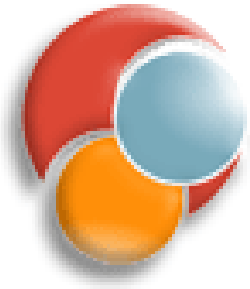


Componentes Web

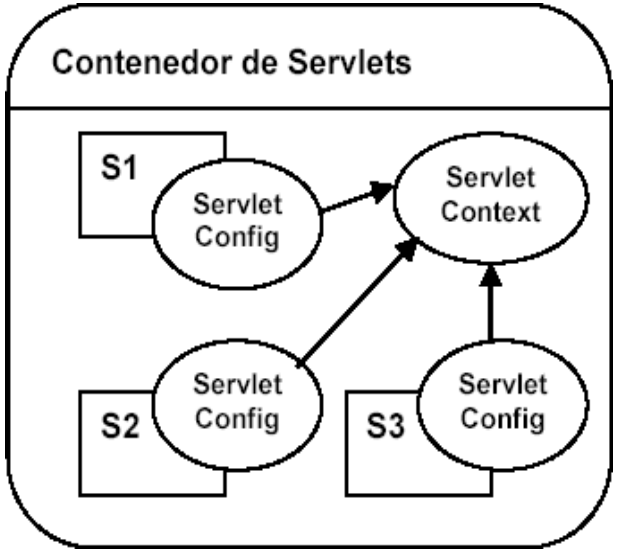
Sesión 4: Contexto y redirecciones

Puntos a tratar

- Contexto de los Servlets
- Parámetros de inicialización
- Acceso a recursos estáticos
- Redirecciones
- Generación de logs

Contexto compartido

- Cada contexto (aplicación web) tiene asociado un único objeto `ServletContext` que lo representa
  - Compartido por todos los componentes de la aplicación web
  - En sistemas distribuidos existe uno por cada VM



- Se obtiene desde dentro de cualquier servlet con

```
ServletContext contexto = getServletContext();
```

Funciones de ServletContext

- Compartir información (objetos Java) entre los distintos componentes de la aplicación web
- Leer los parámetros de inicialización de la aplicación web
- Leer recursos estáticos de la aplicación web
- Hacer redirecciones de la petición a otros recursos
- Escribir *logs*

Atributos del contexto

- Nos permiten compartir información entre los distintos elementos de la aplicación web
- Esta información se almacena en forma de objetos Java
- Establecemos un atributo con

```
contexto.setAttribute(nombre, valor);
```

- Este objeto podrá ser leído desde cualquier componente de la aplicación web con

```
Object valor = contexto.getAttribute(nombre);
```

Listeners del contexto

- Escuchan eventos producidos en el objeto `ServletContext`
- Distinguimos dos tipos:
  - `ServletContextListener`

    Notifica la creación y la destrucción del contexto
  - `ServletContextAttributeListener`

    Notifica los cambios en los atributos del contexto

    Creación de atributos

    Eliminación de atributos

    Modificación de atributos
- Deben declararse en el descriptor de despliegue

ServletContextListener

```
package es.ua.jtech.listener;
import javax.servlet.*;

public class MiContextListener
            implements ServletContextListener {

   public void contextDestroyed(ServletContextEvent sce)
   {
     // Destruccion del contexto
   }

   public void contextInitialized(ServletContextEvent sce)
   {
     // Inicialización del contexto
   }
}
```

- P.ej, nos permite inicializar estructuras de datos al crearse el contexto.

ServletContextAttributeListener

```
package es.ua.jtech.listener;
import javax.servlet.*;

public class MiContextAttributeListener
        implements ServletContextAttributeListener {

   public void attributeAdded(ServletContextAttributeEvent scae)
   {
        // Se ha añadido un nuevo atributo
   }

   public void attributeRemoved(ServletContextAttributeEvent scae)
   {
        // Se ha eliminado un atributo
   }

   public void attributeReplaced(ServletContextAttributeEvent scae)
   {
        // Un atributo ha cambiado de valor
   }
}
```

Declaración de listeners

- El objeto del contexto es global a la aplicación web
  - Los *listeners* del contexto deben ser declarados en el descriptor de despliegue de la aplicación web
  - Los *listeners* declarados deben estar implementados entre las clases de la aplicación web
  - El contenedor web instanciará los *listeners* declarados en este fichero `web.xml` e invocará sus métodos cada vez que se produzca el correspondiente evento en el contexto

```
<listener>
   <listener-class>es.ua.jtech.listener.MiContextListener
   </listener-class>
</listener>

<listener>
   <listener-class>es.ua.jtech.listener.MiContextAttributeListener
   </listener-class>
</listener>
```

Parámetros globales

- Se pueden declarar parámetros globales del contexto en el descriptor de despliegue

```
<context-param>
  <param-name>param1</param-name>
  <param-value>valor1</param-value>
</context-param>
```

- Podemos leer estos parámetros a través del objeto `ServletContext`

```
String valor = contexto.getInitParameter(nombre);
```

Lectura de recursos

- Podemos leer recursos estáticos del contexto con

```
InputStream in = contexto.getResourceAsStream(ruta);
```

- Donde la ruta debe comenzar por "`/`"
  - Es una ruta relativa a la raíz del contexto
  - P.ej. "`/index.htm`" busca el fichero `index.htm` en el directorio raíz del contexto
- Leerá cualquier recurso como estático
  - P.ej, si leemos un recurso "`/index.jsp`" leerá el código fuente del JSP, no procesará este código
  - Si queremos que sea procesado el contenido dinámico deberemos utilizar los métodos `include` o `forward`

Tipos de redirecciones

- Redirecciones en el cliente
  - El servidor envía cabecera de redirección al cliente en la respuesta indicando la URL a la que debe redirigirse
  - El navegador de la máquina cliente realiza una nueva petición a esta URL
- Redirecciones en el servidor
  - *Forward*

    El servidor sirve un recurso distinto al indicado en la URL de forma transparente para el cliente
  - *Include*

    Se incluye el contenido de un recurso dentro del contenido generado por nuestro servlet

Redirecciones en el cliente

- Realizamos estas redirecciones enviando una cabecera de redirección en la respuesta

```
response.sendRedirect(url);
```

- Será el cliente el responsable de interpretar esta cabecera y realizar una nueva petición a la dirección indicada
  - Se mostrará la nueva URL en la barra de dirección del navegador
- Sólo debemos llamar a este método antes de haber generado la respuesta del servlet
  - Si no, se producirá una excepción `IllegalStateException`

Redirecciones en el servidor

- Obtenemos un objeto `RequestDispatcher` a partir del contexto

```
RequestDispatcher rd = contexto.getRequestDispatcher(ruta);
```

- La ruta comenzará con "`/`"
  - Relativa al contexto
- Realizamos la redirección con

```
rd.forward(request, response);
```

- Esto debe hacerse antes de haberse comenzado a escribir la respuesta
  - Si no, se producirá una excepción `IllegalStateException`

Inclusiones

- Podemos utilizar el `RequestDispatcher` para incluir el recurso indicado dentro de la respuesta generada por nuestro servlet

```
rd.include(request, response);
```

- En este caso podremos haber escrito ya contenido en la respuesta
  - El recurso solicitado se escribirá a continuación de este contenido
  - Podremos seguir escribiendo contenido tras la inclusión del recurso

- Tanto con `forward` como con `include` se procesarán los recursos dinámicos solicitados (por ejemplo los JSP)

Escritura de logs

- Podemos escribir mensajes en el fichero de *logs* de Tomcat
  - Registros de eventos sucedidos en nuestra aplicación
  - Depuración de errores
- Escribimos los *logs* utilizando el objeto del contexto

```
contexto.log(mensaje);
```