élogos

GENERACIÓN DE PÁGINAS CON SERVLETS

élogos

ÍNDICE

GENERACIÓN DE PÁGINAS CON SERVLETS

1. Ciclo de vida de un servlet

Instanciación de un servlet

Cuando se realiza una primera petición de un Servlet desde el cliente, por ejemplo:

http://www.servidor.com/aplicacion/servletejemplo.

El contenedor Web crea una instancia del Servlet asociado a la URL /servletejemplo.

A partir de ese momento, comenzarán a ejecutarse una serie de métodos sobre el objeto para atender a la petición, la instancia se mantiene en memoria para poder atender a las siguientes peticiones que se realicen de dicho servlet.

Métodos del ciclo de vida

Después de crear la instancia del Servlet, el contenedor Web llama a los siguiente métodos del mismo:

init. Se ejecuta inmediatamente después de crear la instancia del servlet. Dado que la instancia es compartida por todos los clientes, este método se ejecutará una sola vez durante la vida del servlet. Tiene dos posibles formatos:

public void init()

public void init(ServletConfig config)

service

Conocido como método de servicio, se ejecuta con cada petición del servlet. En el caso de la primera petición, se ejecutará después de init. Es el método más importante del servlet, en él el programador deberá definir las acciones que el servlet tenga que realizar cuando es solicitado desde la capa cliente.

destroy

Se ejecuta justo antes de que la instancia sea destruida. Esto suele ocurrir cuando el contenedor Web se va a detener o, por algún motivo, decide destruir la instancia del servlet.

La Interfaz Servlet y la clase HttpServlet

Para crear servlets se debe tener instalado el fichero servlet.jar dentro del CLASSPATH. Esto se hará automáticamente si utilizas un entorno de desarrollo. Los paquetes de clases principales son:

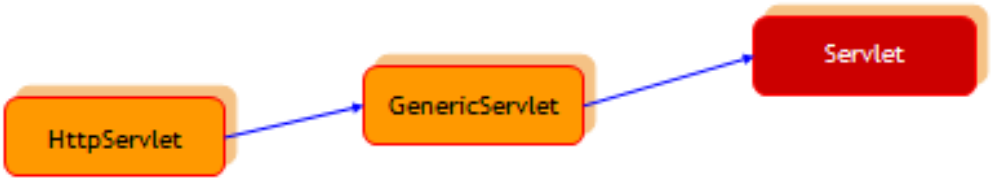
import javax.servlet.*; Para trabajar con servlets genéricos.

import javax.servlet.http.*; Para trabajar con servlets http.

Un servlet debe implementar la **interfaz Servlet**:

javax.servlet.Servlet

En el caso de servlets HTTP, será más cómodo heredar la **clase** HttpServlet:



javax.servlet.HttpServlet javax.servlet.GenericServlet

Métodos de servicio

La clase del Servlet no tiene función main, la programación se realizará en los llamados métodos de servicio que, como hemos indicado, son invocados por el Servlet Engine como respuesta a una solicitud del servlet.

El método de **servicio** definido en HttpServlet es el siguiente:

El objeto **request** contiene los datos de la petición, mientras que **response** nos permitirá generar la respuesta. En la versión original del método, se realiza una llamada a otros **dos métodos** definidos también en la misma clase.

Estos métodos se ejecutan cuando el Servidor envía una **petición GET** (doGet) o **POST** (doPost).

```
protected void doGet (HttpServletRequest request,
HttpServletResponse response)
throws ServletException, IOException
```

```
protected void doPost (HttpServletRequest request,
HttpServletResponse response)
throws ServletException, IOException
```

2. Generación de una respuesta

La interfaz HttpServletResponse

Al crear un servlet que genere una página Web de forma dinámica, tendremos que crear una clase que herede HttpServlet y sobrescribir los métodos de servicio: service(), si las operaciones se deben ejecutar con cualquier tipo de petición o doGet/doPost, si solo nos interesa responder a una petición específica.

Utilizando el objeto HttpServletResponse, podremos construir la cadena de texto HTML que queremos enviar al cliente

La interfaz **HttpServletResponse** tiene los siguientes métodos para poder generar la respuesta:

void setContentType (String type)

PrintWriter getWriter()

El método getWriter()

Esté método devolverá un objeto que nos permitira generar la respuesta del Servlet:

PrintWriter es una clase del paquete java.io, muy similar a PrintStream, que dispone de los métodos **print()** y **println()** para escribir el texto de respuesta.

Este objeto manda los caracteres que tenga implementados la máquina (Sistema Operativo).

Antes de realizar la escritura del texto, se deberá indicar a través de setContentType() el tipo de respuesta a generar.

Ejemplo

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class SaludoServlet extends HttpServlet {
public void service(HttpServletRequest peticion,
HttpServletResponse respuesta)
throws ServletException, IOException
{
respuesta.setContentType("text/html");
PrintWriter out = respuesta.getWriter();
out.println ("<HTML> <HEAD> <TITTLE>");
out.println ("PRIMER SERVLET </TITTLE> </HEAD>");
```

```
out.println ("<BODY>");
out.println ("<I>Bienvenidos a mi servlet </I>");
out.println ("</BODY> </HTML>");
out.close();
} // fin del método service
} // fin de clase.
```

La posibilidad de utilizar instrucciones de control y variables en la creación del texto de respuesta, ofrece grandes posibilidades a la hora de generar de forma dinámica las páginas.

La siguiente imagen corresponde a una página que muestra la tabla de multiplicar de todos los números del 1 al 10:

```
import java.io.*;
import java.io.PrintWriter;
import javax.servlet.*;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.*;
@WebServlet(name = "Tabla", urlPatterns = {"/tabla"})
public class Tabla extends HttpServlet {
   protected void service(HttpServletRequest request, HttpServletResponse response)
         throws ServletException, IOException {
      response.setContentType("text/html;charset=UTF-8");
      PrintWriter out = response.getWriter();
      try {
         out.println("<html>");
         out.println("<head><title>Tabla multiplicar</title></head>");
         out.println("<body>");
         out.println("<table border='1'>");
         for(int i=1;i<=10;i++){
            out.println("<tr>");
            for(int k=1;k<=10;k++){
               out.println("<td>"+i*k+"</td>");
            }
            out.println("</tr>");
         }
         out.println("</table></body></html>");
       } finally {
         out.close();
      }
   }
}
```

3. Códigos de estado HTTP 1.1. Cabecera de respuesta

Deberíamos ser cuidadosos al utilizar los códigos de estado que están disponibles sólo en **HTTP 1.1**, ya que muchos navegadores sólo soportan **HTTP 1.0.**

Si tenemos que usar códigos de estado específicos para **HTTP 1.1**, en la mayoría de los casos querremos chequear explícitamente la versión **HTTP** de la **petición** o reservarlo para situaciones donde no importe el significado de la cabecera **HTTP 1.0.**

Además tenemos cabeceras de **respuesta** más comunes y sus significados.

Código de Estado	Mensaje Asociado	Significado
100	Continue	Continúa con petición parcial (nuevo en HTTP 1.1)
101	Switching Protocols	El servidor cumplirá con la cabecera Upgrade y cambiará a un protocolo diferente. (Nuevo en HTTP 1.1)
200	OK	Todo está bien; los documentos seguidos por peticiones GET y POST. Esto es por defecto para los Servlets, si no usamos setStatus, obtendremos esto.
201	Created	El servidor crea un documento; la cabecera Location indica la URL.
202	Accepted	La petición se está realizando, el proceso no se ha completado.
203	Non-Authoritative Information	El documento está siendo devuelto normalmente, pero algunas cabeceras de respuesta podrían ser incorrectas porque se está usando una copia del documento (Nuevo en HTTP 1.1)
204	No Content	No hay un documento nuevo, el navegador continúa mostrando el documento anterior. Esto es útil si el usuario recarga periódicamente una página y podemos determinar que la página anterior ya está actualizada. Sin embargo, esto no funciona para páginas que se recargan automáticamente mediante cabeceras de respuesta Refresh o su equivalente <META HTTP-EQUIV="Refresh" ...>,ya que al devolver este código de estado se pararán futuras recargas.

205	Reset Content	No hay documento nuevo, pero el navegador debería resetear el documento. Usado para forzar al navegador a borrar los contenidos de los campos de un formulario CGI (Nuevo en HTTP 1.1)
206	Partial Content	El cliente envía una petición parcial con una cabecera Range, y el servidor la ha completado. (Nuevo en HTTP 1.1)
300	Multiple Choices	El documento pedido se puede encontrar en varios sitios. Serán listados en el documento devuelto. Si el servidor tiene una opción preferida, debería listarse en la cabecera de respuesta Location .
301	Moved Permanently	El documento pedido está en algún lugar, y la URL se da en la cabecera de respuesta Location. Los navegadores deberían seguir automáticamente el enlace a la nueva URL.
302	Found	Similar a 301, excepto que la nueva URL debería ser interpretada como reemplazada temporalmente, no permanentemente. Observa: el mensaje era "Moved Temporarily" en HTTP 1.0, y la constante en HttpServletResponse es SC_MOVED_TEMPORARILY, no SC_FOUND. Cabecera muy útil, ya que los navegadores siguen automáticamente el enlace a la nueva URL. Este código de estado es tan útil que hay un método especial para ella, sendRedirect. Usar response.sendRedirect(url) tiene unpar de ventajas sobre hacer response.setStatus(response.SC_MOVED_TEMPORARILY) y response.setHeader("Location", url). Primero, es más fácil. Segundo, con sendRedirect, el servlet automáticamente construye una página que contiene el enlace (para mostrar a los viejos navegadores que no siguen las redirecciones automáticamente). Finalmente, sendRedirect puede manejar URLs relativas, automáticamente las traducen a absolutas. Observa que este código de estado es usado algunas veces de forma intercambiada con 301. Por ejemplo, si erróneamente pedimos http://host/~user (olvidando la última barra), algunos servidores enviarán 301 y otros 302. Técnicamente, se supone que los navegadores siguen automáticamente la redirección si la petición original era GET. Puedes ver la cabecera 307 para más detalles.
303	See Other	Igual que 301/302, excepto que si la petición original era POST, el documento redirigido (dado en la cabecera Location) debería ser recuperado mediante GET. (Nuevo en HTTP 1.1)
304	Not Modified	El cliente tiene un documento en el caché y realiza una petición condicional (normalmente suministrando una cabecera If-Modified-Since indicando que sólo quiere documentos más nuevos que la fecha especificada). El servidor

		quiere decirle al cliente que el viejo documento del caché todavía está en uso.
305	Use Proxy	El documento pedido debería recuperarse mediante el proxy listado en la cabecera Location. (Nuevo en HTTP 1.1)
307	Temporary Redirect	Es idéntica a 302 ("Found" o "Temporarily Moved"). Fue añadido a HTTP 1.1 ya que muchos navegadores siguen erróneamente la redirección de una respuesta 302 incluso si el mensaje original fue un POST, y sólo se debe seguir la redirección de una petición POST en respuestas 303. Esta respuesta es algo ambigua: sigue el redireccionamiento para peticiones GET y POST en el caso de respuestas 303, y en el caso de respuesta 307 sólo sigue la redirección de peticiones GET. Nota: por alguna razón no existe una constante en HttpServletResponse que corresponda con este código de estado. (Nuevo en HTTP 1.1)
400	Bad Request	Mala Sintaxis de la petición.
401	Unauthorized	El cliente intenta acceder a una página protegida por password sin las autorización apropiada. La respuesta debería incluir una cabecera WWW-Authenticate que el navegador debería usar para mostrar la caja de diálogo usuario/password, que viene de vuelta con la cabecera Authorization.
403	Forbidden	El recurso no está disponible, si importa la autorización. Normalmente indica la falta de permisos de fichero o directorios en el servidor.
404	Not Found	No se pudo encontrar el recurso en esa dirección. Ésta la respuesta estándard "no such page". Es tan común y útil esta respuesta que hay un método especial para ella en HttpServletResponse: sendError(message). La ventaja de sendError sobre setStatus es que, con sendErr, el servidor genera automáticamente una página que muestra un mensaje de error.
405	Method Not Allowed	El método de la petición (GET, POST, HEAD, DELETE, PUT, TRACE, etc.) no estaba permitido para este recurso particular. (Nuevo en HTTP 1.1)
406	Not Acceptable	El recurso indicado genera un tipo MIME incompatible con el especificado por el cliente mediante su cabecera Accept. (Nuevo en HTTP 1.1)
407	Proxy Authentication Required	Similar a 401, pero el servidor proxy debería devolver una cabecera Proxy-Authenticate. (Nuevo en HTTP 1.1)
408	Request	El cliente tarda demasiado en enviar la petición. (Nuevo en HTTP 1.1)

	Timeout	
409	Conflict	Usualmente asociado con peticiones PUT. Usado para situaciones como la carga de una versión incorrecta de un fichero. (Nuevo en HTTP 1.1)
410	Gone	El documento se ha ido, no se conoce la dirección de reenvío. Difiere de la 404 en que se sabe que el documento se ha ido permanentemente, no sólo está indisponible por alguna razón desconocida como con 404. (Nuevo en HTTP 1.1)
411	Length Required	El servidor no puede procesar la petición a menos que el cliente envíe una cabecera Content-Length. (Nuevo en HTTP 1.1)
412	Precondition Failed	Alguna condición previa especificada en la petición era falsa (Nuevo en HTTP 1.1)
413	Request Entity Too Large	El documento pedido es mayor que lo que el servidor quiere manejar ahora. Si el servidor cree que puede manejarlo más tarde, debería incluir una cabecera Retry-After. (Nuevo en HTTP 1.1)
414	Request URI Too Long	La URI es demasiado larga. (Nuevo en HTTP 1.1)
415	Unsupported Media Type	La petición está en un formato desconocido. (Nuevo en HTTP 1.1)
416	Requested Range Not Satisfiable	El cliente incluyó una cabecera Range no satisfactoria en la petición. (Nuevo en HTTP 1.1)
417	Expectation Failed	No se puede conseguir el valor de la cabecera Expect. (Nuevo en HTTP 1.1)
500	Internal Server Error	Mensaje genérico "server is confused". Normalmente es el resultado de programas CGI o servlets que se quedan colgados o retornan cabeceras mal formateadas.
501	Not Implemented	El servidor no soporta la funcionalidad de rellenar peticiones. Usado, por ejemplo, cuando el cliente envía comandos como PUT que el cliente no soporta.
502	Bad Gateway	Usado por servidores que actúan como proxies o gateways. Indica que el servidor inicial obtuvo una mala respuesta desde el servidor remoto.

503	Service Unavailable	El servidor no puede responder debido a mantenimiento o sobrecarga. Por ejemplo, un servlet podría devolver esta cabecera si algún almacén de threads o de conexiones con bases de datos están llenos. El servidor puede suministrar una cabecera Retry-After.
504	Gateway Timeout	Usado por servidores que actúan como proxies o gateways. Indica que el servidor inicial no obtuvo una respuesta a tiempo del servidor remoto. (Nuevo en HTTP 1.1)
505	HTTP Version Not Supported	El servidor no soporta la versión de HTTP indicada en la línea de petición. (Nuevo en HTTP 1.1)

Establecimiento de valores

A través de los métodos de la interfaz HttpServletResponse podemos establecer los valores, tanto de los códigos de estado HTTP como de los diferentes encabezados que queramos incluir en la respuesta. Estos métodos son:

- **setHeader (String name, String value).** Establece el valor del encabezado cuyo nombre se especifica en el primer parámetro.
- **setStatus (int codigo).** Establece el código de estado HTTP, en caso de que no se haya producido un error. (como ok, o moved temporarily). Los códigos de estado están definidos en una serie de constantes dentro de HttpServletResponse.
- **sendError (int codigo).** Establece el código de estado HTTP, en caso de que se haya producido un error.