Tema 2
Servlets

J. Gutiérrez

Departament d'Informàtica
Universitat de València

DAW-TS (ISAW).
Curso 14-15

Índice

JavaEE

Java EE (versión 7) es un conjunto de especificaciones para desarrollar aplicaciones empresariales.

Se han desarrollado diferentes implementaciones a partir de estas especificaciones. Por ejemplo:

- GlassFish
- JBoss
- Apache Geronimo
- IBM WebSphere

Java EE

Java EE define diferentes contenedores que son entornos de ejecución y proporcionan servicios a los componentes que gestionan.

Ejemplos de servicios: gestión del ciclo de vida, inyección de dependencias, soporte para transacciones, seguridad, etc.

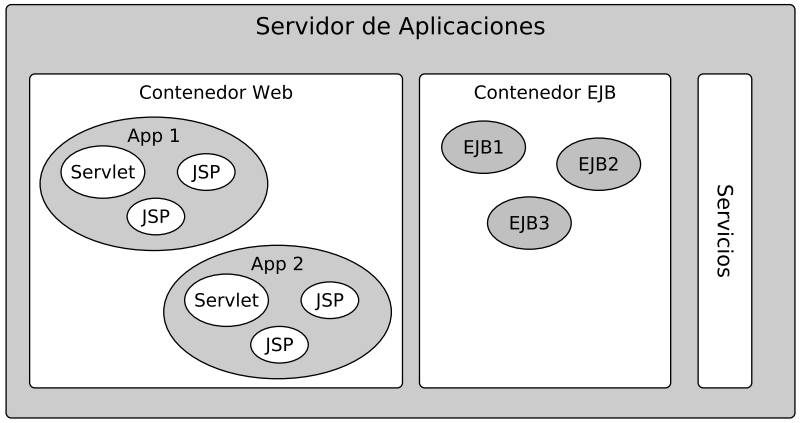
Además ofrece un API para desarrollar estos componentes (un conjunto de ficheros jar y documentación).

Java EE

Java EE ofrece dos contenedores que se ejecutan en el servidor:

- **Contenedor Web** que gestiona Servlets y Java Server Faces (JSF).
- **Contenedor EJB** que gestiona Enterprise Java Beans (EJBs).
  Los componentes del contenedor EJB son los encargados de implementar la lógica de negocio de la aplicación: operaciones sin estado, persistencia, servicios web, etc.
  Estos componentes pueden ser usados por componentes del contenedor Web o desde una aplicación de escritorio.

Java EE



Java EE

Se pueden realizar aplicaciones Web usando un conjunto de componentes del contenedor Web: Servlets (y JSPs, Java Server Pages) y JSF.

Aunque normalmente las aplicaciones Web usan componentes de los dos contenedores.

La comunicación entre los componentes del contenedor Web y los del contenedor EJB se realiza mediante RMI-IIOP (Remote Method Invocation sobre Internet Inter-ORB Protocol)

Se trata de RMI que usa el protocolo de CORBA en lugar del protocolo Java Remote Method Protocol.

Java EE

Los componentes Web que forman la aplicación se empaquetan en un archivo `war` (similar a `jar`) y se despliegan en el servidor.

Las aplicaciones Web constarán en general de:

- Uno o más Servlets
- Varios JSPs
- Recursos estáticos: (X)HTML, CSS, imágenes, ficheros con código JavaScript, etc.
- Acceso a bases de datos

El contenedor Web se encarga de gestionar la interacción entre los clientes y la capa de negocio. En concreto es responsable de:

- generar contenido dinámico,
- recibir la interacción del usuario con la interfaz,
- controlar el flujo de una página a otra,
- gestionar el estado de la sesión,
- mantener datos temporales en objetos JavaBeans.

Además ofrece un contenedor llamado `Application Client Container` que hace de interfaz entre una aplicación Java y el servidor Java EE.

Se ejecuta en la máquina cliente y hace de enlace entre la aplicación cliente y los componentes Java EE que se ejecutan en el servidor.

Esto sirve para realizar peticiones de un modo sencillo a los componentes EJB desde una aplicación de escritorio (que no se esté ejecutando en un contenedor EJB).

¿Qué es un servlet

Un servlet es un componente desarrollado en Java y cuyo ciclo de vida es gestionado por el contenedor web.

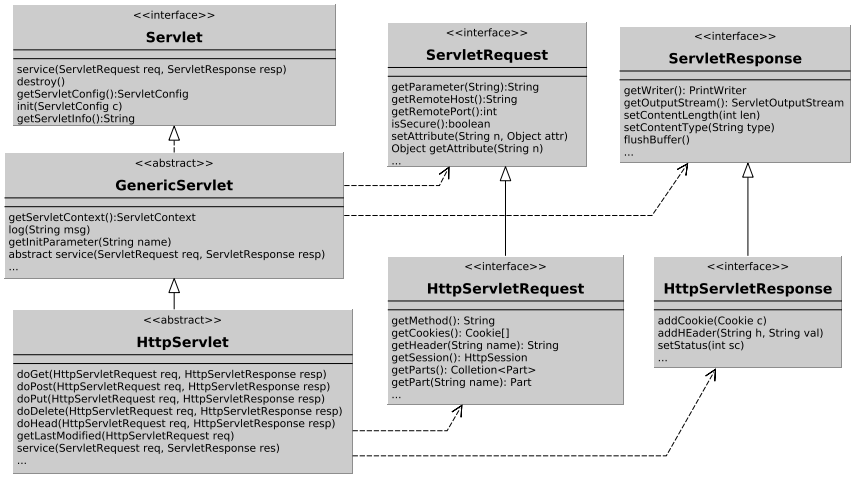
Se trata de una clase Java que extiende a *`javax.servlet.http.HttpServlet`* , en la implementación de sus métodos realiza alguna acción que depende de los parámetros enviados desde el cliente, y genera salida que puede ser una página HTML usando sentencias de escritura en flujo.

También es posible devolver desde el servlet otro tipo de salida (pdfs, imágenes, etc).

Código habitual en los servlets

1. Obtener campos de la cabecera (usando métodos de `HttpServletRequest`)
2. Obtener los parámetros pasados desde el cliente (usando métodos de `HttpServletRequest`)
3. Realizar la tarea en función de los parámetros y/o de los campos de cabecera (acceder a una base de datos, actualizar un objeto de sesión,)
4. Establecer campos de cabecera (usando métodos de `HttpServletResponse`)
5. Escribir la página en el flujo de salida (obtenido de `HttpServletResponse`)

Interfaces y clases importantes



interface Servlet

Interfaz que define los métodos que deben implementar todos los `Servlets`.

- Define métodos relacionados con el ciclo de vida: `destroy()` y `init(.)`.
- Define el método que es llamado cuando un cliente realiza la petición: `service(.)`.
- Y define métodos para obtener el objeto de iniciación: `ServletConfig getServletConfig()` y el método `String getServletInfo()`

Clase abstracta GenericServlet

Clase que implementa a la interfaz `Servlet` y que deja como abstracto únicamente el método `service(.)`.

Sirve para realizar un servlet genérico, independiente del protocolo. Para HTTP es mejor extender la clase `HttpServlet`.

Además de los métodos de `Servlet` define otros métodos:

- Para almacenar información en el `log`: `log(String msg)`
- Para obtener el `ServletContext`: `ServletContext getServletContext()`
- Para obtener un parámetro de iniciación: `String getInitParam(String nombre)`
- Para obtener los nombres de los parámetros de iniciación: `Enumeration<String> getInitParameterNames()`
- ...

Clase abstracta HttpServlet

Clase que extiende a la clase `GenericServlet` y que ofrece métodos específicos para trabajar con el protocolo `HTTP`.

Implementa el método `service(.)` desde donde se realizan llamadas a otros métodos en función del método de la petición:

- Si el método de la petición es `GET`: `doGet(.)`
- Si el método de la petición es `POST`: `doPost(.)`
- Si el método de la petición es `HEAD`: `doHead(.)`
- Si el método de la petición es `PUT`: `doPut(.)`
- Si el método de la petición es `DELETE`: `doDelete(.)`
- ...

interface HttpServletRequest

El contenedor Web crea un objeto de este tipo y lo pasa como primer argumento al enviar el mensaje `service(.)` al `Servlet`.

Con una referencia de este tipo se puede:

- Obtener la sesión asociada a la petición o crear una (si el booleano es true): `HttpSession getSession(boolean c)`
- Obtener un parámetro: `String getParameter(String nombre)`
- Obtener los nombres de los parámetros: `Enumeration<String> getParameterNames()`
- Obtener todos los valores asociados a un parámetro: `String[] getParameterValues(String nombre)`
- Obtener las *cookies* enviadas por el cliente: `Cookie[] getCookies`
- Obtener las partes del cuerpo de la petición: `Collection<Part> getParts()`
- Obtener una parte del cuerpo: `Part getPart()`
- ...

interface HttpServletResponse

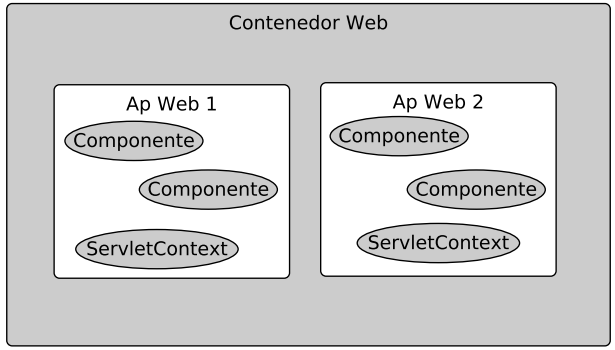
El contenedor Web crea un objeto de este tipo y lo pasa como segundo argumento al enviar el mensaje `service(.)` al `Servlet`.

Con una referencia de este tipo es posible:

- Añadir *cookies*: `addCookie(Cookie c)`
- Añadir campos de cabecera: `addHeader(String nombre, String valor)`
- Obtener un flujo de salida orientado a caracteres: `PrintWriter getWriter()`
- Obtener un flujo de salida orientado a bytes: `ServletOutputStream getOutputStream()`
- Establecer el código de estado: `void setStatus(int sc)`
- ...

ServletContext

El contenedor crea un objeto del tipo `SevletContext` para cada aplicación web:



Los componentes de una aplicación web comparten este objeto (si están en la misma máquina virtual).

ServletContext

Este objeto ofrece entre otros métodos para:

- Obtener la ruta en el disco de un recurso

```
String getRealPath(String path)
```

- Obtener un objeto permite generar una respuesta a partir de la petición (puede ser un Servlet, un JSP o HTML):

```
RequestDispatcher getRequestDispatcher(String path)
```

- Obtener el tipo MIME de un fichero

```
String getMimeType(String fichero)
```

- Establecer un atributo

```
void setAttribute(String nombre, Object attr)
```

- Recuperar un atributo

```
Object getAttribute(String nombre)
```

- ...

interface ServletConfig

Se trata de un objeto que usa el contenedor para pasar información al `Servlet` durante su iniciación.

Ofrece métodos para:

- Obtener el valor de un parámetro inicial:

```
String getInitParameter(String name)
```

- Obtener todos los nombres de los parámetros iniciales:

```
Enumeration<String> getInitParameterNames()
```

- Obtener el `ServletContext`

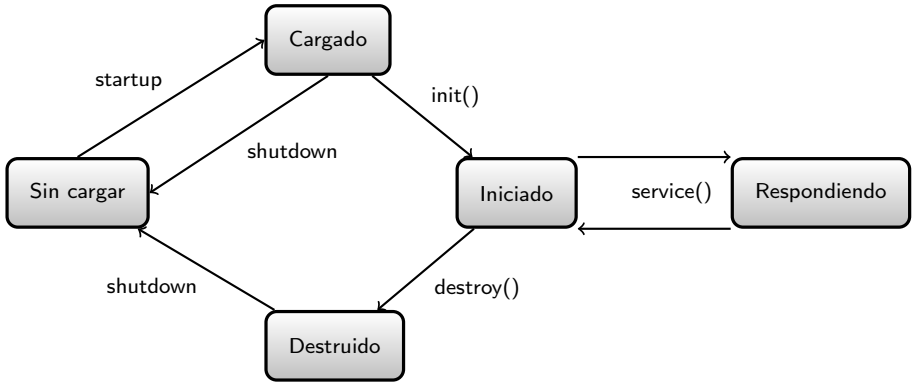
```
ServletContext getServletContext()
```

- Obtener el nombre del servlet

```
String getServletName()
```


Ciclo de vida del servlet

El contenedor se encarga de gestionar el ciclo de vida del servlet. Es decir, nosotros no llamamos explícitamente a esos métodos, sino que es el contenedor quien lo hace.



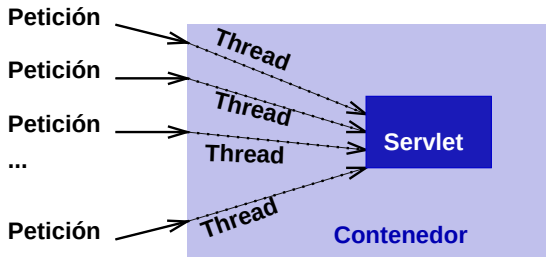
Carga y creación de una instancia del servlet

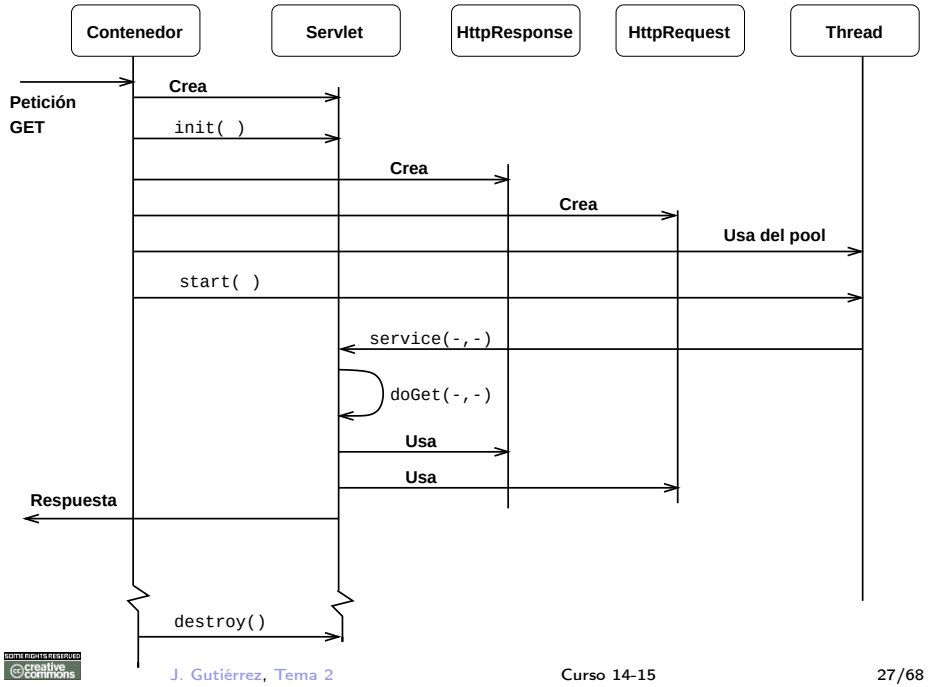
Al instalar la aplicación (fichero war) en un contenedor web, éste busca dentro del war el fichero `web.xml` o en las anotaciones.

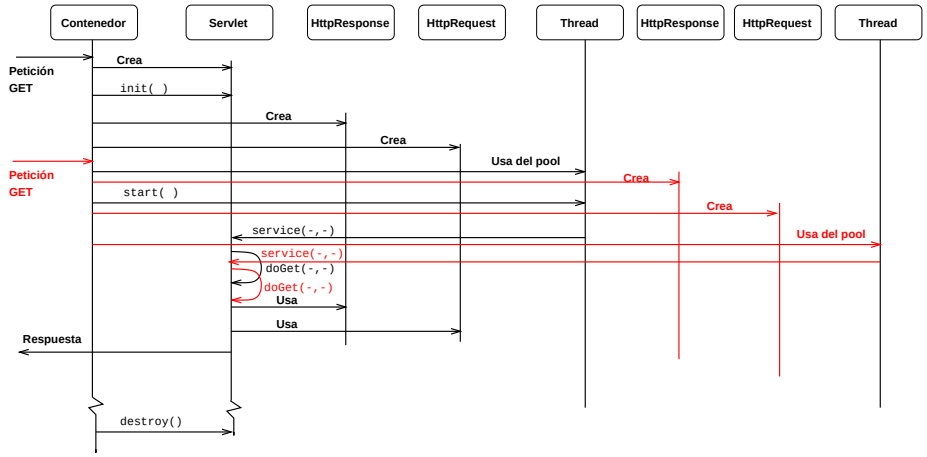
El objetivo es obtener la URL y el nombre de la clase que contiene el código.

El contenedor crea una única instancia de cada servlet.

Cada vez que llega una petición al Servlet se crea (o utiliza de un *pool*) un hilo. A continuación se ejecuta el hilo desde donde se llama a los métodos del Servlet.







En la figura anterior se aprecia que podemos tener dos hilos ejecutando el mismo método del Servlet.

Esto puede provocar errores y para prevenirlos hay que conseguir la exclusión mutua en la ejecución de secciones críticas.

Una forma de conseguirlo es usando bloques `synchronized`:

```
public void doGet(...) ...{
   //Código que puede ser ejecutado por múltiples hilos
   synchronized(ref){
      //Sección crítica: código que puede ser ejecutado por un único hilo
   }
   // Código que puede ser ejecutado por múltiples hilos
```


Tal y como hemos visto, tras crear el Servlet se llama a su método `init()`

En este método se realizan las tareas de iniciación (aquellas que se tienen que ejecutar una sola vez).

Respuesta a las peticiones

Cuando el contenedor recibe una petición para el servlet se llama al método `service(ServletRequest, ServletResponse)` .

Habitualmente no se escribe el código de este método en la clase que estamos realizando sino que se implementan los métodos `doGet(.)` y/o `doPost(.)` , ...

Estos métodos son llamados desde `service(.)` .

Destrucción del servlet

Cuando el contenedor destruye el servlet llama a su método `destroy()` .

En este método debe ir la liberación de recursos.

Una vez se ha destruido el servlet el contenedor no puede usar la instancia.

Un contenedor puede destruir el servlet si necesita recursos.

Índice

interface HttpSession

La sesión permite identificar a un usuario entre diferentes peticiones.

El servidor puede mantener la sesión mediante `cookies` o mediante la reescritura de URLs.

Mediante una referencia de este tipo se puede manejar información sobre la sesión como por ejemplo: el identificador de sesión, el instante de creación, el instante de último acceso o añadir/consultar atributos a la sesión.

HttpSession

- `String getId()`
- `long getCreationTime()`
- `long getLastAccessedTime()`
- `setAttribute(String nombre, Object valor)`
- `Object getAttribute(String nombre)`
- `removeAttribute(String name)`
- `boolean isNew()` devuelve `true` si el cliente no se ha enterado de la sesión o no quiere unirse a ella.
- ...

Índice

Cookie

Una *cookie* tiene un nombre, un valor y atributos adicionales tales como: comentario, ruta, dominio, periodo de validez y número de versión.

El `Servlet` envía *cookies* al navegador como campos de cabecera usando el método `HttpServletResponse.addCookie(Cookie)` .

El navegador envía las *cookies* como campos de cabecera. Se pueden obtener en el `Servlet` usando el método `HttpServletRequest.getCookies()`.

El navegador puede enviar varias *cookies* con el mismo nombre pero con diferente valor del atributo `path`.

Cookie

Ofrece un constructor: `Cookie(String nombre, String valor)` y una serie de métodos para:

- Establecer su periodo de validez: `setMaxAge(int exp)`
- Consultar su periodo de validez: `int getMaxAge()`
- Consultar el nombre: `String getName()`
- Consultar el valor: `String getValue()`
- Establecer el `path`: `setPath(String p)`
- Consultar el `path`: `String getPath()`
- Cambiar el valor: `setValue(String nuevoValor)`
- Establecer la ruta: `setPath(String path)`
- Establecer el dominio: `setDomain(String dom)`
- ...

Índice

Anotaciones

`@WebServlet`

Permite especificar el nombre, la URL que dispara la ejecución del servlet (el *mapping* y los parámetros iniciales (usando otra anotación).

```
@WebServlet("/getMsgs")
class GetMsgs extends HttpServlet{
   ...
}
```

```
@WebServlet(name="getmsg",value="/getMsgs"})
class GetMsgs extends HttpServlet{
   ...
}
```

```
@WebServlet(name="getmsg",urlPatterns={"/getMsgs","/getAllMsgs"})
class GetMsgs extends HttpServlet{
   ...
}
```

Anotaciones

@WebInitParams

Permite especificar los parámetros iniciales. Esta anotación aparece dentro de la anotación anterior.

```
@WebServlet(name="getmsg",value="/getMsgs",
   initParams=@WebInitParam(name="dir",value="/tmp")
)
class GetMsgs extends HttpServlet{
   ...
}
```

```
@WebServlet(name="getmsg",value="/getMsgs",
   initParams={
      @WebInitParam(name="dir",value="/tmp"),
      @WebInitParam(name="maxMsgs",value="80"),
   }
)
class GetMsgs extends HttpServlet{
   ...
}
```


Titulo

Si no se usan anotaciones se pueden especificar parámetros iniciales en el fichero `web.xml`:

```
<servlet>
   <servlet-name>Inicio</servlet-name>
   <servlet-class>es.uv.aw.ServletInicio</servlet-class>
   <init-param>
     <param-name>dir</param-name>
     <param-value>/home/juan/tmpdir</param-value>
   </init-param>
 </servlet>
```


Anotaciones

Dentro del código del `Servlet` se pueden usar los siguientes métodos de la clase `HttpServletRequest` para obtener los parámetros iniciales:

- Para obtener un parámetro de iniciación: `String getInitParam(String nombre)`
- Para obtener los nombres de los parámetros de iniciación: `Enumeration<String> getInitParameterNames()`
- ...

Anotaciones

@MultipartConfig

Sirve para indicar que el servlet espera una codificación del cuerpo de la petición del tipo `multipart/form-data`.
En esta anotación se puede especificar entre otros el directorio donde se guardarán los ficheros o el tamaño máximo del fichero enviado desde el cliente.

Anotaciones

Ejemplo:

```
<form id="frmUploadFile" action="http://server/app"
          method="POST" enctype="multipart/form-data">
   <input type="file" name="fileName" accept="text/*"/><br />
   Selecciona el tipo de procesado: <br>
   <input type="radio" name="procesado" value="latex">Latex <br />
   <input type="radio" name="procesado" value="pdflatex">PdfLatex <br />
   <input type="submit" value="Enviar"/>
</form>
```

El navegador envía el campo de cabecera `Content-Type` que indica que se trata de `multipart/form-data` y cual es la frontera entre los diferentes campos enviados.

```
Content-Type: multipart/form-data; boundary=TEXTO-FRONTERA
```

El valor de la frontera cambia en cada envío.

Anotaciones

```
POST /url HTTP/1.1
Content-Type: multipart/form-data; boundary=TEXTO-FRONTERA
...

TEXTO-FRONTERA
Content-Disposition: form-data; name="fileName"; filename="articulo.tex"
Content-Type: application/octet-stream

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Et nunc quidem quod eam tuetur,
ut de vite potissimum loquar, est id extrinsecus; Igitur neque stultorum quisquam beatus
neque sapientium non beatus.

TEXTO-FRONTERA
Content-Disposition: form-data; name="procesado"

latex
TEXTO-FRONTERA
```


Anotaciones

Cuando un servlet lleva esta anotación se pueden obtener cada parte (`Part`) del cuerpo usando los métodos `getPart` o `getParts`.

Al método `getPart` se le pasa el nombre de la parte que se quiere obtener.

Índice

Desarrollo de Servlets (I)

Ejemplo de generación de una página estática

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

public class EjServlet extends HttpServlet{
   static final long serialVersionUID = 1L;

   /** Método que se llama cuando hay una petición GET */
   protected void doGet(HttpServletRequest request, HttpServletResponse response) throws
         ServletException, IOException {
      // Informamos al navegador del contenido de la respuesta
      response.setContentType("text/html");

      // Obtención del flujo de salida
      PrintWriter pw = response.getWriter();

      // Escritura de la página html
      pw.println("<html><body>");
      pw.println("<h1> Hola </h1>");
      pw.write("</body></html>");
   }

   /** Método que se llama cuando hay una petición POST */
   protected void doPost(HttpServletRequest request, HttpServletResponse response) throws
         ServletException, IOException {
   }
}
```


Desarrollo Servlets (II)

Ejemplo de obtención de un parámetro en la petición

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

public class EjServlet extends HttpServlet{
   static final long serialVersionUID = 1L;

   /** Método que se llama cuando hay una petición GET */
   protected void doGet(HttpServletRequest request, HttpServletResponse response) throws
         ServletException, IOException {
      // Informamos al navegador del contenido de la respuesta
      response.setContentType("text/html");

      // Obtención del parámetro nombre enviado en la petición GET
      String nombre = request.getParameter("nombre");

      // Obtención del flujo de salida y escritura de la página HTML
      PrintWriter pw = response.getWriter();
      response.getWriter().write("<html><body>");
      response.getWriter().write("<h1> Hola " +  nombre + " </h1>");
      response.getWriter().write("</body></html>");
   }

   /** Método que se llama cuando hay una petición POST */
   protected void doPost(HttpServletRequest request, HttpServletResponse response) throws
         ServletException, IOException {
   }
}
```


Desarrollo de Servlets (III)

Ejemplo de obtención de campos de cabecera I

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

public class EjServlet extends HttpServlet{
   static final long serialVersionUID = 1L;

   /** Método que se llama cuando hay una petición GET */
   protected void doGet(HttpServletRequest request, HttpServletResponse response) throws
         ServletException, IOException {
      // Informamos al navegador del contenido de la respuesta
      response.setContentType("text/html");

      // Obtención del campo de cabecera "User-agent"
      String cliente = request.getHeader("User-agent");

      String title="";

      if (cliente.contains("MSIE"))
         title = "Petición realizada desde Internet Explorer";
      else
         title = "Petición no realizada desde Internet Explorer";

      // Obtención del flujo de salida y escritura de la página HTML
      PrintWriter pw = response.getWriter();
      pw.println("<html><head><title>" + title + "</title></head><body>");
```


Desarrollo de Servlets (III)

Ejemplo de obtención de campos de cabecera II

```
      pw.println("<h1> Hola </h1>");
      pw.println("</body></html>");
   }

   /** Método que se llama cuando hay una petición POST */
   protected void doPost(HttpServletRequest request, HttpServletResponse response) throws
         ServletException, IOException {
   }
}
```

```
@WebServlet(urlPatterns = { "/SubirRecurso" }, initParams = {
@WebInitParam(name = "dir", value = "/home/user/tmp/") })
@MultipartConfig
public class SubirRecurso extends HttpServlet {
   private static final long serialVersionUID = 1L;

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

      String ruta = getInitParameter("dir");
      try {
         String fileName = null;
         for (Part part : request.getParts()) {

            InputStream is = request.getPart(part.getName())
                  .getInputStream();
            int i = is.available();
            byte[] b = new byte[i];
            is.read(b);

            fileName = getFileName(part);

            FileOutputStream os = new FileOutputStream(ruta + fileName);
            os.write(b);
            is.close();
         }
         respuesta("El fichero se ha almacenado correctamente");
```



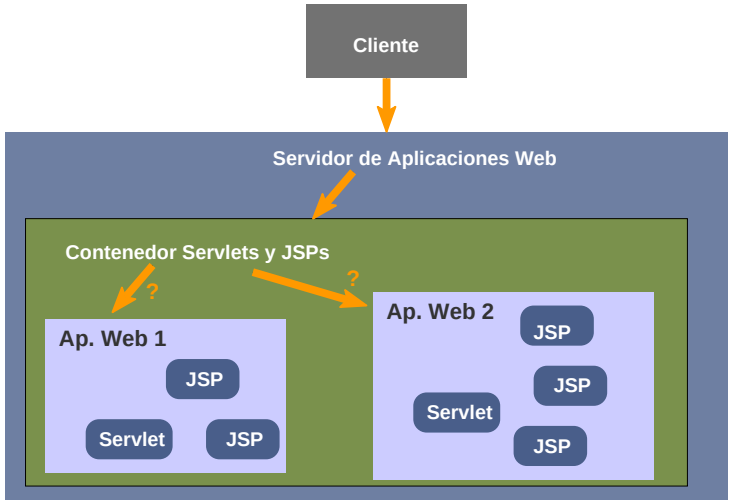
```
      } catch (Exception ex) {
         respuestaError("Se ha producido un error");
      }
   }

   private void respuestaError(String msg) throws IOException{
      PrintWriter pw = response.getWriter();
         pw.println("<html>");
         pw.println("<body>");
         pw.println(msg);
         pw.println("</body>");
         pw.println("</html>");
         pw.flush();
   }

   // Este método sirve para obtener el nombre del fichero (sin la ruta)
   private String getFileName(Part part) {
      for (String cd : part.getHeader("content-disposition").split(";")) {
         if (cd.trim().startsWith("filename")) {
            return cd.substring(cd.indexOf('=') + 1).trim()
                  .replace("\"", "");
         }
      }
      return null;
   }
}
```


Problemas con los servlets

- Si la página es compleja es tedioso escribir con sentencias `out.println()` el código de la página (que puede contener HTML, javascript, ...).
- No se separa la presentación de la lógica de la aplicación (un diseñador tiene que saber programación para modificar la apariencia de la página).

Esto motivó la aparición de JSP y sus posteriores ampliaciones.

Servidor de aplicaciones (I)



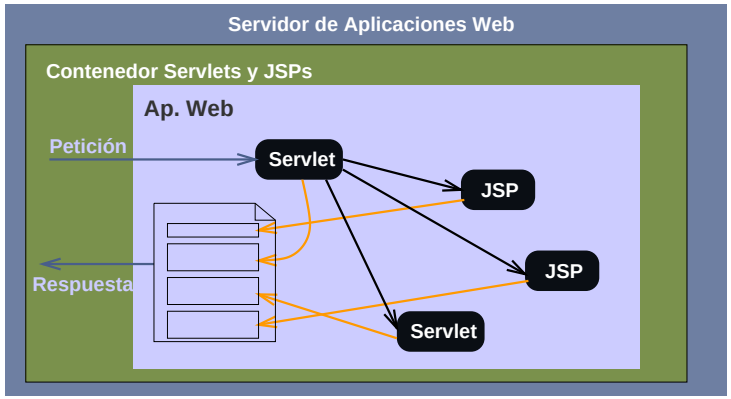
Cuando un servidor recibe una petición debe generar una respuesta

Si se trata de una aplicación web, los componentes que forman la aplicación deben generar la respuesta.

Los componentes pueden colaborar para generar la respuesta de dos modos:

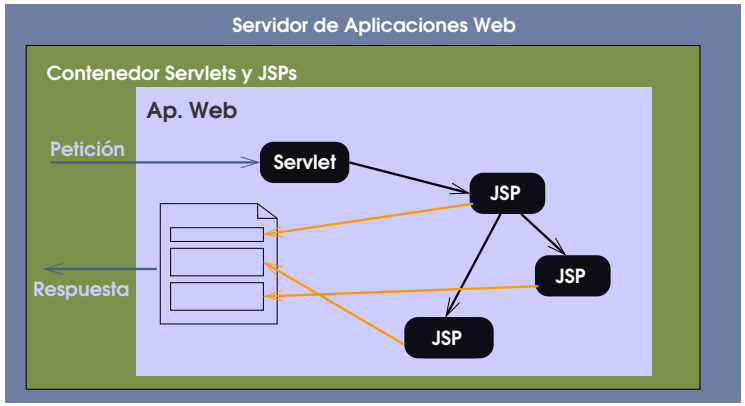
- **Articulación**: cada uno de ellos genera una parte de la respuesta
- **Delegación**: se delega la generación de la respuesta en un componente

Interacción entre componentes Web (I) *Articulación*



Un componente (en este caso un servlet) articula a una serie de componentes para generar de forma colaborativa una respuesta.
http://download.oracle.com/javaee/6/tutorial/doc/bnagi.html

Interacción entre componentes Web (II) *Delegación*



Un componente (en este caso un servlet) delega en otro componente la generación de la respuesta.
http://download.oracle.com/javaee/6/tutorial/doc/bnagi.html

Cuando desarrollamos un aplicación web que está compuesta por diferentes componentes (Servlets y JSPs) podemos necesitar que compartan variables y que cada una de esas variables tenga un determinado tiempo de vida (*scope*) en función de las necesidades.

Este tiempo de vida puede ser a los siguientes niveles:

- **Application**
- **Session**
- **Request**
- **Page**

A continuación se describe cada uno de ellos.

El objeto del tipo `ServletContext` ofrece un método para obtener un elemento de la aplicación Web (servlet, JSP).

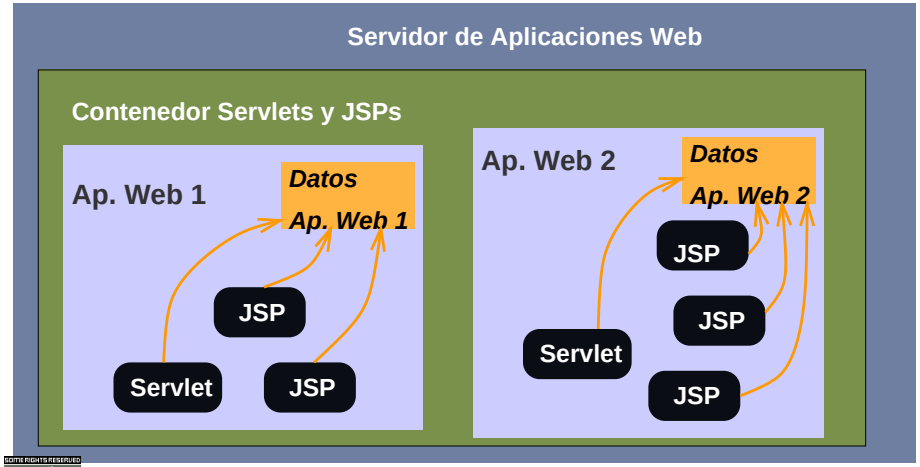
```
// Metodo de ServletContext
RequestDispatcher getRequestDispatcher(String ruta_al_elemento)
```

Una vez se tiene ese elemento lo podemos usar para para que se encargue de tratar la petición (delegación) o para incluir su salida (articulación).

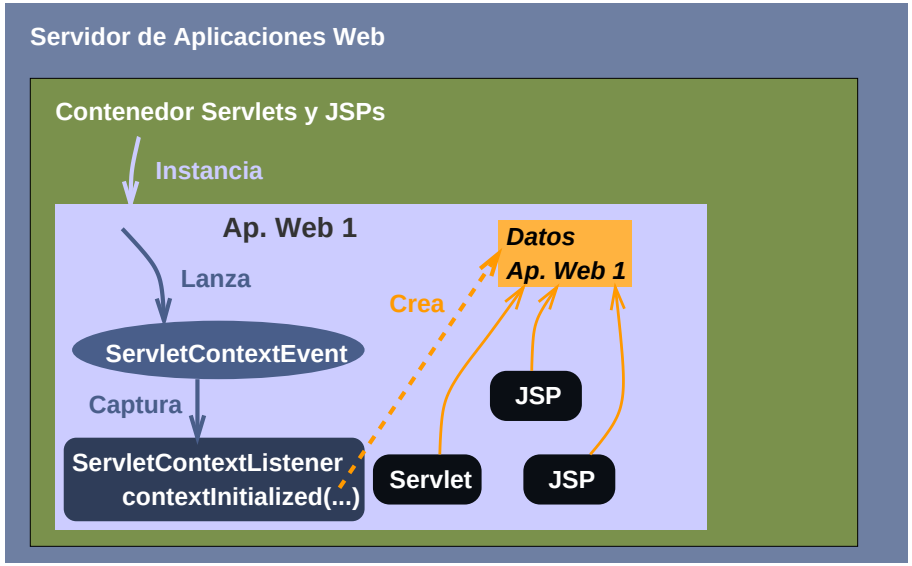
```
// Métodos de RequestDispatcher
void forward(ServletRequest request, ServletResponse response)
        throws ServletException,
               java.io.IOException

void include(ServletRequest request, ServletResponse response)
        throws ServletException,
               java.io.IOException
```


Application scope (I)

¿Cómo se definen variables que sean visibles por todos los componentes de la aplicación web y que se mantienen accesibles (scope) mientras la aplicación web está ejecutándose en el servidor?

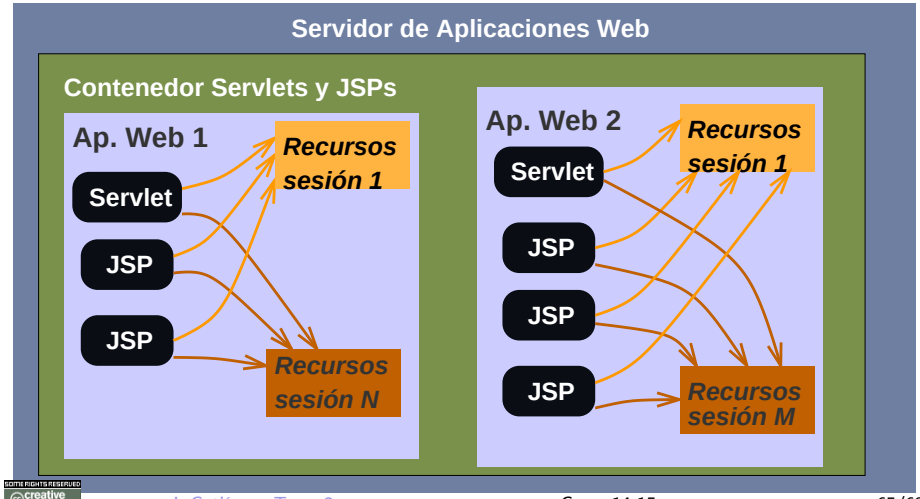


Application scope (II)

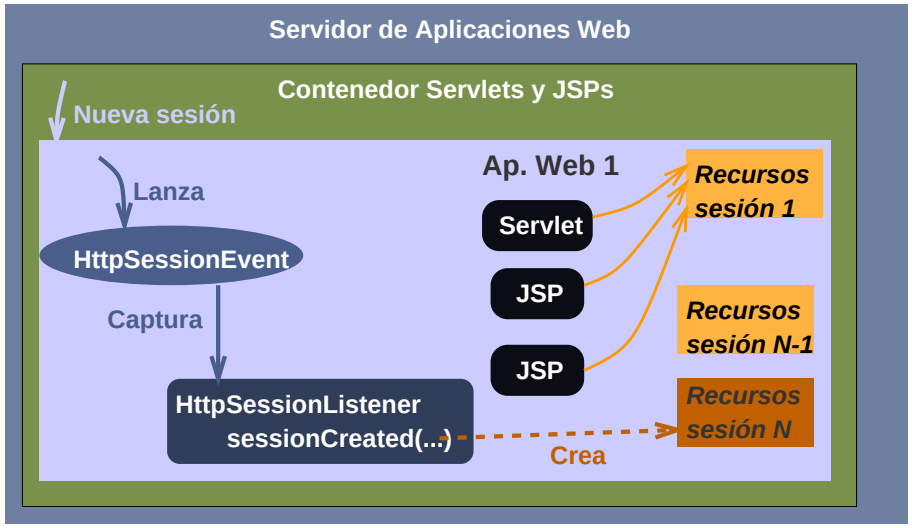


Session scope (I)

¿Cómo se definen variables que sean visibles por todos los componentes de la aplicación web y que se mantienen accesibles (scope) mientras la sesión con un cliente esté activa?

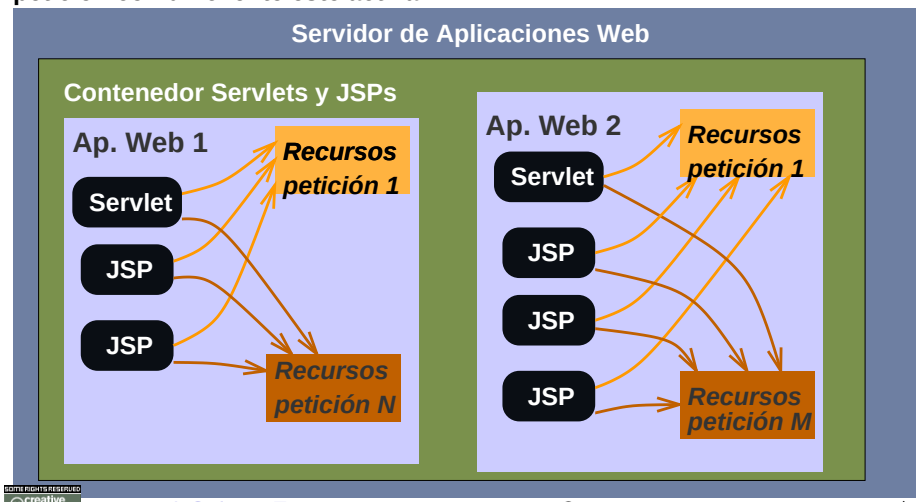


Session scope (II)

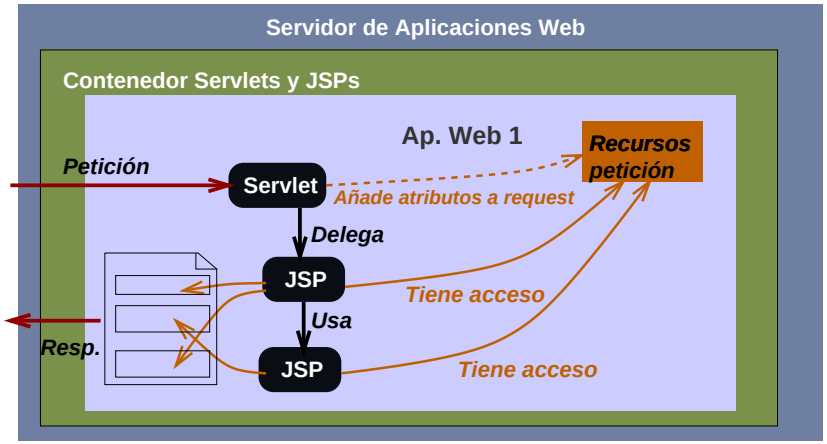


Request scope (I)

¿Cómo se definen variables que sean visibles por todos los componentes de la aplicación web y que se mantienen accesibles (scope) mientras la petición con un cliente esté activa?



Request scope (II)



También está *Page Scope* que contiene objetos que son puestos a disposición de un componente mientras se ejecuta.