBDD.



## 2. INTRODUCCION APLICACIONES WEB

Una aplicación web se despliega en el contenedor Web que todo servidor de aplicaciones compatible con JEE debe poseer.

Este tipo de aplicaciones se componen de los siguientes recursos:

- Servlets; clases java que administra y se ejecutan en el contenedor Web.
- JSP; páginas web compuestas de HTML, CSS, ...etc y código Java.
- Clases Java
- Descriptores de despliegue; archivo xml donde se configura los componentes de la aplicación.

Para poder ejecutar este tipo de aplicaciones necesitamos los siguientes recursos:

- Un servidor de aplicaciones donde se desplegará la aplicación.
- Un navegador web tipo Explorer, Mozilla, Chrome, ...etc.

A lo largo de este modulo iremos conociendo cada uno de los componentes web y su gestión.

### ESTRUCTURA DE UN WAR

Como ya vimos en el modulo 2 Arquitectura JEE una aplicación web se empaqueta en un modulo con extensión .war.

Este modulo se puede desplegar directamente en el contenedor web.

Da igual el entorno de desarrollo que utilicemos ya que la estructura de un war forma parte de la especificación JEE y esto marca un estándar por lo cual siempre será la misma.

En la imagen siguiente podemos ver esta estructura.

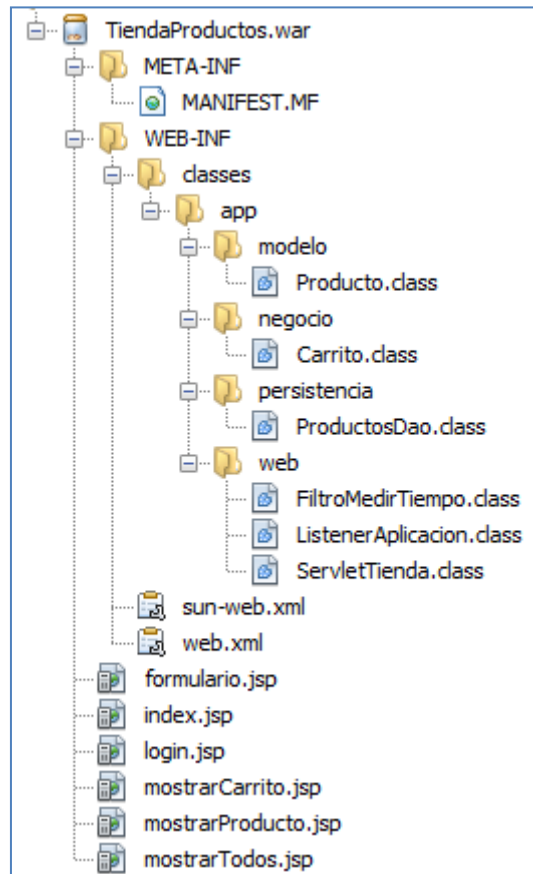


Gráfico 14. Estructura de un modulo .war

- Carpeta **META-INF**; contiene el archivo de manifiesto que utilizamos para almacenar información sobre la aplicación, control de versiones, autor, ...etc.
- Carpeta **WEB-INF**; Esta carpeta contiene:
  - los descriptores de despliegue **web.xml** y **sun-web.xml**; el primero de ellos es un estandar de JEE por lo cual su nombre no debe variar. El segundo toma su nombre del servidor de aplicaciones que estemos utilizando.
  - carpeta **classes**; aquí encontramos todas las clases java compiladas (.class) guardando la misma estructura que los paquetes definidos.
- Contenido web; en esta ubicación tenemos todo el contenido web: paginas **jsp**, **html**, hojas de estilo **css**, archivos **javascript**, **imagenes**, ...etc.

## PATRON MVC

Una aplicación web diseñada con la arquitectura del patrón MVC se caracteriza por lo siguiente:

- Un servlet actúa como **controlador**, que verifica los datos recibidos, actualiza el modelo con dichos datos y selecciona la próxima vista como respuesta.
- Una página de JSP actúa como **vista**. En ella se representa la respuesta HTML, recuperando los datos del modelo necesarios para generar la respuesta, y se proporcionan formularios HTML que permiten la interacción del usuario.
- Las clases Java actúan como **modelo**, que implementa la lógica de negocio de la aplicación web.

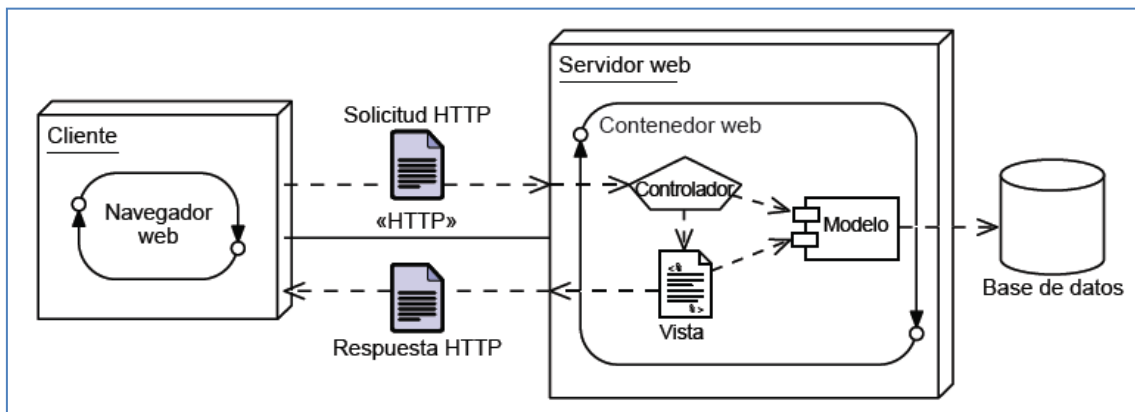


Grafico 15. Patrón MVC en aplicaciones web.



### RECUERDA QUE...

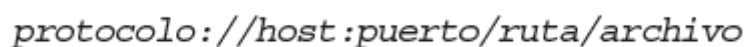
- Un modulo war tiene una determinada estructura que la marca el estándar JEE.
- El patrón MVC en aplicaciones web define los siguientes componentes: El servlet actúa de controlador, las paginas jsp como vistas y las clase java como modelo.

### 3. PETICIONES HTTP

Hypertext Transfer Protocol o HTTP (en español protocolo de transferencia de hipertexto) es el protocolo usado en cada transacción de la World Wide Web.

A través de dicho protocolo podemos emitir peticiones HTTP desde el cliente (navegador web) hasta el servidor. Una vez tramitada la petición, el servidor devolverá una respuesta HTTP al cliente que este interpretará mostrando el resultado al usuario.

Para poder emitir peticiones necesitamos de una URL. Una URL es un nombre canónico que localiza un recurso específico en Internet. Está formado por:



```
protocolo://host:puerto/ruta/archivo
```

Gráfico 16. Formato url

Detallamos cada componente de la url:

- **protocolo**; normalmente usaremos http o https para peticiones seguras.
- **host**; es el nombre o IP del servidor
- **puerto**; puerto de escucha para las aplicaciones web. Suele ser 8080 o únicamente 80.
- **ruta**; es el context path de la aplicación. Es un nombre que hace referencia a la carpeta raíz de la aplicación.
- **archivo**; puede ser una página web, un servlet o cualquier recurso de la aplicación.

Además de los componentes vistos también se pueden enviar datos en la url para su procesamiento. Estos viajarán de diferente forma dependiendo el método http elegido. A continuación vemos los dos más importantes: GET y POST.

#### METODO GET

Es el más simple de los dos. Su principal tarea es pedir al servidor que coja un recurso y lo envíe. Los parámetros se ven en la URL.

En la siguiente imagen vemos como en la url se separa la lista de parámetros con el carácter "?". El parámetro opción viaja con el valor 1. Si hubiese más de un parámetro estos irían unidos con el carácter "&".

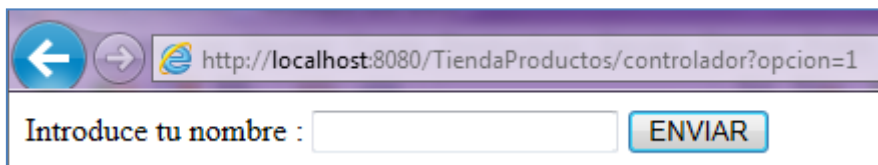


Gráfico 17. Petición url con método get

## METODO POST

Es más seguro. Podemos solicitar algo y al mismo tiempo enviar datos de un formulario al servidor. Los parámetros están ocultos, se envían en el cuerpo del mensaje.

Razones para usar POST en vez de GET

- El numero de caracteres el limitado usando GET (dependiendo del servidor).
- Los datos enviados con GET se añaden al final de la URL.
- No permite almacenar los argumentos del formulario.



Gráfico 18. Petición url con método post

En la imagen anterior vemos como se ha enviado el código del producto a buscar utilizando el método post. Esta vez no vemos el parámetro con su valor correspondiente ya que este viaja dentro del cuerpo de la petición.

## SOBREESCRITURA URL

Podemos adjuntar nuestros propios parámetros sobrescribiendo una url. Esto consiste en añadir el carácter separación (?) y a continuación detallar los parámetros necesarios.

```
<a href="controlador?opcion=1">Consultar todos los productos</a><br>
```

Gráfico 19. Sobreescritura URL

## ENVIO DE PARAMETROS DE UN FORMULARIO

A través del atributo name podemos establecer el nombre del parámetro. El valor será el que rellene el usuario.

También podemos hacer uso de los campos ocultos de esta forma el usuario no verá el dato que enviamos. Lo aconsejable con campos ocultos es utilizar el método POST ya que si utilizásemos GET estos se verían en la url.

En un formulario se puede elegir el método http. En nuestro ejemplo utilizamos POST.

```
<form action="controlador" method="post">
  Introduce Id del producto:
  <input type="text" name="codigo" />
  <input type="hidden" name="opcion" value="2" />
  <input type="submit" value="ENVIAR" />
</form>
```

Gráfico 20. Envío de parámetros en un formulario



### RECUERDA QUE...

- Podemos emitir peticiones a través de los métodos GET o POST.
- Con el método GET los parámetros de la petición se envían en la propia url por lo cual son visibles.
- Con el método POST los parámetros viajan en el cuerpo de la petición por lo cual no son visibles.

## 4. SERVLETS

Los servlets son objetos de negocio, escritos en Java, que residen en el servidor de aplicaciones.

Pueden recibir peticiones de un muchos clientes y generar respuestas (pueden generar documentos HTML, XML, texto plano...).

Una misma instancia de un servlet puede ejecutarse de manera concurrente para satisfacer peticiones solapadas en el tiempo.

Como vemos en la siguiente imagen, el servlet recoge la petición (request) y emite la respuesta (response).

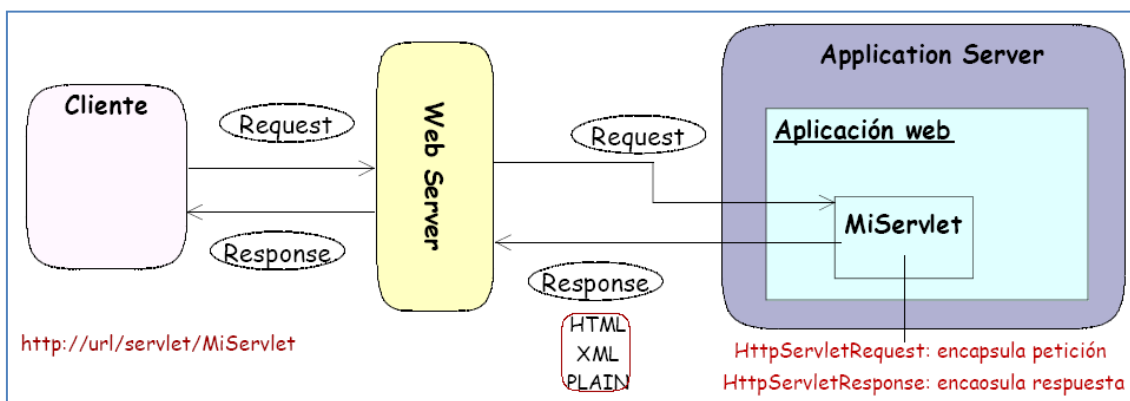


Gráfico 21. Petición y respuesta en un servlet

### CARACTERÍSTICAS DE LOS SERVLETS

- Son independientes del servidor utilizado y de su sistema operativo.
- Los servlets pueden llamar a otros servlets. De esta forma se puede distribuir de forma más eficiente el trabajo a realizar. Por ejemplo, se podría tener un servlet encargado de la interacción con los clientes y que llamara a otro servlet para que a su vez se encargara de la comunicación con una base de datos. De igual forma, los servlets permiten redireccionar peticiones de servicios a otros servlets (en la misma máquina o en una máquina remota).
- Los servlets pueden obtener fácilmente información acerca del cliente (la permitida por el protocolo HTTP), tal como su dirección IP, el puerto que se utiliza en la llamada, el método utilizado (GET, POST, ...), etc.
- Permiten además la utilización de cookies y sesiones, de forma que se puede guardar información específica acerca de un usuario determinado.
- Los servlets pueden actuar como enlace entre el cliente y una o varias bases de datos en arquitecturas cliente-servidor de 3 capas.

- Pueden realizar tareas de proxy para un applet. Debido a las restricciones de seguridad, un applet no puede acceder directamente por ejemplo a un servidor de datos localizado en cualquier máquina remota, pero el servlet sí puede hacerlo de su parte.
- Al igual que los programas CGI, los servlets permiten la generación dinámica de código HTML dentro de una propia página HTML. Así, pueden emplearse servlets para la creación de contadores, banners, etc.

### API DE DESARROLLO DE SERVLETS

Oracle proporciona un conjunto de clases Java para el desarrollo de servlets.

Este API se encuentra diseñado como una extensión del JDK. Consta de dos paquetes `javax.servlet` y `javax.servlet.http`. Este último es una particularización del primero para el caso del protocolo HTTP, que es el más utilizado. Mediante este diseño lo que se consigue es que se mantenga abierta la posibilidad de implementar servlets para otros protocolos existentes (FTP, POP, SMTP).

Una clase Java se dice que es un `HTTPServlet`, si hereda de la clase `javax.servlet.http.HttpServlet` y su entorno de ejecución es un servidor HTTP.

En realidad una clase que herede de `javax.servlet.GenericServlet` ya se considera un servlet, pero lo más común es que trabajemos con `HttpServlet`.



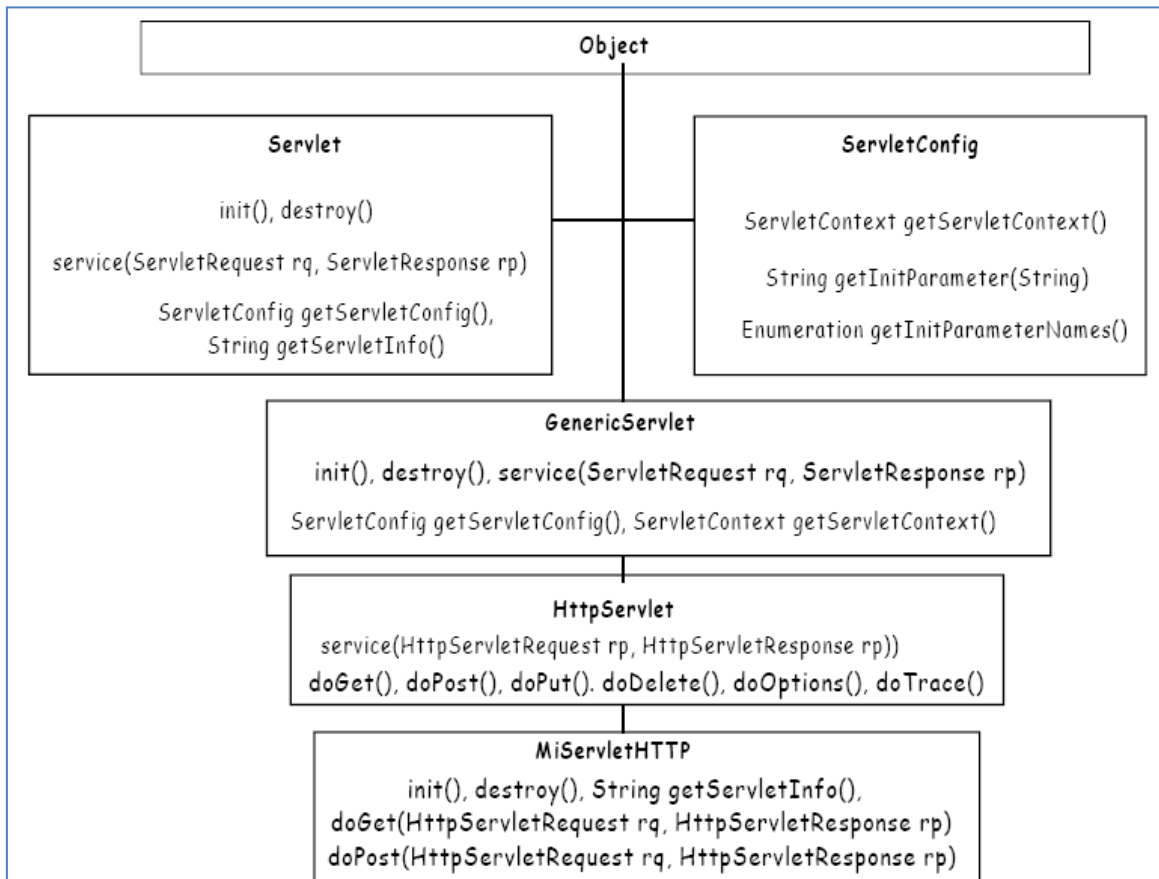


Gráfico 22. Jerarquía del API servlets

El paquete `javax.servlet` define los siguientes elementos:

- Interfaz Servlet
- Interfaz ServletRequest
- Interfaz ServletResponse
- Interfaz ServletConfig
- Interfaz ServletContext
- Clase GenericServlet

## INTERFAZ SERVLET

La deben implementar todos los servlets.

El servidor invoca a los métodos `init()` y `destroy()` para iniciar y detener un servlet.

Los métodos `getServletConfig()` y `getServletInfo()` retornan información acerca del servlet.

El método `service()` es invocado por el servidor para que el servlet realiza su servicio (una petición). Este método dispone de dos parámetros. uno del tipo de la interfaz `ServletRequest` y otro del tipo de la interfaz `ServletResponse`.

## INTERFAZ SERVLETREQUEST

Este interfaz encapsula la petición de servicio de un cliente. Define una serie de métodos tendentes a obtener información del servidor, solicitante y solicitud.

Los métodos más importantes son:

- `int getLength()` Devuelve el tamaño de la petición del cliente o -1 si es desconocido.
- `String getContentType()` Devuelve el tipo de contenido MIME de la petición o null si éste es desconocido.
- `String getProtocol()` Devuelve el protocolo y la versión de la petición como un String en la forma <protocolo>/<versión mayor>.<versión menor>
- `String getScheme()` Devuelve el tipo de esquema de la URL de la petición: http, https, ftp...
- `String getServerName()` Devuelve el nombre del host del servidor que recibió la petición..
- `int getServerPort()` Devuelve el número del puerto en el que fue recibida la petición.
- `String getRemoteAddr()` Devuelve la dirección IP del ordenador que realizó la petición.
- `String getRemoteHost()` Devuelve el nombre completo del ordenador que realizó la petición.
- `String getParameter(String)` Devuelve un String que contiene el valor del parámetro especificado, o null si dicho parámetro no existe. Sólo debe emplearse cuando se está seguro de que el parámetro tiene un único valor.
- `String[] getParameterValues(String)` Devuelve los valores del parámetro especificado en forma de un array de Strings, o null si el parámetro no existe. Útil cuando un parámetro puede tener más de un valor.
- `Enumeration getParameterNames()` Devuelve una enumeración en forma de String de los parámetros encapsulados en la petición. No devuelve nada si el InputStream está vacío.

## INTERFAZ SERVLETRESPONSE

Esta interfaz es utilizada por un servlet para enviar información al solicitante de una petición.

Los métodos más importantes son:

- `ServletOutputStream getOutputStream()` Permite obtener un ServletOutputStream para enviar datos binarios.
- `PrintWriter getWriter()` Permite obtener un PrintWriter para enviar caracteres.
- `setContentType(String)` Establece el tipo MIME de la salida. ("text/html")
- `setContentLength(int)` Establece el tamaño de la respuesta

## **INTERFAZ SERVLETCONFIG**

La utiliza un servidor para pasar información sobre la configuración a un servlet. Sus métodos los utiliza el Servlet para recuperar esta información, por ejemplo los parámetros iniciales del servlet. Un servlet puede recuperar el objeto de esta interfaz a través del método `getServletConfig()`.

Los métodos más importantes son:

- `String getInitParameter(String)` permite obtener el valor de un parámetro inicial, el nombre de este parámetro se especifica como argumento del método.
- `Enumeration getInitParameterNames()` permite obtener todos los nombres de los parámetros iniciales.

## **INTERFAZ SERVLETCONTEXT**

Define el entorno en que se ejecuta el servlet. Proporciona métodos que utilizan los servlets para acceder a información sobre el entorno.

La información acerca del servidor está disponible en todo momento a través de un objeto de la interface `ServletContext`. Un servlet puede obtener dicho objeto mediante el método `getServletContext()` aplicable a un objeto `ServletConfig`.

Los métodos más importantes son:

- `Object getAttribute(String)` Devuelve información acerca de determinados atributos del tipo clave/valor del servidor. Es propio de cada servidor.
- `String getMimeType(String)` Devuelve el tipo MIME de un determinado fichero.
- `public abstract String getRealPath(String)` Traduce una ruta de acceso virtual a la ruta relativa al lugar donde se encuentra el directorio raíz de páginas HTML
- `String getServerInfo()` Devuelve el nombre y la versión del servicio de red en el que está siendo ejecutado el servlet.

## **CLASE GENERICSERVLET**

Implementa el interfaz `Servlet`. Se puede heredar de esta clase para construir clases `Servlet` propias.

La clase `GenericServlet` es una clase abstract puesto que su método `service()` es abstract. Esta clase implementa dos interfaces, de las cuales la más importante es la interface `Servlet`.

Cualquier clase que derive de `GenericServlet` deberá definir el método `service()`. Es muy interesante observar los dos argumentos que recibe este método, correspondientes a las interfaces: `ServletRequest` y `ServletResponse`.

El primer argumento referencia a un objeto que describe por completo la solicitud de servicio que se le envía al servlet. Si la solicitud de servicio viene de un formulario HTML, por medio de ese objeto se puede acceder a los nombres de los campos y a los valores introducidos por el usuario; puede también obtenerse cierta información sobre el cliente.

El segundo argumento es un objeto con una referencia de la interface `ServletResponse`, que constituye el camino mediante el cual el método `service()` se conecta de nuevo con el cliente y le comunica el resultado de su solicitud. Además, dicho método deberá realizar cuantas operaciones sean necesarias para desempeñar su cometido: escribir y/o leer datos de un fichero, comunicarse con una base de datos, etc.

El método `service()` es realmente el corazón del servlet.

## **EL PAQUETE `JAVAX.SERVLET.HTTP`**

En la práctica, salvo para desarrollos muy especializados, todos los servlets deberán construirse a partir de la clase `HttpServlet`, sub-clase de `GenericServlet`.

La clase `HttpServlet` ya no es abstract y dispone de una implementación o definición del método `service()`. Dicha implementación detecta el tipo de servicio o método HTTP que le ha sido solicitado desde el browser y llama al método adecuado de esa misma clase (`doPost()`, `doGet()`, etc..).

Cuando el programador crea una sub-clase de `HttpServlet`, por lo general no tiene que redefinir el método `service()`, sino uno de los métodos más especializados (normalmente `doGet()`, `doPost()`), que tienen los mismos argumentos que `service()`: dos objetos de las clases `HttpServletRequest` y `HttpServletResponse` que implementa las interfaces `ServletRequest` y `ServletResponse`.

## **CLASES DE `JAVAX.SERVLET.HTTP`**

- `HttpServletRequest`; Amplía la interfaz `ServletRequest` y agrega métodos para acceder a los detalles de una solicitud HTTP.
- `HttpServletResponse`; Amplía la interfaz `ServletResponse` y agrega constantes y métodos para devolver respuestas específicas del HTTP.
- `HttpSession`; Se implementa por servlets para permitir sesiones navegador-servidor que abarcan múltiples pares de solicitudes-respuestas.
- `Cookie`; Representa una Cookie HTTP. Una Cookie es un conjunto de información que se genera en el servidor y se almacena en los clientes individuales.
- `HttpServlet`; Amplía de `GenericServlet` con el fin de utilizar interfaces `HttpServletRequest` y `HttpServletResponse`.

## **CICLO DE VIDA DE UN SERVLET**

En un GenericServlet tenemos los siguientes métodos para gestionar su ciclo de vida. Al ser los servlets componentes manejados por el contenedor web estos métodos serán invocados por él.

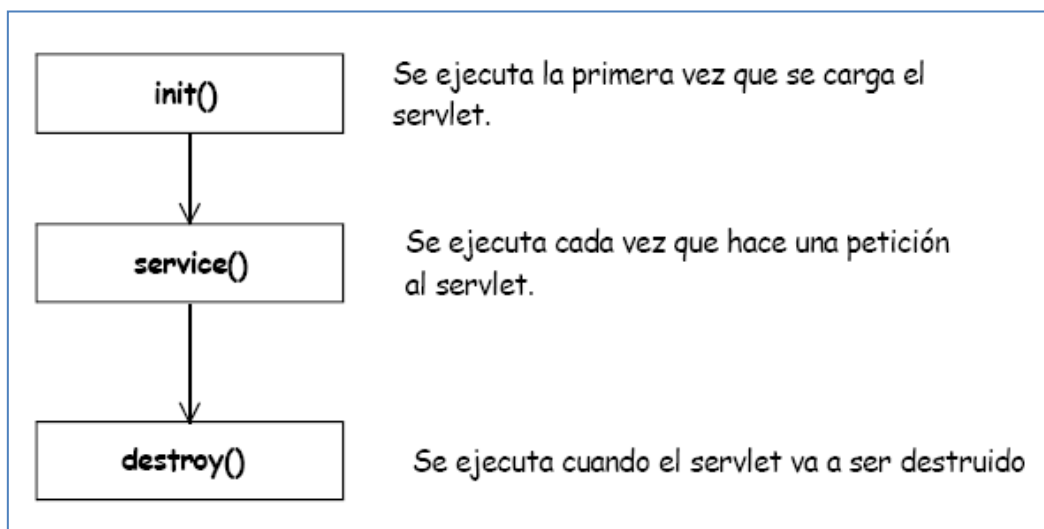


Gráfico 23. Métodos de ciclo de vida de un GenericServlet

El método **init** lo invoca el contenedor de servlets, una sola vez; resulta útil para inicializar recursos que serán necesarios en la ejecución del servlet (abrir ficheros, conectar bases de datos, establecer comunicaciones, etc.).

El método **destroy** se invoca cuando el servlet va a ser descargado del servidor; en su interior se debería programar la liberación de recursos utilizados en el método init.

El método **service** realiza el trabajo “cotidiano” del servlet: dar respuesta a las distintas peticiones de los clientes. Cada vez que un cliente realiza una petición (GET, POST, etc.) el contenedor invoca al método service.

En un HttpServlet los métodos de ciclo de vida son los siguientes:

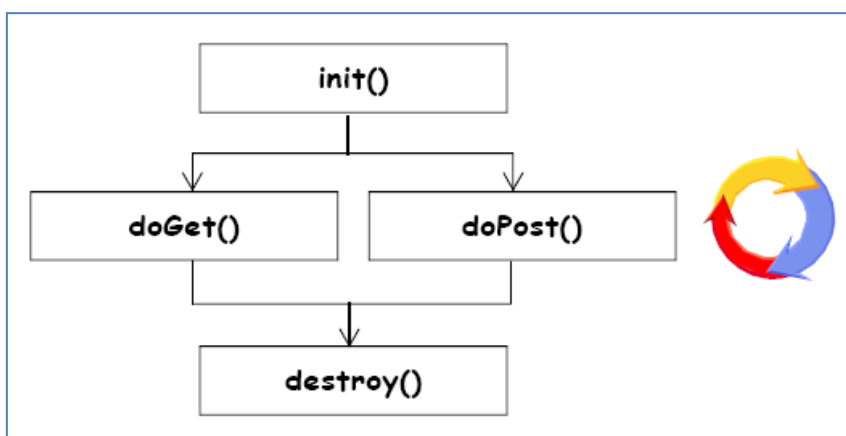


Gráfico 24. Métodos de ciclo de vida de un HttpServlet

Como podemos comprobar el método service se desglosa en dos:

- **goGet();** capturará las peticiones enviadas a través del método GET.
- **doPost();** capturará las peticiones enviadas a través del método POST.

```
public class ServletTienda extends HttpServlet {  
  
    // Este metodo lo invoca el contenedor web en el momento que destruye la instancia del servlet  
    @Override  
    public void destroy() {...}  
  
    // Este metodo lo invoca el contenedor web en el momento que crea la instancia del servlet  
    @Override  
    public void init(ServletConfig config) throws ServletException {...}  
  
    // Este metodo lo invoca el contenedor web cuando entra una petición por el método GET  
    @Override  
    protected void doGet(HttpServletRequest request, HttpServletResponse response)  
        throws ServletException, IOException {...}  
  
    // Este metodo lo invoca el contenedor web cuando entra una petición por el método POST  
    @Override  
    protected void doPost(HttpServletRequest request, HttpServletResponse response)  
        throws ServletException, IOException {...}
```

Gráfico 25. Ejemplo de un HttpServlet

Una vez que tenemos el servlet creado el siguiente paso es mapearlo para poder invocarlo a través de la url.

Actualmente tenemos dos formas de hacerlo:

- A través del descriptor de despliegue web.xml; esta es la forma tradicional como se ha venido haciendo siempre.
- A través de anotaciones; a partir de JEE 6 se contempla esta opción.

## MAPEO DEL SERVLET EN EL DESCRIPTOR WEB.XML

Para mapear el servlet necesitamos de dos secciones:

- La declaración del servlet; que consiste en asociar un alias (<servlet-name>) al nombre de la clase del servlet (servlet-class).
- El mapeo del servlet; consiste en asociar el alias (<servlet-name>) con el patron url (<url-pattern>) que se utilizara para enviar la petición en la url.

```
<!-- Declaración del servlet -->
<servlet>
    <servlet-name>Servlet</servlet-name>
    <servlet-class>app.web.ServletTienda</servlet-class>
</servlet>

<!-- Mapeo del servlet -->
<servlet-mapping>
    <servlet-name>Servlet</servlet-name>
    <url-pattern>/controlador</url-pattern>
</servlet-mapping>
```

Gráfico 26. Mapeo de un servlet en web.xml

Una vez que ya tenemos el servlet mapeado podremos invocarlo a través de una petición.

Veamos un ejemplo de petición mediante un link:

```
<a href="controlador?opcion=1">Consultar todos los productos</a><br>
```

Gráfico 27. Invocar al servlet mediante un link

Como podemos comprobar no utilizamos su alias y tampoco el nombre completo de su clase. El servlet se invoca a través de su patrón url.

Otro ejemplo es como invocarlo a través del acción de un formulario:

```
<form action="controlador" method="post">
    Introduce Id del producto:
    <input type="text" name="codigo" />
    <input type="hidden" name="opcion" value="2" />
    <input type="submit" value="ENVIAR" />
</form>
```

Gráfico 28. Invocar al servlet mediante el atributo action del formulario.

## MAPEO DEL SERVLET A TRAVÉS DE ANOTACIONES

En la siguiente imagen vemos otra forma de mapear servlets.

```
@WebServlet(name="Servlet", urlPatterns={"/controlador"})
public class ServletTienda extends HttpServlet {
```

Gráfico 29. Mapeo de servlet mediante anotaciones.

## ATRIBUTOS

A veces será necesario pasar cierto objeto de un servlet a una página JSP. Existe una forma de hacerlo, que es encapsular el objeto que deseamos pasar en el objeto request.

Veamos el siguiente ejemplo donde guardamos la lista de los productos como atributo de la petición.

```
// Guardamos el producto encontrado como atributo de la petición
request.setAttribute("encontrado", encontrado);

// Elegir la vista para mostrar
RequestDispatcher rd = request.getRequestDispatcher("/mostrarProducto.jsp");
// Redirigir hacia esa vista
rd.forward(request, response);
```

Gráfico 30. Almacenar un atributo en la petición.

El recurso destino, en este caso una jsp, utiliza los objetos request y response como si la petición hubiese sido realizada directamente sobre ella.

Nosotros podemos incluir dentro del request tantos objetos como deseemos que la aplicación destino sea capaz de recoger.

Las instrucciones clave para realizar este proceso están recogidas como métodos propios del request, estos métodos son:

- request.setAttribute(String, Object), para guardar un objeto (Object) en el request, con un identificador, clave (String).
- request.getAttribute(String), para recuperar el objeto guardado con un identificador (String).

En el siguiente fragmento vemos como recuperamos el atributo en la página mostrarTodos.jsp.



```
<% List<Producto> lista = (List)request.getAttribute("listaProductos"); %>
```

Gráfico 31. Recuperar el atributo en la pagina jsp.

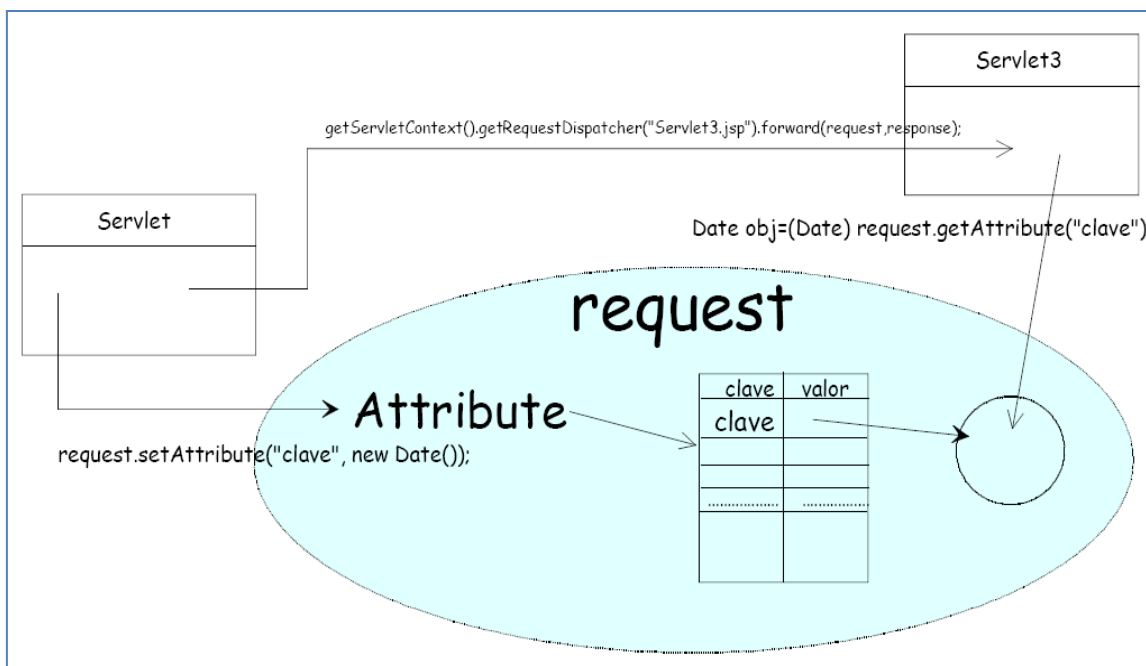


Gráfico 32. Paso de atributos entre servlets

## LANZAMIENTO DE SOLICITUDES

El lanzamiento de solicitudes permite a un servlet lanzar una solicitud a otro servlet, página JSP o página HTML, que será la responsable de cualquier procesamiento posterior y de generar la respuesta.

El API Java Servlet tiene una interfaz especial llamada `javax.servlet.RequestDispatcher` para este propósito.

Esta interfaz tiene dos métodos que le permiten delegar el procesamiento de solicitudes-respuestas en otro recurso, después de que el servlet llamante haya finalizado cualquier procesamiento preliminar: El método `forward()` y el método `include()`.

- El método **`forward()`**; permite reenviar la solicitud a otro servlet o página JSP, o a un archivo HTML en el servidor; este recurso acepta la responsabilidad de producir la respuesta.

```
public void forward(ServletRequest request, ServletResponse response)
    throws ServletException, java.io.IOException
```

Difiere de `HttpServletResponse.sendRedirect()`, que redirige la petición con la ayuda del navegador

- El método **include()**; permite incluir el contenido producido por otro recurso en la respuesta del servlet llamante.

```
public void include(ServletRequest request, ServletResponse response)
    throws ServletException, java.io.IOException
```



### RECUERDA QUE...

- Un servlet es un componente que administra el contenedor web.
- Podemos declarar y mapear el servlet a través de código xml o usando anotaciones.
- Los parámetros de la petición se reciben en el servidor.
- Los atributos se crean en el servidor para poder enviar datos a otros componentes.
- Desde el servlet podemos lanzar solicitudes a otros servlets o jsp.

## 5. JSP

La tecnología JSP está basada en el lenguaje de programación Java y encaminada a facilitar el desarrollo de sitios web.

Mediante el uso de páginas JSP podemos incorporar contenido dinámico en sitios web mediante código Java embebido a través de etiquetas especiales `< % % >`.

Las páginas JSP son archivos de texto con extensión `.jsp` que contienen etiquetas HTML, junto con código Java embebido, que permite el acceso de la página a datos desde ese código Java ejecutado en el servidor.

Cuando se solicita una página JSP, la parte HTML se procesa en el cliente, sin embargo, el código Java se ejecuta en el momento de recibir la petición y el contenido dinámico generado por ese código se inserta en la página antes de devolverla al usuario.

Esto proporciona una separación entre la parte de presentación HTML de la página y la parte de lógica de programación incluida en el código Java.

### CICLO DE VIDA DE LAS PÁGINAS JSP

Todo el tratamiento de la petición HTTP que se hace en el servidor web hasta que se devuelve la respuesta al cliente se resume en 4 pasos:

- El motor JSP analiza la página solicitada y crea un fichero `.java` correspondiente al servlet.
- El servlet generado se compila para obtener el archivo `.class`, que pasa al control del motor servlet, que lo ejecuta del mismo modo que si se tratase de cualquier otro servlet.
- El motor servlet carga la clase del servlet generado para ejecutarlo.
- El servlet se ejecuta y devuelve su respuesta al solicitante.



Gráfico 33. Transformación de una jsp en un servlet.

El servlet se genera con los siguientes métodos:

- `jspInit()`; Inicializa el servlet generado y sólo se llama en la primera petición
- `jspService(request, response)`; Se invoca en cada petición, incluso en la primera. Es quien se encarga de manejar las peticiones.

- `jspDestroy()`; Invocada por el motor para eliminar el servlet.

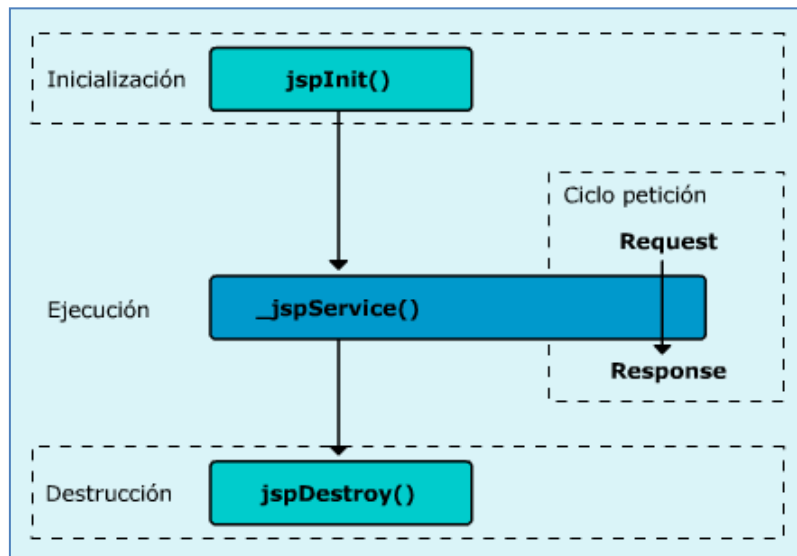


Gráfico 34. Métodos del servlet generado

## INCLUIR CODIGO JAVA EN UNA JSP

Podemos "incrustar" código Java de distintos tipos (declaraciones de variables y/o métodos, expresiones, sentencias) para que lo ejecute el contenedor JSP.

Hay varias formas de insertar código Java en una página JSP:

- Declaraciones
- Expresiones
- Scriptlets
- Comentarios

## DECLARACIONES

Las declaraciones son elementos que se utilizan para declarar una variable o un método que se insertarán dentro del cuerpo del servlet generado.

No generarán ninguna salida, por lo que pueden utilizarse conjuntamente con otros elementos de las páginas JSP. Su sintaxis es la siguiente:

```
<%! DeclaraciónClaseJava %>
```

Gráfico 22. Sintaxis de declaraciones

```
<%! public static final String DEFAULT_NAME = "World"; %>

<%! public String getName(HttpServletRequest request) {
    return request.getParameter("name");
}
%>

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

Gráfico 35. Ejemplos de declaraciones

Una declaración de una variable o método solamente es válida para la página JSP en la que se ha declarado. Las variables conservarán su valor entre sucesivas llamadas a la página, ya que son variables miembro del servlet.

## EXPRESIONES

Las expresiones son un mecanismo que evita tener que escribir el código completo de la sentencia `out.println()`. Su sintaxis:

```
<%= ExpresiónJava %>
```

Gráfico 36. Sintaxis de las expresiones

```
La fecha y hora actual es: <%= new java.util.Date() %>
```

Gráfico 37. Ejemplo de expresión

Las expresiones que se incluyen en estos elementos son evaluados, convertidos a un objeto de tipo `String` e incluidos como parte del código HTML en el lugar donde aparece en la página JSP.

Las expresiones están orientadas a la generación de datos.

## SCRIPTLETS

Un scriptlet es un bloque de código Java insertado en la página y ejecutado durante el procesamiento de la respuesta (en el motor JSP). El resultado del código Java no es necesario enviarlo a la salida. Su sintaxis es la siguiente:

```
<% CódigoJava %>
```

Gráfico 38. Sintaxis de los scriptlets

Los scriptlets no están limitados a una sola línea, pueden ocupar varias.

```
<% if ( i > 10 ) { %>
    Soy un número alto.
<% } else { %>
    Soy un número bajo.
<% } %>
```

Gráfico 39. Ejemplo de scriptlet

Un uso común de los scriptlets es hacer que ciertas partes de código HTML aparezcan o no en función de una condición.

## COMENTARIOS

La documentación es importante para cualquier aplicación. Las páginas de JSP admiten tres tipos de comentarios:

- Comentarios HTML

Los comentarios HTML se consideran texto de plantilla de HTML. Estos comentarios se envían en el flujo de respuesta HTTP. Por ejemplo:

```
<!-- Esto es un comentario HTML. Aparecerá en la respuesta. -->
```

Gráfico 40. Ejemplo comentario HTML

- Comentarios de página de JSP

Los comentarios de página de JSP sólo se ven en el archivo de la página de JSP. Estos comentarios no se incluyen en el código fuente del servlet durante la fase de conversión ni aparecen en la respuesta HTTP. Por ejemplo:

```
<%-- Esto es un comentario de JSP. Sólo se verá en el código JSP.
    No aparecerá en el código del servlet ni en la respuesta.
--%>
```

Gráfico 41. Ejemplo comentario JSP

- Comentarios Java

Se pueden incrustar comentarios Java con etiquetas de scriptlet y de declaración. Estos comentarios se incluyen en el código fuente del servlet durante la fase de conversión, pero no aparecen en la respuesta HTTP. Por ejemplo:

```
<%  
    /* Esto es un comentario Java. Aparecerá en el código de servlet.  
       No aparecerá en la respuesta. */  
%>
```

Gráfico 42. Ejemplo comentario Java

## DIRECTIVAS

Las directivas proporcionan información global de la página utilizada en la fase de traducción. Se utilizan para definir y manipular una serie de atributos dependientes de la página que afectan a todo el JSP y, por lo tanto, influyen en la estructura que tendrá el servlet generado. Para indicar la presencia de una directiva se utiliza el signo de la arroba.

Su sintaxis es:

```
<%@ NombreDirectiva [attr="valor"] * %>
```

Gráfico 43. Sintaxis de las directivas

Hay tres tipos de directivas:

- page
- include
- taglib

## DIRECTIVA PAGE

La directiva page se emplea para especificar atributos para toda la página JSP en su conjunto. Su sintaxis es:

```
<%@ page [atributo="valor" atributo=valor ...]%>
```

Aunque puede haber varias directivas page, sólo se puede declarar un determinado atributo una vez por página. Esto se aplica a todos los atributos excepto import. Una directiva page puede situarse en cualquier lugar del archivo JSP. Es conveniente que la directiva page sea la primera sentencia del archivo JSP.

La directiva page define diversas propiedades dependientes de la página y las comunica al contenedor web durante la conversión.

- El atributo **language** especifica el lenguaje de secuencia de comandos que se debe emplear en la página. El único valor actualmente definido es java, que es el predeterminado.
- El atributo **extends** determina el nombre de clase (completo) de la superclase de la clase de servlet que se genera con la página de JSP.
- El atributo **buffer** define el tamaño del búfer empleado en el flujo de salida (un objeto JspWriter). Su valor es none o Nkb. El tamaño de búfer predeterminado es kilobytes (Kbytes) o más. Por ejemplo: buffer="8kb" o buffer="none".
- El atributo **autoFlush** determina si la salida del búfer debe borrarse automáticamente cuando se llena el búfer o si se lanza una excepción. Su valor es true (borrado automático) o false (lanzar una excepción). El valor predeterminado es true.
- El atributo **session** determina si la página de JSP interviene en una sesión HTTP. Su valor es true (predeterminado) o false.
- El atributo **import** define del conjunto de clases y paquetes que deben importarse en la definición de clase de servlet. El valor de este atributo es una lista delimitada por comas de nombres de clase o paquetes completos. Por ejemplo: import="java.sql.Date,java.util.\*,java.text.\*"
- El atributo **isThreadSafe** permite declarar si la página de JSP está protegida ante subprocesos. Si el valor se define en false, este atributo instruye al analizador JSP para que escriba el código de servlet de manera que sólo se procese simultáneamente una solicitud HTTP. El valor predeterminado es true.
- El atributo **info** define una cadena informativa sobre la página de JSP.
- El atributo **contentType** define un tipo MIME del flujo de salida. El valor predeterminado es text/html.
- El atributo **pageEncoding** determina la codificación de caracteres del flujo de salida. El valor predeterminado es ISO-8859-1. Otras codificaciones de caracteres permiten incluir juegos de caracteres no latinos, como kanji o cirílico.
- El atributo **isELIgnored** especifica si se omiten los elementos de lenguaje de expresiones de la página. Su valor es true o false (predeterminado). Si se define en true, no se evalúa el lenguaje de expresiones de la página.
- El atributo **isErrorPage** establece que la página de JSP se ha diseñado para ser el destino del atributo errorPage de otra página de JSP. Su valor es true o false (predeterminado). Todas las páginas de JSP erróneas tienen acceso automático a la variable implícita exception.
- El atributo **errorPage** indica otra página de JSP que manejará todas las excepciones de tiempo de ejecución lanzadas por esta página de JSP. El valor



es una URL relativa a la jerarquía web actual o a la raíz de contexto. Por ejemplo, `errorPage="error.jsp"` (es relativa a la jerarquía actual) o `errorPage="/error/formErrors.jsp"` (es relativa a la raíz de contexto de la aplicación web).

## DIRECTIVA INCLUDE

La directiva include permite incluir código JSP en una página en tiempo de compilación. El código puede ser una página HTML, un archivo java, un fichero de texto u otra página JSP. La página final que va a procesar el motor JSP es la formada por la página base más el contenido del fichero que se haya incluido. Sintaxis:

```
<%@ include file="nombre del fichero" %>
```

Una vez incluido el fichero, si se modifica el fichero no se verá reflejado en el servlet.

Se suele utilizar para incluir cabeceras y pies de página estándar o cualquier otro texto en formato común en las páginas JSP.

## DIRECTIVA TAGLIB

La directiva taglib permite extender los marcadores de JSP con etiquetas o marcas generadas por el propio usuario (etiquetas personalizadas). Se hace referencia a una biblioteca de etiquetas que contiene código Java compilado definiendo las etiquetas que van a ser usadas y que han sido usadas y que han sido definidas por el usuario. Sintaxis:

```
<%@ taglib uri="taglibraryURI" prefix="tagPrefix"%>
```

La variable uri hace referencia a la dirección que identifica a la biblioteca de etiquetas. Mientras que prefix define el prefijo que se coloca a cada una de las etiquetas de la biblioteca, que se utiliza para distinguir las etiquetas personalizadas.

## ACCIONES

Normalmente sirven para alterar el flujo normal de ejecución de la página (p.ej. redirecciones), aunque tienen usos variados. Las acciones proporcionan al motor JSP información sobre lo que debe hacer a la hora de procesar la página.

Las acciones son marcas estándar, con formato XML, que afectan al comportamiento en tiempo de ejecución del JSP y la respuesta se devuelve al cliente. Hay acciones predefinidas y también se pueden incorporar nuevas acciones personalizadas (incluidas a través de la directiva taglib).

Sintaxis: <nombre\_etiqueta [atr="valor" atr="valor"...]>

...

</nombre\_etiqueta>

<nombre\_etiqueta [atr="valor" atr="valor"...] />

Tenemos siete tipos de acciones diferentes:

- useBean
- setProperty
- getProperty
- include
- forward
- param
- plugin

## ETIQUETA USEBEAN

Si quiere interactuar con un componente JavaBeans utilizando las etiquetas estándar en una página de JSP, primero debe declarar el bean. Para ello se utiliza la etiqueta estándar useBean.

La sintaxis de la etiqueta useBean es:

```
<jsp:useBean id="NombreBean"
             scope="page | request | session | application"
             class="NombreClase" />
```

Gráfico 44. Sintaxis etiqueta <jsp:useBean>

El atributo id especifica el nombre de atributo del bean. El atributo scope indica dónde se almacena el bean. Si no se especifica, scope adopta el valor predeterminado page. El atributo class especifica el nombre de clase completo.

Para una declaración useBean de lo siguiente:

```
<jsp:useBean id="miBean" scope="request"
             class="sl314.beans.CustomerBean" />
```

Gráfico 45. Ejemplo etiqueta <jsp:useBean>

el código Java equivalente podría ser así:

```
CustomerBean miBean =
    (CustomerBean) request.getAttribute("miBean");
if( miBean == null ) {
    miBean = new CustomerBean();
    request.setAttribute("miBean", miBean);
}
```

Gráfico 46. Código java equivalente a <jsp:useBean>

## ETIQUETA SETPROPERTY

La etiqueta setProperty sirve para almacenar datos en la instancia de JavaBeans. La sintaxis de la etiqueta setProperty es:

```
<jsp:setProperty name="NombreBean" property_expression />
```

Gráfico 46. Sintaxis etiqueta <jsp:setProperty>

El atributo name especifica el nombre de la instancia de JavaBeans. Debe coincidir con el atributo id utilizado en la etiqueta useBean. property\_expression puede ser así:

- property="\*"
  - property="NombrePropiedad"
  - property="NombrePropiedad" param="NombreParámetro"
  - property="NombrePropiedad" value="ValorPropiedad"

El atributo **property** especifica la propiedad dentro del bean que se definirá. Por ejemplo, para definir la propiedad email en el ben del cliente, puede utilizar lo siguiente:

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

Esta acción recupera el valor del parámetro de solicitud email y emplea este valor en el método set del bean. El código Java equivalente sería como éste:

```
cust.setEmail(request.getParameter("email"));
```

El atributo **param** se puede suministrar si el nombre del parámetro de solicitud es distinto al nombre del parámetro de bean.

Por ejemplo, si el campo del formulario para la dirección de correo electrónico fuera emailAddress, podría definir la propiedad named email del bean mediante:

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

El código Java equivalente sería como éste:

```
cust.setEmail(request.getParameter("emailAddress"));
```

El atributo **value** puede utilizarse para suministrar el valor que debe emplearse en el método set. Por ejemplo, el valor se podría codificar rígidamente en la página de JSP:

```
<jsp:setProperty name="cust" property="email" value="joe@host.com" />
```

Los valores de los atributos se pueden especificar con expresiones, que se evalúan en tiempo de ejecución. Por ejemplo:

```
<jsp:setProperty name="cust" property="email" value='<%= someMethodToGetEmail()  
%>' />
```

El carácter de **asterisco** (\*) sirve para especificar todas las propiedades del bean.

## ETIQUETA GETPROPERTY

La etiqueta getProperty sirve para recuperar una propiedad de una instancia de JavaBeans y mostrarla en el flujo de salida. La sintaxis de la etiqueta getProperty es:

```
<jsp:getProperty name="NombreBean"  
property="NombrePropiedad" />
```

Gráfico 47. Sintaxis etiqueta <jsp:setProperty>

El atributo name especifica el nombre de la instancia de JavaBeans, mientras que el atributo property determina la propiedad utilizada con el método get.

Para un uso de getProperty como el siguiente:

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

el equivalente en lenguaje Java sería así:

```
out.print(cust.getEmail());
```

La etiqueta getProperty brinda un mecanismo práctico para mostrar las propiedades de una instancia de JavaBeans evitando el código de scriptlet.

## ETIQUETA INCLUDE

Esta acción permite insertar un archivo estático o dinámico en la página que está siendo generada por el motor JSP. Su sintaxis es:

```
<jsp:include page="url" flush="true">
```

```
<jsp:param ... />
```

```
<jsp:param ... />
```

```
</jsp:include>
```

Es importante distinguir entre directiva include y acción include:

- Directiva `<%@ include file="Nombre fichero" />` se añade el código al servlet que se genera para la página en tiempo de compilación y se incluye el contenido existente en el momento inicial.
- Acción `<jsp:include>` no se añade código al servlet, sino que se invoca al objeto en tiempo de ejecución y se ejecuta el contenido existente en el momento de la petición.

## ETIQUETA FORWARD

Esta etiqueta permite que la petición sea redirigida a otra página JSP, a otro servlet o a otro recurso estático para que lo procese.

Cuando el motor JSP encuentra esta etiqueta, la petición se pasa directamente al otro recurso, sin procesar el resto de la página que contenía la acción forward.

Esta acción es muy útil cuando se quiere separar la aplicación en diferentes vistas, dependiendo de la petición interceptada.

Su sintaxis es:

```
<jsp: forward page="url" >
```

```
<jsp:param ... />
```

```
<jsp:param ... />
```

```
<jsp:param ... />
```

```
</jsp: forward >
```

## ETIQUETA PARAM

Este elemento es utilizado dentro de otras acciones para proporcionar información adicional de la forma clave/valor.

Sirve para pasar parámetros a un objeto. Asocia un valor a un nombre y pasa la asociación a otro recurso invocado con `<jsp:include>`, `<jsp:forward>` o `<jsp:plugin>`

Su sintaxis es:

```
<jsp:param name="nombreParametro"
```

```
value="{valorParametro | <%= expresion %>}" />
```

Los dos atributos son obligatorios. El primero corresponde al nombre del parámetro y el segundo con el valor que se le asigna, que puede ser una expresión JSP que se evalúa en el momento de realizar la petición de la página.

## ETIQUETA PLUGIN

Esta acción genera código HTML (las etiquetas object o embed) específico al navegador al que va dirigida la página JSP, que provocará la descarga del software plug-in (en caso de que sea necesario) correspondiente al navegador en el cual se intenta ejecutar un applet o JavaBean.

Esta acción permite que la página JSP incluya un bean o un applet en la página cliente.

Su sintaxis es:

```
<jsp:plugin  
    type="bean|applet"  
    code="nombre de la clase"  
    codebase="directorio donde está el .class"  
    {align="bottom|top|middle|left|right"}  
    {archive="fichero jar donde se ha almacenado la clase"}  
    {height="altura en pixeles"}  
    {hspace="espacio horizontal en pixeles"}  
    {jreversion="numeroversionJRE"}  
    {name="nombre de componente"}  
    {vspace="espacio vertical en pixeles"}  
    {width="anchura en pixeles"}  
    {nspluginurl="URL del plugin Netscape"}  
    {iepluginurl="URL del plugin Explorer"} >  
    <jsp:params>  
        <jsp:param ... />  
    </jsp:params>  
    <jsp:fallback>Problema con el plugin </jsp:fallback>
```

```
</jsp:plugin>
```

## OBJETOS IMPLÍCITOS

El motor de JSP proporciona acceso a las siguientes variables en etiquetas de scriptlet y de expresión. Estas variables representan objetos de uso habitual con los servlets que los desarrolladores de páginas de JSP pueden necesitar. Por ejemplo, puede recuperar datos de parámetros de un formulario HTML con la variable request, que representa al objeto HttpServletRequest.

En la siguiente tabla se recogen los objetos implícitos de JSP.

| Nombre de la variable | Descripción   |
|-----------------------|---|
| request               | El objeto HttpServletRequest asociado a la solicitud.   |
| response              | El objeto HttpServletResponse asociado a la respuesta que se devuelve al navegador.   |
| out                   | El objeto JspWriter asociado al flujo de salida de la respuesta.  |
| session               | El objeto HttpSession asociado a la sesión para el usuario específico de la solicitud. Esta variable sólo es significativa si la página de JSP interviene en una sesión HTTP. |
| application           | El objeto ServletContext para la aplicación web.  |
| config                | El objeto ServletConfig asociado al servlet para esta página de JSP.  |
| pageContext           | El objeto pageContext que encapsula el entorno de una sola solicitud para esta página de JSP.   |
| page                  | La variable page equivale a la variable this en el lenguaje Java.   |
| exception             | El objeto Throwable generado por otra página de JSP. Esta variable sólo está disponible en una página de error de JSP.  |

Gráfico 48. Objetos implícitos de JSP

## LENGUAJE DE EXPRESIONES EL

El lenguaje de expresiones se originó en JSTL 1.0 y está incorporado en la especificación JSP 2.0. Su finalidad es ayudar a crear páginas de JSP sin secuencias de comandos.

### DESCRIPCIÓN DE LA SINTAXIS

La sintaxis del lenguaje de expresiones en una página de JSP es:

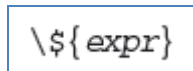


```
${expr}
```

Gráfico 49. Sintaxis EL

En esta sintaxis, *expr* indica una expresión válida del lenguaje de expresiones. Esta expresión se puede mezclar con texto estático y combinar con otras expresiones para formar expresiones más amplias.

Para aplicar escape a los caracteres EL en una página de JSP, utilice la notación de barra invertida (\). Con escape no se evalúa la expresión EL.



```
\${expr}
```

Gráfico 50. Escapar expresiones

Las expresiones EL pueden utilizarse en las páginas de JSP de dos maneras:

- Como valores de atributo en acciones estándar y personalizadas. Cuando una expresión EL se utiliza como valor de un atributo, se evalúa y su valor sirve como valor del atributo. Por ejemplo: `<jsp:include page="${location}">`
- Dentro de texto de plantilla, como HTML. Cuando una expresión EL se utiliza dentro de texto de plantilla, se evalúa y su valor se incorpora al flujo de salida. Por ejemplo: `<h1>Welcome ${name}</h1>`

### ACCESO A BEANS MEDIANTE LENGUAJE DE EXPRESIONES

Con el lenguaje de expresiones es fácil acceder a los beans dentro del espacio de nombres disponible para la página de JSP. Para acceder a un atributo del bean, basta con usar como sigue la notación de punto:



```
${bean.attribute}
```

Gráfico 51. Acceder a las propiedades de un bean con EL

Cuando se especifica un nombre de bean sin ningún ámbito asociado, es preciso ubicar el bean para que el motor de JSP lo busque en los ámbitos. El orden de búsqueda en los ámbitos es: page, request, session y application (página, solicitud, sesión y aplicación).

Las expresiones EL pueden manejar objetos nulos. En un scriptlet, es preciso comprobar un objeto antes de ejecutar un método para evitar una excepción `NullPointerException`. El mecanismo del lenguaje de expresiones maneja el valor nulo mostrando una cadena vacía.

Para indicar el ámbito que contiene el bean, el nombre de atributo se precede con un objeto de ámbito implícito. En el ejemplo siguiente se recupera la propiedad `firstName` del bean `cust` ubicado en el ámbito de sesión:

```
${sessionScope.cust.firstName}
```

Gráfico 52. Ejemplo de acceso a las propiedades del bean

### OBJETOS IMPLÍCITOS DISPONIBLES CON EL LENGUAJE DE EXPRESIONES

| Objeto implícito              | Descripción   |
|-------------------------------|---|
| <code>pageContext</code>      | El objeto <code>PageContext</code> .  |
| <code>pageScope</code>        | Una asignación que contiene los atributos de ámbito de página y sus valores.                        |
| <code>requestScope</code>     | Una asignación que contiene los atributos de ámbito de solicitud y sus valores.                     |
| <code>sessionScope</code>     | Una asignación que contiene los atributos de ámbito de sesión y sus valores.                        |
| <code>applicationScope</code> | Una asignación que contiene los atributos de ámbito de aplicación y sus valores.                    |
| <code>param</code>            | Una asignación que contiene los parámetros de solicitud y valores de cadenas individuales.          |
| <code>paramValues</code>      | Una asignación que contiene los parámetros de solicitud y sus correspondientes matrices de cadenas. |
| <code>header</code>           | Una asignación que contiene los nombres de encabezado y valores de cadenas individuales.            |
| <code>headerValues</code>     | Una asignación que contiene los nombres de encabezado y sus correspondientes matrices de cadenas.   |
| <code>cookie</code>           | Una asignación que contiene los nombres de cookies y sus valores.                                   |

Gráfico 53. Objetos implícitos de EL

Por ejemplo, para acceder a un parámetro de solicitud denominado `username` puede emplearse esta expresión EL: `${param.username}`

Si un bean devuelve una matriz, se puede acceder a un elemento de la matriz mediante su índice: `${paramValues.fruit[2]}`

## OPERADORES ARITMÉTICOS

| Operación aritmética | Operador |
|----------------------|----------|
| Suma                 | +        |
| Resta                | -        |
| Multiplicación       | *        |
| División             | / y div  |
| Resto                | % y mod  |

Gráfico 42. Operadores aritméticos de EL

| Expresión EL             | Resultado |
|--------------------------|-----------|
| $\{3 \text{ div } 4\}$   | 0.75      |
| $\{1 + 2 * 4\}$          | 9         |
| $\{(1 + 2) * 4\}$        | 12        |
| $\{32 \text{ mod } 10\}$ | 2         |

Gráfico 54. Ejemplos de operaciones aritméticas de EL

## OPERADORES DE COMPARACIÓN

| Comparación       | Operador |
|-------------------|----------|
| Igual que         | == y eq  |
| Distinto de       | != y ne  |
| Menor que         | < y lt   |
| Mayor que         | > y gt   |
| Menor o igual que | <= y le  |
| Mayor o igual que | >= y ge  |

Gráfico 55. Operadores de comparación de EL

## OPERADORES LÓGICOS

| Operación lógica | Operador |
|------------------|----------|
| and              | && y and |
| or               | y or     |
| not              | ! y not  |

Gráfico 56. Operadores lógicos de EL

## ETIQUETAS JSTL

En esta sección se resumen las etiquetas disponibles en JSTL por categoría funcional. La siguiente tabla contiene las cinco categorías funcionales de etiquetas en JSTL, sus valores URI (que se usan en las directivas taglib de JSP) y el prefijo típico que se antepone en cada caso.

| Área funcional                                | URI   | Prefijo |
|---|---|---------|
| Acciones básicas                              | <a href="http://java.sun.com/jsp/jstl/core">http://java.sun.com/jsp/jstl/core</a>           | c       |
| Acciones de procesamiento de 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 relacional | <a href="http://java.sun.com/jsp/jstl/sql">http://java.sun.com/jsp/jstl/sql</a>             | sql     |
| Acciones de función                           | <a href="http://java.sun.com/jsp/jstl/functions">http://java.sun.com/jsp/jstl/functions</a> | fn      |

Gráfico 57. Librerías de etiquetas JSTL

## ETIQUETAS DE LA BIBLIOTECA CORE

Las etiquetas del área funcional básica incluyen acciones para enviar al flujo de respuesta, realizar operaciones condicionales y efectuar iteraciones.

| Tag                      | Finalidad  |
|--------------------------|--|
| <code>c:out</code>       | Evalúa una expresión y envía el resultado al objeto <code>JspWriter</code> actual.   |
| <code>c:set</code>       | Define el valor de una variable o propiedad de ámbito.   |
| <code>c:remove</code>    | Elimina una variable de ámbito.  |
| <code>c:catch</code>     | Captura un objeto <code>java.lang.Throwable</code> que se produce en el cuerpo de la etiqueta.                                 |
| <code>c:if</code>        | Evalúa el cuerpo de la etiqueta si la expresión especificada por el atributo de prueba es <code>true</code> .                  |
| <code>c:choose</code>    | Proporciona una condición autoexcluyente.  |
| <code>c:when</code>      | Proporciona una alternativa dentro de un elemento <code>c:choose</code> .  |
| <code>c:otherwise</code> | Proporciona la última alternativa dentro de un elemento <code>c:choose</code> .  |
| <code>c:forEach</code>   | Itera una colección de objetos o un número fijo de ciclos.   |
| <code>c:forTokens</code> | Divide una cadena en tokens y los itera.   |
| <code>c:import</code>    | Importa el contenido de un recurso URL.  |
| <code>c:url</code>       | Reescribe URL relativas.   |
| <code>c:redirect</code>  | Envía un redireccionamiento HTTP al cliente.   |
| <code>c:param</code>     | Agrega parámetros a la solicitud (se utiliza dentro de <code>c:import</code> , <code>c:url</code> y <code>c:redirect</code> ). |

Gráfico 58. Etiquetas de la librería Core

## ETIQUETAS DE LA BIBLIOTECA XML

JSTL incluye etiquetas para procesar documentos XML. A continuación se describen las etiquetas disponibles en esta biblioteca.

| Tag                      | Finalidad  |
|--------------------------|--|
| <code>x:parse</code>     | Analiza un documento XML.  |
| <code>x:out</code>       | Evalúa la expresión XPath y envía el resultado al objeto <code>JspWriter</code> actual.                |
| <code>x:set</code>       | Evalúa la expresión XPath y almacena el resultado en una variable de ámbito.                           |
| <code>x:if</code>        | Evalúa el cuerpo de la etiqueta si la expresión XPath es <code>true</code> .                           |
| <code>x:choose</code>    | Proporciona una condición autoexcluyente.  |
| <code>x:when</code>      | Proporciona una alternativa dentro de un elemento <code>x:choose</code> .                              |
| <code>x:otherwise</code> | Proporciona la alternativa final dentro de un elemento <code>x:choose</code> .                         |
| <code>x:forEach</code>   | Evalúa la expresión XPath y repite el cuerpo de la etiqueta.   |
| <code>x:transform</code> | Aplica una hoja de estilo XSLT a un documento XML.   |
| <code>x:param</code>     | Proporciona parámetros de transformación (se utiliza dentro de un elemento <code>x:transform</code> ). |

Gráfico 59. Etiquetas de la librería xml

## ETIQUETAS DE LA BIBLIOTECA FORMAT

JSTL incluye etiquetas para internacionalización y asignación de formato.

| Tag                              | Finalidad   |
|----------------------------------|---|
| <code>fmt:setLocale</code>       | Almacene la configuración regional especificada en la variable de configuración regional.   |
| <code>fmt:bundle</code>          | Crea un contexto de localización i18n que se utiliza en el cuerpo de la etiqueta.   |
| <code>fmt:setBundle</code>       | Crea un contexto de localización i18n y lo almacena en la variable de ámbito o en la variable de configuración del contexto de localización.            |
| <code>fmt:message</code>         | Busca el mensaje localizado en el lote de recursos.   |
| <code>fmt:param</code>           | Suministra un parámetro para sustitución dentro de un elemento <code>fmt:message</code> .   |
| <code>fmt:requestEncoding</code> | Configura la codificación de caracteres de la solicitud.  |
| <code>fmt:timeZone</code>        | Especifica la zona horaria en la que se procesará la información horaria del cuerpo de la etiqueta.   |
| <code>fmt:setTimeZone</code>     | Almacena la zona horaria especificada en una variable de ámbito o en la variable de configuración de la zona horaria.                                   |
| <code>fmt:formatNumber</code>    | Asigna a un valor numérico un formato personalizado o de configuración regional, como una cifra, una divisa o un porcentaje.                            |
| <code>fmt:parseNumber</code>     | Analiza la representación en una cadena de cifras, divisas y porcentajes a los que se ha asignado un formato personalizado o de configuración regional. |
| <code>fmt:formatDate</code>      | Permite asignar formatos personalizados o de configuración regional de fecha y hora.  |
| <code>fmt:parseDate</code>       | Analiza la representación en una cadena de fechas y horas a las que se ha asignado un formato personalizado o de configuración regional.                |

Gráfico 60. Etiquetas de la librería format

## ETIQUETAS DE LA BIBLIOTECA SQL

JSTL incluye etiquetas para acceso a base de datos relacional.

| Tag                            | Finalidad  |
|--------------------------------|--|
| <code>sql:query</code>         | Consulta la base de datos y almacena los resultados en una variable de ámbito.   |
| <code>sql:update</code>        | Ejecuta una sentencia INSERT, DELETE, UPDATE o SQL DDL y almacena el resultado en una variable de ámbito.  |
| <code>sql:transaction</code>   | Establece un contexto de transacción para los elementos <code>sql:query</code> y <code>sql:update</code> .                                       |
| <code>sql:setDataSource</code> | Exporta una fuente de datos como una variable de ámbito o como variable de configuración de la fuente de datos.                                  |
| <code>sql:param</code>         | Configura los valores de los marcadores de parámetros (se utiliza con los elementos <code>sql:query</code> y <code>sql:update</code> ).          |
| <code>sql:dateParam</code>     | Configura los valores de los marcadores de parámetros de fecha (se utiliza con los elementos <code>sql:query</code> y <code>sql:update</code> ). |

Gráfico 61. Etiquetas de la librería sql

## ETIQUETAS DE LA BIBLIOTECA FUNCTIONS

JSTL incluye etiquetas para funciones, muchas de `java.lang.String`.



| Tag                                | Finalidad  |
|------------------------------------|--|
| <code>fn:contains</code>           | Realiza una prueba de una subcadena especificada con distinción entre mayúsculas y minúsculas; devuelve <code>true</code> o <code>false</code> . |
| <code>fn:containsIgnoreCase</code> | Realiza una prueba de una subcadena especificada sin distinción entre mayúsculas y minúsculas; devuelve <code>true</code> o <code>false</code> . |
| <code>fn:endsWith</code>           | Comprueba si una cadena termina en un sufijo especificado y devuelve <code>true</code> o <code>false</code> .                                    |
| <code>fn:escapeXml</code>          | Aplica escape a caracteres que se interpretarían como marcado XML.   |
| <code>fn:indexOf</code>            | Devuelve la posición de la primera vez que aparece una subcadena especificada.   |
| <code>fn:join</code>               | Une los elementos de una matriz en una cadena.   |
| <code>fn:length</code>             | Devuelve el número de elementos de una colección o el número de caracteres de una cadena.  |
| <code>fn:replace</code>            | Devuelve una cadena después de sustituir una subcadena por otra siempre que aparece.   |
| <code>fn:split</code>              | Divide una cadena en una matriz de subcadenas basándose en un delimitador.   |
| <code>fn:startsWith</code>         | Comprueba si una cadena empieza con un prefijo especificado y devuelve <code>true</code> o <code>false</code> .                                  |
| <code>fn:substring</code>          | Devuelve un subconjunto de una cadena, delimitado por puntos de inicio y fin.  |
| <code>fn:substringAfter</code>     | Devuelve la subcadena que sigue a la subcadena especificada.   |
| <code>fn:substringBefore</code>    | Devuelve la subcadena que precede a la subcadena especificada.   |

Gráfico 62. Etiquetas de la librería Functions

| Tag                         | Finalidad   |
|-----------------------------|---|
| <code>fn:toLowerCase</code> | Convierte los caracteres de una cadena a minúsculas.          |
| <code>fn:toUpperCase</code> | Convierte los caracteres de una cadena a mayúsculas.          |
| <code>fn:trim</code>        | Suprime el espacio en blanco en ambos extremos de una cadena. |

Gráfico 63. Etiquetas de la librería Functions (continuación)



### RECUERDA QUE...

- Las páginas jsp son las vistas de nuestra aplicación.
- Podemos incluir código java en ellas utilizando diferentes formatos: scriptlets, etiquetas, directivas, objetos implícitos, ...etc.
- Una página JSP se traduce en un servlet en tiempo de ejecución.

## 6. MANEJO DE SESIONES

HTTP es un protocolo sin datos de estado. Cada conexión de mensaje de solicitud y respuesta es independiente de todas las demás. Esto es significativo, porque entre solicitud y solicitud (del mismo usuario) el servidor HTTP no conserva una referencia a la solicitud anterior. Por tanto, el contenedor web debe establecer un mecanismo para almacenar la información de sesión de un usuario determinado.

Las sesiones constituyen un mecanismo para almacenar datos específicos del cliente a lo largo de diversas solicitudes HTTP.

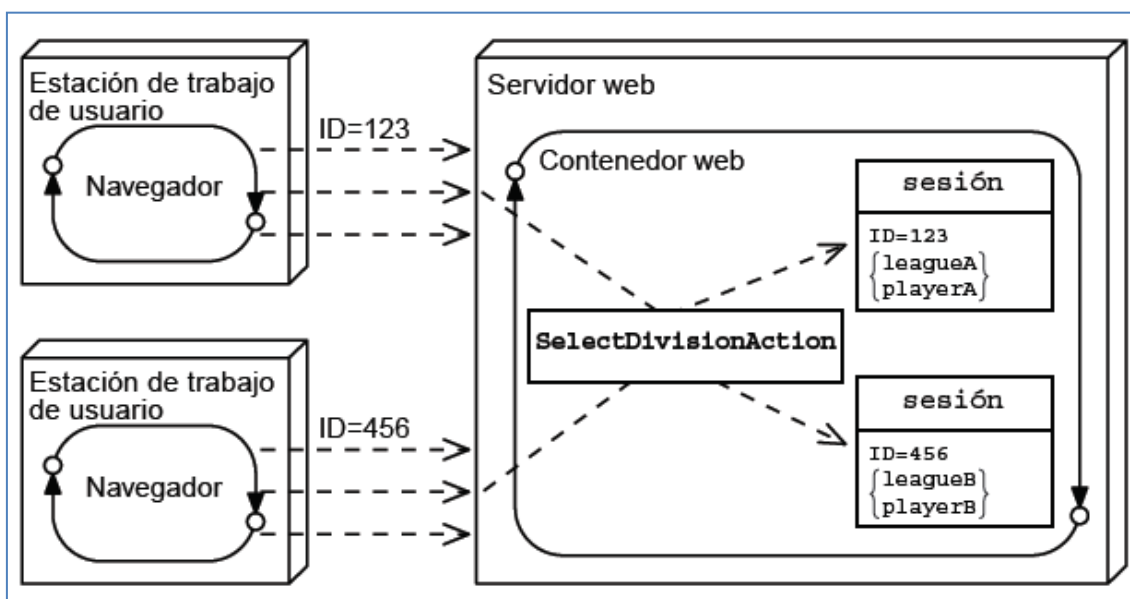


Gráfico 64. Gestión de las sesiones en el contenedor web

Cada cliente recibe un identificador de sesión único que el contenedor web utiliza para identificar el objeto de sesión de ese usuario.

### API DE HTTPSESSION

La especificación de Servlet proporciona una interfaz HttpSession que permite almacenar atributos de sesión. Puede almacenar, recuperar y eliminar atributos del objeto de sesión. El servlet tiene acceso al objeto de sesión con el método getSession del objeto HttpServletRequest.

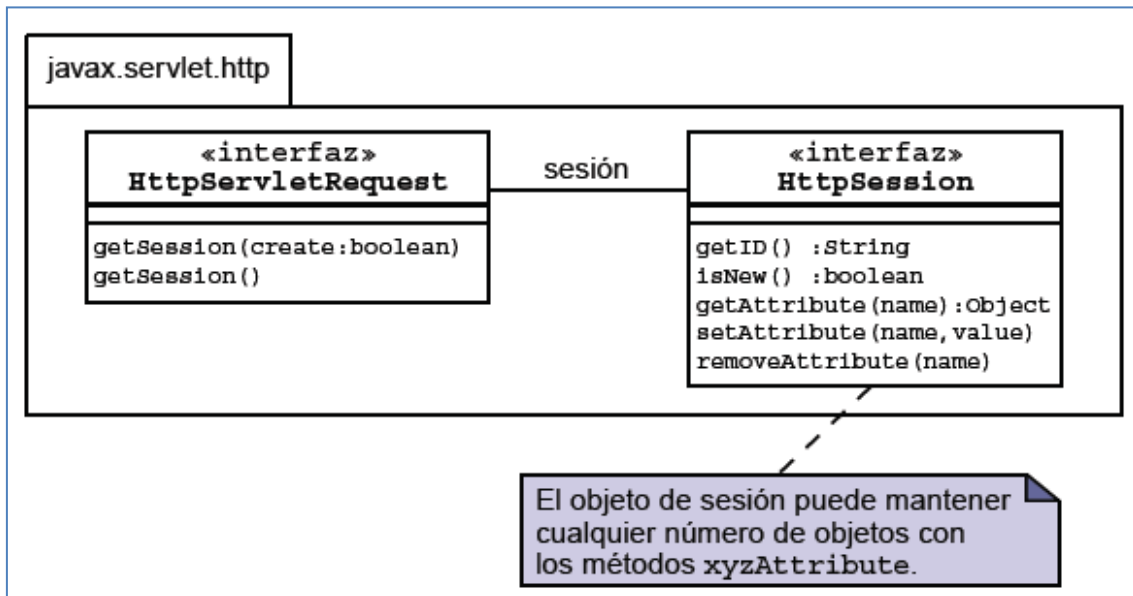


Gráfico 65. La API de HttpSession

## CREAR UNA SESION

La interfaz `HttpServletRequest` también tiene un método `getSession(boolean)`. Si llama a este método con un argumento `true`, se crea un objeto de sesión nuevo si aún no existía. Si llama a este método con un argumento `false`, se devuelve `null` si aún no existía una sesión. Podemos considerar que llamar al método `getSession()` equivale a llamar a `getSession(true)`.

```
HttpSession session = request.getSession();
```

Gráfico 66. Crear o recuperar una sesión

## ALMACENAMIENTO DE ATRIBUTOS DE SESIÓN

A través del método `setAttribute(nombre, valor)` podemos almacenar datos (atributos) en el contexto de sesión.

```
session.setAttribute("carrito", miCarro);
```

Gráfico 67. Almacenar un atributo una sesión

Estos atributos permanecerán todo el tiempo que dure la sesión. La diferencia con los atributos almacenados en la petición (request) es que estos solo existen hasta que se complete la respuesta en el cliente.

## ACCESO A ATRIBUTOS DE SESIÓN

Para recuperar los atributos de la sesión utilizamos el método `getAttribute(nombre)`. Esto nos devuelve un Object por lo cual habrá que hacer el casting al tipo de dato adecuado.

```
Carrito miCarro = (Carrito) sesion.getAttribute("carrito");
```

Gráfico 68. Recuperar un atributo de una sesión

## DESTRUCCIÓN DE LA SESIÓN

Cuando la aplicación web completa una sesión, el controlador puede destruir (efectivamente) la sesión con el método `invalidate`.

Hay otros dos mecanismos para destruir una sesión, ambos gestionados por el contenedor web.

- Puede configurar un parámetro de tiempo de espera en el descriptor de despliegue. El valor del elemento `session-timeout` debe ser un número entero que represente el número de minutos que puede durar una sesión si el usuario la ha dejado inactiva.
- El segundo mecanismo le permite controlar la longitud del intervalo de inactividad para un objeto de sesión específico. Puede utilizar el método `setMaxInactiveInterval` para cambiar el intervalo de inactividad (en segundos) del objeto de sesión.

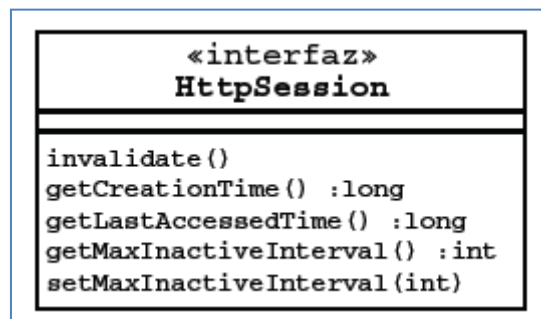


Gráfico 69. Otros métodos de HttpSession

## EJEMPLO

En nuestra aplicación tienda de productos vamos a implementar la opción de comprar productos. Para ello debemos implementar sesiones para que cada usuario tenga su propio carrito.

Desde la página mostrarTodos.jsp se ha añadido una columna para poder añadir un producto al carrito. Al pulsar sobre este link emitimos una petición al servlet con dos parámetros el id del producto a comprar y la opción 3.

Desde el servlet recuperamos o creamos la sesión del cliente y localizamos el atributo carrito, si este está localizamos el producto y lo agregamos a su carrito.

Si no tenemos un atributo carrito es porque la sesión se acaba de crear y es nueva por lo cual debemos crear el carrito y guardarlo como atributo de la sesión. Posteriormente se busca el producto y se almacena en el carrito.

Una vez concluida la operación de añadir el producto al carrito, redirigimos hacia la página mostrarCarrito.jsp donde mostramos los productos comprados, así como el importe total de la compra.

Estos datos se envían como atributos de la petición.



### RECUERDA QUE...

- Las sesiones permiten almacenar datos durante varias peticiones del mismo cliente.
- Se crea un objeto sesion por cada cliente que la necesita.
- Las sesiones utilizan Cookies para almacenar el numero de la sesion.

## 7. PARAMETROS INICIALES DEL SERVLET

Un servlet puede tener cualquier número de parámetros de inicialización.

El método `init` debe obtener los valores a partir del objeto `ServletConfig`. El contenedor web crea el objeto de configuración basándose en los parámetros de inicialización especificados en el descriptor de despliegue.

### DECLARACION PARAMETROS INICIALES DEL SERVLET

En el archivo `web.xml` debemos declarar los parámetros, tal como muestra la imagen.

```
<init-param>
  <param-name>Oferta</param-name>
  <param-value>Hoy todos los monitores al 10% de dto.</param-value>
</init-param>
```

Gráfico 70. Declaración de los parámetros iniciales del servlet

### LA API DE SERVLETCONFIG

La siguiente imagen ilustra la API de servlet y su relación con la interfaz `ServletConfig`.

Cada contenedor web debe implementar la interfaz `ServletConfig`. Las instancias de esta clase se pasan al método `init(ServletConfig)` definido en la interfaz de `Servlet`.

La clase `GenericServlet` (suministrada en la API de servlet) implementa la interfaz de `Servlet`. Implementa el método `init(ServletConfig)`, que almacena el objeto de configuración (`delegate`) y después llama al método `init()`. Se trata de un método `init` sin argumentos que se sustituye en las clases de servlet.

La interfaz `ServletConfig` proporciona el método `getInitParameter`, que permite recuperar los parámetros de inicialización del servlet. Por motivos prácticos, la clase `GenericServlet` también implementa la interfaz `ServletConfig`, y esos métodos delegan las llamadas en el objeto de configuración almacenado. Ello simplifica el código de servlet, porque permite llamar directamente al método `getInitParameter` (sin acceso directo al objeto de configuración).





Este parámetro lo recuperamos en el método `init()` del servlet y lo guardamos como atributo en el ámbito de la aplicación.

Cuando accedemos a la página `mostrarTodos.jsp` vemos el mensaje de la oferta.



#### RECUERDA QUE...

- Los parámetros iniciales del servlet son datos que se recogen en el momento de la inicialización de este.
- Se declaran en el descriptor de despliegue `web.xml` y se recuperan en el método `init()`.

## 8. PARAMETROS INICIALES DE CONTEXTO

Una aplicación web es una colección autocontenida de recursos estáticos y dinámicos: páginas HTML, archivos multimedia, archivos de datos y de recursos, servlets (y páginas de JSP) y otros objetos y clases Java auxiliares. El descriptor de despliegue de la aplicación web sirve para especificar la estructura y los servicios utilizados por una aplicación web.

Un objeto ServletContext es la representación de tiempo de ejecución de la aplicación web.

El objeto de contexto es accesible a todos los servlets en una aplicación web.

En el objeto de contexto se pueden almacenar atributos de ámbito de contexto mediante el método `setAttribute`. Los atributos pueden recuperarse del objeto de contexto mediante el método `getAttribute`.

También es posible eliminar un atributo de contexto con el método `removeAttribute`. El ámbito de contexto también se denomina ámbito de aplicación.

El método `getServletContext` lo suministra la clase `GenericServlet`, ampliada por la clase `HttpServlet` y las clases de servlet del usuario.

### API DE SERVLETCONTEXT

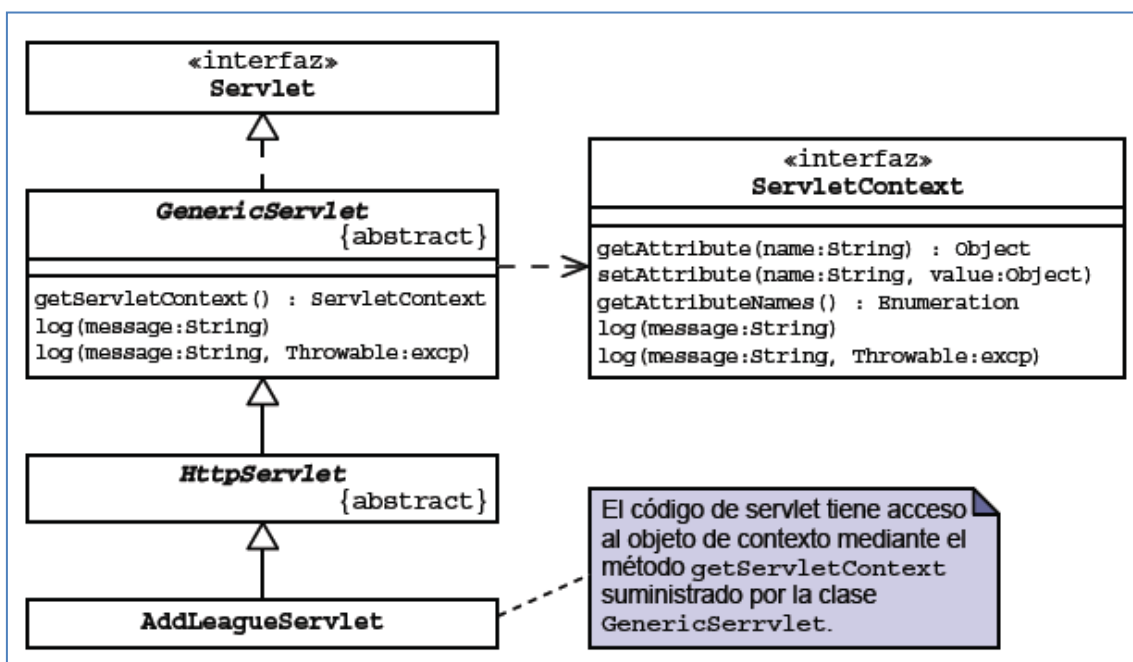


Gráfico 73. API de ServletContext

## DECLARAR LOS PARAMETROS DE CONTEXTO EN EL WEB.XML

```
<context-param>
  <param-name>OfertaMes</param-name>
  <param-value>Este mes las impresoras con un 25% de dto</param-value>
</context-param>
```

Gráfico 74. Declaración de parámetros de contexto

## CICLO DE VIDA DE LA APLICACIÓN WEB

La aplicación web (representada por el objeto de contexto) tiene un ciclo de vida gestionado por el contenedor web. Este ciclo de vida es similar al del servlet.

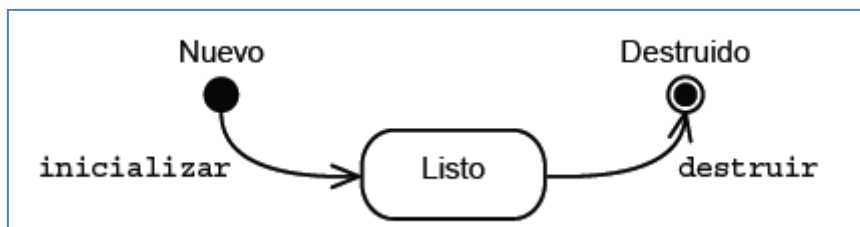


Gráfico 75. Ciclo de vida de las aplicaciones web

Cuando se inicia el contenedor web, se inicializa cada aplicación web.

Cuando se cierra el contenedor web, se destruye cada aplicación web.

Para recibir estos eventos del ciclo de vida de la aplicación web se puede un receptor de eventos de contexto de servlet.

## LISTENERS DE CONTEXTO

Normalmente, los datos compartidos de la aplicación deben hallarse en memoria antes de ejecutar cualquier solicitud HTTP en la aplicación web.

La interfaz `ServletContextListener` tiene dos métodos. El contenedor web invoca el método `contextInitialized` cuando se ha iniciado la aplicación web. El contenedor web invoca el método `contextDestroyed` cuando se está cerrando la aplicación web. Ambos métodos incluyen un argumento de evento. El objeto de evento proporciona acceso al objeto de contexto mediante el método `getServletContext`. Los receptores de eventos de contexto deben implementar esta interfaz

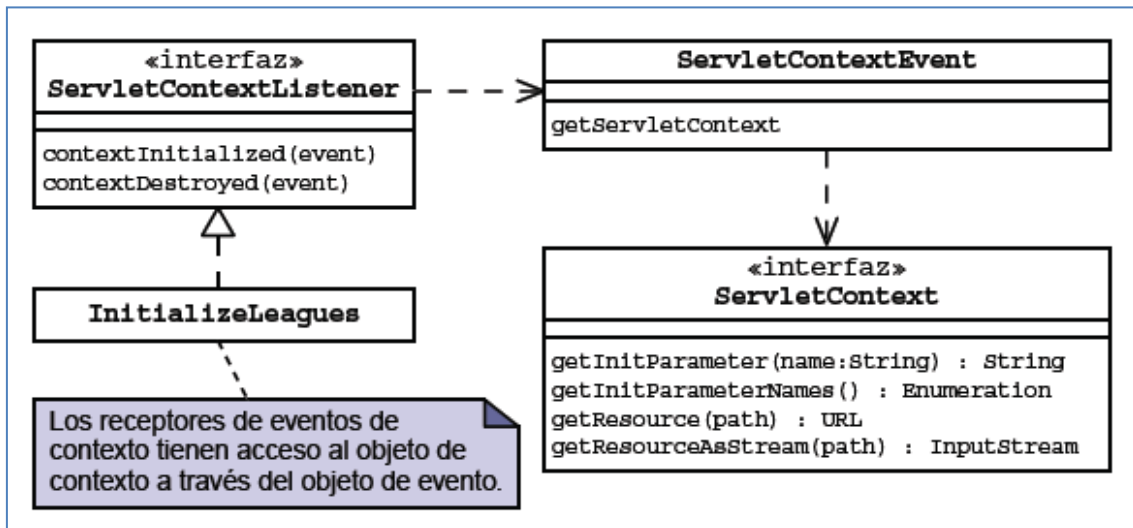


Gráfico 76. API de ServletContextListener

## CONFIGURACIÓN DEL RECEPTOR DE EVENTOS

Debemos configurar el listener de contexto en el descriptor de despliegue web.xml.

```

<listener>
  <description>ServletContextListener</description>
  <listener-class>app.web.ListenerAplicacion</listener-class>
</listener>

```

Gráfico 77. Declaración del listener de contexto

## RECUPERAR LOS PARAMETROS DE CONTEXTO

Los parámetros de contexto los recuperamos en el listener de la aplicación. Concretamente en el método contextInitialized.

```

public void contextInitialized(ServletContextEvent sce) {
    ServletContext aplicacion = sce.getServletContext();
    String mes = aplicacion.getInitParameter("OfertaMes");
    aplicacion.setAttribute("mes", mes);
}

```

Gráfico 78. Recuperar parámetros de contexto

## EJEMPLO

Hemos incluido un parámetro de contexto para las ofertas del mes. Dicho parámetro lo recuperamos dentro de un listener de contexto que nos hemos creado denominado ListenerAplicación en el paquete app.web.

Cuando recuperamos el parámetro de contexto lo guardamos como atributo de la aplicación de esa forma estará disponible antes de realizar ninguna petición.

Prueba de ello es que ahora podemos ver el mensaje de la oferta deslizándose en la página index.jsp.



### RECUERDA QUE...

- Los parámetros de contexto se consideran parámetros iniciales de la aplicación.
- Se declaran en el web.xml como <context-param>
- Se recuperan a través de un Listener de contexto.

## 9. COOKIES

Las cookies son el mecanismo de seguimiento de sesiones más utilizado.

Son archivos de texto almacenados en la maquina del cliente con ayuda de un navegador Web.

Las cookies almacenan información utilizando los pares nombre/valor y la devuelven al servidor que la creó en posteriores peticiones.

### API DE COOKIE

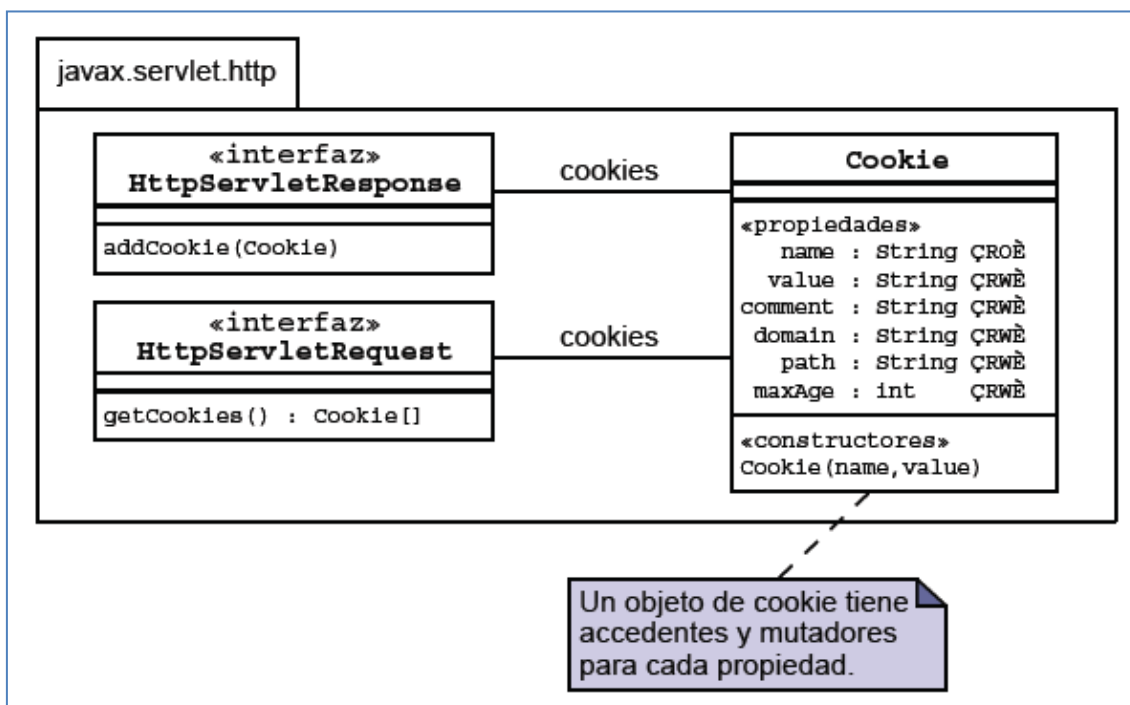


Gráfico 79. API de Cookie

### FUNCIONAMIENTO DE LAS COOKIES

El servidor crea una cookie, la llena de información relevante y la envía al navegador del cliente. El cliente almacena esta cookie en el disco duro y éste envía la cookie de nuevo al servidor en posteriores peticiones. Todas las cookies tienen una fecha de caducidad establecida, por lo que son temporales, pero se puede establecer esta fecha en un futuro lejano si se necesita.

## CREAR COOKIES

Para crear una cookie se crea un objeto de la clase Cookie:

```
Cookie micookie = new Cookie("nombre", "valor");
```

## ESTABLECER EL TIEMPO DE PERMANENCIA

Las cookies tienen un periodo tras el cual expiran, para poder establecer dicho periodo:

```
micookie.setMaxAge(int valor);
```

El valor representa los segundos transcurridos desde que se crea la cookie.

Ejemplos:

- 24\*60\*60 establece 24 horas
- 365\*24\*60\*60 establece un año
- 2\*365\*24\*60\*60 establece dos años

## ALMACENAR LA COOKIE EN EL NAVEGADOR

Una vez creada la cookie y habiendo establecido el tiempo de permanencia ya podemos almacenarla en el navegador del cliente, para ello:

```
response.addCookie(micookie);
```

## BORRAR LA COOKIE

Para eliminar una cookie establecemos el tiempo a cero.

```
micookie.setMaxAge(0);
```

## OBTENER TODAS LAS COOKIES DEL NAVEGADOR

Si queremos saber todas las cookies que tenemos almacenadas en el navegador del cliente crearemos un array de cookies donde las almacenaremos todas.

```
Cookie lista_cookies[] = request.getCookies();
```

## BUSCAR UNA COOKIE DETERMINADA

Para localizar una cookie determinada después de obtenerlas todas en un array de cookies. Lógicamente tendremos que recorrer dicho array y comparar de una en una. Tenemos dos opciones:

- Buscar por nombre: `lista_cookies[i].getName();`
- Buscar por valor: `lista_cookies[i].getValue();`

## MODIFICAR EL VALOR DE LA COOKIE

Para modificar o actualizar el valor de una cookie:

```
micookie.setValue("nuevo valor");
```

## ESTABLECER Y LEER COMENTARIOS DE LA COOKIE

Las cookies nos dan la opción de poner un comentario en ellas que muchas veces nos pueden servir de gran utilidad.

Establecer comentario: `micookie.setComment("comentario");`

Leer comentario: `micookie.getComment();`



## ESPECIFICAR PARA QUE DOMINIO FUE CREADA

Para saber que cookies son nuestras podemos especificar nuestro dominio.

```
micookie.setDomain();
```

Ejemplo: Si mi dominio es Atrium.com

```
micookie.setDomain(Atrium.com);
```

## EJEMPLO

Pretendemos almacenar en una cookie el nombre del usuario con el fin de poder personalizar la página mostrarTodos.jsp con un mensaje de bienvenida con su nombre.

Para poder realizar esta operación en el servlet recuperamos las cookies del cliente cuando solicita ver todos los productos. Buscamos la cookie con su nombre y si esta existe el programa continua.

Si no existe una cookie con su nombre entonces le dirigimos a la página login.jsp para que introduzca su nombre.

Al enviar la petición sobre ese formulario guardamos su nombre en una cookie.



### RECUERDA QUE...

- Las cookies son pequeños ficheros de texto que almacenan la información en el navegador web del usuario.
- Se utilizan para el manejo de sesiones.
- Cuando llega la petición de un cliente en ella viajan las cookies de nuestro dominio.

## 10.FILTROS

Una de las ventajas de la estructura de servlet es que el contenedor web intercepta las solicitudes entrantes antes de que lleguen al código, preprocesando la solicitud y la funcionalidad adicional (como la seguridad). El contenedor también puede posprocesar la respuesta que genera el servlet. En este capítulo aprenderemos a agregar unos componentes denominados filtros para conseguir esta funcionalidad del contenedor. Estos filtros amplían el preprocesamiento de la solicitud y el posprocesamiento de la respuesta.

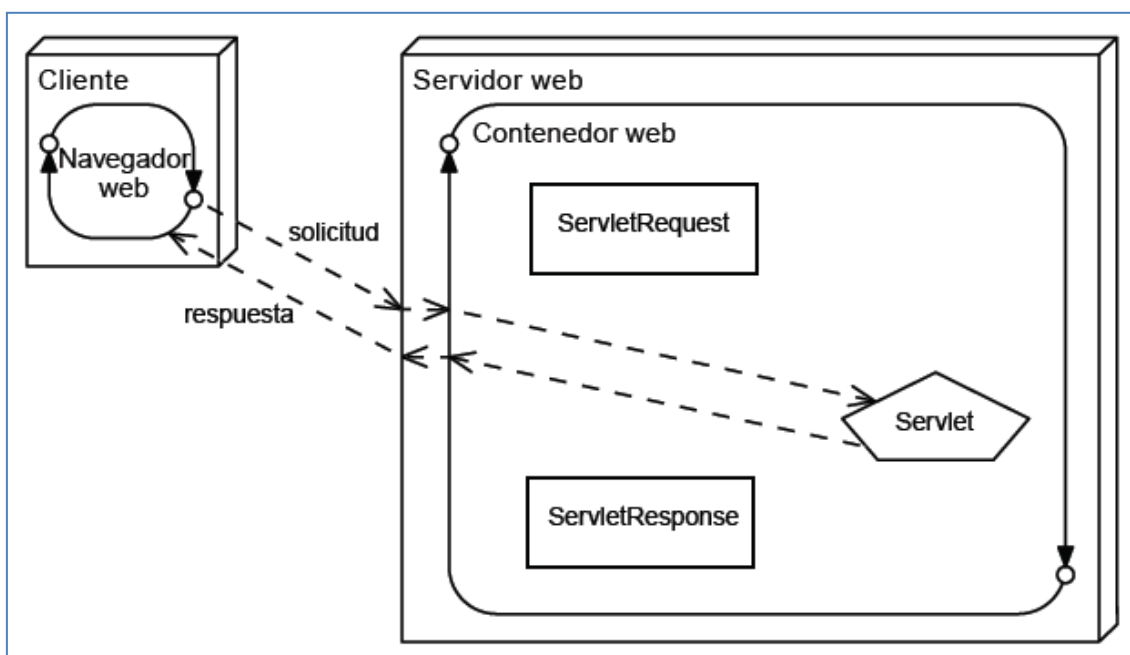


Gráfico 80. Petición y respuesta al servlet

### APLICACIÓN DE FILTROS A SOLICITUDES ENTRANTES

Los filtros son componentes que se pueden escribir y configurar para realizar más tareas de preprocesamiento y posprocesamiento. Cuando el contenedor web recibe una solicitud, se producen varias operaciones:

1. El contenedor web efectúa su preprocesamiento de la solicitud. Lo que sucede en este paso es responsabilidad del proveedor del contenedor.
2. El contenedor web comprueba si algún filtro tiene un patrón URL que coincida con la URL solicitada.
3. El contenedor web busca el primer filtro que tenga un patrón URL coincidente. Se ejecuta el código del filtro.

4. Si otro filtro tiene un patrón URL coincidente, a continuación se ejecuta su código. Este proceso continúa hasta que no quedan filtros que tengan patrones URL coincidentes.
5. Si no se produce ningún error, la solicitud pasa al servlet de destino.
6. El servlet devuelve la respuesta a su llamador. El último filtro aplicado a la solicitud es el primero que se aplica a la respuesta.
7. El último filtro que se aplicó a la solicitud es el primero que se aplica a la respuesta. El contenedor web puede realizar tareas de posprocesamiento en la respuesta.

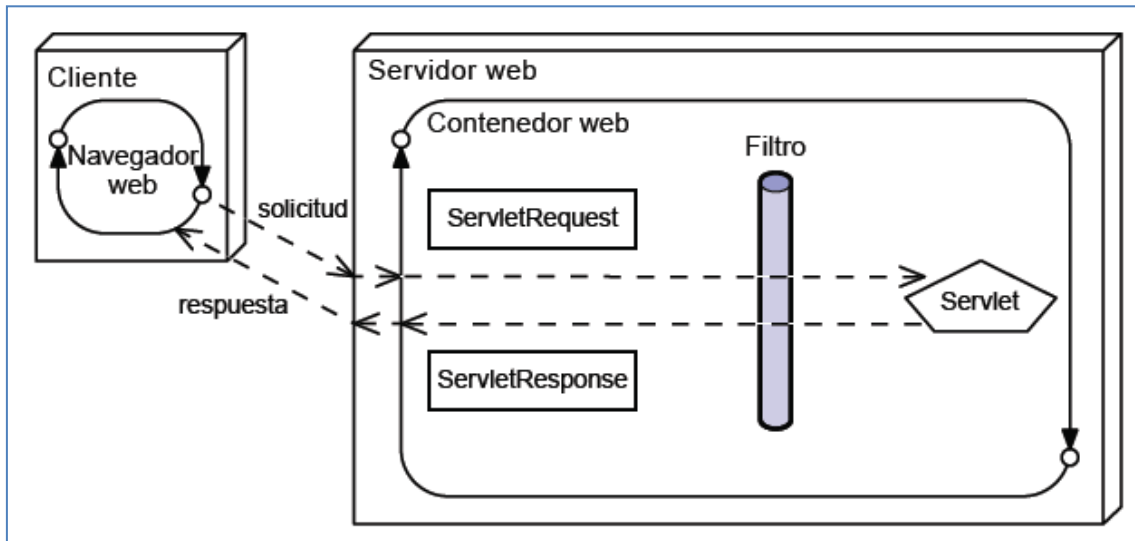


Gráfico 81. Aplicación de un filtro a la petición y respuesta del servlet

## USO DE FILTROS

Se pueden usar filtros para operaciones como:

- Bloquear el acceso a un recurso basándose en la identidad del usuario o la pertenencia a una función.
- Auditar las solicitudes entrantes.
- Comprimir el flujo de datos de la respuesta.
- Transformar la respuesta.
- Medir y registrar el rendimiento del servlet.

## APLICACION DE VARIOS FILTROS

Podemos aplicar varios filtros a distintos componentes web. Estos se ejecutarán en cascada.

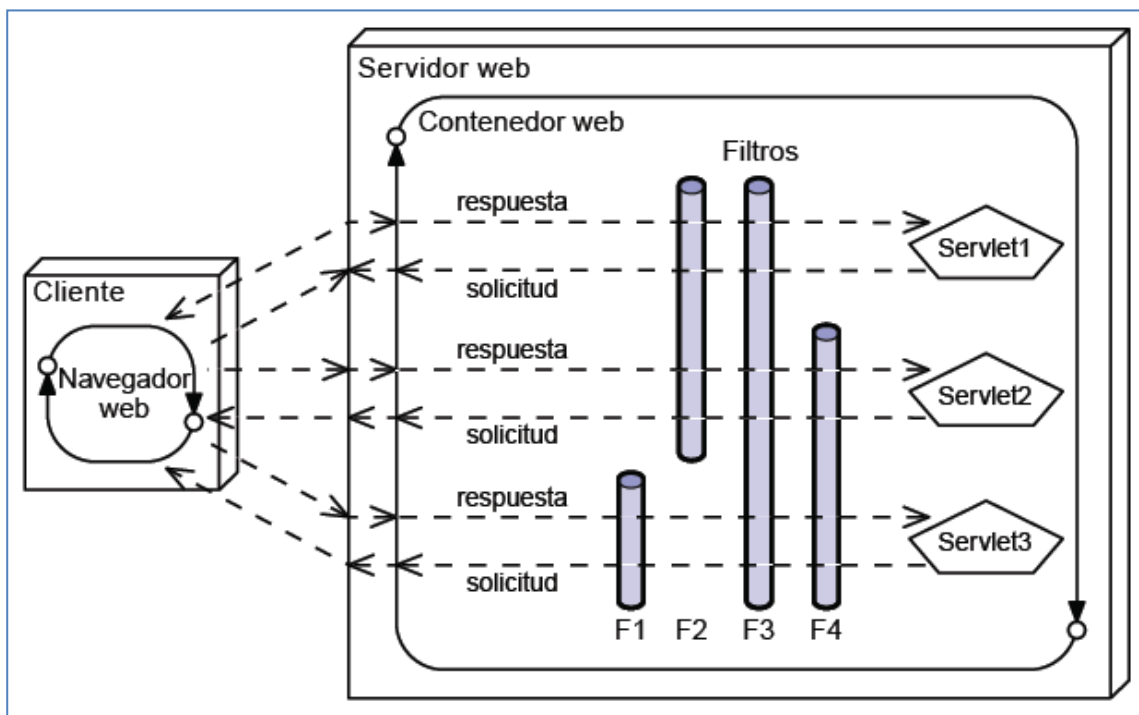


Gráfico 82. Aplicación de varios filtros

## API DE FILTRO

La API de filtro forma parte de la API de servlet básica. Las interfaces se hallan en el paquete `javax.servlet`.

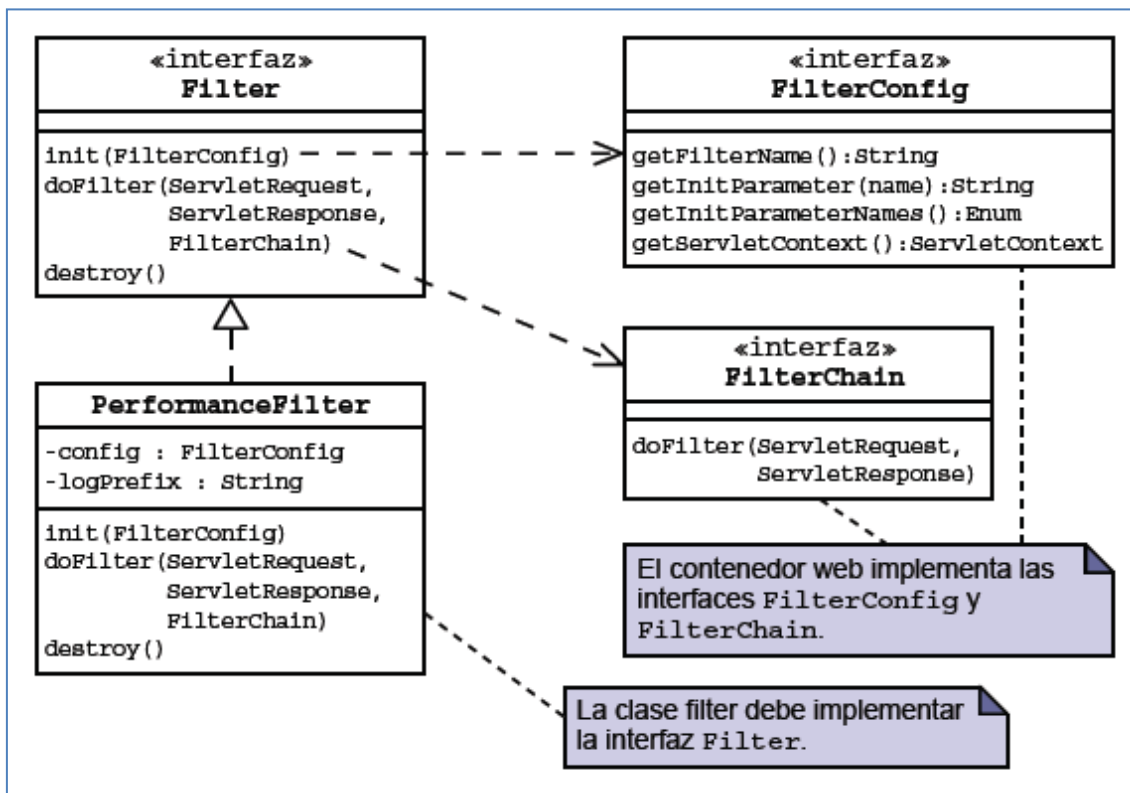


Gráfico 83. API de Filtro

Cuando se escribe un filtro, se implementa la interfaz **Filter**.

El contenedor pasa al método `init` una referencia **FilterConfig**. Debe almacenar la referencia **FilterConfig** como una variable de instancia en el filtro. La referencia **FilterConfig** sirve para obtener parámetros de inicialización del filtro, el nombre del filtro y **ServletContext**.

Cuando se llama al filtro, el contenedor pasa al método `doFilter` las referencias **ServletRequest**, **ServletResponse** y **FilterChain**. La referencia **FilterChain** sirve para pasar los objetos de solicitud y respuesta al siguiente componente de la cadena (filtro o servlet de destino).

## CICLO DE VIDA DE UN FILTRO

### MÉTODO INIT

El método `init` de un filtro se llama una vez cuando el contenedor instancia un filtro. A este método se le pasa **FilterConfig**, que suele almacenarse como una variable miembro para uso posterior. En el método `init` puede realizar cualquier tarea de una sola

inicialización. 'Por ejemplo, si el filtro tiene parámetros de inicialización, se pueden leer en el método `init` aplicando el método `getInitParameter` del objeto `FilterConfig`.

## **MÉTODO DOFILTER**

El método `doFilter` se llama para cada solicitud interceptada por el filtro. 'Es el equivalente en filtro al método `service` de un servlet. Este método recibe tres argumentos: `ServletRequest`, `ServletResponse` y `FilterChain`. El objeto de solicitud se utiliza para obtener información del cliente, por ejemplo, sobre parámetros o encabezado. El objeto de respuesta se utiliza para devolver información al cliente, por ejemplo, valores de encabezado.

Dentro del método `doFilter`, debe decidir si se invoca el siguiente componente de la cadena de filtros o si se bloquea la solicitud. Si prefiere invocar el siguiente componente, llame al método `doFilter` en la referencia `FilterChain`. El siguiente componente de la cadena puede ser otro filtro o un recurso web, como un servlet.

## **MÉTODO DESTROY**

Antes de que el contenedor web elimine una instancia de filtro del servicio, se llama al método `destroy`. 'Este método sirve para realizar las operaciones que deban producirse al final de la vida del filtro.

## **CONFIGURACIÓN DEL FILTRO**

'Como el contenedor web es responsable de los ciclos de vida de los filtros, hay que configurar los filtros en el descriptor de despliegue de la aplicación web.

## **DECLARACIÓN DE UN FILTRO EN EL ARCHIVO WEB.XML**

Como mínimo, una declaración de filtro debe contener los elementos `filter-name` y `filter-class`. Los parámetros de inicialización opcionales se incluyen dentro de la declaración del filtro en elementos `init-param`.

```
<filter>
  <filter-name>FiltroMedirTiempo</filter-name>
  <filter-class>app.web.FiltroMedirTiempo</filter-class>
</filter>
```

Gráfico 84. Declaración de un filtro

## DECLARACIÓN DE UNA ASIGNACIÓN DE FILTRO EN EL ARCHIVO WEB.XML

Los filtros se ejecutan cuando se solicita el recurso al que están asignados.

El elemento url-pattern puede especificar una URL determinada o utilizar un carácter comodín para indicar un conjunto de URL. El elemento url-pattern puede sustituirse por el elemento servlet-name para crear un filtro que se aplique a un servlet determinado.

```
<filter-mapping>
  <filter-name>FiltroMedirTiempo</filter-name>
  <url-pattern>/*</url-pattern>
</filter-mapping>
```

Gráfico 85. Mapeo de un filtro

Los mismos valores de url-pattern y servlet-name pueden emplearse en múltiples asignaciones de filtro para crear una cadena de filtros que se aplican a la solicitud. Los filtros se aplican por el orden en que aparecen en el descriptor de despliegue. Todos los filtros con asignación de URL se aplican antes que los filtros con asignación de servlet.

Los filtros sólo suelen aplicarse a las solicitudes que proceden directamente de un cliente. Si suministra un elemento dispatcher para la asignación de filtro, es posible aplicar filtros tanto a las distribuciones internas como a las solicitudes directas de los clientes.

- Un elemento dispatcher con el valor REQUEST indica que el filtro se aplicará a las solicitudes de un cliente.
- Un elemento dispatcher con el valor INCLUDE invocará el filtro se para las llamadas RequestDispatcher.include correspondientes.
- Un elemento dispatcher con el valor FORWARD invocará el filtro se para las llamadas RequestDispatcher.forward correspondientes.
- Un elemento dispatcher con el valor ERROR indica que el filtro se aplicará cuando se produzca un error (la distribución se debe a una condición de error).

Si quiere aplicar el filtro en múltiples escenarios, puede suministrar múltiples elementos dispatcher.

## EJEMPLO

Vamos a incorporar un filtro en nuestra aplicación para medir el tiempo que se tarda en atender una petición.

Para ello hemos creado la clase `FiltroMedirTiempo` en el paquete `app.web`.

El filtro lo tenemos declarado y mapeado en el `web.xml` para que atienda todas las peticiones.



### RECUERDA QUE...

- Un filtro es un componente que permite interceptar peticiones y respuestas contra una petición determinada.
- Los filtros se declaran y mapean en el `web.xml`



## ÍNDICE DE GRÁFICOS

|  |    |
|--|----|
| Gráfico 1. Esquema de conexión.....  | 7  |
| Gráfico 2. Conexión a través de ODBC.....                                      | 8  |
| Gráfico 3. Conexión a través de Librería nativa .....                          | 9  |
| Gráfico 4. Conexión a través de protocolo nativo .....                         | 10 |
| Gráfico 5. Fragmento para abrir una conexión a Derby .....                     | 11 |
| Gráfico 6. Fragmento para cerrar la conexión .....                             | 11 |
| Gráfico 7. Ejemplo executeUpdate .....   | 12 |
| Gráfico 8. Ejemplo executeQuery .....  | 12 |
| Gráfico 9. Ejemplo PreparedStatement con executeUpdate .....                   | 13 |
| Gráfico 10. Ejemplo PreparedStatement con executeQuery .....                   | 13 |
| Gráfico 11. Ejemplo CallableStatement con executeQuery .....                   | 14 |
| Gráfico 12. Procesar resultados .....  | 15 |
| Gráfico 13. Obtención de los metadatos de la consulta .....                    | 16 |
| Gráfico 14. Estructura de un modulo .war.....                                  | 18 |
| Gráfico 15. Patrón MVC en aplicaciones web.....                                | 19 |
| Gráfico 16. Formato url.....   | 20 |
| Gráfico 17. Petición url con método get.....                                   | 21 |
| Gráfico 18. Petición url con método post.....                                  | 21 |
| Gráfico 19. Sobreescritura URL .....   | 21 |
| Gráfico 20. Envío de parámetros en un formulario .....                         | 22 |
| Gráfico 21. Petición y respuesta en un servlet .....                           | 23 |
| Gráfico 22. Jerarquía del API servlets .....                                   | 25 |
| Gráfico 23. Métodos de ciclo de vida de un GenericServlet.....                 | 29 |
| Gráfico 24. Métodos de ciclo de vida de un HttpServlet .....                   | 29 |
| Gráfico 25. Ejemplo de un HttpServlet .....                                    | 30 |
| Gráfico 26. Mapeo de un servlet en web.xml.....                                | 31 |
| Gráfico 27. Invocar al servlet mediante un link .....                          | 31 |
| Gráfico 28. Invocar al servlet mediante el atributo action del formulario..... | 31 |
| Gráfico 29. Mapeo de servlet mediante anotaciones. ....                        | 32 |
| Gráfico 30. Almacenar un atributo en la petición.....                          | 32 |
| Gráfico 31. Recuperar el atributo en la pagina jsp. ....                       | 33 |

|   |    |
|---|----|
| Gráfico 32. Paso de atributos entre servlets .....            | 33 |
| Gráfico 33. Transformación de una jsp en un servlet. ....     | 35 |
| Gráfico 34. Métodos del servlet generado .....                | 36 |
| Gráfico 22. Sintaxis de declaraciones .....                   | 37 |
| Gráfico 35. Ejemplos de declaraciones .....                   | 37 |
| Gráfico 36. Sintaxis de las expresiones .....                 | 37 |
| Gráfico 37. Ejemplo de expresión .....                        | 37 |
| Gráfico 38. Sintaxis de los scriptlets .....                  | 38 |
| Gráfico 39. Ejemplo de scriptlet.....                         | 38 |
| Gráfico 40. Ejemplo comentario HTML.....                      | 38 |
| Gráfico 41. Ejemplo comentario JSP .....                      | 38 |
| Gráfico 42. Ejemplo comentario Java .....                     | 39 |
| Gráfico 43. Sintaxis de las directivas.....                   | 39 |
| Gráfico 44. Sintaxis etiqueta <jsp:useBean>.....              | 42 |
| Gráfico 45. Ejemplo etiqueta <jsp:useBean>.....               | 42 |
| Gráfico 46. Código java equivalente a <jsp:useBean> .....     | 43 |
| Gráfico 46. Sintaxis etiqueta <jsp:setProperty> .....         | 43 |
| Gráfico 47. Sintaxis etiqueta <jsp:setProperty> .....         | 44 |
| Gráfico 48. Objetos implícitos de JSP.....                    | 47 |
| Gráfico 49. Sintaxis EL .....                                 | 48 |
| Gráfico 50. Escapar expresiones .....                         | 48 |
| Gráfico 51. Acceder a las propiedades de un bean con EL ..... | 49 |
| Gráfico 52. Ejemplo de acceso a las propiedades del bean..... | 49 |
| Gráfico 53. Objetos implícitos de EL .....                    | 50 |
| Gráfico 42. Operadores aritméticos de EL.....                 | 51 |
| Gráfico 54. Ejemplos de operaciones aritméticas de EL.....    | 51 |
| Gráfico 55. Operadores de comparación de EL .....             | 51 |
| Gráfico 56. Operadores lógicos de EL .....                    | 52 |
| Gráfico 57. Librerías de etiquetas JSTL .....                 | 52 |
| Gráfico 58. Etiquetas de la librería Core .....               | 53 |
| Gráfico 59. Etiquetas de la librería xml .....                | 54 |
| Gráfico 60. Etiquetas de la librería format .....             | 55 |
| Gráfico 61. Etiquetas de la librería sql .....                | 56 |
| Gráfico 62. Etiquetas de la librería Functions .....          | 57 |

|   |    |
|---|----|
| Gráfico 63. Etiquetas de la librería Functions (continuación).....              | 57 |
| Gráfico 64. Gestión de las sesiones en el contenedor web .....                  | 59 |
| Gráfico 65. La API de HttpSession .....   | 60 |
| Gráfico 66. Crear o recuperar una sesión .....                                  | 60 |
| Gráfico 67. Almacenar un atributo una sesión .....                              | 60 |
| Gráfico 68. Recuperar un atributo de una sesión .....                           | 61 |
| Gráfico 69. Otros métodos de HttpSession .....                                  | 61 |
| Gráfico 70. Declaración de los parámetros iniciales del servlet .....           | 63 |
| Gráfico 71. API de ServletConfig .....  | 64 |
| Gráfico 72. Recuperar los parámetros iniciales del servlet .....                | 64 |
| Gráfico 73. API de ServletContext.....  | 66 |
| Gráfico 74. Declaración de parámetros de contexto .....                         | 67 |
| Gráfico 75. Ciclo de vida de las aplicaciones web .....                         | 67 |
| Gráfico 76. API de ServletContextListener.....                                  | 68 |
| Gráfico 77. Declaración del listener de contexto.....                           | 68 |
| Gráfico 78. Recuperar parámetros de contexto .....                              | 68 |
| Gráfico 79. API de Cookie.....  | 70 |
| Gráfico 80. Petición y respuesta al servlet.....                                | 74 |
| Gráfico 81. Aplicación de un filtro a la petición y respuesta del servlet ..... | 75 |
| Gráfico 82. Aplicación de varios filtros .....                                  | 76 |
| Gráfico 83. API de Filtro .....   | 77 |
| Gráfico 84. Declaración de un filtro .....                                      | 79 |
| Gráfico 85. Mapeo de un filtro.....   | 79 |

# DOCUMENTACIÓN

## → DOCUMENTACIÓN ADICIONAL

- ▷ [Características generales](#) del lenguaje
- ▷ Clases [esenciales](#)
- ▷ El API de [colecciones](#)
- ▷ [Nuevas características](#) de Java 8

## → DOCUMENTACIÓN ADICIONAL

- ▷ [Tutorial oficial de Java](#)
- ▷ [Libros gratuitos de Java en castellano](#)
- ▷ [Libros gratuitos sobre Java](#)
- ▷ [Libros recomendados: Java Language Specification y Programming Using Java](#)
- ▷ Información sobre [métodos Java](#)
- ▷ Tutorial exhaustivo sobre [Java 8](#)

## → PROGRAMACIÓN ORIENTADA A OBJETOS

- ▷ [Nociones fundamentales](#)
- ▷ [Diagramas de clases bajo UML](#)

## → HERRAMIENTAS UTILIZADAS EN EL CURSO

- ▷ [JSE 8-11](#)
- ▷ [NetBeans 8.X](#)

Todos  
aprendemos.  
Gracias!!



## ¡Seguimos en contacto!

[www.iconotc.com](http://www.iconotc.com)

