Práctica 7.7: Conceptos adicionales sobre *Servlets*

En esta práctica seguiremos analizando conceptos relativos a los *Servlets*:

- Parámetros de inicialización de *Servlet* con *ServletConfig*.
- Parámetros de inicialización de aplicación con *ServletContext*.
- Parámetros de inicialización de servidor.
- Filtros.

1. **Parámetros de inicialización de *Servlet* con *ServletConfig***.

   La clase *ServletConfig* permite al contenedor de *Servlets*, en nuestro caso *Tomcat*, pasar información al *Servlet* durante su inicialización. Los parámetros de inicialización son declarados en el descriptor de despliegue **web.xml** y son recuperados con el método:

   ```
   ServletConfig.getInitParam("nombreParametro")
   ```

   Para probar este concepto crea un nuevo *Dynamic Web Project* llamándolo **Filtros**. Copia y pega los recursos del proyecto **AccesoDatos**: el *Servlet ConsultaServlet.java*, la página **ConsultaLibros.html** y el descriptor de despliegue **web.xml**

   Modifica el fichero **ConsultaLibros.html**, que será el punto de entrada a la aplicación, cambiando la acción del método *get*.

   ```
   <form method="get" action="http://localhost:8080/Filtros/consulta">
   ```

   Modifica el descriptor de despliegue:

   ```
   <display-name>Filtros</display-name>
   ```

   Ahora vamos a modificar el proyecto para probar los parámetros de inicialización del *Servlet*.

   1.1. Primero edita el descriptor de despliegue para añadir al servlet ya existente, **ConsultaServlet**, tres parámetros de inicialización:
      - La URL de la base de datos.
      - El usuario con el que nos vamos a conectar.
      - La contraseña de dicho usuario.

```
<servlet>
    <servlet-name>ConsultaUsuario</servlet-name>
    <servlet-class>ConsultaServlet</servlet-class>
    <init-param>
         <param-name>URLBaseDeDatos</param-name>
         <param-value>jdbc:mysql://localhost/TiendaLibros</param-value>
    </init-param>
    <init-param>
         <param-name>usuario</param-name>
         <param-value>root</param-value>
    </init-param>
    <init-param>
         <param-name>password</param-name>
         <param-value>despliegue</param-value>
    </init-param>
  </servlet>
```

1.2. Ahora modifica el código del *Servlet* para recoger los parámetros en la inicialización del *Servlet*, método *init*.

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

  public class ConsultaServlet extends HttpServlet {
  // El método doGet() se ejecuta una vez por cada petición HTTP GET.

   private String userName;
   private String password;
   private String url;

   @Override
   public void doGet(HttpServletRequest request, HttpServletResponse response)
   throws ServletException, IOException {
   // Establecemos el tipo MIME del mensaje de respuesta
   response.setContentType("text/html");
   // Creamos un objeto para poder escribir la respuesta
   PrintWriter out = response.getWriter();
   Connection conn = null;
   Statement stmt = null;
   try {
   //Paso 1: Cargar el driver JDBC.
   Class.forName("com.mysql.jdbc.Driver").newInstance();
   // Paso 2: Conectarse a la Base de Datos utilizando la clase Connection

   //URL de la base de datos(equipo, puerto, base de datos)

   conn = DriverManager.getConnection(url, userName, password);
   // Paso 3: Crear sentencias SQL, utilizando objetos de tipo Statement
   stmt = conn.createStatement();
```

```
    // Paso 4: Ejecutar las sentencias SQL a través de los objetos Statement
    String sqlStr = "select * from libros where autor = "
    + "'" + request.getParameter("autor") + "'";
    // Generar una página HTML como resultado de la consulta
    out.println("<html><head><title>Resultado de la consulta</title></head><body>");
    out.println("<h3>Gracias por tu consulta.</h3>");
    out.println("<p>Tu consulta es: " + sqlStr + "</p>");
    ResultSet rset = stmt.executeQuery(sqlStr);

    // Paso 5: Procesar el conjunto de registros resultante utilizando ResultSet
    int count = 0;
    while(rset.next()) {
     out.println("<p>" + rset.getString("autor")
     + ", " + rset.getString("titulo")
     + ", " + rset.getDouble("precio") + "</p>");
     count++;
    }
    out.println("<p>==== " + count + " registros encontrados =====</p>");
    out.println("</body></html>");
    } catch (Exception ex) {
    ex.printStackTrace();
    } finally {
    out.close(); // Cerramos el flujo de escritura
    try {

    // Cerramos el resto de recursos
    if (stmt != null) stmt.close();
    if (conn != null) conn.close();
    } catch (SQLException ex) {
    ex.printStackTrace();
    }
    }
    }
  @Override
  public void init(ServletConfig config) throws ServletException {
    super.init(config);
    // Lectura de los parámetros de inicialización del Servlet
    userName=config.getInitParameter("usuario");
    password=config.getInitParameter("password");
    url=config.getInitParameter("URLBaseDeDatos");
}
}
```

1.3. Comprueba cómo funciona de nuevo la aplicación accediendo a la *URL* de la misma, `http://localhost:8080/Filtros/ConsultaLibros.html`. Con la diferencia de que ahora los parámetros de configuración del *Servlet* no residen en el código sino en el descriptor de despligue con las ventajas que ello conlleva.

2. **Parámetros de inicialización de aplicación con *ServletContext*.**

Esta clase encapsula el contexto de una aplicación web por lo que sería más correcto si su nombre fuera *ApplicationContext*. Permite la comunicación entre el contenedor, *Tomcat*, y el *Servlet* y también pasar información entre los diferentes *Servlets* y *JSPs* de la misma aplicación mediante los métodos: *setAttribute("nombre", objeto)* y *getAttribute("nombre")*. Vamos a realizar un cambio en el *Servlet* anterior para que los parámetros de inicialización se conviertan en atributos del contexto disponibles para cualquier *Servlet* o *JSP* de la aplicación.

2.1. Continuando con el ejemplo, modifica el método *init* del *Servlet ConsultaServlet* con el siguiente código:

```
@Override
public void init(ServletConfig config) throws ServletException {
  super.init(config);
  // Se leen los parámetros de inicialización de Servlet y
  // se convierten en atributos del contexto para compartirlos con
  // cualquier servlet y JSP de la aplicación
  ServletContext context = config.getServletContext();
  context.setAttribute("URLBaseDeDatos", config.getInitParameter("URLBaseDeDatos"));
  context.setAttribute("usuario", config.getInitParameter("usuario"));
  context.setAttribute("password", config.getInitParameter("password"));
  // Se recuperan las variables de contexto de la aplicación
  userName=(String)context.getAttribute("usuario");
  password=(String)context.getAttribute("password");
  url=(String)context.getAttribute("URLBaseDeDatos");
}
```

2.2. Prueba de nuevo la aplicación para comprobar su funcionamiento. Con este cambio los parámetros de inicialización ahora se han convertido en variables de contexto utilizables por cualquier *Servlet* o *JSP* de la aplicación.

2.3. También es posible definir parámetros de inicializaicón a nivel de contexto en el descriptor de despliegue y recuperarlos desde cualquier *Servlet* o *JSP* de la aplicación.

2.4. Modifica el descriptor de despliegue añadiendo los siguientes parámetros de contexto. Observa cómo no están dentro de ningún *Servlet* como ocurría con los parámetros de inicialización del *Servlet*.

```
...
<context-param>
    <param-name>URLBaseDeDatos</param-name>
    <param-value>jdbc:mysql://localhost/TiendaLibros</param-value>
</context-param>
<context-param>
    <param-name>usuario</param-name>
    <param-value>root</param-value>
</context-param>
<context-param>
    <param-name>password</param-name>
    <param-value>despliegue</param-value>
</context-param>
</web-app>
```

2.5. Modifica el método *init* del *Servlet ConsultaServlet* con el siguiente código:

```
@Override
public void init(ServletConfig config) throws ServletException {
  super.init(config);
  // Se leen los parámetros de inicialización del contexto
  // Estos parámetros podrán ser utilizados por otros Servlet o JSP de la aplicación
  ServletContext context = config.getServletContext();
  url=context.getInitParameter("URLBaseDeDatos");
  userName=context.getInitParameter("usuario");
  password= context.getInitParameter("password");
```

```
}
```

2.6. Comprueba cómo el proyecto sigue funcionando correctamente.

3. **Parámetros de inicialización de servidor**.

3.1. De manera similiar a como hemos declarado parámetros de inicialización en el descriptor de despliegue de la aplicación, podemos crear parámetros de inicialización a nivel de servidor. Los definiríamos en el fichero **CATALINA_HOME\conf\web.xml** con la misma sintaxis que los parámetros de aplicación.

```
<context-param>
  <param-name>usuario</param-name>
  <param-value>root</param-value>
</context-param>
```

3.2. La diferencia radica en que estos parámetros de inicialización a nivel de servidor pueden ser recuperados por cualquier aplicación utilizando el método *getInitParameter* de *ServletContext*.

4. **Filtros**.

Un filtro es una clase que implementa la interface `javax.servlet.Filter`, y que intercepta las peticiones a recursos del contenedor Web, *Tomcat*, previo al procesamiento de los mismos por parte del contenedor.

Un filtro "envuelve" los componentes del servidor con funcionalidad adicional, interceptando las peticiones del cliente, lo que nos permite una gran versatilidad a la hora de gestionarlas.

4.1. Vamos a añadir un filtro que registre los accesos a nuestra aplicación para ilustrar su uso.

4.2. En nuestro proyecto **Filtros**, en la carpeta **Java Resources\src** pulsa sobre el paquete por defecto y añade un nuevo filtro, llamado **FiltroContador**, Figura 1.

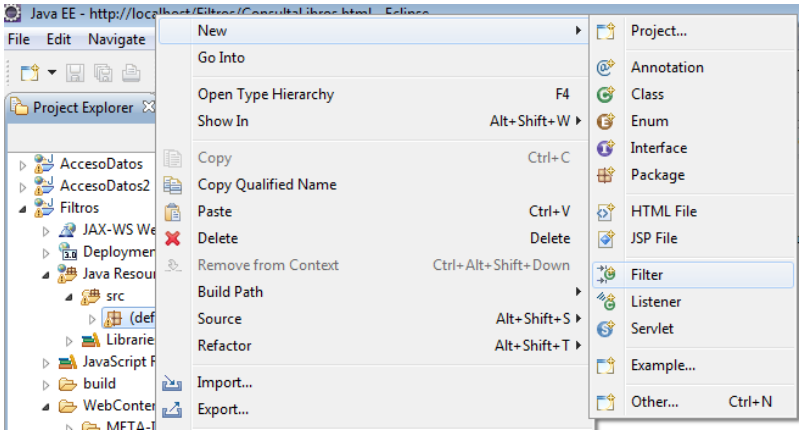

Figura 1: Añadir filtro

4.3. El código del filtro será el siguiente:

```
import java.io.IOException;
import javax.servlet.*;
public class FiltroContador implements Filter {

 // Para tener una referencia al contexto de filtro
 FilterConfig config;

 public void init(FilterConfig config) { this.config = config; }

 public void doFilter(ServletRequest request, ServletResponse response,
        FilterChain chain) throws IOException, ServletException {
 // Tomamos el Contexto de sesión donde almacenaremos el contador
 ServletContext miContexto = config.getServletContext();
 // Vemos si el atributo contador existe dentro del contexto
 // En caso contrario lo creamos e inicializamos a cero
 Integer intContador = (Integer)miContexto.getAttribute("contador");
 if(intContador == null) {
   intContador = new Integer(0);
 }
 intContador = new Integer(intContador.intValue() + 1);
 miContexto.setAttribute("contador", intContador);
 // Instruimos al contenedor para que invoque al siguiente filtro
 chain.doFilter(request, response);
 }
 public void destroy() {}
}
```

4.4. Lo que hace este filtro es crear una variable de aplicación **contador** donde almacena el número de peticiones a ficheros ***.html** de nuestra aplicación.

4.5. A continuación hay que declararlo en el descriptor de despliegue de nuestra aplicación añadiendo la siguiente información.

```
...
<filter>
  <filter-name>FiltroContador</filter-name>
  <filter-class>FiltroContador</filter-class>
   <init-param>
      <param-name>parametro1</param-name>
      <param-value>valor1</param-value>
   </init-param>
</filter>
<filter-mapping>
   <filter-name>FiltroContador</filter-name>
   <url-pattern>*.html</url-pattern>
</filter-mapping>
</web-app>
```

4.6. Observa diferentes aspectos sobre esta declaración:

- Fijate como es parecida a la declaración de un *Servlet*, primero declaramos el Filtro y luego declaramos qué peticiones va a interceptar.

- Recuerda que en *filter-class* deberas poner la ruta completa de la clase si estás utilizando paquetes.
- Observa cómo también se pueden crear parámetros de inicialización del filtro.
- Luego, hemos decidido que nuestro filtro intercepte todas las peticiones a archivos ***.html**. Mira estos dos ejemplos:

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

Interceptaría cualquier petición a nuestra aplicación, ya sea dinámica o estática.

```
<filter-mapping>
   <filter-name>unFiltro</filter-name>
   <servlet-name>unServlet</servlet-name>
</filter-mapping>
```

Interceptaría las peticiones realizadas al *Servlet* **unServlet**.

- Hay que indicar que si declaramos varios filtros, el orden en el que se declaren en el descriptor de despliegue determina el orden en el que se ejecutan.

4.7. Para finalizar modifica el *Servlet* **ConsultaServlet** para mostrar el contador junto con el resultado de la consulta.

```
import java.io.*;
import java.sql.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class ConsultaServlet extends HttpServlet {
// El método doGet() se ejecuta una vez por cada petición HTTP GET.

 private String userName;
 private String password;
 private String url;
 private Integer contador;
 private ServletContext context;

 @Override
 public void doGet(HttpServletRequest request, HttpServletResponse response)
  throws ServletException, IOException {
 // Establecemos el tipo MIME del mensaje de respuesta
 response.setContentType("text/html");
 // Creamos un objeto para poder escribir la respuesta
 PrintWriter out = response.getWriter();
 Connection conn = null;
 Statement stmt = null;
 try {
  //Paso 1: Cargar el driver JDBC.
  Class.forName("com.mysql.jdbc.Driver").newInstance();
  // Paso 2: Conectarse a la Base de Datos utilizando la clase Connection

  //URL de la base de datos(equipo, puerto, base de datos)
```

```
  conn = DriverManager.getConnection(url, userName, password);
  // Paso 3: Crear sentencias SQL, utilizando objetos de tipo Statement
  stmt = conn.createStatement();
  // Paso 4: Ejecutar las sentencias SQL a través de los objetos Statement
  String sqlStr = "select * from libros where autor = "
  + "'" + request.getParameter("autor") + "'";
  // Generar una página HTML como resultado de la consulta
  out.println("<html><head><title>Resultado de la consulta</title></head><body>");
  out.println("<h3>Gracias por tu consulta.</h3>");
  out.println("<p>Tu consulta es: " + sqlStr + "</p>");
  ResultSet rset = stmt.executeQuery(sqlStr);
  // Paso 5: Procesar el conjunto de registros resultante utilizando ResultSet
  int count = 0;
  while(rset.next()) {
   out.println("<p>" + rset.getString("autor")
   + ", " + rset.getString("titulo")
   + ", " + rset.getDouble("precio") + "</p>");
   count++;
  }
  out.println("<p>==== " + count + " registros encontrados =====</p>");
  // Se recupera la variable contador del ServletContext
  contador= (Integer)context.getAttribute("contador");
  out.println("<p>==== " + contador.intValue() + " peticiones *.html===</p>");
  out.println("</body></html>");
 } catch (Exception ex) {
 ex.printStackTrace();
 } finally {
  out.close(); // Cerramos el flujo de escritura
 try {

 // Cerramos el resto de recursos
  if (stmt != null) stmt.close();
  if (conn != null) conn.close();
 } catch (SQLException ex) {
  ex.printStackTrace();
 }
}
}

@Override
public void init(ServletConfig config) throws ServletException {
  super.init(config);
  // Se leen los parametros de inicialización de Servlet y
  // se convierten en atributos del contexto para compartirlos con
  // cualquier servlet y JSP de la aplicación
  // Se guarda el ServletContext que representa el contexto
  // de la aplicación para usarlo en el doGet
  context = config.getServletContext();
  url=context.getInitParameter("URLBaseDeDatos");
```

```
    userName=context.getInitParameter("usuario");
    password= context.getInitParameter("password");
   }
  }
```

4.8. Finalmente ejecuta el proyecto y comprueba cómo el filtro contabiliza las peticiones. Puedes ir actualizando la página **ConsultaLibros.html**, antes de pulsar en **Buscar**, para poder comprobarlo, Figura 2.

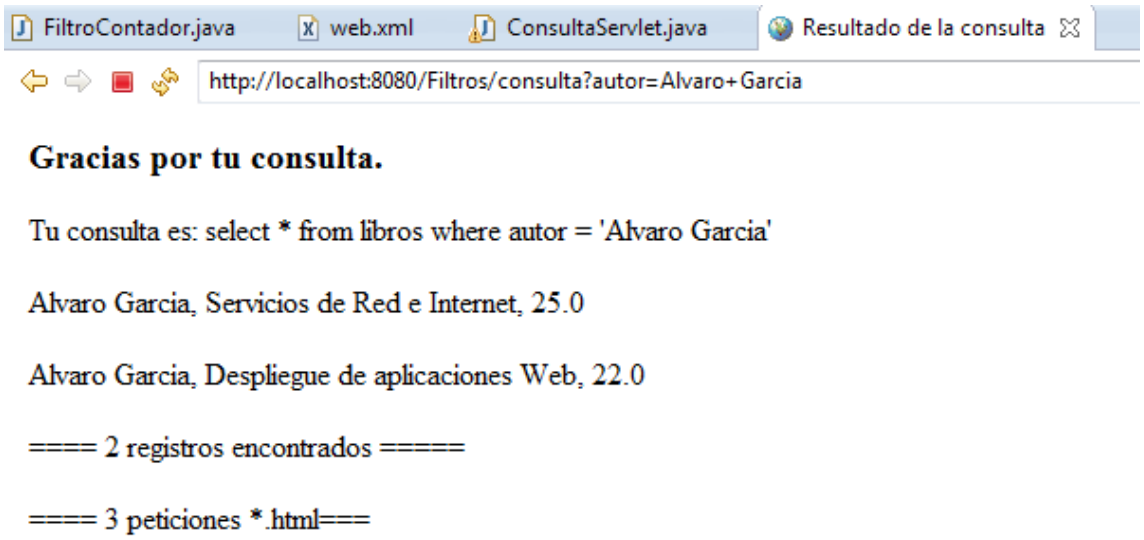

Figura 2: Prueba filtro

◇