SERVLETS

<devAcademy>

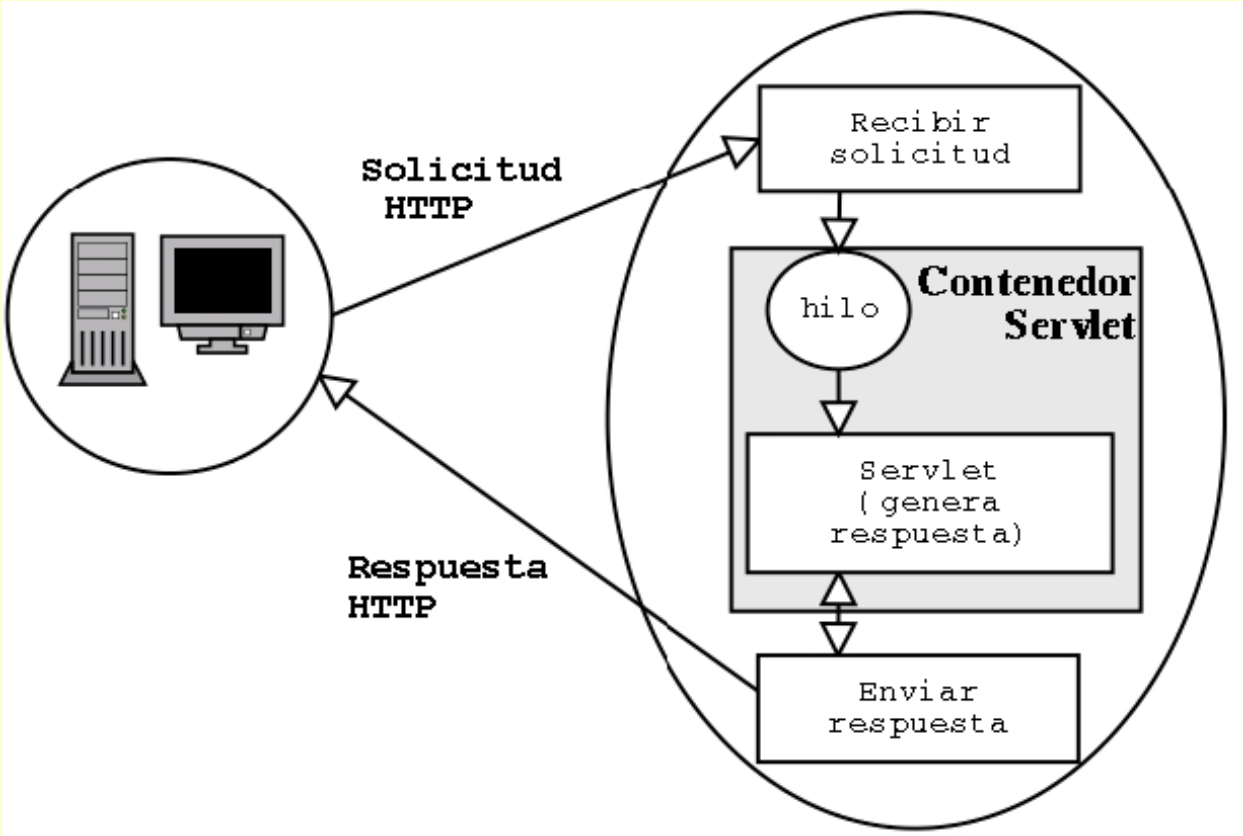
Servlets - Concepto

- En un momento de la evolución de los contenidos web, se pasó de contar con contenidos estáticos a necesitar contenidos dinámicos
- No bastaba con que una web nos diera fotos y textos, si no que se necesitaba funcionalidad

Servlets - Concepto

- Los Servlets, fueron la respuesta estándar de Java a esa necesidad de páginas con contenido dinámico
- Cuando hablamos de Servlets, hablaremos de programas en Java, que están se ejecutan en un Servidor, donde los usuarios después llamarán

Servlets - Concepto



Servlets - Concepto

- En concreto, un Servlet, es una interfaz de Java, que está diseñada para funcionar bajo un patrón de comunicación PETICIÓN-RESPUESTA
- Servlets, es una especificación (no un programa) Cada servidor, ofrece una implementación del servicio JEE

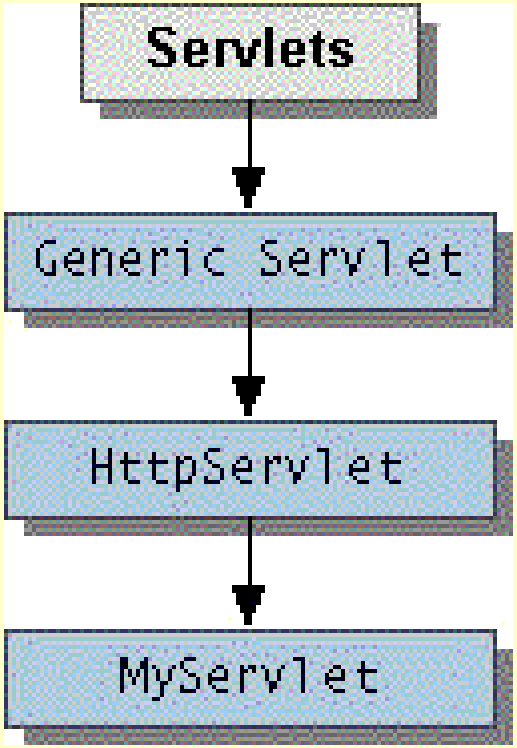
Servlets - JEE

- Las especificaciones, van siendo redefinidas y mejoradas periódicamente mediante documentos JSR (Java Specification Request) e incluídas en diferentes versiones de JAVA EE
- Cada servidor, va desarrollando sus implementaciones en sucesivas versiones

HttpServlet

- Aunque un Servlet está pensado para que interactúe con muchos tipos de clientes, normalmente, trabajaremos con instancias HttpServlet, que están preparadas para atender peticiones HTTP, recibidas desde un navegador (cliente) que emplea dicho protocolo para comunicarse con nuestro Servlet

HttpServlet



Invocar a un servlet

- Para invocar a un Servlet, podemos hacerlo desde:
  - NAVEGADOR HTML
    - Un enlace html
    - Poniendo la dirección en el navegador
    - A través de formularios html <form>
  - SERVIDOR: Desde otro Servlet
  - CLIENTE Telnet

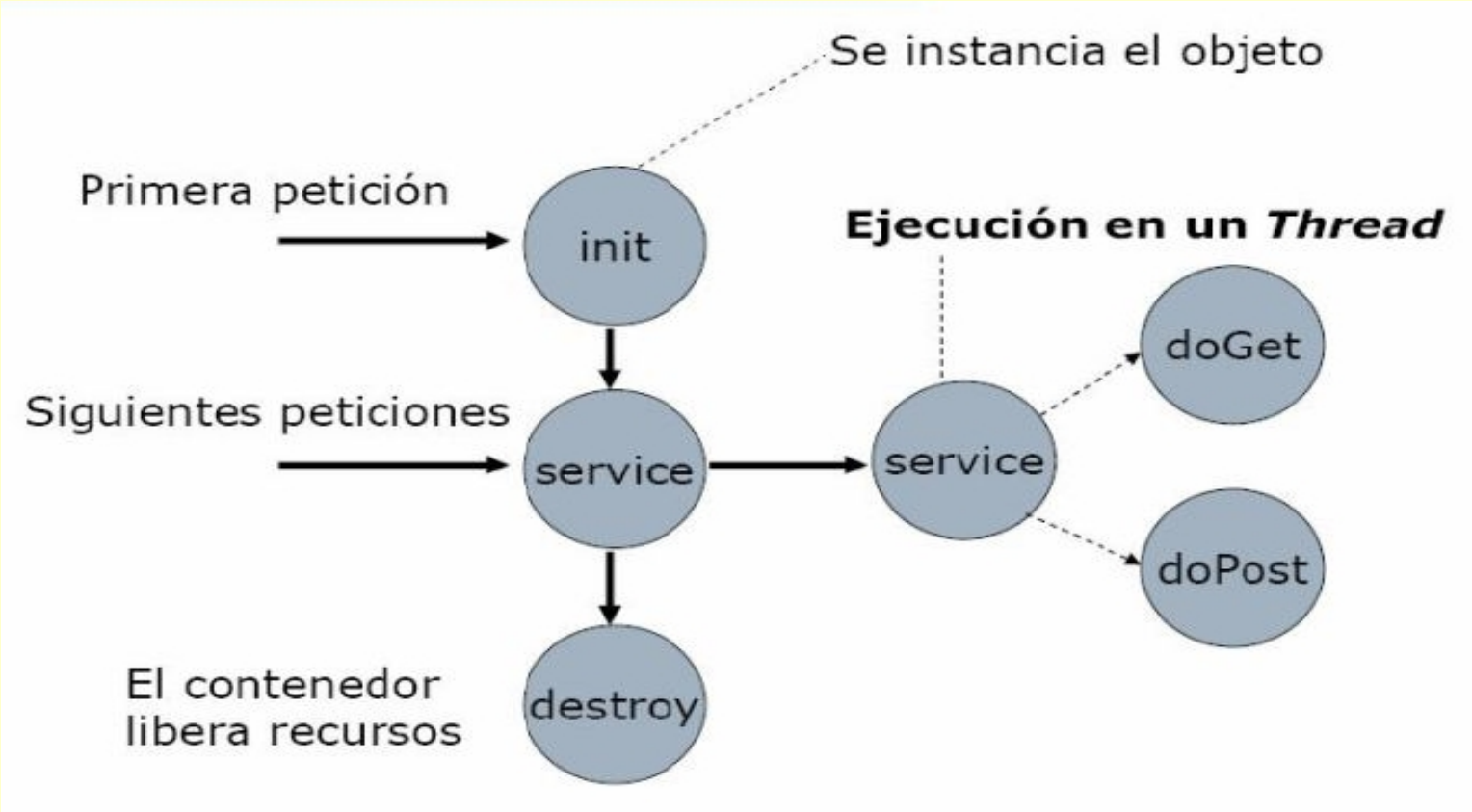
HttpServlet Métodos

- Los principales métodos que implementaremos son
  - GET Cuando recibimos una petición GET
  - POST Cuando recibimos una petición POST
  - SERVICE es invocado siempre antes y luego delega en el método invocado get o post

CICLO DE VIDA

- Cuando el servidor recibe una petición para un Servlet:
  - Crea la instancia (si no se invocó antes)
  - Llama a su método init() //inicialización
  - Llama a su método service()
  - Si necesita liberar espacio, lo elimina llamando a destroy ()

CICLO DE VIDA



Mensajes HTTP

- Tanto las peticiones desde cliente al servidor como las respuestas, se encapsulan en mensajes HTTP, ya que la comunicación es bajo este protocolo
- Un mensaje HTTP se compone de dos secciones diferencias: Cabecera y cuerpo

Mensajes HTTP

```
HTTP/1.1 200 OK
Content-Type: text/html
Header2: ...
...
HeaderN: ...
   (Blank Line)
<!doctype ...>
<html>
<head>...</head>
<body>
...
</body>
</html>
```

CABECERA

CUERPO

GET vs POST

- Aunque son métodos equivalentes en la práctica, presentan algunas diferencias

  - MODO DE CODIFICAR LA INFORMACIÓN
  - SIGNIFICADO DE USO
  - LÍMITE DEL TAMAÑO

GET vs POST

- Modo de codificar la información

  - USANDO GET, la información viaja en la propia URL (no hay body en el mensaje HTTP)

  - USANDO POST, la información viaja en el cuerpo (body) del mensaje HTTP

GET vs POST

- Significado de uso

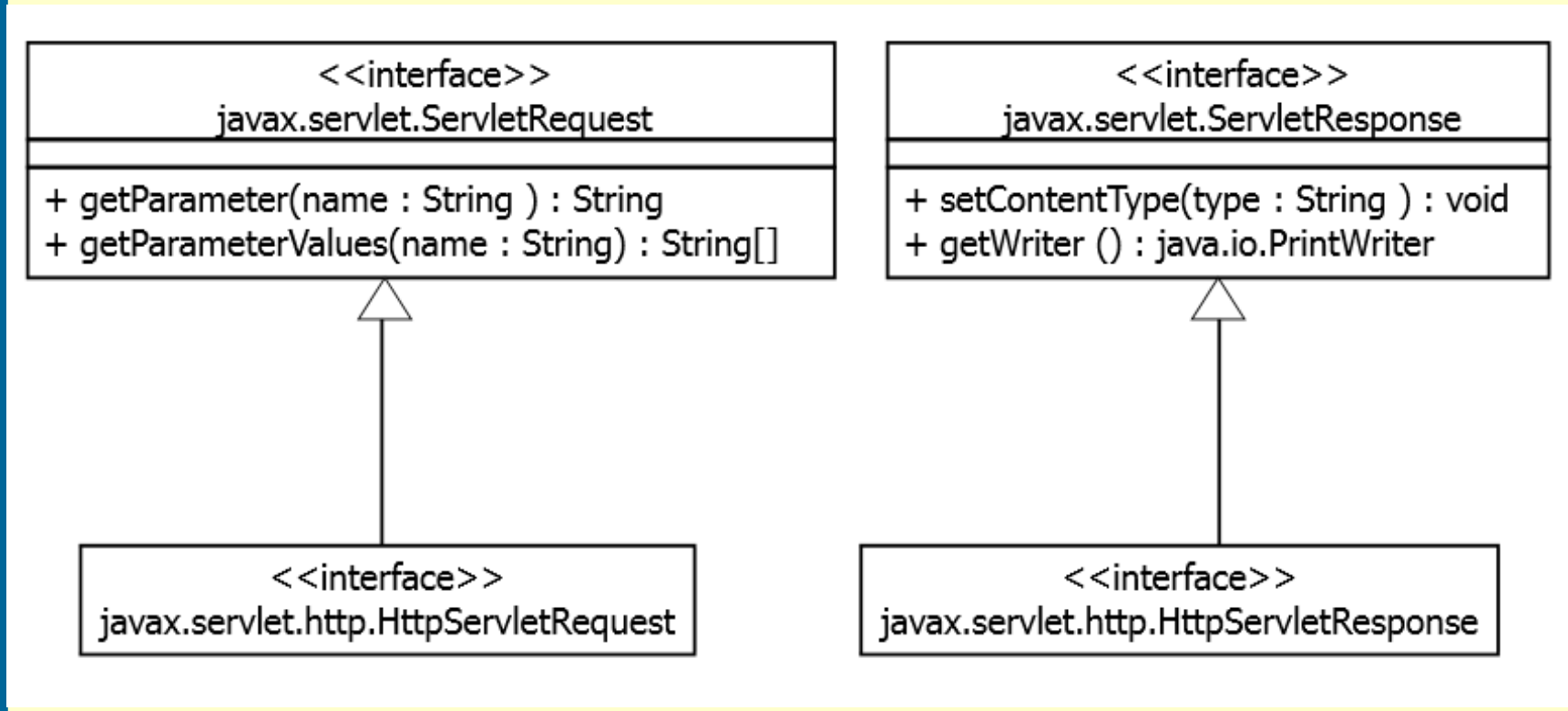
  - GET, para recoger información (consulta)
  - POST, cuando envíamos datos que van a modificar el estado del servidor (insert, delete, update, etc..)

GET vs POST

- Límite del tamaño

  - GET: Limitado al tamaño máximo de una URL (depende del navegador 2083 IE)
  - POST: Sin límite

Request y Reponse



Request y Reponse

- Un servlet, recibe dos objetos HttpServletRequest y HttpServletReponse, que son, respectivamente, las clases definidas para encapsular peticiones y respuestas

HttpServletRequest

- Métodos principales
  - getMethod() Si es GET, POST u otro
  - getRemoteAddr() la Ip del cliente
  - getHeader(paramName) get cabecera
  - getParameter(paramName) get parámetro
  - getQueryString()

HttpServletResponse

- Métodos principales
  - setContentType (tipo MIME respuesta)

  MIME pej text/html text/plain image/png
  - getWriter() Obtenemos el objeto (fichero) donde escribir la salida

Estructura WAR

- La estructura de una aplicación web, también queda definida por la especificación
- El contenedor (servidor) va a buscar los diferentes archivos en una localización predeterminada que debemos respetar

Estructura WAR

- WebContent: raíz (/) html, jsp, css, etc..
- Web-inf/lib: *.jar extras (p ej, log4j)
- Web-inf/web.xml: descriptor
- Web-inf/classes: mis clases compiladas

Cada app desplegada en el servidor, replica esta estructura

Web.xml

- Cuando el servidor se inicia, al recibir una petición, dirige al usuario a una página web por defecto.
- Esa página inicial, queda definida en el archivo WebContent/WEB-INF/web.xml

Web.xml

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3    <display-name>WebMicroForum</display-name>
4    <welcome-file-list>
5      <welcome-file>index.html</welcome-file>
6      <welcome-file>index.htm</welcome-file>
7      <welcome-file>index.jsp</welcome-file>
8      <welcome-file>default.html</welcome-file>
9      <welcome-file>default.htm</welcome-file>
10     <welcome-file>default.jsp</welcome-file>
11   </welcome-file-list>
```

Web.xml

- En el fichero anterior, index.html, queda dentro de la carpeta WebContent.
- Además, en este fichero, deberemos indicar la correspondencia entre los nombres de los servlets y la clase java httpservlet, para que el servidor sepa donde dirigir cada llamada

Web.xml

```
<servlet>
  <description>Ejemplo de prueba</description>
  <display-name>MiServlet</display-name>
  <servlet-name>MiServlet</servlet-name>
  <servlet-class>org.microforum.cursojava.jee.MiServlet</servlet-class>
</servlet>
<servlet-mapping>
  <servlet-name>MiServlet</servlet-name>
  <url-pattern>/MiServlet</url-pattern>
</servlet-mapping>
```

CORRESPONDENCIA -MAPPING

Web.xml

- Existe otra forma alternativa desde la spec 3.0 de Servlets, haciendo con una anotación, explícitamente dentro de la clase, sin tener que mapearlo en web.xml

```
@WebServlet("/miservlet")
public class ServletAnotacion extends
    HttpServlet { …
```

Practicando…

1. DESCARGAR TomcatV7.0
2. CONFIGURAR ECLIPSE (New Server)
3. CREAR UN “New Dinamic Web Project”
4. CREAR UN “Servlet”
5. CONFIGURAR EL web.xml
    1. Página(s) de inicio
    2. Mapping

Practicando…

6. Hacer una llamada desde una página que invoque al método POST de mi servlet

Practicando…

```
<html>
<head><meta charset="utf-8">
<title>Mi Primer Servlet</title>
</head>
<body>
<form action="/WebMicroForum/MiServlet" method=post>

  <BR>Introduzca un nombre y pulse Enviar<BR>
  <input type=text name=NOMBRE>
  <BR><input type=submit title="ENVIAR datos al Servlet"
value="ENVIAR">
</form>
</body>
</html>
```

Practicando…

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


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

Practicando…

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


response.setContentType("text/plain");
PrintWriter out = response.getWriter();
Out.println(“me han pasado a “ + nombre);
```

Practicando…

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

    response.setContentType("text/html");
    PrintWriter out = response.getWriter();
    String titulo = "HTTP Ejemplo lectura cabecera";
    out.println(  "<html>\n" +  "<head><title>" + titulo + "</title></head>\n"+
      "<body bgcolor=\"#f0f0f0\">\n" +
      "<h1 align=\"center\">" + titulo + "</h1>\n" +
      "<table width=\"100%\" border=\"1\" align=\"center\">\n" +
      "<tr bgcolor=\"#949494\">\n" +
      "<th>Cabecera</th><th>Valor</th>\n"+   "</tr>\n");
```

Practicando…

```
    Enumeration headerNames = request.getHeaderNames();
    while(headerNames.hasMoreElements()) {
       String paramName = (String)headerNames.nextElement();
       out.print("<tr><td>" + paramName + "</td>\n");
       String paramValue = request.getHeader(paramName);
       out.println("<td> " + paramValue + "</td></tr>\n");
    }
    //add la ip del cliente
    out.println("<tr><td> IP DEL CLIENTE </td>\n");
    out.println("<td> " + request.getRemoteAddr() + "</td></tr>\n");
    out.println("</table>\n</body></html>");
}
```

Listeners

- Podemos definir clases que implementen ciertas interfaces, cuyos métodos sean invocados bajo callbacks, ante determinadas situaciones o eventos en el servidor
- Por ejemplo: el arranque de una aplicación, el fin de una sesión

Listeners

- Las más importantes, son:

  -**ServletContextListener:** Gestiona el arranque y parada de la aplicación (contexto)

  -HttpSessionListener : Gestionar los eventos de la sesión como creación ,invalidación y destrucción.

  -ServletRequestListener : Se encarga de los eventos de creación y destrucción de peticiones.

Listeners

- Para definir un listener, debemos
  - Implementar la clase
  - Registrarlo
    - Implícitamente con la anotación @Weblistener
    - Explícitamente en web.xml

```
<listener>
    <listener-class>mipaquete.ClaseListener</listener-class>
</listener>
```

HttpSessionListener

- Para escuchar los eventos asociados a una sesión, deberemos implementar esta interfaz
- Ello, nos obligará a definir los métodos:
  - sessionCreated (HttpSessionEvent)
  - sessionDestroyed (HttpSessionEvent)

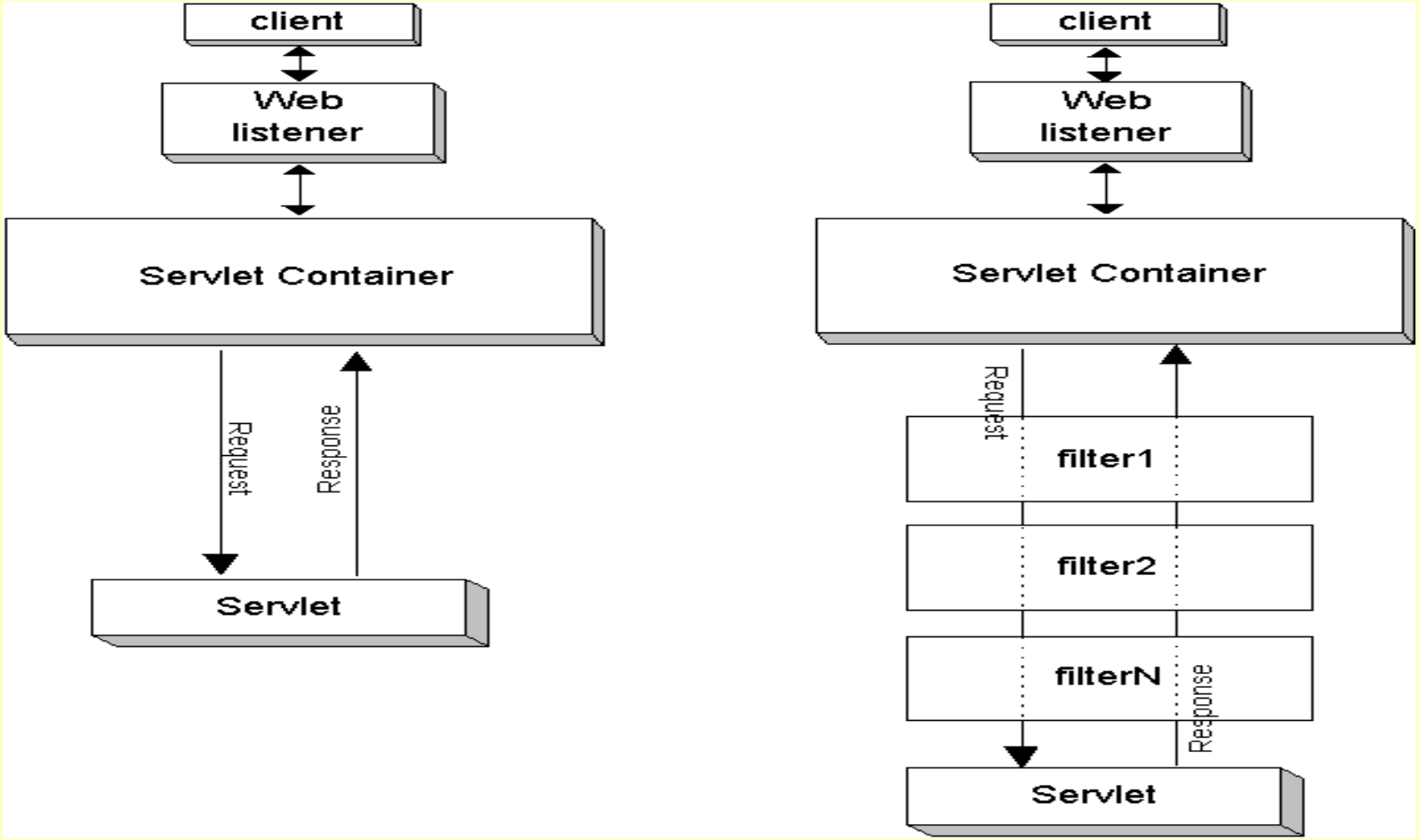
ServletContextListener

- Para escuchar los eventos asociados a un contexto (aplicación), deberemos implementar esta interfaz
- Ello, nos obligará a definir los métodos:
  - contextDestroyed(ServletContextEvent)
  - contextInitialized(ServletContextEvent)

Filtros

- Los filtros son código Java, que se ejecutarán antes y después de la ejecución de un servlet
- Su uso es muy variado: Medir el tiempo que tarda un servlet, control de estadísitcas, de acceso, compresión de archivos, conversión de formatos…

Filtros


client
Web listener
Servlet Container
Request
Response
Servlet
client
Web listener
Servlet Container
Request
filter1
filter2
filterN
Response
Servlet

Filtros – Interface Filter

- Para crear un filtro, lo primero que debemos hacer es crear una clase que impemente Filter
- Ello nos obliga a que nuestra clase implemente el método **doFilter**
- También init() y destroy()

Filtros – Interface Filter

doFilter (**ServletRequest, ServletResponse,FilterChain**)

El primer parámetro es la petición

El segundo la respuesta

Y el tercero, un objeto que permite transitar entre la invocación y el siguiente proceso de la “cadena”

Filtros – Interface Filter

doFilter(**ServletRequest, ServletResponse,FilterChain**)

{

//preprocesamos

FilterChain.doFilter(ServletRequest, ServletResponse)
    //llama al servlet o a otro filtro

//post procesamos

}

Filtros – web.xml

- Para registrar un filtro, es necesario describirlo en el archivo web.xml (descriptor de la aplicación)

```
<filter>
  <filter-name>mifiltro</filter-name>
  <filter-class>org.DEV ACADEMY.cursojava.jee.Filtro</filter-class>
 </filter>
 <filter-mapping>
  <filter-name>mifiltro</filter-name>
  <url-pattern>/*</url-pattern> <! - -Servicio llamado -->
 </filter-mapping>
```

Filtros – web.xml

- Puede darse que haya varios filtros, que concuerden con una llamada. En tal caso, se respeta el orden secuencial expresado en el descriptor

ServletContext

- Hay información que nos puede interesar parametrizar para nuestros servlets (cadena de conexión, driver de la base de datos, constantes, directorios…)
- Para tal fin podemos emplear la clase ServletContext, que nos proporciona información accesible para todos los servlets de nuestra app

ServletContext

- Esta clase, al estar definida por el estándar, se crear y se inicializa automáticamente cuando desplegamos una aplicación en el servidor

ServletContext

- Lo primero, deberemos parametrizar la información en el fichero web.xml, con pares de valores de la forma

```
<context-param>
        <param-name>username</param-name>
        <param-value>system</param-value>
</context-param>
```

ServletContext

- Una vez descrito y desplegada nuestra app en el servidor, podemos acceder a ella mediante

```
getServletContext().getInitParameter("username");
```

ServletConfig

- Si bien ServletContext es un objeto compartido para TODOS los servlets (es decir, hay uno por aplicación), tenemos opción de parametrizar un solo servlet mediante ServletConfig (uno por cada servlet)

ServletConfig

- De forma análoga, debemos incluir la descripción de nuestros parámetros, eso sí, en esta ocasión dentro del servlet

```
<servlet>
    <init-param>
      <param-name> n2 </param-name>
      <param-value> 200 </param-value>
    </init-param>
</servlet>
```

ServletConfig

- El objeto ServletConfig, será instanciado en la fase init() de nuestro servlet. Luego, dentro del service, simplemente usaremos

```
ServletConfig conf=getServletConfig();
String s2=conf.getInitParameter("n2");
```

REALM

- Realm es un mecanismo de validación y autenticación de usuarios para aplicaciones web, ya implementado en el core de los contenedores Apache Tomcat

REALM

- En vez de tener que implementar nosotros el proceso de validación (es decir, tomar los datos del usuario a la entrada del programa – usuario y contraseña-, ir a la base de datos, contrastar que es un usuario con el rol y la contraseña apropiadas, etc…) con sólo parametrizar Realm, éste hará el trabajo por nosotros

REALM PASOS

1 Crear tablas en la base de datos con usuarios y permisos (roles)

```
create table users (
  user_name        varchar(15) not null primary key,
  user_pass        varchar(15) not null
);
```

REALM PASOS

1 Crear tablas en la base de datos con usuarios y permisos (roles)

```
create table user_roles (
  user_name        varchar(15) not null,
  role_name        varchar(15) not null,
  primary key (user_name, role_name)
);
```

REALM PASOS

2 Configurar el contexto META-INF/context.xml

```
<?xml version='1.0' encoding='utf-8'?>
<Context docBase="WebMicroForum" path="/WebMicroForum" reloadable="true">
<Realm className="org.apache.catalina.realm.JDBCRealm"
connectionName="hr" connectionPassword="password"
connectionURL="jdbc:oracle:thin:@localhost:1521:xe"
driverName="oracle.jdbc.driver.OracleDriver" roleNameCol="ROLE_NAME"
userCredCol="USER_PASS" userNameCol="USER_NAME"
userRoleTable="USER_ROLES"
userTable="USERS"/>
</Context>
```

REALM PASOS

3 Incluir el driver de Oracle en la carpeta /lib del servidor

REALM PASOS

4 Definir en el web.xml la restricción de seguridad

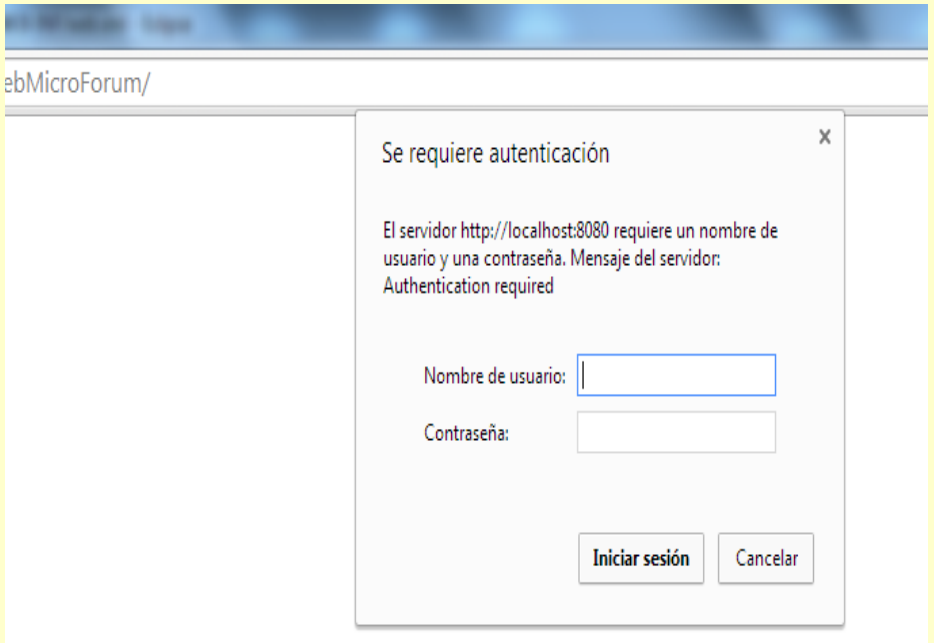
```
<security-constraint>
<web-resource-collection>
<web-resource-name>
Acceso Aplicación
</web-resource-name>
<url-pattern>/*</url-pattern>
<http-method>GET</http-method>
<http-method>POST</http-method>
</web-resource-collection>
```

REALM PASOS

<auth-constraint>
<role-name>0</role-name>
</auth-constraint>
</security-constraint>
<security-role>
<role-name>0</role-name>
</security-role>
<login-config>
<auth-method>BASIC</auth-method>
</login-config>

REALM

- Reiniciamos con todos los cambios y la próxima vez que entremos, nos debe aparecer el siguiente mensaje



REALM – Acceso User

- Una vez que nuestro usuario ha accedido con éxito, podemos acceder a su nombre a través del objeto Request

request.getUserPrincipal().getName()

NULL si el usuario no está autenticado

El nombre, en caso de que sí lo esté

HTTP Sin estado

- Cuando realizamos comunicaciones entre un cliente (navegador) y el servidor mediante HTTP, cada interacción de petición-respuesta, representa una conversación con un principio y un final intrínseco

HTTP Sin estado

- En una segunda petición del mismo cliente, no podemos determinar desde el servidor a priori, si el cliente que nos está llamando, era el mismo que el anterior
- Dicho en otras palabras, HTTP es un protocolo sin estado (stateless)

HTTP Control estado

- Para controlar el estado, JEE nos ofrece varias alternativas

  - SESSION
  - COOKIES

HTTPSESSION

- Para crear una sesión, ejecutamos getSession() sobre el objeto HttpServletRequest
- Con esto, conseguimos un objeto HttpSession, con un identificador único, que permanece asociado al cliente (navegador)

HTTPSESSION

- Para crear una sesión, ejecutamos getSession() sobre el objeto HttpServletRequest
- Con esto, conseguimos un objeto HttpSession, con un identificador único, que permanece asociado al cliente (navegador)

HTTPSESSION

- En sucesivas peticiones de ese cliente, al invocar a getSession, estaremos recuperado el objeto HttpSession creado la vez anterior

HTTPSESSION

- El objeto sesión creado, nos permite además almacenar información en él

- Gracias a este objeto, vamos a poder:

  - Identificar un cliente
  - Asociar información a él

HttpSession - Métodos

- Métodos para gestionar el ciclo de vida
  - getID()
  - setMaxInactiveInterval(secs)
  - isNew()
  - Invalidate()
  - getSession()

HttpSession - Métodos

- Métodos para gestionar el estado
  - setAttribute (Nombre, Valor)
  - getAttribute (Nombre)
  - removeAttribute (Nombre)

Fin de una Sesión

- En algún momento, el usuario finalizará la sesión con el servidor.
- El método invalidate() será invocado y con ello, el objeto HttpSession y los atributos asociados a él, desaparecen

Fin de una Sesión

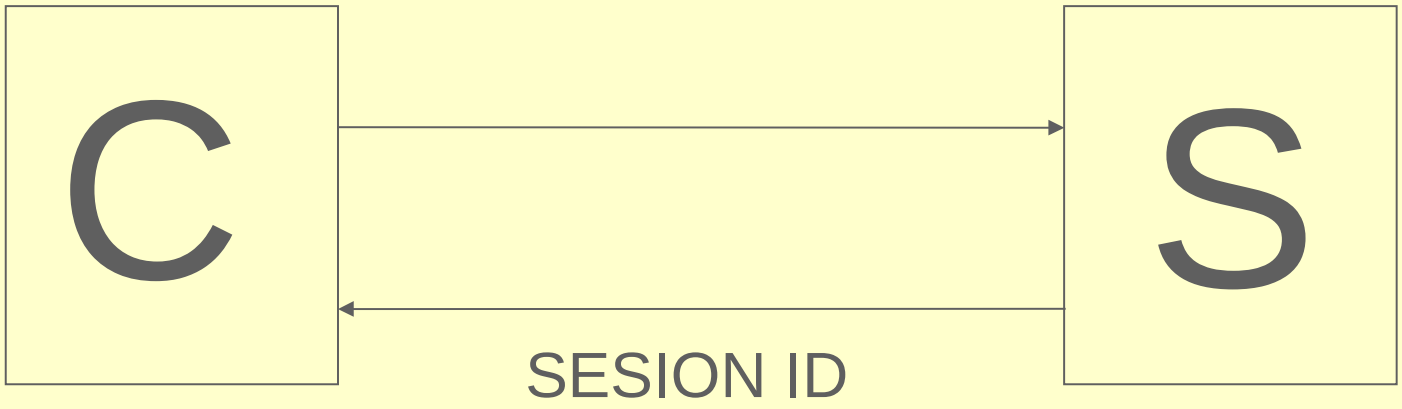
- Podemos llamar a invalidate()
  - Explícitamente (el usuario pulsa salir-logout)
  - Implícitamente, por ausencia de peticiones
    - Por defecto, una sesión muere a los 30’
    - Fijado por setMaxInactiveInterval(secs)
    - Parametrizado web.xml (minutos)

```
<session-config>
    <session-timeout>2</session-timeout>
</session-config>
```

getSession Detalle

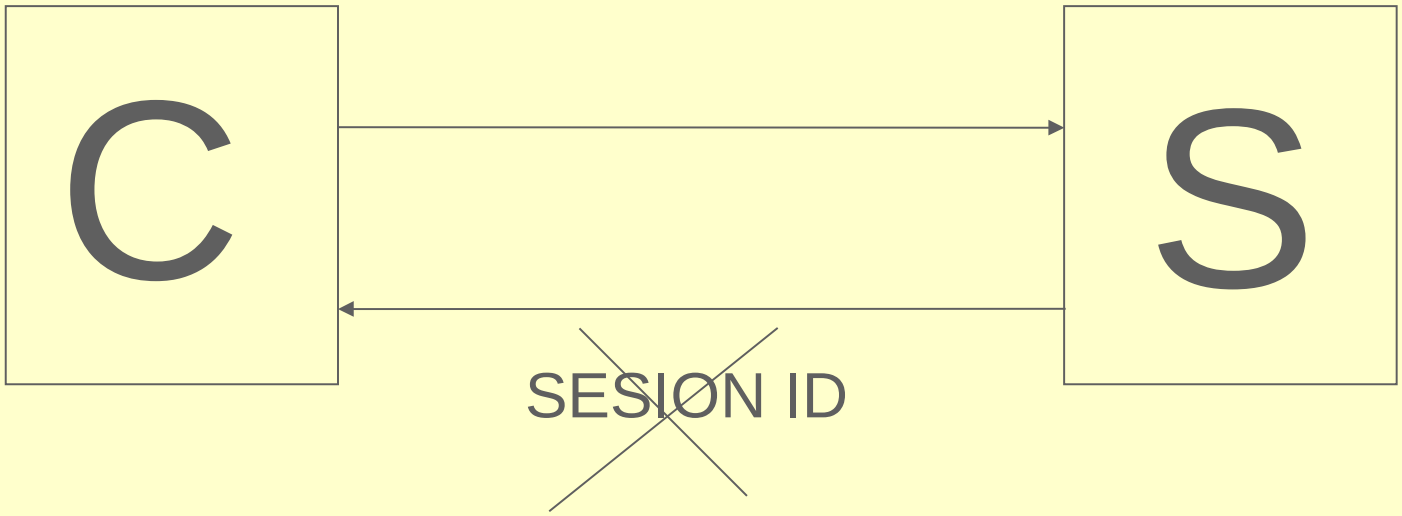
- Para comprobar que la petición, pertenece a una sesión activa, y por tanto es válida, usaremos la versión del método getSession (false) → Comprueba si la petición tiene una sesión asociada. Si es así, la devuelve, si no, devuelve NULL (y no crea una nueva, a diferencia de getSession() o getSession(true)

getSession () - (true)



getSession () - (true)

getSession (false)



getSession (false)

Redirigiendo un Servlet

- Podemos hacer que un Servlet llame a otro, o que nos lleve a otra página o en general ir desde un recurso (servlet, jsp, html, etc..) a otro
- Hay veces, que necesitamos alterar el flujo de ejecución, bien añadiendo más páginas al usuario o más procesos detrás de una petición

RequestDispatcher

- Para conseguir que un servlet se dirija a otro, usaremos el objeto RequestDispatcher, obtenido del objeto Request

request.getRequestDispatcher(recurso)

RequestDispatcher

request.getRequestDispatcher(recurso)

El recurso, será la ruta que identifica al servlet que queremos invocar

request.getRequestDispatcher("/Servlet2")

Forward() e Include()

- Tenemos dos métodos para hacer efectivo la redirección al recurso

  – forward(request, response)
  – include(request, response)

  Ambos, llamarán a la ruta pasada al requestDispatcher(“/ruta”)

Forward() e Include()

- Si uso forward(), la salida devuelta al usuario, será sólo la generada por el segundo recurso
- Si uso include(), cuando finalice el segundo recurso, el flujo de ejecución vuelve al recurso llamante y la salida al usuario es la suma de las salidas del recurso llamante y el llamado

RequestDispatcher

- Ya use un método u otro, la URL que el cliente recibe, es la MISMA que él envío en la petición original

- Los saltos de recursos, son, para él, transparentes

Compartir info Servlets

- Podemos compartir información de un servlet a otro, incluyendo atributos en el objeto request

1er recurso

request.setAttribute (Nombre:str, Valor:obj)

//redirecciono forward o include

2º recurso

request.getAttribute (Nombre:str): Objeto

Redirigiendo un Servlet

- Una forma alternativa de alterar el flujo de ejecución, es la redirección, que cuenta con la interacción del cliente.

- Esta forma alternativa, se usa sobre el objeto response, usando el método sendRedirect(location)

sendRedirect()

- Veamos qué implica el uso de este método:

1. resposne.sendRedirect(“/recurso”)

   Al hacer esta llamada, el servidor envía un mensaje HTTP al cliente, indicándole el valor /recurso en el parámetro de cabecera location

sendRedirect()

2. El cliente recibe un mensaje HTTP con un código de redirección (302) y lo interpreta, llamando al servidor nuevamente con la ruta indicada en location
3. El servidor recibe una nueva petición y el recurso referido en location, es invocado

sendRedirect()

4. Finalmente, el servidor procesa la petición y devuelve al cliente la respuesta final

RequestDispatcher vs sendRedirect()

- De ambas formas alteramos el flujo de navegación pero …
  - Con RequestDispatcher:
    - El cliente no interviene (no se entera)
    - Puedo compartir info entre Servlets
    - Más rápido (ahorro tráfico HTTP)
    - PELIGRO F5 (se vuelve a llamar)
    - No puedo redirigir a un servlet externo

RequestDispatcher vs sendRedirect()

- De ambas formas alteramos el flujo de navegación pero …
  - Con sendRedirect:
    - El cliente lanza una nueva petición GET
    - No podemos compartir info entre Servlets
    - Más lento (genero más tráfico HTTP)

Cookies

- Una cookie es una información generada por el servidor, única para cada cliente, que viaja en la cabecera del HTTP
- Es transmita al servidor y recibida por el cliente (navegador)
- En cada petición, el navegador también incluye dicha información en su cabecera

Cookies

- Es en realidad, un mecanismo casi idéntico al que manejan las sesiones, aunque existen diferencias en cuanto:
  - Al ciclo de vida
  - Al estado

El navegador puede gestionar un máximo de 20 Cookies por SERVIDOR

Cookies - Atributos

- Una cookie está compuesta por:
  - Un nombre
  - Un valor
  - Comentarios (opcionales)
  - Atributos de dominio
  - Edad

Cookies

- El método addCookie() sobre el objeto response añade la cookie al cliente
- En el servidor, al recibir una petición, usaremos el método getCookies sobre request para obtener las cookies de un cliente

Cookies - Ejemplo

```
Cookie cookie = new Cookie(“micokie",“21/12/14 15:35");
cookie.setMaxAge(60*60); //1 hora
response.addCookie(cookie);  //MANDO UNA COOKIE
-----------------------------------------------------------------
Cookie[] cookies = request.getCookies(); //LA RECUPERO
        if (cookies != null) {
            if (cookies[0].getName().equals(“micokie")) {
                loginCookie = cookie;
              }
           }
        }
-----------------------------------------------------------------------
         //ASÍ, HAGO QUE EL NAVEGADOR LA ELIMINE
            cookie.setMaxAge(0); //Y NO ME LA VUELVA A MANDAR
            response.addCookie(cookie);
```

Cookies - Métodos

- **Cookie (nombre:String, valor:String)**
- getName(), setName()
- getValue(), setValue()
- setDomain(), setPath()
- **setMaxAge(segundos),** getMaxAge()

Cookies-Caducidad-Age

- MaxAge Por defecto, vale -1. Cualquier número negativo indica que la cookie desaparece al cerrar el navegador
- MaxAge = 0 Hacemos que la cookie desaparezca el del navegador
- MaxAge > 0 Dura los segundos indicados. Transcurrido el tiempo, el navegador la elimina y no la remite

Manejo de Errores

- En los Servlets, como en todo código Java, pueden ocurrir excepciones o errores, que interrumpan el flujo normal del programa
- ¿Cómo reaccionamos? ¿Le soltamos la traza de pila al usuario?
- Por suerte, hay mejores opciones

Manejo de Errores

- Lo mejor, es que sea un Servlet, el que se encargue de recibir las excepciones y los errores y concentre el tratamiento de los mismos

Manejo de Errores

1 Definir y registrar el servlet en web.xml

2 Añadir al descriptor una información para asociarle los mensajes error del servidor y las excepciones de mis otros servlets

Manejo de Errores

```
<error-page>
    <error-code>404</error-code>
    <location>/ServletError</location>
</error-page>
<error-page>
    <exception-type> javax.servlet.ServletException
    </exception-type >
    <location>/ ServletError </location>
</error-page>
```

Atributos

En el objeto Request que recibe el Servlet que gestiona los errores, recibimos varios atributos, que ayudan a identificar el error

Tipo de expceción - javax.servlet.error.exception

Código HTTP - javax.servlet.error.status_code

Servlet fallido -javax.servlet.error.servlet_name

URL invocada -javax.servlet.error.request_uri

Atributos

Debo hacer request.getAttribute para cada valor, y hacer el casting al tipo de objeto apropiado

Throwable - javax.servlet.error.exception

Integer - javax.servlet.error.status_code

String -javax.servlet.error.servlet_name

String -javax.servlet.error.request_uri

Obteniendo Atributos

```
Throwable excepción = (Throwable)
 request.getAttribute("javax.servlet.error.exception");
 Integer codigoHTTP = (Integer)
 request.getAttribute("javax.servlet.error.status_code");
 String nombreServlet = (String)
 request.getAttribute("javax.servlet.error.servlet_name");
 if (null == nombreServlet ){nombreServlet =
 “Desconocido"; } String uriPedida = (String)
 request.getAttribute("javax.servlet.error.request_uri"); if
 (null== uriPedida ){uriPedida = “Desconocida"; }
```

ÁMBITOS

- Dónde vive una variable?
  - **Página (var local)**
  - **Petición (request.Attribute)**
  - **Sesión (session.Attribute)**
  - **Aplicación (sevletContext.Attribute)**