Componentes Web

Sesión 2 - Procesamiento de peticiones

Índice

Protocolo HTTP

- Comunicación entre cliente y servidor
  - El cliente solicita un documento del servidor
  - El servidor sirve el documento al cliente
- Mecanismo petición/respuesta
  - Se solicita el documento enviando un mensaje de petición HTTP al servidor
  - El servidor devuelve el documento requerido al cliente dentro de un mensaje HTTP de respuesta
  - Si el documento no puede ser servido, devolverá un mensaje HTTP de respuesta indicando el error producido
- Sin estado
  - Cada petición es independiente para el servidor

Mensaje de petición HTTP

- Lo envía el cliente al servidor HTTP
  - Solicita un recurso
- Se compone de:
  - Comando HTTP

    Compuesto por: `Comando URI Protocolo`

    Por ejemplo: `GET /index.htm HTTP/1.1`
  - Cabeceras

    Información sobre la petición

    La sección de cabeceras finaliza con una línea en blanco (`\r\n\r\n`)
  - Contenido adicional

    Bloque de datos de cualquier tipo

Comandos HTTP

- Comandos `GET` y `POST`
  - Se utilizan para solicitar un documento al servidor
  - `GET` proporciona los parámetros en la URI

```
GET /servlet/envia?msg=hola&usr=miguel HTTP/1.1
```

  - `POST` proporciona los parámetros en el bloque de contenido

- Otros comandos:
  - `OPTIONS`: Consulta opciones del servidor
  - `HEAD`: Solicita información sobre el recurso (no su contenido)
  - `PUT`: Guarda un recurso en el servidor
  - `DELETE`: Borra un recurso del servidor
  - `TRACE`: Muestra el camino seguido por la petición

Método HTTP GET

- Se realiza esta petición cuando pulsamos sobre un enlace en una página web
- Si queremos proporcionar parámetros tendremos que incluirlos en la misma URL

```
<a href="pag.jsp?id=123&nombre=pepe">Pulsa Aqui</a>
```

- También se realiza cuando utilizamos formularios con método GET

```
<form action="pag.jsp" method="GET" >
    <input type="text" name="id" value="123">
    <input type="text" name="nombre" value="pepe">
    <input type="submit" value="Enviar">
</form>
```

- Los datos introducidos en el formulario se envían en la URI

```
GET  /pag.jsp?id=123&nombre=pepe  HTTP/1.1
<cabeceras>
```

Método HTTP POST

- Se realiza cuando utilizamos un formulario con método POST

```
<form action="pag.jsp" METHOD=POST>
    <input type="text" name="id" value="123">
    <input type="text" name="nombre" value="pepe">
    <input type="submit" value="Enviar">
</form>
```

- Los parámetros se envían en el bloque de contenido

```
POST  /pag.jsp  HTTP/1.1
<cabeceras>

id=123&nombre=pepe
```

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)
```

  - Sirve tanto para parámetros enviados por GET como por POST

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

```
String getQueryString()
```

Métodos útiles (II)

- Para obtener los datos crudos enviados con POST o PUT

```
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()
```

Mensaje de respuesta HTTP

- El servidor nos responderá con un mensaje HTTP de respuesta

- Este mensaje se compone de:
  - Código de estado:

    Indica si se ha procesado correctamente o si ha habido un error

    Ejemplo: `HTTP/1.1 200 OK`
  - Cabeceras

    Información sobre el recurso y sobre el servidor

    Se definen de la misma forma que las de la petición
  - Contenido

    En el bloque de contenido se incluye el recurso devuelto, si se ha devuelto alguno

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:

```
PrintWriter          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 {
    // ... codigo para una peticion GET
}

public void doPost(HttpServletRequest request,
                   HttpServletResponse response)
              throws ServletException, IOException {
    // ... codigo para una peticion POST ...
    // ... si es diferente a doGet, si no, llamar a doGet
}
```

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 query = 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 ("Enviando al cliente");
```

- Redirigir a otra página

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

Cabeceras de la petición HTTP

- Envían información sobre
  - El agente de usuario (navegador)
  - La petición realizada

- Algunas cabeceras estándar son:

`Accept-Language`	Idiomas aceptados
`Host`	Host y puerto indicado en la URL (requerido)
`If-Modified-Since`	Sólo se desea el documento si ha sido modificado tras esta fecha
`User-Agent`	Tipo de cliente que realiza la petición

- Por ejemplo, según el idioma especificado en la petición, algunos servidores podrán devolver el documento en dicho idioma

Obtención de cabeceras de petición

- El objeto `HttpServletRequest` de la petición tiene un método `getHeader( )` para acceder a valores de cabeceras

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

- Se devuelve el valor de la cabecera `nombre` o `null` si no se ha encontrado la cabecera

- Se tienen otros métodos alternativos como:

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

Obtención de cabeceras concretas

- Se tienen algunos métodos para obtener cabeceras concretas, como:

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

Cabeceras de la respuesta HTTP

- El servidor también puede enviar cabeceras en la respuesta con información sobre
  - El documento devuelto
  - Las características del servidor
- Algunas cabeceras estándar de la respuesta son:

`Content-Length`	Longitud del contenido (en bytes)
`Content-Type`	Tipo MIME del contenido
`Last-Modified`	Fecha de modificación del documento

- Podemos establecer estas cabeceras también desde la cabecera del código HTML de nuestro documento:

```
<META HTTP-EQUIV="Cabecera" CONTENT="Valor">
```

Establecer cabeceras de respuesta

- El objeto `HttpServletResponse` tiene un método `setHeader( )` para establecer valores de cabeceras

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

- Se tienen otros métodos alternativos como:

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

- Con los métodos `add(...)` podemos añadir múltiples valores a una cabecera con el mismo nombre

Establecer cabeceras concretas

- Se tienen algunos métodos para establecer cabeceras concretas, como:

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

Qué son las variables CGI

- Las variables CGI son una forma de recoger información de una petición
- Se derivan de la línea de la petición, cabeceras, el socket, parámetros del servidor, etc.
- Algunos ejemplos:
  - `CONTENT_LENGTH`: número de bytes enviados
  - `PATH_INFO`: información del path junto a la URL
  - `REMOTE_ADDR`: IP del cliente que hizo la petición
  - `REQUEST_METHOD`: tipo de petición (GET, POST...)
  - `SERVER_PORT`: puerto por el que escucha el servidor

Acceso a las variables CGI

- El objeto `HttpServletRequest` y el `ServletContext` del servlet contienen métodos para acceder a las variables CGI:
  - `request.getContentLength()` para `CONTENT_LENGTH`
  - `request.getPathInfo()` accede a `PATH_INFO`
  - `request.getRemoteAddr()` accede a `REMOTE_ADDR`
  - `request.getMethod()` accede a `REQUEST_METHOD`
  - `request.getServerPort()` accede a `SERVER_PORT`
  - ... etc

Código de estado de la respuesta HTTP

- Indica el resultado de la petición
- Encontramos varios grupos de códigos:
  - `1XX`: Códigos de información
  - `2XX`: Códigos de aceptación
    - `200 OK`: Se ha servido correctamente
    - `204 No content`: No hay contenido nuevo
  - `3XX`: Redirecciones, el documento ha sido movido
  - `4XX`: Errores en la petición
    - `400 Bad request`: El mensaje de petición tiene sintaxis errónea
    - `401 Unauthorized`: El usuario no tiene permiso
  - `5XX`: Errores en el servidor
    - `500 Internal Server Error`: Error interno del servidor

Envío de códigos

- Para enviar un código de estado, el objeto `HttpServletResponse` tiene el método `setStatus( )`

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

- Se tienen en la clase varias constantes para representar distintos estados

```
setStatus(HttpServletResponse.SC_NOT_FOUND);    //Codigo 404
```

- Algunos métodos se usan para enviar códigos específicos, porque envían también la cabecera necesaria

```
void sendError(int codigo, String mensaje)      // Código 4XX o 500
void sendRedirect(String url)                   // Código 302
```

Peticiones e hilos

- Un mismo servlet puede atender varias peticiones concurrentemente
- Cada petición tendrá su propio hilo de ejecución
- Puede producir problemas de concurrencia
  - Utilizar bloques synchronized cuando sea necesario
- Hay un número limitado de hilos disponibles para atender peticiones
  - Esto se conoce como un *pool* de hilos
  - Cuando se agote no se podrán atender nuevas peticiones

Procesamiento asíncrono

- Si en una petición se hace una operación de larga duración bloqueará el hilo de la petición
  - Esto hará que los hilos del *pool* se agoten más rápidamente

- Para conseguir una mayor escalabilidad podemos realizar estas peticiones de forma asíncrona
  - Movemos la operación a otro hilo y liberamos el hilo de la petición para que atienda otras peticiones

- Operaciones de larga duración son por ejemplo:
  - Consultas a base de datos
  - Acceso a servicios web remotos
  - Operaciones que dependan del suceso de algún evento o de la interacción del usuario

Implementación de procesamiento asíncrono

- Activar procesamiento asíncrono en el servlet

```
@WebServlet(urlPatterns={"/ServletAsincrono"}, asyncSupported=true)
public class AsincronoServlet extends HttpServlet {
    ...
}
```

- Procesamos la petición de forma asíncrona

```
public void doGet(HttpServletRequest request, HttpServletResponse response) {
  ...
  final AsyncContext ac = request.startAsync();
  ac.start(new Runnable() {
    public void run() {
      HttpServletRequest response = ac.getRequest();
      // Procesa la petición
      ...
      HttpServletResponse response = ac.getResponse();
      // Escribe la respuesta
      ...
      ac.complete();
    }
  }
}
```

Proporciona otro hilo para procesar la petición

Vuelca la respuesta al cliente

¿Preguntas?