Procesamiento de peticiones

Índice

Un servlet maneja peticiones de los clientes a través de su método **`service`**. Con él se pueden manejar peticiones HTTP (entre otras), reenviando las peticiones a los métodos apropiados que las manejan. Por ejemplo, una petición GET puede redirigirse a un método `doGet`. Veremos ahora los elementos principales que intervienen en una interacción vía HTTP.

1. Peticiones: HttpServletRequest

Como hemos visto anteriormente, los objetos **`ServletRequest`** se emplean para obtener información sobre la petición de los clientes. Más en concreto, el subtipo **`HttpServletRequest`** se utiliza en las peticiones HTTP. Proporciona acceso a los datos de las cabeceras HTTP, cookies, parámetros pasados por el usuario, etc, sin tener que parsear nosotros a mano los datos de formulario de la petición.

La clase dispone de muchos métodos, pero destacamos los siguientes:

- Para **obtener los valores de los parámetros** pasados por el cliente, se tienen los métodos:

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

  Con **`getParameterNames()`** se obtiene una lista con los nombres de los parámetros enviados por el cliente. Con **`getParameter()`** se obtiene el valor del parámetro de nombre `nombre`. Si un parámetro tiene varios valores (por ejemplo, si tenemos un array de cuadros de texto con el mismo nombre en un formulario), se pueden obtener todos separados con **`getParameterValues()`**. Los nombres de los parámetros normalmente sí distinguen mayúsculas de minúsculas, deberemos tener cuidado al indicarlos.

- Para **obtener la cadena de una petición GET**, se tiene el método:

```
String getQueryString()
```

  que devuelve todos los parámetros de la petición en una cadena, que deberemos parsear nosotros como nos convenga.

- Para **obtener datos de peticiones POST, PUT o DELETE**, se tienen los métodos:

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

  Con **`getReader()`** se obtiene un `BufferedReader` para peticiones donde esperemos recibir texto. Si esperamos recibir datos binarios, se debe emplear **`getInputStream()`**. Si lo que esperamos recibir son parámetros por POST igual que se haría por GET, es mejor utilizar los métodos `getParameterXXXX(...)` vistos antes.

- Para **obtener información sobre la línea de petición**, se tienen los métodos:

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

Con **`getMethod()`** obtenemos el comando HTTP solicitado (GET, POST, PUT, etc), con **`getRequestURI()`** obtenemos la parte de la URL de petición que está detrás del `host` y el puerto, pero antes de los datos del formulario. Con **`getProtocol()`** obtenemos el protocolo empleado (`HTTP/1.1`, `HTTP/1.0`, etc).

2. Respuestas: HttpServletResponse

Los objetos **`ServletResponse`** se emplean para enviar el resultado de procesar una petición a un cliente. El subtipo **`HttpServletResponse`** se utiliza en las peticiones HTTP. Proporciona acceso al canal de salida por donde enviar la respuesta al cliente.

La clase dispone de muchos métodos, pero destacamos:

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

Con **`getWriter()`** se obtiene un `Writer` para enviar texto al cliente. Si queremos enviar datos binarios, se debe emplear **`getOutputStream()`**.

Si queremos especificar información de cabecera, debemos establecerla ANTES de obtener el `Writer` o el `ServletOutputStream`. Hemos visto en algún ejemplo el método **`setContentType()`** para indicar el tipo de contenido. Veremos las cabeceras con más detenimiento más adelante.

3. Procesamiento de peticiones GET y POST

Como se ha visto anteriormente, el método **`doGet()`** se emplea para procesar peticiones GET. Para realizar nuestro propio procesamiento de petición, simplemente sobreescribimos este método en el servlet:

```
public void doGet(HttpServletRequest request,
                  HttpServletResponse response)
throws ServletException, IOException
{
        // ... codigo para una peticion GET
}
```

Podemos utilizar los métodos del objeto `HttpServletRequest` vistos antes. Así podremos, entre otras cosas:

- Acceder a elementos de la petición, como valores de parámetros:

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

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

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

- Obtener un canal de entrada (`Reader` o `InputStream`) con que leer los datos de la petición:

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

  Esta, sin embargo, no es una buena idea para tomar parámetros de peticiones u otras cosas. Se suele emplear sobre todo para transferencias de ficheros, pero hay que tener en cuenta que si obtenemos un canal de entrada, luego no podremos obtener parámetros u otros valores con métodos `getParameter()` y similares.
- etc.

También podemos utilizar los métodos del objeto **HttpServletResponse** para, entre otras cosas:

- Establecer valores de la cabecera (antes que cualquier otra acción sobre la respuesta):

```
response.setContentType("text/html");
```

- Obtener el canal de salida por el que enviar la respuesta:

```
PrintWriter out = response.getWriter();
out.println ("Enviando al cliente");
```

- Redirigir a otra página:

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

- etc.

De forma similar, el método **`doPost()`**, se emplea para procesar peticiones POST. Igual que antes, debemos sobreescribir este método para definir nuestro propio procesamiento de la petición:

```
public void doPost(HttpServletRequest request,
                   HttpServletResponse response)
throws ServletException, IOException
{
        // ... codigo para una peticion POST
}
```

Las posibilidades de los parámetros `HttpServletRequest` y `HttpServletResponse` son las mismas que para GET. Normalmente muchos servlets definen el mismo código para uno y otro método (hacen que `doPost()` llame a `doGet()` y definen allí el código, o al revés), pero conviene tenerlos separados para poder tratar independientemente uno y otro tipo de peticiones si se quiere.

3.1. Procesamiento secuencial de peticiones

Los servlets normalmente pueden gestionar múltiples peticiones de clientes

concurrentemente. Pero puede suceder que, si los métodos que definimos acceden a un recurso compartido, no nos interese que varios clientes accedan a dicho recurso simultáneamente. Para solucionar este problema, podemos definir bloques de código `synchronized`, o bien hacer que el servlet sólo atienda una petición cada vez.

Para esto último, lo único que hay que hacer es que el servlet, además de heredar de `HttpServlet`, implemente la interfaz **SingleThreadModel**. Esto no supone definir más métodos, simplemente añadimos el `implements` necesario al definir la clase Servlet, y ya está:

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

4. Manejo de formularios

Los datos que se envían como parámetros en una petición (tras el interrogante si es una petición GET, o por otro lado si es POST) se llaman **datos de formulario**. Una vez enviados estos datos como petición, ¿cómo se extraen en el servidor?

Si trabajáramos con CGI, los datos se tomarían de forma distinta si fuese una petición GET o una POST. Para una GET, por ejemplo, tendríamos que tomar la cadena tras la interrogación, y parsearla convenientemente, separando los bloques entre '&', y luego separando el nombre del parámetro de su valor a partir del '='. También hay que descodificar los valores: los alfanuméricos no cambian, pero los espacios se han convertido previamente en '+', y otros caracteres se convierten en '%XX%'.

Con servlets todo este análisis se realiza de forma automática. La clase `HttpServletRequest` dispone de métodos que devuelven la información que nos interesa ya procesada, e independientemente de si es una petición GET o POST. Hemos visto antes los métodos:

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

4.1. Ejemplo

Veamos un ejemplo: supongamos que tenemos este formulario:

```
<html>
<body>
<form action="/appforms/servlet/ejemplos.ServletForm">
        Valor 1: <input type="text" name="texto1">
        <br>
        Valor2:
```

```
        <select name="lista">
        <option name="lista" value="Opcion 1">Opcion 1</option>
        <option name="lista" value="Opcion 2">Opcion 2</option>
        <option name="lista" value="Opcion 3">Opcion 3</option>
        </select>
        <br>
        Valores 3:
        <br>
        <input type="text" name="texto2">
        <input type="text" name="texto2">
        <input type="text" name="texto2">

        <input type="submit" value="Enviar">
</form>
</body>
</html>
```

Al validarlo se llama al servlet `ServletForm`, que muestra una página HTML con los valores introducidos en los parámetros del formulario:

```
package ejemplos;

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

public class ServletForm extends HttpServlet
{
        // Metodo para GET

        public void doGet(HttpServletRequest request,
                          HttpServletResponse response)
        throws ServletException, IOException
        {
                response.setContentType("text/html");

                PrintWriter out = response.getWriter();

                // Mostramos los datos del formulario

                out.println ("<HTML>");
                out.println ("<BODY>");
                out.println ("<H1>Datos del formulario</H1>");
                out.println ("<BR>");

                String valor1 =
                        request.getParameter("texto1");
                String valor2 =
                        request.getParameter("lista");
                String[] valor3 =
                        request.getParameterValues("texto2");

                out.println ("Valor 1:" + valor1);
                out.println ("<BR>");
                out.println ("Valor 2:" + valor2);
                out.println ("<BR>");
                out.println ("Valor 3:");
                out.println ("<BR>");
```

```
                if (valor3 != null)
                        for (int i = 0; i < valor3.length; i++)
                        {
                                out.println (valor3[i]);
                                out.println ("<BR>");
                        }

                out.println ("</BODY>");
                out.println ("</HTML>");
        }

        // Metodo para POST

        public void doPost(HttpServletRequest request,
                           HttpServletResponse response)
        throws ServletException, IOException
        {
                doGet(request, response);
        }
}
```

Para probar el ejemplo que viene en las plantillas, cargamos la URL:

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

5. Cabeceras y códigos

Veremos a continuación cómo tratar las cabeceras HTTP de una petición y de una respuesta, así como los códigos de estado que emite un servidor Web ante una petición, y las variables CGI a las que podemos acceder.

5.1. Cabeceras de petición

Cuando se envía una petición HTTP, se pueden enviar, entre otras cosas, unas cabeceras con información sobre el navegador. Para leer estas cabeceras de una petición desde un servlet, se utiliza el método **`getHeader()`** del objeto **`HttpServletRequest`**.

```
String getHeader(String nombre)
```

El parámetro indica el nombre de la cabecera cuyo valor se quiere obtener. Devuelve el valor de la cabecera, o `null` si la cabecera no ha sido enviada en la petición.

Se tienen otros métodos, como:

```
Enumeration getHeaderNames()
Enumeration getHeaders(String nombre)
int getIntHeader(String nombre)
...
```

Con **`getHeaderNames()`** obtendremos todos los nombres de las cabeceras enviadas. Con **`getHeaders()`** obtendremos todos los valores de la cabecera de nombre dado. También hay métodos como **`getIntHeader()`** que devuelve el valor de una cabecera con un tipo

de dato específico (entero, en este caso). Los nombres de las cabeceras normalmente no distinguen mayúsculas de minúsculas.

Algunas cabeceras son de uso común, y tienen métodos específicos para obtener sus valores, como:

```
Cookie[] getCookies()
String getContentLength()
String getContentType()
...
```

Con **`getCookies()`**obtendremos todas las cookies de la petición (veremos las cookies con más detalle en otro tema). Con **`getContentLength()`** obtenemos el valor de la cabecera `Content-Length`, y con **`getContentType()`** el de la cabecera `Content-Type`.

5.2. Cabeceras de respuesta

En la respuesta de un servidor web a una petición también pueden aparecer cabeceras que informan sobre el documento servido o sobre el propio servidor. Podemos definir cabeceras de respuesta para enviar cookies, indicar la fecha de modificación, etc. Estas cabeceras deben establecerse ANTES de enviar cualquier documento, o antes de obtener el `PrintWriter` si es el caso.

Para enviar cabeceras, el método más general es **`setHeader()`** del objeto **`HttpServletResponse`**.

```
void setHeader(String nombre, String valor)
```

Al que se le pasan el nombre de la cabecera y el valor. Hay otros métodos útiles:

```
void setIntHeader(String nombre, int valor)
void addHeader(String nombre, String valor)
void addIntHeader(String nombre, int valor)
...
```

`setIntHeader()` o **`setDateHeader()`** se utilizan para enviar cabeceras de tipo entero o fecha. Los métodos **`add...()`** se emplean para añadir múltiples valores a una cabecera con el mismo nombre.

Algunas cabeceras tienen métodos específicos de envío, como:

```
void setContentType(String tipo)
void setContentLength(int tamaño)
void sendRedirect(String url)
void addCookie(Cookie cookie)
```

Con **`setContentType()`** se establece la cabecera `Content-Type` con el tipo MIME del documento. Con **`setContentLength()`** se indican los bytes enviados. Con **`sendRedirect()`** se selecciona la cabecera `Location`, y con ella se redirige a la página que le digamos. Finalmente, con **`addCookie()`** se establecen cookies (esto último ya lo veremos con más detalle en otro tema). Es recomendable utilizar estos métodos en lugar

del método `setHeader()` para la cabecera en cuestión.

5.3. Variables CGI

Las variables CGI son una forma de recoger información sobre una petición. Algunas se derivan de la línea de petición HTTP y de las cabeceras, otras del propio socket (como el nombre o la IP de quien solicita la petición), y otras de los parámetros de instalación del servidor (como el mapeo de URLs a los paths actuales).

Mostramos a continuación una tabla con las variables CGI, y cómo acceder a ellas desde servlets:

VARIABLE CGI	SIGNIFICADO	ACCESO DESDE SERVLETS
AUTH_TYPE	Tipo de cabecera Authorization (basic o digest)	`request. getAuthType()`
CONTENT_LENGTH	Número de bytes enviados en peticiones POST	`request. getContentLength()`
CONTENT_TYPE	Tipo MIME de los datos adjuntos	`request. getContentType()`
DOCUMENT_ROOT	Path del directorio raíz del servidor web	`getServletContext(). getRealPath("/")`
HTTP_XXX_YYY	Acceso a cabeceras arbitrarias HTTP	`request. getHeader("Xxx-Yyy")`
PATH_INFO	Información de path adjunto a la URL	`request. getPathInfo()`
PATH_TRANSLATED	Path mapeado al path real del servidor	`request. getPathTranslated()`
QUERY_STRING	Datos adjuntos para peticiones GET	`request. getQueryString()`
REMOTE_ADDR	IP del cliente que hizo la petición	`request. getRemoteAddr()`
REMOTE_HOST	Nombre del dominio del cliente que hizo la petición (o IP si no se puede determinar)	`request. getRemoteHost()`
REMOTE_USER	Parte del usuario en la cabecera Authorization (si se suministró)	`request. getRemoteUser`
REQUEST_METHOD	Tipo de petición (GET, POST, PUT, DELETE, HEAD, OPTIONS, TRACE)	`request. getMethod()`

SCRIPT_NAME	Path del servlet	`request.getServletPath()`
SERVER_NAME	Nombre del servidor web	`request.getServerName()`
SERVER_PORT	Puerto por el que escucha el servidor	`request.getServerPort()`
SERVER_PROTOCOL	Nombre y versión usada en la línea de petición (HTTP/1.0, HTTP/1.1 ...)	`request.getServerProtocol()`
SERVER_SOFTWARE	Información del servidor web	`getServletContext().getServerInfo()`

`request` se asume que es un objeto de tipo `HttpServletRequest`. Para obtener cualquiera de las variables antes mencionadas, sólo hay que llamar al método apropiado desde `doGet()` o `doPost()`.

5.4. Códigos de estado HTTP

Cuando un servidor web responde a una petición, en la respuesta aparece, entre otras cosas, un código de estado que indica el resultado de la petición, y un mensaje corto descriptivo de dicho código.

El envío de cabeceras de respuesta normalmente se planifica junto con el envío de códigos de estado, ya que muchos de los códigos de estado necesitan tener una cabecera definida. Podemos hacer varias cosas con los servlets manipulando las líneas de estado y las cabeceras de respuesta, como por ejemplo reenviar al usuario a otros lugares, indicar que se requiere un password para acceder a un determinado sitio web, etc.

Para enviar códigos de estado se emplea el método **`setStatus()`** de **`HttpServletResponse`**:

```
void setStatus(int estado)
```

Donde se le pasa como parámetro el código del estado. En la clase `HttpServletResponse` tenemos una serie de constantes para referenciar a cada código de estado. Por ejemplo, la constante:

```
HttpServletResponse.SC_NOT_FOUND
```

se corresponde con el código 404, e indica que el documento solicitado no se ha encontrado.

Existen otros métodos para gestión de mensajes de error:

```
void sendError(int codigo, String mensaje)
void sendRedirect(String url)
```

`sendError()` genera una página de error, con código de error igual a `codigo`, y con mensaje de error igual a `mensaje`. Se suele utilizar este método para códigos de error, y `setStatus()` para códigos normales.

`sendRedirect()` genera un error de tipo 302, envía una cabecera `Location` y redirige a la página indicada en `url`. Es mejor que enviar directamente el código, o hacer un `response.setHeader("Location", "http...")`, porque es más cómodo, y porque el servlet genera así una página con el enlace a la nueva dirección, para navegadores que no soporten redirección automática

Si queremos enviar un código en la respuesta, se tiene que especificar antes de obtener el objeto `PrintWriter`.

5.5. Ejemplos

5.5.1. Ejemplo de cabeceras de petición

El siguiente servlet muestra los valores de todas las cabeceras HTTP enviadas en la petición. Recorre las cabeceras enviadas y muestra su nombre y valor:

```
package ejemplos;

import java.io.*;
import java.util.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class ServletCabecerasPeticion
extends HttpServlet
{
   // Metodo para GET

   public void doGet(HttpServletRequest request,
                     HttpServletResponse response)
   throws ServletException, IOException
   {
        PrintWriter out = response.getWriter();

        // Mostramos las cabeceras enviadas
        // en la peticion

        out.println ("<HTML>");
        out.println ("<BODY>");
        out.println ("<H1>Cabeceras</H1>");
        out.println ("<BR>");

        Enumeration cabeceras = request.getHeaderNames();

        while (cabeceras.hasMoreElements())
        {
                String nombre = (String)(cabeceras.nextElement());
                out.println ("Nombre: " + nombre +
```

```
                    ", Valor: " + request.getHeader(nombre));
                out.println ("<BR><BR>");
        }

        out.println ("</BODY>");
        out.println ("</HTML>");
    }

    // Metodo para POST

    public void doPost(HttpServletRequest request,
                       HttpServletResponse response)
    throws ServletException, IOException
    {
        doGet(request, response);
    }
}
```

Se puede probar con este formulario, pinchando el botón:

```
<html>
<body>
<form action=
 "/appcab/servlet/ejemplos.ServletCabecerasPeticion">
        <input type="submit" value="Pulsa aqui">
</form>
</body>
</html>
```

5.5.2. Ejemplo de cabeceras de respuesta

El siguiente servlet espera un parámetro `accion` que puede tomar 4 valores:

- **primos**: El servlet tiene un hilo que está constantemente calculando números primos. Al elegir esta opción se envía una cabecera `Refresh` y recarga el servlet cada 10 segundos, mostrando el último número primo que ha encontrado.
- **redirect**: Utiliza un `sendRedirect()` para cargar la página que se indique como parámetro
- **error**: Utiliza un `sendError()` para mostrar una página de error, con un mensaje de error definido por el usuario, y un código de error a elegir de una lista.
- **codigo**: Envía un código de estado HTTP (con `setStatus()`), a elegir de entre una lista.

```
package ejemplos;

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

public class ServletCabecerasRespuesta
extends HttpServlet implements Runnable
{
    // Ultimo numero primo descubierto
    long primo = 1;
    // Hilo para calcular numeros primos
```

```
Thread t = new Thread(this);

// Metodo de inicializacion

public void init()
{
     t.start();
}

// Metodo para GET

public void doGet(HttpServletRequest request,
                  HttpServletResponse response)
throws ServletException, IOException
{
     String accion = request.getParameter("accion");

     if (accion.equals("primo"))
     {
             // Buscar el ultimo numero
             // primo y enviarlo

             response.setContentType("text/html");
             response.setHeader("Refresh", "10");
             PrintWriter out = response.getWriter();
             out.println ("<HTML><BODY>");
             out.println ("Primo: " + primo);
             out.println ("</BODY></HTML>");

     } else if (accion.equals("redirect")) {

             // Redirigir a otra pagina

             String url = request.getParameter("url");
             if (url == null)
                     url = "http://www.ua.es";
             response.sendRedirect(url);

     } else if (accion.equals("error")) {

             // Enviar error con sendError()

             int codigo = response.SC_NOT_FOUND;
             try
             {
                codigo = Integer.parseInt
                   (request.getParameter("codigoMensaje"));
             } catch (Exception ex) {
                codigo = response.SC_NOT_FOUND;
             }
             String mensaje = request.getParameter("mensaje");
             if (mensaje == null)
                mensaje = "Error generado";
             response.sendError(codigo, mensaje);

     } else if (accion.equals("codigo")) {

             // Enviar un codigo de error
```

```
                int codigo = response.SC_NOT_FOUND;
                try
                {
                   codigo = Integer.parseInt
                      (request.getParameter("codigo"));
                } catch (Exception ex) {
                   codigo = response.SC_NOT_FOUND;
                }
                response.setStatus(codigo);
        }
    }

    // Metodo para POST

    public void doPost(HttpServletRequest request,
                       HttpServletResponse response)
    throws ServletException, IOException
    {
         doGet(request, response);
    }

    ... el resto del codigo es para el hilo,
    para calcular numeros primos
    Puede consultarse en el fichero fuente,
    aqui se quita por simplificar
}
```

Se puede probar con este formulario, eligiendo la acción a realizar, introduciendo los parámetros necesarios en el formulario y pinchando el botón de Enviar Datos:

```
<html>
<body>
<form action=
"/appcab/servlet/ejemplos.ServletCabecerasRespuesta">

<table border="0">

<tr>
<td>
<input type="radio" name="accion" value="primo" selected>
Obtener ultimo numero primo
</td>
<td></td>
<td></td>
</tr>

<tr>
<td>
<input type="radio" name="accion" value="redirect">
Redirigir a una pagina
</td>
<td>
URL:
<input type="text" name="url" value="http://www.ua.es">
</td>
<td></td>
</tr>

<tr>
```

```
<td>
<input type="radio" name="accion" value="error">
Mostrar pagina de error
</td>
<td>
Mensaje:
<input type="text" name="mensaje"
value="Error generado por el usuario">
</td>
<td>
Codigo:
<select name="codigoMensaje">
<option name="codigoMensaje" value="400">400</option>
<option name="codigoMensaje" value="401">401</option>
<option name="codigoMensaje" value="403">403</option>
<option name="codigoMensaje" value="404" selected>404
</option>
</select>
</td>
</tr>

<tr>
<td>
<input type="radio" name="accion" value="codigo">
Enviar codigo de error
</td>
<td>
Codigo:
<select name="codigo">
<option name="codigo" value="200">200</option>
<option name="codigo" value="204">204</option>
<option name="codigo" value="404" selected>404</option>
</select>
</td>
<td></td>
</tr>

</table>

<input type="submit" value="Enviar Datos">

</form>
</body>
</html>
```

5.5.3. Ejemplo de autentificación

El siguiente servlet emplea las cabeceras de autentificación: envía una cabecera de autentificación si no ha recibido ninguna, o si la que ha recibido no está dentro de un conjunto de `Properties` predefinido, con logins y passwords válidos. En el caso de introducir un login o password válidos, muestra un mensaje de bienvenida.

Los logins y passwords están en un objeto `Properties`, definido en el método `init()`. Podríamos leer estos datos de un fichero, aunque por simplicidad aquí se definen como constantes de cadena.

Los datos de autentificación se envían codificados, y se emplea un objeto `sun.misc.BASE64Decoder` para descodificarlos y sacar el login y password.

```
package ejemplos;

import java.io.*;
import java.util.*;
import sun.misc.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class ServletPassword extends HttpServlet
{
   // Conjunto de logins y passwords permitidos
   Properties datos = new Properties();

   // Metodo de inicializacion

   public void init()
   {
        datos.setProperty("usuario1", "password1");
        datos.setProperty("usuario2", "password2");
   }

   // Metodo para GET

   public void doGet(HttpServletRequest request,
                     HttpServletResponse response)
   throws ServletException, IOException
   {
        response.setContentType("text/html");

        // Comprobamos si hay cabecera
        // de autorizacion

        String autorizacion = request.getHeader("Authorization");

        if (autorizacion == null)
        {
                // Enviamos el codigo 401 y
                // la cabecera para autentificacion

                response.setStatus(response.SC_UNAUTHORIZED);
                response.setHeader("WWW-Authenticate",
                   "BASIC realm=\"privileged-few\"");
        }
        else
        {
                // Obtenemos los datos del usuario
                // y comparamos con los almacenados

                // Quitamos los 6 primeros caracteres
                // que indican tipo de autentificación
                // (BASIC)

                String datosUsuario =
                    autorizacion.substring(6).trim();
```

```
                BASE64Decoder dec = new BASE64Decoder();

                String usuarioPassword = new String
                   (dec.decodeBuffer(datosUsuario));

                int indice = usuarioPassword.indexOf(":");

                String usuario =
                   usuarioPassword.substring(0, indice);

                String password =
                   usuarioPassword.substring(indice + 1);

                String passwordReal =
                   datos.getProperty(usuario);

                if (passwordReal != null &&
                    passwordReal.equals(password))
                {
                        // Mensaje de bienvenida

                        PrintWriter out = response.getWriter();
                        out.println ("<HTML><BODY>");
                        out.println ("OK");
                        out.println ("</BODY></HTML>");
                } else {

                        // Pedir autentificacion

                        response.setStatus
                           (response.SC_UNAUTHORIZED);
                        response.setHeader
                           ("WWW-Authenticate",
                           "BASIC realm=\"privileged-few\"");
                }
        }
   }

   // Metodo para POST

   public void doPost(HttpServletRequest request,
                      HttpServletResponse response)
   throws ServletException, IOException
   {
        doGet(request, response);
   }
}
```

Se puede probar cada ejemplo, respectivamente, con:

```
http://localhost:8080/appcab/inicioCabecerasPeticion.html
http://localhost:8080/appcab/inicioCabecerasRespuesta.html
http://localhost:8080/appcab/servlet/ejemplos.ServletPassword
```

Un ejemplo de login y password válidos para el tercer ejemplo es: login=`usuario1`, password=`password1`.