Servlet Context
Servlet Event Listeners
Proteger Recursos Compartidos
RequestDispatcher

La interface ServletContext

- La interface **ServletContext** es una **representación en ejecución** de la **aplicación web**, es una **vista de la aplicación web**.
- Cada **Contenedor Web ofrece una implementación específica** de la interface **jakarta.servlet.ServletContext**. La funcionalidad es idéntica, no depende de la implementación, está establecida por la interface.
- A través del objeto **ServletContext** un **servlet puede**:
  - **Acceder a parámetros de inicialización** de la aplicación web.
  - **Obtener referencias a urls de los recursos web estáticos** de la aplicación.
  - ***Loggear* eventos** de la aplicación.
  - **Compartir objetos con alcance "aplicación"** para que otros servlets puedan accederlos.
  - **Interactuar con otros servlets**.
- Existe un **único objeto ServletContext por aplicación web**, por JVM. En el caso de una instalación multi-server, la aplicación web tendrá un objeto ServletContext por JVM.
- **ServletContext** está asociado a una **ruta** dentro del Contenedor Web llamada **context root.**

Parámetros de Inicialización de la Aplicación Web

- Los **parámetros de inicialización** de la aplicación web ofrecen a **TODOS** los servlets de **información de configuración inicial** de la aplicación.
- El objeto **ServletContext** permite recuperar estos parámetros a través de los métodos:

Permiten acceder a parámetros de inicialización de la aplicación web

```
String getInitParameter(String nombre)
Enumeration getInitParameterNames()
```

Los **parámetros de inicialización de la aplicación web** se especifican en el **web.xml**:

```
<web-app>
    <context-param>
        <param-name>email</param-name>
        <param-value>admin@info.unlp.edu.ar</param-value>
    </context-param>
    <servlet>
        <servlet-name>HolaMundo</servlet-name>
        <servlet-class>com.servlets.HolaMundo</servlet-class>
<servlet>
```

Las ocurrencias del tag <context-param> deben aparecer antes que cualquier ocurrencia del tag <servlet>

- El objeto **ServletContext** está contenido en el objeto **ServletConfig**, que es provisto al servlet por el Contenedor Web en el momento de la inicialización.
- ¿Cómo acceder al objeto **ServletContext**? Hay de 2 maneras:
  - A través del objeto ServletConfig: **HttpServlet.getServletConfig().getServletContext()**
  - Directamente a través de la instancia del servlet: **HttpServlet. getServletContext()**

Parámetros de Inicialización de la Aplicación Web

```
package linti.servlets;
import java.io.*;
import jakarta.servlet.*;
import jakarta.servlet.http.*;
public class PaginaError extends HttpServlet {
    public void doGet(HttpServletRequest req, HttpServletResponse resp)
                                                    throws IOException, ServletException{

    resp.setContentType("text/html");
    PrintWriter out=resp.getWriter();
    out.println("<html>");
    out.println("<head>");

    out.println("<title> Ocurrió un Error </title>");
    out.println("</head>");
    out.println("<body>");
    ServletContext sc=this.getServletContext();
    String mail=sc.getInitParameter("email");
    out.println("<h1> Página de Error </h1>");
    out.println("Ha ocurrido una error inesperado. Por favor, repórtelo a: "+ mail+ ".");
    out.println("</body>");
    out.println("</html>");
    }
}
```

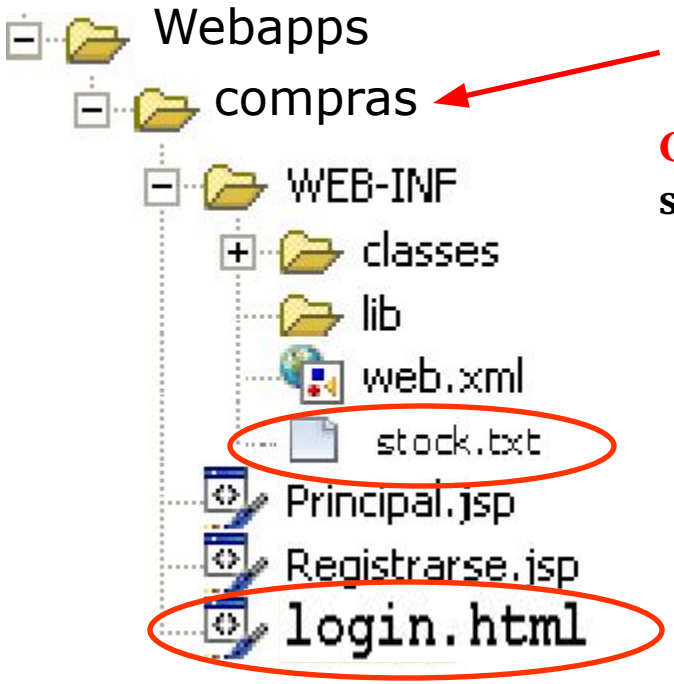
```
<web-app>
 <context-param>
    <param-name>email</param-name>
     <param-value>admin@info.unlp.edu.ar</param-value>
 </context-param>
```

- **Los parámetros de inicialización de la aplicación web pueden usarse en todas las componentes web.**
- **Los parámetros se pueden cambiar fácilmente.**

ServletContext sc=this.getServletConfig().getServletContext();

Recursos Web Estáticos

El objeto **ServletContext** está asociado a una **ruta en el servidor** y provee **acceso a la jerarquía de documentos estáticos** que forman parte de la aplicación web, como por ej. archivos HTML, GIF, JPEG.



document root

Context Root o Context Path: identifica a una aplicación web en un servidor JEE, comienza con "/" y finaliza con un string.

http:// www.miempresa.com/compras

Directorios Virtuales

/WEB-INF/stock.txt

/login.html

Todos los recursos de una aplicación web son abstraídos en **directorios virtuales** a partir del **context root**. Un **directorio virtual** comienza con "/" y continúa con una ruta virtual a directorios y recursos.

Recursos Web Estáticos

Métodos de **ServletContext** para **acceder a recursos web estáticos**:

URL getResource (String path)

Devuelve la URL al recurso que coincide con la **ruta virtual** dada como parámetro. La ruta debe comenzar con "*/*" y es relativa a la raíz del **context root**. Este método permite recuperar un recurso de la aplicación, independientemente si está ubicado en un archivo JAR, dentro del WAR o en el file system. El método retorna null si no existe el recurso en la ruta especificada.

```
URL una_url=sc.getResource("/login.html");
```

```
/C:/Documents and Settings/claudiaq/jdev/cursoj2ee06/practiac3/public_html/login.html
```

InputStream getResourceAsStream(String path)

Devuelve el recurso ubicado en la ruta virtual especificada como parámetro, como un objeto **InputStream**. Los datos en el InputStream pueden ser de cualquier tipo y longitud. El método retorna null si no existe el recurso en la ruta especificada. La ruta debe empezar con "*/*" y es relativa a la raíz del **context root**. Este método permite al Contenedor Web hacer disponible al servlet un recurso ubicado en cualquier lugar (archivo WAR, en el file system).

```
InputStream is = this.getServletContext().getResourceAsStream("/WEB-INF/stock.txt");
```

Recursos Web Estáticos

String getRealPath(String path)

Devuelve un String que contiene la **ruta real** de la ruta virtual especificada como parámetro. Devuelve null si la aplicación se cargó desde un archivo WAR.

getRealPath("imagenes/fuego.jpg");

"C:\Documents and Settings\claudiaq\jdev\cursoj2ee06\practiac3\public_html\imagenes\fuego.html"

Set getResourcePaths(String path)

Devuelve un Set que contiene los **nombres de todos los recursos en la ruta virtual** especificada. La ruta debe empezar con "/". Las rutas retornadas son relativas al document root.

getResourcePaths("/") devuelve {"/bienvenido.html", "/catalogo/", "/cliente/", "/WEB-INF/"}

getResourcePaths("/catalogo/") devuelve {"/catalogo/inicio.html", "/catalogo/productos.html", /catalogo/ofertas/"}

compras

- /bienvenido.html
- /catalogo/inicio.html
- /catalogo/productos.html
- /catalogo/ofertas/libros.html
- /catalogo/ofertas/musica.html
- /cliente/login.jsp
- /WEB-INF/web.xml
- /WEB-INF/classes/compras.ordenes.OrdendeCompras.class

Recursos Web Estáticos

```
public class LeeRecursos extend HttpServlet {
public void doGet(HttpServletRequest req, HttpServletResponse resp)
                                                    throws IOException, ServletException {
//TODO
try {
InputStream is = this.getServletContext().getResourceAsStream("/WEB-INF/recursos/catalogo.txt");
BufferedReader r = new BufferedReader(new InputStreamReader(is));
int posComa=0;
String linea = r.readLine();
    while (linea != null) {
      posComa = linea.indexOf(",");
      String nom = linea.substring(0, posComa);
      String precio = linea.substring(posComa + 1);
      linea = r.readLine();
     }
}
catch (IOException e) {
// TODO
}
}
```

Se procesa el archivo catalogo.txt

El uso más común y práctico de acceso a recursos web estáticos es a archivos planos que forman parte de la aplicación web. Pueden ser archivos de configuración o simplemente archivos para inicialización de datos.

Logging de Eventos de la Aplicación

Métodos para logging de información a través del objeto ServletContext.

- **void log(String mensaje)**

  Escribe el String especificado en el archivo de logs del servlet o en el repositorio de logs.

- **void log(String mensaje, java.lang.Throwable e)**

  Escribe el mensaje y el *stack trace* de la excepción pasada como parámetro. El propósito del mensaje es que sea una explicación breve de la excepción.

La especificación de Servlets no indica dónde se guardará o mostrará la información de log.

Ventajas de los métodos log()

Proveen un lugar común en donde enviar información sobre eventos de la aplicación web.

Desventajas de los métodos log()

Solamente el código que tiene acceso al objeto **ServletContext** puede guardar información de log fácilmente.

Una mejor solución y más comúnmente usada para hacer logging de aplicaciones web es disponer de una API que permita desde cualquier clase acceder y usar el logging. Ejemplo: **Log4j**, **http://jakarta.apache.org/log4j** o con la **API estándar para logging (java.util.logging).** Cuando se requiere un mecanismo de logging robusto, ambas soluciones son preferibles al mecanismo de logging de la API de Servlets.

Compartir información
Objetos de alcance aplicación

- El objeto **ServletContext** permite **vincular y recuperar** objetos **server-side** con **alcance aplicación**. Los objetos se vinculan mediante nombres de atributos.

  void setAttribute(String nombre, Object attr)
  Object getAttribute(String nombre)
  void removeAttribute(String nombre)
  Enumeration getAttributeNames()

- El **alcance aplicación** es un espacio de memoria ideal para que diferentes componentes web JAVA **compartan objetos**. Una vez que un objeto se ligó a este alcance permanecerá en él mientras la aplicación esté corriendo.
- El **alcance aplicación** debe usarse con moderación: los objetos ligados al ServletContext no serán eliminados por el GC hasta que el ServletContext sea removido (esto ocurre cuando la aplicación se da de baja o es reiniciada).

```
ServletContext sc=this.getServletContext();
String s= sc.getInitParameter("urlBD");
sc.setAttribute("url",s);
```

¿De dónde se lee este dato?

Liga un objeto al *alcance aplicación* asociándolo el nombre url

```
ServletContext sc=this.getServletContext();
String dbURL=(String) sc.getAttribute("url");
```

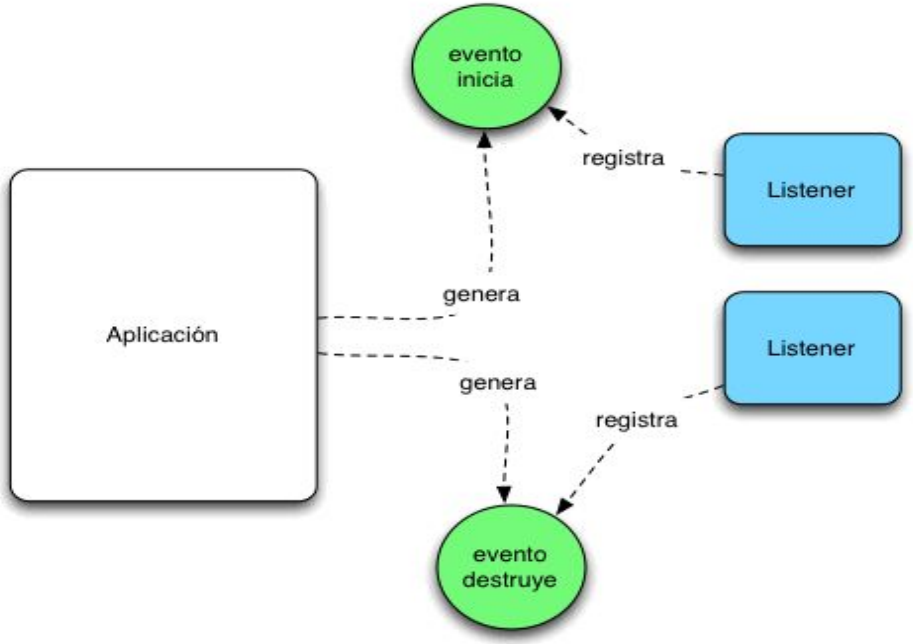
Recupera el objeto identificado como url del *alcance aplicación*

Eventos del Ciclo de Vida de una Aplicación Web

El **manejo de eventos de la aplicación WEB** le ofrece al desarrollador de un **amplio control sobre el ciclo de vida** de los objetos **ServletContext**, **HttpSession** y el **ServletRequest.** Asimismo, mejoran la **factorización del código** e **incrementan la eficiencia en el gerenciamiento de recursos** que usa la aplicación Web.

El **modelo de eventos del ciclo de vida de una aplicación web** funciona de una manera similar a los eventos de componentes de GUI en JAVA: AWT y Swing.

Ciclo de Vida de una Aplicación Web



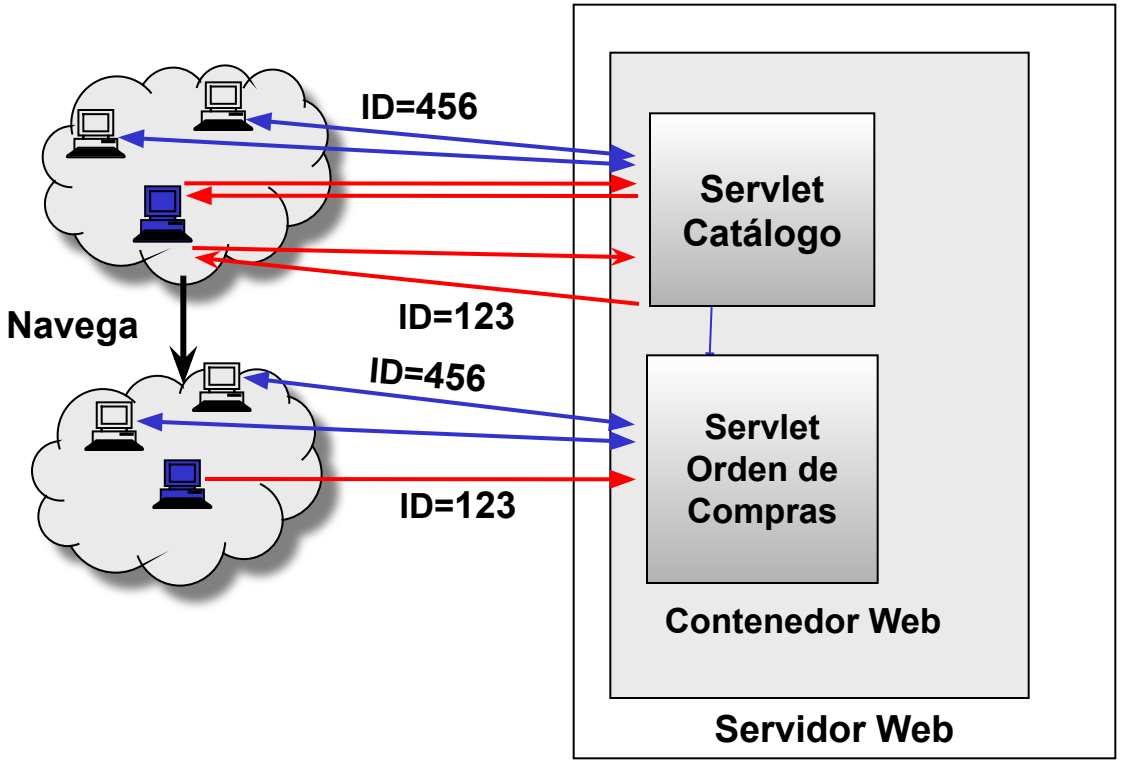
ServletContextListener

Es un listener que gestiona los eventos generales de una aplicación

- Cuando el **Contenedor Web inicia**, cada una de las aplicaciones web se **inicializan** – **Evento: Aplicación Nueva**.
- Cuando se da de **baja al Contenedor Web**, cada una de las aplicaciones web se **destruyen** – **Evento: Aplicación Destruida**.
- Es posible **monitorear** y **reaccionar** ante los **eventos del ciclo de vida** de una aplicación web.

Conociendo los eventos del ciclo de vida de una aplicación es posible escribir el código apropiado, por ejemplo para cargar datos de una bd en el arranque de la aplicación.

Ciclo de Vida de las Sesiones



Los eventos del ciclo de vida de una sesión son:

- **creación de una sesión nueva**
- **destrucción de una sesión**

Con Servlets es posible hacer seguimiento de clientes, sin embargo es más sencillo disponer de un objeto que reaccione cuando un cliente inicia una sesión nueva y cuando la sesión es invalidada o finaliza por time-out.

Conociendo **los eventos del ciclo de vida de una sesión**, es posible escribir el código apropiado, por ejemplo para incrementar en uno la cantidad de usuarios de la aplicación cada vez que se crea una sesión nueva y descontar en uno dicha cantidad cuando una sesión se invalida o finaliza por *time-out*.

Servlet Event Listener
¿Qué son? ¿Para qué sirven?

Los **Servlet Event Listeners** ofrecen un mecanismo para que el **Contenedor Web** **notifique** a la aplicación web sobre **eventos** importantes

- A partir de la versión 2.3, la API de Servlet incorpora las componentes **Servlet Event Listeners.**
- Todos los **Servlet Event Listener** están definidos mediante interfaces.
- Existen interfaces **listeners** para todos los **eventos importantes relacionados a la aplicación web**.
- Para que la **aplicación web sea notificada acerca** de los eventos importantes que ocurren, es necesario **escribir una clase** que **implemente la interface listener** adecuada y anotarla o describirla en el **web.xml**.
- Los **listeners** son instanciados y registrados por el Contenedor Web.

Interfaces Listeners:

jakarta.servlet.ServletContextListener → **Escucha eventos del ciclo de vida de una aplicación**

jakarta.servlet.ServletContextAttributeListener

jakarta.servlet.http.HttpSessionListener → **Escucha eventos del ciclo de vida de una sesión**

jakarta.servlet.http.HttpSessionActivationListener

jakarta.servlet.http.HttpSessionAttributeListener

jakarta.servlet.http.HttpSessionBindingListener

jakarta.servlet.ServletRequestListener → **Escucha eventos del ciclo de vida de un requerimiento**

jakarta.servlet.ServletRequestAttributeListener

Se agregaron en la versión 2.4 de la API de Servlets

Servlet Event Listener
Interfaces Listener

Los **Servlet Event Listeners** soportan **notificación de eventos** por cambios de estado en los objetos **ServletContext**, **HttpSession** y **ServletRequest**.

	Evento a monitorear	Interface a implementar
Objeto	**Evento**	**Interface Listener**
Contexto de la Aplicación: ServletContext	Inicialización Destrucción	jakarta.servlet.ServletContextListener
	Atributo agregado Atributo removido Atributo reemplazado	jakarta.servlet.ServletContextAttributeListener
Sesiones: HttpSession	Creación Invalidación Time-Out	jakarta.servlet.http.HttpSessionListener
	Atributo agregado Atributo removido Atributo reemplazado	jakarta.servlet.http.HttpSessionAttributeListener

Servlet Event Listener
Interfaces Listener

Evento a monitorear **Interface a implementar**

Objeto	Evento	Interface Listener
Sesiones: HttpSession	Activación Pasivación	jakarta.servlet.http.HttpSessionActivationListener
Requerimiento: ServletRequest	Creación Destrucción	jakarta.servlet.ServletRequestListener
	Atributo agregado Atributo removido Atributo reemplazado	jakarta.servlet.ServletRequestAttributeListener

Los **Listeners de Contexto** se usan para gerenciar recursos o estados mantenidos por la aplicación web (en una JVM).

Los **Listeners de Sesión** se usan para gerenciar recursos o estado asociado con una serie de peticiones realizadas a la aplicación web desde el mismo cliente.

Los **Listeners de Requerimientos** se usan para gerenciar recursos o estado asociado con el ciclo de vida de cada requerimiento.

Listeners de los objetos ServletContext y HttpSession

ServletContextListener: **escuchan** los **eventos del ciclo de vida de una aplicación web**. Cuando la aplicación se **carga**, se **agrega** o se da de **baja**, los **métodos apropiados se ejecutan**. Son ideales para inicializar recursos compartidos. Ejemplo: pool de conexiones a la bd.

HttpSessionListener: **escuchan** los e**ventos de sesión** tales como la **creación**, **invalidación** o **finalización por tiempo** de una sesión.

ServletContextAttributeListener: **escuchan eventos** tales como **agregar**, **remover** o **reemplazar atributos** del **contexto de la aplicación web** (del objeto ServletContext).

HttpSessionAttributeListener: **escuchan** eventos tales como **agregar**, **remover** o **reemplazar atributos** de la **sesión de usuario** (del objeto HttpSession)

La interface ServletContextListener

Un **objeto** que **implementa** la interface **jakarta.servlet.ServletContextListener** **escucha los eventos del ciclo de vida de una aplicación web**. Asociada con esta interface listener está la clase **jakarta.servlet.ServletContextEvent** que representa el **evento ocurrido** y permite acceder al objeto **ServletContext.**

La interface **ServletContextListener** es usada típicamente para **inicializar recursos compartidos**, por ejemplo el pool de conexiones JDBC, que se crea una sola vez y se re-usa en toda aplicación web.

Tiene dos métodos:

getServletContext()

void contextInitialized(ServletContextEvent e)

Es invocado cuando una **aplicación web está lista para ejecutarse por primera vez**, esto es cuando **arranca el Contenedor**, cuando **la aplicación es agregada o recargada**. Los requerimientos http a la aplicación no son atendidos hasta que este método haya terminado de ejecutarse.

void contextDestroyed(ServletContextEvent e)

Es invocado cuando una **aplicación web está por removerse**, esto es cuando **se da de baja al Contenedor** o cuando **una aplicación es removida**.

Ejemplo de ServletContextListener

```
package mislisteners;
public class InicializaListaDeProductos implements ServletContextListener {
    public void contextInitialized(ServletContextEvent sce) {
        ServletContext contexto=sce.getServletContext();
        String archCatalago=contexto.getInitParameter("archivoCatalago");
        BufferedReader cat=null;
        Hashtable catalogo=new Hashtable();
        try {
            cat=new BufferedReader (new
            InputStreamReader(contexto.getResourceAsStream(archCatalogo)));
         // TODO: Parsea el archivo y construye una lista de productos en el objeto catalogo
            contexto.setAttribute("stock", catalogo);
            contexto.log("lista de Productos creada ");
        } catch(Exception e) {contexto.log("Ocurrió una excepción mientras….",e));
        }finally{
                if (cat!=null) {
                    try{cat.close();} catch(Exception e) {}
         }
      }
    }
    public void contextDestroyed(ServletContextEvent sce) {// TODO}
} // Fin de la clase
```

Parámetro de inicialización del web.xml

Lee del web.xml la ruta virtual del archivo que está asociado al parámetro "archivoCatalgo"

Guarda el objeto catalogo como atributo del ServletContext

Escribe el archivo de log

Descripción de Servlet Listeners en el web.xml

- Los **servlet listeners** deben **configurarse** en el archivo **web.xml** o **anotarse** y de esta manera el **Contenedor Web se entera** de su existencia. Se usa el tag **<listener>**.
- El **Contenedor Web** crea **una instancia de cada clase listener** declarada y **registra** dicho objeto para ser **notificado ante la ocurrencia de eventos**.
- Si el **orden en que deben invocarse** los listeners, servlets y filtros es **importante**, entonces debe usarse el **web.xml**.

- Es posible declarar múltiples listeners para cada tipo de evento.
- El orden en que son declarados determina el orden en el que son invocados por el Contenedor Web durante el arranque. El orden se invierte cuando se de baja la aplicación.
- El tag listener se declara después del tag <context-param> y antes de la definición de todos los servlets de la aplicación web.

```
<web-app>

<!--Inicializa el catálogo de compras-->
 <context-param>
   <param-name> archivoCatalogo </param-name>
  <param-value> WEB-INF/catalogo.txt </param-value>
 </context-param>

 <listener>
   <listener-class> mislisteners.InicializaListaDeProductos </listener-class>
 </listener>

</web-app>
```

Cuando se definen listeners, servlets y filtros, el orden en que son invocados no está especificado

Anotar Listeners

```
package mislisteners;
@WebListener
public class InicializaListaDeProductos implements ServletContextListener {
    public void contextInitialized(ServletContextEvent sce) {// TO DO}
    public void contextDestroyed(ServletContextEvent sce) {// TO DO}
}
```

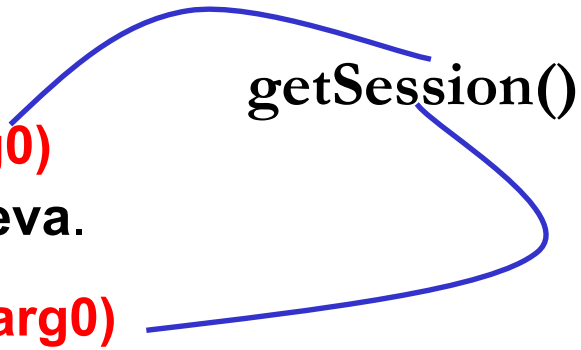
La anotación **@WebListener** está disponible a partir de la versión 3.0 de la API de Servlets y permite **registrar listeners**.

Si se necesita un control exhaustivo de los listeners, por ejemplo establecer un orden de registración, entonces se recomienda usar el web.xml.

La interface HttpSessionListener

Un objeto que implementa la interface **jakarta.servlet.http.HttpSessionListener** **es notificado** de los **cambios que se producen en la lista de sesiones activas** de la aplicación web. Asociada con esta interface listener está la clase **jakarta.servlet.http.HttpSessionEvent** que representa el **evento ocurrido** y permite acceder al objeto **HttpSession.**

Tiene dos métodos:



void sessionCreated(HttpSessionEvent arg0)

Es **invocado** cuando se **crea una sesión nueva**.

void sessionDestroyed(HttpSessionEvent arg0)

Es **invocado** cuando una **sesión es invalidada** porque expiró el tiempo (time-out) o porque se invocó al método invalidate().
También es invocado cuando se da de **baja la aplicación** antes de ser notificados los listeners de contexto, dado que el listener de sesión debe enterarse que se invalidó la sesión antes que el de contexto se entere que se dio de baja a la aplicación.

Ejemplo de HttpSessionListener

```
package mislisteners;
public class SeguimientoDeUsuarios implements HttpSessionListener {
        private static int users=0;
        public void sessionCreated(HttpSessionEvent e) {
                users++;
        }
        public void sessionDestroyed(HttpSessionEvent e){
                users--;
        }
        public static int getCantUsrs(){
                return users;
        }
} // Fin de la clase
```

```
<listener>
        <listener-class> mislisteners.InicializaListaDeProductos </listener-class>
</listener>
<listener>
        <listener-class> mislisteners.SeguimientoDeUsuarios </listener-class>
</listener>
```

web.xml

Ejemplo de HttpSessionListener

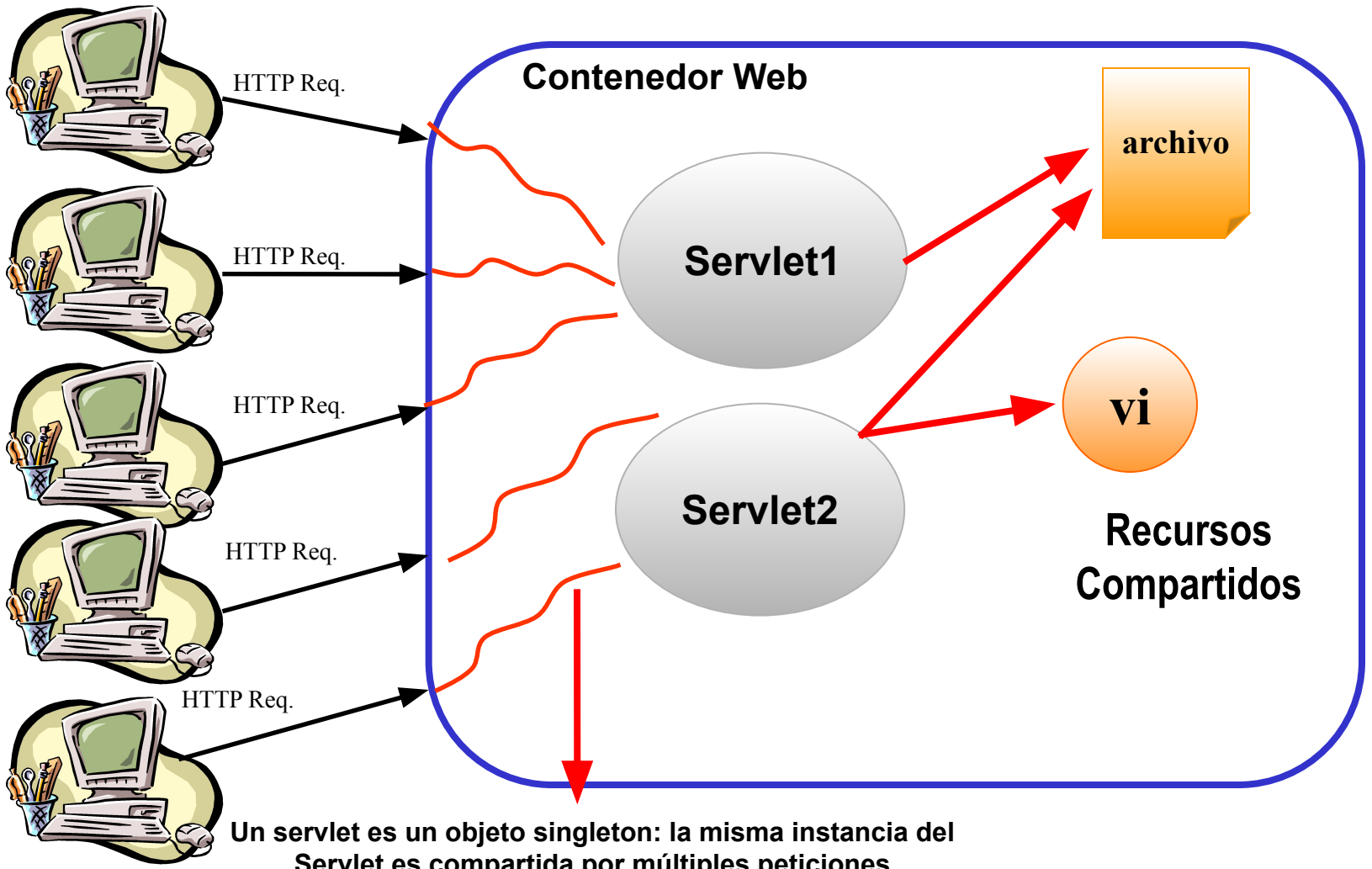
```
package misservlets;
public class MostrarUsuarios implements HttpServlet {

    public void doGet(HttpServletRequest req, HttpServletResponse resp)
                        throws IOException, ServletException {
    req.getSession();
    resp.setContentType("html/text");
    PrintWriter out=resp.getWriter();
    out.println("<html>");
    out.println("Usuarios: ");
    out.println(SeguimientoDeUsuarios.getCantUsrs());
    out.println("</html>");
    }
} // Fin de la clase
```

El navegador despliega una página HTML que muestra la cantidad de usuarios que están usando la aplicación web

```
<listener>
    <listener-class> misservlets.InicializaListaDeProductos </listener-class>
</listener>
<listener>
    <listener-class> misservlets.SeguimientoDeUsuarios </listener-class>
</listener>
```

web.xml

Problemas de Concurrencia en Aplicaciones Web



Variables de instancia, variables de clase, archivos que residen en el file system

Un servlet es un objeto singleton: la misma instancia del Servlet es compartida por múltiples peticiones simultáneas provenientes de diferentes clientes

Problemas de Concurrencia en Aplicaciones Web

Alcance	Thread-Safe	Ejemplo

Variables Instancia	NO	class Hola {
		private double d;
		}

Variables Clase	NO	class Hola {
		private static double d;
		}

Parámetros de Métodos	SI	void doGet(HttpServletRequest req, HttpServletResponse resp) {...}

Variables Locales	SI	void doGet(HttpServletRequest req,
		HttpServletResponse resp) {
		double d;
		}

Datos de Sesión	NO	HttpSession se= req.getSession();
Datos de Aplicación	NO	ServletContext ctx= this.getServletContext();

Problemas de Concurrencia
Proteger el Estado Sesión y Aplicación

- Los objetos **HttpSession** y **ServletContext** son **recursos compartidos**.
- El objeto **HttpSession** puede ser accedido simultáneamente por dos requerimientos diferentes desde un mismo cliente.
- El objeto **ServletContext** puede ser accedido por múltiples componentes web a la vez.

```
ServletContext ctx=this.getServletContext();
synchronized (ctx) {
    PoolConexion p=(PoolConexion)ctx.getAttribute("pool");
    //Código que testea si hay conexiones libres, en cuyo caso la obtiene
    // y actualiza el pool de conexiones p
    ctx.setAttribute("pool",p);
}
```

Redireccionar la Respuesta

Redireccionar la respuesta HTTP

sendRedirect(String URL) de la interface jakarta.servlet.HttpServletResponse

Redirecciona el requerimiento del cliente a la URL especificada en el parámetro.

¿Cómo funciona?: usa el código de respuesta HTTP 302 que indica **"El recurso que está buscando el cliente fue temporariamente movido".**

Toma el String que recibe como parámetro, que representa una URL, y automáticamente le setea el código HTTP 302. El navegador, sin informar al usuario, direcciona el requerimiento a la URL enviada.

resp.sendRedirect("/compras/Productos");

resp.sendRedirect("http://www.google.com.ar");

Delegar el Requerimiento

En las aplicaciones web frecuentemente es útil **delegar** el procesamiento del requerimiento original en otro servlet o **incluir** la salida de un servlet en la respuesta original.
La interface **RequestDispatcher** provee un mecanismo para lograrlo.

La interface RequestDispatcher

Es un "wrapper" que **encapsula un recurso web** ubicado en una ruta particular o de acuerdo a un nombre dado.
Permite **transferir el control** a otro recurso web para que le responda al cliente.
El objetivo del **RequestDispatcher** es "wrappear" servlets, pero también permite crear objetos **RequestDispatcher** para diferentes tipos de recursos web.
Es posible **obtener el RequestDispatcher** de un recurso web usando los siguientes métodos del objeto **ServletContext:**

getRequestDispatcher(String path) el path es relativo a la raíz del context root y comienza con "/"

getNamedDispatcher(String nom) nom es el nombre de un servlet declarado en el web.xml

¿Cómo delegar el requerimiento HTTP?

Para delegar el procesamiento del requerimiento se usa el siguiente método del **RequestDispatcher**:

foward(jakarta.servlet.ServletRequest, jakarta.servlet.ServletResponse)

Delega el requerimiento y la respuesta al objeto **RequestDisptacher**.
El proceso íntegro de delegación del requerimiento se realiza del lado del servidor. A diferencia del redireccionamiento de la respuesta (senRedirect), no requiere de ninguna acción por parte del cliente ni del envío de información extra entre el cliente y el servidor. Permite pasar el requerimiento actual a otro servlet para que continúe el procesamiento y responda al cliente.

Delegar el Requerimiento

```
@WebServlet("/origen")
public class OrigenServlet extends HttpServlet {
    public void doGet(HttpServletRequest request, HttpServletResponse response)
                                                    throws ServletException, IOException {
      // Obtiene el RequestDispatcher y reenvía la solicitud al otro servlet
      request.setAttribute("mensaje", "Hola desde OrigenServlet");
      RequestDispatcher dispatcher = getServletContext().getRequestDispatcher("/destino");
      dispatcher.forward(request, response);
    }
}
```

Detiene la ejecución del servlet Origen y pasa el control al servlet Destino

```
@WebServlet("/destino")
public class DestinoServlet extends HttpServlet {
   public  void doGet(HttpServletRequest request, HttpServletResponse response)
                                                    throws ServletException, IOException {
      response.setContentType("text/html;charset=UTF-8");
      PrintWriter out = response.getWriter();
      String mensaje = (String) request.getAttribute("mensaje");
      out.println("<html><body>");
      out.println("<h2>Servlet de Destino</h2>");
      out.println("<p>" + mensaje + "</p>");
      out.println("</body></html>");
      }
}
```

```
RequestDispatcher dispatcher = request.getRequestDispatcher("destino");
```

Incluir Recursos Programáticamente

Incluir Recursos

Para incluir contenido “server-side” se usa el siguiente método del **RequestDispatcher**:

include(jakarta.servlet.ServletRequest, javax.servlet.ServletResponse)

El servlet que es receptor del método include() tiene acceso completo al requerimiento original, pero es limitado el acceso a la respuesta: puede escribir pero no está permitido cambiar el código de estado de la respuesta ni cambiar o modificar el header de la respuesta ni modificar cookies.

El servet original continúa la ejecución

Se ejecuta la componente web (que queremos incluir) y el resultado de la ejecución se incluye en la respuesta del servlet que invoca al include().

```
ServletContext ctx=this.getServletContext();
RequestDispatcher dispatcher=ctx.getRequestDispatcher(“/banner”);
if (dispatcher!=null) dispatcher.include(request,response);
```

```
RequestDispatcher dispatcher=request.getRequestDispatcher(“header.html”);
if (dispatcher!=null) dispatcher.include(request,response);
```

Es posible en los métodos **getRequestDispatcher()** de ServletContext y de ServletRequest usar rutas que contengan parámetros

```
String path =“/banner.jsp?titulo=JAVA ”;
RequestDispatcher rd = this.getServletContext().getRequestDispatcher(path);
rd.include(request,response);
```