



Servlets y JSP

- Sesión 1: Introducción a los servlets



Puntos a tratar

- Concepto de servlet
- Configuración de servlets en aplicaciones web
- Ejemplos básicos de servlets
- Logging en servlets



Definición de servlet

- Un *servlet* es un programa Java que se ejecuta en un servidor web y construye o sirve páginas web.
- Permite la construcción dinámica de páginas, en función de determinados parámetros de entrada
- Más *sencillo* que un CGI, más *eficiente* (se arranca un hilo por petición, y no un proceso entero), más *potente* y más *portable*.



Recursos de servlets y JSP

- Servlets y JSP son dos conceptos muy interrelacionados
- Para trabajar con ellos se necesita:
 - Un *servidor Web* con soporte para servlets / JSP (*contenedor* de servlets y JSP: Tomcat, WebLogic...)
 - Las *librerías* o clases necesarias (proporcionadas por el servidor)
 - Recomendable también la *documentación de la API* de servlets / JSP
- Son útiles las direcciones:
 - <http://java.sun.com/j2ee>
 - <http://java.sun.com/products/jsp>
 - <http://java.sun.com/products/servlets>



Arquitectura del paquete servlet

- En el paquete *javax.servlet* tenemos toda la infraestructura para trabajar con servlets
- El elemento central es la interfaz *Servlet*
- La clase *GenericServlet* es una clase abstracta que la implementa para un servlet genérico independiente del protocolo
- La clase *HttpServlet* en el paquete *javax.servlet.http* hereda de la anterior para definir un servlet vía web utilizando HTTP

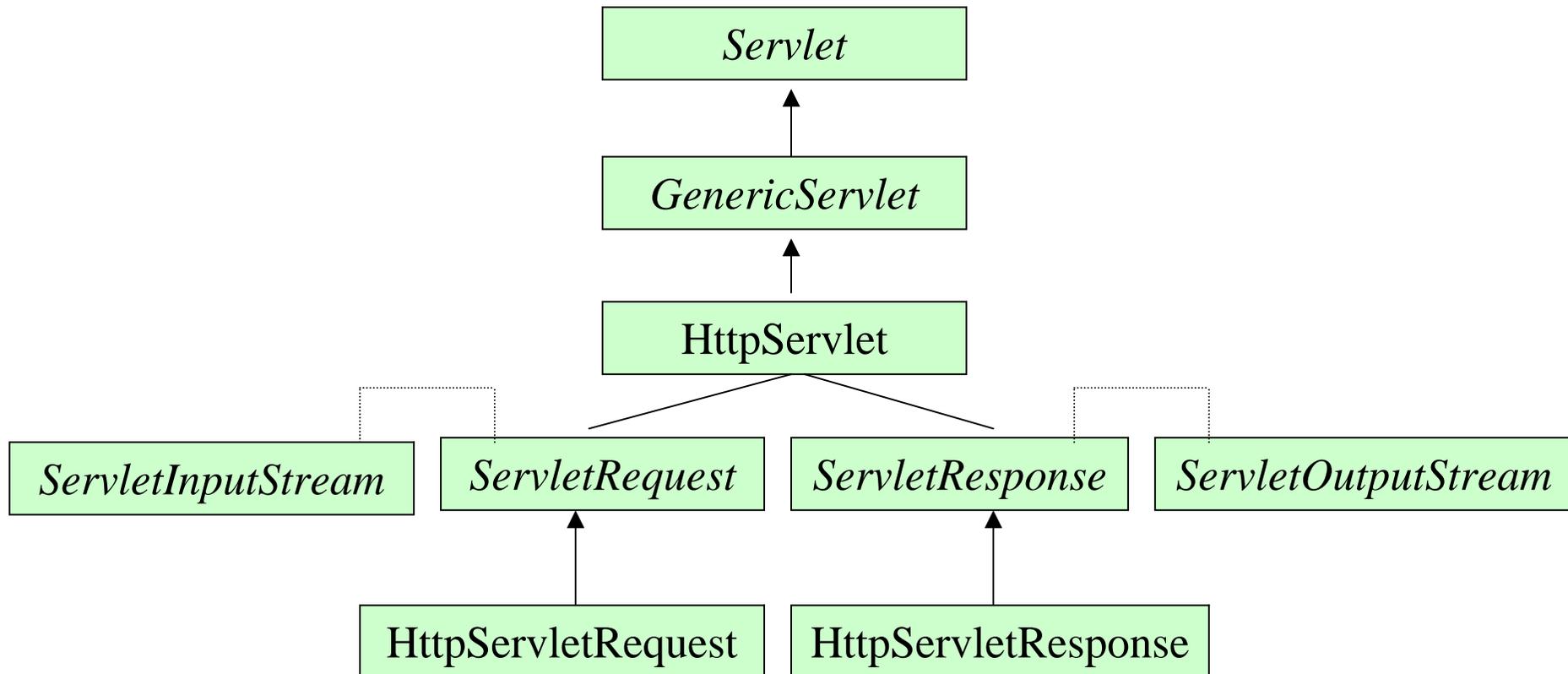


Arquitectura del paquete servlet

- Cuando un servlet recibe una petición de un cliente, se reciben dos objetos:
 - Un objeto *ServletRequest* con los datos de la petición (información entrante: parámetros, protocolo, etc)
 - Se puede obtener un *ServletInputStream* para leer los datos como un stream de entrada
 - La subclase *HttpServletRequest* procesa peticiones HTTP
 - Un objeto *ServletResponse* donde se colocarán los datos de respuesta del servlet ante la petición
 - Se puede obtener un *ServletOutputStream* o un *Writer* para escribir esos datos en la salida
 - La subclase *HttpServletResponse* trata respuestas HTTP



Arquitectura del paquete servlet





Ciclo de vida de un servlet

- Todos los servlets tienen el mismo ciclo de vida:
 - El servidor carga e inicializa el servlet
 - El servlet procesa N peticiones
 - El servidor destruye el servlet
- *Inicialización*: para tareas que se hagan una sola vez al iniciar el servlet

```
public void init() throws ServletException  
{ ... }
```

```
public void init(ServletConfig conf) throws ServletException  
{  
    super.init(conf);  
    ...  
}
```



Ciclo de vida de un servlet

- *Procesamiento de peticiones*: cada petición llama al método `service()`

```
public void service(HttpServletRequest request,  
                    HttpServletResponse response)  
    throws ServletException, IOException
```

- Según el tipo de petición, llama a uno de los métodos (todos con los mismos parámetros y excepciones que `service()`):

```
public void doGet(...)  
public void doPost(...)  
public void doPut(...)  
public void delete(...)  
public void doOptions(...)  
public void doTrace(...)
```



Ciclo de vida de un servlet

- *Destrucción*: método *destroy()*
`public void destroy() throws ServletException`
- Se debe deshacer todo lo construido en *init()*
- Se llama a este método cuando todas las peticiones han concluido, o cuando ha pasado un determinado tiempo (en este caso, se debe controlar por código que se destruya cuando debe)



Estructura básica de un servlet

```
import javax.servlet.*;
import javax.servlet.http.*;

public class ClaseServlet extends HttpServlet {

    public void doGet(HttpServletRequest req,HttpServletResponse res)
    throws ServletException, IOException {
        ... // CODIGO PARA UNA PETICION GET
    }

    public void doPost(HttpServletRequest req,HttpServletResponse res)
    throws ServletException, IOException {
        ... // CODIGO PARA UNA PETICION POST
    }
}
```

NOTA: esta es la estructura más común de un servlet. Además, se puede incluir cualquiera de los métodos (*init()*, *destroy()*, *doPut(...)*, etc, vistos antes



Llamada directa de servlets

- Para utilizar un servlet en una aplicación web, se coloca en el directorio *WEB-INF/classes*, con su estructura de paquetes y subpaquetes
- Después, podemos llamar al servlet directamente con:

```
http://host:puerto/<dir-aplicacion>/servlet/<nombre-servlet>
```

- O si lo hemos colocado en el *root* del servidor:

```
http://host:puerto/servlet/<nombre-servlet>
```

- Por ejemplo:

```
http://localhost:8080/miapp/servlet/paquete1.subpaquete1.MiServlet
```



Mapeado de servlets y JSP

- Otra opción es incluir en el fichero descriptor un nombre identificativo del servlet o la página JSP:

```
<servlet>
```

```
  <servlet-name>nombre</servlet-name>
```

```
  <servlet-class>unpaquete.ClaseServlet</servlet-class>
```

```
</servlet>
```

```
<servlet>
```

```
  <servlet-name>nombre2</servlet-name>
```

```
  <jsp-file>/mipagina.jsp</servlet-class>
```

```
</servlet>
```

- Con lo que podremos llamar al servlet o página:

```
http://host:puerto/<dir>/servlet/unpaquete.ClaseServlet
```

```
http://host:puerto/<dir>/servlet/nombre
```

```
http://host:puerto/<dir>/mipagina.jsp
```

```
http://host:puerto/<dir>/nombre2
```



Asignar URLs a servlets o páginas JSP

- Podemos mapear una URL concreta con un nombre de servlet o página JSP (tras `<servlet>`):

```
<servlet-mapping>  
    <servlet-name>nombre</servlet-name>  
    <url-pattern>/ejemploservlet</url-pattern>  
</servlet-mapping>
```

- Con lo que podremos llamar al servlet o página:

```
http://host:puerto/ejemploservlet
```

- También podemos usar comodines para mapear un conjunto de direcciones a un servlet o página JSP:

```
<servlet-mapping>  
    <servlet-name>nombre</servlet-name>  
    <url-pattern>/ejemploservlet/*.jsp</url-pattern>  
</servlet-mapping>
```



Parámetros de inicio en servlets y JSP

- Al asignar un nombre a un servlet o página JSP podemos definirle uno o más parámetros de inicio:

```
<servlet>  
  <servlet-name>nombre</servlet-name>  
  <servlet-class>ClaseServlet</servlet-class>  
  <init-param>  
    <param-name>param1</param-name>  
    <param-value>valor1</param-value>  
  </init-param>  
</servlet>
```

- Si llamamos al servlet por su nombre o por su *servlet-mapping*, podemos en su código acceder a estos parámetros con:

```
String s = getServletConfig().getInitParameter("param1");
```



Deshabilitar servlets

- Podemos deshabilitar la llamada al alias *servlet* en cada aplicación Web añadiendo en el descriptor:

```
<servlet>
  <servlet-name>error</servlet-name>
  <servlet-class>ErrorServlet</servlet-class>
</servlet>
...
<servlet-mapping>
  <servlet-name>error</servlet-name>
  <url-pattern>/servlet/*</url-pattern>
</servlet-mapping>
```

- En algunos servidores (Tomcat, p. ej.) podemos colocar esto en el *web.xml* global (*conf/web.xml*) para que sirva para cualquier aplicación. Aunque estos servidores a veces traen su propia forma de deshabilitar el alias *servlet*.



Cargar servlets al inicio

- Podemos indicar que un servlet se cargue nada más iniciar el servidor Web:

```
<servlet>
  <servlet-name>nombre</servlet-name>
  <servlet-class>ClaseServlet</servlet-class>
  <load-on-startup>2</load-on-startup>
</servlet>
```

- El parámetro numérico es opcional (la etiqueta puede abrirse y cerrarse sin más), e indica el orden en que cargar los servlets al inicio, si hay varios que cargar



Ejemplos

- Servlet que genera texto plano
- Servlet que genera una página HTML
- Servlet que toma parámetros de inicio



Prueba de los ejemplos

- Definimos un fichero descriptor *web.xml* que permite llamar a cada ejemplo de 3 formas distintas:

```
http://localhost:8080/appserv1/servlet/ejemplos.ClaseServlet
```

```
http://localhost:8080/appserv1/servlet/ejemplo1_1
```

```
http://localhost:8080/appserv1/ejemploservlet
```

```
http://localhost:8080/appserv1/servlet/ejemplos.ClaseServletHTML
```

```
http://localhost:8080/appserv1/servlet/ejemplo1_2
```

```
http://localhost:8080/appserv1/ejemploservletHTML
```

```
http://localhost:8080/appserv1/servlet/ejemplos.ClaseServletInit
```

```
http://localhost:8080/appserv1/servlet/ejemplo1_3
```

```
http://localhost:8080/appserv1/ejemploservletInit
```



Logging en aplicaciones web

- Utilizaremos *Log4J* encapsulado en *commons-logging* para enviar mensajes de log en aplicaciones web, como en una aplicación normal
 - Añadimos los JAR de *commons-logging* y *log4j* a *WEB-INF/lib*
 - Definimos los ficheros *commons-logging.properties* y *log4j.properties* dentro de la carpeta *WEB-INF/classes* de la aplicación
 - *Volcado automático desde carpeta "resources"*
 - Colocar los mensajes en los servlets



Generación de mensajes en servlets

```
import javax.servlet.*;
import javax.servlet.http.*;
import org.apache.commons.logging.*;

public class ClaseServlet extends HttpServlet {
    static Log logger = LogFactory.getLog(ClaseServlet.class) ;

    public void doGet(...){
        ...
        logger.info("Atendiendo peticion");
        ...
        logger.warn("Error al obtener parametros!");
        ...
    }
}
```