Servlets y JSP

- Sesión 2: Procesamiento de peticiones

Puntos a tratar

- Peticiones
- Respuestas
- Procesamiento de peticiones GET y POST
- Manejo de formularios
  - Ejemplo
- Pruebas de servlets con Cactus

ServletRequest y *HttpServletRequest*

- Los objetos *ServletRequest* se emplean para obtener información sobre las peticiones de los clientes
- En concreto, el subtipo *HttpServletRequest* se emplea en las peticiones HTTP
  - Proporciona acceso a los datos de las cabeceras HTTP
  - Proporciona acceso a las cookies
  - Permite ver los parámetros pasados por el usuario
  - ... y todo sin tener que procesar nosotros la petición para obtener los datos (*datos de formulario*)

Métodos útiles

- En esta clase se tienen, entre otros, los métodos:
  - Para obtener nombres y valores de parámetros de una petición (cuidando mayúsculas y minúsculas)

```
Enumeration getParameterNames()
String getParameter(String nombre)
String getParameterValues(String nombre)
```

  - Para obtener los parámetros de una petición GET (devuelve una cadena que deberemos parsear nosotros)

```
String getQueryString()
```

Métodos útiles (II)

- Para obtener los datos enviados con POST, PUT o DELETE

```
BufferedReader getReader()
ServletInputStream getInputStream()
```

- Para obtener el método HTTP, la URI (parte de la URL tras el host, sin contar los datos del formulario) o el protocolo

```
String getMethod()
String getRequestURI()
String getProtocol()
```

ServletResponse y *HttpServletResponse*

- Los objetos *ServletResponse* se emplean para enviar en ellos una respuesta a una petición de un cliente

- En concreto, el subtipo *HttpServletResponse* se emplea para enviar respuestas HTTP

Métodos útiles

- Destacan los métodos para obtener el canal de salida donde escribir la respuesta:

```
Writer getWriter()
ServletOutputStream getOutputStream()
```

- Si se quieren indicar cabeceras, se deben indicar ANTES de obtener estos objetos

Métodos para atender peticiones

- Hemos visto que los métodos *doGet(...)* y *doPost(...)* atienden las peticiones GET y POST:

```
public void doGet(HttpServletRequest request,
                  HttpServletResponse response)
throws ServletException, IOException
{
   ... // Código del método
}

public void doPost(HttpServletRequest request,
                   HttpServletResponse response)
throws ServletException, IOException
{
   ... // Código del método, si es diferente a doGet
   ... // Si no, llamar a doGet y ya está
}
```

Qué hacer con la petición

- Acceder a valores de parámetros

```
String nombreUsuario = request.getParameter("nombre");
```

- Acceder a los parámetros de la petición y procesarlos como queramos

```
String peticion = request.getQueryString();
```

- Obtener un canal de entrada

```
BufferedReader r = request.getReader();
```

  - Esta no es buena idea si tomamos parámetros de peticiones. Se suele usar para transferencia de ficheros

Qué hacer con la respuesta

- Establecer valores de cabecera (antes que nada)

```
response.setContentType(“text/html”);
```

- Obtener el canal de salida por el que enviar la respuesta, y enviar contenido

```
PrintWriter out = response.getWriter();
out.println (“<HTML><BODY>...”);
```

- Redirigir a otra página

```
response.sendRedirect(“http://localhost:8080/pag.html”);
```

Procesamiento secuencial de peticiones

- Los servlets pueden gestionar múltiples peticiones concurrentemente
- Si, por algún recurso compartido u otra causa, queremos que hagan un procesamiento secuencial, podemos:
  - Utilizar bloques de código *synchronized* en el código del servlet necesario
  - Hacer que el servlet implemente la interfaz *SingleThreadModel* (sin métodos)

```
public class MiServlet extends HttpServlet
implements SingleThreadModel
{ ...
```

Manejo de formularios

- Los datos que se envían tras el ? en una petición GET o en una línea aparte en una petición POST se llaman *datos de formulario*.
- Los servlets procesan estos datos de forma automática y extraen la información útil, con métodos como:

```
Enumeration getParameterNames()
String getParameter(String nombre)
String getParameterValues(String nombre)
```

Ejemplo

- Para agrupar todo lo visto, tenemos el siguiente ejemplo donde:
  - Tenemos un formulario con una serie de campos
  - El formulario llama a un servlet que muestra en una página generada por él mismo los valores introducidos en el formulario

- Para probarlo en la plantilla, conectad con la URL siguiente:

```
http://localhost:8080/appforms/index_form.html
```

Pruebas de servlets con Cactus

- Ya vimos en Servidores web cómo configurar la aplicación para usar Cactus
- Veremos ahora cómo escribir un caso de prueba para un servlet, teniendo la aplicación ya configurada
- Lo primero, importar los paquetes necesarios

```
import org.apache.cactus.*;
import junit.framework.*;
```

- Después, hacer una clase en el mismo paquete que el servlet a probar (pero en la carpeta *test*), que herede de *ServletTestCase*

```
public class ClaseServletTest extends ServletTestCase
{
    ...
```

Métodos adicionales

- *setUp* y *tearDown*: Se usan como en JUnit, para ejecutar código antes de cada caso de prueba (inicializar variables, etc)
  - Se ejecutan en el lado del SERVIDOR: podemos acceder a los objetos de la petición/respuesta, y leer/modificar valores
- *testXXX*: el método principal de la prueba. Se ejecutan en el SERVIDOR. Podremos:
  - Instanciar la clase (servlet) a probar
  - Configurar objetos de petición y respuesta
  - Llamar al método a probar (*doGet*, por ejemplo)
  - Realizar aserciones (*assertXXX*) en los lugares donde queramos comprobar algo, como en JUnit.
- *beginXXX* y *endXXX*: equivalentes a *setUp* y *tearDown* pero...
  - Se ejecutan en el lado del CLIENTE, antes y después de la prueba *testXXX* correspondiente. Podemos usarlos para leer cookies, establecer cabeceras de petición, etc.

Ejemplo completo

```
import org.apache.cactus.*;
public class PruebaServletTest extends ServletTestCase {
    public void beginXXX(WebRequest theRequest) {
          // Inicializa los parámetros HTTP relacionados
          theRequest.setURL("jakarta.apache.org", "/mywebapp",
                                "/test/test.jsp", null, null);
          theRequest.addCookie("cookiename", "cookievalue");
    }
    public void testXXX() {
          //Servlet a probar
          PruebaServlet servlet = new PruebaServlet();
          servlet.init(config);
          // Llama al metodo a probar, por ejemplo, doGet
          try { servlet.doGet(request, response); } catch (Exception ex) {}
          // Realiza algunas asercion en el lado del servidor
          assertEquals("someValue", session.getAttribute("someAttribute"));
          assertEquals("jakarta.apache.org", request.getServerName());
    }
    public void endXXX(WebResponse theResponse) {
          // Aserta la respuesta HTTP
          Cookie cookie = theResponse.getCookie("someCookie");
          assertEquals("someValue2", cookie.getValue());
          assertEquals("some content here", theResponse.getText());
    }
}
```

Prueba de los casos

- Para probar los casos, primero debemos tener ejecutando la aplicación en el servidor (*Run As – Run on Server*)
- Después, ejecutamos la prueba como un caso de prueba JUnit (*Run As – JUnit test*)
  - También podemos utilizar un script de ant con tareas de cactus, como se vio en Servidores Web